Gerard O'Regan

A Brief History of Computing

Second Edition

Springer

A Brief History of Computing

Gerard O'Regan

A Brief History of Computing

Second Edition

Springer

Gerard O'Regan
11 White Oaks
Mallow, Co. Cork
Ireland

ISBN 978-1-4471-2358-3 e-ISBN 978-1-4471-2359-0
DOI 10.1007/978-1-4471-2359-0
Springer London Dordrecht Heidelberg New York

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Control Number: 2012934180

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

To my wonderful nieces and nephews
Jamie, Tiernan, Cian, Aoife, Lorna, Daniel
and Niamh

Preface

Overview

The objective of this book is to provide an introduction into a selection of key events in the history of computing. The computing field is a vast area, and a truly comprehensive account of its history would require several volumes. The aims of this book are more modest, and it aims to give the reader a flavour of some of the important events in the history of computing. It is hoped that this will stimulate the interested reader to study the more advanced books and articles available.

The history of computing has its origins in the dawn of civilisation. Early hunter gatherer societies needed to be able to perform elementary calculations such as counting and arithmetic. As societies evolved into towns and communities, there was a need for more sophisticated calculations.

Our account commences with the contributions of the Egyptians and Babylonians. We then move swiftly onwards to more modern times. Along the journey, we consider some early computers such as Zuse's machines, the Atanasoff-Berry Computer and ENIAC. We move forward to consider developments such as the SAGE air defence system, the IBM 360 and the PDP family of minicomputers. Revolutions in computing such as the invention of the personal computer and the invention of the World Wide Web are considered.

We also discuss a selection of programming languages and the history of the software engineering field. Finally, we consider a selection of individuals who have made important contributions to the computing field. These include historical figures such as George Boole, Charles Babbage, Alan Turing, Claude Shannon and John von Neumann.

Organisation and Features

The first chapter discusses the contributions made by early civilisations to computing. This includes work done by the Babylonians, Egyptians and Greeks. Egyptians applied their mathematics to solving practical problems such as the construction of pyramids. The Greeks made a major contribution to mathematics and geometry, and most students are familiar with the work of Euclid.

Chapter 2 discusses the question as to what a computer is. Early mechanical calculators are considered, and the discussion then moves on to analog and digital computers. The evolution of the digital computer from vacuum tube technology to transistor and integrated circuit technology is considered.

Chapter 3 discusses a selection of early computers. These include Zuse's machines; the Atanasoff-Berry Computer; Colossus developed at Bletchley Park during the Second World War; the ENIAC, EDVAC and UNIVAC computers; the Manchester Mark 1; and the EDSAC and LEO computers developed in England.

Chapter 4 considers a selection of computers developed in the 1950s–1970s. These include the SAGE system designed as an air defence system for the United States, the IBM 360 family of computers, the PDP minicomputers developed by Digital and early home computers such as the Altair 8080, the Apple I and II and the Commodore PET.

Chapter 5 considers the revolutions in computing that took place in the 1980s and 1990s. This included the introduction of the IBM personal computer, the move from mainframes to networks of personal computers and client-server architecture and the development of mobile phone technology which transformed the existing communication paradigm of that between places to communication between people. Finally, the World Wide Web and its impact on society are discussed.

Chapter 6 considers the history of IBM from its evolution as a company producing tabulators to its position as the industry leader in the computing field.

Chapter 7 considers a selection of technology companies that have made important contributions to the computing field. These include Microsoft, Digital, Digital Research, Intel, Apple, Amdahl, HP, Oracle and Motorola.

Chapter 8 discusses the birth of the World Wide Web. Its invention by Tim Berners-Lee built upon existing technologies such as the Internet, mouse and hypertext to find a solution to the problem of keeping track of the work of visiting researchers at CERN. Its invention has revolutionised the computing field.

Chapter 9 considers a selection of various programming languages developed since the birth of the computing field. These include machine code languages that use actual machine instructions, assembly languages and high-level programming languages. The earliest high-level language developed was called Plankalkül, developed by the German engineer Zuse. The syntax and semantics of programming languages are discussed.

Chapter 10 considers the important field of software engineering. It discusses the birth of software engineering at the NATO conference at Garmisch in 1968. This conference discussed the crisis with software which included problems with projects being delivered late or with poor quality. The software engineering field is concerned with sound techniques to engineer high-quality software to meet customers' requirements.

Chapter 11 considers a selection of individuals who have made important contributions to the computing field. These include Zuse, von Neumann, Amdahl; Brooks, Knuth, Hoare, Dijkstra, Parnas, Berners-Lee, Stallman, Leibniz and Archimedes.

Chapter 12 considers foundational work done by Boole and Babbage in the nineteenth century. Boolean logic is fundamental to the working of all modern computers. Babbage did pioneering work on the difference engine as well as designing the analytic engine. The difference engine was essentially a mechanical calculator while the analytic engine was essentially the design of the world's first computer. Lady Ada Lovelace designed the first programs to run on the analytic engine, and she believed that the machine would be applicable to many disciplines.

Chapter 13 considers the contribution of Claude Shannon. His influential master's thesis showed how Boolean logic could be applied to simplify the design of circuits and is the foundation of digital computing. He also made important contributions to information theory and to cryptography.

Chapter 14 considers the contribution of Alan Turing. His work on computability showed that whatever was computable was computable by his theoretical Turing machine. He was involved with the code-breaking work at Bletchley Park during the Second World War, and he also made contributions to the artificial intelligence field.

Chapter 15 considers artificial intelligence which is a multidisciplinary field concerned with the problem of producing intelligence in machines. This chapter considers some of the key disciplines in the field including philosophy, psychology, linguistics, neural networks and so on. The Turing test was proposed by Alan Turing to judge whether a machine is intelligent. Searle's Chinese room argument is a rebuttal and argues that even if a machine passes the Turing Test, it still may not be considered intelligent.

Audience

This book is suitable for the general reader who is interested in the history of computing, and it will also be of interest to computer science students.

Acknowledgements

I am deeply indebted to friends and family who supported my efforts in this endeavour. I would like to express thanks to all copyright owners for permission to use their photos, and my thanks to Springer for their professional help.

I would like to express a special thanks to Dr. Robert L. Baber who read an earlier draft and for sending many helpful comments and suggestions. I am, of course, responsible for all remaining errors and omissions.

Cork, Ireland Gerard O'Regan

Contents

List of Figures

List of Tables

Chapter 1
Computing in Early Civilisations

Key Topics

Babylonian Mathematics
Egyptian Civilisation
Greek and Roman Civilisation
Counting and Numbers
Solving Practical Problems
Syllogistic Logic
Algorithms
Early Ciphers

1.1 Introduction

It is difficult to think of western society today without modern technology. The last decades of the twentieth century have witnessed a proliferation of high-tech computers, mobile phones, text messaging, the Internet and the World Wide Web. Software is now pervasive, and it is an integral part of automobiles, airplanes, televisions and mobile communication. The pace of change as a result of all this new technology has been extraordinary. Today consumers may book flights over the World Wide Web as well as keep in contact with family members in any part of the world via e-mail or mobile phone. In previous generations, communication often involved writing letters that took months to reach the recipient. Communication improved with the telegrams and the telephone in the late nineteenth century. Communication today is instantaneous with text messaging, mobile phones and e-mail, and the new generation probably views the world of their parents and grandparents as being old-fashioned.

The new technologies have led to major benefits[1] to society and to improvements in the standard of living for many citizens in the western world. It has also reduced the necessity for humans to perform some of the more tedious or dangerous manual tasks, as many of these may now be automated by computers. The increase in productivity due to the more advanced computerised technologies has allowed humans, at least in theory, the freedom to engage in more creative and rewarding tasks.

Early societies had a limited vocabulary for counting: for example, 'one, two, three, many' is associated with some primitive societies and indicates primitive computation and scientific ability. It suggests that there was no need for more sophisticated arithmetic in the primitive culture as the problems dealt with were elementary. These early societies would typically have employed their fingers for counting, and as humans have 5 fingers on each hand and 5 toes on each foot, then the obvious bases would have been 5, 10 and 20. Traces of the earlier use of the base 20 system are still apparent in modern languages such as English and French. This includes phrases such as 'three score' in English and '*quatre vingt*' in French.

The decimal system (base 10) is used today in western society, but the base 60 was common in computation *circa* 1,500 B.C. One example of the use of base 60 today is still evident in the subdivision of hours into 60 min and the subdivision of minutes into 60 s. The base 60 system (i.e. the sexagesimal system) is inherited from the Babylonians [Res:84]. The Babylonians were able to represent arbitrarily large numbers or fractions with just two symbols. The binary (base 2) and hexadecimal (base 16) systems play a key role in computing (as the machine instructions that computers understand are in binary code).

The achievements of some of these ancient societies were spectacular. The archaeological remains of ancient Egypt such as the pyramids at Giza and the temples of Karnak and Abu Simbal are impressive. These monuments provide an indication of the engineering sophistication of the ancient Egyptian civilisation. The objects found in the tomb of Tutankamun[2] are now displayed in the Egyptian museum in Cairo and demonstrate the artistic skill of the Egyptians.

The Greeks made major contributions to western civilisation including contributions to mathematics, philosophy, logic, drama, architecture, biology

[1] Of course, it is essential that the population of the world moves towards more sustainable development to ensure the long-term survival of the planet for future generations. This involves finding technological and other solutions to reduce greenhouse gas emissions as well as moving to a carbon neutral way of life. The solution to the environmental issues will be a major challenge for the twenty-first century.

[2] Tutankamun was a minor Egyptian pharaoh who reigned after the controversial rule of Akenaten. Tutankamun's tomb was discovered by Howard Carter in the Valley of the Kings, and the tomb was intact. The quality of the workmanship of the artefacts found in the tomb is extraordinary, and a visit to the Egyptian museum in Cairo is memorable.

and democracy.[3] The Greek philosophers considered fundamental questions such as ethics, the nature of being, how to live a good life and the nature of justice and politics. The Greek philosophers include Parmenides, Heraclitus, Socrates, Plato and Aristotle. The Greeks invented democracy, and their democracy was radically different from today's representative democracy.[4] The sophistication of Greek architecture and sculpture is evident from the Parthenon on the Acropolis and the Elgin marbles[5] that are housed today in the British Museum, London.

The Hellenistic[6] period commenced with Alexander the Great and led to the spread of Greek culture throughout most of the known world. The city of Alexandria became a centre of learning and knowledge during the Hellenistic period. Its scholars included Euclid who provided a systematic foundation for geometry. His work is known as 'The Elements', and consists of 13 books. The early books are concerned with the construction of geometric figures, number theory and solid geometry.

There are many words of Greek origin that are part of the English language. These include words such as psychology which is derived from two Greek words: *psyche* (ψυχε) and *logos* (λογος). The Greek word '*psyche*' means mind or soul, and the word '*logos*' means an account or discourse. Other examples are anthropology derived from '*anthropos*' (αντροπος) and '*logos*' (λογος).

The Romans were influenced by the Greek culture. The Romans built aqueducts, viaducts and amphitheatres. They also developed the Julian calendar, formulated laws (*lex*) and maintained peace throughout the Roman Empire (*pax Romano*). The ruins of Pompeii and Herculaneum demonstrate their engineering capability. Their numbering system is still employed in clocks and for page numbering

[3] The origin of the word 'democracy' is from demos (δημος), meaning people, and kratos (κρατος), meaning rule. That is, it means rule by the people. It was introduced into Athens following the reforms introduced by Cleisthenes. He divided the Athenian city state into 30 areas. Twenty of these areas were inland or along the coast and ten were in Attica itself. Fishermen lived mainly in the ten coastal areas, farmers in the ten inland areas and various tradesmen in Attica. Cleisthenes introduced ten new clans where the members of each clan came from one coastal area, one inland area on one area in Attica. He then introduced a Boule (or assembly) which consisted of 500 members (50 from each clan). Each clan ruled for $^1/_{10}$th of the year.

[4] The Athenian democracy involved the full participations of the citizens (i.e. the male adult members of the city state who were not slaves) whereas in representative democracy the citizens elect representatives to rule and represent their interests. The Athenian democracy was chaotic and could also be easily influenced by individuals who were skilled in rhetoric. There were teachers (known as the Sophists) who taught wealthy citizens rhetoric in return for a fee. The origin of the word 'sophist' is the Greek word σοφος, meaning wisdom. One of the most well known of the sophists was Protagorus. The problems with the Athenian democracy led philosophers such as Plato to consider alternate solutions such as rule by philosopher kings. This is described in Plato's Republic.

[5] The Elgin marbles are named after Lord Elgin who moved them from the Parthenon in Athens to London in 1806. The marbles show the Pan-Athenaic festival that was held in Athens in honour of the goddess Athena after whom Athens is named.

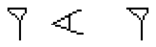
[6] The origin of the word Hellenistic is from Hellene ('Ελλην), meaning Greek.

in documents. However, it is cumbersome for serious computation. The collapse of the Roman Empire in Western Europe led to a decline in knowledge and learning in Europe. However, the eastern part of the Roman Empire continued at Constantinople until its sacking by the Ottomans in 1453.

1.2 The Babylonians

The Babylonian[7] civilisation flourished in Mesopotamia (in modern Iraq) from about 2,000 B.C. until about 300 B.C. Various clay cuneiform tablets containing mathematical texts were discovered and later deciphered in the nineteenth century [Smi:23]. These included tables for multiplication, division, squares, cubes and square roots and the measurement of area and length. Their calculations allowed the solution of a linear equation and one root of a quadratic equation to be determined. The late Babylonian period (c. 300 B.C.) includes work on astronomy.

They recorded their mathematics on soft clay using a wedge-shaped instrument to form impressions of the *cuneiform* numbers. The clay tablets were then baked in an oven or by the heat of the sun. They employed just two symbols (1 and 10) to represent numbers, and these symbols were then combined to form all other numbers. They employed a positional number system[8] and used the base 60 system. The symbol representing 1 could also (depending on the context) represent 60, 60^2, 60^3, etc. It could also mean $^1/_{60}$, $^1/_{3,600}$ and so on. There was no zero employed in the system and there was no decimal point (no 'sexagesimal point'), and therefore the context was essential.



The example above illustrates the cuneiform notation and represents the number 60 + 10 + 1 = 71. The Babylonians used the base 60 system for computation, and this base is still in use today in the division of hours into minutes and the division of minutes into seconds. One possible explanation for the use of the base 60 notation is the ease of dividing 60 into parts. It is divisible by 2, 3, 4, 5, 6, 10, 12, 15, 20 and 30. They were able to represent large and small numbers and had no difficulty in working with fractions (in base 60) and in multiplying fractions. The Babylonians maintained tables of reciprocals (i.e. $^1/_n$, $n = 1, \ldots 59$) apart from numbers like 7, 11, etc. which cannot be written as a finite sexagesimal expansion (i.e. 7, 11, etc. are not of the form $2^\alpha 3^\beta 5^\gamma$).

The modern sexagesimal notation [Res:84] 1;24,51,10 represents the number $1 + {}^{24}/_{60} + {}^{51}/_{3,600} + {}^{10}/_{216,000} = 1 + 0.4 + 0.0141666 + 0.0000462 = 1.4142129$.

[7] The hanging gardens of Babylon were one of the seven wonders of the ancient world.

[8] A positional numbering system is a number system where each position is related to the next by a constant multiplier. The decimal system is an example, for example $546 = 5 * 10^2 + 4 * 10^1 + 6$.

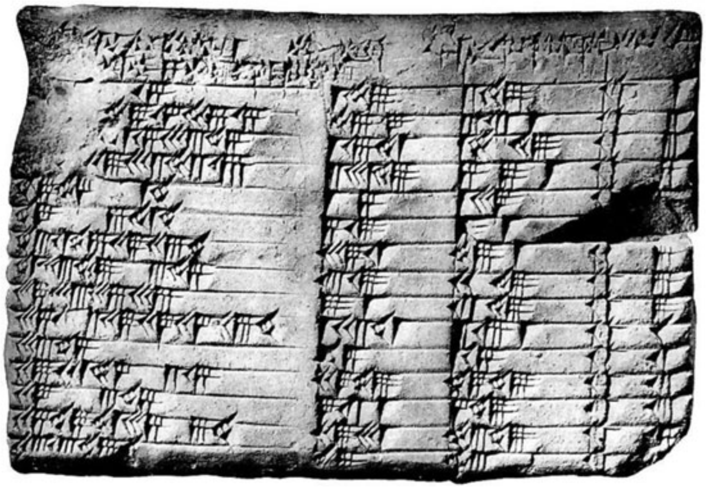


Fig. 1.1 The Plimpton 322 tablet

This is the Babylonian representation of the square root of 2. They performed multiplication as follows: for example consider 20 * sqrt 2 = (20) * (1;24,51,10):

$$20 * 1 = 20$$
$$20 *; 24 = 20 * {}^{24}\!/_{60} = 8$$
$$20 * {}^{51}\!/_{3{,}600} = {}^{51}\!/_{180} = {}^{17}\!/_{60} = ;17$$
$$20 * {}^{10}\!/_{216{,}000} = {}^{3}\!/_{3{,}600} + {}^{20}\!/_{216{,}000} = ;0{,}3{,}20.$$

Hence, the product 20 * sqrt 2 = 20; + 8; + ;17 + ;0,3,20 = 28;17,3,20.

The Babylonians appear to have been aware of Pythagoras's theorem about 1,000 years before the time of Pythagoras. The Plimpton 322 tablet records various Pythagorean triples, that is triples of numbers (a, b, c) where $a^2 + b^2 = c^2$. It dates from approximately 1,700 B.C. (Fig. 1.1).

They developed algebra to assist with problem-solving, and their algebra allowed problems involving length, breadth and area to be discussed and solved. They did not employ notation for representation of unknown values (e.g. let x be the length and y be the breadth), and instead they used words like 'length' and 'breadth'. They were familiar with and used square roots in their calculations, and they were familiar with techniques that allowed one root of a quadratic equation to be solved.

They were familiar with various mathematical identities such as $(a + b)^2 = (a^2 + 2ab + b^2)$ as illustrated geometrically in Fig. 1.2. They also worked on astronomical problems, and they had mathematical theories of the cosmos to make predictions of when eclipses and other astronomical events would occur. They were also interested in astrology, and they associated various deities with the

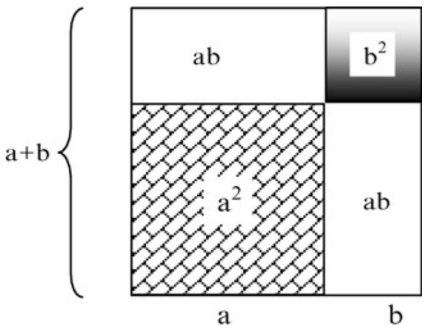


Fig. 1.2 Geometric representation of $(a + b)^2 = (a^2 + 2ab + b^2)$

heavenly bodies such as the planets, as well as the sun and moon. They associated various clusters of stars with familiar creatures such as lions, goats and so on.

The Babylonians used counting boards to assist with counting and simple calculations. A counting board is an early version of the abacus and was usually made of wood or stone. The counting board contained grooves which allowed beads or stones could be moved along the groove. The abacus differed from counting boards in that the beads in abaci contained holes that enabled them to be placed in a particular rod of the abacus.

1.3 The Egyptians

The Egyptian civilisation developed along the Nile from about 4,000 B.C. and the pyramids were built around 3,000 B.C. They used mathematics to solve practical problems such as measuring time, measuring the annual Nile flooding, calculating the area of land, bookkeeping and accounting and calculating taxes. They developed a calendar circa 4,000 B.C. which consisted of 12 months with each month having 30 days. There were then five extra feast days to give 365 days in a year. Egyptian writing commenced around 3,000 B.C. and is recorded on the walls of temples and tombs.[9] A reed-like parchment termed 'papyrus' was used for writing, and three Egyptian writing scripts were employed. These were Hieroglyphics, the Hieratic script and the Demotic script.

Hieroglyphs are little pictures and are used to represent words, alphabetic characters as well as syllables or sounds. The deciphering of Hieroglyphics was done by Champollion with his work on the Rosetta stone. This was discovered during the Napoleonic campaign in Egypt, and it is now in the British Museum in London. It contains three scripts: Hieroglyphics, Demotic script and Greek. The key to its decipherment was that the Rosetta stone contained just one name

[9] The decorations of the tombs in the Valley of the Kings record the life of the pharaoh including his exploits and successes in battle.

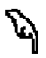
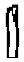
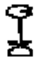
					
100,000	10,000	1000	100	10	1

Fig. 1.3 Egyptian numerals

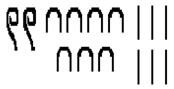


Fig. 1.4 Egyptian representation of a number

'Ptolemy' in the Greek text, and this was identified with the hieroglyphic characters in the cartouche[10] of the Hieroglyphics. There was just one cartouche on the Rosetta stone, and Champollion inferred that the cartouche represented the name 'Ptolemy'. He was familiar with another multilingual object which contained two names in the cartouche. One he recognised as Ptolemy and the other he deduced from the Greek text as 'Cleopatra'. This led to the breakthrough in the translation of the Hieroglyphics [Res:84].

The Rhind Papyrus is a famous Egyptian papyrus on mathematics. It was purchased by the Scottish Egyptologist, Henry Rhind, in 1858, and it is a copy created by an Egyptian scribe called Ahmose.[11] It is believed to date to 1,832 B.C. It contains examples of many kinds of arithmetic and geometric problems, and it may have been used by students as a textbook to develop their mathematical knowledge. This would allow them to participate in the pharaoh's building programme.

The Egyptians were familiar with geometry, arithmetic and elementary algebra. They had techniques to find solutions to problems with one or two unknowns. A base 10 number system was employed separate with symbols for one, ten, a hundred, a thousand, a ten thousand, a hundred thousand and so on. These hieroglyphic symbols are represented in Fig. 1.3:

For example, the representation of the number 276 in Egyptian Hieroglyphics is given by Fig. 1.4.

The addition of two numerals is straightforward and involves adding the individual symbols, and where there are ten copies of a symbol, it is then replaced by a single symbol of the next higher value. The Egyptian employed unit fractions (e.g. $1/n$ where n is an integer). These were represented in hieroglyphs by placing the symbol representing a 'mouth' above the number. The symbol 'mouth' represents part of. For example, the representation of the number $^1/_{276}$ is given by Fig. 1.5.

[10] The cartouche surrounded a group of hieroglyphic symbols enclosed by an oval shape. Champollion's insight was that the group of hieroglyphic symbols represented the name of the Ptolemaic pharaoh 'Ptolemy'.

[11] The Rhind papyrus is sometimes referred to as the Ahmes papyrus in honour of the scribe who wrote it in 1832 B.C.

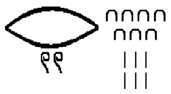


Fig. 1.5 Egyptian representation of a fraction

The problems on the papyrus included the determination of the angle of the slope of the pyramid's face. They were familiar with trigonometry, including sine, cosine, tangent and cotangent, and knew how to build right angles into their structures by using the ratio 3:4:5. The papyrus also considered problems such as the calculation of the number of bricks required for part of a building project. Multiplication and division was cumbersome in Egyptian mathematics as they could only multiply and divide by 2.

Suppose they wished to multiply a number n by 7. Then $n * 7$ is determined by $n * 2 + n * 2 + n * 2 + n$. Similarly, if they wished to divide 27 by 7, they would note that $7 * 2 + 7 = 21$ and that $27 - 21 = 6$ and that therefore the answer was $3 \cdot {}^{6}/_{7}$. Egyptian mathematics was cumbersome, and the writing of their mathematics was long and repetitive. For example, they wrote a number such as 22 by 10 + 10 + 1 + 1.

The Egyptians calculated the approximate area of a circle by calculating the area of a square ${}^{8}/_{9}$ of the diameter of a circle. That is, instead of calculating the area in terms of our familiar πr^2, their approximate calculation yielded $({}^{8}/_{9} * 2r)^2 = {}^{256}/_{81}\, r^2$ or 3.16 r^2. Their approximation of π was ${}^{256}/_{81}$ or 3.16. They were able to calculate the area of a triangle and volumes. The Moscow papyrus includes a problem to calculate the volume of the frustum. The formula for the volume of a frustum of a square pyramid[12] was given by $V = {}^{1}/_{3}\, h(b_1^{\,2} + b_1 b_2 + b_2^{\,2})$, and when b_2 is 0, then the well-known formula for the volume of a pyramid is given, that is ${}^{1}/_{3}\, hb_1^{\,2}$.

1.4 The Greeks

The Greeks made major contributions to western civilisation including mathematics, logic, astronomy, philosophy, politics, drama and architecture. The Greek world of 500 B.C. consisted of several independent city states, such as Athens and Sparta, and various city states in Asia Minor. The Greek polis (πολισ) or city state tended to be quite small and consisted of the Greek city and a certain amount of territory outside the city state. Each city state had political structures for its citizens, and these varied from city state to city state. Some were oligarchs where political power was maintained in the hands of a few individuals or aristocratic families. Others were ruled by tyrants (or sole rulers) who sometimes took power by force, but who often had a lot of support from the public. The tyrants included people such as Solon, Peisistratus and Cleisthenes in Athens.

[12] The length of a side of the bottom base of the pyramid is $b_1$and the length of a side of the top base is b_2.

The reforms by Cleisthenes led to the introduction of the Athenian democracy. Power was placed in the hands of the citizens who were male (women or slaves did not participate). It was an extremely liberal democracy where citizens voted on all important issues. Often, this led to disastrous results as speakers who were skilled in rhetoric could exert significant influence. This led to Plato to advocate rule by philosopher kings rather than by democracy.

Early Greek mathematics commenced approximately 600–500 B.C. with work done by Pythagoras and Thales. Pythagoras was a philosopher and mathematician who had spent time in Egypt becoming familiar with Egyptian mathematics. He lived on the island of Samoa and formed a secret society known as the Pythagoreans. They included men and women and believed in the transmigration of souls and that number was the essence of all things. They discovered the mathematics for harmony in music with the relationship between musical notes being expressed in numerical ratios of small whole numbers. Pythagoras is credited with the discovery of Pythagoras's theorem, although this theorem was probably known by the Babylonians about 1,000 years earlier. The Pythagorean society was dealt a major blow[13] by the discovery of the incommensurability of the square root of 2, that is there are no numbers p, q such that $\surd 2 = {}^{p}/_{q}$.

Thales was a sixth-century (B.C.) philosopher from Miletus in Asia Minor who made contributions to philosophy, geometry and astronomy. His contributions to philosophy are mainly in the area of metaphysics, and he was concerned with questions on the nature of the world. His objective was to give a natural or scientific explanation of the cosmos, rather than relying on the traditional supernatural explanation of creation in Greek mythology. He believed that there was single substance that was the underlying constituent of the world, and he believed that this substance was water.

He also contributed to mathematics [AnL:95], and a well-known theorem in Euclidean geometry is named after him. It states that if A, B and C are points on a circle, and where the line AC is a diameter of the circle, then the angle $<ABC$ is a right angle.

The rise of Macedonia led to the Greek city states being conquered by Philip of Macedonia in the fourth century B.C. His son, Alexander the Great, defeated the Persian Empire, and extended his empire to include most of the known world. This led to the Hellenistic Age with Greek language and culture spread to the known world. The city of Alexandra was founded by Alexander, and it became a major centre of learning. However, Alexander's reign was very short as he died at the young age of 33 in 323 B.C.

Euclid lived in Alexandria during the early Hellenistic period. He is considered the father of geometry and the deductive method in mathematics. His systematic

[13] The Pythagoreans took a vow of silence with respect to the discovery of incommensurable numbers. However, one member of the society is said to have shared the secret result with others outside the sect, and an apocryphal account is that he was thrown into a lake for his betrayal and drowned. The Pythagoreans obviously took mathematics seriously back then.

treatment of geometry and number theory is published in the 13 books of the Elements [Hea:56]. It starts from 5 axioms, 5/1 postulates and 23 definitions to logically derive a comprehensive set of theorems. His method of proof was often constructive in that as well as demonstrating the truth of a theorem the proof would often include the construction of the required entity. He also used indirect proof to show that there are an infinite number of primes:

1. Suppose there is a finite number of primes (say n primes).
2. Multiply all n primes together and add 1 to form N:

$$(N = p_1 * p_2 * \ldots * p_n + 1).$$

1. N is not divisible by p_1, p_2, …, p_n as dividing by any of these gives a remainder of 1.
2. Therefore, N must either be prime or divisible by some other prime that was not included in the list.
3. Therefore, there must be at least n + 1 primes.
4. This is a contradiction as it was assumed that there was a finite number of primes n.
5. Therefore, the assumption that there is a finite number of primes is false.
6. Therefore, there are an infinite number of primes.

Euclidean geometry included the parallel postulate or Euclid's fifth postulate. This postulate generated interest as many mathematicians believed that it was unnecessary and could be proved as a theorem. It states that:

Definition 1.1 (Parallel Postulate). *If a line segment intersects two straight lines forming two interior angles on the same side that sum to less than two right angles, then the two lines, if extended indefinitely, meet on that side on which the angles sum to less than two right angles.*

This postulate was later proved to be independent of the other postulates with the development of non-Euclidean geometries in the nineteenth century. These include the hyperbolic geometry discovered independently by Bolyai and Lobachevsky and elliptic geometry developed by Riemann. The standard model of Riemannian geometry is the sphere where lines are great circles.

The material in the Euclid's Elements is a systematic development of geometry starting from the small set of axioms, postulates and definitions, leading to theorems logically derived from the axioms and postulates. Euclid's deductive method influenced later mathematicians and scientists. There are some jumps in reasoning and Hilbert later added extra axioms to address this.

The Elements contains many well-known mathematical results such as Pythagoras's theorem, Thales Theorem, Sum of Angles in a Triangle, Prime Numbers, Greatest Common Divisor and Least Common Multiple, Euclidean Algorithm, Areas and Volumes, Tangents to a point and Algebra.

The Euclidean algorithm is one of the oldest known algorithms and is employed to produce the greatest common divisor of two numbers. It is presented in the

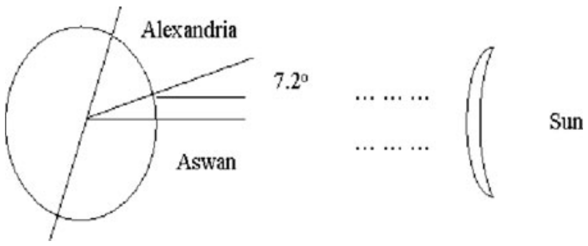


Fig. 1.6 Eratosthenes measurement of the circumference of the earth

Elements but was known well before Euclid. The algorithm to determine the gcd of two natural numbers, a and b, is given by:

1. Check if b is zero. If so, then a is the gcd.
2. Otherwise, the gcd (a,b) is given by gcd $(b, a \bmod b)$.

It is also possible to determine integers p and q such that $ap + bq = \gcd(a, b)$.

The proof of the Euclidean algorithm is as follows. Suppose a and b are two positive numbers whose gcd has to be determined, and let r be the remainder when a is divided by b:

1. Clearly $a = qb + r$ where q is the quotient of the division.
2. Any common divisor of a and b is also a divisor or r (since $r = a - qb$).
3. Similarly, any common divisor of b and r will also divide a.
4. Therefore, the greatest common divisor of a and b is the same as the greatest common divisor of b and r.
5. The number r is smaller than b, and we will reach $r = 0$ in finitely many steps.
6. The process continues until $r = 0$.

Comment 1.1. *Algorithms are fundamental in computing as they define the procedure by which a problem is solved. A computer program implements the algorithm in some programming language.*

Eratosthenes was a Hellenistic mathematician and scientist who worked in the famous library in Alexandria. This ancient library was once the largest library in the world. It was build during the Hellenistic period in the third century B.C. and destroyed by fire in 391 A.D.

Eratosthenes devised a system of latitude and longitude and became the first person to estimate of the size of the circumference of the earth. His calculation proceeded as follows (Fig. 1.6):

1. On the summer solstice at noon in the town of Aswan[14] on the Tropic of Cancer in Egypt the sun appears directly overhead.
2. Eratosthenes believed that the earth was a sphere.

[14] The town of Aswan is famous today for the Aswan high dam which was built in the 1960s. There was an older Aswan dam built by the British in the late nineteenth century. The new dam led to a rise in the water level of Lake Nasser and flooding of archaeological sites along the Nile. Several archaeological sites such as Abu Simbel and the temple of Philae were relocated to higher ground.

3. He assumed that rays of light came from the sun in parallel beams and reached the earth at the same time.
4. At the same time in Alexandria, he had measured that the sun would be 7.2° south of the zenith.
5. He assumed that Alexandria was directly north of Aswan.
6. He concluded that the distance from Alexandria to Aswan was $^{7.2}/_{360}$ of the circumference of the earth.
7. Distance between Alexandria and Aswan was 5,000 stadia (approximately 800 km).
8. He established a value of 252,000 stadia or approximately 396,000 km.

Eratosthenes's calculation was an impressive result for 200 B.C. The errors in his calculation were due to:

1. Aswan is not exactly on the Tropic of Cancer but it is actually 55 km north of it.
2. Alexandria is not exactly north of Aswan, and there is a difference of 3° longitude.
3. The distance between Aswan and Alexandria is 729 km, not 800 km.
4. Angles in antiquity could not be measured with a high degree of precision.
5. The angular distance is actually 7.08° and not 7.2°.

Eratosthenes also calculated the approximate distance to the moon and sun, and he also produced maps of the known world. He developed a very useful algorithm for determining all of the prime numbers up to a specified integer. The method is known as the Sieve of Eratosthenes, and the steps are as follows:

1. Write a list of the numbers from 2 to the largest number that you wish to test for primality. This first list is called A.
2. A second list B is created to list the primes. It is initially empty.
3. The number 2 is the first prime number and is added to the list of primes in B.
4. Strike off (or remove) 2 and all multiples of 2 from List A.
5. The first remaining number in List A is a prime number, and this prime number is added to List B.
6. Strike off (or remove) this number and all multiples of this number from List A.
7. Repeat steps 5–7 until no more numbers are left in List A.

Comment 1.2. *The Sieve of Eratosthenes method is a well-known algorithm for determining prime numbers.*

Archimedes was a Hellenistic mathematician, astronomer and philosopher who lived in Syracuse in the third century B.C. He discovered the law of buoyancy known as Archimedes' principle:

The buoyancy force is equal to the weight of the displaced fluid.

He is believed to have discovered the principle while sitting in his bath. He was so overwhelmed with his discovery that he rushed out onto the streets of Syracuse shouting 'Eureka', but forgot to put on his clothes to announce the discovery.

The weight of the displaced liquid will be proportional to the volume of the displaced liquid. Therefore, if two objects have the same mass, the one with greater volume (or smaller density) has greater buoyancy. An object will float if its buoyancy force (i.e. the weight of liquid displaced) exceeds the downward force of gravity (i.e. its weight). If the object has exactly the same density as the liquid, then it will stay still, neither sinking nor floating upwards.

For example, a rock is generally a very dense material and will generally not displace its own weight. Therefore, a rock will sink to the bottom as the downward weight exceeds the buoyancy weight. However, if the weight of the object is less than the liquid it would displace, then it floats at a level where it displaces the same weight of liquid as the weight of the object.

Archimedes also made good contributions to mathematics including a good approximation to π, contributions to the positional numbering system, geometric series, and to physics. He also solved several interesting problems, for example the calculation of the composition of cattle in the herd of the Sun god by solving a number of simultaneous Diophantine equations. The herd consisted of bulls and cows with one part of the herd consisting of white, second part black, third spotted and the fourth brown. Various constraints were then expressed in Diophantine equations, and the problem was to determine the precise composition of the herd. Diophantine equations are named after Diophantus who worked on number theory in the third century.

Archimedes also calculated an upper bound of the number of grains of sands in the known universe. The largest number in common use at the time was a myriad myriad (100 million), where a myriad is 10,000. Archimedes' numbering system goes up to $8 * 10^{16}$, and he also developed the laws of exponents, that is $10^a 10^b = 10^{a+b}$. His calculation of the upper bound includes not only the grains of sand on each beach but on the earth filled with sand and the known universe filled with sand. His final estimate of the upper bound for the number of grains of sand in a filled universe was 10^{64}.

It is possible that he may have developed the odometer,[15] and this instrument could calculate the total distance travelled on a journey. An odometer is described by the Roman engineer Vitruvius around 25 B.C. It employed a wheel with a diameter of 4 ft, and the wheel turned 400 times in every mile.[16] The device included gears and pebbles and a 400-tooth cogwheel that turned once every mile and caused one pebble to drop into a box. The total distance travelled was determined by counting the pebbles in the box.

Aristotle was born in Macedonia and became a student of Plato in Athens. Plato had founded a school (known as Plato's academy) in Athens in the fourth century B.C., and this school remained open until 529 A.D. Aristotle founded his own school (known as the Lyceum) in Athens. He was also the tutor of

[15] The origin of the word 'odometer' is from the Greek words 'οδοζ' (meaning journey) and 'μετρον' meaning (measure).

[16] The figures given here are for the distance of one Roman mile. This is given by $\pi 2^2 * 400 = 12.56 * 400 = 5{,}024$ (which is less than 5,280 ft for a standard mile in the Imperial system).

Fig. 1.7 Plato and Aristotle

Alexander the Great. He made contributions to physics, biology, logic, politics, ethics and metaphysics.

Aristotle's starting point to the acquisition of knowledge was the senses. He believed that the senses were essential to acquire knowledge. This position is the opposite from Plato who argued that the senses deceive and should not be relied upon. Plato's writings are mainly in dialogues involving his former mentor Socrates (Fig. 1.7).[17]

Aristotle made important contributions to formal reasoning with his development of syllogistic logic. His collected works on logic is called the Organon, and it was used in his school in Athens. Syllogistic logic (also known as term logic) consists of reasoning with two premises and one conclusion. Each premise consists

[17] Socrates was a moral philosopher who deeply influenced Plato. His method of enquiry into philosophical problems and ethics was by questioning. Socrates himself maintained that he knew nothing (Socratic ignorance). However, from his questioning, it became apparent that those who thought they were clever were not really that clever after all. His approach obviously would not have made him very popular with the citizens of Athens. Socrates had consulted the oracle at Delphi to find out who was the wisest of all men, and he was informed that there was no one wiser than him. Socrates was sentenced to death for allegedly corrupting the youth of Athens, and the sentence was carried out by Socrates being forced to take hemlock (a type of poison). The juice of the hemlock plant was prepared for Socrates to drink.

Table 1.1 Syllogisms: relationship between terms

Relationship	Abbreviation
Universal affirmation	A
Universal negation	E
Particular affirmation	I
Particular negation	O

of two terms, and there is a common middle term. The conclusion links the two unrelated terms from the premises. For example:

Premise 1	All Greeks are mortal.
Premise 2	Socrates is a Greek.
	- - - - - - - - - - - - - -
Conclusion	Socrates is mortal.

The common middle term is 'Greek', which appears in the two premises. The two unrelated terms from the premises are 'Socrates' and 'Mortal'. The relationship between the terms in the first premise is that of the universal, that is, anything or any person that is a Greek is mortal. The relationship between the terms in the second premise is that of the particular, that is, Socrates is a person that is a Greek. The conclusion from the two premises is that Socrates is mortal, that is a particular relationship between the two unrelated terms 'Socrates' and 'Mortal'.

The syllogism above is a valid syllogistic argument. Aristotle studied the various possible syllogistic arguments and determined those that were valid and invalid. There are several candidate relationships that may exist between the terms in a premise, and these are defined in Table 1.1. In general, a syllogistic argument will be of the form:

$$\begin{array}{c} \text{S}\ x\ \text{M} \\ \text{M}\ y\ \text{P} \\ \text{- - - -} \\ \text{S}\ z\ \text{P} \end{array}$$

where x, y, z may be universal affirmation, universal negation, particular affirmation and particular negation. Syllogistic logic is described in more detail in [ORg:06]. Aristotle's work was highly regarded in classical and mediaeval times, and Kant believed that there was nothing else to invent in logic. There was another competing system of logic proposed by the stoics in Hellenistic times, that is an early form of propositional logic that was developed by Chrysippus[18] in the third century B.C. Aristotelian logic is mainly of historical interest today.

[18] Chrysippus was the head of the Stoics in the third century B.C.

Aquinas,[19] a thirteenth-century Christian theologian and philosopher, was deeply influenced by Aristotle and referred to him as the philosopher. Aquinas was an empiricist (i.e. he believed that all knowledge was gained by sense experience), and he used some of Aristotle's arguments to offer five proofs of the existence of God. These arguments included the cosmological argument and the design argument. The cosmological argument used Aristotle's ideas on the scientific method and causation. Aquinas argued that there was a first cause, and he deduced that this first cause is God:

1. Every effect has a cause.
2. Nothing can cause itself.
3. A causal chain cannot be of infinite length.
4. Therefore, there must be a first cause.

The Antikythera [Pri:59] was an ancient mechanical device that is believed to have been designed to calculate astronomical positions. It was discovered in 1902 in a wreck off the Greek island of Antikythera and dates from about 80 B.C. It is one of the oldest known geared devices, and it is believed that it was used for calculating the position of the sun, moon, stars and planets for a particular date entered.

The Romans appear to have been aware of a device similar to the Antikythera that was capable of calculating the position of the planets. The island of Antikythera was well known in the Greek and Roman period for its displays of mechanical engineering.

1.5 The Romans

Rome is said to have been founded[20] by Romulus and Remus about 750 B.C. Early Rome covered a small part of Italy, but it gradually expanded in size and importance. It destroyed Carthage[21] in 146 B.C. to become the major power in the Mediterranean. The Hellenistic world was colonised by the Romans, and the Romans were influenced by Greek culture and mathematics. Julius Caesar conquered the Gauls in 58 B.C. (Fig. 1.8).

[19] Aquinus's (or St. Thomas's) most famous work is Sumna Theologicae.

[20] The Aenid by Virgil suggests that the Romans were descended from survivors of the Trojan war and that Aeneas brought surviving Trojans to Rome after the fall of Troy.

[21] Carthage was located in Tunisia, and the wars between Rome and Carthage are known as the Punic wars. Hannibal was one of the great Carthaginan military commanders, and during the second Punic war, he brought his army to Spain, marched through Spain and crossed the Pyrnees. He then marched along southern France and crossed the Alps into Northern Italy. His army also consisted of war elephants. Rome finally defeated Carthage and destroyed the city.

Fig. 1.8 Julius Caesar

Fig. 1.9 Roman numbers

I = 1
V = 5
X = 10
L = 50
C = 100
D = 500
M = 1000

The Gauls consisted of several disunited Celtic[22] tribes. Vercingetorix succeeded in uniting them, but he was defeated by at the siege of Alesia in 52 B.C. (Fig. 1.9).

The Roman number system uses letters to represent numbers, and a number consists of a sequence of letters. The evaluation rules specify that if a number follows a smaller number, then the smaller number is subtracted from the larger number, for example IX represents 9 and XL represents 40. Similarly, if a smaller number followed a larger number, they were generally added, for example MCC represents 1,200. They had no zero in their system.

The use of Roman numerals was cumbersome in calculation, and an abacus was often employed. An abacus is a device that is usually of wood and has a frame that

[22] The Celtic period commenced around 1000 B.C. in Hallstaat (near Salzburg in Austria). The Celts were skilled in working with Iron and Bronze, and they gradually expanded into Europe. They eventually reached Britain and Ireland around 600 B.C. The early Celtic period was known as the ‘Hallstaat period’, and the later Celtic period is known as ‘La Téne’. The later La Téne period is characterised by the quality of ornamentation produced. The Celtic museum in Hallein in Austria provides valuable information and artefacts on the Celtic period. The Celtic language would have similarities to the Irish language. However, the Celts did not employ writing, and the Ogham writing used in Ireland was developed in the early Christian period.

Fig. 1.10 Caesar cipher

Alphabet Symbol	abcde	fghij	klmno	pqrst	uvwxyz
Cipher Symbol	dfegh	ijklm	nopqr	stuvw	xyzabc

holds rods with freely sliding beads mounted on them. It is used as a tool to assist calculation, and it is useful for keeping track of the sums and the carries of calculations.

It consists of several columns in which beads or pebbles are placed. Each column represented powers of 10, that is 10^0, 10^1, 10^2, 10^3, etc. The column to the far right represents one, the column to the left 10, next column to the left 100 and so on. Pebbles[23] (calculi) were placed in the columns to represent different numbers: for example, the number represented by an abacus with 4 pebbles on the far right, 2 pebbles in the column to the left and 3 pebbles in the next column to the left is 324. Calculations were performed by moving pebbles from column to column.

Merchants introduced a set of weights and measures (including the *libra* for weights and the *pes* for lengths). They developed an early banking system to provide loans for business and commenced minting money about 290 B.C. The Romans also made contributions to calendars, and the Julian calendar was introduced in 45 B.C. by Julius Caesar. It has a regular year of 365 days divided into 12 months, and a leap day is added to February every 4 years. It remained in use up to the twentieth century but has since been replaced by the Gregorian calendar. The problem with the Julian calendar is that too many leap years are added over time. The Gregorian calendar was first introduced in 1582.

The Romans employed the mathematics that had been developed by the Greeks. Caesar employed a substitution cipher on his military campaigns to enable important messages to be communicated safely. It involves the substitution of each letter in the plaintext (i.e. the original message) by a letter a fixed number of positions down in the alphabet. For example, a shift of three positions causes the letter B to be replaced by E, the letter C by F and so on. It is easily broken, as the frequency distribution of letters may be employed to determine the mapping. The cipher is defined as (Fig. 1.10):

The process of enciphering a message (i.e. plaintext) involves looking up each letter in the plaintext and writing down the corresponding cipher letter. For example the encryption of 'summer solstice' involves:

Plaintext:	Summer Solstice
Cipher Text	vxpphu vrovwleh

The decryption involves the reverse operation: that is, for each cipher letter, the corresponding plaintext letter is identified from the table.

[23] The origin of the word 'Calculus' is from Latin and means a small stone or pebble used for counting.

Cipher Text	vxpphu vrovwleh
Plaintext:	Summer Solstice

The encryption may also be represented using modular arithmetic. The numbers 0–25 represent the alphabet letters, and addition (modulo 26) is used to perform the encryption. The encoding of the plaintext letter x is given by:

$$c = x + 3(\mathrm{mod}\ 26).$$

Similarly, the decoding of a cipher letter represented by the number c is given by:

$$x = c - 3(\mathrm{mod}\ 26)$$

The emperor Augustus[24] employed a similar substitution cipher (with a shift key of 1). The Caesar cipher remained in use up to the early twentieth century. However, by then frequency analysis techniques were available to break the cipher. The Vignère cipher uses a Caesar cipher with a different shift at each position in the text. The value of the shift to be employed with each plaintext letter is defined using a repeating keyword.

1.6 Islamic Influence

Islamic mathematics refers to mathematics developed in the Islamic world from the birth of Islam in the early seventh century up until the seventeenth century. The Islamic world commenced with Mohammed in Mecca and spread throughout the Middle East, North Africa and Spain. Islamic scholars translated the works of the Greeks into Arabic, and this led to the preservation of the Greek texts during the Dark Ages in Europe. The Islamic scholars developed the existing mathematics further.

The Moors[25] invaded Spain in the eighth century A.D., and they ruled large parts of the Iberian Peninsula for several centuries. The Moorish influence[26] in Spain

[24] Augustus was the first Roman emperor, and his reign ushered in a period of peace and stability following the bitter civil wars. He was the adopted son of Julius Caesar and was called Octavion before he became emperor. The earlier civil wars were between Caesar and Pompey, and following Caesar's assassination, civil war broke out between Mark Anthony and Octavion. Octavion defeated Anthony and Cleopatra at the battle of Actium.

[25] The origin of the word 'Moor' is from the Greek work μυοροζ meaning very dark. It referred to the fact that many of the original Moors who came to Spain were from Egypt, Tunisia and other parts of North Africa.

[26] The Moorish influence includes the construction of various castles (*alcazar*), fortresses (*alcalzaba*) and mosques. One of the most striking Islamic sites in Spain is the palace of Alhambra in Granada, and this site represents the zenith of Islamic art.

continued until the time of the Catholic Monarchs[27] in the fifteenth century. Ferdinand and Isabella united Spain, defeated the Moors and expelled them from the country.

Islamic mathematicians and scholars were based in several countries including the Middle East, North Africa and Spain. Early work commenced in Baghdad, and the mathematicians were influenced by the work of Hindu mathematicians who had introduced the decimal system and decimal numerals. This system was adopted by Al-Khwarizmi[28] in the ninth century, and the resulting system is known as the Hindu–Arabic number system.

Many caliphs (Muslim rulers) were enlightened and encouraged scholarship in mathematics and science. This led to the translation of the existing Greek texts, and a centre of translation and research was set up in Baghdad. This included the works of Euclid, Archimedes, Apollonius and Diophantus. Al-Khwarizmi made contributions to early classical algebra, and the word algebra comes from the Arabic word '*al jabr*' that appears in a textbook by Al-Khwarizmi.

The Islamic contribution to algebra was an advance on the achievements of the Greeks. They developed a broader theory that treated rational and irrational numbers as algebraic objects and moved away from the Greek concept of mathematics as being essentially geometry. Later Islamic scholars applied algebra to arithmetic and geometry. This included contributions to reduce geometric problems such as duplicating the cube to algebraic problems. Eventually this led to the use of symbols in the fifteenth century such as:

$$x^n \cdot x^m = x^{m+n}.$$

The poet, Omar Khayyam, was also a mathematician.[29] He did work on the classification of cubic equations with geometric solutions. Others applied algebra to geometry and aimed to study curves by using equations. Other scholars made contributions to the theory of numbers, for example a theorem that allows pairs of amicable numbers to be found. Amicable numbers are two numbers such that each is the sum of the proper divisors of the other. They were aware of Wilson's theory in number theory, that is for p prime then p divides $(p - 1)! + 1$.

Moorish Spain became a centre of learning, with Islamic and other scholars coming to study at its universities. Many texts on Islamic mathematics were translated from Arabic into Latin, and these were invaluable in the renaissance in European learning and mathematics from the thirteenth century.

[27] The Catholic Monarchs refer to Ferdinand of Aragon and Isabella of Castille who married in 1469. They captured Granada (the last remaining part of Spain controlled by the Moors) in 1492.

[28] The origin of the word 'algorithm' is from the name of the Islamic scholar Al-Khwarizmi.

[29] I am aware of no other mathematician who was also a poet.

1.7 Chinese and Indian Mathematics

The development of mathematics commenced in China about 1,000 B.C. and was independent of developments in other countries. The emphasis was on problem-solving rather than on conducting formal proofs. It was concerned with finding the solution to practical problems such as the calendar, the prediction of the positions of the heavenly bodies, land measurement, conducting trade and the calculation of taxes.

The Chinese employed counting boards as mechanical aids for calculation from the fourth century B.C. These are similar to abaci and are usually made of wood or metal and contained carved grooves between which beads, pebbles or metal discs were moved.

Early Chinese mathematics was written on bamboo strips and included work on arithmetic and astronomy. The Chinese method of learning and calculation in mathematics was learning by analogy. This involves a person acquiring knowledge from observation of how a problem is solved, and then applying this knowledge for problem-solving to similar kinds of problems.

They had their version of Pythagoras's theorem and applied it to practical problems. They were familiar with the Chinese remainder theorem, the formula for finding the area of a triangle, as well as showing how polynomial equations (up to degree 10) could be solved. They showed how geometric problems could be solved by algebra, how roots of polynomials could be solved, how quadratic and simultaneous equations could be solved and how the area of various geometric shapes such as rectangles, trapezia and circles could be computed. Chinese mathematicians were familiar with the formula to calculate the volume of a sphere. The best approximation that the Chinese had to π was 3.14159, and this was obtained by approximations from inscribing regular polygons with 3×2^n sides in a circle.

The Chinese made contributions to number theory including the summation of arithmetic series and solving simultaneous congruences. The Chinese remainder theorem deals with finding the solutions to a set of simultaneous congruences in modular arithmetic. Chinese astronomers made accurate observations which were used to produce a new calendar in the sixth century. This was known as the Taming calendar, and it was based on a cycle of 391 years.

Indian mathematicians have made important contributions such as the development of the decimal notation for numbers that is now used throughout the world. This was developed in India sometime between 400 B.C. and 400 A.D. Indian mathematicians also invented zero and negative numbers and also did early work on the trigonometric functions of sine and cosine The knowledge of the decimal numerals reached Europe through Arabic mathematicians, and the resulting system is known as the Hindu-Arabic numeral system.

The Sulva Sutras is a Hindu text that documents Indian mathematics, and it dates from about 400 B.C. They were familiar with the statement and proof of Pythagoras's theorem, rational numbers, quadratic equations as well as the calculation of the square root of 2–5 decimal places.

1.8 Review Questions

1. Discuss the strengths and weaknesses of the various numbering system.
2. Describe the ciphers used during the Roman civilisation and write a program to implement one of these.
3. Discuss the nature of an algorithm and its importance in computing.
4. Discuss the working of an abacus and its application to calculation.
5. What are the differences between syllogistic logic and propositional and predicate logic?

1.9 Summary

Software is pervasive throughout society and has transformed the world in which we live in. New technology has led to improvements in all aspects of our lives including medicine, transport, education and so on. The pace of change of new technology is relentless, with new versions of technology products becoming available several times a year.

This chapter considered some of the contributions of early civilisations to computing. We commenced our journey with an examination of some of the contributions of the Babylonians. We then moved forward to consider some of the achievements of the Egyptians, the Greek and Romans, Islamic scholars and the Indians and Chinese.

The Babylonians developed a base 60 number system and recorded their mathematical knowledge on clay cuneiform tablets. These tablets included tables for multiplication, division, squares and square roots and the calculation of area. They were familiar with techniques that allowed the solution of a linear equation and one root of a quadratic equation to be determined.

The Egyptian civilisation developed along the River Nile and lasted over 3,000 years. They applied their knowledge of mathematics to solve practical problems such as measuring the annual Nile flooding and constructing temples and pyramids.

The Greeks and the later Hellenistic period made important contributions to western civilisation. This included contributions to philosophy, architecture, politics, logic, geometry and mathematics. The Euclidean algorithm is used to determine the greatest common divisor of two numbers. Eratosthenes developed an algorithm to determine the prime numbers up to a given number. Archimedes invented the 'Archimedes screw', the 'Archimedes claw' and a type of heat ray.

The Islamic civilisation helped to preserve western knowledge that was lost during the dark ages in Europe, and they also continued to develop mathematics and algebra. Hindu mathematicians introduced the decimal notation that is familiar today. It was adopted by Islamic mathematicians, and the resulting system is known as the Hindu-Arabic system.

Chapter 2
What Is a Computer?

Key Topics

Analog Computers
Digital Computers
Vacuum Tubes
Shockley
Transistors
Integrated Circuits
von Neumann Architecture
Generations of Computers
Hardware
Software

2.1 Introduction

Computers are an integral part of modern society, and new technology has transformed the world into a global village. Communication is now conducted using text messaging, e-mail, mobile phones, video calls over the Internet using Skype and social media sites such as Facebook. The new technology makes it easier for people to keep in touch with friends and family around the world and allows business to be conducted in a global market.

A computer is a programmable electronic device that can process, store and retrieve data. The data is processed by a set of instructions termed a *program*. All computers consist of two basic parts, namely, *hardware* and *software*. The hardware is the physical part of the machine, and a digital computer contains memory for short-term storage of data or instructions, a central processing unit for carrying out arithmetic and logical operations, a control unit responsible for the execution of computer instructions in memory and peripherals that handle the input and output operations. The underlying architecture is referred to as the von Neumann architecture (Fig. 2.5) and is described in Sect. 2.5. Software is a set of instructions that tells the computer what to do and is created by one or more programmers. It differs from hardware in that it is intangible.

The original meaning of the word '*computer*' referred to someone who carried out calculations rather than an actual machine. The early digital computers built in the 1940s and 1950s were enormous machines consisting of several thousand vacuum tubes.[1] They typically filled a large room or building but their computational power was a fraction of the power of the computers used today.

There are two distinct families of computing devices, namely, *digital computers* and the historical *analog computer*. These two types of computer operate on quite different principles, and the earliest computers were analog not digital.

The representation of data in an analog computer reflects the properties of the data that is being modelled. For example, data and numbers may be represented by physical quantities such as electric voltage in an analog computer, whereas in a digital computer, they are represented by a stream of binary digits.

A digital computer is a sequential device that generally operates on data one step at a time. The data in a digital computer are represented in binary format, and the binary number system uses just two digits: namely, 0 and 1. A single transistor has two states, that is on and off, and is used to represent a single binary digit. Several transistors are used to represent larger numbers. The earliest digital computers were developed in the 1940s.

2.2 Mechanical Calculators

There was important progress made on mechanical calculation in the seventeenth century. Napier introduced logarithms as a way to simplify calculation. This allowed the multiplication or division of two numbers to be performed, respectively, as the addition or subtraction of their logarithms:

$$\log xy = \log x + \log y$$
$$\log x/y = \log x - \log y$$

The slide rule was invented by William Oughtred and others in 1622, and it allowed multiplication and division to be carried out significantly faster than calculation by hand. The slide rule evolved over the centuries to include

[1] The Whirlwind computer (developed in the early 1950s and discussed in Chap. 4) occupied an entire building. One room contained the operating console consisting of several input and output devices.

reciprocals, squares, square roots, logarithms, exponentials and other functions. Slide rules were used by engineers who were working on the Apollo program to perform many of their calculation. They were replaced by electronic calculators in the early 1970s.

A mechanical calculator was a device used to perform the basic operations of arithmetic. The first mechanical calculator was invented by Blaise Pascal in 1642. He was just 19 years old when he produced his first prototype machine, and the machine was designed to ease the calculation burden of his father's work. His father worked as a tax commissioner, and he desired a device to speed up calculation to reduce his workload. Pascal produced several prototype machines before the machine finally appeared in 1645. It was called the Pascaline, and it could add or subtract two numbers. Multiplication or division could be performed by repeated addition or subtraction.

The Pascaline machine was the first calculator to be used in an office and the first calculator to be commercialized. Pascal built over 20 of these machines, and there are about 9 still existing today in various museums in France, Germany and the United States.

Gottfried Wilhelm Leibniz,[2] a German mathematician and philosopher, invented a mechanical calculator in 1672. It was called the 'Step Reckoner', and it was an advance on Pascal's machine in that it was the first calculator that could perform all four arithmetic operations, that is addition, subtraction, multiplication and division.

The operating mechanism for the machine was called the 'stepped cylinder' or 'Leibniz wheel'. This device continued to be used in calculating machines for over 200 years. Leibniz built two prototype machines but only one survives today. It is in the National Library of Lower Saxony in Hannover.

2.3 Analog Computers

Early foundational work on analog computation was done in the nineteenth century by James Thompson (who was the brother of the physicist, Lord Kelvin). Thompson invented a wheel-and-disk integrator which was used in mechanical analog devices, and he worked with Kelvin to construct a device to perform the integration of a product of two functions. Kelvin later described a general-purpose analog machine for integrating linear differential equations of any order. He built a tide-predicting analog device that remained in use at the Port of Liverpool up to the 1960s.

The operations in an analog computer are performed in parallel, and they are useful in simulating dynamic systems. They have been applied to flight simulation, nuclear power plants and industrial chemical processes.

[2] Leibniz is credited (along with Newton) with the development of the calculus.

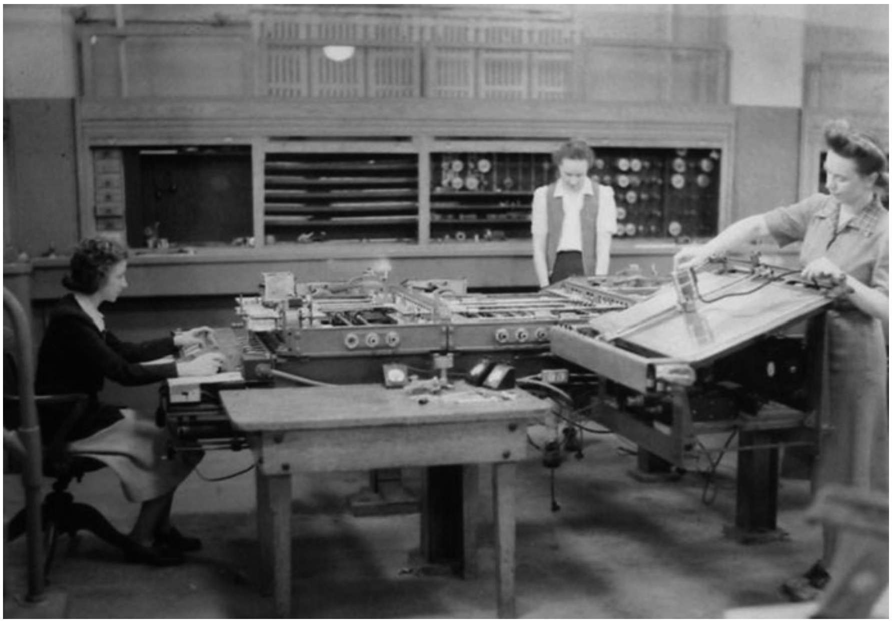


Fig. 2.1 Differential analyser at Moore School of Engineering, University of Pennsylvania

The first large-scale general-purpose mechanical analog computer was developed by Vannevar Bush and others at the Massachusetts Institute of Technology in the late 1920s. This mechanical analog computer was Bush's differential analyser, and it was designed to solve sixth order differential equations by integration, using wheel-and-disk mechanisms to perform the integration. It allowed integration and differential equations problems to be solved more rapidly. The machine took up the space of a large table in a room and weighed 100 tons (Fig. 2.1).

It consisted of wheels, disks, shafts and gears to perform the calculations and required a considerable effort to be set up by technicians to solve a particular equation. It contained 150 motors and miles of wires connecting relays and vacuum tubes.

Data representation in an analog computer is compact but may be subject to corruption with noise. A single capacitor can store one continuous variable in an analog computer, whereas several transistors are required in a digital computer. Analog computers were replaced by digital computers after the Second World War.

2.4 Digital Computers

Early digital computers used vacuum tubes to store binary information. A vacuum tube could represent the binary value '0' or '1'. However, the tubes were large and bulky and generated a significant amount of heat. Air conditioning was required to cool the machine, but there were problems with the reliability of the tubes.

The transistor was invented by Shockley and others in the 1950s, and they replaced vacuum tubes from the late 1950s. Transistors are small and require very little power. The resulting machines were smaller, faster and more reliable.

Integrated circuits were introduced in the 1960s, and a massive amount of computational power may be placed in a very small chip. They are small with little power consumed and may be mass produced to very high quality standard. Over a billion transistors may be placed on an integrated circuit today. However, integrated circuits are difficult to modify or repair and nearly always need to be replaced.

The development of the microprocessor was another revolution in the computing field, and it allowed a single chip to contain all of the components of a computer from the CPU and memory to input and output controls. The microprocessor could fit into the palm of the hand whereas the early computers filled an entire room.

The fundamental architecture of a computer has remained basically the same since it was proposed by von Neumann and others in the 1940s. It includes a central processing unit which includes the control unit, the arithmetic-logic unit, an input and output unit and memory.

2.4.1 Vacuum Tubes

A vacuum tube is a device that relies on the flow of an electric current through a vacuum. Vacuum tubes (thermionic valves) were widely used in electronic devices such as televisions, radios and computers until they were replaced by transistors from the late 1950s.

The first generation of computers used several thousand of these bulky vacuum tubes. A machine contained several racks of vacuum tubes which took up the space of a large room. A vacuum tube consists of three basic parts: a cathode (also known as filament), a grid and a plate. The vacuum tube was used to represent one of two binary states, that is the binary value ‘0’ or ‘1’.

The basic idea of a vacuum tube is that a current passes through the filament which then heats it up so that it gives off electrons. The electrons are negatively charged and are attracted to the small positive plate (or anode) within the tube. A unidirectional flow is established between the filament and the plate.

The filament of a vacuum tube becomes unstable over time. In addition, if air leaks into the tube, then oxygen will react with the hot filament and damage it. The size and unreliability of vacuum tubes motivated research into more compact and reliable technologies. This led to invention of the transistor in the mid-1950s.

The first generation of digital computers all used vacuum tubes, for example the Atanasoff-Berry computer (ABC) developed at the University of Iowa in 1942, Colossus developed at Bletchley Park in 1943, ENIAC developed in 1946, UNIVAC I developed in 1951 and the Whirlwind computer developed in 1951.

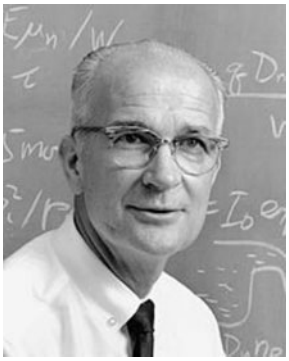


Fig. 2.2 William Shockley (Courtesy of Chuck Painter, Stanford News Service)

2.4.2 Transistors

The transistor is a fundamental building block in modern electronic systems, and its invention revolutionised the field of electronics. It led to smaller, cheaper and more reliable computers.

The transistor was developed at Bell Labs in the early 1950s. The goal of the research was to find a solid-state alternative to vacuum tubes, as these were too bulky and unreliable. Three inventors at Bell Labs (Shockley, Bardeen and Brattan) were awarded the Nobel Prize in physics in 1956 for their invention of the transistor.

It is a three-terminal, solid-state electronic device that can control electric current or voltage between two of the terminals by applying an electric current or voltage to the third terminal. The three-terminal transistor enables an electric switch to be made which can be controlled by another electrical switch. Complicated logic circuits may be built up by cascading these switches (switches that control switches that control switches and so on).

These logic circuits may be built very compactly on a silicon chip with a density of a million transistors per square centimetre. The switches may be turned on and off very rapidly (e.g. every 0.000000001 s). These electronic chips are at the heart of modern electron devices.

William Shockley did his Ph.D. at Massachusetts Institute of Technology in 1936, and he joined Bell Labs shortly afterwards. He was involved in radar research and anti-submarine operations research during the Second World War, and after the war he led a research group including Bardeen and Brattan to find a solid-state alternative to the glass-based vacuum tubes (Fig. 2.2).

Bardeen and Brattan succeeded in creating a point contact transistor in 1947 independently of Shockley who was working on a junction-based transistor. Shockley believed that the point-contact transistor would not be commercially viable, and the junction point transistor was announced in 1951.

Shockley was not an easy person to work with, and relations between him and the others deteriorated. He formed Shockley Semiconductor Inc. (part of Beckman Instruments) in 1955. However, his style of management alienated several of his

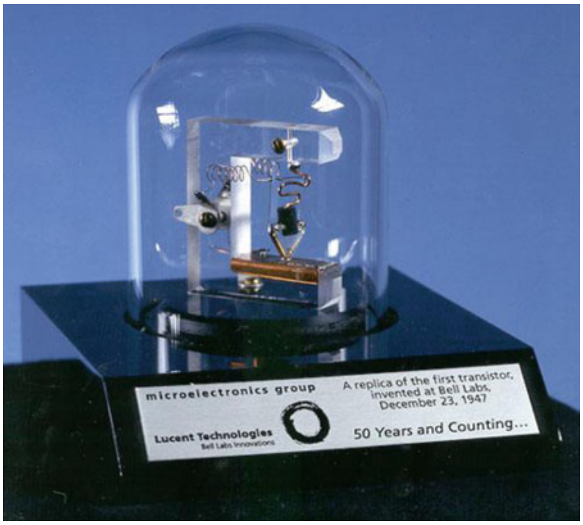


Fig. 2.3 Replica of transistor (Courtesy of Lucent Bell Labs)

employees and led to the resignation of eight key researchers in 1957 following his decision not to continue research into silicon-based semiconductors.

This gang of eight went on to form Fairchild Semiconductors and other companies in the Silicon Valley area in the following years. They included Gordon Moore and Robert Noyce who founded Intel in 1968. National Semiconductors and Advanced Micro Devices were later formed by employees from Fairchild (Fig. 2.3).

The second generation of computers used transistors instead of vacuum tubes. The University of Manchester's experimental Transistor Computer was one of the earliest transistor computers. The prototype machine appeared in 1953, and the full-size version was commissioned in 1955. The prototype machine had 92 point-contact transistors.

2.4.3 *Integrated Circuits*

The first Integrated Circuit was invented in 1958 by Jack Kilby who was working for Texas Instruments. Kilby together with Robert Noyce of Fairchild Semiconductors came up with a solution to the problem of large numbers of components with the development of the integrated circuit. The Nobel Prize in Physics was awarded to Kirby in 2000 for his contribution to its invention.

Their insight was that instead of making transistors one by one that several transistors could be made at the same time on the same piece of semiconductor. This allowed transistors and other electric components such as resistors, capacitors and diodes to be made by the same process with the same materials.

The number of transistors per unit area has been doubling every 2 years over the last 30 years. This amazing progress in circuit fabrication is known as *Moore's Law*, after Gordon Moore (one of the founders of Intel). This trend was originally described by Moore in his 1965 paper [Mor:65] and refined further in the 1970s.

Kilby was connecting many germanium[3] wafers of discrete components together by stacking each wafer on top of one another. Connections were made by running wires up the sides of the wafers.

He saw this process as unnecessarily complicated and realised that if a piece of germanium was engineered properly that it could act as many components simultaneously. This was the idea that led to the birth of the first integrated circuit, and its development involved miniaturising transistors and placing them on silicon chips called semiconductors. The use of semiconductors led to third-generation computers which were smaller, faster and cheaper than their predecessors.

The third generation of computers used integrated circuits and keyboards and monitors were used to interface via an operating system that allowed the computer to run many different applications at one time with a central program that monitored the memory.

2.4.4 Microprocessors

The development of the microprocessor led to the fourth generation of computers with thousands of integrated circuits placed onto a single silicon chip. A single chip could now contain all of the components of a computer from the CPU and memory to input and output controls. It could fit in the palm of the hand, whereas first generation of computers filled an entire room.

The Intel P4004 microprocessor is considered to be the world's first microprocessor, and it was released in 1969. It was the first semiconductor device that provided, at the chip level, the functions of a computer (Fig. 2.4).

It was created for use in calculators, but often an invention has applications other than those originally intended. This was the case with the Intel 4004 microprocessor, and it provides the basic building blocks used in today's microcomputers including the arithmetic and logic unit and the control unit. The 4-bit Intel 4004 ran at a clock speed of 108 kHz and contained 2,300 transistors. It processed data in 4 bits, but its instructions were 8 bits long. It could address up to 1 kB of program memory and up to 4 kB of data memory.

Gary Kildall of Digital Research was one of the early people to recognise the potential of a microprocessor as a computer in its own right. He worked as a consultant with Intel and began writing experimental programs for the Intel 4004 microprocessor. He later developed the CP/M operating system for the Intel 8080 chip and set up Digital Research as it is the company that Kildall set up to market and sell the operating system.

[3] Germanium is an important semiconductor material used in transistors and other electronic devices.

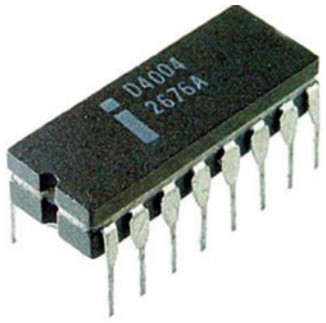


Fig. 2.4 Intel 4004 Microprocessor (Public domain)

Silicon is a type of semiconductor, and during chip production, silicon wafers are used to create many individual chips. Fairchild Semiconductor invented the modern silicon diffusing process in 1959. The integrated circuit development process gradually evolved over time, and computer-aided design was employed from 1967.

2.5 von Neumann Architecture

The earliest computers were fixed programs machines and were designed to do a specific task. This proved to be a major limitation as it meant that a complex manual rewiring process was required to enable the machine to solve a different problem (Fig. 2.5).

The computers used today are general-purpose machines designed to allow a variety of programs to be run on the machine. The fundamental architecture underlying modern computers was described by von Neumann and others in the mid-1940s. It is known as von Neumann architecture.

It arose on work done by von Neumann, Eckert, Mauchly and others on the design of the EDVAC computer. This was the successor to ENIAC computer, and von Neumann's draft report on EDVAC [VN:45] described the new architecture. The EDVAC computer was built in 1949.

von Neumann architecture led to the birth of stored program computers where a single store is used for both machine instructions and data. Its key components are shown in Table 2.1.

The key approach to building a general-purpose device according to von Neumann was in its ability to store not only its data and intermediate results of computation but also to store the instructions or commands for the computation. The computer instructions can be part of the hardware for specialised machines, but for general-purpose machines, the computer instructions must be as changeable as the data that is acted upon by the instructions. His insight was to recognise that both the machine instructions and data could be stored in the same memory (Fig. 2.6).

The key advantage of the von Neumann architecture was that it was much simpler to reconfigure a computer to perform a different task. All that was required was to enter

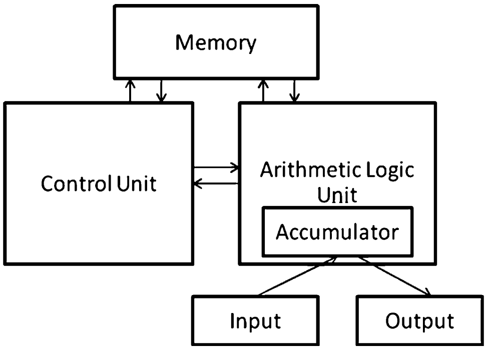


Fig. 2.5 von Neumann architecture

Table 2.1 von Neumann architecture

Component	Description
Arithmetic unit	The arithmetic unit is capable of performing basic arithmetic operations
Control unit	The program counter contains the address of the next instruction to be executed. This instruction is fetched from memory and executed. This is the basic Fetch and Execute cycle
	The control unit contains a built-in set of machine instructions
Input-output unit	The input and output unit allows the computer to interact with the outside world
Memory	There is a one-dimensional memory that stores all of the program instructions and data. These are usually kept in different areas of memory
	The memory may be written to or read from, i.e. it is random access memory (RAM)
	The program instructions are binary values, and the control unit decodes the binary value to determine the particular instruction to execute

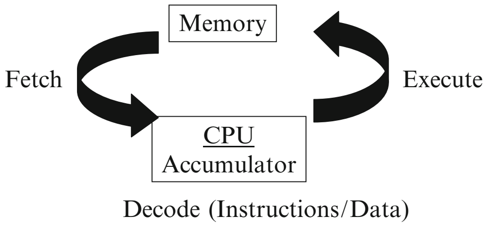


Fig. 2.6 Fetch Execute cycle

new machine instructions in memory, rather than physically rewiring a machine as was required with ENIAC. The limitations of von Neumann architecture include that it is limited to sequential processing rather than parallel processing.

2.6 Hardware and Software

Hardware is the physical part of the machine. It is tangible and may be seen and touched. It includes punched cards, vacuum tubes, transistors and circuit boards, integrated circuits and microprocessors. The hardware of a personal computer includes a keyboard, network cards, a mouse, a DVD drive, hard disk drive, printers, scanners and so on.

Software is intangible in that it is not physical, and instead it consists of a set of instructions that tells the computer what to do. It is an intellectual creation of a programmer or a team of programmers. There are several types of software such as system software and application software.

The system software manages the hardware and resources and acts as an intermediary between the application programs and the computer hardware. This includes the UNIX operating system, the various Microsoft Windows operating system for the personal computer and the Mac operating system for the Apple Macintosh computers. There are also operating systems for mobile phones, video games and so on. Application software is designed to perform a specific task such as banking or accounting.

Early software consisted of instructions in machine code that could be immediately executed by the computer. A program consisted of a sequence of machine code instructions, but these were difficult to read and debug. This led to assembly languages that represented a particular machine code instruction by a mnemonic, and the assembly language was translated into machine code by an assembler. Assembly languages were an improvement on machine code but were still difficult to use. This led to the development of high-level programming languages (e.g. FORTRAN and COBOL) where a program was written in the high-level language and compiled to the particular code of the target machine.

2.7 Review Questions

1. Explain the differences between an analog computer and a digital computer.
2. Explain how a transistor can represent a binary digit.
3. Explain the difference between a vacuum tube and a transistor.
4. Describe the components of the von Neumann architecture.
5. Explain the difference between hardware and software.

2.8 Summary

The objective of this chapter was to discuss the nature of a computer, and to briefly describe the evolution of the computing field from the invention of the slide rule and mechanical calculators in the seventeenth century to the development of analog computers and up to the birth of the digital computer.

A computer is a programmable electronic device that can process, store and retrieve data. All computers consist of two basic parts, namely, *hardware* and *software*.

The hardware is the physical part of the machine, and software is a set of instructions that tells the computer what to do.

The early digital computers built in the 1940s and 1950s were large machines consisting of thousands of vacuum tubes. Their computational power was a fraction of that of the personal computers used today.

Transistors began to replace vacuum tubes from the mid-1950s. These are small and use very little power, and so the resulting machines were smaller, faster and more reliable. Integrated circuits were introduced in the 1960s, and over a billion transistors may be placed on an integrated circuit. This allows a massive amount of computational power to be placed on a very small chip.

The development of the microprocessor allowed a single chip to contain all of the components of a computer from the CPU and memory to input and output controls. The microprocessor could fit into the palm of the hand, whereas the early computers filled an entire room.

The fundamental architecture of a computer has remained basically the same since it was proposed by von Neumann and others in the 1940s. It includes a central processing unit which includes the control unit and the arithmetic unit, an input and output unit and memory.

Chapter 3
Early Computers

Key Topics

Atanasoff-Berry Computer (ABC)
Zuse's Machines
ENIAC
EDVAC
UNIVAC
COLOSSUS
Manchester 'Baby'
Manchester Mark 1
Ferranti Mark 1
CSIRAC
EDSAC and LEO

3.1 Introduction

This chapter considers some of the early computers developed in the United States, Britain, Germany and Australia. The Second World War motivated researchers to investigate faster ways to perform calculation to solve practical problems. This led to research into the development of machines to provide faster methods of computation.

The early computers were large bulky machines consisting of several thousand vacuum tubes. A computer took up the space of a large room; it was slow and unreliable and had a fraction of the computational power of today's computers.

The early computers considered in this chapter include the Z1, Z2 and Z3 machines developed by Zuse in Germany; the ABC developed by Atanasoff and Berry in the United States; the ENIAC, EDVAC and UNIVAC computers developed by Eckert, Mauchly and others in the United States; the COLOSSUS machine developed as part of the Lorenz code breaking work at Bletchley Park in England during the Second World War; the Manchester 'Baby' and Manchester Mark 1 computers developed by Williams, Kilburn and others at Manchester University; the EDSAC and LEO computers developed in England; and finally the CSIRAC developed by the Australian Research and Industrial Organization (CSIR).

The early digital computers used vacuum tube technology. Later generations of computers used transistors and integrated circuits which were faster and more reliable. These are discussed in later chapters.

3.2 Zuse's Machines

Konrad Zuse was a German engineer and was born in Bonn in 1910. He is considered 'the father of the computer' in Germany as he built the world's first programmable machine (the Z3) in 1941. He initially worked as a design engineer at the Henschel aircraft factory in eastern Germany, but resigned from his job to set up a company to build a programmable machine. He returned to work with Henschel during the war. He was unaware of computer-related developments in Germany or in other countries and independently implemented the principles of modern digital computers in isolation.

He commenced work on his first machine called the Z1 in 1936, and the machine was operational by 1938. It was demonstrated to a small number of people who saw it rattle and compute the determinant of a three by three matrix. It was essentially a binary electrically driven mechanical calculator with limited programmability. It was capable of executing instructions read from the program punch cards, but the program itself was never loaded into the memory (Fig. 3.1).

It employed the binary system and metallic shafts that could slide from position 0 to position 1 and vice versa. The machine was essentially a 22-bit floating-point value adder and subtracter. A decimal keyboard was used for input, and the output was decimal digits. The machine included some control logic which allowed it to perform more complex operations such as multiplications and division. These operations were performed by repeated additions for multiplication and repeated subtractions for division. The multiplication took approximately 5 s. The computer memory contained sixty-four 22-bit words. Each word of memory could be read from and written to by the program punch cards and the control unit. It had a clock speed of 1 Hz, and two floating-point registers of 22 bits each. However, the machine was unreliable. A reconstruction of it is in the Deutsches Technikmuseum in Berlin.

His next attempt was the creation of the Z2 machine which aimed to improve on the Z1. This was a mechanical and relay computer created in 1939. It used a similar mechanical memory but replaced the arithmetic and control logic with 600 electrical relay circuits. It used 16-bit fixed point arithmetic instead of the 22-bit

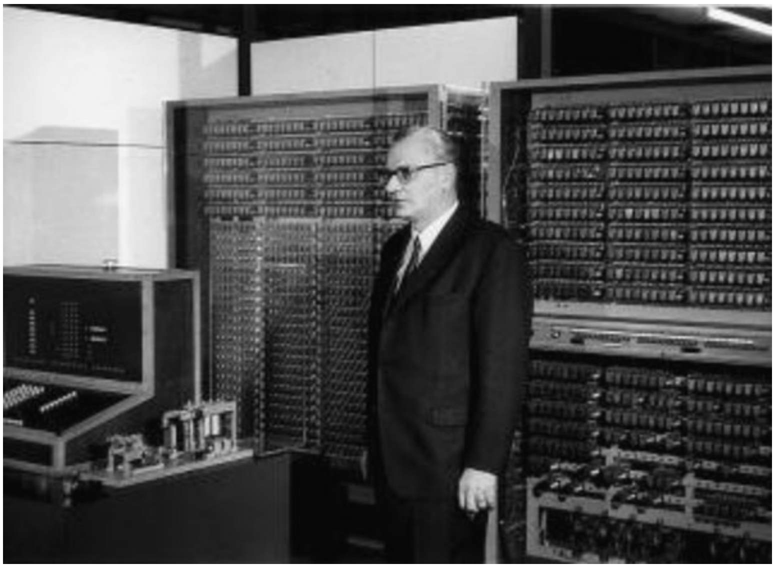


Fig. 3.1 Zuse and the reconstructed Z3 (Courtesy of Horst Zuse, Berlin)

used in the Z1. It had a 16-bit word size, and the size of its memory was 64 words. It had a clock speed of 3 Hz.

The Z3 machine was the first functional tape-stored-program-controlled computer and was created in 1941. It used 2,600 telephone relays and the binary number system and could perform floating-point arithmetic. It had a clock speed of 5 Hz, and multiplication and division took 3 s. The input to the machine was with a decimal keyboard, and the output was on lamps that could display decimal numbers. The word length was 22 bits, and the size of the memory was 64 words.

It used a punched film for storing the sequence of program instructions. It could convert decimal to binary and back again. It was the first digital computer since it predates the Atanasoff-Berry computer by 1 year. It was proven to be Turing-complete in 1998. There is a reconstruction of the Z3 computer in the Deutsches Museum in Munich.

The Z4 was almost complete before the fall of Berlin to the advancing Soviet Army. Zuse fled to Bavaria with the Z4, and this completed machine was the world's first commercial computer when it was introduced in 1950.

3.3 Atanasoff-Berry Computer (ABC)

John Atanasoff was born in New York in 1903 and studied engineering at the University of Florida. He did a Masters in Mathematics at Iowa State College and earned a Ph.D. from the University of Wisconsin. He became an assistant professor

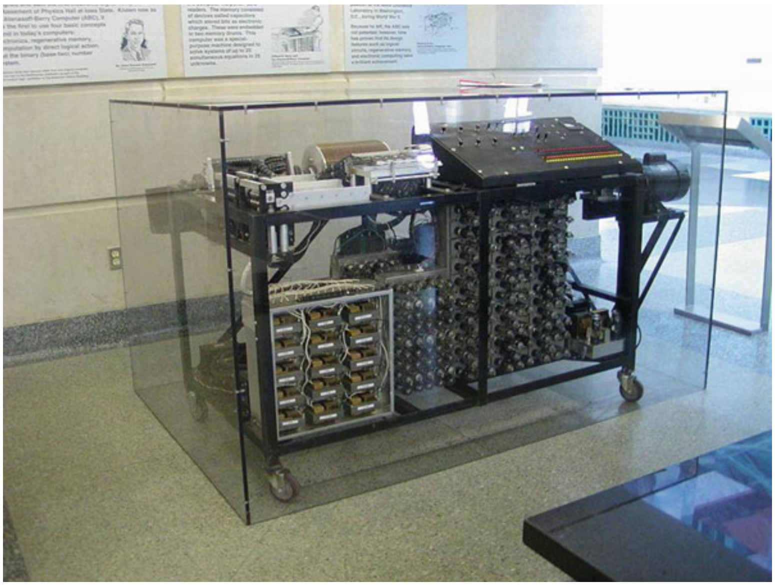


Fig. 3.2 Replica of ABC computer (Courtesy of Iowa State University)

at Iowa State College and became interested in finding faster methods of computation. He developed the concept of the ABC in the late 1930s, and using his research grant of $650 and with the assistance of his graduate student, Clifford Berry, the ABC was built from 1939 to 1942.

The Atanasoff-Berry computer was ruled to be the first electronic digital computer in a 1973 court case in the United States. The court case arose from a patent dispute, and Atanasoff was called as an expert witness in the case. The court ruled that Eckert and Mauchly did not invent the first electronic computer since the ABC existed as *prior art* at the time of their patent application. It is fundamental in patent law that an invention is novel and that there is no existing prior art. This meant that the Mauchly and Eckert patent application for ENIAC was invalid, and Atanasoff was named by the US court as the inventor of the first digital computer (Fig. 3.2).

The ABC was approximately the size of a large desk and had approximately 270 vacuum tubes. Two hundred and ten tubes controlled the arithmetic unit, 30 tubes controlled the card reader and card punch and the remaining tubes helped maintain charges in the condensers. It employed rotating drum memory with each of the two drum memory units able to hold thirty 50-bit numbers.

Atanasoff became interested in mechanising calculation from the mid-1930s, and the ABC was designed to solve linear equations. The existing computing devices were mechanical, electromechanical or analog. Atanasoff and Berry built a working prototype of the electronic digital computer, called the Atanasoff-Berry computer (ABC). However, the ABC was slow, required constant operator monitoring and was not programmable.

It used binary mathematics and Boolean logic to solve simultaneous linear equations. It used over 270 vacuum tubes for digital computation, it had no central processing unit (CPU) and it was not programmable. It was designed for a specific purpose (i.e. solving linear equations) rather than as a general-purpose computer.

It weighed over 300 kg and used 1.6 km of wiring. Data was represented by 50-bit fixed point number. It performed 30 additions or subtractions per second. The memory and arithmetic units could operate and store 60 such numbers at a time ($60 * 50 = 3{,}000$ bits). The arithmetic logic unit was unit fully electronic and implemented with vacuum tubes.

The input was in decimal format with standard IBM 80-column punch cards, and the output was decimal via a front panel display. A paper card reader was used as an intermediate storage device to store the results of operations too large to be handled entirely within electronic memory. The ABC pioneered important elements in modern computing including:

- Binary arithmetic and Boolean logic.
- All calculations were performed using electronics rather than mechanical switches.
- Computation and memory were separated.

It was tested and operational by 1942, and Atanasoff then commenced a Second World War assignment.

Controversy (Mauchly and Atanasoff)
Mauchly visited Atanasoff on several occasions and they discussed the implementation of the ABC computer. Mauchly subsequently developed the ENIAC, EDVAC and UNIVAC computers. It had been believed for many years that ENIAC was the first digital computer until a 1973 legal case ruled that the Atanasoff Berry Computer (ABC) existed as prior art at the time of the patent application. The court ruled that the ABC was the first digital computer and stated that the inventors of ENIAC had derived the subject matter of the electronic digital computer from Atanasoff.

3.4 The Bletchley Park Contribution

Bletchley Park is located near Milford Keynes in England, and it played an important role in breaking German cipher codes during the Second World War. These codes were used by the Germans for the encryption of naval messages which prevented third parties from unauthorised viewing of the messages. The plaintext (i.e. the original message) was converted by the Enigma machine into the encrypted text, and these messages were then transmitted by the Germans to their submarines in the Atlantic or their bases throughout Europe.

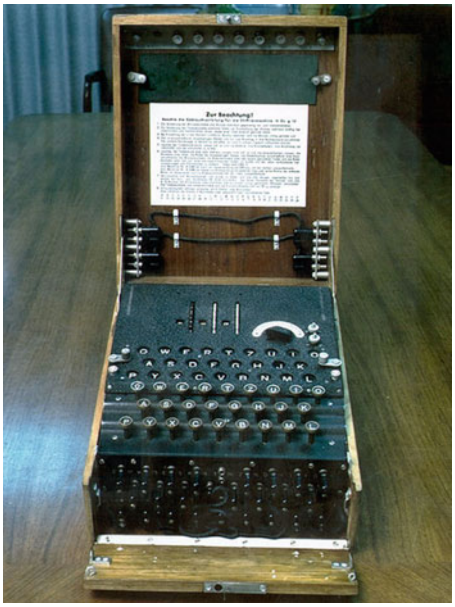


Fig. 3.3 The Enigma machine (Public domain)

The Enigma codes were cracked by the team of cryptanalysts in Bletchley Park using a machine called the Bombe. The Poles had done some work on code breaking prior to the war, and they passed their knowledge to the British following the German invasion of Poland. Alan Turing and Gordon Welchman built on the Polish research to develop the 'Bombe' machine. They used the fact that enciphered German messages often contained common words or phrases, such as general's names or weather reports, and this enabled them to guess short parts of the original message. These guesses were called 'cribs'.

The Enigma machine does not allow a letter to be enciphered as itself, and this also helped in reducing the potential number of settings that the Enigma machine could be in on that particular day (Fig. 3.3).

The first Bombe was installed in early 1940, and by the end of the war, there were over 200 Bombes in operation. Each Bombe weighed over a ton, and it was named after a cryptological device designed in 1938 by the Polish cryptologist, Marian Rejewski. They were over 7 ft long, 6 ft high and 2 ft deep. The Bombes were built by the British Tabulating Machine Company, and they were destroyed after the war. A replica of the British Turing Bombe machine was rebuilt at Bletchley Park by a team of volunteers and was switched on by the Duke of Kent in 2008.

The code breaking team then wired the Bombe to check the reduced set of potential settings based on the information that they had gained from the cribs. The Bombe found potential Enigma settings not by proving a particular setting, but by disproving every incorrect one in turn.

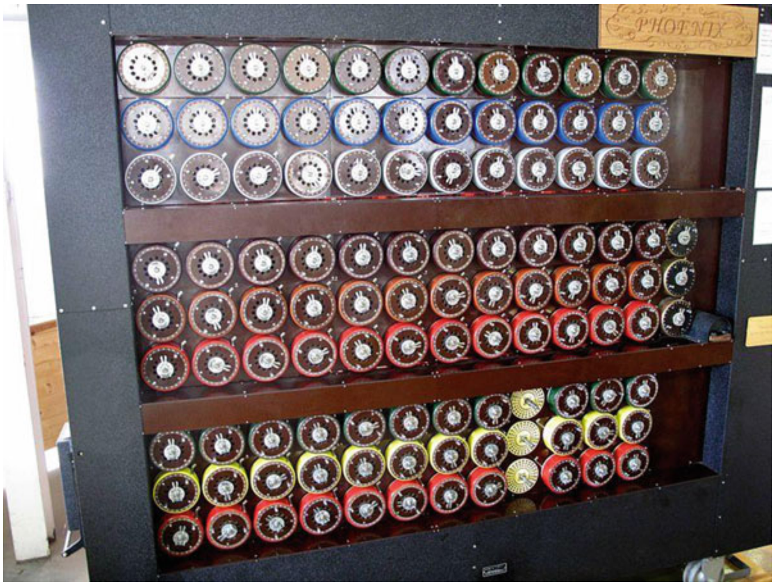


Fig. 3.4 Rebuilt Bombe (Photo public domain)

A standard Enigma machine employed a set of three rotors. Each rotor could be in any of 26 positions. The Bombe tried each possible rotor position and applied a test. The test eliminated almost all the 17,576 positions (i.e. 26^3) and left a smaller number of cases to be dealt with. The test required the cryptologist to have a suitable ‘crib’, that is a section of cipher text for which he could guess the corresponding plaintext (Fig. 3.4).

For each possible setting of the rotors, the Bombe employed the crib to perform a chain of logical deductions. The Bombe detected when a contradiction had occurred; it then ruled out that setting and moved onto the next. Most of the possible settings would lead to contradictions and could then be discarded. This would leave only a few settings to be investigated in detail.

The Lorenz codes were a more complex cipher than the Enigma codes. The enciphering was performed by the Lorenz SZ40/42 machine, and it was used exclusively for the most important messages passed between the German Army Field marshals and their Central High Command in Berlin. It was not a portable device like the Enigma machine.

The Bletchley Park code breakers called the machine ‘Tunny’ and the coded messages ‘Fish’. The code breaking was performed by carrying out complex statistical analyses on the intercepted messages. The Colossus Mark 1 machine was specifically designed for code breaking rather than as a general-purpose computer. It was semi-programmable and helped in deciphering messages encrypted using the Lorenz machine. A prototype was available in 1943, and a working version was

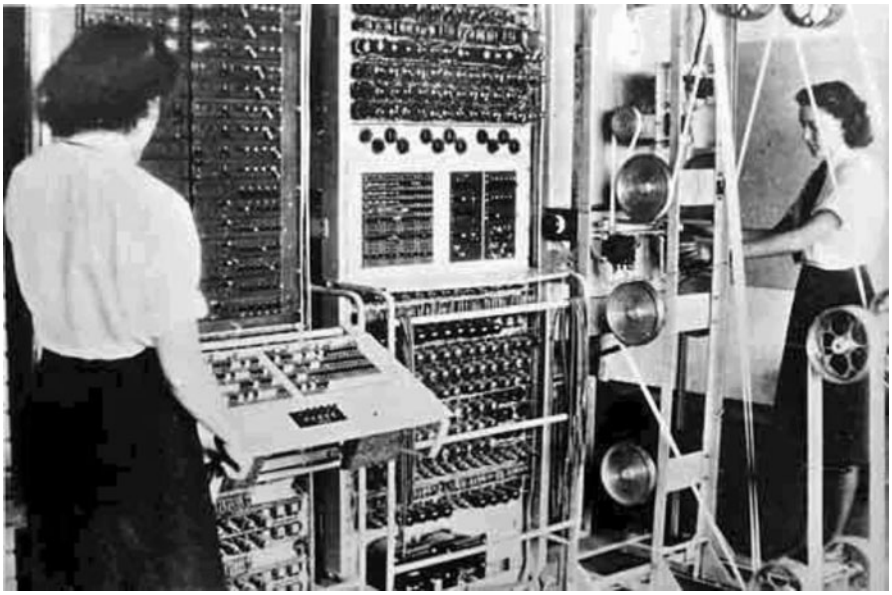


Fig. 3.5 Colossus Mark 2 (Photo courtesy of UK government)

available in early 1944 at Bletchley Park. The Colossus Mark 2 was introduced just prior to the Normandy landings in June 1944 (Fig. 3.5).

Colossus contained 1,500 vacuum tubes and used 15 kW of power. It was partially programmable and could process 5,000 characters of paper tape per second. It enabled a large amount of mathematical work to be done in hours rather than in weeks. There were ten colossi machines working at Bletchley Park by the end of the war. A replica of the Colossus was rebuilt by a team of volunteers led by Tony Sale from 1993 to 1996 and is in Bletchley Park museum.

The original machine was designed by Tommy Flowers and others at the Post Office Research Station at Dollis Hill in London. It was used to find possible key combinations for the Lorenz machines rather than decrypting an intercepted message in its entirety.

It did this by comparing two data streams to identify possible key settings for the Lorenz machines. The first data stream was the encrypted message, and it was read at high speed from a paper tape. The other stream was generated internally, and was an electronic simulation of the Lorenz machine at various trial settings. If the match count for a setting was above a certain threshold, it would be sent as output to an electric typewriter.

The contribution of Bletchley Park to the cracking of the German Enigma and Lorenz codes and to the development of computing remained clouded in secrecy until recent times. The museum at Bletchley Park provides insight to the important contribution made by this organisation during the Second World War.

3.5 ENIAC, EDVAC and UNIVAC

The Electronic Numerical Integrator and Computer (ENIAC) was one of the first large general-purpose electronic digital computers. It was used to integrate ballistic equations and to calculate the trajectories of naval shells. It was completed in 1946 and remained in use until 1955. The original cost of the machine was approximately $500,000.

The ENIAC had to be physically rewired in order to perform different tasks, and it was clear that there was a need for an architecture that would allow a machine to perform different tasks without physical rewiring each time. This eventually led to the concept of the stored program which was implemented in the successor to ENIAC. The idea of a stored program is that the program is stored in memory, and when the task to be computed needs to be changed, then all that is required is to place a new program in memory rather than rewiring the machine. EDVAC (the successor of ENIAC) implemented the concept of a stored program in 1949 just after its implementation on the Manchester Baby prototype machine in England. The concept of a stored program and von Neumann architecture is detailed in von Neumann's report on EDVAC [VN:45] (Fig. 3.6).

ENIAC was a large bulky machine and was over 100 ft long, 10 ft high and 3 ft deep. Its development commenced in 1943 at the University of Pennsylvania,

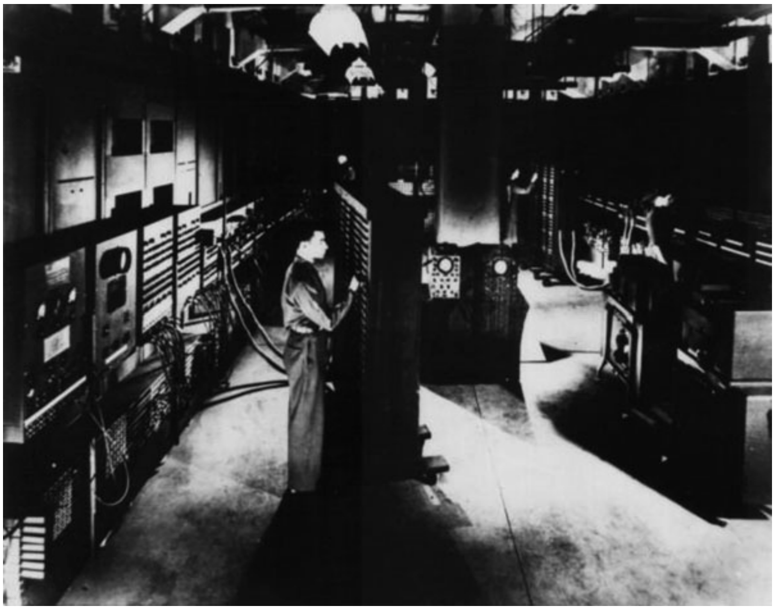


Fig. 3.6 Setting the switches on ENIAC's function tables (US Army photo)

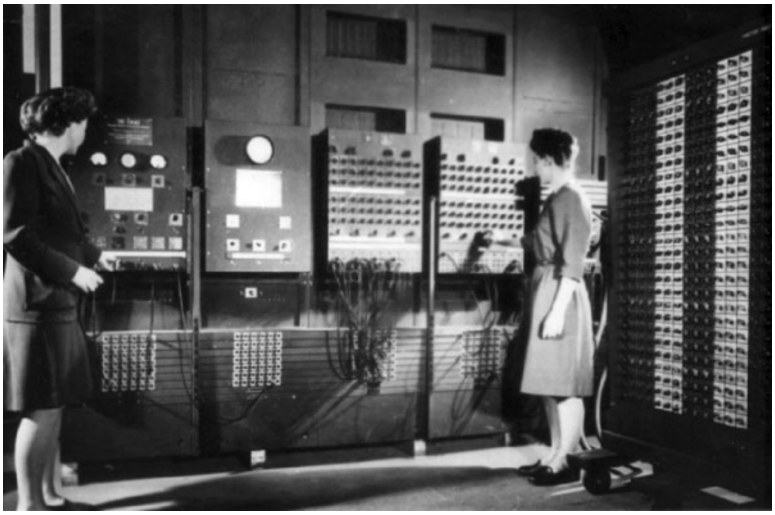


Fig. 3.7 Replacing a valve on ENIAC (US Army photo)

and it was built for the US Army's Ballistics Research Laboratory. The project team included Presper Eckert as chief engineer, John Mauchly as a consultant, von Neumann as an external advisor and several others. The machine generated a vast quantity of heat since each of the vacuum tubes generated heat like a light bulb, and there were over 18,000 tubes in the machine. The machine employed 150 kW of power, and air conditioning was employed to cool the machine. It weighed about 30 tons and employed IBM paper card readers.

It employed decimal numerals rather than binary. It could add 5,000 numbers, do 357 10-digit multiplications or thirty-five 10-digit divisions in 1 s.

It could be programmed to perform complex sequences of operations, and this included loops, branches and subroutines. However, the task of taking a problem and mapping it onto the machine was complex and usually took weeks to perform. The first step was to determine what the program was to do on paper; the second step was the process of manipulating the switches and cables to enter the program into ENIAC, and this usually took several days. The final step was verification and debugging, and this often involved single-step execution of the machine (Fig. 3.7).

There were problems initially with the reliability of ENIAC as several vacuum tubes burned out most days. This meant that the machine was often nonfunctional as high-reliability tubes were not available until the late 1940s. However, most of these problems occurred during the warm-up and cool-down periods, and therefore it was decided not to turn the machine off. This led to improvements in its reliability to the acceptable level of one tube every 2 days. The longest continuous period of operation without a failure was 5 days.

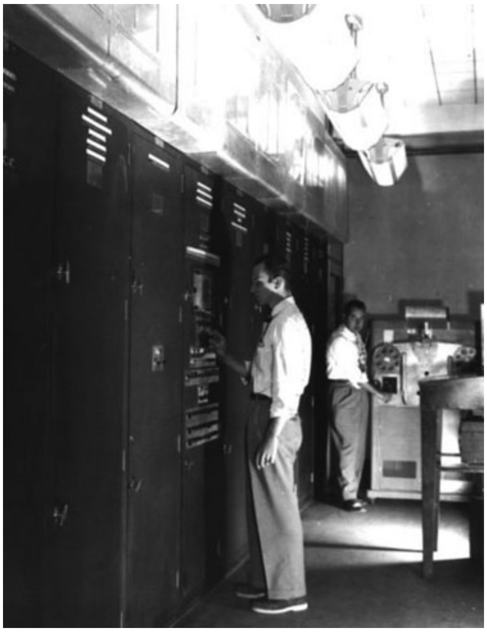


Fig. 3.8 The EDVAC computer (US Army photo)

The very first problem run on ENIAC required only 20 s and was checked against an answer obtained after 40 h of work with a mechanical calculator. One of the earliest problems solved was related to the feasibility of the hydrogen bomb. It involved the input of 500,000 punch cards, and the program ran for 6 weeks and gave an affirmative reply. ENIAC was preceded in development by the Atanasoff-Berry computer (ABC) and the Colossus computer.

The EDVAC (Electronic Discrete Variable Automatic Computer) was the successor to the ENIAC computer. It was a stored program computer, and it cost $500,000. It was proposed by Eckert and Mauchly in 1944 and design work commenced prior to the completion of ENIAC. It was delivered to the Ballistics Research Laboratory in 1949; it commenced operations in 1951 and ran until 1961.

It employed 6,000 vacuum tubes, and its power consumption was 56,000 W. It had 5.5 kB of memory (Fig. 3.8).

The ENIAC was a major milestone in the development of computing, and Eckert and Mauchly set up a company shortly afterwards called the Eckert-Mauchly Computer Corporation. They received an order from the National Bureau of Standards to develop the Universal Automatic Computer (UNIVAC) which was one of the first commercially available computers. The Univac 1 computer was delivered in late1950, and it was a machine for general processing (Fig. 3.9).

Their company was taken over by the Remington Rand Corporation in 1950, and it later became Unisys (Fig. 3.10).

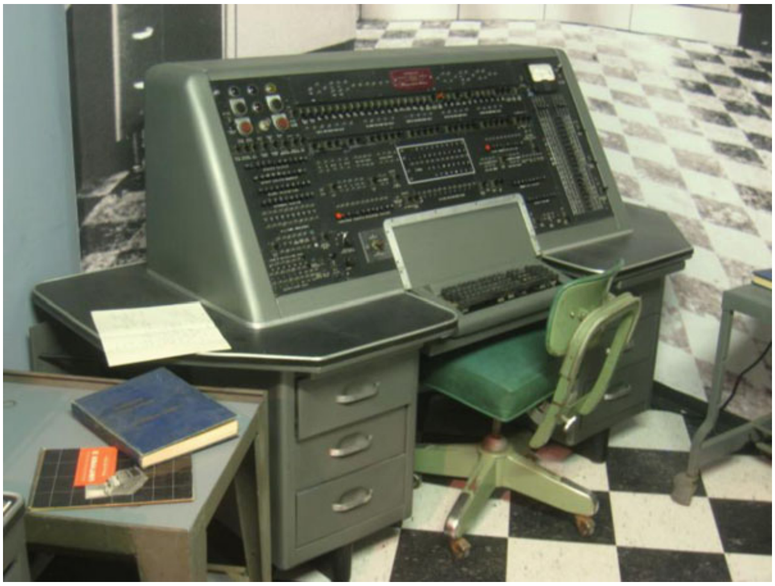


Fig. 3.9 The UNIVAC operator console (Photo public domain)

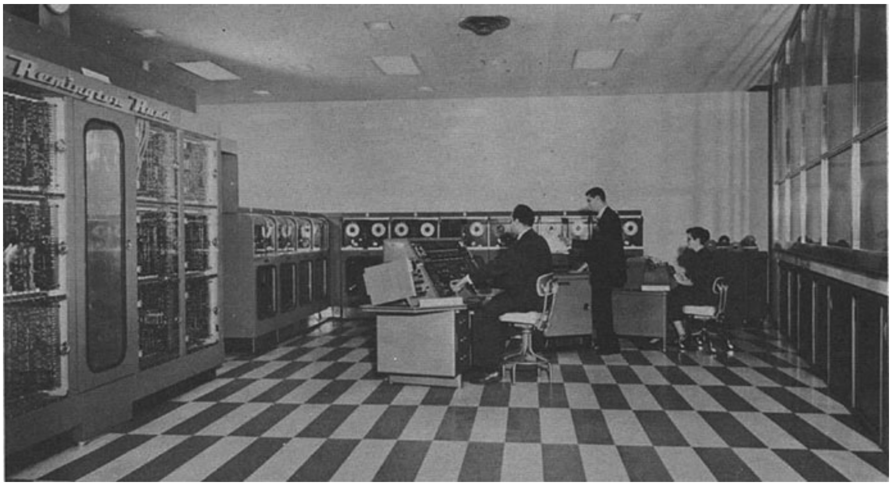


Fig. 3.10 The UNIVAC computer (Photo public domain)

3.6 The Manchester University Contribution

Sir Frederick Williams and Tom Kilburn co-invented the Williams-Kilburn Tube (also known as the Williams Tube), which was a cathode-ray tube used to store binary data. It was the first random access digital storage device and was popular in the computer data-storage field for several years until it was outdated by core memory in 1955. It provided the first large amount of random access memory (RAM), and it did not require rewiring each time the data was changed.

Williams had succeeded in storing one bit of information on a cathode-ray tube, and Kilburn began working with him in the mid-1940s to improve its digital storage ability. Kilburn devised an improved method of storing bits which increased the storage capacity to 2,048 bits. They were now ready to build a computer based on the Williams Tube.

3.6.1 Manchester Baby

The Manchester Small Scale Experimental Computer (better known by its nickname ‘Baby’) was developed at the University of Manchester. It was the *first stored program computer*, and it was designed and built at Manchester University in England by Frederic Williams, Tom Kilburn and others.

Kilburn was assisted by Geoff Tootill in the design and construction of the prototype machine, and the prototype machine demonstrated the ability of the Williams Tube. It was also the first stored program computer; that is the instructions to be executed were loaded into memory rather than by rewiring the computer to run a new program. That is, all that was required was to enter the new program into the computer memory. *For the first time in history, a computer used a stored program*, and Kilburn wrote and executed that short computer program in 1948 (Fig. 3.11).

The prototype machine demonstrated the feasibility and potential of the stored program. The memory of the machine consisted of thirty-two 32-bit words, and it took 1.2 ms to execute one instruction, that is 0.00083 MIPS (million instructions per second). Today’s computers are rated at speeds of up to 1,000 MIPS. The team in Manchester developed the machine further, and in 1949, the Manchester Mark 1 was available.

3.6.2 Manchester Mark 1

The Manchester Automatic Digital Computer (MADC), also known as the Manchester Mark 1, was developed at the University of Manchester. It was one of the earliest stored program computers, and it was the successor to the earlier prototype computer nicknamed ‘Baby’. It was designed and built by Frederic Williams, Tom Kilburn and others (Fig. 3.12).

Each word could hold one 40-bit number or two 20-bit instructions. The main memory consisted of two pages (i.e. two Williams tubes with each holding 32×40 bit words or 1,280 bits). The secondary backup storage was a magnetic drum consisting of 32 pages (this was updated to 128 pages in the final specification).

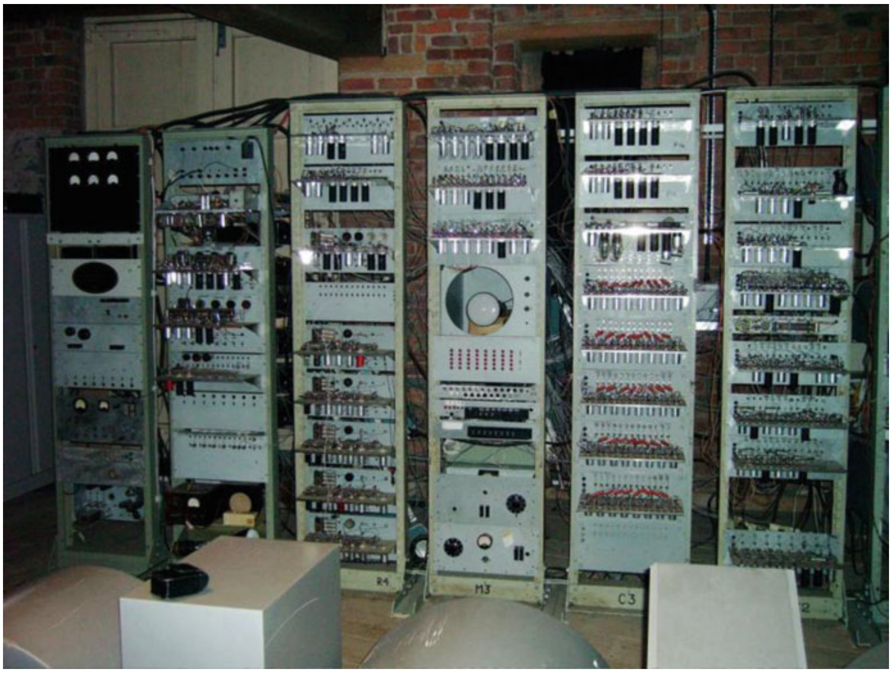


Fig. 3.11 Replica of the Manchester Baby (Photo courtesy of Tommy Thomas)

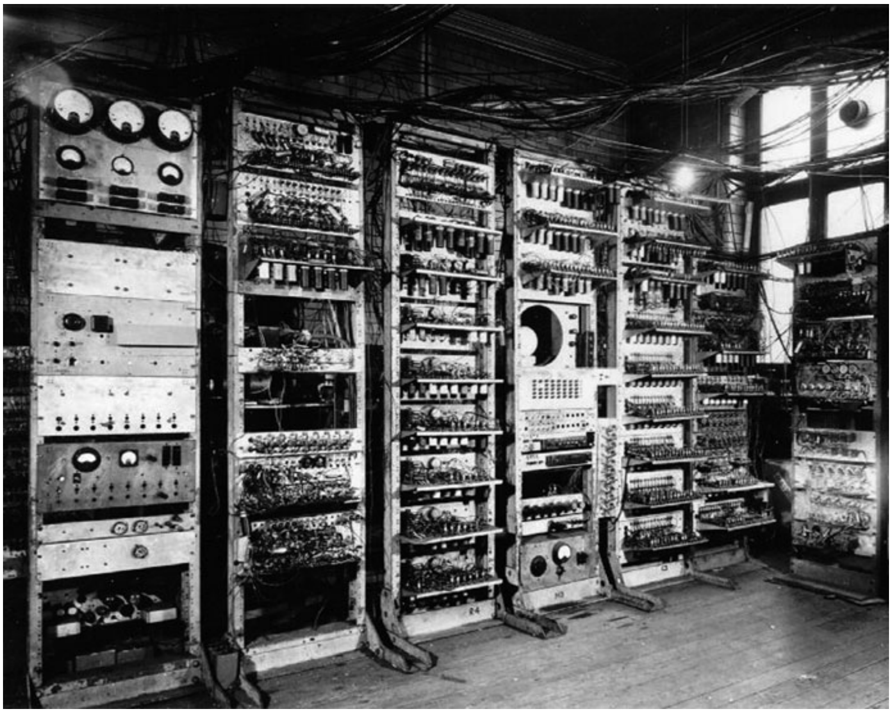


Fig. 3.12 The Manchester Mark 1 computer (Courtesy of the University of Manchester)

Each track consisted of two pages (2,560 bits). One revolution of the drum took 30 ms, and this allowed the 2,560 bits to be transferred to main memory.

It contained 4,050 vacuum tubes and had a power consumption of 25,000 W. The standard instruction cycle was 1.8 ms; however, multiplication was much slower. The machine had 26 defined instructions, and the programs were entered into the machine in binary form as assembly languages were not yet available.

It had no operating system, and its only systems software were some basic routines for input and output. Its peripheral devices included a teleprinter and a five-hole paper tape reader and punch.

A display terminal used with the Manchester Mark 1 computer mirrored what was happening within the Williams Tube. A metal detector plate placed close to the surface of the tube detected changes in electrical discharges. The metal plate obscured a clear view of the tube, but the technicians could monitor the tubes used with a video screen. Each dot on the screen represented a dot on the tube's surface; the dots on the tube's surface worked as capacitors that were either charged and bright or uncharged and dark. The information translated into binary code (0 for dark, 1 for bright) became a way to program the computer.

3.6.3 Ferranti Mark 1

Ferranti Ltd. (a British company) and Manchester University collaborated to build one of the world's first commercially available general-purpose electronic computer. This machine was the Ferranti Mark 1, and it was basically an improved version of the Manchester Mark 1. The first machine off the production line was delivered to the University of Manchester in 1951 and shortly before the release of the UNIVAC 1 electronic computer in the United States.

The Ferranti Mark 1's instruction set included a 'hoot command' which allowed auditory sounds to be produced. It also allowed variations in pitch. Christopher Strachey (who later did important work on the semantics of programming languages) programmed the Ferranti Mark 1 to play tunes such as 'God save the King', and the Ferranti Mark 1 was one of the earliest computers to play music. The earliest was the CSIRAC computer in Australia. The parents of Tim Berners-Lee (the inventor of the World Wide Web) both worked on the Ferranti Mark 1.

The main improvements of the Ferranti Mark 1 over the Manchester Mark 1 were improvements in size of primary and secondary storage, a faster multiplier and additional instructions.

It had eight pages of random access memory (i.e. eight Williams tubes each with a storage capacity of sixty-four 20-bit words or 1,280 bits). The secondary storage was provided by a 512-page magnetic drum which stored two pages per track, and its revolution time was 30 ms.

It used a 20-bit word stored as a single line of dots on the Williams tube display, with each tube storing a total of 64 lines of dots (or 64 words). Instructions were stored in a single word, while numbers were stored in two words.

The accumulator was 80 bits, and it could also be addressed as two 40-bit words. There were about 50 instructions, and the standard instruction time was 1.2 ms. Multiplication could be completed in 2.16 ms. There were 4,050 vacuum tubes employed.

3.7 EDSAC and LEO

The Electronic Display Storage Automatic Computer (EDSAC) was an early British computer developed by Maurice Wilkes and others at the University of Cambridge in England. The machine was inspired by von Neumann's report on EDVAC [VoN:45], and it was the first practical stored program computer. EDSAC ran its first program in May 1949 when it calculated a table of squares and a list of prime numbers.

It was used to support the industry needs of the university. It used mercury delay lines for memory and vacuum tubes for logic; input was via a five-hole punched tape; output was via a teleprinter. The accumulator could hold 71 bits, including the sign, and this allowed two 35-bit numbers to be multiplied.

J. Lyons & Co., a British food manufacturing and catering company, supported the project with funding of £3,000. The board of Lyon was forward thinking and realised that a computer could support its business needs. Once EDSAC was successfully completed, the board of Lyons decided to start on the construction of their own machine. This machine was christened the Lyons Electronic Office (LEO), and it was one of the earliest commercial computers. The first business application to run on LEO was Bakery Valuations which was run in 1951.

The machine was used initially for valuation jobs and extended over time to payroll, inventory and other applications. Its clock speed was 500 kHz, with most instructions taking 1.5 ms to execute.

Lyons formed LEO Computers Ltd. in 1954 following its decision to proceed with LEO II computer. The LEO III was developed in 1961, and LEO computers merged into the English Electric Company in 1963. This company merged with International Computers and Tabulators to form International Computers Ltd. in 1968.

3.8 The Australian Contribution

The CSIRAC (Council for Scientific and Industrial Research Automatic Computer) was Australia's first digital computer. It was also the fourth stored program computer in the world. It was first run in November 1949, and it is one of only a handful of first-generation computers still existing in the world. It is on permanent display at the Melbourne Museum (Fig. 3.13).

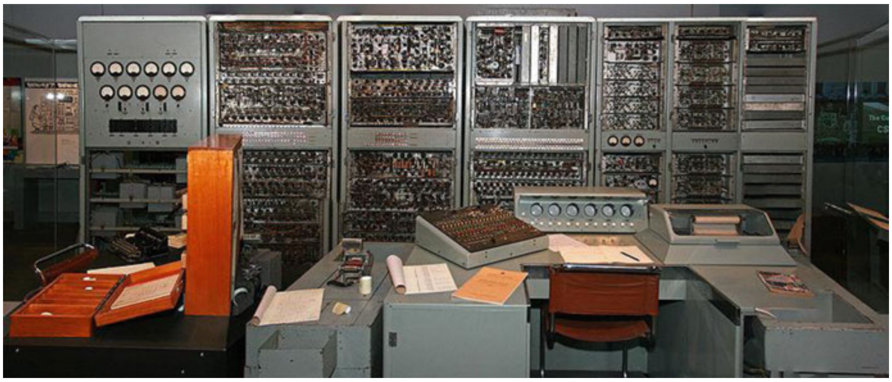


Fig. 3.13 CSIRAC (Photo courtesy of John O'Neill)

It was constructed by a team led by Trevor Pearcey and Maston Beard at the CSIR in Sydney. The machine had 2,000 vacuum valves and used 30 kW of power during operation. The input to the machine was by a punched paper tape, and output was to a teleprinter or to punched tape. The machine was controlled through a console which allowed programs to be stepped through.

The CSIRAC was the first digital computer to play music, and this took place in 1950. The machine was moved to the University of Melbourne in the mid-1950s, and today the machine is on permanent display at the Melbourne Museum.

3.9 Review Questions

1. Discuss the contribution of Zuse.
2. Discuss the contribution of Bletchley Park to cracking cipher codes and to the development of computing.
3. Discuss the contribution of Manchester University to computing.
4. Discuss the importance of the ENIAC, EDVAC and UNIVAC computers.
5. Discuss the importance of the Atanasoff-Berry computer.

3.10 Summary

This chapter considered some of the early computers developed in the United States, Britain, Germany and Australia. The Second World War motivated researchers to investigate faster ways to perform calculations to solve practical problems.

This led to an interest in the development of machines to provide faster methods of computation.

The earliest computer developed was the Z3 machine developed by Konrad Zuse in Germany in 1941. Zuse set up the first computer company, and the first commercial computer, the Z4, was delivered to the Technical University of Zurich in 1950.

The earliest computer developed in the United States was the Atanasoff-Berry computer developed by John Atanasoff and his graduate student, Clifford Berry. Mauchly met with Atanasoff on a number of occasions and Mauchly and Eckert developed ENIAC at the Moore School of Engineering at the University of Pennsylvania. This was a large bulky vacuum tube computer.

von Neumann later became involved in the design of EDVAC (the successor to the ENIAC), and he published a draft paper that describes the fundamental architecture underlying digital computers. This is known as the 'von Neumann architecture', and the names of Mauchly and Eckert were removed from the draft report as they had resigned from the Moore School to start their own computer company.

The earliest computer developed in Britain was the COLOSSUS machine developed at Bletchley Park in England as part of the code breaking work during the Second World War. The Manchester 'Baby' and Manchester Mark 1 computers were developed by Williams, Kilburn and others at Manchester University. The University of Cambridge received funding from Lyons to assist them in the development of the EDSAC Computer. This led to the development of the LEO computer for Lyons in 1951.

Chapter 4
Developments in the 1950s–1970s

Key Topics

IBM 360
IBM 701
IBM 650
SAGE
PDP
Spacewar
Home Computers
Altair 8080
Commodore PET
Apple I and Apple II

4.1 Introduction

This chapter considers a selection of computers developed during the 1950s–1970s. The initial driver for the design and development of more powerful computers was the perceived threat of the Soviet Union. This led to an arms race between the two superpowers, and it was clear that computing technology would play an important role in developing sophisticated weapon and defence systems. The SAGE air defence system developed for the United States and Canada was an early example of the use of computer technology for the military.

The machines developed during this period were mainly large proprietary mainframes and minicomputers. They were developed for business, scientific and government use. They were expensive, and this led vendors such as IBM and DEC to introduce families of computers in the 1960s where a customer could choose a smaller cheaper member of the family and to upgrade to a larger computer as their needs expanded.

It was only in the mid-1970s that the idea of a home computer was born. Early home computers included the Altair-8080, Commodore and the Apple I and Apple II computers. The home computer later led to major changes for the industry and the eventual replacement of the mainframe computer with networks of personal computers and servers.

The industry witnessed major technology changes during this period. The large bulky vacuum tubes were replaced by compact transistors (invented by Shockley and others) leading to smaller and more reliable machines. The invention of the integrated circuit by Kirby allowed a large number of transistors to be placed on a single piece of silicon. Finally, the invention of the microprocessor by Intel (which provided all of the functionality of a computer on a single chip) led to the development of home computers from the mid-1970s.

The introduction of the IBM 360 series of computers was an important milestone in computing. It was a paradigm shift for the industry as up to then each computer was designed independently and customised for a particular customer. The System/360 was a family of small to large computers, and it offered the customer a choice of the size and computational power rather than a one-size-fits-all approach. The family ranged from minicomputers with 24 kB of memory to supercomputers for US missile defence systems.

This allowed a customer to start with a small member of the S/360 family and to upgrade over time in accordance to their needs to a larger computer in the family. This helped to make computers more affordable for businesses and stimulated growth in computer use.

All of the computers is the S/360 family had the same user instruction set. The main difference was that the larger computers implemented the more complex machine instructions with hardware, whereas the smaller machines used microcode.

A selection of the PDP computers designed and developed by the Digital Equipment Corporation (DEC) is discussed. The PDP family was very popular in business and scientific computing, and at its peak DEC was the second largest computer company in the world. Finally, the introduction of home computers such as the Apple I and Apple II computers are discussed.

4.2 SAGE

The Semi-Automated Ground Environment (SAGE) was an automated system for tracking and intercepting enemy aircraft in North America. It was used by the North American Aerospace Defence Command (NORAD) which is located in Colorado in the United States. The SAGE system was used from the late 1950s until the 1980s.

The interception of enemy aircraft was extremely difficult prior to the invention of radar during the Second World War. Its introduction allowed fighter aircraft to be

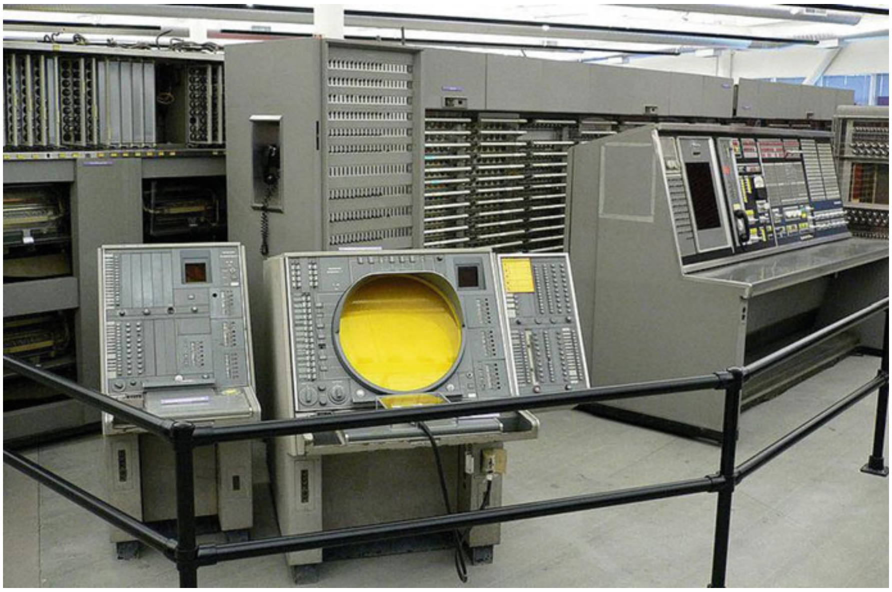


Fig. 4.1 SAGE photo (Courtesy of Steve Jurvetson)

scrambled just in time to meet the enemy threat. The radar stations were ground-based, and therefore needed to send interception instructions to the fighter aircraft.

However, after the war, the speed of aircraft increased considerably, thereby reducing the time available to scramble fighter aircraft. This necessitated a more efficient and automatic way to transmit interception instructions, and the SAGE system was designed to solve this problem and to provide security to the United States. SAGE analysed the information that it received from the various radar stations around the country in real time, and it then automated the transmission of interception messages to fighter aircraft (Fig. 4.1).

IBM and MIT played an important role in the design and development of SAGE. Some initial work on real-time computer systems had been done at Massachusetts Institute of Technology on a project for the US Navy. This project was concerned with building an aircraft flight simulator computer for training bombing crews, and it led to the development of the Whirlwind computer. This computer was originally intended to be analog but instead became the Whirlwind digital computer used for experimental development of military combat information systems.

Whirlwind was the first real-time computer, and George Valley and Jay Forrester wrote a proposal to employ Whirlwind for air defence. This led to the Cape Cod system which demonstrated the feasibility of an air defence system covering New England. Following its successful deployment in 1953, work on the design and development of SAGE commenced.

IBM was responsible for the design and manufacture of the AN/FSQ-7 vacuum tube computer used in SAGE. Its design was based on the Whirlwind II computer,

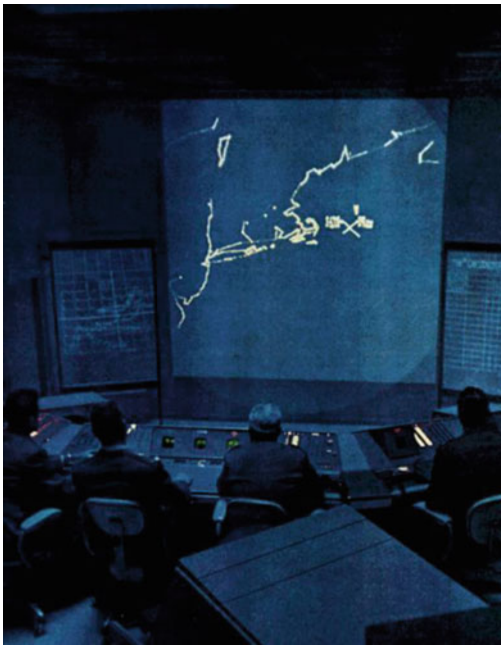


Fig. 4.2 SAGE sector control room (Photo public domain)

which was intended to be the successor to Whirlwind. However, the Whirlwind II was never built, and the AN/FSQ-7 computer weighed 275 tons and included 500,000 lines of assembly code.

The AN/FSQ holds the current world record for the largest computer ever built. It contained 55,000 vacuum tubes, covered an area over 18,000 ft^2 and used about 3 MW of power (Fig. 4.2).

There were 24 SAGE direction centres and three SAGE combat centres located in the United Sates. Each SAGE site included two computers for redundancy, and each centre was linked by long-distance telephone lines. Burroughs provided the communications equipment to enable the centres to communicate with one another, and this was one of the earliest computer networks.

Each site was connected to multiple radar stations with tracking data transmitted by modem over a standard telephone wire. The SAGE computers then collected the tracking data for display on a cathode ray tube (CRT). The console operators at the centre could select any of the targets on the display to obtain information on the tracking data. This enabled aircraft to be tracked and identified, and the electronic information was presented to operators on a display device.

The engineering effort in the SAGE project was immense, and the total cost is believed to have been several billion US dollars. It was a massive construction project which involved erecting buildings and building power lines and communication links between the various centres and radar stations.

SAGE influenced the design and development of the Federal Aviation Authority (FAA) automated air traffic control system.

4.3 IBM Contributions

Chapter 6 presents a short account of the history of IBM and its contributions to the computing field. The company has a long and distinguished history, and the objective of this section is to give a flavour of some of important developments at the company during the 1950s and 1960s. The reader is referred to Chap. 6 for more detailed information.

IBM introduced its first large computer, the 701, in 1952. This was based on vacuum tube technology, and it was used mainly for government work and business applications.

It introduced the popular 650 in 1954. This was an intermediate-sized computer, which was widely used in business computing during the 1950s up to the early 1960s. It was a very successful product for IBM, and over 2,000 of these machines built and sold from its product launch in 1954 to its retirement in 1962.

The 704 was a large computer introduced by IBM in 1954. It was designed by Gene Amdahl and John Backus, and it was used for scientific and commercial applications.

IBM's first completely transistorised machine was the 608 which was introduced in 1957. This was followed by the 7090 which was a large-scale transistorised computer that was used for scientific applications. It was introduced in 1958 (Fig. 4.3).

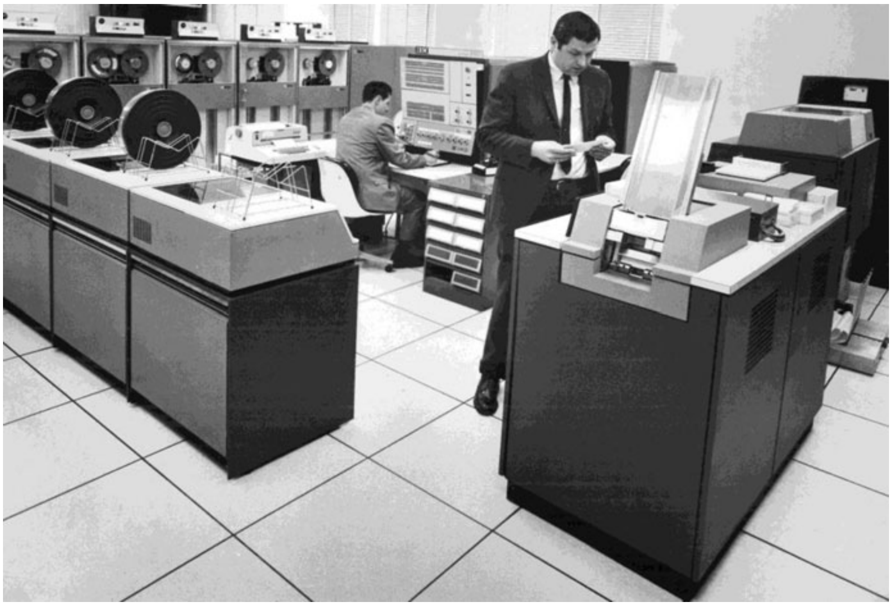


Fig. 4.3 IBM 360 Model 30 (Courtesy of IBM archives)

IBM introduced a new generation of electronic computing equipment known as the IBM System/360 in 1964. The IBM chairman, Thomas Watson, called the event the most important product announcement in the company's history. The chief architect for the IBM 360 was Gene Amdahl, and the S/360 project manager was Fred Brooks.

The IBM 360 was a family of small to large computers, and it offered a choice of five processors and 19 combinations of power, speed and memory. There were 14 models in the family. The concept of a '*family of computers*' was a *paradigm shift* away from the traditional *'one-size-fits-all' philosophy of the computer industry*, as up until then, every computer model was designed independently. The family of computers ranged from minicomputers, with 24 kB of memory, to supercomputers for US missile defence systems. However, all these computers had the same user instruction set, and the main difference was that the larger computers implemented the more complex machine instructions with hardware, whereas the smaller machines used microcode.

The S/360 was used extensively in the Apollo mission to place man on the moon. The contribution by IBM computers and personnel were essential to the success of the project. IBM invested over $5 billion in the design and development of the S/360. However, the gamble paid off, and it was a very successful product line for the company.

4.4 PDP Minicomputers

The programmable data processor (PDP) computer refers to a series of minicomputers developed by the Digital Corporation[1] (also known as DEC) from the 1960s until the 1990s. The various PDP machines can be grouped by word length. These minicomputers were very popular in the scientific and engineering communities during the 1970s and 1980s.

The first PDP computer developed was the PDP 1 which was introduced in 1960. This was an 18-bit machine with an early time-sharing operating system. One of the earliest computer games, Spacewar, was developed for this computer by Steve Russell and others at MIT. The machine could be used to play music in four-part harmony.

It had 9 kB of main memory which was upgradable to 144 kB. It had a clock speed of 200 kHz, and most arithmetic instructions took 10 μs (100,000 operations per second). An instruction had two memory cycles: one for the instruction and one for the data fetch operand.

[1] Digital Corporation was taken over by Compaq in the late 1990s. Compaq was subsequently taken over by HP.

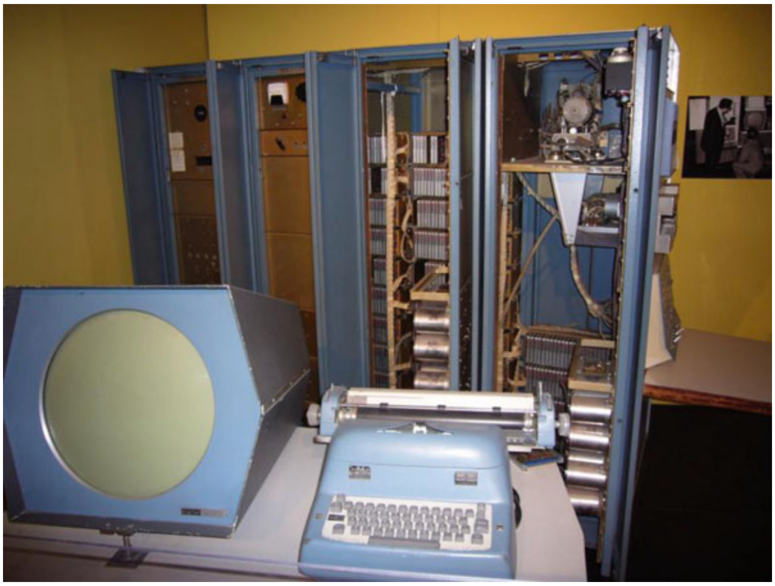


Fig. 4.4 PDP 1 computer (Photo courtesy of Matthew Hutchinson)

It used punched paper tape as the primary storage medium. This was a cumbersome approach as paper tape is difficult to edit. It represented signed numbers in one's complement notation (Fig. 4.4).

The first PDP-1 computer was delivered to Bolt, Beranek and Newman (BBN) Technologies in 1960.

4.4.1 Spacewar Computer Game

Spacewar was one of the earliest computer games. It was developed by Steve Russell and others at Massachusetts Institute of Technology. The game was developed on the PDP-1 computer, and its first version was available in early 1962. The game involves two armed spaceships trying to shoot one another while manoeuvring to avoid a nearby star.

Each ship fires missiles, and it has a limited amount of missiles and fuel available. Each player controls one of the ships and must attempt to shoot the other ship and avoid the star. The player has various controls including rotation, thrust, fire and hyperspace. Hyperspace could be used as a last resort by a player to avoid a missile; however, re-entry from hyperspace was random, and its use increased the probability of the machine exploding the next time that it was used (Fig. 4.5).

The PDP-8 was a very successful 12-bit minicomputer introduced by DEC in 1965. Its product sales were in excess of 50,000 machines. There were

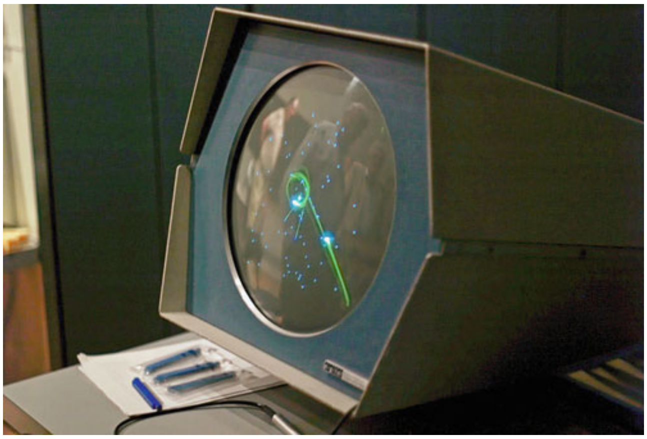


Fig. 4.5 Spacewar (Photo courtesy of Joi Ito)

several models of the machine, and the PDP-8/E was well regarded with several types of I/O devices available for it. It was often configured as a general-purpose computer, and its low cost made the computer available to many new users.

The PDP-11 was a series of 16-bit minicomputer originally introduced in 1970 with sales continuing up to the mid-1990s. It was a commercial success for DEC, and the VAX11/780 super minicomputer is a 32-bit extension of the PDP-11. It was introduced in 1977.

4.5 Home Computers

The Altair 8080 was the first commercially successful home computer, and it was developed by Ed. Roberts at MITS (Micro Instrumentation Telemetry Systems). It featured on the cover of the January 1975 edition of the magazine, *Popular Electronics*. The cost of the machine assembly kit was $439, and it was initially purchased and assembled by computer hobbyists. The Altair 8080 kit came with a front panel, a CPU board with the Intel 8080 microprocessor, 256 bytes of RAM, a 4-slot back plane and an 8-amp power supply. MITS had sold over 5,000 machines by August 1975.

Bill Gates and Paul Allen saw the magazine and started writing software for the machine. They spent a short time working with MITS and founded Microsoft later that year. Their first product was the Altair BASIC interpreter.

Apple Computers was founded by Steve Jobs and Steve Wozniak on 1 April 1976. The Apple I computer was introduced later that year, and the machine had 4 K of RAM (expandable to 8 K), 256 bytes of ROM and was intended to be used

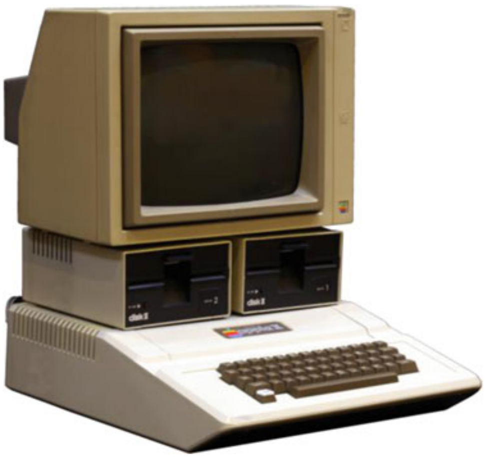


Fig. 4.6 Apple II computer (Photo public domain)

those for whom computing was a hobby. It did not have a case, power supply, keyboard or display, and these needed to be supplied by the user.

The Apple II computer was a significant advance on the Apple I, and it was released in 1977. It was a popular 8-bit home computer and was one of the earliest computers to have a colour display. It had the BASIC programming language built in; it contained 4 K of RAM (which was could be expanded to 48 K). The VisiCalc spreadsheet program was released on the Apple II, and this helped to transform the computer into a credible business machine. The Apple II and its successors were very successful (Fig. 4.6).

The Commodore PET (Personal Electronic Transactor) was a home computer introduced by Commodore International in early 1977. It was the first computer produced by Commodore, which was up to then an electronics company that sold calculators. Commodore realised that the future was in computers rather than calculators, and it purchased MOS Technology in 1976 as part of its strategy of building a home computer. MOS had designed a simple home computer based on their new 6502 microprocessor.

The Commodore PET was a reasonably successful machine, and it became popular in schools and became known as ‘Teacher’s Pet’. It was a single board computer design in a metal case. However, it was not as popular as a home computer since its sound and graphics capabilities were limited. This was addressed in later models such as the VIC-20 which sold over 2.5 million units.

Its keyboard was quite small and was not popular with users. The machine used the 6502 microprocessor, 4 K or 8 K of RAM and a small monochrome monitor with 40 × 25 graphics. The computer’s main board contained four expansion ports, extra memory and a parallel port.

Commodore introduced the Commodore 64 in the early 1980s, and this popular home computer sold in excess of 22 million units. It had excellent sound and graphics.

4.6 Review Questions

1. Describe the IBM System/360 and its impact on the computer field.
2. Describe the SAGE defence system.
3. Describe the PDP family of computers developed by DEC.
4. Describe the development of the home computer.

4.7 Summary

There were major changes to the design of computers between the 1950s and 1970s. Initially, computers employed vacuum tubes, but these were large, bulky and unreliable. The development of the transistor by Shockley and others at Bell Labs provided an alternative to this bulky technology. Transistors were smaller and more reliable. These were later replaced by integrated circuits which allowed hundreds of thousands of transistors to be placed on a piece of silicon.

The SAGE system was developed as part of the airborne defence of the United States in the late 1950s. It was a massive engineering project involving the construction of buildings, power lines and communication infrastructure.

IBM developed several computers during the period. Its first vacuum tube computer was the 701 which was introduced in 1952. This was followed by the popular 650, which was used for business computing. It was a very successful machine with over 2,000 of these machines sold. Its first transistorised machine was the 608, and this was followed by the introduction of a large transistorised machine, the 7090, in 1958. This machine was used for scientific applications.

The IBM System/360 was a family of small to large computers, and it offered the customer a choice of the size and computational power rather than a one-size-fits-all approach. It ranged from minicomputers with 24 kB of memory to supercomputers for US missile defence systems.

It allowed a customer to start with a small member of the S/360 family and to upgrade over time in accordance to their needs to a larger computer in the family.

The PDP computers were designed and developed by the Digital Corporation. The PDP-1 was released in 1960, and this was an 18-bit machine. It could be used to play music in four-part harmony and one of the earliest computer games.

The early home computers were of interest to mainly computer hobbyists rather than a mass consumer market. The Altair 8080 was the first successful home computer, and it was introduced in early 1975. The Apple I computer was introduced by Apple in 1976. It was followed by the Apple II computer in 1976, and this machine was popular for business use as well as home use.

Chapter 5
Revolutions in the 1980s and 1990s

Key Topics

IBM PC Revolution
Mobile Phone Technology
World Wide Web

5.1 Introduction

The 1980s and 1990s were a time of fundamental change in the computing field. The industry moved from a world dominated by mainframe computers to a brave new world dominated by networks of personal computers. The invention of the World Wide Web was a revolution in computing, and it has altered consumer and business behaviour. The speed of microprocessors improved dramatically during the period, and there were large increases in memory and storage in personal computers. This increase in processing power has transformed computers from machines dedicated to business or scientific use to sophisticated machines that may play music or videos or engage in multimedia communication.

Personal computers were now affordable to ordinary consumers, and this led to a large consumer market for home computers and the application software to run on the computers. It was now feasible to have a personal computer on every employee's desk, and this led to an increase in employee productivity as well as a demand for business applications for personal computers. The early applications included editors such as the 'Brief programming editor' which was popular with computer programmers, the 'Wordstar word processor' application which was a text-based word-processing system which employed a markup language and the Norton Utilities which included a set of tools to analyse the hard disk of a machine and allowed a file that had been accidently deleted to be recovered.

The Lotus 123 spreadsheet application was developed in the mid-1980s. It is a spreadsheet program and included charting and graphing capabilities, and it was developed by the Lotus Corporation (now part of IBM). However, 123 was later eclipsed by Microsoft Excel.

The Analogue Mobile Phone System (AMPS) was a first-generation mobile phone system standard developed by Bell Labs. It was introduced into North America in the early 1980s and was a paradigm shift in communication. Communication had been traditionally between places, but with mobile phone technology it was now between people. By the late 1980s, mobile phone technology was being introduced worldwide by companies such as Motorola and Ericsson.

The analog mobile phone networks allowed voice communication only, as text messaging and more advanced features only became available in later generations of mobile telephony. The quality of the voice communication was weak compared to today's technology, as it was susceptible to static and noise, and had no protection from eavesdropping using a scanner. It used frequency division multiple access (FDMA) with separate frequencies (or channels) used for each conversation.

The World Wide Web was invented by Tim Berners-Lee at CERN in the early 1990s. It is a major milestone in computing and has altered business and consumer behaviour. It allows business to operate in a globalised world and is an essential part of modern business.

5.2 The Personal PC Revolution

IBM introduced the IBM Personal Computer (or PC) in 1981 as a machine to be used by small businesses and personal users in the home. The IBM strategy at the time was to get into the home computer market that was then dominated by Commodore, Atari and Apple. It was the cheapest computer produced up to then and was priced at $1,565. It offered 16 kB of memory (expandable to 256 kB), a floppy disk and a monitor. The IBM personal computer became an immediate success and became the industry standard.

The goal was to get the personal computer to the market quickly, and IBM assembled a small team of twelve people led by Don Estridge to achieve this. They designed and developed the IBM PC within 1 year, and as time to market was a key driver, they built the machine with 'off-the-shelf' parts from a number of equipment manufacturers. They had intended using the IBM 801 processor developed at the IBM Research Centre in Yorktown Heights, but decided instead to use the Intel 8088 microprocessor which was inferior to the IBM 801. They chose the PC-DOS operating system from Microsoft rather than developing their own operating system (Fig. 5.1).

The unique IBM elements in the personal computer were limited to the system unit and keyboard. The team decided on an open architecture so that other manufacturers could produce and sell peripheral components and software without purchasing a licence. They published the IBM PC Technical Reference Manual

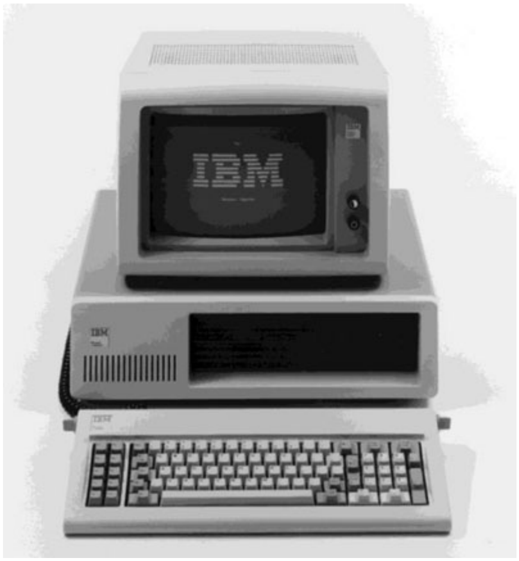


Fig. 5.1 IBM personal computer (Courtesy of IBM archives)

which included the complete circuit schematics, the IBM ROM BIOS source code and other engineering and programming information.

The open architecture led to a new industry of ‘IBM-compatible’ computers which had all of the essential features of the IBM PC but were cheaper. The terms of the licensing of PC-DOS operating system gave Microsoft the rights to the MS-DOS operating system used on the IBM compatibles and led inexorably to the rise of the Microsoft Corporation. The IBM Personal Computer XT was introduced in 1983. This model had more memory, a dual-sided diskette drive and a high-performance fixed-disk drive. It was followed by the Personal Computer/AT introduced in 1984.

The development of the IBM PC meant that computers were now affordable to ordinary users, and this led to a huge consumer market for personal computers and software. It led to the development of business software such as spreadsheets and accountancy packages, banking packages, programmer developer tools such as compilers and specialised editors, and computer games.

The Apple Macintosh was announced during a famous television commercial aired during the Super Bowl in 1984. It was quite different from the IBM PC in that it included a friendly and intuitive graphical user interface, and the machine was much easier to use than the standard IBM PC. The latter was a command-driven operating system and required the users to be familiar with its commands. The introduction of the Mac GUI was an important milestone in the computing field.

However, the Apple Macintosh was more expensive than the IBM PC, and cost proved to be a decisive factor for consumers when purchasing a personal computer. The IBM PC and the various IBM-compatible computers remained dominant.

The introduction of the personal computer represented a paradigm shift in computing, and it led to a fundamental change in the way in which people worked.

It placed the power of the computer directly in the hands of millions of people. The previous paradigm was that an individual user had limited control over a computer, and the access privileges of the individual users were strictly controlled by the system administrators. The subsequent introduction of the client-server architecture led to the linking of the personal computers (clients) to larger computers (servers). These servers contained large amounts of data that could be shared with the individual client computers.

The IBM strategy in developing the IBM personal computer was deeply flawed and cost the company dearly. IBM had traditionally produced all of the components for its machines, but with its open architecture model, any manufacturer could produce an IBM compatible machine. It outsourced the development of the microprocessor chip to Intel, and this led to Intel eventually becoming the dominant power in the microprocessor industry. The development of the operating system, PC-DOS (PC disk operating system), was outsourced to a small company called Microsoft. This proved to be a major error by IBM as the terms of the deal with Microsoft were favourable to the latter and allowed it to sell its own version of PC-DOS (MS-DOS) to other manufactures as the operating system for IBM compatibles.

5.3 The Mobile Phone Revolution

The origins of the mobile phone revolution dates back to work done on radio technology from the 1940s. Bell Labs had proposed the idea of a cellular system back in 1947, and it was eventually brought to fruition by researchers at Bell Labs and Motorola.

The world's first commercial mobile phone went on sale in 1983. It was the Motorola DynaTAC 8000X and was popularly known as the 'brick' due to its size and shape. It weighed 28 ounces and was 13.5″ in length and 3.5″ in width; it had a LED display and could store 30 numbers; it had a talk time of 30 min and 8 h of standby. It cost $3,995 and was too expensive apart from wealthy consumers.

The team at Motorola that developed the DynaTAC8000X was led by Martin Cooper, and he made the first mobile phone call. The DynaTAC today is a collectors' item as most analog networks are now obsolete (Fig. 5.2).

The Analogue Mobile Phone System (AMPS) was a first-generation mobile phone system standard developed by Bell Labs, and it was introduced into North America in the early 1980s. It allowed voice communication only, and the voice communication was susceptible to static and noise and had no protection from eavesdropping using a scanner. It used frequency division multiple access (FDMA), and a separate frequency (or channel) was used for each conversation.

The second generation (2G) of mobile technology was digital, and it was designed to replace the analog technology. It used the Global Systems Mobile (GSM) standard developed by the European Telecommunications Standards Institute (ETSI). It is a cellular network which means that mobile phones connect to it by searching cells in

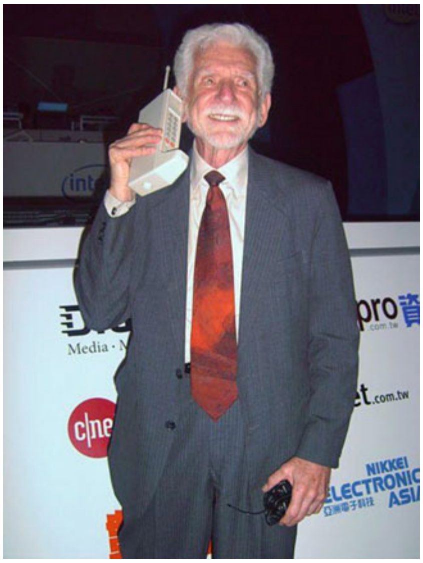


Fig. 5.2 Martin Cooper re-enacts DynaTAC call (Public domain)

the immediate vicinity. The first GSM call was made by the Finnish prime minister in Finland in 1991, and the first short message service (SMS) or text message was sent in 1992.

One key feature of GSM is its use of the Subscriber Identity Module (SIM) card. This is a detachable smartcard that contains the user's subscription information and phone book. The SIM card may be used on other GSM phones, and this is useful when the user purchases a replacement phone. GSM provides an increased level of security with communication between the subscriber and base station encrypted.

GSM networks continue to evolve, and GPRS (2.5G) became available in 2000. Third-generation mobile UMTS (3G) provides mobile broadband multimedia communication. Mobile phone technology has successfully transformed the earlier paradigm of communication between places to communication between people.

5.4 Birth of the World Wide Web

The invention of the World Wide Web by Tim Berners-Lee in 1990 at CERN in Switzerland is described in Chap. 8. It is a revolutionary milestone in computing, and it has transformed the Internet from mainly academic use to where it is now an integral part of peoples' lives. Users may surf the web, that is hyperlink among the millions of computers in the world and obtain information easily.

Berners-Lee had been working on a key problem facing CERN in the late 1980s. This international centre had a large permanent staff and many visiting scientists, and it was a challenge to keep track of people, computers, documents and databases. Further, it lacked an effective way to share information among the scientists. Berners-Lee solution to this problem was the invention of the World Wide Web, and so while intended initially as a solution to a particular problem, the invention had global applications.

The major leap that Berners-Lee made was essentially a marriage of the existing technologies of the Internet, hypertext and the mouse into what has become the World Wide Web. He created a system that gives every web page a standard address called the universal resource locator (URL). Each page is accessible via the hypertext transfer protocol (HTTP), and the page is formatted with the hypertext markup language (HTML). Each page is visible using a web browser. Its features include:

The early browsers included Gopher, developed at the University of Minnesota, and Mosaic, developed at the University of Illinois. These were replaced in later years by Netscape which dominated the browser market until Microsoft developed Internet Explorer. Microsoft gave away its browser for free and bundled it in its windows operating system, and thereby led to the dominance of Internet Explorer. The browser wars are discussed in Chap. 7.

The development of the graphical browsers led to the commercialisation of the World Wide Web. Today, there are many browsers available (e.g. Mozilla Firefox, Google Chrome and Safari). Most of these browsers are available free of charge to the user, and users are unlikely to be interested in purchasing a browser given that there are existing high-quality open-source browsers such as Firefox available free of charge.

5.5 Review Questions

1. Discuss the invention of the personal computer and its impacts on society.
2. Discuss the invention of the mobile phone and its impact on society.
3. Discuss the invention of the World Wide Web and its impact on society.

5.6 Summary

The 1980s and 1990s were a time of fundamental change in the computing field. The industry moved from a world dominated by mainframe computers to a world dominated by networks of personal computers. The invention of the World Wide Web was a revolution, and it has transformed consumer and business behaviour leading to a world that is, in effect, a global village. The speed of microprocessors improved dramatically during the period, and there were large increases in memory and storage of personal computers.

The development of the personal computer meant that computers were now affordable to ordinary consumers as well as business. This led to a consumer market for home computers and for software applications to run on these computers.

The introduction of the mobile phone was a paradigm shift from communication between places to that between people. By the late 1980s, mobile phone technology was being introduced worldwide by companies such as Motorola and Ericsson.

The analog mobile phone networks allowed voice communication only. The quality of the voice communication was poor as it was susceptible to static and noise. Further, there was no protection from eavesdropping using a scanner, but these problems were addressed by later generations of mobile phone technology. Text messaging and more advanced features only became available in the later GSM system.

Chapter 6
IBM

Key Topics

Hermann Hollerith
Thomas Watson Sr. and Jr.
Harvard Mark 1 (IBM ASCC)
IBM 701
IBM system/360
CICS
IBM personal computer

6.1 Introduction

This chapter considers the history of International Business Machines (IBM). This company is a household name and has a long and distinguished history. It is a major corporation that has made major contributions to the computing field and in developing the computers that we are familiar with today. Its origins go back to the processing of the 1880 population census of the United States.

The processing of this census was done manually, and it took 7 years to complete. It was predicted that the 1890 census would take in excess of 10 years to process, and the US Census Bureau recognised that its current methodology was no longer fit for purpose. It held a contest amongst its employees to find a more efficient methodology to tabulate the census data, and the winner was Hermann Hollerith who was the son of a German immigrant.

His punch card tabulating machine used an electric current to sense holes in punch cards, and it kept a running total of the data. The new methodology enabled the results of the 1890 census to be available in 6 weeks, and the population was recorded to be over 62 million.

Fig. 6.1 Hermann Hollerith (Courtesy of IBM Archives)

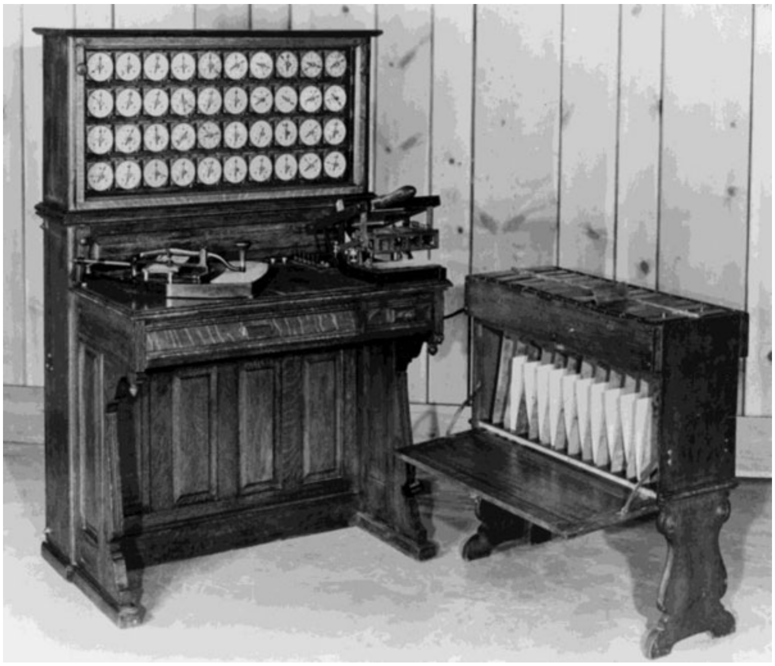


Fig. 6.2 Hollerith's tabulator (1890) (Courtesy of IBM Archives)

Hollerith formed the Tabulating Machine Company in Washington, DC, in 1896, and this was the first electric tabulating machine company. It later merged with the International Time Recording Company to form the Computing Tabulating Recording Company (CTR) in 1911. The company changed its name to IBM in 1924 (Fig. 6.1).

Thomas Watson Sr. became president of CTR in 1914 and transformed the company into a highly successful international selling organisation based around punch card tabulating machines. He was responsible for the famous 'THINK' signs that have been associated with IBM for many years. They were introduced in 1915 (Fig. 6.2).

Fig. 6.3 Thomas Watson Sr. (Courtesy of IBM Archives)

Watson considered the motivation of the sales force to be an essential part of his job, and the sales people were required to attend an IBM sing-along. The verses in the songs were in praise of IBM and its founder Thomas Watson Sr.

These songs were published as a book titled *Songs of the IBM* in 1931 and included ‘Ever Onward’, ‘March on with IBM’ and ‘Hail to the IBM’. Watson renamed CTR to International Business Machines (IBM) in 1924. It employed over 3,000 people, had revenues of $11 million and a net income of $2 million. It had manufacturing plants in the United States and Europe. By 2005, IBM had revenues of over $91 billion, net income of $7.9 billion, and it employed over 300,000 people.

IBM has successfully adapted its business to a changing world for over a 100 years. Its early products were designed to process, store and retrieve information from tabulators and time-recording clocks. Today, the company is an industry leader that produces powerful computers and global networks and provides professional services to its customers around the world (Fig. 6.3).

It developed the popular IBM punch card in the late 1920s which provided almost double the capacity of existing cards. The company introduced a mechanism by which staff could make improvement suggestions in the 1920s.

The great depression of the 1930s affected many Americans. Its impact on IBM was minimal, and IBM’s policy was to take care of its employees. It was one of the first corporations to provide life insurance and paid vacations for its employees. Watson kept his workers busy during the depression by producing new machines even while demand was slack. He also won a major government contract to maintain employment records for over 26 million people.

He recognised the importance of research and development and created a division in the early 1930s to lead the engineering and research and development efforts for the entire IBM product line. The education and development of employees was seen as essential to the success of its business. IBM launched an employee and customer magazine called ‘Think’ in the 1930s. This magazine included topics such as education and science.

IBM placed all of its plants at the disposal of the US government during the Second World War, and it expanded its product line to include military equipment. It commenced work on computers during the war years with the Harvard Mark I

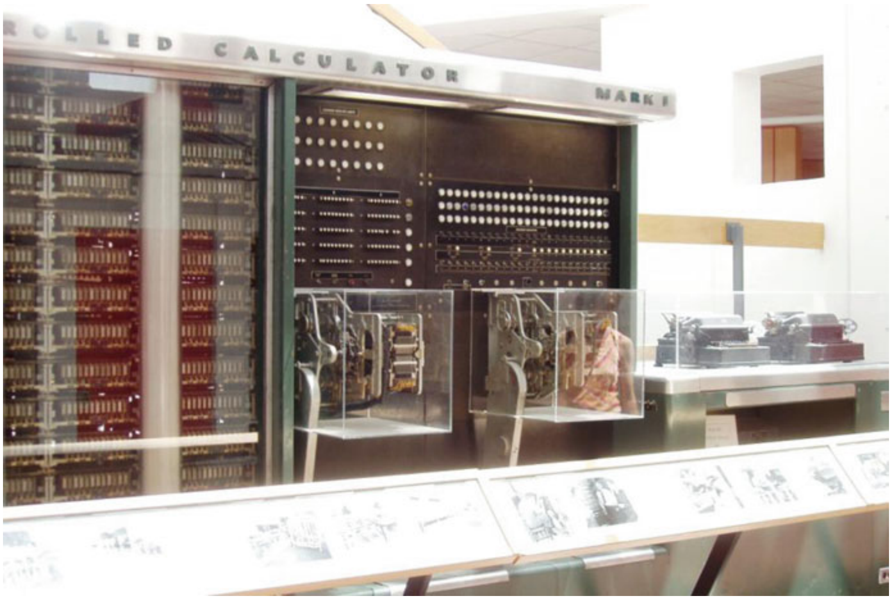


Fig. 6.4 Harvard Mark 1 (IBM ASCC) (Photo public domain)

(also known as the IBM Automatic Sequence Controlled Calculator (ASCC)). It was completed in 1944 and presented to Harvard University. It was essentially an electromechanical calculator that could perform large computations automatically.

The machine was designed by Howard Aiken of Harvard University to assist in the numerical computation of differential equations. It was funded and built by IBM and was 50 ft long, 8 ft high and weighed 5 tons. It performed additions in less than a second, multiplications in 6 s and division in about 12 s. It used electromechanical relays to perform the calculations.

The ASCC could execute long computations automatically. It used 500 miles of wiring and over 700,000 components. It was the industry's largest electromechanical calculator and had 60 sets of 24 switches for manual data entry. It could store 72 numbers, each 23 decimal digits long (Fig. 6.4).

The announcement of the Harvard Mark 1 led to tension between Aiken and IBM, as Aiken announced himself as the sole inventor without acknowledging the important role played by IBM.

6.2 Early IBM Computers

The company developed the Vacuum Tube Multiplier in 1943 which was an important move from electromechanical to electronic machines. It was the first complete machine to perform arithmetic electronically by substituting vacuum tubes for electric relays. The key advantages of the vacuum tubes is that they

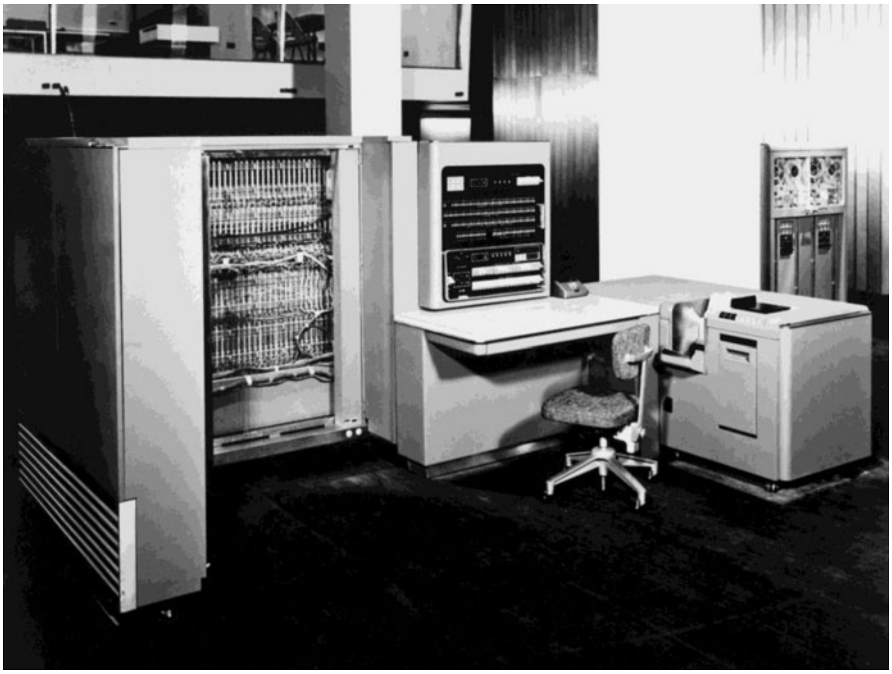


Fig. 6.5 IBM 701 (Courtesy of IBM Archives)

Fig. 6.6 Thomas Watson Jr. (Courtesy of IBM Archives)

were faster, smaller and easier to replace than the electromechanical switches used in the Harvard Mark I. This allowed engineers to process information thousands of times faster (Fig. 6.5).

It introduced its first large computer based on the vacuum tube in 1952. This machine was called the IBM 701, and it executed 17,000 instructions per second. It was used mainly for government work and for business applications. Thomas Watson, Sr., retired in 1952, and his son, Thomas Watson, Jr., became chief executive officer the same year (Fig. 6.6).

Thomas Watson, Jr., believed that the future of IBM was in computers rather than in tabulators. He recognised the future role that computers would play in business and realised that IBM needed to change to adapt to the new technology. He played a key role in the transformation of IBM to a company that would become the world leader in the computer industry.

IBM introduced the IBM 650 (Magnetic Drum Calculator) in 1954. This was an intermediate-sized electronic computer designed to handle accounting and scientific computations. It was the first mass-produced computer and was used by universities and businesses from the 1950s to the early 1960s. It was a very successful product for IBM with over 2,000 built and sold between its product launch in 1954 and its retirement in 1962. The machine included a central processing unit, a power unit and a card reader.

The IBM 704 data processing system was a large computer introduced in 1954. It included core memory and floating-point arithmetic and was used for scientific and commercial applications. It included high-speed memory which was faster and much more reliable than the cathode ray tube memory storage mechanism used in earlier machines. It also had a magnetic drum storage unit which could store parts of the program and intermediate results.

The interaction with the system was either by magnetic tape or punch cards entered through the card reader. The program instructions or data were initially produced on punch cards. They were then either converted to magnetic tape or read directly into the system, and data processing performed. The output from the processing was then sent to a line printer, magnetic tape or punch cards. Multiplication and division was performed in 240 µs.

The designers of the IBM 704 included John Backus and Gene Amdahl. Backus was one of the key designers of the FORTRAN programming language introduced by IBM in 1957.This was the first scientific programming language and is used by engineers and scientists. Gene Amdahl later founded the Amdahl Corporation in 1970, and this company later became a rival to IBM in the mainframe market.

The first IBM product to use transistor circuits without vacuum tubes was the 608. This transistorised calculator was introduced in late 1957, and it contained 3,000 germanium transistors. It was similar to the operation of the older vacuum tube 604.

The IBM 7090 which was introduced in 1958 was one of the earliest commercial computers with transistor logic. It was designed for large-scale scientific applications. It was over 13 times faster than the older vacuum tube IBM 701 and could perform 229,000 calculations per second. It used a 36-bit word and had an address space of 32,768 words. It was used by the US Air Force to provide an early warning system for missiles and also by NASA to control space flights. It cost approximately $3 million, but it could be rented for over $60 K per month.

IBM introduced the first computer disk storage system in 1957. This medium was called the Random Access Method of Accounting and Control (RAMAC), and it became the basic storage medium for transaction processing. The RAMAC's random access arm could retrieve data stored on any of the 50 spinning disks.

IBM was involved in the development of the Semi-Automatic Ground Environment (SAGE) early warning system during the Cold War. This was described in more detail in Chap. 4. IBM provided the hardware for the air defence system. The initial installation was completed in 1958, and the system was fully implemented in 1963. It remained operational until 1984.

There were 24 SAGE direction centres and 3 SAGE combat centres located in the United Sates. Each centre was linked by long-distance telephone lines, and Burroughs provided the communications equipment that allowed the centres to communicate with one another. It was one of the earliest computer networks. Each centre contained a large digital computer that automated the information flow and provided real-time control information on aircraft and on weapons systems. It tracked and identified aircraft and presented the electronic information to operators on a display device (cathode ray tube).

The IBM 1401 data processing system and the IBM 1403 printer were launched in 1959. The 1401 was an all-transitorised data processing system, and it was aimed at small businesses. This computer replaced punch card–based tabulating equipment. It included high-speed card punching and reading, magnetic tape input and output and high-speed printing. The 1403 printer was four times faster than any competitor printer. IBM introduced a program termed ‘Speak Up’ to enhance staff communication in 1959. It opened its research division headquarters at Yorktown Heights, New York, in 1961.

6.3 The IBM System/360

Thomas Watson announced the new System/360 to the world at a press conference in 1964 and said:

> The System/360 represents a sharp departure from concepts of the past in designing and building computers. It is the product of an international effort in IBM’s laboratories and plants and is the first time IBM has redesigned the basic internal architecture of its computers in a decade. The result will be more computer productivity at lower cost than ever before. This is the beginning of a new generation – not only of computers – but of their application in business, science and government.

The IBM 360 was a family of small to large computers, and the concept of a ‘family of computers’ was a paradigm shift away from the traditional ‘one size fits all’ philosophy of the computer industry. The family ranged from minicomputers with 24 kB of memory to supercomputers for US missile defence systems. However, all these computers had the same user instruction set, and the main difference was that the larger computers implemented the more complex machine instructions with hardware, whereas the smaller machines used microcode (Fig. 6.7).

Its architecture potentially allowed customers to commence with a lower-cost computer model and to then upgrade over time to a larger system to meet their evolving needs. The fact that the same instruction set was employed meant that the time and expense of rewriting software was avoided.

The S/360 was used extensively in the Apollo project to place man on the moon. The contribution by IBM computers and personnel were essential to the success of the project. It was a very successful product line.

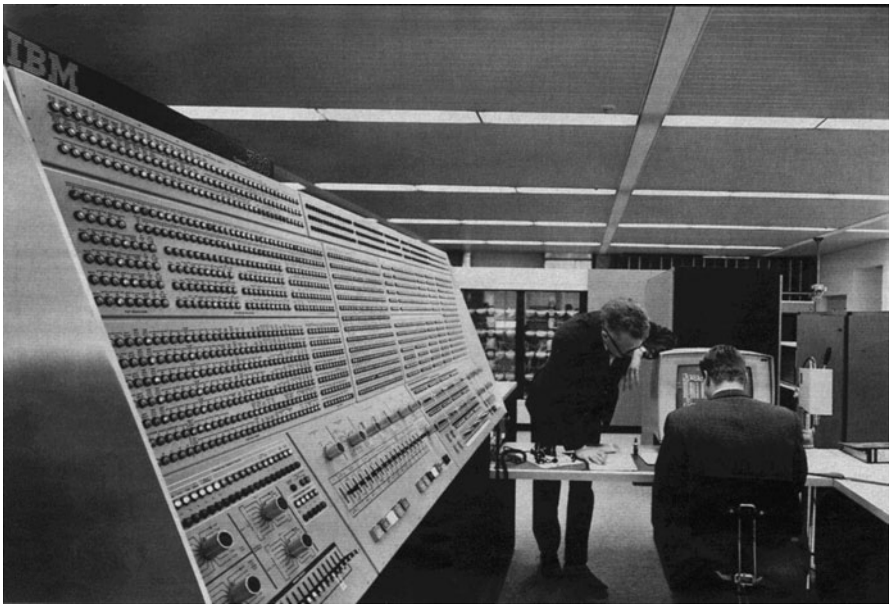


Fig. 6.7 IBM System/360 Model 91 at NASA in the 1960s (Public domain)

6.4 The 1970s

IBM introduced the Customer Information Control System (CICS) in 1969. This is a transaction processing system designed for online and batch processing. It was originally developed at IBM's Palo Alto laboratory, but development moved to IBM's laboratory at Hursley, England, from the mid-1970s.

It is used by banks and insurance companies in the financial sector for their core business functions. It can support several thousand transactions per second and up to 300 billion transactions flow through CICS every day. It is available on large mainframes and on several operating systems, including Z/OS, AIX, Windows and Linux. CICS applications have been written in COBOL, PL/1, C and Java.

The IBM System/370 was introduced in 1970. It was backwards compatible with the older 360 system (i.e. programs that ran on the 360 could still run on the 370). This made it easier for customers to upgrade from their System/360 to the System/370. The S/370 employed virtual memory.[1]

[1] Virtual memory was developed for the Atlas Computer at Manchester University in England in the early 1960s. It allowed the actual memory space of a computer to appear much larger by using the space available on the hard drive. The Atlas Computer was a joint venture between the Manchester University, Ferranti, and Plessey.

The floppy disk was introduced in 1971, and it became the standard for storing personal computer data. The IBM 3340 Winchester disk drive was introduced in 1973. It doubled the information density on disk surfaces and included a small light read/write head that was designed to ride on an air film that was 18×10^{-6} in. thick. Winchester technology was employed up to the early 1990s.

IBM introduced the Systems Network Architecture (SNA) networking protocol in 1974, and this protocol provided a set of rules and procedures for communication between computers. It remained an important standard until the open architecture standards appeared in the 1990s.

It introduced the IBM 5100 Portable Computer in 1975 which cost under $20,000. This was a desktop machine used by engineers and scientists. IBM's Federal Systems Division built the flight computers and special hardware for the space-shuttle program.

IBM developed the Data Encryption Standard (DES) in the mid-1970s. DES provides a high degree of security during the transmission of data over communication channels. It specifies an algorithm that enciphers and deciphers a message. The effect of enciphering is to make the message meaningless to unauthorised viewing as the task of breaking the encryption algorithm is extremely difficult.

6.5 The IBM Personal Computer Revolution

This section provides a brief overview of the introduction of the IBM Personal Computer, and it was discussed in more detail in Chap. 5. The IBM Personal Computer (or PC) was introduced in 1981, and it was intended to be used by small businesses and personal users in the home. Its price was $1,565, and it provided 16 kB of memory (that was expandable to 256 kB), a floppy disk and a monitor. The IBM Personal Computer became an immediate success and became the industry standard.

It led to a new industry of 'IBM-compatible' computers which had all of the essential features of the IBM PC but were cheaper. The IBM Personal Computer XT was introduced in 1983. This model had more memory, a dual-sided diskette drive and a high-performance fixed-disk drive. It was followed by the Personal Computer/AT introduced in 1984.

The development of the IBM PC meant that computers were now affordable to ordinary users, and this led to a huge consumer market for personal computers and software.

The introduction of the personal computer represented a paradigm shift in computing, and it placed the power of the computer directly in the hands of millions of people.

IBM had traditionally produced all of the components for its machines. However, it outsourced the production of components to other companies for the

IBM PC. This proved to be a major mistake as they outsourced the production of the processor chip to a company called Intel,[2] and the development of the Disk Operating System (DOS) was outsourced to a small company called Microsoft.[3] Intel and Microsoft would later become technology giants.

6.6 The 1980s and 1990s

The IBM token-ring local area network was introduced in 1985. This enabled personal computer users to share printers, files and information within a building. It is essentially a computer network in which all of the computers are arranged in a circle (or ring). There is a special data frame termed a token, and the token moves from computer to the next computer until it arrives at a computer that needs to transmit data. This computer then converts the token frame into a data frame for transmission; that is, the computer that wishes to transmit catches the token, attaches a message to it and then sends it around the network. The token-ring network later became the IEEE 802.5 standard.

The Ethernet local area network was developed by Robert Metcalfe at Xerox, and its performance was superior to the IBM token-ring network. Ethernet was first published as a standard in 1980, and it was later published as the IEEE 802.2 standard. Metcalfe formed the technology company 3-Com to exploit the Ethernet technology. IBM introduced the Advanced Peer-To-Peer Networking (APPN) architecture in 1984. This was widely used for communication by mid-range systems, and it allowed individual computers to talk to one another without a central server.

IBM developed the reduced instruction set computer (RISC) architecture. This technology boosts computer speed by using simplified machine instructions for frequently used functions. It reduces the time to execute commands and is the basis of most workstations in use today. This led to the design of the RS/6000 and the subsequent development of the Power PC architecture. The RISC System/6000 was introduced in 1990. It is a family of workstations that were amongst the fastest and most powerful in the industry.

IBM introduced the next generation of personal computers termed the Personal System/2 (PS/2) in 1987. It included a new operating system called Operating System/2 (OS/2). The latter gave users of personal computers access to multiple applications and very large programs and data and allowed concurrent communication with other systems. It was the first offering in IBM's Systems Application Architecture (SAA) which was designed to make application programs look and

[2] Intel was founded by Bob Noyce and Gordon Moore in 1968.

[3] Microsoft was founded by Bill Gates and Paul Allen in 1975.

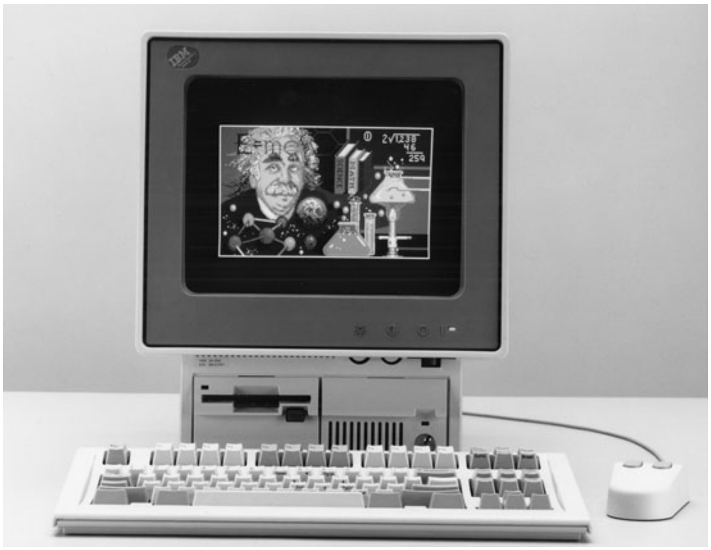


Fig. 6.8 Personal System/2. Model 25. (Courtesy of IBM Archives)

work in the same manner across different systems such as personal computers, mid-range systems and larger systems (Fig. 6.8).

A research group at IBM developed a suite of antivirus tools to protect personal computers from attacks from viruses. This led to the establishment of the High Integrity Computing Laboratory (HICL) at IBM. This laboratory went on to pioneer the science of computer viruses.

IBM researchers introduced very fast computer memory chips in 1988. These chips could retrieve a single bit of information in 2×10^{-8} of a second. IBM introduced the IBM Application System/400 (AS/400) in 1988. This was a new family of easy-to-use computers designed for small and intermediate-sized companies. It became one of the world's most popular business computing systems.

A team of IBM researchers succeeded in storing a billion bits of information (i.e. a gigabit) on a single square inch of disk space in 1989. IBM introduced a laptop computer in 1991 to give customers computing capabilities on the road or in the air.

IBM, Apple Computers and Motorola entered an agreement in 1991 to link Apple computers to IBM networks and to develop a new reduced instruction set microprocessors for personal computers. IBM and Motorola completed development and fabrication of the PowerPC 620 microprocessor in 1994. The new open-systems environment allowed both IBM AIX and Macintosh software programs to run on RISC-based systems from both companies (Fig. 6.9).

IBM created the world's fastest and most powerful general-purpose computer in 1994. This was a massively parallel computer capable of performing 136 billion calculations per second. The increase in computational power of computers was becoming phenomenal.

The Deep Blue computer-programmed chess program defeated Garry Kasparov in 1997. Kasparov was then the existing world champion in chess, and the IBM

Fig. 6.9 Deep Blue processors (Courtesy of IBM Archives)

victory showed that the computational power of computers could match or exceed that of man. It was also the first time that a computer had defeated a top-ranked chess player in tournament play. Deep Blue had phenomenal calculating power, and it could calculate 200 million chess moves per second.

IBM and the US Energy Department introduced Blue Pacific which was the world's fastest computer in 1998. It was capable of performing 3.9 trillion[4] calculations per second and had over 2.6 trillion bytes of memory. An indication of its computability is given by the fact that the amount of calculations that this machine could perform in one second would take a person using a calculator over 63,000 years.

6.7 Challenges for IBM

IBM had traditionally provided a complete business solution to its clients with generally one key business person making the decision to purchase the IBM computer system for the company. The personal computer market and the client/server architecture had now fragmented this traditional market, as departments and individuals could now make their own purchasing decisions. The traditional customer relationship that IBM had with its clients was fundamentally altered. Further, the decision to outsource the development of the microprocessor and the operating system proved to be costly mistakes.

It took IBM some time to adjust to this changing world, and it incurred huge losses of over $8 billion in 1993. The company embarked on cost-cutting measures

[4] We are using the US system with a trillion defined as 10^{12} and a billion defined as 10^9.

which involved reducing its work force and rebuilding its product line. IBM's strength in providing integrated business solutions proved to be an asset in adapting to the brave new world.

IBM faced further challenges to adapt to the rise of the Internet and to network computing. The Internet was another dramatic shift in the computing industry, but IBM was better prepared this time after its painful adjustment in the client/server market. IBM's leadership helped to create the e-business revolution, and IBM actually coined the term 'e-business'. IBM outlined to customers and to its employees how the Internet had the ability to challenge older business models and to transform the nature of transactions between businesses and individuals.

IBM is a highly innovative company and is awarded more patents[5] in the United States than any other company. It earned over 3,000 patents in 2004. The company is a household name, and it has a long and distinguished history. The history of computing is, in many ways, closely related to the history of IBM, as the company has played a key role in the development of the computers that we are familiar with today.

6.8 Review Questions

1. Discuss the contribution of IBM to computing.
2. Discuss the development of the Systems/360 and its importance.
3. Discuss the introduction and impact of the introduction of the IBM personal computer.

6.9 Summary

IBM is a major corporation with a long and distinguished history. Its origins go back to the development by Hermann Hollerith of a tabulating machine to process the population census of the United States in 1890.

IBM became involved in the computer field during the war years with the development of the Harvard Mark 1. This machine was designed by Howard Aiken of Harvard University to assist in the numerical computation of differential equations. It was funded and built by IBM.

IBM played a key role in the implementation of the SAGE air defence system in North America. It provided the hardware for each of the SAGE centres, with each

[5] A patent is legal protection that is given to an inventor and allows the inventor to exploit the invention for a fixed period of time (typically 20 years).

centre containing a large digital computer to automate the information flow, and provided real-time control information on aircraft and weapons systems.

Its first computer, the 701, was introduced in 1952. This was a large vacuum tube computer, and it was followed by the 650 in 1954. This was a popular machine used for business computing. It introduced its first transistorised machine, the 608, in 1957, and this was followed in 1958 by the 7090, a large transistorised machine for business computing.

It introduced the IBM 360 family of small to large computers in 1964, and the family ranged from minicomputers with 24 kB of memory to supercomputers for US missile defence systems. However, all these computers had the same user instruction set, and the main difference was that the larger computers implemented the more complex machine instructions with hardware, whereas the smaller machines used microcode. The S/360 was used extensively in the Apollo project to place man on the moon.

It introduced the IBM Personal Computer (or PC) in 1981, and this machine was intended to be used by small businesses and personal users in the home. The IBM Personal Computer became an immediate success and became the industry standard. However, IBM's strategy in the introduction of the personal computer was flawed, and it enabled companies such as Microsoft and Intel to become industry giants.

IBM is a highly innovative company that has made important contributions to computing.

Chapter 7
Technology Companies

Key Topics

Microsoft
Apple
Digital Research
Digital
Intel
HP
Oracle
Amdahl
Motorola

7.1 Introduction

This chapter considers a selection of technology companies that have made important contributions to the computing field. It is not possible due to space constraints to consider all companies that merit inclusion.

Microsoft was founded in the mid-1970s, and it initially developed software for the Altair 8080 home computer. It was awarded a contract by IBM to develop the Disk Operating System (DOS) for the IBM PC, and the rest is history. It has grown to become a major corporation and has developed operating systems such as MS-DOS, Microsoft Windows and the Microsoft Office suite of products.

Apple was founded in the mid-1970s and developed the Apple I and Apple II home computers. The Apple Macintosh personal computer appeared in the mid-1980s, and it was a much friendlier machine than the IBM PC with a nice graphical user interface (GUI). MS-DOS was command-driven and required the user to know more of the technical details. Apple has been a highly innovative company, and it has developed exciting products that have captured the imagination of consumers. These include the iPod, the iPhone and the iPad.

Other companies discussed in this chapter include HP, Digital, Amdahl, Digital Research, Intel, Motorola, Oracle and Siemens.

7.2 Microsoft

Microsoft was created by Bill Gates and Paul Allen in 1975. Steve Ballmer joined the company in 1980. The first real success of the company was with the Disk Operating System (DOS) for personal computers. IBM had originally intended awarding the contract for the operating system to Digital Research for a version of their CP/M operating system on the forthcoming IBM Personal Computer. However, negotiations between IBM and Digital Research failed in 1981, and IBM awarded the contract to Microsoft to produce a version of the CP/M operating system for its personal computers (Fig. 7.1).

Microsoft purchased a CP/M clone called QDOS and enhanced it for the IBM Personal Computer. IBM renamed the new operating system to PC-DOS, and Microsoft created its own version of the operating system called MS-DOS. The terms of the contract with IBM allowed Microsoft to have control of its own QDOS derivative. This proved to be a major mistake by IBM, as MS-DOS became popular in Europe, Japan and South America. The flood of PC clones on the market allowed Microsoft to gain major market share with effective marketing of its operating system to the various manufacturers of the cloned PCs. This led to Microsoft becoming a major player in the personal computer market.

The company released its first version of Microsoft Word in 1983, and this would later become the world's most popular word processing package. It released

Fig. 7.1 Bill Gates (Public domain)

its first version of Microsoft Windows in 1985, and this product was a graphical extension of its MS-DOS operating system. Microsoft and IBM commenced work in 1985 on a new operating system called Operating System 2 (OS/2) for the IBM PS/2 personal computer.

The Microsoft Office suite of products was introduced in 1989, and these include Microsoft Word, Microsoft Excel and PowerPoint. Microsoft introduced a new operating system, Windows 3.0, in 1990, and it included a friendly graphical user interface. Windows (and its successors) became the most popular operating systems in the coming years. Microsoft's office suite gradually became the dominant office suite with a far greater market share than its competitors such as WordPerfect and Lotus 1-2-3. It released its Windows 3.1 operating system and its Access database software in 1992. Windows was the most widely used GUI operating system by 1993. Windows 95 was released in 1995, Windows NT in 1996, Windows 2000 in 2000 and Windows XP in 2001.

7.2.1 Microsoft Windows and Apple GUI

Apple Computers took a copyright infringement lawsuit against Microsoft in 1988 to prevent Microsoft from using its GUI elements in the Windows operating system. The legal arguments lasted 5 years, and the final ruling in 1993 was in favour of Microsoft. Apple had claimed that the look and feel of the Macintosh operating system was protected by copyright including 189 GUI elements. However, the judge found that 179 of these had already been licensed to Microsoft (as part of the Windows 1.0 agreement), and that most of the ten other GUI elements were not copyrightable.

This legal case generated a lot of interest as some observers considered Apple to be the villain, as they were using legal means to dominate the GUI market and restrict the use of an idea that was of benefit to the wider community. Others considered Microsoft to be the villain with their theft of Apple's work, and their argument was that if Microsoft succeeded, a precedent would be set in allowing larger companies to steal the core concepts of any software developer's work.

The court's judgement seemed to invalidate the copyrighting of the 'look and feel' of an application. However, the judgement was based more on contract law rather than copyright law, as Microsoft and Apple had previously entered into a contract with respect to licensing of Apple's icons on Windows 1.0. Apple had not acquired a software patent to protect the intellectual idea of the look and feel of its Macintosh operating system.

7.2.2 The Browser Wars

Microsoft was initially slow to respond to the rise of the Internet. However, it developed Microsoft Network (MSN) to compete directly against America Online (AOL).

It developed some key Internet technologies such as ActiveX, VBScript and Jscript, and it released a new version of Microsoft SQL Server with built-in support for Internet applications. It released a new version of Microsoft Office and Internet Explorer 4.0 (its Internet browser) in 1997.

This was the beginning of Microsoft dominance of the browser market. Netscape had dominated the market, but as Internet Explorer 4.0 (and its successors) was provided as a standard part of the Windows operating system (and also on Apple computers), it led to the replacement of Netscape by Internet Explorer.

This led to a filing against Microsoft stating that Microsoft was engaged in anticompetitive practices by including the Internet Explorer browser in the Windows operating system, and that Microsoft had violated an agreement signed in 1994. The leaking of internal Microsoft company memos caused a great deal of controversy in 1998 as these documents went into detail of the threat that open-source software posed to Microsoft and mentioned possible legal action against Linux and other open-source software.

The legal action taken by the US Department of Justice against Microsoft concluded in mid-2000, and the judgement called the company an ‘abusive monopoly’. The judgement stated that the company should be split into two parts. However, this ruling was subsequently overturned on appeal.

7.3 Apple

Apple was founded by Steven Wozniak and Steven Jobs at Cupertino in California in 1976. Jobs and Wozniak were two college dropouts, and they released the Apple I computer in 1977. It retailed for $666.66. They then proceeded to develop the Apple II computer. This machine included colour graphics, and it came in its own plastic casing. It retailed for $1,299, and it was one of the first computers to come pre-assembled. The Apple II was a commercial success, and Apple became a public listed company in 1980 (Fig. 7.2).

The Apple Macintosh was announced during a famous television commercial aired during the Super Bowl in 1984. It was quite different from the IBM PC in that it included a friendly and intuitive graphical user interface, and the machine was much easier to use than the standard IBM PC. The latter was a command-driven operating system and required the users to be familiar with its commands. The introduction of the Mac GUI was an important milestone in the computing field.

Jobs got the idea of the graphical user interface from Xerox’s PARC research centre in Palo Alto in California. Apple intended that the Macintosh would be an inexpensive and user-friendly personal computer that would rival the IBM PC and its clones. However, it was more expensive as it retailed for $2,495, and initially it had limited applications available. The IBM PC had spreadsheets, word processors and databases applications.

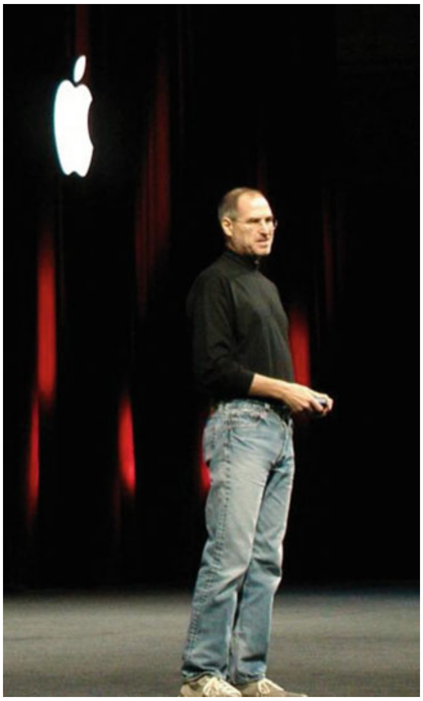


Fig. 7.2 Steve Jobs at MACworld 2005 (Photo public domain)

Apple went through financial difficulty in the mid-1980s, and Jobs resigned in 1985. Microsoft and Apple signed a contract in the mid-1980s that granted Microsoft permission to use some of the Macintosh GUIs. This contract led to the failure of the subsequent copyright infringement lawsuit taken by Apple against Microsoft in 1988.

It released the Newton Message Personal Digital Assistant (PDA) in 1993. However, this product was unsuccessful due mainly to reliability problems. IBM, Apple and Motorola entered an alliance in the early 1990s aimed at challenging the dominance of the Windows and Intel architecture. This initiative involved IBM and Motorola designing and producing the power CPUs for the new desktop computers and servers. The responsibilities of Apple were to port its Mac OS operating system to the new architecture. Despite some success, this initiative eventually failed, and Apple began using Intel microprocessors from 2005.

Jobs became the acting head of Apple in 1997, and he developed an alliance between Apple and Microsoft. The iMac was released in 1998 and was a major commercial success. It released iBook (a line of personal laptops) in 1999 and the iPod in 2001, and these were major commercial successes.

The iPod is a portable hard-disk MP3 player which has a capacity of 5 GB and could hold up to 1,000 MP3 songs.

It entered the mobile phone market with the release of the iPhone in 2007. This is an Internet-based multimedia smartphone. Its features include a video camera, e-mail, web browsing, text messaging and voice.

It released the iPad in 2010, and this is a large screen tablet-like device. It uses a touch-based operating system.

Apple is a highly innovative company and has made major contributions to the computing field.

7.4 Digital Research

Digital Research Inc. (DRI) was a Californian company founded by Gary Kildall and his wife Dorothy in the mid-1970s. It was initially called Galactic Digital Research Inc., and it was set up to develop, market and sell the CP/M operating system.

Kildall was one of the early people to recognise the potential of the microprocessor as a computer in its own right, and he began writing experimental programs for the Intel 4004 microprocessor. The Intel 4004 was the first commercially available microprocessor, and Kildall worked as a consultant with Intel on the 8008 and 8080 microprocessors. He developed the first high-level programming language for a microprocessor (PL/M) in 1973, and he developed the CP/M operating system (Control Program for Microcomputers) in the same year. This operating system allowed the 8080 to control a floppy drive, and CP/M combined all of the essential components of a computer at the microprocessor level.

Kildall made CP/M hardware independent by creating a separate module called the BIOS (Basic Input/Output System). He added several utilities such as an editor, debugger and assembler, and by 1977, several manufactures were including CP/M with their systems.

IBM approached DRI in 1980 with a view to licensing the CP/M operating system for the new IBM Personal Computer. IBM decided to use the Intel 8088 for its new personal computer product; Digital Research was working on CP/M-86 for the Intel 16-bit 8086 microprocessor that Intel had introduced in 1978. The 8088 was a lower cost and slower version of the 8086 and was introduced in 1979.

IBM and Digital Research did not reach an agreement on the licensing of CP/M for the PC. This may have been due to Kildall holding out for higher royalty payments and inflexibility on the part of Digital Research. Bill Gates of Microsoft had been negotiating a Microsoft BASIC licence agreement with IBM, and he saw an opportunity and offered to provide a DOS/BASIC package to IBM on favourable terms. The rest, as they say, is history.

Tim Patterson had developed a simple quick and dirty version of CP/M (called QDOS) for the 8086 microprocessor for Seattle Computer Products (SCP). Bill Gates licensed QDOS for $50,000 and hired Patterson to modify it to run on the IBM PC for the 8088 microprocessor. DRI released CP/M-86 shortly after IBM released DOS 1.0. IBM offered both CP/M-86 and DOS, but as CP/M was priced at $240 and DOS at $60, few PC owners were willing to pay the extra cost.

Digital Research considered suing Microsoft for copying all of the CP/M system calls in DOS 1.0. This would also have involved suing IBM, and DRI decided not to proceed with legal action due to its limited resources. Perhaps, if Kildall had played his hand differently, he could have been in the position that Bill Gates is in today, and Digital Research could have been a Microsoft. It shows that technical excellence is not in itself sufficient for business success. Digital Research was purchased by Novell in 1991.

7.5 Digital Corporation

Digital Equipment Corporation (DEC) was founded in 1957 by Ken Olsen and Harian Anderson. Olsen and Anderson were engineers who had worked on early machines at the Massachusetts Institute of Technology.

DEC was a leading vendor of computer systems from the 1960s to the 1990s, and its PDP and VAX products were very popular with the engineering and scientific communities. It was the second largest computer company in the world at its peak in the late 1980s when it employed over 140,000 people.

Its first computer, the Programmed Data Processor (PDP-1), was released in 1961. The PDP-1 was an 18-bit machine, and one of the earliest computer games, Spacewar, was developed for this machine.

The PDP series of minicomputers became popular in the 1960s. The PDP-8 minicomputer was a 12-bit machine with a small instruction set and was released in 1965. It was a major commercial success for DEC with many sold to schools and universities. The PDP-11 was a highly successful series of 16-bit minicomputer sold from the 1970s to the 1990s.

The VAX 11 series was derived from the PDP-11 and was the first widely used 32-bit minicomputer. The VAX 11/780 was released in 1978 and was a major success for the company. The VAX product line was a competitor to the IBM System/370 series of computers.

The rise of the microprocessor and microcomputer led to the availability of low-cost personal computers, and this later challenged DEC's product line. DEC was slow in recognising the importance of these developments, and Olsen's well-known statement 'There is no need for any individual to have a computer in his home' shows how unprepared DEC was for this new challenge.

DEC responded with their personal computer after the launch of the IBM PC. The DEC machine easily outperformed the PC but was more expensive and incompatible with IBM PC hardware and software. DEC's microcomputer efforts were a failure, but its PDP and VAX products were continuing to sell, and by the late 1980s, they were threatening IBM's number one spot in the industry. However, the increasing power of the newer generations of microprocessors began to challenge DEC's minicomputer product range.

Ultimately, the company was too late in responding to the threats of changing technology in the industry, and this proved to be fatal. Its sales were falling in the early 1990s, and indecision and infighting inside the company delayed an appropriate response.

Robert Palmer became the new CEO and was given the responsibility to return the company to profitability. He attempted to change the business culture and sold off noncore businesses. Eventually, Digital was acquired by Compaq in 1998 for $9.8 billion, and Compaq later merged with HP.

7.6 Intel Corporation

Robert Noyce and Gordon Moore were among the founders of Fairchild Semiconductor which was established in 1957. They founded Intel in 1968. It is the largest semiconductor manufacturer in the world, with major plants in the United States, Europe and Asia. It has had a major impact on the computing field with its invention of microprocessors.

The microprocessor is essentially a computer on a chip, and it made handheld calculators and personal computers (PCs) possible. Intel's microprocessors are used on the majority of PCs worldwide.

The initial focus of Intel was to create large-scale integrated (LSI) semiconductor memory, and they introduced a number of products including the 1103 a 1-kB dynamic RAM (DRAM).

The company made a major impact on the computer industry with its introduction of the Intel 4004 microprocessor in 1971. This was the world's first microprocessor, and although it was initially developed as an enhancement to allow users to add more memory to their units, it soon became clear that the microprocessor had great potential for everything, from calculators to cash registers and traffic lights.

The 4004 was designed by Ted Hoff, and it contained a central processing unit (CPU) on one chip. It contained 2,300 transistors on a one-eighth by one-sixth-inch chip. It had the power of the ENIAC computer, which used 18,000 vacuum tubes.

It introduced the 8-bit 8080 microprocessor in 1974, and this was the first general-purpose microprocessor. It was sold for $360: that is, a whole computer on one chip was sold for $360, while conventional computers sold for thousands of dollars. The 8080 soon became the industry standard, and Intel became the industry leader in the 8-bit market.

The 16-bit 8086 was introduced in 1978, but it took 2 years to achieve for it to be used widely. Motorola had developed the 68000 microprocessor which appeared to be selling faster. However, IBM chose the Intel chip for its personal computer and took a 20% stake in Intel, leading to strong ties between both companies.

Intel released the 80486 microprocessor in 1989. It was described by Business Week as 'a verifiable mainframe on a chip'. This microprocessor included 1.2 million transistors and the first built-in math coprocessor. It was 50 times faster than the 4004, the first microprocessor. It released the Pentium generation of microprocessor in 1993.

Intel dominates the microprocessor market and has a broad product line including motherboards, flash memory, switches and routers. It has made a major contribution to the computing field.

7.7 Oracle

Larry Ellison, Bob Miner and Ed Oates founded Software Development Laboratories in 1977. Ellison became the chief executive of the company and was responsible for sales and marketing, while Miner was responsible for software development. The company changed its name to Oracle Systems in 1983. Ellison was familiar with Codd's theoretical work on databases at IBM and was aware of the IBM research work on databases (i.e. the IBM System R project). He saw an opportunity to exploit and commercialise relational database technology before IBM.

There were no commercial relational databases in the world at the time, and the launch of Oracle changed business processes and computing. The database product was called Oracle in memory of a CIA-funded project that they had previously worked on. The release of Oracle V.2 in 1979 was a major milestone in the history of computing as it was the world's first commercial SQL relational database management system.

The concept of a relational database was described in a paper 'A Relational Model of Data for Large Shared Data Banks' by Edgar Codd [Cod:70]. Codd was born in England, and he worked for IBM in the United States. A relational database is a database that conforms to the relational model, and it may be defined as a set of relations (or tables).

The Structured Query Language (SQL) is a computer language that tells the relational database what to retrieve and how to display it. A relational database management system (RDBMS) is a system that manages data using the relational model, and examples of such systems include Oracle, Informix and Microsoft SQL Server.

A relation is defined as a set of tuples and is usually represented by a table, where a table is data organised in rows and columns. The data stored in each column of the table is of the same data type. Constraints may be employed to provide restrictions on the kinds of data that may be stored in the relations. Constraints are Boolean expressions which indicate whether the constraint holds or not, and are a way of implementing business rules into the database.

Most relations have one or more keys associated with them, and the key uniquely identifies the row of the table. An index is a way of providing quicker access to the data in a relational database as it allows the tuple in a relation to be looked up directly (using the index) rather than checking all of the tuples in the relation.

A stored procedure is executable code that is associated with the database. It is usually written in an imperative programming language, and it is used to perform common operations on the database.

An Oracle database consists of a collection of data managed by an Oracle database management system. Today, Oracle is the main standard for database technology, and Oracle databases are used by companies throughout the world. It has grown to become one of the largest software companies in the world with over 100,000 employees.

It is also involved in enterprise application market and has acquired companies such as PeopleSoft and Siebel. This had led to it becoming a competitor to SAP in the enterprise resource planning (ERP) and customer relationship management (CRM) market. This led to a legal dispute between Oracle and SAP over allegations that an acquired subsidiary of SAP, Tomorrow Now, had illegally downloaded Oracle software.

The success of the Oracle database led to competitor products from IBM (DB2), Sybase (later taken over by SAP), Informix (later taken over by IBM) and Microsoft (SQL server which was developed from Sybase software).

Oracle is recognised as a world leader in relational database technology, and its products play a key role in business computing.

7.8 Amdahl

Amdahl was founded by Gene Amdahl in 1970. Amdahl joined IBM in 1952 and became the chief designer of the IBM 704 computer which was released in 1954. He was the principal architect for the hugely successful System/360 family of mainframe computers which was introduced in 1964. Amdahl was appointed an IBM fellow in recognition of his contribution to IBM and given the freedom to pursue his own research projects.

He resigned from IBM in 1970 to set up Amdahl. The company received funding from Fujitsu, who were interested in a joint development program, and also from Nixdorf who were interested in representing Amdahl in Europe.

Amdahl launched its first product, the Amdahl 470, in 1975. Its first customer was NASA's Goddard Spaceflight Centre in New York and further sales to universities and other institutions followed. The 470 competed directly against the IBM System/370 family of mainframes which was released in 1972. It was compatible with IBM hardware and software but cheaper than the IBM product; that is, the Amdahl machines provided better performance for less money.

IBM's machines were water-cooled, while Amdahl's were air-cooled, which decreased installation costs significantly. The use of large-scale integration (LSI) with many integrated circuits on each chip meant the Amdahl 470 was one-third of the size of IBM's 360. Further, it performance was over twice as fast and sold for about 10% less than the IBM 360.

Machine sales were slow initially due to concerns over the company's survival. However, Amdahl had 50 units in place by 1977 and began to pose a serious challenge to IBM in large-scale computer placements. Amdahl decided to sell rather than lease its equipment, and this led to a price war with IBM.

IBM launched a new product, the IBM 3033, to compete with the Amdahl 470. However, Amdahl responded with a new machine that was one and a half times faster than the 3033 and was only slightly more expensive. Customers voted with their feet and chose Amdahl as their supplier, and it was clear that Amdahl was a major competitor to IBM in the high-end mainframe market. It had a 24% market share and annual revenues of $2 billion by 1989.

Amdahl worked closely with Fujitsu to improve circuit design, and Fujitsu's influence on the company increased following Gene Amdahl's departure from the company in 1980. Amdahl moved into large system multiprocessor design from the mid-1980s, and it launched a new product line, the 5990 processor, in 1988. This processor outperformed IBM by 50%.

However, by the late 1990s, it was clear that Amdahl could no longer effectively compete against IBM's 64-bit zSeries as Amdahl had only 31-bit servers to sell. The company estimated that it would take $1 billion to create an IBM-compatible 64-bit system. Further, there were declining sales in the mainframe market due to the popularity of personal computers.

Amdahl became a wholly owned subsidiary of Fujitsu in 1997, and its headquarters are in California.

7.9 HP

Hewlett Packard was founded by Bill Hewlett and Dave Packard in 1939. They were classmates at Stanford University and graduated in engineering in 1934. Packard then took a position with General Electric, and Hewlett continued with graduate studies in Stanford/MIT. They built their first product, an electronic test instrument used by sound engineers, in a Palo Alto garage, and Disney Studios was an early HP customer. The company began to grow during the early 1940s with orders from the US government.

Hewlett and Packard created a management style for the company that became known as the HP Way. This included a strong commitment by the company to its employees and a strong belief in the basic goodness of people and in their desire to do a good job. They believed that if employees are given the proper tools to do their job, they would then do a good job. It was a core value that each person had the right to be treated with respect and dignity. The HP Way was highly effective, and its corporate culture was later copied by several technology companies.

The HP management technique is known as 'management by walking around' and is characterised by the personal involvement of management. This includes good listening skills by the manager and the recognition that everyone in a company wants to do a good job. The HP Way involves management by objectives; that is, senior managers communicate the overall objectives clearly to their employees. Employees are then given the flexibility to work towards those goals in ways that are best for their own area of responsibility. The HP Way was refined further in the late 1950s, and the company objectives included several areas. These included profit, customers, growth, people, management and citizenship.

HP also established an open-door policy to create an atmosphere of trust and mutual understanding. The open-door policy encouraged employees to discuss problems with a manager without fear of reprisals or adverse consequences. HP addressed employees by their first name and provided good benefits such as free medical insurance and the provision of regular parties for employees. HP was the first company to introduce flexitime in the late 1960s. The concept was based on trust, and it allowed employees to arrive early or late for work as long as they worked a standard number of hours.

HP entered the microwave field during the Second World War, and it later became a leader in the technology. It became a public quoted company in 1957, and set up two plants in Europe in the late 1950s.

It moved into the medical device sector in the 1960s and developed its first computer in the mid-1960s. HP's research laboratory was established in 1966, and it became a leading commercial research centre.

The company introduced the hand held scientific calculator (the HP-35) in the early 1970s, and this invention made the slide rule obsolete. It introduced the first programmable calculator, the HP-65, in 1974. This programmable pocket calculator could fit inside a shirt pocket, and its computational power was greater than that of the large bulky early computers. It was the first programmable handheld calculator to be used in space as part of the 1975 Apollo Soyuz Test project. HP became a major player in the computer industry in the 1980s, and its products included desktop computers and minicomputers as well as inkjet and laser printers. It introduced a touch screen personal computer in the early 1980s.

HP merged with Compaq in 2002, and the new company is known as HP. It has revenues of over $90 billion, employs over 150,000 people and has more than one billion customers in over 160 countries. It is a market leader in business infrastructure, including servers, storage, software, imaging and printing, personal computers, and personal access devices. It is a leading consumer technology company offering a range of technology tools from digital cameras to PCs to handheld devices. The rise of HP and insight into its business practices, culture and management style is described by David Packard [Pac:96].

7.10 Siemens

Siemens is a European technology giant which employs 475,000 people around the world and has annual sales of €87 billion. It is one of the world's leading electrical engineering and electronics companies, and its major competitors are General Electric, ABB and Hitachi. The company operates in several segments including automation and control, power, transportation, medical and information and communications.

It was founded in 1847 by Werner von Siemens and J. Halske as Siemens & Halske. Siemens, a former artillery officer in the Prussian army and an engineer, was the driving force in the company. Its early projects involved manufacturing and installing telegraph networks in Prussia and Russia and building an Indo-European[1]

[1] The construction took 2 years, and the route was over 11,000 km. It extended from London to Calcutta. It took 1 min for a dispatch to reach Teheran, whereas it took 28 min to reach Calcutta.

telegraph network. It laid the first direct transatlantic cable (from Ireland to the United States) in 1874. Siemens played a role in the introduction of facsimile telegraphy in the late 1920s which involved scanning of photographs. It was popular with the press as it made it easier to transmit photographs. It played a role in launching the world's first public telex network with Deutsche Reichspost in 1933. It benefited from the German rearmament during the 1930s, and it manufactured a wide range of equipment for the German armed services. It was involved in the development and manufacturing of the V2 rocket used to launch attacks on London towards the end of the war.

It was the first company to succeed in manufacturing the ultrapure silicon needed for semiconductor components. This was done in 1953 independently of the research being done in the United States. It introduced its first mainframe computer in 1955. The company became involved in satellite communication from the mid-1960s. It was the official supplier of telecommunications and data processing equipment to the 1972 Olympic Games in Munich.

Siemens played an important role in the development of mobile phone technology. It entered the mobile phone market in the early 1980s, and at that time, there was a very expensive public mobile phone network[2] in place in Germany. Siemens supplied new mobile radio system equipment to Deutsche Bundespost in the mid-1980s, and this technology allowed subscribers to be reached at their own numbers. It introduced its first mobile phone (C1) in the mid-1980s. This phone closely resembled a car battery, and it weighed 8.8 kg or 19 pounds. It came with a suitcase attachment. Today, many mobile phones weigh less than 200 g.

Siemens has a number of joint ventures with companies such as Fujitsu, Nokia and Bosch. Fujitsu Siemens' portfolio ranges from high-performance servers, PCs and notebooks. Nokia-Siemens Networks has a research centre dedicated to advance the development of product platforms and services for the next generation of fixed and mobile networks.

7.11 Motorola

Motorola was founded as the Galvin Manufacturing Corporation in 1928 by the Galvin brothers. The company introduced one of the first commercially successful car radios in 1930, and this led to the brand name 'Motorola'.

The company is recognised for its innovation and its dedication to customer satisfaction. It has played a leading role in transforming the way in which people communicate. Its products have included walkie-talkies, a two-way radio

[2] This network supported 11,000 users in Germany, and its 600 staff were responsible for switching the calls of the subscribers. The use of this network was the preserve of the super rich in Germany as the cost of phones and calls were prohibitive for the general public.

system used for communication during the Second World War, televisions, cellular infrastructure equipment and mobile phones. The company was strong in semiconductor technology including integrated circuits used in computers. This included the microprocessors used in the Apple Macintosh and Power Macintosh computers.

The company is renowned for its dedication to quality and customer satisfaction. It developed the Six Sigma™ quality principles in 1983 and was awarded the first Malcolm Baldrige National Quality Award in 1988 in recognition of this work. Six Sigma is a rigorous customer-focused, data-driven management approach to business improvement. Its objectives are to eliminate defects from every product and process and to improve processes to do things better, faster and at a lower cost. It can be used to improve every activity and step of the business that is concerned with cost, timeliness and quality of results.

The company introduced a pager in 1956 that allowed radio messages to be sent to a particular individual. Motorola's radio equipment was used by Neil Armstrong on the Apollo 11 lunar module. A Motorola FM radio receiver was used on NASA's lunar roving vehicle to provide a voice link between the Earth and the moon.

Motorola set up a research lab in 1952 to take advantage of the potential of transistors and semiconductors, and by 1961, it was mass-producing semiconductors at a low cost. It introduced a transistorised walkie-talkie in 1962 as well as transistors for its Quasar televisions. It introduced the 8-bit 6800 microprocessors in 1974, which contained over 4,000 transistors, and a 16-bit microprocessor in 1979 which was adopted by Apple for its Macintosh personal computers. It introduced the MC68020, the first true 32-bit microprocessor in 1984. This microprocessor contained 200,000 transistors on a three-eighths-inch square chip. Motorola became the main supplier for the microprocessors used in Apple Macintosh and Power Macintosh personal computers.

It presented a prototype for the world's first portable telephone in 1973. This used cellular radio technology and linked people not places. Bell Labs introduced the idea of cellular communications in 1947, and Motorola was the first company to incorporate the technology into a portable device designed for use outside of an automobile. It introduced the first portable mobile phone in 1983 (the DynaTAC) which cost $3,500 and weighed 28 ounces.

The signals from a transmitter cover an area called a cell. As a user moves from one cell into a new cell, a handover to the new cell takes place without any noticeable transition. The signals in the adjacent cell are sent and received on different channels than the previous cell's signals, and so there is no interference. Analog is the original mobile phone system but has been replaced by more sophisticated systems in recent years. These include GSM, CDMA, GPRS and UMTS. The old analog phones were susceptible to noise and static. They were also subject to eavesdropping.

Motorola dominated the analog mobile phone market. However, it was slow to adapt to the GSM standard, and it paid the price with a loss of market share to Nokia and other competitors. The company was very slow to see the potential of a mobile phone as a fashion device.[3]

The Iridium constellation is the largest commercial satellite constellation in the world. Motorola played a key role in its design and development, and the system was launched 1998. It provided global satellite voice and data coverage with complete coverage of the earth including the oceans, airways and polar regions.

Iridium routes phone calls through space, and there are four earth stations. As satellites leave the area of an Earth base station, the routing tables change, and frames are forwarded to the next satellite just coming into view of the Earth base station. The system is used extensively by the US Department of Defense.

Motorola Mobility became part of Google in 2011.

7.12 Philips

Philips is a European technology giant founded by Gerard Philips[4] in Eindhoven, Holland, in 1891. The company initially made carbon-filament lamps, and today it is a technology giant that employs over 120,000 people with sales of approximately $27 billion. Its headquarters moved to Amsterdam in the late 1990s.

Philips and other companies pioneered the development of lamps that used tungsten wire after the carbon-filament lamp became obsolete in 1907. These produced three times as much light for the same amount of electricity. Philips was involved in improvements in the manufacture of filament wire which led to the production of incandescent lamps.

The company began manufacturing vacuum tubes and manufacturing radios in the 1920s. It introduced consumer products such as the electric razor in the 1930s and introduced the compact cassette in the 1960s. By the mid-1960s, it was producing integrated circuits, and its semiconductor sector played an important role in its business. It invented the compact audio cassette tape in 1962.

The company introduced the laser disk player in the late 1970s, the world's first compact disk in 1982 and the DVD in the late 1990s. Phillips is a large manufacturer of computer products such as personal digital assistants, CD-ROM drives and various other computer components and consumer products.

It sold off a majority stake of its semiconductor business to a consortium of private equity investors in 2005. The company has gone through a major process of change, and it plans to focus on health care, lifestyle and technology in the future.

[3] The attitude of Motorola at the time seemed to be similar to that of Henry Ford; i.e. they can have whatever colour they like as long as it is black.

[4] Gerard Philips was an engineer and a cousin of Karl Marx.

7.13 Review Questions

1. Describe the contribution of Microsoft to the computing field.
2. Discuss the contribution of Motorola to mobile phone technology.
3. Discuss the controversy between Microsoft and Apple and the controversy between Microsoft and Netscape.
4. Describe the contribution of Digital Research to the development of the CP/M operating system and how it missed the opportunity to become the operating system for the IBM Personal Computer.
5. Describe the HP Way.
6. Describe the contribution of the Digital Corporation to the computing field.

7.14 Summary

This chapter considered the history of a small selection of some of well-known technology companies including Microsoft, Apple, Digital Research, Digital, Oracle, Intel, Amdahl, HP and Motorola. Some of the companies considered (e.g. Microsoft and Intel) continue to thrive in business, while others (e.g. Amdahl, Digital and Digital Research) no longer exist.

Microsoft was founded in the mid-1970s, and it has grown to become a major corporation. It has developed operating systems such as MS-DOS, Microsoft Windows NT, Windows 2000 and Windows XP. The company was controversial from its early days with its use of a clone of Digital Research's CP/M operating system to develop its version of PC-DOS and MS-DOS operating system for the IBM Personal Computer and clones. Other controversies were to follow in later years with legal cases being taken against it for abusing its monopoly position.

Apple was founded in the mid-1970s. It has developed many innovative products including the Apple Macintosh, the iPhone, the iPad and so on.

DEC was founded by Olsen and Andersen, and it developed the PDP and VAX families of computers. It grew to become the second largest computer company in the world in the late 1980s, and its decline and failure to adapt to the changing technology led to its decline and eventual takeover by Compaq.

Motorola was founded by the Galvin brothers in 1928. Its initial business was in the production of radios for cars, and it became a world leader in radio and telecommunications. It was very successful initially in mobile phone technology and in semiconductors. However, it was slow in adapting to changing technology trends, and it had to exit the semiconductor business and to the mobile network infrastructure market. Motorola Mobility is now part of Google.

Chapter 8
The Internet Revolution

Key Topics

ARPANET
TCP/IP
The Internet
The World Wide Web
Dot-Com Bubble
E-Software Development
Facebook
The Twitter Revolution
Skype

8.1 Introduction

The vision of the Internet and World Wide Web goes back to an article by Vannevar Bush in the 1940s. Bush was an American scientist who had done work on submarine detection for the US Navy. He designed and developed the differential analyser which was a mechanical computer whose function was to evaluate and solve first-order differential equations. It was funded by the Rockefeller Foundation and developed by Bush and others at MIT in the early 1930s. Bush supervised Claude Shannon at MIT, and Shannon's initial work was to improve the differential analyser.

Bush became director of the office of Scientific Research and Development, and he developed a win-win relationship between the US military and universities. He arranged large research funding for the universities to carry out applied research to assist the US military. This allowed the military to benefit from the early exploitation of research results, and it also led to better facilities and laboratories at the universities. It led to close links and cooperation between universities such as

Fig. 8.1 Vannevar Bush

Harvard and Berkeley, and this would eventually lead to the development of ARPANET by DARPA.

Bush outlined his vision of an information management system called the '*memex*' (memory extender) in a famous essay 'As We May Think' [Bus:45]. He envisaged the memex as a device electronically linked to a library and able to display books and films. It describes a proto-hypertext computer system and influenced the later development of hypertext systems (Fig. 8.1).

> *A memex is a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory.*
>
> *It consists of a desk, and while it can presumably be operated from a distance, it is primarily the piece of furniture at which he works. On the top are slanting translucent screens, on which material can be projected for convenient reading. There is a keyboard, and sets of buttons and levers. Otherwise it looks like an ordinary desk.*

Bush predicted that:

> *Wholly new forms of encyclopedias will appear, ready made with a mesh of associative trails running through them, ready to be dropped into the memex and there amplified.*

This description motivated Ted Nelson and Douglas Engelbart to independently formulate ideas that would become hypertext. Tim Berners-Lee would later use hypertext as part of the development of the World Wide Web.

8.2 The ARPANET

There were approximately 10,000 computers in the world in the 1960s. These were expensive machines ($100K–$200K) with limited processing power. They contained only a few thousand words of magnetic memory, and programming and debugging was difficult. Further, communication between computers was virtually nonexistent.

However, several computer scientists had dreams of worldwide networks of computers, where every computer around the globe is interconnected to all of the other computers in the world. For example, Licklider[1] wrote memos in the early 1960s on his concept of an intergalactic network. This concept envisaged that everyone around the globe would be interconnected and able to access programs and data at any site from anywhere.

The US Department of Defense founded the Advanced Research Projects Agency (ARPA) in the late 1950s. ARPA embraced high-risk, high-return research, and Licklider became the head of its computer research program. He developed close links with MIT, UCLA and BBN Technologies.[2] The concept of packet switching[3] was invented in the 1960s, and several organisations including the National Physical Laboratory (NPL), the RAND Corporation and MIT commenced work on its implementation.

The early computers had different standards for data representation, and the standard employed by each computer needed to be known for communication. This led to recognition of the need for common standards in data representation, and a US government committee developed ASCII (American Standard Code for Information Interchange) in 1963. This was the first universal standard for data, and it allowed machines from different manufacturers to exchange data. The standard allowed a 7-bit binary number to stand for a letter in the English alphabet, an Arabic numeral or a punctuation symbol. The use of 7 bits allowed 128 distinct characters to be represented. The development of the IBM 360 mainframe standardised the use of 8 bits for a word, and 12-bit or 36-bit words became obsolete.

The first wide area network connection was created in 1965. It involved the connection of a computer at MIT to a computer in Santa Monica via a dedicated telephone line. This showed that a telephone line could be used for data transfer. ARPA recognised the need to build a network of computers in the mid-1960s, and this led to the ARPANET project in 1966 which aimed to implement a packet-switched network with a network speed of 56 Kbps. ARPANET was to become the world's first packet-switched network.

BBN Technologies was awarded the contract to implement the network. The first two nodes were based at UCLA and SRI with plans for a total of 19 nodes. The network management was performed by interconnected 'Interface Message Processors' (IMPs) in front of the major computers. The IMPs eventually evolved to become the network routers that are used today.

[1] Licklider was an early pioneer of AI and wrote an influential paper '*Man-Computer Symbiosis*' in 1960 which outlined the need for simple interaction between users and computers.

[2] BBN Technologies (originally Bolt, Beranek and Newman) is a research and development technology company. It played an important role in the development of packet switching and in the implementation and operation of ARPANET. The '@' sign used in an e-mail address was a BBN innovation.

[3] Packet switching is a message communication system between computers. Long messages are split into packets which are then sent separately so as to minimise the risk of congestion.

The team at UCLA called itself the Network Working Group and saw its role as developing the Network Control Protocol (NCP). This was essentially a rule book that specified how the computers on the network should communicate. The first host-to-host connection was made between a computer in UCLA and a computer at the Stanford Research Institute (SRI) in late 1969. Several other nodes were added to the network until it reached its target of 19 nodes in 1971.

The Network Working Group developed the telnet protocol and file transfer protocol (FTP) in 1971. The telnet program allowed the user of one computer to remotely log in to the computer of another computer. The file transfer protocol allows the user of one computer to send or receive files from another computer. A public demonstration of ARPANET was made in 1972, and it was a huge success. One of the earliest demos was that of Weizenbaum's ELIZA program. This is a famous AI program that allows a user to conduct a typed conversation with an artificially intelligent machine (psychiatrist) at MIT.

The viability of packet switching as a standard for network communication had been clearly demonstrated. Ray Tomlinson of BBN Technologies developed a program that allowed electronic mail (e-mail) to be sent over the ARPANET. Over 30 institutions were connected to the ARPANET by the early 1970s.

8.3 TCP/IP

ARPA was renamed to DARPA (Defence Advanced Research Projects Agency) in 1973. It commenced a project to connect seven computers on four islands using a radio-based network and a project to establish a satellite connection between a site in Norway and in the United Kingdom. This led to a need for the interconnection of the ARPANET with other networks. The key problems were to investigate ways of achieving convergence between ARPANET, radio-based networks and the satellite networks, as these all had different interfaces, packet sizes and transmission rates. Therefore, there was a need for a network-to-network connection protocol.

An International Network Working Group (INWG) was formed in 1973. The concept of the transmission control protocol (TCP) was developed at DARPA by Bob Kahn and Vint Cerf, and they presented their ideas at an INWG meeting at the University of Sussex in England in 1974 [KaC:74]. TCP allowed cross network connections, and it began to replace the original NCP protocol used in ARPANET.

TCP is a set of network standards that specify the details of how computers communicate, as well as the standards for interconnecting networks and computers. It was designed to be flexible and provides a transmission standard that deals with physical differences in host computers, routers and networks. It is designed to transfer data over networks which support different packet sizes and which may sometimes lose packets. It allows the inter-networking of very different networks which then act as one network.

Table 8.1 TCP layers

Layer	Description
Network interface layer	This layer is responsible for formatting packets and placing them onto the underlying network
Internet layer	This layer is responsible for network addressing. It includes the Internet protocol and the address resolution protocol
Transport layer	This layer is concerned with data transport and is implemented by TCP and the user datagram protocol (UDP)
Application layer	This layer is responsible for liaising between user applications and the transport layer. It includes the file transfer protocol (FTP), telnet, domain naming system (DNS) and simple mail transfer program (SMTP)

The new protocol standards were known as the transport control protocol (TCP) and the Internet protocol (IP). TCP details how information is broken into packets and reassembled on delivery, whereas IP is focused on sending the packet across the network. These standards allow users to send e-mail or to transfer files electronically, without needing to concern themselves with the physical differences in the networks. TCP/IP consists of four layers (Table 8.1).

The Internet protocol (IP) is a connectionless protocol that is responsible for addressing and routing packets. It breaks large packets down into smaller packets when they are travelling through a network that supports smaller packets. A connectionless protocol means that a session is not established before data is exchanged, and packet delivery with IP is not guaranteed as packets may be lost or delivered out of sequence. An acknowledgement is not sent when data is received, and the sender or receiver is not informed when a packet is lost or delivered out of sequence. A packet is forwarded by the router only if the router knows a route to the destination. Otherwise, it is dropped. Packets are dropped if their checksum is invalid or if their time to live is zero. The acknowledgement of packets is the responsibility of the TCP protocol. The ARPANET employed the TCP/IP protocols as a standard from 1983.

8.4 Birth of the Internet

The use of ARPANET was initially limited to academia and to the US military, and in the early years, there was little interest from industrial companies. It allowed messages to be sent between the universities that were part of ARPANET. There were over 2,000 hosts on the TCP/IP-enabled network by the mid-1980s.

It was decided to shut down the network by the late 1980s, and the National Science Foundation (NSF) commenced work on the NSFNET in the mid-1980s. This network consisted of multiple regional networks connected to a major backbone. The original links in NSFNET were 56 Kbps, but these were updated to 1.544 Mbps T1 links in 1988. The NSFNET T1 backbone initially connected 13 sites, but this increased due to growing academic and industrial from around the world. The NSF realised that the Internet had significant commercial potential.

The Internet began to become more international with sites in Canada and several European countries connected. DARPA formed the computer emergency response team (CERT) to deal with any emergency incidents arising from the operation of the network.

The independent not-for-profit company, Advanced Network Services (ANS), was founded in 1991. It installed a new network (ANSNET) that replaced the NSFNET T1 network and operated over T3 (45 Mbps) links. It was owned and operated by a private company rather than the US government, with the NSF focusing on research aspects of networks rather than the operational side.

The ANSNET network was a distributive network architecture operated by commercial providers such as Sprint, MCI and BBN. The network was connected by major network exchange points, termed network access points (NAPs). There were over 160,000 hosts connected to the Internet by the late 1980s.

8.5 Birth of the World Wide Web

The World Wide Web was invented by Tim Berners-Lee in 1990 at CERN in Switzerland. CERN is a key European centre for research in the nuclear field, and it employs several thousand physicists and scientists. Berners-Lee obtained a degree in physics in the mid-1970s at Oxford University in England. His parents had been involved in the programming of the Ferranti Mark I computer in the 1950s.

The invention of the World Wide Web was a revolutionary milestone in computing. It transformed the use of the Internet from mainly academic use to where it is now an integral part of peoples' lives. Users could now surf the web, that is, hyperlink among the millions of computers in the world and obtain information easily. It is revolutionary in that:

- No single organisation is controlling the web.
- No single computer is controlling the web.
- Millions of computers are interconnected.
- It is an enormous marketplace of billions of users.
- The web is not located in one physical location.
- The web is a space and not a physical thing.

One of the problems that scientists that CERN faced in late 1989 was keeping track of people, computers, documents and databases. The centre had many visiting scientists who spent several months there, as well as a large pool of permanent staff. There was no efficient and effective way to share information among scientists.

A visiting scientist might need to obtain information or data from a CERN computer or to make the results of their research available to CERN. Berners-Lee developed a program called 'Enquire' to assist with information sharing and in

Table 8.2 Features of World Wide Web

Feature	Description
URL	Universal resource identifier (later renamed to universal resource locator (URL)) provides a unique address code for each web page
HTML	Hypertext Markup Language (HTML) is used for designing the layout of web pages
HTTP	The Hypertext Transport Protocol (HTTP) allows a new web page to be accessed from the current page
Browser	A browser is a client program that allows a user to interact with the pages and information on the World Wide Web

keeping track of the work of visiting scientists. He returned to CERN in the mid-1980s to work on other projects and devoted part of his free time to consider solutions to the information sharing problem.

He built on several existing inventions such as the Internet, hypertext and the mouse. Hypertext was invented by Ted Nelson in the 1960s, and it allowed links to be present in text. For example, a document such as a book contains a table of contents, an index and a bibliography. These are all links to material that is either within the book itself or external to the book. The reader of a book is able to follow the link to obtain the internal or external information. The mouse was invented by Doug Engelbart in the 1960s, and it allowed the cursor to be steered around the screen.

The major leap that Berners-Lee made was essentially a marriage of the Internet, hypertext and the mouse into what has become the World Wide Web. His vision and its subsequent realisation benefited CERN and the wider world.

He created a system that gives every web page a standard address called the universal resource locator (URL). Each page is accessible via the Hypertext Transfer Protocol (HTTP), and the page is formatted with the Hypertext Markup Language (HTML). Each page is visible using a web browser. The key features of Berners-Lee invention are presented in Table 8.2.

Berners-Lee invented the well-known terms such as URL, HTML and World Wide Web, and he wrote the first browser program that allowed users to access web pages throughout the world. Browsers are used to connect to remote computers over the Internet and to request, retrieve and display the web pages on the local machine.

The early browsers included Gopher developed at the University of Minnesota, and Mosaic developed at the University of Illinois. These were replaced in later years by Netscape which dominated the browser market until Microsoft developed Internet Explorer. The development of the graphical browsers led to the commercialisation of the World Wide Web.

The World Wide Web creates a space in which users can access information easily in any part of the world. This is done using only a web browser and simple web addresses. The user can then click on hyperlinks on web pages to access further relevant information that may be on an entirely different continent. Berners-Lee is now the director of the World Wide Web Consortium, and this MIT-based organisation sets the software standards for the web.

8.6 Applications of the World Wide Web

Berners-Lee realised that the World Wide Web offered the potential to conduct business in cyberspace, rather than the traditional way of buyers and sellers coming together to do business in the marketplace:

Anyone can trade with anyone else except that they do not have to go to the market square to do so.

The growth of the World Wide Web has been phenomenal since its invention. Exponential growth rate curves became a feature of newly formed Internet companies and their business plans. The World Wide Web has been applied to many areas including:

- Travel industry (booking flights, train tickets and hotels)
- E-marketing
- Online shopping (e.g. www.amazon.com)
- Portal sites (such as Yahoo)
- Recruitment services (such as www.jobserve.com)
- Internet banking
- Online casinos (for gambling)
- Newspapers and news channels
- Social media (Facebook)

The prediction in the early days was that the new web-based economy would replace traditional bricks and mortar companies. It was expected that most business would be conducted over the web, with traditional enterprises losing market share and going out of business. Exponential growth of e-commerce companies was predicted, and the size of the new web economy was estimated to be in trillions of US dollars.

New companies were formed to exploit the opportunities of the web, and existing companies developed e-business and e-commerce strategies to adapt to the brave new world. Companies providing full e-commerce solutions were concerned with the selling of products or services over the web to either businesses or consumers. These business models are referred to as business-to-business (B2B) or business-to-consumer (B2C). The characteristics of e-commerce websites are presented in Table 8.3.

8.7 Dot-Com Companies

The success of the World Wide Web was phenomenal, and it led to a boom in the formation of ‘new economy’ businesses. These businesses were conducted over the web and included the Internet portal company, Yahoo; the online bookstore, Amazon; and the online auction site, eBay. Yahoo provides news and a range

Table 8.3 Characteristics of e-commerce

Feature	Description
Catalogue of products	The catalogue of products details the products available for sale and their prices
Well-designed and easy to use	This is essential as otherwise the website will not be used
Shopping carts	This is analogous to shopping carts in a supermarket
Security	Security of credit card information is a key concern for users of the web, as users need to have confidence that their credit card details will remain secure
Payments	Once the user has completed the selection of purchases there is a checkout facility to arrange for the purchase of the goods
Order fulfilment/order enquiry	Once payment has been received, the products must be delivered to the customer

of services, and most of its revenue comes from advertisements. Amazon initially sold books but now sells a collection of consumer and electronic goods. eBay brings buyers and sellers together in an online auction space.

Boo.com was an online fashion company that failed dramatically in late 1999. Pets.com was an online pet supplies and accessory company that lasted 1 year in business. Priceline.com is an online travel firm that offers airlines and hotels a service to sell unfilled seats or rooms cheaply. ETrade.com is an online share dealing company. Some of these new technology companies were successful and remain in business. Others were financial disasters due to poor business models, poor management and poor implementation of the new technology.

Some of these technology companies offered an Internet version of a traditional bricks and mortar company, with others providing a unique business offering. For example, eBay offers an auctioneering Internet site to consumers worldwide which was a totally new service and quite distinct from traditional auctioneering.

Yahoo was founded by David Filo and Jerry Yang who were students at Stanford in California. It was used by them as a way to keep track of their personal interests and the corresponding websites on the Internet. Their list of interests grew over time and became too long and unwieldy. Therefore, they broke their interests into a set of categories and then subcategories, and this is the core concept of the website.

There was a lot of interest in the site from other students, family and friends and a growing community of users. The founders realised that the site had commercial potential, and they incorporated it as a business in 1995. The company launched its initial public offering (IPO) 1 year later in April 1996, and the company was valued at $850 million (or approximately 40 times its annual sales).

Yahoo is a portal site and offers free e-mail accounts to users, a search engine, news, shopping, entertainment, health and so on. The company earns a lot of its revenue from advertisement (including the click through advertisements that appear on a yahoo web page). It also earns revenue from services such as web hosting, web tools, larger mailboxes and so on.

Amazon was founded by Jeff Bezos in 1995 as an online bookstore. Its product portfolio diversified over time to include the sale of CDs, DVDs, toys, computer software and video games. Its initial focus was to build up the ‘Amazon’ brand throughout the world, and its goal was to become the world’s largest bookstore. It initially sold books at a loss by giving discounts to buyers in order to build market share. It was very effective in building its brand through advertisements, marketing and discounts.

It has become the largest online bookstore in the world and has a sound business model with a very large product catalogue; a well-designed website with good searching facilities, good checkout facilities and good order fulfilment. It also developed an associate model, which allows its associates to receive a commission for purchases of Amazon products made through the associate site.

eBay was founded in California by Pierre Omidyar in 1995. Omidyar was a French-American programmer born in Paris, and he moved to the United States with his parents. He later studied computing and worked as a programmer at Claris prior to setting up eBay. The eBay site brings buyers and sellers together and allows buyers to bid for items. The company earns revenue by charging a commission for each transaction. The IPO of eBay took place in 1998 and was highly successful.

Millions of items are listed, bought and sold on eBay every day. The sellers are individuals and international companies who are selling their products and services. Any legal product that does not violate the company’s terms of service may be bought or sold on the site. A buyer makes a bid for a product or service and competes against several other bidders. The highest bid is successful, and payment and delivery is then arranged. The revenue earned by eBay includes fees to list a product and commission fees that are applied whenever a product is sold. It is an international company with a presence in over 20 countries.

There have been a number of strange offerings on eBay. One man offered one of his kidneys for auction as part of the market for human organs. Other unusual cases have been towns that have been offered for sale (as a joke). Any product lists that violate eBay’s terms of service are removed from the site as soon as the company is aware of them. The company also has a fraud prevention mechanism which allows buyers and sellers to provide feedback on each other and to rate each other following the transaction. The feedback may be positive, negative or neutral, and relevant comments included. This offers a way to help to reduce fraud as unscrupulous sellers or buyers will receive negative ratings and comments.

Priceline was founded by Jay Walker and offers a service to airlines and hotels to sell unfilled seats or rooms cheaply. It was valued at $10 billion at its IPO, despite the fact that unlike airlines it had no assets and was actually selling flights at a loss.

8.7.1 Dot-Com Failures

Several of the companies formed during the dot-com era were successful and remain in business today. Others had inappropriate business models or poor management and failed in a spectacular fashion. This chapter considers some of the dot-com failures and highlights the reasons for failure.

Webvan.com was an online grocery business based in California. It delivered products to a customer's home within a 30-min period of their choosing. The company expanded to several other cities before it went bankrupt in 2001. Many of its failings were due to management as the business model was reasonable. The management was inexperienced in the supermarket or grocery business, and the company spent excessively on infrastructure. It had been advised to build up an infrastructure to deliver groceries as quickly as possible, rather than developing partnerships with existing supermarkets. It built warehouses, purchased a fleet of delivery vehicles and top of the range computer infrastructure before running out of money.

Boo.com was founded in 1998 by Ernst Malmsten and others as an online fashion retailer based in the United Kingdom. The company spent over $135 million of shareholder funds in less than 3 years and went bankrupt in 2000. Its website was poorly designed for its target audience and went against many of the accepted usability conventions of the time. The website was designed in the days before broadband with most users employing 56 K modems. However, its design included the latest Java and Flash technologies, and it took several minutes to load the first page of the website for most users for the first released version of the site. The navigation of the website was inconsistent and changed as the user moved around the site. The net effect was that despite extensive advertising by the company, users were not inclined to use the site.

Other reasons for failure included poor management and leadership, lack of direction, lack of communication between departments, spirally costs left unchecked and hiring staff and contractors in large numbers leading to crippling payroll costs. Further, a large number of products were returned by purchasers, and there was no postage charge applied for this service. The company incurred a significant cost for covering postage for these returns. The company went bankrupt in 2000, and an account of its formation and collapse is in the book, *Boo Hoo* [MaP:02]. This book is a software development horror story, and the poor software development practices employed is evident from the fact that while it had up to 18 contractor companies working to develop the website, the developers were working without any source code control mechanism in place.

Pets.com was an online pet supply company founded in 1998 by Greg McLemore. It sold pet accessories and supplies. It had a well-known advertisement as to '*why one should shop at an online pet store*?'. The answer to this question was: '*Because Pets Can't Drive*!'. Its mascot (the Pets.com sock puppet) was well known. It launched its IPO in February 2000 just before the dot-com collapse.

Pets.com made investments in infrastructure such as warehousing and vehicles. It needed a critical mass of customers in order to break even, and its management believed that it needed $300 million of revenue to achieve this. They expected that this would take a minimum of 4–5 years, and, therefore, there was a need to raise further capital. However, following the dot-com collapse, there was negative sentiment towards technology companies, and it was apparent that it would

Table 8.4 Characteristics of business models

Constituent	Description
Value proposition	This describes how the product or service that the company provides is a solution to a customer problem
Market segment	This describes the customers that will be targeted, and it may include several market segments
Value chain structure	This describes where the company fits into the value chain [Por:98]
Revenue generation and margins	This describes how revenue will be generated and includes the various revenue streams (e.g. sales, investment income, support income, subscriptions)
Position in value network	This involves identifying competitors and other players that can assist in delivering added value to the customer
Competitive strategy	This describes how the company will develop a competitive advantage to be a successful player in the market

be unable to raise further capital. They tried to sell the company without success, and it went into liquidation 9 months after its IPO.

Kozmo.com was founded by Joseph Park and Yong Kang in New York in 1998 as an online company that promised free 1-h delivery of small consumer goods. It provided point-to-point delivery usually on a bicycle and did not charge a delivery fee. Its business model was deeply flawed, as it is expensive to offer point-to-point delivery of small goods within a 1-h period without charging a delivery fee. The company argued that they could make savings to offset the delivery costs as they did not require retail space. It expanded into several cities in the United States and raised about $280 million from investors. It had planned to launch an IPO, but this was abandoned due to the dot-com collapse. The company ceased trading in 2001.

8.7.2 Business Models

A business model converts a technology idea or innovation into a commercial reality and needs to be appropriate for the company and its intended operating market. A company with an excellent business idea but with a weak business model may fail, whereas a company with an average business idea but an excellent business model may be quite successful. Several of the business models in the dot-com era were deeply flawed, and the eventual collapse of many of these companies was predictable. Chesbrough and Rosenbroom [ChR:02] have identified six key components in a business model (Table 8.4).

8.7.3 Bubble and Burst

Netscape was founded as Mosaic Communications by Marc Andreessen and Jim Clark in 1994. It was renamed as Netscape in late 1994. Its initial public offering in 1995 demonstrated the incredible value of the new Internet companies.

The company had planned to issue the share price at $14 but decided at the last minute to issue it at $28. The share price reached $75 later that day. This was followed by what became the dot-com bubble where there were a large number of public offerings of Internet stock, and where the value of these stocks reached astronomical levels. Eventually, reality returned to the stock market when it crashed in April 2000, and share values returned to more realistic levels.

The vast majority of these companies were losing substantial sums of money, and few expected to deliver profits in the short term. Financial instruments such as the balance sheet, profit and loss account and price to earnings ratio are normally employed to estimate the value of a company. However, investment bankers argued that there was a new paradigm in stock market valuation for Internet companies. This paradigm suggested that the potential future earnings of the stock be considered in determining its appropriate value. This was used to justify the high prices of shares in technology companies, as frenzied investors rushed to buy these overpriced and overhyped stocks. Common sense seemed to play no role in decision making. The dot-com bubble included features such as:

- Irrational exuberance on the part of investors
- Insatiable appetite for Internet stocks
- Incredible greed from all parties involved
- A lack of rationality and common sense by all concerned
- Traditional method of company valuation not employed
- Interest in making money rather than in building the business first
- Following herd mentality
- Questionable decisions by Federal Reserve Chairman Alan Greenspan
- Questionable analysis by investment firms
- Conflict of interest for investment banks
- Market had left reality behind

There were winners and losers in the boom and collapse. Many made a lot of money from the boom, with others including pension funds and life assurance funds making significant losses. The investment banks typically earned 5–7% commission on each successful IPO, and it was therefore in their interest not to question the boom too closely. Those who bought and disposed early obtained a good return. Those who kept their shares for too long suffered losses.

The full extent of the boom can be seen in the rise and fall of the value of the Dow Jones and NASDAQ from 1995 through 2002 (Fig. 8.2).

The extraordinary rise of the Dow Jones from a level of 3,800 in 1995 to 11,900 in 2000 represented a 200% increase over 5 years or approximately 26% annual growth (compound) during this period. The rise of the NASDAQ over this period is even more dramatic. It rose from a level of 751 in 1995 to 5,000 in 2000, representing a 566% increase during the period. This is equivalent to a 46% compounded annual growth rate of the index (Fig. 8.3).

The fall of the indices has been equally as dramatic especially in the case of the NASDAQ. It peaked at 5,000 in March 2000 and fell to 1,200 (a 76% drop) by September 2002. It had become clear that Internet companies were rapidly going through the cash raised at the IPOs, and analysts noted that a significant number

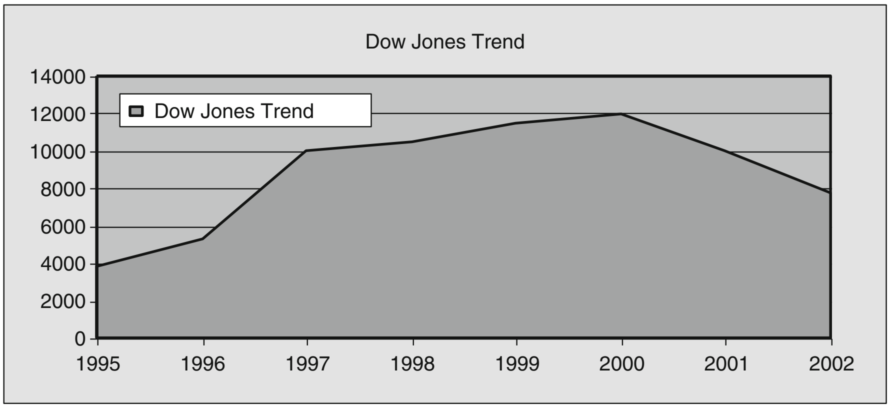


Fig. 8.2 Dow Jones (1995–2002)

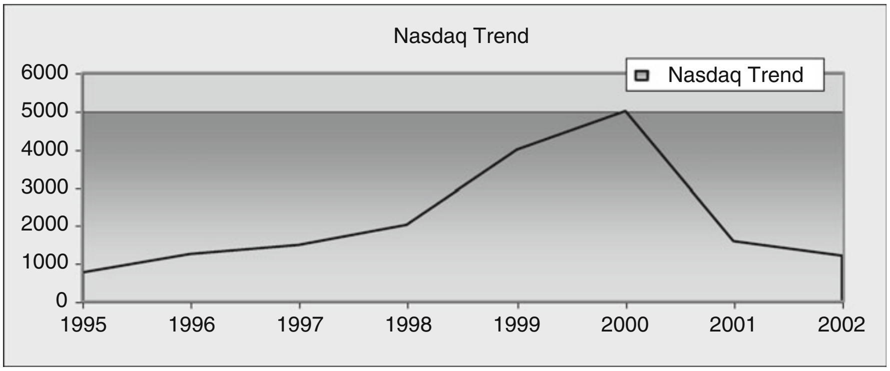


Fig. 8.3 NASDAQ (1995–2002)

would be out of cash by end of 2000. Therefore, these companies would either go out of business or would need to go back to the market for further funding. This led to questioning of the hitherto relatively unquestioned business models of many of these Internet firms. Funding is easy to obtain when stock prices are rising at a rapid rate. However, when prices are static or falling, with negligible or negative business return to the investor, then funding dries up. The actions of the Federal Reserve in rising interest rates to prevent inflationary pressures also helped to correct the irrational exuberance of investors. However, it would have been more appropriate to have taken this action 2–3 years earlier.

Some independent commentators had recognised the bubble, but their comments and analysis had been largely ignored. These included 'The Financial Times' and the 'Economist' as well as some commentators in the investment banks. Investors rarely queried the upbeat analysis coming from Wall Street and seemed to believe that the boom would never end. They seemed to believe that rising stock prices would be a permanent feature of the US stock markets. Greenspan had argued that it is difficult to predict a bubble until after the event, and that even if the bubble had been identified, it could not have been corrected without causing a contraction. Instead, the responsibility of the Fed according to Greenspan was to mitigate the fallout when it occurs.

There have, of course, been other stock market bubbles throughout history. For example, in the 1800 s, there was a rush on railway stock in England, leading to a bubble and eventual burst of railway stock prices in the 1840 s. There has been a recent devastating property bubble and collapse (2002–2009) in the Republic of Ireland. The failure of the Irish political class, the Irish Central bank and financial regulators, the Irish Banking sector in their irresponsible lending policies and failures of the media in questioning the bubble are deeply disturbing. Its legacy will remain in the country for many years and will require resilience in dealing with its aftermath.

8.8 Facebook and Twitter

Facebook is a social medial site that was founded by Mark Zuckerberg and other Harvard students in 2004. It allows users to add a personal profile, to add photos or albums of photos, to add other users as friends and to send messages to other friends. It was initially used by just Harvard University students, but it is now widely used throughout the world. It is the most widely used social media site with in excess of 600 million users.

Facebook allows users to set their own privacy settings which allow them to choose which users may see specific parts of their profile. Most of its revenue is generated from advertisement such as banner ads.

Twitter is a social networking and mini-blogging service that was founded by Jack Dorsey in 2006. It allows users to send or receive short messages of 140 characters called tweets. The company was founded in 2006 and has 200 million users. It is described as the SMS of the Internet.

The use of Twitter fluctuates with important external events leading to a spike in its use. Twitter messages are public, but senders may also send short private messages that are visible just to their followers.

8.9 E-Software Development

An organisation that conducts part or all of its business over the World Wide Web will need to ensure that its website is fit for purpose. It needs to be of high quality, reliable and usable. Web applications are quite distinct from other software systems in that:

- They may be accessed from anywhere in the world.
- They may be accessed by many different browsers.

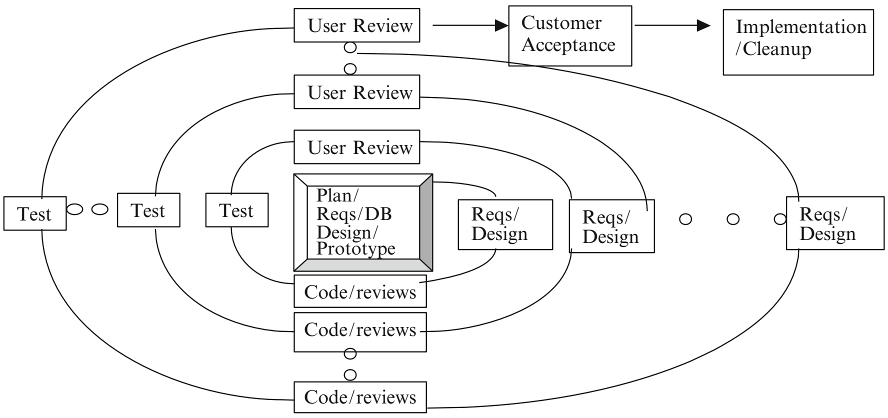


Fig. 8.4 Spiral life cycle model

- It is a distributed system with millions of servers and billions of users.
- The usability and look and feel of the application is a key concern.
- The performance and reliability of the website are key concerns.
- Security threats may occur from anywhere.
- A large number of transactions may occur at any time.
- There are strict availability constraints (typically 24 $\times$ 7 $\times$ 365).

Rapid application development (RAD) or joint application development (JAD) lifecycles are often employed for website development. The spiral life cycle is used rather than waterfall, as it is often the case that the requirements for web-based systems will not be fully defined at project initiation.

The spiral is, in effect, a reusable prototype, and the customer examines the current iteration and provides feedback to the development team. This is addressed in the next spiral. The approach is to partially implement the system as this leads to a better understanding of the requirements of the system, and it then feeds into the next development cycle. The process repeats until the requirements and product are fully complete. The spiral model is shown in Fig. 8.4.

There are various roles involved in web-based software development including content providers who are responsible for providing the content on the web, web designers who are responsible for graphic design of the website, programmers who are responsible for the implementation and administrators who are responsible for administration of the website.

Sound software engineering practices need to be employed to design, develop and test the website. The project team needs to produce similar documentation as the waterfall model, except that the chronological sequence of delivery of the documentation is more flexible. The joint application development is important as it allows early user feedback to be received on the look and feel and correctness of the application. The approach is often ‘design a little, implement a little, and test a little’.

Various technologies are employed in web development. These include HTML which is used to design simple web pages; CGIs (Common Gateway Interface) are often employed in sending a completed form to a server; cookies are employed to enable the server to store client-specific information on client's machine. Other popular technologies are Java, JavaScript, VB Script, Active X and Flash. There are tools such as Dreamweaver to assist with website design.

Testing plays an important role in assuring quality, and various types of web testing include:

- Static testing
- Unit testing
- Functional testing
- Browser compatibility testing
- Usability testing
- Security testing
- Load/performance/stress testing
- Availability testing
- Post-deployment testing

The purpose of post-deployment testing is to ensure that the performance of the website remains good, and this is generally conducted as part of a service level agreement (SLA). Service level agreements typically include a penalty clause if the availability of the system or its performance falls below defined parameters. Consequently, it is important to identify as early as possible potential performance and availability issues before they become a problem.

Most websites are operating $24 \times 7 \times 365$ days a year, and there is the potential for major financial loss in the case of a major outage of the electronic commerce website.

8.10 E-Commerce Security

The World Wide Web consists of unknown users and suppliers with unpredictable behaviour operating in unknown countries around the world. These users and websites may be friendly or hostile and the issue of trust arises:

- Is the other person whom they claim to be?
- Can the other person be relied upon to deliver the goods on payment?
- Can the other person be trusted not to inflict malicious damage?
- Is financial information kept confidential on the server?

Hostility may manifest itself in various acts of destruction. For example, malicious software may attempt to format the hard disk of the local machine, and if successful, all local data will be deleted. Other malicious software may attempt to steal confidential data from the local machine including bank account or credit card details. The denial of service attack is when a website is overloaded by a malicious attack, and where users are therefore unable to access the website for an extended period of time.

The display of web pages on the local client machine may involve the downloading of programs from the server and running the program on the client machine (e.g. Java Applets). Standard HTML allows the static presentation of a web page, whereas many web pages include active content (e.g. Java Applets or Active X). There is a danger that a Trojan Horse[4] may be activated during the execution of active content.

Security threats may be from anywhere (e.g. client side, server side, transmission) in an e-commerce environment, and therefore a holistic approach to security is required. Internal and external security measures need to be considered. Internal security is generally implemented via procedures and access privileges.

It is essential that the user is confident in the security provided as otherwise they will be reluctant to pass credit card details over the web for purchases. This has led to technologies such as Secure Socket Layer (SSL) and Secure HTTP (S-HTTP) to ensure security.

8.11 Review Questions

1. Describe the development of the Internet.
2. Describe the development of the World Wide Web and its key constituents.
3. Describe the applications of the World Wide Web.
4. Describe the key constituents of an electronic commerce site.
5. Describe a successful dot-com company that you are familiar with. What has made the company successful?
6. Describe a dot-com failure that you are familiar with. What caused the company to fail?
7. Discuss the key components of a business model.
8. Discuss software development in a web environment.

[4] The origin of the term 'Trojan Horse' is from Homer's Illiad and concerns the Greek victory in the Trojan War. The Greek hero, Odysseus, and others hid in a wooden horse, while the other Greeks sailed away from Troy. This led the Trojans to believe that the Greeks had abandoned their attack and were returning to their homeland leading behind a farewell gift for the citizens of Troy. The Trojans brought the wooden horse into the city, and later that night Odysseus and his companions opened the gates of Troy to the returning Greeks, and the mass slaughter of the citizens of Troy commenced. Hence, the phrase 'Beware of Greeks bearing gifts'. Troy is located at the mouth of the Dardanelles in Turkey.

8.12 Summary

This chapter considered the evolution of the Internet from the early work on packet switching and ARPANET, to the subsequent development of the TCP/IP network protocols that specify how computers communicate and the standards for interconnecting networks and computers.

TCP/IP provides a transmission standard that deals with physical differences in host computers, routers and networks. It is designed to transfer data over networks which support different packet sizes and which may sometimes lose packets. TCP details how information is broken into packets and reassembled on delivery, whereas IP is focused on sending the packet across the network.

The invention of the World Wide Web by Tim Berners-Lee was a revolutionary milestone in computing. It transformed the Internet from mainly academic use to commercial use and led to a global market of consumers and suppliers. Today, the World Wide Web is an integral part of peoples' lives.

The growth of the World Wide Web was exponential, and the boom led to the formation of many 'new economy' businesses. These new companies conducted business over the web as distinct from the traditional bricks and mortar companies. Some of these new companies were very successful (e.g. Amazon) and remain in business. Others were financial disasters due to poor business models, poor management and poor implementation of the new technology.

Chapter 9
History of Programming Languages

Key Topics

Generations of Programming Languages
Imperative Languages
ALGOL
FORTRAN and COBOL
Pascal and C
Object-Oriented Languages
Functional Programming Languages
Logic Programming Languages
Syntax and Semantics

9.1 Introduction

Hardware is physical and may be seen and touched, whereas software is intangible and is an intellectual undertaking by a team of programmers. Software is written in a particular programming language, and hundreds of languages have been developed. Programming languages have evolved over time with the earliest languages using machine code to program the computer. The next development was the use of assembly languages to represent machine language instructions. These were then translated into machine code by an assembler. The next step was the development of high-level programming languages such as FORTRAN and COBOL. These were easier to use than assembly languages and machine code and helped to improve quality and productivity.

A first-generation programming language (or 1GL) is a machine-level programming language that consists of 1s and 0s. Their main advantage of these languages is execution speed as they may be directly executed on the computer. These languages do not require a compiler or assembler to convert from a high-level language or assembly language into the machine code.

However, writing a program in machine code is difficult and error prone as it involves writing a stream of binary numbers. This made the programming language difficult to learn and difficult to correct should any errors occur. The programming instructions were entered through the front panel switches of the computer system, and adding new code was difficult. Further, the machine code was not portable as the machine language for one computer could differ significantly from that of another computer. Often, the program needed to be totally rewritten for the new computer.

First-generation languages were mainly used on the early computers. A program written in a high-level programming language is generally translated by the compiler[1] into the machine language of the target computer for execution.

Second-generation languages, or 2GL, are low-level assembly languages that are specific to a particular computer and processor. However, assembly languages are unlike first-generation programming languages in that the assembly code can be read and written more easily by a human. They require considerably more programming effort than high-level programming languages, and are more difficult to use for larger applications. The assembly code is converted by the assembler into the actual machine code to run on the computer. The assembly language is specific to a particular processor family and environment and is therefore not portable.

A program written in assembly language for a particular processor family needs to be rewritten for a different platform. However, since the assembly language is in the native language of the processor, it has significant speed advantages over high-level languages. Second-generation languages are still used today, but they have generally been replaced by high-level programming languages.

The third-generation languages, or 3GL, are high-level programming languages such as Pascal, C or FORTRAN. They are general-purpose languages and have been applied to business and scientific applications. They are designed to be easier for a human to understand and include features such as named variables, conditional statements, iterative statements, assignment statements and data structures. Early examples of third-generation languages are FORTRAN, ALGOL and COBOL.

[1] This is true of code generated by native compilers. Other compilers may compile the source code to the object code of a Virtual Machine, and the translator module of the Virtual Machine translates each byte code of the Virtual Machine to the corresponding native machine instruction. That is, the Virtual Machine translates each generalised machine instruction into a specific machine instruction (or instructions) that may then be executed by the processor on the target computer. Most computer languages such as C require a separate compiler for each computer platform (i.e. computer and operating system). However, a language such as Java comes with a virtual machine for each platform. This allows the source code statements in these programs to be compiled just once, and they will then run on any platform.

Later examples are languages such as C, C++ and Java. The advantages of these high-level languages over the older second-generation languages were:

– Ease of readability
– Clearly defined syntax (and semantics[2])
– Suitable for business or scientific applications
– Machine independent
– Portability to other platforms
– Ease of debugging
– Execution speed

These languages are machine independent and may be compiled for different platforms. The early 3GLs were procedural in that they focus on how something is done rather than on what needs to be done. The later 3GLs were object-oriented,[3] and the programming tasks were divided into objects. Objects may be employed to build larger programs, in a manner that is analogous to building a prefabricated building. Examples of modern object-oriented language are the Java language that is used to build web applications, C++ and Smalltalk.

High-level programming languages allow programmers to focus on problem-solving rather than on the low-level details associated with assembly languages. They are easier to debug and to maintain than assembly languages.

Fourth-generation languages specify what needs to be done rather than how it should be done. They are designed to reduce programming effort and include report generators and form generators. Report generators take a description of the data format and the report that is to be created, and then automatically generate a program to produce the report. Form generators are used to generate programs to manage online interactions with the application system users. However, 4GLs are slow when compared to compiled languages.

A fifth-generation programming language, or 5GL, is a programming language that is based around solving problems using constraints applied to the program rather than using an algorithm written by the programmer. Fifth-generation languages are designed to make the computer (rather than the programmer) solve the problem. The programmer specifies the problem and the constraints to be satisfied and is not concerned with the algorithm or implementation details. These languages are mainly used for research purposes especially in the field of artificial intelligence. Prolog is one of the best-known fifth-generation languages, and it is a logic programming language.

[2] The study of programming language semantics commenced in the 1960s. It includes work done by Hoare on axiomatic semantics, work done by Bjørner and Jones at IBM in Vienna on operational semantics and work done by Scott and Strachey on denotational semantics.

[3] Object-oriented programming was originally developed by Norwegian Research with Simula 67 in the late 1960s.

The task of deriving an efficient algorithm from a set of constraints for a particular problem is non-trivial, and to date, this step has not been successfully automated. Fifth-generation languages are used mainly in academia.

9.2 Plankalkül

The earliest high-level programming language was Plankalkül developed by Konrad Zuse in 1946. It means ‘Plan’ and ‘Kalkül’, that is, a calculus of programs. It is a relatively modern language for a language developed in 1946. There was no compiler for the language, and the Free University of Berlin developed a compiler for it in 2000. This allowed the first Plankalkül program to be run over 55 years after its conception.

It employs data structures and Boolean algebra and includes a mechanism to define more powerful data structures. Zuse demonstrated that the Plankalkül language could be used to solve scientific and engineering problems, and he wrote several example programs including programs for sorting lists and searching a list for a particular entry. The main features of Plankalkül are:

– A high-level language
– Fundamental data types are arrays and tuples of arrays
– While construct for iteration
– Conditionals are addressed using guarded commands
– There is no GOTO statement
– Programs are non-recursive functions
– Type of a variable is specified when it is used

The main constructs of the language are variable assignment, arithmetical and logical operations, guarded commands and while loops. There are also some list and set processing functions.

9.3 Imperative Programming Languages

Imperative programming is a programming style that describes computation in terms of a program state and statements that change the program state. The term ‘imperative’ in a natural language, such as English, is a command to carry out a specific instruction or action. Similarly, imperative programming consists of a set of commands to be executed on the computer and is therefore concerned with *how* the program will be executed. The execution of an imperative command generally results in a change of state.

Imperative programming languages are quite distinct from functional and logical programming languages. Functional programming languages, like Miranda, have no global state, and programs consist of mathematical functions that have no side effects. In other words, there is no change of state, and the variable *x* will have the same value later in the program as it does earlier. Logical programming languages, like Prolog, define '*what*' is to be computed rather than '*how*' the computation is to take place.

Most commercial programming languages are imperative languages, with interest in functional programming languages and relational programming languages being mainly academic. Imperative programs tend to be more difficult to reason about due to the change of state. Assembly languages and machine code are imperative languages.

High-level imperative languages use program variables and employ commands such as assignment statements, conditional commands, iterative commands and calls to procedures. An assignment statement performs an operation on information located in memory and stores the results in memory. The effect of an assignment statement is a change of the program state. A conditional statement allows a statement to be executed only if a specified condition is satisfied. Iterative statements allow a statement (or group of statements) to be executed a number of times.

High-level imperative languages allow the evaluation of complex expressions. These may consist of arithmetic operations and function evaluations, and the resulting value of the expression is assigned to memory.

FORTRAN was one of the earliest programming languages and was developed at IBM in the 1950s. ALGOL was designed by an international committee in the late 1950s and 1960s, and it became a popular language for the expression of algorithms. COBOL was designed by a committee in the late 1950s as a programming language for business use. BASIC (Beginner's All-Purpose Symbolic Instruction Code) was designed by George Kemeny and Tom Kurtas in 1963 as a teaching tool. Pascal was developed in the early 1970s as a teaching language by Niklaus Wirth. The C programming language was developed by Denis Ritchie in the early 1970s at Bell Laboratories.

The ADA language was developed for the US military in the early 1980s. Object-oriented languages are imperative in style but include features to support objects. Bjarne Stroustrup designed an object-oriented extension of the C language called C++, which was implemented in 1985. Java was released by Sun Microsystems in 1996.

9.3.1 FORTRAN and COBOL

FORTRAN (FORmula TRANslator) was the first high-level programming language to be implemented. It was developed by John Backus at IBM in the mid-1950s, and the first compiler was available in 1957. The language includes named variables, complex expressions and subprograms. It was designed for scientific and engineering applications and remains the most important programming language for these applications. The main statements of the language include:

– Assignment statements (using the = symbol)
– IF statements
– GOTO statements
– DO loops

Fortran II was developed in 1958, and it introduced subprograms and functions to support procedural (or imperative) programming. Each procedure (or subroutine) contains computational steps to be carried out and may be called at any point during program execution. This could include calls by other procedures or by itself. However, recursion was not allowed until Fortran 90. Fortran 2003 provides support for object-oriented programming.

The basic types supported in FORTRAN include Boolean, Integer and Real. Support for double precision and complex numbers was added later. The language included relational operators for equality (.EQ.), less than (.LT.) and so on. It was good at handling numbers and computation which was especially useful for mathematical and engineering problems. The following code (written in Fortran 77) gives a flavour of the language.

```
      PROGRAM HELLOWORLD
C     FORTRAN 77 SOURCE CODE COMMENTS FOR HELLOWORLD
        PRINT '(A)', 'HELLO WORLD'
        STOP
      END
```

FORTRAN remains a popular scientific programming language for applications such as climate modelling, simulations of the solar system, modelling the trajectories of artificial satellites, and simulation of automobile crash dynamics.

It was initially weak at handling input and output, which was important to business computing. This led to the development of the COBOL programming language by the CODASYL committee in the late 1950s.

The Common Business-Oriented Language (COBOL) was the first business programming language and was introduced in 1959. It was developed by Grace Murray Hopper[4] and a group of computer professionals called the Conference on Data Systems Languages (CODASYL).

Its objective was to improve the readability of software source code for business users, and it has an English-like syntax designed to make it easy for the average business user to learn the language. The only data types in the language were numbers and strings of text. The language allowed for these to be grouped into arrays and records so that data could be tracked and organised better. For example, the operation of division is performed by the verbose statement (Fig. 9.1):

DIVIDE A BY B GIVING C REMAINDER D

[4] Grace Murray Hopper was a programmer on the Mark 1, Mark II and Mark III and UNIVAC 1 computers. She was the technical advisor to the CODASYL committee.

Fig. 9.1 Grace Murray Hopper (U.S. Government Archives)

COBOL was the first computer language whose use was mandated by the US Department of Defense. The language remains in use today, and there is an object-oriented version of the language.

9.3.2 ALGOL

ALGOL (ALGOrithmic Language) is a family of imperative programming languages. It was originally developed in the mid-1950s and was subsequently revised in ALGOL 60 and ALGOL 68. It was designed to address some of the problems in FORTRAN, but it did not become widely used. This may have been due to the refusal of IBM to support ALGOL and the dominance of IBM in the computing field.

The language was designed by a committee of American and European computer scientists, and it had a significant influence on later language design. ALGOL 60 [Nau:60] was the most popular member of the family, and Edsger Dijkstra developed an early ALGOL 60 compiler. John Backus and Peter Naur developed a method for describing the syntax of the ALGOL 58 programming language called Backus Naur Form (or BNF).

ALGOL includes data structures and block structures. Block structures were designed to allow blocks of statements to be created (e.g. for procedures or functions). A variable defined within a block may be used within the block but is out of scope outside of the block.

ALGOL 60 introduced two ways of passing parameters to subprograms, and these are '*call by value*' and '*call by name*'. The call by value parameter passing technique involves the evaluation of the arguments of a function or procedure before the function or procedure is entered. The values of the arguments are passed to the function or procedure, and any changes to the arguments within the called function or procedure have no effect on the actual arguments. The call by name parameter passing technique is the default parameter passing technique in ALGOL 60. This involves reevaluating the actual parameter expression each time the formal parameter is read. Call by name is used today in C/C++ macro expansion.

ALGOL 60 includes conditional statements and iterative statements. It supports recursions, that is, it allows a function or procedure to call itself. It includes:

- *Dynamic arrays* – These are arrays in which the subscript range is specified by variables.
- *Reserved words* – These are keywords that are not allowed to be used as identifiers by the programmer.
- *User-defined data types* – These allow the user to design their own data types.
- ALGOL uses bracketed statement blocks, and it was the first language to use *begin-end* pairs for delimiting blocks.

It was used mainly by researchers in the United States and Europe. There was a lack of interest to its adoption by commercial companies due to the absence of standard input and output facilities in its description. ALGOL 60 became the standard for the publication of algorithms, and it had a major influence on later language development.

ALGOL evolved during the 1960s but not in the right direction. The ALGOL 68 committee decided on a very complex design rather than the simple and elegant ALGOL 60 specification. Tony Hoare remarked that:

ALGOL 60 was a great improvement on its successors.

9.3.3 Pascal and C

The Pascal programming language was developed by Niklaus Wirth in the early 1970s. It is named after Blaise Pascal (a seventeenth century French mathematician). It was based on the ALGOL programming language and was intended as a language to teach students structured programming.

The debate on structured programming was quite intense in the late 1960s. Structured programming [Dij:68] is concerned with rigorous techniques to design and develop programs. Dijkstra argued against the use of the GOTO statement 'GOTO Statement considered harmful' [Dij:68]. This led to several languages that did not include the GOTO statement. Today, it is agreed that the use of the GOTO statement should be avoided (Fig. 9.2).

The Pascal language includes constructs such as the conditional if statement; the iterative while, repeat and for statements; the assignment statement; and the case statement (which is a generalised if statement). The statement in the body of

Fig. 9.2 Niklaus Wirth (Photo public domain)

the repeat statement is executed at least once, whereas the statement within the body of a while statement may never be executed.

The language has several reserved words (known as keywords) that have a special meaning, and these may not be used as program identifiers. The Pascal program that displays 'Hello World' is given by:

```
program HELLOWORLD (OUTPUT);

begin
    WRITELN ('Hello, World!')
end.
```

Pascal includes several simple data types such as Boolean, Integer, Character and Reals. Its more advanced data types include arrays, enumeration types, ordinal types and pointer data types. It allows complex data types to be constructed from existing data types. Types are introduced by the reserved word 'type'.

```
type
   c = record
          a: integer;
          b: char
        end;
```

Pascal includes a 'pointer' data type, and this data type allows linked lists to be created by including a pointer type field in the record. The variable LINKLIST is a pointer to the data type B in the example below where B is a record.

```
type
  BPTR = ^B;
  B = record
        A : integer;
        C : BPTR
      end;

var
  LINKLIST : BPTR;
```

Pascal is a block-structured language with programs structured into procedures and function blocks. These can be nested to any depth, and recursion is allowed. Each block has its own constants, types, variables and other procedures and functions which are defined within the scope of the block.

Pascal was criticised as being unsuitable for serious programming by Brian Kernighan and others [Ker:81]. Many of these deficiencies were addressed in later versions of the language. However, by then, Richie at Bell Labs had developed the C programming language which became popular in industry. It is a general-purpose and a systems programming language.

It was originally designed as a language to write the kernel for the UNIX operating system, which had traditionally been written in assembly languages. The success of C in writing the UNIX kernel led to its use on several other operating systems such as Windows and Linux. It also influenced later language development such as C++ and is one of the most commonly used systems programming languages. The language is described in detail in [KeR:78].

The language provides high-level and low-level capabilities, and a C program that is written in ANSI C with portability in mind may be compiled for a very wide variety of computer platforms and operating systems with minimal changes to the source code. The C language is now available on a wide range of platforms.

C is a procedural programming language and includes conditional statements such as the 'if statement', the 'switch statement', iterative statements such as the 'while' statement or 'do' statement and the assignment statement.[5]

- If Statement

```
if (A == B)
    A =  A + 1;
else

    A = A– 1;5
```

- Assignment Statement

```
i  = i + 1;
```

[5] The semi-colon in Pascal is used as a statement separator, whereas it is used as a statement terminator in C.

One of the first programs that people write in C is the Hello World program. This is given by:

```
main()
{
    printf("Hello, World\n");
}
```

It includes several predefined data types including integers and floating point numbers:

– int (integer)
– long (long integer)
– float (floating point real)
– double (double precision real)

It allows more complex data types to be created using ‘structs’ that are similar to records in Pascal. It allows the use of pointers to access memory locations which allows the memory locations to be directly referenced and modified. The result of the following example is to assign 5 to the variable x:

```
 int   x;
 int   *ptr_x;

 x = 4;
 ptr_x = &x;
*ptr_x =5;
```

C is a block-structured language, and a program is structured into functions (or blocks). Each function block contains its own variables and functions. A function may call itself (i.e. recursion is allowed).

One key criticism of C is that it is very easy to make errors in C programs, and to thereby produce undesirable results. For example, one of the easiest mistakes to make is to accidently write the assignment operator (=) for the equality operator (==). This totally changes the meaning of the original statement, as can be seen below:

```
if (a == b)
      a++;                              …. Program fragment A
 else
      a--

if (a = b)
      a++;                              …. Program fragment B
 else
      a--
```

Both program fragments are syntactically correct, and the intended meaning of a program is easily changed. The philosophy of C is to allow statements to be written as concisely as possible, and this is potentially dangerous.[6] The use of pointers potentially leads to problems as uninitialised pointers may point anywhere in memory, and may therefore write anywhere in memory. Therefore, the effective use of C requires experienced programmers, well-documented source code and formal peer reviews of the source code by other developers.

9.4 Object-Oriented Languages

Object-oriented programming is a paradigm shift in programming. The traditional view of programming is that a program is a collection of functions or a list of instructions to be performed on the computer. Object-oriented programming considers a computer program to be a collection of objects that act on each other. Each object is capable of sending and receiving messages and processing data. That is, each object may be viewed as an independent entity or actor with a distinct role or responsibility.

An object is a 'black box' which sends and receives *messages*. A black box consists of *code* (computer instructions) and *data* (information which these instructions operate on). The traditional way of programming kept code and data separate. For example, functions and data structures in the C programming language are not connected. However, in the object-oriented world, code and data are merged into a single indivisible thing called an object.

The reason that an object is called a black box is that the user of an object never needs to look inside the box, since all communication to it is done via messages. Messages define the *interface* to the object. Everything an object can do is represented by its message interface. Therefore, there is no need to know anything about what is in the black box (or object) in order to use it. The approach is to access to an object only through its messages, while keeping the internal details private is called *information hiding*,[7] and this dates back to work done by Parnas in the early 1970s.

The origins of object-oriented programming go back to the invention of Simula 67 at the Norwegian Computing Research Centre[8] in the late 1960s. It introduced the notion of a class and instances of a class.[9] The Smalltalk object-oriented language was developed at Xerox in the mid-1970s. Xerox introduced the term

[6] It is very easy to write incomprehensible code in C, and even a 1 line of C code can be incomprehensible. The maintenance of poorly written code is a challenge unless programmers follow good programming practice. This discipline needs to be enforced by formal reviews of the source code.

[7] Information hiding is a key contribution by Parnas to computer science. He has also done work on mathematical approaches to software quality using tabular expressions [ORg:06].

[8] The inventors of Simula 67 were Ole-Johan Dahl and Kristen Nygaard.

[9] Dahl and Nygaard were working on ship simulations and were attempting to address the huge number of combinations of different attributes from different types of ships. Their insight was to group the different types of ships into different classes of objects, with each class of objects being responsible for defining its own data and behaviour.

Table 9.1 Object-oriented paradigm

Feature	Description
Class	A class defines the abstract characteristics of a thing, including its attributes (or properties) and its behaviours (or methods). The members of a class are termed objects
Object	An object is a particular instance of a class with its own set of attributes. The set of values of the attributes of a particular object is called its state
Method	The methods associated with a class represent the behaviours of the objects in the class
Message passing	Message passing is the process by which an object sends data to another object or asks the other object to invoke a method
Inheritance	A class may have subclasses (or children classes) that are more specialised versions of the class. A subclass inherits the attributes and methods of the parent class. This allows the programmer to create new classes from existing classes. The derived classes inherit the methods and data structures of the parent class
Encapsulation (information hiding)	One fundamental principle of the object-oriented world is encapsulation (or information hiding). The internals of an object are kept private to the object and may not be accessed from outside the object. That is, encapsulation hides the details of how a particular class works, and it requires a clearly specified interface around the services provided
Abstraction	Abstraction simplifies complexity by modelling classes and removing all unnecessary detail. All essential detail is represented, and non-essential information is ignored
Polymorphism	Polymorphism is behaviour that varies depending on the class in which the behaviour is invoked. Two or more classes may react differently to the same message. The same name is given to methods in different subclasses, i.e. one interface and multiple methods

'*object-oriented programming*' for the use of objects and messages as the basis for computation. Many modern programming languages (e.g. Java and C++) support object-oriented programming. The main features of object-oriented languages are presented in Table 9.1.

Object-oriented programming has become popular in large-scale software engineering and became the dominant paradigm in programming from the late 1980s. Its proponents argue that it is easier to learn and simpler to develop and maintain. Its growth in popularity was helped by the rise in popularity of graphical user interfaces (GUI), as the development of GUIs is well suited to object-oriented programming. The C++ language has become very popular, and it is an object-oriented extension of the C programming language.

Object-oriented features have been added to many existing languages including Ada, BASIC, Lisp, FORTRAN and Pascal.

9.4.1 C++ and Java

Bjarne Stroustrup developed the C++ programming language in 1983 as an object-oriented extension of the C programming language. It was designed to use the power of object-oriented programming and to maintain the speed and portability of C. It provides a significant extension of C's capabilities but does not force the programmer to use the object-oriented features of the language.

A key difference between C++ and C is the concept of a class. A *class* is an extension to the C concept of a structure. The main difference is that while a C data structure can hold only data, a C++ class may hold both data and functions. An *object* is an instantiation of a class; that is, the class is essentially the type, whereas the object is essentially the variable of that type. Classes are defined in C++ by using the keyword class as follows:

```
class class_name
{
      access_specifier_1:
           member1;
      access_specifier_2:
           member2;
       ...
}
```

The members may be either data or function declarations, and an access specifier is included to specify the access rights for each member (e.g. private, public or protected). Private members of a class are accessible only by other members of the same class; public members are accessible from anywhere where the object is visible; protected members are accessible by other members of same class and also from members of their derived classes. This is illustrated in the example of the definition of the class rectangle:

```
class CRectangle
{
   int x, y;
      public:
        void set_values (int,int);
        int area (void);
} rect;
```

Java is an object-oriented programming language developed by James Gosling and others at Sun Microsystems in the early 1990s. The syntax of the language was influenced by C and C++. The language was designed with portability in mind, and the objective is to allow a program to be written once and executed anywhere. Platform independence is achieved by compiling the Java code into Java bytecode. The latter is simplified machine instructions specific to the Java platform.

This code is then run on a Java Virtual Machine (JVM) that interprets and executes the Java bytecode. The JVM is specific to the native code on the host hardware. The problem with interpreting bytecode is that it is slow compared to traditional compilation. However, Java has a number of techniques to address this including just-in-time compilation and dynamic recompilation. Java also provides automatic garbage collection. This is a very useful feature as it protects programmers who forget to deallocate memory (thereby causing memory leaks).

Java is a proprietary standard that is controlled through the Java Community Process. Sun Microsystems makes most of its Java implementations available without charge. The following is an example of the Hello World program written in Java:

```
class HelloWorld
{

        public static void main (String args[])
        {
                System.out.println ("Hello World!");
        }
}
```

9.5 Functional Programming Languages

Functional programming is quite distinct from imperative programming in that computation for functional programs involves the evaluation of mathematical functions. Imperative programming involves the execution of sequential (or iterative) commands that change the system state. For example, the assignment statement alters the value of a variable, and for any imperative program, the value of a given variable x may change during program execution.

There are no changes of state for functional programs. The fact that the value of x will always be the same makes it easier to reason about functional programs than imperative programs. Functional programming languages provide referential transparency; that is, equals may be substituted for equals, and if two expressions have equal values, then one can be substituted for the other in any larger expression without affecting the result of the computation.

Functional programming languages use higher-order functions,[10] recursion, lazy and eager evaluation, monads[11] and Hindley-Milner–type inference systems.[12]

[10] Higher-order functions are functions that take functions as arguments or return a function as a result. They are known as operators (or functionals) in mathematics.

[11] Monads are used in functional programming to express input and output operations without introducing side effects. The Haskell functional programming language makes use of this feature.

[12] This is the most common algorithm used to perform type inference. Type inference is concerned with determining the type of the value derived from the eventual evaluation of an expression.

These languages have mainly been used in academia, but there has been some industrial use, including the use of Erlang for concurrent applications in industry. The roots of functional programming languages are in the lambda calculus developed by Church in the 1930s. Lambda calculus provides an abstract framework for describing mathematical functions and their evaluation. Church employed lambda calculus to prove that there is no solution to the decision problem for first-order arithmetic in 1936.

Any computable function can be expressed and evaluated using lambda calculus or Turing machines. The question as to whether two arbitrary expressions in the lambda calculus are equivalent cannot be solved by a general algorithm. Lambda calculus uses transformation rules, and one of these rules is variable substitution. The original calculus developed by Church was untyped; however, typed lambda calculi have been developed in recent years. Lambda calculus has influenced functional programming languages such as Lisp, ML and Haskell.

Functional programming uses the notion of higher-order functions. Higher-order functions take other functions as arguments and may return functions as results. The derivative function $^{d}/_{dx} f(x) = f'(x)$ is a higher-order function. It takes a function as an argument and returns a function as a result. The derivative of the function sin(x) is given by cos(x). Higher-order functions allow currying which is a technique developed by Schönfinkel. It allows a function with several arguments to be applied to each of its arguments one at a time, with each application returning a new (higher-order) function that accepts the next argument.

John McCarthy developed LISP at MIT in the late 1950s, and this language includes many of the features found in modern functional programming languages.[13] Scheme built upon the ideas in LISP and simplified and improved upon the language.

Kenneth Iverson developed APL[14] in the early 1960s, and this language influenced Backus's FP programming language. The ML programming language was created by Robin Milner at the University of Edinburgh in the early 1970s. David Turner developed Miranda at the University of Kent in the mid-1980s. The Haskell programming language was released in the late 1980s.

9.5.1 Miranda

Miranda was developed by David Turner at the University of Kent in the mid-1980s [Turn:85]. It is a non-strict functional programming language; that is, the arguments to a function are not evaluated until they are actually required within the function being called. This is also known as lazy evaluation, and one of its main advantages

[13] Lisp is a multi-paradigm language rather than a functional programming language.

[14] Iverson received the Turing Award in 1979 for his contributions to programming language and mathematical notation. The title of his Turing Award paper was 'Notation as a tool of thought'.

is that it allows an infinite data structures to be passed as an argument to a function. Miranda is a pure functional language in that there are no side-effect features in the language. The language has been used for:

– Rapid prototyping
– Specification language
– Teaching language

A Miranda program is a collection of equations that define various functions and data structures. It is a strongly typed language with a terse notation:

```
    Z = sqr p / sqr q
sqr k = k * k
    p = a + b
    q = a – b
    a = 10
    b = 5
```

The scope of a formal parameter (e.g. the parameter k above in the function sqr) is limited to the definition of the function in which it occurs.

One of the most common data structures used in Miranda is the list. The empty list is denoted by [], and an example of a list of integers is given by [1, 3, 4, 8]. Lists may be appended to by using the '++' operator. For example:

[1, 3, 5] ++ [2, 4] is [1, 3, 5, 2, 4].

The length of a list is given by the '#' operator:

[1, 3] = 2.

The infix operator ':' is employed to prefix an element to the front of a list. For example:

5: [2, 4, 6] is equal to [5, 2, 4, 6].

The subscript operator '!' is employed for subscripting: For example:

Nums = [5, 2, 4, 6] then Nums!0 is 5.

The elements of a list are required to be of the same type. A sequence of elements that contains mixed types is called a tuple. A tuple is written as follows:

Employee = ('Holmes', '222 Baker St. London', 50, 'Detective').

A tuple is similar to a record in Pascal, whereas lists are similar to arrays. Tuples cannot be subscripted, but their elements may be extracted by pattern matching. Pattern matching is illustrated by the well-known example of the factorial function:

```
fac 0 = 1
fac (n+1) = (n+1) * fac n
```

The definition of the factorial function uses two equations, distinguished by the use of different patterns in the formal parameters. Another example of pattern matching is the reverse function on lists:

```
reverse [] = []
reverse (a:x) = reverse x ++ [a]
```

Miranda is a higher-order language, and it allows functions to be passed as parameters and returned as results. Currying is allowed, and this allows a function of n-arguments to be treated as n-applications of a function with 1-argument. Function application is left associative, that is, f x y means (f x) y. That is, the result of applying the function f to x is a function, and this function is then applied to y. Every function of two or more arguments in Miranda is a higher-order function.

9.5.2 Lambda Calculus

Lambda calculus (**λ**-calculus) was designed by Alonzo Church in the 1930s to study computability. It is a formal system that may be used to study function definition, function application, parameter passing and recursion. It may be employed to define what a computable function is, and any computable function may be expressed and evaluated using the calculus. Church used lambda calculus in 1936 to give a negative answer to Hilbert's Entscheidungs problem.

The lambda calculus is equivalent to the Turing machine formalism. However, lambda calculus emphasises the use of transformation rules, whereas Turing machines are concerned with computability on primitive machines. Lambda calculus consists of a small set of rules:

– Alpha-conversion rule (α-conversion)[15]
– Beta-reduction rule (β-reduction)[16]
– Eta-conversion (η-conversion)[17]

Every expression in the **λ**-calculus stands for a function with a single argument. The argument of the function is itself a function with a single argument and so on. The definition of a function is anonymous in the calculus. For example, the function that adds one to its argument is usually defined as $f(x) = x + 1$. However, in λ-calculus, the function is defined as:

$$\lambda x.x + 1$$

[15] This essentially expresses that the names of bound variables is unimportant.

[16] This essentially expresses the idea of function application.

[17] This essentially expresses the idea that two functions are equal if and only if they give the same results for all arguments.

The name of the formal argument x is irrelevant, and an equivalent definition of the function is $\lambda\, z.\, z + 1$. The evaluation of a function f with respect to an argument (e.g. 3) is usually expressed by $f(3)$. In **λ**-calculus, this would be written as $(\lambda\, x.\, x + 1)\ 3$, and this evaluates to $3 + 1 = 4$. Function application is left associative, that is, $f\,x\,y = (f\,x)\,y$. A function of two variables is expressed in lambda calculus as a function of one argument which returns a function of one argument. This is known as currying and was discussed earlier. For example, the function $\mathrm{f}(x, y) = x + y$ is written as $\lambda\, x.\ \lambda\, y.\ x + y$. This is often abbreviated to $\lambda\, x\, y.\ x + y$.

λ-Calculus is a simple mathematical system, and its syntax is defined as follows:

```
<exp> ::= <identifier>         |
      λ <identifier>.<exp>     | --abstraction
      <exp> <exp>              | --application
      ( <exp> )
```

-- **Syntax of Lambda Calculus** --

λ-Calculus's four lines of syntax plus *conversion* rules are sufficient to define Booleans, integers, data structures and computations on them. It inspired Lisp and modern functional programming languages.

9.6 Logic Programming Languages

Logic programming languages describe what is to be done rather than how it should be done. These languages are concerned with the statement of the problem to be solved rather than how the problem will be solved.

These languages use mathematical logic as a tool in the statement of the problem definition. Logic is a useful tool in developing a body of knowledge (or theory), and it allows rigorous mathematical deduction to derive further truths from the existing set of truths. The theory is built up from a small set of axioms or postulates, and rules of inference derive further truths logically.

The objective of logic programming is to employ mathematical logic to assist with computer programming. Many problems are naturally expressed as a theory, and the statement of a problem to be solved is often equivalent to determining if a new hypothesis is consistent with an existing theory. Logic provides a rigorous way to determine this, as it includes a rigorous process for conducting proof.

Computation in logic programming is essentially logical deduction, and logic programming languages use first-order[18] predicate calculus. These languages employ theorem proving to derive a desired truth from an initial set of axioms.

[18] First-order logic allows quantification over objects but not functions or relations. Higher-order logics allow quantification of functions and relations.

These proofs are constructive[19] in that more than existence is demonstrated: in fact, an actual object that satisfies the constraints is produced. Logic programming specifies the objects, the relationships between them and the constraints that must be satisfied for the problem.

1. The set of objects involved in the computation
2. The relationships that hold between the objects
3. The constraints for the particular problem

The language interpreter decides how to satisfy the particular constraints. Artificial intelligence influenced the development of logic programming, and John McCarthy[20] demonstrated that mathematical logic could be used for expressing knowledge. The first logic programming language was Planner developed by Carl Hewitt at MIT in 1969. It uses a procedural approach for knowledge representation rather than McCarthy's mathematical logic.

The best-known logic programming languages is Prolog which was developed in the early 1970s by Alain Colmerauer and Robert Kowalski. It stands for ***pro***gramming in ***log***ic. It is a goal-oriented language that is based on predicate logic. Prolog became an ISO standard in 1995. The language attempts to solve a goal by tackling the subgoals that the goal consists of

goal :- subgoal$_1$, . . ., subgoal$_n$.

That is, in order to prove a particular goal, it is sufficient to prove subgoal$_1$ through subgoal$_n$. Each line of a typical Prolog program consists of a rule or a fact, and the language specifies what exists. The following program fragment has one rule and two facts:

```
grandmother(G,S) :- parent(P,S), mother(G,P).
mother(sarah, issac).
parent(issac, jacob).
```

The first line in the program fragment is a rule that states that G is the grandmother of S if there is a parent P of S and G is the mother of P. The next two statements are facts stating that issac is a parent of jacob and that sarah is the mother of issac. A particular goal clause is true if all of its subclauses are true:

goalclause(V$_g$) :- clause$_1$(V$_1$),..,clause$_m$(V$_m$)

A Horn clause consists of a goal clause and a set of clauses that must be proven separately. Prolog finds solutions by *unification*, that is, by binding a variable

[19] For example, the statement $\exists x$ such that $x = \sqrt{4}$ states that there is an x such that x is the square root of 4, and the constructive existence yields that the answer is that $x = 2$ or $x - -2$; i.e. constructive existence provides more the truth of the statement of existence, and an actual object satisfying the existence criteria is explicitly produced.

[20] John McCarthy received the Turing Award in 1971 for his contributions to artificial intelligence. He also developed the programming language Lisp.

to a value. For an implication to succeed, all goal variables V*g* on the left side of :- must find a solution by binding variables from the clauses which are activated on the right side. When all clauses are examined and all variables in V*g* are bound, the goal succeeds. But if a variable cannot be bound for a given clause, that clause fails. Following the failure of a fails, Prolog *backtracks*. This involves going back to the left to previous clauses to continue trying to unify with alternative bindings. Backtracking gives Prolog the ability to find multiple solutions to a given query or goal.

Most logic programming languages use a simple searching strategy to consider alternatives:

- If a goal succeeds and there are more goals to achieve, then remember any untried alternatives and go on to the next goal.
- If a goal is achieved and there are no more goals to achieve, then stop with success.
- If a goal fails and there are alternative ways to solve it, then try the next one.
- If a goal fails and there are no alternate ways to solve it, and there is a previous goal, then go back to the previous goal.
- If a goal fails and there are no alternate ways to solve it, and no previous goal, then stop with failure.

Constraint programming is a programming paradigm where relations between variables can be stated in the form of constraints. Constraints specify the properties of the solution and differ from the imperative programming languages in that they do not specify the sequence of steps to execute.

9.7 Syntax and Semantics

There are two key parts to any programming language, and these are its syntax and semantics. The syntax is the grammar of the language, and a program needs to be syntactically correct with respect to its grammar. The semantics of the language is deeper and determines the meaning of what has been written by the programmer. The semantics of a language determines what a syntactically valid program will compute. A programming language is therefore given by

$$\text{Programming Language} = \text{Syntax} + \text{Semantics}$$

The theory of the syntax of programming languages is well established, and Backus Naur Form[21] (BNF) is employed to specify the grammar of languages. The grammar of a language may be input into a parser, which determines whether the program is

[21] Backus Naur Form is named after John Backus and Peter Naur. It was created as part of the design of Algol 60 and used to define the syntax rules of the language.

syntactically correct. Chomsky[22] identified a hierarchy of grammars (regular, context free, context sensitive). A BNF specification consists of a set of rules such as

<symbol> ::= <expression with symbols>

where <symbol> is a *non-terminal*, and the expression consists of sequences of symbols and/or sequences separated by the vertical bar 'I' which indicates a choice. Symbols that never appear on a left side are called *terminals*.

The definition of the syntax of various statements in a programming language is given below:

<loop statement> ::= <while loop> | <for loop>.
<while loop> ::= while (<condition>) <statement>
<for loop>::= for (<expression>) < statement>
<statement>::= <assignment statement> | <loop statement>
<assignment statement>::= <variable>:= <expression>

The example above is a partial definition of the syntax of various statements in the programming language. It includes various non-terminals (<loop statement>, <while loop>, <for loop>, <condition>, <expression>, <statement>, <assignment statement> and <variable>). The terminals include 'while', 'for', ':=', '('and')'. The production rules for <condition≥ and <expression> are not included.

There are various types of grammars such as regular grammars, context-free grammars and context-sensitive grammars. The grammar of a language is translated by a parser into a parse table. Each type of grammar has its own parsing algorithm to determine whether a particular program is valid with respect to its grammar.

9.7.1 Programming Language Semantics

The formal semantics of a programming language is concerned with the meaning of programs. A program is written according to the rules of the language. Its syntax is then checked by the compiler, and if it is syntactically correct, the compiler then generates the equivalent machine code.[23]

The compiler must preserve the semantics of the language, and syntax gives no information on the meaning of a program. It is possible to write syntactically correct programs that behave in quite a different way from the intentions of the programmer.

The formal semantics of a language is given by a mathematical model that describes the possible computations described by the language. There are three main approaches to programming language semantics, namely, axiomatic semantics, operational semantics and denotational semantics (Table 9.2).

[22] Chomsky made important contributions to linguistics and the theory of grammars. He is more widely known today as a critic of US foreign policy.

[23] Of course, what the programmer has written may not be what the programmer had intended.

Table 9.2 Programming language semantics

Approach	Description
Axiomatic semantics	Axiomatic semantics gives meaning to phrases of the language by describing the logical axioms that apply to them. It is based on mathematical logic and employs pre- and post-condition assertions to specify what happens when the statement executes. The relationship between the initial assertion and the final assertion gives the semantics of the phrase
Operational semantics	This approach was developed by Gordon Plotkin [Plo:81]. It describes how a valid program is interpreted as a sequence of computational steps An abstract machine (SECD machine) may be defined to give meaning to phrases, by describing the transitions they induce on states of the machine The semantics may also be given by a precise mathematical interpreter such as the lambda calculus
Denotational semantics	Denotational semantics (originally called mathematical semantics) provides meaning to programs in terms of mathematical objects such as integers, tuples and functions It was developed by Christopher Strachey and Dana Scott at Oxford in the mid-1960s Each phrase in the language is translated into a mathematical object that is the *denotation* of the phrase

9.8 Review Questions

1. Describe the five generations of programming languages.
2. Describe the early use of machine code including its advantages and disadvantages.
3. Describe the early use of assembly languages including their advantages and disadvantages.
4. Describe the key features of FORTRAN and COBOL.
5. Describe the key features of Pascal and C.
6. Discuss the key features of object-oriented languages.
7. Discuss the similarities and differences between imperative programming languages and functional programming languages.
8. Discuss the key features of logic programming languages.
9. Discuss the similarities and differences between syntax and semantics of programming languages.
10. Discuss the three main types of programming language semantics.

9.9 Summary

This chapter considered the evolution of programming languages from the early machine languages, to the low-level assembly languages, to high-level programming languages and object-oriented languages and to functional and logic programming languages.

The advantages of the machine-level programming languages were execution speed and efficiency. It was difficult to write programs in these languages as the statements are just a stream of binary numbers. These languages were not portable, as the machine language for one computer could differ significantly from that of another.

The second-generation languages, or 2GL, are low-level assembly languages that are specific to a particular computer and processor. These are easier to write and understand, and they must be converted into the actual machine code to run on the computer. The assembly language is specific to a particular processor family and environment and is therefore not portable. However, their advantage is the execution speed, as the assembly language is the native language of the processor.

The third-generation languages, or 3GL, are high-level programming languages. They are general-purpose languages and have been applied to business and scientific applications. They are designed to be easier for a human to understand and to allow the programmer to focus on problem-solving. Their advantages include ease of readability, ease of portability and ease of debugging and maintenance. The early 3GLs were procedure-oriented, and the later 3GLs were object-oriented.

Fourth-generation languages, or 4GLs, are languages that consist of statements similar to human language. They specify what needs to be done rather than how it should be done.

Fifth-generation programming languages, or 5GLs, are programming languages that is based around solving problems using logic programming or applying constraints to the program. They are designed to make the computer (rather than the programmer) solve the problem. The programmer only needs to be concerned with the specification of the problem and the constraints to be satisfied and does not need to be concerned with the algorithm or implementation details.

Chapter 10
History of Software Engineering

Key Topics

Birth of Software Engineering
Floyd
Hoare
Formal Methods
Cleanroom and Software Reliability
Software Inspections and Testing
Project Management
Software Process Maturity Models

10.1 Introduction

The NATO Science Committee organised two famous conferences on software engineering in the late 1960s. The first conference was held in Garmisch, Germany, in 1968, and this was followed by a second conference in Rome in 1969. The Garmisch conference was attended by over 50 people from 11 countries.

The conferences highlighted the problems that existed in the software sector in the late 1960s, and the term '*software crisis*' was coined to refer to these problems. These included budget and schedule overruns of projects and problems with the quality and reliability of the delivered software. This led to the birth of *software engineering* as a separate discipline, and the realisation that programming is quite distinct from science and mathematics. Programmers are like engineers in the sense that they design and build products; however, they need an appropriate education to design and develop software.[1]

The construction of bridges was problematic in the nineteenth century, and many people who presented themselves as qualified to design and construct bridges did not have the required knowledge and expertise. Consequently, many bridges collapsed, endangering the lives of the public. This led to legislation requiring an engineer to be licensed by the Professional Engineering Association prior to practising as an engineer. These engineering associations identify a core body of knowledge that the engineer is required to possess, and the licensing body verifies that the engineer has the required qualifications and experience. The licensing of engineers by most branches of engineering ensures that only personnel competent to design and build products actually do so. This in turn leads to products that the public can safely use. In other words, the engineer has a responsibility to ensure that the products are properly built and are safe for the public to use.

Parnas argues that traditional engineering be contrasted with the software engineering discipline where there is no licensing mechanism and where individuals with no qualifications can participate in the design and building of software products.[2] However, the fact that the maturity frameworks such as the CMMI or ISO 9000 place a strong emphasis on qualifications and training may help to deal with this.

The Standish Group conducted research in the late 1990s [Std:99] on the extent of current problems with schedule and budget overruns of IT projects. This study was conducted in the United States, but there is no reason to believe that European or Asian companies perform any better. The results indicate serious problems with on-time delivery.[3] Fred Brooks has argued that software is inherently complex, and that there is no silver bullet that will resolve all of the problems associated with software such as schedule overruns and software quality problems [Brk:75, Brk:86].

Poor-quality software can cause minor irritation or it may seriously disrupt the work of an organisation leading. It has in a very small number of cases led to the

[1] Software companies that are following approaches such as the CMM or ISO 9000:2000 consider the qualification of staff before assigning staff to performing specific tasks. The qualifications and experience required for the role are considered prior to appointing a person to carry out a particular role. Mature companies place significant emphasis on the education and continuous development of their staff and in introducing best practice in software engineering into their organisation. There is a growing trend among companies to mature their software processes to enable them to deliver superior results. One of the purposes that the original CMM served was to enable the US Department of Defense (DOD) to have a mechanism to assess the capability and maturity of software subcontractors.

[2] Modern HR recruitment specifies the requirements for a particular role, and interviews with candidates aim to establish that the candidate has the right education and experience for the role.

[3] It should be noted that these are IT projects covering diverse sectors including banking, telecommunications, etc., rather than pure software companies. Mature software companies using the CMM tend to be more consistent in project delivery with high quality.

deaths of individuals, for example, in the case of the Therac-25.[4] The Y2K problem, where dates were represented in a 2-digit format, required major rework for year 2000 compliance. Clearly, well-designed programs would have hidden the representation of the date, thereby minimising the changes required for year 2000 compliance. The quality of software produced by mature software companies committed to continuous improvement tends to be superior.

Mathematics plays a key role in engineering and may potentially assist software engineers deliver high-quality software products that are safe to use. Several mathematical approaches that can assist in delivering high-quality software are described in [ORg:06]. There is a lot of industrial interest in software process maturity for software organisations, and approaches to assess and mature software companies are described in [ORg:02, ORg:10].[5] These focus mainly on improving the effectiveness of the management, engineering and organisation practices related to software engineering.

10.2 What Is Software Engineering?

Software engineering involves multi-person construction of multi-version programs. The IEEE 610.12 definition of software engineering is:

Definition 10.1 (Software Engineering). *Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software, and the study of such approaches.*

Software engineering includes:

1. Methodologies to determine requirements, design, develop, implement and test software to meet customers' needs.
2. The philosophy of engineering, that is, an engineering approach to developing software is adopted. That is, products are properly designed, developed and tested, with quality and safety properly addressed.

[4] Therac-25 was a radiotherapy machine produced by the Atomic Energy of Canada Limited (AECL). It was involved in at least six accidents between 1985 and 1987 in which patients were given massive overdoses of radiation. The dose given was over 100 times the intended dose, and three of the patients died from radiation poisoning. These accidents highlighted the dangers of software control of safety-critical systems. The investigation subsequently highlighted the poor software design of the system and the poor software development practices employed.

[5] Approaches such as the CMM or SPICE (ISO 15504) focus mainly on the management and organisational practices required in software engineering. The emphasis is on defining and following the software process. In practice, there is often insufficient technical detail on requirements, design, coding and testing in the models, as the models focus on what needs to be done rather how it should be done.

3. Mathematics[6] may be employed to assist with the design and verification of software products. The level of mathematics to be employed will depend on the safety-critical nature of the product, as systematic peer reviews and testing are often sufficient in building quality into the software product.
4. Sound project management and quality management practices are employed.

Software engineering requires the engineer to state precisely the requirements that the software product is to satisfy and then to produce designs that will meet these requirements. Engineers provide a precise description of the problem to be solved; they then proceed to producing a design and validate the correctness of the design; finally, the design is implemented, and testing is performed to verify its correctness with respect to the requirements. The software requirements need to be unambiguous and should clearly state what is required, and it should also be evident what is not required.

Classical engineers produce the product design, and then analyse their design for correctness. Classical engineers will always use mathematics in their analysis as this is the basis of confirming that the specifications are met. The level of mathematics employed will depend on the particular application and calculations involved. The term '*engineer*' is generally applied only to people who have attained the necessary education and competence to be called engineers and who base their practice on mathematical and scientific principles. Often, in computer science, the term engineer is employed rather loosely to refer to anyone who builds things rather than to an individual with a core set of knowledge, experience and competence.

Parnas[7] is a strong advocate of the classical engineering approach, and he argues that computer scientists should have the right education to apply scientific and mathematical principles to their work. This includes mathematics and design, to enable them to be able to build high-quality and safe products. Baber has argued [Bab:11] that mathematics is the language of engineering. He argues that students should be shown how to turn a specification into a program using mathematics.

Parnas has argued that computer science tends to include a small amount of mathematics, whereas mathematics is a significant part of an engineering course and is the language of classical engineering. He argues that students are generally taught programming and syntax, but not how to design and analyse software. He advocates an engineering approach to the teaching of mathematics with an emphasis on its application to developing and analysing product designs.

He argues that software engineers need education on engineering mathematics, specification and design, converting designs into programs, software inspections

[6] There is no consensus at this time as to the appropriate role of mathematics in software engineering. My view is that the use of mathematics should be mandatory in the safety-critical and security-critical fields as it provides an extra level of quality assurance in these critical fields.

[7] Parnas's key contribution to software engineering is information hiding which is used in the object-oriented world. He has also done work (mainly of academic interest) on mathematical approaches to software quality.

and testing. The education should enable the software engineer to produce well-designed programs that will correctly implement the requirements.

Parnas argues that software engineers have individual responsibilities as professional engineers.[8] They are responsible for designing and implementing high-quality and reliable software that is safe to use. They are also accountable for their own decisions and actions[9] and have a responsibility to object to decisions that violate professional standards. Professional engineers have a duty to their clients to ensure that they are solving the real problem of the client. Engineers need to be honest about current capabilities, and when asked to work on problems that have no appropriate technical solution, this should be stated honestly rather than accepting a contract for something that cannot be done.

The licensing of a professional engineer provides confidence that the engineer has the right education and experience to build safe and reliable products. Otherwise, the profession gets a bad name as a result of poor work carried out by unqualified people. Professional engineers are required to follow rules of good practice and to object when the rules are violated.[10] The professional engineering body is responsible for enforcing standards and certification. The term 'engineer' is a title that is awarded on merit, but it also places responsibilities on its holder.

Engineers have a professional responsibility and are required to behave ethically with their clients. The membership of the professional engineering body requires the member to adhere to the code of ethics of the profession. Most modern companies have a code of ethics that employees are required to adhere to. It details the required ethical behaviour and responsibilities.

The approach used in current software engineering is to follow a well-defined software engineering process. The process includes activities such as project management, requirements gathering, requirements specification, architecture design,

[8] The concept of accountability is not new; indeed, the ancient Babylonians employed a code of laws ca. 1750 B.C. known as the Hammurabi Code. This code included the law that if a house collapsed and killed the owner, then the builder of the house would be executed.

[9] However, it is unlikely that an individual programmer would be subject to litigation in the case of a flaw in a program causing damage or loss of life. Most software products are accompanied by a comprehensive disclaimer of responsibility for problems rather than a guarantee of quality. Software engineering is a team-based activity involving several engineers in various parts of the project, and it could be potentially difficult for an outside party to prove that the cause of a particular problem is due to the professional negligence of a particular software engineer, as there are many others involved in the process such as reviewers of documentation and code and the various test groups. Companies are more likely to be subject to litigation, as a company is legally responsible for the actions of their employees in the workplace, and the fact that a company is a financially richer entity than one of its employees. However, the legal aspects of licensing software may protect software companies from litigation including those companies that seem to place little emphasis on software quality. However, greater legal protection for the customer can be built into the contract between the supplier and the customer for bespoke software development.

[10] Software companies that are following the CMMI or ISO 9000 will employ audits to verify that the rules have been followed. Auditors report their findings to management, and the findings are addressed appropriately by the project team and affected individuals.

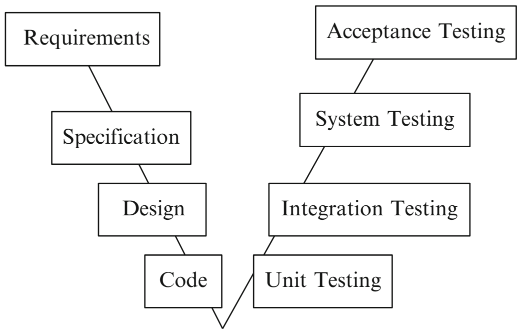


Fig. 10.1 Waterfall life cycle model (V-model)

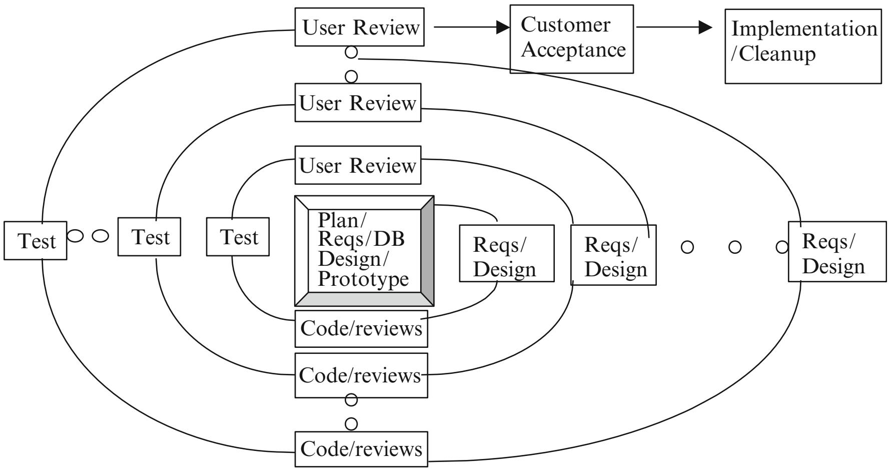


Fig. 10.2 Spiral life cycle model

software design, coding and testing. Most companies use a set of templates for the various phases. The waterfall model [Roy:70] and spiral model [Boe:88] are popular software development life cycles.

The waterfall model (Fig. 10.1) starts with requirements, followed by specification, design, implementation and testing. It is typically used for projects where the requirements can be identified early in the project life cycle or are known in advance. The waterfall model is also called the 'V' life cycle model, with the left-hand side of the 'V' detailing requirements, specification, design and coding and the right-hand side detailing unit tests, integration tests, system tests and acceptance testing. Each phase has entry and exit criteria which must be satisfied before the next phase commences. There are several variations of the waterfall model.

The spiral model (Fig. 10.2) is useful where the requirements are not fully known at project initiation. There is an evolution of the requirements during development which proceeds in a number of spirals, with each spiral typically involves updates to the requirements, design, code, testing and a user review of the particular iteration or spiral.

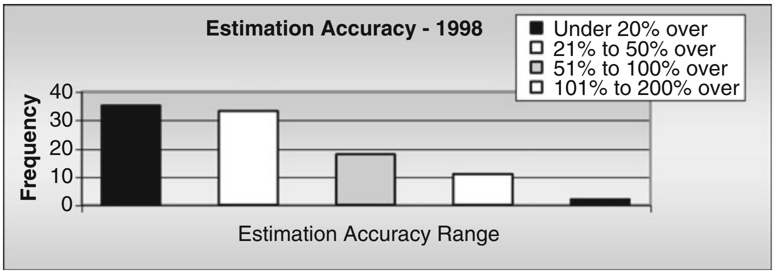


Fig. 10.3 Standish Group report: estimation accuracy

The spiral is, in effect, a reusable prototype, and the customer examines the current iteration and provides feedback to the development team to be included in the next spiral. The approach is to partially implement the system. This leads to a better understanding of the requirements of the system, and it then feeds into the next cycle in the spiral. The process repeats until the requirements and product are fully complete.

There are other life cycle models; for example, the Cleanroom approach to software development includes a phase for formal specification, and its approach to testing is quite distinct from other models as it is based on the predicted usage of the software product. Finally, the Rational Unified Process (RUP) has become popular in recent years.

The challenge in software engineering is to deliver high-quality software on time to customers. The Standish Group research (Fig. 10.3) on project cost overruns in the USA during 1998 indicates that 33% of projects are between 21% and 50% overestimate, 18% are between 51% and 100% overestimate and 11% of projects are between 101% and 200% overestimate.

Accurate project estimation of cost and effort are key challenges, and organisations need to determine how good their current estimation process actually is and to make improvements as appropriate. The use of software metrics allows effort estimation accuracy to be determined by computing the variance between actual project effort and the estimated project estimate.

Risk management is a key part of project management, and its objectives are to identify potential risks to the project, determine the probability of the risks occurring, assess the impact of each risk if it materialises, identify actions to eliminate the risk or to reduce its probability of occurrence, design contingency plans in place to address the risk if it materialises and finally, track and manage the risks throughout the project.

The concept of process maturity has become popular with the Capability Maturity Model, and the SEI has collected empirical data to suggest that there is a close relationship between software process maturity and the quality and the reliability of the delivered software. However, the main focus of the CMMI is management and organisation practices rather than on the technical engineering practices.

The implementation of the CMMI helps to provide a good engineering approach, as it places strict requirements on the processes and their characteristics that a company needs to have in place to provide a good engineering solution. The processes required include:

- Developing and managing requirements
- Doing effective design
- Planning and tracking projects
- Building quality into the product with peer reviews
- Performing rigorous testing
- Performing independent audits

There has been a growth of popularity among software developers in lightweight methodologies such as XP [Bec:00]. These methodologies view documentation with distaste, and often, software development commences prior to the full specification of the requirements.

10.3 Early Software Engineering

Robert Floyd was born in New York in 1936 and attended the University of Chicago. He became a computer operator in the early 1960s, an associate professor at Carnegie Mellow University in 1963 and a full professor of computer science at Sanford University in 1969. He did pioneering work on software engineering from the 1960s and made valuable contributions to the theory of parsing, the semantics of programming languages, program verification and methodologies for the creation of efficient and reliable software.

Mathematics and computer science were regarded as two completely separate disciplines in the 1960s, and software development was based on the assumption that the completed code would always contain defects. It was therefore better and more productive to write the code as quickly as possible and to then perform debugging to find the defects. Programmers then corrected the defects, made patches and retested and found more defects. This continued until they could no longer find defects. Of course, there was always the danger that defects remained in the code that could give rise to software failures.

Floyd believed that there was a way to construct a rigorous proof of the correctness of the programs using mathematics. He showed that mathematics could be used for program verification, and he introduced the concept of assertions that provided a way to verify the correctness of programs.

Flowcharts were employed in the 1960s to explain the sequence of basic steps for computer programs. Floyd's insight was to build upon flowcharts and to apply an invariant assertion to each branch in the flowchart. These assertions state the essential relations that exist between the variables at that point in the flowchart. An example relation is '$R = Z > 0, X = 1, Y = 0$'. He devised a general flowchart

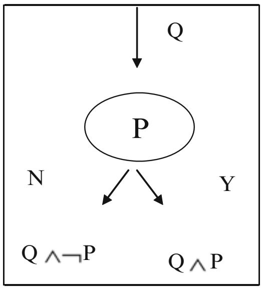


Fig. 10.4 Branch assertions in flowcharts

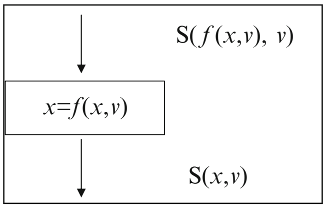


Fig. 10.5 Assignment assertions in flowcharts

language to apply his method to programming languages. The language essentially contains boxes linked by flow of control arrows [Flo:67].

Consider the assertion Q that is true on entry to a branch where the condition at the branch is P. Then, the assertion on exit from the branch is Q ∧ ¬P if P is false and Q ∧ P otherwise (Fig. 10.4).

The use of assertions may be employed in an assignment statement. Suppose x represents a variable and v represents a vector consisting of all the variables in the program. Suppose $f(x,v)$ represents a function or expression of x and the other program variables represented by the vector v. Suppose the assertion $S(f(x,v), v)$ is true before the assignment $x = f(x,v)$. Then the assertion $S(x,v)$is true after the assignment. This is given by (Fig. 10.5):

Floyd used flowchart symbols to represent entry and exit to the flowchart. This included entry and exit assertions to describe the program's entry and exit conditions.

Floyd's technique showed how a computer program is a sequence of logical assertions. Each assertion is true whenever control passes to it, and statements appear between the assertions. The initial assertion states the conditions that must be true for execution of the program to take place, and the exit assertion essentially describes what must be true when the program terminates.

His key insight was the recognition that if it can be shown that the assertion immediately following each step is a consequence of the assertion immediately preceding it, then the assertion at the end of the program will be true, provided the appropriate assertion was true at the beginning of the program.

He published an influential paper, 'Assigning Meanings to Programs', in 1967 [Flo:67], and this paper influenced Hoare's work on preconditions and post-conditions, leading to Hoare logic. This is a formal system of logic used for programming language semantics and for program verification and was originally published in Hoare's 1969 paper 'An axiomatic basis for computer programming' [Hor:69].

Hoare recognised that Floyd's approach provided an effective method for proving the correctness of programs. He built upon Floyd's work to include all of the familiar constructs of high-level programming languages. This led to the axiomatic approach to defining the semantics of every statement in a programming language with axioms and proof rules. He introduced what has become known as the Hoare triple, and this describes how the execution of a fragment of code changes the state. A Hoare triple is of the form:

$$P\{Q\}R$$

where P and R are assertions and Q is a program or command. The predicate P is called the *precondition*, and the predicate R is called the *post-condition*.

Definition 10.2 (Partial Correctness). *The meaning of the Hoare triple above is that whenever the predicate P holds of the state before the execution of the command or program Q, then the predicate R will hold after the execution of Q. The brackets indicate partial correctness as if Q does not terminate then R can be any predicate.*

Total correctness requires Q to terminate, and at termination, R is true. Termination needs to be proved separately. Hoare logic includes axioms and rules of inference rules for the constructs of imperative programming language. Hoare and Dijkstra were of the view that the starting point of a project should always be the specification, and that the proof of the correctness of the program should be developed hand in hand along with the program itself. That is, one starts off with a mathematical specification of what a program is supposed to do, and mathematical transformations are applied to the specification until it is turned into a program that can be executed. The resulting program is then known to be correct by construction.

10.4 Software Engineering Mathematics

Mathematics plays a key role in the classical engineer's work. For example, bridge designers will develop a mathematical model of a bridge prior to its construction. The model is a simplification of the reality, and an exploration of the model enables a deeper understanding of the proposed bridge to be gained. Engineers will model the various stresses on the bridge to ensure that the bridge design can deal with the projected traffic flow. The engineer applies mathematics and models to the design of the product, and the analysis of the design is a mathematical activity.

Mathematics allows a rigorous analysis to take place and avoids an over-reliance on intuition. The emphasis is on applied mathematics to solve practical problems and to develop products that are fit for use. The objective is therefore to teach students how to use and apply mathematics to program well and to solve practical problems. There is a rich body of classical mathematics available that may be applied to software engineering. This includes:

- Set theory
- Relations
- Functions
- Logic
- Calculus
- Functions
- Probability theory
- Graph theory
- Matrix theory

Mathematical approaches to software engineering are described in [ORg:06]. Next, we consider various formal methods that may be employed to assist in the development of high-quality software.

10.5 Formal Methods

The term 'formal methods' refers to various mathematical techniques used in the software field for the specification and formal development of software. Formal methods consist of formal specification languages or notations and employ a collection of tools to support the syntax checking of the specification, as well as the proof of properties of the specification. Abstraction is employed, and this allows questions to be asked about what the system does to be answered independently of the implementation. Furthermore, the unambiguous nature of mathematical notation avoids the problem of speculation about the meaning of phrases in an imprecisely worded natural language description of a system. Natural language is inherently ambiguous, whereas mathematics employs a precise notation with sound rules of inference. Spivey [Spi:92] defines formal specification as:

Definition 10.3 (Formal Specification). *Formal specification is the use of mathematical notation to describe in a precise way the properties which an information system must have, without unduly constraining the way in which these properties are achieved.*

The formal specification thus becomes the key reference point for the different parties involved in the construction of the system. It may be used as the reference point in the requirements, program implementation, testing and program documentation. The formal specification is a valuable means of promoting a common understanding for all those concerned with the system. The term '*formal methods*' is used to describe a formal specification language and a method for the design and implementation of computer systems.

The specification is written in a mathematical language, and the implementation is derived from the specification via stepwise refinement.[11] The refinement step makes the specification more concrete and closer to the actual implementation. There is an associated proof obligation that the refinement is valid, and that the concrete state preserves the properties of the more abstract state. Thus, assuming that the original specification is correct and the proofs of correctness of each refinement step are valid, then there is a very high degree of confidence in the correctness of the implemented software. Stepwise refinement is illustrated as follows: the initial specification S is the initial model M_0; it is then refined into the more concrete model M_1, and M_1 is then refined into M_2, and so on until the eventual implementation $M_{\mathrm{n}} = E$ is produced:

$$S = M_0 \subseteq M_1 \subseteq M_2 \subseteq M_3 \subseteq \ldots \subseteq M_n = \mathrm{E}$$

Requirements are the foundation from which the system is built, and irrespective of the best design and development practices, the product will be incorrect if the requirements are incorrect. The objective of requirements validation is to ensure that the requirements are correct and reflect what is actually required by the customer (in order to build the right system). Formal methods may be employed to model the requirements, and the model exploration yields further desirable or undesirable properties. The ability to prove that certain properties are true of the specification is very valuable, especially in safety-critical and security-critical applications. These properties are logical consequences of the definition of the requirements, and, if appropriate, the requirements may need to be amended appropriately. Thus, formal methods may be employed for requirements validation and in a sense to debug the requirements.

The use of formal methods generally leads to more robust software and to increased confidence in its correctness. The challenges involved in the deployment of formal methods in an organisation include the education of staff in formal specification, as formal specification and the use of mathematical techniques may be a culture shock to many staff.

Formal methods have been applied to a diverse range of applications, including the security-critical field, the safety-critical field, the railway sector, microprocessor verification, the specification of standards and the specification and verification of programs.

Formal methods have been criticised by Parnas and others on the grounds as enumerated in Table 10.1.

[11] It is questionable whether stepwise refinement is cost effective in mainstream software engineering, as it involves rewriting a specification ad nauseam. It is time-consuming to proceed in refinement steps with significant time also required to prove that the refinement step is valid. It is more relevant to the safety-critical field. Others in the formal methods field may disagree with this position.

Table 10.1 Criticisms of formal methods

No.	Criticism
1.	Often the formal specification is as difficult to read as the program[a]
2.	Many formal specifications are wrong[b]
3.	Formal methods are strong on syntax but provide little assistance in deciding on what technical information should be recorded using the syntax[c]
4.	Formal specifications provide a model of the proposed system. However, a precise unambiguous mathematical statement of the requirements is what is needed[d]
5.	Stepwise refinement is unrealistic. It is like, e.g. deriving a bridge from the description of a river and the expected traffic on the bridge. There is always a need for a creative step in design[e]
6.	Much unnecessary mathematical formalisms have been developed rather than using the available classical mathematics[f]

[a]Of course, others might reply by saying that some of Parnas's tables are not exactly intuitive, and that the notation he employs in some of his tables is quite unfriendly. The usability of all of the mathematical approaches needs to be enhanced if they are to be taken seriously by industrialists

[b]Obviously, the formal specification must be analysed using mathematical reasoning and tools to provide confidence in its correctness. The validation of a formal specification can be carried out using mathematical proof of key properties of the specification, software inspections or specification animation

[c]Approaches such as VDM include a method for software development as well as the specification language

[d]Models are extremely valuable as they allow simplification of the reality. A mathematical study of the model demonstrates whether it is a suitable representation of the system. Models allow properties of the proposed requirements to be studied prior to implementation

[e]Stepwise refinement involves rewriting a specification with each refinement step producing a more concrete specification (that includes code and formal specification) until eventually the detailed code is produced. However, tool support may make refinement easier

[f]Approaches such as VDM or Z are useful in that they add greater rigour to the software development process. They are reasonably easy to learn, and there have been some good results obtained by their use. Classical mathematics is familiar to students, and, therefore, it is desirable that new formalisms are introduced only where absolutely necessary

However, formal methods are potentially quite useful and reasonably easy to use. The use of a formal method such as Z or VDM forces the software engineer to be precise and helps to avoid ambiguities present in natural language. Clearly, a formal specification should be subject to peer review to provide confidence in its correctness. New formalisms need to be intuitive to be usable by practitioners. The advantage of classical mathematics is that it is familiar to students.

10.5.1 Why Should We Use Formal Methods?

There is a very strong motivation to use best practices in software engineering in order to produce software adhering to high-quality standards. Flaws in software may at best cause minor irritations or major damage to a customer's business including loss of life. Consequently, companies need to employ best practice to develop high-quality software, and formal methods are one leading-edge technology which may be of benefit to companies in reducing the occurrence of defects in software products. Brown [Bro:90] argues that for the safety-critical field:

Comment 10.1 (Missile Safety). *Missile systems must be presumed dangerous until shown to be safe, and that the absence of evidence for the existence of dangerous errors does not amount to evidence for the absence of danger.*

It is quite possible that a software company may be sued for software which injures a third party, and this suggests that companies will need a quality assurance program that will demonstrate that every reasonable practice was considered to prevent the occurrence of defects.

There is some evidence to suggest that the use of formal methods provides savings in the cost of the project. For example, a 9% cost saving is attributed to the use of formal methods during the CICS project; the T800 project attributes a 12-month reduction in testing time to the use of formal methods. These are discussed in more detail in chapter one of [HB:95].

10.5.2 Applications of Formal Methods

Formal methods have been employed to verify correctness in the nuclear power industry, the aerospace industry, the security technology area and the railroad domain. These sectors are subject to stringent regulatory controls to ensure safety and security. Several organisations have piloted formal methods with varying degrees of success. These include IBM, who developed VDM at its laboratory in Vienna; IBM (Hursley) piloted the *Z* formal specification language in the CICS (Customer Information Control System) project.

The mathematical techniques developed by Parnas (i.e. requirements model and tabular expressions) have been employed to specify the requirements of the A-7 aircraft as part of a research project for the US Navy.[12] Tabular expressions have also been employed for the software inspection of the automated shutdown software of the Darlington nuclear power plant in Canada.[13] These are two successful uses of mathematical techniques in software engineering.

There are examples of the use of formal methods in the railway domain, and examples dealing with the modelling and verification of a railroad gate controller and railway signalling are described in [HB:95]. Clearly, it is essential to verify safety-critical properties such as '*when the train goes through the level crossing, then the gate is closed*'.

[12] However, the resulting software was never actually deployed on the A-7 aircraft.

[13] This was an impressive use of mathematical techniques, and it has been acknowledged that formal methods must play an important role in future developments at Darlington. However, given the time and cost involved in the software inspection of the shutdown software, some managers have less enthusiasm in shifting from hardware to software controllers [Ger:94].

10.5.3 Tools for Formal Methods

One key criticism of formal methods is the lack of available or usable tools to support the software engineer in writing the formal specification or in doing the proof. Many of the early tools were criticised as not being of industrial strength. However, in recent years, more advanced tools to support the software engineer's work in formal specification and formal proof have become available, and this is likely to continue in the coming years.

The tools include syntax checkers to check that the specification is syntactically correct, specialised editors to ensure that the written specification is syntactically correct, tools to support refinement, automated code generators to generate a high-level language corresponding to the specification, theorem provers to demonstrate the presence or absence of key properties and to prove the correctness of refinement steps and to identify and resolve proof obligations and specification animation tools where the execution of the specification can be simulated.

The B-Toolkit from B-Core is an integrated set of tools that supports the B-Method. These include syntax and type checking, specification animation, proof obligation generator, an auto-prover, a proof assistor and code generation. Thus, in theory, a complete formal development from initial specification to final implementation may be achieved, with every proof obligation justified, leading to a provably correct program.

The IFAD Toolbox[14] is a support tool for the VDM-SL specification language, and it includes support for syntax and type checking, an interpreter and debugger to execute and debug the specification and a code generator to convert from VDM-SL to C++. It also includes support for graphical notations such as the OMT/UML design notations.

10.5.4 Model-Oriented Approach

There are two key approaches to formal methods, namely, the model-oriented approach of VDM or Z and the algebraic or axiomatic approach. The latter includes the process calculi such as the calculus communicating systems (CCS) or communicating sequential processes (CSP).

A model-oriented approach to specification is based on mathematical models. A mathematical model is a mathematical representation or abstraction of a physical entity or system. The representation or model aims to provide a mathematical explanation of the behaviour of the system or the physical world. A model is considered suitable if its properties closely match the properties of the system,

[14] The IFAD Toolbox has been renamed to VDM Tools as IFAD sold the VDM Tools to CSK in Japan. The tools are expected to be available worldwide and will be improved further.

and if its calculations match and simplify calculations in the real system and if predictions of future behaviour may be made. The physical world is dominated by models, for example, models of the weather system, that enable predictions of the weather to be made, and economic models that enable predictions of the future performance of the economy may be made.

It is fundamental to explore the model and to consider the behaviour of the model and the behaviour of the physical world entity. The adequacy of the model is the extent to which it explains the underlying physical behaviour and allows predictions of future behaviour to be made. This will determine its acceptability as a representation of the physical world. Models that are ineffective will be replaced with newer models which offer a better explanation of the manifested physical behaviour. There are many examples in science of the replacement of one theory by a newer one. For example, the Copernican model of the universe replaced the older Ptolemaic model, and Newtonian physics was replaced by Einstein's theories on relativity. The structure of the revolutions that take place in science is described in [Kuh:70].

The model-oriented approach to software development involves defining an abstract model of the proposed software system. The model acts as a representation of the proposed system, and the model is then explored to assess its suitability. The exploration of the model takes the form of model interrogation, that is, asking questions and determining the effectiveness of the model in answering the questions. The modelling in formal methods is typically performed via elementary discrete mathematics, including set theory, sequences, functions and relations.

The modelling approach is adopted by the Vienna Development Method (VDM) and Z. VDM arose from work done in the IBM laboratory in Vienna in formalising the semantics for the PL/1 compiler, and it was later applied to the specification of software systems. The Z specification language had its origins in work done at Oxford University in the early 1980s.

10.5.5 Axiomatic Approach

The axiomatic approach focuses on the properties that the proposed system is to satisfy, and there is no intention to produce an abstract model of the system. The required properties and behaviour of the system are stated in mathematical notation. The difference between the axiomatic specification and a model-based approach is may be seen in the example of a stack.

The stack includes operators for pushing an element onto the stack and popping an element from the stack. The properties of *pop* and *push* are explicitly defined in the axiomatic approach. The model-oriented approach constructs an explicit model of the stack, and the operations are defined in terms of the effect that they have on the model. The specification of the *pop* operation on a stack is given by axiomatic properties, for example, $pop(push(s,x)) = s$.

Comment 10.2 (Axiomatic Approach). *The property-oriented approach has the advantage that the implementer is not constrained to a particular choice of implementation, and the only constraint is that the implementation must satisfy the stipulated properties.*

The emphasis is on the identification and expression of the required properties of the system, and implementation issues are avoided. The focus is on the specification of the underlying behaviour, and properties are typically stated using mathematical logic or higher-order logics. Mechanised theorem-proving techniques may be employed to prove results.

One potential problem with the axiomatic approach is that the properties specified may not be satisfiable in any implementation. Thus, whenever a 'formal axiomatic theory' is developed, a corresponding 'model' of the theory must be identified, in order to ensure that the properties may be realised in practice. That is, when proposing a system that is to satisfy some set of properties, there is a need to prove that there is at least one system that will satisfy the set of properties.

10.5.6 Proof and Formal Methods

The word *proof* has several connotations in various disciplines; for example, in a court of law, the defendant is assumed innocent until proven guilty. The proof of the guilt of the defendant may take the form of certain facts in relation to the movements of the defendant, the defendant's circumstances, the defendant's alibi, statements taken from witnesses, rebuttal arguments from the defence and certain theories produced by the prosecution or defence. Ultimately, in the case of a trial by jury, the defendant is judged guilty or not guilty depending on the extent to which the jury has been convinced by the arguments made by the prosecution and defence.

A mathematical proof typically includes natural language and mathematical symbols, and often, many of the tedious details of the proof are omitted. The strategy of proof in proving a conjecture is often *a divide and conquer* technique, that is, breaking the conjecture down into subgoals and then attempting to prove the subgoals. Most proofs in formal methods are concerned with cross-checking on the details of the specification or are concerned with checking the validity of refinement steps or proofs that certain properties are satisfied by the specification. There are often many tedious lemmas to be proved, and theorem provers[15] are essential in assisting with this. Machine proof needs to be explicit, and reliance on some brilliant insight is avoided. Proofs by hand are notorious for containing errors or jumps in reasoning, while machine proofs are often extremely lengthy and unreadable.

[15] Most existing theorem provers are difficult to use and are for specialist use only. There is a need to improve the usability of theorem provers.

A mathematical proof consists of a sequence of formulae, where each element is either an axiom or derived from a previous element in the series by applying a fixed set of mechanical rules. One well-known theorem prover is the Boyer/Moore theorem prover [BoM:79]. There is an interesting case in the literature concerning the proof of correctness of the VIPER microprocessor[16] [Tie:91], and the actual machine proof consisted of several million formulae.

Theorem provers are invaluable in resolving many of the thousands of proof obligations that arise from a formal specification, and it is not feasible to apply formal methods in an industrial environment without the use of machine assisted proof. Automated theorem proving is difficult, as often mathematicians prove a theorem with an initial intuitive feeling that the theorem is true. Human intervention to provide guidance or intuition improves the effectiveness of the theorem prover.

The proof of various properties about a program increases confidence in its correctness. However, an absolute proof of correctness[17] is unlikely except for the most trivial of programs. A program may consist of legacy software which is assumed to work; it is created by compilers which are assumed to work correctly. Theorem provers are programs which are assumed to function correctly. The best that formal methods can claim is increased confidence in correctness of the software rather than an absolute proof of correctness.

10.5.7 The Future of Formal Methods

The debate concerning the level of use of mathematics in software engineering is still ongoing. Most practitioners are against the use of mathematics and avoid its use. They tend to employ methodologies such as software inspections and testing to improve confidence in the correctness of the software. They argue that in the current competitive industrial environment where time to market is a key driver, the use of such formal mathematical techniques would seriously impact the market opportunity. Industrialists often need to balance conflicting needs such as quality, cost and delivering on time. They argue that the commercial necessities require methodologies and techniques that allow them to achieve their business goals.

The other camp argues that the use of mathematics helps in the delivery of high-quality and reliable software, and that if a company does not place sufficient emphasis on quality will pay a price in terms of a poor reputation in the marketplace.

[16] This verification was controversial with RSRE and Charter overselling VIPER as a chip design that conforms to its formal specification.

[17] This position is controversial with others arguing that if correctness is defined mathematically, then the mathematical definition (i.e. formal specification) is a theorem, and the task is to prove that the program satisfies the theorem. They argue that the proofs for non-trivial programs exist, and that the reason why there are not many examples of such proofs is due to a lack of mathematical specifications.

It is generally accepted that mathematics and formal methods must play a role in the safety-critical and security-critical fields. Apart from that, the extent of the use of mathematics is a hotly disputed topic. The pace of change in the world is extraordinary, and companies face competitive forces in a global marketplace. It is unrealistic to expect companies to deploy formal methods unless they have clear evidence that it will support them in delivering commercial products to the marketplace ahead of their competition, at the right price and with the right quality. Formal method needs to prove that it can do this if it wishes to be taken seriously in mainstream software engineering. The issue of technology transfer of formal methods to industry is discussed in [ORg:06].

10.6 Propositional and Predicate Calculus

Propositional calculus associates a truth value with each proposition and is widely employed in mathematics and logic. There are a rich set of connectives employed in the calculus for truth functional operations, and these include $A \Rightarrow B, A \wedge B, A \vee B$ which denote, respectively, the conditional of A and B, the conjunction of A and B and the disjunction of A and B. A truth table may be constructed to show the results of these operations on the binary values of A and B. That is, A and B have the binary truth values of *true* and *false*, and the result of the truth functional operation is to yield a binary value. There are other logics that allow more than two truth values. These include, for example, the logic of partial functions which is a 3-valued logic. This logic allows a third truth value (the undefined truth value) for the proposition as well as the standard binary values of true and false.

Predicate calculus includes variables, and a formula in predicate calculus is built up from the basic symbols of the language. These symbols include variables; predicate symbols, including equality; function symbols, including the constants; logical symbols, for example, $\exists$, $\wedge$, $\vee$, $\neg$, etc.; and the punctuation symbols, for example, brackets and commas. The formulae of predicate calculus are built from terms, where a *term* is a key construct and is defined recursively as a variable or individual constant or as some function containing terms as arguments. A formula may be an atomic formula or built from other formulae via the logical symbols. Other logical symbols are then defined as abbreviations of the basic logical symbols.

An interpretation gives meaning to a formula. If the formula is a sentence (i.e. it does not contain any free variables), then the given interpretation is true or false. If a formula has free variables, then the truth or falsity of the formula depends on the values given to the free variables. A formula with free variables essentially describes a relation say, $R(x_1,x_2,\ldots, x_n)$ such that $R(x_1,x_2,\ldots, x_n)$ is true if $(x_1,x_2,\ldots, x_n)$ is in relation R. If a formula with free variables is true irrespective of the values given to the free variables, then the formula is true in the interpretation.

A valuation function is associated with the interpretation, and this gives meaning to the formulae in the language. Thus, associated with each constant c is a constant c_Σ in some universe of values Σ; with each function symbol f, we have a function symbol f_Σ in Σ; and for each predicate symbol P, we have a relation P_Σ in Σ. The valuation function, in effect, gives a semantics to the language of the predicate calculus L. The truth of a proposition P is then defined in the natural way, in terms of the meanings of the terms, the meanings of the functions, predicate symbols and the normal meanings of the connectives.

Mendelson [Men:87] provides a rigorous though technical definition of truth in terms of satisfaction (with respect to an interpretation M). Intuitively, a formula F is *satisfiable* if it is *true* (in the intuitive sense) for some assignment of the free variables in the formula F. If a formula F is satisfied for every possible assignment to the free variables in F, then it is *true* (in the technical sense) for the interpretation M. An analogous definition is provided for *false* in the interpretation M.

A formula is *valid* if it is true in every interpretation; however, as there may be an uncountable number of interpretations, it may not be possible to check this requirement in practice. M is said to be a model for a set of formulae if and only if every formula is true in M.

There is a distinction between proof theoretic and model theoretic approaches in predicate calculus. *Proof theoretic* is essentially syntactic, and we have a list of axioms with rules of inference. In this way, the theorems of the calculus may be logically derived (written as $\vdash A$). In essence, the logical truths are a result of the syntax or form of the formulae rather than the *meaning* of the formulae. *Model theoretical*, in contrast, is essentially semantic. The truths derive essentially from the meaning of the symbols and connectives rather than the logical structure of the formulae. This is written as $\vdash_M A$.

A calculus is *sound* if all the logically valid theorems are true in the interpretation, that is, proof theoretic $\Rightarrow$ model theoretic. A calculus is complete if all the truths in an interpretation are provable in the calculus, that is, model theoretic $\Rightarrow$ proof theoretic. A calculus is *consistent* if there is no formula A such that $\vdash$ A and $\vdash \neg$A.

10.7 Unified Modelling Language

The unified modelling language (UML) is a visual modelling language for software systems, and it facilitates the understanding of the architecture of the system and in managing the complexity of large systems. It was developed by Jim Rumbaugh, Grady Booch and Ivar Jacobson [Jac:99a] as a notation for modelling object-oriented systems.

It allows the same information to be presented in several different ways, and there are UML diagrams for alternate viewpoints of the system. Use cases describe scenarios or sequences of actions for the system from the user's viewpoint. For example, typical user operations at an ATM machine include the balance inquiry operation, the withdrawal of cash and the transfer of funds from one account to another. These operations can be described with UML use case diagrams.

Class and object diagrams are a part of UML, and the concept of class and objects is taken from object-oriented design. The object diagram is related to the class diagram in that the object is an instance of the class. There will generally be several objects associated with the class. The class diagram describes the data structure and the allowed operations on the data structure. Two key classes are customers and accounts for an ATM system, and this includes the data structure for customers and accounts and also the operations on customers and accounts. The operations include adding or removing a customer and operations to debit or credit an account. The objects of the class are the actual customers of the bank and their corresponding accounts.

Sequence diagrams show the interaction between objects/classes in the system for each use case. UML activity diagrams are similar to flowcharts. They are used to show the sequence of activities in a use case and include the specification of decision branches and parallel activities. State diagrams (or state charts) show the dynamic behaviour of a class and how different operations result in a change of state. There is an initial state and a final state, and the different operations result in different states being entered and exited.

UML offers a rich notation to model software systems and to understand the proposed system from different viewpoints. The main advantage of UML includes the fact that it is an expressive visual modelling language that allows a study of the proposed system prior to implementation. It allows the system to be visualised from different viewpoints and provides an effective mechanism to communicate the proposed behaviour of the software system.

10.8 Software Inspections and Testing

Software inspections and testing play a key role in building quality into software products and verifying that the products are of high quality. The Fagan Inspection Methodology was developed by Michael Fagan of IBM in the mid-1970s [Fag:76]. It is a seven-step process that identifies and removes errors in work products. There is a strong economic case for identifying defects as early as possible, as the cost of correction increases the later a defect is discovered in the life cycle. The methodology mandates that requirement documents, design documents, source code and test plans are all formally inspected by independent experts to ensure quality.

There are several *roles* defined in the process including the *moderator* who chairs the inspection; the *reader's* responsibility is to read or paraphrase the particular deliverable; the *author* is the creator of the deliverable and has a special interest in ensuring that it is correct; and the *tester* role is concerned with the testing viewpoint.

The inspection process will consider whether a design is correct with respect to the requirements and whether the source code is correct with respect to the design. There are seven stages in the inspection process [ORg:02]:

- Planning
- Overview

- Prepare
- Inspect
- Process improvement
- Rework
- Follow-up

The errors identified in an inspection are classified into various types, and mature organisations record the inspection data in a database for further analysis. Measurement allows the effectiveness of the organisation in identifying errors in phase and detecting defects out of phase to be determined and improved. Tom Gilb has defined an alternate inspection methodology [Glb:94].

Software testing plays a key role in verifying that a software product is of high quality and conforms to the customer's quality expectations. Testing is both a constructive activity in that it is verifying the correctness of functionality, and it is also a destructive activity in that the objective is to find as many defects as possible in the software. The testing verifies that the requirements are correctly implemented as well as identifies whether any defects are present in the software product.

There are various types of testing such as unit testing, integration testing, system testing, performance testing, usability testing, regression testing and customer acceptance testing. The testing needs to be planned to ensure that it is effective. Test cases will need to be prepared and executed, the results reported and any issues corrected and retested. The test cases will need to be appropriate to verify the correctness of the software. The quality of the testing is dependent on the maturity of the test process, and a good test process will include:

- Test planning and risk management
- Dedicated test environment and test tools
- Test case definition
- Test automation
- Formality in handover to test department
- Test execution
- Test result analysis
- Test reporting
- Measurements of test effectiveness
- Postmortem and test process improvement

Metrics are generally maintained to provide visibility into the effectiveness of the testing process. Testing is described in more detail in [ORg:02, ORg:10].

10.9 Process Maturity Models

The Software Engineering Institute (SEI) developed the Capability Maturity Model (CMM) in the early 1990s as a framework to help software organisations to improve their software process maturity and to implement best practice in software and

Fig. 10.6 W. Edwards Deming (Courtesy of W. Edwards Deming Institute)

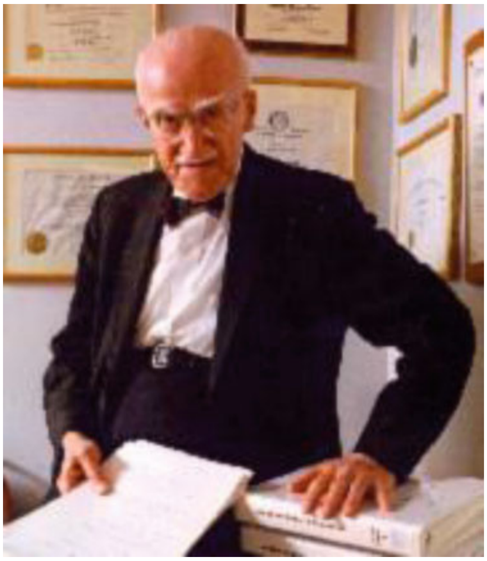


Fig. 10.7 W. Joseph Juran (Courtesy of Juran Institute)

systems engineering. The SEI believes that there is a close relationship between the maturity of software processes and the quality of the delivered software product. The CMM applied the ideas of Deming [Dem:86], Juran [Jur:00] and Crosby [Crs:79] to the software field. These quality gurus were influential in transforming manufacturing companies with quality problems to effective quality-driven organisations with a reduced cost of poor quality (Fig. 10.6).

They recognised the need to focus on the process, and software organisations need to improve their software development processes as well as the product. Watt Humphries did early work on software process improvement at IBM [Hum:89], and he moved to the SEI in the late 1980s. This led to the first version of the CMM in 1991. It is now called the Capability Maturity Model Integration (CMMI®) [CKS:11] (Fig. 10.7).

Fig. 10.8 Watts Humphrey (Courtesy of Watts Humphrey)

The CMMI consists of five maturity levels, with each maturity level (except level 1) consisting of several process areas. Each process area consists of a set of goals that must be satisfied for the process area to be satisfied. The goals for the process area are implemented by practices related to that process area, and the implementation of these practices leads to an effective process. Processes need to be defined and documented. The users of the process need to receive appropriate training to enable them to carry out the process, and processes need to be enforced by independent audits (Fig. 10.8).

The emphasis on level 2 of the CMMI is on maturing management practices such as project management, requirements management, configuration management and so on. The emphasis on level 3 of the CMMI is to mature engineering and organisation practices. This maturity level includes peer reviews and testing, requirements development, software design and implementation practices and so on. Level 4 is concerned with ensuring that key processes are performing within strict quantitative limits and adjusting processes, where necessary, to perform within these defined limits. Level 5 is concerned with continuous process improvement which is quantitatively verified.

Maturity levels may not be skipped in the staged implementation of the CMMI. There is also a continuous representation of the CMMI which allows the organisation to focus on improvements to key processes. However, in practice, it is often necessary to implement several of the level 2 process areas before serious work can be done on implementing a process at a higher maturity level. The use of metrics [Fen:95, Glb:76] becomes more important as an organisation matures, as metrics allow the performance of an organisation to be objectively judged. The higher CMMI maturity levels set quantitative levels for processes to perform within.

The CMMI allows organisations to benchmark themselves against other similar organisations. This is done by formal SEI approved SCAMPI appraisals conducted by an authorised SCAMPI lead appraiser. The results of a SCAMPI appraisal are generally reported back to the SEI, and there is a strict qualification process to become an authorised SCAMPI lead appraiser. An appraisal is useful in verifying that an organisation has improved, and it enables the organisation to prioritise improvements for the next improvement cycle.

The time required to implement the CMMI in an organisation depends on the current maturity and size of the organisation. It generally takes 1–2 years to implement maturity level 2, and a further 1–2 years to implement level 3.

10.10 Review Questions

1. Describe the crisis in software in the 1960s and the birth of software engineering.
2. Describe waterfall and spiral life cycle models including their advantages and disadvantages.
3. Discuss Floyd's contribution to software engineering and how it led to Hoare's axiomatic semantics.
4. Describe the mathematics that is potentially useful in software engineering.
5. Describe formal methods and their applications to software engineering. Explain when their use should be considered in software engineering.
6. Discuss any tools to support formal methods that you are familiar with.
7. Discuss the similarities and differences between Z and VDM.
8. Discuss the similarities and differences between the model-oriented approach and the axiomatic approach to formal methods.
9. Discuss UML and its applicability to software engineering.
10. Discuss the applicability of software inspections and testing to software engineering.
11. Discuss the Capability Maturity Model and its applicability to software engineering.

10.11 Summary

This chapter considered a short history of some important developments in software engineering from its birth at the Garmisch conference in 1968. It was recognised that there was a crisis in the software field, and there was a need for sound methodologies to design, develop and maintain software to meet customer needs.

Classical engineering has a successful track record in building high-quality products that are safe for the public to use. It is therefore natural to consider using an engineering approach to developing software, and this involves identifying the customer requirements, carrying out a rigorous design to meet the requirements, developing and coding a solution to meet the design and conducting appropriate inspections and testing to verify the correctness of the solution.

Mathematics plays a key role in engineering to assist with design and verification of software products. It is therefore reasonable to apply appropriate mathematics in software engineering (especially for safety-critical systems) to assure that the delivered systems conform to the requirements. The extent to which mathematics will need to be used is controversial with strong views on both sides. In many cases, peer reviews and testing will be sufficient to build quality into the software product. In other cases, and especially with safety and security-critical applications, it is desirable to have the extra assurance that may be provided by mathematical techniques.

Various mathematical approaches were considered including Z, VDM, propositional calculus and predicate calculus. The nature of mathematical proof and supporting theorem provers were discussed.

There is a lot more to the successful delivery of a project than just the use of mathematics or peer reviews and testing. Sound project management and quality management practices are essential, as a project that is not properly managed will suffer from schedule, budget or cost overruns as well as problems with quality.

Maturity models such as the CMMI can assist organisations in maturing key management and engineering practices that are essential for the successful delivery of high-quality software. The use of the CMMI helps companies in their goals to deliver high-quality software systems that are consistently on time and consistently meet business requirements.

Chapter 11
People in Computing

Key Topics

Konrad Zuse
John von Neumann
Mauchly and Eckert
Gene Amdahl
Fred Brooks
Donald Knuth
C.A.R Hoare
Edsger Dijkstra
David Parnas
Dennis Ritchie
Richard Stallman
Ed Yourdan
Tim Berners-Lee
Wilhelm Gottfried Leibniz
Archimedes

11.1 Introduction

The objective of this chapter is to give a flavour of some of the people who have made important contributions to the computing field. A small selection is considered as it is not feasible, due to space constraints, to consider all those who merit inclusion.

The individuals considered include Konrad Zuse who is considered to be the 'father of the computer' in Germany and John von Neumann who did important work on early computers and who developed (along with Mauchly and Eckert) the von Neumann architecture which is the fundamental architecture of the computer systems used today. Gene Amdahl was the chief designer of the IBM 360 series of computers, and he later founded the Amdahl Corporation. Fred Brooks was the project manager for the IBM 360 project, and he later wrote the influential book, *The Mythical Man Month*, which describes the challenge of delivering a large project in which software is a major constituent on time, on budget and with the right quality.

Donald Knuth is considered to be the father of the analysis of algorithms, and he has published the ultimate guide to program development in his massive four volume work *The Art of Computer Programming*. He also developed the TEX and METAFONT typesetting systems.

C.A.R Hoare is a famous British computer scientist who has made fundamental contributions to computing including the development of the quicksort algorithm and the development of axiomatic semantics and CSP.

Edsger Dijkstra made important contributions to the computing field with his development of graph algorithms, his work on Algol 60 and his development of the calculus of weakest preconditions.

David Parnas has made important contributions to the software field and his ideas on the specification, design, implementation, maintenance and documentation of computer software remain important.

Dennis Ritchie developed the C programming language and codeveloped the UNIX operating system with Ken Thompson. Richard Stallman is the founder of the free software movement with the GNU project. Ed Yourdan has made contributions to systems analysis and design methodologies. Tim Berners-Lee invented the World Wide Web which has revolutionised the computing field. Finally, we discuss the contributions of Leibniz and Archimedes.

11.2 Zuse

Konrad Zuse was born in Berlin in 1910. He built the world's first programmable computer in 1941. He studied civil engineering at the Technical University of Berlin, and he was talented at mechanical construction. He won several prizes as a student for his constructions.

He commenced working as a stress analyser for Henschel after his graduation in 1935. Henschel was an airline manufacturer, and Zuse stayed in his position for less than a year. He resigned with the intention of forming his own company to build automatic calculating machines.

His parents provided financial support, and he commenced work on what would become the Z1 machine in 1936. Zuse employed the binary system for the calculator and metallic shafts that could shift from position 0 to 1 and vice versa. The Z1 was operational by 1938 (Fig. 11.1).

He served in the German Army on the Eastern Front for 6 months in 1939 at the start of the Second World War. Henschel helped Zuse to obtain a deferment from the army and made the case that he was needed as an engineer and not as a soldier.

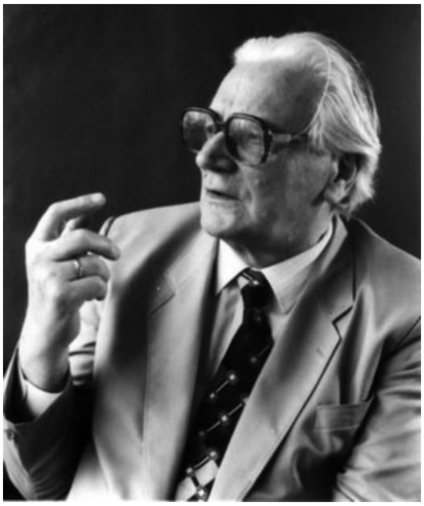


Fig. 11.1 Konrad Zuse (Courtesy of Horst Zuse)

Zuse recommenced working at Henschel in 1940 for the duration of the war, and he developed the Z2 and Z3 machines there. The Z3 was operational in 1941 and was the world's first programmable computer. He started his own company in 1941, and this was the first company founded with the sole purpose of developing computers. The Z4 was almost complete as the Red Army advanced on Berlin in 1945, and Zuse left Berlin for Bavaria with the Z4 prior to the Russian advance.

He designed the world's first high-level programming language between 1943 and 1945. This language was called Plankalkül, and it was discussed in Chap. 9. He later restarted his company (Zuse KG), and he completed the Z4 in 1950. This was the first commercial computer as it was completed ahead of the Ferranti Mark 1, UNIVAC and LEO computers. Its first customer was the Technical University of Zurich.

Zuse's results are all the more impressive given that he was working alone in Germany, and he was unaware of the developments taking place in other countries. He had no university support or graduate students to assist him.

11.3 von Neumann

John von Neumann was a Hungarian mathematician who made fundamental contributions to mathematics, set theory, computer science, economics and quantum theory. He was born in Budapest, Hungary, in 1903 and moved to the United States in the early 1930s (Fig. 11.2).

His PhD degree was concerned with the axiomatisation of set theory and dealing with the problem of Russell's paradox. Bertrand Russell had posed the question whether the set S of all sets that do not contain themselves as members contains itself as a member, that is, does $S \in S$? In either case, a contradiction arises since

Fig. 11.2 John von Neumann. Los Alamos (Courtesy US Government Archives)

if $S \in S$, then as S is a set that does not contain itself as a member, and, therefore, $S \notin S$. Similarly, if $S \notin S$, then S is a set that does not contain itself as a member, and, therefore, $S \in S$.

von Neumann showed how the contradiction in set theory can be avoided in two ways. One approach is to employ the axiom of foundation, and the other approach is to employ the concept of a class.

He considered the problem of the axiomatisation of quantum theory and showed how the physics of quantum theory could be reduced to the mathematics of linear operators on Hilbert spaces [VN:32]. His theory included the Schödinger wave mechanical formulation and Heisenberg matrix mechanical formulation as special cases. von Neumann and Birkhoff later proved that quantum mechanics requires a logic that is quite different from classical logic.

He made contributions to economics and game theory. He was interested in practical problems and worked as a consultant to the US Navy, IBM, the CIA and the Rand Corporation. He became an expert in the field of explosions and discovered that large bombs are more devastating if they explode before touching the ground. He contributed to the development of the hydrogen bomb and to improving methods to utilise nuclear energy.

He gave his name to the von Neumann architecture used in almost all computers. Eckert and Mauchly were working with him on this concept during their work on ENIAC and EDVAC. The architecture includes a central processing unit which includes the control unit and the arithmetic unit, an input and output unit and memory.

He also created the field of cellular automata. He is also credited as the inventor of the merge-sort algorithm (in which the first and second halves of an array are each sorted recursively and then merged). He also invented the Monte Carlo method that allows complicated problems to be approximated through the use of random numbers.

He died at the relatively young age of 54 in 1957. There is an annual von Neumann medal awarded by IEEE for outstanding achievements in computer science.

11.4 Mauchly and Eckert

John Mauchly and Presper Eckert designed ENIAC which was one of the earliest digital computers. They went on to design EDVAC and the UNIVAC 1 computer, and they started one of the earliest computer companies, the Eckert-Mauchly Computer Corporation (EMCC). They pioneered some fundamental computer concepts such as the ‘stored program’, ‘subroutines’, and ‘programming languages’.

Mauchly was born in Ohio in 1907. He studied electrical engineering for his undergraduate degree at Johns Hopkins University and then did a PhD in Physics at the university. He became a professor of physics at Ursinus College in Philadelphia.

Eckert was born in Philadelphia in 1919, and he initially studied business at the University of Pennsylvania. He transferred to the Moore School of Electrical Engineering at the university in 1940 and worked on radar and improving the differential analyser at the school. Moore and Eckert met on a course at the Moore School in 1941.

Mauchly joined the lecturing staff at the Moore School in 1941, and he made a proposal to build an electronic computer using vacuum tubes that would be much faster and more accurate than the existing differential analyser. Mauchly discussed his idea further with representatives with the US Army, and this led to funding to build the machine in 1943. This machine was ENIAC, and it was completed in 1945. Mauchly focused on the design and Eckert on the hardware engineering side. ENIAC was discussed in Chap. 3.

Mauchly and Eckert were aware of the limitations of ENIAC, and they commenced work on the design of a successor computer, EDVAC, in early 1945. von Neumann became involved in some of the engineering discussions, and he produced a draft report describing EDVAC. This draft report was intended to be internal, but circumstances changed. This was due to legal issues with respect to inventions and patents in the Moore School which led to the resignation of Mauchly and Eckert to set up their own company. This was in order that they could maintain their patents on ENIAC and EDVAC. The Moore School then removed the names of Mauchly and Eckert from the report and circulated it to the wider community. The report mentioned the architecture that is known today as the ‘von Neumann architecture’, and unfortunately, no credit was given to Mauchly and Eckert.

Eckert and Mauchly launched the Eckert-Mauchly Computer Corporation in 1947. They produced the UNIVAC 1 and BINAC computers and were later taken over by Remington Rand.

11.5 Gene Amdahl

Gene Amdahl is an American computer scientist and entrepreneur. He was born in South Dakota in 1922, and he served in the American Navy during the Second World War. He obtained a degree in engineering physics at South Dakota University in 1948 and was awarded a PhD in 1952 from the University of Wisconsin-Madison.

Fig. 11.3 Gene Amdahl (Photo Courtesy of Perry Pkivolowitz)

His PhD thesis detailed the design of his first computer, the Wisconsin Integrally Sequential Computer (WISC), and the machine was built during the period 1951 to 1954 (Fig. 11.3).

He joined IBM in 1952 and became the chief designer of the IBM 704 computer which was released in 1954. He was the principal architect for the hugely successful System/360 family of mainframe computers which was introduced in 1964. Amdahl was appointed an IBM fellow in recognition of his contribution to IBM and given freedom to pursue his own research projects.

He resigned from IBM in 1970 to found the Amdahl Corporation, and this company became a major competitor to IBM in the mainframe market. Amdahl computers were compatible with IBM computers and software but delivered an improved performance at a lower cost. Customers could run S/360 and S/370 applications without buying IBM hardware.

He resigned from Amdahl in 1979 to pursue other interests, and at that time, Amdahl had over $1 billion of mainframes sales. He set up a new company called Trilogy Systems with the goal of creating an integrated chip for cheaper mainframes, and the company later worked on very-large-scale integration (VLSI) with the goal of creating a supercomputer to rival IBM. However, Trilogy was not successful, and Amdahl left the company in 1989.

11.6 Fred Brooks

Fred Brooks was born in North Carolina in 1931 and obtained a PhD in Applied Mathematics from Harvard University in 1956. His doctoral advisor was Howard Aiken who had designed the Harvard Mark 1. He joined IBM in New York in 1956

Fig. 11.4 Fred Brooks (Photo by Dan Sears)

and worked on the architecture of various IBM machines. He became manager for the project to deliver the System/360 family of computers and the IBM OS/360 operating system.

The System/360 project involved 5,000 man-years of effort at IBM, and Brooks later wrote a famous book based on his experience as project manager on this project. This book, *The Mythical Man Month*, considers the challenge of delivering a major project (of which software is a key constituent) on time, on budget and with the right quality. Brooks describes his book as 'my belated answer to Tom Watson's probing question as to why programming is hard to manage' (Fig. 11.4).

Brooks law is a famous statement made by Brooks in the book. It states '*Adding manpower to a late software project makes it later*'. Most project managers agree with this statement on schedule slippage. The problem is that project staff will need to devote time to training new team members to become familiar with the project. This leads to a loss of team productivity, and it takes time for the new team members to become productive and effective. There are also increased communication overheads.

Brooks has another famous quote on schedule slippage: '*How does a project get to be a year behind schedule? One day at a time*'. This shows the importance of schedule monitoring on a daily and weekly basis during the project, and taking early action to ensure that the schedule remains on track. Often, there are more effective solutions to schedule slippage other than adding more staff to the project.

He wrote a famous paper '*No Silver Bullet – Essence and Accidents of Software Engineering*' in 1986. This paper argues that 'There is no single development in technology or management techniques which by itself promises even one order of magnitude [tenfold] improvement within a decade in productivity, in reliability and simplicity'. Brooks states that while there is no silver bullet to achieve this goal, that a series of innovations could lead to significant improvements and perhaps greater than tenfold improvements over a 10-year period.

Brooks founded the Department of Computer Science at the University of North Carolina in 1964 and chaired it for over 20 years.

11.7 Donald Knuth

Donald Knuth was born in 1938 in Wisconsin in the United States. He studied physics and mathematics at Case Institute of Technology and published two scientific papers while still an undergraduate. He became aware of computers in 1957 when he saw the IMB 650. He was immediately fascinated, and he later stated that it provided the inspiration for his future work. He wrote a user manual and instructional software for the IBM 650 and wrote computer software to crunch the statistics of the institute's basketball team in 1958.

He obtained a PhD in Mathematics from the Californian Institute of Technology in 1963, and was one of the earliest mathematicians to take computer science seriously. He introduced rigour and elegance into programming which was characterised at that time by people writing and rewriting code until all known defects were removed. Knuth's emphasis on rigour aimed to provide a proof that the program was correct (Fig. 11.5).

He is considered the father of the analysis of algorithms, and he has made important contributions to the rigorous analysis of the computational complexity of algorithms. He is the author of the influential book *The Art of Computer Programming* which is published in four volumes. This is regarded as the discipline's definitive reference guide, and it has been a lifelong project for Knuth. The first volume was published in 1968, the second in 1969, the third in 1973 and the fourth appeared in 2011.

He is the creator of the TeX and METAFONT systems for computer typesetting. These programs are used extensively for scientific publishing. He is the author of over 26 books and over 160 papers.

He has received numerous awards including the first Grace Hopper Award in 1971, the Turing Award in 1974 and the National Medal of Science in 1979.

Fig. 11.5 Donald Knuth at Open Content Alliance 2005 (Courtesy of Jacob Applebaum)

11.8 C.A.R Hoare

Charles Anthony Richard (C.A.R or Tony) Hoare studied philosophy (including Latin and Greek) at Oxford University. He studied Russian at the Royal Navy during his National Service in the late 1950s. He then studied statistics and went to Moscow University as a graduate student to study machine translation of languages and probability theory (Fig. 11.6).

He discovered the well-known sorting algorithm, Quicksort, while investigating efficient ways to look up words in a dictionary. He returned to England in 1960 and worked as a programmer for Elliot Brothers, which was a company that manufactured scientific computers. He led a team to successfully produce a commercial compiler for Algol 60. This language had been designed by an international committee, and its concise specification was 21 pages long in Backus Naur Form (BNF) [Nau:60]. Peter Naur's report gave the implementer accurate information to implement a compiler for the language, and there was no need for the implementer to communicate with the language designers.

He then led a project team to implement an operating system, and this project was a disaster. He managed to steer a recovery from the disaster, and he then moved to the research division in the company.

Hoare argues for simplicity in language design and argued against languages such as Algol 68 and ADA which attempted to be all things to all people. He noted the flaws in the design of the Algol 68 language with the statement:

> *Algol 60 was a great achievement in that it was a significant advance over most of its successors.*

Fig. 11.6 C.A.R Hoare

He took a position at Queens University in Belfast in 1968, and his research goals included examining techniques to assist with the implementation of operating systems. He was especially interested in to see if advances in programming methodologies could assist with the problems of concurrency. He also published material on the use of assertions to prove program correctness, and he developed the axiomatic semantics of programming languages.

An assertion, for example, ($x - y > 5$), may or may not be satisfied by a state of the program during execution. It is true in the state where the values of the variables x and y are 7 and 1, respectively, and false in the state where x and y have values 4 and 2, respectively. Assertions were applied by Floyd to flowcharts in 1967, and Hoare built upon Floyd's work to show how the semantics of a small programming language could be defined with precondition and postcondition assertions.

The semantics of a program fragment a is given by an expressions of the form $P\{a\}$ Q where P is the precondition, a is the program fragment and Q is the postcondition. The precondition P is a predicate (or input assertion), and the postcondition Q is a predicate (output assertion). He then showed how this could be applied to prove partial correctness of programs. His quote on simplicity in software design is well known:

There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complex that there are no obvious deficiencies.

He moved to Oxford University in 1977 following the death of Christopher Strachey (who was well known for his work in denotational semantics), and he built up the programming research group. This group produced the Z specification language and Communicating Sequential Processes (CSP). The latter is a mathematical approach to the study of communication and concurrency and is applicable to the specification and design of computer systems that continuously interact with their environment.

He received the Turing Award in 1980, and following his retirement from Oxford, he took up a position as senior researcher at Microsoft Research in the United Kingdom. Hoare has made fundamental contributions to computing including the Quicksort algorithm, the axiomatic approach to program semantics and programming constructs for concurrency.

11.9 Edsger Dijkstra

Edsger Dijkstra was born in Rotterdam in Holland and studied mathematics and physics at the University of Leyden. He obtained a PhD in Computer Science from the University of Amsterdam in 1959. He commenced his programming career at the Mathematics Centre in Amsterdam in the early 1950s and developed various efficient graph algorithms to determine the *shortest* or *longest* paths from a vertex u to a vertex v in a graph.

Fig. 11.7 Edsger Dijkstra (Courtesy of Brian Randell)

He contributed to the definition of Algol 60 and designed and coded the first Algol 60 compiler. He became a professor of mathematics at Eindhoven University in the early 1960s. His work on operating systems showed that they can be built as synchronised sequential processes, and he introduced ideas such as semaphores and deadly embrace (Fig. 11.7).

The approach to software development was informal in the 1960s. John McCarthy argued in 1962 that the focus should be to prove that the programs have the desired properties rather than testing the program ad nauseum. The NATO 1968 software engineering conference highlighted the extent of the s*oftware crisis* with projects being delivered late and with poor quality. Dijkstra made important contributions to structured programming, and his article against the use of the GOTO statement in programming '*Go to statement considered harmful*' was influential in the trend towards structured programming.

He advocated simplicity, precision, and a formal approach to program development using the calculus of weakest preconditions. He insisted that programs should be composed correctly using mathematical techniques and not debugged into correctness. He considered testing to be an inappropriate means of building quality into software, and his statement on software testing was influential:

Testing a program shows that it contains errors never that it is correct.

His calculus of weakest preconditions was used to develop reliable programs. This led to a science of programming [Gri:81] and the use of logic to provide formal proof of program correctness. The idea is that the program and its proof should be developed together with proofs involving weakest preconditions and the formal definition of the programming constructs (e.g. assignment and iteration).

Programming is viewed as a goal-oriented activity in that the desired result (i.e. the postcondition R) plays a more important role in the development of the program than the precondition Q. Programming is employed to solve a problem, and the problem needs to be clearly stated with precise pre- and postconditions.

He received the Turing award in 1972, and moved to the University of Texas in Austin in 1984.

11.10 David Parnas

David Lorge Parnas has been influential in the computing field, and his ideas on the specification, design, implementation, maintenance and documentation of computer software remain relevant today. He has won numerous awards (including ACM best paper award in 1979, two most influential paper awards from ICSE in 1978 and 1984, the ACM SigSoft outstanding researcher award in 1998 and an honorary doctorate from the ETH in Zurich and the Catholic University of Louvain in Belgium) for his contribution to computer science. Software engineers today continue to use his ideas in their work.

He was born in the United States on February 10, 1941, (or 41,41 in Parnas speak). He studied at Carnegie Mellon University in the United States and was awarded BS, MS and PhD degrees in Electrical Engineering by the university. He has worked in both industry and academia, and his approach aims to achieve a middle way between theory and practice. His research has focused on real industrial problems that engineers face and on finding solutions to these practical problems. Several organisations such as Phillips in the Netherlands, the Naval Research Laboratory (NRL) in Washington, IBM Federal Systems Division and the Atomic Energy Board of Canada have benefited from his talent, expertise and advice.

His ideas on the specification, design, implementation, maintenance and documentation of computer software remain important. He advocates a solid engineering approach to the development of high-quality software and argues that the role of the engineer is to apply scientific principles and mathematics to design and develop useful products. He argues that computer scientists should be educated as engineers and provided with the right scientific and mathematical background to do their work effectively (Fig. 11.8).

His contributions to software engineering include:

- *Tabular expressions*
- *Mathematical documentation*
- *Requirements specification*
- *Software design*

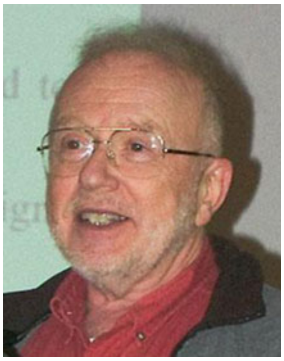


Fig. 11.8 David Parnas (Courtesy of Hubert Baumeister)

- *Software inspections*
- *Predicate logic*
- *Information hiding*

He played an important role in criticising the Strategic Defence Initiative (SDI) launched by President Reagan in the early 1980s. This initiative planned to use ground and space-based systems to protect the United States from nuclear attack. The initiative was too ambitious and unrealistic and threatened to reignite a new arms race. Parnas has argued that engineers have ethical responsibilities as engineers and that engineers should be licensed.

11.11 Dennis Ritchie

Dennis Ritchie was born in New York in 1941, and he did a degree in physics and mathematics at Harvard University. He joined Bell Labs in 1967 and was involved in the Multics operating system project. He designed and implemented the C programming language at Bell Labs in the early 1970s. The origin of this language is closely linked to the development of the UNIX operating system, and C was originally used for systems programming. It later became very popular for both systems and application programming and influenced the development of other languages such as C++ and Java.

Brian Kernighan wrote the first tutorial on C, and Kernighan and Ritchie later wrote the well-known book *The C Programming Language* which appeared in 1978.

The UNIX operating system was developed by Ken Thompson, Dennis Ritchie and others at Bell Labs in the early 1970s. It is a multitasking and multiuser operating system written almost entirely in C. UNIX arose out of work by Massachusetts Institute of Technology and Bell Labs on the development of an experimental operating system called Multics (Fig. 11.9).

This project failed but several of the Bell Lab researchers decided to redo the work on a smaller-scale operating system on a PDP-7 and later on a PDP-11 computer. The result was UNIX, and it became a widely used operating system that was used initially by universities and the US government. Ritchie later became head of Lucent Technologies Systems Software Research Department.

Ritchie and Thompson received the Turing Award in 1983 in recognition of their achievements in their implementation of the UNIX operating system. They received the National Medal in Technology from Bill Clinton in 1999.

11.12 Richard Stallman

Richard Stallman was born in New York in 1953 and became interested in computers while still at high school, after spending a summer working at IBM. He obtained a degree in physics at Harvard University in 1974 and joined the

Fig. 11.9 Ken Thompson and Dennis Ritchie with President Clinton in 1999

Fig. 11.10 Richard Stallman (Courtesy of Victor Powell)

AI Lab at MIT as a programmer. He had become involved in the hacker community while at Harvard and became a critic of restricted computer access in the lab.

He was convinced that software users should have the freedom to share with their neighbours and to be able to study and make changes to the software that they use. He left his position at MIT in 1984 to launch a free software movement (Fig. 11.10).

He launched the GNU project in 1984 which is a free software movement involving software developers from around the world. He also formed the Free Software Foundation which is a non-profit organisation to promote the free software movement. It developed a legal framework for the free software movement which provides a legal means to protect the modification and distribution rights of free software.

The initial focus of GNU was to create a free operating system called GNU (GNU's Not UNIX). This was achieved in 1992 when the final part of GNU required, the kernel, was provided by the release by Linus Torvalds of the Linux operating system kernel as free software.

The founding goal of the GNU project was to develop '*a sufficient body of free software to get along without any software that is not free*'. The GNU manifesto was written by Stallman to gain support from other developers for the project. Stallman outlines the meaning of free software in the manifesto and lists four freedoms essential to software development:

1. Freedom to run the program
2. Freedom to access the code
3. Freedom to redistribute the program to anyone
4. Freedom to improve the software

The GNU project uses software that is free for users to copy, edit and distribute. It is free in the sense that users can change the software to fit individual needs. Stallman has written many essays on software freedom and is a key campaigner for the free software movement.

11.13 Ed Yourdan

Ed Yourdon was born in 1944 and obtained a degree in applied mathematics from Massachusetts Institute of Technology in 1965. He commenced his computing career in 1964 as a senior programmer with Digital Equipment Corporation. He developed the Fortran Math library for the PDP-5 computer and wrote an assembler for the PDP-8 computer (Fig. 11.11).

He became one of the leading developers of the structured analysis and design methods in the 1970s. These are methods for analysing and converting business requirements into specification and programs. His contribution to object-oriented analysis and design in the late 1980s and 1990s include codeveloping the Yourdan/Whitehead method of object-oriented analysis and design and the Coad/Yourdon object-oriented methodology.

Structured analysis involves drawing diagrams to represent the information flow in the system, as well as showing the key processes at the level of granularity required. Data dictionaries are used to describe the data, and CASE tools are employed to support the drawing of diagrams and recording the objects drawn in a data dictionary.

Fig. 11.11 Edward Yourdan (Courtesy of Edward Yourdan)

He founded an international consulting business, Yourdan Inc., in 1974 with the goal of providing education, publishing and consulting services to clients in software engineering technologies. His consulting business was very successful, and it trained over 250,000 people around the world in structured programming, structured analysis, structured design, logical data modelling and project management. It was sold in the mid-1980s and is now part of CGI. He is the author of over 500 technical articles and over 20 books.

11.14 Stephan Wolfram

Stephan Wolfram is a British scientist who developed the *Mathematica* software application. He was born in 1959 and obtained a PhD in Particle Physics from California Institute of Technology in 1979.

He cofounded Wolfram Research in 1987 and the company develops and markets the computer algebra system, Mathematica. This is a computational software program used in the scientific, engineering and mathematical fields.

Mathematica includes a function library, tools for matrices, support for complex numbers, visualisation tools for 2D and 3D data, tools for solving various types of equations, tools for visualising and analysing graphs, tool support for calculus and many more.

He published *A New Kind of Science* [Wol:02], and this controversial book argues that the universe is digital in nature with its fundamental laws described by simple computer programs. He developed an answer engine called Wolfram|Alpha in 2009. This is an online answer system that answers factual queries using natural language processing. Current search engines provide a list of documents or search pages in response to a query, whereas Wolfram|Alpha computes the answer from a core knowledge base of structured data.

11.15 Tim Berners-Lee

Tim Berners-Lee is a British computer scientist and the inventor of the World Wide Web. He was born in London in 1955 and obtained a degree in physics in 1976 from Oxford University. Both his parents had been involved in the programming of the Ferranti Mark I computer in the 1950s.

He went to CERN in 1980 for a short-term programming assignment. CERN is a key European centre for research in the nuclear field and employs several thousand scientists. He returned to CERN in the mid-1980s to work on other projects, and he devoted part of his free time to consider solutions to the problem of information sharing between scientists at CERN and overseas. His solution to the problem was the invention of the World Wide Web in 1990.

He built on several existing inventions such as the Internet, hypertext and the mouse. Hypertext was invented by Ted Nelson in the 1960s, and it allowed links to be present in text. For example, a document such as a book contains a table of contents, an index and a bibliography. These are all links to material that is either within the book itself or external to the book. The reader of a book is able to follow the link to obtain the internal or external information. The mouse was invented by Douglas Engelbart in the 1960s, and it allowed the cursor to be steered around the screen.

The major leap that Berners-Lee made was essentially a marriage of the Internet, hypertext and the mouse into what has become the World Wide Web. His vision is described in [BL:00] (Fig. 11.12):

> *Suppose that all information stored on computers everywhere were linked. Program computer to create a space where everything could be linked to everything.*

The World Wide Web creates a space in which users can access information easily in any part of the world. This is done using only a web browser and simple web addresses. The user can then click on hyperlinks on web pages to access further relevant information that may be on an entirely different continent. Berners-Lee is now the director of the World Wide Web Consortium, and this MIT-based organisation sets the software standards for the web.

Fig. 11.12 Tim Berners-Lee (Courtesy of Uldis Bojārs)

11.16 Wilhelm Gottfried Leibniz

Wilhelm Gottfried Leibniz was a German philosopher, mathematician and inventor in the field of mechanical calculators. He was born in Leipzig, Germany, to a wealthy Lutheran family. He studied law and philosophy at the University of Leipzig and completed a Doctorate in Law at the University of Altdorf. He spent a few years in Paris in the 1670s and developed links with some of the leading intellectual figures of the seventeenth century. He received lessons in mathematics from the physicist, Christopher Huygens, and his flair for mathematics soon became apparent (Fig. 11.13).

Chapter 2 discussed the calculating machines developed by Leibniz and Pascal. Leibniz's machine was superior to the existing Pascaline machine, and it could perform addition, subtraction, multiplication, division and the extraction of roots. Pascal's machine could just perform addition and subtraction. Leibniz's machine was called the Step Reckoner, and it was first developed in 1673. A metallic version of the machine was presented to the French Academy of Sciences in 1675.

The calculating machine used a cylinder or stepped drum known as the 'Leibniz Wheel'. It consists of a set of wheels of incremental length that were used with a counting wheel in the calculating engine. The Leibniz machine remained in use for a couple of hundred years.

Leibniz recognised the potential of the binary number system and was an advocate of its use. A binary digit can be represented by an on off switch, and when computers became electronic, this was the appropriate way to proceed. However, his calculating machine used decimal numbers rather than binary.

He developed the differential and integral calculus independently of Newton, and this led to a major controversy between them. The controversial question was who developed the calculus first, with Leibniz accused of having plagiarised Newton's work. The consensus today is that both Newton and Leibniz independently developed the calculus. Leibniz's notation was easier to use than Newton's and is widely used today. The last years of Leibniz's life were spent embroiled in this controversy.

Fig. 11.13 Wilhelm Gottfried Leibniz (Public domain)

11.17 Archimedes

Archimedes was a Hellenistic mathematician, astronomer and philosopher and was born in Syracuse[1] in the third century B.C. He was a leading scientist in the Greco-Roman world, and he is credited with designing various innovative machines.

His inventions include the ‘Archimedes’ Screw’ which was a screw pump that is still used today in pumping liquids and solids. Another of his inventions was the ‘Archimedes’ Claw’, which was a weapon used to defend the city of Syracuse. It was also known as the ‘ship shaker’, and it consisted of a crane arm from which a large metal hook was suspended. The claw would swing up and drop down on the attacking ship. It would then lift it out of the water and possibly sink it. Another of his inventions was said to be the ‘Archimedes’ Heat Ray’. This device is said to have consisted of a number of mirrors that allowed sunlight to be focused on an enemy ship thereby causing it to go on fire (Fig. 11.14).

There is a well-known anecdote concerning Archimedes and the crown of King Hiero II. The king wished to determine whether his new crown was made entirely of solid gold, and that no substitute silver had been added by the goldsmith. Archimedes was required to solve the problem without damaging the crown, and as he was taking a bath, he realised that if the crown was placed in water, the water displaced would give him the volume of the crown. From this he could then determine the density of the crown and therefore whether it consisted entirely of gold.

Archimedes is likely to have used a more accurate method using Archimedes’ Principle than that described in the anecdote. He discovered the law of buoyancy known as Archimedes’ principle:

The buoyancy force is equal to the weight of the displaced fluid.

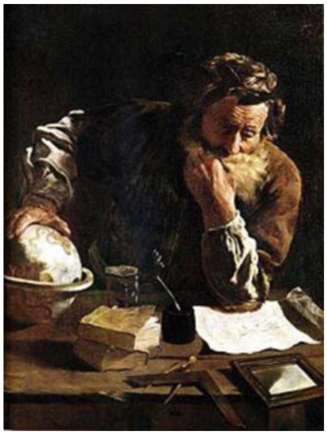


Fig. 11.14 Archimedes in thought by Fetti

[1] Syracuse is located on the island of Sicily in Southern Italy.

He is believed to have discovered the principle while sitting in his bath. He was so overwhelmed with his discovery that he rushed out onto the streets of Syracuse shouting ‘eureka’ but forgot to put on his clothes to announce the discovery.

Archimedes also made good contributions to mathematics including a good approximation to π, contributions to the positional numbering system, geometric series and to mathematical physics.

He also solved several interesting problems, for example, the calculation of the composition of cattle in the herd of the Sun god by solving a number of simultaneous Diophantine equations.

Archimedes also calculated an upper bound of the number of grains of sands in the known universe. He challenged the prevailing view that the number of grains of sand was too large to be counted, and in order to provide an upper bound, he needed to develop a naming system for large numbers.

11.18 Review Questions

1. Describe the contribution of Gene Amdahl to computing.
2. Discuss the importance of the Mythical Man Month.
3. Describe the approach of Dijkstra, Hoare and Parnas. How successful has their work been?
4. What are your views of the free software movement?
5. Describe the components of the von Neumann architecture.
6. Describe the contribution of Knuth.
7. Explain your views on Dijkstra’s position on testing.
8. Explain your views on Brooks law.

11.19 Summary

This chapter discussed a small selection of individuals who have made important contributions to the computing field. The selection included John von Neumann who did important work on early computers and who developed (along with Mauchly and Eckert) the von Neumann architecture that is the fundamental architecture underlying computers.

Gene Amdahl was the chief designer of the IBM 360 series of computers, and he later formed the Amdahl Corporation. This company became a rival to IBM in the mainframe market.

Fred Brooks was the project manager for the IBM 360 project, and he later wrote *The Mythical Man Month*. This influential book describes the challenge of delivering a large project in which software is a major constituent on time, on budget and with the right quality.

Donald Knuth is considered the father of the analysis of algorithms, and he has published the ultimate guide to program development in his massive four-volume work *The Art of Computer Programming*.

Hoare, Dijkstra and Parnas have made fundamental contributions to the software engineering field. Hoare developed axiomatic semantics and CSP. Dijkstra developed shortest graph algorithms and advocated a formal approach to software development. Parnas developed information hiding and a mathematical approach to software engineering using tabular expressions.

Dennis Ritchie developed the C programming language and codeveloped the UNIX operating system with Ken Thompson.

Chapter 12
Foundations (Boole and Babbage)

Key Topics

Boolean Algebra
Switching Circuits
Difference Engine
Analytic Engine
Lady Ada Lovelace

12.1 Introduction

This chapter considers the work of George Boole and Charles Babbage who are considered grandfathers of computing. George Boole was a nineteenth-century English mathematician who made contributions to logic, probability theory and the differential and integral calculus. His calculus of logic (Boolean logic) acts as the foundation of all modern digital computers.

Charles Babbage was a nineteenth-century scientist and inventor who did pioneering work on calculating machines. He invented the difference engine (a sophisticated calculator that could be used for the production of mathematical tables), and he also designed the analytic engine (the world's first mechanical computer). The design of the analytic engine included a processor, memory and a way to input information and output results. However, it was never built in Babbage's lifetime.

Babbage intended that the program be stored on read-only memory using punch cards, and that input and output for the analytic engine be carried out using punch cards. He intended that the machine would be able to store numbers and intermediate results in memory where they could then be processed. This machine would have employed features subsequently used in modern computers such as sequential control, branching and looping. He even intended that the machine would be capable of parallel processing where several calculations could be performed at once. It would have been the first mechanical device to be Turing complete.

Lady Ada Lovelace became familiar with Babbage's ideas on the analytic engine at a dinner party, and she was fascinated by the idea of such a machine. She wrote what is considered the first computer program and may be considered the world's first computer programmer.

12.2 George Boole

George Boole was born in Lincoln, England, in 1815. His father (a cobbler who was interested in mathematics and optical instruments) taught him mathematics and showed him how to make optical instruments. Boole inherited his father's interest in knowledge and was self-taught in mathematics and Greek. He was taught Latin by a tutor. Boole taught in various schools near Lincoln and developed his mathematical knowledge by working his way through Newton's Principia, as well as applying himself to the work of mathematicians such as Laplace and Lagrange (Fig. 12.1).

He published regular papers from his early twenties onwards, and these included contributions to probability theory, differential equations and finite differences. He developed Boolean algebra which is the foundation for modern computing, and he is considered (along with Babbage) to be one of the grandfathers of computing. His work was theoretical; he was not an engineer and never actually built a computer or calculating machine. However, Boole's symbolic logic provided the perfect mathematical model for switching theory and for the design of digital circuits. Claude Shannon recognised its applicability and applied it successfully to switching circuits.

Boole was interested in formulating a calculus of reasoning, and he published a pamphlet titled 'Mathematical Analysis of Logic' in 1847 [Boo:48]. This article developed novel ideas on a logical method, and he argued that logic should be considered as a separate branch of mathematics, rather than being considered

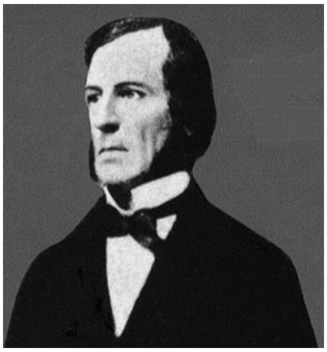


Fig. 12.1 George Boole

a part of philosophy. Boole argued that there are mathematical laws to express the operation of reasoning in the human mind, and he showed how Aristotle's syllogistic logic could be rendered as algebraic equations. He corresponded regularly on logic with the British mathematician, Augustus De Morgan[1] on logic.

Boole had no formal university qualification, and he had difficulty in obtaining a university position. However, his publications were recognised as excellent,[2] and he was awarded the position as the first professor of mathematics at the newly founded Queens College Cork,[3] Ireland, in 1849.

Boole's influential paper on a calculus of logic introduced two quantities 0 and 1. He used the quantity 1 to represent the universe of thinkable objects (i.e. the universal set), with the quantity 0 representing the absence of any objects (i.e. the empty set). He then employed symbols such as x, y, z, etc. to represent collections or classes of objects given by the meaning attached to adjectives and nouns. Next, he introduced three operators (+, $-$ and $\times$) that combined classes of objects.

For example, the expression xy (i.e. x multiplied by y or $x \times y$) combines the two classes x, y to form the new class xy (i.e. the class whose objects satisfy the two meanings represented by the classes x and y). Similarly, the expression $x + y$ combines the two classes x, y to form the new class $x + y$ (that satisfies either the meaning represented by class x or by class y). The expression $x - y$ combines the two classes x, y to form the new class $x - y$. This represents the class that satisfies the meaning represented by class x but not class y. The expression $(1 - x)$ represents objects that do not have the attribute that represents class x.

Thus, if $x =$ black and $y =$ sheep, then xy represents the class of black sheep. Similarly, $(1 - x)$ would represent the class obtained by the operation of selecting all things in the world except black things, $x\ (1 - y)$ represents the class of all things that are black but not sheep and $(1 - x)\ (1 - y)$ would give us all things that are neither sheep nor black.

He showed that these symbols obeyed a rich collection of algebraic laws and could be added, multiplied, etc. in a manner that is similar to real numbers. He showed how these symbols could be used to reduce propositions to equations, and algebraic rules could be used to solve the equations. The algebraic rules satisfied by his system included:

1.	$x + 0 = x$	(Additive identity)
2.	$x + (y + z) = (x + y) + z$	(Associativity)
3.	$x + y = y + x$	(Commutativity)

[1] De Morgan was a nineteenth-century British mathematician based at University College London. De Morgan's laws in set theory and logic state that $(A \cap B)^c = A^c \cap B^c$ and $\neg (A \vee B) = \neg A \wedge \neg B$.

[2] Boole was awarded the Royal Medal from the Royal Society of London in 1844 in recognition of his publications. The Irish Mathematician, Sir Rowan Hamilton (who invented Quaternions), was another famous recipient of this prize.

[3] Queens College Cork is now called University College Cork (UCC) and has about 18,000 students. It is located in Cork city in the south of Ireland.

4.	$x + (1 - x) = 1$	(Not operator)
5.	$x\,1 = x$	(Multiplicative identity)
6.	$x\,0 = 0$	
7.	$x + 1 = 1$	
8.	$xy = yx$	(Commutativity)
9.	$x(yz) = (xy)z$	(Associativity)
10.	$x(y + z) = xy + xz$	(Distributive)
11.	$x(y - z) = xy - xz$	(Distributive)
12.	$x^2 = x$	(Idempotent)
13.	$x^n = x$	

These operations are similar to the modern laws of set theory with the set union operation represented by '+', and the set intersection operation is represented by multiplication. The universal set is represented by '1' and the empty by '0'. The associative and distributive laws hold. Finally, the set complement operation is given by $(1 - x)$.

Boole applied the symbols to encode propositions of Aristotle's Syllogistic Logic, and he showed how the syllogisms could be reduced to equations. This allowed conclusions to be derived from premises by eliminating the middle term in the syllogism.

Boole refined his ideas on logic further in his book *An Investigation of the Laws of Thought* [Boo:58] published in 1854. This book aimed to identify the fundamental laws underlying reasoning in the human mind and to give expression to these laws in the symbolic language of a calculus. He considered the equation $x^2 = x$ to be a fundamental laws of thought. It allows the principle of contradiction to be expressed (i.e. for an entity to possess an attribute and at the same time not to possess it):

$$\begin{aligned} & x^2 = x \\ \Rightarrow\ & x - x^2 = 0 \\ \Rightarrow\ & x(1 - x) = 0 \end{aligned}$$

For example, if x represents the class of horses then $(1 - x)$ represents the class of 'not horses'. The product of two classes represents a class whose members are common to both classes. Hence, $x\ (1 - x)$ represents the class whose members are at once both horses and 'not horses', and the equation $x\ (1 - x) = 0$ expresses that fact that there is no such class whose members are both horses and 'not horses'. Another words, it is the empty set.

Boole made contributions to other areas in mathematics including differential equations and on finite differences.[4] He also contributed to the development of probability theory.

[4] Finite differences are a numerical method used in solving differential equations.

He married Mary Everest in 1855, and they lived in Lichfield Cottage in Ballintemple, Cork. She was a niece of the surveyor of India, Sir George Everest, after whom the world's highest mountain is named. They had five daughters, and his daughter Ethel Lilian was the author of the novel *The Gadfly*.[5] Boole died from fever at the early age of 49 in 1864.

Queens College Cork honoured his memory by installing a stained glass window in the *Aula Maxima* of the college. This shows Boole writing at a table with Aristotle and Plato in the background. An annual Boole Prize is awarded by the Mathematics Department at University College Cork. Des McHale has written an interesting biography of Boole [McH:85].

Boole's work on logic appeared to have no practical use. However, *Claude Shannon* became familiar with Boole's work in the 1930s, and his 1937 Masters thesis showed how Boolean algebra could optimise the design of systems of electromechanical relays which were then used in telephone routing switches. He also proved that circuits with relays could solve Boolean algebra problems.

The use of the properties of electrical switches to process logic is the basic concept that underlies all modern electronic digital computers. All digital computers today use the binary digits 0 and 1, and Boolean logical operations may be implemented by electronic AND, OR and NOT gates. More complex circuits (e.g. arithmetic) may be designed from these fundamental building blocks.

12.2.1 Modern Boolean Algebra

Boolean algebra consists of propositions that are either true or false. The proposition '$2 + 2 = 4$' is true, whereas the proposition '$2 \times 5 = 11$' is false. Variables (A, B, etc.) are used to stand for propositions, and propositions may be combined using logical connectives to form new propositions. The standard logical connectives are 'and', 'or' and 'not', and these are represented by the symbols '$\wedge$', '$\vee$' and '$\neg$', respectively. There are other logical connectives that may be used such as implication ($\Rightarrow$) and equivalence ($\Leftrightarrow$).

There are several well-known properties of Boolean algebra as described in Table 12.1.

The Boolean constant 'True' is the identity operation for conjunction. In other words, the conjunction of any operand A with the Boolean value 'True' yields the proposition A. Similarly, the Boolean constant 'False' is the identity operation for disjunction.

Truth tables define the truth values of a compound proposition from its constituent propositions. The conjunction of A and B ($A \wedge B$) is true if and only if

[5] This is a novel about the struggle of an international revolutionary. Shostakovich wrote the score for the film of the same name that appeared in 1955.

Table 12.1 Properties of Boolean algebra

Property	Example
Commutative	$A \wedge B \equiv B \wedge A$
	$A \vee B \equiv B \vee A$
Associative	$A \wedge (B \wedge C) \equiv (A \wedge B) \wedge C$
	$A \vee (B \vee C) \equiv (A \vee B) \vee C$
Identity	$A \wedge \text{True} \equiv A$
	$A \vee \text{False} \equiv A$
Distributive	$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
	$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$
De Morgan	$\neg (A \wedge B) \equiv \neg A \vee \neg B$
	$\neg (A \vee B) \equiv \neg A \wedge \neg B$
Idempotent	$A \wedge A \equiv A$
	$A \vee A \equiv A$

Table 12.2 Truth tables for conjunction and disjunction

A	B	$A \wedge B$
T	T	T
T	F	F
F	T	F
F	F	F

A	B	$A \vee B$
T	T	T
T	F	T
F	T	T
F	F	F

Table 12.3 Truth table for not operation

A	$\neg A$
T	F
F	T

both *A* and *B* are true. The disjunction of *A* and *B* ($A \vee B$) is true if either *A* or *B* is true. These are defined in Table 12.2.

There are other logical connectives that may be used [ORg:06] such as implication and equivalence. The ‘not’ operator (¬) is a unary operator such that ¬A is true if A is false and is false if A is true (Table 12.3).

Complex Boolean expressions may be formed from simple Boolean expressions using the logical connectives.

12.2.2 Switching Circuits and Boolean Algebra

Claude Shannon showed in his influential masters thesis that Boolean algebra was applicable to telephone routing switches. It could optimise the design of systems of electromechanical relays, and circuits with relays could solve Boolean algebra problems.

Modern electronic computers use millions of transistors that act as switches and can change state rapidly. The use of switches to represent binary values is the foundation of modern computing. A high voltage represents the binary value 1 with low voltage representing the binary value 0. A silicon chip may contain thousands

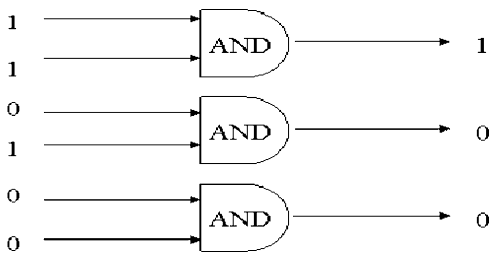


Fig. 12.2 Binary AND operation

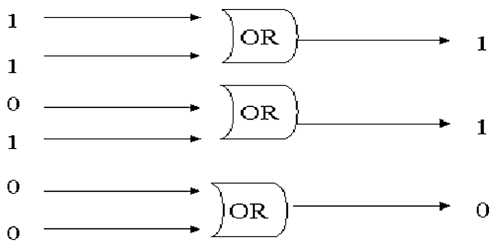


Fig. 12.3 Binary OR operation

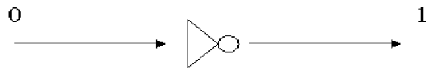


Fig. 12.4 NOT operation

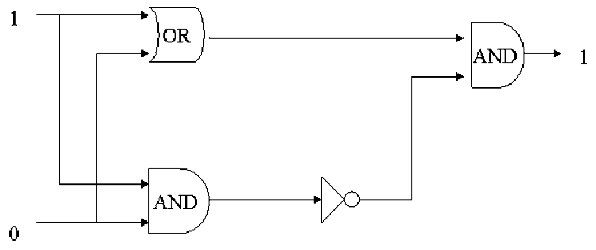


Fig. 12.5 Half adder

of tiny electronic switches arranged into logical gates. The basic logic gates are AND, OR and NOT. These gates may be combined in various ways to allow the computer to perform more complex tasks such as binary arithmetic. Each gate has binary value inputs and outputs.

The example in Fig. 12.2 below is that of an ‘AND’ gate, and this gate produces the binary value 1 as output only if both inputs are 1. Otherwise, the result will be the binary value 0.

The example in Fig. 12.3 is of an ‘OR’ gate, and it produces the binary value 1 as output if any of its inputs is 1. Otherwise, it will produce the binary value 0 as output.

Finally, a NOT gate accepts only a single input which it reverses. That is, if the input is ‘1’, the value ‘0’ is produced and vice versa. The NOT gate may be combined with the AND to yield the NAND gate (NOT AND) (Fig. 12.4).

The logic gates may be combined to form more complex circuits. The example in Fig. 12.5 is that of a half adder of 1 + 0.

The inputs to the top OR gate are 1 and 0, and this yields the result of 1. The inputs to the bottom AND gate are 1 and 0 which yields the result 0, which is then inverted through the NOT gate to yield binary 1. Finally, the last AND gate receives two 1s as input, and the binary value 1 is the result of the addition. The half adder computes the addition of two arbitrary binary digits, but it does not calculate the carry for the operation. The half adder may be extended to a full adder that provides a carry for addition.

12.3 Charles Babbage

Charles Babbage is considered (along with George Boole) to be one of the grandfathers of computing. He was born in Devonshire, England, in 1791 and was the son of a banker. He studied mathematics at Cambridge University in England and was appointed to the Lucasian Chair in Mathematics at Cambridge in 1828. He made contributions to several areas including mathematics, statistics, astronomy, philosophy, railways and lighthouses. He founded the British Statistical Society and the Royal Astronomical Society (Fig. 12.6).

Babbage was interested in accurate mathematical tables as these are essential for navigation and scientific work. However, there was a high error rate in the existing tables due to human error introduced during calculation. His approach to solving this problem was to investigate a mechanical method to perform the calculations as this would eliminate errors introduced by human calculation. Pascal and Leibniz did early work on calculating machines.

He designed the difference engine (no. 1) in 1821 for the production of mathematical tables. A difference engine is essentially a mechanical calculator

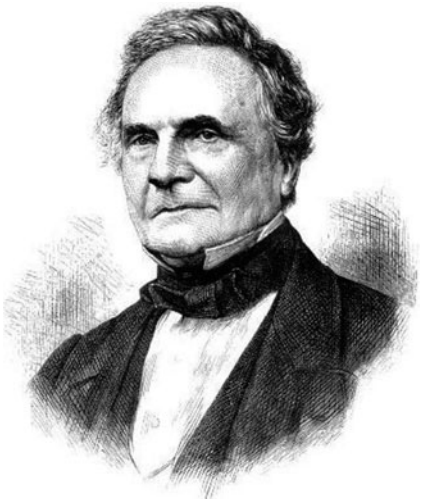


Fig. 12.6 Charles Babbage

(analogous to modern electronic calculators), and it was designed to compute polynomial functions. It could also compute logarithmic and trigonometric functions such as sine or cosine as these may be approximated by polynomials.[6]

The accurate approximation of trigonometric, exponential and logarithmic functions by polynomials depends on the degree of the polynomials, the number of decimal digits that it is being approximated to and the error function. A higher degree polynomial is able to approximate the function more accurately.

Babbage produced prototypes for parts of the difference engine but he never actually completed it. Babbage's Engine was intended to operate on sixth-order polynomials of 20 digits. A polynomial of degree 6 is of the form $p(x) = ax^6 + bx^5 + cx^4 + dx^3 + ex^2 + fx + g$.

He also designed the analytic engine (the world's first mechanical computer). The design of the analytic engine included a processor, memory and a way to input information and output results.

12.3.1 Difference Engine

The first working difference engine was built in 1853 by the Swedish engineers George and Edward Scheutz. They based their plans on Babbage's design and received funding from the Swedish government. The Scheutz machine could compute polynomials of degree 4 on 15-digit numbers. A copy of the third Scheutz Difference Engine is on display in the Science Museum in London.

This machine was the first to compute and print mathematical tables mechanically. The working engine produced by the Scheutz brothers was accurate, and it was used by scientists and engineers in their calculations. It showed the potential of mechanical machines as a tool for scientists and engineers (Fig. 12.7).

The difference engine consists of N columns (numbered 1–N). Each column is able to store one decimal number, and the numbers are represented by wheels. The difference engine (no. 1) has seven columns with each column containing 20 wheels. Each wheel consists of ten teeth, and these represent the decimal digits. Each column could therefore represent a decimal number with up to 20 digits. The seven columns allowed the representation of polynomials of degree six.

The only operation that the difference engine can perform is the addition of the value of column $n + 1$ to column n, and this results in a new value for column n.

[6] The power series expansion of the sine function is given by $\sin(x) = x - x^3/3! + x^5/5! - x^7/7! + \ldots$. The power series expansion for the cosine function is given by $\cos(x) = 1 - x^2/2! + x^4/4! - x^6/6! + \ldots$. Functions may be approximated by interpolation and the approximation of a function by a polynomial of degree n requires $n + 1$ points on the curve for the interpolation. That is, the curve formed by the polynomial of degree n that passes through the $n + 1$ points on the function to be approximated is an approximation to the function.

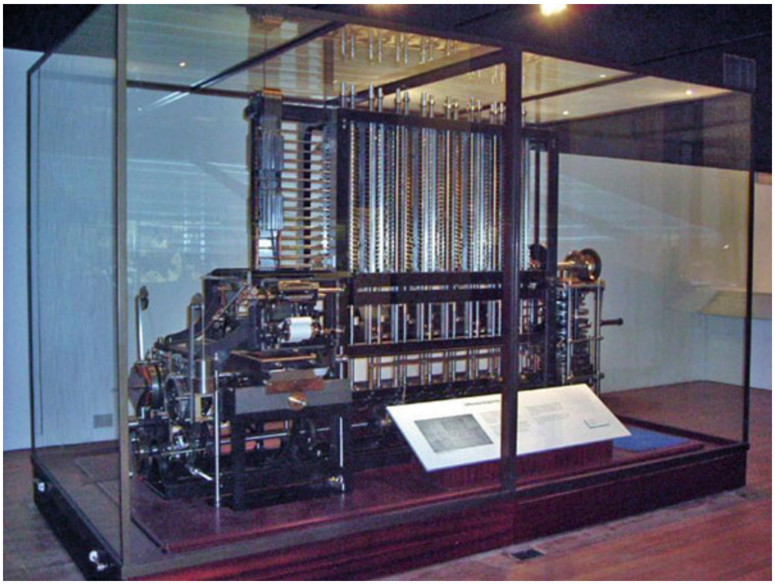


Fig. 12.7 Difference engine no. 2 (Photo public domain)

Column N can only store a constant and column 1 displays the value of the calculation for the current iteration. The machine is programmed prior to execution by setting initial values to each of the columns. Column 1 is set to the value of the polynomial at the start of computation; column 2 is set to a value derived from the first and higher derivatives of the polynomial for the same value of x. Each column n from 3 to N is set to a value derived from the $(n - 1)$ first and higher order derivatives of the polynomial.

The Scheutz's difference engine was comprised of shafts, shelves and wheels. The scientist could set numbers on the wheels[7] and turn a crank to start the computation. Then, by reading down each shaft, he could find the result of the calculation. The difference engine was able to print out the answers to the computation. The decimal numbering system was employed, and there was also a carry mechanism.

The machine is unable to perform multiplication or division directly. Once the initial value of the polynomial and its derivatives are calculated for some value of x, the difference engine can calculate any number of nearby values using the numerical method of finite differences. This method replaces computational intensive tasks involving multiplication or division by an equivalent computation which just involves addition or subtraction.

[7] Each wheel represented a decimal digit (each wheel consisted of ten teeth).

Table 12.4 Finite Differences (Calculation)

x	$p(x)$	Diff. 1	Diff. 2
1	6		
2	15	9	
3	28	13	4
4	45	17	4
5	66	21	4

Table 12.5 Finite Differences (In Use)

x	$f(x)$	Diff. 1	Diff. 2
1	6	9	4
2	15	13	4
3	28	17	4
4	45	21	4
5	66	25	4

12.3.2 Finite Differences

A finite difference is a mathematical expression of the form $f(x + h) - f(x)$. If a finite difference is divided by h, then the resulting expression is similar to a differential quotient, except that it is discrete.

Finite differences may be applied to approximate derivatives, and they are often used to find numerical solution to differential equations. They provide a very useful way to calculate the value of a polynomial used to approximate a given function.

The method of finite differences is used in the production of tables for polynomials using the difference engine. Consider the quadratic polynomial $p(x) = 2x^2 + 3x + 1$ and consider Table 12.4.

The first difference is computed by subtracting two adjacent entries in the column of $p(x)$. For example, $15 - 6 = 9$, $28 - 15 = 13$ and so on. Similarly, the second difference is given by subtracting two adjacent entries in the Difference 1 column; for example, $13 - 9 = 4$, $17 - 13 = 4$ and so on. The entries in the second difference column are the constant 4. In fact, for any n-degree polynomial the entries in the n-difference column are always a constant.

The difference engine performs the computation of the table in a similar manner, although the approach is essentially the reverse of the above. Once the first row of the table has been determined, the rest of the table may be computed using just additions of pairs of cells in the table (Table 12.5).

The first row is given by the cells 6, 9 and 4 which allows the rest of the table to be determined. The numbers in the table below have been derived by simple calculations from the first row. The procedure for calculation of the table is as follows:

1. The Difference 2 column is the constant 4.
2. The calculation of the cell in row i for the Difference 1 column is given by Diff. 1 $(i - 1)$ + Diff. 2 $(i - 1)$.
3. The calculation of the cell in row i for the function column is given by $f(i - 1)$ + Diff. 1 $(i - 1)$.

In other words, to calculate the value of a particular cell, all that is required is to add the value in the cell immediately above it to the value of the cell immediately to its right. Therefore, in the second row, the calculations 6 + 9 yield 15, and 9 + 4 yields 13, and, since the last column is always a constant, it is just repeated. Therefore, the second row is 15, 13 and 4, and $f(2)$ is 15. Similarly, the third row yields $15 + 13 = 28$, $13 + 4 = 17$ and so on. This is the underlying procedure of the difference engine.

The initial problem is to compute the first row which allows the other rows to be computed. Its computation is more difficult for complex polynomials. The other problem is to find a suitable polynomial to represent the function, and this may be done by interpolation. However, once these problems are solved, the engine produces pages and columns full of data.

Babbage received £17 K of taxpayer funds to build the difference engine, but for various reasons he only produced prototypes of the intended machine. Therefore, the British government was reluctant to continue funding, and the project was cancelled in 1842.

The prototypes built by Babbage and his engineer Joseph Clement were limited to the computation of quadratic polynomials of six-digit numbers. The difference engine envisaged by Babbage was intended to operate on sixth-order polynomials of 20 digits.

He designed an improved difference engine (no. 2) in 1849. It could operate on seventh-order differences (i.e. polynomials of order 7) and 31-digit numbers. The machine consisted of eight columns with each column consisting of 31 wheels. However, it was over 150 years later before it was built (in 1991) to mark the 200th anniversary of his birth. The science museum also built the printer that Babbage designed, and both the machine and the printer worked correctly according to Babbage's design (after a little debugging).

12.3.3 Analytic Engine

The difference engine was designed to produce mathematical tables and required human intervention to perform the calculation. Babbage recognised its limitations and proposed a revolutionary solution. His plan was to construct a new machine that would be capable of executing all tasks that may be expressed in algebraic notation. The analytic engine envisioned by Babbage consisted of two parts (Table 12.6).

Babbage intended that the operation of the analytic engine would be analogous to the operation of the Jacquard loom.[8] The latter is capable of weaving (i.e. executing

[8] The Jacquard loom was invented by Joseph Jacquard in 1801. It is a mechanical loom which used the holes in punch cards to control the weaving of patterns in a fabric. The use of punch cards allowed complex designs to be woven from the pattern defined on the punch card. Each punch card corresponds to one row of the design, and the cards were appropriately ordered. It was very easy to change the pattern of the fabric being weaved on the loom, as this simply involved changing cards.

Table 12.6 Analytic engine

Part	Function
Store	This contains the variables to be operated upon as well as all those quantities which have arisen from the result of intermediate operations
Mill	The mill is essentially the processor of the machine into which the quantities about to be operated upon are brought

on the loom) a design pattern that has been prepared by a team of skilled artists. The design pattern is represented by punching holes on a set of cards, and each card represents a row in the design. The cards are then ordered and placed in the Jacquard loom, and the exact pattern is produced by the loom.

The Jacquard loom was the first machine to use punch cards to control a sequence of operations. It did not perform computation, but it was able to change the pattern of what was being weaved by changing cards. This gave Babbage the idea to use punch cards to store programs to perform the analysis and computation in the analytic engine.

The use of the punch cards in the analytic engine allowed the formulae to be manipulated in a manner dictated by the programmer. The cards commanded the analytic engine to perform various operations and to return a result. Babbage distinguished between two types of punch cards:

– Operation cards
– Variable cards

Operation cards are used to define the operations to be performed, whereas the variable cards define the variables or data that the operations are performed upon. This planned use of punch cards to store programs in the analytic engine is similar to the idea of a stored computer program in von Neumann architecture. However, Babbage's idea of using punch cards to represent machine instructions and data was over 100 years before digital computers. Babbage's analytic engine is therefore an important step in the history of computing.

The analytic engine was designed in 1834 as the world's first mechanical computer [Bab:42]. It included a processor, memory and a way to input information and output results. However, the machine was never built as Babbage was unable to receive funding from the British Government.

Babbage intended that the program be stored on read-only memory using punch cards, and that the input and output would be carried out using punch cards. He intended that the machine would be able to store numbers and intermediate results in memory that could then be processed. There would be several punch card readers in the machine for programs and data. He envisioned that the machine would be able to perform conditional jumps as well as parallel processing where several calculations could be performed at once (Fig. 12.8).

Lady Ada was introduced into Babbage's ideas on the analytic engine at a dinner party. She was a mathematician and the daughter of Lord Byron. She was fascinated by the idea of the analytic engine and communicated regularly with Babbage with ideas on its applications. She predicted that such a machine could be used

Fig. 12.8 Lady Ada Lovelace

to compose music, produce graphics as well as solving mathematical and scientific problems. She suggested to Babbage that a plan be written for how the engine would calculate Bernoulli numbers. This plan is considered to be the first computer program, and Lady Ada Lovelace is therefore considered to be the first computer programmer. The Ada programming language is named in her honour.

12.4 Review Questions

1. Describe how Boole's symbolic logic is used in digital circuit design.
2. Explain how finite differences are used to produce tables in the difference engine.
3. Describe the analytic engine and compare/contrast to von Neumann architecture.

12.5 Summary

This chapter considered the contribution of George Boole and Charles Babbage who are considered to be the grandfathers of the computing field.

Boole was a nineteenth-century English mathematician who made contributions to logic, probability theory and calculus. His calculus of logic (Boolean logic) introduced two quantities (i.e. 0 and 1) and is the foundation of all modern digital computers. It was Claude Shannon who recognised the applicability of Boolean logic to circuit design in his influential 1937 masters thesis. He showed that Boole's symbolic logic provided the perfect mathematical model for switching theory and for the design of digital circuits.

Babbage was a nineteenth-century scientist and inventor who did pioneering work on calculating machines. He invented the difference engine (a sophisticated calculator that could be used for the production of mathematical tables), and he designed the analytic engine (the world's first mechanical computer). The design of the analytic engine included a processor, memory and a way to input information and output results.

Chapter 13
Claude Shannon

Key Topics

Boolean Algebra and Switching Circuits
Information Theory
Cryptography

13.1 Introduction

Claude Shannon was an American mathematician and engineer who made fundamental contributions to computing. He was born in Michigan in 1916, and his primary degree was in mathematics and electrical engineering at the University of Michigan in 1936. He was awarded a PhD in mathematics from the Massachusetts Institute of Technology (MIT) in 1940.

He was the first person[1] to see the applicability of Boolean algebra to simplify the design of circuits and telephone routing switches. He showed that Boole's symbolic logic developed in the nineteenth century provided the perfect mathematical model for switching theory and for the subsequent design of digital circuits and computers.

His influential master's thesis is a key milestone in computing, and it shows how to lay out circuits according to Boolean principles. It provides the theoretical foundation of switching circuits, and his insight of using the properties of electrical switches to do Boolean logic is the basic concept that underlies all electronic digital computers.

[1] Victor Shestakov at Moscow State University also proposed a theory of electric switches based on Boolean algebra around the same time as Shannon. However, his results were published in Russian in 1941, whereas Shannon's were published in 1937.

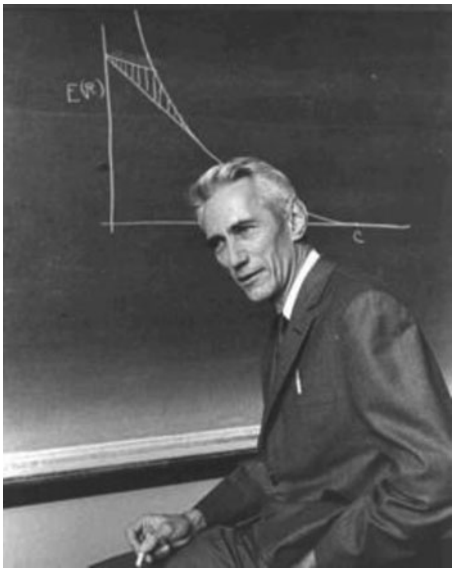


Fig. 13.1 Claude Shannon

Shannon realised that you could combine switches in circuits in such a manner as to carry out symbolic logic operations. The implications of this were enormous, as it allowed binary arithmetic and more complex mathematical operations to be performed by relay circuits. He showed the design of a circuit which could add binary numbers; he moved on to realising circuits which could make comparisons and thus is capable of performing a conditional statement. This was the birth of digital logic (Fig. 13.1).

He moved to the mathematics department at Bell Labs in the 1940s and commenced work that would lead to the foundation of modern information theory. This is concerned with defining a quantitative measure of information and using this to solve related problems. It includes the well-known 'channel capacity theorem' and is also concerned with the problem of the reliable transfer of messages from a source point to a destination point. The messages may be in any communications medium, for example, television, radio, telephone and computers. The fundamental problem is to reproduce at a destination point either exactly or approximately the message that has been sent from a source point. The problem is that information may be distorted by noise, leading to differences between the received message and that which was originally sent.

Shannon provided a mathematical definition and framework for information theory in '*A Mathematical Theory of Communication*' [Sha:48]. He proposed the idea of converting data (e.g. pictures, sounds or text) to binary digits, that is, binary bits of information. The information is then transmitted over the communication medium. Errors or noise may be introduced during the transmission, and the objective is to reduce and correct them. The received binary information is then converted back to the appropriate medium.

Shannon is considered the father of digital communication, and his theory was an immediate success with communications engineers with wide reaching impacts.

He also contributed to the field of cryptography in '*Communication Theory of Secrecy Systems*' [Sha:49].

He also made contributions to genetics and invented a chess playing computer program in 1948. He was a talented gadgeteer who built some early robot automata, game playing devices and problem-solving machines. He was able to juggle while riding a unicycle, and he designed machines to juggle and to ride unicycle like machines. He received many honours and awards and died in 2001.

13.2 Boolean Algebra and Switching Circuits

Vannevar Bush[2] was Shannon's supervisor at MIT, and Shannon's initial work was to improve Bush's mechanical computing device known as the differential analyser.

The differential analyzer was an analog computer made of gears, pulleys and rods. Its function was to evaluate and solve first-order differential equations. The machine was developed by Vannevar Bush and others in 1931 and funded by the Rockefeller Foundation. It weighed over a 100 tons, had 2,000 vacuum tubes, several thousand relays and automated punch-tape access units.

It had a complicated control circuit that was composed of 100 switches that could be automatically opened and closed by an electromagnet. Shannon's insight was the realisation that an electronic circuit is similar to Boolean algebra. He showed how Boolean algebra could be employed to optimise the design of systems of electromechanical relays used in Bush's analog computer, and that circuits with relays could solve Boolean algebra problems.

He showed in his master's thesis '*A Symbolic Analysis of Relay and Switching Circuits*' [Sha:37] that the binary digits (i.e. 0 and 1) can be represented by electrical switches. The convention employed in Shannon's thesis is the opposite to the convention used today. In his thesis, the digit 1 is represented by a switch that is turned off and the symbol 0 by a switch that is turned on. The *convention today is that the digit* 1 *is represented by a switch that is turned on, whereas the digit* 0 *is represented by a switch that is turned off*. We shall follow the convention in Shannon's thesis in this chapter for historical reasons.

He used Boolean algebra to show that complex operations could be performed automatically on these electrical circuits. The implications of true and false being denoted by the binary digits 1 and 0 were enormous since it allowed binary arithmetic and more complex mathematical operations to be performed by relay circuits. This provided electronics engineers with the mathematical tool they needed to design digital electronic circuits, and these techniques are the foundation of digital electronic design.

The design of circuits and telephone routing switches could be simplified with Boolean algebra. Shannon showed how to lay out circuitry according to Boolean

[2] Vannevar Bush was discussed in an earlier chapter.

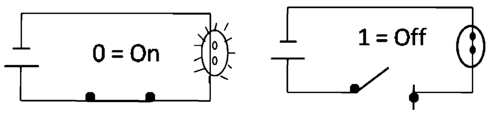


Fig. 13.2 Switching circuit representing Boolean logic

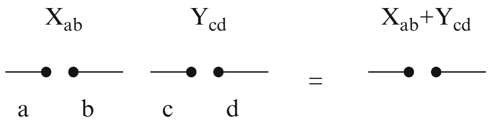


Fig. 13.3 Serial circuits

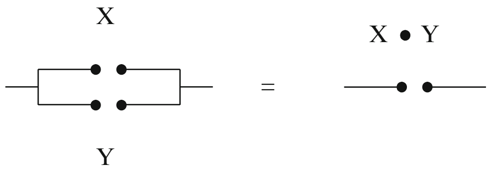


Fig. 13.4 Parallel circuits

principles, and his influential master's thesis became the foundation for the practical design of digital circuits. These circuits are fundamental to the operation of modern computers and telecommunication systems, and Shannon's master's thesis is considered a key milestone in the development of modern computers.

His insight of using the properties of electrical switches to do Boolean logic is the basic concept that underlies all electronic digital computers (Fig. 13.2).

A circuit may be represented by a set of equations with the terms in the equations representing the various switches and relays in the circuit. He developed a calculus for manipulating the equations, and this calculus is similar to Boole's propositional logic. The design of a circuit consists of algebraic equations, and the equations may be manipulated to yield the simplest circuit. The circuit may then be immediately drawn.

A switching circuit X_{ab} between two terminals *a* and *b* is either open (with $X_{ab} = 1$) or closed (with $X_{ab} = 0$). The symbol 0 is employed to denote a closed circuit, and the symbol 1 is used to denote an open circuit.

The expression $X_{ab} + Y_{cd}$ (usually written as X + Y) denotes the circuit formed when the circuit X_{ab} is joined serially to the circuit Y_{cd}. Similarly, the expression $X_{ab}\bullet Y_{cd}$ (written as XY) denotes the circuit formed when the circuit X_{ab} is placed in parallel to the circuit Y_{cd}. The operators + and • are, of course, different from the standard addition and multiplication operators (Figs. 13.3 and 13.4).

13.2.1 Properties of Circuits

It is evident from the above definitions that the laws found in Table 13.1 are true.

The circuit is either open or closed at a given moment of time, that is, X = 0 or X = 1. The algebraic properties found in Table 13.2 hold:

The negation of a circuit X is denoted by X′. X′ is open when X is closed and vice versa. Its properties are shown in Table 13.3.

Table 13.1 Properties of circuits

Property name	Property	Interpretation
Idempotent property	$0 \bullet 0 = 0$	A closed circuit in parallel with a closed circuit is a closed circuit
	$1 + 1 = 1$	An open circuit in series with an open circuit is an open circuit
Additive identity (0)	$1 + 0 = 0 + 1 = 1$	An open circuit in series with a closed circuit (in either order) is an open circuit
Multiplicative identity (1)	$1 \bullet 0 = 0 \bullet 1 = 0$	A closed circuit in parallel with an open circuit (in either order) is a closed circuit
Additive identity (0)	$0 + 0 = 0$	A closed circuit in series with a closed circuit is a closed circuit
Multiplicative identity (1)	$1 \bullet 1 = 1$	An open circuit in parallel with an open circuit is an open circuit

Table 13.2 Properties of circuits (ctd.)

Property name	Property
Commutative property	$x + y = y + x$
	$x \bullet y = y \bullet x$
Associative property	$x + (y + z) = (x + y) + z$
	$x \bullet (y \bullet z) = (x \bullet y) \bullet z$
Distributive property	$x \bullet (y + z) = (x \bullet y) + (x \bullet y)$
	$x + (y \bullet z) = (x + y) \bullet (x + z)$
Identity property	$x + 0 = x = 0 + x$
	$1 \bullet x = x \bullet 1 = x$
	$1 + x = x + 1 = 1$
	$0 \bullet x = x \bullet 0 = 0$

Table 13.3 Properties of circuits (ctd.)

Property name	Property
Negation	$X + X' = 1$
	$X \bullet X' = 0$
De Morgan's law	$(X')' = X$
	$(X + Y)' = X' \bullet Y'$
	$(X \bullet Y)' = X' + Y'$

Functions of variables may also be defined in the calculus in terms of the '+', '•' and negation operations. For example, the function $f(X,Y,Z) = XY' + X'Z' + XZ'$ is an example of a function in the calculus, and it describes a circuit.

13.2.2 Digital Circuits and Boolean Logic

The calculus for circuits is analogous to Boole's symbolic logic. The symbol X denotes a circuit in digital circuits and a proposition in Boolean logic;

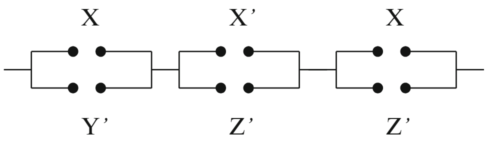


Fig. 13.5 Sample circuit

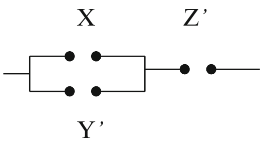


Fig. 13.6 Simplified circuit

0 denotes a closed circuit and the proposition false; 1 denotes an open circuit and the proposition true. The expression X + Y denotes serial connection of two circuits and denotes disjunction in Boolean logic; the expression XY denotes the parallel connection of two circuits and denotes logical conjunction in Boolean algebra; the expression X' denotes the complement of the circuit and the complement in Boolean algebra. De Morgan's laws hold for circuits and propositional logic.

Any expression formed with the additive, multiplicative and negation operators forms a circuit containing serial and parallel connections only. To find the simplest circuit with the least number of contacts, all that is required is to manipulate the mathematical expression into the form in which the fewest variables appear. For example, the circuit represented by $f(X,Y,Z) = XY' + X'Z' + XZ'$ is given by Fig. 13.5:

However, this circuit may be simplified by noting that (Fig. 13.6):

$$\begin{aligned} f(X,Y,Z) &= XY' + X'Z' + XZ' \\ &= XY' + (X' + X)Z' \\ &= XY' + 1Z' \\ &= XY' + Z'. \end{aligned}$$

13.2.3 Implementation of Boolean Logic

Digital circuits may be employed to implement Boolean algebra with the Boolean value 0 represented by a closed circuit and the Boolean value 1 represented by an open circuit. The logical conjunction and disjunction operations may be represented by combining circuits in parallel and series.

Complex Boolean value functions can be constructed by combining these digital circuits. Next, we discuss Shannon's contribution to information theory.

13.3 Information Theory

Early work on information theory was done by Nyquist and other engineers at Bell Labs in the 1920s. Nyquist investigated the speed of transmission of information over a telegraph wire [Nyq:24] and proposed a logarithmic rule ($W = k \log m$)[3] that set an upper limit on the amount of information that may be transmitted. This rule is a special case of Shannon's later logarithmic law.

There were several communication systems in use prior to Shannon's 1948 paper. These included the telegraph machine which dated from the 1830s, the telephone dating from the 1870s, the AM radio from the early 1900s and early television from the 1930s. These were all designed for different purposes and used various media. Each of these was a separate field with its own unique problems, tools and methodologies.

Shannon is recognised as the father of information theory due to his classic 1948 paper [Sha:48] which provided a unified theory for communication and a mathematical foundation for the field. The key problem in the field is the reliable transmission of a message from a source point over a communications channel to a destination point.[4] There may be noise in the channel that distorts the message, and the engineer wishes to ensure that the message received is that which has been sent. Information theory provides answers as to how rapidly or reliably a message may be sent from the source point to the destination point. The meanings of the messages are ignored as they are irrelevant from an engineering viewpoint. Shannon identified five key parts of an information system, namely (Fig. 13.7):

- Information source
- Transmitter
- Channel
- Receiver
- Destination

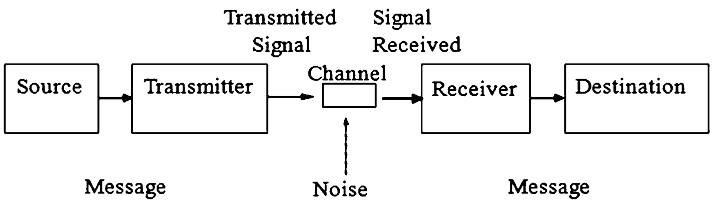


Fig. 13.7 Information theory

[3] W stands for the speed of transmission of information; m is the number of voltage levels to choose from at each step; and k is a constant.

[4] The system designer may also place a device called an encoder between the source and the channel and a device called a decoder between the output of the channel and the destination.

He derived formulae for the information rate of a source and for the capacity of a channel including noiseless and noisy cases. These were measured in bits per second, and he showed that for any information rate R less than the channel capacity C,[5] it is possible (by suitable encoding) to send information at rate R, with an error rate less than any preassigned positive ε, over that channel.

Shannon's theory of information is based on probability theory and statistics. One important concept is that of *entropy*[6] which measures the level of uncertainty in predicting the value of a random variable. For example, the toss of a fair coin has maximum entropy as there is no way to predict what will come next. A single toss of a fair coin has entropy of one bit.

The concept of entropy is used by Shannon as a measure of the average information content missing when the value of the random variable is not known. English text has fairly low entropy as it is reasonably predictable because the distribution of letters is far from uniform.

Shannon proposed two important theorems that establish the fundamental limits on communication. The first theorem deals with communication over a noiseless channel, and the second theorem deals with communication in a noisy environment.

The first theorem (Shannon's source coding theorem) essentially states that the transmission speed of information is based on its entropy or randomness. It is possible to code the information (based on the statistical characteristics of the information source) and to transmit it at the maximum rate that the channel allows. Shannon's proof showed that an encoding scheme exists, but it did not show how to construct one. This result was revolutionary as communication engineers at the time thought that the maximum transmission speed across a channel was related to other factors and not on the concept of information.

Shannon's *noisy-channel coding theorem* states that reliable communication is possible over noisy channels provided that the rate of communication is below a certain threshold called the 'channel capacity'. This result was revolutionary as it showed that a transmission speed arbitrarily close to the channel capacity can be achieved with an arbitrarily low error. The assumption at the time was that the error rate could only be reduced by reducing the noise level in the channel. Shannon showed that this assumption was not quite true, and he showed that a transmission speed arbitrarily close to the channel capacity can be achieved by using appropriate encoding and decoding systems.

Shannon's theory also showed how to design more efficient communication and storage systems.

[5] The channel capacity C is the limiting information rate (i.e. the least upper bound) that can be achieved with an arbitrarily small error probability. It is measured in bits per second.

[6] The concept of entropy is borrowed from the field of thermodynamics.

13.4 Cryptography

Shannon is considered the father of modern cryptography with his influential 1949 paper 'Communication Theory of Secrecy Systems' [Sha:49]. He established a theoretical basis for cryptography and for cryptanalysis and defined the basic mathematical structures that underlie secrecy systems.

A secrecy system is defined to be a transformation from the space of all messages to the space of all cryptograms. Each possible transformation corresponds to encryption with a particular key, and the transformations are reversible. The inverse transformation allows the original message to be obtained from the cryptogram provided that the key is known. The basic operation of a secrecy system is described in Fig. 13.8.

The first step is to select the key and to send it securely to the intended recipient. The choice of key determines the particular transformation to be used for the encryption of the message. The transformation converts the message into a cryptogram (i.e. the encrypted text). The cryptogram is then transmitted over a channel that is not necessarily secure to the receiver, and the recipient uses the key to apply the inverse transformation to decipher the cryptogram into the original message.

The enciphering of a message is a functional operation. Suppose M is a message, K is the key and E is the encrypted message, then:

$$E = f(M, K)$$

This is often written as a function of one variable $E = T_i M$ where the index i corresponds to the particular key being used. It is assumed that there are a finite number of keys $K_1, \ldots K_m$ and a corresponding set of transformations $T_1, T_2, \ldots, T_m$. Each key has a probability p_i of being chosen as the key. The encryption of a message M with key K_i is therefore given by:

$$E = T_i M$$

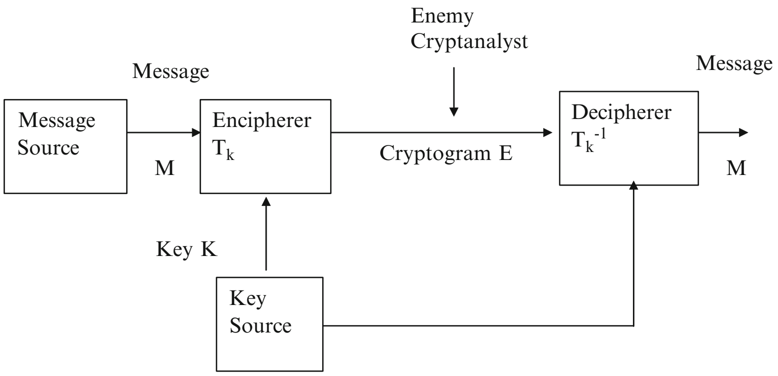


Fig. 13.8 Cryptography

It is then possible to retrieve the original message from the received encrypted message by:

$$M = T_i^{-1}E.$$

The channel may be intercepted by an enemy who will attempt to break the encrypted text. The enemy will examine the cryptogram and attempt to guess the key and the message from the cryptogram.

Shannon also showed that Vernam's cipher (also known as the *onetime pad*) is an unbreakable cipher. He showed that perfect secrecy of the cipher requires a secret random key with length greater than or equal to the number of information bits to be encrypted in the message. The secret key must be used once only, and the key needs to be sent securely to the receiver. This cipher was invented by Gilbert Vernam at Bell Labs.

13.5 Review Questions

1. Explain how Boolean algebra is applied to switching circuits.
2. Describe information theory and explain the importance of Shannon's fundamental theorems on communication.
3. Explain encryption and decryption in cryptology.
4. Describe the Vernam cipher.

13.6 Summary

Claude Shannon was an American mathematician and engineer who made fundamental contributions to computing. He was the first person to see the applicability of Boolean algebra to simplify the design of circuits and telephone routing switches. He showed how to lay out circuits according to Boolean principles, and his insight of using the properties of electrical switches to do Boolean logic is the basic concept that underlies all electronic digital computers.

He laid the foundation of modern information theory which is concerned with the problem of reliable transfer of messages from a source point to a destination point. This includes the transfer of messages over any communications medium, for example, television, radio, telephone and computers. His influential 1948 paper provided a mathematical foundation and framework for information theory that remains the standard today. He proposed two key theorems that establish the fundamental limits on communication in a noiseless and in a noisy channel.

He also made important contributions to the emerging field of cryptography, and his 1949 paper provided a theoretical foundation for cryptography and cryptanalysis.

Chapter 14
Alan Turing

Key Topics

Turing Machine
Bletchley Park
Turing Test

14.1 Introduction

Alan Turing was a British mathematician and computer scientist who made fundamental contributions to mathematics and computer science. He made important contributions to computability with his theoretical Turing machine, cryptology and breaking the German Enigma naval codes at Bletchley Park code-breaking centre during the Second World War; he contributed to the development of software for the Manchester Mark 1 at Manchester University; and he contributed to the emerging field of artificial intelligence.

He was born in London 1912, and his father worked as a magistrate and tax collector in the Indian Civil Service. He attended the Sherborne[1] which was a famous public school in England. Turing's interests were in science, mathematics and chess, and he applied himself to solving advanced problems in science and mathematics.

He excelled at long-distance running at the Sherborne, and in later life, he completed the marathon in 2 h and 46 min. This time would have made him a candidate for the 1948 Olympic Games that were held in Wembley stadium

[1] The Sherborne is located in Dorset, England. Its origins go back to the eighth century when the school was linked with the Benedictine Abbey in the town.

Fig. 14.1 Alan Turing

in London. However, he was injured shortly before the games, and this prevented him from future competition (Fig. 14.1).

Turing attended King's College, Cambridge, from 1931 to 1934 and graduated with a distinguished degree. He was elected a fellow of the college in 1935, and the computer room at King's College is named after him. He published a key paper on computability in 1936. This paper introduced a theoretical machine known as the Turing machine, and this mathematical machine is capable of performing any conceivable mathematical problem that has an algorithm.

He proved that anything that is computable is computable by this theoretical machine. The *Church-Turing thesis* states that anything that is computable may be computed by a Turing machine. There are other equivalent definitions of computability such as Church's lambda calculus.

He contributed to the code-breaking work carried out at Bletchley Park during the Second World War. The team at Bletchley succeeded in breaking the German Enigma codes.

He did important work in artificial intelligence and devised the famous 'Turing test' as a test of machine intelligence. He also did work at Manchester University prior to his premature death in the early 1950s.

14.2 Turing Machines

The theoretical Turing machine was introduced by Turing in 1936. It consists of a head and a potentially infinite tape that is divided into frames. Each frame may be either blank or printed with a symbol from a finite alphabet of symbols. The input tape may initially be blank or have a finite number of frames containing symbols. At any step, the head can read the contents of a frame; the head may erase a symbol on the tape, leave it unchanged or replace it with another symbol. It may then move

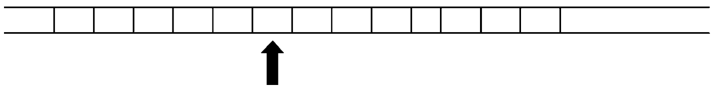


Fig. 14.2 Potentially infinite tape

one position to the right, one position to the left or not at all. If the frame is blank, the head can either leave the frame blank or print one of the symbols.

Turing believed that every calculation could be done by a human with finite equipment and with an unlimited supply of paper to write on. The unlimited supply of paper is formalised in the Turing machine by a paper tape marked off in squares. The finite number of configurations of the Turing machine was intended to represent the finite states of mind of a human calculator (Fig. 14.2).

Definition 14.1 (Turing Machine). A Turing machine M $= (Q, \Gamma, b, \Sigma, \delta, q_0, F)$ is a 7-tuple as defined formally in [HoU:79] as:

- Q is a finite set of *states*.
- Γ is a finite set of the *tape alphabet/symbols*.
- $b \in \Gamma$ is the *blank symbol* (This is the only symbol that is allowed to occur infinitely often on the tape during each step of the computation).
- Σ is the set of input symbols and is a subset of Γ (i.e. $\Gamma = \Sigma \cap \{b\}$).
- δ: $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$[2] is the transition function. This is a partial function where L is left shift, R is right shift.
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is the set of final or accepting states.

The Turing machine is a simple machine that is equivalent to an actual physical computer in the sense that it can compute exactly the same set of functions. It is much easier to analyse and prove things about than a real computer, but it is not suitable for programming and therefore does not provide a good basis for studying programming and programming languages.

It is essentially a finite-state machine (FSM) with an unbounded tape. The tape is potentially infinite and unbounded, whereas real computers have a large but finite store. The machine may read from and write to the tape. The FSM is essentially the control unit of the machine, and the tape is essentially the store. However, the store in a real computer may be extended with backing tapes and disks and, in a sense, may be regarded as unbounded. However, the maximum amount of tape that may be read or written within n steps is n.

A Turing machine has an associated set of rules that defines its behaviour. Its actions are defined by the transition function. It may be programmed to solve any problem for which there is an algorithm. However, if the problem is unsolvable, then the machine will either stop or compute forever. The solvability of a problem may not be determined beforehand. There is, of course, some answer (i.e. either the machine halts or it computes forever).

[2] We may also allow no movement of the tape head to be represented by adding the symbol 'N' to the set.

Fig. 14.3 David Hilbert

Turing showed that there was no solution to the *Entscheidungsproblem* (i.e. the decision problem) posed by the German mathematician David Hilbert. This problem asserted that the truth or falsity of a mathematical problem can always be determined by a mechanical procedure. Hilbert was asserting that first-order logic is decidable; that is, there is a decision procedure to determine if an arbitrary sentence in the formula is valid. Turing was sceptical on the decidability of first-order logic, and he believed that whereas some problems may be solved, some others cannot be solved. The Turing machine played a key role in refuting Hilbert's claim of the decidability of first-order logic, as whatever is computable may be computed on a Turing machine (Fig. 14.3).

The question as to whether a given Turing machine halts or not can be formulated as a first-order statement. If a general decision procedure exists for first-order logic, then the statement of whether a given Turing machine halts or not is within the scope of the decision algorithm. However, Turing had already proved that the halting problem for Turing machines is not computable; that is, it is not possible algorithmically to decide whether or not any given Turing machine will halt or not. Therefore, since there is no general algorithm that can decide whether any given Turing machine halts, there is no general decision procedure for first-order logic. The only way to determine whether a statement is true or false is to try to solve it; however, if one tries but does not succeed, this does not prove that an answer does not exist.

Turing also introduced the concept of a universal Turing machine, and this machine is able to simulate any other Turing machine. His results on computability were proved independently of Church's lambda calculus equivalent results.[3] Turing studied at Princeton University in 1937 and 1938 and was awarded a PhD from the university in 1938. His research supervisor was Alonzo Church.[4] His results on computability and decidability were proved when he was just 23. He returned to Cambridge in 1939.

[3] However, Turing machines are in my view more intuitive than the lambda calculus.

[4] Alonzo Church was a famous American mathematician and logician who developed the lambda calculus. He also showed that Peano arithmetic and first-order logic were undecidable. Lambda calculus is equivalent to Turing machines in that whatever is computed may be computed by Lambda calculus or a Turing machine. This is known as the Church-Turing thesis.

14.3 Bletchley Park and the Enigma Codes

Turing and the team at Bletchley Park played an important role in the defeat of Nazi Germany during the Second World War. Turing's contribution was the role that he played at Bletchley Park in breaking the Enigma codes. These codes were used by the Germans for the transmission of naval messages. They allowed messages to be passed secretly to submarines using encryption and were designed to ensure that any unauthorised third-party interception of the messages would be unintelligible to the third party. The plain text (i.e. the original message) was converted by the Enigma machine into the encrypted text, and these messages were then transmitted by the Germans to their submarines in the Atlantic or to their bases throughout Europe (Fig. 14.4).

These messages contained top secret information on German submarine and naval activities in the Atlantic, and the threat that they posed to British and Allied shipping. The team at Bletchley Park succeeded in breaking of the Enigma codes, and this contributed to the Allied victory in Europe.

Turing worked in Hut 8 in the code-breaking centre at Bletchley Park which was located in northwest London. He designed an electromechanical machine known as the bombe, and this machine was designed to help break the Enigma codes by finding the right settings for the Enigma machine. This became the main tool to read the Enigma traffic during the war. It was first installed in early 1940, and by the end of the war, there were over 200 bombes in operation. The bombe weighed over a ton, and it was named after a cryptological device designed in 1938 by the Polish cryptologist Marian Rejewski.

A standard Enigma machine employed a set of three rotors. Each rotor could be in any of 26 positions. The bombe tried each possible rotor position and applied

Fig. 14.4 Bletchley Park

a test. The test eliminated almost all the 17,576 positions (i.e. 26^3) and left a smaller number of cases to be dealt with. The test required the cryptologist to have a suitable 'crib', that is, a section of cipher text for which he could guess the corresponding plain text.

Turing travelled to the United States in late 1942 and worked with US cryptanalysts on bombe construction. He also worked with Bell Labs to assist with the development of secure speech. He returned to Bletchley Park in early 1943 and designed a portable machine for secure voice communications. It was developed too late to be used in the war. Turing was awarded the Order of the British Empire[5] (OBE) in 1945 in recognition of his wartime services to Britain.

14.4 National Physical Laboratory

Turing worked as a scientific officer at the National Physical Laboratory (NPL) after the war, and he worked on the design of the Automatic Computing Engine (ACE).

He presented his design to NPL management in late 1945, and it included detailed logical circuit diagrams and an estimated cost £11,000. He proposed a high-speed memory, and the machine would implement subroutine calls which set it apart from the EDVAC. It also used an early form of programming language termed *Abbreviated Computer Instructions*. It was not possible for Turing to discuss his wartime experience at Bletchley Park including the work done on the Colossus computer due to the official secrets act in the UK. Turing's colleagues at NPL thought that his design was too ambitious.

Turing's design of ACE was the first complete design of a stored-program computer in Britain. The ACE computer was intended to be a smaller and more advanced computer than ENIAC. However, there were delays with funding to start the project, and Turing became disillusioned with NPL.

Turing's original machine was never built, and, instead, the NPL decided to build a smaller version of his machine. This first version was called the Pilot Model ACE, and it had 1,450 vacuum tubes and used mercury delay lines for its main memory. Each of the 12 delay lines could store 32 instructions or data words of 32 bits. The pilot machine ran its first program on May 10, 1950, and it was then the fastest computer in the world with a clock speed of 1 MHz.

He moved to the University of Manchester to work on the software for the Manchester Mark 1 computer. He also did some work on a chess program (Fig. 14.5).

[5] The British Empire was essentially over at the end of the Second World War with the United States becoming the dominant world power. India achieved independence from Britain in 1947, and former colonies gradually declared independence from Britain in the following years.

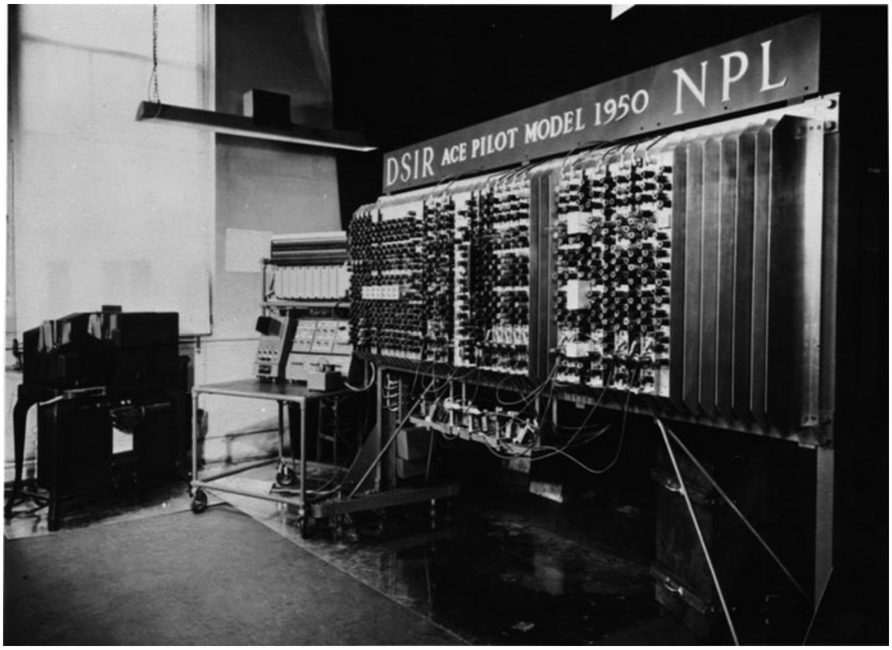


Fig. 14.5 NPL Pilot ACE (Courtesy of NPL © Crown Copyright 1950)

14.5 Turing Test in AI

Turing contributed to the debate concerning artificial intelligence in his 1950 paper on computing, machinery and intelligence [Tur:50]. His paper considered whether it could be possible for a machine to be conscious and to think. He predicted that it would be possible to speak of 'thinking machines', and he devised a famous experiment that would allow a computer to be judged as a conscious and thinking machine. This is known as the 'Turing test'.

The test is an adaptation of a party game which involves three participants. One of them, the judge, is placed in a separate room from the other two: one of whom is male and the other is female. Questions and responses are typed and passed under the door. The objective of the game is for the judge to determine which participant is male and which is female. The male is allowed to deceive the judge, whereas the female is supposed to assist.

Turing adapted this game to allow a computer to play the role of the male. The computer is said to pass the Turing test if the judge is unable to determine which of the participants is human and which is machine. Another words, if a computer can convince the judge that it is intelligent, then it is said to have passed the Turing test. The test is applied to test the linguistic capability of the machine rather than to the audio capability, and the conversation between the judge and the parties is conducted over a text-only channel.

Table 14.1 Objections to AI

Argument	Description
Theological objections	God has given an immortal soul to every man and woman. Thinking is a function of man's immortal soul. God has not given an immortal soul to animals or inanimate machines. Therefore, animals or machines cannot think
Head in the sand objections	The consequences of thinking machines would be dreadful to mankind. Let us hope that this never happens as it would undermine the superiority of man over animals and machines
Mathematical objections	There is a limitation to the power of discrete machines, and consequently, there are certain things that these machines cannot do. Gödel's theorem is one example of a limitation of what machines may do. However, there are also limits on what humans may do
Consciousness	A machine cannot feel or experience emotion such as joy, anger, happiness and so on
Lady Lovelace's objection	Ada Lovelace stated that the analytic engine can do only whatever we know or tell it to do. In other words, a machine can never do anything really new. However, as Turing points out, machines and computers often surprise us with new results
Informality of behaviour	This argument is along the lines that if each man has a set of laws (or rules of behaviour) that regulate his life, then he would be no better than a machine. But there are no such rules, and, therefore, men cannot be machines. However, the statement that there are no such rules or laws of behaviour is debatable
Learning machines	If a machine is to possess humanlike intelligence, it will need to be capable of learning. That is, the machine will need to be a learning machine that acquires knowledge by experience as well as having an initial core set of knowledge inserted into the machine. Turing argued that researchers should aim to produce a computer with a childlike mind and develop its knowledge by teaching it

Turing's work on 'thinking machines' caused public controversy, as defenders of traditional values attacked the idea of machine intelligence. The paper led to a debate concerning the nature of intelligence. John Searle later argued [Sea:80] in his Chinese room thought experiment that even if a machine passes the Turing test that it still may not be considered to be intelligent.

Turing strongly believed that machines would eventually be developed that would stand a good chance of passing the 'Turing test'. He considered the operation of 'thought' to be equivalent to the operation of a discrete state machine. Such a machine may be simulated by a program that runs on a single, universal machine, that is, a computer. He considered various objections to his position on artificial intelligence and attempted to refute these (Table 14.1).

There are a number of recent objections to the Turing test, including Searle's Chinese room argument, and this essentially argues that the simulation of human language behaviour is weaker than true intelligence. This is discussed in the next chapter.

Turing later did some work in biology in the area of morphology which is concerned with the form and structure of plants and animals.

14.6 Later Life

Turing was a homosexual which was illegal under the 1885 Criminal Law Amendment Act. This legislation prohibited acts of gross indecency between males. The Irish playwright Oscar Wilde was prosecuted in 1895 for indecent acts with Alfred Douglas under this law. Turing had a brief relationship with Arnold Murray in the early 1950s. However, Murray and an accomplice burgled Turing's house when the relationship ended in 1952. Turing reported the matter to the police, and at the criminal trial allegations of homosexuality were made against him.

He was charged and convicted. He was given a choice between imprisonment and probation. The terms of the probation were severe and required him to receive oestrogen hormone injections for a year. There were side affects with the treatment, and it led to depression and despair. Further, his conviction led to a removal of his security clearance and the end of his work on cryptography for the British government.

Turing committed suicide in June 1954. The medical examiner noted that the cause of death was by cyanide poisoning caused by eating an apple that was laced with cyanide.

He made outstanding contributions to mathematics, computer science, artificial intelligence and biology. The Association for Computing Machinery awards the annual 'Turing Award' to individuals who have made outstanding contributions to the computing field.

14.7 Review Questions

1. Explain the difference between finite state automata and the Turing machine.
2. Explain how the Turing machine was used to show that first-order logic is not decidable.
3. Discuss Turing's contributions to the development of the bombe at Bletchley Park.
4. Discuss Turing's contribution to the field of AI.

14.8 Summary

Turing was a British mathematician who made fundamental contributions to mathematics and computer science. He made important contributions to computability with the development of the theoretical Turing machine. He played an important role in breaking the German Enigma naval codes at Bletchley Park during the Second World War. He contributed to the work in the development of the first stored program computer at Manchester University, and he contributed to the emerging field of artificial intelligence.

His theoretical mathematical machine (i.e. the Turing machine) is capable of performing any conceivable mathematical problem that has an algorithm. He proved that whatever is computable is computable by this theoretical machine.

He contributed to the code-breaking work carried out at Bletchley Park during the Second World War. The team at Bletchley succeeded in breaking the German Enigma codes which contributed to the Allied victory in Europe.

He did important work in artificial intelligence and devised the famous ‘Turing test’. This test is employed to judge whether a machine is intelligent. He also did work at Manchester University on the Mark 1 computer prior to his premature death in the early 1950s.

Chapter 15
Artificial Intelligence

Key Topics

Turing Test
Searle's Chinese Room
Philosophical Problems in AI
Cognitive Psychology
Linguistics
Logic and AI
Robots
Cybernetics
Neural Networks
Expert Systems

15.1 Introduction

The long-term[1] goal of artificial intelligence is to create a thinking machine that is intelligent, has consciousness, has the ability to learn, has free will and is ethical. The field involves several disciplines such as philosophy, psychology, linguistics, machine vision, cognitive science, mathematics, logic and ethics. Artificial intelligence is a young field, and the term was coined by John McCarthy and others in 1956. Alan Turing had earlier devised the Turing test as a way to test the intelligent behaviour of a machine. There are deep philosophical problems in artificial intelligence, and some researchers believe that its goals are impossible or incoherent. These views are shared by Hubert Dreyfus and John Searle. Even if

[1] This long-term goal may be hundreds or even thousands of years.

artificial intelligence is possible, there are moral issues to consider such as the exploitation of artificial machines by humans and whether it is ethical to do this. Weizenbaum[2] has argued that artificial intelligence is unethical.

One of the earliest references to creating life by artificial means is that of the classical myth of Pygmalion. Pygmalion was a sculptor who carved a woman out of ivory. The sculpture was so realistic that he fell in love with it and offered the statue presents and prayed to Aphrodite, the goddess of love. Aphrodite took pity on him and brought the statue (Galathea) to life.

There are several stories of attempts by man to create life from inanimate objects, for example, the creation of the monster in Mary Shelly's Frankenstein. The monster is created by an overambitious scientist who is punished for his blasphemy of creation (in that creation is for God alone). The monster feels rejected following creation and inflicts a horrible revenge on its creator.

There are stories of the creation of the golem in Prague dating back to sixteenth century. A golem was created from mud by a holy person who was able to create life. However, the life that the holy person could create would always be a shadow of that created by God. Golems became servants to the holy men but were unable to speak.

The most famous golem story involved Rabbi Judah Loew of Prague. He is said to have created a golem from clay on the banks of the Vltava River to defend the Jews of Prague from attack. The golem was brought to life by the Rabbi by incantations in Hebrew. The golem became violent over time and started killing people in Prague. The Rabbi destroyed the golem by erasing the first letter of the word '*emet*' from the golem's forehead to make the Hebrew word '*met*', meaning death.

The story of the golem was given a more modern version in the Czech play 'Rossum's Universal Robots'. This science fiction play by Capek appeared in Prague in 1921. It was translated into English and appeared in London in 1923. It contains the first reference to the term 'robot', and the play considers the exploitation of artificial workers in a factory. The robots (or androids) are initially happy to serve humans but become unhappy with their existence over a period of time. The fundamental questions that the play is considering are whether the robots are being exploited, and, if so, is this ethical, and what should be the response of the robots to their exploitation. It eventually leads to a revolt by the robots and the extermination of the human race.

The story of Pinocchio as a fictional character first appeared in 1883 in the 'Adventures of Pinocchio' by Carlo Collodi. He was carved as a wooden puppet by a woodcarver named Geppetto, but he dreamt of becoming a real boy. Pinocchio's nose became larger whenever he told a lie.

[2] Weizenbaum was a psychologist who invented the ELIZA program. This program simulated a psychologist in dialogue with a patient. He was initially an advocate of artificial intelligence but later became a critic.

15.2 René Descartes

René Descartes was an influential French mathematician, scientist and philosopher. He was born in a village in the Loire valley in France in 1596 and studied law at the University of Poitiers. He never practised as a lawyer and instead served Prince Maurice of Nassau in the Netherlands. He invented the Cartesian coordinate system that is used in plane geometry and algebra. In this system, each point on the plane is identified through two numbers: the x-coordinate and the y-coordinate (Fig. 15.1).

He made important contributions to philosophy and attempted to derive a fundamental set of principles which can be known to be true. His approach was to renounce any idea that could be doubted. He rejected the senses since they can deceive and are not a sound source of knowledge. For example, during a dream, the subject perceives stimuli that appear to be real, but these have no existence outside the subject's mind. Therefore, it is inappropriate to rely on one's senses as the foundation of knowledge.

He argued that a powerful 'evil demon or mad scientist' could exist who sets out to manipulate and deceive subjects, thereby preventing them from knowing the true nature of reality. The evil demon could bring the subject into existence including an implanted memory. The question is how one can know for certain what is true given the limitations of the senses. The '*brain in the vat thought experiment*' is a more modern formulation of the idea of an evil spirit or mad scientist. A mad scientist could remove a person's brain from their body and place it in a vat and connects its neurons by wires to a supercomputer. The computer provides the disembodied brain with the electrical impulses that the brain would normally receive. The computer could then simulate reality, and the disembodied brain would have conscious experiences and would receive the same impulses as if it were inside a person's skull. There is no way to tell whether the brain is inside the vat or inside a person.

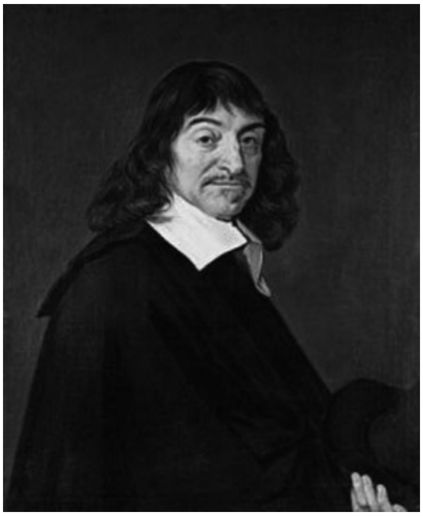


Fig. 15.1 René Descartes

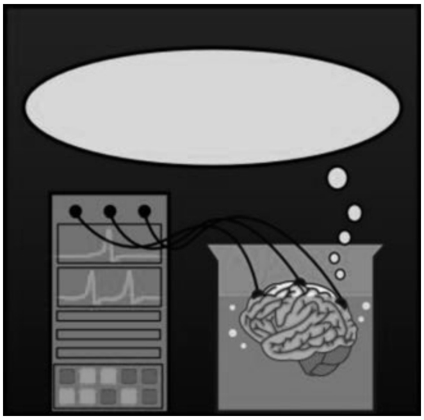


Fig. 15.2 Brain in a VAT thought experiment

That is, at any moment, an individual could potentially be a brain connected to a sophisticated computer program or inside a person's skull. Therefore, if you cannot be sure that you are not a brain in a vat, then you cannot rule out the possibility that all of your beliefs about the external world are false. This sceptical argument is difficult to refute.

The perception of a 'cat' in the case where the brain is in the vat is false and does not correspond to reality. It is impossible to know whether your brain is in a vat or inside your skull; it is therefore impossible to know whether your belief is valid or not (Fig. 15.2).

From this, Descartes deduced that there was one single principle that must be true. He argued that even if he is being deceived, then clearly he is thinking and must exist. This principle of existence or being is more famously known as 'cogito, ergo sum' (I think, therefore I am). Descartes argued that this existence can be applied to the present only, as memory may be manipulated and therefore doubted. Further, the only existence that he sure of is that he is a '*thinking thing*'. He cannot be sure of the existence of his body as his body is perceived by his senses which he has proven to be unreliable. Therefore, his mind or thinking thing is the only thing about him that cannot be doubted. His mind is used to make judgements and to deal with unreliable perceptions received via the senses.

Descartes constructed a system of knowledge from this one principle using the deductive method. He deduced the existence of a benevolent God using the ontological argument. He argues [Des:99] that we have an innate idea of a supremely perfect being (God), and that God's existence may be inferred immediately from the innate idea of a supremely perfect being:

1. I have an innate idea of a supremely perfect being (i.e. God).
2. Necessarily, existence is a perfection.
3. Therefore, God exists.

He then argued that since God is benevolent that he can have some trust in the reality that his senses provide. God has provided him with a thinking mind and does not wish to deceive him. He argued that knowledge of the external world can be obtained by both perception and deduction, and that reason or rationalism is the only reliable method of obtaining knowledge. His proof of the existence of God and the external world are controversial.

Descartes was a *dualist,* and he makes a clear *mind-body* distinction. He states that there are two substances in the universe: mental substances and bodily substances. The mind-body distinction is very relevant in AI, and the analogy of the human mind and brain is software running on a computer.

This thinking thing (*res cogitans* or mind/soul) was distinct from the rest of nature (*res extensa*) and interacted with the world through the senses to gain knowledge. Knowledge was gained by mental operations using the deductive method, where starting from the premises that are known to be true, further truths could be logically deduced. Descartes founded what would become known as the Rationalist school of philosophy where knowledge was derived solely by human reasoning. The *analogy of the mind in AI would be an AI program running on a computer* with knowledge gained by sense perception with sensors and logical deduction.

Descartes believed that the bodies of animals are complex living machines without feelings. He dissected (including vivisection) many animals for experiments. Vivisection has become controversial in recent years. His experiments led him to believe that the actions and behaviour of non-human animals can be fully accounted for by mechanistic means, and without reference to the operations of the mind. He realised from his experiments that a lot of human behaviour (e.g. physiological functions and blinking) is like that of animals in that it has a mechanistic explanation.

Descartes was of the view that well-designed automata[3] could mimic many parts of human behaviour. He argued that the key differentiators between human and animal behaviour were that humans could adapt to widely varying situations and also had the ability to use language. The use of language illustrates the power of the use of thought, and it clearly differentiates humans from animals. Animals do not possess the ability to use language for communication or reason. This, he argues, provides evidence for the presence of a soul associated with the human body. In essence, animals are pure machines, whereas humans are machines with minds (or souls).

The significance of Descartes in the field of artificial intelligence is that the Cartesian dualism that humans seem to possess would need to be reflected amongst artificial machines. Humans seem to have a distinct sense of ‘I’ as distinct from the body, and the ‘I’ seems to represent some core sense or essence of being that is unchanged throughout the person’s life. It somehow represents personhood, as distinct from the physical characteristics of a person that are inherited genetically.

[3] An automaton is a self-operating machine or mechanism that behaves and responds in a mechanical way.

The challenge for the AI community in the longer term is to construct a machine that (in a sense) possesses Cartesian dualism. That is, the long-term[4] goal of AI is to produce a machine that has awareness of itself as well as its environment.

15.3 The Field of Artificial Intelligence

The origin of the term 'artificial intelligence' is in work done on the proposal for Dartmouth Summer Research Project on Artificial Intelligence. This proposal was written by John McCarthy and others in 1955, and the research project took place in the summer of 1956.

The success of early AI went to its practitioners' heads, and they believed that they would soon develop machines that would emulate human intelligence. They convinced many of the funding agencies and the military to provide research grants as they believed that real artificial intelligence would soon be achieved. They had some initial (limited) success with machine translation, pattern recognition and automated reasoning. However, it is now clear that AI is a long-term project. Artificial intelligence is a multidisciplinary field and includes disciplines such as:

- Computing
- Logic and philosophy
- Psychology
- Linguistics
- Neuroscience and neural networks
- Machine vision
- Robotics
- Expert systems
- Machine translation
- Epistemology and knowledge representation

The British mathematician, Alan Turing, contributed to the debate concerning thinking machines, consciousness and intelligence in the early 1950s [Tur:50]. He devised the famous 'Turing test' to judge whether a machine was conscious and intelligent. Turing's paper was very influential as it raised the idea of the possibility of programming a computer to behave intelligently.

Shannon considered the problem of writing a chess program in the late 1940s and distinguished between a brute force strategy where the program could look at every combination of moves or a strategy where knowledge of chess could be used to select and examine a subset of available moves. The ability of a program to play

[4] This long-term goal may be hundreds of years as there is unlikely to be an early breakthrough in machine intelligence as there are deep philosophical problems to be solved. It took the human species hundreds of thousands of years to evolve to its current levels of intelligence.

Fig. 15.3 John McCarthy (Courtesy of John McCarthy)

chess is a skill that is considered intelligent, even though the machine itself is not conscious that it is playing chess.

Modern chess programs have been quite successful and have advantages over humans in terms of computational speed in considering combinations of moves. Kasparov was defeated by the IBM chess program 'Deep Blue' in a six-game match in 1997. The result of the match between Kasparov and X3D Fritz in 2003 was a draw.

Herbert Simon and Alan Newell developed the first theorem prover with their work on a program called 'Logic Theorist' or 'LT' [NeS:56]. This program could independently provide proofs of various theorems[5] in Russell's and Whitehead's Principia Mathematica [RuW:10]. It was demonstrated at the Dartmouth conference and showed that computers had the ability to encode knowledge and information and to perform intelligent operations such as solving theorems in mathematics.

John McCarthy proposed a program called the Advice Taker in his influential paper 'Programs with Common Sense' [Mc:59]. The idea was that this program would be able to draw conclusions from a set of premises, and McCarthy states that a program has common sense if it is capable of automatically deducing for itself 'a sufficiently wide class of immediate consequences of anything it is told and what it already knows' (Fig. 15.3).

The Advice Taker uses logic to represent knowledge (i.e. premises that are taken to be true), and it then applies the deductive method to deduce further truths from the relevant premises.[6] That is, the program manipulates the formal language

[5] Russell is said to have remarked that he was delighted to see that the Principia Mathematica could be done by machine, and that if he and Whitehead had known this in advance, they would not have wasted 10 years doing this work by hand in the early twentieth century. The LT program succeeded in proving 38 of the 52 theorems in Chap. 2 of Principia Mathematica. Its approach was to start with the theorem to be proved and to then search for relevant axioms and operators to prove the theorem.

[6] Of course, the machine would somehow need to know what premises are relevant and should be selected for to apply the deductive method from the many premises that are already encoded.

(e.g. predicate logic) and provides a conclusion that may be a statement or an imperative. McCarthy envisaged that the Advice Taker would be a program that would be able to learn and improve. This would be done by making statements to the program and telling it about its symbolic environment. The program will have available to it all the logical consequences of what it has already been told and previous knowledge. McCarthy's desire was to create programs to learn from their experience as effectively as humans do.

The McCarthy philosophy is that commonsense knowledge and reasoning can be formalised with logic. A particular system is described by a set of sentences in logic. These logic sentences represent all that is known about the world in general and what is known about the particular situation and the goals of the systems. The program then performs actions that it infers are appropriate for achieving its goals. That is, commonsense[7] knowledge is formalised by logic and commonsense problems are solved by logical reasoning.

15.3.1 Turing Test and Strong AI

Turing contributed to the debate concerning artificial intelligence in his 1950 paper on computing, machinery and intelligence [Tur:50]. Turing's paper considered whether it could be possible for a machine to be conscious and to think. He predicted that it would be possible to speak of machines thinking, and he devised a famous experiment that would allow a computer to be judged as a conscious and thinking machine. This is known as the 'Turing test'. The test itself is an adaptation of a well-known party game which involves three participants. One of them, the judge, is placed in a separate room from the other two: one is a male and the other is a female. Questions and responses are typed and passed under the door. The objective of the game is for the judge to determine which participant is male and which is female. The male is allowed to deceive the judge whereas the female is supposed to assist.

Turing adapted this game by allowing the role of the male to be played by a computer. The test involves a judge who is engaged in a natural language conversation with two other parties, one party is a human and the other is a machine. If the judge cannot determine which is machine and which is human, then the machine is said to have passed the 'Turing test'. That is, a machine that passes the Turing test must be considered intelligent, as it is indistinguishable from a human. The test is applied to test the linguistic capability of the machine rather than the audio capability, and the conversation is limited to a text-only channel.

[7] Common sense includes basic facts about events, beliefs, actions, knowledge and desires. It also includes basic facts about objects and their properties.

Turing's work on 'thinking machines' caused a great deal of public controversy as defenders of traditional values attacked the idea of machine intelligence. It led to a debate concerning the nature of intelligence. There has been no machine developed, to date, that has passed the Turing test.

Turing strongly believed that machines would eventually be developed that would stand a good chance of passing the 'Turing test'. He considered the operation of 'thought' to be equivalent to the operation of a discrete state machine. Such a machine may be simulated by a program that runs on a single, universal machine, that is, a computer.

Turing's viewpoint that a machine will one day pass the Turing test and be considered intelligent is known as '*strong artificial intelligence*'. It states that a computer with the right program would have the mental properties of humans. There are a number of objections to strong AI, and one well-known rebuttal is that of Searle's Chinese room argument [Sea:80].

15.3.1.1 Strong AI

The computer is not merely a tool in the study of the mind, rather the appropriately programmed computer really *is* a mind in the sense that computers given the right programs can be literally said to *understand* and have other cognitive states [Searle's 1980 Definition].

15.3.1.2 Weak AI

Computers just *simulate* thought, their seeming understanding is not real understanding (just as-if), their seeming calculation is only as-if calculation, etc. Nevertheless, computer simulation is useful for *studying* the mind (as for studying the weather and other things).

The Chinese room thought experiment was developed by John Searle to refute the feasibility of the strong AI project. It rejects the claim that a machine has or will someday have the same cognitive qualities as humans. Searle argues that brains cause minds and that syntax does not suffice for semantics.

A man is placed into a closed room into which Chinese writing symbols are input to him. He is given a rulebook that shows him how to manipulate the symbols to produce Chinese output. He has no idea as to what each symbol means, but with the rulebook, he is able to produce the Chinese output. This allows him to communicate with the other person and appear to understand Chinese. The rulebook allows him to answer any questions posed, without the slightest understanding of what he is doing or what the symbols mean.

1. Chinese characters are entered through slot 1.
2. The rulebook is employed to construct new Chinese characters.
3. Chinese characters are outputted to slot 2.

The question ‘Do you understand Chinese?’ could potentially be asked, and the rulebook would be consulted to produce the answer ‘Yes, of course’ despite of the fact that the person inside the room has not the faintest idea of what is going on. It will appear to the person outside the room that the person inside is knowledgeable on Chinese. The person inside is just following rules without understanding.

The process is essentially that of a computer program which has an input, performs a computation based on the input and then finally produces an output. Searle has essentially constructed a machine which can never be mental. Changing the program essentially means changing the rulebook, and this does not increase understanding. The strong artificial intelligence thesis states that given the right program, *any* machine running it would be mental. However, Searle argues that the program for this Chinese room would not understand anything and that therefore the strong AI thesis must be false. In other words, Searle’s Chinese room argument is a rebuttal of strong AI by showing that a program running on a machine that appears to be intelligent has no understanding whatsoever of the symbols that it is manipulating. That is, given any rulebook (i.e. program), the person would never understand the meanings of those characters that are manipulated.

That is, just because the machine acts like it knows what is going on, it actually only knows what it is programmed to know. It differs from humans in that it is not aware of the situation like humans are. It suggests that machines may not have intelligence or consciousness, and the Chinese room argument applies to any Turing equivalent computer simulation.

There are several rebuttals of Searle’s position,[8] and one well-known rebuttal attempt is the ‘System Reply’ argument. This reply argues that if a result associated with intelligence is produced, then intelligence must be found somewhere in the system. The proponents of this argument draw an analogy between the human brain and its constituents. None of its constituents have intelligence, but the system as a whole (i.e. the brain) exhibits intelligence. Similarly, the parts of the Chinese room may lack intelligence, but the system as a whole is intelligence.

15.4 Philosophy and AI

Artificial intelligence includes the study of knowledge and the mind, and there are deep philosophical problems (e.g. the nature of mind, consciousness and knowledge) to be solved.

[8] I don’t believe that Searle’s argument proves that strong AI is impossible. However, I am not expecting to see intelligent machines anytime soon.

Early work on philosophy was done by the Greeks as they attempted to understand the world and the nature of being and reality. Thales and the Miletians[9] attempted to find an underlying principle that would explain the nature of the world. Pythagoras believed that mathematics was the basic principle, and that everything (e.g. music) could be explained in terms of number. Plato distinguished between the world of appearances and the world of reality. He argued that the world of appearances resembles the flickering shadows on a cave wall, whereas reality is in the world of ideas[10] or forms, in which objects of this world somehow participate. Aristotle proposed the framework of a substance which includes form plus matter. For example, the matter of a wooden chair is the wood that it is composed of, and its form is the general form of a chair.

Descartes had a significant influence on the philosophy of mind and AI. Knowledge is gained by mental operations using the deductive method. This involves starting from premises that are known to be true and deriving further truths. He distinguished between the mind and the body (Cartesian dualism), and the analogy of the mind is an AI program running on a computer with sensors and logical deduction used to gain knowledge.

British empiricism rejected the Rationalist position and stressed the importance of empirical data in gaining knowledge about the world. It argued that all knowledge is derived from sense experience. It included philosophers such as Locke, Berkeley[11] and Hume. Locke argued that a child's mind is a blank slate (tabula rasa) at birth and that all knowledge is gained by sense experience. Berkeley argued that the ideas in a man's mind have no existence outside his mind [Ber:99], and this philosophical position is known as idealism.[12] David Hume formulated the standard empiricist philosophical position in 'An Enquiry concerning Human Understanding' [Hum:06] (Fig. 15.4).

[9] The term 'Miletians' refers to inhabitants of the Greek city state Miletus which is located in modern Turkey. Anaximander and Anaximenes were two other Miletians who made contributions to early Greek philosophy approx 600 B.C.

[10] Plato was an idealist, that is, that reality is in the world of ideas rather than the external world. Realists (in contrast) believe that the external world corresponds to our mental ideas.

[11] Berkeley was an Irish philosopher (not British). He was born in Dysart castle in Kilkenny, Ireland; educated at Trinity College, Dublin; and served as bishop of Cloyne in Co. Cork. He planned to establish a seminary in Bermuda for the sons of colonists in America, but the project failed due to lack of funding from the British government. Berkeley University in San Francisco is named after him.

[12] Berkeley's theory of ontology is that for an entity to exist, it must be perceived, that is, '*Esse est percipi*'. He argues that 'It is an opinion strangely prevailing amongst men, that houses, mountains, rivers, and in a world all sensible objects have an existence natural or real, distinct from being perceived'.

This led to a famous Limerick that poked fun at Berkeley's theory. 'There once was a man who said God; Must think it exceedingly odd; To find that this tree, continues to be; When there is no one around in the Quad'.

The reply to this Limerick was appropriately: 'Dear sir, your astonishments odd; I am always around in the Quad; And that's why this tree will continue to be; Since observed by, yours faithfully, God'.

Fig. 15.4 George Berkely. Bishop of Cloyne

Fig. 15.5 David Hume

Hume argued that all objects of human knowledge may be divided into two kinds: *matters of fact* propositions that are based entirely on experience or *relation of ideas* propositions that may be demonstrated (such as geometry) via deduction reasoning in the operations of the mind. He argued that any subject[13] proclaiming knowledge that does not adhere to these empiricist principles should be committed to the flames[14] as such knowledge contains nothing but sophistry and illusion (Fig. 15.5).

[13] Hume argues that these principles apply to subjects such as Theology as its foundations are in faith and divine revelation which are neither matters of fact nor relations of ideas.

[14] 'When we run over libraries, persuaded of these principles, what havoc must we make? If we take in our hand any volume; of divinity or school metaphysics, for instance; let us ask, *Does it contain any abstract reasoning concerning quantity or number?* No. *Does it contain any experimental reasoning concerning matter of fact and existence?* No. Commit it then to the flames: for it can contain nothing but sophistry and illusion'.

Kant's Critique of Pure Reason [Kan:03] was published in 1781 and is a response to Hume's theory of empiricism. Kant argued that there is a third force in human knowledge that provides concepts that cannot be inferred from experience. Such concepts include the laws of logic (e.g. modus ponens), causality and so on, and Kant argued that the third force was the manner in which the human mind structures its experiences. These structures are called categories.

The continental school of philosophy included thinkers such as Heidegger and Merleau-Ponty who argued that the world and the human body are mutually intertwined. Heidegger emphasised that existence can only be considered with respect to a changing world. Merleau-Ponty emphasised the concept of a body-subject that actively participates both as the perceiver of knowledge and as an object of perception.

Philosophy has been studied for over two millennia, but to date, very little progress has been made in solving its fundamental questions. However, it is important that it be considered as any implementation of AI will make philosophical assumptions, and it is important that these be understood.

15.5 Cognitive Psychology

Psychology arose out of the field of psychophysics in the late nineteenth century with work by German pioneers in attempting to quantify perception and sensation. Fechner's mathematical formulation of the relationship between stimulus and sensation is given by

$$S = k \log I + c$$

The symbol S refers to the intensity of the sensation, the symbols k and c are constants and the symbol I refers to the physical intensity of the stimulus. Psychology was defined by William James as the science of mental life.

One of the early behavioural psychologist's was Pavlov who showed that it was possible to develop a conditional reflex in a dog. He showed that it is possible to make a dog salivate in response to the ringing of a bell. This is done by ringing a bell each time before meat is provided to the dog, and the dog therefore associates the presentation of meat with the ringing of the bell after a training period.

Skinner developed the concept of conditioning further using rewards to reinforce desired behaviour and punishment to discourage undesired behaviour. Positive reinforcement helps to motivate the individual to behave in the desired way, with punishment used to deter the individual from performing undesired behaviour. The behavioural theory of psychology explains many behavioural aspects of the world. However, it does not really explain complex learning tasks such as language development.

Merleau-Ponty[15] considered the problem of what the structure of the human mind must be in order to allow the objects of the external world to exist in our minds in the form that they do. He built upon the theory of phenomenology as developed by Hegel and Husserl. Phenomenology involves a focus and exploration of phenomena with the goal of establishing the essential features of experience. Merleau-Ponty introduced the concept of the body-subject which is distinct from the Cartesian view that the world is just an extension of our own mind. He argued that the world and the human body are mutually intertwined. The Cartesian view is that the self must first be aware of and know its own existence prior to being aware of and recognising the existence of anything else.

The body has the ability to perceive the world, and it plays a double role in that it is both the subject (i.e. the perceiver) and the object (i.e. the entity being perceived) of experience. Human understanding and perception is dependent on the body's capacity to perceive via the senses and its ability to interrogate its environment. Merleau-Ponty argued that there is a symbiotic relationship between the perceiver and what is being perceived, and he argues that as our consciousness develops, the self imposes richer and richer meanings on objects. He provides a detailed analysis of the flow of information between the body-subject and the world.

Cognitive psychology is a branch of psychology that is concerned with learning, language, memory and internal mental processes. Its roots lie in Piaget's child development psychology and in Wertheimer's Gestalt psychology. The latter argues that the operations of the mind are holistic and that the mind contains a self-organising mechanism. Holism argues that the sum of the parts is less than the whole and is the opposite of logical atomism[16] developed by Bertrand Russell. Russell (and also Wittgenstein) attempted to identify the atoms of thought, that is, the elements of thought that cannot be divided into smaller pieces. Logical atomism argues was that all truths are ultimately dependent on a layer of atomic facts. It had an associated methodology whereby by a process of analysis, it attempted to construct more complex notions in terms of simpler ones.

Cognitive psychology was developed in the late 1950s and is concerned with how people understand, diagnose and solve problems, as well as the mental processes that take place during a stimulus and its corresponding response. It argues that solutions to problems take the form of rules, heuristics and sudden insight, and it considers the mind as having a certain conceptual structure. The dominant paradigm in the field has been the *information processing model*, which considers the mental processes of thinking and reasoning as being equivalent to software

[15] Merleau-Ponty was a French philosopher who was strongly influenced by the phenomenology of Husserl. He was also closely associated with the French existentialist philosophers such as Jean-Paul Sartre and Simone De Beauvoir.

[16] Atomism actually goes back to the work of the ancient Greeks and was originally developed by Democritus and his teacher Leucippus in the fifth century B.C. Atomism was rejected by Plato in the dialogue the Timaeus.

running on the computer, that is, the brain. It has associated theories of input, representation of knowledge, processing of knowledge and output.

Cognitive psychology has been applied to artificial intelligence from the 1960s, and some of the research areas include:

- Perception
- Concept formation
- Memory
- Knowledge representation
- Learning
- Language
- Grammar and linguistics
- Thinking
- Logic and problem solving

It is clear that for a machine to behave with intelligence, it will need to be able to perceive objects in the physical world. It must be able to form concepts and to remember knowledge that it has already been provided with. It will need an understanding of temporal events. Knowledge must be efficiently represented to allow easy retrieval for analysis and decision making. An intelligent machine will need the ability to produce and understand written or spoken language. A thinking machine must be capable of thought, analysis and problem solving.

15.6 Linguistics

Linguistics is the theoretical and applied study of language, and human language is highly complex. It includes the study of phonology, morphology, syntax, semantics and pragmatics. Syntax is concerned with the study of the rules of grammar, and the syntactically valid sentences and phrases are formed by applying the rules of the grammar. Morphology is concerned with the formation and alteration of words, and phonetics is concerned with the study of sounds and how sounds are produced and perceived as speech (or non-speech).

Computational linguistics is an interdisciplinary study of the design and analysis of natural language processing systems. It includes linguists, computer scientists, experts in artificial intelligence, cognitive psychologists and mathematicians.

Early work on computational linguistics commenced with machine translation work in the United States in the 1950s. The objective was to develop an automated mechanism by which Russian language texts could be translated directly into English without human intervention. It was naively believed that it was only a matter of time before automated machine translation would be done.

However, the initial results were not very successful, and it was realised that the automated processing of human languages was considerably more complex. This led to the birth of a new field called computational linguistics, and the objective of this field is to investigate and develop algorithms and software for processing natural languages. It is a subfield of artificial intelligence and deals with the comprehension and production of natural languages.

The task of translating one language into another requires an understanding of the grammar of both languages. This includes an understanding of the syntax, the morphology, semantics and pragmatics of the language. For artificial intelligence to become a reality, it will need to make major breakthroughs in computational linguistics.

15.7 Cybernetics

The interdisciplinary field of cybernetics[17] began in the late 1940s when concepts such as information, feedback and regulation were generalised from engineering to other systems. These include systems such as living organisms, machines, robots and language. The term '*cybernetics*' was coined by Norbert Wiener, and it was taken from the Greek word 'κυβερνητη' (meaning steersman or governor). It is the study of communications and control and feedback in living organisms and machines to ensure efficient action.

The name is well chosen as a steersman needs to respond to different conditions and feedback while steering the boat to achieve the goal of travel to a particular destination. Similarly, the field of cybernetics is concerned with the interaction of goals, predictions, actions, feedback and responses in all kinds of systems. It uses models of organisations, feedback and goals to understand the capacity and limits of any system.

It is concerned with knowledge acquisition through control and feedback. Its principles are similar to human knowledge acquisition where learning is achieved through a continuous process of feedback from parents and peers which leads to adaptation and transformation of knowledge rather than its explicit encoding.

The conventional belief in AI is that knowledge may be stored inside a machine, and that the application of stored knowledge to the real world in this way constitutes intelligence. External objects are mapped to internal states on the machine, and machine intelligence is manifested by the manipulation of the internal states. This approach has been reasonably successful with rule-based expert systems, but it has made limited progress in creating intelligent machines. Therefore, alternative approaches such as cybernetics warrant further research. Cybernetics views information (or intelligence) as an attribute of an interaction, rather than something that is stored in a computer.

[17] Cybernetics was defined by Couffignal (one of its pioneers) as the art of ensuring the efficacy of action.

15.8 Logic and AI

Mathematical logic is used in the AI field to formalise knowledge and reasoning. Commonsense reasoning is required for solving problems in the real world, and therefore McCarthy [Mc:59] argues that it is reasonable for logic to play a key role in the formalisation of commonsense knowledge. This includes the formalisation of basic facts about actions and their effects, facts about beliefs and desires and facts about knowledge and how it is obtained. His approach allows commonsense problems to be solved by logical reasoning.

Its formalisation requires sufficient understanding of the commonsense world, and often the relevant facts to solve a particular problem are unknown. It may be that knowledge thought relevant may be irrelevant and vice versa. A computer may have millions of facts stored in its memory, and the problem is how to determine the relevant facts from its memory to serve as premises in logical deduction.

McCarthy influential 1959 paper discusses various commonsense problems such as getting home from the airport. Other examples are diagnosis, spatial reasoning and understanding narratives that include temporal events. Mathematical logic is the standard approach to express premises, and it includes rules of inferences that are used to deduce valid conclusions from a set of premises. It provides a rigorous definition of deductive reasoning showing how new formulae may be logically deduced from a set or premises.

Propositional calculus associates a truth value with each proposition and includes logical connectives to produce formulae such as $A \Rightarrow B$, $A \wedge B$ and $A \vee B$. The truth values of the propositions are normally the binary values of *true* and *false*. There are other logics, such as three-valued logic or fuzzy logics that allow more than two truth values for a proposition. Predicate logic is more expressive than propositional logic and includes quantifiers and variables. It can formalise the syllogism 'All Greeks are mortal; Socrates is a Greek; Therefore, Socrates is mortal'. The predicate calculus consists of:

- Axioms
- Rules for defining well-formed formulae
- Inference rules for deriving theorems from premises

A formula in predicate calculus is built up from the basic symbols of the language. These include variables, predicate symbols such as equality; function symbols, constants logical symbols such as $\exists$, $\wedge$, $\vee$ and $\neg$ and punctuation symbols such as brackets and commas. The formulae of predicate calculus are built from terms, where a *term* is defined recursively as a variable or individual constant or as some function containing terms as arguments. A formula may be an atomic formula or built from other formulae via the logical symbols.

There are several rules of inference associated with predicate calculus, and the most important of these are modus ponens and generalisation. The rule of modus ponens states that given two formulae p, and $p \Rightarrow q$, we may deduce q. The rule of generalisation states that given a formula p that we may deduce $\forall(x)p$.

McCarthy's approach to programs with common sense has been criticised by Bar-Hillel and others on the grounds that common sense is fairly elusive and the difficulty that a machine would have in determining which facts are relevant to a particular deduction from its known set of facts. However, logic remains an important area in AI.

15.9 Computability, Incompleteness and Decidability

An algorithm (or procedure) is a finite set of unambiguous instructions to perform a specific task. The term 'algorithm' is named after the Persian mathematician Al-Khwarizmi. The concept of an algorithm was defined formally by Church in 1936 and independently by Turing. Church defined computability in terms of the lambda calculus, and Turing defined computability in terms of the theoretical Turing machine. These formulations are equivalent.

Formalism was proposed by Hilbert as a foundation for mathematics in the early twentieth century. A formal system consists of a formal language, a set of axioms and rules of inference. Hilbert's program was concerned with the formalisation of mathematics (i.e. the axiomatisation of mathematics) together with a proof that the axiomatisation was consistent. The specific objectives of Hilbert's program were to:

– Develop a formal system where the truth or falsity of any mathematical statement may be determined
– A proof that the system is consistent (i.e. that no contradictions may be derived)

A proof in a formal system consists of a sequence of formulae, where each formula is either an axiom or derived from one or more preceding formulae in the sequence by one of the rules of inference. Hilbert believed that every mathematical problem could be solved and therefore expected that the formal system of mathematics would be complete (i.e. all truths could be proved within the system) and decidable, that is, that the truth or falsity of any mathematical proposition could be determined by an algorithm. However, Church and Turing independently showed this to be impossible in 1936, and the only way to determine whether a statement is true or false is to try to solve it.

Russell and Whitehead published *Principia Mathematica* in 1910, and this three-volume work on the foundations of mathematics attempted to derive all mathematical truths in arithmetic from a well-defined set of axioms and rules of inference. The questions remained whether the Principia was *complete* and *consistent*. That is, is it possible to derive all the truths of arithmetic in the system, and is it possible to derive a contradiction from the Principia's axioms?

Gödel's second incompleteness theorem [Goe:31] showed that first-order arithmetic is incomplete, and that the consistency of first-order arithmetic cannot be proved within the system. Therefore, if first-order arithmetic cannot prove its own consistency, then it cannot prove the consistency of any system that contains first-order arithmetic. These results dealt a fatal blow to Hilbert's program.

15.10 Robots

The first use of the term 'robot' was by the Czech playwright Karel Capek in his play '*Rossum's Universal Robots*' performed in Prague in 1921. The word 'robot' is from the Czech word for forced labour. The theme explored is whether it is ethical to exploit artificial workers in a factory and what response the robots should make to their exploitation. Capek's robots were not mechanical or metal in nature and were instead created through chemical means. Capek rejected the idea that machines created from metal could think or feel.

Asimov wrote several stories about robots in the 1940s including the story of a robotherapist.[18] He predicted the rise of a major robot industry, and he also introduced a set of rules (or laws) for good robot behaviour. These are known as the three Laws of Robotics, and a fourth law was later added by Asimov (Table 15.1).

The term 'robot' is defined by the Robot Institute of America as:

Definition 15.1 (Robots). *A re-programmable, multifunctional manipulator designed to move material, parts, tools, or specialized devices through various programmed motions for the performance of a variety of tasks.*

Joseph Engelberger and George Devol are considered the fathers of robotics. Engelberger set up the first manufacturing company 'Unimation' to make robots, and Devol wrote the necessary patents. Their first robot was called the 'Unimate'. These robots were very successful and reliable and saved their customer's (General Motors) money by replacing staff with machines. The robot industry continues to play a major role in the automobile sector.

Robots have been very effective at doing clearly defined repetitive tasks, and there are many sophisticated robots in the workplace today. These are industrial manipulators that are essentially computer-controlled 'arms and hands'. However, fully functioning androids are many years away.

Robots can also improve the quality of life for workers as they can free human workers from performing dangerous or repetitive jobs. Further, it leads to work for robot technicians and engineers. Robots provide consistently high-quality products

Table 15.1 Laws of robotics

Law	Description
Law 0	A robot may not injure humanity or, through inaction, allow humanity to come to harm
Law 1	A robot may not injure a human being or, through inaction, allow a human being to come to harm, unless this would violate a higher order law
Law 2	A robot must obey orders given by human beings, except where such orders would conflict with a higher order law
Law 3	A robot must protect its own existence as long as such protection does not conflict with a higher order law

[18] The first AI therapist was the ELIZA program produced by Weizenbaum in the mid-1960s.

and can work tirelessly 24 h a day. This helps to reduce the costs of manufactured goods thereby benefiting consumers. They do not require annual leave but will, of course, from time to time require servicing by engineers or technicians. However, there are impacts on workers whose jobs are displaced by robots.

15.11 Neural Networks

The term 'neural network' refers to an interconnected group of processing elements called nodes or neurons. These neurons cooperate and work together to produce an output function. Neural networks may be artificial or biological. A biological network is part of the human brain, whereas an artificial neural network is designed to mimic some properties of a biological neural network. The processing of information by a neural network is done in parallel rather than in series.

A unique property of a neural network is fault tolerance; that is, it can still perform (within certain tolerance levels) its overall function even if some of its neurons are not functioning. Neural network may be trained to learn to solve complex problems from a set of examples. These systems may also use the acquired knowledge to generalise and solve unforeseen problems.

A biological neural network is composed of billions of neurons (or nerve cells). A single neuron may be physically connected to thousands of other neurons, and the total number of neurons and connections in a network may be enormous. The human brain consists of many billions of neurons, and these are organised into a complex intercommunicating network. The connections are formed through axons[19] to dendrites,[20] and the neurons can pass electrical signals to each other. These connections are not just the binary digital signals of *on* or *off*, and, instead, the connections have varying strength which allows the influence of a given neuron on one of its neighbours to vary from very weak to very strong.

That is, each connection has an individual weight (or number) associated with it that indicates its strength. Each neuron sends its output value to all other neurons to which it has an outgoing connection. The output of one neuron can influence the activations of other neurons causing them to fire. The neuron receiving the connections calculates its activation by taking a weighted sum of the input signals. Networks learn by changing the weights of the connections. Many aspects of brain function, especially the learning process, are closely associated with the adjustment of these connection strengths. Brain activity is represented by particular patterns of firing activity amongst the network of neurons. This simultaneous

[19] These are essentially the transmission lines of the nervous system. They are microscopic in diameter and conduct electrical impulses. The axon is the output from the neuron, and the dendrites are input.

[20] Dendrites extend like the branches of a tree. The origin of the word dendrite is from the Greek word (δενδρον) for tree.

cooperative behaviour of a huge number of simple processing units is at the heart of the computational power of the human brain.[21]

Artificial neural networks aim to simulate various properties of biological neural networks. They consist of many hundreds of simple processing units which are wired together in a complex communication network. Each unit or node is a simplified model of a real neuron which fires[22] if it receives a sufficiently strong input signal from the other nodes to which it is connected. The strength of these connections may be varied in order for the network to perform different tasks corresponding to different patterns of node firing activity. The objective is to solve a particular problem, and artificial neural networks have been applied to speech recognition problems, image analysis and so on.

The human brain employs massive parallel processing whereas artificial neural networks have provided simplified models of the neural processing that takes place in the brain. The largest artificial neural networks are tiny compared to biological neural networks. The challenge for the field is to determine what properties individual neural elements should have to produce something useful representing intelligence.

Neural networks are quite distinct from the traditional von Neumann architecture which is based on the sequential execution of machine instructions. The origins of neural networks lie in the attempts to model information processing in biological systems. This relies more on parallel processing as well as on implicit instructions based on pattern recognition from sense perceptions of the external world.

The nodes in an artificial neural network are composed of many simple processing units which are connected into a network. Their computational power depends on working together (parallel processing) on any task, and computation is related to the dynamic process of node firings rather than sequential execution of instructions. This structure is much closer to the operation of the human brain and leads to a computer that may be applied to a number of complex tasks.

15.12 Expert Systems

An expert system is a computer system that contains domain knowledge of one or more human experts in a narrow specialised domain. It consists of a set of rules (or knowledge) supplied by the domain experts about a specific class of problems and allows knowledge to be stored and intelligently retrieved. The effectiveness of the expert system is largely dependent on the accuracy of the rules provided, as incorrect inferences will be drawn with incorrect rules. Several commercial expert systems have been developed since the 1960s.

Expert systems have been a success story in the AI field. They have been applied to the medical field, equipment repair and investment analysis. They employ

[21] The brain works through massive parallel processing.

[22] The firing of a neuron means that it sends off a new signal with a particular strength (or weight).

Table 15.2 Expert systems

Component	Description
Knowledge base	The knowledge base is represented as a set of rules of form (IF condition THEN action)
Inference engine	Carries out reasoning by which expert system reaches conclusion
Explanatory facility	Explains how a particular conclusion was reached
User interface	Interface between user and expert system
Database/memory	Set of facts used to match against IF conditions in knowledge base

a logical reasoning capability to draw conclusions from known facts as well as recommending an appropriate course of action to the user.

An expert system consists of the following components: a knowledge base, an inference engine, an explanatory facility, a user interface and a database (Table 15.2).

Human knowledge of a specialty is of two types: namely, public knowledge and private knowledge. The former includes the facts and theories documented in text books and publications, whereas the latter refers to knowledge that the expert possesses that has not found its way into the public domain. The latter often consists of rules of thumb or heuristics that allow the expert to make an educated guess where required, as well as allowing the expert to deal effectively with incomplete or erroneous data. It is essential that the expert system encodes public and private knowledge to enable it to draw valid inferences.

The inference engine is made up of many inference rules that are used by the engine to draw conclusions. Rules may be added or deleted without affecting other rules, and this reflects the normal updating of human knowledge. Out of date facts may be deleted and are no longer used in reasoning, while new knowledge may be added and applied in reasoning. The inference rules use reasoning which is closer to human reasoning. There are two main types of reasoning with inference rules, and these are backward chaining and forward chaining. Forward chaining starts with the data available and uses the inference rules to draw intermediate conclusions until a desired goal is reached. Backward chaining starts with a set of goals and works backwards to determine if one of the goals can be met with the data that is available.

The expert system makes its expertise available to decision makers who need answers quickly. This is extremely useful as often there is a shortage of experts and the availability of an expert computer with in-depth knowledge of specific subjects is therefore very attractive. Expert systems may also assist managers with long-term planning. There are many small expert systems available that are quite effective in a narrow domain. The long-term goal is to develop expert systems with a broader range of knowledge. Expert systems have enhanced productivity in business and engineering, and there are several commercial software packages available to assist.

Several expert systems (e.g. Mycin, Colossus and Dendral) have been developed. Mycin was developed at Stanford University in the 1970s. It was written in LISP and was designed to diagnose infectious blood diseases and to recommend appropriate antibiotics and dosage amounts corresponding to the patient's body weight. It had a knowledge base of approximately 500 rules and a fairly simple inference engine. Its approach was to query the physician running the program with a long list of yes/no questions. Its output consisted of various possible bacteria that could correspond to the blood disease, along with an associated probability that indicated the confidence in the diagnosis. It also included the rationale for the diagnosis and a course of treatment appropriate to the diagnosis.

Mycin had a correct diagnosis rate of 65%. This was better than the diagnosis of most physicians who did not specialise in bacterial infections. However, its diagnosis rate was less than that of experts in bacterial infections who had a success rate of 80%. Mycin was never actually used in practice due to legal and ethical reasons on the use of expert systems in medicine. For example, if the machine makes the wrong diagnosis, who is to be held responsible?

Colossus was an expert system used by several Australian insurance companies. It was used to help insurance adjusters assess personal injury claims, and helped to improve consistency, objectivity and fairness in the claims process. It guides the adjuster through an evaluation of medical treatment options, the degree of pain and suffering of the claimant and the extent that there is permanent impairment to the claimant, as well as the impact of the injury on the claimant's lifestyle. It was developed by Computer Sciences Corporation (CSC).

Dendral (Dendritic Algorithm) was developed at Stanford University in the mid-1960s, and it was the first use of artificial intelligence in medical research. Its objectives were to assist the organic chemist with the problem of identifying unknown organic compounds and molecules by computerised spectrometry. This involved the analysis of information from mass spectrometry graphs and knowledge of chemistry. Dendral automated the decision-making and problem-solving process used by organic chemists to identify complex unknown organic molecules. It was written in LISP, and it showed how an expert system could employ rules, heuristics and judgement to guide scientists in their work.

15.13 Review Questions

1. Discuss Descartes and his Rationalist philosophy and his relevance to artificial intelligence.
2. Discuss the Turing test and its relevance on strong AI. Discuss Searle's Chinese room rebuttal arguments. What are your own views on strong AI?
3. Discuss the philosophical problems underlying artificial intelligence.
4. Discuss the applicability of logic to artificial intelligence.
5. Discuss neural networks and their applicability to artificial intelligence.
6. Discuss expert systems.

15.14 Summary

Artificial intelligence is a multidisciplinary field, and its branches include logic and philosophy, psychology, linguistics, machine vision, neural networks and expert systems.

Turing believed that machine intelligence was achievable, and he devised the 'Turing test' to judge whether a machine was intelligent. Searle's Chinese room argument is a rebuttal of strong AI and aims to demonstrate that a machine will never have the same cognitive qualities as a human even if it passes the Turing test.

McCarthy proposed programs with commonsense knowledge and reasoning formalised with logic. He argued that human-level intelligence may be achieved with a logic-based system.

Cognitive psychology is concerned with cognition and some of its research areas include perception, memory, learning, thinking and logic and problem solving. Linguistics is the scientific study of language and includes the study of syntax and semantics.

Artificial neural networks aim to simulate various properties of biological neural networks. They consist of many hundreds of simple processing units which are wired together in a complex communication network. Each unit or node is a simplified model of a real neuron which fires if it receives a sufficiently strong input signal from the other nodes to which it is connected. The strength of these connections may be varied in order for the network to perform different tasks corresponding to different patterns of node firing activity. Artificial neural networks have been successfully applied to speech recognition problems and to image analysis.

An expert system is a computer system that allows knowledge to be stored and intelligently retrieved. It is a program that is made up of a set of rules (or knowledge). These rules are generally supplied by the domain experts about a specific class of problems. They include a problem-solving component that allows analysis of the problem to take place, as well as recommending an appropriate course of action to solve the problem.

Glossary

ABC	Altanasoff-Berry Computer
AI	Artificial Intelligence
AIX	Advanced IBM UNIX
ALGOL	Algorithmic language
AOL	America On-Line
AMN	Abstract Machine Notation
AMPS	Advanced Mobile Phone Services
ANS	Advanced Network Service
APPN	Advanced Peer-To-Peer Networking
ARPA	Advanced Research Project Agency
ASCC	Automatic Sequence Controlled Calculator
ASCII	American Standard Code for Information Interchange
AT&T	American Telephone and Telegraph Company
ATM	Automated Teller Machine
B2B	Business to Business
B2C	Business to Consumer
BASIC	Beginners All purpose Symbolic Instruction Code
BBN	Bolt Beranek and Newman
BNF	Backus Naur Form
CCS	Calculus Communicating Systems
CDMA	Code Division Multiple Access
CEO	Chief Executive Officer
CERN	Conseil European Recherche Nucleaire
CERT	Computer Emergency Response Team
CGI	Common Gateway Interface
CICS	Customer Information Control System
CMM®	Capability Maturity Model
CMMI®	Capability Maturity Model Integration
COBOL	Common Business Oriented Language
CODASYL	Conference on Data Systems Languages
COM	Component Object Model

CP/M	Control Program for Microcomputers
CPU	Central Processing Unit
CRM	Customer Relationship Management
CRT	Cathode Ray Tube
CSIRAC	Council for Scientific and Industrial Research Automatic Computer
CSP	Communication Sequential Processes
CTR	Computing Tabulating Recording Company
DARPA	Defence Advanced Research Project Agency
DB	Database
DEC	Digital Equipment Corporation
DES	Data Encryption Standard
DNS	Domain Naming System
DOS	Disk Operating System
DRAM	Dynamic Random Access Memory
DSDM	Dynamic Systems Development Method
DMAIC	Define, Measure, Analyse, Improve, Control
DRAM	Dynamic RAM
DVD	Digital Versatile Disc
EJB	Enterprise Java Beans
EBCDIC	Extended Binary Coded Decimal Interchange Code
EDVAC	Electronic Discrete Variable Automatic Computer
ENIAC	Electronic Numerical Integrator and Computer
ERP	Enterprise Resource Planning
ESI	European Software Institute
ETSI	European Telecommunication Standards Institute
FAA	Federal Aviation Authority
FCC	Federal Communications Commission
FDMA	Frequency Division Multiple Access
FTP	File Transfer Protocol
FSM	Finite State Machine
GNU	GNU's Not Unix
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communication
GUI	Graphical User Interface
HICL	High Integrity Computing Laboratory
HP	Hewlett Packard
HTML	Hypertext Markup Language
HTTP	Hyper Text Transport Protocol
IBM	International Business Machines
ICL	International Computer Ltd.
IEEE	Institute of Electrical and Electronic Engineers
IMP	Interface Message Processors
IP	Internet Protocol
IPO	Initial Public Offering

IPTO	Information Processing Technology Office
ISO	International Standards Organization
JAD	Joint Application Development
JVM	Java Virtual Machine
LAN	Local Area Network
LEO	Lyons Electronic Office
LSI	Large Scale Integration
LT	Logic Theorist
MADC	Massachusetts Automatic Digital Computer
MIPS	Million Instructions Per Second
MIT	Massachusetts Institute of Technology
MITS	Micro Instrumentation and Telemetry System
MOD	Ministry of Defence (U.K.)
MSN	Microsoft Network
NAP	Network Access Point
NCP	Network Control Protocol
NORAD	North American Aerospace Defence Command
NPL	National Physical Laboratory
NSF	National Science Foundation
NWG	Network Working Group
OSI	Open Systems Interconnection
OS	Operating System
PC	Personal Computer
PET	Personal Electronic Transactor
PDP	Programmed Data Processor
PL/M	Programming Language for Microcomputers
PDA	Personal Digital Assistant
PIN	Personal Identification Number
PS/2	Personal System 2
RACE	Research Advanced Communications Europe
RAD	Rapid Application Development
RAISE	Rigorous Approach to Industrial Software Engineering
RAM	Random Access Memory
RDBMS	Relational Database Management System
RISC	Reduced Instruction Set Computer
RSL	RAISE Specification Language
RSRE	Royal Signals and Radar Establishment
SAA	Systems Application Architecture
SAGE	Semi-Automatic Ground Environment
SCORE	Service Creation in an Object Reuse Environment
SDI	Strategic Defence Initiative
SDL	Specification and Description Language
SECD	Stack, Environment, Core, Dump
SEI	Software Engineering Institute

SIM	Subscriber Identity Module
SLA	Service Level Agreement
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
SNA	Systems Network Architecture
SPI	Software Process Improvement
SPICE	Software Process Improvement and Capability determination
SQA	Software Quality Assurance
SQL	Structured Query Language
SQRL	Software Quality Research Laboratory
SRI	Stanford Research Institute
SSL	Secure Socket Layer
TCP	Transport Control Protocol
UCLA	University of California (Los Angeles)
UDP	User Datagram Protocol
UML	Unified Modelling Language
UMTS	Universal Mobile Telecommunications System
URL	Universal Resource Locator
VAX	Virtual Address eXtension
VDM	Vienna Development Method
VLSI	Very Large Scale Integration
VDM$^{♣}$	Irish School of VDM
VM	Virtual Memory
W3C	World Wide Web Consortium
WISC	Wisconsin Integrally Synchronized Computer
WWW	World Wide Web
XP	eXtreme Programming

References

[AnL:95]	Anglin, W.S., Lambek, J.: The Heritage of Thales. Springer, New York (1995)
[Bab:11]	Baber, R.L.: The Language of Mathematics. Utilizing Math in Practice. Wiley, Hoboken (2011)
[Bab:42]	Menabrea, L.F.: Sketch of the Analytic Engine. Invented by Charles Babbage. Bibliothèque Universelle de Genève. Translated by Lada Ada Lovelace (1842)
[Bec:00]	Extreme Programming Explained. Kent Beck. Addison-Wesley (2000)
[Ber:99]	Berkeley, G.: Principles of Human Knowledge. Oxford University Press, Oxford (1999) (Originally published in 1710)
[BL:00]	Berners-Lee, T.: Weaving the Web. Collins Book, New York (2000)
[Boe:88]	Boehm, B.: A spiral model for software development and enhancement. Computer **21**, 61–72 (1988)
[BoM:79]	Program Verification. R.S. Boyer and J.S. Moore. Journal of Automated Reasoning **1**, (1985)
[Boo:48]	Boole, G.: The calculus of logic. Camb. Dublin Math. J. **III**, 183–198 (1848)
[Boo:58]	Boole, G.: An Investigation into the Laws of Thought. Dover Publications, New York (1958) (First published in 1854)
[Brk:75]	Brooks, F.: The Mythical Man Month. Addison Wesley, Reading (1975)
[Brk:86]	Brooks, F.: No silver bullet. Essence and accidents of software engineering. In: Information Processing. Elsevier, Amsterdam (1986)
[Bro:90]	Rational for the development of the U.K. Defence Standards for Safety Critical Software. Compass Conference (1990)
[Bus:45]	Bush, V.: As we may think. The Atlantic Monthly **176**(1), 101–108 (1945)
[ChR:02]	Chesbrough, H., Rosenbloom, R.: The role of the business model in capturing value from innovation: Evidence from Xerox Corporation's Technology spin-off companies. Ind. Corp. Chang. **11**(3), 529–555 (2002)
[CKS:11]	Chrissis, M.B., Konrad, M., Shrum, S.: CMMI. Guidelines for Process Integration and Product Improvement. SEI Series in Software Engineering, 3rd edn. Addison Wesley, Reading (2011)
[Cod:70]	Codd, E.F.: A relational model of data for large shared data banks. Commun. ACM **13**(6), 377–387 (1970)
[Crs:79]	Crosby, P.: Quality Is Free. The Art of Making Quality Certain. McGraw Hill, New York (1979)
[Dem:86]	Deming, W.E.: Out of Crisis. MIT Press, Cambridge (1986)
[Des:99]	Descartes, R.: Discourse on Method and Meditations on First Philosophy, 4th edn. Translated by Donald Cress. Hackett Publishing Company, Indianapolis (1999)
[Dij:68]	Dijkstra, E.W.: Go to statement considered harmful. Commun. ACM **51**, 7–9 (1968)

[Fag:76] Fagan, M.: Design and code inspections to reduce errors in software development. IBM Syst. J. **15**(3), 182–211 (1976)

[Fen:95] Fenton, N.: Software Metrics: A Rigorous Approach. Thompson Computer Press, London (1995)

[Flo:67] Floyd, R.: Assigning meanings to programs. Proc. Symp. Appl. Math. **19**, 19–32 (1967)

[Ger:94] Experience with formal methods in critical systems. Susan Gerhart, Dan Creighton and Ted Ralston. IEEE Software (Jan 1994)

[Glb:94] Gilb, T., Graham, D.: Software Inspections. Addison Wesley, Reading (1994)

[Glb:76] Gilb, T.: Software Metrics. Winthrop Publishers, Inc., Cambridge (1976)

[Goe:31] Goedel, K.: Undecidable Propositions in Arithmetic. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I. Monatshefte für Mathematik und Physik **38**, 173–198 (1931)

[Gri:81] Gries, D.: The Science of Programming. Springer, Berlin (1981)

[HB:95] Hinchey, M., Bowen, J. (eds.): Applications of Formal Methods. Prentice Hall International Series in Computer Science. Prentice Hall, New York (1995)

[Hea:56] Euclid.: The Thirteen Books of the Elements, vol. 1, Translated by Sir Thomas Heath. Dover Publications, New York (First published in 1925) (1956)

[Hor:69] Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM **12** (10), 576–585 (1969)

[HoU:79] John E. Hopcroft and Jeffrey D. Ullman: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)

[Hum:06] Hume, D.: An Enquiry concerning Human Understanding. Digireads.com, Stilwell (2006) (Originally published in 1748)

[Hum:89] Humphry, W.: Managing the Software Process. Addison Wesley, Reading (1989)

[Jac:99a] The Unified Modelling Language. Reference Manual. (2nd. Edition). Grady Booch, James Rumbaugh and Ivar Jacobson. Addison-Wesley (2004)

[Jur:00] Juran, J.: Juran's Quality Handbook, 5th edn. McGraw Hill, New York (2000)

[KaC:74] Kahn, B., Cerf, V.: Protocol for packet network interconnections. IEEE Trans. Commun. Technol. **22**, 637–648 (1974)

[Kan:03] Critique of Pure Reason. Immanuel Kant. Dover Publications (2003)

[KeR:78] Kernighan, B., Ritchie, D.: The C Programming Language. Prentice Hall Software Series, 1st edn. Prentice Hall, Englewood Cliffs (1978)

[Ker:81] Kernighan, B.: Why Pascal Is Not My Favourite Language. AT&T Bell Laboratories, Murray Hill (1981)

[Knu:11] Knuth, D.: The Art of Computer Programming, 4 Volumes 1 to 4A. Addison-Wesley Longman (2011)

[Kuh:70] Kuhn, T.: The Structure of Scientific Revolutions. University of Chicago Press, Chicago (1970)

[MaP:02] Malmsten, E., Portanger, E.: Boo Hoo. $135 Million, 18 Months. A Dot.Com Story from Concept to Catastrophe. Arrow (2002)

[Mc:59] McCarthy, J.: Programs with common sense. In: Proceedings of the Teddington Conference on the Mechanization of Thought Processes. Her Majesty's Stationery Office, London (1959)

[McH:85] McHale, D.: Boole. Cork University Press, Cork (1985)

[Men:87] Mendelson, E.: Introduction to Mathematical Logic. Wadsworth and Cole/Brook, Advanced Books & Software, Monterey (1987)

[Mor:65] Moore, G.: Cramming more components onto integrated circuits. Elect. Mag. **38**, 14–117 (1965)

[Nau:60] Naur, P. (eds.): Report on the algorithmic language: ALGOL 60. Commun. ACM. **3**(5), 299–314 (1960)

[NeS:56] Newell, A., Simon, H.: The logic theory machine. IRE Trans. Inf. Theory **2**, 61–79 (1956)

[Nyq:24] Nyquist, H.: Certain Factors affecting telegraph speed. Bell Syst. Tech. J. **3**, 324–346 (1924)

[ORg:02] O'Regan, G.: A Practical Approach to Software Quality. Springer, New York (2002)

[ORg:06] O'Regan, G.: Mathematical Approaches to Software Quality. Springer, London (2006)

[ORg:10] O'Regan, G.: Introduction to Software Process Improvement. Springer, New York (2010)

[Pac:96] Packard, D.: The HP Way. How Bill Hewlett and I Built Our Company. Collins Press, New York (1996)

[Plo:81] Plotkin, G.: A structural approach to operational semantics. Technical Report DAIM FN-19. Computer Science Department. Aarhus University, Denmark (1981)

[Por:98] Porter, M.E.: Competitive Advantage. Creating and Sustaining Superior Performance. Free Press, New York (1998)

[Pri:59] de Solla Price, D.J.: An ancient Greek computer. Sci. Am. **200**, 60–67 (1959)

[Res:84] Resnikoff, H.L., Wells, R.O.: Mathematics in Civilisation. Dover Publications, New York (1984)

[Roy:70] Managing the development of large software systems. Winton Royce. Proceedings of IEEE WESTCON(26) (1970)

[RuW:10] Russell, B., Whitehead, A.: Principia Mathematica. Cambridge University Press, Cambridge (1910)

[Sea:80] Searle, J.: Minds, brains, and programs. Behav. Brain Sci. **3**, 417–457 (1980)

[Sha:37] Shannon, C.: A symbolic analysis of relay and switching circuits. Masters Thesis, Massachusetts Institute of Technology (1937)

[Sha:48] Shannon, C.: A mathematical theory of communication. Bell Syst. Tech. J. **27**, 379–423 (1948)

[Sha:49] Shannon, C.E.: Communication theory of secrecy systems. Bell Syst. Tech. J. **28**(4), 656–715 (1949)

[Smi:23] Smith, D.E.: History of Mathematics, vol. 1. Dover Publications, New York (1923)

[Spi:92] Spivey, J.M.: The Z Notation. A Reference Manual. Prentice Hall International Series in Computer Science. Prentice Hall, Englewood Cliffs (1992)

[Std:99] Standish Group Research Note.: Estimating: art or science. Featuring Morotz Cost Expert (1999)

[Tie:91] The Evolution of Def. Standard 00-55 and 00-56. An Intensification of the Formal Methods Debate in the U.K. Margaret Tierney. Research Centre for Social Sciences, University of Edinburgh (1991)

[Tur:50] Turing, A.: Computing, machinery and intelligence. Mind **49**, 433–460 (1950)

[Turn:85] Turner, M.D.: Proceedings IFIP Conference, Nancy France, Springer LNCS (201) (1985)

[VN:32] von Neumann, J.: Mathematische Grundlagen der Quantenmechan (The Mathematical Foundations of Quantum Mechanics). Springer, Berlin (1932)

[VN:45] von Neumann, J.: First Draft of a Report on the EDVAC. University of Pennsylvania (1945)

[Wol:02] Wolfram, S.: A New Kind of Science. Wolfram Media, Champaign (2002)

Index