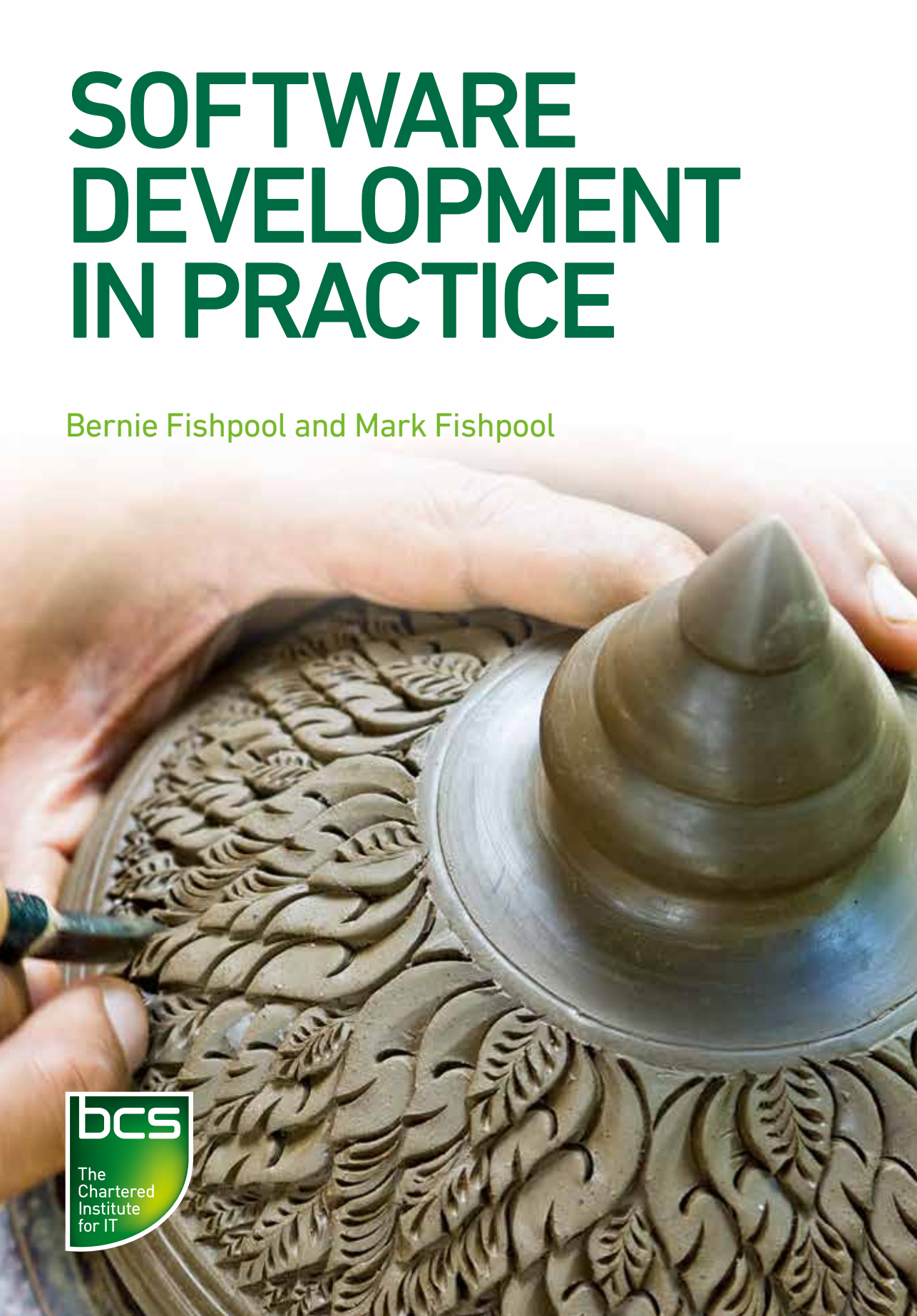
SOFTWARE DEVELOPMENT IN PRACTICE

Bernie Fishpool and Mark Fishpool



bcs
The Chartered Institute for IT

'A remarkable book that provides a unique perspective on modern software development. A distinctive and unusual feature is the way modern software development principles are explained holistically in terms of all project activities. The focus on key employment skills and knowledge also makes it a must read for aspiring developers.'

***Chris Beaumont PhD FBCS FHEA**, Chair of Examiners, NCC Education*

'I wish this book had been around when I was starting out 30 years ago. It's a manual for all aspects of software development and the scope of the role in business, rather than focusing on being a "coder". I particularly like the fact it includes client aspects, which are usually forgotten!'

***Andy Doyle**, Director, Nice Group (SW) Ltd*

'As a leader of many software development teams, this book will be indispensable to modern developers and managers alike. It will not teach you how to write .net, but will help when someone who does tries to bamboozle you with jargon. It is brilliantly written and easy to digest.'

***Paul Leonard**, Group Technology & Infrastructure Manager, DCC plc*

'A comprehensive, practical overview of what awaits you in the real world of professional software development.'

***Karl Beecher**, Author of* Computational Thinking

'*Software Development in Practice* takes the guesswork out of your journey into tech. From term definitions to Agile practices and clean code tips, this book is my go-to resource for anyone breaking into the tech industry. I especially appreciate the emphasis on communication, collaboration and user experience.'

***Sjoukje Ijlstra**, Software Engineer, JP Morgan*

'There are many books which describe various technical and theoretical aspects of software development. However, few describe what's actually involved in day-to-day software development. This book is one of those few and should be of real interest to prospective and early-career software developers.'

***Dr Patrick Hill**, R&D Director, QPC Ltd*

'As a security researcher and advocate for embedding security in the software development process, it is enlightening to see this book dedicate some detailed coverage to consider use of defensive coding techniques, GDPR from a developer's point of view, and specific vulnerabilities and associated mitigations taken direct from the OWASP Top10.'

***Adrian Winckles**, Director of Cyber and Networking, Anglia Ruskin University*

'A great book for those thinking of working or progressing in the commercial software development industry. The book gives insight into working practices, identifying positives and negatives to each of them. Deliberately avoiding specific programming languages (other than to explain some points), the book will be a perfect addition for any dev team in any development environment.'

***Martin Thorne**, Technical Director, Montpellier Integrated*

'This book provides the framework to apply knowledge of how to code into the real world of being a software developer. It is the theory and thought processes that you can't learn without doing the job first – until now! If you're considering a career path in software development this book should be the first port of call on your journey.'

***Kieran Purdie**, Pro AV Channel Manager & Business Development / Technical Manager, NETGEAR Business, UK & Ireland*

'If you want a guide on what you need to do to become a fantastic software developer, then this book is for you. The book's in-depth topic coverage will provide you with all the tools and information you will need to succeed in the software development Industry.'

***Anthony Davis**, Senior Manager Platform Engineering, Sixt*

'IT now permeates almost every area of business. In an environment where the pace is ever increasing, it is essential for those aspiring to work as a software developer to gain knowledge, skills and experience in many areas. *Software Development in Practice* covers the areas to master to become a productive member of a software development team.'

Chris Galley FBCS CITP

SOFTWARE DEVELOPMENT IN PRACTICE

BCS, THE CHARTERED INSTITUTE FOR IT

BCS, The Chartered Institute for IT, is committed to making IT good for society. We use the power of our network to bring about positive, tangible change. We champion the global IT profession and the interests of individuals, engaged in that profession, for the benefit of all.

Exchanging IT expertise and knowledge
The Institute fosters links between experts from industry, academia and business to promote new thinking, education and knowledge sharing.

Supporting practitioners
Through continuing professional development and a series of respected IT qualifications, the Institute seeks to promote professional practice tuned to the demands of business. It provides practical support and information services to its members and volunteer communities around the world.

Setting standards and frameworks
The Institute collaborates with government, industry and relevant bodies to establish good working practices, codes of conduct, skills frameworks and common standards. It also offers a range of consultancy services to employers to help them adopt best practice.

Become a member
Over 70,000 people including students, teachers, professionals and practitioners enjoy the benefits of BCS membership. These include access to an international community, invitations to a roster of local and national events, career development tools and a quarterly thought-leadership magazine. Visit www.bcs.org/membership to find out more.

Further information
BCS, The Chartered Institute for IT,
3 Newbridge Square,
Swindon, SN1 1BY, United Kingdom.
T +44 (0) 1793 417 417
(Monday to Friday, 09:00 to 17:00 UK time)
www.bcs.org/contact
http://shop.bcs.org/

SOFTWARE DEVELOPMENT IN PRACTICE

Bernie Fishpool and Mark Fishpool

bcs
The Chartered Institute for IT

Published by BCS Learning & Development Ltd, a wholly owned subsidiary of BCS, The Chartered Institute for IT, 3 Newbridge Square, Swindon, SN1 1BY, UK.
www.bcs.org

Paperback ISBN: 978-1-78017-497-6
PDF ISBN: 978-1-78017-498-3
ePUB ISBN: 978-1-78017-499-0
Kindle ISBN: 978-1-78017-500-3

British Cataloguing in Publication Data.
A CIP catalogue record for this book is available at the British Library.

Publisher's acknowledgements
Reviewers: Patrick Hill, Karl Beecher
Publisher: Ian Borthwick
Commissioning editor: Rebecca Youé
Production manager: Florence Leroy
Project manager: Hazel Bird
Copy-editor: Hazel Bird
Proofreader: Barbara Eastman
Indexer: Sally Roots
Cover design: Alex Wright
Cover image: Shutterstock/nofilm2011
Typeset by Lapiz Digital Services, Chennai, India.

CONTENTS

LIST OF FIGURES AND TABLES

FIGURES

TABLES

AUTHORS

Bernie Fishpool is a consultant subject matter expert and author of vocational computing and IT texts and learning resources, who taught programming, systems analysis, project management and personal skills development in post-16 education for more than 20 years. On leaving teaching in 2007, she worked for Edexcel as a curriculum development manager. Bernie led the development of the AQA Tech-levels in IT between 2014 and 2016 as a sector strategist for IT and computing, and has written many Level 2 and Level 3 BTEC IT units for Pearson Edexcel over the years. More recently Bernie was a member of the team that developed the BCS Trailblazer Apprenticeships, and she has recently worked on the BCS Essential Digital Skills Qualification (EDSQ). She continues to develop teaching and learning resources.

Mark Fishpool has taught various computing subjects from Year 7 to degree level in schools and colleges since the age of 19. He has lead-written nationally recognised BTEC and AQA computing qualifications and was head of school for computing and Centre of Vocational Excellence manager at Gloucestershire College for over seven successful years before returning to the commercial sector, specialising as a senior full-stack developer. He has recently come back to training and education as senior technical learning consultant (cyber) at QA Ltd, specialising in Python, C and Linux. Somehow in the middle of this 30-year career he's also written 11 books, mostly with his wife, Bernie.

ABBREVIATIONS

ACID	atomicity, consistency, isolation and durability
API	application programming interface
AppSec	application security
AWS	Amazon Web Services
BIDMAS	brackets, indices, division, multiplication, addition, subtraction
BODMAS	brackets, orders, division, multiplication, addition, subtraction
CIW	Certified Internet Web Professional
CLI	command line interface
CMM	Capability Maturity Model
CMS	content management system
CPD	continuing professional development
CPU	central processing unit
CRUD	create, read, update, delete
CSS	Cascading Style Sheets
CSV	Comma-Separated Values
DCL	Data Control Language
DDaT	Digital, Data and Technology Professions
DDL	Data Definition Language
DevOps	developer and operations
DevSecOps	developer, security and operations
DFD	data flow diagram
DML	Data Manipulation Language
DQL	Data Query Language
DRY	don't repeat yourself
EAFP	easier to ask for forgiveness than it is to get permission
EC2	Elastic Compute Cloud (Amazon)
EE	Enterprise Edition (Java)
ERD	entity relationship diagram
EU	European Union
FAQs	frequently asked questions

FE	further education
FIFO	first in, first out
FTP	File Transfer Protocol
GDPR	General Data Protection Regulation
GUI	graphical user interface
HNC	Higher National Certificate
HND	Higher National Diploma
HTML	Hypertext Mark-up Language
HTTP	Hypertext Transfer Protocol
I/O	input and output
IaaS	infrastructure as a service
IAM	Identity and Access Management (Amazon)
IDE	integrated development environment
IoT	Internet of Things
JSON	JavaScript Object Notation
LAMP	Linux, Apache, MySQL and PHP
LBYL	look before you leap
LIFO	last in, first out
LMGTFY!	let me Google that for you!
MAMP	Mac, Apache, MySQL and PHP
MCA	Microsoft Certified Associate
MCSA	Microsoft Certified System Administrator
MCSD	Microsoft Certified Solutions Developer
MCSE	Microsoft Certified Solutions Expert
MVC	model-view-controller
MVP	minimum viable product
NoSQL	Not only SQL
OCA	Oracle Certified Associate
OCM	Oracle Certified Master
OCP	Oracle Certified Professional
OOP	object-oriented programming
OS	operating system
OTS	off-the-shelf
OWASP	Open Web Application Security Project
PaaS	platform as a service
PCAP	Certified Associate in Python Programming
PCEP	Certified Entry-Level Python Programmer
PCPP	Certified Professional in Python Programming
PEP8	Python Enhancement Proposal 8

PYPL	Popularity of Programming Language
QAT	quality assurance testing
RAD	rapid application development
RAM	random access memory
RDBMS	relational database management system
RESTful	representational state transfer
ROI	return on investment
SaaS	software as a service
SDLC	software development lifecycle
SE	Standard Edition (Java)
SFIA	Skills Framework for the Information Age
SHA1	Secure Hash Algorithm 1
SMS	short message service
SQL	Structured Query Language
SRP	single responsibility principle
TDD	test-driven development
UI	user interface
UML	Unified Modelling Language
UX	user experience
VCS	version control system
VM	virtual machine
WAMP	Windows, Apache, MySQL and PHP
WET	write everything twice (or we enjoy typing)
XML	Extensible Mark-up Language
XP	extreme programming
XXE	XML external entity
YAML	YAML Ain't Mark-up Language

PREFACE

'A software engineer, a developer, and coder walk into a bar.
"Here come the programmers!", says the bartender.'

– Kim (2018)

This book covers good practice and gives you some things to consider no matter where you are in the software development lifecycle.

From a historical perspective, programmers traditionally wrote code – that was their job. They didn't always get involved in analysis or testing, and they certainly wouldn't have been expected to have in-depth knowledge of hardware, networking or databases. That's perhaps a gross simplification, but the kernel of truth remains: software development could be a very isolated – and insular – activity.

The reality today is that for most software developers their job is **not just about coding**. They are expected to have a much wider field of expertise and almost certainly have an extensive list of essential and desirable IT skills, whether these relate to industry-trending code frameworks, modern methodologies, or experience with particular operating systems and popular IDE (integrated development environment). Having a working knowledge of a single programming language just doesn't cut the mustard any more.

For example, a commercial online organisation might require you to demonstrate 'full-stack' web development skills. This would mean knowing the principles of data design, database development, and server- and client-side interaction (e.g. JavaScript); HTML5 (Hypertext Mark-up Language version 5) and CSS3 (Cascading Style Sheets version 3); email and FTP (File Transfer Protocol) server integration; and much more. On the other hand, if you were developing for custom hardware, you might need to understand different signal types in Linux and couple this with advanced low-level C skills. Although both examples would require the skills of a software developer, the roles themselves, the underpinning knowledge and the daily challenges would be quite different.

What's clear is that the role of a software developer is a moving feast, and individuals need to continually improve and adapt their skills over time. This is especially the case for contract programmers as they move between commercially diverse projects.

In this role you'll find that learning never stops. To fill gaps in your own skill set, you may need to take additional courses or gain new professional qualifications to help you to prepare for this type of role. Above all, in these information-sensitive times, you should care about data protection principles and the role of cybersecurity and defensive coding.

A push in the late 1980s led the industry to embrace the notion of software engineering principles rather than simply using the term 'programmer' to define development, and this has now become the norm. The profession has become somewhat more

formalised, more professional and far less 'wild west' – although, in some development environments, that pioneering spirit still plays a crucial part in thinking outside the proverbial box and creating amazing innovations.

Despite the passing of time, key concepts remain important: generality, consistency, incremental development, anticipation of change, abstraction, modularity, and the concepts 'separation of concerns', 'the single responsibility principle' and 'don't repeat yourself'. While rapidly evolving technology produces improved frameworks and encourages better workflows that enable teams to seamlessly share development assets, the underlying challenge of producing robust, reliable and – increasingly important – secure solutions within (typically) tight timescales steadfastly remains.

Many graduate programmers leave education with a sound theoretical understanding, having been taught using a series of tried-and-tested problems that can be resolved using well-chosen tools and techniques in a prescribed way to produce sensible (and somewhat predictable) solutions. Unfortunately, the real world is much messier than this, and few problems encountered 'in the wild' will decompose quite so neatly. This doesn't mean the textbooks should be thrown out or that the educational journey was not worthwhile (far from it), but there should be an acceptance that there is always going to be a great deal still to learn. And, of course, this is no bad thing.

For many new developers, consideration of the well-established software development lifecycle (see Chapter 4) can be a good place to start because it instils structure and process. However, as you will see, there are many different approaches and development models in vogue at any particular time, as they go in and out of fashion or are adapted and remixed for the next generation.

This book will help you to explore the workings of software development and will furnish you with the tools and understanding to make informed decisions as a software developer.

Ultimately, software development should be based on a series of fundamental principles that help developers to successfully navigate software projects. Which ones you choose to follow are up to you – software development is a broad church, and **everyone** is welcome.

WHO THIS BOOK IS FOR

This book is intended for those who aspire to work in a software development role within the IT sector.

It will be of interest to you if you are thinking of starting a software development career or have completed initial education and training programmes, such as A-levels, a Pearson BTEC, T-Levels or a Level 3 apprenticeship and are contemplating your next step. Of course, it will also be relevant to those who have worked for a while in a related or unrelated sector and are simply contemplating a career change.

Whatever your reason for considering a career in software development, you will need to understand the context of the profession, explore the end-to-end process involved,

and get an idea of the good and bad practices, the tools and techniques, and the myriad knotty problems and rock-hard challenges that lie ahead.

Although written from an insider's perspective, this book has (hopefully) not lost the outsider's excitement of looking through a window into a world that appears to be tremendous fun, nose pressed against the glass, wondering how best to get involved. So, whether you consider yourself a budding software engineer, a developer or just a coder – welcome, programmers and aspiring programmers all!

1 GETTING STARTED IN SOFTWARE DEVELOPMENT

'If you think it's expensive to hire a professional, wait until you hire an amateur.'
– Red Adair

'Whether you want to uncover the secrets of the universe, or you just want to pursue a career in the 21st century, basic computer programming is an essential skill to learn.'
– Stephen Hawking

The aim of this chapter is to consider some of the ways in which people get started in the software development industry. With so many potential entry points, it is never too late to think about a role in software development, regardless of your background.

ENTRY POINTS INTO A SOFTWARE DEVELOPMENT ROLE

Malcolm Gladwell is a well-known journalist, author and public speaker. One of his most-quoted books, *Outliers: The Story of Success* (2008), includes a reference to what he calls the '10,000-hour rule'.

Simply put, this rule suggests that achieving success in any field requires 10,000 hours of practising related tasks. That works out at about 20 hours of work a week for 10 years. Only the core working hours are included – the times when your skills are focused and being ever more finely tuned by experiential learning.

Does this represent a true, aspirational metric for successful software development practice? Perhaps.

The IT sector is large, somewhat diverse and constantly growing, and demand for software developers is increasing as content requirements soar, particularly now that everyone has a supercomputer in their pocket disguised as a mobile phone.

The first generation of programmers, who learned programming using primitive 8-bit microcomputers back in the 1970s, is now heading towards well-earned retirement and, as you've probably heard, nature abhors a vacuum. Of course, software development project teams also hate empty developer chairs. This is where you come in!

There are multiple entry points into a software development role, and as such there is no guaranteed 'this always works' career plan. However, there are obvious things you can do to improve your appeal.

Getting started: age is no barrier

The first thing to understand is that, just like in any other occupation, developers come in all shapes and sizes. As such, entry points can occur at different ages. For example, junior developers may begin their careers at the age of 16, 18, 20, 30 or even beyond, depending on their life choices, educational background and relevant work history.

The following information will give you some ideas on how to begin your career in software development, depending on your situation.

Under 18s

If you are aged under 18, you are entitled to a free education, which can be studied full time in sixth form or at a further education (FE) college. You have the opportunity to study a combination of A-levels, T-levels and vocational options, all of which can be studied at Level 3 and can lead you into employment and/or higher education.

A-Levels

Most of the larger awarding organisations offer A-levels in computer science or computer science and IT. These focus on mathematical concepts, communication technologies, computer hardware and architecture, software, security, data modelling, algorithms, problem-solving, computer programming and file handling.

While this option might seem attractive because the content is broad, it is unlikely that you would be able to go directly into a job from an A-level programme because A-levels are classified as academic and are more theoretical than practical. Additionally, modern A-levels focus on examined assessment at the end of the course. With many development jobs now requiring candidates to undertake a practical programming task as part of the recruitment process, the lack of extensive practical coding in an A-level programme may make successfully completing the task a significant challenge.

If you are considering pursuing this option with a view to university progression, you should also consider studying mathematics at AS or A-level, as some higher education programmes now require this in addition to the A-level, especially for degree courses such as programming or software development.

Vocational qualifications

If you enjoy computing but prefer to be assessed with more coursework and fewer exams, the alternative to an A-level programme is to study vocational qualifications from a range of awarding organisations.

Pearson BTECs, City & Guilds and OCR Cambridge Technicals programmes have been developed in line with the National Occupational Standards for software developers (NOS n.d.). This means that their content is more directly targeted to the job with a much reduced focus on architecture, hardware and so on. Instead, there is much more emphasis on the software development lifecycle phases, including understanding the problem, designing a solution, programming, and then testing, implementing and maintaining a product. In this instance, a single course developed to meet these standards is equivalent to two or three A-levels and it will prepare you for either the workplace or progression into higher education.

Both A-levels and vocational qualifications are publicly funded for under 18s, which means that you can study without any cost to you or any requirement to repay the fees. In some circumstances, these courses can also be studied in FE colleges by students over 18, funded by a student loan, but this must be repaid when you are employed.

Apprenticeships

Apprenticeships are available to anybody over the age of 16. Level 3 is on the same stratum as an A-level, and in some contexts you can then progress to a Level 4 apprenticeship after completing your full-time studies at Level 3.

Apprenticeships are studied while working for an employer, often on day release, which means one day studying at a university or FE college or with a training provider, and four days at work learning on the job. The apprenticeship may well be a Trailblazer Apprenticeship. These apprenticeships were developed in partnership with employers to replace a range of outdated apprenticeships with a new format with more rigour, focused learning and a single End Point Assessment.

> Because of the popularity of apprenticeships as an alternative to university, the government has created a website that serves as a central point for those looking for an apprenticeship opportunity: https://www.apprenticeships.gov.uk.

Whereas in higher education you have to cover the whole cost of the fees yourself (usually through a student loan), in an apprenticeship your employer will make a contribution to your education. You may, however, be expected to contribute to your own development (e.g. by attending courses and seminars in your own time, or even buying textbooks).

> The National Apprenticeship Service has released a comprehensive list of all IT-related apprenticeships included under the heading 'Digital': see https://tinyurl.com/y3c95uaw.

Let's examine the different levels of apprenticeship available in more detail.

Level 2 (intermediate)

The entry point for apprenticeships is Level 2, which is usually referred to as 'intermediate'. This level is judged to be equivalent to five GCSEs at grades 4 to 9 (previously C to A*), with grade 4 considered a standard pass.

The current qualifications are still relatively generalist and cover a large area of study, as many of the fundamentals need to be learned before moving on to a Level 3 apprenticeship.

Level 3 (advanced)

Level 3 is equivalent to three A-levels at A* to E or a vocational Extended Diploma at pass, merit, distinction or distinction*. There are currently a number of IT apprenticeships that can be studied, including Cyber Security Technologist, Data Analyst, Digital and Technology Solution Specialist, Infrastructure Technician, Network Engineer, Unified Communications Technician and the Level 3 most suited to those wishing to pursue software development: Software Development Technician.

Successful completion of the Software Development Technician qualification prepares learners for junior roles in various fields, including:

- software development;
- mobile application development;
- web development;
- games development;
- application development.

In addition, there may be opportunities as junior or assistant programmers or automated test developers.

The Software Development Technician apprenticeship requires learners to focus on three key areas:

- knowledge and understanding;
- competence;
- underpinning skills, attitudes and behaviours.

The expectation is that learners will gain technical knowledge in areas such as security, data, analysis, quality and problem-solving, but also develop the ability to apply knowledge in an organisational or business context, together with an appreciation of the attitudes and behaviours that employers require. Employers expect their employees to be able to problem-solve, act professionally, work independently and, importantly, show that they can use their initiative. In addition, good communication skills are highly sought-after when you consider that a developer must be able to communicate effectively (and professionally) with clients and users, as well as their managers and fellow team members. The ability to work as part of an effective team is essential as very few development opportunities will ever be undertaken by an individual working alone.

The entry requirement for most Level 3 IT and digital qualifications is five GCSEs at levels 5 to 9, including English and mathematics.

T-levels

In addition to A-levels, vocational levels and apprenticeships, there are T-levels. These are designed to become the technical equivalent of A-levels, and the first T-levels will be available in schools and colleges from September 2020. In the first instance, these qualifications will not be available in every institution and a number of organisations across the UK have been selected to pilot the new schemes.

Based on the same standards as apprenticeships, T-levels will contain both a classroom element and a work experience or industrial placement. However, the key difference between apprenticeships and T-levels will be the amount of

classroom study. In an apprenticeship, candidates spend most of their time in the workplace, undertaking less classroom study. In contrast, T-level candidates will spend most of their time in the classroom while also undertaking significant work-based activity (approximately 315 hours).

The first T-level for IT will be called Digital Production, Design and Development.

Level 4 (higher)

A Level 4 apprenticeship (also called a higher apprenticeship) can be attractive to those who wish to continue their studies but who do not want to access full-time higher education.

At this level there are significantly more apprenticeship options, but the two that are important in a software development context are Software Developer and Software Tester. This is not to suggest that software developers do not carry out testing activities (they most certainly do) – software developers carry out formative testing, eradicating errors as code is developed, and they also carry out summative testing at predefined points in development. However, there are new roles in industry for those who wish to focus on the testing phase of the development lifecycle in a summative sense, using a wide range of testing tools and techniques to improve test development outcomes before implementation and hand-over to the client.

The entry requirement for Level 4 IT and digital apprenticeships is three A-levels, a vocational Extended Diploma or a Level 3 apprenticeship, but again there is an emphasis on English and mathematics if these have not been achieved already.

After a Level 4 apprenticeship, candidates can progress to Levels 5, 6 and 7, where their technical learning is further advanced and combined with management.

Employer role and commitment

A relatively recent development in the world of apprenticeships is that there is now a greater role for the employer, who makes a financial contribution to the apprenticeship in addition to allowing the apprentice time to study. This makes the modern apprenticeship much more of a partnership between the employer and employee, both of whom make financial and practical commitments to the programme. You should be aware, however, that in some cases the employer may seek to recover some of their financial input from the apprentice if they immediately leave their employment at the end of the apprenticeship to go to another job.

Higher education

With a combination of Higher National Certificate (HNC), Higher National Diploma (HND) and degree programmes available across the UK, there are many potential routes through the higher education landscape.

A modern HNC is the equivalent of the first year of a degree, the HND is equivalent to the second year and the final top-up year of study makes up the full degree. HNC and

HND programmes can be studied at university or an FE college, although for the degree top-up you would probably need to go to university.

Each level can be studied either full time or part time, so you may be able to pursue these qualifications while you continue to work in your current role. You may even find that your employer will pay for, or make a contribution towards, your studies, particularly if they will be of direct benefit to the employer by making you more skilled.

Most higher education courses require you to take a student loan, which you will be expected to repay when you are employed.

Professional and vendor qualifications

Professional and vendor qualifications are also available, from a range of organisations, such as BCS, The Chartered Institute for IT, Microsoft, Oracle and Red Hat.

In recent years, professional bodies that represent the sector as a whole have developed qualifications with the heavy involvement of employer partners. BCS, for example, offers the Practitioner Certificate in Systems Development Essentials, which includes, on successful completion, a level of BCS membership. Professional qualifications such as this are not linked to a particular product but deliver a wide curriculum to candidates who already have technical skills and expertise but who lack the ability to apply their existing knowledge in a wider context.

A vendor qualification is one that is linked to a particular software or hardware component. Vendor qualifications have existed for many years, with Microsoft and Cisco being early adopters of the concept. Their vendor qualifications were specifically linked to their products and these certifications became valuable currency for candidates applying for roles in those areas. Chapter 17 examines these types of qualification in more detail.

You will study with a training provider or possibly at an FE college. Some can be studied remotely if there are online options.

Professional certification can be a very useful way of making you stand out in comparison to other job candidates. Typically, the first vendor or professional qualification has no specific entry requirements and is equally accessible to those with qualifications and those who do not have qualifications but have extensive industrial experience. Therefore, these qualifications can be an opportunity to get professional recognition for something you already know!

These courses are not usually publicly funded, which means that you will have to pay for them yourself or, in some circumstances, your employer may be prepared to pay. These courses are usually shorter – anything from a few weeks to a number of months – and are unlikely to require you to study for a full year.

With the cost of higher education still making qualifications at Level 4 and above inaccessible for some, apprenticeships and professional qualifications have seen a large surge in learner numbers in recent years. These qualifications have industry recognition and are increasingly popular for those who do not wish to face large student debts.

Already in the workplace?

Progressing from junior software development to a more senior role (such as senior or lead developer) can take considerable time and patience. Often, employers assess eligibility for promotion based on involvement in and delivery of successful projects. However, there are some simple steps to follow when opportunities arise:

- If you are offered training in new technologies or techniques, **always** do so, especially if it is an industry-recognised certification.
- If you are given the opportunity to stretch into new areas, generally it is a good idea to accept. Any learning, especially that which is conducted 'on the job', can be very rewarding.
- Pair programming techniques (see Chapter 7) are beneficial, particularly when you can work alongside a more experienced developer.
- Keep your programming skills up to date, and embrace new concepts and ideas. Working on a pet project outside work hours is a fun way to hone your skills without extra pressure.
- Grow with new responsibilities – particularly in terms of leading teams and undertaking different roles.
- Above all, enjoy challenges and solve them as best you can!

SOFTWARE DEVELOPER SKILLS

In the context of software development, across the whole lifecycle, the skills needed can generally be divided into two types – transferable skills and development skills. These include (but are not limited to) the skills below.

Transferable skills

These are skills which you develop during your working life that can be transferred and used in different contexts.

- **Ability to work individually and as part of a team:** most software development is based around team activity unless the project is small enough to be undertaken by an individual, but this is increasingly unlikely given the relative complexity (and technological diversity) of modern projects.
- **Active listening skills:** this involves listening to the opinions of your team and the needs of your client, and concentrating on what is being communicated and the way it is being communicated, as non-verbal communication often adds important context.
- **Emotional intelligence:** this means being co-operative and compassionate when dealing with peers, and being constructive and supportive in your contributions, especially when giving feedback to others.

- **Negotiation skills:** you must be able to have discussions and, more importantly, make compromises to agree a resolution that suits both parties. This is not always as easy as it may sound, particularly when you are trying to balance the needs of a large group of stakeholders and a range of business drivers.
- **Written and verbal communication skills:** you must be clear and (preferably) concise both in writing and in discussions and presentations to your client. To achieve this, you must ensure that you are fully prepared by gaining a comprehensive understanding of the business needs, known constraints and technical challenges that lie ahead. Periodically immersing yourself in a new business sector is usually also a good idea.
- **Attention to detail:** ensuring documents, emails, discussions and so on are complete and correct will help to make sure that every member of the team (and the client) has a full understanding of the development process and current progress, targets and goals. Missing out key details can have catastrophic effects in a development process.
- **Problem-solving:** not everyone is a natural problem-solver but most of us have valuable insights and contributions we can make as part of a larger design or implementation process. The key is knowing when these are appropriate (or not) and how to effectively express your ideas. Formalised problem-solving provides the developer with a guaranteed step-by-step approach: (1) understanding the problem, (2) planning a solution (considering alternatives) by breaking it down into smaller problems and (3) finally implementing the chosen solution. Everything else (languages, frameworks, libraries etc.) is just the **tools** you use to support this activity and these will undoubtedly change over time.
- **Project management:** managing software development relies on two key components: effective and realistic planning, and appropriate (thorough) monitoring of activities. A good project manager will ensure that a product is delivered on time and within budget.
- **Trouble-shooting:** unlike problem-solving, which is usually carried out in the context of the development and its deliverables, trouble-shooting requires much faster responses, immediate reaction to issues and an ability to find resolutions quickly.
- **Mentoring skills:** everyone has to start somewhere, so less experienced team members should be supported to help them to build confidence that they are approaching the development in the right way. This will make them better team members for future developments.
- **Good understanding of business context:** this is the most difficult of the transferable skills to acquire and is almost always only achieved with experience. There are, however, some key areas you can investigate to give you at least some theoretical understanding of business context:
  - be able to identify the different functional areas of an organisation;
  - understand what each functional area does;
  - understand what data the functional areas need to operate and what each generates as a result of its activities.

Development-specific skills

Some skills are traditionally associated with the role of a software developer. These include but are not limited to:

- **Analysis skills:** good analysis is based on a person's ability to examine something in detail. Superficial analysis will almost always become a factor that leads to the failure of a software development project. Sometimes analysis is less in-depth due to time constraints, which is why your approach should be planned to ensure that the analysis you undertake is focused and is likely to produce the level of information that you need. Analysis may be needed in software development for various reasons – for example:
  - determine how to access data items from a complex data structure;
  - debug complex algorithms to improve their efficiency;
  - assess benchmarks from code execution to determine which techniques are fastest;
  - make development choices based on research into good coding practice.
- **Programming (language-specific syntax but also a wider understanding of programming constructs, tools and techniques):** experience using the language that will be used in a development project is key. Some would argue that understanding common algorithms is a transferrable skill and that all that differs is the actual language syntax. While that may be true to some degree, you will find differences even between versions of the same language which can slow you down when coding (e.g. changes in the way components work or where you find them within the environment). You must make sure that you are fully oriented within the relevant language and, if it is completely new to you, ensure that you take time to learn key techniques and that you know how to access essential tools. It is also worth considering the evolution of programming languages; for example, coding in **classic** C++ is a very different proposition to doing so in **modern** C++.
- **Structured Query Language (SQL) skills:** as shown in Chapter 9, most applications use database connections as a source of external data, and as such you should be able to perform the basic data operations using the world's most popular database language, for example:
  - SQL Data Definition Language (DDL), which is used for creating databases and tables, altering structures, renaming, dropping (removing the table and its data) and truncating (deleting all data from a table);
  - SQL Data Manipulation Language (DML), which is used for inserting, deleting and updating rows;
  - SQL Data Control Language (DCL), which is used for granting and revoking user privileges to database objects;
  - SQL Data Query Language (DQL), which is used for creating queries using the `select` statement (probably the most important).
- **Hardware skills:** it is always a good idea to have a basic working knowledge of the technology used by your applications – for example, you should understand

how programs use computer memory to enable you to avoid memory errors caused by poor code. Another example is understanding the architecture of the central processing unit (CPU) so that code can be more efficiently written and take advantage of the hardware's natural features.

- **Networking skills:** many applications will be network aware, whether these are traditional desktop applications, mobile applications or commercial web applications. As such, it is useful to have a working knowledge of basic networking principles such as addresses, protocols and sockets.
- **Knowledge of an operating system (OS):** software developers' role is not limited to producing program code; they also have to prepare development environments and deployment environments (i.e. where the code will eventually be run). This very often involves installing software, managing configuration files and working with files (and permissions). Much of this should be done from the operating system's command line interface (CLI) – that is, Windows' command prompt or Linux's shell – rather than the graphical user interface (GUI), although note that on a server the latter may not be available. As such, a working knowledge of the target operating system is crucial.
- **Regular expressions:** often abbreviated to 'regex', this is a popular tool used by experienced developers and supported in most modern programming languages and applications. This intricate and elegant pattern-matching metalanguage can be used to match, extract or substitute data in a string without requiring the use of complex and messy combinations of different string functions.
- **Diagramming skills:** a diagram is often much more useful at explaining existing or new systems than extensive text. There is a host of software applications that you can use, such as Lucidchart, SmartDraw and Draw.io. They are fundamentally very similar in that you work with shapes, arrows and lines to produce a representation of a system or processes within a system.

Flowcharts example

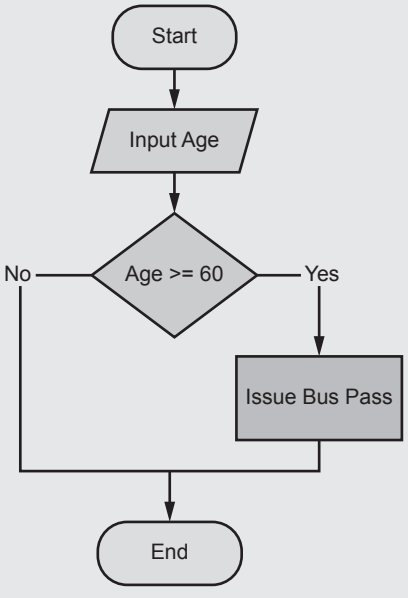
The flowchart in Figure 1.1 was created using Lucidchart. The flowchart represents the simple process of issuing a bus pass to travellers aged 60 or above.

Firstly, the user inputs the age of the traveller. The program then evaluates the age and either issues the bus pass (if the age is correct) or ends the program.

If you investigate different diagramming software you may see variations in the shapes. It is important that whichever version you use, you use it consistently.

You might find it helpful to identify a particular software application and practise using it to generate diagrams of systems and processes.

Figure 1.1 A simple flowchart



- **Design skills:** once you understand the problem to be addressed and the requirements, you may well be able to visualise a solution or components that will contribute towards a solution. Designing a solution as part of a team is often a good way of producing a high-quality product as you can draw on a large range of ideas.
- **Technical writing skills:** these skills enable you to produce a range of important documentation, including:
  - development documentation, which begins with a copy of the development brief and then records details of the investigation, the specification, the design, the implementation and testing activities, and (particularly importantly) the rationale for the decisions made;
  - technical documents that explain the final system as developed to aid future maintenance and upgrades;
  - user documentation to support new users of the software.

Tips for user guides

- Avoid waffle. Users appreciate short, succinct and accurate instructions to help them navigate the software.
- Ensure that you consider users with disabilities by offering the guide in different formats.
- Ideally, test the guide on users during the user-test part of the development process.
- Documentation may be paper-based, electronic or a combination of both.

- **Estimation/costing skills:** these are essential if you are required to prepare a costing or estimate for a client. Keep in mind the following:
  - Ensure that you understand the objectives of the development.
  - You must have a full understanding of the scope of the project (this will ensure that your project does not suffer from scope creep). Scope creep is where the project expands into unexpected areas which were not part of the agreed development (see Chapter 4 for more information on scope creep).
  - Consider what will be needed in terms of both functional requirements (what the software is expected to do) and non-functional requirements (which will also include usability, security, and interoperability with other systems and maintenance).
  - Consider adding additional time to facilitate recovery points for when the development project experiences problems.
  - Consider using project management estimation tools, either to help you to carry out the costing or to confirm the results of a costing you have undertaken by other means.
  - Your skills in this area will improve over time; however, as you are building up experience, draw on the knowledge and experiences of others in your team.
- **Testing:** programs which have not been robustly tested can be disastrous for a number of reasons. The program may fail under load (large number of users) and may become slow. Alternatively, functionality may not work as expected. Testing is about planning, executing, documenting and resolving issues to make sure that the product is fully functioning and meets the needs of the client and the users. This requires a highly structured approach to ensure that the testing is appropriately comprehensive. As a developer you will choose the range of data and the data items that will be used to test the software product. A popular technique is the use of unit testing, which devolves testing responsibility to small parts of the solution (e.g. individual functions which perform a specific task). More on testing can be found in Chapter 10.

INTERVIEW SKILLS

Seeing a suitable vacancy and applying for it are only the first steps. Once you have received an invitation to attend an interview, you need to prepare in advance. Simply turning up and hoping for the best will probably not result in success.

Firstly, don't panic. You will be there on merit, having impressed the interview panel through your CV, covering letter and any other documentation you were required to send. While it is outside the remit of this book to describe good interview technique, we can pull back the curtain to discuss common testing strategies you might face in an interview.

Within the IT sector, it is quite common for software development applicants to sit at least two timed tests:

- **An intelligence test:** this is often a written test which combines elements such as numerical reasoning, letter sequences, vocabulary, spelling, punctuation, lateral thinking and homonyms (two words with the same spelling but different meanings, e.g. 'bright').
- **An aptitude and ability test:** this will be designed specifically for software developers and typically involves solving a programming problem. Often it isn't very complex – for example, you might be asked to create graduated bonus features for a simple guessing game, such as personalisation and a table of high scores. What is significant is that the time limit will be deliberately tight, forcing you to make quick decisions and prioritise your problem-solving. It is quite common for the interview panel to provide constructive feedback on your efforts, offering you a chance to explain your design and development decisions. This can be critical as the panel will usually be genuinely interested not just in your coding abilities but also in your problem-solving and decision-making, particularly when under pressure.

An alternative test might involve completing programming tasks using a remote learning platform which times and grades your efforts. Ultimately, it is up to the organisation how it chooses to judge your ability and performance. For more senior roles, a screening telephone interview may be used initially.

In short, any testing is designed to do two things: put you under pressure and see how you solve problems. Both are completely representative of the everyday challenges you will face as a software developer.

TIPS FOR GETTING STARTED AS A SOFTWARE DEVELOPER

The following tips will help you to get started as a software developer:

- Be 100% committed to this career pathway.
- Target an initial programming language and/or technology stack to learn – it's generally not too difficult to transfer your skills to a new one.

- Program for fun, not just work. Personal projects are rewarding and permit time for experimentation without the risk of impacting ongoing professional projects.
- Use industry tools and workflows – for example, use popular version control tools and gain experience with different command line environments, current project management platforms and so on.
- Don't insulate yourself; investigate wider technologies that often dovetail with yours (e.g. CPUs and system architecture, networking, web standards, databases and emerging cloud technologies). This will help you to communicate effectively with stakeholders in other disciplines.
- Learn to read technical documentation. Much of a software developer's job is working with new libraries, web-based application programming interfaces (APIs), operating systems and so on. You may need to delve into written material to find answers on those rare occasions when a search engine can't help.
- Be prepared to work under pressure and within strict time limits. From a job interview's programming test to partaking in weekly sprints, the challenge of getting the job done within the given constraints to the best standard possible is ever present.
- Be aware of potential vulnerabilities and the best defensive coding practices in order to reduce the likelihood of successful cyberattacks and improve the robustness of your solutions.
- Don't have a precious ego. Everyone, no matter how good they are, can learn from other software developers (even relatively inexperienced ones). Read other people's code to absorb new ideas and accept feedback and criticism of your code without taking personal offence – everything can (and should) be improved over time.
- Be prepared for rejection. The industry is highly competitive, and you will undoubtedly end up vying for a job with other developers who have more experience.
- Invest (time and money) in your skills. Regularly attend workshops, seminars, training courses and so on.
- Be active in the software development community – whether you publish a blog, have a LinkedIn profile, lead an open source passion project, contribute magazine articles, or attend conferences and meetings, be visible. It will help potential employers to form a positive impression of you.
- Be prepared for lifelong learning. IT is a fast-changing sector to work in, and software development doubly so. Your skills need to evolve to meet the challenges and demands that will be placed on you.

SUMMARY

As you can see, there is no single route, educational background or area of life experience that is mandated for stepping into the role of software developer. Gaining recognised qualifications will almost certainly be helpful and many different options exist. Training

in the workplace is also invaluable and can be combined with vocational qualifications to good effect.

However, don't think you'll stop learning once you've got your desired role! The IT industry, and most notably those who develop professionally within it, face a constant battle to keep their skills up to date and competitive.

The next chapter examines target roles in the software development industry.

INDEX

SOFTWARE DEVELOPMENT IN PRACTICE

Bernie Fishpool and Mark Fishpool

Software development is becoming recognised more and more as an essential skill and profession in today's increasingly digital world. Familiarity with basic programming concepts is no longer sufficient to succeed as a software developer, and today's developers require a wider field of expertise and a holistic, customer-focused approach.

This book is a pragmatic guide to software development in practice. It explores the inner workings of software development in the context of the industry, covering good practice for software developers and providing you with tools and practical understanding you'll need to advance within the software development world.

- **Understand key software development tasks, methodologies and languages**
- **Explore writing, testing and debugging good quality code**
- **Consider commercial aspects and deploying in enterprise environments**
- **Ideal for aspiring and early career software developers and career movers**
- **Aligned with the BCS Level 4 Software Developer Digital Apprenticeship**

ABOUT THE AUTHORS

Bernie Fishpool is an educational specialist and subject matter expert, experienced in qualification and curriculum development. Mark Fishpool is a senior commercial trainer, experienced software developer and specialist in curriculum development and management. Together, Bernie and Mark have written over 10 books between them.

You might also be interested in:

A remarkable book that provides a unique perspective on modern software development. The focus on key employment skills and knowledge also makes it a must read for aspiring developers.

Chris Beaumont PhD FBCS FHEA, Chair of Examiners, NCC Education

Software Development in Practice takes the guesswork out of your journey into tech. I especially appreciate the emphasis on communication, collaboration and user experience.

Sjoukje Ijlstra, Software Engineer, JP Morgan

Software Development in Practice covers the areas to master to become a productive member of a software development team.

Chris Galley FBCS CITP

Information Technology, Business

ISBN 978-1-78017-497-6

9 781780 174976

bcs The Chartered Institute for IT

Paperback available