Chapter 1
Introduction

The term 'software quality assurance' rarely evokes much excitement. It is often perceived as a form of applied pedantry, lacking the challenge and creativity that is required to actually design and build a software system. In short, it is often seen as a necessary chore. Nobody wants to read and learn about necessary chores. To some, a book on quality assurance is about as enticing as a book on dish-washing.

In reality such preconceptions are wildly misguided (surely something you would hope for, given that you have nine chapters on the topic ahead of you). The idea that quality assurance stands separately from other software development activities is not true. Nor is the perception that it is particularly pedantic or lacks creativity (though pedantry can be useful!). It is not a chore, and if it is treated as an afterthought, any non-trivial software project is bound to fail.

In this book we will show how quality assurance and software development are inextricably linked. A software system cannot succeed and be sustained without the framework of practices and concepts to continuously assess, ensure, and improve its quality. The real challenge of software development is not merely to assimilate the right source code instructions, but to do so in a way that the code can be readily understood, maintained, and shown to be correct, and to achieve all of this within appropriate time and cost bounds.

1.1 Consistency, Complexity, and Change

Software quality assurance is, as we shall see, notoriously difficult to pin down as a concept. Every software development activity, from drawing up the requirements all the way through to deployment and maintenance, can at some level contribute to (or detract from) the quality of the final software system. Loosely put, the term 'quality assurance' refers to the various strategies and techniques that can be adopted to convey a certain level of confidence in the quality of the final system.

A successful quality assurance strategy should also ensure a degree of repeatability and reliability. It should provide safeguards and mechanisms that will not depend

N. Walkinshaw, *Software Quality Assurance*, Undergraduate
Topics in Computer Science, DOI 10.1007/978-3-319-64822-4_1

on the capabilities of the individual developers. In other words, one overarching goal is to ensure *consistency*.

The challenge is to achieve this in the face of complexity and change. Complexity comes in a variety of forms – from the problem domain for which the software is being developed, the existing base of software libraries and frameworks, and the organisational structure within which the software is being developed. Change also comes in several forms. Capricious customers can continuously change their minds about requirements and priorities. Development teams can comprise large numbers of developers, often located in different locations and time-zones, tweaking the code base at the same time. During development, some activities can take longer than expected, others less.

A key challenge of quality assurance is to ensure that processes are in place that are capable of detecting, accommodating and controlling these various forms of complexity and change. This has to be achieved in a manner is not counter-productive. Quality assurance processes must not hamper development by placing an unnecessary burden on the developers, being overly restrictive in terms of flexibility, or causing unsustainable increases in time or expense.

Finding or developing a suitable quality assurance process is in and of itself a challenge. There are a huge number of potential frameworks, techniques, and tools for every aspect of software development. The most appropriate combination of these approaches invariably depends on various factors that are specific to a given organisation or project. Finding a suitable process therefore relies upon a capacity for introspection – the ability to experiment with different tools and techniques, to determine which approaches work, and which ones need to be refined or replaced.

1.2 Synopsis

This book aims to provide a concise introduction to the approaches to quality assurance that span every aspect of software development. To ensure concision, the book places an emphasis on the underlying foundations of modern quality assurance techniques – on emphasising the *whys* instead of the *hows*. Although the book will delve into some specific techniques, it does not seek to provide a comprehensive reference. The main objective is to provide the reader with a comprehensive understanding of where software quality fits into the development life-cycle (spoiler: everywhere), and what the key quality assurance activities are.

Although software development is relatively new as a discipline, many of the quality assurance principles and techniques have roots in other, more established disciplines. In order to fully appreciate these techniques it is necessary to be aware of their back-stories. This is not only useful from a pedagogical perspective. An appreciation of the historical roots of areas such as software process improvement and agile development supports an often under-appreciated but very important point: software development today has been largely shaped by some of the major advances in manufacturing that shaped the world as we know it.

The reader is encouraged to read the book in a sequential fashion from start to end. The chapters have been structured in such a way that we start from some fo the most general notions (e.g. quality and development process), and gradually home-in on the more specific activities, assuming knowledge of basic notions established in prior chapters.

Chapter 2: What is Software Quality, and Why Does it Matter?

It is impossible to understate the importance of software quality. Software is increasingly pervasive, controlling many devices that we depend upon, both as individuals and as a society. Poor software quality can have wide-ranging, and often unexpected consequences. In this chapter we look at some notable examples. We also use these examples to illustrate just how broad the term "software quality" can be, and explain why it is consequently so notoriously difficult to pin down. The rest of the chapter then looks at some of the key attempts over the last 40 years to formalise such definitions as software quality models.

Chapter 3: Software Development Processes and Process Improvement

The 'quality' of a product is dependent upon the process that was used to develop it – a realisation that underpinned the revolution in industrial manufacturing across the globe. In this chapter we start with some of the key innovations within manufacturing that shaped the principles that have influenced software engineering development processes. We then go on to cover three particularly popular software engineering processes: The Waterfall model, Iterative and Incremental Development, and finally Agile software development (with an emphasis on the SCRUM methodology). The chapter concludes by introducing frameworks to tailor and improve processes within the context of a particular organisation, with a focus on the CMMI framework.

Chapter 4: Managing Requirements and Code

Software development is ultimately about the ability to turn a set of (implicit or explicit) requirements into source code. Given that the requirements can be complex, and the development process can involve many developers, this process needs to be carefully managed. This chapter is split into two parts - the first part examines the challenge of managing requirements: How to elicit and capture them, dealing with questions such as security, how to prioritise them, and trace them to the various development artefacts. The second part is concerned with managing the corresponding source code as it evolves by adopting design and coding conventions and using version repositories.

Chapter 5: Planning Activities and Predicting Cost

Ultimately, sound management of a project requires a degree of planning. A typical project involves lots of different types of activities, that can require varying degrees of time and effort. It is necessary to predict how long different activities will take, and to figure out how to schedule them. Poor planning can lead to cost-overruns or even abandoned projects. In this chapter we look at some of the most popular techniques that can serve to attenuate this risk. The chapter starts by presenting two generic planning tools - the PERT technique and Gantt charts. This is followed by various techniques that can be used to predict the effort or cost required for a project by either using historical data with linear regression techniques, the off-the-shelf COCOMO model, and Planning poker.

Chapter 6: Testing

Testing - the execution of a software system to identify failures or faults - is a particular activity that is most obviously linked with quality assurance. As such, this chapter treats testing with a greater degree of detail. It introduces the key concepts that are involved in software testing. It then proceeds to cover white-box testing (testing techniques that are predicated upon access to the source code), including the various notions of code coverage that can be used to assess test adequacy, test generation techniques that aim to achieve coverage, the cases for and against said coverage metrics, and mutation testing. It concludes with an overview of black-box testing techniques, which cater for the scenario that the source code is not available. This part introduces specification-based testing, random testing, and Fuzz-testing techniques.

Chapter 7: Inspections, Reviews, and Safety Arguments

Software inspection sits alongside software testing as an activity that is specifically geared to identify faults in the source code. In contrast to testing, inspection does not depend on the ability to execute the code. Although the activity has its roots in the traditional ‘Fagan inspections’, this book only briefly covers these in favour of focussing on the less formal but more widespread activities that are broadly referred to as ‘Modern Code Review’, where version repository-based tools such as Gerrit and Pull-requests are used to orchestrate light-weight code reviews. This is followed briefly by an introduction to specific automated and manual code review approaches. The chapter concludes with an overview of an inspection approach that is specific to safety-critical systems, by introducing the concept of safety-arguments, and a graphical language that can be used to express these arguments.

Chapter 8: Measurement

Quality assurance relies fundamentally on the ability to measure: measuring the progress of the development, or the size, complexity, and cost of the system, the complexity of the code-base, etc. This chapter starts off by presenting some general background theory to measurement - what constitutes a valid measurement, and the different types of scales that can be used. It then moves on to cover a variety of software metrics, which analyse software size, modularity, and maintainability. It then concludes with a section that discusses the validity of metrics specifically in the context of software measurement, and how the Goal Question Metric technique can be used to ensure validity.

Chapter 9: Future Challenges and Opportunities

Software development is evolving at a fast pace. Software systems are continuously growing in terms of complexity and expense, which poses enormous challenges in terms of the overhead required by quality assurance activities. Software is increasingly incorporating data-intensive Machine Learning algorithms, the behaviour of which can be very difficult to constrain and validate. These algorithms are being placed into devices that have an increasing bearing on safety and society, with self-driving cars being a prime example. On the other hand, software development has also changed to alleviate some of the long-standing problems. This chapter examines some of these changes, and discusses what they mean for software development.

Key Points and Exercises

Each chapter will be concluded by a section called “Key Points” which summarises the main points that have been covered within the chapter.

Throughout the book you will find “exercises”:

> Exercises will be presented in a highlighted box, such as this one.

These are to be treated as reading-aids, to remind you of relevant parts in the book that have been covered previously, or to give you the opportunity to reflect on a particular topic and refer to related references.