



Abstract

This chapter is the concluding chapter in which we summarize the journey that we have travelled in this book.

We embarked on a long journey in this book and set ourselves the objective of providing a concise introduction to the software engineering field to students and practitioners. The book was based on the author's experience at leading industrial companies, and it covered both theory and practice. The objective was to give the reader a grasp of the fundamentals of the software engineering field, as well as guidance on how to apply the theory in an industrial environment.

Customers today have very high expectations on quality and expect high-quality software to be consistently delivered on time and on budget. The focus on quality requires that sound software engineering practices be employed to enable quality software to be consistently produced. Further, it is an accepted view in the software quality field that the quality of the delivered software is closely related to the quality of the underlying processes used to build the software, and on adherence to them.

Many processes are employed in the design and development of software, and companies need to determine the extent to which the underlying processes used to design, develop, test, and manage software projects are fit for purpose. The process will need to be continuously improved, and often model-based improvement using a framework such as the CMMI is employed. There is also the need to focus on best practice in software engineering, as well as emerging technologies from various research programs. Piloting or technology transfer of innovative technology is an important part of continuous improvement. Companies need to focus on customer satisfaction and software quality, and they need to ensure that the desired quality is built into the software product.

We discussed project planning and tracking, software lifecycles, software inspections and testing, configuration management, software quality assurance, etc. The capability maturity model integrated was discussed, and it provides a

framework that assists organizations in software process improvement. The appraisal of an organization against the CMMI allows the organization to determine the current capability or maturity of selected software processes and to prioritize improvements.

We discussed the professional responsibility of software engineers, where engineers have a professional responsibility to behave ethically with their clients. The professional engineering body requires its member to adhere to the code of ethics of the profession.

We discussed ethical software engineering including the ethical impacts of technical decisions as part of the software engineering process. The ethical software engineer needs to examine both the technical and the ethical dimensions of decisions that affect wider society.

We introduced project management, and discussed project estimation; project planning and scheduling, project monitoring and control, risk management, and managing project quality.

We discussed requirements engineering including activities such as requirements gathering, requirements elicitation, requirements analysis, requirements management, and requirements verification and validation.

We then discussed design and development, including the high-level architectural design, the low-level design of individual programs, and software development and reuse. The views of Hoare and Parnas on software design were discussed, and we discussed the historical function-oriented design and object-oriented design. We discussed software development topics such as software reuse, customized-off-the-shelf software, and open-source software development.

We discussed software inspections including Fagan inspections, as well as the less formal review and walkthrough methodologies. Software testing was then discussed, including the various types of testing that may be carried out, and we discussed test planning, test case definition, test tracking, test metrics, test reporting, and testing in an e-commerce environment.

We then discussed ethics and privacy where professional ethics are a code of conduct that governs how members of a profession deal with each other and with third parties. It expresses ideals of human behaviour, and the fundamental principles of the organization, and is an indication of its professionalism. Privacy is defined as “the right to be left alone”, and specifies there should be no intrusion upon seclusion, and no public disclosure of private facts or false information.

We then discussed metrics and problem solving, including the balanced score card and GQM, as well as presenting a collection of sample metrics for an organization.

We then discussed outsourcing including the selection and management of a software supplier, and we described how candidate suppliers may be formally evaluated, selected, and managed during the project.

We then discussed software configuration management including the concept of a baseline. Configuration management is concerned with identifying those deliverables that are subject to change control and controlling changes to them.

We discussed software quality assurance and the importance of process quality, and the discussion included audits and described how they are carried out.

We discussed the Agile methodology which has become the dominant paradigm in software engineering. It is a popular lightweight approach to software development that has a strong collaborative style of working. It advocates adaptive planning and evolutionary development.

We then discussed software reliability and dependability, and covered topics such as software reliability and software reliability models; the Cleanroom methodology; system availability; safety and security critical systems, and dependency engineering.

We discussed formal methods, which are often employed in the safety critical and security critical fields. These consist of a set of mathematical techniques to specify and derive a program from its specification. Formal methods may be employed to rigorously state the requirements of the proposed system; they may be employed to derive a program from its mathematical specification; and they provide a rigorous proof that the implemented program satisfies its specification.

We discussed the Z specification language, which was developed at the Programming Research Group at Oxford University in the early 1980s. Z specifications are mathematical, and the use of mathematics ensures precision, and allows inconsistencies and gaps in the specification to be identified. Theorem provers may be employed to demonstrate that the software implementation meets its specification.

We then discussed the unified modelling language, which is a visual modelling language for software systems, and it is used to present several views of the system architecture. We presented various UML diagrams such as use case diagrams, sequence diagrams and activity diagrams.

We then discussed the important field of software process improvement, and discussed the idea of a software process, and discussed the benefits that may be gained from software process improvement.

We gave an overview of the CMMI model and discussed its five maturity levels and their constituent process areas. We discussed both the staged and continuous representations of the CMMI.

We then discussed a selection of tools to support various software engineering activities, including tools to support project management, requirements engineering, configuration management, design and development activities and software testing.

We then discussed some innovative developments in the computer field, such as distributed systems, service-oriented architecture, software as a service, cloud computing and embedded systems. This led to a discussion of the many innovations in the software engineering, and the need for continuous innovation.

We then discussed legal aspects of computing including the application of the legal system to the computing field. This includes the protection of intellectual property such as patents, copyright, trademarks and trade secrets, and the resolution of disputes between parties.

Finally, we discussed cybersecurity and cybercrime. Computer crime (or cybercrime) is a crime that involves a computer and a network. The computer may be the vehicle by which the crime was conducted, or it may be the target of the crime. Cybersecurity is the protection of information through good security practices, including the protection of confidentiality, integrity, and availability of data. It is achieved through policies that ensure consistency in employee behaviour in the use of computer resources, as well as training and awareness of security in the workplace.

25.1 The Future of Software Engineering

Software engineering has come a long way since the 1950s and 1960s, when it was accepted that the completed software would always contain lots of defects, and that the coding should be done as quickly as possible, to enable these defects to be quickly identified and corrected.

The software crisis in the late 1960s highlighted problems with budget and schedule overruns, as well as problems with the quality and reliability of the delivered software. This led to the birth of software engineering as a discipline, and the realization that programming is quite distinct from science and mathematics.

The software engineering field is highly innovative, and many new technologies and systems have been developed over the decades. These include object-oriented design and development; formal methods and UML; the waterfall and spiral models; software inspections and software testing; software process improvement and the CMMI; and the Agile methodology.

Software engineering will continue to be fundamental to the success of projects. There is not a one size that fits all: some companies (e.g., in the safety critical or security critical fields) are likely to focus on formal methods and software process maturity models such as the CMMI. For other areas, the lightweight Agile methodology may be the appropriate software development methodology.

Companies are likely to measure the cost of poor quality in future, as driving down the cost of poor quality will become more important. Software components and the verification of software components is likely to become important, to speed up software development and to shorten time to market. Software reuse and open-source software development is likely to grow in popularity, and continuous innovation will continue in the software engineering field.