The objective is to demonstrate that the product satisfies the business requirements and meets the customer expectations. Upon its successful completion the customer is happy to accept the product.

2.5.6 Maintenance

This phase continues after the release of the software product to the customer. Any problems that the customer notes with the software are reported as per the customer support and maintenance agreement. The support issues will require investigation, and the issue may be *a defect in the software*, *an enhancement to the software*, or *due to a misunderstanding*. The support and maintenance team will identify the causes of any identified defects, and will implement an appropriate solution to resolve. Testing is conducted to verify that the solution is correct, and that the changes made have not adversely affected other parts of the system. Mature organizations will conduct post mortems to learn lessons from the defect,[12] and will take corrective action to prevent a re-occurrence.

The presence of a maintenance phase suggests an acceptance of the reality that problems with the software will be identified post release. The goal of building a correct and reliable software product the first time is very difficult to achieve, and the customer is always likely to find some issues with the released software product. It is accepted today that quality needs to be built into each step in the development process, with the role of software inspections and testing to identify as many defects as possible prior to release, and minimize the risk that that serious defects will be found post-release.

The more effective the in-phase inspections of deliverables, the higher the quality of the resulting implementation, with a corresponding reduction in the number of defects detected by the test groups. The testing group plays a key role in verifying that the system is correct, and in providing confidence that the software is fit for purpose. The approach to software correctness almost seems to be a "*brute force*" approach, where quality is achieved by testing and re-testing, until the testing group is confident that all defects have been eliminated. Dijkstra [16] noted that:

> Testing a program demonstrates that it contains errors, never that it is correct.

That is, irrespective of the amount of time spent testing, it can never be said with absolute confidence that the program is correct, and, at best, statistical techniques may be employed to give a measure of the confidence in its correctness. That is, there is no guarantee that all defects have been found in the software.

[12] This is essential for serious defects that have caused significant inconvenience to customers (e.g., a major telecoms outage). The software development organization will wish to learn lessons to determine what went wrong in its processes that prevented the defect from been identified during peer reviews and testing. Actions to prevent a reoccurrence will be identified and implemented.

Many software companies may consider one defect per thousand lines of code (KLOC) to be reasonable quality. However, if the system contains one million lines of code this is equivalent to a thousand post-release defects, which is unacceptable.

Some mature organizations have a quality objective of three defects per million lines of code. This goal is known as six-sigma (6σ) and it was developed by Motorola. It was originally applied it to its manufacturing businesses and subsequently applied to its software organizations. The goal is to reduce variability in manufacturing processes and to ensure that the processes performed within strict process control limits. Motorola was awarded the first Malcom Baldridge Quality award for its six-sigma initiative and its commitment to quality.

2.6 Software Inspections

Software inspections were discussed in Chap. 1 and they are used to build quality into software products. There are a number of well-known approaches such as the Fagan Methodology [20]; Gilb's approach [24]; and Prince 2's approach.

Fagan inspections were developed by Michael Fagan of IBM It is a seven-step process that identifies and removes errors in work products. The process mandates that requirement documents, design documents, source code, and test plans are all formally inspected by experts independent of the author of the deliverable to ensure quality.

There are various *roles* defined in the process including the *moderator* who chairs the inspection. The *reader's* responsibility is to read or paraphrase the particular deliverable, and *the author* is the creator of the deliverable and has a special interest in ensuring that it is correct. The *tester* role is concerned with the test viewpoint.

The inspection process will consider whether the design is correct with respect to the requirements, and whether the source code is correct with respect to the design. Software inspections play an important role in reducing the cost of poor quality in the organization.

2.7 Software Project Management

The timely delivery of quality software requires good management and engineering processes. Software projects have a history of being delivered late or over budget, and good project management practices include the following activities:

- Estimation of cost, effort and schedule for the project
- Identifying and managing risks
- Preparing the project plan
- Preparing the initial project schedule and key milestones
- Obtaining approval for the project plan and schedule
- Staffing the project

- Monitoring progress, budget, schedule, effort, risks, issues, change requests and quality
- Taking corrective action
- Re-planning and re-scheduling
- Communicating progress to affected stakeholders
- Preparing status reports and presentations

The project plan will contain or reference several other plans such as the project quality plan; the communication plan; the configuration management plan; and the test plan.

Project estimation and scheduling are difficult as often software projects are breaking new ground and differ from previous projects. That is, previous estimates may often not be a good basis for estimation for the current project. Often, unanticipated problems can arise for technically advanced projects, and the estimates may often be optimistic. Gantt charts are often employed for project scheduling, and these show the work breakdown for the project, as well as task dependencies and allocation of staff to the various tasks.

The effective management of risk during a project is essential to project success. Risks arise due to uncertainty and the risk management cycle involves[13] risk identification; risk analysis and evaluation; identifying responses to risks; selecting and planning a response to the risk; and risk monitoring. The risks are logged, and the likelihood of each risk arising and its impact is then determined. The risk is assigned an owner and an appropriate response to the risk determined.

2.8 CMMI Maturity Model

The CMMI is a framework to assist an organization in the implementation of best practice in software and systems engineering. It is an internationally recognized model for process improvement and assessment, and is used world-wide by thousands of organizations. It provides a solid engineering approach to the development of software, and helps in the definition of high-quality processes for the various software engineering and management activities.

It was developed by the Software Engineering Institute (SEI) who adapted the process improvement principles used in the manufacturing field to the software field. They developed the original CMM model and its successor the CMMI. The CMMI states *what the organization needs to do* to mature its processes rather than *how this should be done*.

The CMMI consists of five maturity levels with each maturity level consisting of several process areas. Each process area consists of a set of goals, and these goals are implemented by practices related to that process area. Level two is focused on management practices; level three is focused on engineering and organization practices; level four is concerned with ensuring that key processes are performing

[13] These are the risk management activities in the Prince 2 methodology.

within strict quantitative limits; level five is concerned with continuous process improvement. Maturity levels may not be skipped in the staged implementation of the CMMI, as each maturity level is the foundation for the next level.

The CMMI allows organizations to benchmark themselves against other organizations. This is done by a formal appraisal conducted by an authorized lead appraiser. The results of the appraisal are generally reported back to the SEI, and there is a strict qualification process to become an *authorized lead appraiser*. An appraisal is useful in verifying that an organization has improved, and it enables the organization to prioritize improvements for the next improvement cycle. The CMMI is discussed in more detail in a later chapter.

2.9 Formal Methods

Dijkstra and Hoare have argued that the way to develop correct software is to derive the program from its specifications using mathematics, and to employ *mathematical proof* to demonstrate its correctness with respect to the specification. This offers a rigorous framework to develop programs adhering to the highest quality constraints. However, in practice mathematical techniques have proved to be cumbersome to use, and their widespread deployment in industry is unlikely at this time.

The *safety-critical area* is one domain to which mathematical techniques have been successfully applied: for example, demonstrating the presence or absence of safety properties such as "*when a train is in a level crossing, then the gate is closed*". There is a need for extra rigour in the software development process used in the safety critical field, and mathematical techniques can demonstrate the presence or absence of certain desirable or undesirable properties.

Spivey [62] defines a "*formal specification*" as 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. It describes *what* the system must do, as distinct from *how* it is to be done. This abstraction away from implementation enables questions about what the system does to be answered, independently of the detailed code. 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.

The formal specification thus becomes the key reference point for the different parties concerned with the construction of the system, and is a useful way 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 avoids the problem of ambiguity inherent in a natural language specification. The derivation of an implementation from the specification may be achieved via *step-wise refinement*. Each refinement step makes the specification more concrete and closer to the actual implementation.

There is an associated *proof obligation* that the refinement be valid, and that the concrete state preserves the properties of the more abstract state. Thus, assuming 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.

Formal methods have been applied to a diverse range of applications, including circuit design, artificial intelligence, specification of standards, specification and verification of programs, etc. They are described in more detail Chap. 17.

2.10 Review Questions

1. Discuss the research results of the Standish Group the current state of IT project delivery?
2. What are the main challenges in software engineering?
3. Describe various software lifecycles such as the waterfall model and the spiral model.
4. Discuss the benefits of Agile over conventional approaches. List any risks and disadvantages?
5. Describe the purpose of software inspections? What are the benefits?
6. Describe the main activities that take place in software testing.
7. Describe the main activities in project management?

2.11 Summary

The birth of software engineering was at the NATO conference held in 1968 in Germany. This conference highlighted the problems that existed in the software sector in the late 1960s, and the term "*software crisis*" was coined to refer to these. This led to the realization that programming is quite distinct from science and mathematics, and that software engineers need to be properly trained to enable them to build high-quality products that are safe to use.

The Standish group conducts research on the extent of problems with the delivery of projects on time and budget. Their research indicates that it remains a challenge to deliver projects on time, on budget and with the right quality.

Programmers are like engineers in the sense that they build products. Therefore, programmers need to receive an appropriate education in engineering as part of their training. The education of traditional engineers includes training on product design, and an appropriate level of mathematics.

Software engineering involves multi-person construction of multi-version programs. It is a systematic approach to the development and maintenance of the software, and it requires a precise statement of the requirements of the software product, and then the design and development of a solution to meet