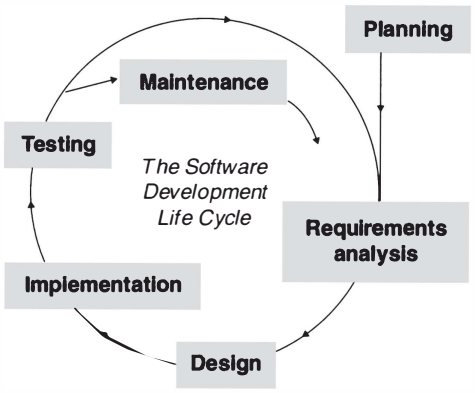
5

Quality in the Software Process



- What are the principles of managing quality?
- How do you plan for "quality?"
- What are inspections and how do you carry them out?
- How do you carry out reviews and audits?
- How do you measure and improve software processes?
- In what way does CMMI assess organizational quality?
- What does a software quality plan look like?

Figure 5.1 The context and learning goals for this chapter

This chapter discusses the manner in which we integrate and manage quality throughout the software development process. This is an integral part of project planning, and it affects every phase of development. As part of planning, documents are produced detailing the quality approach to be taken, including specific quality goals, techniques to be employed, metrics to be collected, tests to be run, and the manner in which

- Quality principles
  —overarching quality guidelines
- Quality planning
  —quality plan defines overall approach to quality
- Inspections
  —peer processes focused on quality
- Reviews and audits
  —external quality assessment
- Defect management
  —identification, tracking, and resolution of defects
- Process improvement
  —continuous upgrading of process effectiveness
- Organizational quality
  —engineering competence levels (e.g., CMMI)

Figure 5.2 Quality in the software process

defects are to be handled. All of these contribute to an effective development process with a focus on quality, resulting in the development of a high-quality software product. The integration of quality in the software process is summarized in Figure 5.2. Each topic mentioned in the figure is described in this chapter.

5.1 PRINCIPLES OF MANAGING QUALITY

An overall approach to quality is guided by several principles. The planning and practices described in this chapter are implemented with these principles in mind, as follows.

First and foremost, quality is something the entire development team is responsible for, not just the quality assurance team. This is the meaning of the first point in Figure 5.3: focusing "continuously on quality." It is thus an integral part of the development process, and each phase must incorporate quality practices. For example, peer inspections are carried out for each artifact when it is generated. Depending on the artifact under review, stakeholders from different functions such as marketing, customer service, software development, customers, and so on participate in the inspection. Inspections are discussed in detail in Section 5.4. Another example is test-driven development (TDD), which is prevalent in various agile processes. It dictates

1. Focus continuously on quality.
2. A quality assurance process must be defined.
3. The organization must follow its quality assurance process.
4. Find and repair defects as early in the development process as possible.

Figure 5.3 Key principles of managing quality

Reason (by one estimate):	If defect found . . .	
	. . . soon after creation	. . . at integration time
Hours to ..		
.. detect	0.7 to 2	0.2 to 10
.. repair	0.3 to 1.2	9+
Total	**1.0 to 3.2**	**9.2 to 19+**

Figure 5.4 Cost to repair a defect, depending on time elapsed since injection

that developers write tests before coding, and then write code to pass their tests. TDD is discussed in more detail in Chapter 27.

Quality assurance (QA) refers to actions taken to assure that a product under construction attains required levels of quality. The second and third principles listed in Figure 5.3 state that a QA process must be defined and followed. This is how so much quality has been encouraged by the International Standards Organization (ISO) standard 9001 "Quality Systems—Model for quality assurance in design, development, production, installation, and servicing." Many companies have created documents describing their official QA process. ISO insists on the existence of such documents for companies seeking its certification. However, ISO recognized that a documented procedure is only a part of quality assurance. In practice, documented quality procedures are often ignored with impunity; hence ISO's second principle, which ensures that employees actually following quality procedures. Quality planning is discussed in more detail in Section 5.3.

The fourth software quality principle listed in Figure 5.4 is to find and fix defects as early as possible: i.e., to prevent them from persisting for more than a single phase. Figure 5.4 provides data from one of the author's clients on the relative cost of allowing a defect to persist. Many organizations have tried to measure the difference in cost, and it is always found to be very large. A rule of thumb sometimes used is that a defect costing a dollar to find and repair soon after its creation costs a hundred dollars to find and repair if delivered to the customer. The main consequence of this principle is that it most often pays to spend significant resources finding and fixing defects as early as possible rather than allowing them to remain undetected and active.

5.2 MANAGING QUALITY IN AGILE PROCESSES

Agile and non-agile projects are equally concerned with producing quality products, but agile methods go about this in different ways. In particular, agile responses to the principles of Section 5.1 are as follows:

1. Focusing continuously on quality
   Agile processes fulfill this principle by creating increments from close customer contact, by testing them during development, and via prearranged customer acceptance tests.

2. Defining a quality assurance process
   Agile processes rely on the activities just listed for a QA process, rather than on documentation such as the IEEE standards discussed in this chapter. Whether this is sufficient, especially for large projects, remains a subject of debate.
3. Following the organization's quality assurance process
   Whether or not one doubts the adequacy of agility for quality in large projects, team members are generally well motivated, and they usually follow agreed-upon methods very faithfully.
4. Finding and repairing defects as early in the development process as possible
   Here, agile methods show their greatest strength, since they are code centric and thus try out implementation and testing of the pieces as soon as it is possible.

5.3 QUALITY PLANNING

We begin planning for quality early in the development life cycle. Documents are written that define an overall approach, including quality goals and techniques for every development phase (the Software Quality Assurance Plan), how the product will be verified and validated (the Software Verification and Validation Plan), and how testing is planned, specified, and reported (the Software Test Documents). The Software Quality Assurance Plan (SQAP) is described below, the Software Verification and Validation Plan (SVVP) is described in Chapter 9, and the Software Test Documents (STD) are described in the testing chapters of Part VII.

5.3.1 Software Quality Assurance Plan

The Software Quality Assurance Plan (SQAP) specifies the overall quality plan, policies, and procedures that will be followed by the team. It provides guidelines so that quality actions are not an afterthought but something consciously and deliberately striven for and practiced throughout the development process. It orients the project toward prevention—defining procedures for proactively monitoring and improving processes and quality, ensuring that quality practices are defined and implemented, and ensuring that problems are identified and resolved as early as possible. To do this, the SQAP answers the questions posed in Figure 5.5.

The rest of this section addresses these questions in turn.

1. Who will be Responsible for Quality?

An individual or group is identified as being responsible for assuring quality on a project. The individual or group is tasked with ensuring that quality is integrated into all activities: their existence does not alter the principle that quality is everyone's responsibility. For very small projects, there may be no explicit QA organization, making QA the responsibility of the project management plan. Large projects require several QA people, led by a QA manager. Sometimes an organization's QA manager is the QA lead on several projects. For truly large projects, the QA function has its own management hierarchy. In the author's experience, a project requires roughly one QA person-month for every 3 to 7 developer person-months. This excludes developer testing, which is usually considered internal because it requires intimate knowledge of the design and implementation. An example of estimated QA requirements is summarized in Figure 5.6.

2. What Quality Documentation is Generated?

The SQAP specifies the documents to be generated for a project and how they are to be checked for accuracy. Examples of documents are the Software Requirements Specification (SRS), Software Design Document

1. **Who** will be responsible for quality?

   A person, a manager, a group, an organization, etc.

2. What **documentation** will be generated to guide development, verification and validation, use and maintenance of the software?

3. What **standards** will be used to ensure quality?

   Documentation standards, coding standards, etc.

4. What **metrics** will be used to monitor quality?

   Product and process metrics

5. What **procedures** will be used to **manage** the quality process?

   Meetings, audits, reviews, etc.

6. What kind of **testing** will be performed?

7. What quality assurance **techniques** will be used?

   Inspections, proofs of correctness, tests, etc.

8. How will **defects** be handled?

Figure 5.5 What's needed from a quality plan

(SDD), Software Verification and Validation Plan (SVVP), User Documentation, and the Software Configuration Management Plan (SCMP). The SRS defines the requirements for the software, and the SDD describes how the software is designed to meet the requirements. The SRS and requirements are covered in Part IV. The SDD and software design are covered in Part V.

The SVVP describes methods used to verify that the requirements specified in the SRS are the right set of requirements, that the requirements are correctly implemented by the SDD, and that the design in the SDD is correctly implemented by the code. Verification methods include inspection, analysis, and testing. Inspection is described in detail in Section 5.4. User documentation describes how the software shall be successfully and correctly used. It also describes error messages and corrective actions to take as a result. The SCMP details how changes to software and documents are managed, and is described in greater detail in Chapter 7.

- 1 QA person per 3–7 developers
- Excludes developer testing counted as developer time
- Includes post-developer testing
- Ideally performed by external QA personnel

Figure 5.6 Typical estimate of QA requirements per developer

3. What Standards will be Used to Ensure Quality?

Adopting consistent document templates contributes to the production of quality documentation. The use of IEEE standard documents—as outlined in this text, for example—ensures that documents contain all the necessary content and can be easily read and understood by team members.

Coding standards dictate such things as variable and module naming, commenting, and logic structure. Consistently developed code aids in different team members being able to better understand it and make updates.

4. What Metrics will be Used to Monitor Quality?

The SQAP defines the quality metrics to be collected and analyzed. Examples of quality metrics include defect density, mean time to failure, customer problems, and customer satisfaction. Metrics are discussed in detail in Chapters 3 and 9, among others.

5. What Procedures will be Used to Manage the Quality Process?

Meetings, audits, and reviews are some of the techniques employed to manage the quality process. They are described in more detail in Section 5.5.

6. What Kind of Testing will be Performed?

Both the Software Verification and Validation Plan and the Software Test Documents specify the tests to be executed.

7. What Quality Assurance Techniques will be Used?

Inspections, proofs of correctness, tests, and so on are all techniques used to assure product quality. Section 5.4 in particular describes the inspection process in greater detail. Proofs of correctness and testing are described in this book as well.

8. How will Defects be Handled?

As defined in Chapter 2, defects are deviations from requirements. Procedures for identifying and managing them are defined in the SQAP. Defect management is covered in detail in Section 5.6.

5.3.2 IEEE Quality Documents

The IEEE has published several standards related to software quality, as follows:

- IEEE 730-2002: Standard for Software Quality Assurance Plans
- IEEE 1012-2004: Standard for Software Verification and Validation
- IEEE 829-1998: Standard for Software Test Documentation

These documents relate to the software development phases as shown in Figure 5.7.

Figures 5.8 and 5.9 summarize the contents of IEEE Standard 730-2002 Standard for Software Quality Assurance Plans. The case study section near the end of this chapter contains a sample SQAP for the Encounter video game.

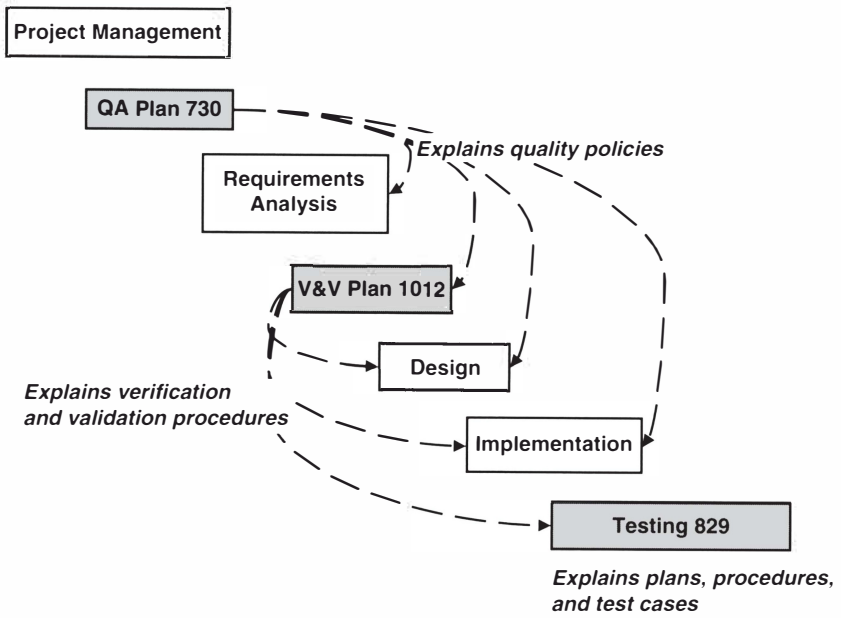


Figure 5.7 The main IEEE software quality documents

1. **Purpose**
2. **Referenced documents**
3. **Management**
   - 3.1 Organization
   - 3.2 Tasks
   - 3.3 Responsibilities
   - 3.4 QA estimated resources
4. **Documentation**
   - 4.1 Purpose
   - 4.2 Minimum documentation requirements
   - 4.3 Other
5. **Standards, practices, conventions, and metrics**
   - 5.1 Purpose
   - 5.2 Content
6. **Reviews**
   - 6.1 Purpose
   - 6.2 Minimum requirements
     - 6.2.1 Software specifications review
     - 6.2.2 Architecture design review
     - 6.2.3 Detailed design review
     - 6.2.4 V&V plan review
     - 6.2.5 Functional audit
     - 6.2.6 Physical audit
     - 6.2.7 In-process audits
     - 6.2.8 Managerial review
     - 6.2.9 SCMP review
     - 6.2.10 Post-implementation review
   - 6.3 Other reviews and audits
7.–**15**. See next figure

Figure 5.8 IEEE Software Quality Assurance Plan table of contents 1 of 2

Source: IEEE Std 730–2002.

7. **Test**

— may reference Software Test Documentation.

8. **Problem reporting and corrective action**

9. **Tools, techniques, and methodologies**

— may reference SPMP.

10. **Media control**

11. **Supplier control**

12. **Records collection, maintenance, and retention**

13. **Training**

14. **Risk management**

— may reference SPMP.

15. **Glossary**

16. **SQAP change procedure and history**

Figure 5.9 IEEE Software Quality Assurance Plan table of contents 2 of 2

Source: IEEE Std 730–2002.

5.4 INSPECTIONS

An *inspection* is a quality technique that focuses on reviewing the details of a project artifact (requirements, designs, code, etc.) in an organized and thorough manner. Inspections are performed periodically during all software engineering phases by the artifact's author and by other engineers. The purpose is to assure the artifact's correctness by seeking defects. A meeting of inspectors is held at which defects are identified. The repair of defects is the author's responsibility.

The inspection concept was developed by Michael Fagan [1] while he was at IBM. He observed that the author of a work is usually able to repair a defect once he recognizes its presence. Thus, a process is needed whereby the defects in a work are called to the author's attention as early as possible. This implies that inspections should be a peer process because inspections are performed on work in process rather than on finished product.

Since inspections were originally introduced to improve code, they are often referred to as "code inspections," but their value has been shown to be greatest when used early in the process, long before code is produced. They are used profitably as soon as the first project documents are produced. For example, requirements specifications, project management, and configuration plans should all be inspected.

5.4.1 Inspection Principles

Guiding principles for conducting inspections are listed below. The sections that follow describe each of these:

1. Peer process
2. Specified roles
3. Defect detection instead of defect repair
4. Use of checklists
5. Artifact readiness
6. Adequate preparation
7. Metrics collection
8. Time limit

1. Peer Process

Inspections are conducted by a group of individuals who are familiar with the artifact under review. They may be software engineers, members of other departments such as software quality assurance, marketing, sales, or customers. The inspection is a peer process with an overriding goal of discovering defects. The work in progress is under inspection, not the performance of the author, and therefore it is not a supervisor-subordinate process. An author is responsible mainly for the product he or she submits *after* inspection, not before. The work brought to the inspection should be the author's best effort, not a draft.

2. Specified Roles

Inspections work best when specific roles are assigned to each participant.

The *moderator* is responsible for leading the inspection and seeing that it is conducted in a productive and efficient manner. The moderator schedules the meeting and ensure that it starts and ends on time, with the latter being accomplished by maintaining an appropriate pace throughout. It is very easy for participants to get bogged down in details and to try to solve problems during the meeting. It is the job of the moderator to prevent this from happening and to keep the meeting moving along. However, the moderator must also ensure that the pace is not too fast as to miss defects. When defects are identified, the moderator is responsible for ensuring that there is consensus from the team. Thus, to be effective a moderator should be technically competent. The moderator is also an inspector. The job can sometimes involve sensitive issues, such as having to moderate among hostile participants with differing opinions. As described by Wiegers, "Moderators should be trained in how to conduct inspections, including how to keep participants with strong technical skills but low social skills from killing each other." [2]

The *author* is the person responsible for the work product itself, and repairs all defects found (offline). The author is also an inspector, looking for defects along with other inspectors. In order to avoid bias, the author should not serve in any other role. Authors must remain objective throughout the inspection and not become defensive. This can be hard as others are finding "faults" with their work.

The *recorder* is responsible for writing down descriptions and classifications of the defects found, and for recording the action items. If the inspection team is very small, the moderator could also take on this role, but it usually is best to have someone else assume this responsibility. The recorder is also an inspector.

A *reader* is responsible for leading the team through the work in an appropriate and thorough manner. This person selects the most effective sequence for presenting the work product and answer questions posed by the inspection team. The reader is also an inspector. In many cases, the role of the reader is handled by the moderator or author.

An *inspector* is a participant who attempts to identify defects in the artifact under review.

All participants in an inspection act as inspectors, in addition to any other responsibilities they may have. It is important to invite people who can make a significant contribution and have the expertise to identify defects to the inspection. Examples of people who should attend are other project members, those responsible for testing or maintaining the product, and depending on what artifact is being developed, customer and business representatives. Depending on the artifact being inspected, there may be a requirement for specialized inspectors. For example, a *focused inspector* inspects for a specific criterion (e.g., reliability). A *specialized inspector* is a specialist in the area covered by the artifact under inspection (e.g., a radar expert for a radar control application).

3. Defect Detection, not Defect Repair

Inspections should focus on identifying defects, and specifically exclude any discussion of their repair. The repair process is left to the author, and no time should be spent during inspections even suggesting repairs. All repair suggestions should be made offline. This is typically one of the hardest things to control—many participants love to discuss potential solutions and it is quite easy to fall into interminable technical discussions regarding the best approach. This is where a strong moderator is essential. It is key to not let this type of discussion continue and instead to remind people that the goal of the meeting is to identify defects, not repair them.

4. Checklists

A checklist can be very useful in providing guidance to inspectors, describing specific areas to pay attention to. Each type of artifact such as project plan, requirements specification, design specification, configuration management plan, source code, and so on should have its own checklist, as each requires different areas of focus. Example questions to ask when examining a requirements specification might be as follows: Is each requirement verifiable by testing? Is each requirement uniquely identified? For code inspections, questions to ask might be: Are there any variables that are unused or redundant? Is the code adequately commented? Good examples of checklists can be found in [3].

5. Artifact Readiness

The artifact under review should be in the best state of readiness that the author is capable of. For documents there should not be many, if any, sections with "to be determined." Code should compile cleanly. When a group of people takes time to identify a defect that the author would have found with reasonable effort, a significant amount of time is wasted.

6. Adequate Preparation

Inspections are not the same as reviews, management overviews, or education sessions. Inspectors have to work at the same level of detail as the author. (This is what makes inspections time-consuming and thus expensive.) The artifact under review is distributed several days before the meeting, allowing inspectors adequate time to study the material, identify defects, and prepare questions to ask at the inspection. Inspection-aiding software can save time by allowing inspectors to enter descriptions of defects they discover while preparing. The recorder accesses and edits these at inspection meetings.

7. Metrics Collection

During inspections, metrics are collected and analyzed. Examples of metrics to collect are as follows:

- Number of defects discovered, by severity and type
- Number of defects discovered by each category of stakeholder inspecting the artifact

- Number of defects per page reviewed
- Review rate (number of pages/hour)

For severity, a simple classification such as *trivial*, *minor*, and *severe* can be used. For type, categories such as missing function, incorrect function, performance, and usability can be used. This classification topic is discussed in Section 5.6.1.

The metrics collected from inspections are analyzed and the results used for several purposes. First, they are used to predict the efficiency of future reviews. If a company has information on the rate of review for a particular artifact, it can use that to predict how long to schedule for a future review of a similar artifact. Second, counting the number of defects discovered during each stage of development is useful in predicting the number of defects in future projects. Even more specific, counting the number of defects discovered by a *particular stakeholder* (e.g., marketing, customer, QA) is also useful. For example, if during a requirements review fewer than expected defects are discovered by the customer representative, it could indicate either a lack of preparation or misunderstanding of requirements by that person. In either case, this would raise a red flag and a follow-up meeting with the customer would ensue to understand the cause for the discrepancy.

8. Time Limit

Inspections should not be marathon sessions, which can occur if the moderator does not keep the meeting moving along. After a certain amount of time, participants will not be as focused as needed. Each session should be kept to a maximum of two hours, and if it is not completed by then a follow-up session should be rescheduled.

5.4.2 Inspection Process

The inspection process follows an orderly flow of steps, each adhering to the principles described above. The steps are summarized in Figure 5.10.

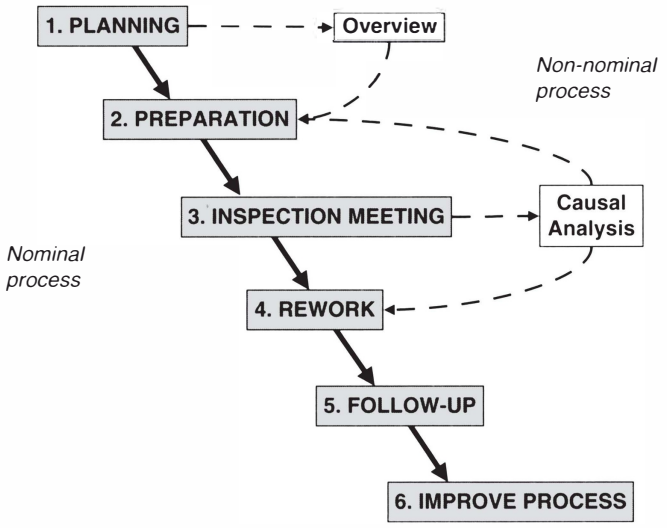


Figure 5.10 The parts of the inspection process, emphasizing the main sequence of activities

1. **Planning.** The process begins with planning. This includes deciding which inspection metrics to collect, identifying tools to be used in recording and analyzing these data, deciding who will participate in inspections and when they will take place, and distributing the materials several days prior to the meeting. Typically, the moderator is responsible for these tasks.

   1A. **Overview meeting.** If necessary, an overview meeting can be organized to explain the artifact under inspection. Since meetings are expensive, this should be avoided unless obviously necessary.

2. **Preparation.** The next step for each inspection consists of preparation. Here, inspectors review the work in complete detail at their own workstations (e.g., checking that the code under inspection correctly implements the detailed design), possibly using checklists provided for guidance. What makes the inspection process valuable, but also expensive, is the fact that this time-consuming process is performed by several people in complete detail. The process is not a "review," because inspectors work at the same level of detail as the author. Inspectors frequently enter the defects they find into a database (e.g., Web-accessible) together with descriptions and classifications. This helps to prevent duplication, and it minimizes unnecessary meeting time. Some prefer to use paper to record their defects, and some consider the number of inspectors who recognize a given defect to be a useful metric.

3. **Inspection meeting.** When every participant is prepared, the inspection meeting takes place. During this meeting, the participants honor their designated roles. Of particular importance is to not try and *solve* problems that are raised, but instead to ensure that they are recognized as defects, to record them as action items only, and to move on.

4. **Rework.** Normally, the author is able to repair all defects, working alone. This is the rework phase. If the inspection meeting decides, however, that the defects are so pervasive that a reinspection is required, then the item is recycled through the process.

   4A. **Causal analysis.** If the defects are due to a misunderstanding or widespread misconception, it may be necessary to call a separate meeting at which these causes are analyzed and discussed. Again, since meetings are expensive, these should not be scheduled casually.

5. **Follow-up.** After the author repairs the defects identified during the inspection meeting, a brief follow-up meeting is conducted at which the moderator and the author confirm that the defects have indeed been repaired. This is not intended to be a detailed review by the moderator. The onus for repair is on the author, who is responsible for the work. If the number of defects repaired is high, a follow-up inspection may be required.

6. **Improve process.** Organizations should always analyze the efficacy of their processes and strive to improve them. For inspections, the group meets from time to time to review the inspection process itself, and decides how it can be improved. They examine the metrics collected, including the list of defects, and decide how the development process can be improved to reduce and/or prevent the same types of defects in the future.

Figure 5.11 shows the average relative times for each inspection process step, used as reference data by one of the author's clients.

The inspection meeting time in Figure 5.11 is shown as one hour for the sake of reference. Individual companies or development groups record inspection times and quantities inspected, and they estimate future inspections based on these historical data. The times shown in Figure 5.11 may disturb the uninitiated, who may wonder whether it really does take that long to check code. Producing professional-quality products does indeed take a substantial amount of time, and any failure to recognize the true costs ends up consuming far more time in the end. Inspections have been estimated by Gehani and McGettrick [4] to consume as much as 10-15 percent of the development budget.

	One company's estimates
Planning	**1 hr ×(1 person)**
Overview	*1 hr × (3–5 people)*
Preparation	**1 hr × (2–4 people)**
Inspection meeting	**1 hr × (3–5 people)**
Rework	**1 hr × (1 person)**
Analysis	*1 hr × (3–5 people)*
Total:	**7–21 person-hours**

Figure 5.11 Inspection time/costs per 100 non-commented lines of code

Inspections are expensive because they consume the time of several engineers. Nevertheless, various studies (for example, [1]) have shown that they pay off handsomely. This is due to the astronomical cost of detecting and repairing defects undiscovered until close to delivery time. Quantified benefits of inspections from sources such as IBM, ICL, and Standard Bank, for example, are summarized in [5].

Appendix D of the case study shows an example of a form for reporting the results of an inspection.

5.5 QA REVIEWS AND AUDITS

The QA organization works with the project leadership to define the manner in which external quality assessment will be performed, specifies this in the SQAP, and executes it during the course of the project. External QA activities are either *reviews* (sometimes called *walk-throughs*), which are scheduled in advance, or *audits*, which are not so scheduled. A project can expect both types of activities. Figures 5.12 and 5.13 show the range of options for QA involvement in reviews and audits, starting from the most invasive to the least.

- Participate in all meetings
  Including formative sessions and inspections.
- Review all documents
  Participate in all inspections
  (but do not attend all meetings).
- Attend final reviews and review all completed documents
- Review select completed documents
  But do not participate otherwise.

Figure 5.12 Options for QA reviews

- Audit at unrestricted random times
  Includes visiting with engineers.
  Includes inspecting any document at any time.
- Audit random meetings
- Audit randomly from a specified list of meetings
- Audit with notice
- No auditing

Figure 5.13 Options for QA audits

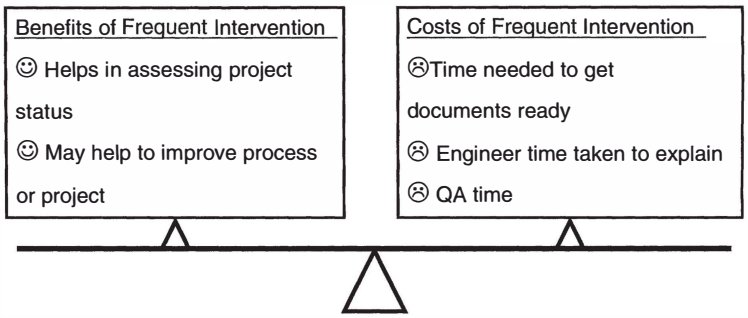


Figure 5.14 Comparing frequent vs. occasional QA intervention

Frequent intervention by QA has the advantage of providing QA personnel with greatly improved understanding of the health of the project, but it may disrupt the project by frequently calling for documents or for engineers' time. This trade-off is summarized in Figure 5.14.

5.6 DEFECT MANAGEMENT

Projects encourage the submission of defect reports by all team members. Usually, a screening process that ensures defect reports are valid and accurate is put in place. To manage defects, QA standardizes on the manner in which defects are defined and tracked. These data can then be compared between different projects so that estimates can be made regarding new projects. We explain this standardization next.

5.6.1 Classifying Defects

Teams classify each defect by its *severity* (how serious it is), *priority* (indicating when it will be handled), its *type* (the kind of problem) and its *source* (the phase during which it was injected). These classifications are illustrated in Figure 5.15.

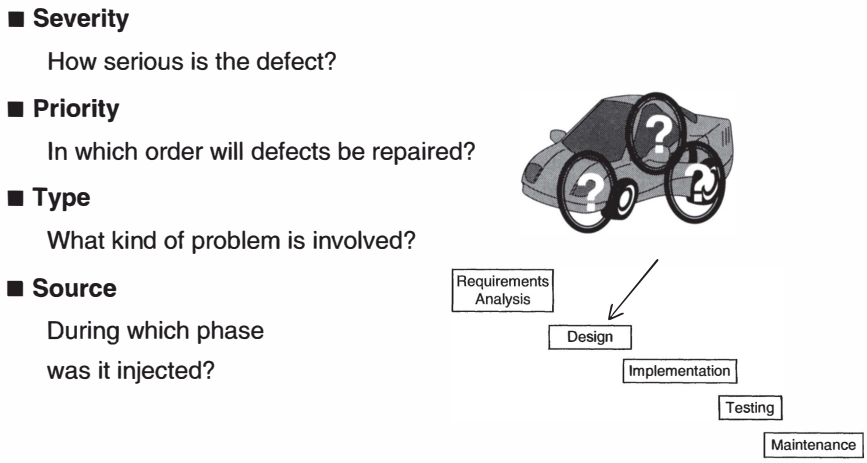


Figure 5.15 Defect classification

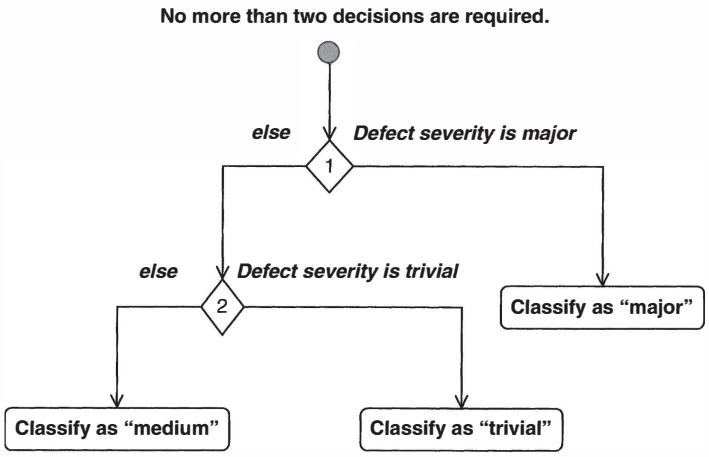


Figure 5.16 The triage decision method applied to defect severity classification

A single severity classification scheme can be used for all artifacts. It is possible to be very discriminating among levels of severity, but this consumes time. The results should be worth the time. One of the authors has frequently encountered teams using discriminating classification schemes that consume significant decision-making time and yet the data that they produce are not utilized by the organization.

Deciding the severity level of a defect falls into can sometimes be difficult. Triage is a useful and simple decision-making technique when confronted with a large number of possibilities, a situation occurring often in software engineering. Utilizing triage for defect classification has the advantage of being fast to perform because each classification requires answering just one or two questions as shown in Figure 5.16. Once a list of defects develops, the team approaches the major ones first (they can be placed in order with the "major" category if necessary). Once they have been attended to, the medium severity defects can be handled. The team gets to the trivial ones only after this, if time allows.

A simple way to classify the severity of defects when using triage is shown in Figure 5.17.

Although triage is fast, the three-category classification lumps together all defects that fail requirements, whether serious or not. Thus, "name field is not purple, as required" and "system crashes every minute" (a "showstopper" since its effects are catastrophic for the application) are given the same severity, which is rather extreme. Showstoppers usually need to be separated. To address this, the IEEE has defined a more refined classification of defect severity as shown in Figure 5.18. The "None" category is added to collect unprioritized defects.

- **Major**
  Causes a requirement to be unsatisfied.
- **Medium**
  Neither *major* nor *trivial*.
- **Trivial**
  Defect doesn't affect operation or maintenance.

Figure 5.17 Classifying defect severity by triage

- **Urgent**
  Failure causes system crash, unrecoverable data loss, or jeopardizes personnel.
- **High**
  Causes impairment of critical system functions, and no workaround solution exists.
- **Medium**
  Causes impairment of critical system functions, though a workaround solution does exist.
- **Low**
  Causes inconvenience or annoyance.
- **None**
  None of the above.

Figure 5.18 IEEE 1044.1 Severity classification

A defect's *priority* is the order in which the team plans to address it. This is somewhat aligned with the defect's severity but is not identical. For example, a defect can be classified as "urgent" but it may not be necessary to repair it immediately. For the sake of project efficiency, for example, its repair may be batched with a set of repairs to the same part of the product.

Different phases use different defect *type classifications*. For example, a defect in the requirements may concern ambiguity but "ambiguity" does not apply very well to code. On the other hand, code may have a logic defect but "logic" does not apply to requirements in the same way. Some types of defects *are* common to all artifacts, as listed in Figure 5.19.

The *source* of a defect is the phase in which it was introduced (or *injected*). For example, you may discover while testing a video store application that it does not—but should—accept the video *It's A Mad, Mad, Mad, Mad World* because the requirements document stated that no word in a title should be repeated more than once (supposedly to prevent bad data entry). Even though the defect may have been discovered during the testing phase, its *source* phase was requirements analysis.

5.6.2 Tracking Defects

Table 5.1 shows a typical way to track known defects. The information is used to manage ongoing work and to record defect history for postmortem and estimation purposes.

- **Omission**
  Something is missing.
- **Unnecessary**
  The part in question can be omitted.
- **Nonconformance with standards**
- **Inconsistency**
  The part in question contradicts other part(s).
- **Unclassified**
  None of the above.

Figure 5.19 Common defect types across all artifacts

Many bug-tracking programs are indispensable in tracking and managing defects. *Bugzilla* is an example of an open source defect-tracking system. It was originally written by Terry Weissman. Bugzilla features include the following:

> inter-bug dependencies, dependency graphing, advanced reporting capabilities extensive configurability, a very well-understood and well-thought-out natural bug resolution protocol, email, XML, console, and HTTP APIs. (It is) available integrated with automated software configuration management systems, including CVS [6].

Appendix B of the case study contains an example of the way in which defects can be reported, and Appendix A contains an example of the way in which they can be tracked.

5.7 PROCESS IMPROVEMENT AND PROCESS METRICS

Even a very good process has to adapt to changes such as new technology and new kinds of requirements. For these reasons, very effective software development organizations include a *meta-process* with every software process: a process for improving the process itself. This usually takes the form of meetings at the end of phases to review the effectiveness of the process used with a view to improving it for future projects. A meta-process is outlined in Figure 5.20.

To measure the effectiveness of a process, an organization has to use it for several projects and then compare the project metrics. Table 5.2 gives an example.

These results suggest that process U is the most effective, since it produces the best results in every category. Process V is the worst for the opposite reasons. Matters are not often so simple, however. For example, suppose that we had to choose between Waterfall and "Waterfall + Incremental." The latter process is superior in defect count, developer cost, and customer satisfaction, but it scores worse in the other categories. Making the call depends on the relative importance of the factors to the organization.

Figure 5.21 lists a sequence of actions that can be taken throughout the life of a project in order to continually improve the process.

Figure 5.22 is an example of the kind of data that can be collected about the process. It is applied to the process of collecting detailed requirements, which is covered in Part IV, but the table is applicable to most phases. The numbers are illustrative only, and should not be regarded as industry standards. A comparison with the organization's normative (typical) data reveals deficiencies in the team's meeting process and in their individual execution (i.e., the actual writing process). This exposes problems in meetings, for example, which were subjectively evaluated 2 out of 10 by the team. It was determined (though not visible in the data) that the meeting process would improve if the straw man proposal brought to the meeting was more complete i.e. an explicit proposal that can be used as a basis for discussion.

The other problem observed in the example shown in Figure 5.22 seems to occur during the execution step, where the actual work of writing the requirements is performed. The defect rate is higher than normal (5 versus 3) and the self-assessed quality is a little below average (4). Compared with company norms, there appears to be room to spend more time executing the work (i.e., as individuals), thereby reducing the defect count and improving the subjective self-assessment. You can observe from this process that a standard for counting the parts of the phase is fundamental to our ability to measure it. In this case, we are counting "detailed requirements," a concept that will be explained in Chapter 4.

Even in the absence of historical data, the team predicts what the values of the metrics should and will be. With these advance predictions, teams tend to work better, and they tend to remember results. The data collected become the basis for future historical data. Managing all of this is not technically difficult, but it has

Table 5.1 A typical way to track known defects

Defect Tracking									
No.	Name	Description	Discovering engineer	Responsible engineer	Date opened	Source	Severity	Type	Status
1	Checkout flicker	Checkout screen 4 flickers when old DVDs are checked out by hitting the Checkout button.	Kent Bain	Fannie Croft	1/4/04	Integration	Med	GUI	Being worked; begun 2/10/04
2	Bad fine	Fine not correct for first-run DVDs checked out for 2 weeks, as displayed on screen 7.	Fannie Croft	April Breen	1/4/06	Requirements	High	Math	Not worked yet
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	Tested with suite 9023
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	Resolved
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

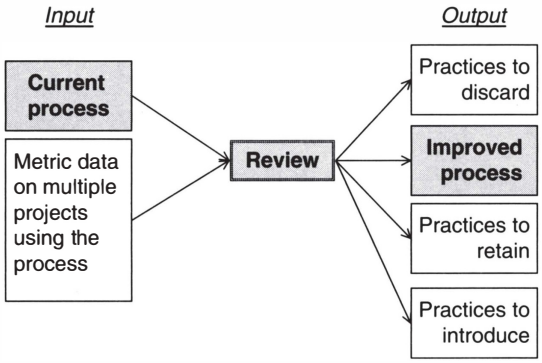


Figure 5.20 The process improvement meta-process

to be done at the same time as many other urgent activities. For this reason, the assignment of clear responsibilities and the regular review of the metric data are worked out at this early stage in the process. As you will see in the next section, process improvement feedback separates great development organizations from merely good ones.

Table 5.2 Data from multiple projects used to compare processes

Process →	Waterfall	Waterfall + Incremental	Process U	Process V
Average over 10 projects:				
Major defects identified within first 3 months per 1000SLOC in delivered product	1.3	0.9	0.7	2.1
Development cost per detailed requirement	$120	$100	$85	$135
Developer satisfaction index (1 to 10 = best)	4	3	4	3
Customer satisfaction index (1 to 10 = best)	4	6	6	2
Cost per maintenance request	$130	$140	$95	$165
Variance in schedule on each phase : $100 \times \frac{\text{actual duration} - \text{projected duration}}{\text{projected duration}}$	+20%	+70%	−10%	+80%
Variance in cost : $100 \times \frac{\text{actual cost} - \text{projected cost}}{\text{projected cost}}$	+20%	+65%	−5%	+66%
Design fraction : $\frac{total\ design\ time}{total\ programming\ time}$ Humphrey: Should be at least 50%.	23%	51%	66%	20%

1. **Identify and define** metrics team will use by phase.
   Include . . . **time spent on research, execution, and review**
   . . . **size (e.g., lines of code)**
   . . . **number of defects detected per unit (e.g., lines of code)**
   include source
   . . . **quality self-assessment of each on scale of 1–10**
   maintain bell-shaped distribution
2. **Document** these in the SQAP.
3. **Accumulate** historical **data** by phase.
4. Decide **where** the metric data will be **placed.**
   As the project progresses SQAP? SPMP? Appendix?
5. **Designate engineers** to manage collection by phase.
   QA leader or phase leaders (e.g., design leader)
6. **Schedule reviews** of data for lessons learned.
   Specify when and how to feed back improvement.

Figure 5.21 A process for gathering process metrics

Requirements Document: 200 detailed requirements	**Meeting**	**Research**	**Execution**	**Personal Review**	**Inspection**
Hours spent	0.5 × 4	4	5	3	6
% of total time	10%	20%	25%	15%	30%
% of total time: norm for the organization	*15%*	*15%*	*30%*	*15%*	*25%*
Self-assessed quality 1–10	2	8	5	4	6
Defects per 100	N/A	N/A	N/A	5	6
Defects per 100: organization norm	*N/A*	*N/A*	*N/A*	*3*	*4*
Hours spent per detailed requirement	0.01	0.02	0.025	0.015	0.03
Hours spent per detailed requirement: organization norm	0.02	0.02	0.04	0.01	0.03
Process improvement	**Improve strawman brought to meeting**		**Spend 10% more time executing**		
Summary	**Productivity: 200/22 = 9.9 detailed requirements per hour**				

Figure 5.22 Collecting project metrics for phases

5.8 ORGANIZATION-LEVEL QUALITY AND THE CMMI

Software development organizations periodically assess and upgrade their overall capability. In the 1980s the nonprofit Software Engineering Institute (SEI) established a classification of capabilities for contractors on behalf of the U.S. Department of Defense (DoD). The DoD's plan was to restrict bidding on government software development contracts to contractors with specified capability levels. The SEI's system, now expanded into the *Capability Maturity Model Integration* (CMMI), succeeded in providing concrete software engineering competence goals for organizations. Actually, the scope of CMMI is larger than software engineering, but we will restrict our attention to the latter, termed *CMMI-SW*. Many organizations, defense and commercial alike, have used the CMMI and its predecessor, the CMM, to assess the quality of their development process.

Software development organizations are assessed at a CMMI level between 1 (worst) and 5 (best). The CMMI distinguishes two kinds of assessments: *staged* and *continuous*. The staged assessment consists of identifiable steps. We will not discuss the continuous kind here. The CMMI classifies staged organizational capability with the steps shown in Figure 5.23 (from [7]). These are elaborated on in Table 5.3.

5.8.1 Level 1: Initial

The "Initial" CMMI level is the most primitive status that a software organization can have. Organizations at level 1 can be said only to be capable of producing software (i.e., "something"). The organization has no recognized process for software production, and the quality of products and projects depends entirely on the individuals performing the design and implementation. Teams depend on methods provided by members of the group who take initiative on the process to be followed. From project to project, these could be very good or very poor. The success of one project has little relation to the success of another unless they are similar and employ the same software engineers. When a project is completed, nothing is known or recorded about its cost, schedule, or quality. New projects are as uncertain of success as past ones.

For most organizations, this is unacceptable.

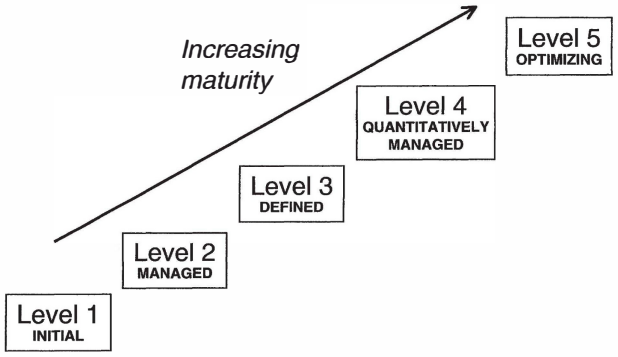


Figure 5.23 CMMI model for organization with staged processes

Table 5.3 SEI specifications of capabilities required for each level (see [7] for a full description)

Level, Title, and Summary	Expected Outcome	Characteristics
1. INITIAL		
Undefined; ad hoc	Unpredictable; depends entirely on team individuals	Organizations often produce products, but they are frequently over budget and miss their schedules.
2. MANAGED	Preceding level plus:	
Measurement and control	Project outcomes are qualitatively predictable	Respect organizational policies Follow established plans Provide adequate resources Establish responsibility and authority Provide training Establish configuration management Monitor and control: Take corrective action Evaluate process and product relative to plans: address deviations
3. DEFINED	Preceding level plus:	
Processes standardized	Projects consistent across the organization; qualitatively predictable	Establish process objectives Ensure process objectives met Establish orderly tailoring Describe processes rigorously Be proactive
4.QUANTITATIVELY MANAGED	Preceding level plus:	
Processes measured	Metrics available on process; quantitatively predictable	Set quantitative goals for key subprocesses Control key subprocesses with statistical techniques Identify and remedy variants
5. OPTIMIZING	Preceding level plus:	
Improvement meta-process	Processes improved and adapted using process metrics	Establish quantitative process improvement objectives Identify and implement innovative process improvements Identify and implement incremental process improvements Evaluate process improvements against quantitative objectives

5.8.2 Level 2: Managed

The "Managed" CMMI level applies to organizations capable of tracking projects as they unfold. Such organizations make plans, manage configurations, and maintain records of project costs and schedules. They also describe the functionality of each product in writing. It is therefore possible to predict the cost and schedule of very similar projects performed by a very similar team.

Although level 2 organizations track projects, no standard development process for the organization is guaranteed.

5.8.3 Level 3: Defined

The "Defined" CMMI level applies to organizations that document and enforce a standard process. Such a process is typically one of those described in the previous chapters (waterfall, spiral, etc.). Some organizations adopt existing standards such as IEEE's, while others define their own. Roughly speaking, as long as management enforces coordinated, professional standards, and engineers implement them uniformly, the organization is at level 3. Training is typically required for an organization to reach level 3. Teams are allowed flexibility to tailor the organization's standards for special circumstances. Level 3 includes organization-wide standards for project management, complete configuration management, inspections, design notation, and testing.

Level 3 organizations lack general predictive capability. They are able to make predictions only for projects that are very similar to ones performed in the past.

5.8.4 Level 4: Quantitatively Managed

The "Quantitatively Managed" CMMI level applies to organizations that can predict the cost and schedule of jobs. A common way to do this is to use statistical techniques and historical data. Such an organization classifies jobs and their components; it measures and records the cost and time to design and implement these; then it uses these metric data to predict the cost and schedule of subsequent jobs. As Humphrey has pointed out, this is not "rocket science," but it does require a significant amount of organization, as well as an ability to analyze the data. Level 4 requires a complete, metric-oriented quality plan.

Even though level 4 is a high level of capability, it is not the highest. Software engineering changes rapidly. For example, the object-oriented paradigm made rapid inroads in the 1990s: reuse and new component concepts are having a growing impact. Future improvements and paradigms are unpredictable. Thus, the capabilities of a level 4 organization that does not adapt and improve may actually decline over time.

5.8.5 Level 5: Optimizing

Rather than trying to predict future changes, it is preferable to institute permanent *procedures* for seeking and exploiting new and improved methods and tools. Level 5 organizations, at the "Optimizing" CMMI level, build in process improvement. In other words, their process (actually a meta-process) includes a systematic way of evaluating the organization's process itself, investigating new methods and technologies, and then improving the organization's process. Many organizations aspire to this impressive capability.

5.8.6 Relationship of the CMMI to the PSP, TSP

As we have seen, the CMMI is a measure of capability at the organizational/corporate level. Watts Humphrey and the Software Engineering Institute have also defined coordinated process models at the team level and at

the individual engineer level. These are called the *Team Software Process* (TSP) and the *Personal Software Process* (PSP), respectively. The PSP, TSP, and CMMI frameworks form a relatively coordinated set of capabilities and procedures at the individual, team, and organizational levels, respectively. We will describe the TSP in the project management chapters and the PSP in the implementation chapters.

5.8.7 Relationship of the CMMI to Agile Methods

CMMI appears to be almost contradictory to agility because it is process heavy. However, CMMI and agility are at different levels. CMMI is used to assess the capability of organizations as a whole to produce good applications. Those that use agile methods are just as interested in such an assessment as those that are not, and it is reasonable to assume that CMMI will evolve to assess development organizations of all kinds.

5.9 CASE STUDY: SOFTWARE QUALITY ASSURANCE PLAN FOR ENCOUNTER

The Software Quality Assurance Plan (SQAP) for Encounter is based on *IEEE Std 730-2002 Standard for Software Quality Assurance Plans*. The table of contents was outlined in Figures 5.8 and 5.9.

The table of contents of this SQAP follows that of IEEE standard 730-2002 in most essentials.

ENCOUNTER SOFTWARE QUALITY ASSURANCE PLAN

The author is indebted to his students for inspections of this document and for their improvements to the original.

> Note to the Student:
> Organizations usually have a standard quality assurance procedure. The SQAP for each project relies on this and provides project-specific QA plans. We will effectively combine two such documents here.

Approvals:

Title	Signature	Date
Engineering Manager	*P. Jones*	6/15/04
QA Manager	*L Wilenz*	6/11/04
Project Manager	*A. Pruitt*	6/7/04
Author	*E. Braude*	6/1/04

Revision History

> This assumes the existence of a method whereby revision numbers of documents are assigned.

Version 1

1.0.0 E. Braude: created 1/17/99

1.1.0 R. Bostwick: reviewed 5/30/99, added substance to Sections 7 through end

1.1.1 E. Braude: integrated and revised contents 5/31/99

Version 2

2.0.0 E. Braude: significant edits of Section 5.2 1/18/04

2.0.1 E. Braude: edits 1/23/04

2.1.0 E. Braude: edits in many sections responding to reviews by students 6/17/04

Table of Contents

1. Purpose
2. Referenced Documents

3. Management
   3.1 Organization
   3.2 Tasks
   3.3 Responsibilities
   3.4 QA Estimated Resources
4. Documentation
   4.1 Purpose
   4.2 Minimum documentation requirements
   4.3 Other
5. Standards
   5.1 Purpose
   5.2 Content
6. Reviews
   6.1 Purpose
   6.2 Minimum requirements
   6.3 Other reviews and audits
7. Test
   7.1 Unit Test
   7.2 Integration Test
   7.3 System Test
   7.4 User Acceptance Test
8. Problem Reporting and Corrective Action
9. Tools, Techniques, and Methodology
10. Media Control
11. Supplier Control
12. Records Collection, Maintenance, and Retention
13. Training
14. Risk Management
15. Glossary
16. Sqap Change Procedure and History

Appendix A – Metric History for this Project
Appendix B – Problem Reporting Form
Appendix C – Change Reporting Form
Appendix D – Inspection Meeting Report Form
Appendix E – Document Baselining Checklist

> Typically, most of the content of a SQAP are common to all of the organization's projects. Engineers would need simply to tailor parts to the project in question.

1. Purpose

This document describes the means by which the Encounter project will produce and maintain a high-quality product. It is also intended to describe mechanisms for improving the quality assurance process itself. "Quality" is defined as the degree to which the application satisfies its requirements.

> There is no separate "scope" section in the IEEE standard, so we have inserted it here. "Scope" contains vital information such as whether this document applies to the first release only or to maintenance as well.

The scope of this document comprises the artifacts of all releases.

2. Referenced Documents

See Section 4.2.

3. Management

3.1 Organization

> State the roles that are involved in ensuring quality. Actual names are provided in Section 3.3. The process described in this case is a mix of internal and external quality assurance.

This section assumes the description of the organizational structure of the Encounter project in 4.2 of the SPMP.

This document will refer to the quality work of developers as "internal." In addition, for the first three iterations of Encounter, a quality assurance engineer

will be designated who will be responsible to the manager of QA. The QA leader will take the lead for project-wide quality issues. QA tasks will be referred to as "external."

3.2 Tasks

> Summarize what needs to be done.

External QA tasks shall include, for all iterations:

- Maintaining this document
- Documenting the quality of the evolving product and associated artifacts
- Managing external review meetings
- Ensuring that verification takes place and logging verification
- Preparing for and attending all inspections
- Post-unit testing as per the Software Test Documentation
- Engaging in activities designed to improve the quality assurance process itself
- Keeping the project leader apprised of QA progress through weekly written reports
- Carrying out the audits specified in Section 6 of this document
- Providing the development team with feedback from QA's activities
- Assign defect repair to software engineers
- Identifying methods and tools for collecting and maintaining metrics

Internal QA tasks shall include, for all iterations:

- Each team member responsible for the quality of his or her work, as defined in this document
- Maintaining an issue database
- Collecting the metrics designate by this and other project documents
- Carrying out the reviews and inspections specified in Section 6 of this document
- Unit testing as per GCI manual 023.2

3.3 Responsibilities

> State what roles will fulfill what job functions.

It is the quality assurance leader's responsibility to see to it that the tasks in Section 3.2 are performed and to ensure that the prescriptions in this document are followed, including scheduling the reviews specified. For the first three iterations, the QA leader will perform all of the quality control (QC) functions.

For subsequent iterations, the QC team, appointed by the QA department manager, will take over this function. The quality leader and the QC team are responsible for all non-unit testing. (See Part VII of the book for background information on these types of tests). A description of the QC team will be supplied.

The project leader will be responsible for ensuring that inspections and unit tests are performed. The schedules are to be placed in the SPMP.

> No names are assigned to these roles here because they are in the Software Project Management Plan, which is their proper place. Duplicating a name here would require us to update more than one document when there are changes.

The leaders designated in the SPMP are responsible for the quality of their respective areas (requirements, design, etc.). They shall ensure that the designated metrics are collected and that the quality self-assessments as outlined in Section 5.2 of this document are conducted.

Each member of the Encounter development team is responsible for quality as specified in Section 4.5 of the company document "Quality responsibilities of engineers at GCI." This includes testing individual methods and combinations of methods in a class ("unit testing").

3.4 QA Estimated Resources

The estimated resources required for QA on Encounter are as follows:

- One engineer working half time for the first third of the project
- One engineer working full time for the second third of the project
- One engineer working full time for the last third of the project
- An additional engineer working half time for the last third of the project

4. Documentation

4.1 Purpose

The purpose of this section is to identify the documentation that will be used to ensure quality.

4.2 Minimum Documentation Requirements

> This section lists all of the project documentation, since the documentation is a major factor ensuring the quality of the product. Note that the User's Manual is excluded: it is a deliverable rather than a project document.

The following documents will be produced:

- Software Quality Assurance Plan (SQAP; this document)
- Software Configuration Management Plan (SCMP)
- Software Project Management Plan (SPMP)
- Software Requirements Specifications (SRS)
- Software Design Document (SDD)
- Software Test Documentation and the test document that it refers to (STD)
- Software Verification and Validation Plan (SVVP)
- Software Verification and Validation Report (SVVR) (*Note:* This book does not cover the SVVR.)
- Maintenance Plan

In addition to these documents, the Java source code will utilize *Javadoc* to generate package-, class-, and function-level documentation. (See http://java.sun.com/j2se/javadoc/ for Javadoc specifications.)

4.3 Other

—intentionally left blank

> It is customary to make a comment like this so that no one wastes time looking for what may be missing.

5. Standards, Practices, Conventions, and Metrics

5.1 Purpose

This section describes the standards, practices, conventions, and metrics to be used for the Encounter project. These are intended not only to ensure quality of the Encounter product but also to obtain quantitative metric data on the SQA process itself. These data are to be used to help elevate the CMMI level of Gaming Consolidated Industries (GCI) from level 2 to level 3.

5.2 Content

> Describe the standards, practices, conventions, and metrics to be used. Organization-wide quality goals can be supplied here or in a separate appendix. The contents of this section should be as specific as possible. For example, statements such as "quality should be as high as possible" should be avoided.

Standards:

The IEEE documentation standards as of July 1, 2004, with appropriate modifications, are to be used for all documentation. The standards for Javadoc commenting will be followed as found at http://java.sun.com/j2se/javadoc/writingdoccomments/index.html and http://java.sun.com/j2se/javadoc/writingapispecs/index.html. Documentation standards or templates developed by the company may supersede these at the discretion of management.

Unified Modeling Language standards, as specified in . . . shall be used in this project.

Refer to the Conventions section below for additional standards.

Practices:

1. Because delaying quality is expensive, GCI Inc. strongly encourages engineers to apply quality precepts while working, rather than as an afterthought. This is referred to in the company as "internal quality." It includes all unit testing.
2. GCI Inc.'s policy is that the QA department provides independent, external testing. The QA department at GCI also has a role in educating engineers in the practice of internal quality and working with project mangers to ensure that it takes place.
3. All project artifacts are inspected and are made easily available to the team once released by the developer.
4. All project artifacts are placed under configuration management, where the contents can be seen by anyone in the team at any time (see the Software Configuration Management Plan for details).
5. The development process is to be reviewed at least once for improvement, and the written results forwarded to the software engineering laboratory (see Section 6.2.10).

Conventions:

Where feasible, writing conventions should conform to the suggestions in *Writing for Computer Science: The Art of Effective Communication* by Justin Zobel (Springer Verlag.

The coding conventions as found at http://java.sun.com/docs/codeconv/html/CodeConventions.doc.html will be followed.

Metrics:

> This section is liable to be extensive. The numbers used here would be based on historical data obtained from the group that is developing Encounter.

GCI's list of standard metrics, found at http://xxx, should be gathered as specified there. The following includes some of them.

For every process and document, metrics shall include the following:

1. Time spent by individuals on preparation and review.
2. Number of defects per unit (e.g., lines of code), classified per Section 8 of this document.
3. Quality self-assessment of the QA process and performance on a scale of 1 through 10, approximately in a bell-shaped distribution; self-assessment scores will not be used for the evaluation of personnel by management; failure to produce them, however, may negatively affect the evaluation of an engineer by management, however.

> We would specify all metrics to be collected on this project. Possible metrics are described in Chapter 6.

The standard for defect classification is given in Section 8 of this document.

Quality Goals:

GCI quality goals for delivered products are as follows, measured in terms of defects detected within two months of delivery.

- No known "critical" or "serious" defects remain in any delivered artifact.

In addition:

- Requirements: No more than one "medium," and no more than three "trivial" defective detailed requirements per 100
- Design: No more than one "medium" defect per five diagrams. A "diagram" is any figure that uses about a page of easily legible parts.
- Pseudocode: No more than two "medium" defects per 1000 lines pseudocode is described in Chapter 19.
- Code: No more than two "medium" defects per KLoC (1000 lines of non-commented code)

The data from this project are to be reported as Appendix 1 to this document.

6. Reviews and Audits

6.1 Purpose

The purpose of reviews and audits is to continually focus engineers' attention on the quality of the application as it develops. *Reviews* effect this in a scheduled and thorough manner. *Audits* do so on the basis of random sampling with short notice.

6.2 Minimum Requirements

> Large projects require the full set of reviews and audit listed here. Student teams should try to conduct reviews and inspections of requirements and design, as well as postmortem reviews. "Reviews" are discussions of proposed artifacts. "Inspections" are conducted on completed artifacts presented to the team. The SQAP does not call out inspections as a heading, so the author has inserted section "A"'s for this purpose.

6.2.1 Software Requirements Reviews

These are walk-throughs of all proposed requirements in the presence of the entire team and at least one responsible customer representative. They will be led by the requirements leader, who will determine their frequency and scope.

6.2.1A Software Requirements Inspections

After they have been reviewed, all requirements will be inspected in accordance with GCI, Inc.'s document GCI 345678 "Inspections and Review procedures at GCI." Requirements sections must be completed and signed within a week of beginning the design.

6.2.2 Architecture Design Reviews

This is a review of alternative architectures with the entire team. The review will be led by the design leader in accordance with GCI 345678 on a frequency to be determined. The team will provide feedback, which will be reflected in the final design.

6.2.2A Architecture Design Inspections

After they have been reviewed, architectures will be inspected in accordance with GCI, Inc.'s inspection process manual, document GCI 345678. Architecture sections must be completed and signed off within a week of beginning detailed design.

6.2.3 Detailed Design Reviews

These are reviews of all proposed detailed designs in the presence of the entire development team. They will be led by the design leader, who will determine their frequency and scope, but at least one design review will be conducted per iteration. If possible, the architecture will be decomposed into detailed designs of its parts, and these will undergo separate detailed design reviews.

6.2.3A Detailed Design Inspections

After they have been reviewed, detailed designs will be inspected in accordance with GCI, Inc.'s inspection process manual, document GCI 345678. Detailed design sections must be completed and signed off within a week of beginning implementation.

6.2.3 Test Plan Reviews

These are reviews of all proposed test plans in the presence of the entire team. They will be led by the

QA leader, who will determine their frequency and scope. The test plan will be decomposed into parts, and these will undergo separate reviews.

6.2.3A Test Plan Inspections

After they have been reviewed, test plans will be inspected in accordance with GCI, Inc.'s inspection process manual, document GCI 345678. Test plan sections must be completed and signed off within a week of beginning testing.

6.2.4 Verification and Validation Plan Review

V&V is to be conducted by an independent team (i.e., not associated with QA), following the process detailed in GCI 345678. The QA engineer will review the SVV plan prior to its execution.

6.2.5 Functional Audits

The QA leader shall be responsible for auditing the product relative to the SRS. The audit will follow company guidelines in "GCI 8902: Release Procedures."

6.2.6 Physical Audits

Prior to each delivery, the QA leader is responsible for checking that the physical software and its documentation designated for delivery are complete and the correct version.

6.2.7 In-Process Audits

Project personnel should expect random audits of their work. These will consist of visits to the work site by individuals or teams designated by division management. A day's notice shall usually be given for all visits, but audits without notice shall take place as well. The subject of these audits will be the current work of teams and individuals that has been allocated to the project. All project artifacts will be made freely available to all team members and auditors at all times. It will be organized in a clear, standard fashion, so that audits will be possible without any notice.

6.2.8 Managerial Review

The Encounter project shall be reviewed by the VP for Engineering during the first week of every month: exceptions are at the discretion of the VP for Engineering. It is the project leader's responsibility to schedule this review and provide the appropriate documentation and software demonstrations.

6.2.9 SCMP Review

The QA leader shall review the status of configuration management on a monthly basis in a manner independent of the procedures specified in the SCMP.

6.2.10 Post-Implementation Review

As with all GCI projects, the Encounter team shall conduct post-implementation reviews to provide data for future projects. These will include reviews of the project phase just completed and reviews of the QA process itself. The QA team or QA leader shall file a process improvement report for every phase, and for the QA process itself, with the manager of the software engineering laboratory.

6.3 Other Reviews and Audits

—intentionally left blank

7. Test

> This section describes how testing is to be managed. The text here should refer to, but not duplicate, the software test documentation.

The responsibilities for testing were described in Section 3.3. Refer to the Software Test Documentation for details on testing Encounter.

8. Problem Reporting and Corrective Action

> This section explains how defects come to be recognized, described, and repaired. They do not follow the details of IEEE standards. The reader is referred to the IEEE standards, as well as to Humphrey [8] for additional defect severity and type classifications.

The team will use the Bugzilla defect management system.

> Instead of describing Bugzilla, we will describe a hypothetical manual defect system for the sake of clarity, even though a software-based system is common practice and far preferable. Such systems also allow for a dialog thread involving, typically, testers and developers.

The Problem Reporting Form to be used by the Encounter development team in response to a software problem report generated by QA is shown in Appendix B.

To use this form, engineers should retrieve the form from www.a.b.c.d. The defect number will appear automatically, and the site will ensure that the appropriate fields are filled in.

The values for *severity* are as follows:

- *Critical*: Causes the application to crash with significant frequency
- *Serious*: Causes at least one documented requirement to be unmet
- *Trivial*: Could be allowed to stand ad infinitum without impeding the user from exercising a required feature
- *Medium* : Is neither serious nor trivial

The documentation defect types are as follows:

- Incorrect
- Missing material
- Unclear
- Ambiguous
- Incomplete
- Redundant (within or between documents)
- Contradictory
- Obsolete

The code *and pseudocode* defects types are as follows:

- Syntax
- Logic
- Data (i.e., allows a wrong variable value)
- Insecure (allows unacceptable security breach)

The workflow of a defect status is:

1. Open: defect found by tester
2. Assigned: defect assigned to an engineer
3. Corrected: defect fixed by engineer
4. Closed: defect closed by tester

If the defect is reopened, the defect will move from the Corrected state to an Open state.

The QA leader will create and the QA team will maintain a database of problem reports that describe the deficiencies, discrepancies, and anomalies for Encounter. They will ensure that defects are consistently recorded on this form and that they are routed and repaired in a consistent manner. Problem reports shall be routed in accordance with the SCMP. Full traceability as to their effects and status shall be maintained, including after they are repaired.

After iteration three, when a problem is encountered, the QA manager will distribute the problem report to the members of the Change Control Board (CCB). For the first three releases, the configuration specialist will carry out the functions of the QA team, and the project leader will perform all of the CCB functions in accordance with the SPMP. The CCB evaluates the problem report and then assigns a priority to the report of either *immediate*, *to be done*, or *optional*. The problem report is then assigned by the CCB to the Encounter development team, QA, or CM for resolution. The CCB determines the schedule for problem report resolution based on problem report priority and analysis report results. After the problem in the report is corrected, the QA team reviews the results and the QA manager reports on the review to the CCB. If necessary, the process is repeated.

9. Tools, Techniques, and Methodologies

SQA techniques include the auditing of standards, requirements tracing, design verification, software inspections, and the verification of formal methods. The SQA tools consist of software verification programs, checklists, media labels, and acceptance stamps. Checklists will be obtained from the company's software engineering laboratory, and tailored for Encounter. These are augmented by NASA checklists at [9]. Checklists include the following:

- Review checklists are used at formal meetings, for document reviews, and for inspections.
- Checklists will be used for verifying the quality of the following activities and documents: Preliminary Design Review, Critical Design Review, Test Readiness Review, Functional Configuration Audit, Physical Configuration Audit, SRS, SDD, SPMP, and Software Development Folders.
- Separate checklists and forms are used for software audit purposes.

> This book contains checklists throughout. These checklists involve meeting and inspection procedures, for example. Teams often begin with published checklists, and augment them according to additional specific needs of their projects.

Additional SQA tools, techniques, and methodologies for configuration management are described in the SPMP.

10. Media Control

> Describe the means by which disks, tapes, and so on will be managed.

The SQA team verifies that the software media are built and configured per the SCMP and that authorized changes have been installed and tested. In addition, the SQA team verifies that the software media are duplicated using only the procedures identified in the SCMP. SQA acceptance is indicated by an SQA stamp on the media label. The SQA audit reports for media control are intended as further evidence that QA procedures have been followed. All backup media will be stored offsite as described in the SCMP.

11. Supplier Control

> This section concerns relationships with suppliers of software and hardware. It describes how and by whom these relationships are to be handled.

The SQA team verifies all commercial third-party products provided by the suppliers during incoming inspection by reviewing the packing slips that identify the products and their version numbers. The QA manager is responsible for ensuring that all third-party software and hardware meets the expected requirements. The products will be validated by the QA manager through installation and acceptance tests. A QA representative will be responsible for testing all new versions. He will also be responsible for the relationship with the external vendor.

12. Records Collection, Maintenance, and Retention

> This section describes how physical records will be handled and who will be responsible for them. Include disk files that are not under configuration control.

The SQA records collected and archived shall include the following:

- Task reports
- Anomaly reports not handled by the regular problem-reporting mechanism

- Memos, including recommendations to responsible parties
- Logbooks of SQA activities
- Audit reports
- Signed-off checklists from reviews and audits
- Minutes of inspections
- Metrics for the QA process itself

Besides verifying the archive procedures specified in the SCMP, SQA shall separately archive its own records at least once a week. These records are retained throughout the operation and maintenance phase.

13. Training

Includes SQA training specific to this project.

The SQA organization will conduct an initial four-hour orientation on quality for the development team. This will include a presentation on the metrics to be used and a workshop on how to use tools for recording metrics. SQA will also conduct monthly three-hour classes for development team members to keep them informed of quality goals, tools, and techniques. Team members can waive attendance at these meetings by scoring perfectly on a multiple-choice quiz available at GCI/monthly/SQA/quiz.

Each team member is required to attend a three-hour course on writing styles and conventions conducted by QA.

14. Risk Management

SQA team members are encouraged to identify risks as early as possible and direct them to the project leader. The procedures for risk management are specified in Section 5.4 of the SPMP.

15. Glossary

. . . .

16. SQAP Change Procedure and History

. . .

(The material in the following appendices was supplied by Jane Dyson.)

Appendix A: Metric History for This Project

	Defects			Preparation Time		Review Time	
Product	Defect Unit	Expected Rate	Actual Rate	Expected Time	Actual Time	Expected Time	Actual Time
SCMP	Section	1 minor defect per section					
SPMP	Section	1 minor defect per section					
SQAP	Section	1 minor defect per section					
SRS	Section	1 minor defect per section					
SDD	Section	1 minor defect per section					
STP	Section	1 minor defect per section					
SVVP	Section	1 minor defect per section					
SVVR	Section	1 minor defect per section					
Requirements	D-Requirement	1 medium and 3 trivial defects per 100					
Design	Diagram	1 minor defect per 5					
Pseudocode	KLOC	2 medium defects					
Code	KLOC	2 medium defects					

Code Defects

A free online tool called "RefactorIt" from http://www.refactorit.com shall used by the Project Manager to estimate LOC, NCLOC, and CLOC.

Appendix B: Problem Reporting Form

1. Defect number:_____
2. Proposed by: _____
3. Documents / sections affected: _____
4. Document defect type:
   a. _____ Missing material
   b. _____ Unclear
   c. _____ Ambiguous
   d. _____ Incomplete
   e. _____ Redundant (within or between documents)
   f. _____ Contradictory
   g. _____ Obsolete

   Source code affected (for source code defects):
5. Package(s) _____
6. Class(es)_____
7. Method(s)_____
8. Severity:
   a. _____ High
   b. _____ Medium
   c. _____ Low
9. Code defect type:
   a. _____ Syntax
   b. _____ Logic
   c. _____ Data (i.e., allows a wrong variable value)
   d. _____ Insecure (allows unacceptable security breach)

10. Phase injected (earliest phase with the defect):

    a. _____ Requirements

    b. _____ Architecture

    c. _____ Detailed design

    d. _____ Code

    e. _____ Implementation

11. Detailed description: _____

12. Priority: a. _____ Immediate b. _____ Intermediate c. _____ Deferred

13. Resolution: _____

14. Status: a. _____ Closed b. _____ Open

    Sign-off:

15. Description and plan inspected: _____

16. Resolution code and test plan inspected: _____

17. Change approved for incorporation: _____

Appendix C: Change Reporting Form

1. Change number: _____
2. Proposed by: _____
3. Documents/sections affected: _____
4. Reason for change or addition: _____

    . . .

5. Enhancement: _____

    _____

    . . .

6. Details: _____

    . . .

    Signatures:

    Description and plan inspected (QA team leader) _____

    Resolution code and test plan inspected (QA team leader) _____

    Change approved for incorporation (Team leader) _____

Appendix D: Inspection Meeting Report Form

A completed Inspection Meeting Report must be included at the end of each document. Only the latest report is required.

Inspection Meeting Report

PROJECT:		
DATE:	STARTING TIME:	ENDING TIME:
PRODUCED BY:		
TYPE OF INSPECTION:	☐ INITIAL INSPECTION	☐ REWORK INSPECTION
DOCUMENT NUMBER AND REVISION:		
BRIEF DESCRIPTION:		
APPRAISAL OF THE WORK UNIT: ☐ Inspection not completed (continuation scheduled for _____) ☐ No further inspection required ☐ Minor revisions required ☐ Major revisions required		
MATERIALS PRODUCED ☐ Issues List (Not Author's Responsibility) ☐ Issues List for Author ☐ Traceability Issues?		
Participant	**Role(s)**	**Org./Dept.**
	Author	
	Moderator	
	Recorder	
	Inspector	
	Inspector	
	Inspector	
	Inspector	
	Inspector	

Inspection Meeting Report Issues List

ISSUES, NOT FOR AUTHOR			
Issue	Issue must be resolved prior to Document Baseline? Y/N	Assigned To	Response
1.			
2. etc.			

ISSUES FOR AUTHOR	
Issue	Response
1.	
2. etc.	

Appendix E: Document Baselining Checklist

Document Baselining Checklist

	Item	Comment / Initials
☐	Inspection Meeting Report completed and all issues resolved.	
☐	Revised document sent out for consensus after Inspection.	
☐	All comments accepted via Word's Track Changes. Track changes feature then turned off.	
☐	Check for clean requirements traceability completed. (*SRS only*)	
☐	All high-level requirements have been allocated to detailed requirements. (*SRS only*)	
☐	Final traceability report generated and stored. (*SRS only*)	

5.10 SUMMARY

Quality practices are integrated throughout the development process. They start with planning, when specific documents such as the Software Quality Assurance Plan (SQAP) and the Software Verification and Validation Plan (SVVP) are produced, describing the quality policies, procedures, and practices to be implemented.

The SQAP is the master quality plan and specifies the people and organizations responsible for quality, the project documentation to be produced, the metrics to be collected, the procedures to be practiced, and the techniques to be implemented. It is an important document as it sets the expectations regarding quality early in a project and helps guide its implementation.

Inspections are an important quality technique that involve peer review of all project artifacts including documents, test plans, and code. Their goal is to identify defects as close to their introduction as possible so that they can be repaired quickly. Considerable research has shown that the cost to repair a defect increases dramatically the longer it is allowed to persist in the software.

Quality reviews and audits by a quality assurance (QA) organization are beneficial. An external QA group is independent of the people developing the software and other artifacts and can objectively assess quality. Again, assessing quality and addressing it throughout the development process helps identify problems as early as possible.

Problems are identified by team members throughout the development process and are submitted as *defects*. A screening process is implemented to ensure that the defects are accurate and valid. A classification system is employed to understand the severity of each defect and priority for its repair. In addition, maintaining metrics such as number of defects, time to repair, and so on is useful for comparison with other projects and in making estimates for future projects.

Improving the effectiveness of the overall software process is accomplished through implementation of a meta-process. This includes the collection of process metrics, and meetings at the end of projects phases to analyze the metrics. In addition, lessons-learned meetings are conducted at the end of a project to analyze project successes and identify areas of improvement.

The Software Engineering Institute has developed a comprehensive meta-process for assessing an organization's overall capability. The process is called the Capability Maturity Model Integration (CMMI), and it defines several levels of capability and maturity an organization can measure itself against. The levels range from the lowest, Level 1 Initial, to the highest, Level 1 Optimizing. At the Initial level an organization can produce software but has no recognized process. At the Optimizing level, an organization implements a meta-process for process improvement.

5.11 EXERCISES

1. In a paragraph, explain why it is important to document quality procedures at the beginning of a project rather than later on.
2. The number of people attending an inspection can have a direct impact on the effectiveness of the review. List the disadvantages of having either too few or too many people attend an inspection.
3. Give two advantages and two disadvantages to using standards for documentation of the various software phases.
4. Why is it generally a good idea to have a cross-functional group be part of an inspection team? What type of review (that is, during what development phase) might it not be a good idea?

5. Your instructor will pair up student project teams. Conduct an inspection of an artifact produced by the other team, such as a SCMP or SPMP. Use an inspection checklist, such as one found in [3], to guide your inspection.
6. Give an example of a defect that might be classified with a high severity but a low priority. Be specific with your answer.
7. (a) In your own words, describe each of the CMMI levels.
   (b) How does applying the CMMI levels promote organizational quality? Explain this in a paragraph or two, using your own words.

BIBLIOGRAPHY

1. Fagan, M. "Design and Code Inspections to Reduce Errors in Program Development." *IBM Systems Journal*, Vol. 15, No. 3, 1976, pp. 182–211.
2. Wiegers, Karl, "Improving Quality Through Software Inspections," *Process Impact*, 1995. http://www.processimpact.com/articles/inspects.html [accessed November 15, 2009].
3. Wiegers, Karl, "Goodies for Peer Reviews," *Process Impact*, 'http://www.processimpact.com/pr_goodies [accessed November 15, 2009].
4. Gehani, Narain, and A. McGettrick, *"Software Specification Techniques,"* International Computer Science Series, Addison-Wesley, 1985.
5. Gilb, T., and D. Graham, *"Software Inspection,"* Addison-Wesley, 1993.
6. Bugzilla. http://bugzilla.org/about.html [accessed December 10, 2009].
7. Capability Maturity Model® Integration (CMMI℠), Version 1.1, http://www.sei.cmu.edu/pub/documents/oz-reports/pdf/02trf012.pdf [accessed December 8, 2009].
8. Humphrey, Watts S., *"A Discipline for Software Engineering,"* SEI Series in Software Engineering, Addison-Wesley, 1995.
9. NASA Goddard Space Flight Center Software Assurance Web site. http://sw-assurance.gsfc.nasa.gov/disciplines/quality/index.php (2006) [accessed November 15, 2009].