CHAPTER 7

Inspection as an Up-Front Quality Technique

Don O'Neill

7.1 Origin and Evolution

Software inspections are considered a best industry practice for detecting software defects early and learning about software artifacts. Software inspections and software walkthroughs are peer reviews and are integral to software product engineering activities. A collection of coordinated knowledge, skills, and behaviors facilitates the best possible practice of peer reviews. Software inspections are the most rigorous form of peer reviews and fully utilize the elements of practice in detecting defects. These elements include the structured review process, standard of excellence product checklists, defined roles of participants, and the forms and reports. Software walkthroughs draw selectively upon the elements in assisting the producer to obtain the deepest understanding of an artifact and reaching a consensus among participants. Measured results reveal that software inspections produce an attractive return on investment obtained through accelerated learning and early defect detection. For best results, they are rolled out within an organization through a defined program of policy and procedure preparation, practitioners and managers training, measurement definition and collection within a database structure, and roll out of a sustaining infrastructure.

Software inspections provide value by improving reliability, availability, and maintainability [1]. IBM Corporation originated and adopted software inspections in the early 1970s and recognized Michael Fagan with an Outstanding Contribution Award for his pioneering work [2, 3]. Software inspections are known to add economic value in detecting and correcting defects early at greatly reduced cost [4]. IBM reported an 83% defect detection rate resulting from software inspections practice; AT&T Corp., 92% [5].

Gerald Weinberg and Daniel Freedman gave real thought to the dynamics of the software inspection role players providing deep and interesting insights useful to practitioners [6]. Robert Ebenau provided leadership in the roll out of software inspections at AT&T Corp. and documented his knowledge [7], as did Tom Gilb and Dorothy Graham [8].

The Software Engineering Institute (SEI) identified software inspections as an industry practice essential to managing the software process [9] and offered

practitioner training [5, 10]. Peer reviews are included in the SEI Capability Maturity Model® (CMM®) for Software as a level 3 key process area [11]. The ongoing Capability Maturity Model Integration® (CMMI®) project spanning software, systems engineering, and integrated product development includes peer reviews in its product verification process area.

7.2 Context of Use

Software inspections are considered a best industry practice for use on software projects. Senior managers consistently ranked peer reviews as the significant enabler of software product quality among all key process areas [12]. These are integral to the software product engineering life-cycle activities associated with software requirements and specifications, designs and code, and test plans and procedures [11]. The best practices for software management and engineering on the project and the context of use for peer reviews are shown in Figure 7.1.

7.3 Scope

Peer reviews are composed of software inspections and software walkthroughs [13, 14]. Software inspections are the most rigorous form of peer reviews. Both software inspections and software walkthroughs are composed of a collection of coordinated knowledge, skills, and behaviors associated with process, standards, roles,

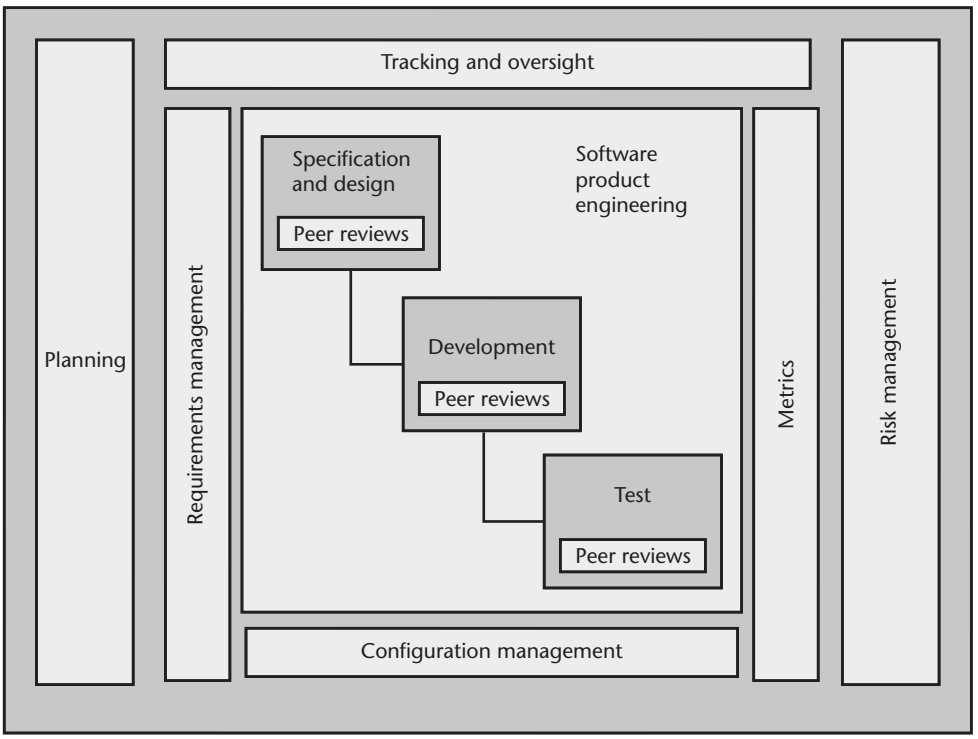

Figure 7.1 Best software practices.

and measurement. Peer reviews are conducted as an integral part of each life-cycle activity. See Figure 7.2.

7.3.1 Software Inspections and Walkthroughs Distinguished

Peer reviews are a group activity organized to systematically reason about a software artifact. There are two types of peer reviews: software walkthroughs and software inspections. Each of these serves different purposes. Software walkthroughs are an informal review used to confirm the understanding of the producer and validate the approach being taken. Software inspections are a formal review used to verify that the artifact complies with the standard of excellence. In a life-cycle activity, the software inspection is the exit criteria or gate that concludes the activity. See Table 7.1.

7.3.1.1 Software Walkthrough

The software walkthrough is organized to serve the needs of the producer or author of the software artifact in acquiring superior knowledge of all aspects of the software artifact. It is a learning experience. A desirable side effect of the software walkthrough is the forging of a shared vision among the reviewers and consensus among participants on the approaches taken, product and engineering practices applied, completeness and correctness of capabilities and features, and rules of construction for the domain product. Since the software walkthrough caters to the needs of the author, it is the author who initiates the session. Consequently, there may be several walkthroughs in each life-cycle activity. Software walkthroughs yield open issues and action items. While these issues and action items may be tracked to closure, the only measurement taken is a count of the software walkthroughs held.

7.3.1.2 Software Inspection

The software inspection is structured to serve the needs of quality management in verifying that the software artifact complies with the standard of excellence for software

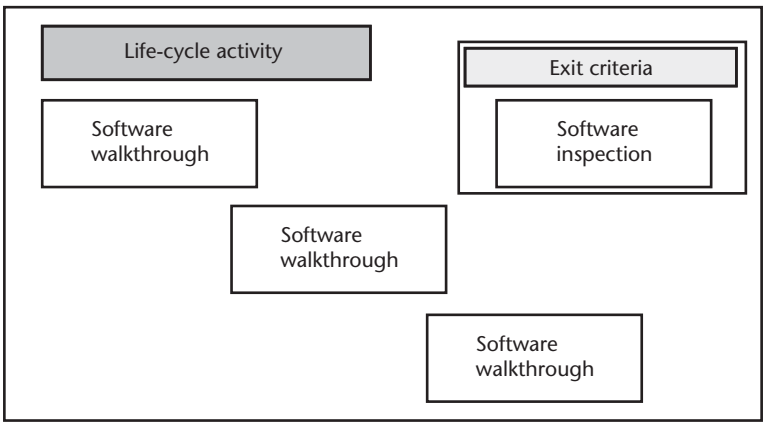

Figure 7.2 Life-cycle activity and peer reviews.

Table 7.1 Peer Reviews Scope

Focus	*Software Inspection*	*Software Walkthrough*
Purpose	Do the job right	Do the right job
	Detect defects	Learning
	Conformance	Consensus
Initiation	Exit criteria for life-cycle activity	Author request
Measurement	Product and process measurements	Instances

engineering artifacts. The focus is one of verification, on doing the job right. The software inspection is a formal review held at the conclusion of a life-cycle activity and serves as a quality gate with an exit criteria for moving on to subsequent activities.

The software inspection utilizes a structured review process of planning, preparation, entry criteria, conduct, exit criteria, report out, and follow-up. It ensures that a close and strict examination of the product artifact is conducted according to the standard of excellence criteria, which spans completeness, correctness, style, rules of construction, and multiple views and may also include technology and metrics. This close and strict examination results in the early detection of defects. The software inspection is led by a moderator and assisted by other role players including recorder, reviewer, reader, and producer. The software inspection is initiated as an exit criteria for each activity in the life cycle. Product and process measurements are recorded during the software inspection session and recorded on specially formatted forms and reports. These issues and defects are tracked to closure.

7.4 Elements

The software inspections process is made up of several elements: the structured review process, standard of excellence, system of checklists, defined roles of participants, and forms and reports. See Tables 7.2 and 7.3.

1. A structured review process is a systematic procedure integrated with the activities of the life-cycle model. The process is composed of planning, preparation, entry criteria, conduct, exit criteria, reporting, and follow-up [2, 15].
2. A system of checklists governs each step in the structured review process and the review of the product itself, objective by objective. Process checklists are used as a guide for each activity of the structured review process. Product checklists house the strongly preferred indicators that set the standard of excellence for the organization's software products [16].
3. The role of each participant in the structured review process is defined. The roles include the moderator, producer, reader, reviewer, recorder, manager, and consumer. Each role is characterized by particular skills and behaviors [6].
4. Forms and reports provide uniformity in recording issues at all software inspections, reporting the results to management, and building a database useful in process management. Data collection utilizes three recording instruments: Inspection Record, Inspection Reporting Form, and Report

Table 7.2 Peer Reviews: Entry, Task, Verification, Exit

Entry Criteria	*Task*	*Exit Criteria*
Product artifact	Structured review process	Forms and reports
Participants in defined roles		
Standard of excellence		
	Verification	
	Product checklists	
	Process checklists	

Table 7.3 Elements of Peer Reviews

Elements	*Software Inspection*	*Software Walkthrough*
Structured review process	Planning	Planning: optional
	Preparation	
	Conduct	Conduct
	Report out	
	Follow up	Follow up
Standard of excellence	Completeness	Completeness
	Correctness	Correctness
	Style	
	Rules of construction	Rules of construction
	Multiple views	
		Product and engineering practice
Defined roles of participants	Moderator	Moderator: optional
	Recorder	
	Producer	Producer
	Reviewer	Reviewer
	Reader	
Forms and reports	Inspection record	
	Inspection reporting form	
	Report summary form	
		Open issues
		Action items

Summary Form. The results of software walkthroughs are recorded as open issues and action items [7].

7.4.1 Structured Review Process

The activities of the structured review process are organized for software inspections. Software walkthroughs may employ variations for planning, conduct, and follow-up.

7.4.1.1 Planning

The structured review process begins early in the project when the manager plans for software inspections of requirements, specifications, designs, code, and test procedures. The schedule for each software inspection is recorded in the project's

software development plan (SDP) or project plan. A trained moderator is assigned to each software inspection, and moderator training is scheduled as necessary. The Software Quality Assurance (SQA) Plan also discusses the use of software walkthroughs and software inspections in terms of their contribution to product verification and validation.

7.4.1.2 Preparation

Preparation is initiated by the moderator a week before the inspection session. The readiness of the product for inspection is assessed by the moderator and the producer. The moderator obtains the reviewers, recorder, and reader and briefs them on their roles along with the key principles of software inspections.

In assessing product volatility, the moderator ascertains the status of the baseline change activity and the completion of the preceding life-cycle activity. The producer conducts a brief overview of the product to be inspected to assist the inspection team in their preparation for the inspection session.

The inspection materials are distributed to team members, the time and place for the inspection session are announced, and reviewers are encouraged to prepare individually for the inspection session. Individual preparation is the mother's milk of the software inspections process.

7.4.1.3 Entry Criteria

Entry criteria are checked by the moderator at the start of the inspection session. Before the conduct activity begins, the moderator determines that the software product is ready to be inspected and the inspection team is prepared to inspect it. The moderator again assesses the product volatility indicators. Inspection team members are asked for their preparation effort, and the recorder notes this information. Where the entry criteria are not satisfactorily met, the moderator may reschedule the inspection session.

The moderator script for directing the entry criteria includes:

1. Has the preceding life-cycle activity been concluded?
2. Are there any changes to the baseline?
3. Are review participants in place and briefed?
4. Have all participants received all the review materials and checklists?
5. How many minutes of preparation did each participant perform?

7.4.1.4 Conduct

The inspection session including entry, conduct, and exit is directed by the moderator and attended by the producer, reviewers, recorder, and reader. The manager does not attend. Some key principles govern the inspection session:

1. The inspection is limited to periods of peak concentration (1 to 2 hours).
2. The product is reviewed, not the producer.
3. Issues are identified, not proposed solutions.

Each product component is inspected using the strongly preferred indicators found in the appropriate software product checklists. Each inspection team member in turn is asked if there is an issue to be raised for the product component and product checklist now before the group. If so, the issue is stated, discussed, and recorded. The producer may wish to obtain clarification of the issue at the time it is raised, but there is no need for the producer to defend or even explain the approach taken. The producer will have the opportunity to resolve the issue during the follow-up activity.

The moderator script for directing the conduct activity includes:

1. Are there any issues in completeness?
2. Are there any issues in correctness?
3. Are there any issues in style?
4. Are there any issues in rules of construction?
5. Are there any issues in multiple views?

7.4.1.5 Exit Criteria

Exit criteria are checked by the moderator at the close of the inspection session. The moderator verifies that all product components have been inspected and that the intended product checklists have been utilized. The recorder verifies that all the metrics have been recorded including preparation effort of each team member, the duration of the inspection session, and the size of the product being inspected as well as the defect type, defect category, defect severity, and defect origin for each issue raised. Finally, the moderator asks the producer for any closing comments, permitting the producer to have the last word.

The moderator script for directing the exit criteria includes:

1. Have all product elements been inspected?
2. Have all checklists been processed?
3. Have the inspection results been recorded?
4. Have metrics been collected?
5. Would the recorder read back the issues?
6. What should be the disposition of the inspection?
7. Would the producer like an opportunity to comment?

7.4.1.6 Reporting

The moderator, with the help of the recorder, reports the findings of the inspection session to the manager within a week. This report provides a review summary, the preparation effort and conduct time expended, the types of defects detected, and follow-up recommendations.

7.4.1.7 Follow-Up

The follow-up rework on the product is performed by the producer. The follow-up actions are prepared jointly by the manager and the producer and entered in the project action log. As these follow-up actions are completed, the action log reflects

the closure. Tracking issues to closure is an important indicator of software process maturity.

7.4.2 System of Checklists

Checklists are at the heart of software inspections. They fuel the structured review process and form the standard of excellence expected for the software product. Checklists provide the criteria for evaluating the quality of the product as well as progress within the process. Process and product checklists promote uniformity in the use of software inspections throughout a project and across an organization. Process checklists are used as a guide for each activity of the structured review process to ensure that the software inspections process runs smoothly. Product checklists provide reviewers with the thorough technical focus needed to guide the review of each product component from all viewpoints [10]. The system of checklists helps to overcome human limitations on information processing by focusing on just one consistent perspective at a time. See Tables 7.4 through 7.8 for examples of Requirements, Specification, Architecture, Design and Code, and Test Procedure Checklists [10].

Table 7.4 Requirements Checklist

Completeness
1. Do the requirements specified carry out the mission in a consistent fashion?
2. Do the requirements include the essential needs of the user, operational, and maintenance communities?
3. Does each requirement stand alone or have clearly stated dependencies?
4. Is the requirements document complete with all TBDs eliminated?
5. Are any requirements missing?
6. Are necessary requirements distinguished from those that are simply nice to have?
Correctness
1. Does each requirement not conflict with any other requirement?
2. Does the organization of the requirements facilitate traceability to design and code?
3. Is each specific requirement identified by unique paragraph number?
Style
1. Are the requirements clearly understandable?
2. Are changes clearly distinguished between editorial and functional?
3. Is the nomenclature of terms and their definition complete?
Rules of Construction
1. Are requirements that are likely to change and evolve distinguished from those that are likely to be stable?
2. Has all available tool assistance been applied it the production, analysis, and review the requirements?
3. Does the requirements document follow the documentation standard?
4. Is each requirement testable?
Multiple Views
1.Have users participated in the production, analysis, and review of these requirements?
2. Have operational personnel participated in the production, analysis, and review of these requirements?
3. Have maintenance personnel participated in the production, analysis, and review of these requirements?
4. Have software engineering designers participated in the production, analysis, and review of these requirements?
5. Has the software development team participated in the production, analysis, and review of these requirements?
6. Have test engineers participated in the production, analysis, and review of these requirements?
7. Has the test team participated in the production, analysis, and review?

Table 7.5 Architecture Checklist

Completeness
1. Is the commonality of functions and components identified? 2. Is permissible variability of inputs and parameters defined?
Correctness
1. Are responses derivable from stimuli? 2. Are responses and stimuli identified for all permissible states and modes? 3. Are adaptation parameters governing permissible variability properly defined?
Style
1. Are architecture artifacts recorded in accordance with project selected templates?
Rules of Construction
1. Does the architecture utilize the underlying algorithms and data stores intrinsic to any existing domain specific reference architecture? 2. Are guidelines for commonality followed to facilitate alignment of interfaces and components and preserve open system possibilities? 3. Does the architecture provide for explicit identification of reusable components and commercial off-the-shelf products choices? 4. Does the architecture support reasoning about scalability and capacities?
Multiple Views
1. Does the architecture support reasoning about performance? 1.1 For each stimulus mode, source and frequency are identified. 1.2 For architectural adaptations computer resources, their arbitration (queuing policy), and loading are identified. 1.3 For each response latency, throughput, and precedence are identified. 2. Does the architecture support reasoning about modifiability? 2.1 Anticipated or likely additions, modifications, and deletions are appropriately encapsulated and separated. 3. Does the architecture support reasoning about availability and ensuring continuous operation including hardware and software faults and failures, hardware and software redundancy, service levels, and fail soft backup?
Technology
1. Have appropriate logical structures been specified? 2. Are the principles of good software engineering applied including: abstraction, information hiding, and separation of concerns? 3. Have the appropriate mechanisms been used to present information including text, tables, lists, matrices, equations, logical diagrams, diagrams, and pictures?

7.4.2.1 Completeness

Completeness is based on traceability among software product artifacts of various types including requirements, specifications, designs, code, and test procedures. Completeness analysis may be assisted by tools that trace the components of a product artifact of one type to the components of another type. Completeness analysis of predecessor and successor artifacts reveals what sections are missing and what fragments may be extra. A by-product of the completeness analysis is a clear view of the relationship of requirements to the code product: straightforward (one to one), simple analysis (many to one), and complex (one to many).

The moderator script for inquiring about completeness includes:

Table 7.6 Specification Checklist

Completeness

1. Scope and Traceability
   1.1 Is each operational capability fully described in terms of its purpose and function?
   1.2 Are all identified functions and data traceable to the higher level specification?
2. Completeness of Detail
   2.1 Is the decomposition of each identified function sufficiently fine-grained and uniquely allocated to a physical component?
   2.2 Have all TBS and TBD indicators been replaced with the required information?
   2.3 Are all logical operations specified completely?
   2.4 Are all exception conditions explicitly identified and all necessary error processing defined?
3. Data and Interface
   3.1 Has the necessary and sufficient set of input data been specified for each function?
   3.2 For each function specified, are the required output data derived from the input data and accessible retained data?
   3.3 Is the boundary of each identified function described in terms of all required input data and all required output data?
   3.4 Is all the data identified that must be retained within the system for reference?
   3.5 Is each interfunction data flow or signal accounted for by both senders and receivers?
   3.6 Are the semantics for each interfunction data flow or signal the same for all senders and all receivers?
   3.7 Does each data item have appropriately specified initialization values and required range of permissible values?
4. Testability
   4.1 Does each requirement have a corresponding test requirement that identified how it is to be verified and the required testing limits? (Verification methods include: by test and review of data, by inspection or observation, by data collection and analysis, or not required.)
   4.2 Has the appropriate level of test been identified for verifying each requirement? (Levels of test include: unit, computer program, element, or system test level.)
   4.3 Have the stress test and regression test requirements for each test requirement been described?

Correctness

1. Have the higher level requirements been allocated correctly and appropriately to the specification?
2. Does each function specification and its inputs correctly produce the required outputs?
3. Is each equation correctly specified for performing the needed computations?
4. Is each logical operation correctly defined?
5. Is each interface signal identified in the interface design specification properly reflected in the specification?
6. Is each external interface signal in the specification properly reflected in the respective interface design specification?

Style

1. Does the format of the specification follow the documentation style guide?
2. For each mnemonic used, is it defined when first used and entered in the glossary with the page number of this definition?

Rules of Construction

1. Is interprocessor communication properly addressed?
2. Are interfunction notifications properly addressed?
3. Are alternate modes considerations included?
4. Are data representation considerations property accounted for?
5. Are design constraints properly accounted for?
6. Are control requirements properly accounted for?

Multiple Views

1. Have initialization considerations been considered?
2. Has computer resource loading been assessed including memory, timing, and I/O?
3. Have alternate mode considerations been properly addressed?
4. Are the requirements sufficiently complete to support high level design?
5. Have user interface and display impacts been properly accounted for?

Table 7.6 (continued)

Technology

1. Have appropriate logical design structures been specified?
   1.1 Are the principles of good design applied including: abstraction, information hiding, and separation of concerns?
2. Have the appropriate mechanisms been used to present information including text, tables, lists, matrices, equations, logical diagrams, diagrams, and pictures?

1. Has traceability been assessed?
2. Have all predecessor requirements been accounted for?
3. Were any product fragments revealed not to have traceability to the predecessor requirements?
4. Was traceability found to be straightforward, simple, or complex?

7.4.2.2 Correctness

Correctness is based on reasoning about programs through the use of informal verification and correctness questions derived from the prime constructs of structured programming and their composite use in proper programs [17, 18]. Input domain and output range are analyzed for all legal values and all possible values. State data is similarly analyzed. Adherence to project-specified disciplined data structures is analyzed. Asynchronous processes and their interaction and communication are analyzed [19].

The moderator script for inquiring about correctness includes:

1. Is the function commentary satisfied?
2. Are programs limited to single entry and single exit?
3. Is the loop initialized and terminated properly?
4. Does the input domain span all legal values?
5. Is there systematic exception handling for illegal values?
6. Are disciplined data structures used?

7.4.2.3 Style

Style is based on project-specified style guidance. This guidance is expected to call for block-structured templates. Naming conventions and commentary are checked for consistency of use along with alignment, highlighting, and case. More advanced style guidance may call for templates for repeating patterns and semantic correspondence among software product artifacts of various types.

The moderator script for inquiring about style includes:

1. Are style conventions for block structuring followed?
2. Are naming conventions followed?
3. Are style conventions for commentary followed?
4. Are the semantics of the product component traceable to the requirements?
5. Are templates used for repeating patterns?

Table 7.7 Design and Code Checklist

Completeness

1. Has traceability been assessed?
2. Has available tool assistance been applied in assessing traceability?
3. Have all predecessor requirements been accounted for?
4. Were any product fragments revealed not to have traceability to the predecessor requirements?
5. What is the relationship of requirements to product component:
   5.1 One to one <straightforward>?
   5.2 Many to one <simple analysis>?
   5.3 One to many <complex>?

Correctness

1. Are structured programming prime constructs used correctly:
   1.1 Sequence: Is the function commentary satisfied for the sequence?
   1.2 If-then: If the test is true, is the function commentary satisfied?
   1.3 If-then-else: If the test is true, is the function commentary satisfied?
   1.4 While-loop: If the condition is true, is the function commentary satisfied? Does the loop terminate?
   1.5 Loop-until: If the condition is false, is the function commentary satisfied? Does the loop terminate? Is a one time loop acceptable?
   1.6 For-do: Is the function commentary satisfied? Are there discrete steps through the loop? Is the control variable not modified in the loop? Is the loop initialized and terminated properly?
   1.7 Case: For each leg, is the function commentary satisfied? Is the domain partitioned exclusively and exhaustively?
2. Are proper programs composed of multiple prime programs limited to single entry and single exit?
3. Are disciplined data structures used to manipulate and transform data?
4. Does the input domain span all legal values?
5. Does the input domain span all possible values, with systematic exception handling for illegal values?
6. Does the output range span all legal values?
7. For modules, does the state data span all legal values?

Style

1. Are style conventions for block structuring defined and followed?
2. Are naming conventions defined and followed?
3. Are the semantics of the product component traceable to the requirements?
4. Are style conventions for commentary defined and followed?
5. Are style conventions for alignment, upper/lower case, and highlighting defined and followed?
6. Are templates used for repeating patterns?

Rules of Construction

1. Are guidelines for program unit construction followed?
2. Is the interprocess communication protocol followed?
3. Are data representation conventions followed?
4. Is the system standard time defined and followed?
5. Are encapsulation, localization, and layering used to achieve object orientation?
6. Is logical independence achieved through event driven and process driven paradigms, late binding, and implicit binding?
7. Is scalability achieved through uniformity, parameterization, and portability?
8. Are fault tolerance, high availability, and security achieved?

Multiple Views

1. Has the logical view of user interface and object orientation considerations been assessed?
2. Has the static view of packaging considerations been assessed including program unit construction, program generation process, and target machine operations?
3. Has the dynamic view of operational considerations been assessed including communications, concurrency, synchronization, and failure recovery?
4. Has the physical view of execution considerations been assessed including timing, memory use, input and output, initialization, and finite word effects?
5. Has the product component been assessed for safety considerations?
6. Has the product component been assessed for open systems considerations?
7. Has the product component been assessed for security considerations?
8. Has the product component been assessed for innovation considerations?

Table 7.8 Test Procedure Checklist

Completeness
1. Have objectives been established for each test case?
2. Have all predecessor requirements been accounted for?
3. Has available tool assistance been applied in assessing traceability?
4. Were any test cases revealed not to have traceability to the predecessor requirements?
5. Have test prerequisite conditions been established?
6. Have test input conditions been established?
7. Have expected test results been established?
Correctness
1. Is the test case testable within the test category (Review, Lab, Field)?
2. Does the test case objective reflect requirements?
3. Are the test prerequisite conditions necessary and complete?
4. Are the input conditions correct and obtainable?
5. Do the test results reflect the requirements?
6. Do the test procedures satisfy the test case objectives?
7. Are the test procedure steps correct and in logical order?
Style
1. Are style conventions for test procedure structuring defined and followed?
2. Are procedures written in "Device/Action/Observation/Comment" format?
3. Are style conventions for alignment, upper/lower case, and highlighting defined and followed?
4. Has a version number been assigned?
5. Are templates used for repeating patterns?
Multiple Views
1. Have test cases been assessed for integration considerations, such as input and output, integration of multiple components, and test efficiency?
2. Have the test procedures been assessed for packaging considerations?
Metrics
1. Have the total pages inspected been recorded?
2. Have the total minutes of inspection preparation effort been recorded?
3. Have the total minutes of inspection conduct time been recorded?
4. For each defect, have defect category, severity, type, and origin been recorded?

7.4.3 Rules of Construction

Rules of construction are based on the software application architecture and the specific protocols, templates, and conventions used to carry it out. For example, these include interprocess communication protocols, tasking and concurrent operations, program unit construction, and data representation.

The moderator script for inquiring about rules of construction includes:

1. Are guidelines for program unit construction followed?
2. Is the interprocess communication protocol followed?
3. Are data representation conventions followed?
4. Is the system standard time defined and followed?
5. Are encapsulation, localization, and layering used to achieve object orientation?
6. Is logical independence achieved through event driven and process driven paradigms, late binding, and implicit binding?
7. Is scalability achieved through uniformity, parameterization, and portability?
8. Are fault tolerance, high availability, and security achieved?

7.4.4 Multiple Views

Multiple views are based on the various perspectives and view points required to be reflected in the software product. During execution many factors must operate harmoniously as intended including initialization, timing of processes, memory management, input and output, and finite word effects. In building the software product, packaging considerations must be coordinated including program unit construction, program generation process, and target machine operations. Product construction disciplines of systematic design and structured programming must be followed as well as interactions with the user, operating system, and physical hardware.

The moderator script for inquiring about multiple views includes:

1. Has the logical view of user interface and object orientation considerations been assessed?
2. Has the static view of packaging considerations been assessed including program unit construction, program generation process, and target machine operations?
3. Has the dynamic view of operational considerations been assessed including communications, concurrency, synchronization, and failure recovery?
4. Has the physical view of execution considerations been assessed including timing, memory use, input and output, initialization, and finite word effects?

7.4.5 Defined Roles of Participants

Software inspections are a reasoning activity performed by practitioners playing the defined roles of moderator, recorder, reviewer, reader, and producer. Some may name these roles facilitator, scribe, inspector, and author. Each role carries with it the specific behaviors, skills, and knowledge needed to achieve the expert practice of software inspections [6].

Individuals attending the inspection session may take on more than one role. For example, the producer may also be a reviewer. The moderator and recorder roles are demanding ones, and the individual assigned is usually dedicated to the single role. The reader role is not always utilized in software inspections. When the reader is used, one of the reviewers, not the producer, is assigned this role. In software walkthroughs, the producer serves as reader.

7.4.5.1 Manager

The manager is active in the planning, preparation, reporting, and follow-up activities. In planning, the manager identifies and schedules all software inspections in the project plan. The manager identifies personnel resource needs in terms of labor hours and allocates them to each inspection. The moderator is assigned by the manager, who ensures that only trained moderators are appointed.

The manager generally does not attend the inspection session. Practitioners are wary that managers attending an inspection session might use the results in the personnel performance appraisal of the producer. In addition, reviewers are reluctant to identify defects in the artifacts of their peers in the presence of managers. However, if the manager is an expert in the application and must be present to make a

technical contribution, the manager must first convincingly check management and organizational behaviors at the door and attend the inspection as a technical peer.

After the software inspection is conducted, the manager receives the moderator's report, meets with the producer to plan the follow-up, and administers the follow-up oversight.

7.4.5.2 Moderator

The moderator is the keystone of the software inspections process and is active in the preparation, entry criteria, conduct, exit criteria, and reporting activities. The moderator directs the activities of the software inspection. During the preparation activity, the moderator briefs the inspection team members on their roles in the structured review process, asks the producer to overview the software product to be inspected, distributes the inspection materials, and announces the time and place for the inspection session.

During the inspection session, the moderator directs the entry criteria, conduct, and exit criteria activities and facilitates the interaction among the inspection team members. The moderator intervenes as little as possible and as much as necessary to ensure that an effective and efficient software inspection session takes place.

A skillful moderator recognizes the role specific needs of inspection team members. For example, a producer with a "good catch" on his own product is called upon first. A talkative reviewer with little preparation effort is controlled. Where the moderator has issues to bring up, it is good form to insert these after the other team members have spoken. The moderator collaborates with the recorder in preparing the report for the manager on the findings of the inspection session.

7.4.5.3 Producer

The producer is active during the preparation, entry criteria, conduct, exit criteria, and follow-up activities. The producer is responsible for creating the materials to be inspected. The producer attends the inspection as reviewer and is expected to raise issues. From time to time the producer may offer a technical explanation of the product as necessary.

The producer expects criticism of the product and need not offer any defense as issues are raised. It is understood that the producer may be in a protective state of mind with respect to the product being inspected. What is asked of the producer is that the protective state not be exhibited as defensive behavior. Where an issue is surfaced that is not understood by the producer, a dialogue may be needed to obtain clarification.

At the conclusion of the conduct activity, the producer is afforded the opportunity to comment on the inspection session and to acknowledge the value of the issues raised. The producer meets with the manager to plan the rework and performs the follow-up actions resulting from the inspection.

7.4.5.4 Recorder

The recorder is active in the preparation, entry criteria, conduct, exit criteria, and reporting activities. The recorder completes the Inspection Record, the Inspection

Reporting Form, and the Report Summary Form. The practice of the recorder is "to leave no bits on the floor." In other words, the recorder is expected to record every issue without exception.

During the entry criteria, the recorder notes the preparation effort of each inspection team member, the start and stop time of the meeting, the project and product name and size, and the life-cycle activity for which the inspection is an exit criteria. As issues are raised, the recorder describes each issue and notes defect category, defect severity, defect type, and defect origin. The recorder uses the key word "investigate" in recording issues that may be defects but require additional research following the meeting. At the conclusion, the recorder tabulates the issues by defect type, severity, and category.

The role of the recorder is to be transparent to the inspection session and to record all issues completely and accurately. This requires a high degree of concentration, judgment, and technical knowledge.

7.4.5.5 Reviewer

Reviewers are active in the preparation, entry criteria, conduct, and exit criteria activities. A reviewer is expected to spend sufficient time preparing and to raise issues and concerns about the software product. Reviewers are asked to refrain from proposing solutions and to direct their comments at the product not the producer. Reviewers accept the discipline imposed by the round robin, checklist structure of the inspection session. In return for accepting these responsibilities and disciplines, each reviewer is assured of an uninterrupted opportunity to raise issues.

7.4.5.6 Reader

The reader is active in the preparation, entry criteria, conduct, and exit criteria activities. Where necessary, the moderator may ask the reader to read parts of the product aloud so as to focus attention on a particular trouble spot. The reader does this by paraphrasing not by reading line by line. Using the reader for this task helps promote the egoless behavior of the producer. The reader is responsible for bringing to the inspection session and being prepared to navigate any background materials, such as, baseline documentation and style guide.

7.4.6 Forms and Reports

All data collected and reported during the software inspections process is recorded by the recorder. This includes data about the product being inspected and about the inspection process itself. The requirements for data collection are defined and focus on three recording instruments: Inspection Record, Inspections Reporting Form, and Report Summary Form.

7.4.6.1 Inspection Record

The Inspection Record is initiated during the entry criteria activity when the recorder gathers the preparation effort from each inspection team member. The

name of the project and the product component are recorded along with the size of the product to be inspected. The life-cycle activity for which this inspection serves as the exit criteria is recorded. The start time for the inspection session is entered at the beginning of the meeting, and the stop time is recorded at the close. Also at the close of the session, the disposition is recorded in terms of acceptance, re-inspection, or conditional. See Figure 7.3.

7.4.6.2 Inspection Reporting Form

During the conduct activity as issues are raised, the recorder documents a description of each issue and assigns attributes that characterize the issue. Each issue is assigned a sequence number, and the page and line number are pinpointed. Similar issues that occur a few times are recorded as separate issues. An issue type that occurs an unaccountably large number of times is recorded once.

A defect category is assigned as missing, wrong, or extra. A defect severity is assigned as major or minor. The defect origin is noted as the life-cycle activity during which this defect was inserted. The defect type is entered [7]. See Figure 7.4.

A major defect affects execution; a minor defect does not. In practice, some prefer to extend defect severity to include the extremes of critical and trivial. Other severity gradations used to classify defects, faults, and failures detected in testing and field operations are not used in software inspections. These test and operational execution-based severities often revolve around the criticality of the defect and its impact on sustaining testing or operations. Another dimension of defect severity is related to the effort needed to correct the defect.

The appropriate defect type is assigned as follows:

Inspection Record

Project Name: Date:
Product Component: Size:_____ lines _____pages
Start Time: _____ Stop Time: _____ Elapsed Time: _____

Role	Name	Preparation Minutes	Additional Comments
Moderator			
Recorder			
Producer			
Reviewer			
Reviewer			
Reviewer			
Reader			
Total Prep Effort			

Checklists Used		Disposition	Life-Cycle Activity
Completeness Correctness Style Rules of Construction Multiple Views Technology Metrics		Accept ____ Conditional ____ Reinspect ____	Planning Requirements Specification Design Code Test

Figure 7.3 Inspection Record.

Inspection Reporting Form

Issue Number	Page/ Line	Checklist	Defect Category	Defect Severity	Defect Type	Defect Origin	Defect Description

Defect Category: Missing, Wrong, Extra
Defect Severity: Major, Minor

Defect Type: Interface, Data, Logic, UO, Performance, Functionality, Human Factors, Standards, Documentation, Syntax, Maintainability, Other

Figure 7.4 Inspection Reporting Form.

1. Interface: error in parameter list;
2. Data: error in data definition, initial value setting, or use of disciplined data structures;
3. Logic: error revealed through informal correctness questions spanning prime constructs of structured programming;
4. I/O: error in formatting, commanding, or controlling I/O operations;
5. Performance: error in managing or meeting constraints in computer resource allocations and capacities for CPU, memory, or I/O;
6. Functionality: error in stating intended function or in satisfying intended function through refinement or elaboration;
7. Human Factors: error in externally visible user or enterprise interface or interaction;
8. Standards: error in compliance with product standards for construction or integration including programming style guidelines, open systems interfaces, or guidelines for the application domain architecture;
9. Documentation: error in guidance documentation;
10. Syntax: error in language defined syntax;
11. Maintainability: error in uniformity and consistency;
12. Other: any other error.

7.4.6.3 Report Summary Form

During the exit criteria, the recorder completes the meeting stop time, verifies the completeness of all recorded results, and completes the Report Summary Form. This

form is a frequency count of issues presented as a matrix of defect types by defect severity and defect category. This form serves several purposes. Since it cannot be constructed unless the recorder has completed the Inspection Reporting Form, it serves as an on–the-spot check of the recorded results. Once completed, weaknesses are highlighted and some opportunities for defect prevention suggest themselves. When the results of numerous inspection sessions are overlaid on the Report Summary Form, these frequency counts divided by the total defects serve as the probability of occurrence for each defect type, defect severity, and defect category. See Figure 7.5.

7.5 Preparation for Expert Use

A collection of coordinated knowledge, skills, and behaviors facilitates the best possible practice of peer reviews. As Deming reminded us, there is no substitute for superior knowledge. In conducting peer reviews, superior knowledge is sought in the application domain, the computing platform both hardware and operating system, and programming language. In addition, participants must be knowledgeable in the peer review process and the standard of excellence expected in the product artifact.

For best results, participants filling certain defined roles must possess particular skills. The moderator needs facilitation, conflict identification, and conflict resolution skills. The recorder needs listening, synthesizing, and recording skills. The reviewer needs code reading skills.

Participants in peer reviews are expected to adopt certain behaviors known to contribute to effective and harmonious review sessions. First, the rules of civility apply. For example, one person speaks at a time, and personal attacks are not permitted. Second, since people make mistakes sometimes, it is necessary to

Report Summary Form						
Defect Types	Major Defects			Minor Defects		
	Missing	Wrong	Extra	Missing	Wrong	Extra
Interface						
Data						
Logic						
I/O						
Performance						
Functionality						
Human Factors						
Standards						
Documentation						
Syntax						
Maintainability						
Other						

Figure 7.5 Report Summary Form.

decriminalize defects so that they do not remain hidden. Recognizing that, assigning blame for defects is discouraged. Third, participants are encouraged to direct their comments towards the product not the person who authored the artifact being reviewed. Finally, everyone is encouraged to give way to the individual who possesses superior knowledge.

7.6 Measurements

While many organizations have adopted software inspections, few have published their results. Those that have published results typically have done so following early successes in the new practice adoption cycle. Organizations with published results have included the Jet Propulsion Laboratory [20], Litton Data Systems [21], Bull HN Information Systems, Inc. [22], AT&T Corp. [23], and Lockheed Martin Corporation [24]. While they all used software inspections and have documented measured results, the particular adaptations are not well aligned, and the results do not lend themselves to systematic comparison.

7.6.1 National Software Quality Experiment

In 1992 the DOD Software Technology Strategy set the objective to reduce software problem rates by a factor of 10 by 2000. The National Software Quality Experiment is being conducted to benchmark the state of software product quality. The experiment has measured progress towards the national objective [16, 25, 26] and continues with the measurements. Industry problem rates ranged from 1 to 10 defects per thousand lines of source code. Meeting the objective shifts the range to 0.1 to 1 defect per thousand lines of source code.

The centerpiece of the experiment is the Software Inspection Lab where data collection procedures, product checklists, and participant behaviors are packaged for operational project use. The uniform application of the experiment and the collection of consistent measurements are guaranteed through rigorous training of each participant.

Approximately 3,000 participants from nearly 60 organizations have populated the experiment database with nearly 15,000 defects of all types along with pertinent information needed to pinpoint their root causes. These results are highlighted below in the discussion of the common problems, Inspection Lab operations, defect type ranking, and return on investment.

7.6.2 Common Problems Revealed

Analysis of the issues raised in the experiment has revealed common problems that reoccur from session to session. Typical organizations that desire to reduce their software problem rates should focus on preventing the following types of defects:

1. Software product source code components are not traced to requirements. As a result, the software product is not under intellectual control, verification procedures are imprecise, and changes cannot be managed.

2. Software engineering practices for systematic design and structured programming are applied without sufficient rigor and discipline. As a result, high defect rates are experienced in logic, data, interfaces, and functionality.
3. Software product designs and source code are recorded in an ad hoc style. As a result, the understandability, adaptability, and maintainability of the software product are directly impacted.
4. The rules of construction for the application domain are not clearly stated, understood, and applied. As a result, common patterns and templates are not exploited in preparation for later reuse.
5. The code and upload development paradigm is becoming predominant in emerging e-commerce applications.

As a result, the enterprise code base services only the short term planning horizon where code rules and heroes flourish, but it mortgages the future where traceable baseline requirements, specification, and design artifacts are necessary foundations.

7.6.3 Inspection Lab Operations

The Inspection Lab is the consistent operation of software inspection sessions as part of the National Software Quality Experiment. These sessions apply the elements of software inspections including the entry, conduct, and exit processes; defined roles of participants; product checklists; and forms and reports. Through 2002, 3,040 participants conducted inspection sessions. A total of 1,020,229 source lines of code have received strict and close examination in the Software Inspection Lab. There have been 181,471 minutes of preparation effort and 71,283 minutes of conduct time expended to detect 14,903 defects. See Figure 7.6. For each metric, control panels are derived by ordering all values for the metric and selecting the data points at the 20th percentile, 50th percentile, and 80th percentile. With these values the Software Inspections Control Panel in Figure 7.7 is produced.

Of these 14,903 defects, 2,512 were classified as major, and 12,391 as minor. A major defect effects execution; a minor defect does not. It required 12.18 minutes of preparation effort on the average to detect a defect. To detect a major defect required 72.24 minutes of preparation effort on the average. On the average, 0.858 thousand source lines of code were examined each inspection conduct hour. There were 2.46 major defects detected in each thousand lines, and 12.15 minor defects. There were 4.90 defects detected in inspecting 335.60 lines per session. The preparation effort was 0.64 of conduct effort. The Software Inspection Labs produced a return on investment of 4.50.

7.6.4 Defect Type Ranking

The foremost defect types that accounted for more than 90% of all defects detected include the following (see Figure 7.8):

- *Documentation:* 40.51% error in guidance documentation;
- *Standards:* 23.20% error in compliance with product standards;

Sessions	Prep Effort	Conduct Time	Major Defects	Minor Defects	Size in Lines
3,040	181,471	71,283	2,512	12,391	1,020,229
Metrics:					
1.	12.18	Minutes of preparation effect per defect			
2.	72.24	Minutes of preparation effort per major defect			
3.	2.46	Major defects per thousand lines			
4.	12.15	Minor defects per thousand lines			
5.	858.74	Lines per conduct hour			
6.	4.90	Defects per session			
7.	0.64	Preparation/conduct effort			
8.	335.60	Lines per session			
9.	4.50	Return on investment			

Figure 7.6 Inspection Lab operations.

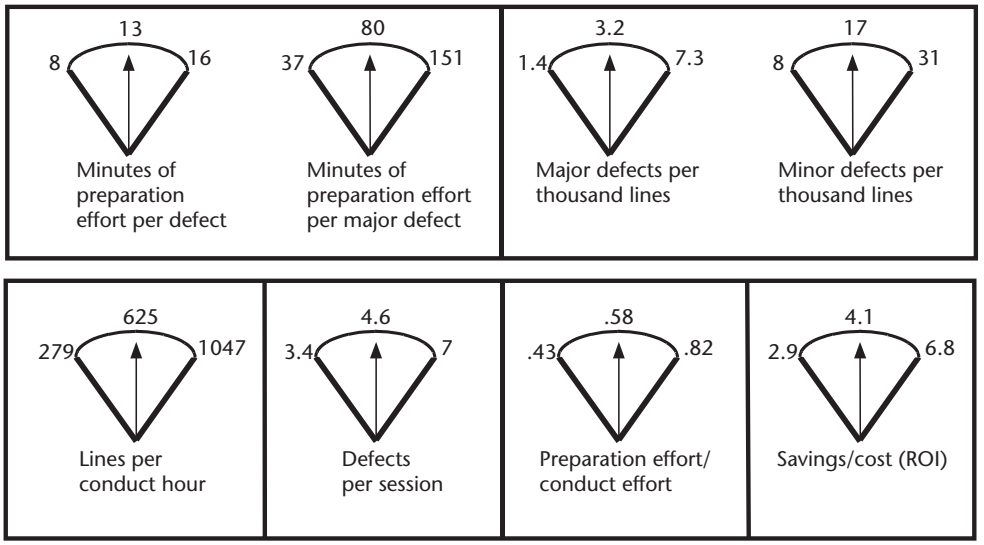

Figure 7.7 Software Inspections Control Panel.

- *Logic:* 7.22% error revealed through informal correctness questions function;
- *Functionality:* 6.57% error in stating or meeting intended;
- *Syntax:* 4.79% error in language defined syntax compliance;
- *Data:* 4.62% error in data definition, initial value setting, or use;
- *Maintainability:* 4.09% error in good practice impacting the supportability and evolution of the software product.

7.6.5 Return on Investment

Managers are interested in knowing the return on investment to be derived from software process improvement actions. The software inspections process gathers the data needed to determine this [4].

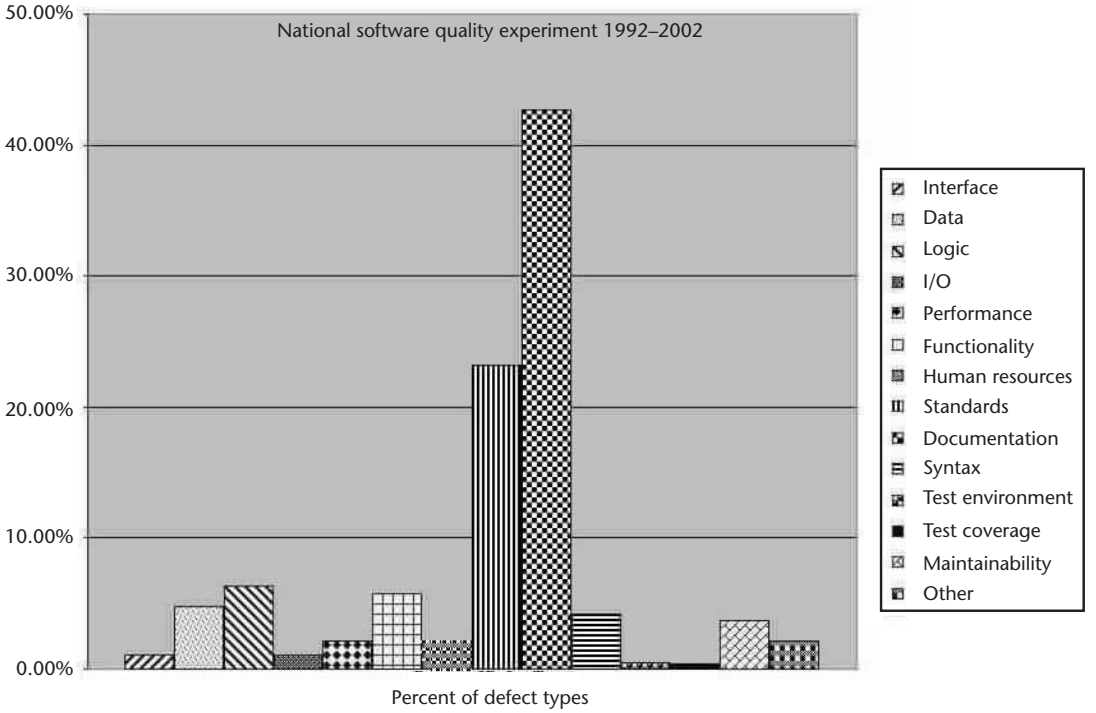

Figure 7.8 Defect type distribution.

The return on investment for software inspections is defined as net savings divided by detection cost, where net savings is cost avoidance less cost to repair and detection cost is the cost of preparation effort and the cost of conduct effort. The defined measurements collected in the Software Inspections Lab may be combined in complex ways to form this derived metric.

The model for return on investment bases the savings on the cost avoidance associated with detecting and correcting defects earlier rather than later in the product evolution cycle. A major defect that leaks from development to test may cost as much as 10 times to detect and correct. Some defects, undetected in test, continue to leak from test to customer use and may cost an additional 10 times to detect and correct. A minor defect may cost two to three times to correct later.

Figure 7.9 is a graph showing the return on investment measurements for each organization participating in the National Software Quality Experiment. This graph suggests that the return on investment for software inspections ranges from 4:1 to 8:1. For every dollar spent on software inspections, the organization can expect to avoid $4 to $8 on higher rework cost. Table 7.9 provides the return on investment expressions needed to perform the calculation [27].

7.7 Transition from Cost to Quality

In using software inspections, the goals vary with the maturity of the software product engineering method used, transitioning from cost to quality. Three levels of achievement of software product engineering are identified:

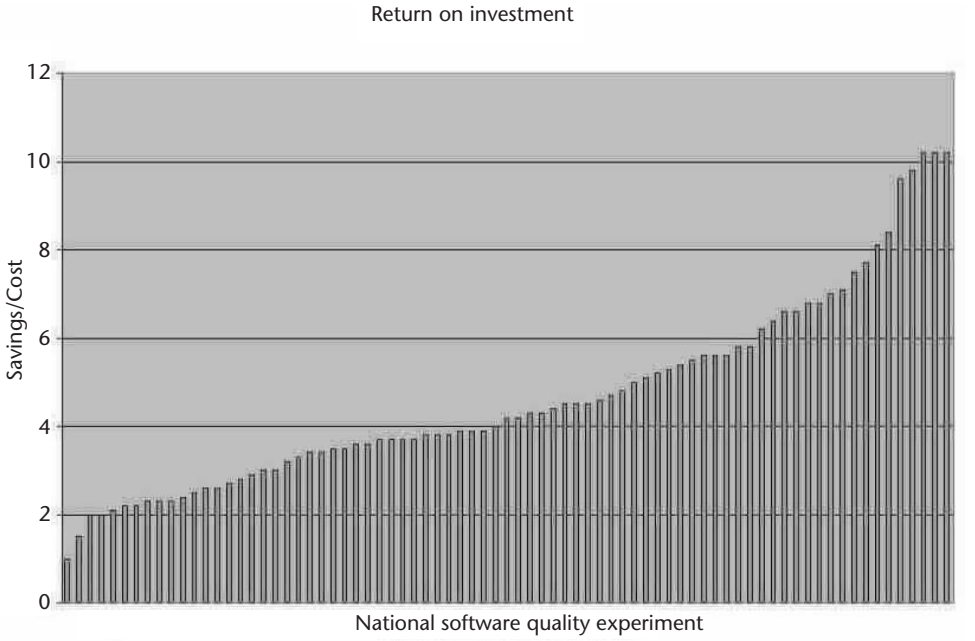

Figure 7.9 Return on investment measurements.

Table 7.9 Return on Investment Expressions

Return on Investment
Return on Investment is Net Savings divided by Detection Cost, or ROI = Net Savings/Detection Cost
Net Savings
Net Savings is Cost Avoidance minus Cost to Repair Now, or Net Savings = Cost Avoidance – Cost to Repair Now
Cost Avoidance
Cost Avoidance = Major Defects*{(M1*DD)+(M1*DD)*(M2*TL)*C1}+Minor Defects*M3
Net Savings
Net Savings = Major Defects*{(M1*DD)+(M1*DD)*(M2*TL)*C1-C1}+Minor Defects*(M3-C2) where: • M1: (2–10) Additional Cost to Repair Multiplier for Development to Test Major Deflect Leakage • M2: (2–10) Additional Cost to Repair Multiplier for Test to Customer Use Major Defect Leakage • M3: (2–4) Additional Cost to Repair for Minor Defect Leakage • DL: (0.5–0.95) Defect Detection Rate for Development to Test • TL: (0.05–0.5) Test Leakage Rate for Test to Customer Use • C1: Average Cost to Repair Major Defect • C2: Average Cost to Repair Minor Defect

1. Ad hoc programming is characterized by a code and upload life cycle and a hacker coding style. This is common in low software process maturity organizations especially those facing time-to-market demands.

2. Structured software engineering employs structured programming, modular design, and defined programming style and pays close attention to establishing and maintaining traceability among requirements, specification, architecture, design, code, and test artifacts.
3. Disciplined software engineering is more formal and might be patterned after Clean Room software engineering, Personal and Team Software Process, and Extreme Programming techniques [14, 19].

By necessity, the focus of ad hoc programming practitioners is on reducing cost by detecting as many defects as possible. With 40 to 60 defects inserted, a defect detection rate of 0.50–0.65, and an additional cost multiplier of 8 to 10, the result is a net savings of 234.80 to 285 labor hours and a defect leakage expectation of 8.75 to 12.50 per thousand lines of code, numbers that promote a focus on cost. For this group, finding defects is like finding free money, and there are always more defects to find; however, managers struggle to meet cost and schedule commitments.

Structured software engineering focus is split between reducing cost and improving quality. With 20 to 30 defects inserted, a defect detection rate of 0.70–0.80, and an additional cost multiplier of 5 to 7, the result is a net savings of 65.00 to 85.23 labor hours and a defect leakage expectation of 2.5 to 3.75 per thousand lines of code, numbers that promote an attraction to both goals. For this group, there is constant dithering between cost and schedule.

Without question, the focus of disciplined software engineering practitioners is on eliminating every possible defect even if defect detection costs exceed net savings and the return on investment falls below the break even point. With 10 to 15 defects inserted, a defect detection rate of 0.85 to 0.95, and an additional cost multiplier of 2 to 4, the result is a net savings of 12.49 to 18.55 labor hours and a defect leakage expectation of 0.3125 to 0.9375 per thousand lines of code, numbers that promote a focus on quality. For this group, every practitioner is riveted on achieving perfection.

7.8 Software Inspections Roll Out

Software inspections have application throughout the software organization. Consequently a systematic approach is needed to introduce it to all participants. The steps in the defined program for rolling out software inspections within the organization include [28, 29]:

1. Assess software inspections practice.
2. Obtain management commitment.
3. Conduct software inspections training for practitioners, managers, and executives.
4. Prepare and disseminate software inspections policy and procedure documents.
5. Establish a coordination infrastructure, assign personnel, and formulate a program agenda.
6. Establish an inspection-based measurement program and database.

7. Set management objectives for planning, training, conducting, and using software inspections and measurements.
8. Continue to evolve the organization's process and product checklists.

The initial step in the roll out is to conduct an assessment of the software inspections practice. This will identify the strengths and weaknesses in planning, training, conduct, and reporting and use of results. Armed with this assessment, the change agent seeks management commitment and sponsorship for the improvements needed to offset the weaknesses. The following key practice indicators are assessed at regular intervals for each project in terms of approach, deployment, and results:

1. The project follows a documented organization policy for performing software inspections.
2. Adequate resources and funding are provided for performing software inspections on each software work product to be reviewed.
3. Moderators and reviewers receive required training in how to lead software inspections, as well as in objectives, principles, and methods of software inspections.
4. Software inspections are planned and documented.
5. Software inspections are performed according to a documented procedure.
6. Data on the conduct of the software inspections is recorded.
7. Measurements are made and used to determine the status of the software inspections activities.
8. The software quality assurance group reviews and/or audits the activities and work products for inspections and reports the results.

A clear management commitment is needed to approve and fund the training program including the cost of the instructor, training site, labor and burden for student attendance, and scheduling and administration of student enrollment and attendance.

With training underway and inspections being initiated by projects, peer reviews policy and procedure documents provide the guidelines for the well-defined process being deployed. The policy applies to all software development projects and states that each project involved in software development shall prepare, plan, conduct, and utilize software inspections. The purpose of the procedure is to provide the step-by-step instructions needed to carry out consistent examinations of work products on the project.

To fully stimulate the software inspections roll out program among project personnel, an infrastructure of coordinators drawn from active projects is convened. These project coordinators meet periodically to share experiences, compare results, and discuss common problems.

As inspection results and measurements are accumulated, a measurement database and the operations for analyzing, reporting, and acting upon these measurements are established. The measurements are recorded in the Software Inspection Lab. These measurements include preparation effort, conduct time, and the artifact size in lines of code or pages inspected and for each defect detected the defect severity, defect category, defect type, and defect origin. Using the measurements, metrics

are derived to continually assess the efficiency and effectiveness of the process and its operation. These metrics include:

1. Minutes of preparation effort per defect;
2. Minutes of preparation effort per major defect;
3. Major defects per thousand lines;
4. Minor defects per thousand lines;
5. Lines per conduct hour;
6. Defects per session;
7. Preparation/conduct effort;
8. Lines per session;
9. Return on Investment.

7.9 Future Directions

In reasoning about future trends of peer reviews, the topics considered include increasing rate of software problems, improving the practice of defect prevention and prediction, extending the practice of peer reviews to systems engineering, understanding the process of experimentation in software development, exploiting technology in automating the peer reviews, and adapting to changes in business environment.

Software problem rates are not decreasing. The results of the National Software Quality Experiment 1999 show no systematic improvement towards fulfilling the national goal of a 10 times reduction in software problems set in 1992. The defect rates continue to range from 1 to 10 defects per thousand lines of source code.

The factors that may be contributing to defect rates include:

1. The emphasis on quicker, better, and cheaper;
2. The trend towards code and upload practice as the life-cycle model;
3. The preoccupation on improving software process maturity and mastering the management track practices of the Software Engineering Institute's CMMI® for Development, Maturity Level 2, an obstacle to many;
4. The downsizing of middle management and senior technical staff known to hold the line on product quality.

While software inspections have been in use for more than 25 years, defect prevention remains an immature practice. Causal analysis and resolution are a CMMI® for Development Maturity level 5 process area whose purpose is to identify causes of defects and take action to prevent them, and some organizations have achieved level 5. As more organizations seek to adopt the practice of defect prevention, its benefits and methods may become better understood, stimulating others to adopt the practice.

Similarly, defect prediction remains an underdeveloped practice. If software defects, faults, and failures can be predicted, perhaps they can be detected, controlled, and prevented. Model-based techniques calibrated with defect detection early in the life cycle to predict defect rates in later life-cycle activities have been

demonstrated [30]. More modest efforts utilizing software inspections data to estimate the number of defects remaining to be found in testing are being applied on the project [31]. However, there is insufficient defect, fault, and failure data available from the nation's factory floor [26]. In addition there is insufficient process, method, and tooling to combine defect data obtained through software inspections practice, software fault data obtained through software product test and use, and software failure data obtained through software system operation into predictions of trustworthy software system operation [32].

While the benefits and usage of software inspections on code artifacts is well known, there is increasing interest in extending software inspections to all phases of the life cycle including requirements, specifications, design, code, and test artifacts. The CMMI® model with the inclusion of peer reviews in the product verification process area extends peer reviews to both systems engineering and software artifacts.

To achieve the best possible practice of software inspections, both managers and technical practitioners are encouraged to decriminalize defects. People make mistakes sometimes, yet software must be bit perfect. When managers and technical participants view with alarm the defects detected in software inspections, it produces a negative impact. On the other hand, when managers genuinely decriminalize defects and use their detection as a means to prevent their recurrence, it produces a positive result. During a software inspection session, the litmus test for decriminalization lies in the reaction of participants when a major defect is detected. Does the group say "good catch" or "bummer"?

With the growing recognition that fielding software involves a process of experimentation and with the increasing pressures of competition and demand for innovation, software walkthroughs may experience increasing usage. Software walkthroughs encourage and support the learning essential to experimentation. In favoring the group interaction needed to achieve consensus, software walkthroughs may contribute to increased innovation in software products.

There is interest in automating software inspections. The value of programming languages with strong typing, robust compilers, static analyzers and traceability tools, and complexity metrics [18] is recognized. However, software inspections practice is a reasoning activity and will remain essentially a human activity. The use of information technology innovations to support the logistics of preparation, scheduling, conduct, and results repository operations are sources for improved industry practice.

Software inspections are being conducted effectively using groupware tools. However, where global software development teams conducting geographically dispersed inspection sessions are using "follow the sun" software development tactics, software inspection participants may be separated by both geography and time zones, complicating the logistics of their application [33].

Software inspections usage is increasing in e-commerce applications where code and upload is the typical life-cycle practice. In an environment of rapid change and frequent releases, there is an absence of robust testing and sometimes even regression testing.

7.10 Conclusion

Software inspections deliver value to the organization through the close and strict examinations on life-cycle product artifacts that detect defects early and promote the deepest possible understanding of the artifact. The organization that sets the standard of excellence for its software engineered products in terms of completion, correctness, style, rules of construction, and multiple views and disciplines its practitioners to meet the standard set is able to reap an attractive return on investment while earning higher customer satisfaction. The measurements taken during software inspections promote an understanding of common problems and reveal opportunities for product and process improvement.

References

[1] O'Neill, D., "Software Inspections," *Software Technology Guide*, Software Engineering Institute, January 10, 1997.

[2] Fagan, M., "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal*, Vol. 15, No. 3, 1976, pp. 182–211.

[3] Fagan, M., "Advances in Software Inspections," *IEEE Transactions on Software Engineering*, Vol. 12, No. 7, 1987.

[4] McGibbon, T., "A Business Case for Software Process Improvement," *Rome Laboratory DACS Report*, September 30, 1996.

[5] O'Neill, D., "Software Inspections Course and Lab," Software Engineering Institute, 1989.

[6] Freedman, D. P., and G. M. Weinberg, *Handbook of Walkthroughs, Inspections, and Technical Reviews*, New York: Dorset House, 1990, pp. 89–161.

[7] Ebenau, R. G., and S. H. Strauss, *Software Inspection Process*, New York: McGraw-Hill, 1994, pp. 236–240.

[8] Gilb, T., and D. Graham, *Software Inspection*, Reading, MA: Addison-Wesley, 1993.

[9] Humphrey, W. S., *Managing the Software Process*, Reading, MA: Addison-Wesley, 1989, pp. 171–190.

[10] Special permission to reproduce O'Neill, D., and A. L. Ingram, "Software Inspections Tutorial," as contained in the *Software Engineering Institute Technical Review*, 1988, pp. 92–120, by Carnegie Mellon University, is granted by the Software Engineering Institute.

[11] Paulk, M. C., et al., *The Capability Maturity Model: Guidelines for Improving the Software Process*, Reading, MA: Addison-Wesley, 1995, pp. 270–276.

[12] Johnson, D. L., and J. G. Broadman, "Realities and Rewards of Software Process Improvement," *IEEE Software*, Vol. 13, No. 6, © IEEE, November 1996.

[13] Humphrey, W. S., *Managing the Software Process*, Reading, MA: Addison-Wesley, 1989, pp. 463–486.

[14] Humphrey, W. S., *A Discipline for Software Engineering*, Addison-Wesley, 1995, page 233.

[15] Gilb, T., and D. Graham, *Software Inspection*, Reading, MA: Addison-Wesley, 1993, pp. 40–136.

[16] O'Neill, D., "National Software Quality Experiment: A Lesson in Measurement 1992–1997," *CrossTalk*, Vol. 11, No. 12, Web Addition, December 1998.

[17] Linger, R. C., H. D. Mills, and B. I. Witt, *Structured Programming: Theory and Practice*, Reading, MA: Addison-Wesley, 1979, pp. 147–212.

[18] McCabe, T. J. and A. H. Watson, "Software Complexity," *CrossTalk*, Vol. 7, No. 12, December 1994, pp. 5–9.

[19] Prowell, S. J., et al., *Cleanroom Software Engineering: Technology and Process*, Reading, MA: Addison-Wesley, 1999, pp. 17, 33–90.

[20] Kelly, J., and J. Sherif, "An Analysis of Defect Densities Found During Software Inspections," *Proceedings of the Fifteenth Annual Software Engineering Workshop*, Goddard Space Flight Center, Greenbelt, MD, December 1990.

[21] Madachy, R., L. Little, and S. Fan, "Analysis of a Successful Inspection Program," *Proceedings of the Eighteenth Annual Software Engineering Workshop*, Goddard Space Flight Center, Greenbelt, MD, December 1993, pp. 176–188.

[22] Weller, E. F., "Lessons from Three Years of Inspection Data," *IEEE Software*, September 1993, pp. 38–45.

[23] Ebenau, R. G., "Predictive Quality Control with Software Inspections," *CrossTalk*, Vol.7, No. 6, June 1994, pp. 9–16.

[24] Bourgeois, K. V., "Process Insights from a Large-Scale Software Inspections Data Analysis," *CrossTalk*, Vol. 9, No. 10, October 1996, pp. 17–23.

[25] O'Neill, D., "National Software Quality Experiment: A Lesson in Measurement 1992–1996," *Quality Week Conference*, San Francisco, CA, May 1997, and *Quality Week Europe Conference*, Brussels, November 1997, pp. 1–25.

[26] O'Neill, D., "National Software Quality Experiment: A Lesson in Measurement 1992–1997," *First Annual International Software Assurance Certification Conference*, Chantilly, Virginia, March 1, 1999, pp. 1–14.

[27] O'Neill, D., "Determining Return on Investment Using Software Inspections," *CrossTalk*, March 2003, pp. 16–21, http://members.aol.com/ONeillDon/roi-essay.html.

[28] O'Neill, D., "Issues in Software Inspection," *IEEE Software*, Vol. 14, No 1, January 1997, pp. 18–19.

[29] O'Neill, D., "Setting Up a Software Inspections Program," *CrossTalk*, Vol. 10, No. 2, February 1997, pp. 11–13.

[30] Gaffney, J. E., "Software Defect Estimation, Prediction, and the CMM®," *Metrics '97 Conference*, 1997.

[31] Harding, J. T., "Using Inspection Data to Forecast Test Defects," *CrossTalk*, Vol. 11, No. 5, May 1998, pp. 19–24.

[32] Wallace, D. R., L. M. Ippolito, and H. Hecht, "Error, Fault, and Failure Data Collection and Analysis," *Quality Week*, San Francisco, CA, May 1997.

[33] Carmel, E., *Global Software Teams: Collaborating Across Borders and Time Zones*, Englewood Cliffs, NJ: Prentice-Hall, 1999, pp. 27–33.