SE-3002
SOFTWARE QUALITY ENGINEERING

RUBAB JAFFAR
RUBAB.JAFFAR@NU.EDU.PK

Part III-Other Quality Assurance Techniques

Defect Prevention,
Software Inspection, review & walk through, Formal Verification
Fault tolerance, Fault containment

TODAY'S OUTLINE

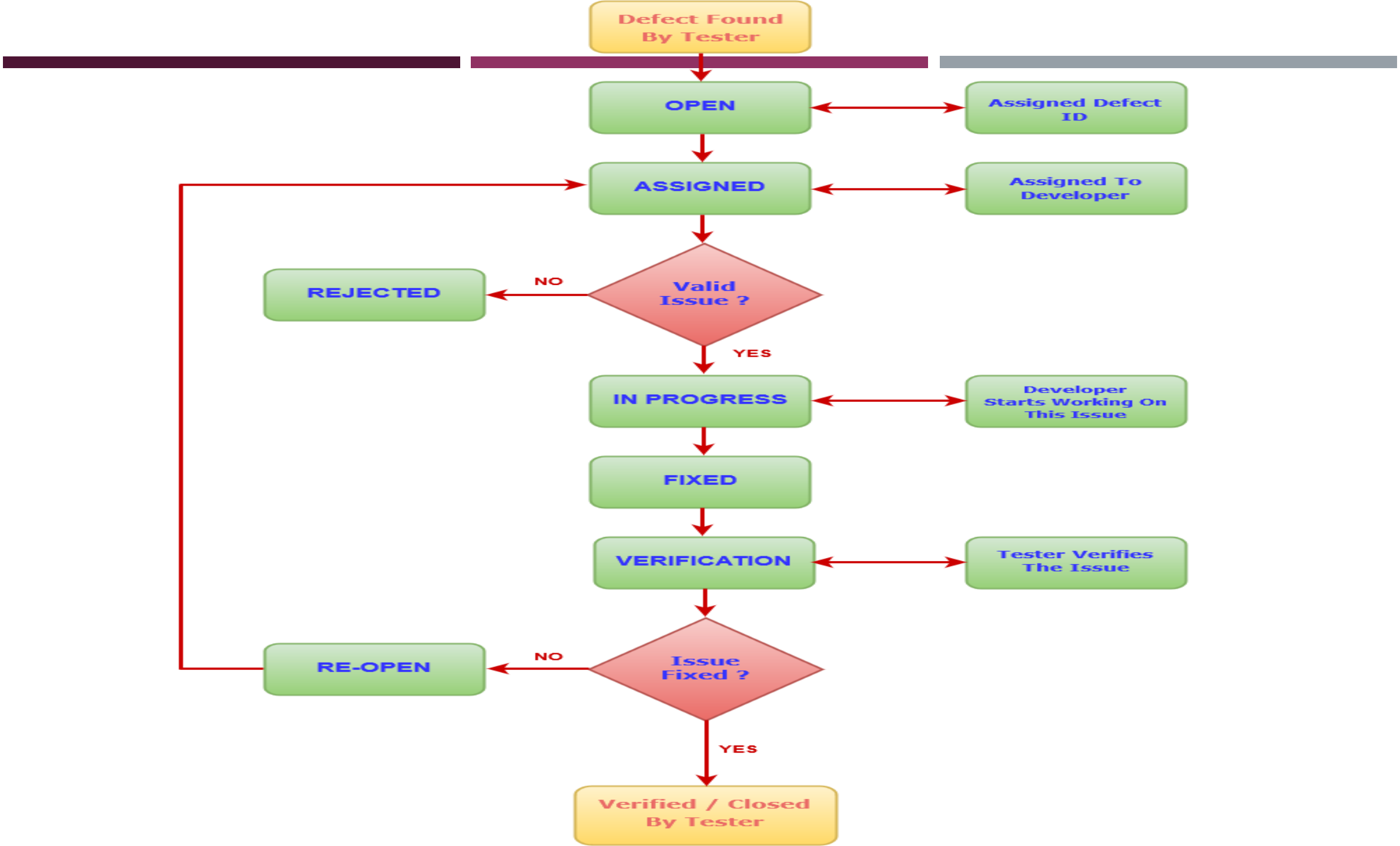
- Defect prevention(Chapter 13)
- Software Inspection (Chapter 14)
  - Fagan inspections, Gilb inspection, Software reviews, Inspection checks and metrics
- Desk check
- Review
- Walkthrough
- Formal Verification (Chapter 15)
- Fault tolerance (Chapter 16)
- Fault containment
- Comparison of different QA techniques w.r.t (Chapter 17)
  - Defect perspective
  - Problem type
  - Interpretation
  - Level of difficulty

DEFECT

- A *defect or a bug is an error in coding or logic that causes a program to malfunction or to produce incorrect or unexpected results.*

DEFECT LIFE CYCLE

- In software development process, the bug has a lifecycle. The bug should go through the life cycle to be closed.
- A specific life cycle ensures that the process is standardized.
- The bug attains different states in its lifecycle.


Defect Found By Tester
OPEN
Assigned Defect ID
ASSIGNED
Assigned To Developer
Valid Issue ?
NO
REJECTED
YES
IN PROGRESS
Developer Starts Working On This Issue
FIXED
VERIFICATION
Tester Verifies The Issue
Issue Fixed ?
NO
RE-OPEN
YES
Verified / Closed By Tester

SEVERITY OF DEFECT

- Indicate the **impact** each defect has on **testing efforts** or users and administrators of the application under test. This information is used by developers and management as the **basis** for assigning **priority** of work on defects.
- A sample guideline for assignment of priority levels during the product test phase includes:
  - **Critical**
  - **High**
  - **Average**
  - **Low**

DEFECT MANAGEMENT STRATEGIES IN SOFTWARE DEVELOPMENT

- Defect prevention
- Defect detection
- Defect containment

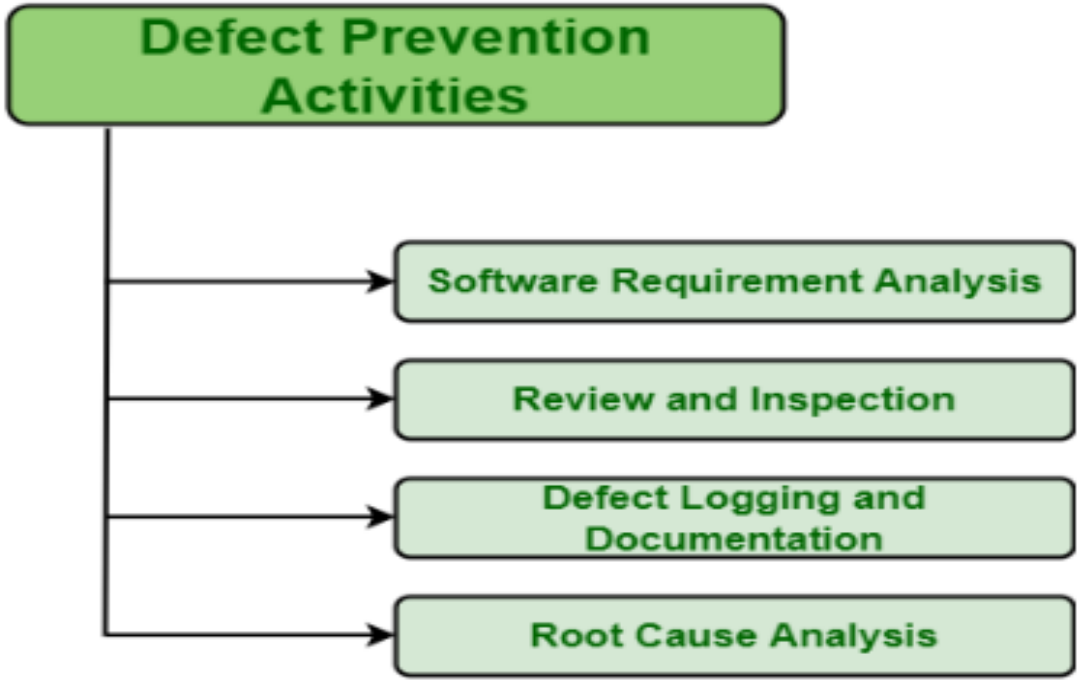
DEFECT PREVENTION

- Defect prevention techniques can be a very effective and efficient way to deal with quality problems by preventing the injections of faults into the software systems.
- The primary advantage of these techniques is in the effective savings resulted from not having to deal with numerous software faults that would otherwise be injected without applying these defect prevention techniques.
- There are two general strategies for defect prevention:
- **Error blocking:** Identifying common errors, which are defined to be missing or incorrect human actions, and blocking them to prevent fault injections. Various techniques, such as following well-defined processes, standards, and methodologies, or using appropriate tools, can help block identified common errors.
- **Error source removal:** Identifying common error sources and removing them, thus preventing fault injections. Various activities focused on people and their product and process knowledge can be carried out to removal these identified error sources, such as through education and training, process maturity and improvement initiatives, and other techniques specifically based on causal analysis of injected or potential faults.

DEFECT PREVENTION

- However, for new products to be developed in a new environment for an emerging market, it would be hard to collect existing defect data and analyze them to formulate an overall defect prevention strategy.
- Anticipated defects, based on past experience or general knowledge, can be used for this purpose. However, the effectiveness of the selected defect prevention techniques would be somewhat questionable, and should be closely monitored and adjusted whenever necessary.
- In either of the above situations with mature products or new ones, there are still some defects whose related errors or error sources we couldn't identify practically.
- This observation is particularly true for large software systems, where it would be infeasible to analyze all the actual and potential defects to find their corresponding causes in errors and in error sources.
- In addition, we cannot expect the selected defect prevention techniques to be 100% effective in blocking all the identified errors or removing all the identified error sources.
- These facts indicate that there would still be some defects that evade the defect prevention activities and get injected into large software systems.
- These defects then need to be dealt with by other software QA alternatives we cover in this book.

SOME EXAMPLE TECYHNIQUES:



SOFTWARE INSPECTION

- Most commonly performed software quality assurance (QA) activity besides testing.
- Inspection directly detects and corrects software problems without resorting to execution, therefore it can be applied to many types of software artifacts.
- Depending on various factors, such as the techniques used, software artifacts inspected, the formality of inspection, number of people involved, etc., inspection activities can be classified and examined individually, and then compared to one another.
- Software inspection deals with finding software defects through critical examination by human inspectors.
- As a result of this direct examination, the detected software defects are typically precisely located, and therefore can be fixed easily in the follow-up activities.

THE CASE FOR INSPECTION

- The main difference between the object types of inspection and testing, namely executable programs for testing and all kinds of software artifacts for inspection,
- The primary reason for the existence of inspection: One does not have to wait for the availability of executable programs before one can start performing inspection.
- Consequently, the urgent need for QA and defect removal in the early phases of software development can be supported by inspection, but not by testing.
- In addition, various software artifacts available late in the development can be inspected but not tested, including product release and support plans, user manuals, project schedule and other management decisions, and other project documents.
- Basically, anything tangible can be inspected.

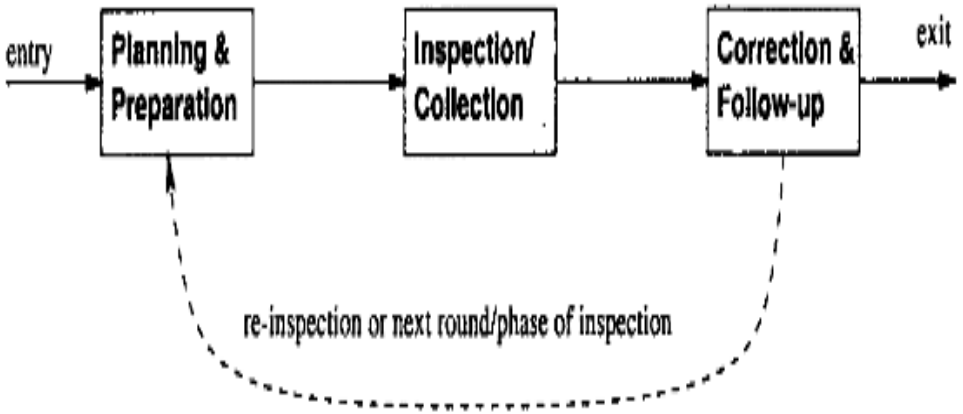
INSPECTION TECHNIQUES W.R.T OBJECTS

- Wide variety of objects for inspection, so inspection techniques also vary considerably.
- For example, code inspection can use the program implementation details as well as product specifications and design documents to help with the inspection.
- Inspection of test plans may benefit from expected usage scenarios of the software product by its target customers.
- Inspection of product support plans must take into account the system configuration of the product in operation and its interaction with other products and the overall operational environment.
- Consequently, different inspection techniques need to be selected to perform effective inspection on specific objects.

DEGREE OF FORMALITY

- Similarly, there are different degrees of formality, ranging from informal reviews and checks to very formal inspection techniques associated with precisely defined individual activities and exact steps to follow.
- Even at the informal end, some general process or guidelines need to be followed so that some minimal level of consistency can be assured, and adequate coverage of important areas can be guaranteed.
- In addition, some organizational and tool support for inspection is also needed.

GENERIC INSPECTION PROCESS



Generic inspection process

PLANNING AND PREPARATION

- Inspection planning needs to answer the general questions about the inspection, including:
  - What are the objectives or goals of the inspection?
  - What are the software artifacts to be inspected or the objects of the inspection?
  - Who are performing the inspection?
  - Who else need to be involved, in what roles, and with what specific responsibilities?
  - What are the overall process, techniques, and follow-up activities of the inspection?

INSPECTION OR COLLECTION

- This step roughly corresponds to the execution of QA activities in generic quality engineering process.
- This step is also referred to as collection or collection meeting.
- The focus of this step is to detect faults in the software artifacts inspected, and record the inspection results so that these faults can be resolved in the next step.

CORRECTION AND FOLLOW-UP

- The discovered faults need to be corrected by people who are responsible for the specific software artifacts inspected. For example, in design or code inspection, the responsible designer or programmer, often labeled as design or code "owners" in industry, need to fix the design or code.
- There should be some follow-up activities to verify the fix.
- Sometimes, new inspection rounds can be planned and carried out,

FAGAN INSPECTION

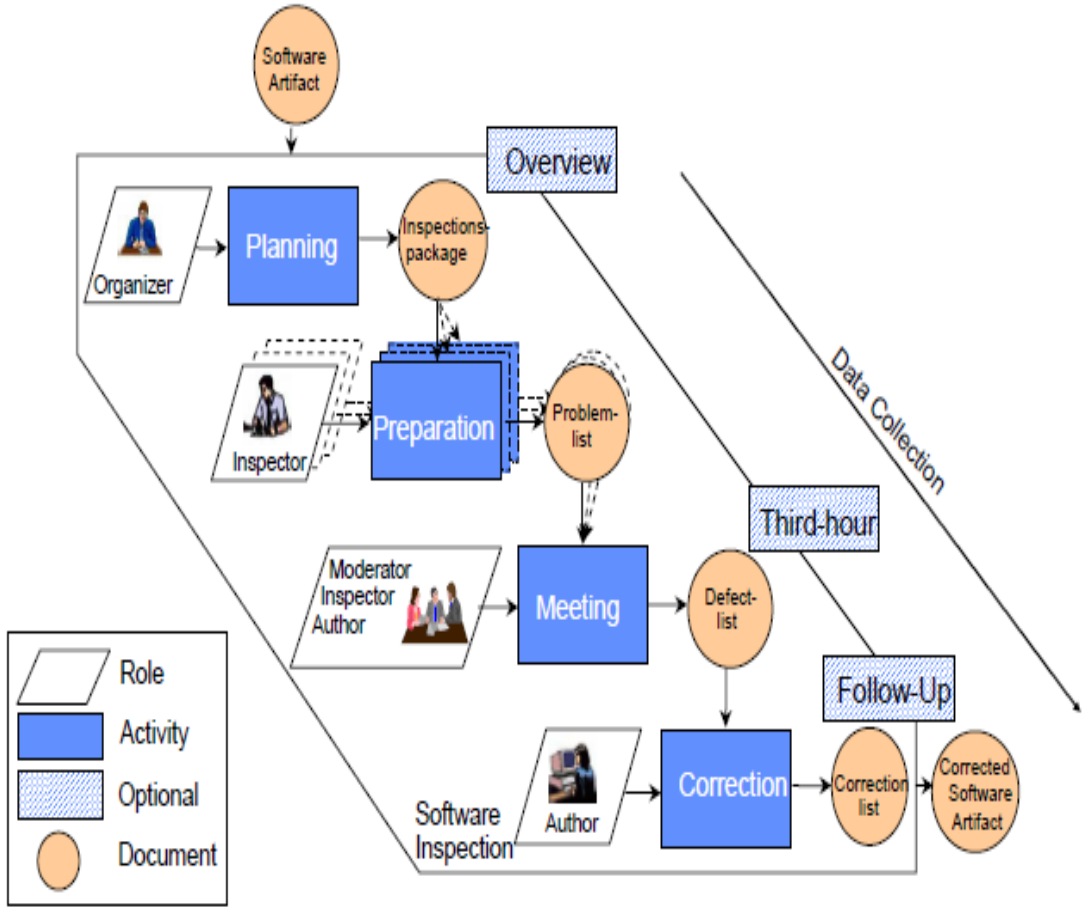
- The earliest and most influential work in software inspection is Fagan inspection (Fagan, 1976), which is almost synonymous with the term "inspection" itself.
- Fagan inspection has been used widely across different industrial boundaries and on many different software artifacts, although most often on program code.
- Almost all the other inspection processes and techniques can be considered as derivatives of Fagan inspection, by enhancing, simplifying, or modifying it in various ways to fit specific application environment or to make it more effective or efficient with respect to certain criteria.

PROCESS AND PARTICIPANTS

- Planning: Deciding what to inspect, who should be involved, in what role, and if inspection is ready to start.
- Overview meeting: The author meets with and gives an overview of the inspection object to the inspectors. Assignment of individual pieces among the inspectors is also done.
- Preparation: Individual inspection is performed by each inspector, with attention focused on possible defects and question areas.

PROCESS AND PARTICIPANTS

- Inspection meeting to collect and consolidate individual inspection results. Fault identification in this meeting is carried out as a consensus building process.
- Rework: The author fixes the identified problems or provides other responses.
- Follow-up: Closing the inspection process by final validation.



GENERIC INSPECTION PROCESS AND FAGAN INSPECTION PROCESS

- We can adapt the generic inspection program (slide 7) to depict Fagan inspection in the following:
- The **"planning and preparation"** block can be expanded into three sequential steps, "planning", "overview", and "preparation" in Fagan inspection.
- The **"inspection/collection"** is directly mapped to the "inspection" step.
- The **"correction and follow-up"** block can be expanded into two sequential steps, "correction", and "follow-up".
- The dotted link for the next round of inspection is eliminated.

INSPECTION TEAM

- Fagan inspection typically involves about four people in the inspection team
- The potential inspectors are identified in the planning stage (Step 1) from those designers, developers, testers, or other software professionals or managers, who are reasonably familiar with the software artifacts to be inspected, but not necessarily those who directly work on it.
- An ideal mix would include people with different roles, background, experience, and different personal or professional characteristics, to bring diverse views and perspectives to the inspection.

INSPECTION WORK

- The assignment of individual pieces for inspection among the inspectors needs to take two issues into consideration: overall coverage and areas of focus.
- On the one hand, different inspectors will be assigned different pieces so as not to unnecessarily duplicate inspection effort.
- On the other hand, some important or critical pieces may need the focused attention of more than one inspector.
- The inspection meeting should be an organized event, with one inspector identified as the leader or moderator, who oversees the meeting and ensures that it fulfills its main purpose of defect identification and consolidation.
- The meeting typically lasts two hours or less.

INSPECTION WORK

- The focus is on defect detection and consolidation only, but not on defect resolution,
- A group of people working together would find and confirm problems that individuals may not.
- However, each individual must be fully prepared and bring forward candidate problems for the team to examine together.
- In this group process, false alarms will be eliminated, and consolidated defects will be confirmed, recorded, and handed over for authors to fix.

GENERAL OBSERVATIONS AND FINDINGS

- The importance of preparation
- Variations with team size, moderator role, and session coordination
- Defect detection techniques used in inspection
- Additional use of inspection feedback

OTHER INSPECTIONS AND RELATED ACTIVITIES

- Variations to Fagan inspection have been proposed and used to effectively conduct inspection under different environments.
- Some of them are direct responses to some of the general findings of Fagan inspection described above. We organize these inspection techniques and processes along two dimensions:
  - size and scope of the inspection,
  - formality of the inspection.

INSPECTIONS OF REDUCED SCOPE OR TEAM SIZE

- Fagan inspection teams typically consist of four members.
- However, some software artifacts are small enough to be inspected by one or two inspectors.
- Similarly, such reduced-size inspection teams can be used to inspect software artifacts of limited size, scope, or complexity.
- This so-called two-person inspection is the simplification form of Fagan inspection, with an author-inspector pair, but following essentially the same process for Fagan inspection.
- This technique is cheaper and more suitable for smaller-scale programs, small increments of design and/or code in the incremental or iterative development, or other software artifacts of similarly smaller size.

INSPECTIONS OF REDUCED SCOPE OR TEAM SIZE

- Another implementation of two-person inspection is the reversible author- inspector pair, that is, the individuals in the pair complement their roles by inspecting each other's software artifacts.
- easier to manage technique because of the mutual benefit to both individuals instead of the asymmetric relation in Fagan inspection, where the author is the main beneficiary while the inspectors are performing "service" to others or to the company.
- The idea of two-person inspection is also found in the new development paradigm called agile development and extreme programming, where the so-called paired programming resembles the author-inspector pair.
- Informal inspection
- Reduce cost

INSPECTIONS OF ENLARGED SCOPE OR TEAM SIZE

- A common extension to Fagan inspection is based on the observation that during Fagan inspection meeting, people tend to linger on discovered defects and try to both find the causes for them and suggest fixes.
- These additional activities in the meeting would interfere with the main task of defect detection and confirmation in Fagan inspection and tend to prolong the meeting.
- On the other hand, these activities do add valuable information to the feedback that can be used to improve the overall inspection process and product quality.
- A solution to this problem is proposed in the Gilb inspection.

GILB INSPECTION

- In Gilb inspection an additional step, called "process brainstorming", is added right after the inspection meeting in Fagan inspection.
- The focus of this step is root cause analysis aimed at preventive actions and process improvement in the form of reduced defect injections for future development activities.
- There are several other special features to Gilb inspection, as characterized below:
- The input to the overall inspection process is the product document, rules, checklists, source documents, and kin documents. The emphasis is that any technical documentation, even diagrams, can be inspected.
- The output from the overall inspection process is the inspected (and corrected) input documents, change requests, and suggested process improvements.
- The inspection process forms a feedback loop, with the forward part resembling Fagan inspection but with the added step for process brainstorming, and the feedback part consisting of inspection statistics and adjustment to inspection strategies.
- Multiple inspection sessions are likely through this feedback loop

GILB INSPECTION

- The Gilb inner inspection steps are as follows (with Fagan inspection equivalent given inside parenthesis):
  - 1. planning (same),
  - 2. kickoff (overview),
  - 3. individual checking (preparation),
  - 4. logging meeting (inspection),
  - 5a. edit (rework),
  - 5b. process brainstorming (),
  - 6. edit audit (follow-up).
- 5a and 5b are carried out in parallel in Gilb inspection.
- The team size is typically about four to six.
- Checklists are extensively used, particularly for step 3, individual checking.

PHASED INSPECTION

- Another variation to the above is the phased inspection, where the overall inspection is divided into multiple phases with each focusing on a specific area or a specific class of problems.
- These problems not only include the defects (correctness problems), but also issues with portability, maintainability, etc.
- This inspection is typically supported by some form of checklist and related software tools.
- The dynamic team make-up reflects the different focus and skill requirements for individual phases.

INFORMAL DESK CHECKS, REVIEWS, AND WALKTHROUGHS

- **Desk check** typically refers to informal check or inspection of technical documents produced by oneself, which is not too different from proofreading one's own writings to catch and correct obvious mistakes.
- Advance software tools can detect things as mis-spelling, format, syntactical errors.
- Instead, desk checks should focus on logical and conceptual problems, to make effective use of the valuable time of software professionals.

REVIEW

- **Review** typically refers to informal check or inspection of technical documents, but in this case, produced by someone else, either organized as individual effort, or as group effort in meetings, conference calls, etc.
- The focus of these reviews should be similar to desk checks, that is, on logical and conceptual problems.
- The differences in views, experience, and skill set are the primary reasons to use some reviews to complement desk checks.
- In most companies, the completion of a development phase or sub-phase and important project events or milestones are typically accompanied by a review, such as requirement review, design review, code review, test case review, etc.

REVIEWS

- A group of people carefully examine part or all of a software system and its associated documentation.
- Purpose: to find defects (errors) before they are passed on to another software engineering activity or released to the customer.

(INFORMAL) REVIEWS

Why make informal reviews?

- Everyone makes mistakes
- Create open atmosphere (increase productivity)
- Programming is a social activity (or should be)
- Find errors in program early (before it is run the first time)
- Find quality issues
- Improve programming skills of all involved
- Anything can be reviewed
  - (. . . , use cases, documentation, . . .)

HOW TO HOLD A REVIEW MEETING?

Purpose: to *evaluate* a software product to:

- determine its suitability for its intended use
- identify discrepancies from specifications & standards

Participants read documents in advance

- then bring their comments to a meeting for discussion

A review:

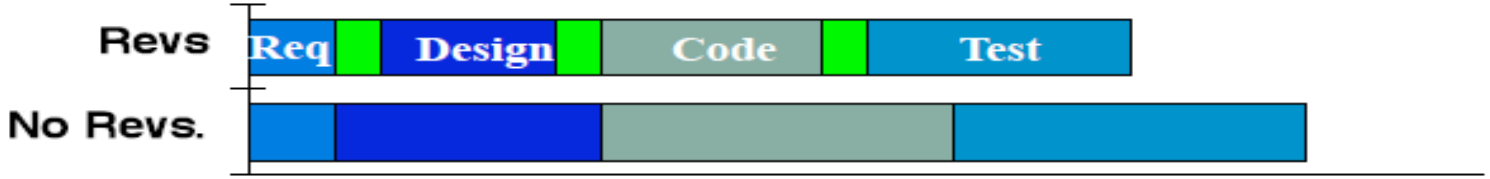
- may provide recommendations and suggest alternatives
- may be held at any time during a project
- need not reach conclusions on all points

REVIEW MEETING - WHAT TO AVOID?

- Improvements to the program
- Blaming programmers
- Finger pointing

WHY REVIEW? IF WE ALREADY TEST!

Reviews improve schedule performance.



Reviews reduce rework.

- Rework accounts for 44% of dev. cost!
- Reqs (1%), Design (12%), Coding (12%), Testing (19%)

Reviews are *pro-active tests*.

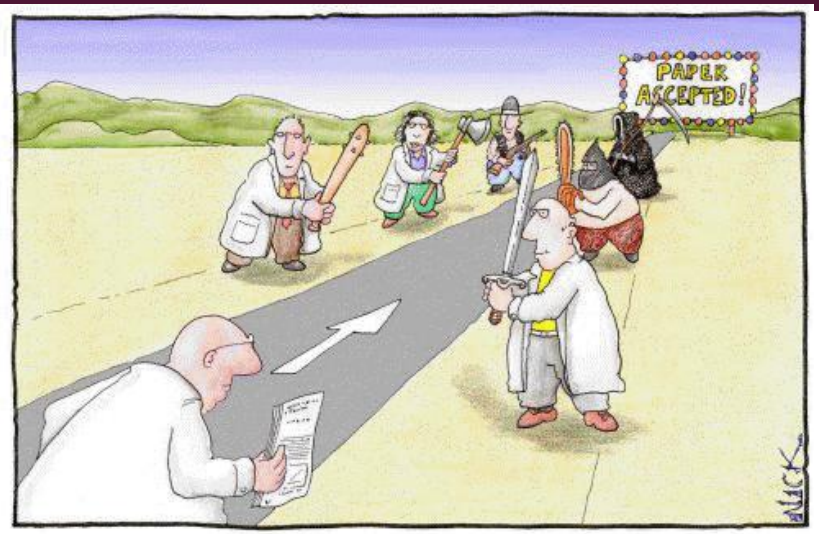
- Find errors not possible through testing.

Reviews are training.

- Domain, corporate standards, group

PEER REVIEW

- In software development, peer review is a type of software review in which a work product (normally some form of document) is examined by its author and one or more colleagues, in order to evaluate its technical content and quality.
- Peer reviews identify problems and fix them early in the lifecycle.



FORMAL TECHNICAL REVIEWS (FTR)

- FTR is one of the techniques for improving software quality.
- The FTR is a software quality assurance activity with the objectives to uncover errors in function, logic or implementation for any representation of the software.
- FTR (Formal Technical Review) is also a learning ground for junior developers to know more about different approaches to software analysis, design and implementation.

FORMAL TECHNICAL REVIEWS (CONT..)

- FTR activities include walkthroughs and inspections.
- Formal technical review methods began in 1976 with the publication of Fagan's code inspection techniques.
- Reviews are the most widely used approach for assessing software quality.

WALKTHROUGH

- A special form of review is called walkthrough,
- a more organized review typically applied to software design and code.
- Meetings are usually used for these walkthroughs.
- The designer or the code owner usually leads the meeting, explaining the intentions and rationales for the design or the code, and the other reviewers (meeting participants) examine these design code for overall logical and environmental soundness and offer their feedback and suggestions.
- Defect detection is not the focus.
- Typically, these meetings require less time and preparation by the participants except for the owners.
- In practical applications, these informal checks, reviews, and walkthroughs can be used in combination with formal inspections.

AUDIT

Finding the problem is more important than the solution

ALBERT EINSTEIN

SOFTWARE AUDITS

A software audit is an independent evaluation of software products or processes to ascertain compliance to standards, specifications, and procedures based on objective criteria that includes documents that specify:

- The form or content of the product to be produced
- The process by which the products shall be produced
- How compliance to standards or guidelines shall be measured”

INSPECTION LIMITATIONS

- As a human-intensive QA alternative, inspection also suffers from its own limitations, including:
  - Difficulties in dealing with dynamic and complex interactions often present among many different components or functions in large software systems;
  - Difficulties with task automation because of the human expertise involved. Communication and analysis tools can only help inspection to a limited degree, but are not able to replace human inspectors whose ability to detect conceptual problems cannot be matched by any software tool.

DIFFERENCE BETWEEN REVIEW AND INSPECTION

Aspect	Review	Inspection
Definition	A general evaluation of a product, process, or document to ensure quality.	A formal, structured process to identify defects in a product or document.
Formality	Informal or semi-formal.	Highly formal and process-driven.
Focus	Broad focus on overall quality, functionality, and completeness.	Specific focus on defect identification and correction.
Participants	May involve peers, managers, or stakeholders.	Involves trained moderators, reviewers, and producers of the document.
Process	Flexible, may not follow a strict protocol.	Follows a defined protocol, including pre-meetings, checklists, and records.
Output	Suggestions and feedback for improvement.	Detailed defect logs and a formal report.
Examples	Code reviews, document reviews, design reviews.	Software code inspections, design inspections, or test case inspections.

FORMAL VERIFICATION

- The basic idea of formal verification is to verify the correctness, or absence of faults, of some given program code or design against its formal specifications.
- Therefore, the existence of formal specifications is a prerequisite for formal verifications.
- Both formal specification techniques and formal verification techniques are referred to as formal methods
- When formal methods are used for software development, formal specifications are used upstream for requirement analysis and product specifications, and formal verifications or analyses are used downstream to verify the design and code before additional verification and validation (V&V) activities are carried out.

THE USE OF FORMAL METHODS CAN BE SUMMARIZED IN THE FOLLOWING TWO-STEP PROCESS

- **1. Constructing formal specifications:** The expected behavior and other properties of the software artifacts are represented in formal models. These models of program code, design, and expected behavior are typically product-dependent, which can be specifically constructed for formal verification and analysis purposes.
- However, to reduce cost as well as to benefit from formal development methods, these models can be the same as the formal specifications or adapted by formalizing informal specifications for the product.
- **2. Performing formal verifications:** Formal analysis techniques are applied on the product components, typically product code or formal designs, to verify their correctness with respect to their formal specifications, or to check for certain properties. These techniques are typically organized as a product-independent framework of rules that serve as the basis of formal inferences or analyses.

FORMAL SPECIFICATIONS PRODUCED IN SEVERAL DIFFERENT FORMS

- Two general categories: descriptive specifications and operational specifications
- **Descriptive specifications** focus on the properties or conditions associated with software products and their components. For example:
  - Entity-relationship diagrams are commonly used to describe product components and connections.
  - Logical (or logic) specifications focus on the formal properties associated with different product components or the product as a whole.
  - Algebraic specifications focus on functional computation carried out by a program or program-segment and related properties.
- **Operational specifications** focus on the dynamic behavior of the software systems. For example:
  - Data flow diagrams specify information flow among the major functional units.
  - UML diagrams specify individual behavior for major objects or product components.
  - Finite-state machines (FSMs) describe control flow in state transitions.

APPLICATIONS, EFFECTIVENESS, AND INTEGRATION ISSUES

- As a QA alternative, formal verification and analysis techniques have not gained the wide usage and popularity anywhere close to that for inspection or testing.
- Besides their own limitations related to cost, process, and other issues, one key reason is the general lack of the required expertise, which can be partially solved or alleviated by our education system.
- Any product can potentially benefit from the use of these formal techniques. However, due to the required expertise for the personnel involved in the verification and analysis activities and the related cost, these techniques are mostly used in small software, or in a small subset of larger software systems that require ultra-high quality or where the damage of failures is substantial.
- Such large software systems include software for safety critical systems, or critical components or functions for large software systems.

DEFECT CONTAINMENT

- **Defect containment** refers to the strategies and processes used to identify, isolate, and mitigate defects in software during the development lifecycle, preventing them from propagating to later stages or to the final product. It aims to reduce the cost and impact of defects by catching them early in the development process.

DEFECT CONTAINMENT : FAULT TOLERANCE AND FAILURE CONTAINMENT

- Defect containment techniques attempt to contain the defects through two generic means:
- **Fault tolerance techniques** limit defect manifestation to a local area to avoid global failures, through the use of some duplication designed into the software systems or their operations.
- **Failure containment techniques** reduce the impact or damage associated with certain system failures so that some accidents can be avoided or the related damage can be minimized.
- Accidents are a subset of failures with severe consequences, and the pre-conditions to such accidents are called hazards. Most of the failure containment activities are associated with safety-critical systems, where the main concerns are for the system to be as safe or to be as accident free as possible.
- Hazard analysis and resolution play an important role in identifying hazards and dealing with them to contain failures related to potential accidents, thus ensuring system safety.

FAULT TOLERANCE

- **Fault-tolerance** is the process of working of a system in a proper way in spite of the occurrence of the failures in the system. Even after performing the so many testing processes there is possibility of failure in system. Practically a system can't be made entirely error free. hence, systems are designed in such a way that in case of error availability and failure, system does the work properly and given correct result.
  - **N-version programming**
  - **Recovery Blocks**
  - **Check-pointing and Rollback Recovery**

EXAMPLES:

- For example, a simple air pressure sensor in a car tire pressure monitoring system (TPMS) can detect the air overfill and notify the driver via the car dashboard.
- In this case, the detection and display is the only acceptable tolerance level for this fault event. The customer can safely disengage the air hose before rupturing the tire.
- For instance, in the case of overpressure of petroleum products in a vessel, the system is triggered by relevant pressure sensors. It opens the safety pressure valve and exhausts the vapors out in the flare stack.
- In this example, the containment is carried out by diverting the high-pressure flammable vapor to the exhaust stack, protecting the system from fire or explosion.

EFFECTIVENESS COMPARISON

- Different QA alternatives can be compared by examining the specific perspectives of defect they are dealing with, what kind of problems they are good at addressing, their suitability to different defect levels and pervasiveness, and their ability to provide additional information for quality improvement.

EFFECTIVENESS COMPARISON : DEFECT PERSPECTIVE

- Among the different defect related perspectives and concepts, the QA alternatives can be compared by examining whether they are dealing with error sources, errors, faults, failures, or accidents.
- This examination can be broken down further into two parts:
- Detection or observation of specific problems from specific defect perspectives during the performance of specific QA activities.
- Types of follow-up actions that deal with the observed or detected problems in specific ways as examined from the defect perspectives.

EFFECTIVENESS COMPARISON: DEFECT PERSPECTIVE

QA Alternative	Defect Perspective	
	At Observation	At Follow-up (& Action)
testing	failures	fault removal
defect prevention	errors & error sources	reduced fault injection
inspection	faults	fault removal
formal verification	(absence of) faults	fault absence verified
fault tolerance	local failures	global failures avoided
failure containment	accidents	hazards resolution & damage reduction

EFFECTIVENESS COMPARISON: PROBLEM TYPES

- Different QA alternative might be effective for different types of problems, including dealing with different perspectives of defects, ranging from different errors and error sources, various types of faults, and failures of different severity and other characteristics.

Main problem types dealt with by different QA alternatives

QA Alternative	Problem Types
testing	dynamic failures & related faults
defect prevention	systematic errors or conceptual mistakes
inspection	static & localized faults
formal verification	logical faults, indirectly
fault tolerance	operational failures in small areas
failure containment	accidents and related hazards

EFFECTIVENESS COMPARISON: PROBLEM TYPES

- Defect prevention works to block some errors or to remove error sources to prevent the injection of related faults. Therefore, it is generally good at dealing with conceptual mistakes made by software designers and programmers.
- Once such conceptual mistakes can be identified as error sources, they can be effectively eliminated.
- One key difference between inspection and testing is the way faults are identified: inspection identifies them directly by examining the software artifact, while failures are observed during testing and related faults are identified later by utilizing the recorded execution information.
- This key difference leads to the different types of faults commonly detected using these two techniques.
- Inspection is usually good at detecting static and localized faults which are often related to some common conceptual mistakes, while testing is good at detecting dynamic faults involving multiple components in interactions.

THE REASONS OF DIFFERENCES BETWEEN THE TWO TYPES OF QA ALTERNATIVES

- Inspection involves static examination while testing involves dynamic executions. Therefore, static problems are more likely to be found during inspection, while dynamic problems are more likely to be found during testing.
- It is hard for human inspectors to keep track of multiple components and complicated interactions over time, while the same task may not be such a difficult one for computers. Therefore, testing is generally better at detecting interaction problems involving multiple components.
- Human inspectors can focus on a small area and perform in-depth analysis, leading to effective detection of localized faults.

EFFECTIVENESS COMPARISON

- Formal verification deals with logical (or mathematical) correctness, and can be interpreted as extremely formalized inspection. Therefore, it shares some of the characteristics of inspection in dealing with static and logical problems.
- Problem identification is only a side-effect of failing to produce a correctness proof.
- Fault tolerance and failure containment are designed to work with dynamic operational problems that may lead to global failures or accidents.
- Fault tolerance techniques are good at isolating faults to only cause local failures but not global ones, while failure containment works to contain failures that may lead to accidents by dealing with hazards or reducing damage related to accidents.

EFFECTIVENESS COMPARISON: DEFECT LEVEL AND PERVASIVENESS

- Different QA techniques may be suitable for different defect levels or pervasiveness.

Defect levels where different QA alternatives are suitable

QA Alternative	Defect Level	
testing	low – medium	
defect prevention	low – high	(particularly pervasive problems)
inspection	medium – high	
formal verification	low	
fault tolerance	low	
failure containment	lowest	

EFFECTIVENESS COMPARISON: RESULT INTERPRETATION AND CONSTRUCTIVE INFORMATION

- Ease of result interpretation plays an important role in the application of specific QA techniques. A good understanding of the results is a precondition to follow-up actions.
- For example, both inspection and testing are aimed at defect removal. However, inspection results are much easier to interpret and can be used directly for defect removal. Testing results need to be analyzed by experienced software professionals to locate the faults that caused the failures observed during testing, and only then can these faults be removed.
- Result interpretation for formal verification, fault tolerance, and failure containment is harder than that for inspection and testing. A significant amount of effort is needed to analyze these results to support follow-up actions.
- For example, in a fault tolerant system using recovery blocks, repeated failures need to be dealt with off-line by analyzing the dynamic records. Much information related to unanticipated environment and usage not covered in the pre-planned testing activities may be included in these records.
- Similarly, failure containment results typically need additional analysis support.

EFFECTIVENESS COMPARISON: RESULT INTERPRETATION AND CONSTRUCTIVE INFORMATION

Ease of result interpretation for different QA alternatives and amount of constructive information/measurements

QA Alternative	Result Interpretation	Information/Measurement
testing	moderate	executions & failures
defect prevention	(intangible)	experience
inspection	easy	faults, already located
formal verification	hard	fault absence verified
fault tolerance	hard	(unanticipated) environments/usages
failure containment	hard	accident scenarios and hazards

COMPARISON SUMMARY

General comparison for different QA alternatives

QA Alternative	Applicability	Effectiveness	Cost
testing	code	occasional failures	medium
defect prevention	known causes	systematic problems	low
inspection	s/w artifacts	scattered faults	low – medium
formal verification	formal spec.	fault absence	high
fault tolerance	duplication	rare-cond. failures	high
failure containment	known hazards	rare-cond. accidents	highest

That is all