

Software Defect Prevention

Presentation By
Chethana Kuloor
Dept. Of Electrical and Computer Engg.

Abstract

With the increase in demand for software products their cost and complexity go on increasing from day to day. In order to achieve higher quality and increased productivity with reduced time to market organizations must adopt one or other process improvement technique. One way to reduce the cost and time is to prevent the injection of defects into the software product.

In this report the concept of *Software Defect Prevention* is taken into consideration. The purpose of defect prevention is explained. The difference between defect detection and defect prevention is briefly explained. The major causes for the defect is briefly discussed. The various defect categories are mentioned. The two major approaches, *Causal analysis approach* and *Cleanroom approach* are described. The effect of cost in implementing the process is mentioned. Costs and benefits of defect prevention is also considered. Finally it concludes on the necessity of defect prevention in an organization.

Table of Contents

1. [Introduction](#)
2. [What are defects ?](#)
3. [Defect detection and Defect Prevention](#)
4. [Why defect prevention is necessary ?](#)
5. [Principles of Defect Prevention](#)
6. [Various methods for defect prevention](#)
 1. [A process integrated approach to defect prevention](#)
 - [Metric Data Collection and Defect reporting](#)

- [The Kickoff Meeting](#)
- [Causal Analysis](#)
- [Cause Analysis Charts and Diagrams](#)
- [Cause Analysis Meeting](#)
- [The Action Team](#)
- [Action Team Meeting](#)
- [Action Progress Tracking](#)
- [Prevention Feedback](#)
- [Cost and Benefits of Defect Prevention](#)
- [Management's Role](#)

2. [Clean Room Software Approach](#)

- [Management Practice](#)
 - [Incremental Development](#)
 - [Team Ownership](#)
- [Development Practice](#)
 - [Black Box Specification](#)
 - [Clear Box Design](#)
 - [Abstract Data Model](#)
 - [State Box Design](#)
- [Team Review](#)
- [Testing Practices](#)
 - [Usage Modeling](#)
 - [Statistical Testing](#)

3. [The Personal Software Process](#)

4. [Prototyping](#)

5. [Formal Specification](#)

7. [Conclusion](#)

8. [References](#)

1. Introduction

The cost and complexity of the software process goes on increasing from day to day. In order to achieve higher quality and increased productivity with reduced time to market organizations must adopt process improvement techniques. Once an organization has reached a stage at which it can establish process measurements and quality plans, it can make use of these capabilities to process improvement. In this competitive world, maintaining productivity and quality of the product with reduced time to market is very important for an organization. To achieve a competitive priority in the business world, each organization should adopt one or other way to optimize its development process. Recent effort to improve quality in software have concentrated on defect detection and prevention. One of the main causes for both delay in the delivery of software products and the high cost of developing systems is due to defects in system. Most of the development time is used in finding the errors and correcting them. The errors can crop up at any stage in the process life cycle and can lead to system failure during their work time or even before their completion. Therefore defect prevention is a necessary action to be taken.

2. What are defects ?

The word *defect* refers to something that is wrong with a program, such as a syntax error, a misspelling, a punctuation mistake, or an incorrect program statement [1]. They can be extra statements, incorrect statements, or omitted program sections. There can be defects even in architecture, design, and testing. A defect is therefore anything that deviates the program's ability to work in the expected manner.

3. Defect Prevention Vs Defect Detection

Defect prevention is different from defect detection. The process of preventing an error from occurring is defect prevention whereas the process of finding the errors which are already present in program is defect detection. As already mentioned the task of finding defects is always difficult. Fixing the errors is expensive and time consuming. An error occurred during the design stage, if not prevented may cause problems further in the implement and testing stage. ***The process of improving the quality and productivity by preventing the injection of defects into a software product*** is called defect prevention [2]. Defect prevention should be considered at the beginning of

every process stage. Using the historical data available, all the defects that can occur during that stage should be taken into consideration. Once a defect is injected into the product the only way is to find it and correct it. Defects can be detected during the validation and inspection process. The defects remaining even after inspection can be detected during testing. Some defects need fixing even after customer release. By having a proper Inspection and verification process and effective testing the defects incorporated in product can be eliminated.

Errors can occur at any stage of a software development. Finding the errors in early stage of the software development cycle and taking necessary actions to prevent their repetition is defect prevention. The actions may be short term (immediate actions) or long term(to be implemented in future).

4. Why defect prevention is necessary ?

To meet the increasing demand for software product, progressively large and complex programs will be needed. Since these large, complex programs will be used in increasingly sensitive applications, software defects will become more dangerous to society. For the foreseeable future, programs will continue to be designed by the error-prone humans. This means that with the present methods the number and severity of the errors encountered by the system will increase. An error detected in an early stage of development is easy to analyze and correct. By preventing defects, their cumulative effects to the future development cycle can be reduced. Fixing an error at an early stage of development is less expensive. The costs of finding and fixing errors increase exponentially in the later stages.

Studies showed that, if defects can be ironed out during the development stage rather than at the end of the life cycle by using a suitable defect prevention technique, up to 90% of total defects can be reduced [2]. If defects can be prevented even before their entry to the system more time can be spent on improving and adding extra functionality to the system. When the total error content is reduced, the test and inspection processes are more effective. When a program contains a large number of trivial errors, the reviewers will likely be distracted by the volume of detail and may overlook serious problems. Sometimes the errors in the system may be life threatening. Thus a defect prevention process is important in an organization for optimization.

6. Principles of Defect Prevention

Like all other software processes, defect prevention also needs certain procedures to be followed. It can not be established by one or two people. There should be commitment to the process.

The principles of software defect prevention are [\[3\]](#):

1. Programmers should evaluate their own errors. They should be involved themselves in the analysis of errors because,
 - The best person to determine the cause of an error is the person who made it. Their involvement provides a more accurate assessment of what really happened in creating the error.
 - They are more likely to suggest meaningful ideas for correction because they understand the error more thoroughly.
 - If only one person is assigned to do the causal analysis for an entire project, he/she will soon suffer burnout. So there is a chance that the analyzer may overlook some of causes.
2. Causal analysis should be a part of the process. If the activities for problem analysis are not established as part of the process, the analysis work will be done in a haphazard way or it will fade away soon. Establishing it as a part of the process spreads the work, involves the appropriate people, develops responsibilities in people, and creates an environment that emphasizes prevention of errors as a daily activity.
3. Feedback should be a part of the process. Even though programming process have been used for quite sometimes, detected errors and statistics have seldom been used as feedback to improve those processes. A *looping* effect for the detection and correction of errors has been defined, but an equivalent feedback of data for improvement to the process or for the education of programmers has not been fully exploited. Errors and statistics must be made visible to the programmers to allow them to refine their technique and to learn from their mistakes.
4. There is no single cure that can solve all the problems. Defect prevention technique is implemented in incremental steps. The causes of errors are removed one at a time.
5. Process improvement must be an integral part of the software process. As the volume of software process change grows, equal effort and discipline should be invested in defect prevention, and defect detection and correction. This requires that the process must be architected, designed, inspections and test be conducted, baselines established, problem reports written, all changes tracked and controlled.

7. Various methods for defect prevention

Defect injection to a software process can be reduced by adopting many methods. Some of them are as follows:

- A process integrated approach which was established at IBM.
 - Cleanroom Software Approach which was also first established at IBM.
 - The Personal Software Process
 - Prototyping.
 - Formal Specification.
-

7.1. A process integrated approach to defect prevention

This method explains how defect prevention can be achieved by systematically implementing various procedures such as causal analysis, feedback mechanism to the existing software process. Although the software process stages themselves remain the same with new methodology, additional activities are needed to facilitate defect prevention, such as kickoff meetings, causal analysis meetings, and action team follow-up [\[2\]](#), [\[4\]](#), [\[5\]](#), [\[6\]](#).

The following figure shows a normal software process model represented in ETVX form [\[4\]](#).

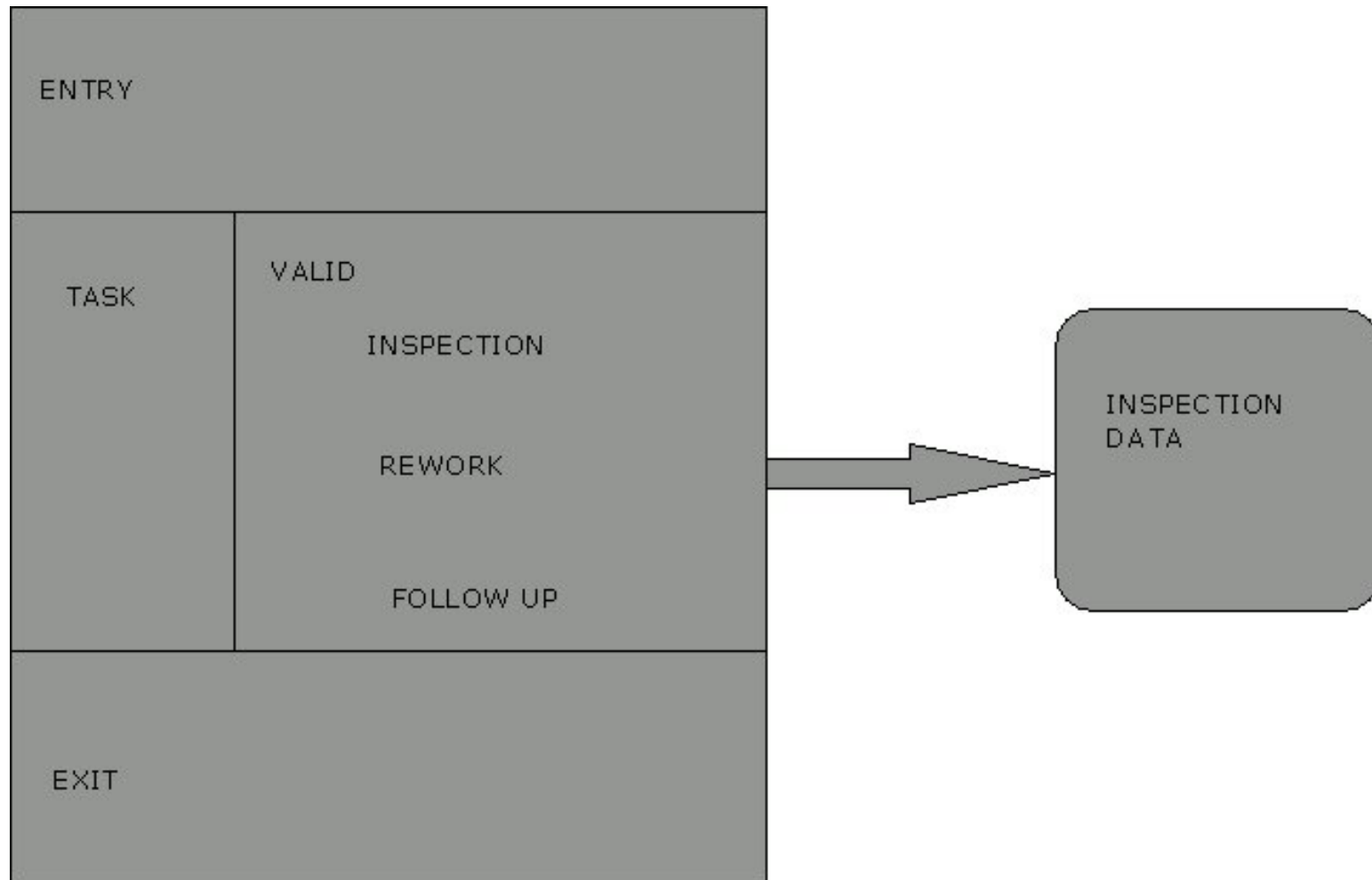


Figure 2. A Process Stage Without Defect Prevention

7.1.1. Metric Data Collection and Defect reporting

Metric data collection and defect reporting is an important step in a defect prevention process. Defect reports are prepared at each stage , and metrics data such as faults per line of code, rework time for each corrected fault are documented and updated. Using data collected

from various projects, an organization wide metrics repository can be built up.

7.1.2. The Kickoff Meeting

At the **ENTRY** sub stage kickoff meeting is conducted. The defect prevention methodology adds a formal kickoff meeting to the **ENTRY** sub stage. At the beginning of a stage the team assigned to the unit of work meets together with the following objectives [\[4\]](#):

- *Review what is available as input.* The input to the stage is evaluated to make sure that all members of the team understand what is available as input and whether or not everything from the previous stage is complete.
- *Review process.* The team discusses the output requirements for the stage going over the examples, guidelines, explicit expectations of the team leader. Examples are extremely useful in eliminating contradictory interpretations of the guidelines or descriptions of what the output should be. Sample outputs help to clarify exactly what is expected. Guidelines and kickoff packages from previous meetings should be available for each stage to ensure consistency between teams and to provide documentation that can be incrementally improved.
- *Review error checklists.* The most common causes of error for the particular stage are reviewed from a checklist. From this the developers will know what are the common possible errors that may occur in that stage and they can be cautious. These error checklists are prepared using defect reports. They are made available to each stage and are constantly updated.
- *Set team goals.* Normally quality goal will be set for a product for each stage and will be documented in a quality plan.

After the kickoff meeting the task is performed. The product is inspected. The errors found during this stage are recorded. Any rework suggested is also reported. These information are necessary to conduct the causal analysis. Since rework is usually done by the person who created the error, she/he can determine the categories of the errors, can also suggest some preventive actions. All these information are collected and stored in a database.

7.1.3. Causal Analysis

The data gathered for the defects found during the validation and inspection process are analyzed to determine the origin, cause and types of errors. Cause analysis should be conducted as early as possible after a defect is found. To be most efficient, a modest backlog is required for each session. Shortly after the completion of the last module cause analysis meetings should be held for all the defects found during that stage. Cause analysis reviews must be held on a product after a reasonable number of user problems have been found after customer release.

How defects are caused?: To prevent an error, it is essential to understand the cause, location and type of the error occurred. The stage where the error originated is noted. Various projects have been carried out to learn about the distribution of errors in a software process. Four major categories which describe any error are [4]:

1. *Communication* errors result from a breakdown in communication between groups or among team members. For example, A design concept stated by a higher level designer is misinterpreted by a lower level designer.
2. *Education* errors occur when a team member's failure to understand something causes the error. Educational errors can be further divided into the following:
 - New Function: The programmer does not understand the function and makes an error.
 - Old Function: The base code or function is not well understood, and when a new function is added to it, the implementation causes the problem. (i.e. the programmer does not understand the base code or function well enough to know that the addition of new function causes regression problem.
 - Other: The programmer needs education in a subject other than the function being developed. (e.g., compiler knowledge)
3. An *Oversight* arises when all the possible cases or conditions are not considered or handled. (e.g., an error condition is missed)
4. A *Transcription* error occurs when the programmer knows every thing in detail about a program, but simply makes mistake. (e.g., types in the wrong label)

Studies have been conducted by James S. Collofello and Jeffery J.Buck, [5] to find the origin and types of errors that are common in a software process. They selected a version of a large, Centrex-type telephone switching system for their study. They adopted a postmortem defect categorization scheme to capture relevant information about each error. They documented when the defect was introduced, into what subsystem it was introduced, its type, and its cause. Based on information in the defect report each identified defect was analyzed later.

Their defect report had four data fields:

1. The origin field identified when the defect was introduced into the system. Defects were found to originate in the following four fields :
 - Defects were found by the execution of feature verifying a new feature or capability that was added to the existing product.
 - Defects were found by the execution of a feature with added new feature, verifying the feature that existed in the previous version of the product. Here the addition of new feature caused defect in the existing feature.

- Defects were found by the execution of a feature with added corrections to detected error, verifying a feature which was working properly on the previous version. Applied corrections caused errors here.
- Defects were found by the execution of a feature that existed in the base version of the product. Here the errors were those carried over from the previous version.

The distribution of defect origin can be graphically represented as in [Figure 3](#).

2. The physical location field identified in which area (E.g. subprogram) the defect was detected and corrected.
 3. The defect type field documented what type of defects were detected. The major types of errors found were documentation, logic, interface, standards, environment, and miscellaneous. The distribution of total types of defects found in their project is given by [Figure 4](#).
 4. The defect cause fields identify the cause of the defect detected. They are explained at the beginning of this section.
-

Distribution of Defect Origin

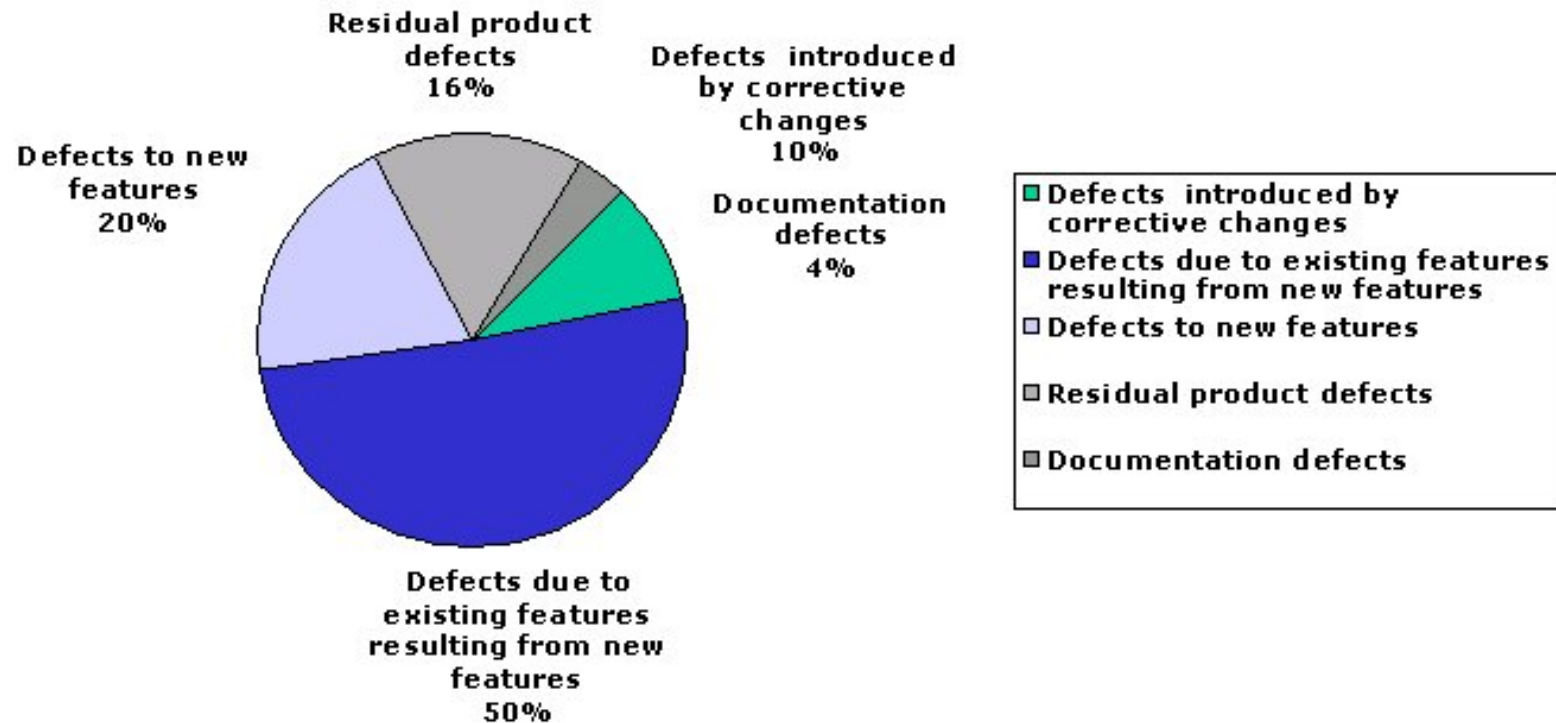


Figure 3. Distribution of Defect Origin

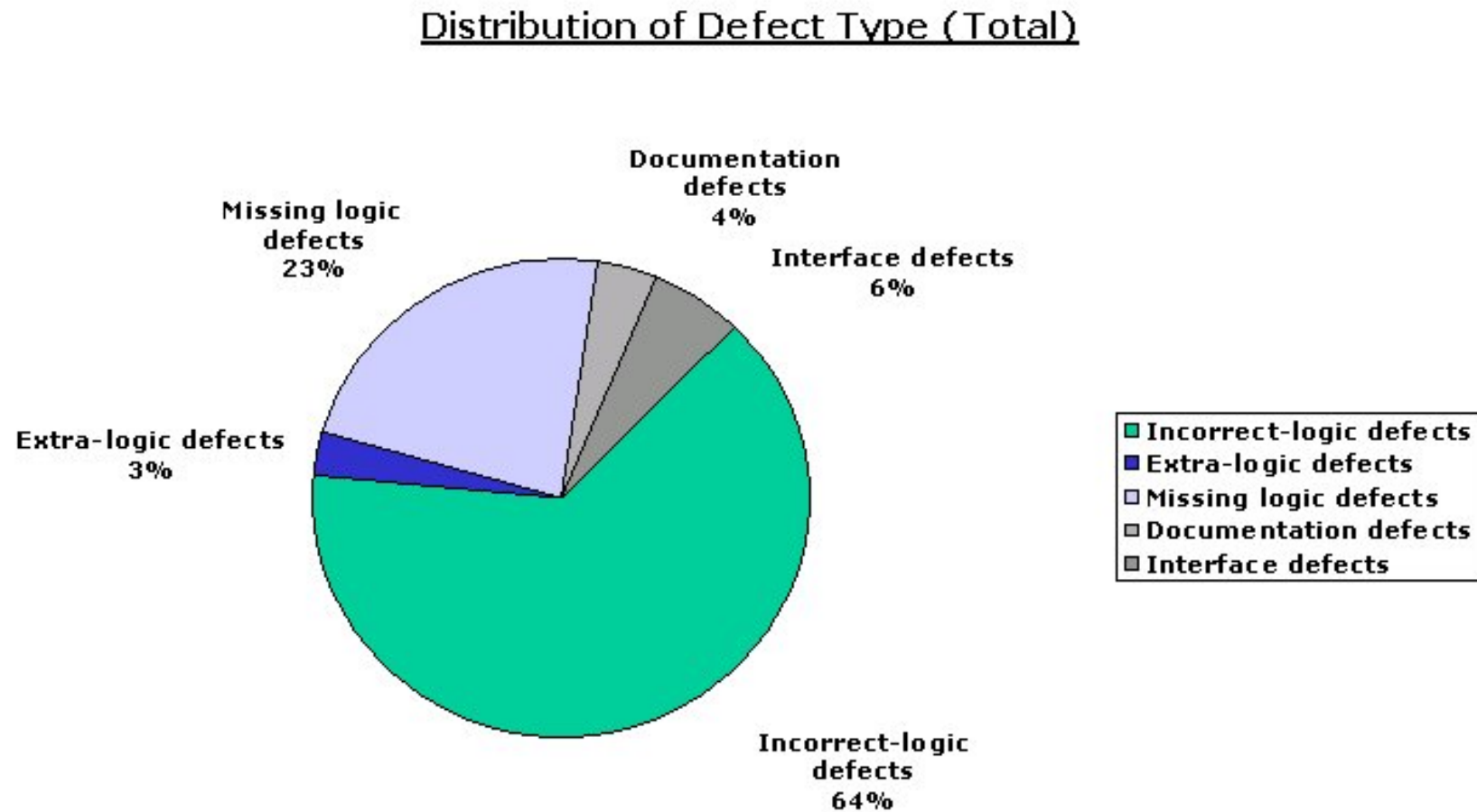


Figure 4. Distribution of Total Defect Types

The above example gives a clear picture of how and where defects are caused in a process. This knowledge is very important to adopt a defect prevention technique.

7.1.4 Cause Analysis Charts and Diagrams

Various charts and diagrams have been used for the purpose of cause analysis. They help to determine the priority of the action items to be taken to prevent or correct the defects. And some of them help to determine the various causes related to a particular defect. Some of them are as follows:

- *Pareto Charts*. A vertical bar graph used after data collection to rank problems or their causes are called Pareto charts [\[11\]](#). They are created by plotting the cumulative frequencies of the relative frequency data (event counts) in descending order. They are used to prioritize conflicting or competing problems so that the attention can be given to the most important areas. The major limitation of this chart is that the data should be in terms of counts or costs i.e., attributed data is needed. Examples of of Pareto charts are shown in [Figure 5](#) and in [Figure 6](#). [\[3\]](#)
- *Cause/Effect Diagrams*. These diagrams which are also called **Ishikawa Diagrams** are used to associate multiple possible causes with a single defect [\[11\]](#). Thus given a particular defect the Cause/Effect Diagram is constructed to identify and organize possible causes for it. The example cause/effect diagram is shown in [Figure 7](#). [\[3\]](#). The primary branch represents the defect which is labeled in the right side of the diagram. Each major branch of the diagram corresponds to a major cause that is directly associated with the defect. Minor branches represent more detailed causal factors.

The Pareto Chart and Cause/Effect Diagram examples are shown for the causal analysis process for a compiler project conducted by Grady and Caswell. [\[3\]](#)

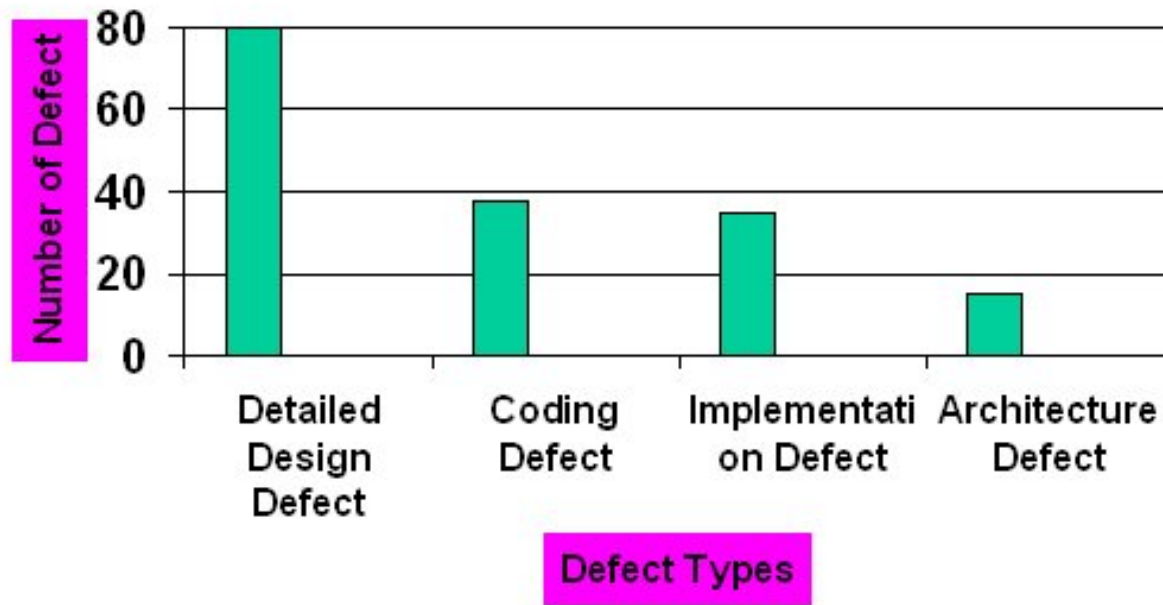


Figure 5. Pareto Chart for Compiler Defects. [\[3\]](#)

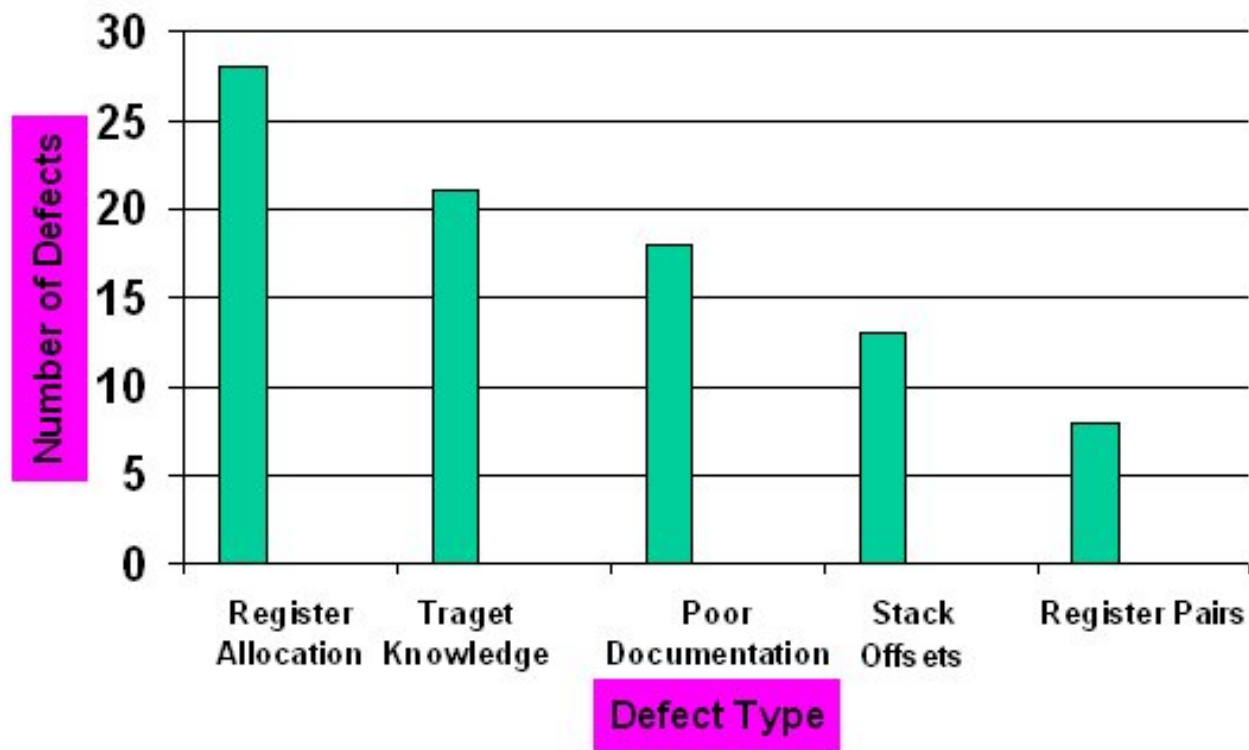


Figure 6. Pareto Chart for Compiler Design Defects [3].

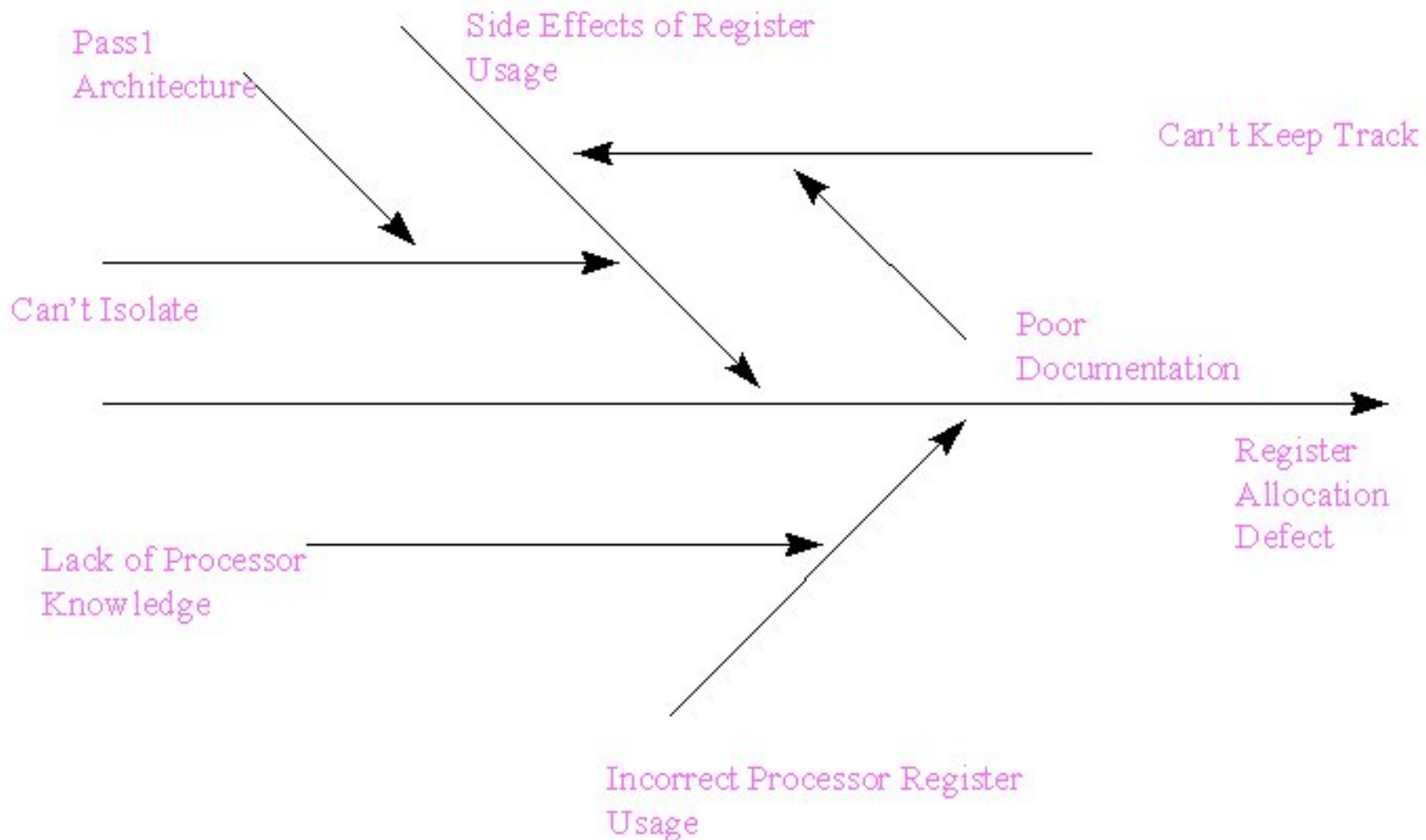


Figure 7. Cause/Effect Diagram for Register Allocation Defect [3].

7.1.5. Causal Analysis Meeting

A causal analysis meeting is held at the **EXIT** sub stage. At this stage it is ensured that all the required work is complete and the team goal is achieved. They are similar in many ways to inspections. The team has a leader and a recorder and includes all the involved programmers and some of their peers. It is a short meeting and it should not include managers. The causal analysis meeting should

follow an agenda.

After selecting the team leader and the team members, preliminary data on each defect is provided to the team members. They review this in advance to become familiar with each defect and to list questions to ask the implementers during the meeting. (As said above implementers should be present for the meeting as they are the best people who can provide some suggestions to correct or prevent the errors caused by them.)

The main objectives of the cause analysis meeting are as follows [4]:

- *To analyze defects.* The causal analysis meeting itself is a brain storming session to produce not only creative actions for individual task but also comprehensive actions for recurring groups of problems. During the sessions, all team members learn from the errors of other team members. In this way the entire team can create an action plan.
- *Evaluate results against team goals.* The results of tasks performed are compared with the goals set in the kickoff meeting. This comparison enables the team to understand their efficiency in achieving their target and triggers a discussion on how to improve future work.
- *Process Stage Evaluation.* After the completion of the causal analysis, the team should discuss general process improvements, such as how the inspections can be improved, what tools would be useful, what positive actions were taken during the stage that should be added to the process or kickoff package.

Answer to the following questions are collected for each defect.

- What stage originated the error?
- What is the error cause category?
- How was the error introduced?

The answers to the above questions are explained previously.

- *What caused the error?:* The purpose of this question is to identify the exact cause of the error before trying to take some actions. Causes vary from error to error. Causes such as inadequate schedule time, too many interruptions, or too little tools support often occur when least expected.
- *How to avoid the error?:* This question calls for a suggestion that would prevent the error from occurring in future. This may

include the necessary education, training that are to be provided for the developers.

- What corrective actions are recommended?: This question is to determine the specific action to be taken to correct or prevent the error. This question focuses on how to get the solution implemented. For example if an instruction class is recommended then the person who will take the responsibility of arranging the class and the person who will teach etc. are selected. Time and tools needed are arranged.

The result of the meeting is a report on each defect, and a report on the meeting itself. The report should include the basic data on the meeting such as preparation time, meeting time, defects handled, and the number of actions recommended.

The idea of these meetings is to provide immediate feedback as to the progress of the development so far.

After the completion of the meeting the leader enters all the actions to a database. The actions are linked to the defects at this time. The actions suggested during the process evaluation session of the meeting are also recorded. The team leader prepares a report of the actions and their associated defects for distribution to all the team members. The report includes all the details of the causal analysis and identifies the actions that have been turned over to the Action team. The action system should be available for every interested team member in order to determine the status of an action at any time.

Samples of actual defects and actions can be shown by the following table.

Table 1. Samples defects and corrective actions. [3]

| | |
|---------|---|
| Error: | WXTRN was coded when an EXTRN was needed. |
| Categ: | Education-base code/other. |
| Cause: | EXTRN statement not understood. |
| Action: | Add this item to the common error list. |
| Error: | Several program error conditions were overlooked. |
| Categ: | Oversight. |
| Cause: | Last-minute addition caused multiple errors. |

| | |
|---------|---|
| Action: | Add item to the common error list to warn people that the last minute changes are more prone to error. |
| Error: | Program functions were coded in incorrect order. |
| Categ: | Communication. |
| Cause: | Communications failed between high level and low level. |
| Action: | Change the process to include regularly scheduled communication sessions to be attended by the designers. |

A process stage with Kickoff meeting and Causal analysis can be represented by [Figure 8](#)

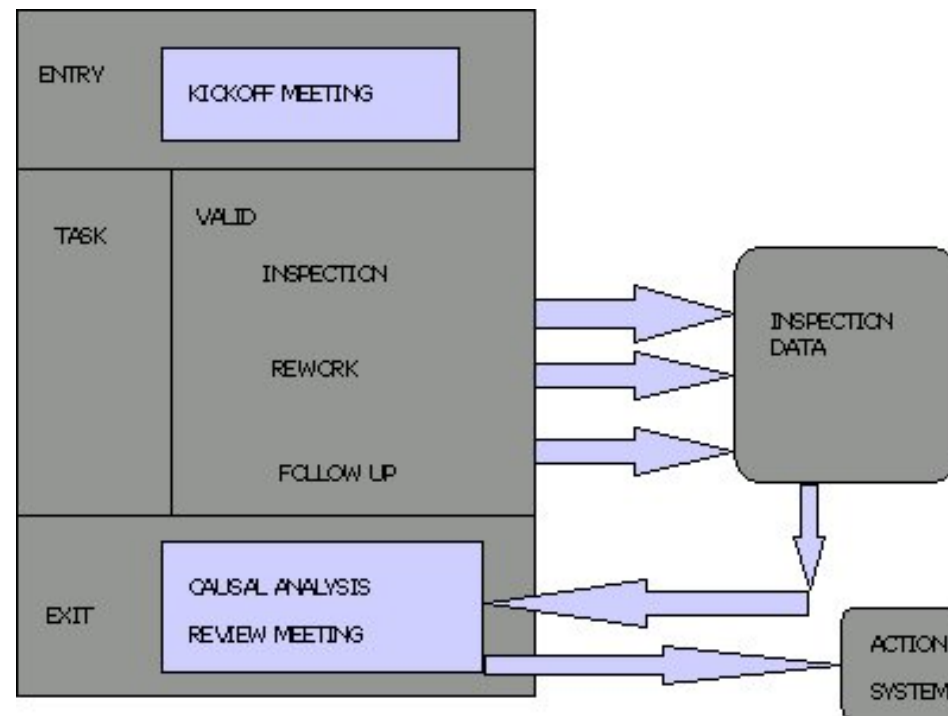


Figure 8. A process stage with kickoff meeting and causal analysis [\[4\]](#).

7.1.6. The Action Team

An action team is a set-up to ensure that the preventive and corrective actions are implemented and tracked. This team is not specific to one project and may be involved in every project that the organization carries out. The action team should include the manager, a process group representative, the education manager, a tools and methods specialist, a quality assurance group representative, configuration management. Since action team will be a long term assignment, team membership is rotated occasionally.

The major responsibilities of an action team are:

- Prioritizing all actions
- Establishing an implementation plan for the highest priority items.
- Assigning responsibilities.
- Tracking implementation.
- Reporting to manager on progress.
- Ensuring that all the success stories are recognized and the responsible individuals are identified.

7.1.7. Action Team Meeting

The action team meets regularly to review the defects and actions to prevent them which were discussed in the causal analysis meeting. A team member to whom the responsibility is assigned, brings all the records needed for the meeting. This member also updates the data base with new assignments. It starts with an initial review of the recommendations made to date and a decision on the priority of the items to be addressed. The important areas are handled first, actions are promptly launched to handle them.

Some of the preventive actions that are implemented are:

- *Process improvement, refinement, documentation.* These are actions to improve existing methods of development.
- *Tools.* Such tools could be code checkers which can scan sections of code to ensure correctness.
- *Education.* Improving knowledge of the people who are developing the system.
- *Product changes:* These are actions which make it less error prone for defects to occur when improving the product itself.

- *Communication.* It is important that any changes made to a system are documented and made aware of, as the change could have implications further down the line.

7.1.8. Action Progress Tracking

Every action must be logged and tracked and its status must be periodically reported. There should be follow-up reports, open item lists, priority lists. Once a prevention action has been approved, it should be instituted with all possible speed. Action records must be maintained for all actions, starting on the date they were originally suggested by the cause analysis teams. Actions are removed from this record only if they are implemented or superseded by other actions or by the action team's decision that they are no longer pertinent. Periodic status reports show the action item identification, its priority, the product involved, the date the item was submitted, the person responsible for implementation, the date assigned, and the plan completion date, cost, the expected quality impact, intermediate checkpoints, a summary of the action, and a record of the error types.

A typical action record is shown below:

Table 2. A typical Action Record. [3]

| | |
|-----------------|--|
| ACTION # | A unique number to identify the action. |
| PRODUCT | The product name or identifier. |
| PROGRAMMER | Programmer submitting the action. |
| CREATE DATE | Date action was introduced into the system. |
| PRIORITY | 1,2,3 or 4. |
| AREA CODE | Where the implementation occurred. |
| LINE ITEM | Specific item within the area. |
| CHECKPOINT INFO | Current status of the entry. |
| COST ESTIMATE | Number of programmer days expected for implementation. |
| TARGET DATE | Date of expected completion. |

| | |
|--|--|
| CLOSE DATE | Closing reason codes, date. |
| FINAL COST | Number of programmer days implementation actually took. |
| ABSTRACT OF ACTION | A short description of the action. |
| ASSOCIATED DEFECTS | List of all defects linked to this action. |
| ANSWER TEXT | A full description of the action that took place. |
| LOG OF ACTIVITIES AGAINST THE DATABASE | A track record of all activities check pointed against the action. |

Thus every action implementation is carefully tracked and updated records are maintained.

7.1.9. Prevention Feedback

After the root cause of the problem has been determined, this information should be disseminated through reports, presentations, or recorded minutes of causal analysis meetings. This feedback be utilized to perform the appropriate corrective actions. The feedback may be anything such as tools, dissertations, memos, management reports, defect records, newsletters, process documentation updates, classes, kickoff packages, error checklists. If possible an organization should build an organization wide feedback repository, so that feedback can be provided when ever it is necessary.

The entire software process with defect prevention technique can be summarized as in [Figure 9](#)

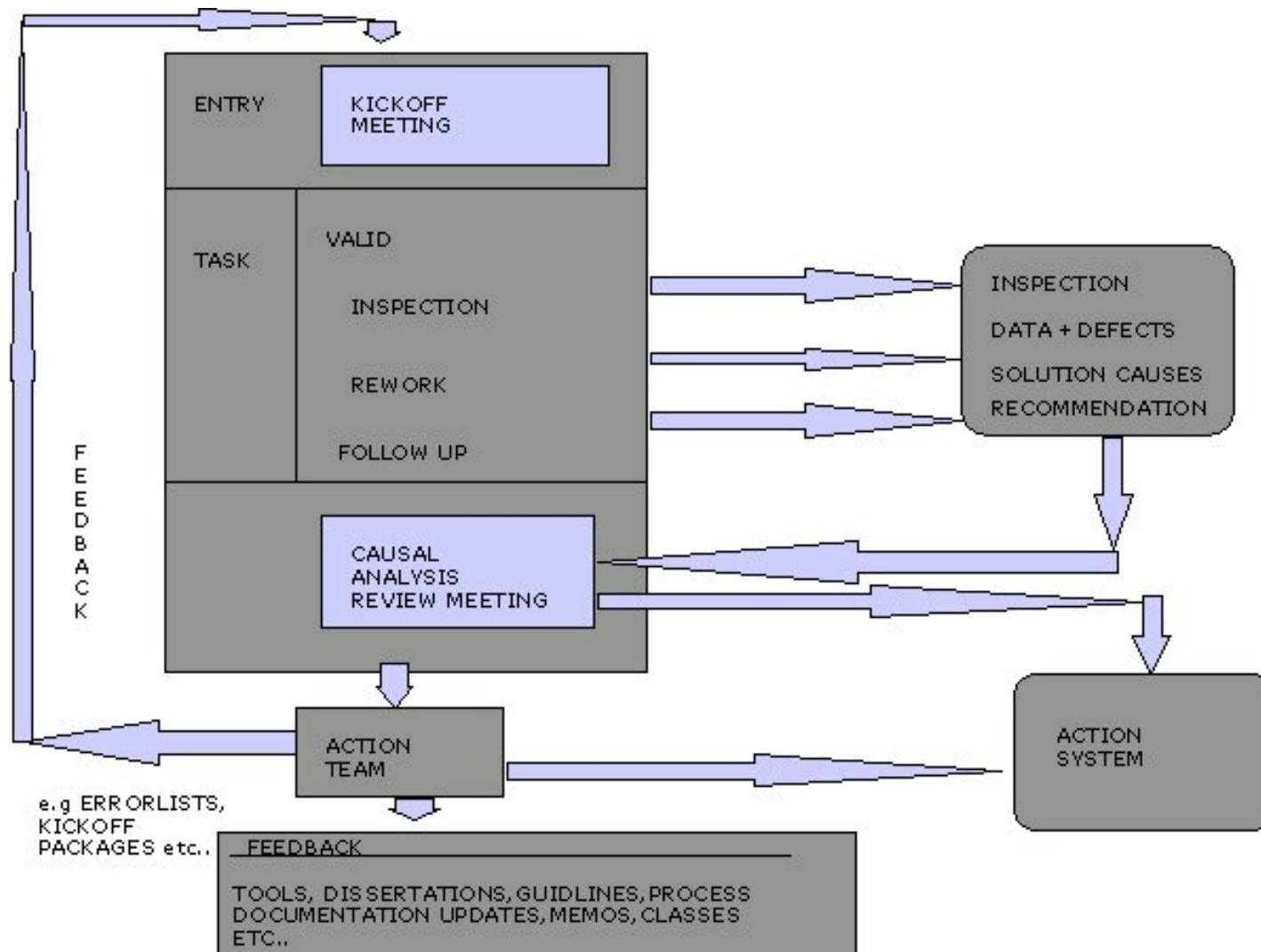


Figure 9. A Software Process With Defect Prevention Technique. [4]

7.1.10. Cost and Benefits of Defect Prevention

The major constraint that prevents an organization from implementing any defect prevention techniques is the cost of its implementation and maintenance. This is a reasonable concern. The major costs involved in defect prevention practice are the cost of conducting several meetings such as kickoff meeting, causal analysis meeting, action team meeting, the cost of gathering needed data, the cost of implementing a database system which is needed in tracking and reporting various process steps, the cost of new tools and the costs of conducting educational training and classes. While the costs are generally obvious the benefits are not.

But those meetings usually last for one to one and a half hours. So if we consider them as the part of the regular process then the cost will not be too much. It is very clear that in a normal software process a lot of time is spent on testing, finding bugs and fixing them. If we adopt defect prevention practice, the developer effort needed will be less due to reduced amount of errors needing to be fixed and test effort will be less. Since the cost of fixing bugs in a later stage of development would be ten to twenty times more than the cost of preventing them, it is wise to have a defect prevention practice. Also reducing the number of errors will save a lot of developer time. This extra time can be spent on process improvement.

Another benefit is that of improved documentation. Since documentation is a fundamental part of the defect prevention strategy, the errors due to communication problems and ambiguity can be reduced. Defect prevention process is a team work. It involves more and more employees deeply in their work and makes them understand the problems more clearly and precisely.

Improvements to the system are not simply effective for that one project, but are cumulative throughout further projects, resulting in a much more efficient and productive development organization. With reduced after release defects customer satisfaction can be greatly increased. This will improve the chance of that development organization receiving further work from their customers in the future.

7.1.11. Management's Role

Together with all the workers management should also play an active role in implementing the defect prevention technique. After deciding to move ahead with defect prevention the steps to be followed by the management are [\[3\]](#):

- Should meet the process team and make a joint commitment to the defect prevention.
- Should select a trial area and develop an introduction plan.
- Should select the action team and the appropriate team manager.
- Should provide the necessary tools and methods to conduct various process.
- Should make necessary arrangements to conduct various training and classes.

- Should track the process and take necessary actions when ever needed.
- After some experience, should modify the program to correct deficiencies.

Defect prevention takes time. The management should have the patience to wait until the benefits are apparent. Care should be taken to maintain a good relationship with all the workers in the organization.

7.2. Clean Room Software Approach

The Cleanroom approach for defect prevention was first used in the early 1980s at IBM's Federal System Division [7]. Cleanroom software approach combines successful techniques of precision manufacturing with the best practices of software engineering. Its goal is to produce almost error free codes before system testing begins. Cleanroom software engineering practices are grouped into 3 clusters Basic, Intermediate and Advanced practices. Projects that use the practices of the basic cluster are typically looking for short term improvements in business competitiveness. The advanced cluster adds specialized techniques that are needed to achieve high reliability in complex systems environments. The intermediate cluster of Cleanroom practice is intended for software based organizations. Intermediate Cleanroom practices address both quality improvement and development cost reduction through early defect prevention. The tasks involved in the Cleanroom practices are grouped under four sections *Management*, *Development*, *Review* and *Testing*. Technical reviews result in designs that are nearly error free before any testing begins. The separation of development from testing is a defining characteristic of Cleanroom.

7.2.1. Management Practices

- *Incremental Development*. It is the life cycle model most often associated with the Cleanroom. An increment is a functional subset of the eventual software product. It can be tested in an environment similar to that of the final software. By having many increments within a project, the result of one increment can be used to tailor the process for subsequent increments.
- *Team Ownership*. Cleanroom is team based approach. Every work product is owned by team that reviews its development, approves for completion and suggests any changes or corrections. Each team member is responsible for leading the creation of work product and analyzing it through reviews. In Cleanroom the teams are usually small ranging from from 3 to 7 people. Depending upon the size of the project the Cleanroom software practice would have different numbers of Project Teams. A small project would have a single Project Team. These Project Teams would review all work products that pertain to the project as a whole, such as the requirements, highest level specifications, and test plans. A sub team of perhaps 4 members would form the

Development Team. Their responsibility is to produce increments of software products that are ready for testing. A different subset of Project team would form the Test Team. A large project would have several sub teams in both development and test side. Each sub team then would report to the Development Team. The Development Team and Test Team together form the Project Team.

7.2.2. Development Practices

- *Black Box Specification.* Black Box behavior specifications emphasize an abstract view of the *behavior* of a software entity, hiding details of its design and implementation. Most specifications evolves from precise natural language toward more than rigorous notations. Even an advanced cleanroom uses natural language to support and explain its formal specifications. Black box specification describes a programs behavior as a mapping from input data into output data, without revealing the details of intermediate procedures. The objective of the Black box specification is to provide a complete definition of the externally visible behavior of the system, module or object. Conditional rule notations are used to express Black box specifications at advanced levels to eliminate the ambiguity from natural language specifications. It is programming like notation which is more formal at low level in the design hierarchy, with more natural language at higher level.
- *Clear Box Design.* Clear boxes are the framework for expressing algorithms. A clear box combines a control structure (Such as if-then-else or sequence) with a set of sub specifications for the software work product being developed. These sub specifications are subsequently represented as a clear box. This provides a hierarchy of specifications that helps in review and verification. This hierarchical structure gives the developers a way to maintain intellectual control over software development.
- *Abstract Data Models.* These are the models used in the Black box specification of an object that summarizes and retains data from one use to the next. This form of a conceptual data model, an abstraction of what is inside the system or object helps to distinguish the user's view of a system's behavior from the developer's view. The design data (concrete model) which is needed for the development is very much different from the abstract data model. But the later is much more meaningful to the user.
- *State Box Design.* The State box is represented by the choice of implementation in terms of concrete structures. A State box combines the concrete design data objects with the methods used to manipulate them. Both the data object and the methods may require further design specifications either using conditional rules or abstract models. A hierarchy of specifications, state boxes and clear boxes is thus formed which would guide the development and support the intellectual control.

7.2.3. Team Review

Cleanroom software team reviews are characterized by the following features:

- They are iterative
- Each review is conducted with the same team members.
- They focus on all quality aspects.

Suppose a team is developing a software component, the steps followed in a Cleanroom practice team review are:

- *Specification.*
 - Prototype reviews of initial specification sketches. The prototype reviews use informal discussion to examine the direction for that work product.
 - Development reviews of evolving specifications. These reviews provide closer scrutiny to completeness and validity issues and design quality issues.
 - Completion reviews of final specification. Since the reviewers would have seen the specification and design many times before, at the completion stage it is easy to determine its suitability for approval.
- *Design.* In addition to the prototype, development and completion reviews, Clear box and State box verification are performed.
 - Clear Box verification. After planning and implementing, the design needs to be verified that it meets the specified behavior. This is done by the reviewing team and the result is combined with design quality evaluation.
 - State Box verification. This verification evaluates the correctness of the state box. Here the mapping between the abstract data model and its concrete realization are focused.

The review process is iterative and if there is any problem or if anything is not clear, it is possible to refer back to the earlier steps any time. The reviews are not limited to exposing defects, but to verify that the software is valid and correct with respect to specification and has the appropriate design qualities. The review takes place throughout the development process and will be conducted for all development work products.

7.2.4. Testing Practices

The primary purpose of Cleanroom testing is to measure the software quality. Behavioral testing based on expected usage is a key aspect of the Cleanroom practice.

- *Usage Modeling.* Here a model of system's expected usage is developed and the test cases are selected or generated based on that model. The usage model assigns a probability to each possible use of the system. Once usage model is constructed, test cases can

be created, acquired, or generated that reflect estimates in that model. Usage model can be constructed using a number of techniques, including instrumenting an existing system or similar system, conducting interviews and surveys, or by best engineering judgment. Because the software is tested the same way it will be used, the most likely failure will tend to arise earliest and most often in testing. The user visible quality of the work product can be seen earlier in the testing. The usage modeling provides feedback both to the specifications and to the designers.

- *Statistical Testing*. Statistical testing and usage modeling go hand-in-hand. The usage model defines a probability distribution for all sequences of inputs. In statistical testing distributions are sampled, tests are conducted and field reliability is estimated based on the results. This testing is a tool that enables reliability estimation. But this testing is not efficient in finding bugs.

Thus by applying Cleanroom Practices in an organization a systematic quality and productivity improvement can be achieved. But there are many disadvantages in using this practice since a complete and stable requirement specification is needed for the development. The box structures and verification processes are fully dependent on the clear and stable requirement specifications. But in reality the requirement specifications are not fully known and they are often unstable. In addition to that the Cleanroom practice has a long development cycle.

7.3. The Personal Software Process (PSP)

The personal software process provides a framework for training software engineers about the software process. It is a defined and measured, individualized process for consistently and efficiently developing high quality software modules or small programs [10]. The PSP courses incorporate what has been called a *Self Convincing Learning Strategy* that uses data from the engineer's own performance to improve learning and motivate use. The course introduces the PSP practices in steps corresponding to seven PSP process levels. Each level builds on the capabilities developed and historical data collected in the previous level. There is a level called **Personal Quality Management** (i.e Level 2) in which each individual can learn to detect errors in an early stage of development and to prevent them. As engineers progress through the PSP training, the number of faults injected and therefore removed per certain lines of code decreases. After PSP training, when measured in customer-found defects, the quality of components developed by an engineer was five times better than the quality of components developed before the training. Thus by properly training individual programmer or employee the personal performance can be improved which in turn improves the overall quality and productivity.

7.4. Prototyping

In his article *A Personal Commitment to Software Quality*, Watts Humphrey suggests that *While there are many reasons for developing prototypes, they all stem from a desire to avoid making mistakes*. A prototype which is effectively a test model is constructed to obtain a better understanding of the problem under development. It helps users to confirm that the requirements have been fully understood by both parties and the finished product is going to be functionally correct. Even though prototyping is an expensive way of developing the system, a lot of time and cost spent on reviews and on fixing the bugs can be saved by prototyping. Thus we can consider prototyping as a defect preventing technique.

7.5. Formal Specification

This technique uses mathematical symbols and equations to provide complete and unambiguous specifications for parts of a system. It can help to identify the areas which are not clear-cut and reduce the risk of defects injection into the system. Such defects can be categorized under Education or Communication cause of defects.

The major drawback of this process is that it is difficult to learn this technique. But once mastered it can prove a very exact way of defining functions and thereby eliminating the errors.

12. Conclusion

Due to its thoroughness and investment in both money and time, many organizations could step back from using the defect prevention process. But reduced number of after release errors will result in increased user satisfaction, a benefit which is difficult to place a value upon. Improvements to the system are not only effective for that particular project, but also cumulative throughout further projects, resulting in a much more efficient and productive organization. When used together with models such as Capability Maturity Model the defect prevention technique will help to build a product which is of a higher quality and standard, a principle which is highly important in today's critical world. Several methods are be used to prevent defects. But irrespective of the methodology used, there must be an aggressive effort to learn from the available data. Without analyzing the statistics and defects, the available information can not be properly utilized for fine-tuning the development process. By implementing process such as Personal Software Process, the individual working efficiency of each worker can be improved which helps to improve the overall development cycle.

So I conclude that regardless of its implementation cost and development time each and every organization must adopt one or other practices to prevent the defects that are injected into the work products during the development life cycle. In addition to the above said methods with automation and reuse of certain existing components, it is possible to prevent injection of defects to some extent.

13. References

1. *Introduction to the Personal Software Process*. Watts S Humphrey.
2. <http://www.yacc.co.uk/~rej/defects.html>
3. *Managing the Software Process*. Watts S.Humphrey.
4. *A process-integrated approach to defect prevention*. Jones , C.L. *IBM Systems Journal*, Vol.24, no.2,1985.
5. *Software Quality Assurance for Maintenance*. James S.Collofello,Jeffery J.Buck. *IEEE Software*, September 1987.
6. *Learning from our Mistakes with Defect Causal Analysis*. David.N.card, *IEEE Software*,September 1998.
7. *Cleanroom Practice: A Theme and Variations*. Michael Deck, cleanroom Software Engineering, Inc.
8. *Novice Mistakes: Are The Folk Wisdoms Correct?*. James C.Spohrer and Elloit Soloway, *Communications of the ACM*, Vol.29, no.7,1986.
9. <http://www.sei.cmu.edu/technology/cmm/obtain.cmm.html>
10. <http://www.sei.cmu.edu/products/publications/97.reports/97tr001/97tr001abstarct.html>
11. <http://deming.eng.clemson.edu/pub/tutorials/qctools/homepg.htm>

[Back to table of content](#)

[Presentation slides](#)

[Back to main page](#)

Software Defect Prevention

2/19/98

[Click here to start](#)

[Goto Presentation Document](#)

[Back to main page](#)

Table of Contents

Author: Chethana Kuloor

Email: kuloor@enel.ucalgary.ca

[Software Defect Prevention](#)

[Overview](#)

[Cont'd](#)

[Defects](#)

[Defect Detection](#)

[Defect Prevention](#)

[Why it is necessary ?](#)

[Principles of Defect Prevention](#)

[Defect Prevention Techniques](#)

[A process integrated approach](#)

[A process integrated approach
\(Cont'd\)](#)

[PPT Slide](#)

[The Kickoff Meeting](#)

[Causal Analysis](#)

[PPT Slide](#)

[PPT Slide](#)

[PPT Slide](#)

[Defect Types](#)

[PPT Slide](#)

[Causal analysis charts](#)

[PPT Slide](#)

[PPT Slide](#)

[PPT Slide](#)

[Action Plan Development and
Implementation](#)

[PPT Slide](#)

[PPT Slide](#)

[Other Methods](#)

[Cleanroom software approach](#)

[Cleanroom software approach](#)

[Cleanroom Software Approach](#)

[Cleanroom Practice Overview](#)

[Cleanroom software approach 1.
Management practices:](#)

[Cleanroom software approach 2.
development practices](#)

[Cleanroom software approach 2.
development practices\(continued\)](#)

[Cleanroom software approach 2.
development practices\(continued\)](#)

[Cleanroom software approach 2.
development practices\(continued\)](#)

[Cleanroom software approach 2.
development practices\(continued\)](#)

[Cleanroom software approach 2.
development practices\(continued\)](#)

[Cleanroom software approach
3.Team Review\(continued\)](#)

[Cleanroom software approach
3.Testing practice](#)

[Cleanroom software approach](#)

[3. Testing practice\(continued\)](#)

[Cleanroom software approach](#)

[3. Testing practice\(continued\)](#)

[Cleanroom software approach](#)

[3. Testing practice\(continued\)](#)

[Cleanroom software approach:](#)

[Summary](#)

[Conclusion](#)

[Questions and comments](#)

Software Defect Prevention

Chethana Kuloor
and
Mike.M.Hong



Slide 1 of 46

Overview

- **Introduction**
- **What are defects ?**
- **Defect detection and Defect prevention**
- **Why Defect prevention is necessary ?**
- **Principles of defect prevention**



Slide 2 of 46

Cont'd

• **Methods of defect prevention**

- A process integrated approach to defect prevention
- Cleanroom software approach
- Other methods
 - Prototyping
 - Formal specification
 - Personal software approach

• **Conclusion**



Slide 3 of 46

Defects

- **What are defects?**
- **Defects cause low-quality/delay**



Slide 4 of 46

Defect Detection

- **Process of finding errors and fixing them**
- **Can be detected during inspection, reviews, testing, and even after customer release**



Slide 5 of 46

Defect Prevention

“Process of improving the quality and productivity by preventing the injection of errors into the software product”



Slide 6 of 46

Why it is necessary ?

- ⦿ **Prevention is better than cure**
- ⦿ **Reduces development time and cost**
- ⦿ **Inspection and Testing can be more effective**
- ⦿ **Increases Customer satisfaction**



Slide 7 of 46

Principles of Defect Prevention

- ⊗ Programmers should evaluate their own errors
- ⊗ Causal Analysis Should be a part of the process
- ⊗ Feedback should be a part of the process
- ⊗ Should be implemented in incremental steps
- ⊗ Process Improvement must be an integral part of the software process



Slide 8 of 46

Defect Prevention Techniques

- **A process integrated approach**
- **Cleanroom software approach**
- **Other methods**
 - **The personal software process**
 - **Prototyping**
 - **Formal Specification**



Slide 9 of 46

A process integrated approach

- The Kickoff Meeting
- Causal Analysis and defect reporting
- The Action Plan Development
- Action Implementation
- Performance Tracking



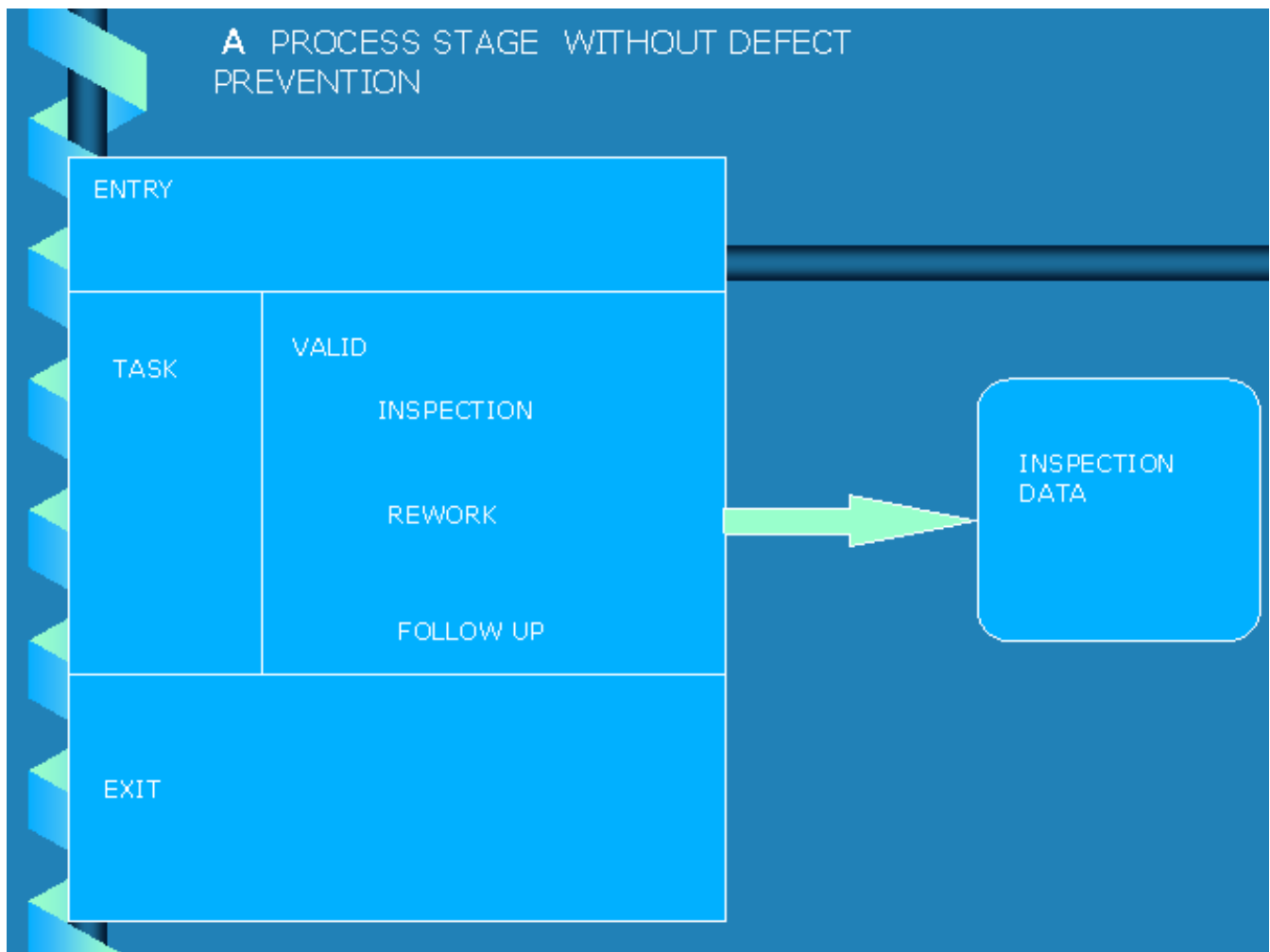
Slide 10 of 46

A process integrated approach (Cont'd)

- Prevention Feed back
- Defect prevention in testing
- Starting Over
- Cost and benefits
- Management's role



Slide 11 of 46



Slide 12 of 46

The Kickoff Meeting

- **Review available input**
- **Review process**
- **Review error checklists**
- **Set team goal**



Slide 13 of 46

Causal Analysis

- **Causal analysis team**
- **General defect cause categories**
- **Examples of causal analysis**
- **Causal analysis charts and diagrams**
- **Causal analysis meeting**



Slide 14 of 46

How Defects are caused?

- **Defect cause categories**

- Communication
- Education
 - New Function
 - Old Function
 - Other
- Oversight
- Transcription



Slide 15 of 46

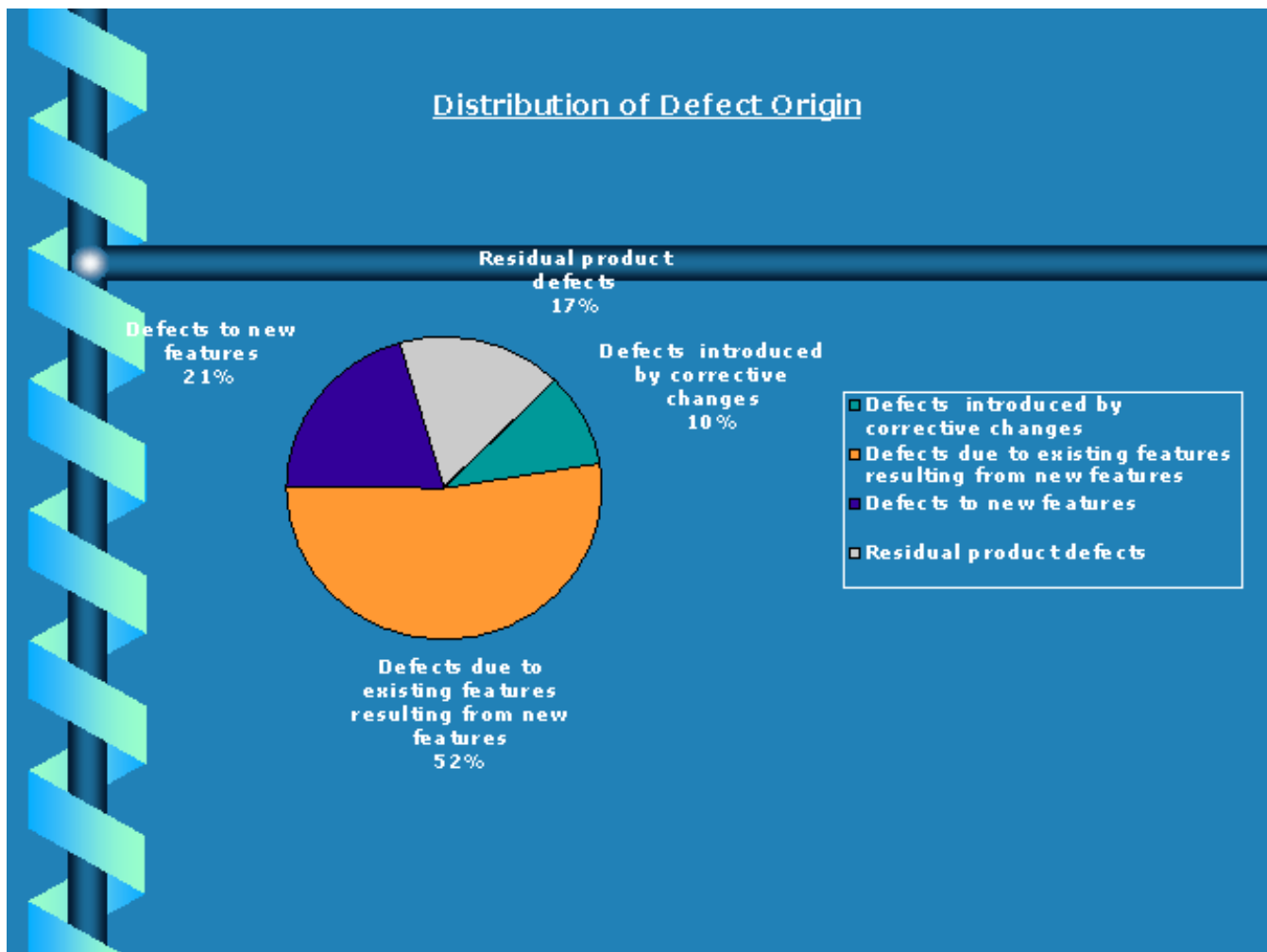
Example

Defects found :

- In a new feature when it is added to the existing feature.
- In an existing feature when a new feature is added to it.
- In an existing feature when corrections are applied to the defects in it.
- In an existing feature due to residual errors.



Slide 16 of 46



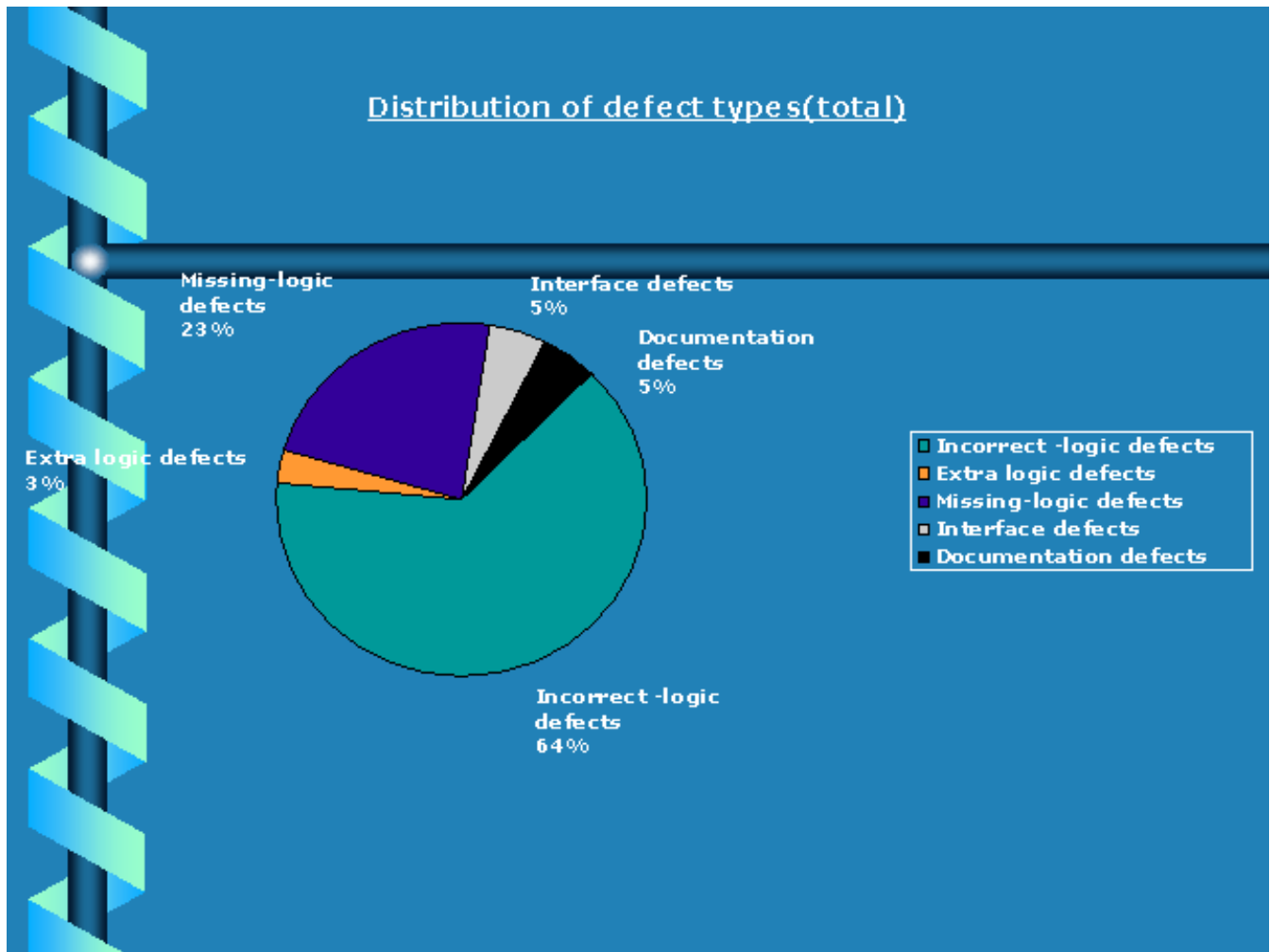
Slide 17 of 46

Defect Types

- The major defect types found are:
 - **Incorrect-logic defects**
 - **Extra logic defects**
 - **Missing logic defects**
 - **Documentation defects**
 - **Interface defects**



Slide 18 of 46



Slide 19 of 46

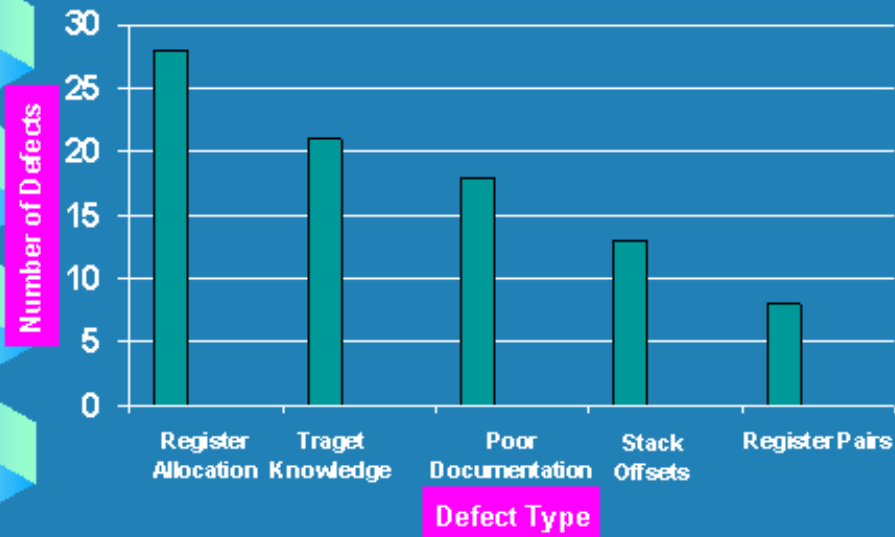
Causal analysis charts

- Pareto Charts
- Example
- Cause/Effect Diagrams
- Example



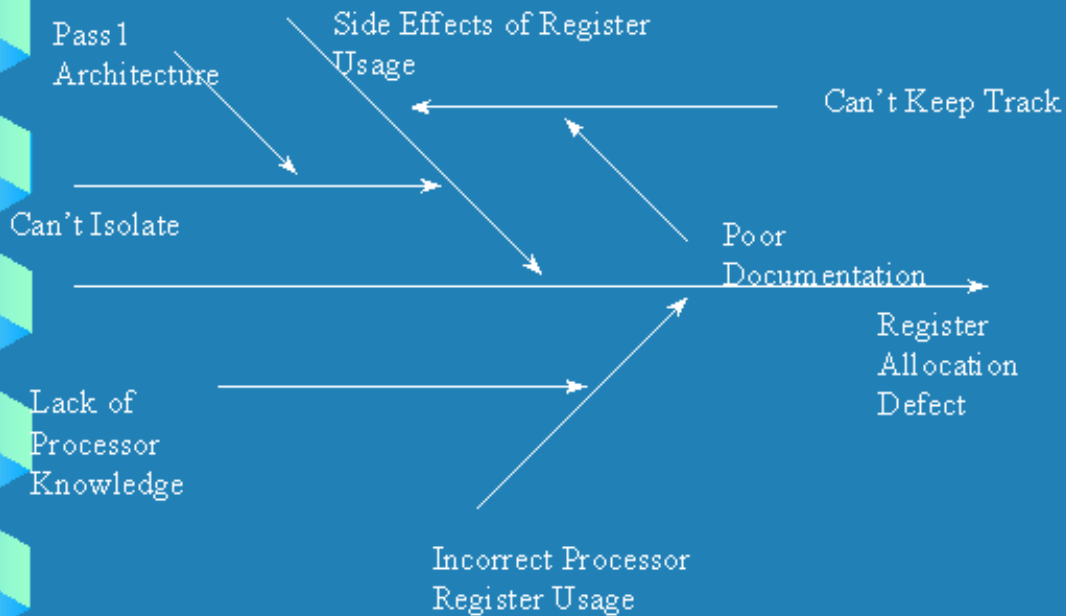
Slide 20 of 46

Pareto Chart For Compiler Design Defects



Slide 21 of 46

Cause/Effect Diagram



Slide 22 of 46

Causal Analysis Meeting

- **Objectives of causal analysis meeting**

- To analyze defects
- Evaluate results against team goals
- Process stage evaluation



Slide 23 of 46

Action Plan Development and Implementation

- Action Team
- Action team Meeting
 - **Prioritize all actions**
 - **Establish Implementation plan for highest priority item**
 - **Assign Responsibilities**
 - **Track all actions**
 - **Report to manager**



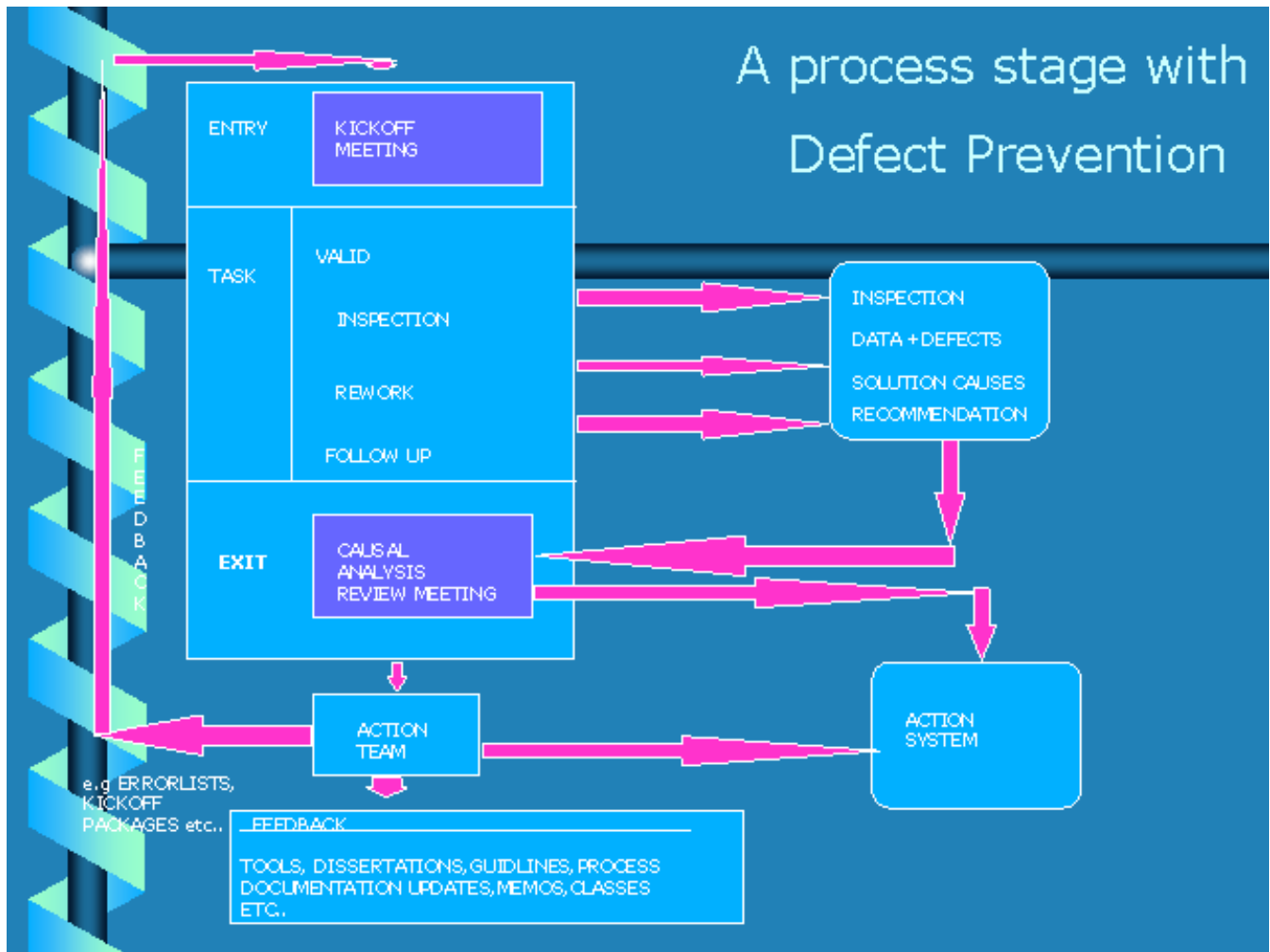
Slide 24 of 46

Table 1. A typical Action Record.

| | |
|--|--|
| ACTION # | A unique number to identify the action. |
| PRODUCT | The product name or identifier. |
| PROGRAMMER | Programmer submitting the action. |
| CREATE DATE | Date action was introduced into the system. |
| PRIORITY | 1,2,3 or 4. |
| AREA CODE | Where the implementation occurred. |
| LINE ITEM | Specific item within the area. |
| CHECKPOINT INFO | Current status of the entry. |
| COST ESTIMATE | Number of programmer days expected for implementation. |
| TARGET DATE | Date of expected completion. |
| CLOSE DATE | Closing reason codes, date. |
| FINAL COST | Number of programmer days implementation actually took. |
| ABSTRACT OF ACTION | A short description of the action. |
| ASSOCIATED DEFECTS | List of all defects linked to this action. |
| ANSWER TEXT | A full description of the action that took place. |
| LOG OF ACTIVITIES AGAINST THE DATABASE. | A track record of all activities check pointed against the action. |



Slide 25 of 46



Other Methods

- **The personal software process**
- **Prototyping**
- **Formal specification**



Slide 27 of 46

Cleanroom software approach

- Introduction:
 - The Cleanroom approach was first used in the early 1980s at IBM's Federal System Division.
 - The approach takes its name from the clean rooms used in precision manufacturing, where statistical quality control techniques emphasize defect prevention over defect removal.



Slide 28 of 46

Cleanroom software approach

- Three main areas:
 - Management
 - Development
 - Review and Testing



Slide 29 of 46

Cleanroom Software Approach

– Three basic practices:

– The basic practices

» typically looking for short term improvements in business competitiveness.

– The intermediate practices

» address both quality improvement and development cost reduction through early defect prevention.

– The advanced practices

» adds specialized techniques that are needed to achieve high reliability in complex systems environments.



Slide 30 of 46

Cleanroom Practice Overview

| Basic | Intermediate | Advanced |
|-------------------------|--|--|
| Incremental Development | | |
| Team Ownership | | |
| Black Box Specification | Conditional Rule Process Specification | Formal Specification |
| | Abstract Data Models | Stimulus History Models |
| | | Formal Models of Real-Time and Concurrency |
| Clear Box Design | State Box Designs | |
| Team Review | Clear Box Verification | Correctness Proof |
| | State Box Verification | |
| Behavioral Testing | Usage Modeling | Advanced Usage Model |
| | Statistical Testing | Product Warranties |



Slide 31 of 46

Cleanroom software approach

1. Management practices:

- Two main features:
 - Incremental development
 - Team Ownership



Slide 32 of 46

Cleanroom software approach

2. development practices

- Software development process
 - is based on box structures.
- Box structures:
 - Black box;
 - State box;
 - Clear box



Slide 33 of 46

Cleanroom software approach 2. development practices(continued)

- Black box:
 - is used in specification stage
- State box and Clear box:
 - are used in design stage



Slide 34 of 46

Cleanroom software approach 2. development practices(continued)

- The black box design :
 - it describes the external behavior of the program,
 - It hides data and process implementations.
 - It maps all the input data to the output data with no detail explanation of how it is being done.
 - it provides a common ground between the developers and the customers.



Slide 35 of 46

Cleanroom software approach 2. development practices(continued)

- State Box Design:
 - The state box is evolved from the black box data specification.
 - It exposes the data implementation, but hides the process implementation.
 - The state box expresses the encapsulates data in a more concrete form.



Slide 36 of 46

Cleanroom software approach 2. development practices(continued)

- State Box Design (continued)
 - It looks at the mapping between abstract data model and its realization.
 - A state box consists of state data and the specification for each process operation.



Slide 37 of 46

Cleanroom software approach 2. development practices(continued)

- Clear box design
 - It is evolved from black box process specification or state box design.
 - A clear box design exposes both data and process implementation.



Slide 38 of 46

Cleanroom software approach

3.Team Review(continued)

- Team reviews features:
 - They are iterative, and if anything is not clear, it is possible to refer back to the earlier steps any time.
 - The reviews are not limited to exposing defects, but to verify that the software is valid and correct
 - The review takes place throughout the development process



Slide 39 of 46

Cleanroom software approach

3. Testing practice

- The primary purpose:
 - to measure the software quality.
 - expected usage testing is a key aspect
 - the strategy and tactics of cleanroom testing are fundamentally different from conventional testing approaches.



Slide 40 of 46

Cleanroom software approach

3. Testing practice(continued)

- Testing practice feature

- Conventional methods derive a set of test cases to uncover design and coding errors.
- The goal of cleanroom testing is to validate software requirements .



Slide 41 of 46

Cleanroom software approach

3. Testing practice(continued)

- Testing in Cleanroom approach:
 - Purpose:
 - To measure the quality of software.
 - Not to find bugs and try to fix it, since most bugs have been found in the review meetings
 - Method:
 - Statistical test method is used



Slide 42 of 46

Cleanroom software approach

3. Testing practice(continued)

- The statistical testing
 - It does not require the detail internal software knowledge. It emphasizes on the user-visible part of the software, and is being used to determine the likelihood of the failures.



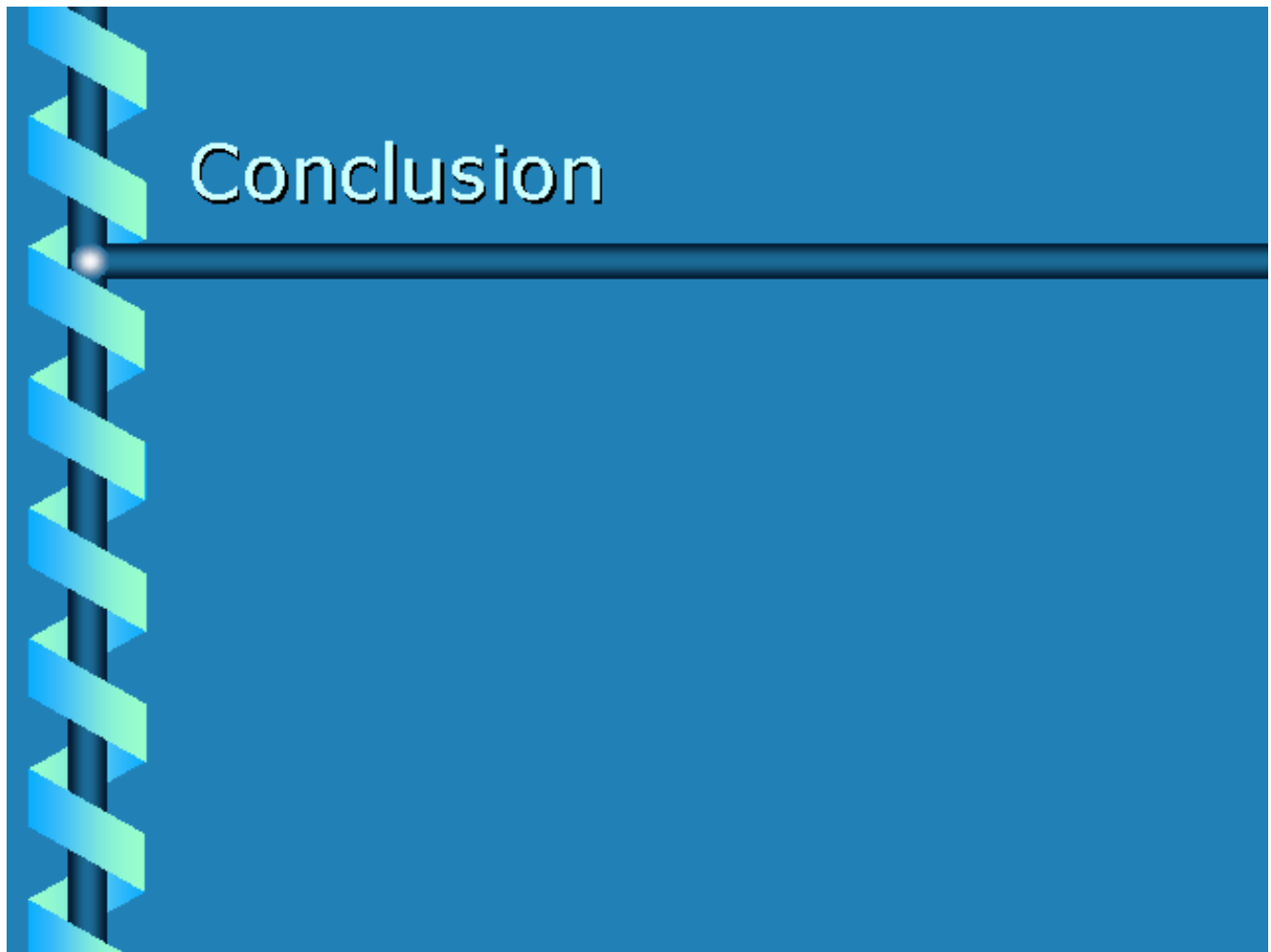
Slide 43 of 46

Cleanroom software approach: Summary

- Cleanroom software practice is one of the approaches for software defect prevention
- Its goal:
 - To improve software process quality.
 - To realize the defect prevention(error_free)



Slide 44 of 46



Slide 45 of 46

Questions and comments

- Welcome
 - Questions?
 - Comments?



Slide 46 of 46