Software Reliability

Definition 1: The reliability of a software product essentially denotes its trustworthiness or dependability.

Definition 2: The reliability of a software product can also be defined as the probability of the working "correctly" over a given period of time.

- The 10 per cent instructions are often called the core1 of a program. The rest 90 per cent of the program statements are called non-core working "correctly" over a given period of time.
- We can say that reliability of a product **depends not** only on the number of **latent errors** but also on the exact **location** of the errors.

A. Hardware versus Software Reliability:

- Hardware components fail due to very different reasons as compared to software components. Hardware components fail mostly due to wear and tear, whereas software components fail due to bugs.
- Example: A logic gate may be stuck at 1 or 0, or a resistor might short circuit. To fix a hardware fault, one has to either replace or repair the failed part. In contrast, a software product would continue to fail until the error is tracked down and either the design or the code is changed to fix the bug.
- For this reason, when a hardware part is repaired its reliability would be maintained at the level that existed before the failure occurred; whereas when a software failure is repaired, the reliability may either increase or decrease (reliability may decrease if a bug fix introduces new errors).
- A comparison of the changes in failure rate over the product life time for a typical hardware product as well as a software product is sketched in Figure 11.1. Observe that the plot of change of reliability with time for a hardware component (Figure 11.1(a)) appears like a "**bath tub**".
- For a software component the failure rate is initially high, but decreases as the faulty components identified are either repaired or replaced.

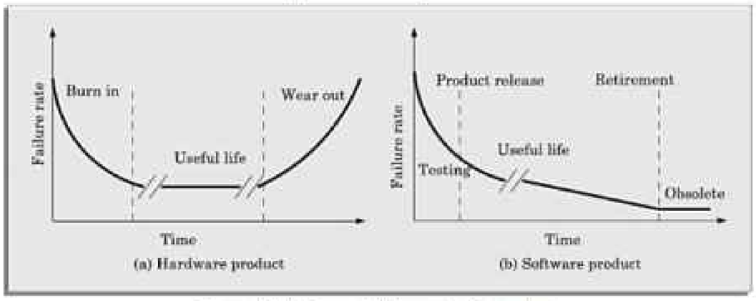


Figure 11.1: Change in failure rate of a product.

B. Reliability Metrics of Software Products

1. Rate of occurrence of failure (ROCOF): ROCOF measures the frequency of occurrence of failures.

2. Mean time to failure (MTTF): MTTF is the time between two successive failures, averaged over a large number of failures. Let the failures occur at the time instants t1, t2, ..., tn. Then, MTTF can be calculated as

$$\sum_{i=1}^{n} \frac{t_{i+1}-t_i}{(n-1)}.$$

3. Mean time to repair (MTTR): Once failure occurs, some time is required to fix the error.
4. Mean time between failures (MTBF): The MTTF and MTTR metrics can be combined to get the MTBF metric:

MTBF=MTTF+MTTR.

5. Probability of failure on demand (POFOD): POFOD measures the likelihood of the system failing when a service request is made.

6. Availability: Availability of a system is a measure of how likely would the system be available for use over a given period of time.

Reliability Growth Modelling

A reliability growth model can be used to predict when (or if at all) a particular level of reliability is likely to be attained. Thus, reliability growth modelling can be used to determine when to stop testing to attain a given reliability level.

Ex: The **Jelinski-Moranda** (J-M) model is one of the earliest software reliability models. The reliability increases by a constant increment each time an error is detected and repaired.

The program failure rate at the ith failure interval is given by

$$\lambda(t_i) = \phi[N-(i-1)], \quad i = 1, 2, ..., N$$

Where
ϕ = a proportional constant, the contribution any one fault makes to the overall program
N = the number of initial faults in the program
t_i = the time between the (i-1)th and the (i)th failures.

Ex: **Littlewood and Verall's model :** This model allows for negative reliability growth to reflect the fact that when a repair is carried out, it may introduce additional errors.

STATISTICAL TESTING

Statistical testing is a testing process whose objective is to determine the **reliability** of the product rather than discovering errors.

Steps in Statistical Testing:

1) The first step is to determine the **operation profile** of the software.
2) The next step is to generate a set of **test data** corresponding to the determined operation profile.
3) The third step is to apply **the test cases** to the software and record the time between each failure.
4) After a statistically significant **number of failures** have been observed, the reliability can be computed.

SOFTWARE QUALITY METRICS

Software Quality: Traditionally, the quality of a product is defined in terms of its **fitness of purpose**. That is, a good quality product does exactly what the users want it to do.

The modern view of a quality associates with a software product several quality factors (or attributes) such as the following:

1) **Portability**: if it can be easily made to work in **different hardware** and **operating system** environments.
2) **Usability**: if different categories of users (i.e., both expert and novice users) can easily invoke the functions of the product.
3) **Reusability**: if different modules of the product can easily be reused to develop new products.
4) **Correctness:** if different requirements as specified in the SRS document.
5) **Maintainability**: if errors can be easily corrected, new functions can be easily added to the product, and the functionalities of the product can be easily modified, etc.

SOFTWARE QUALITY MANAGEMENT SYSTEM

- **Quality Management System:** (often referred to as quality system) is the principal methodology used by organizations to ensure that the products they develop have the desired quality.
- **Quality system activities:**
  The quality system activities encompass the following:
  - ✓ Auditing of projects to check if the processes are being followed.
  - ✓ Collect process and product metrics and analyse them to check if quality goals are being met.
  - ✓ Review of the quality system to make it more effective.
  - ✓ Development of standards, procedures, and guidelines.
  - ✓ Produce reports for the top management summarising the effectiveness of the quality system in the organisation.
- **Evolution of Quality Systems**
  - **Quality control (QC)** focuses not only on detecting the defective products and eliminating them, but also on determining the causes behind the defects, so that the product rejection rate can be reduced.
  - Thus, quality control aims at correcting the causes of errors and not just rejecting the defective products. The next breakthrough in quality systems was the development of the **quality assurance** (**QA**) principles.

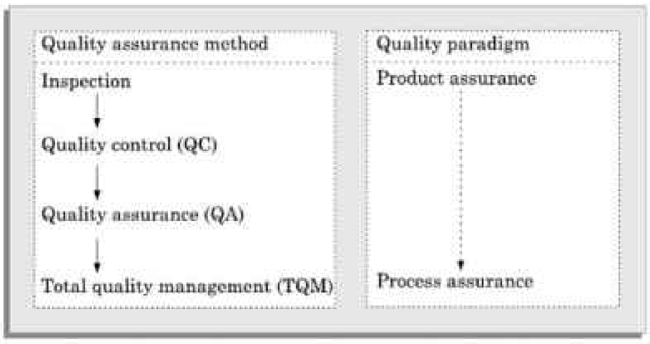


Figure 11.3: Evolution of quality system and corresponding shift in the quality paradigm.

- Total quality management (**TQM**) goes a step further than quality assurance and aims at continuous process improvement.
- **Product Metrics versus Process Metrics**: Product metrics help measure the characteristics of a product being developed, whereas process metrics help measure how a process is performing.

ISO 9000

- International standards organisation (ISO) is a consortium of 63 countries established to formulate and foster standardisation. ISO published its 9000 series of standards in 1987.
- ISO 9000 standard specifies the guidelines for maintaining a quality system. The ISO 9000 series of standards are based on the premise that if a proper process is followed for production, then good quality products are bound to follow automatically.
- ISO 9000 certification serves as a reference for contract between independent parties
- ISO 9000 is a series of three standards—ISO 9001, ISO 9002, and ISO9003.

ISO 9001: This standard applies to the organisations engaged in design, development, production, and servicing of goods. This is the standard that is applicable to most software development organisations.

ISO 9002: This standard applies to those organisations which do not design products but are only involved in **production**. Examples of this category of industries include steel and car manufacturing industries who buy the product and plant designs from external sources and are involved in only manufacturing those products. Therefore, ISO 9002 is **not applicable** to software development organisations.

ISO 9003: This standard applies to organisations involved only in installation and testing of products.

Why Get ISO 9000 Certification?

There is a mad scramble among software development organisations for obtaining ISO certification due to the benefits it offers. Let us examine some of the benefits that accrue to organisations obtaining ISO certification:

1. Confidence of customers in an organisation increases when the organisation qualifies for ISO 9001 certification.
2. ISO 9000 requires a well-documented software production process to be in place.
3. ISO 9000 makes the development process focused, efficient, and costeffective.
4. ISO 9000 certification points out the weak points of an organisation and recommends remedial action.
5. ISO 9000 sets the basic framework for the development of an optimal process and TQM.

Salient Features of ISO 9001 Requirements

- **Document control**: All documents concerned with the development of a software product should be properly managed, authorised, and controlled. This requires a configuration management system to be in place.
- **Planning**: Proper plans should be prepared and then progress against these plans should be monitored.
- **Review**: Important documents across all phases should be independently checked and reviewed for effectiveness and correctness.

- **Testing**: The product should be tested against specification.
- **Organisational aspects**: Several organisational aspects should be addressed e.g., management reporting of the quality team.

Shortcomings of ISO 9000 Certification:

1. ISO 9000 requires a software production process to be adhered to, but does not guarantee the process to be of high quality.
2. ISO 9000 certification process is not fool-proof and no international accreditation agency exists. Therefore it is likely that variations in the norms of awarding certificates can exist among the different accreditation agencies and also among the registrars.
3. Organisations getting ISO 9000 certification often tend to downplay domain expertise and the ingenuity of the developers. These organisations start to believe that since a good process is in place, the development results are truly person-independent.
4. In other words, software development is a creative process and individual skills and experience are important.
5. ISO 9000 does not automatically lead to continuous process improvement. In other words, it does not automatically lead to TQM.

ISO 9000-2000

ISO revised the quality standards in the year 2000 to fine tune the standards. The major changes include a mechanism for continuous process improvement. There is also an increased emphasis on the role of the top management, including establishing measurable objectives for various roles and levels of the organisation. The new standard recognizes that there can be many processes in an organisation.

SEI CAPABILITY MATURITY MODEL

- ✓ SEI CMM is suited for large organisations.
- ✓ SEI **C**apability **M**aturity **M**odel (SEI CMM) was proposed by Software Engineering Institute of the Carnegie Mellon University, USA. In simple words, CMM is a reference model for apprising the software process maturity into different levels.
- ✓ This can be used to predict the most likely outcome to be expected from the next project that the organisation undertakes.
- ✓ The different levels of SEI CMM have been designed so that it is easy for an organisation to slowly build its quality system starting from scratch.
- ✓ Capability maturity model integration (**CMMI)** is the successor of the CMM.
- ✓ SEI CMM classifies software development industries into the following five maturity levels:

Level 1: Initial:

1. A software development organisation at this level is characterised by adhoc activities.
2. Very few or no processes are defined and followed.
3. Since software production processes are not defined, different engineers follow their own process and as a result development efforts become chaotic.
4. Therefore, it is also called **chaotic level**.
5. The success of projects depends on individual efforts and heroics.
6. When a developer leaves the organisation, the successor would have great difficulty in understanding the process that was followed and the work completed.

Level 2: Repeatable

1. At this level, the basic project management practices such as tracking cost and schedule are established.
2. Configuration management tools are used on items identified for configuration control.
3. Size and cost estimation techniques such as function point analysis, COCOMO, etc.,are used.
4. The necessary process discipline is in place to repeat earlier success on projects with **similar** applications.
5. Though there is a rough understanding among the developers about the process being followed, the process is not documented.
6. For this reason, the successful development of one product by such an organisation does not automatically imply that the next product development will be successful.

Level 3: Defined

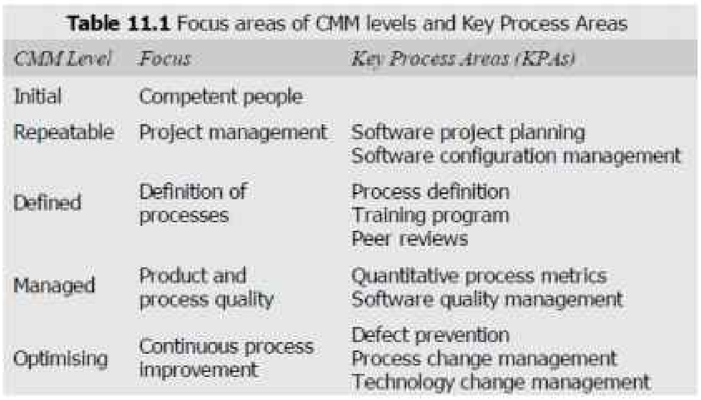


Table 11.1 Focus areas of CMM levels and Key Process Areas

CMM Level	*Focus*	*Key Process Areas (KPAs)*
Initial	Competent people	
Repeatable	Project management	Software project planning Software configuration management
Defined	Definition of processes	Process definition Training program Peer reviews
Managed	Product and process quality	Quantitative process metrics Software quality management
Optimising	Continuous process improvement	Defect prevention Process change management Technology change management

1. At this level, the processes for both management and development activities are defined and documented. There is a common organisation-wide understanding of activities, roles, and responsibilities.
2. The processes though defined, the process and **product qualities** are **not** measured.
3. At this level, the organisation builds up the capabilities of its employees through periodic training programs.
4. Also, review techniques are emphasized and documented to achieve phase containment of errors.
5. ISO 9000 aims at achieving this level.

Level 4: Managed:

1. At this level, the focus is on software metrics. Both process and product metrics are collected. Quantitative quality goals are set for the products and at the time of completion of development it was checked whether the quantitative quality goals for the product are met.
2. Various tools like Pareto charts, fishbone diagrams, etc. are used to measure the product and process quality.

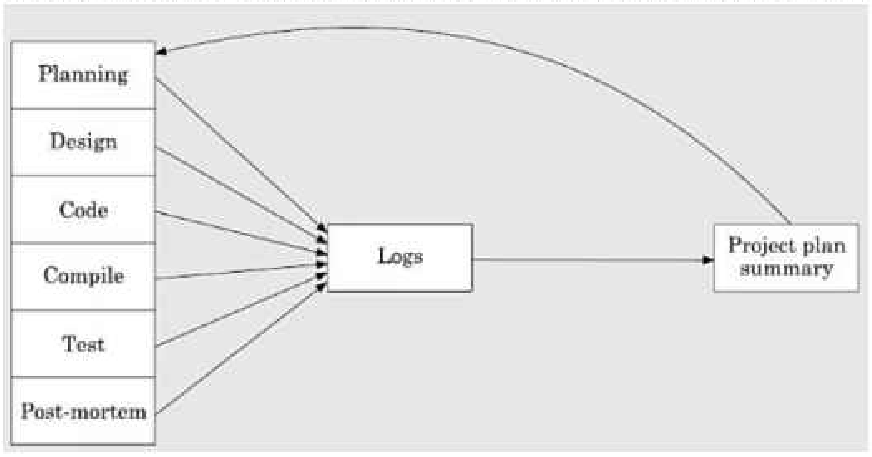
Level 5: Optimizing:

1. At this stage, process and product metrics are collected. Process and product measurement data are analyzed for continuous process improvement.
2. At CMM level 5, an organisation would identify the best software engineering practices and innovations (which may be tools, methods, or processes) and would transfer these organisation-wide.

Personal Software Process (PSP)

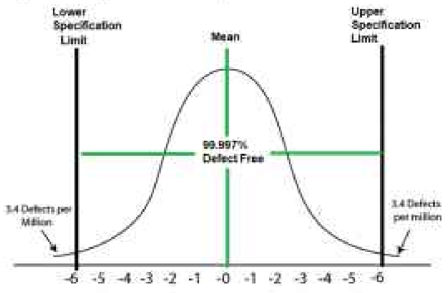
- ✓ PSP is based on the work of David Humphrey [Hum97].
- ✓ PSP is a scaled down version of industrial software process.
- ✓ PSP is suitable for individual use.
- ✓ PSP recognises that the process for individual use is different from that necessary for a **team.**
- ✓ PSP is a framework that helps engineers to measure and improve the way they work.
- ✓ It helps in developing personal skills and methods by estimating, planning, and tracking performance against plans, and provides a defined process which can be tuned by individuals.

Time measurement: PSP advocates that developers should rack the way they spend time. For example, he may stop the clock when attending a telephone call, taking a coffee break, etc. An engineer should measure the time he spends for various development activities such as **designing**, **writing code, testing**, etc.
The PSP is schematically shown in Figure . While carrying out the different phases, an individual must record the log data using time measurement. During post-mortem, they can compare the log data with their project plan to achieve better planning in the future projects, to improve his process, etc.



SIX SIGMA

- ✓ General Electric (GE) corporation first began Six Sigma in 1995.
- ✓ The purpose of Six Sigma is to improve processes to do things better, faster, and at lower cost. It can be used to improve every facet of business, from production, to human resources, to order entry, to technical support.
- ✓ Six Sigma can be used for any activity that is concerned with cost, timeliness, and quality of results. Therefore, it is applicable to virtually every industry.
- ✓ The statistical representation of Six Sigma describes quantitatively how a process is performing
- ✓ To achieve Six Sigma, a process must not produce more than 3.4 defects per million opportunities



Six Sigma Curve

Six Sigma Methodologies:

Two methodologies used in the Six Sigma projects are DMAIC and DMADV.

DMAIC is used to enhance an **existing** business process.

The DMAIC project methodology has five phases:

1. Define
2. Measure
3. Analyze
4. Improve
5. Control

DMADV is used to create **new** product designs or process designs. The DMADV project methodology also has five phases:

1. Define
2. Measure
3. Analyze
4. Design
5. Verify

Computer Aided Software Engineering (CASE)

CASE tools helps to improve software development effort and maintenance effort.

CASE AND ITS SCOPE

Definition: CASE tool can mean any tool used to automate some activity associated with software development. CASE tools assist in phase-related tasks such as specification, structured analysis, design, coding, testing, etc. and others to non-phase activities such as project management and configuration management.

The primary objectives in using any CASE tool are:

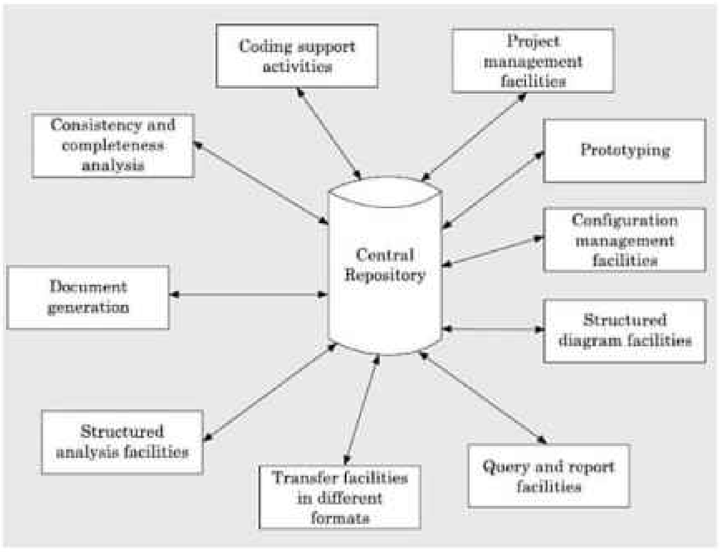
1. To increase productivity.
2. To help produce better quality software at lower cost.

CASE ENVIRONMENT

CASE environment facilitates the automation of the step-by-step methodologies for software development (just like programming environment which is an integrated collection of tools to support only the coding).

Benefits of CASE Environment:

1. Cost saving through all developmental phases.
2. improvement in quality
3. CASE tools help produce high quality and consistent documents
4. Impact on the style of working of a company, and makes it oriented towards the structured and orderly approach.


Coding support activities
Project management facilities
Consistency and completeness analysis
Prototyping
Configuration management facilities
Central Repository
Document generation
Structured diagram facilities
Structured analysis facilities
Transfer facilities in different formats
Query and report facilities

CASE Support in Software Life Cycle

Prototyping Support: The prototyping CASE tool's requirements are as follows:

- Define user interaction.
- Define the system control flow.
- Store and retrieve data required by the system.
- Incorporate some processing logic.

Structured Analysis and Design:

- Several diagramming techniques are used for structured analysis and structured design.
- A CASE tool should support one or more of the structured analysis and design technique.
- The CASE tool should support effortlessly drawing analysis and design diagrams.
- The CASE tool should support drawing fairly complex diagrams and preferably through a hierarchy of levels.

Code Generation: CASE tool during code generation phase are the following

- The CASE tool should support generation of module skeletons or templates in one or more popular languages.
- It should be possible to include copyright message, brief description of the module, author name and the date of creation in some selectable format.
- The tool should generate records, structures, class definition automatically from the contents of the data dictionary in one or more popular programming languages.
- It should generate database tables for relational database management systems.

Test Case Generator: The CASE tool for test case generation should have the following features:

- It should support both design and requirement testing
- It should generate test set reports in ASCII format which can be directly imported into the test plan document.

Project Management: It should support collecting, storing, and analyzing information on the Software project's progress such as the estimated task duration, scheduled and actual task start, completion date, dates and results of the reviews, etc.

Reverse Engineering Support: The tool should support generation of structure charts and data Dictionaries from the existing source codes.

CHARACTERISTICS OF SOFTWARE MAINTENANCE

Maintenance is inevitable for almost any kind of product. However, most products need maintenance due to the wear and tear caused by use. On the other hand, **software products** do not need maintenance on this count, but need maintenance **to correct errors, enhance features, port to new platforms**, etc.

Types of Software Maintenance

There are three types of software maintenance, which are described as follows:

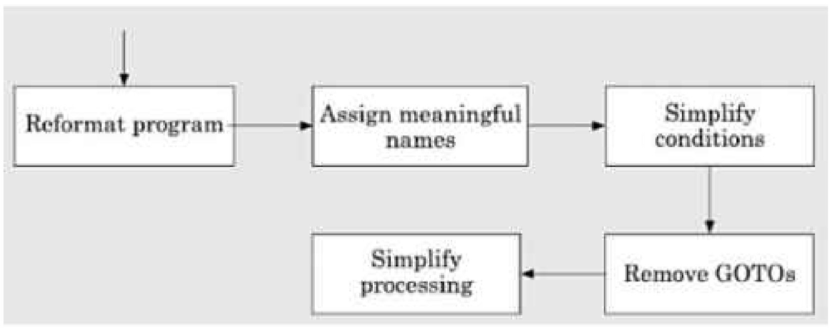
1. **Corrective**: Corrective maintenance of a software product is necessary either to rectify the bugs observed while the system is in use.
2. **Adaptive**: A software product might need maintenance when the customers need the product to run on new platforms, on new operating systems, or when they need the product to interface with new hardware or software.
3. **Perfective**: A software product needs maintenance to support the new features that users want it to support, to change different functionalities of the system according to customer demands, or to enhance the performance of the system.

SOFTWARE REVERSE ENGINEERING

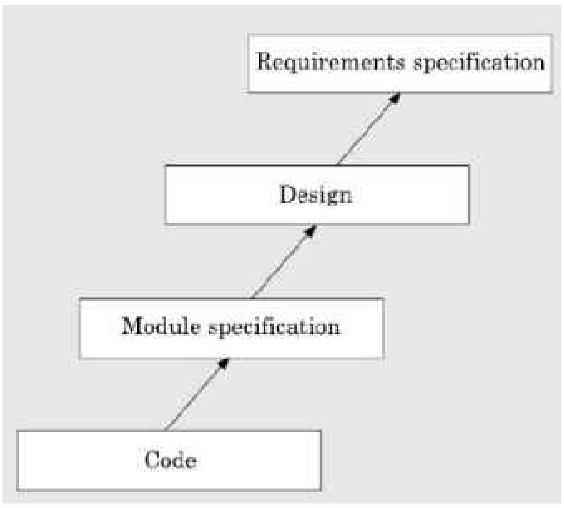
- Software reverse engineering is the process of recovering the **design** and the requirements specification of a product from an **analysis of its code**.
- The purpose of reverse engineering is to facilitate maintenance work by improving the understandability of a system and to produce the necessary documents for a legacy system.
- Reverse engineering is becoming important, since legacy software products lack proper documentation, and are highly unstructured.

Steps in Software Reverse Engineering

Step 1: The first stage of reverse engineering usually focuses on carrying out **cosmetic** changes to the code to improve its readability, structure, and understandability, without changing any of its functionalities.



Step-2: After the cosmetic changes have been carried out on legacy software, the process of extracting the code, design, and the requirements specification can begin.

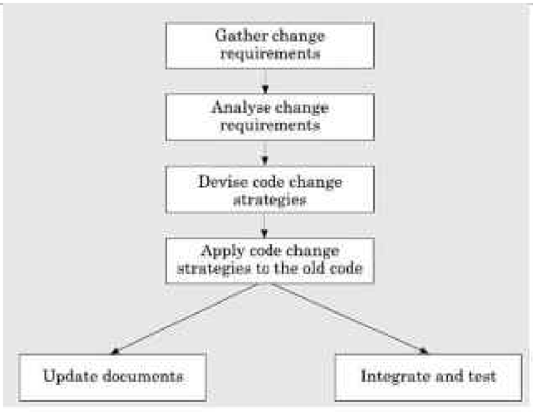


SOFTWARE MAINTENANCE PROCESS MODELS

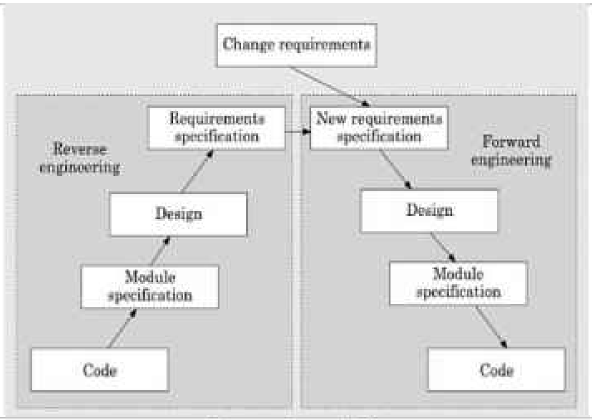
Two broad categories of process models can be proposed.

First model:

The first model is preferred for projects involving small reworks where the code is changed directly and the changes are reflected in the relevant documents later. This maintenance process is graphically presented in Figure 13.3.


Gather change requirements
Analyse change requirements
Devise code change strategies
Apply code change strategies to the old code
Update documents
Integrate and test

First model



Second model

Second model:

The second model is preferred for projects where the amount of rework required is significant. This approach can be represented by a reverse engineering cycle followed by a forward engineering cycle. Such an approach is also known as **software re-engineering**. This process model is depicted in Figure 13.4.

ESTIMATION OF MAINTENANCE COST

Boehm [1981] proposed a formula for estimating maintenance costs as part of his COCOMO cost estimation model. Boehm's maintenance cost estimation is made in terms of a quantity called the annual change traffic (ACT). Boehm defined ACT as the fraction of a software product's source instructions which undergo change during a typical year either through addition or deletion.

$$\text{ACT} = \frac{\text{KLOC}_{\text{added}} + \text{KLOC}_{\text{deleted}}}{\text{KLOC}_{\text{total}}}$$

where, KLOC added is the total kilo lines of source code added during maintenance.
KLOC deleted is the total KLOC deleted during maintenance.
Thus, the code that is changed should be counted in both the code added and code deleted.
The annual change traffic (ACT) is multiplied with the total development cost to arrive at the maintenance cost:

$$\text{Maintenance cost} = \text{ACT} \times \text{Development cost}$$

BASIC ISSUES IN ANY REUSE PROGRAM

WHAT CAN BE REUSED

The prominent items that can be effectively reused are:

1. Requirements specification
2. Design
3. Code
4. Test cases
5. Knowledge

BASIC ISSUES IN ANY REUSE PROGRAM

The following are some of the basic issues that must be clearly understood for starting any reuse program:

1. Component creation.
2. Component indexing and storing.
3. Component search.
4. Component understanding.
5. Component adaptation.
6. Repository maintenance.

Component creation: For component creation, the reusable components have to be first identified. Selection of the right kind of components having potential for reuse is important.

Component indexing and storing: Indexing requires classification of the reusable components so that they can be easily searched when we look for a component for reuse. The components need to be stored in a relational database management system (RDBMS) or an object-oriented database system (ODBMS) for efficient access when the number of components becomes large.

Component searching: The programmers need to search for right components matching their requirements in a database of components. To be able to search components efficiently, the programmers require a proper method to describe the components that they are looking for.

Component understanding: The programmers need a precise and sufficiently complete understanding of what the component does to be able to decide whether they can reuse the component.

Component adaptation: Often, the components may need adaptation before they can be reused, since a selected component may not exactly fit the problem at hand.

Repository maintenance: A component repository once is created requires continuous maintenance.

A REUSE APPROACH

Domain analysis approach to create reusable components.

Evolution of a reuse domain:

- **Stage 1** : There is no clear and consistent set of notations. Obviously, no reusable components are available. All software is written from scratch.
- **Stage 2** : H e r e , only experience from similar projects are used in a development effort. This means that there is only knowledge reuse.
- **Stage 3**: At this stage, the domain is ripe for reuse. The set of concepts are stabilised and the notations standardized. Standard solutions to standard problems are available. There is both knowledge and component reuse.
- **Stage 4**: The domain has been fully explored. The software development for the domain can largely be automated. Programs are not written in the traditional sense any more. Programs are written using a domain specific language, which is also known as an **application generator**.

Steps to be followed

1. **Components** need to be properly classified in order to develop an effective indexing and storage scheme.
2. The domain **repository** may contain thousands of reuse items. In such large domains, what is the most efficient way to **search** an item that
3. **Repository maintenance** involves entering new items, retiring those items which are no more necessary, and modifying the search attributes of items to improve the effectiveness of search.
4. Reuse without Modifications: One can directly plug in the parts to develop his application. The specification usually is written using **4GL**

REUSE AT ORGANISATION LEVEL

Achieving organisation-level reuse requires adoption of the following steps:

1. Assess of an item's potential for reuse.
2. Refine the item for greater reusability.
3. Enter the product in the reuse repository.

1. Assessing a product's potential for reuse : Assessment of a components reuse potential can be obtained from an analysis of a questionnaire circulated among the developers. The questionnaire can be devised to assess a component's reusability. The programmers working in similar application domain can be used to answer the questionnaire about the product's reusability.

A sample questionnaire to assess a component's reusability is the following:

- Is the component's functionality required for implementation of systems in the future?
- How common is the component's function within its domain?
- Would there be a duplication of functions within the domain if the component is taken up?
- Is the component hardware dependent?
- Can we parameterize a non-reusable component so that it becomes reusable?

2. Refining products for greater reusability: For a product to be reusable, it must be relatively easy to adapt it to different contexts. Machine dependency must be abstracted out or localized using data encapsulation techniques. The following refinements may be carried out:

- Name generalization
- Operation generalization
- Exception generalization

3.Enter the product in the reuse repository. : The items should enter into a repository for between availability.