UNIT-5: SOFTWARE QUALITY, RELIABILITY AND OTHER ISSUES

S/W reliability, Statistical testing, S/W quality and management, ISO-9000, SEI Capability maturity model (CMM), Personal S/W process (PSP), Six sigma, S/W quality metrics, CASE and its scope, CASE environment, CASE support in S/W life cycle, Characteristics of s/w maintenance, S/W reverse engineering, S/W maintenance process model, Estimation maintenance cost, Basic issues in any reuse program, Reuse approach, Reuse at organization level.

SOFTWARE RELIABILITY

Reliability of S/W product essentially denotes its trustworthiness or dependability. Alternatively, reliability of S/W product can also be defined as the probability of the product working correctly over a given period of time. It has been experimentally observed by analyzing the behaviour of a large number of programs that 90% of the execution time of a typical program is spent in executing only 10% of the instructions in the program. These most used 10% instructions are often called the core of the program. The rest 90% of the program statements are called non-core and are executed only for 10% of the total execution time. It therefore may not be very surprising to note that removing 60% product defects from the least used parts of a system would typically lead to only 3% improvement to the product reliability. It is clear that the quantity by which the overall reliability of a program improves, due to the correction of a single error, depends on how frequently the corresponding instruction is executed.

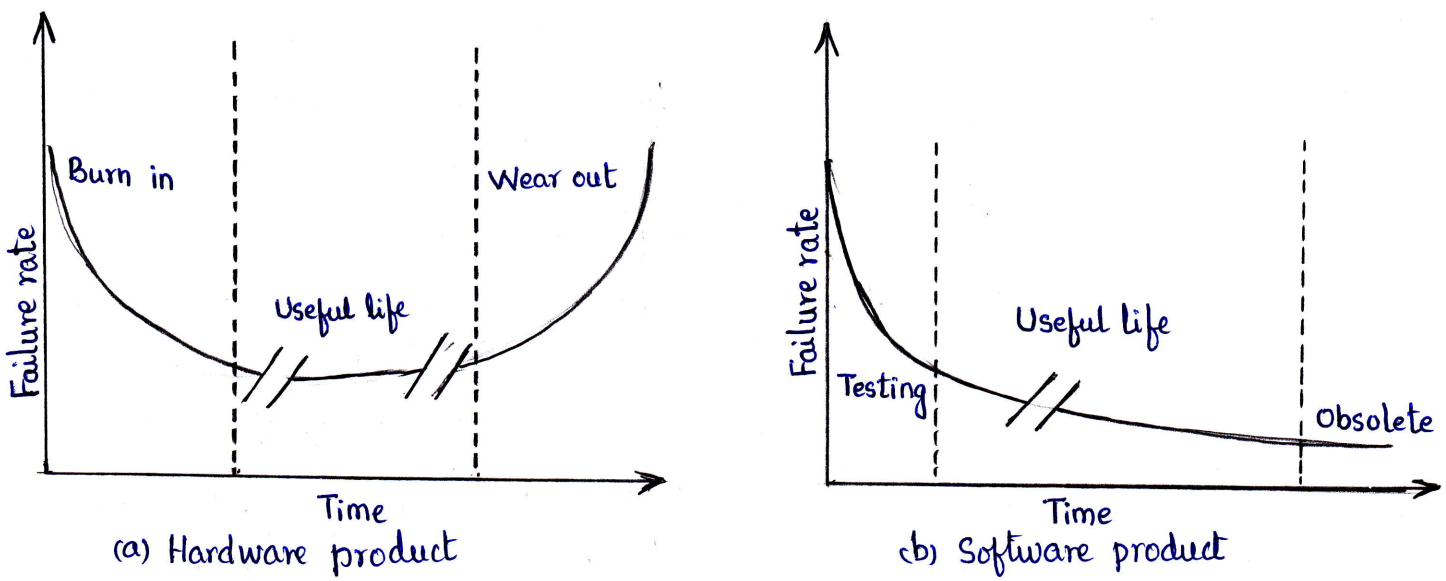
Thus, the reliability of a product depends not only on the number of latent errors but also on the exact location of the errors. Apart from this, reliability also depends upon how the product is used, i.e. on its execution profile. If we select input data to the system such that only the correctly implemented functions are executed, none of the errors will be exposed and the perceived reliability of the product will be high.

The s/w reliability is difficult to measure can be summarized as follows:

- The reliability improvement due to fixing a single bug depends on where the bug is located in the code.
- The perceived reliability of a s/w product is highly observer-dependent.
- The reliability of a product keeps changing as errors are detected and fixed.

1. Hardware Vs Software Reliability: Reliability behaviour for h/w and s/w is very different. For example, h/w failures are inherently different from s/w failures. Most h/w failures are due to component wear and tear. A logic gate can get stuck at a 1 or 0, or a resistor might have short-circuited. To fix h/w faults, one has to either replace or repair failed part. On the other hand, a s/w product would continue to fail until the error is tracked down either the design or the code is changed. When a s/w failure is repaired, the reliability may increase or decrease.

The changes in failure rate over the product life time for a typical h/w product and a s/w product are sketched below:



(a) Hardware product (b) Software product

Fig: Change in failure rate of a product

Observe that the failure rate is high initially but decreases as the faulty components are identified and removed. The system then enters its useful life. After some time the components wear out, and the failure rate increases. This gives the plot of h/w reliability over time its characteristics bath tub shape. On the other hand, for s/w the failure rate is at its highest during integration and testing phase. As the system is tested more and more errors are identified and moved resulting in a reduced failure rate. This error removal continues at a slower pace during the useful life of the product. As the s/w becomes obsolete, no more error correction occurs and the failure rate remains unchanged.

2. Reliability Metrics :

The reliability requirements for different categories of s/w products may be different. For this reason, it is necessary that the level of reliability required for a s/w product should be specified in the SRS document. In order to be able to do this, we need some metrics to quantitatively express the reliability of a s/w product. A good reliability measure should be observer-independent, so that different people can agree on the degree of reliability that a system has. However, in practice, it is very difficult to formulate a precise reliability measurement technique. There are six reliability metrics which can be used to quantify the reliability of s/w products.

1. Rate of Occurrence of Failure (ROCOF) : ROCOF measures the frequency of occurrence of unexpected behavior/failures. The ROCOF measure of a s/w product can be obtained by obtaining the behaviour of a s/w product in operation over a specified time interval and then calculating the total number of failures during the interval.

2. Mean Time to Failure (MTTF) : MTTF is the average time between two successive failures, observed over a large number of failures. To measure MTTF, we can record the failure data for n failures. Let the failures

occur at the time instants $t_1, t_2, \ldots t_n$. 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. MTTR measures the average time it takes to track the errors causing the failure and then to fix them.

4. Mean Time Between Failures (MTBF) : We can combine the MTTF and MTTR metric to get the MTBF metric : MTBF = MTTF + MTTR. Thus, MTBF of 300 hours indicates that once a failure occurs, the next failure is expected to occur only after 300 hours. In this case, the time measurements are real time and not the execution time as in MTTF.

5. Probability of Failure on Demand (POFOD) :- POFOD measures the likelihood of the system failing when a service request is made. For example, a POFOD of 0.001 would mean that 1 out of every 1000 service requests would result in a failure.

6. Availability : Availability of a system is a measure of how likely will the system be available for use over a given period of time. This metric not only considers the number of failures occurring during a time interval, but also takes into account the repair time of a system when a failure occurs. This metric is important for systems such as telecommunication systems and operation systems, which are supposed to be never down and where repair and restart time are significant and loss of service during that time is important.

3. Reliability Growth Modelling : A reliability growth model is a mathematical model of how S/W reliability improves as errors are detected and repaired. A reliability growth model can be used to predict when 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. Although several different

reliability growth models have been proposed, the below are two simple reliability growth models.

Jelinski and Moranda Model : The simplest reliability growth model is a step function model where it is assumed that the reliability increases by a constant increment each time an error is detected and repaired. However, this simple model of reliability which implicitly assumes that all errors contribute equally to reliability growth, is highly unrealistic since we already know that corrections of different errors contribute differently to reliability growth.

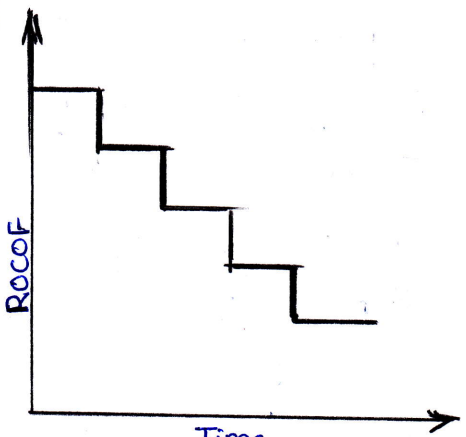


Fig: Step function model of reliability growth

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. It also models the fact that as errors are repaired, the average improvement is reliability per repair decreases.

There are more complex reliability growth models, which give greater accurate approximation to the reliability growth. However, these models are beyond the scope of this text.

STATISTICAL TESTING

Statistical testing is a process whose objective is to determine the reliability of the product rather than discovering errors. The test cases designed for statistical testing have an entirely different objective from that of conventional testing. To carry out statistical testing, need to define first the operation profile of the product.

Operation Profile : Different categories of users may use a s/w for very different purposes. Formally, we can define the operation profile of a s/w as the probability distribution of the input of an average user. If we divide the input into a number of classes $\{c_i\}$, the probability value of

a class represents the probability of an average user selecting his next input from this class. Thus, the operation profile assigns a probability value {Pi} to each input class {Ci}.

How to define the operation profile for a product? : We need to divide the input data into a number of input classes. Then need to assign a probability value to each input class; this probability value would signify probability for an input value from that class to be selected. The operation profile of a s/w product can be determined by observing and analyzing the usage pattern of a number of users.

Steps in statistical Testing : The first step is to determine the operation profile of the s/w. The next step is to generate a set of test data corresponding to the determined operation profile. The third step is to apply the test cases to the s/w and record the time between each failure. After a statistically significant number of failures have been observed, the reliability can be computed.

For accurate results, statistical testing requires some fundamental assumptions to be satisfied. It requires a statistically significant number of test cases to be used. It further requires that a small percentage of test inputs that are likely to cause system failure be included.

Pros and Cons of Statistical Testing : It allows one to concentrate on testing parts of the system that are most likely to be used. Therefore, it results in a system that the users find more reliable. Also, the reliability estimation arrived at by using statistical testing is more accurate compared to other methods. However, it is not easy to do statistical testing properly. There is no simple and repeatable way of defining operation profiles. Also, the number of test cases with which the system is to be tested should be statistically significant.

SOFTWARE QUALITY & MANAGEMENT

A quality management system is the principal methodology used by organizations to ensure that the products they develop have the desired quality. A quality system consists of the following:

- **Managerial structure and individual responsibilities:** A quality system is actually the responsibility of the organization as a whole. However, many organizations have a separate quality department to perform several quality system activities. The quality system of an organization should have the support of the top management.
- **Quality System Activities:** The quality system activities encompass the following:
  - Auditing of the projects
  - Review of the quality system
  - Development of standards, procedures, and guidelines, etc.
  - Production of reports for the top management summarizing the effectiveness of the quality system in the organization.

A good quality system must be well documented. Without a properly documented quality system, the application of quality controls and procedures become ad hoc, resulting in large variations in the quality of the products delivered. Also an undocumented quality system sends clear messages to the staff about the attitude of the organization towards quality assurance. International Standards such as ISO 9000 provide guidance on how to organize a quality system.

Evolution of Quality Systems Quality systems have rapidly evolved over the last five decades. Quality systems of organizations have undergone through four stages of evolution. The initial product inspection method gave way to quality control (QC). Quality control focuses not only on detecting the defective products and eliminating them but also on determining the causes behind the defects. 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 quality assurance principles.

The modern quality paradigm includes certain guidance for recognizing, defining, analyzing, and improving the production process. Total quality management (TQM) advocates that the process followed by an organization must be continuously improved through process measurements. TQM goes a step further than quality assurance and aims at continuous process improvement. It goes beyond documenting processes with a view to optimizing them through redesign. A term related to TQM is Business Process Reengineering (BPR). BPR aims at reengineering the way business is carried out in an organization.

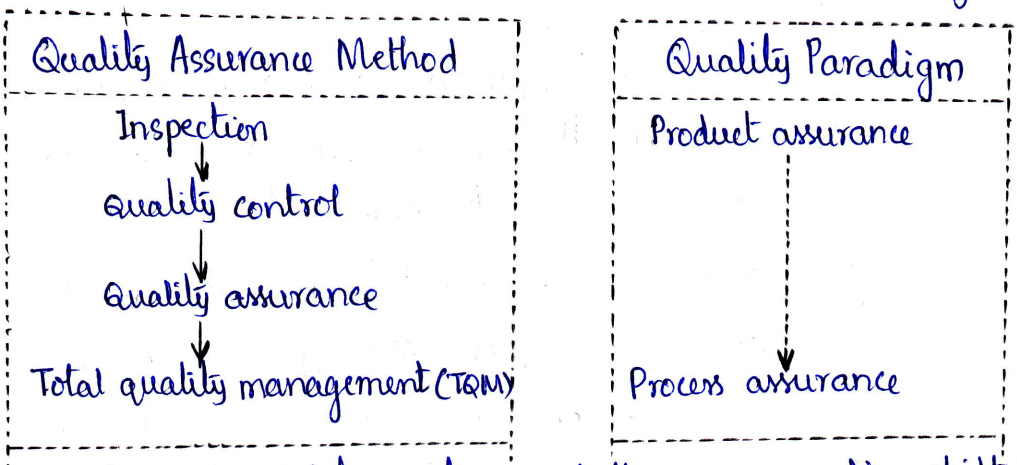


Fig: Evolution of quality system and the corresponding shift in the quality paradigm.

ISO 9000

ISO (International Standards Organization) is a consortium of 63 countries established to formulate and foster standardization. ISO published its 9000 series of standards in 1987.

What is ISO 9000 Certification? ISO 9000 certification serves as a reference for contract between independent parties. The ISO9000 standard

specifies the guidelines for maintaining a quality system. The ISO 9000 standard mainly addresses operational aspects of organizational aspects such as responsibilities, reporting, etc. It is important to realize that ISO 9000 standard is a set of guidelines for the production process and is not directly concerned with the product itself.

ISO 9000 is a series of three standards: ISO 9001, ISO 9002, and ISO 9003. The types of s/w industries to which the different ISO standards apply are as follows:

ISO 9001: This standard applies to the organizations engaged in design, development, production, and servicing of goods. This is the standard that is applicable to most s/w development organizations.

ISO 9002: This standard is applied to those organizations which do not design products but are only involved in production. Example 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 s/w development organizations.

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

ISO 9000 for S/W Industry: ISO 9000 is a generic standard that is applicable to a large gamut of industries, starting from a steel manufacturing industry to a service rendering company. Therefore, many of the clauses of ISO 9000 documents are written using generic technologies and it is very difficult to interpret them in the context of s/w development organizations. There are two primary reasons behind this:

- S/w is intangible and therefore difficult to control. It is difficult to control and manage anything that we cannot see and feel. Therefore it is easy to accurately determine how much work has been completed and to estimate how much more time will it take.
- During s/w development, the only raw material consumed is data. In contrast, large quantities of raw materials are consumed during the development of any other product. ISO 9000 standards have many clauses corresponding to raw material control. These are obviously not relevant to s/w development organizations.

Due to such radical differences between s/w and other types of product development, it was difficult to interpret various clauses of the original Iso standard in the context of s/w industry. Therefore, ISO released a separate document called ISO 9000 part-3 in 1991 to help interpret the ISO standard for s/w industry.

Why Get ISO 9000 Certification?

- Confidence of customers in an organization increases when the organization qualifies for ISO certification. This is especially true in the international market.
- ISO 9000 requires a well-documented s/w production process to be in place. A well-documented s/w production process contributes to repeatable and higher quality of the developed s/w.
- ISO 9000 makes the development process focused, efficient, and cost effective.
- ISO 9000 certification points out the weak points of an organization and recommends remedial action.
- ISO 9000 sets the basic framework for the development of an optimal process and TQM.

How to get ISO 9000 Certification?

An organization intending to obtain ISO 9000 certification applies to an ISO 9000 registrar for registration. The ISO 9000 registration process consists of the following stages:

Application: Once an organization decides to go for ISO 9000 certification, it applies to a registrar for registration.

Pre-assessment: During this stage, the registrar makes a rough assessment of the organization.

Document review and adequacy of audit: During this stage, the registrar reviews the documents submitted by the organization and makes suggestions for possible improvements.

Compliance audit: During this stage, the registrar checks whether the suggestions made by it during review have been compiled with by the organization or not.

Registration: The registrar awards the ISO 9000 certificate after successful completion of all previous phases.

Continued Surveillance: The registrar continues to monitor the organization, through periodically.

ISO mandates that a certified organization can use the certificate for corporate advertisements but cannot use the certificate for advertising any of its products. This is probably due to the fact that the ISO 9000 certificate is issued for an organization's process and does not apply to any specific product of the organization. An organization using ISO certificate for product advertisements faces the risk of withdrawal of the certificate. In India, ISO 9000 certification is offered by (BIS) Bureau of Indian Standards, STQC (Standardization, Testing, and Quality Control), and IRQS (Indian Register Quality System). IRQS has been accredited by the Dutch council of certifying bodies.

SEI CAPABILITY MATURITY MODEL (SEICMM)

Software Engineering Institute Capability Maturity Model helped organizations to improve the quality of the s/w they develop and therefore adoption of SEICMM model has significant business benefits.

SEI CMM can be used in two ways: capability evaluation and s/w process assessment. Capability evaluation and s/w process assessment differ in motivation, objective, and the final use of the result. Capability evaluation provides a way to assess the s/w capability of an organization. The results of capability evaluation indicates the likely contractor performance if the contractor is awarded a work. Therefore, the results of s/w process capability assessment is used by an organization with the objective to improve its process capability. Thus, this type of assessment is for purely internal use.

SEI CMM classifies s/w development industries into the following five maturity levels. The different levels of SEI CMM have been designed so that it is easy for an organization to slowly build its quality systems starting from scratch.

Level 1 : Initial. A s/w development organization at this level is characterized by ad hoc activities. Very few or no processes are defined and followed. Since s/w production processes are not defined different engineers follow their own process and as a result development efforts become chaotic. Therefore, it is also called chaotic level. The success of projects depends on individual efforts and heroics. When engineers leave, the successors have great difficulty in understanding the process followed and the work completed. Since formal project management practices are not followed, under time pressure shortcuts are tried out leading to low quality.

Level 2 : Repeatable. At this level, the basic project management practices such as tracking cost and schedule are established. Size and cost estimation techniques like function point analysis, cocomo, etc. are used. The necessary process discipline is in place to repeat earlier success on projects with similar applications.

Level 3 : Defined. At this level the process for both management and development activities are defined and documented. There is a common organization-wide understanding of activities, roles and responsibilities. The process though defined, the process and product qualities are not measured. ISO 9000 aims at achieving this level.

Level 4 : Managed. At this level, the focus is on s/w metrics. Two types of metrics are collected. Product metrics measure the characteristics of the product being developed, such as its size, reliability, time complexity, understandability, etc. Process metrics reflect the effectiveness of the process being used, such as average defect correction time, productivity, average number of defects found per hour inspection, average number of failures detected during testing per LOC, etc. The s/w process and product quality are measured and quantitative quality requirements for the product are met. Various tools like Pareto Charts, Fishbone diagrams, etc. are used to measure the product and process quality. The process metrics are used to check if a project performed satisfactorily. Thus, the results of process measurements are used to evaluate project performance rather than improve the process.

Level 5 : Optimizing At this stage, process and product metrics are collected. Process and product measurement data are analyzed for continuous process improvement. Continuous process improvement is achieved both by carefully analyzing the quantitative feedback

from the process measurements and also from application of innovative ideas and technologies. Such an organization identifies the best s/w engineering practices and innovations which may be tools, methods, or processes. These best practices are transferred throughout the organization.

Key process areas (KPA) of a s/w organization:

Except for SEI CMM level 1, each maturity level is characterized by several key process areas that includes the areas an organization should focus to improve its s/w process to the next level. The focus of each level and the corresponding key process areas are shown in the below table:

CMM Level	Focus	Key process areas
1. Initial	Competent people	
2. Repeatable	Project Management	S/w project planning S/w Configuration Mgmt.
3. Defined	Definition of processes	Quantitative process metrics S/w quality management
4. Managed	Product and process quality	Process definition Training program Peer reviews
5. Optimizing	Continuous process improvement	Defect prevention Process change management Technology change mgmt.

Fig: The focus of each SEI CMM level and the corresponding Key process areas

SEI CMM provides a list of key process areas on which to focus to take an organization from one level of maturity to the next. Thus, it provides a way for gradual quality improvement over several stages. Each stage has been carefully designed such that one

stage enhances the capability already built up. For example, it considers that trying to implement a defined process (level 3) before a repeatable process (level 2) would be counterproductive as it becomes difficult to follow the defined process due to schedule and budget pressures.

Applicability of SEI CMM to organizations:

SEI CMM model is perfectly applicable for large organizations. But small organizations typically handle applications such as internet, e-commerce, and are without an established product range, revenue base, and experience on past projects, etc. For such organizations, a CMM-based appraisal is probably excessive. These organizations need to operate more efficiently at the lower levels of maturity. For example, they need to practice effective project management, reviews, configuration management, etc.

PERSONAL SOFTWARE PROCESS (PSP)

Personal S/w Process is a scaled down version of the industrial s/w process. PSP is useful for individual use. PSP recognizes that the process for individual use is different from that necessary for a team. The quality and productivity of an engineer is to a great extent dependent on his process. 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 and planning, by showing how to track performance against plans, and provides a defined process which can be tuned by individuals.

Time Measurement: PSP advocates that engineers should rack the

the way they spend time. Because, boring activities seem longer than actual and interesting activities seem short. Therefore, the actual time spent on a task should be measured with the help of a stop-clock to get an objective picture of the time spent. An engineer should measure the time he spends for designing, writing code, testing, etc.

PSP Planning : Individuals must plan their project. They must estimate the maximum, minimum, and the average LOC required for the product. They should use their productivity in minutes/LOC to calculate the maximum, minimum, and the average development time. They must record the plan data in a project plan summary.

The PSP is schematically shown in below figure, while carrying out the different phases, they 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 their process, etc.

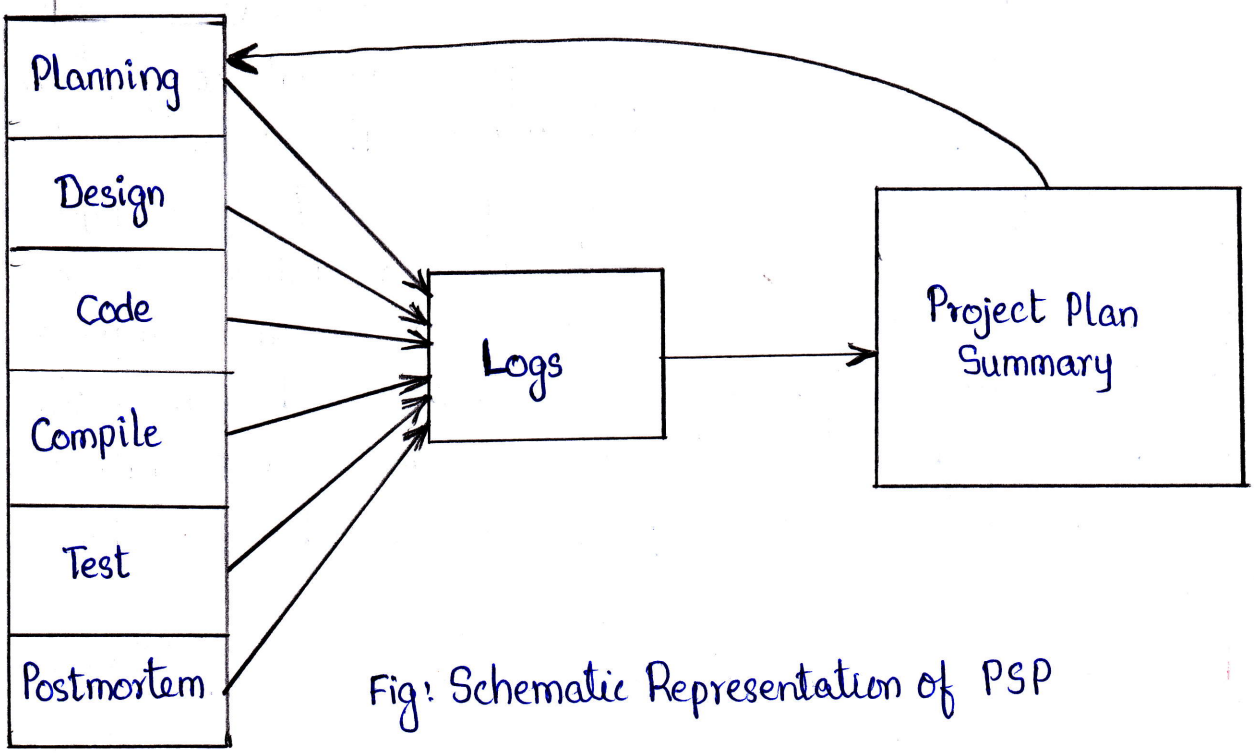


Fig: Schematic Representation of PSP

PSP2 introduces defect management via the use of checklists for code and design reviews. The checklists are developed from gathering and analyzing defect data earlier projects.

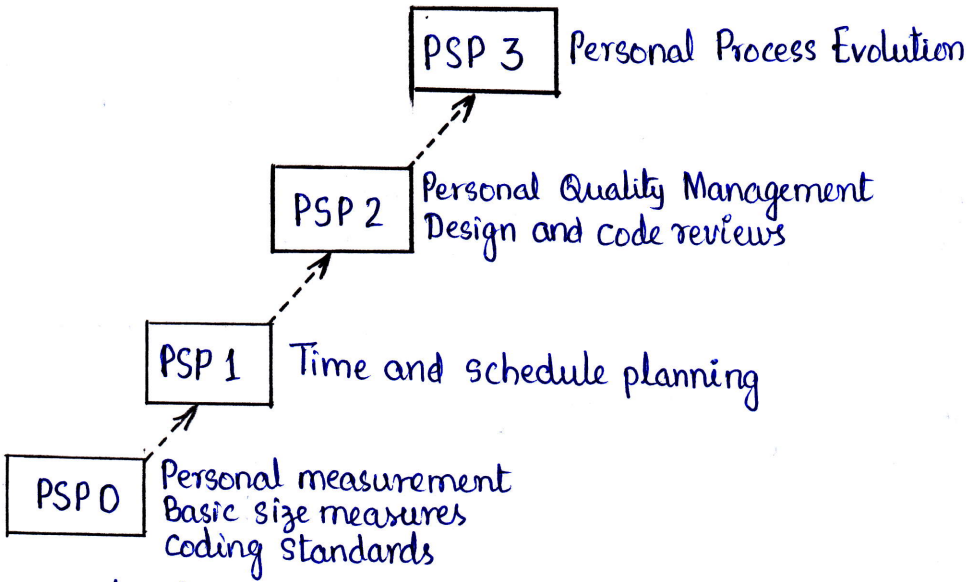


Fig: Levels of PSP

SIX SIGMA

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, timelines, and quality of results. Therefore, it is applicable virtually to every industry.

Six Sigma at many organizations simply means striving for near perfection. Six Sigma is a disciplined, data-driven approach to eliminate defects in any process - from manufacturing to transactional and product to service.

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. A Six Sigma defect is defined as any system behavior

that is not as per customer specifications. Total number of Six Sigma opportunities is then the total number of chances for a defect. Process sigma can easily be calculated using a Six Sigma calculator.

The fundamental objective of Six Sigma methodology is the implementation of a measurement-based strategy that focuses on process improvement and variation reduction through the application of Six Sigma improvement projects. This is accomplished through the use of two six sigma sub-methodologies: DMAIC and DMADV. The Six Sigma DMAIC process (define, measure, analyze, improve, control) is an improvement system for existing processes falling below specification and looking for incremental improvement. The Six Sigma DMADV (define, measure, analyze, design, verify) is an improvement system used to develop new processes or products at Six Sigma quality levels. It can also be employed if a current process requires more than just incremental improvement. Both Six Sigma processes are executed by six Sigma Green Belts and Six Sigma Black Belts, and are overseen by Six Sigma Master Black Belts.

Many frameworks exist for implementing the Six Sigma methodology. Six Sigma Consultants all over the world have also developed proprietary methodologies for implementing Six Sigma quality, based on the similar change management philosophies and applications of tools.

SOFTWARE QUALITY METRICS

S/W Quality Metrics are a subset of s/w metrics that focus on the quality aspects of the product, process and project. These are more closely associated with process and product metrics than project metrics.

S/W quality metrics can be further divided into three categories :

- Product quality metrics
- In-process quality metrics
- Maintenance quality metrics

1. **Product Quality Metrics** : This metrics includes the following:
   - Mean Time to Failure
   - Defect Density
   - Customer Problems
   - Customer Satisfaction

Mean Time to Failure : It is the time between failures. This metric is mostly used with safety critical systems such as the airline traffic control systems, avionics and weapons.

Defect Density : It measures the defects relative to the s/w size expressed as lines of code or function point, etc. i.e. it measures code quality per unit. This metrics is used in many commercial s/w systems.

Customer Problems : It measures the problems that customers encounter when using the product. It contains the customer's perspective towards the problem space of the s/w, which includes the non-defect oriented problems together with the defect problems. The problems metric is usually expressed in terms of problems per User-Month (PUM).

PUM = Total problems that customers reported (true defect and non-defect oriented problems) for a time period + Total number of license months of the s/w during the period.

Where, Number of license-months of the software = Number of install license of the s/w X Number of months in the calculation year.

PUM is usually calculated for each month after the s/w is released to the market, and also for monthly averages by year.

Customer Satisfaction: It is often measured by customer survey data through the five point scale-

- Very satisfied
- Satisfied
- Neutral
- Dissatisfied
- Very dissatisfied

Satisfaction with the overall quality of the product and its specific dimensions is usually obtained through various methods of customer surveys. Based on the five-point-scale data, several metrics with slight variations can be constructed and used, depending on the purpose of analysis, for example-

- Percent of completely satisfied customers
- Percent of satisfied customers
- Percent of dissatisfied customers
- Percent of non-satisfied customers

2. In-Process Quality Metrics: In-process Quality Metrics deals with the tracking of defect arrival during formal machine testing for some organizations. This metric includes-

- Defect density during machine testing
- Defect arrival pattern during machine testing
- Phase-based defect removal pattern
- Defect removal effectiveness

3. Maintenance Quality Metrics :- Although much cannot be done to alter the quality of the product during this phase, following are the fixes that can be carried out to eliminate the defects as soon as possible with excellent fix quality.

- Fix backlog and backlog management index
- Fix response time and ~~and~~ fix responsiveness
- Percent delinquent fixes
- Fix quality

CASE AND ITS SCOPE :

CASE stands for Computer Aided S/w Engineering. CASE tools promise reduction in s/w development and maintenance costs. CASE tools help develop better quality products more efficiently.

A CASE tool is a generic term used to denote any form of automated support for S/w Engg. Many CASE tools are now available. Some of these tools assist in phase-related tasks such as specification, structured analysis, design, coding, testing, etc, and others are related to non-phase activities such as project management and configuration management. The primary objectives of deploying CASE tool are :

- To increase productivity
- To produce better quality S/w at lower cost.

CASE ENVIRONMENT

CASE tools are characterized by the stage or stages of s/w development life cycle on which they focus. Since different tools covering different stages share common information, it is required that they integrate through some central repository to have a consistent view of information associated with the s/w. This central repository usually a data dictionary containing the definitions of all composite and elementary data items. Through the central repository, all the CASE tools in a CASE environment share common information among themselves. Thus a CASE environment facilitates the automation of the step-by-step methodologies for s/w development. In contrast to a CASE environment, a programming environment is an integrated collection of tools to support only the coding phase of s/w development. The tools commonly integrated in a programming environment are a text editor, a compiler, and a debugger. The different tools are integrated to the extent that once the compiler detects an error, the editor automatically goes to the statements in error and the error statements are highlighted. Examples of popular programming environments are Turbo C environment, Visual Basic, Visual C++, etc. A schematic representation of a CASE environment, standard programming environments such as Turbo C, Visual C++, and more come equipped with a program editor, compiler, debugger, linker, etc. All these tools are integrated. If you click on an error reported by the compiler, not only does it take you into the editor, but also takes the cursor to the specific line or statement causing the error.

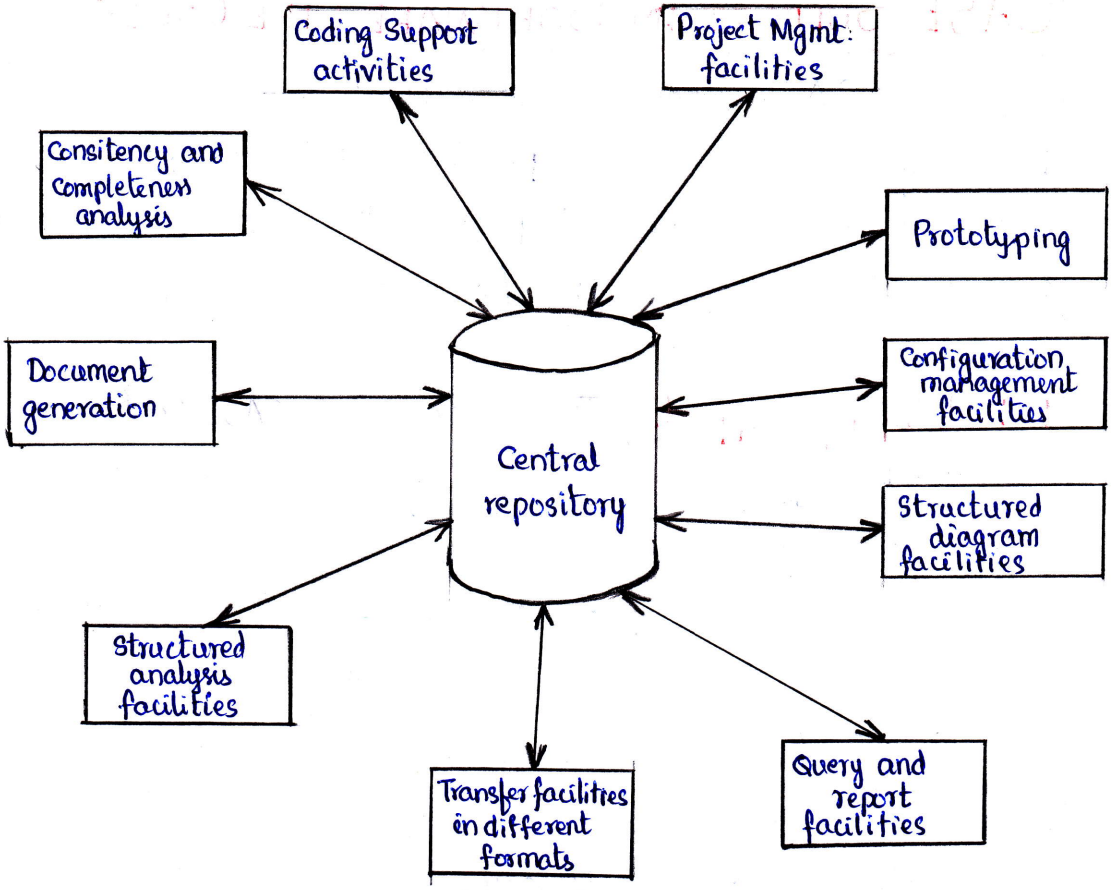


Fig: A CASE environment

Benefits of CASE: Several benefits accrue from the use of a CASE environment or even from the use of isolated CASE tools. Let us examine some of these benefits:

- A key benefit arising out of the use of a CASE environment is cost saving through all developmental phases. CASE put the effort reduction between 30% to 40%.
- Use of CASE tools leads to considerable improvements to quality.
- CASE tools help produce high quality and consistent documents.
- CASE tools reduce the drudgery (lazy at work) in a s/w engineer's work.
- CASE tools have led to revolutionary cost savings in s/w maintenance efforts.
- Use of a CASE environment has an impact on the style of working of a company, and makes it conscious of structured and orderly approach.

CASE Support in Software Life Cycle

CASE tools should support a development methodology, help enforce the same, and provide certain amount of consistency checking between different phases. The kind of support that CASE tools usually provide in the s/w development life cycle is elucidated below:

1. **Prototyping Support:** The prototyping CASE tools requirements are as follows:
   - Define user interaction
   - Define the system control flow
   - Store and retrieve data required by the system
   - Incorporate some processing logic

There are several standalone prototyping tools. But a tool that integrates with the data dictionary can make use of the entries in the data dictionary, help in populating the data dictionary and ensure the consistency between the design data and the prototype.

A good prototyping tool should support the following features:

- A prototyping CASE tool should support the user to create a GUI using a graphic editor. The user should be allowed to define all data entry forms, menus and controls.
- It should integrate with the data dictionary of a CASE environment.
- If possible, it should be able to integrate with the external user-defined modules written in C or in some popular high level programming languages.
- The user should be able to define the sequence of states through which a created prototype can run. The user should also be allow to control the running of the prototype.

- The run-time system of the prototype should support mock up run of the actual system and management of the input and output data.

2. Structured Analysis and Design: Several diagramming techniques are used for structured analysis and design. A CASE tool should support one or more of the structured analysis and design techniques. It should support effortlessly making of the analysis and design diagrams. It should also support making of the fairly complex diagrams and preferably through a hierarchy levels. The CASE tool should provide easy navigation through different levels of design and analysis.

3. Code Generation: As far as code generation is concerned, the general expectation from a case tool is quite low. A reasonable requirement is traceability from source file to design data. More pragmatic support expected from a CASE tool during the code generation phase comprises the following:

- The CASE tool should support generation of module skeletons or templates in one or more popular programming languages. It should be possible to include copyright message, brief description of the module, author name and date of creation in some selectable format.
- The tool should generate records, structures, class definitions, 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.
- The tool should generate code for user interface from prototype definition for X-Windows, and MS-Windows-based applications.

4. 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.

CHARACTERISTICS OF SOFTWARE MAINTENANCE

S/w maintenance is becoming an important activity of a large number of organizations. This is no surprise, given the rate of h/w obsolescence, the immortality of a s/w product, and the demand of the user community to see the existing s/w products run on newer platforms, run in newer environments, and/or with enhanced features. Also, whenever the support environment of a s/w product changes, the s/w product requires rework to cope with the newer interface. Thus, every s/w product continues to evolve after its development through maintenance efforts.

1. Types of s/w maintenance : The requirement of s/w maintenance arises on account of the three main reasons :

Corrective : Corrective maintenance of a s/w product becomes necessary to rectify the bugs observed while system is in use.

Adaptive : A s/w 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 be interfaced with new h/w or s/w.

Perfective : A s/w 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.

2. Characteristics of S/W Evolution : Lehman and Belady have studied the characteristics of evolution of several s/w products. They have observed in the form of laws.

Lehman's first law : A s/w product must change continually or become progressively less useful.

Lehman's Second Law : The structure of a program tends to degrade as more and more maintenance is carried on it.

Lehman's Third Law : over a program's lifetime, its rate of development is approximately constant.

3. Special Problems Associated with S/W Maintenance :

- During maintenance it is necessary to thoroughly understand someone else's work and then carry out the required modifications and extensions.
- Another problem associated with maintenance work is that the majority of s/w products needing maintenance are legacy(aged) products.

SOFTWARE REVERSE ENGINEERING

S/W 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 to facilitate maintenance work by improving the understandability of a system and to produce the necessary documents for a legacy system. The reverse engineering is becoming important, since legacy s/w products lack proper documentation, and are highly unstructured. Even well-designed products becomes legacy s/w as their structure degrades because of a series of maintenance efforts implemented a period of use.

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. A program can be formatted using any of the several available pretty-printer programs which layout the program neatly. Many legacy s/w products are difficult to comprehend with complex control structure and unthoughtful variable names. All variables, data structures, and functions should be assigned meaningful names wherever possible. Complex nested conditionals in the program can be replaced by simpler conditional statements or whenever appropriate by 'case' statements.

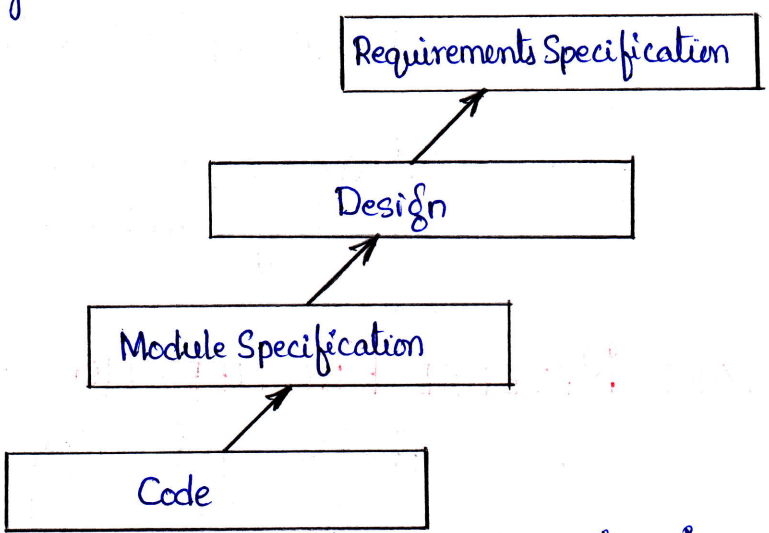


Fig: A process model for reverse engineering

After the cosmetic changes have been carried out on a legacy s/w, the process of extracting the code, design and the requirements specification can begin. Some automatic tools can be used to derive the data flow and the control flow diagram from the code. The structure chart (module invocation sequence and data interchange among modules) should also be extracted. The SRS document can be written once the full code has been thoroughly understood and the design extracted.

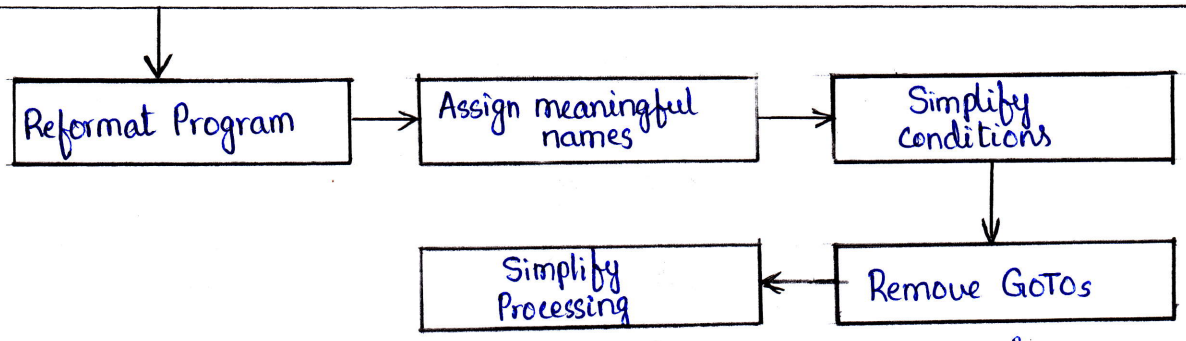


Fig: Cosmetic changes carried out before reverse engineering

SOFTWARE MAINTENANCE PROCESS MODELS

The activities involved in a s/w maintenance project are not unique and depend on several factors such as : (i) the extent of modification to the product required, (ii) the resources available to the maintenance team, (iii) the expected project risks etc. When the changes needed to a s/w product are minor and straightforward the code can be directly modified and the changes approximately, reflected in all the documents.

However, more elaborate activities are required when the required changes are not so trivial. Usually, for complex maintenance projects for legacy systems, the s/w process can be represented by a reverse engineering cycle followed by a forward engineering cycle with an emphasis on as much reuse as possible from the existing code and other documents.

Since the scope for different maintenance projects vary widely, no single maintenance process model can be developed to suit every kind of maintenance project. However, two broad categories of process models can be proposed. The first model is preferred for projects involving small reworks where the code is changed directly and the changes are reflected in the relevant document later.

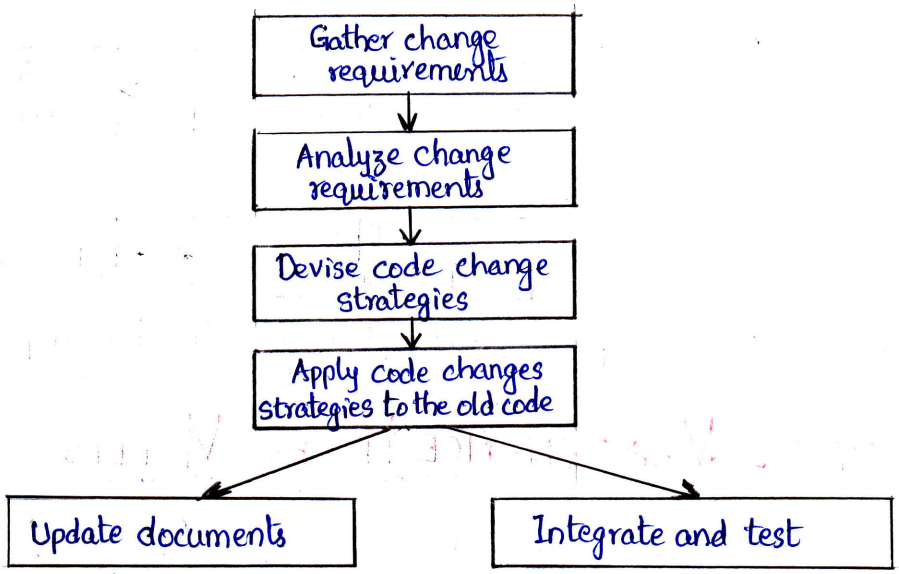


Fig: Maintenance process model-1

In this approach, the project starts by gathering the requirements for changes. The requirements are next analyzed to formulate the strategies to be adopted for code change. At this stage, the association of at least a few members of the original development team goes a long way in reducing the cycle time, especially for projects involving unstructured and inadequately documented code. The availability of a working old system to the maintenance engineers at the maintenance site greatly facilitates the task of the engineers as the get a good insight into the working of the old system and also can compare the working of their modified system with the old system

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 s/w reengineering. The reverse engineering cycle is required for legacy products. During the reverse engineering, the old code is analyzed to extract the module specifications. The module specifications are then analyzed to produce the design. The design is analyzed to produce the original

requirements specification. The change requests are then applied to this requirements specification to arrive at the new requirements specification. At this point a forward engineering is carried out to produce the new code. At the design, module specification, and coding stages, a substantial reuse is made from the reverse engineered products. An important advantage of this approach is that it produces a more structured design than what the original product, produces good documentation, and very often results in increased efficiency. However, this approach is costlier than the first one. An empirical study indicates that process 1 is preferred when the amount of re-work is no more than 15%. Besides the amount of rework, several other factors might affect the decision regarding using model 1 over process 2.

- Reengineering might be preferable for products which exhibit a high failure rate.
- Reengineering might also be preferable for legacy problems having poor design and code structure.

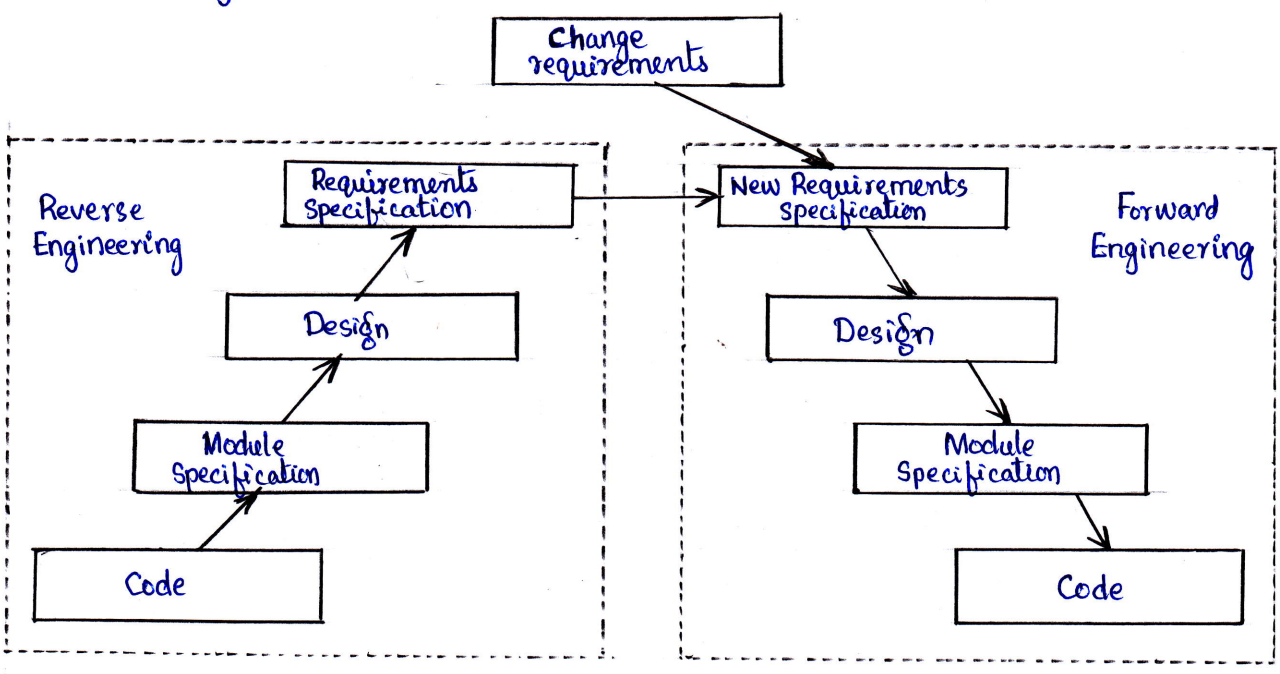


Fig: Maintenance Process Model-2

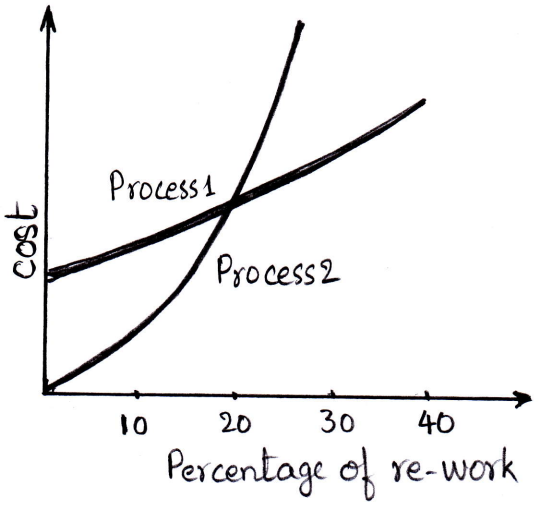


Fig. Empirical estimation of maintenance cost Vs. percentage rework

Estimation of Maintenance Cost

Maintenance efforts constitute about 60% of the total life cycle cost for a typical s/w product. However, maintenance costs vary widely from one application domain to another. For embedded systems, the maintenance cost can be as much as 2 to 4 times the development cost.

Boehm 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 quality called the Annual Change Traffic (ACT). Boehm defined ACT as the fraction of a s/w product's source instructions which undergo change during a typical year either through addition or deletion.

$$ACT = \frac{KLOC_{added} + KLOC_{deleted}}{KLOC_{total}}$$

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 the code deleted.

The ACT is multiplied with the total development cost to arrive at the maintenance cost: maintenance cost = ACT x development cost.

Most maintenance cost estimation models, however, give approximate results because they do not take into account several factors such as the experience level of engineers, and familiarity of engineers with the product, h/w requirements, s/w complexity, etc.

BASIC ISSUES IN ANY REUSE PROGRAM

S/w products are expensive. S/w project managers are worried about the high cost of s/w development and are desperately looking for ways to cut development costs. A possible way to reduce the development cost is to reuse parts from the previously developed s/w. In addition to achieving reduced development cost and time, reuse also ensures a high quality of the developed products since the reusable components have already stood the test of reliability.

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

1. **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.

2. **Compont 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.

3. **Component searching:** The programmers need to search for right components by matching their requirements with components stored in a database. To be able to search components efficiently, the programmers required a proper method to describe the components that they are looking for.

4. **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. To facilitate understanding the components should be well documented and should do something simple.

5. Component adaptation : Often, the components may need adaptation before they can be reused, since a selected component not exactly fit the problem at hand. However, tinkering with the code is also not a satisfactory solution because this is very likely to be a source of bugs.

6. Repository maintenance : A component repository once created, requires continuous maintenance. New components, as and when created into the repository. The faulty components have to be tracked. Further, when new applications emerge, the older applications become obsolete. In this case, the obsolete components might have to be removed from the repository.

A Reuse Approach

A promising approach that is being adopted by many organizations is to introduce a building block approach into the s/w development process. For this, the reusable components need to be identified after every development project is completed. The reusability of the identified components has to be enhanced and these have to be catalogued into a component library.

1. Domain Analysis : The aim of domain analysis is to identify the reusable components for a problem domain.

Reuse domain. A reuse domain is a technically related set of application areas. A body of information is considered to be a problem domain for reuse, if a deep and comprehensive relationship exists among the information items as characterized by patterns of similarity among the development components of the s/w product.

Evolution of reuse domain : The ultimate result of domain analysis is the development of problem-oriented languages. These are also known as application generators. These application generators, once developed from application development standards. The domains slowly develops, we may distinguish the various stages it undergoes :

Stage 1 : There is no clear and consistent set of notations. Obviously, no reusable components are available. All s/w is written from scratch.

Stage 2 : Here, only experience from similar projects is used in a new 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 stabilized 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 s/w development for domain can be largely 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.

2. Component Classification : Components need to be properly classified in order to develop an effective indexing and storage scheme. At the lowest level, the components are described in several forms : natural language description, logic schema, timing information, etc. The higher the level at which a component is described, the more is the ambiguity. This has motivated the Prieto-Diaz's classification scheme.

Prieto-Diaz's Classification Scheme : Each component is best described using a number of different characteristics or facets. For example, objects can be classified using the following:

- Actions they embody
- Objects they manipulate
- Data structures used
- Systems they are part of, and so forth

Prieto-Diaz's faceted classification scheme requires choosing an n-tuple that best fits a component. Faceted components has advantages over the enumerative classification.

3. Searching : A popular search technique that has proved to be very effective is the one that provides a web interface to the repository. Using such a web interface, one would search for an item using an approximate automated search and using keywords, and then from these results do a browsing using the links provided to lookup the related items.

4. Repository Maintenance : Repository maintenance involves entering new items which are no more necessary, and modifying the search attributes of items to improve the effectiveness of search. Also, the links relating the

different items may need to be modified to improve the effectiveness of search.

5. Reuse without Modifications: Once the standard solutions emerge, no modifications to the program parts may be necessary. One can plug in the parts directly to develop one's application. Reuse without modification is much more useful than the classical program libraries. These can be supported by compilers through linkage to run-time support routines (application generators).

Reuse At Organization Level

Reusability should be a standard part in all s/w development activities including specification, design, implementation, test, etc. Ideally, there should be a steady flow of reusable components. In practice, things are not so simple.

Extracting reusable components from projects that were completed in the past presents a real difficulty not encountered while extracting a reusable component from an ongoing project - typically, the original developers are no longer available for consultation. Development of new systems can lead to an assortment of products, since reusability ranges from items whose reusability is immediate to those items whose reusability is highly improbable.

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

- Assessing a product's potential for reuse.
- Refining a product for greater reusability.
- Improving reusability of a product by handling portability problems.

Assessing a product's potential for reuse: Assessment of a component's 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 domains can be asked to answer the questionnaire about the product's reusability. Depending on the answers given by the programmers, either the component is

taken up for reuse as it is, or it is modified and refined before it is entered into the reuse repository, or it is ignored. A sample questionnaire to assess a component's reusability as follows:

- Would the component's functionality be 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 h/w dependent?
- Is the design of the component optimized enough?
- If the component is non-reusable, then can it be decomposed to yield some reusable components?
- Can we parametrize a non-reusable component so that it becomes reusable?

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. The names should be general, rather than being directly related to a specific application.

Operation generalization. Operations should be added to make the component more general. Also, operations that are too specific to an application can be removed.

Exception generalization. This involves checking each component to see which exceptions it might generate. For a general component, several types of exceptions might have to be handled.

Handling portability Problems: Programs typically make some assumptions regarding the representation of information in the underlying machine. These assumptions are in general not true for all machines. The programs also often need to call some operating system functionality and these calls may not be the same on all machines. Also, programs use some function libraries, which may not be available on all host machines. The portability solution suggests that rather than call the operating system and I/O procedures directly, abstract versions of these should be called by the application program. Also, all platform-

related calls should be routed through the portability interface. One problem with this solution is the significant overhead incurred, which makes it inapplicable to many real-time systems and applications requiring very fast response.

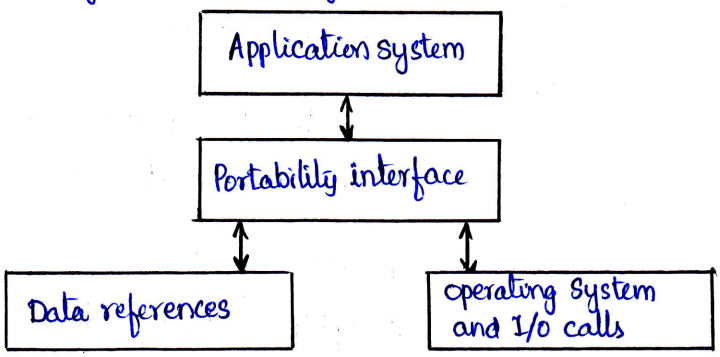


Fig: Improving reusability of a component by using a portability interface

Current State of Reuse: In spite of all the short comings of the state-of-the art reuse techniques, it is the experience of several organizations that most of the factors inhibiting an effective reuse program are non-technical. Some of these factors are the following:

- Need for commitment from the top management
- Adequate documentation to support reuse
- Adequate incentive to reward those who reuse. Both the people contributing new reusable components and those reusing the existing components should be rewarded to start a reuse program and keep it going.
- Providing access to and information about reusable components. Organizations are often hesitant to provide an open access to the reuse repository for the fear of the reuse components finding a way to their competitors.

— * END OF UNIT-5 * —