BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 6

DCA3245

SOFTWARE PROJECT MANAGEMENT

Unit 8

Configuration Management

Table of Contents

1. INTRODUCTION

In the last unit, we have discussed risk management. In this unit, let's discuss configuration management, which is an important activity in project development. Change is an integral part of the software development process; means there will always be changes that needed to be done in the system being developed for various reasons. As these changes are brought in to the project, this leads to lot of confusion to the team members if the changes are not handled properly. By handling we mean that the changes are to be properly recorded, analyzed, and the decision on the change should be properly communicated in order that the changes are implemented in a controlled manner which will improve the quality of the product and reduce the error. In this unit, we shall learn how to record various software changes.

1.1 Objectives:

After studying this unit, you should be able to:

- *Explain software configuration management process*
- *Identify objects in software configuration*
- *Describe version control and change control*
- *Discuss configuration audit and status reporting*
- *List goals of software configuration management*

Evaluation Process: A Large sums of money are spent on software systems because of how important they are to running a company. These assets are essential to the survival of the company, thus keeping them fresh and current is essential. Most of a company's software spending goes towards maintenance and upgrades rather than brand new software development. An iterative cycle of software updates is depicted by the "spiral model" of development and evolution.

Changes in the system are driven by proposed modifications. These should be connected to parts of the system that will be impacted by the modification, allowing for an assessment of the modification's financial and operational consequences. Over the course of a system's existence, change detection and evolution will continue to occur.

The processes of software evolution rely on:

1. Type of maintained software;
2. Methods of development employed;
3. Competence and experience of those involved.

Implementing a change is similar to going through a development cycle when changes are made and tested in small increments. The first step in implementing a modification may include learning the programme, especially if the original system's creators aren't in charge of the project. At this stage, you should have a firm grasp on the program's structure, the functionality it provides, and the potential effects of the proposed change.

Agile Methods:

Regression testing that is automated is especially helpful when modifications have been made to a system. It's possible to communicate modifications by adding user stories.

The transition from development to evolution is seamless since agile methods are based on incremental development. Simply said, evolution is the ongoing process of system development that results from regular updates.

Program evaluation dynamics: Lehman and Belady suggested a set of universal 'rules' of system evolution after reviewing the results of a number of large-scale empirical research. Instead of rules, there are commonsense observations. They can be used with the systems that huge corporations design and implement.

Computer programme evolution dynamics is the study of how computer programmes change over time. Due to environmental changes, the system's requirements are likely to evolve during development, making it likely that the final product may fall short of expectations. There is a strong coupling between systems and their surroundings. When a system is installed, it alters its surrounding environment, which in turn necessitates adjustments to the system's specifications. In order for a system to continue serving its purpose, it must undergo some sort of modification.

Software Maintenance: During routine upkeep, the underlying structure of the system is rarely altered significantly. The system's current parts are altered, and new ones are added, to bring about the desired effect.

After a programme has been deployed into production, its maintenance phase focuses on making adjustments to it. The word is most commonly associated with modifying proprietary software. It is commonly held that generic software products evolve into newer versions throughout time. It is commonly held that generic software products evolve into newer versions throughout time. Used to modify tailor-made software

Types of software maintenance involved:

Maintenance to add to or modify the system's functionality: modifying the system to satisfy new requirements.

- Maintenance to repair software faults: changing a system to correct deficiencies in the way meets its requirements.
- Maintenance to adapt software to a different operating environment: changing a system so that it operates in a different environment (computer, OS, etc.) from its initial implementation.
- Maintenance to add to or modify the system's functionality: modifying the system to satisfy new requirements.

Maintenance Costs: The cost of maintenance typically exceeds the initial cost of development by a factor of two to one hundred. As software is maintained, costs rise due to both technical and non-technical issues. When software is maintained, its structure is corrupted, which makes future maintenance more difficult. It can be expensive to maintain older software (such as obsolete programming languages, compilers, etc.).

Factors that affect maintenance costs include:

- Long-term involvement from the same staff members reduces maintenance costs.
- There is no incentive to plan for future change if the developers of a system have no contractual responsibility for maintenance.

- Maintenance workers typically lack experience and broad domain knowledge.
- Age and structure in programmes: when programmes age, their structure deteriorates, making them more difficult to understand and update.

Factors Influencing Maintenance Prediction: Predictive maintenance analyses potential trouble spots and costly wear and tear in a system. Counting changes demands familiarity with the interconnections between a system and its surroundings. When the external environment shifts, adjustments must be made to the tightly coupled system.

Influencing this connection are:

- how many and how complicated the system's interfaces are;
- Variability in the number of system requirements;
- Where in the company the system is put into action.

Predictions of maintain ability:

The complexity of the system's parts can be evaluated to make predictions about its maintainability. Research has demonstrated that only a few of system parts require the majority of maintenance time and resources.

- Assessing the complexity of the system's components allows for maintenance feasibility judgements to be made.
- Research has shown that a comparatively small number of system components get the bulk of the maintenance effort.
- An object's, a procedure's, or a module's complexity will vary depending on a number of factors.

Process Metrics:

It may be used to assess maintainability; if any or all of these is increasing, this may indicate a decline in maintainability: Maintainability can be evaluated with process metrics; an increase in any of these metrics may point to a problem with the system's ability to be easily maintained.

- Requests for preventative maintenance;
- Median impact analysis runtime;
- How long it typically takes to make a change;
- The total number of outstanding change orders.

System Re-Engineering in Software Development

Redesigning, rebuilding, or enhancing an existing software system to boost performance, maintainability, scalability, or other attributes without altering its core functionality is what is meant by "system re-engineering," also known as "software system re-engineering" or "system modernization." Typical motivations for such action include the need to update outmoded hardware or software, improve reliability, tighten security, or adapt to shifting organisational needs. In the context of long-lived software systems, system re-engineering is a complex but frequently essential activity. It is common practise to do so when the original system has become obsolete, ineffective, or difficult to maintain and a replacement must be found.

Re-Engineering process includes the following activities:

- **Source code translation:** As part of this process, code written in one programming language will be translated into another. When moving a project to a new platform or environment, this can be helpful.
- **Reverse engineering:** Understanding the structure, functionality, and logic of a programme is the goal of this exercise. It's common practise to do this when official documentation is scarce and you want to learn more about how a system functions.
- **Program structure improvement:** This is the process of reorganizing a program's code automatically such that it is easier to read, modify, and understand. This can simplify the codebase for developers.
- **Program modularization:** Modularization is the process of restructuring a programme into smaller, more manageable modules or components. The modularization of code can increase its reusability and ease of maintenance.

- **Data reengineering:** Data in the system is being cleaned up and reorganised as part of this task. Data validation, normalisation, transformation, and migration are all possible steps in this process, with the goal of improving data quality and uniformity..

Refactoring: Refactoring is a technique in software engineering for enhancing the architecture and structure of a codebase without altering its behaviour. The purpose of refactoring is to improve the readability, maintainability, and performance of a program's source code. It is a methodical process for removing clutter from code in order to make it more readable and manageable.

- Refactoring is the process of enhancing a programme in order to reduce its deterioration as a result of evolution.
- You might conceive of remodelling as "preventative maintenance" that lessens future change problems.
- Refactoring a programme is not the time to add new features; instead, focus on making the existing ones better.
- When the cost of maintaining a system steadily rises over time, it may be time for a reengineering.

Bad smells' of code: A Stereotypical situations in which a program's code can be improved through refining are known as "bad smells of code." Code that is same or almost identical in multiple places in a programme can be easily removed and implemented as a single method or function that is called only when necessary. Code of the program can be improved through refactoring: The following are some of the steps to be followed to eliminate the concept of bad smell of a code

- Long Methods: methods that take too long should be broken down into several shorter ones. Methods and functions that are too long can be challenging to understand and keep up to date. It's recommended to split up lengthy procedures into simpler ones.
- Similarly, classes or modules that are overly big can be cumbersome to work with. They might be spreading themselves too thin and would benefit from being broken up into more manageable pieces. Similarly, classes or modules that are overly big can be

cumbersome to work with. They might be spreading themselves too thin and would benefit from being broken up into more manageable pieces.

- Switch (case) statements frequently involve duplication, as the switch may depend on the type of a value or be scattered throughout a programme. The same effect can be achieved through the use of polymorphism in object-oriented languages.
- Data clumping occurs when the same set of information appears many times in a programme (for example, the same set of fields in a class or the same set of parameters in multiple methods). In many cases, these can be replaced with a single object that contains all the necessary information.

Legacy system Management: Legacy system management entails keeping outmoded hardware and software running at peak performance for as long as possible. These systems may be antiquated, but they are usually embedded in the procedures of a business, making their replacement impractical or expensive. Organizations that rely on legacy systems must choose a strategy for evolving these systems.

The chosen strategy should depend on the system quality and its business value:

- Low quality, low business value: should be scrapped.
- Low-quality, high-business value: make an important business contribution but are expensive to maintain. Should be re-engineered or replaced if a suitable system is available.
- High-quality, low-business value: replace with COTS, scrap completely, or maintain.
- High-quality, high business value: continue in operation using normal system maintenance.

2. SOFTWARE CONFIGURATION MANAGEMENT (SCM)

Babich explains software configuration management as follows:

> *Software configuration management* (SCM) is an umbrella activity that is applied throughout the software process. Because change can occur at any time, SCM (software configuration management) activities are developed to
>
> (1) Identify change
> (2) Control change
> (3) Ensure that change is being properly implemented
> (4) Report changes to others who may have an interest.

Before we look into details of SCM, we should have clear distinction between the software configuration management and the software support. When a software product is released, the product needs to be supported after it is delivered to customers. The support activity can be in terms of helping the end user to properly in using the application, resolving a defect, preparing and releasing a patch release to stabilize the product etc. On the other hand, configuration management is something that is applied when a product is developed. It could be the changes to the specification of the product to be developed, or change in technology etc. In essence, the SCM can be thought of integral part of development project.

Because many work products are produced when software is built, each must be uniquely identified. Once this is accomplished, mechanisms for version and change control can be established. In order to make sure that the quality constraints are met as the changes are made, the process is audited and to ensure that those with a need to know are informed about changes, reporting is conducted.

The most important objective of software engineering practice is to ensure that process maturity is reached in the project being carried out so that any changes that come along are easily incorporated with minimal effort.

The artifacts produced by software development processes can be broadly classified into three categories. (1) Computer program (both in source code level and executable form) that the models the system being developed. (2) The documents that describe the system, the

design and testing mechanism and any other supporting information (3) the data – which is part of the software or external to the software.

Thus all the items that carry certain information about the product being developed and that are produced as part of software processes are known as *software configuration items* (***SCI***). It is pretty clear that as the project progresses, the number of software configuration items grow rapidly. For example, at the beginning of the project, only Scope and Vision document would describe the system at the very high level. This document becomes the basis to further elaboration of Project plan and the Software Requirement specification as the project progresses.

Change is an integral part of software development and can happen at any phase of the project. As endorsed by the first law of system engineering, the change can occur at any stage of product life cycle

There are six fundamental sources of change.

- Changes in the Business dynamics requires the changes in product being developed.
- Change in the customer needs requires the corresponding modifications to the product.
- Different functionality needed from the product being developed.
- Project priorities are modified due to organizational changes or re-structuring
- Some changes in the team composition forces change in strategy for development of software product.
- Change in scope of the project due to financial or time constraint.

As the changes are prominent throughout the life cycle of the project, the Software configuration management aims to manage the changes in an orderly fashion. SCM is a set of activities that have been developed to manage change throughout the life cycle of computer software. SCM can be considered as a software quality assurance exercise that is applied throughout the software process. In the sections that follow, we examine major SCM tasks and important concepts that help us to manage change.

2.1 Baselines

Change is un-avoidable part of the software development life cycle. Change originates from every key player in the project starting from customers to the developers who are working on the project. The examples of changes can be that Customers want to change the requirement, and change the scope of the project, the manager might want to change the schedule, strategy of the project, the developers might want to change the technology used to implement the product and so on. If we try to find the answer to question that why so many changes originate during the course of the project, we can see that as the project goes through different phases, the team becomes more knowledgeable on the product being developed. This additional knowledge makes the team to define the items more clearly hence causing the changes to the originally stated requirements/standards, or processes etc. In a way, every change ends up being justified with the grounds of having more clearer picture on the overall system being developed.

The Baseline is a SCM concept that helps to implement the justifiable change in a non-destructive fashion. Baseline gives the snapshot of the previously agreed upon state of the project with collection of artifacts (source code, documents etc.) produced so far. The change to be implemented is compared against the baseline to know the impact of the change and the quantum of effort involved to implement the change. The IEEE defines the baseline as:

A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures.

Any processing of the change should be carried out against the baseline. During the development of a software product, if the development of a work item (code or document or any other artifact) is still under the sub-team working on the item the changes can be done internally and informally. But once the work item is made part of the overall system repository, and base lined after review, any change to the work item should be done through software configuration processes described later in the chapter.

Activities during this process:

- Facilitate construction of various versions of an application
- Defining and determining mechanisms for managing various versions of these work products
- The functional baseline corresponds to the reviewed system requirements
- Widely used baselines include functional, developmental, and product baselines

2.2 Software configuration items Identification (SCI)

We already know that the software configuration item (SCI) refers to the artifacts produced through the software processes during the course of the project. The SCI can be documents like SRS (software requirement specification), design document, or test plan, or it can be the source code like a Java package. It can also be the Data modeling scripts used for the project.

Additionally, the SCI's might include the tools used in the project also. For example, if a project is using a particular version of Requirement gathering tool (like Rational Rose) the team would like to carry out the entire project using the same across all the sub-teams of the project so that the consistency in deliverables are maintained. In such scenario, the CASE tool being used is also treated as the SCI so that tool and version is recorded in the project database for consistency.

In practice, the software configuration items are maintained as the **configuration objects** in the project repository in the form of catalog. By catalog, we mean each item is version and any relation with other configuration object is also recorded in the repository. Figure 8.1 refers to a particular configuration objects arrangement in some project. As we can see, the configuration object will have name, attribute and is connected to other objects by some relationship. Between two configuration objects we might have composition, aggregation, or interrelation (bidirectional) relationships or there may be no apparent relation between two objects treating them as independent of each other. Representing these relationships become important during the change management. For example, from the diagram we can see that if there is any change done to the **Component N**, then **Test Specification** object will also have to be modified accordingly.

Activities during this process:

- Identification of configuration Items like source code modules, test case, and requirements specification.
- Identification of each CSCI in the SCM repository, by using an object-oriented approach
- The process starts with basic objects which are grouped into aggregate objects. Details of what, why, when and by whom changes in the test are made

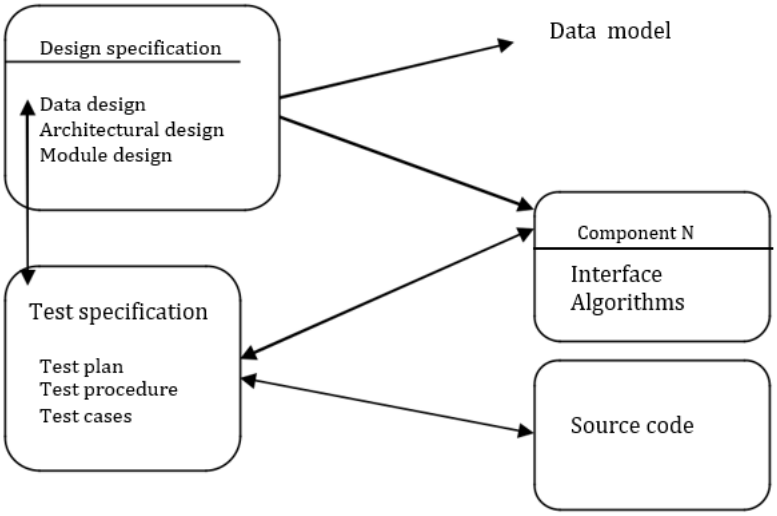

Fig. 8.1: Configuration Objects

SELF-ASSESSMENT QUESTIONS - 1

1. Software Configuration Management (SCM) is an umbrella activity that is applied throughout the software process. (True / False)
2. A _____ becomes the basis to further elaboration of Project plan and the Software Requirement specification as the project progresses.
3. A _____ has a name, attributes, and is "connected" to other objects by relationships.

3. SCM PROCESS

Software Configuration Management (SCM) is essentially a mechanism of controlling the change in the software project. It can also be considered as one of the quality assurance exercises undertaken by the project. In order to control the change the SCM must also be capable of the identifying the artifacts that need to controlled in the project. In other words, the SCM will also include the activity of identifying the SCI's, their versions, and also audit information regarding the SCI's to get the history of any changes that SCI would have undergone. Typical SCM activities would be based on following complex questions:

- How should the organization identify the different version of a SCI and what should be the mechanism to propagate the change through these different versions?
- What should be the minimum set of processes need to be followed before the software is delivered to the customer?
- What should be the change approval mechanism and hierarchy?
- How to ensure that the changes are implemented as expected?
- How should the changes communicated to others in the team so that entire team is aware of the current change?

4. IDENTIFICATION OF OBJECTS IN THE SOFTWARE CONFIGURATION

In order to control and manage the SCI's, each item must be separately identified. For identification we need a separate name and versioning approach. For this purpose, we can follow object oriented representation approach representing the SCI's as objects. The objects are classified into two categories: *basic object* and the *aggregate objects*. As the name indicates, the basic object corresponds to some work item created by a software developer. A section of requirement document, a section of code etc. are examples of basic object. Aggregate object is collection of basic objects. Looking at Fig 8.1 we can see that **Component N** is a basic object whereas the **Design Specification** is an aggregate object.

Each object has a set of distinct features that identify it uniquely: a name, a design, a list of resources, and a "realization." The object name is a character string that identifies the object unambiguously. The object description is a list of data items that identify the SCI type (e.g., document, program, and data) represented by the object project identifier change and/or version information. Resources are entities that are provided, processed, referenced or otherwise required by the object. For example, data types, specific functions, or even variable names may be considered to be object resources. The realization is a pointer to the "unit of text" for a basic object and null for an aggregate object.

Configuration object identification must also consider the relationships that exist between named objects. An object can be identified as **<part-of>** an aggregate object. The relationship **<part-of>** defines a hierarchy of objects. For example, using the simple notation we can create a hierarchy of SCIs.

E-R diagram <part-of> data model;

Data model <part-of> design specification;

It is unrealistic to assume that the only relationships among objects in an object hierarchy are along direct paths of the hierarchical tree. In many cases, objects are interrelated across branches of the object hierarchy. For example, a data model is interrelated to data flow diagrams (assuming the use of structured analysis) and also interrelated to a set of test cases

for a specific equivalence class. These cross structural relationships can be represented in the following manner:

Data model < interrelated > data flow model;

Data model < interrelated > test case class m;

In the first case, the interrelationship is between a composite object, while the second relationship is between an aggregate object (**data model**) and a basic object (**test case class m**). The interrelationships between configuration objects can be represented with a *module interconnection language* (MIL). A MIL describes the interdependencies among configuration objects and enables any version of a system to be constructed automatically.

The identification scheme for software objects must recognize that objects evolve throughout the software process. Before an object is baseline, it may change many times, and even after a baseline has been established, changes may be quite frequent. It is possible to create an evolution graph for any object. The *evolution graph* describes the change history of an object, as illustrated in figure 8.2. Configuration object 1.0 undergoes revision and becomes object 1.1. Minor corrections and changes result in versions 1.1.1 and 1.1.2, which is followed by a major update that is object 1.2. The evolution of object 1.0 continues through 1.3 and 1.4, but at the same time, a major modification to the object results in a new evolutionary path, version 2.0 both versions are currently supported.

Changes may be made to any version, but not necessarily to all versions. How does the developer reference all components, documents, and test cases for version 1.4? How does the marketing department know what customers currently have version 2.0 or 1.0? How can we be sure that changes to the version 2.1 source are properly reflected in the corresponding design documentation? A key element in this answer to all these questions is identification.

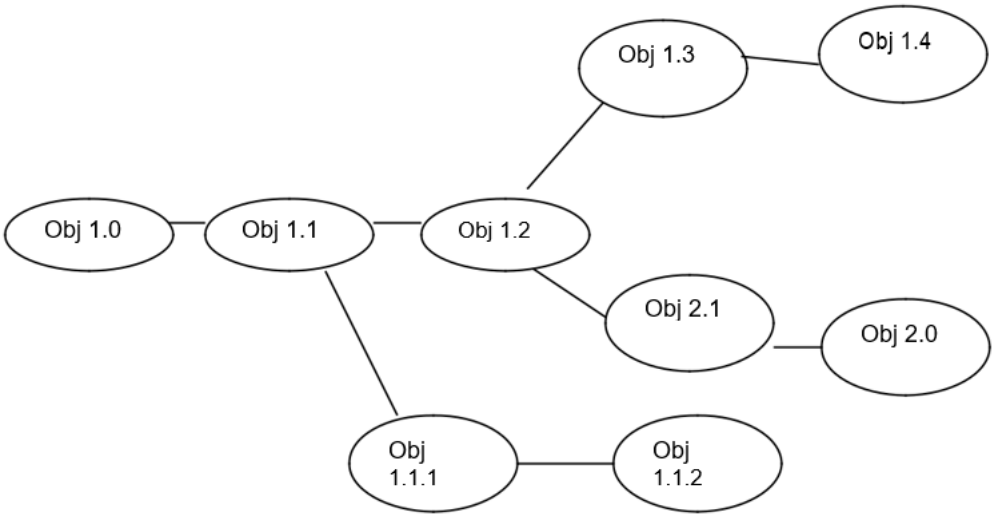

Fig. 8.2: Evolution graph showing object's change history

A variety of automated SCM tools has been developed to aid in identification (and other SCM) tasks. In some cases, a tool is designed to maintain full copies of only the most recent version. To achieve earlier versions (of documents or programs) changes (cataloged by the tool) are "subtracted" from the most recent version. Scheme makes the current configuration immediately available and allows other versions to be derived easily.

5. VERSION CONTROL

Version *control* combines procedures and tools to manage different versions of configuration objects that are created during the software process. Clam describes version control in the context of SCM as follows:

Configuration management allows a user to specify alternative configurations of the software system through the selection of appropriate versions. This is supported by associating attributes with each software version, and then allowing a configuration to be specified [and constructed] by describing the set of desired attributes. These "attributes" mentioned can be as simple as a specific version number that is attached to each object or as complex as a string of Boolean variables (switches) indicating specific types of functional changes that have been applied to the system.

One representation of the different versions of a system is the evolution graph presented in figure 8.2. Each node on the graph is an aggregate object, that is, a complete version of the software. Each version of the software may be composed of different variants.

To illustrate this concept, consider a version of a simple program that is composed of entities 1, 2, 3, 4, and 5. Entity 4 is used only when the software is implemented using color displays. Entity 5 is implemented when monochrome displays are available. Therefore, two variants of the version can be defined: (1) entities 1, 2, 3, and 4; (2) entities 1, 2, 3, and 5.

To construct the appropriate *variant* of a given version of a program, each entity can be assigned an "attribute-tuple" – a list of features that will define whether the entity should be used when a particular variant of a software version is to be constructed. One or more attributes is assigned for each variant. For example, a color attribute could be used to define which entity should be included when color displays are to be supported.

Another way to conceptualize the relationship between entities, variants and versions (revisions) is to represent them as an *object pool*. Referring to figure 8.3, the relationship between configuration objects and entities, variants and versions can be represented in a three-dimensional space. An entity is composed of a collection of objects at the same revision level. A variant is a different collection of objects at the same revision level and therefore

coexists in parallel with other variants. A new version is defined when major changes are made to one or more objects.

A number of different automated approaches to version control have been proposed over the past decade. The primary difference in approaches is the sophistication of the attributes that are used to construct specific versions and variants of a system and the mechanics of the process for construction.

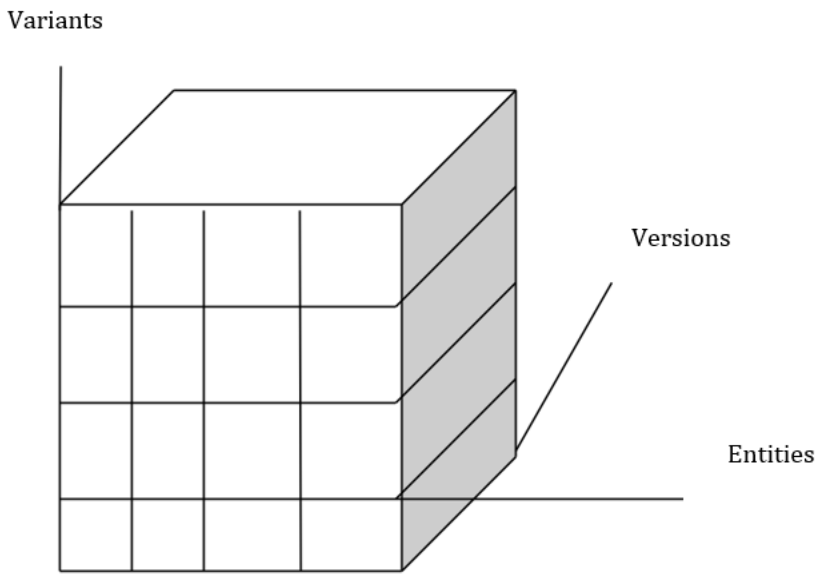

Fig. 8.3: Object pool representation- Versions, Entities and Variants

SELF-ASSESSMENT QUESTIONS - 2

4. Software configuration management is not an element of software quality assurance. (True / False)
5. Two types of objects that can be identified in configuration management are _____ and _____.
6. _____ combines procedures and tools to manage different versions of configuration objects that are created during the software process.

6. CHANGE CONTROL

The reality of *change control* in modern software engineering context has been summed up beautifully by James Bach as follows:

Change control is vital. But the forces that make it necessary also make it annoying. We worry about change because a tiny error in the code can create a big failure in the product. But it can also fix a big failure or enable wonderful new capabilities. We worry about change because a single developer could sink the project; yet brilliant ideas originate in the minds of those people, and a burdensome change control process could effectively discourage them from doing creative work.

Bach recognizes that we face a balancing act. Too much change control creates lots of problems. For a large software engineering project, uncontrolled change rapidly leads to chaos. For such projects, change control combines human procedures and automated tools to provide a mechanism for the control of change. The change control contains the following activities:

1) A *change request* is submitted and evaluated to assess technical merit, potential side effects, overall impact on other configuration objects and system functions, and the projected cost of the change.
2) The results of the evaluation are presented as a change report, which is used by ***a Change Control Authority (CCA) –*** a person or group who makes a final decision on the status and priority of the change.
3) An ***Engineering Change Order (ECO)*** is generated for each approved change. The ECO describes the change to be made, the constraints that must be respected, and the criteria/or review and audit. The object to be changed is "die out" of the project database, the change is made, and appropriate SQA activities are applied. The object is then "checked in" to the database and appropriate version control mechanisms are used to create the next version of the software.

The "check-in" and "check-out" process implements two important elements of change control – access control and synchronization control. Access *control governs* which software engineers have the authority to access and modify a particular configuration object. *Synchronization* control helps to ensure that parallel changes, formed by two different people, don't overwrite one another. Access and synchronization control flow are illustrated schematically in figure 8.4. A software engineer checks out a configuration project based on the approved change request and ECO.

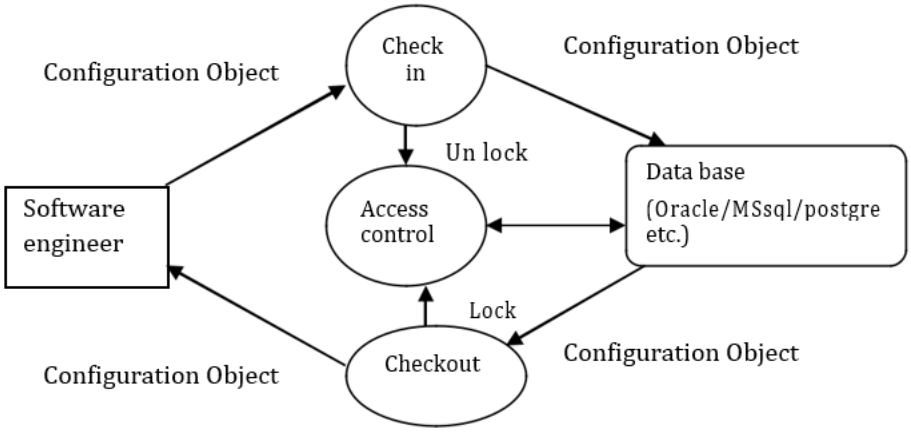

Fig. 8.4: Flow of access and synchronization control

An access control function ensures that the software engineer has authority to check out the object, and synchronization control *LOCKS* the object in the project database so that no updates can be made to it until the currently check out version has been replaced. Note that other copies can be checked-out, but other updates cannot be made. A copy of the baseline object, called the extracted version is modified by the software engineer. After appropriate SQA and testing, the modified version of the object is *checked* in and the new baseline object is unlocked.

Without proper maintains, change control can causes the delay of graft and creates unnecessary red tape. To avoid the problems of change control, most of software developers have formed a number of layers of change control mechanism.

Before SCI becoming a baseline, it should be essential to apply informal change control. The developers of the Software configuration object (SCI) defines the changes which are discussed and justified by project and technical requirements (this change not effects expansive system requirements; it stays outside the developer possibility work). After defining the SCI object, it has been technically reviewed and then creates approved baseline. Once an SCI become baseline, project level change control is implemented. If change is local, the developer should take approval from project manager to implement that change. If change affects other SCIs, developer should take approval from CCA.

Configuration status accounting activities:

Keeping tabs on the current and past states of configuration items (CIs) is what configuration status accounting (CSA) is all about. CSA is an integral part of configuration management (CM). The major objective of CSA is to document all CI updates, versions, and current status.

Activities during this process:

- Keeps a record of all the changes made to the previous baseline to reach a new baseline
- Identify all items to define the software configuration
- Monitor status of change requests
- Complete listing of all changes since the last baseline
- Allows tracking of progress to next baseline
- Allows to check previous releases/versions to be extracted for testing

7. CONFIGURATION AUDIT AND REVIEW

Identification, version control, and change control help the software developer to maintain order in what would otherwise be a chaotic and fluid situation. However, even the most successful control mechanisms track a change only until an ECO is generated. How can we ensure that the change has been properly implemented? The answer is two-fold:

(1) Formal Technical Reviews (FTR)
(2) The software configuration audit

A *software configuration audit* complements the formal technical review by assessing a configuration object for characteristics that are generally not considered during review

The formal technical review focuses on the technical correctness of the configuration object that has been modified. The reviewers assess the SCI to determine consistency with other SCIs, omissions, or potential side effects. A formal technical review should be conducted for all but the most trivial changes.

The audit asks and answers the following questions:

1) Has the change specified in the ECO been made? Have any additional modifications been incorporated?
2) Has a formal technical review been conducted to assess technical correctness?
3) Has the software process been followed and have software engineering standards been properly applied?
4) Has the change been "highlighted" in the SCI? Have the change date and change author been specified? Do the attributes of the configuration object reflect the change?
5) Have SCM procedures for noting the change, recording it, and reporting it been followed?
6) Have all related SCIs been properly updated?

Participant of SCM Process

- Configuration manager
- Developer
- Project Manager

- User

Configuration Manager: The Configuration Manager is in charge of tracking down the many components that make up the configuration. Team adherence to the SCM process is monitored by CM. It is up to him/her to accept or reject requests for adjustments.

Developer: The code must be updated in accordance with normal development procedures and user requirements. He is in charge of the code's configuration. A developer's job is to monitor transitions and resolve conflicts.

Project Manager: Make sure the product is developed in a reasonable amount of time.

Tracks development and flags any problems with the SAM procedure. Produce reports about the software's current state Make sure that procedures and policies for making and revising and putting anything to the test are adhered to.

User: Making use of the created software or system. Giving suggestions for improving the software's features and performance. Communicating concerns and faults discovered in use. Respecting the needs and preferences of an individual user's setting. Working together with the coders on UAT (user acceptance testing).Requesting improvements or alterations in a way that is consistent with established change management procedures, these are all the responsibilities of the end users

8. STATUS REPORTING

Configuration status reporting usually can be performed by specifying below things:

1. What happened?
2. When did status reported?
3. To whom status can be reported?
4. What else should be affected?

Configuration status reporting plays a vital role in the success of a large software development project. When many people are involved, it is likely that "the left hand not knowing what the right hand is doing" syndrome will occur.

Two developers may attempt to modify the same SCI with different and conflicting intents. A software engineering team may spend months of effort building software to an obsolete hardware specification. The person who would recognize serious side effects for a proposed change is not aware that the change is being made. Status reporting helps to eliminate the problems by improving communication among all people involved.

9. GOALS OF SCM

- Configuration Identification – What code are we working with?
- Configuration Control – Controlling the release of a product and its changes.
- Status Accounting – Recording and reporting the status of components.
- Review – Ensuring completeness and consistency among components.
- Build Management Managing the process and tools used for builds.
- Process Management – Ensuring adherence to the organization's development process.
- Environment Management – Managing the software and hardware that host our system.
- Teamwork – Facilitate team interactions related to the process.
- Defect Tracking – making sure every defect has traceability back to the source

Version Control:

Tracking and controlling modifications to computer code is known as "version control," "source control," or "change management."

Fundamental to the software development process, version control allows numerous programmers to work on the same codebase at the same time while keeping track of what has changed. This keeps the codebase tidy so that teams of programmers can collaborate effectively without worrying about accidentally erasing one another's changes.

The process of monitoring and controlling modifications to computer programmes is known as "version control" or "source control." A version control system is a software tool used by development teams to track and keep track of all of the different versions of their code. Now more than ever, version control systems are crucial to the success of software development teams in today's fast-paced workplaces. DevOps teams can benefit greatly from them since they shorten the time it takes to develop new features and improve the rate at which new features are successfully deployed. DevOps teams can benefit greatly from them since they shorten the time it takes to develop new features and improve the rate at which new features are successfully deployed.

Need of version control:

- In the event of a disaster or the more subtle degradation caused by human mistake or unexpected effects, version control safeguards the source code.
- The source code for a piece of software is the equivalent of the crown jewels; it is a valuable asset that must be safeguarded at all costs.
- The source code is a repository of the invaluable knowledge and understanding about the problem domain that the developers have accumulated and improved through careful effort for the majority of software teams.
- Source code version control safeguards against accidental degradation and accidental consequences of human error.

It's not uncommon for multiple team members to make modifications to the codebase at the same time, with one developer working on a new feature and another fixing a bug in completely unrelated code.

Teams of software developers are constantly adding new lines of code to the source repository and making modifications to old lines. In most cases, the code for a project, app, or software component is organised in a folder structure, sometimes known as a "file tree."

Benefits of version control:

1. **Version control is used to prevent conflicts:**

   In addition, new software should never be relied upon before it has been thoroughly tested, as every change to the code can result in unexpected errors. So, until a new version is complete, testing and development continue simultaneously.

   - The use of version control helps teams avoid these conflicts by keeping track of each change made by each contributor.
   - It is possible for changes made by one developer to be incompatible with those made by another developer working on the same piece of software at the same time.
   - Finding and fixing this issue should proceed in a methodical fashion that doesn't hold up the rest of the team's efforts.

- Git is free and open source. Regardless of what they are called, or which system is used, the primary benefits you should expect from version control
- Using version control software is a best practice for high performing software and DevOps teams.
- Version control also helps developers move faster and allows software teams to preserve efficiency and agility as the team scales to include more developers.

Branching and merging: Creating a branch in a Vis toolset ensures that many work streams remain autonomous from one another. While it's obvious that teams should be working together, it's also important for individuals to be able to work on separate streams of change. Having the option to integrate the two sets of changes back together allows developers to ensure that their work does not conflict.

Traceability: This is essential for developers to have any hope of accurately estimating future work, and it can be especially useful when working with legacy code.

It is helpful to be able to track each change made to the software and connect it to project management and issue tracking software like Jira, as well as to be able to annotate each change with a message describing the purpose and objective of the change. Developers can make changes that are correct and beneficial to the system as a whole by consulting the annotated history of the code while reading the code and trying to understand what it is doing and why it is structured the way it is.

Two approaches to deal with version control system

1. Local Version Control System
2. Centralised Version Control System
3. Decentralised Version Control System

1. Local Version Control System: The local machine of the developer is where version control is administered in a Local VCS. Taking snapshots of a project directory at various times is a common practise. Because of its ease of use, this method is widely employed; yet, it is also fraught with significant room for error. It's easy to make a mistake when working with files and directories if you lose track of where you are.

Copying files to a new location (maybe one with a timestamp, if you're feeling really clever) is a common method for version control. To address this problem, developers created local VSSes with a straightforward database that tracked all file edits for easy revision tracking.

2. Centralised Version Control System: A central server houses the repository in a centralised VCS. Software developers work off of a centralised repository of code that they may check out, modify locally, and then upload to the server. When programmers "check out" code from the repository, it is copied to their local system.

   They "commit" their locally made modifications to the main repository.

   As a result, other programmers can simply update their own local copy to reflect the most recent modifications.

3. Decentralised Version Control System: Each developer in a Decentralised VCS has their own local, fully-functional copy of the repository. As a result, version control can be more adaptable and redundant. The entire repository can be cloned by developers, providing them with a local copy of all files and revision history.

   They only need intermittent network connectivity to make commits to their local repository.

   Alterations can be communicated directly between repositories.

Change Control: Modifications to a product, system, or project require a methodical and organised procedure known as "change control." It guarantees a methodical and well-documented approach to change management throughout the planning, evaluation, and implementation stages. This method aids in preserving the lasting quality, dependability, and durability of the product or system. Change management is the process of checking that modifications achieve their goals with no unwanted side effects by examining, approving, documenting, testing, and implementing them.

Change having controls in place to ensure the integrity of the product by controlling changes to and release of approved baseline throughout the life of the product.

Procedure:

- ✓ Select, identify, and classify the components/items to be placed under CM control
- ✓ Creation of an identification scheme that reflects the software hierarchy (e.g., represents the architecture of the product)
- ✓ Identifying and maintaining relationships between components
- ✓ Identifying baselines for a product

Configuration Management Process:

The 'CM Change Control' method ensures that configuration item changes are monitored and managed. It lessens the likelihood that a change will be made accidentally that will have a negative impact on functionality.

The configuration management team will benefit from better communication about the changes that have been requested, a standardised method for addressing those changes, and less uncertainty about the final product.

The Configuration Change Control includes the following types of changes to the Project Repository:

- ✓ Add/Remove/Edit Configuration Items
- ✓ Add/Edit/Remove User and Permissions

Entry Criteria:

Result of configuration audits, collated in a Report, and corrective actions specified. The process and project teams will ensure that these points are tracked to closure.

- Configuration Change request received.
- Inputs
- Change request information.
- Output
- Updated Configuration Change Log

Methods for Distributing Software

The objective of the Software Release process is to guarantee that software is released in a timely manner and in accordance with the project's approved plan. Each product is updated with the correct version before release, and all related pull requests are fulfilled.

Entry Criteria: Completed Testing and Verification of Released Software Products.

Enter Software Release Products (Including Software and Documentation).

Release Notes for New Software

Exit Requirements Completion and Submission of Final Work Products: the documented findings of configuration audits, with recommendations for corrective action. The process and project teams will follow up to make sure these items are resolved.

CM Audits and Reviews Process

The software development and configuration management process relies heavily on the completion and submission of final work deliverables, such as documentation of configuration audit results and recommendations for corrective action. In general, this entails the following steps:

To ensure that the established requirements, standards, and specifications are met, configuration audits are performed on the configuration items (CIs) and any supporting documentation.

Depending on the system or project at hand, auditors may conduct either a functional configuration audit (FCA) or a physical configuration audit (PCA).

The software or system configuration is in accordance with specifications, standards, and best practises after the completion and submission of final work products following configuration audits. Through all stages of a project's or system's lifespan, it aids in preserving quality, reliability, and compliance.

Functional Configuration Audit (FCA) verifies that the software actually satisfies the specified software requirements Physical Configuration Audit (PCA) determines whether or not the design and reference documents represent the software that was built

- ✓ The goal of Configuration Audit is to verify that all software products have been produced, correctly identified and described, and that change requests have been resolved. Informal audits are conducted at key phases of the software life cycle.
- ✓ Informal audits are conducted at key phases of the software life cycle.
- ✓ Two types of formal audits need to be conducted before the software is delivered to the customer:

Metrics of CM Audit

The success of the configuration management process may be measured and improvement points can be pinpointed with the use of audit metrics. In audits of Configuration Management, these metrics are frequently used:

- Identification of Configuration Items: Recognised and recorded configuration items (CIs) count. Identification success rate (the proportion of CIs that were correctly detected).
- Management by Baseline: The total number of baselines that have been created for various releases. How often new or updated baselines are made available.
- Manage the Change: The total sum of authorised change requests. How long it takes to review and authorise a request for a change. Success rate in putting in place changes that have been approved.
- Revision Management: There are a total of n revisions of the code available in the repository. Distribution of code updates.
- Status Reporting for Configurations: The percentage of correct status reports, as recorded in the records.

How often you'll get updates.

- Audit Results: The total number of inconsistencies or non-compliances discovered by the audit.

Degrees to which the differences are problematic (e.g., major, moderate, minor).

- Action to Correct: The total number of suggested course corrections.

The amount of time needed to make necessary adjustments.

- Observance of Prescribed Procedures: Compliance with established norms and practises for configuration management in both the industry and the organisation.

Completeness and accuracy of documentation:

Concept of Change Control

The new version of the programme is built using version control methods when the altered item has been checked in to the project database.

To prevent accidental overwriting of one another's work during simultaneous edits, synchronisation controls are employed.

A request for modification is made and evaluated in terms of its technical value and impact on the remaining configuration projects and the budget.

The findings of the evaluation are presented in the "Change Report."

The final decision on the status and priority of the change is made by the Change Control Authority (CCA) based on the Change Report (CR), and an Engineering Change Order (E) is generated for each change that is approved (describes the change, lists the objections, and specifies the criteria for review).

Checked-out from the project database in accordance with the object's access control parameters; subjected to the appropriate SQs and testing procedures for the modified object.

Configuration Audit

An audit of a system's configuration management process, methods, and documentation is called a "Configuration Audit." Its purpose is to guarantee that the system is set up correctly, follows industry norms, and fulfils all necessary conditions. System integrity, dependability, and security can all be maintained through regular configuration audits. Integrity and quality

of a system's configuration are essential to its operation throughout its lifecycle, and configuration audits play a key part in this. They guarantee the system's conformance to requirements and the efficacy of configuration management practices for handling modifications.

Objective of Configuration audit:

- Verification of Compliance: Compliance with organisational policies, industry standards, and legal mandates must be maintained throughout the configuration management process.
- Completeness and precision: Making sure everything in the configuration management system has a correct identifier, description, and storage location.
- Checking the Status Quo: Verifying sure baselines are established at key points in the process and faithfully capture the system's status at that juncture.
- Efficient Management of Change: Evaluating how well the CI change control method is working to manage CI updates.
- Safety Measures and Permit Obtainable: Protecting the privacy and safety of configuration data by installing appropriate safeguards.
- Analysing Danger: Locating and suggesting solutions for configuration management's identified weaknesses or threats.
- Efficiency Gains: Making suggestions and offering criticism to better the configuration management process and boost its efficacy.

Configuration Management Plan Recommends

Items that can be changed and those that need to be changed formally are both listed in the configuration management strategy, as is the method for making and implementing those changes.

Items that can be configured, those that need formal change control, and the process for controlling such changes are all laid out in the configuration management plan.

The following is what we learn from configuration planning:

- Which parts of the project can be altered as needed?
- Which all pieces (say Scope Statement, Work Breakdown Structure Dictionary) require formal change control and
- what would the process of controlling these changes look like?

CM audit and review process:

When it comes to software, it's important to make sure it lives up to its requirements. A Functional Configuration Audit (FCA) does just that. A Physical Configuration Audit (PCA) verifies if the code matches the specifications in the documentation.

The purpose of a configuration audit is to ensure that all requested changes have been implemented and that all software products have been developed and identified correctly. Informational audits are performed at pivotal points in the software development life cycle.

Before the software can be shipped to the client, it must pass two types of formal audits:

- Informal audits are conducted at key phases of the software life cycle.
- Two types of formal audits need to be conducted before the software is delivered to the customer:

Configuration Management Plan Recommends

- A tool to manage Configurable Items,
- A versioning scheme. For example, a Document Version will have two segments like aa, bb, cc, .dd.
- The first segment will represent the product; the second will represent deliverable, etc..

Two Views of Configuration Management

1. Product-Level Configuration:
2. Project Level Configuration:

1. Product Level Configuration: When discussing the output of a project, often a tangible good, the term "product-level configuration" is used to describe the configuration items (CIs) related to that product. The finished product is made up of these artefacts and parts. All configuration data for a given product is kept up-to-date during its entire lifecycle, which includes creation, testing, release, and regular upkeep and improvement.

   Example: Materials that are integral to the product's operation and functionality, including but not limited to user guides, technical manuals, source code, executables, design specifications, user interfaces, databases, and so on.

2. Project Level Configuration: The configuration items at the project level are those that pertain to the project itself and not the final output. The management and execution of the project are governed by these documents, plans, and resources. Configuration elements at the project level are kept up-to-date to guarantee that work is being done in accordance with established protocols. Throughout the project's lifetime, they are monitored, revised, and adjusted as needed to account for developments and obstacles.

   Example: Documents relating to the project as a whole, including the Work Breakdown Structure (WBS) dictionary, project management plans, project timetables, risk assessments, quality assurance plans, and change management procedures.

Configuration Status Accounting

It indicates whether or not a CI has pending revisions.

Where we stand with accepted alterations. It guarantees that all relevant information is recorded for a specific alteration.It also details the total number of rejected modification requests and the reasons for those rejections.

The purpose of configuration accounting is to keep track of every change that has been made to a system, including any and all CI versions. Each version's current status is also displayed. The status could read something like this: "I Position within the life cycle. Any of the following states—draft, under review, approved, not effective, effective, retired, or

obsolete—would suffice as an example stage. These are only examples; depending on the nature of the project, additional or alternative stages may be required.

Details on any pending alterations to an SGID.

Example: As modifications are made in accordance with the configuration management strategy, the state of the configuration must be recorded. Things include updating the version, keeping a version log, etc.

If I'm going to approve the revised scope statement, I need to know that a revised version of this document is on the way. And after the change is implemented, the Configuration Management system will make sure everyone is using the most recent build.

Configuration Verification and Audit

The purpose of configuration verification and auditing is as follows.

It guarantees that the projects are constructed in accordance with their prescribed documentation. This is a formal audit, performed at least once each release.

The project team needs a system in place to regularly check and audit the configuration management plan.

Possible verification criteria include: is the team managing the Is version as defined;

Is the team keeping track of revisions, can the old version be generated by the system without hiccups, etc.?

It is not meant to replace previous analyses and forecasts. It's similar to having a quality assurance team come in and do an audit to make sure the configuration management plan is being adhered to.

Configuration verification:

This verifies the configuration specifications have been followed and the desired functionality has been achieved.

The correct composition of a project's configuration items and the recording, assessment, approval, tracking, and implementation of corresponding changes are guaranteed by conducting configuration audits and verifications.

However, we are not directly affecting anything when we add or remove issues from the issue log. For this reason, we skipped out on formal change control.

It's also important to realise that not all of the artworks adhere to the rules of conventional currency exchange. We should focus on more productive endeavours instead. Here's an example: We believe formal change control for the WBS dictionary is necessary in a project. Changes in acceptance criteria, test case coding, and resource allocation, all of which are related to a certain WBS dictionary, could be the cause.

SELF-ASSESSMENT QUESTIONS - 3

7. For a large software engineering project, uncontrolled change rapidly leads to chaos. (True / False)
8. The results of the evaluation are presented as a change report, which is used by a _____.
9. _____ helps to eliminate the problems by improving communication among all people involved.

10. SUMMARY

Let's summarize the important points covered in the unit:

- Software configuration management is an umbrella activity that is applied thought the software process. SCM identifies controls, audits, and reports modifications that invariably occur while software is being developed and after it has been released to a customer. All artifacts produced as part of software engineering processes during the project execution are termed as software configuration items (SCI).
- The SCI's are arranged in the project database or repository in the form of configuration objects depicting the relation between the objects. This depiction of relations among the configuration objects helps SCM in determining if the change done to a section of software has any impact on other parts of the application.
- In addition to documents, programs, and data, the development environment that is used to create software can also be placed under configuration control.
- Once a configuration object has been developed and reviewed, it becomes baseline. Changes to a baseline object result in the creation of a new version object. The evolution of a program can be tracked by examining the revision history of all SCI's arranged as configuration objects. Version control is the set of procedures, tools for managing the use of these objects.
- Change control is a procedural activity that ensures quality and consistency as changes are made to a configuration object. The change control process begins with a change request, leads to a decision to make or reject the request for change culminates with a controlled update of the SCI that is to be changed.
- The configuration audit is an SQA activity that helps to ensure that quality is maintained as changes are made. Status reporting provides information about each change to those with a need to know.

11. TERMINAL QUESTIONS

1. What is configuration management? Explain in detail.
2. Explain software configuration management process in detail.
3. Explain the goals of the SCM.
4. What is the difference between SCM audit and a formal technical review?
5. What happens to software code, when too many changes occur? Explain in your words.
6. What is version control? Explain how it helps to reduce too many changes.

12. ANSWERS

Self Assessment Questions

1. True
2. Scope and Vision document
3. Configuration Object
4. False
5. Base Objects, Aggregate Objects
6. Version Control
7. True
8. Change Control Authority
9. Status Reporting

Terminal Questions

1. Software Configuration Management is a set of activities that have been developed to manage change throughout the life cycle of computer software. SCM can be considered as a software quality assurance exercise that is applied throughout the software process. (Refer Section 2)
2. Software Configuration Management (SCM) is essentially a mechanism of controlling the change in the software project. It can also be considered as one of the quality assurance exercises undertaken by the project. In order to control the change the SCM must also be capable of the identifying the artifacts that need to controlled in the project. In other words the SCM will also include the activity of identifying the SCI's,

their versions, and also audit information regarding the SCI's to get the history of any changes that SCI would have undergone. (Refer Section 3)

3. Configuration Identification, Configuration Control, Status Accounting. (Refer Section 9)
4. The formal technical review focuses on the technical correctness of the configuration object that has been modified. The reviewers assess the SCI to determine consistency with other SCIs, omissions, or potential side effects. A formal technical review should be conducted for all but the most trivial changes. (Refer Section 7)
5. Too much change control creates lots of problems. For a large software engineering project, uncontrolled change rapidly leads to chaos. For such projects, change control combines human procedures and automated tools to provide a mechanism for the control of change. (Refer Section 6)
6. Version control combines procedures and tools to manage different versions of configuration objects that are created during the software process. (Refer Section 5)