Module 12

Software Project Monitoring and Control

Lesson 31

Risk Management and Software Configuration Management

Specific Instructional Objectives

At the end of this lesson the student would be able to:

- Identify the main categories of risks which can affect a software project.
- Explain how to assess a project risk.
- Identify the main strategies to plan for risk containment.
- Explain what risk leverage is.
- Explain how to handle the risk of schedule slippage.
- Explain what is meant by configuration of a software product.
- Differentiate among release, version and revision of a software product.
- Explain the necessity of software configuration management.
- Identify the principal activities of software configuration management.
- Identify the activities carried out during configuration identification.
- Identify the activities carried out during configuration control.
- Identify the popular configuration management tools.

Risk management

A software project can be affected by a large variety of risks. In order to be able to systematically identify the important risks which might affect a software project, it is necessary to categorize risks into different classes. The project manager can then examine which risks from each class are relevant to the project. There are three main categories of risks which can affect a software project:

Project risks. Project risks concern varies forms of budgetary, schedule, personnel, resource, and customer-related problems. An important project risk is schedule slippage. Since, software is intangible, it is very difficult to monitor and control a software project. It is very difficult to control something which cannot be seen. For any manufacturing project, such as manufacturing of cars, the project manager can see the product taking shape. He can for instance, see that the engine is fitted, after that the doors are fitted, the car is getting painted, etc. Thus he can easily assess the progress of the work and control it. The invisibility of the product being developed is an important reason why many software projects suffer from the risk of schedule slippage.

Technical risks. Technical risks concern potential design, implementation, interfacing, testing, and maintenance problems. Technical risks also include ambiguous specification, incomplete specification, changing specification, technical uncertainty, and technical obsolescence. Most technical risks occur due to the development team's insufficient knowledge about the project.

Business risks. This type of risks include risks of building an excellent product that no one wants, losing budgetary or personnel commitments, etc.

Risk assessment

The objective of risk assessment is to rank the risks in terms of their damage causing potential. For risk assessment, first each risk should be rated in two ways:

- The likelihood of a risk coming true (denoted as r).
- The consequence of the problems associated with that risk (denoted as s).

Based on these two factors, the priority of each risk can be computed:

$$p = r * s$$

Where, p is the priority with which the risk must be handled, r is the probability of the risk becoming true, and s is the severity of damage caused due to the risk becoming true. If all identified risks are prioritized, then the most likely and damaging risks can be handled first and more comprehensive risk abatement procedures can be designed for these risks.

Risk containment

After all the identified risks of a project are assessed, plans must be made to contain the most damaging and the most likely risks. Different risks require different containment procedures. In fact, most risks require ingenuity on the part of the project manager in tackling the risk.

There are three main strategies to plan for risk containment:

Avoid the risk: This may take several forms such as discussing with the customer to change the requirements to reduce the scope of the work, giving incentives to the engineers to avoid the risk of manpower turnover, etc.

Transfer the risk: This strategy involves getting the risky component developed by a third party, buying insurance cover, etc.

Risk reduction: This involves planning ways to contain the damage due to a risk. For example, if there is risk that some key personnel might leave, new recruitment may be planned.

Risk leverage

To choose between the different strategies of handling a risk, the project manager must consider the cost of handling the risk and the corresponding reduction of risk. For this the risk leverage of the different risks can be computed.

Risk leverage is the difference in risk exposure divided by the cost of reducing the risk. More formally,

> risk leverage = (risk exposure before reduction – risk exposure after reduction) / (cost of reduction)

Risk related to schedule slippage

Even though there are three broad ways to handle any risk, but still risk handling requires a lot of ingenuity on the part of a project manager. As an example, it can be considered the options available to contain an important type of risk that occurs in many software projects – that of schedule slippage. Risks relating to schedule slippage arise primarily due to the intangible nature of software. Therefore, these can be dealt with by increasing the visibility of the software product. Visibility of a software product can be increased by producing relevant documents during the development process wherever meaningful and getting these documents reviewed by an appropriate team. Milestones should be placed at regular intervals through a software engineering process to provide a manager with regular indication of progress. Completion of a phase of the development process before followed need not be the only milestones. Every phase can be broken down to reasonable-sized tasks and milestones can be scheduled for these tasks too. A milestone is reached, once documentation produced as part of a software engineering task is produced and gets successfully reviewed. Milestones need not be placed for every activity. An approximate rule of thumb is to set a milestone every 10 to 15 days.

Software configuration management

The results (also called as the deliverables) of a large software development effort typically consist of a large number of objects, e.g. source code, design document, SRS document, test document, user's manual, etc. These objects are usually referred to and modified by a number of software engineers through out the life cycle of the software. The state of all these objects at any point of time is called the configuration of the software product. The state of each deliverable object changes as development progresses and also as bugs are detected and fixed.

Release vs. Version vs. Revision

A new version of a software is created when there is a significant change in functionality, technology, or the hardware it runs on, etc. On the other hand a new revision of a software refers to minor bug fix in that software. A new release is created if there is only a bug fix, minor enhancements to the functionality, usability, etc.

For example, one version of a mathematical computation package might run on Unix-based machines, another on Microsoft Windows and so on. As a software is released and used by the customer, errors are discovered that need correction. Enhancements to the functionalities of the software may also be needed. A new release of software is an improved system intended to replace an old one. Often systems are described as version m, release n; or simple m.n. Formally, a history relation is version of can be defined between objects. This relation can be split into two sub relations *is revision of* and *is variant of*.

Necessity of software configuration management

There are several reasons for putting an object under configuration management. But, possibly the most important reason for configuration management is to control the access to the different deliverable objects. Unless strict discipline is enforced regarding updation and storage of different objects, several problems appear. The following are some of the important problems that appear if configuration management is not used.

Inconsistency problem when the objects are replicated. A scenario can be considered where every software engineer has a personal copy of an object (e.g. source code). As each engineer makes changes to his local copy, he is expected to intimate them to other engineers, so that the changes in interfaces are uniformly changed across all modules. However, many times an engineer makes changes to the interfaces in his local copies and forgets to intimate other teammates about the changes. This makes the different copies of the object inconsistent. Finally, when the product is integrated, it does not work. Therefore, when several team members work on developing an object, it is necessary for them to work on a single copy of the object, otherwise inconsistency may arise.

Problems associated with concurrent access. Suppose there is a single copy of a problem module, and several engineers are working on it. Two engineers may simultaneously carry out changes to different portions of the same module, and while saving overwrite each other. Though the problem associated with concurrent access to program code has been explained, similar problems occur for any other deliverable object.

Providing a stable development environment. When a project is underway, the team members need a stable environment to make progress. Suppose somebody is trying to integrate module A, with the modules B and C, he cannot make progress if developer of module C keeps changing C; this can be especially frustrating if a change to module C forces him to recompile A. When an effective configuration management is in place, the manager freezes the objects to form a base line. When anyone needs any of the objects under configuration control, he is provided with a copy of the base line item. The requester makes changes to his private copy. Only after the requester is through with all modifications to his private copy, the configuration is updated and a new

base line gets formed instantly. This establishes a baseline for others to use and depend on. Also, configuration may be frozen periodically. Freezing a configuration may involve archiving everything needed to rebuild it. (Archiving means copying to a safe place such as a magnetic tape).

System accounting and maintaining status information. System accounting keeps track of who made a particular change and when the change was made.

Handling variants. Existence of variants of a software product causes some peculiar problems. Suppose somebody has several variants of the same module, and finds a bug in one of them. Then, it has to be fixed in all versions and revisions. To do it efficiently, he should not have to fix it in each and every version and revision of the software separately.

Software configuration management activities

Normally, a project manager performs the configuration management activity by using an automated configuration management tool. A configuration management tool provides automated support for overcoming all the problems mentioned above. In addition, a configuration management tool helps to keep track of various deliverable objects, so that the project manager can quickly and unambiguously determine the current state of the project. The configuration management tool enables the engineers to change the various components in a controlled manner.

Configuration management is carried out through two principal activities:

- Configuration identification involves deciding which parts of the system should be kept track of.
- Configuration control ensures that changes to a system happen smoothly.

Configuration identification

The project manager normally classifies the objects associated with a software development effort into three main categories: controlled, precontrolled, and uncontrolled. Controlled objects are those which are already put under configuration control. One must follow some formal procedures to change them. Precontrolled objects are not yet under configuration control, but will eventually be under configuration control. Uncontrolled objects are not and will not be subjected to configuration control. Controllable objects include both controlled and precontrolled objects. Typical controllable objects include:

- Requirements specification document
- Design documents

- Tools used to build the system, such as compilers, linkers, lexical analyzers, parsers, etc.
- Source code for each module
- Test cases
- Problem reports

The configuration management plan is written during the project planning phase and it lists all controlled objects. The managers who develop the plan must strike a balance between controlling too much, and controlling too little. If too much is controlled, overheads due to configuration management increase to unreasonably high levels. On the other hand, controlling too little might lead to confusion when something changes.

Configuration control

Configuration control is the process of managing changes to controlled objects. Configuration control is the part of a configuration management system that most directly affects the day-to-day operations of developers. The configuration control system prevents unauthorized changes to any controlled objects. In order to change a controlled object such as a module, a developer can get a private copy of the module by a reserve operation as shown in fig. 12.5. Configuration management tools allow only one person to reserve a module at a time. Once an object is reserved, it does not allow any one else to reserve this module until the reserved module is restored as shown in fig. 12.5. Thus, by preventing more than one engineer to simultaneously reserve a module, the problems associated with concurrent access are solved.

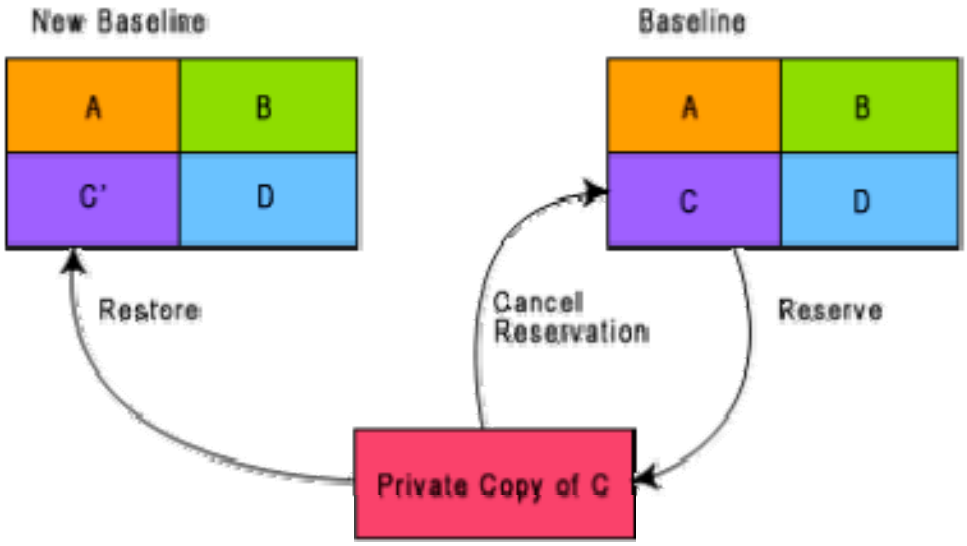

Fig. 12.5: Reserve and restore operation in configuration control

It can be shown how the changes to any object that is under configuration control can be achieved. The engineer needing to change a module first obtains a private copy of the module through a reserve operation. Then, he carries out all necessary changes on this private copy. However, restoring the changed module to the system configuration requires the permission of a change control board (CCB). The CCB is usually constituted from among the development team members. For every change that needs to be carried out, the CCB reviews the changes made to the controlled object and certifies several things about the change:

1. Change is well-motivated.
2. Developer has considered and documented the effects of the change.
3. Changes interact well with the changes made by other developers.
4. Appropriate people (CCB) have validated the change, e.g. someone has tested the changed code, and has verified that the change is consistent with the requirement.

The change control board (CCB) sounds like a group of people. However, except for very large projects, the functions of the change control board are normally discharged by the project manager himself or some senior member of the development team. Once the CCB reviews the changes to the module, the project manager updates the old base line through a restore operation (as shown in fig. 12.5). A configuration control tool does not allow a developer to replace an object he has reserved with his local copy unless he gets an authorization from the CCB. By constraining the developers' ability to replace reserved objects, a stable environment is achieved. Since a configuration management tool allows only one engineer to work on one module at any one time, problem of accidental overwriting is eliminated. Also, since only the manager can update the baseline after the CCB approval, unintentional changes are eliminated.

Configuration management tools

SCCS and RCS are two popular configuration management tools available on most UNIX systems. SCCS or RCS can be used for controlling and managing different versions of text files. SCCS and RCS do not handle binary files (i.e. executable files, documents, files containing diagrams, etc.) SCCS and RCS provide an efficient way of storing versions that minimizes the amount of occupied disk space. Suppose, a module MOD is present in three versions MOD1.1, MOD1.2, and MOD1.3. Then, SCCS and RCS stores the original module MOD1.1 together with changes needed to transform MOD1.1 into MOD1.2 and MOD1.2 to MOD1.3. The changes needed to transform each base lined file to the next version are stored and are called deltas. The main reason behind storing the deltas rather than storing the full version files is to save disk space.

The change control facilities provided by SCCS and RCS include the ability to incorporate restrictions on the set of individuals who can create new versions, and facilities for checking components in and out (i.e. reserve and restore operations). Individual developers check out components and modify them. After they have made all necessary changes to a module and after the changes have been reviewed, they check in the changed module into SCCS or RCS. Revisions are denoted by numbers in ascending order, e.g., 1.1, 1.2, 1.3 etc. It is also possible to create variants or revisions of a component by creating a fork in the development history.

The following questions have been designed to test the objectives identified for this module:

1. Compare the relative advantages of the functional and the project approaches to organizing a development center.
2. Suppose you are the chief executive officer (CEO) of a software development center. Which organization structure would you select for your organization and why?
3. Why the functional format is not suitable for small organizations handling just one or two projects?
4. Name the different ways in which software development teams are organized. For the development of a challenging satellite-based mobile communication product which type of project team organization would you recommend? Justify your answer.
5. Suppose you are the project manager of a large software product development team and you have to make a choice between democratic and chief programmer team organizations, which one would you adopt for your team? Explain the reasoning behind your answer.
6. What is egoless programming? How can it be realized?
7. In what units can you measure the productivity of a software development team? List three important factors that affect the productivity of a software development team.
8. As a project manager, identify the characteristics that you would look for in a software engineer while trying to select personnel for your team.
9. List three common types of risks that a typical software project might suffer from.
10. Explain how you can identify the risks that your project is susceptible to. Suppose you are the project manager of a large software development project, point out the main steps you would follow to manage risks in your software project.

11. What is meant by risk leverage?
12. Schedule slippage is a very common form of risk that almost every project manager has to overcome. Explain how would you manage the risk of schedule slippage as the project manager of a medium-sized project?
13. What do you understand by software configuration?
14. Differentiate among release, version and revision of a software product.
15. Why is software configuration management crucial to the success of large software product development projects?
16. How can you effectively manage software configuration?
17. Discuss how SCCS and RCS can be used to efficiently manage the configuration of source code.

Mark all options which are true.

1. When are the software project monitoring and control activities undertaken?

   □ before the development starts to plan the activities to be undertaken during development
   □ once the development activities start with the aim of ensuring that the development proceeds as per plan
   □ at the end of the development
   □ none of the above

2. Job specialization is one of the main advantages in case of which organization structure?

   □ project format
   □ function format
   □ either project format or function format
   □ both of project format and function format

3. Pure egoless programming is encouraged by which team organization?

   □ chief programmer team structure
   □ democratic team structure
   □ mixed control team structure
   □ none of the above

4. In which type of team organization a single point failure of development occurs when an individual leaves the team?

□ chief programmer team structure
□ democratic team structure
□ mixed control team structure
□ none of the above

5. The primary purpose of risk management is

□ risk containment
□ risk assessment
□ risk identification
□ all of the above

6. Schedule slippage is a type of

□ business risk
□ project risk
□ technical risk
□ none of the above

7. A development team's insufficient knowledge of the product being developed is one of the main factors contributing to

□ business risk
□ project risk
□ technical risk
□ none of the above

8. Visibility of a software product can be increased by

□ producing relevant documents during the development process
□ properly reviewing those relevant documents by an expert team
□ placing milestones at regular intervals through a software engineering process
□ all of the above

9. A new version of a software is produced when there is a
□ minor bug fix
□ minor enhancements to the functionality, usability etc.
□ significant change in functionality, technology, or the hardware the software runs on
□ all of the above

10. A revision of a software refers to

□ minor bug fix

□ minor enhancements to the functionality, usability etc.
□ significant change in functionality, technology, or the hardware the software runs on
□ all of the above

11. If configuration management is not during a software development effort used then which of the following problems are likely to appear:

□ inconsistency problem when the objects are replicated
□ problems associated with concurrent access
□ problems with handling several variants
□ all of the above

12. If we develop several versions of the same software product without using any configuration management tools then the problems that we would face are

□ bug fixing in any version would require manually fixing the same bug in all versions
□ large storage requirements would be needed
□ difficulty in keeping track the updated configurations of various versions
□ all of the above

13. Revisions to different software products are handled by

□ version control
□ change control
□ neither version control nor change control
□ both version control and change control

14. For effective configuration control, in order to change a controlled object such as a module, a developer can get a private copy of the module by using:

□ restore operation
□ reserve operation
□ update operation
□ none of the above

Mark the following as either True or False. Justify your answer.

1. Software organizations achieve efficient manpower utilization by adopting a project-based organization structure.
2. In order to handle complex software projects requiring knowledge of specialized domain areas, software organization would surely prefer functional organization structure.
3. The pure democratic team organization is well suited to handle extremely large and complex projects.
4. Technical knowledge in the area of the project i.e. domain knowledge is an important factor determining the productivity of an individual for a particular project.
5. For high productivity of a developer, strong interpersonal skills are essential.
6. It is possible to do the configuration management of a software project without using an automated tool.
7. A version and a release of a software product are synonyms.
8. Source Code Control System (SCCS) and RCS provide an efficient way of storing versions that minimizes the amount of occupied disk space.