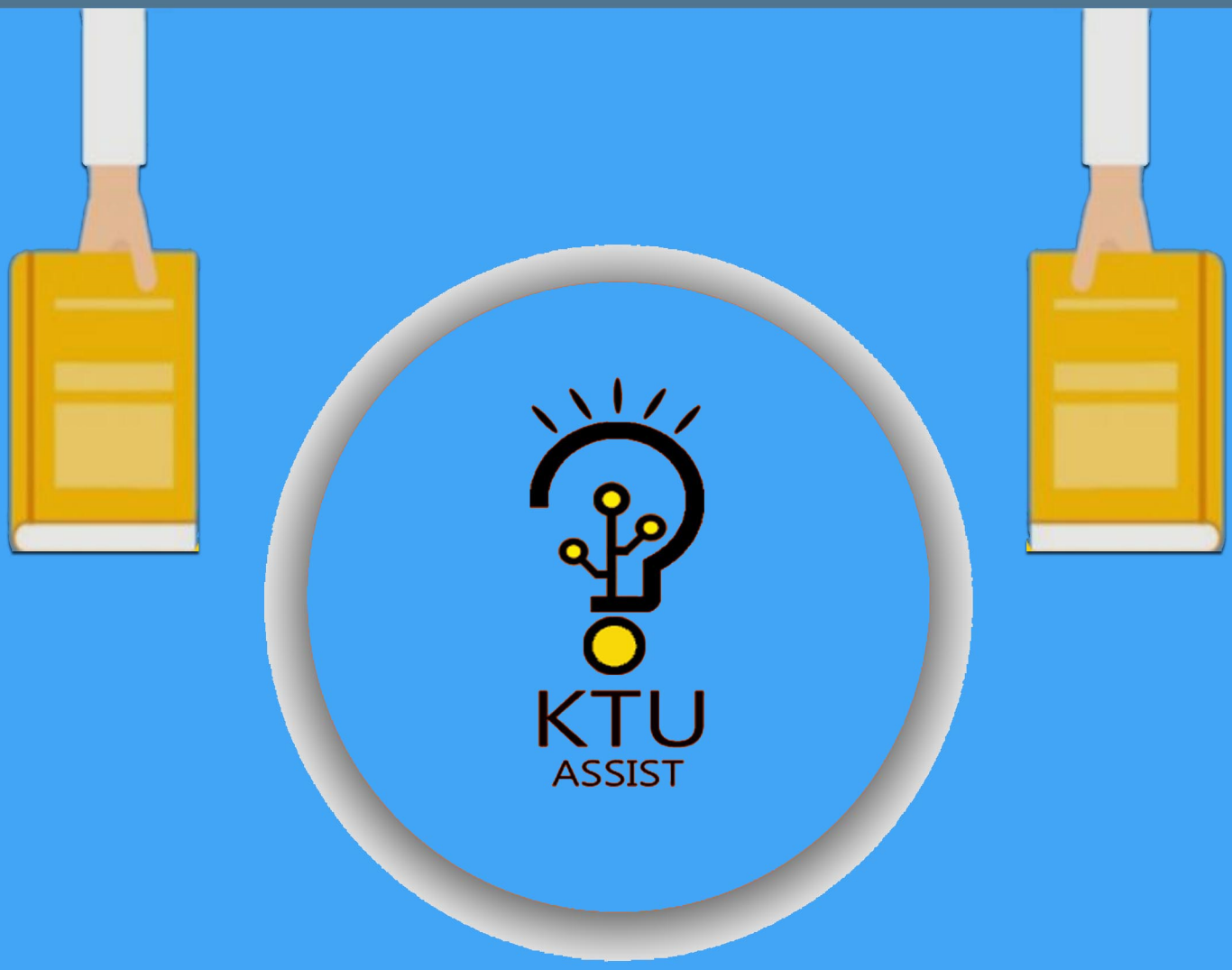
APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

STUDY MATERIALS



Project scheduling and Software configuration management

1. PROJECT SCHEDULING AND TRACKING

1.1. BASIC CONCEPTS

Project scheduling is one of the most difficult jobs for a project manager. cheduling is the culmination of a planning activity that is a primary component of ftware project management.

hy are Projects Late?

Although there are many reasons why software is delivered late, most can be aced to one or more of the following root causes:

- An unrealistic deadline established by someone outside the software development group
- Changing customer requirements that are not reflected in schedule changes;
- An honest underestimate of the amount of effort and/or the number of resources that will be required to do the job;
- Predictable and/or unpredictable risks that were not considered when the project commenced;
- Technical difficulties that could not have been foreseen in advance;
- Human difficulties that could not have been foreseen in advance;
- Miscommunication among project staff that results in delays;
- A failure by project management to recognize that the project is falling behind schedule and a lack of action to correct the problem

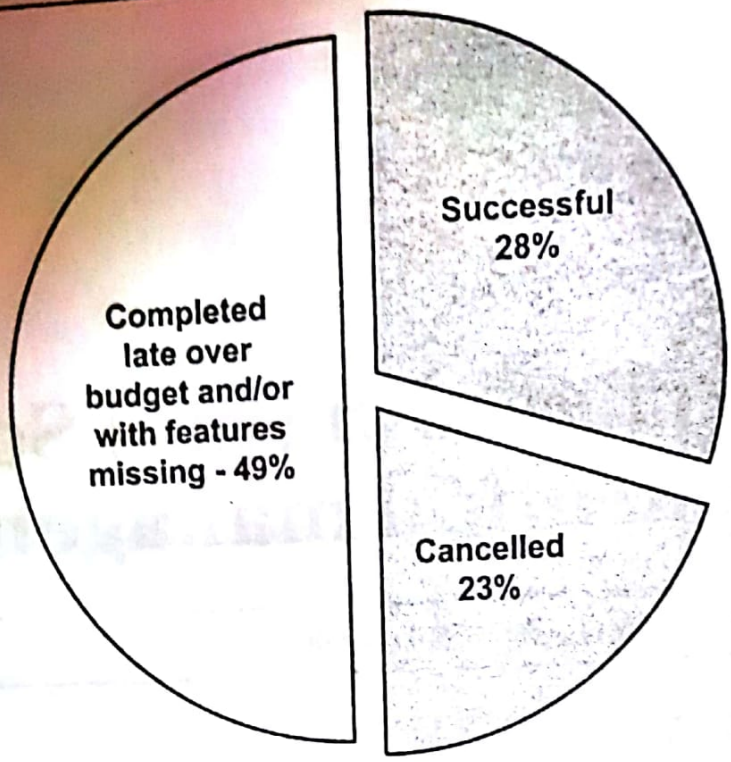


Fig. 6.1. Project scheduling

Project Scheduling

- Scheduling has the ultimate goal of deciding on how to distribute effort over time.
- Making decision to go on without considering team organization and careful scheduling will also cause problem.
- It will be impossible to know the progress of the project and whether more people can improve the progress.
- A 100 person-months project does not mean can be completed by 1 person in 100 months or 100 persons in 1 month. Usually, more people mean more communication paths and lower efficiency – Putting more people into a late project may make it later.

When project scheduling, the project manager must know

- Duration of each activity.
- Order of which the activities will be performed.
- Start and end times for each activity
- Who will be assigned to each specific task.
- Tasks that are dependent on other activities.

Several graphical planning aids can help a project manager in the scheduling process.

- Gantt charts
- PERT (Program Evaluation and Review Technique)
- CPM (Critical path Method)

Features of Tasks Scheduling

- Design and Test planning can run in parallel.
- Detail design and Coding of each unit can run in parallel.
- All tasks converge with the test plan before Integration test.
- Milestones typically take the form of a document produced by the corresponding progress review.
- It is used as a checkpoint to ensure the project is on schedule.
- In a task network, a milestone can be shown as a task without duration.

1.2. CHALLENGES IN SOFTWARE PROJECT MANAGEMENT

One of the major challenges in software project management is the difficulty to dhere to schedules. The common reasons for a late delivery of software project are n unrealistic deadline, changing customer requirements, honest underestimate of ffort or resources, overlooked risks, unforeseen technical difficulties or human ifficulties, miscommunication and failure by project manager to recognize the delay arly and take appropriate measures.

Software project scheduling is an activity that distributes estimated effort across ne duration of project cycle by allocating effort to each specific task that is ssociated with all process. The basic principles that guides software project cheduling is compartmentalization of the project into a number of manageable tasks, orrect allocation of time, correct effort validation ,defining responsibility for each ask to a team member, defining outcomes or work product for each task and defining nilestones for a task or group of tasks as appropriate.

A Task set is a collection of software tasks, milestones and deliverables that must e completed for the project to be successfully accomplished. Task sets are defined or being applicable to different type of project and degree of rigor.

The types of projects commonly encountered are Concept Development projects, New applications, Development projects, Application enhancement projects, Application maintenance projects and Re-engineering projects. The degree of rigor with which the software process is applied may be casual, structured, strict or quick reaction (used for emergency situation). For the project manager to develop a systematic approach for selecting degree of rigor for the type of project project adaptation criteria are defined and a task set selector value is computed based on the characteristics of the project.

6.1.3. SCHEDULING PRINCIPLES

Software project scheduling is an activity that distributes estimated effort across the planned project duration by allocating the effort to specific software engineering tasks.

The schedule evolves over time. During early stages of project planning, a macroscopic schedule is developed. This types of schedule identifies all major software engineering activities and the product functions to which they are applied. As the project gets under way, each entry on the macroscopic schedule is refined into a detailed schedule. Here specific software tasks (required to accomplish an activity) are identified and scheduled.

- Compartmentalization - the product and process must be decomposed into a manageable number of activities and tasks
- Interdependency - tasks that can be completed in parallel must be separated from those that must completed serially
- Time allocation - every task has start and completion dates that take the task interdependencies into account
- Effort validation - project manager must ensure that on any given day there are enough staff members assigned to completed the tasks within the time estimated in the project plan
- Defined Responsibilities - every scheduled task needs to be assigned to a specific team member
- Defined outcomes - every task in the schedule needs to have a defined outcome (usually a work product or deliverable)

- Defined milestones - a milestone is accomplished when one or more work products from an engineering task have passed quality review

6.1.4. RELATIONSHIP BETWEEN PEOPLE AND EFFORT

"If we fall behind schedule, we can always add more programmers and catch up later in the project"

Why does this not work?

- The most important point to get across is that adding people to a project in an arbitrary manner does not reduce the project completion time (and may in fact lengthen the completion time).
- There are times when a project schedule has slipped so badly that adding people cannot save it and the only option a manager has it to renegotiate the completion date with the customer.

Example:

Consider four software engineers, each capable of producing 5000 LOC/year when working on an individual project. When these four engineers are placed on a team project, six potential communication paths are possible. Each communication path requires time that could otherwise be spent developing software. We shall assume that team productivity (when measured in LOC) will be reduced by 250 LOC/year for each communication path, due to the overhead associated with communication.

Therefore, team productivity is 20,000 (250 x 6) = 18,500 LOC/year - 7.5 percent less than what we might expect.

The one-year project on which the team is working falls behind schedule, and with two months remaining, two additional people are added to the team. The number of communication paths escalates to 14.

The productivity input of the new staff is the equivalent of 840 x 2 = 1680 LOC for the two months remaining before delivery. Team productivity now is 20,000 + 1680 (250 x 14) = 18,180 LOC/year.

Although the example is a gross over simplification of real-world circumstances, it does illustrate another key point: The relationship between the number of people working on a software project and overall productivity is not linear.

Based on the people/work relationship, are teams counterproductive? The answer is an emphatic "no," if communication improves software quality. In fact, formal technical reviews (see Chapter 8) conducted by software teams can lead to better analysis and design, and more important, can reduce the number of errors that go undetected until testing (thereby reducing testing effort). Hence, productivity and quality, when measured by time to project completion and customer satisfaction, can actually improve.

An Empirical Relationship

We can demonstrate the highly nonlinear relationship between chronological time to complete a project and human effort applied to the project. The number of delivered lines of code (source statements), L, is related to effort and development time by the equation:

$$L = P \times E1/3t4/3$$

Where E is development effort in person-months, P is a productivity parameter that reflects a variety of factors that lead to high-quality software engineering work (typical values for Prange between 2,000 and 12,000), and tis the project duration in calendar months. Rearranging this software equation, we can arrive at an expression for development effort E:

$$E = L3/(P3t4)$$

Where Eis the effort expended (in person-years) over the entire life cycle for software development and maintenance and t is the development time in years. The equation for development effort can be related to development cost by the inclusion of a burdened labor rate factor ($/person-year). This leads to some interesting results. Consider a complex, real-time software project estimated at 33,000 LOC, 12 person-years of effort. If eight people are assigned to the project team, the project can be completed in approximately 1.3 years. If, however, we extend the end-date to 1.75 years, the highly nonlinear nature of the model described in Equation (7-1) yields: E = L3/(P3t4) ~ 3.8 person-years.

This implies that, by extending the end-date six months, we can reduce the number of people from eight to four! The validity of such results is open to debate, but the implication is clear: Benefit can be gained by using fewer people over a somewhat longer time span to accomplish the same objective.

Project Effort Distribution

- How should effort be distributed across the software process workflow?
- A recommended distribution of effort across the software process is often referred to as the 40-20-40 rule. Forty percent of all effort is allocated to front-end analysis and design. A similar percentage is applied to back-end testing. You can correctly infer that coding (20 percent of effort) is de-emphasized.

The 40-20-40 rule

- 40% front-end analysis and design
- 20% coding
- 40% back-end testing

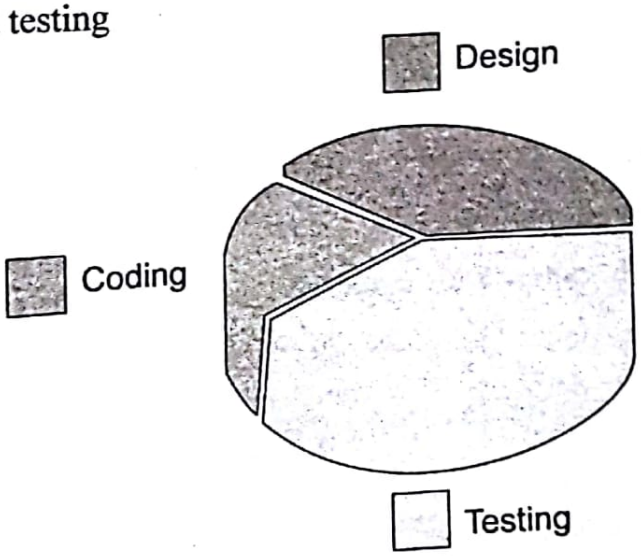


Fig. 6.2.

Generally accepted guidelines are

This effort distribution should be used as a guideline only. The characteristics of each project must dictate the distribution of effort. Work expended on project planning rarely accounts for more than 2–3 percent of effort, unless the plan commits an organization to large expenditures with high risk.

- 02-03 % planning
- 10-25 % requirements analysis
- 20-25 % design
- 15-20 % coding
- 30-40 % testing and debugging

6.1.5. DEFINING A TASK SET FOR THE SOFTWARE PROJECT

A task set is a collection of software engineering work tasks, milestones, and deliverables that must be accomplished to complete a particular project.

- The software process selected for project provides much guidance in determining the task set.
- Task set is also dependent on the project type.
- Three Steps
- Selecting Software Engineering Tasks
- Refinement of Major Tasks
- Defining a Task Network
- Determine type of project
- Assess the degree of rigor required
- Identify adaptation criteria
- Compute task set selector (TSS) value
- Interpret TSS to determine degree of rigor

Three Steps

- Selecting Software Engineering Tasks
- Refinement of Major Tasks
- Defining a Task Network

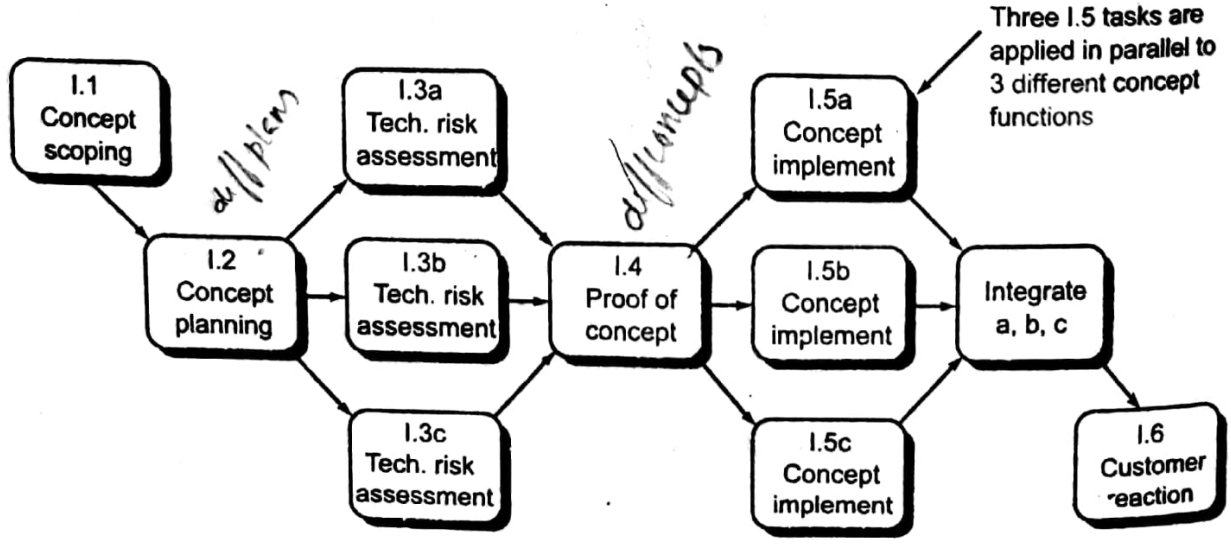


Fig. 6.3. A task network for concept development

Tasks sets are designed to accommodate different types of projects and different degrees of rigor.

Typical project types

- Concept Development Projects
- New Application Development Projects
- Application Enhancement Projects
- Application Maintenance Projects
- Reengineering Projects

6.1.6. DEGREE OF RIGOR

Even for a project of a particular type, the degree of rigor with which the software process is applied may vary significantly. The degree of rigor is a function of many project characteristics. As an example, small, non-business-critical projects can generally be addressed with somewhat less rigor than large, complex business-critical applications

Four different degrees of rigor can be defined

Casual - All process framework activities are applied, but only a minimum task set is required. In general, umbrella tasks will be minimized and documentation requirements will be reduced. All basic principles of software engineering are still applicable.

Structured - The process framework will be applied for this project. Framework activities and related tasks appropriate to the project type will be applied and umbrella activities necessary to ensure high quality will be applied. SQA, SCM, documentation, and measurement tasks will be conducted in a streamlined manner.

Strict - The full process will be applied for this project with a degree of discipline that will ensure high quality. All umbrella activities will be applied and robust work products will be produced

Quick reaction - The process framework will be applied for this project, but because of an emergency situation only those tasks essential to maintaining good quality will be applied. "Back-filling" (i.e., developing a complete set of documentation, conducting additional reviews) will be accomplished after the application/product is delivered to the customer.

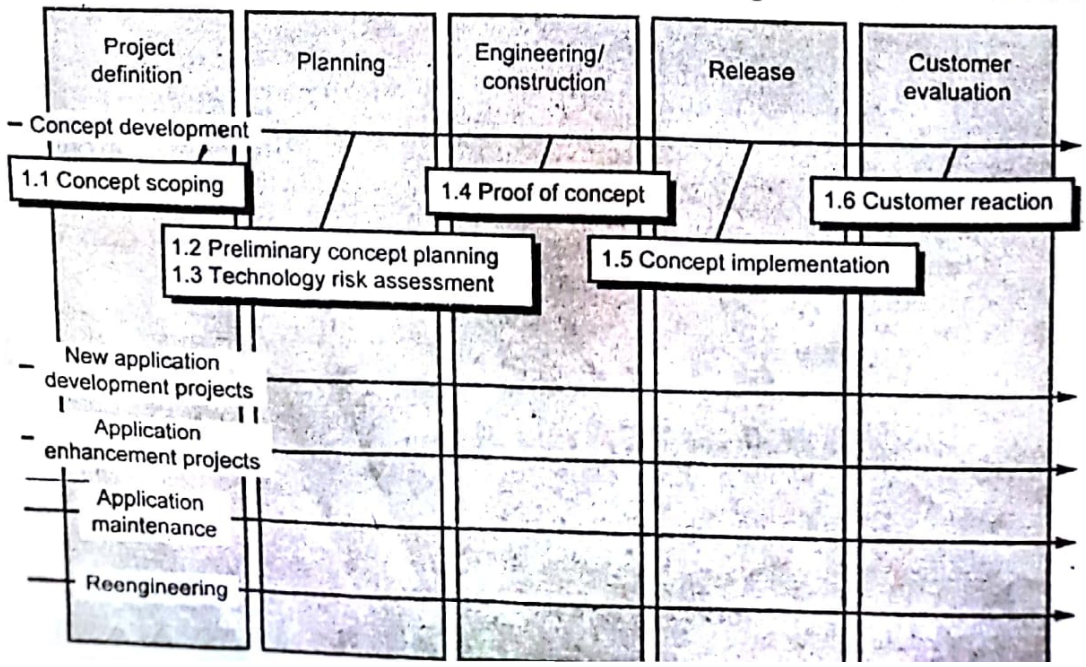
6.1.7. SELECTING SOFTWARE ENGINEERING TASKS

- Scheduling involves taking the software engineering task set and distributing it on the project time line.
- The details of how to do this will depend on whether the software process model is linear, iterative or evolutionary.

In order to develop a project schedule, a task set must be distributed on the project time., The task set will vary depending upon the project type and the degree of rigor. Each of the project types may be approached using a process model that is linear sequential, iterative (e.g., the prototyping or incremental models), or evolutionary (e.g., the spiral model). In some cases, one project type flows smoothly into the next.

For example, concept development projects that succeed often evolve into new application development projects. As a new application development project ends, an application enhancement project sometimes begins. This progression is both natural and predictable and will occur regardless of the process model that is adopted by an organization. Therefore, the major software engineering tasks described in the sections that follow are applicable to all process model flows. As an example, we consider the software engineering tasks for a concept development project.

Concept development projects are initiated when the potential for some new technology must be explored. There is no certainty that the technology will be applicable, but a customer (e.g., marketing) believes that potential benefit exists.



Concept development projects are approached by applying the following major tasks:

Concept scoping determines the overall scope of the project.

- **Preliminary concept planning** establishes the organization's ability to undertake the work implied by the project scope.
- **Technology risk assessment** evaluates the risk associated with the technology to be implemented as part of project scope.
- **Proof of concept** demonstrates the viability of a new technology in the software context.
- **Concept implementation** implements the concept representation in a manner that can be reviewed by a customer and is used for "marketing" purposes when a concept must be sold to other customers or management.
- **Customer reaction to the concept** solicits feedback on a new technology concept and targets specific customer applications.

Refinement of Major Tasks

Refining a major scheduling task (concept scooping) into smaller activities needed to create a detailed project schedule.

The macroscopic schedule must be refined to create a detailed project schedule. Refinement begins by taking each major task and decomposing it into a set of subtasks (with related work products and milestones).

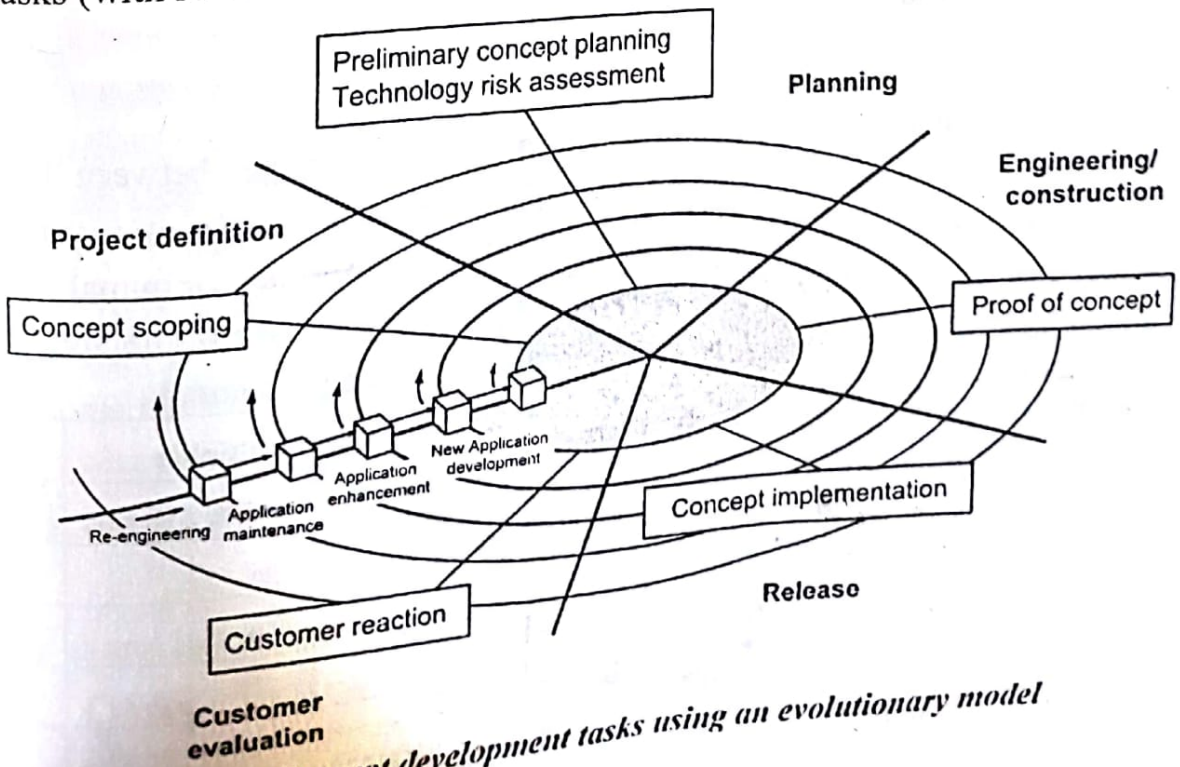


Fig. 6.5. Concept development tasks using an evolutionary model

As an example of task decomposition, consider concept scoping for a development project, Task refinement can be accomplished using an outline format, a process design language approach is used to illustrate the flow of the concept scoping activity. [Figure 6.5].

6.1.8. FACTORS AFFECTING TASK SET/ DEfiNING ADAPTATION CRITERIA

Adaptation criteria are used to determine the recommended degree of rigor with which the software process should be applied on a project. Eleven adaptation criteria are defined for software projects.

- Size of project
- Number of potential users
- Mission criticality
- Application longevity
- Requirement stability
- Ease of customer/developer communication
- Maturity of applicable technology
- Performance constraints
- Embedded/non-embedded characteristics
- Project staffing
- Reengineering factors

Each of the adaptation criteria is assigned a grade that ranges between 1 and 5, where 1 represents a project in which a small subset of process tasks are required and overall methodological and documentation requirements are minimal, and 5 represents a project in which a complete set of process tasks should be applied and overall methodological and documentation requirements are substantial.

Adaptation Criteria	Grade	Weight	Conc.	Entry Point Multiplier				Product
				NDev.	Enhan.	Maint.	Reeng.	
Size of project	2	1.2	-	1	-	-	-	2.4
Number of users	3	1.1	-	1	-	-	-	3.3
Business criticality	4	1.1	-	1	-	-	-	4.4
Longevity	3	0.9	-	1	-	-	-	2.7

Adaptation Criteria	Grade	Weight	Conc.	Entry Point Multiplier				Product
				NDev.	Enhan.	Maint.	Reeng.	
Stability of requirements	2	1.2	-	1	-	-	-	2.4
Ease of communication	2	0.9	-	1	-	-	-	1.8
Maturity of technology	2	0.9	-	1	-	-	-	1.8
Performance constraints	3	0.8	-	1	-	-	-	2.4
Embedded/nonembedded	3	1.2	-	1	-	-	-	3.6
Project staffing	2	1.0	-	1	-	-	-	2.0
Interoperability	4	1.1	-	1	-	-	-	4.4
Reengineering factors	0	1.2	-	0	-	-	-	0.0
Task set selector (TSS)								2.8

Computing a Task Set Selector Value

To select the appropriate task set for a project, the following steps should be conducted:

1. Review each of the adaptation criteria and assign the appropriate grades (1 to 5) based on the characteristics of the project. These grades should be entered into Table.
2. Review the weighting factors assigned to each of the criteria. The value of a weighting factor ranges from 0.8 to 1.2 and provides an indication of the relative importance of a particular adaptation criterion to the types of software developed within the local environment. If modifications are required to better reflect local circumstances, they should be made.
3. Multiply the grade entered in Table 7.1 by the weighting factor and by the entry point multiplier for the type of project to be undertaken. The entry point multiplier takes on a value of 0 or 1 and indicates the relevance of the adaptation criterion to the project type.

The result of the product

grade × weighting factor × entry point multiplier

is placed in the Product column of Table for each adaptation criteria individually.

1. Compute the average of all entries in the Product column and place the result in the space marked task set selector (TSS). This value will be used to help select the task set that is most appropriate for the project.

Interpreting the TSS Value and Selecting the Task Set

Once the task set selector is computed, the following guidelines can be used to select the appropriate task set for a project:

Task set selector value	Degree of rigor
TSS < 1.2	casual
1.0 < TSS < 3.0	structured
TSS > 2.4	strict

The overlap in TSS values from one recommended task set to another is purposeful and is intended to illustrate that sharp boundaries are impossible to define when making task set selections. In the final analysis, the task set selector value, past experience, and common sense must all be factored into the choice of the task set for a project.

From the above table, The project manager selects the grades shown in the Grade column. The project type is new application development. Therefore, entry point multipliers are selected from the NDev column. The entry in the Product column is computed using

Grade × Weight × NewDev entry point multiplier

The value of TSS (computed as the average of all entries in the product column) is 2.8. Using the criteria discussed previously, the manager has the option of using either the structured or the strict task set. The final decision is made once all project factors have been considered.

Defining a Task Network

- Building a task graph or activity network is the key to building a feasible schedule.
- The task graph represents inter-task dependencies very clearly.
- This allows managers to determine which tasks may be done in parallel and which tasks need to done first.

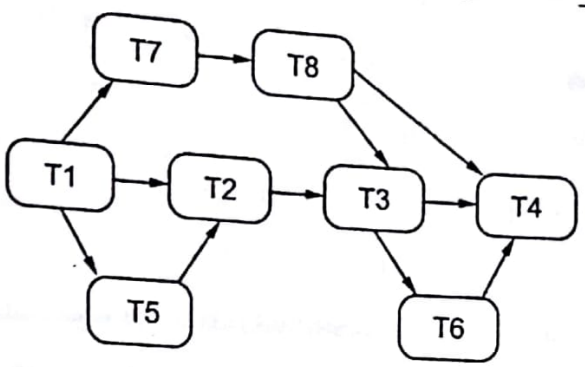


Fig. 6.6.

Project scheduling (Figure 6.7) involves separating the total work involved in a project into separate activities and judging the time required to complete these activities.

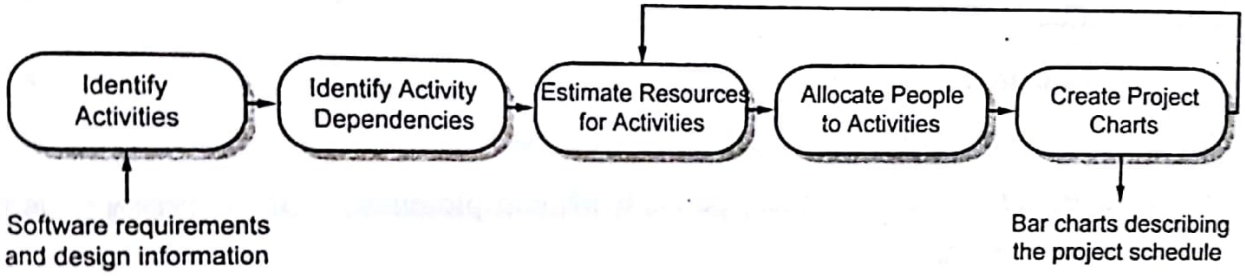


Fig. 6.7. The Project Scheduling Process

Usually, some of these activities are carried out in parallel. You have to coordinate these parallel activities and organize the work so that the workforce is used optimally.

Basic steps followed are: once the tasks dictated by the software process model is refined based on the functionality of the system , effort and duration are allocated for each task and an activity network is created that allows the project to meets its deadlines. The work product of this activity is the project schedule and in order that it is accurate it is required to check all tasks are covered in the activity network, effort and timing are appropriately allocated, interdependencies are correctly indicated, resources are allocated tasks in a right manner and closely spaced milestones are defined to track the project easily.

Project scheduling methods

- Program evaluation and review technique (PERT)
- critical path method (CPM)

Timeline Charts

- Gantt charts

Tracking the Schedule

Several ways to track a project schedule:

- conducting periodic project status meeting
- evaluating the review results in the software process
- determine if formal project milestones have been accomplished (Figure)
- compare actual start date to planned start date for each task
- informal meeting with practitioners

Earned Value Analysis (EVA)

Earned value -

- is a measure of progress
- provides a quantitative indication of progress
- enables us to assess the "percent of completeness" of a project using quantitative analysis
- Earned Value Analysis (EVA) is a quantitative technique for assessing progress.

The total hours to complete the entire project are estimated and each task is given an earned value based on its estimated percentage contribution to the total

These techniques are driven by information such as estimates of effort, decomposition of the product function, the selection of process model, task set and decomposition of tasks. The interdependencies among tasks are defined using a task network. According to basic PERT, expected task duration is calculated as the weighted average of the most pessimistic, the most optimistic and most probable time estimates. The expected duration of any path on the network is found by summing the expected durations of tasks. PERT gives appropriate results when there is a single dominant path in the network. The time needed to complete the project is defined by the longest path in the network which is called critical path. CPM allows software planner to determine the critical path and establish most likely time estimates.

While creating schedule a timeline chart also called as Gantt chart can be generated. This can be developed for entire project or separately for each function or individual.

Large organizations usually employ a number of specialists who work on a project when needed.

In Figure 6.11, you can see that Mary and Jim are specialists who work on only a single task in the project. This can cause scheduling problems.

If one project is delayed while a specialist is working on it, this may have a knock-on effect on other projects. They may also be delayed because the specialist is not available.

6.2. SOFTWARE CONFIGURATION MANAGEMENT

Why Software Configuration Management?

- The problem:
  - Multiple people have to work on software that is changing
  - More than one version of the software has to be supported:
  - Released systems
  - Custom configured systems (different functionality)
  - System(s) under development
  - Software must run on different machines and operating systems
- Need for coordination
- Software Configuration Management
  - manages evolving software systems
  - controls the costs involved in making changes to a system

The process of software development and maintenance is controlled is called configuration management. The configuration management is different in development and maintenance phases of life cycle due to different environments.

Software Configuration Management (SCM) is a set of management disciplines within the software engineering process to develop a baseline.

Software Configuration Management encompasses the disciplines and techniques of initiating, evaluating and controlling change to software products during and after the software engineering process.

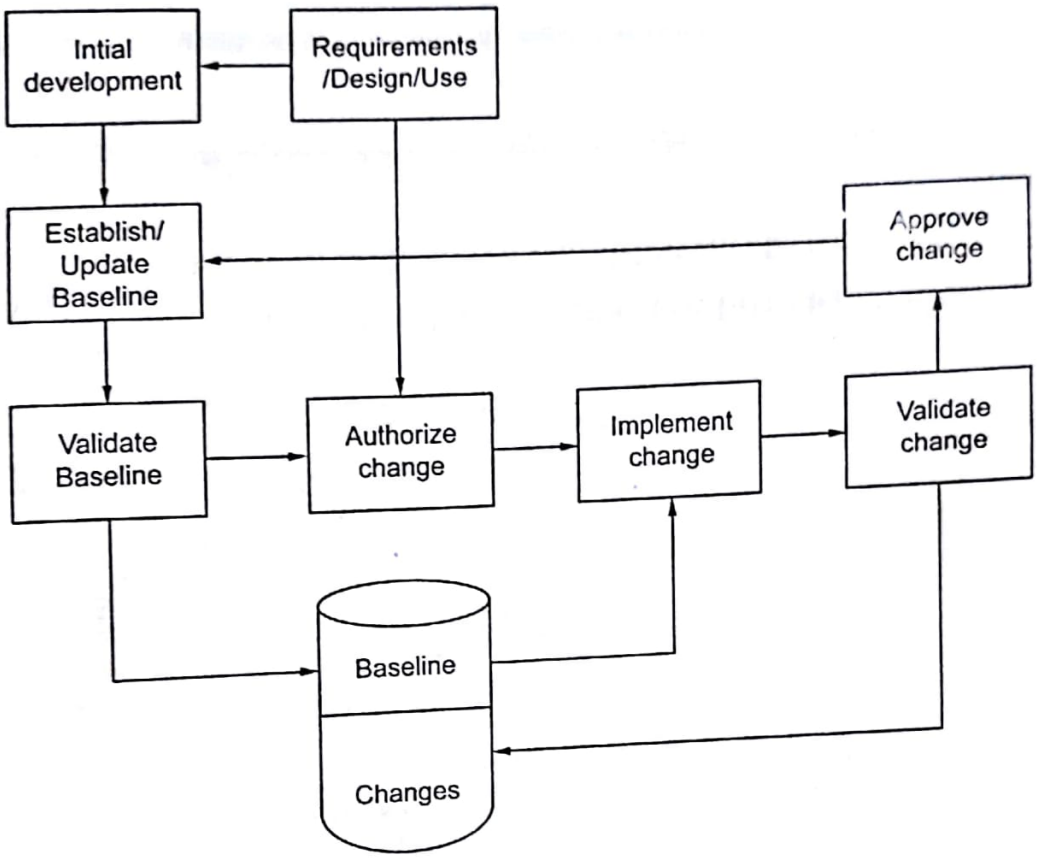


Fig. 6.12.

Configuration managers are responsible for keeping track of the differences between software versions, for ensuring that new versions are derived in a controlled way and for releasing new versions to the right customers at the right time.

The CM plan should be organised into a number of sections that:

1. Define what is to be managed (the configuration items) and the scheme that you should use to identify these entities.
2. Set out who is responsible for the configuration management procedures and for submitting controlled entities to the configuration management team.
3. Define the configuration management policies that all team members must use for change control and version management.
4. Specify the tools that you should use for configuration management and the processes for using these tools.
5. Describe the structure of the configuration database that is used to record configuration information and the information that should be maintained in that database (the configuration records).

The configuration item identification scheme must assign a unique name to all documents under configuration control.

Therefore, you might define a hierarchical naming scheme with names such as:

Example:

PCL-TOOLS/EDIT/FORMS/DISPLAY/AST-INTERFACE/CODE

PCL-TOOLS/EDIT/HELP/QUERY/HELPFRAMES/FR-1

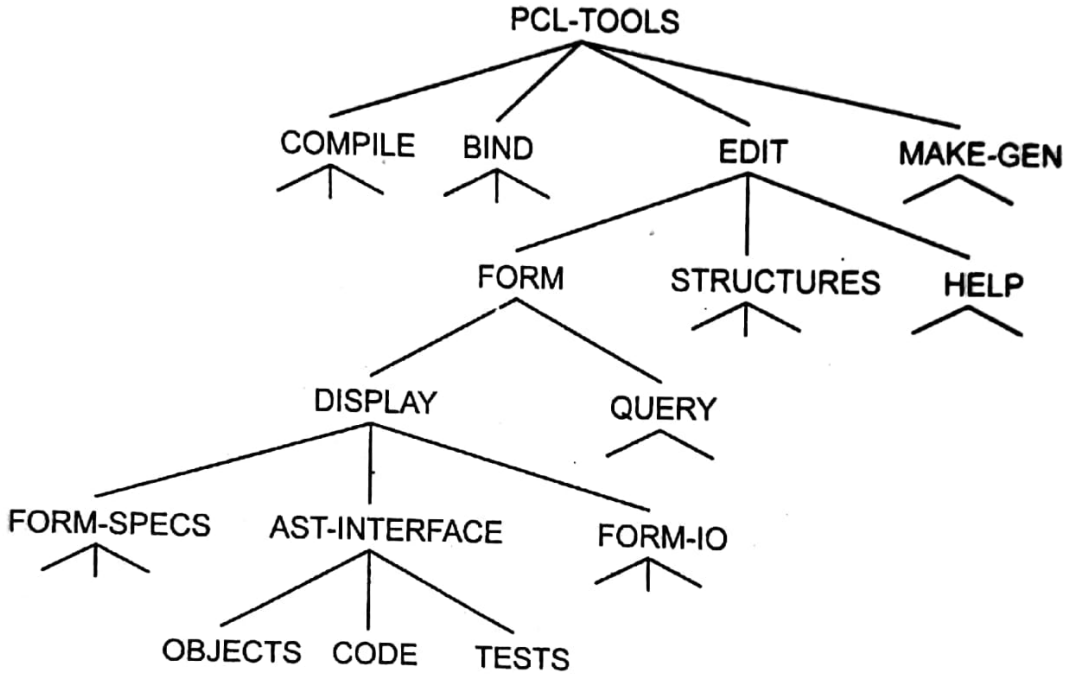


Fig. 6.13. A configuration hierarchy used to assign item identifiers

The configuration database

The configuration database is used to record all relevant information about system configurations and configuration items.

- Use the CM database to help assess the impact of system changes and to generate reports for management about the CM process. As part of the CM planning process, you should define the CM database schema, the forms to collect information to be recorded in the database and procedures for recording and retrieving project information.
- A configuration database does not just include information about configuration items.
- It may also record information about users of components, system customers, execution platforms, proposed changes and so forth.

Software Configuration Management (SCM) Activities

- Configuration item identification
  - ➢ modeling of the system as a set of evolving components
- Promotion management
  - ➢ is the creation of versions for other developers
- Release management
  - ➢ is the creation of versions for the clients and users
- Branch management
  - ➢ is the management of concurrent development
- Variant management
  - ➢ is the management of versions intended to coexist
- Change management
  - ➢ is the handling, approval and tracking of change requests

The following documents are required for these activities

- Project plan
- Software requirements specification document
- Software design description document
- Source code listing
- Test plans / procedures / test cases
- User manuals

SCM Roles

- Configuration Manager
  - ➢ Responsible for identifying configuration items. The configuration manager can also be responsible for defining the procedures for creating promotions and releases
- Change control board member
  - ➢ Responsible for approving or rejecting change requests
- Developer
  - ➢ Creates promotions triggered by change requests or the normal activities of development. The developer checks in changes and resolves conflicts

- Auditor
  - ➢ Responsible for the selection and evaluation of promotions for release and for ensuring the consistency and completeness of this release

What are,

- Configuration Items
- Baselines
- SCM Directories
- Versions, Revisions and Releases

Configuration Item

"An aggregation of hardware, software, or both, that is designated for configuration management and treated as a single entity in the configuration management process."

- ❖ Software configuration items are not only program code segments but all type of documents according to development, e.g
  - all type of code files
  - drivers for tests
  - analysis or design documents
  - user or developer manuals
  - system configurations (e.g. version of compiler used)

In some systems, not only software but also hardware configuration items (CPUs, bus speed frequencies) exist.

Baseline

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

Controlled systems are sometimes called baselines because they are a starting point for further, controlled evolution.

Examples:

- Baseline A: The API of a program is completely defined; the bodies of the methods are empty.

- Baseline B: All data access methods are implemented and tested programming of the GUI can start.
- Baseline C: GUI is implemented, test-phase can start.

SCM Directories

- Programmer's Directory (IEEE: Dynamic Library)
  - Library for holding newly created or modified software entities. The programmer's workspace is controlled by the programmer only.
- Master Directory (IEEE: Controlled Library)
  - Manages the current baseline(s) and for controlling changes made to them. Entry is controlled, usually after verification. Changes must be authorized.
- Software Repository (IEEE: Static Library)
  - Archive for the various baselines released for general use. Copies of these baselines may be made available to requesting organizations.

Version vs. Revision vs. Release

Version

- An initial release or re-release of a configuration item associated with a complete compilation or recompilation of the item. Different versions have different functionality.

Revision

- Change to a version that corrects only errors in the design/code, but does not affect the documented functionality.

Release

- The formal distribution of an approved version.

Software Versions

- Two types of versions namely revisions (replace) and variations (variety).

Version Control

Version: An instance of a system which is functionally distinct in some way from other system instances.

A version control tool is the first stage towards being able to manage multiple versions. Once it is in place, a detailed record of every version of the software must be kept.

This comprises the

- Name of each source code component, including the variations and revisions
- The versions of the various compilers and linkers used
- The name of the software staff who constructed the component
- The date and the time at which it was constructed

Example

```
// BANKSEC project (IST 6087)
//
// BANKSET-TOOLS/AUTH/RBAC/USER_ROLE
//
// Object: currentRole
// Author: N. Perwaiz
// Creation date: 10th November 2002
//
// (c) Lancaster University 2002
//
// Modification history
// Version   Modifier    Date        Change      Reason
// 1.0       J. Jones    1/12/2002   Add header  Submitted to CM
// 1.1       N. Perwaiz  9/4/2003    New field   Change req. R07/02
```

A system version is an instance of a system that differs, in some way, from other instances. Versions of the system may have different functionality, enhanced performance or repaired software faults. Some versions may be functionally equivalent but designed for different hardware or software configurations.

Variants

Versions with only small differences are sometimes called variants. It is an instance of a system which is functionally identical but non-functionally distinct from other instances of a system

A system release is a version that is distributed to customers. Each system release should either include new functionality or should be intended for a different hardware platform. There are normally many more versions of a system than releases.

Versions are created within an organisation for internal development or testing and are not intended for release to customers.

Version identification

To create a particular version of a system, you have to specify the versions of the system components that should be included in it.

In a large software system, there are hundreds of software components, each of which may exist in several different versions. There must therefore be an unambiguous way to identify each component version to ensure that the right components are included in the system. However, you cannot use the configuration item name for version identification because there may be several versions of each identified configuration item.

Instead, three basic techniques are used for component version identification

1. Version numbering: The component is given an explicit, unique version number. This is the most commonly used identification scheme.
2. Attribute-based identification: Each component has a name (such as the configuration item name, which is not unique across versions) and an associated set of attributes for each version (Estublier and Casallas, 1994). Components are therefore identified by specifying their name and attribute values.
3. Change-oriented identification: Each component is named as in attribute-based identification but is also associated with one or more change requests (Munch,et al., 1993). That is, it is assumed that each version of the component has been created in response to one or more change requests. The component version is identified by the set of change requests that apply to the component.

Change Control Process

Change control process comes into effect when the software and associated documentation are delivered to configuration management change request form (as shown in fig.), which should record the recommendations regarding the change.

Change management process is depicted below:

Request change by completing a change request form
Analyze change request
if change is valid **then**
    Assess low change might be implemented
    Assess change cost
    Record change request in database
    Submit request to change control board
    if change is accepted **then**
        repeat
            make changes to software
            record changes and link to associated change request
            submit changed software for quality approval
        until software quality is adequate
        create new system version
    else
        reject change request
else
    reject change request

Fig. 6.14. Change Control Process

Change Request Form	
Project: Proteus/PCL-Tools	**Number:** 23/02
Change requester: 1. Sommerville	**Date:** 1/12/02
Requested change: When a component is selected from the structure, display the name of the file where it is started.	
Change analyser: G. Dean	**Analysis date:** 10/12/02
Components affected: Display-Icon, Select, Display-Icon, Display	
Associated components: File Table	
Change assessment: Relatively simple to implement as a file name table is available. Requires the design and implementation of a display field. No changes to associated components are required.	

Change priority: Low	
Change implementation:	
Estimated effort: 15/12/02	**CCB decision date:** 1/2/03
CCB decision: Accept change. Change to be implemented in **Release** 2.1.	
Change implementer:	**Date of change:**
Date submitted to QA:	**QA decision:**
Date submitted to CM:	
Comments	

Fig. 6.15. Change Request Form

The first stage in the change management process is to complete a change request form (CRF) describing the change required to the system. As well as recording the change required, the CRF records the recommendations regarding the change, the estimated costs of the change and the dates when the change was requested approved, implemented and validated. The CRF may also include a section where an analyst outlines how the change is to be implemented.

Once a change request form has been submitted, it should be registered in the configuration database. The change request is then analysed to check that the change requested is necessary.

System building

System building is the process of compiling and linking software components into a program that executes on a particular target configuration.

Example

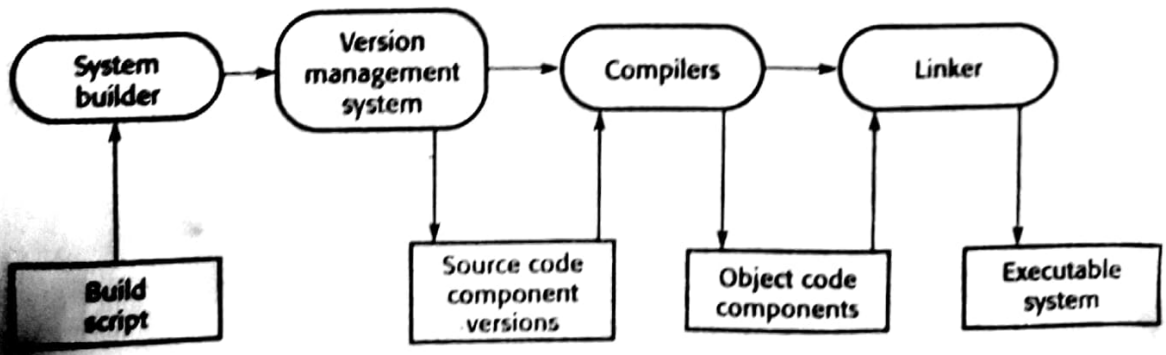


Fig. 6.16. System building

Release management

Release: An instance of a system which is distributed to users outside of the development team

A system release is a version of the system that is distributed to customers. System release managers are responsible for deciding when the system can be released to customers, managing the process of creating the release and the distribution media, and documenting the release to ensure that it may be re-created exactly as distributed if this is necessary.

A system release is not just the executable code of the system.

The release may also include:

1. Configuration files defining how the release should be configured for particular installations
2. Data files that are needed for successful system operation
3. An installation program that is used to help install the system on target hardware
4. Electronic and paper documentation describing the system

Tools for Software Configuration Management

- VSS – Visual Source safe is a centralized version control system from Microsoft.
- CVS - Concurrent version system is an open source version controlling system which lets group of people work simultaneously on group of files
- Rational Clear Case is a configuration management tool developed by IBM.
- SVN- Subversion is the most popular open source Configuration Management tool used for version controlling
- Perforce is a version controlling software developed by Perforce Software Inc. here the users connect to shared file repository and files are transferred between the repositories and individual workstation
- Tortoise SVN
- IBM Rational team concert
- IBM Configuration management version management
- Razor

- Quma version control system
- Source Anywhere

6.2.1. CASE TOOLS FOR CONFIGURATION MANAGEMENT

This chapter discusses the use of computer-aided software engineering (CASE) tools that can assist software engineering managers and practitioners with every activity associated with the software development process.

CASE stands for Computer Aided Software Engineering which is software that supports one or more software engineering activities within a software development process, and is gradually becoming popular for the development of software as they are improving in the capabilities and functionality and are proving to be beneficial for the development of quality software.

Computer-aided software engineering (CASE) tools assist software engineering managers and practitioners in every activity associated with the software process. They automate project management activities, manage all work products produced throughout the process, and assist engineers in their analysis, design, coding and test work. CASE tools can be integrated within a sophisticated environment.

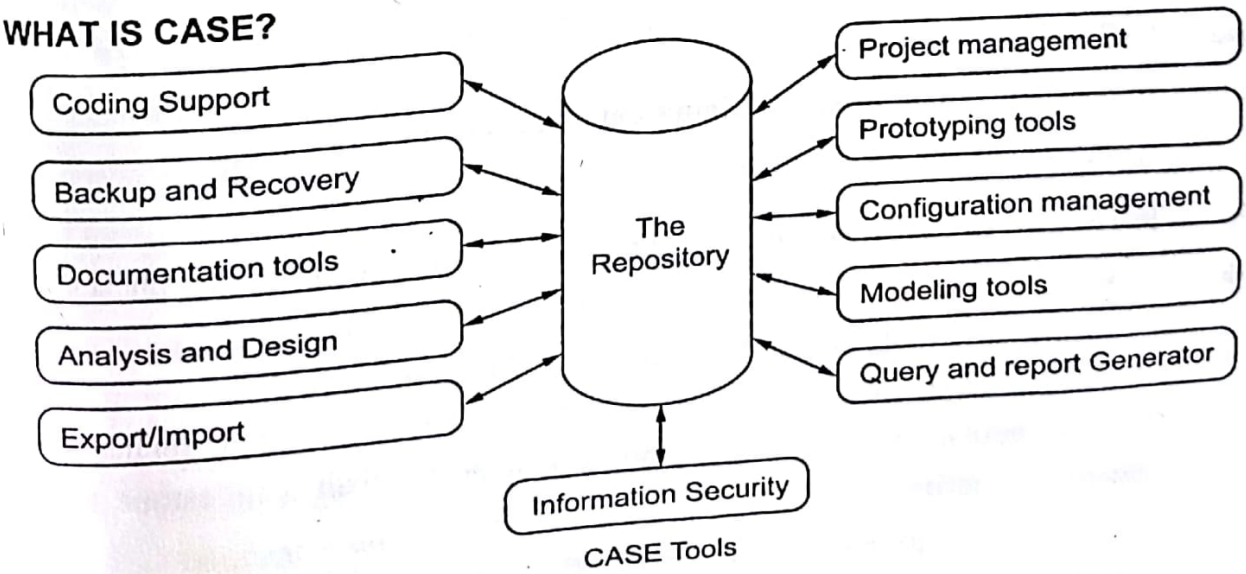


Fig. 6.17. Environment having CASE

A good workshop for any craftsperson - a mechanic, a carpenter, or a software engineer - has three primary characteristics:

(1) a collection of useful tools that will help in every step of building a product,
(2) an organized layout that enables tools to be found quickly and used efficiently, and
(3) a skilled artisan who understands how to use the tools in an effective manner.

The workshop for software engineering has been called an integrated project support environment and the tools that fill the workshop are collectively called computer-aided software engineering

Categories of CASE Tools

Sometimes CASE tools are classified in to following categories due to their activities:

1. UPPER CASE Tools
2. LOWER CASE Tools
3. INTEGRATED CASE Tools

Upper CASE tools

They support the analysis and the design phase. They include tools for analysis modeling, reports and forms generation.

Lower CASE tools

They support the coding phase, configuration management, etc.

Integrated CASE tools

- It is known as I-CASE and also supports analysis, design and coding phases.
- CASE tools can automate management activities and can manage work products. CASE tools can assist engineers with analysis, design, coding, and testing work. Software engineering is hard work and tools that reduce the effort required to produce a work product or accomplish a milestone have substantial benefits.
- CASE tools assist the software engineer in producing high quality work products. Tools can provide the software engineer with improved insight into the software product and make decisions that lead to improved software quality.
- CASE tool support is essential for configuration management

These tools can be combined to create a configuration management workbench to support all CM activities. There are two types of CM workbench:

1. Open workbenches Tools for each stage in the CM process are integrated through standard organizational procedures for using these tools.
2. Integrated workbenches These workbenches provide integrated facilities for version management, system building and change tracking.

It involves in the support of the below

1. Change management
2. System building
3. Version Management

Support for change management

Each person involved in the change management process is responsible for some activity. They complete this activity, then pass on the forms and associated configuration items to someone else.

The procedural nature of this process means that a change process model can be designed and integrated with a version management system. This model may then be interpreted so that the right documents are passed to the right people at the right time.

There are several change management tools available, from relatively simple, open-source tools such as Bugzilla to comprehensive integrated systems such as Rational ClearQuest.

These tools provide some or all of the following facilities to support the process:

1. A form editor that allows change proposal forms to be created and completed by people making change requests.
2. A workflow system that allows the CM team to define who must process the change request form and the order of processing. This system will also automatically pass forms to the right people at the right time and inform the relevant team members of the progress of the change. E-mail is used to provide progress updates for those involved in the process.
3. A change database that is used to manage all change proposals and that may be linked to a version management system. Database query facilities allow the CM team to find specific change proposals.

4. A change-reporting system that generates management reports on the status of change requests that have been submitted.

Support for version management

- Version management involves managing large amounts of information and ensuring that system changes are recorded and controlled.
- Version management tools control a repository of configuration items where the contents of that repository are immutable (i.e., cannot be changed). To work on a configuration item, you must check it out of the repository into a working directory. After you have made the changes to the software, you check it back into the repository and a new version is automatically created.

All version management systems provide a comparable basic set of capabilities although some have more sophisticated facilities than others. Examples of these capabilities are:

1 Version and release identification Managed versions are assigned identifiers when they are submitted to the system.

2. Storage management To reduce the storage space required by multiple versions that are largely the same, version management systems provide storage management facilities so that versions are described by their differences from some master version. Differences between versions are represented as a delta, which encapsulates the instructions required to recreate the associated system version.

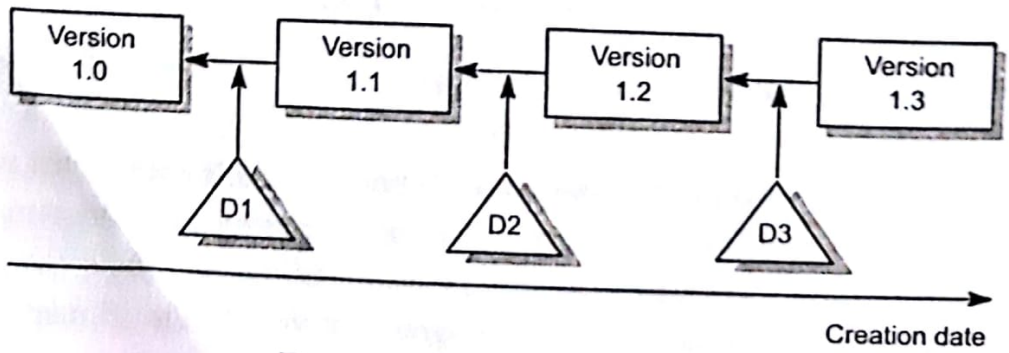


Fig. 6.18. Delta based versioning

3. Change history recording All of the changes made to the code of a system or component are recorded and listed. In some systems, these changes may be used to select a particular system version.

4. Independent development Multiple versions of a system can be developed in parallel and each version may be changed independently. For example, release 1 can be modified after development of release 2 is in progress by adding new level-1 deltas. The version management system keeps track of components that have been checked out for editing and ensures that changes made to the same component by different developers do not interfere. Some systems allow only one instance of a component to be checked out for editing; others resolve potential clashes when the edited components are checked back into the system.
5. Project support The system can support multiple projects as well as multiple files. In project support systems, such as CVS, it is possible to check in and check out all of the files associated with a project rather than having to work with one file at a time.

Support for system building

System building is a computationally intensive process. Compiling and linking all of the components of a large system can take several hours. There may be hundreds of files involved, with the consequent possibility of human error if these are compiled and linked manually. System-building tools automate the build process to reduce the potential for human error and, where possible, minimise the time required for system building.

System-building tools may be standalone, such as derivatives of the Unix make utility (Oram and Talbott, 1991), or may be integrated with version management tools.

Facilities provided by system-building CASE tools may include:

1. A dependency specification language and associated interpreter Component dependencies may be described and recompilation minimized.
2. Tool selection and instantiation support The compilers and other processing tools that are used to process the source code files may be specified and instantiated as required.
3. Distributed compilation Some system builders, especially those that are part of integrated CM systems, support distributed network compilation. Rather than all compilations being carried out on a single machine, the system builder

looks for idle processors on the network and sets off a number of parallel compilations. This significantly reduces the time required to build a system.

4. Derived object management Derived objects are objects created from other source objects. Derived object management links the source code and the derived objects and rederives only an object when this is required by source code changes.

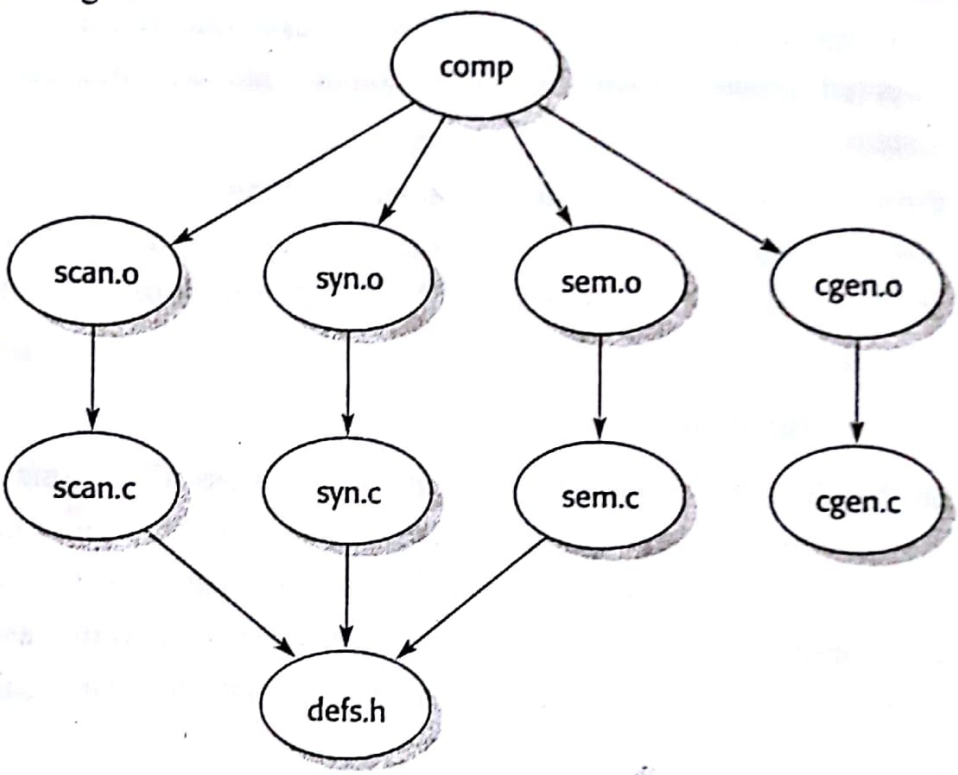


Fig. 6.19. Variable and constant declarations called defs.h shared by scan.c, syn.c and sem.c

Managing derived objects and minimising recompilation is best explained using a simple example. Consider a situation where a compiler program called comp is created out of four object modules named scan.o, syn.o, sem.o and cgen.o. Each object module is created from a source code module with corresponding names (scan.c, syn.c, sem.c and cgen.c).

A file of variable and constant declarations called defs.h is shared by scan.c, syn.c and sem.c (Figure 6.19). In Figure 6.19, the arrows mean 'depends on'—the entity at the base of the arrow depends on the entity at its head. Therefore, comp depends on scan.o, syn.o, sem.o and cgen.o, scan.o depends on scan.c, and so on.

If scan.c is changed, the system-building tool can detect that the derived object scan.o must be re-created. It does this by comparing the modification times of scan.o

and scan.c and detects that scan.c has been modified after scan.o. It then calls the C compiler to compile scan.c to create a new derived object, scan.o.

The build tool then uses the dependency link between comp and scan.o to detect that comp must be recreated by linking scan.o, syn.o, sem.o and cgen.o. The system can detect that the other object code components are unchanged, so recompilation of their source code is not required.

Prerequisites to Software Tool Use

- Collection of useful tools that help in every step of building a product
- Organized layout that enables tools to be found quickly and used efficiently
- Skilled craftsperson who understands how to use the tools effectively

CASE Building Blocks

- CASE tools.
- Integration framework (specialized programs allowing CASE tools to communicate with one another).
- Portability services (allow CASE tools and their integration framework to migrate across different operating systems and hardware platforms without significant adaptive maintenance).
- Operating system (database and object management services).
- Hardware platform.
- Environmental architecture (hardware and system support).

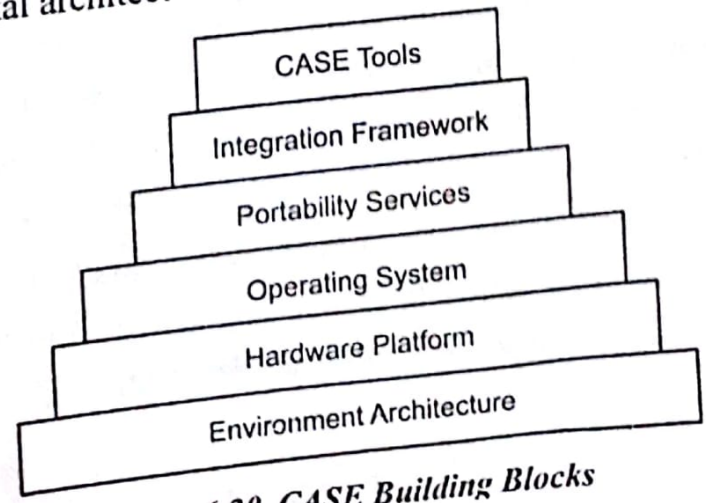


Fig. 6.20. CASE Building Blocks

CASE Tool Components

- Integration framework
- specialized programs allowing CASE tools to communicate
- Portability services
- Operating system
- database and object management services
- Hardware platform

The environment architecture, composed of the hardware platform and system support (including networking software, database management, and object management services), lays the ground work for CASE. But the CASE environment itself demands other building blocks.

A set of portability services provides a bridge between CASE tools and their integration framework and the environment architecture.

The integration framework is a collection of specialized programs that enables individual CASE tools to communicate with one another, to create a project database, and to exhibit the same look and feel to the end-user (the software engineer).

Portability services allow CASE tools and their integration framework to migrate across different hardware platforms and operating systems without significant adaptive maintenance.

Most CASE tools in use today have not been constructed using all these building blocks. In fact, some CASE tools remain "point solutions." That is, a tool is used to assist in a particular software engineering activity (e.g., analysis modeling) but does not directly communicate with other tools, is not tied into a project database, is not part of an integrated CASE environment (ICASE). Although this situation is not ideal, a CASE tool can be used quite effectively, even if it is a point solution.

Point-solution CASE tools can provide substantial individual benefit, but a software team needs tools that talk to one another. Integrated tools help the team develop, organize, and control work products. Use them.

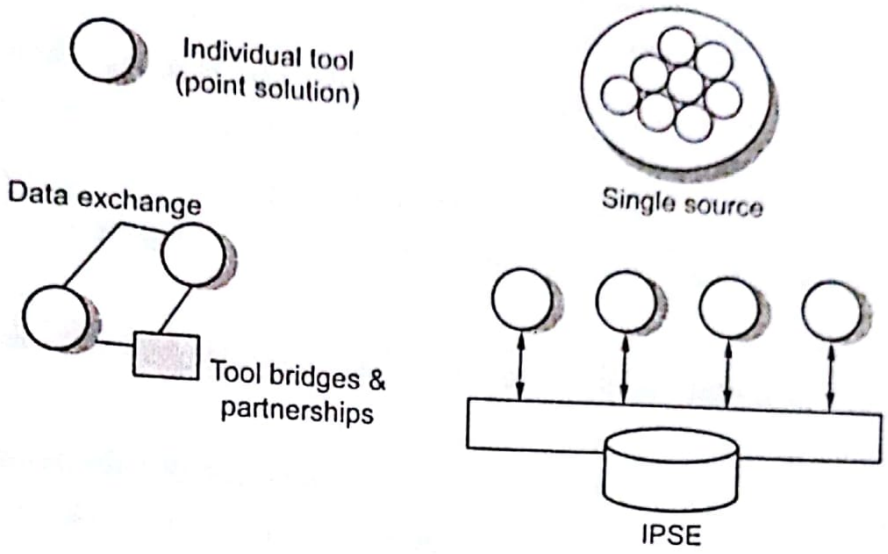


Fig. 6.21. CASE Tool Components

The relative levels of CASE integration are shown in Figure above. At the low end of the integration spectrum is the individual (point solution) tool. When individual tools provide facilities for data exchange (most do), the integration level is improved slightly. Such tools produce output in a standard format that should be compatible with other tools that can read the format. In some cases, the builders of complementary CASE tools work together to form a bridge between the tools (e.g., an analysis and design tool that is coupled with a code generator).

Using this approach, the synergy between the tools can produce end products that would be difficult to create using either tool separately.

Single-source integration occurs when a single CASE tools vendor integrates a number of different tools and sells them as a package. Although this approach is quite effective, the closed architecture of most single-source environments precludes easy addition of tools from other vendors.

At the high end of the integration spectrum is the integrated project support environment (IPSE). Standards for each of the building blocks described previously have been created. CASE tool vendors use IPSE standards to build tools that will be compatible with the IPSE and therefore compatible with one another.

6.2.2. CASE TOOL TAXONOMY

CASE tools can be classified by function, by their role as instruments for managers or technical people, by their use in the various steps of the software

engineering process, by the environment architecture that supports them, or even by their origin or cost. The taxonomy presented here uses functions a primary criterion.

- Business process engineering tools - represent business data objects, their relationships, and flow of the data objects between company business areas
- Process modeling and management tools - represent key elements of processes and provide links to other tools that provide support to defined process activities
- Project planning tools - used for cost and effort estimation, and project scheduling
- Risk analysis tools - help project managers build risk tables by providing detailed guidance in the identification and analysis of risks
- Requirements tracing tools - provide systematic database-like approach to tracking requirement status beginning with specification
- Metrics and management tools -management oriented tools capture project specific metrics that provide an overall indication of productivity or quality, technically oriented metrics determine metrics that provide greater insight into the quality of design or code
- Documentation tools - provide opportunities for improved productivity by reducing the amount of time needed to produce work products
- System software tools - network system software, object management services, distributed component support, and communications software
- Quality assurance tools -metrics tools that audit source code to determine compliance with language standards or tools that extract metrics to project the quality of software being built
- Database management tools - RDMS and OODMS serve as the foundation for the establishment of the CASE repository
- Software configuration management tools -uses the CASE repository to assist with all SCM tasks (identification, version control, change control, auditing, status accounting)
- Analysis and design tools -enable the software engineer to create analysis and design models of the system to be built, perform consistency checking between models

- PRO/SIM tools - prototyping and simulation tools provide software engineers with ability to predict the behavior of real-time systems before they are built and the creation of interface mockups for customer review
- Interface design and development tools - toolkits of interface components, often part environment with a GUI to allow rapid prototyping of user interface designs
- Prototyping tools - enable rapid definition of screen layouts, data design, and report generation
- Programming tools - compilers, editors, debuggers, OO programming environments, fourth generation languages, graphical programming environments, applications generators, and database query generators
- Web development tools - assist with the generation of web page text, graphics, forms, scripts, applets, etc.
- Integration and testing tools: In their directory of software testing tools, Software Quality Engineering [SQE95] defines the following testing tools categories Data acquisition, Static measurement, Dynamic measurement, Simulation, Test management, Cross-functional tools.
  - ❖ Data acquisition (get data for testing)
  - ❖ static measurement (analyze source code without using test cases)
  - ❖ dynamic measurement (analyze source code during execution)
  - ❖ simulation (simulate function of hardware and other externals)
  - ❖ test management (assist in test planning, development, and control)
  - ❖ cross-functional (tools that cross test tool category boundaries)
- Static analysis tools - code-based testing tools, specialized testing languages, requirements-based testing tools
- Dynamic analysis tools - intrusive tools modify source code by inserting probes to check path coverage, assertions, or execution flow, non-intrusive tools use a separate hardware processor running in parallel with processor containing the program being tested
- Test management tools - coordinate regression testing, compare actual and expected output, conduct batch testing, and serve as generic test drivers

- Client/server testing tools - exercise the GUI and network communications requirements for the client and server
- Reengineering tools
  - ❖ reverse engineering to specification tools - generate analysis and design models from source code, where used lists, and other design information
  - ❖ code restructuring and analysis tools -analyze program syntax, generate control flow graph, and automatically generates a structured program
  - ❖ on-line system reengineering tools - used to modify on-line DBMS

6.2.3. INTEGRATED CASE ENVIRONMENTS

Integration of a variety of tools and information that enables closure of communication among tools, between people and across the software process. Combination of CASE tools in an environment where interface mechanisms are standardized.

The benefits of integrated CASE (I-CASE) include

(1) smooth transfer of information (models, programs, documents, data) from one tool to another and one software engineering step to the next;

(2) a reduction in the effort required to perform umbrella activities such as software configuration management, quality assurance, and document production;

(3) an increase in project control that is achieved through better planning, monitoring, and communication; and

(4) improved coordination among staff members who are working on a large software project.

But I-CASE also poses significant challenges. Integration demands consistent representations of software engineering information, standardized interfaces between tools, a homogeneous mechanism for communication between the software engineer and each tool, and an effective approach that will enable I-CASE to move among various hardware platforms and operating systems. Comprehensive I-CASE environments have emerged more slowly than originally expected. However, integrated environments do exist and are becoming more powerful as the years pass.

The term integration implies both combination and closure. I-CASE combines a variety of different tools and a spectrum of information in a way that enables closure

of communication among tools, between people, and across the software process. Tools are integrated so that software engineering information is available to each tool that needs it; usage is integrated so that a common look and feel is provided for all tools; a development philosophy is integrated, implying a standardized software engineering approach that applies modern practice and proven methods.

6.2.4. INTEGRATION FRAMEWORK DIAGRAM

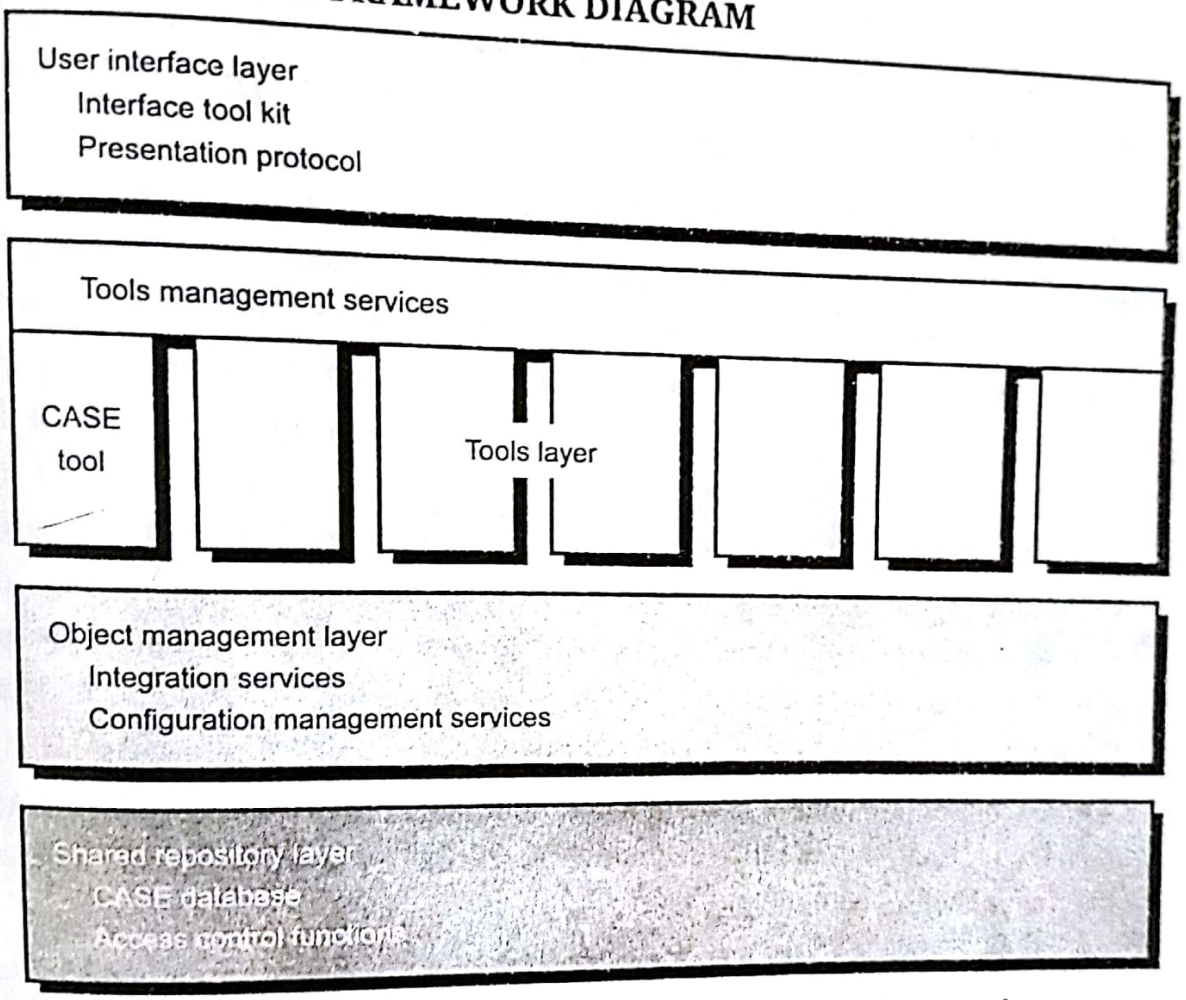


Fig. 6.22. Architectural model for the integration framework

A software engineering team uses CASE tools, corresponding methods, and a process framework to create a pool of software engineering information. The integration framework facilitates transfer of information into and out of the pool.

To accomplish this, the following architectural components must exist: a database must be created (to store the information); an object management system must be built (to manage changes to the information); a tools control mechanism must be constructed (to coordinate the use of CASE tools); a user interface must provide a

consistent pathway between actions made by the user and the tools contained in the environment. Most models (e.g., [FOR90], [SHA95]) of the integration framework represent these components as layers.

- The user interface layer consists of interface tool kit and presentation protocols. The user interface layer incorporates a standardized interface tool kit with a common presentation protocol. The interface tool kit contains software for human/computer interface management and a library of display objects. Both provide a consistent mechanism for communication between the interface and individual CASE tools.
- The presentation protocol is the set of guidelines that gives all CASE tools the same look and feel. Screen layout conventions, menu names and organization, icons, object names, the use of the keyboard and mouse, and the mechanism for tools access are all defined as part of the presentation protocol.
- The tools layer incorporates a set of tools management services with the CASE tools themselves. Tools management services (TMS) control the behavior of tools within the environment. If multitasking is used during the execution of one or more tools, TMS performs multitask synchronization and communication, coordinates the flow of information from the repository and object management system into the tools, accomplishes security and auditing functions, and collects metrics on tool usage.
- The presentation protocol decides a common look and feel of the presentation interface. Then it comes a tools layer. IT consists of set of tools management services (TMS).
- The object management layer (OML) performs the configuration management functions described in Chapter 9. In essence, software in this layer of the framework architecture provides the mechanism for tools integration. Every CASE tool is "plugged into" the object management layer. Working in conjunction with the CASE repository, the OML provides integration services—a set of standard modules that couple tools with the repository. In addition, the OML provides configuration management services by enabling the identification of all configuration objects, performing version control, and providing support for change control, audits, and status accounting.

- The shared repository layer is the CASE database and the access control functions that enable the object management layer to interact with the database.
- The services of this layer allow the identification of all the configuration objects. So that the case tool can be plugged into the integrated CASE environment.
- The bottom most layer is shared repository layer. It consists of CASE database and access control functions. These access control functions help the object management layer to access the CASE database.

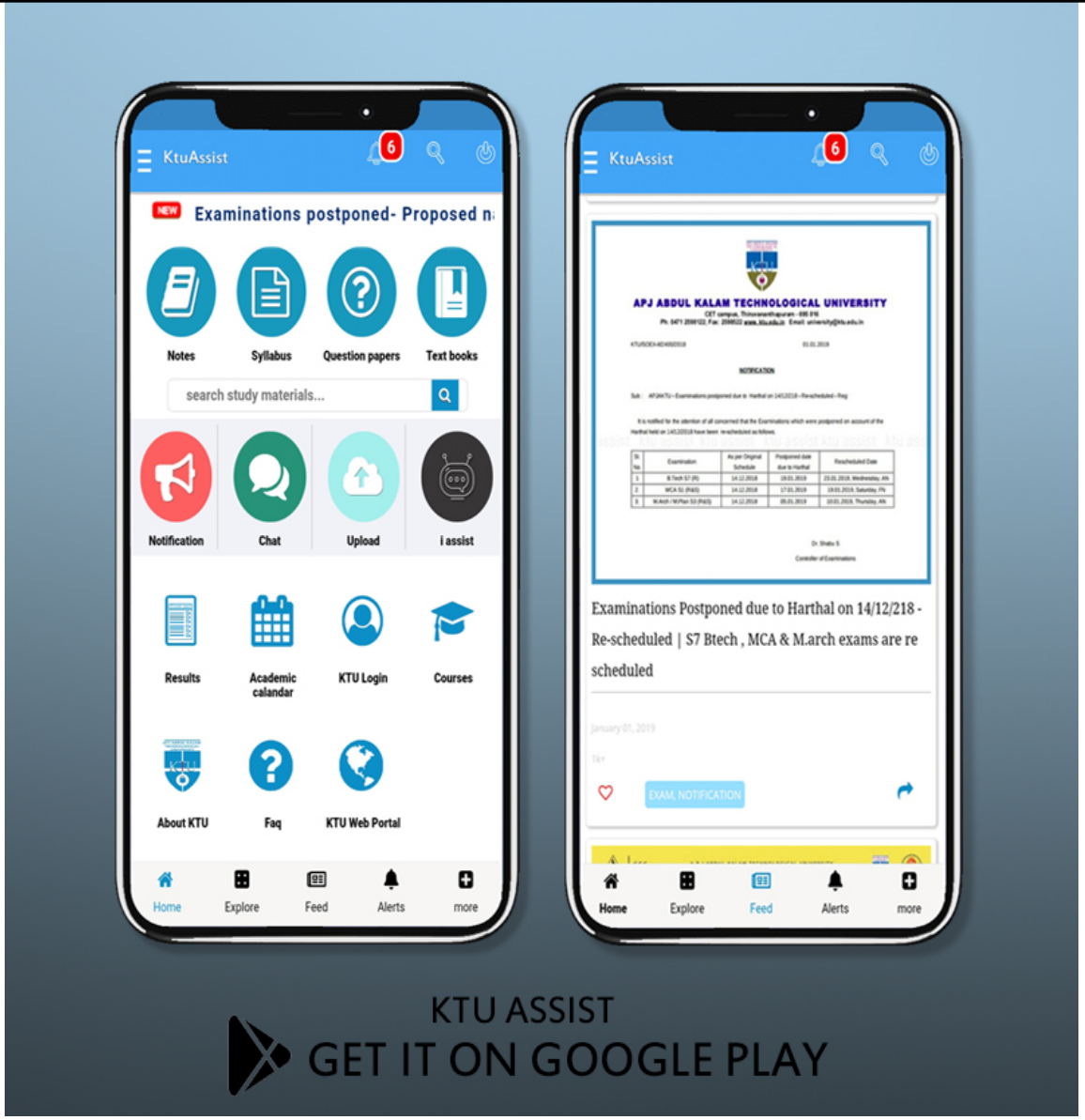
These provides

- Provide mechanism for sharing information among all tools contained in the environment
- Enable changes to items to be tracked to other information items
- Provide version control and overall configuration management
- Allow direct access to any tool contained in the environment
- Establish automated support for the chosen software process model, integrating CASE tools and SCI's into a standard work break down structure
- Enable users of each tool to experience a consistent look and feel at the human-computer interface
- Support communication among software engineers
- Collect both management and technical metrics to improve the process and the product

CASE Repository Functions (The Role of the Repository in I-CASE)

The repository for an I-CASE environment is the set of mechanisms and data structures that achieve data/tool and data/data integration. It provides the obvious functions of a database management system, but in addition, the repository performs or precipitates the following functions [FOR89b]:

- Data integrity - includes functions to validate entries to the repository and ensure consistency among related objects
- Information sharing - provides mechanism for sharing information among multiple developers and multiple tools, controls modification of information



END

facebook.com/ktuassist

instagram.com/ktu_assist