

UNIT I

SOFTWARE PROCESS AND PROJECT MANAGEMENT

Introduction to Software Engineering, Software Process, Perspective and Specialized Process Models – Software Project Management: Estimation – LOC and FP Based Estimation, COCOMO Model – Project Scheduling – Scheduling, Earned Value Analysis - Risk Management.

Prescriptive Process Models

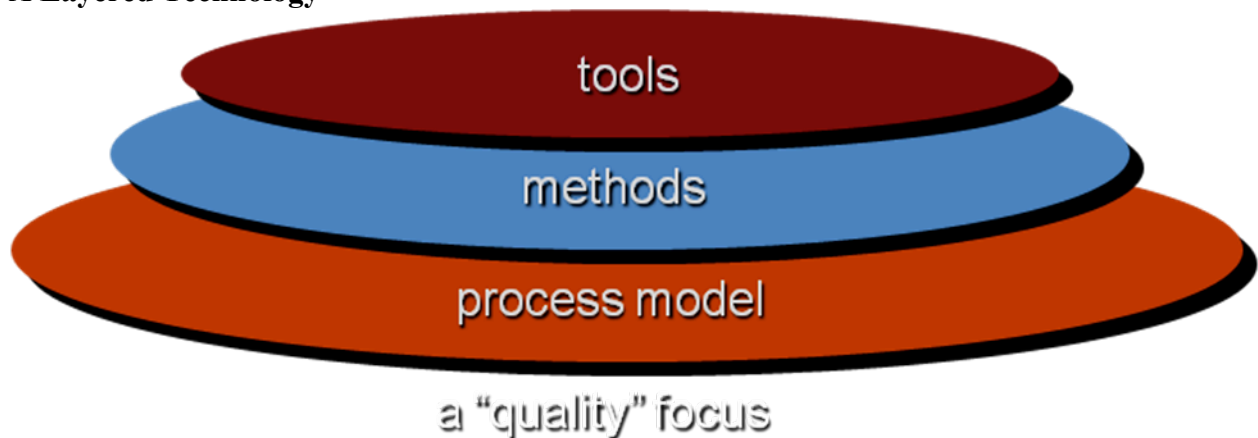
- ✓ Generic process framework
- ✓ Traditional process models
- ✓ Specialized process models
- ✓ The unified process

Process: A Generic View

Software Engineering

- ☐ The application of
 - ☐ systematic, disciplined, quantifiable approach
 - ☐ to the development, operation, and maintenance of software;
 - ☐ that is, the application of engineering to software.
- ☐ The study of approaches as pointed above.

A Layered Technology



Process

- ✓ Defines as a collection of work activities, actions, and tasks that are performed when some work product is to be created.
- ✓ Defines who is doing what, when, and how to reach a certain goal.
- ✓ Forms the basis for management control of software projects.

Methods

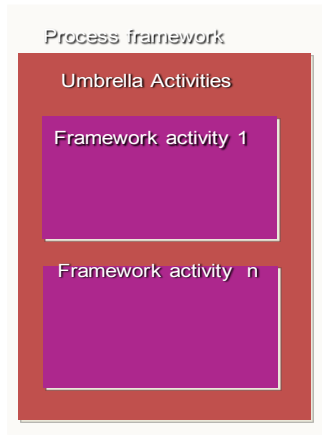
- ✓ technical “how to” s for building software
- ✓ Broad array of tasks: communication, requirements analysis, design modeling, program construction, testing, and support

Tools

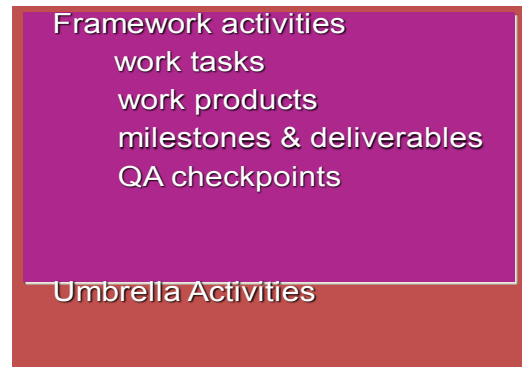
- ✓ Automated support for process and method

A Process Framework

Software Process



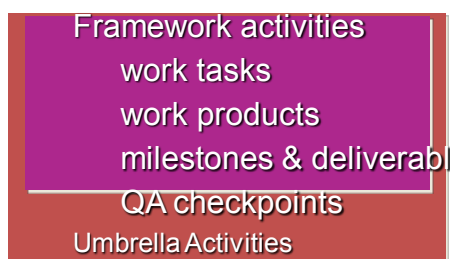
Process framework



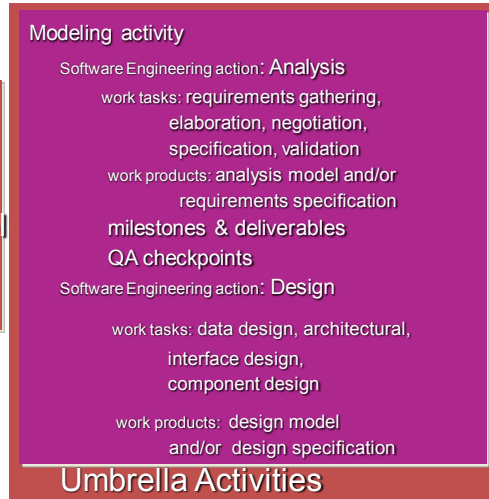
7

A Process Framework

Process framework



Process framework



8

Activity: Communication

- ✓ Task set for a relatively simple project
 - Make a list of stakeholders for project
 - Invite all stakeholders for an informal meeting
 - Ask each stakeholder to make a list of required features and functions
 - Discuss requirements and build a final list
 - Prioritize requirements
 - Note areas of uncertainty
- ✓ Task set for a larger project

Activity: Communication , Action: requirement gathering

- ✓ Task set for a relatively simple project ...
- ✓ Task set for a larger project
- ✓ Make a list of stakeholders for project
- ✓ Interview each stakeholder separately
- ✓ Build a preliminary list of required features and functions
- ✓ Schedule a series of facilitated requirement gathering meetings
- ✓ Conduct meetings
- ✓ Produce informal user scenarios as part of each meeting
- ✓ Refine user scenarios based on stakeholder feedback
- ✓ Build a revised list of stakeholder requirements
- ✓ Use quality function deployment techniques to prioritize requirements
- ✓ Package requirements so that they can be delivered incrementally
- ✓ Note constraints and restrictions that will be placed on the system
- ✓ Discuss methods for validating the system

Typical Umbrella Activities

Software project tracking and control

- To maintain schedule

Risk management

Software quality assurance

Formal technical reviews

- Uncover and remove errors before they propagate to the next action

Measurement

- Process, project and product measures

Software configuration management

- Manages the effects of change

Reusability management

Work product preparation and production

- Create models, documents, logs, forms, lists

The Process Model: Adaptability

- ✓ the framework activities will always be applied on every project ... BUT
- ✓ the tasks (and degree of rigor) for each activity will vary based on:
 - the type of project
 - characteristics of the project
 - common sense judgment; concurrence of the project team

The CMMI: Capability Maturity Model Integration

- ✓ The CMMI defines each process area in terms of “specific goals” and the “specific practices” required to achieve these goals.
- ✓ Specific goals establish the characteristics that must exist if the activities implied by a process area are to be effective.
- ✓ Specific practices refine a goal into a set of process-related activities.

Process Patterns

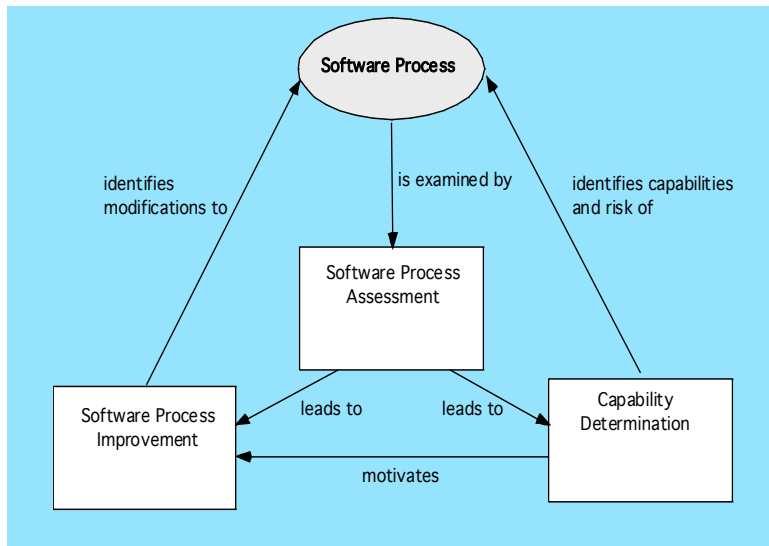
- ✓ Process patterns define a set of activities, actions, work tasks, work products and/or related behaviors
- ✓ A template is used to define a pattern
- ✓ Typical examples:

- ✓ Customer communication (a process activity)
- ✓ Analysis (an action)
- ✓ Requirements gathering (a process task)
- ✓ Reviewing a work product (a process task)
- ✓ Design model (a work product)

Process Assessment

The process should be assessed to ensure that it meets a set of basic process criteria that have been shown to be essential for a successful software engineering.

Assessment and Improvement



18

Personal Software Process (PSP)

Recommends five framework activities:

- ✓ Planning
- ✓ High-level design
- ✓ High-level design review
- ✓ Development
- ✓ Postmortem

Stresses the need for each software engineer to identify errors early and as important, to understand the types of errors

Plannings

1. This activity isolates requirements and based on these, develops both size and resource estimates.
2. Defect estimates are made.
3. All metrics are recorded on worksheets or templates.
4. Development task are identified and a project schedule is created.

High-level design

- ✓ External specification for each component to be constructed are developed and a component design is created.
- ✓ Prototypes are built when uncertainty exists.
- ✓ All issues are recorded and tracked.

High-level design view

- ✓ Formal verification methods are applied to uncover errors in the design.
- ✓ Metrics are maintained for all important tasks and work results.

Team Software Process (TSP)

- ✓ Each project is “launched” using a “script” that defines the tasks to be accomplished
- ✓ Teams are self-directed
- ✓ Measurement is encouraged
- ✓ Measures are analyzed with the intent of improving the team process

The Primary Goal of Any Software Process: *High Quality*

Remember:

High quality = project timeliness

Why?

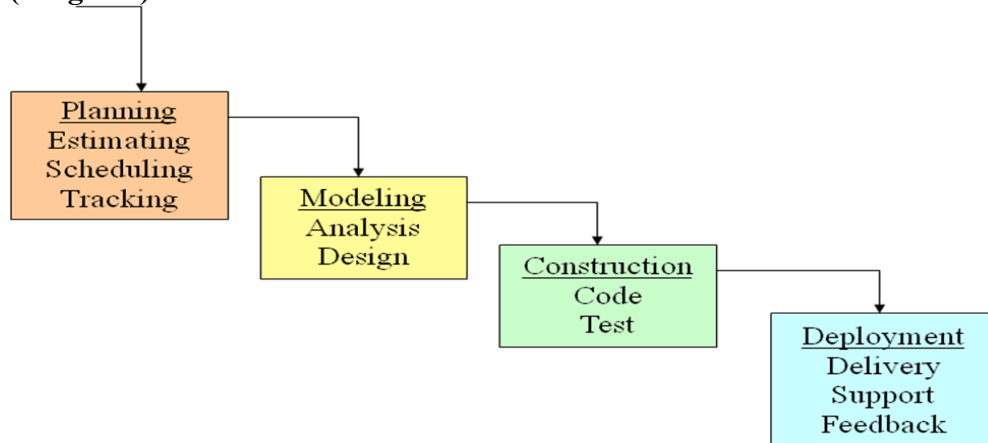
Less rework!

Traditional Process Models

Prescriptive Process Model

- ✓ Defines a distinct set of activities, actions, tasks, milestones, and work products that are required to engineer high-quality software
- ✓ The activities may be linear, incremental, or evolutionary

Waterfall Model (Diagram)



Waterfall Model

(Description)

- ✓ Oldest software lifecycle model and best understood by upper management
- ✓ Used when requirements are well understood and risk is low
- ✓ Work flow is in a linear (i.e., sequential) fashion
- ✓ Used often with well-defined adaptations or enhancements to current software

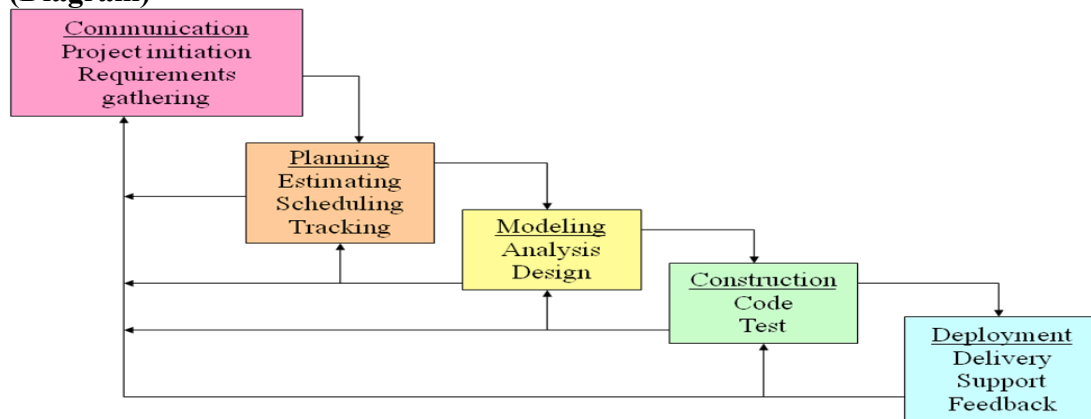
Waterfall Model

(Problems)

- ✓ Doesn't support iteration, so changes can cause confusion
- ✓ Difficult for customers to state all requirements explicitly and up front
- ✓ Requires customer patience because a working version of the program doesn't occur until the final phase
- ✓ Problems can be somewhat alleviated in the model through the addition of feedback loops

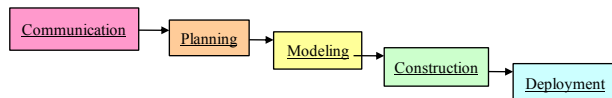
Waterfall Model with Feedback

(Diagram)

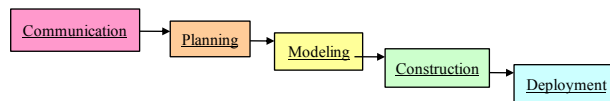


Incremental Model (Diagram)

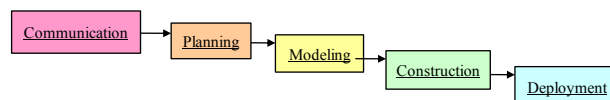
Increment #1



Increment #2



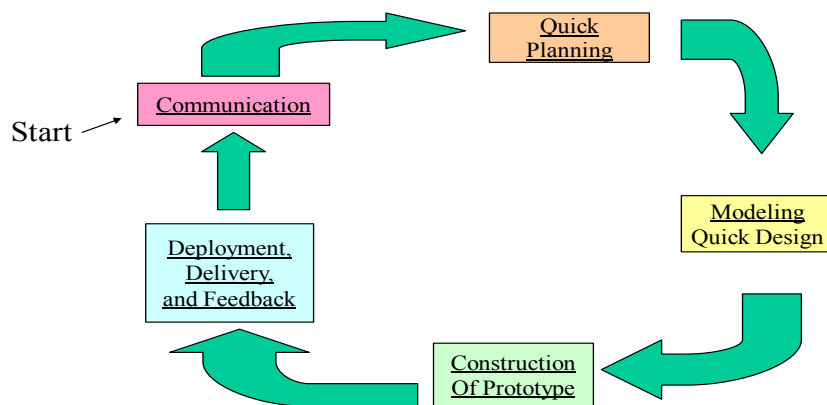
Increment #3



Incremental Model (Description)

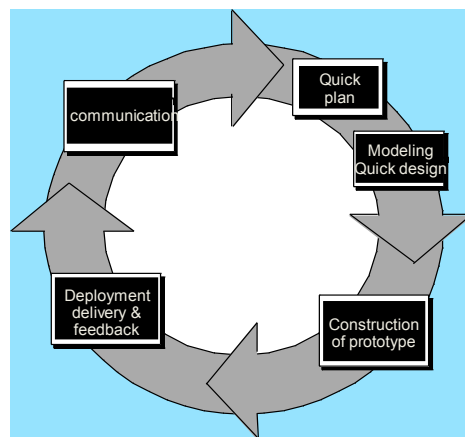
- ✓ Used when requirements are well understood
- ✓ Multiple independent deliveries are identified
- ✓ Work flow is in a linear (i.e., sequential) fashion within an increment and is staggered between increments
- ✓ Iterative in nature; focuses on an operational product with each increment
- ✓ Provides a needed set of functionality sooner while delivering optional components later
- ✓ Useful also when staffing is too short for a full-scale development

Prototyping Model (Diagram)



35

Evolutionary Models: Prototyping



These slides are designed to accompany
*Software Engineering: A Practitioner's
Approach*, 7/e (McGraw-Hill, 2009). Slides
copyright 2009 by Roger Pressman

36

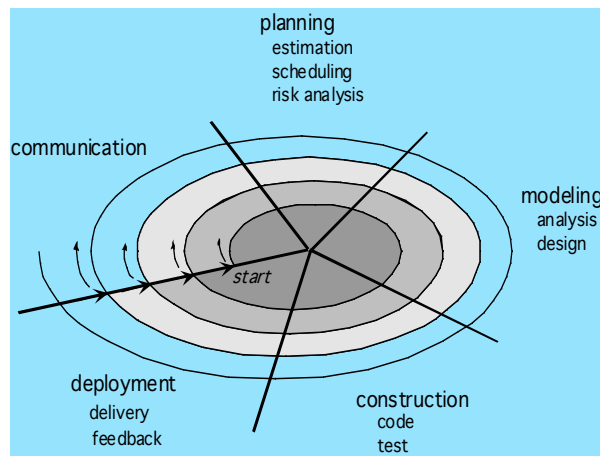
Prototyping Model (Description)

- ✓ Follows an evolutionary and iterative approach
- ✓ Used when requirements are not well understood
- ✓ Serves as a mechanism for identifying software requirements
- ✓ Focuses on those aspects of the software that are visible to the customer/user
- ✓ Feedback is used to refine the prototype

Prototyping Model (Potential Problems)

- ✓ The customer sees a "working version" of the software, wants to stop all development and then buy the prototype after a "few fixes" are made
- ✓ Developers often make implementation compromises to get the software running quickly (e.g., language choice, user interface, operating system choice, inefficient algorithms)
- ✓ Lesson learned
 - Define the rules up front on the final disposition of the prototype before it is built
 - In most circumstances, plan to discard the prototype and engineer the actual production software with a goal toward quality

Evolutionary Models: The Spiral



39

Spiral Model (Description)

- ✓ Invented by Dr. Barry Boehm in 1988 while working at TRW
- ✓ Follows an evolutionary approach
- ✓ Used when requirements are not well understood and risks are high
- ✓ Inner spirals focus on identifying software requirements and project risks; may also incorporate prototyping
- ✓ Outer spirals take on a classical waterfall approach after requirements have been defined, but permit iterative growth of the software

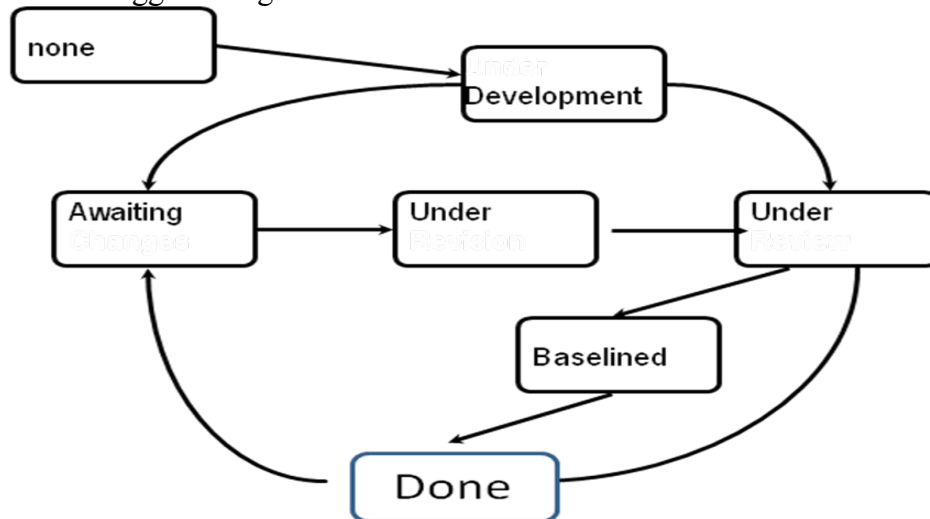
- ✓ Operates as a risk-driven model...a go/no-go decision occurs after each complete spiral in order to react to risk determinations
- ✓ Requires considerable expertise in risk assessment
- ✓ Serves as a realistic model for large-scale software development

General Weaknesses of Evolutionary Process Models

- ✓ Prototyping poses a problem to project planning because of the uncertain number of iterations required to construct the product
- ✓ Evolutionary software processes do not establish the maximum speed of the evolution
 - If too fast, the process will fall into chaos
 - If too slow, productivity could be affected
- ✓ Software processes should focus first on flexibility and extensibility, and second on high quality
 - We should prioritize the speed of the development over zero defects
 - Extending the development in order to reach higher quality could result in late delivery

The Concurrent Development Model

- ✓ Also called 'concurrent engineering'.
- ✓ Process represented by a network of activities.
- ✓ Each activity has a 'state'.
- ✓ Each activity exists simultaneously with other activities. Events from activities can trigger changes of state in other activities.



Specialized Process Models

Component-based Development Model

- ✓ Consists of the following process steps
 - Available component-based products are researched and evaluated for the application domain in question
 - Component integration issues are considered
 - A software architecture is designed to accommodate the components
 - Components are integrated into the architecture
 - Comprehensive testing is conducted to ensure proper functionality
- ✓ Relies on a robust component library

- ✓ Capitalizes on software reuse, which leads to documented savings in project cost and time

Formal Methods Model

(Description)

- ✓ Encompasses a set of activities that leads to formal mathematical specification of computer software
- ✓ Enables a software engineer to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation
- ✓ Ambiguity, incompleteness, and inconsistency can be discovered and corrected more easily through mathematical analysis
- ✓ Offers the promise of defect-free software
- ✓ Used often when building safety-critical systems

Formal Methods Model

(Challenges)

- ✓ Development of formal methods is currently quite time-consuming and expensive
- ✓ Because few software developers have the necessary background to apply formal methods, extensive training is required
- ✓ It is difficult to use the models as a communication mechanism for technically unsophisticated customers

The Unified Process

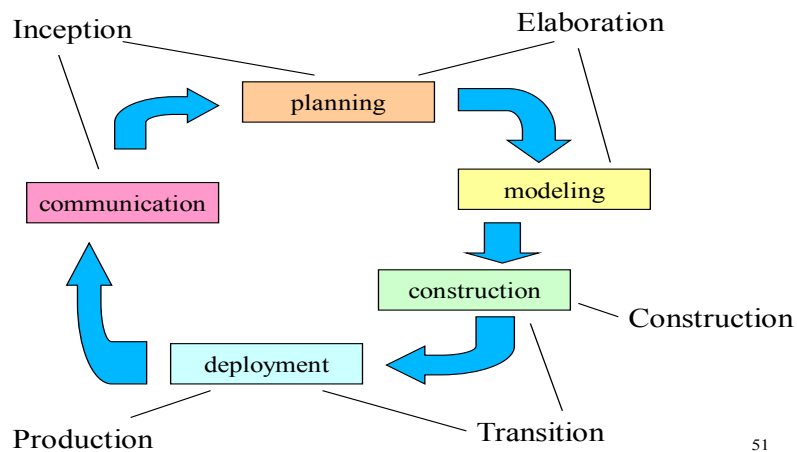
Background

- ✓ Birthed during the late 1980's and early 1990s when object-oriented languages were gaining wide-spread use
- ✓ Many object-oriented analysis and design methods were proposed; three top authors were Grady Booch, Ivar Jacobson, and James Rumbaugh
- ✓ They eventually worked together on a unified method, called the Unified Modeling Language (UML)
 - UML is a robust notation for the modeling and development of object-oriented systems
 - UML became an industry standard in 1997
 - However, UML does not provide the process framework, only the necessary technology for object-oriented development

Background (continued)

- ✓ Booch, Jacobson, and Rumbaugh later developed the unified process, which is a framework for object-oriented software engineering using UML
 - Draws on the best features and characteristics of conventional software process models
 - Emphasizes the important role of software architecture
 - Consists of a process flow that is iterative and incremental, thereby providing an evolutionary feel
- ✓ Consists of five phases: inception, elaboration, construction, transition, and production

Phases of the Unified Process



Inception Phase

- ✓ Encompasses both customer communication and planning activities of the generic process
- ✓ Business requirements for the software are identified
- ✓ A rough architecture for the system is proposed
- ✓ A plan is created for an incremental, iterative development
- ✓ Fundamental business requirements are described through preliminary use cases
 - A use case describes a sequence of actions that are performed by a user

Elaboration Phase

- ✓ Encompasses both the planning and modeling activities of the generic process
- ✓ Refines and expands the preliminary use cases
- ✓ Expands the architectural representation to include five views
 - Use-case model
 - Analysis model
 - Design model
 - Implementation model
 - Deployment model
- ✓ Often results in an executable architectural baseline that represents a first cut executable system
- ✓ The baseline demonstrates the viability of the architecture but does not provide all features and functions required to use the system

Construction Phase

- ✓ Encompasses the construction activity of the generic process
- ✓ Uses the architectural model from the elaboration phase as input
- ✓ Develops or acquires the software components that make each use-case operational
- ✓ Analysis and design models from the previous phase are completed to reflect the final version of the increment
- ✓ Use cases are used to derive a set of acceptance tests that are executed prior to the next phase

Transition Phase

- ✓ Encompasses the last part of the construction activity and the first part of the deployment activity of the generic process
- ✓ Software is given to end users for beta testing and user feedback reports on defects and necessary changes
- ✓ The software teams create necessary support documentation (user manuals, trouble-shooting guides, installation procedures)
- ✓ At the conclusion of this phase, the software increment becomes a usable software release

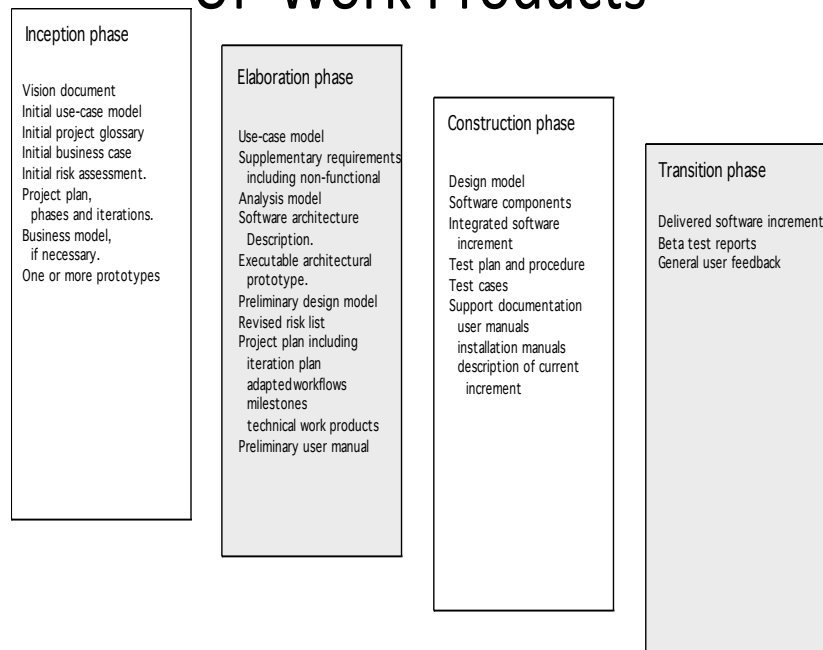
Production Phase

- ✓ Encompasses the last part of the deployment activity of the generic process
- ✓ On-going use of the software is monitored
- ✓ Support for the operating environment (infrastructure) is provided
- ✓ Defect reports and requests for changes are submitted and evaluated

Unified Process Work Products

- ✓ Work products are produced in each of the first four phases of the unified process
- ✓ In this course, we will concentrate on the analysis model and the design model work products
- ✓ Analysis model includes
 - Scenario-based model, class-based model, and behavioral model
- ✓ Design model includes
 - Component-level design, interface design, architectural design, and data/class design

UP Work Products



Estimation for Software Projects

■ Software Project Planning

The overall goal of project planning is to establish a pragmatic strategy for controlling, tracking, and monitoring a complex technical project.

Why?

So the end result gets done on time, with quality!

Project Planning Task Set-I

- Establish project scope
- Determine feasibility
- Analyze risks
 - Risk analysis is considered in detail in Chapter 25.
- Define required resources
 - Determine require human resources
 - Define reusable software resources
 - Identify environmental resources

Project Planning Task Set-II

- Estimate cost and effort
 - Decompose the problem
 - Develop two or more estimates using size, function points, process tasks or use-cases
 - Reconcile the estimates
- Develop a project schedule
 - Scheduling is considered in detail in.
- Establish a meaningful task set
- Define a task network
- Use scheduling tools to develop a timeline chart
- Define schedule tracking mechanisms

Estimation

- Estimation of resources, cost, and schedule for a software engineering effort requires
 - experience
 - access to good historical information (metrics)
 - the courage to commit to quantitative predictions when qualitative information is all that exists
- Estimation carries inherent risk and this risk leads to uncertainty

Write it Down!



5

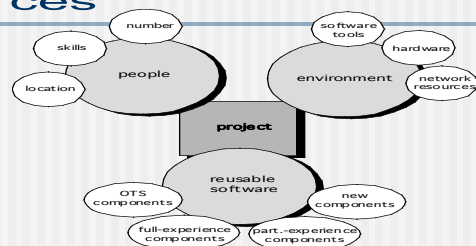
To Understand Scope ...

- Understand the customers needs
- understand the business context
- understand the project boundaries
- understand the customer's motivation
- understand the likely paths for change
- understand that ...

What is Scope?

- *Software scope* describes
 - the functions and features that are to be delivered to end-users
 - the data that are input and output
 - the “content” that is presented to users as a consequence of using the software
 - the performance, constraints, interfaces, and reliability that *bound* the system.
- Scope is defined using one of two techniques:
 - A narrative description of software scope is developed after communication with all stakeholders.
 - A set of use-cases is developed by end-users.

Resources



6

Project Estimation

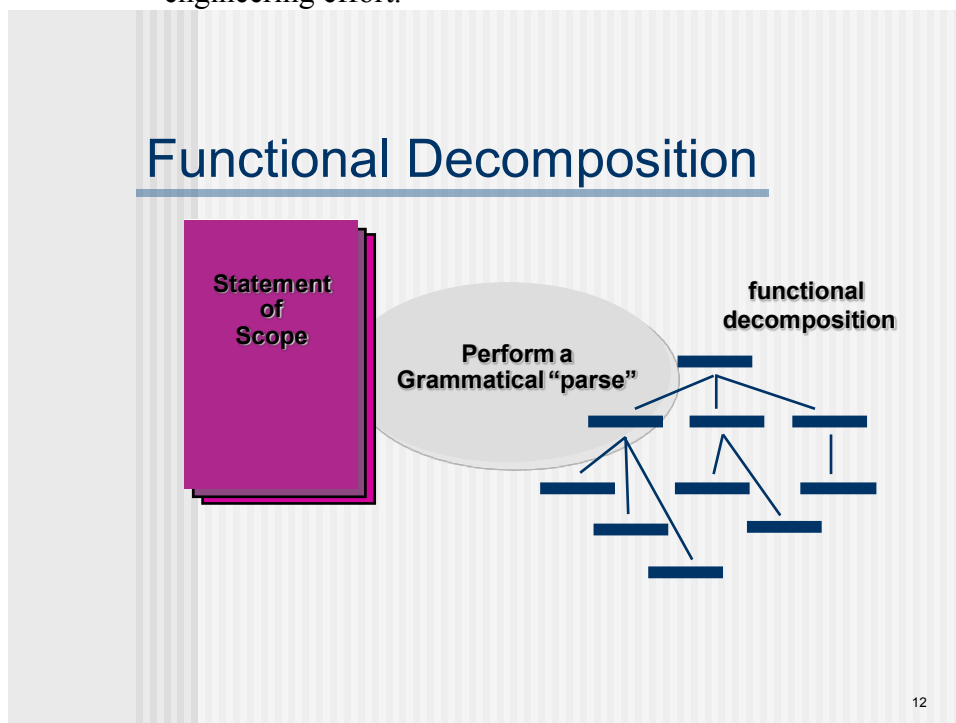
- Project scope must be understood
- Elaboration (decomposition) is necessary
- Historical metrics are very helpful
- At least two different techniques should be used
- Uncertainty is inherent in the process

Estimation Techniques

- Past (similar) project experience
- Conventional estimation techniques
 - task breakdown and effort estimates
 - size (e.g., FP) estimates
- Empirical models
- Automated tools

Estimation Accuracy

- Predicated on ...
 - the degree to which the planner has properly estimated the size of the product to be built
 - the ability to translate the size estimate into human effort, calendar time, and dollars (a function of the availability of reliable software metrics from past projects)
 - the degree to which the project plan reflects the abilities of the software team
 - the stability of product requirements and the environment that supports the software engineering effort.



Conventional Methods: LOC/FP Approach

- compute LOC/FP using estimates of information domain values
- use historical data to build estimates for the project

Example: LOC Approach

Function	Estimated LOC
user interface and control facilities (UICF)	2,300
two-dimensional geometric analysis (2D GA)	5,300
three-dimensional geometric analysis (3D GA)	6,800
database management (DBM)	3,300
computer graphics display facilities (GGDF)	4,900
peripheral control (PC)	2,100
design analysis modules (DAM)	8,400
<i>estimated lines of code</i>	33,200

Average productivity for systems of this type = 620 LOC/pm.

Burdened labor rate = \$8000 per month, the cost per line of code is approximately \$13.

Based on the LOC estimate and the historical productivity data, the total estimated project cost is **\$431,000** and the **estimated effort is 54 person-months**.

14

Example: FP Approach

Information Domain Value	opt.	likely	pess.	est. count	weight	FP count
number of inputs	20	24	30	24	4	97
number of outputs	12	15	22	16	5	78
number of inquiries	16	22	28	22	5	88
number of files	4	4	5	4	10	42
number of external interfaces	2	2	3	2	7	15
count-total						321

The estimated number of FP is derived:

$$FP_{\text{estimated}} = \text{count-total} \cdot 3 [0.65 + 0.01 \cdot 3 \cdot S(F_i)]$$

$$FP_{\text{estimated}} = 375$$

organizational average productivity = 6.5 FP/pm.

burdened labor rate = \$8000 per month, approximately \$1230/FP.

Based on the FP estimate and the historical productivity data, **total estimated project cost is \$461,000** and **estimated effort is 58 person-months**.

15

COCOMO-II

- COCOMO II is actually a hierarchy of estimation models that address the following areas:

- *Application composition model*. Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity are paramount.
- *Early design stage model*. Used once requirements have been stabilized and basic software architecture has been established.
- *Post-architecture-stage model*. Used during the construction of the software.

The Software Equation

The Software Equation

A dynamic multivariable model

$$E = [\text{LOC} \times B^{0.333}/P]^3 \times (1/t^4)$$

where

E = effort in person-months or person-years

t = project duration in months or years

B = "special skills factor"

P = "productivity parameter"

17

Estimation for OO Projects-I

- Develop estimates using effort decomposition, FP analysis, and any other method that is applicable for conventional applications.
- Using object-oriented requirements modeling (Chapter 6), develop use-cases and determine a count.
- From the analysis model, determine the number of key classes (called analysis classes in Chapter 6).
- Categorize the type of interface for the application and develop a multiplier for support classes:

Interface type	Multiplier
■ No GUI	2.0
■ Text-based user interface	2.25
■ GUI	2.5
■ Complex GUI	3.0

Estimation for OO Projects-II

- Multiply the number of key classes (step 3) by the multiplier to obtain an estimate for the number of support classes.

- Multiply the total number of classes (key + support) by the average number of work-units per class. Lorenz and Kidd suggest 15 to 20 person-days per class.
- Cross check the class-based estimate by multiplying the average number of work-units per use-case

Estimation for Agile Projects

- Each user scenario (a mini-use-case) is considered separately for estimation purposes.
- The scenario is decomposed into the set of software engineering tasks that will be required to develop it.
- Each task is estimated separately. Note: estimation can be based on historical data, an empirical model, or “experience.”
 - Alternatively, the ‘volume’ of the scenario can be estimated in LOC, FP or some other volume-oriented measure (e.g., use-case count).
- Estimates for each task are summed to create an estimate for the scenario.
 - Alternatively, the volume estimate for the scenario is translated into effort using historical data.
- The effort estimates for all scenarios that are to be implemented for a given software increment are summed to develop the effort estimate for the increment.

Project Scheduling

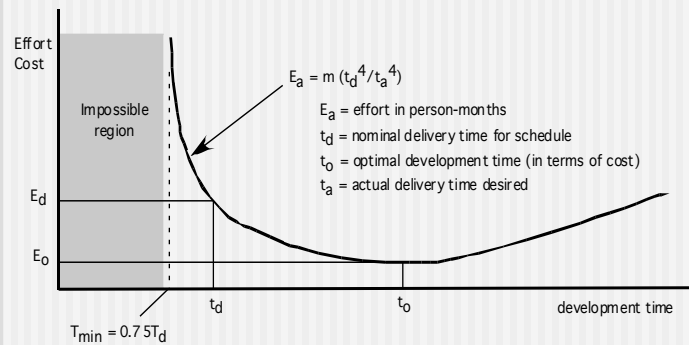
Why Are Projects Late?

- 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

Scheduling Principles

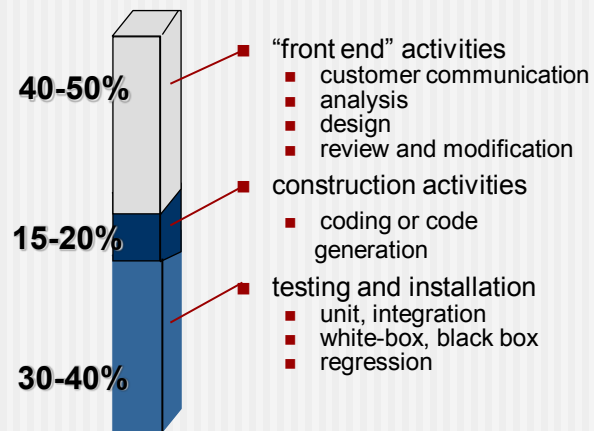
- compartmentalization—define distinct tasks
- interdependency—indicate task interrelationship
- effort validation—be sure resources are available
- defined responsibilities—people must be assigned
- defined outcomes—each task must have an output
- defined milestones—review for quality

Effort and Delivery Time



3

Effort Allocation



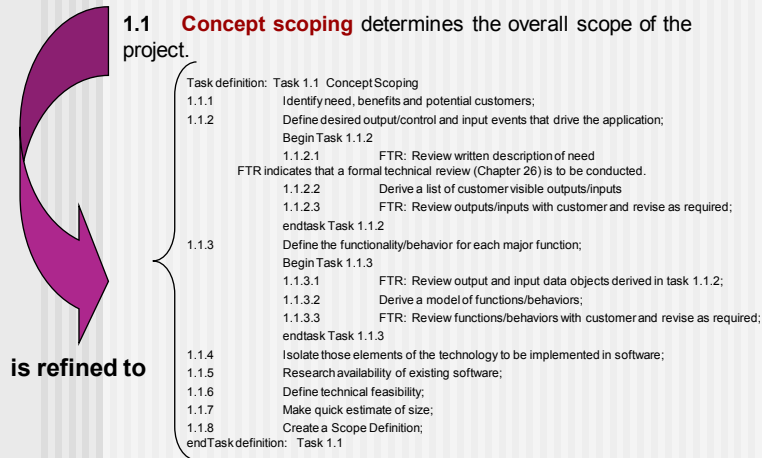
4

Defining Task Sets

- determine type of project
- assess the degree of rigor required
- identify adaptation criteria
- select appropriate software engineering tasks

Task Set Refinement

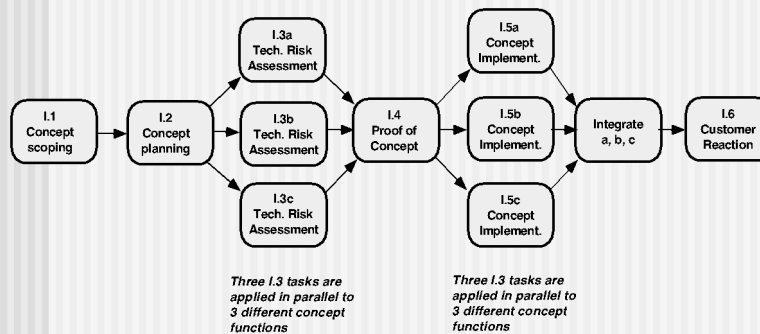
Task Set Refinement



6

Define a Task Network

Define a Task Network



7

Timeline Charts

Timeline Charts

Tasks	Week 1	Week 2	Week 3	Week 4		Week n
Task 1	■	■				
Task 2		■	■			
Task 3		■	■	■	■	
Task 4			■	■	■	
Task 5			■	■		
Task 6		■	■			
Task 7				■	■	
Task 8				■	■	■
Task 9			■	■	■	
Task 10				■	■	■
Task 11					■	■
Task 12		■	■	■		

8

Schedule Tracking

- conduct periodic project status meetings in which each team member reports progress and problems.
- evaluate the results of all reviews conducted throughout the software engineering process.
- determine whether formal project milestones (the diamonds shown in Figure 27.3) have been accomplished by the scheduled date.
- compare actual start-date to planned start-date for each project task listed in the resource table (Figure 27.4).
- meet informally with practitioners to obtain their subjective assessment of progress to date and problems on the horizon.
- use earned value analysis (Section 27.6) to assess progress quantitatively.

Progress on an OO Project-I

- *Technical milestone: OO analysis completed*
 - All classes and the class hierarchy have been defined and reviewed.
 - Class attributes and operations associated with a class have been defined and reviewed.
 - Class relationships (Chapter 8) have been established and reviewed.
 - A behavioral model (Chapter 8) has been created and reviewed.
 - Reusable classes have been noted.
- *Technical milestone: OO design completed*
 - The set of subsystems (Chapter 9) has been defined and reviewed.
 - Classes are allocated to subsystems and reviewed.
 - Task allocation has been established and reviewed.
 - Responsibilities and collaborations (Chapter 9) have been identified.

- Attributes and operations have been designed and reviewed.
- The communication model has been created and reviewed.

Progress on an OO Project-II

■ *Technical milestone: OO programming completed*

- Each new class has been implemented in code from the design model.
- Extracted classes (from a reuse library) have been implemented.
- Prototype or increment has been built.

■ *Technical milestone: OO testing*

- The correctness and completeness of OO analysis and design models has been reviewed.
- A class-responsibility-collaboration network (Chapter 6) has been developed and reviewed.
- Test cases are designed and class-level tests (Chapter 19) have been conducted for each class.
- Test cases are designed and cluster testing (Chapter 19) is completed and the classes are integrated.
- System level tests have been completed.

Earned Value Analysis (EVA)

■ Earned value

- is a measure of progress
- enables us to assess the “percent of completeness” of a project using quantitative analysis rather than rely on a gut feeling
- “provides accurate and reliable readings of performance from as early as 15 percent into the project.” [Fle98]

Computing Earned Value-I

- The *budgeted cost of work scheduled* (BCWS) is determined for each work task represented in the schedule.
 - $BCWS_i$ is the effort planned for work task i .
 - To determine progress at a given point along the project schedule, the value of BCWS is the sum of the $BCWS_i$ values for all work tasks that should have been completed by that point in time on the project schedule.
- The BCWS values for all work tasks are summed to derive the *budget at completion*, BAC. Hence,

$$BAC = \sum (BCWS_k) \text{ for all tasks } k$$

Computing Earned Value-II

- Next, the value for *budgeted cost of work performed* (BCWP) is computed.
 - The value for BCWP is the sum of the BCWS values for all work tasks that have actually been completed by a point in time on the project schedule.
- “the distinction between the BCWS and the BCWP is that the former represents the budget of the activities that were planned to be completed and the latter represents the budget of the activities that actually were completed.” [Wil99]

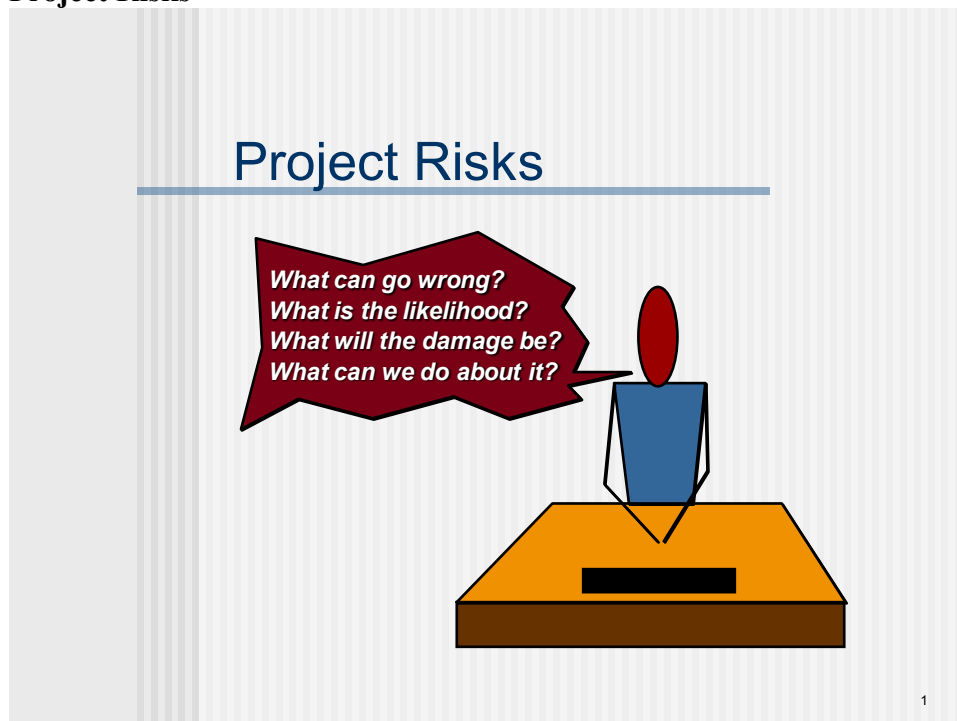
- Given values for BCWS, BAC, and BCWP, important progress indicators can be computed:
 - Schedule performance index, $SPI = BCWP/BCWS$
 - Schedule variance, $SV = BCWP - BCWS$
 - SPI is an indication of the efficiency with which the project is utilizing scheduled resources.

Computing Earned Value-III

- Percent scheduled for completion = $BCWS/BAC$
 - provides an indication of the percentage of work that should have been completed by time t .
- Percent complete = $BCWP/BAC$
 - provides a quantitative indication of the percent of completeness of the project at a given point in time, t .
- *Actual cost of work performed*, ACWP, is the sum of the effort actually expended on work tasks that have been completed by a point in time on the project schedule. It is then possible to compute
 - Cost performance index, $CPI = BCWP/ACWP$
 - Cost variance, $CV = BCWP - ACWP$

Risk Management.

Project Risks



Reactive Risk Management

- project team reacts to risks when they occur
- mitigation—plan for additional resources in anticipation of fire fighting
- fix on failure—resources are found and applied when the risk strikes

- crisis management—failure does not respond to applied resources and project is in jeopardy

Proactive Risk Management

- formal risk analysis is performed
- organization corrects the root causes of risk
 - TQM concepts and statistical SQA
 - examining risk sources that lie beyond the bounds of the software
 - developing the skill to manage change

Seven Principles

Maintain a global perspective—view software risks within the context of system and the business problem

Take a forward-looking view—think about the risks that may arise in the future; establish contingency plans

Encourage open communication—if someone states a potential risk, don't discount it.

Integrate—a consideration of risk must be integrated into the software process

Emphasize a continuous process—the team must be vigilant throughout the software process, modifying identified risks as more information is known and adding new ones as better insight is achieved.

Develop a shared product vision—if all stakeholders share the same vision of the software, it likely that better risk identification and assessment will occur.

Encourage teamwork—the talents, skills and knowledge of all stakeholder should be pooled



Risk Identification

Product size—risks associated with the overall size of the software to be built or modified.

Business impact—risks associated with constraints imposed by management or the marketplace.

Customer characteristics—risks associated with the sophistication of the customer and the developer's ability to communicate with the customer in a timely manner.

Process definition—risks associated with the degree to which the software process has been defined and is followed by the development organization.

Development environment—risks associated with the availability and quality of the tools to be used to build the product.

Technology to be built—risks associated with the complexity of the system to be built and the "newness" of the technology that is packaged by the system.

Staff size and experience—risks associated with the overall technical and project experience of the software engineers who will do the work.

Assessing Project Risk-I

- Have top software and customer managers formally committed to support the project?
- Are end-users enthusiastically committed to the project and the system/product to be built?
- Are requirements fully understood by the software engineering team and their customers?
- Have customers been involved fully in the definition of requirements?
- Do end-users have realistic expectations?

Assessing Project Risk-II

- Is project scope stable?
- Does the software engineering team have the right mix of skills?
- Are project requirements stable?
- Does the project team have experience with the technology to be implemented?
- Is the number of people on the project team adequate to do the job?
- Do all customer/user constituencies agree on the importance of the project and on the requirements for the system/product to be built?

Risk Components

performance risk—the degree of uncertainty that the product will meet its requirements and be fit for its intended use.

cost risk—the degree of uncertainty that the project budget will be maintained.

support risk—the degree of uncertainty that the resultant software will be easy to correct, adapt, and enhance.

schedule risk—the degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time.

Risk Projection

Risk projection, also called *risk estimation*, attempts to rate each risk in two ways

the likelihood or probability that the risk is real

the consequences of the problems associated with the risk, should it occur.

There are four risk projection steps:

- establish a scale that reflects the perceived likelihood of a risk
- delineate the consequences of the risk
- estimate the impact of the risk on the project and the product,
- note the overall accuracy of the risk projection so that there will be no misunderstandings.

Building a Risk Table

Building a Risk Table

Risk	Probability	Impact	RMMM
			Risk Mitigation Monitoring & Management

11

Building the Risk Table

Estimate the probability of occurrence

Estimate the impact on the project on a scale of 1 to 5, where

1 = low impact on project success

5 = catastrophic impact on project success

sort the table by probability and impact

Risk Exposure (Impact)

The overall *risk exposure*, RE, is determined using the following relationship [Hal98]:

$$RE = P \times C$$

where

P is the probability of occurrence for a risk, and

C is the cost to the project should the risk occur.

Risk Exposure Example

Risk identification. Only 70 percent of the software components scheduled for reuse will, in fact, be integrated into the application. The remaining functionality will have to be custom developed.

Risk probability. 80% (likely).

Risk impact. 60 reusable software components were planned. If only 70 percent can be used, 18 components would have to be developed from scratch (in addition to other custom software that has been scheduled for development). Since the average component is 100 LOC and local data indicate that the software engineering cost for each LOC is \$14.00, the overall cost (impact) to develop the components would be $18 \times 100 \times 14 = \$25,200$.

Risk exposure. $RE = 0.80 \times 25,200 \sim \$20,200$.

Risk Mitigation, Monitoring, and Management

mitigation—how can we avoid the risk?

monitoring—what factors can we track that will enable us to determine if the risk is becoming more or less likely?

management—what contingency plans do we have if the risk becomes a reality?

Risk Due to Product Size

Risk Due to Product Size

Attributes that affect risk:

- estimated size of the product in LOC or FP?
- estimated size of product in number of programs, files, transactions?
- percentage deviation in size of product from average for previous products?
- size of database created or used by the product?
- number of users of the product?
- number of projected changes to the requirements for the product? before delivery? after delivery?
- amount of reused software?

16

Risk Due to Business Impact

Risk Due to Business Impact

Attributes that affect risk:

- affect of this product on company revenue?
- visibility of this product by senior management?
- reasonableness of delivery deadline?
- number of customers who will use this product
- interoperability constraints
- sophistication of end users?
- amount and quality of product documentation that must be produced and delivered to the customer?
- governmental constraints
- costs associated with late delivery?
- costs associated with a defective product?

17

Risks Due to the Customer

Risks Due to the Customer

Questions that must be answered:

- Have you worked with the customer in the past?
- Does the customer have a solid idea of requirements?
- Has the customer agreed to spend time with you?
- Is the customer willing to participate in reviews?
- Is the customer technically sophisticated?
- Is the customer willing to let your people do their job—that is, will the customer resist looking over your shoulder during technically detailed work?
- Does the customer understand the software engineering process?

18

Risks Due to Process Maturity

Risks Due to Process Maturity

Questions that must be answered:

- Have you established a common process framework?
- Is it followed by project teams?
- Do you have management support for software engineering
- Do you have a proactive approach to SQA?
- Do you conduct formal technical reviews?
- Are CASE tools used for analysis, design and testing?
- Are the tools integrated with one another?
- Have document formats been established?

19

Technology Risks

Technology Risks

Questions that must be answered:

- Is the technology new to your organization?
- Are new algorithms, I/O technology required?
- Is new or unproven hardware involved?
- Does the application interface with new software?
- Is a specialized user interface required?
- Is the application radically different?
- Are you using new software engineering methods?
- Are you using unconventional software development methods, such as formal methods, AI-based approaches, artificial neural networks?
- Are there significant performance constraints?
- Is there doubt the functionality requested is "do-able?"

20

Staff/People Risks

Staff/People Risks

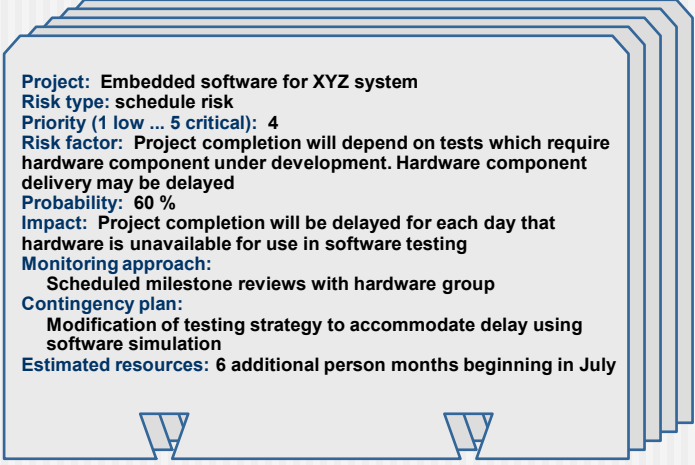
Questions that must be answered:

- Are the best people available?
- Does staff have the right skills?
- Are enough people available?
- Are staff committed for entire duration?
- Will some people work part time?
- Do staff have the right expectations?
- Have staff received necessary training?
- Will turnover among staff be low?

21

Recording Risk Information

Recording Risk Information



Project: Embedded software for XYZ system
Risk type: schedule risk
Priority (1 low ... 5 critical): 4
Risk factor: Project completion will depend on tests which require hardware component under development. Hardware component delivery may be delayed
Probability: 60 %
Impact: Project completion will be delayed for each day that hardware is unavailable for use in software testing
Monitoring approach:
Scheduled milestone reviews with hardware group
Contingency plan:
Modification of testing strategy to accommodate delay using software simulation
Estimated resources: 6 additional person months beginning in July