COMPUTER SCIENCE AND ENGINEERING

SOFTWARE DEVELOPMENT METHODOLOGY (AIT508)

VI SEMESTER

Prepared by:
Mr. J Thirupathi
Mr. C Praveen Kumar

UNIT –I

INTRODUCTION, A GENERIC VIEW OF PROCESS AND PROCESS MODELS

CONTENTS

Introduction to Software Engineering

Software products

- Generic products
  - Stand-alone systems that are marketed and sold to any customer who wishes to buy them. management tools; CAD software; software for specific markets such as appointments systems for dentists.
- Customized products
  - Software that is commissioned by a specific customer to meet their own needs.
  - Embedded control systems, air traffic control software, traffic monitoring systems.

Features of Software?

- Its characteristics that make it different from other things human being build.
- Features of such logical system:
- Software is developed or engineered, it is not manufactured in the classical sense which has quality problem.
- Software doesn't "wear out. but it deteriorates (due to change). Hardware has bathtub curve of failure rate (high failure rate in the beginning, then drop to steady state, then cumulative effects of dust, vibration, abuse occurs).
- Although the industry is moving toward component-based construction (e.g. standard screws and off-the-shelf integrated circuits), most software continues to be custom-built. Modern reusable components encapsulate data and processing into software parts to be reused by different programs.

  E.g. graphical user interface, window, pull-down menus in library etc.

Software Myths

–Software myths beliefs about software and the process used to build it.

Management Myths

- Myth – We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?
- Reality – The book of standards may very well exist but is it used? Are software practitioners aware of its existence? Does it reflect modern software engineering practice? Is it adaptable?
- Myth – If we get behind schedule, we can add more programmers and catch up
- Reality – Software development is not a mechanistic process like manufacturing.

Software Myths

–Software myths beliefs about software and the process used to build it

–Management Myths

- ➢ Myth – If I decide to outsource the software project to a third party, I can relax and let that firm build it
- ➢ Reality – If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it is outsourced.

Software Myths

–**Customer Myths**

- Myth – A general statement of objectives is sufficient to begin writing programs – we can fill in the details later
- Reality – An ambiguous statement of objectives is a recipe for disaster. Unambiguous requirements are developed only through effective and continuous communication between customer and developer
- Myth – Project requirements change, but change can be easily accommodated because software is flexible
- Reality – When requirement changes are requested early, cost impact is relatively small. With time, cost impact grows rapidly, and a change can cause additional resources and major design modifications.

Software Myths

–Practitioner Myths

- Myth – Once we write the program and get it to work.
- Reality – The sooner you begin writing code, the longer it will take you to get done. Between 60 to 80 percent of all effort spent on software will be spent after it is delivered to the customer for the first time
- Myth–Until I get the program running, I have no way of assessing its quality
- Reality – Software reviews are a duality filter that have found to be more effective than testing for finding certain classes of software errors

Software Myths

–Practitioner Myths

- Myth – The only deliverable work product for a successful project
- Reality – A working program is only one part of a software configuration that includes many elements. Documentation provides a foundation for successful engineering and guidance for software support
- Myth – Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down
- Reality – Software engineering is not about creating documents, it si about creating quality. Better quality leads to reduced rework. Reduced rework results in faster delivery times

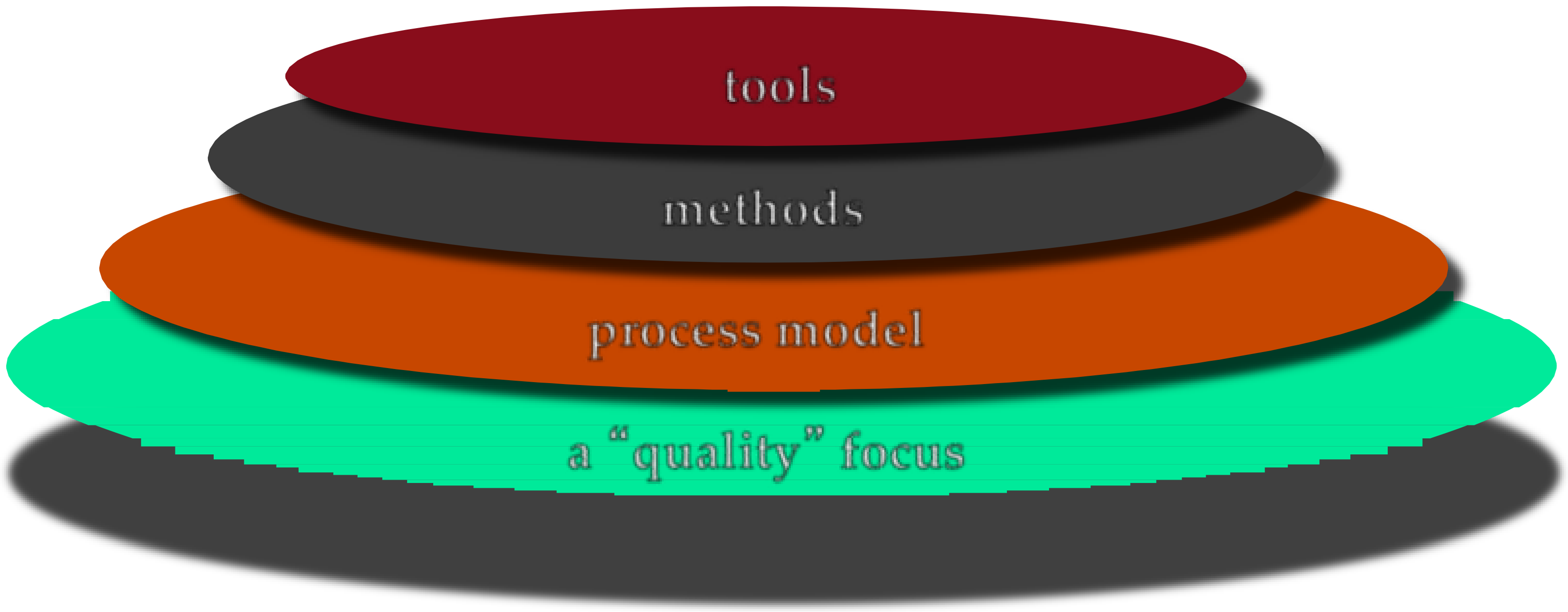
Software Applications

- **System software:** such as compilers, editors, file management utilities
- **Application software:** stand-alone programs for specific needs.
- **Engineering/scientific software:** Characterized by number crunching such as automotive stress analysis, molecular biology, orbital dynamics etc
- Embedded software resides within a product or system. (key pad control of a microwave oven, digital function of dashboard display in a car)
- Product-line software focus on a limited marketplace to address mass consumer market. (word processing, graphics, database management)
- WebApps (Web applications) network centric software. As web 2.0 emerges, more sophisticated computing environments is supported integrated with remote database and business applications.
- AI software uses non-numerical algorithm to solve complex problem. Robotics, expert system, pattern recognition game playing

Importance of SE

- Weneed to be able to produce reliable and trustworthy
- systems economically and quickly.
- It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems rather than just write the programs as if it was a personal programming project. For most types of system, the majority of costs are the costs of changing the software after it has gone into use.
- Any engineering approach must rest on organizational commitment to quality which fosters a continuous process improvement culture.

Software Layered Technology



Process layer as the foundation defines a framework with activities for effective delivery of software engineering technology. Establish the context where products (model, data, report, and forms) are produced, milestone are established, quality is ensured and change is managed.

Method provides technical how-to's for building software. It encompasses many tasks including communication, requirement analysis, design modeling, program construction, testing and support.

- **Tools** provide automated or semi-automated support for the process and methods.
- Communication: communicate with customer to understand objectives and gather requirements.
- Planning: creates a map defines the work by describing the tasks, risks and resources, work products and work schedule.
- Modeling: Create a sketch what it looks like architecturally, how the constituent parts fit together and other characteristics.
- Construction: code generation and the testing.
- Deployment: Delivered to the customer who evaluates the products and provides feedback based on the evaluation.
- These five framework activities can be used to all software development regardless of the application domain, size of the project, complexity of the efforts etc, though the details will be different in each case.
- For many software projects, these framework activities are applied **iteratively** as a project progresses. Each iteration produces a software increment that provides a subset of overall software features and functionality.

Adapting a Process Model

The process should be **agile and adaptable** to problems. Process adopted for one project might be significantly different than a process adopted from another project. (to the problem, the project, the team, organizational culture). Among the differences are:

- The overall flow of activities, actions, and tasks and the interdependencies among Them
- The degree to which actions and tasks are defined within each framework activity
- The degree to which work products are identified and required
- The manner which quality assurance activities are applied
- The manner in which project tracking and control activities are applied
- The overall degree of detail and rigor with which the process is described
- The degree to which the customer and other stakeholders are involved with the project
- The level of autonomy given to the software team
- The degree to which team organization and roles are prescribed

The process should be **agile and adaptable** to problems. Process adopted for one project might be significantly different than a process adopted from another project. (to the problem, the project, the team, organizational culture). Among the differences are:

- The overall flow of activities, actions, and tasks and the interdependencies among them
- The degree to which actions and tasks are defined within each framework activity
- The degree to which work products are identified and required
- The manner which quality assurance activities are applied
- The manner in which project tracking and control activities are applied
- The overall degree of detail and rigor with which the process is described
- The degree to which the customer and other stakeholders are involved with the project
- The level of autonomy given to the software team
- The degree to which team organization and roles are prescribed

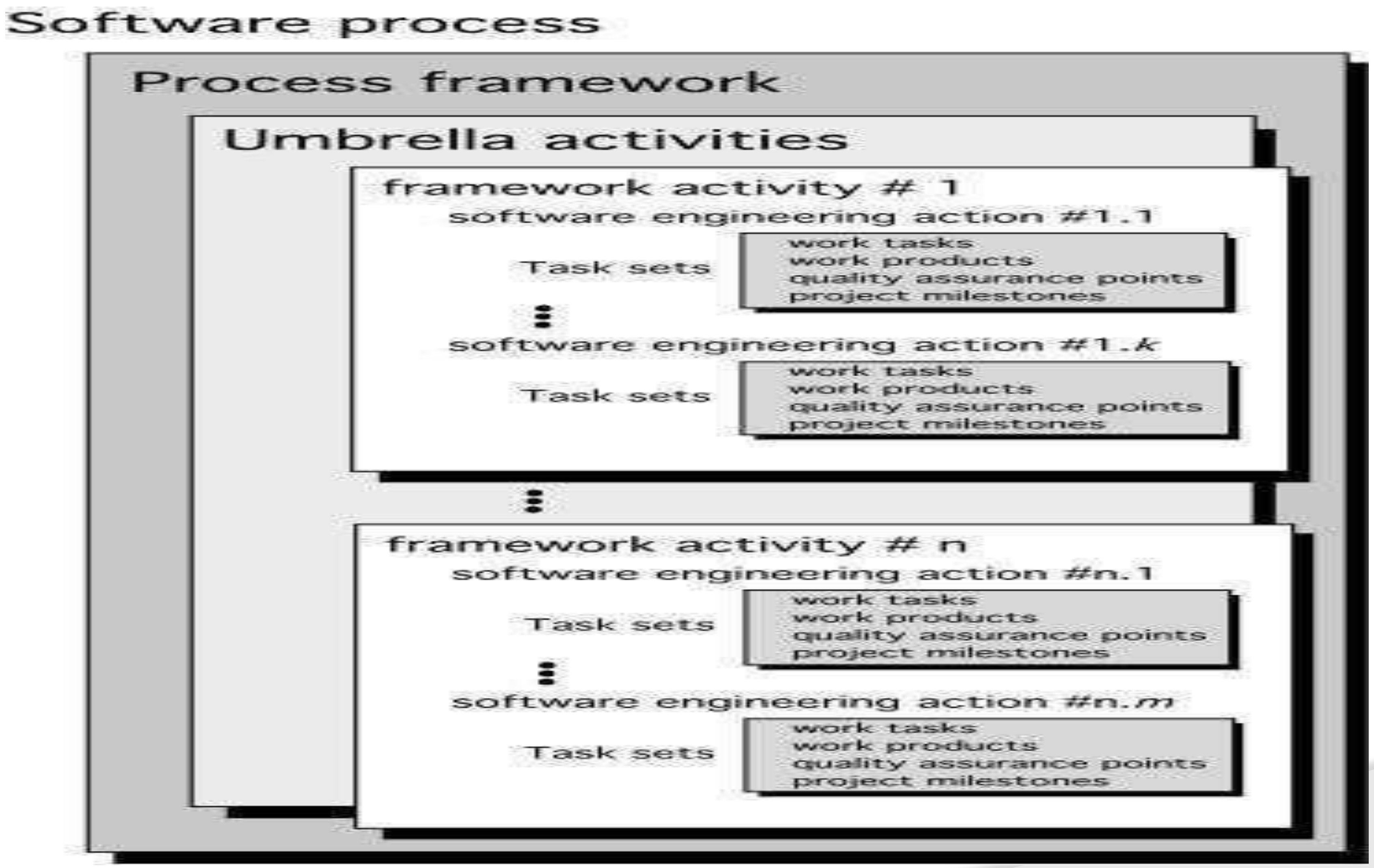
Plan the Solution

- Have you seen similar problems before? Are there patterns that are recognizable in a potential solution? Is there existing software that implements the data, functions, and features that are required?
- Has a similar problem been solved? If so, are elements of the solution reusable?
- Can subproblems be defined? If so, are solutions readily apparent for the subproblems?

Carry Out the Plan

- Does the solutions conform to the plan? Is source code traceable to the design model?
- Is each component part of the solution provably correct? Has the design and code been reviewed, or better, have correctness proofs been applied to algorithm?

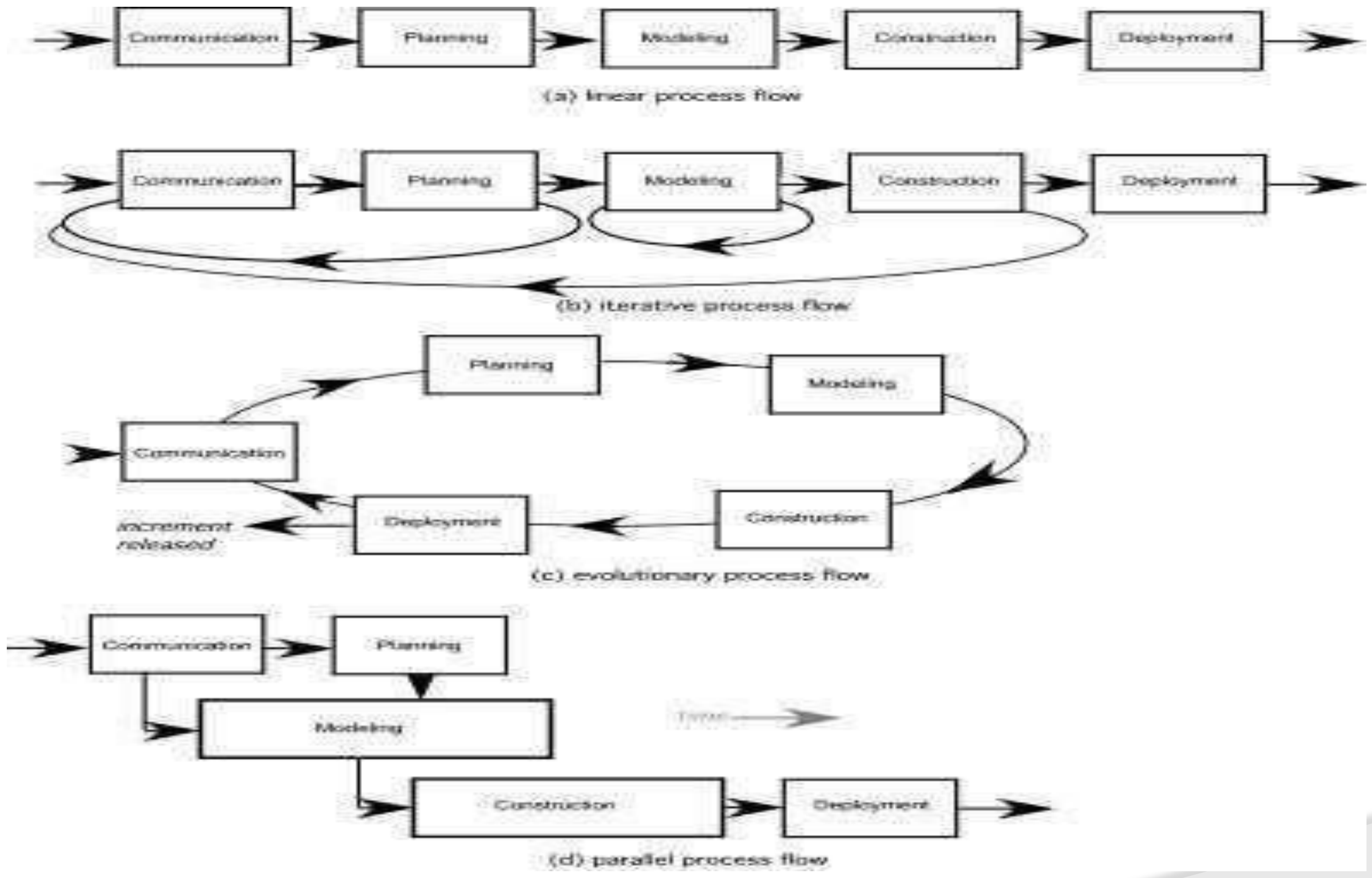
A Generic Process Model



A Generic Process Model

- A generic process framework for software engineering defines five framework activities communication, planning, modeling, construction, and deployment.
- In addition, a set of umbrella activities- project tracking and control, risk
- management, quality assurance, configuration management, technical reviews, and others are applied throughout the process.
- Next question is: how the framework activities and the actions and tasks that occur within each activity are organized with respect to sequence and time? See the **process flow** for answer.

Process Flow



(a) linear process flow
(b) iterative process flow
(c) evolutionary process flow
(d) parallel process flow

Cont.

- Linear process flow executes each of the five activities in sequence.
- An iterative process flow repeats one or more of the activities before proceeding to the next.
- An evolutionary process flow executes the activities in a circular manner.
- Each circuit leads to a more complete version of the software.
- A parallel process flow executes one or more activities in parallel with other activities modeling for one aspect of the software in parallel with construction of another aspect of the software.

Identifying a Task Set

- Before you can proceed with the process model, a key question: what **actions** are appropriate for a framework
- A task set defines the actual work to be done to accomplish the objectives of a software engineering action.
  - A list of the task to beaccomplished
  - A list of the work products to beproduced
  - A list of the quality assurance filters to be applied

For example, a small software project requested by one person with simple requirements, the communication activity might encompass little more than a phone all with the stakeholder. Therefore, the only necessary action is phone conversation, the work tasks of this action are:

- Make contact with stakeholder via telephone.
- Discuss requirements and take notes.
- Organize notes into a brief written statement of requirements.
- E-mail to stakeholder for review and approval.

Examples of a Selection process

The task sets for Requirements gathering action for a **simple** project may include:

1. Make a list of stakeholders for the project.
2. Invite all stake holders to an informal meeting.
3. Ask each stakeholder to make a list of features
4. Discuss requirements and build a finallist.
5. Prioritize requirements.
6. Note areas of uncertainty.

Ex. For Task Set Elicitation

- Make a list of stakeholders for the project.
- Interview each stakeholders separately to determine overall wants and needs.
- Build a preliminary list of functions and features based on stakeholder input.
- Schedule a series of facilitated application specification 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.

Both Do The Same Work With Different Depth And Formality. Choose The Task sets That Achieve The Goal And Still Maintain quality And Agility.

Process patterns

- A *process pattern*
  - describes a process-related problem that is encountered during software engineering work, identifies the environment in which the problem has been encountered, and suggests one or more proven solutions to the problem.
  - Stated in more general terms, a process pattern provides you with a *template* [Amb98]—a consistent method for describing problem solutions within the context of the software process. (defined at different levels of abstraction)
  - Problems and solutions associated with a complete process model (e.g. prototyping).
  - Problems and solutions associated with a framework activity (e.g. planning) or an action with a framework activity (e.g. project estimating).

Process patterns

- **STAGE PATTERNS**—defines a problem associated with a framework activity for Requirements Gathering and others.
- **TASK PATTERNS**—defines a problem associated with a software engineering action or work task and relevant to successful software engineering practice
- **Phase patterns**—define the sequence of framework activities that occur with the process, even when the overall flow of activities is iterative in nature. Example includes Spiral Model or Prototyping.

An Example of ProcessPattern

- Describes an approach that may be applicable when stakeholders have a general idea of what must be done but are unsure of specific softwarerequirements.
- Pattern name. Requirement Unclear
- Intent pattern describes an approach for building a model that can be assessed iteratively by stakeholders in an effort to identify or solidify software requirements.
- Initial context. Conditions must be met (1) stakeholders have been identified; (2) a mode of communication between stakeholders and the software team has been established; (3) the overriding software problem to be solved has been identified by stakeholders ; (4) an initial understanding of project scope, basic business requirements and project constraints has been developed.
- Problem. Requirements are hazy or nonexistent. stakeholders are unsure of whatthey want.
- Solution.Adescription of the prototyping process would be presentedhere.
- Resulting context. A software prototype that identifies basic requirements. (modes of interaction, computational features, processing functions) is approved by stakeholders. Following this, 1. This prototype may evolve through a series of increments to become the production software or 2. the prototype may be discarded.

Process Assessment and Improvement

- The existence of a software process is no guarantee that software will be delivered on time, that it will meet the customer’s needs, or that it will exhibit the technical characteristics that will lead to long-term quality characteristics.

  A number of approaches to software process improvement have been proposed over different r the past few decades:
- **Standard CMMI Assessment Method for Process Improvement (SCAMPI)—** provides a five-step process assessment model that incorporates five phases: initiating, diagnosing, establishing, acting, and learning. The SCAMPI method uses the SEI CMMI as the basis for assessment [SEI00].
- **CMM-Based Appraisal for Internal Process Improvement (CBA IPI)—** provides a diagnostic technique for assessing the relative maturity of a software organization; uses the SEI CMM as the basis for the assessment [Dun01].

- **SPICE (ISO/IEC15504)—**a standard that defines a set of requirements for software process assessment. The intent of the standard is to assist organizations in developing an objective evaluation of the efficacy of any defined software process[ISO08].

- **ISO 9001:2000 for Software—**a generic standard that applies to any organization that wants to improve the overall quality of the products, systems, or services that it provides. Therefore, the standard is directly applicable to software organizations and companies [Ant06].

Prescriptive Models

- Prescriptive process models were originally proposed to bring order tochaos.
- Prescriptive process models advocate an orderly approach to software engineering. However, will some extent of chaos (less rigid) be beneficial to bring some creativity?

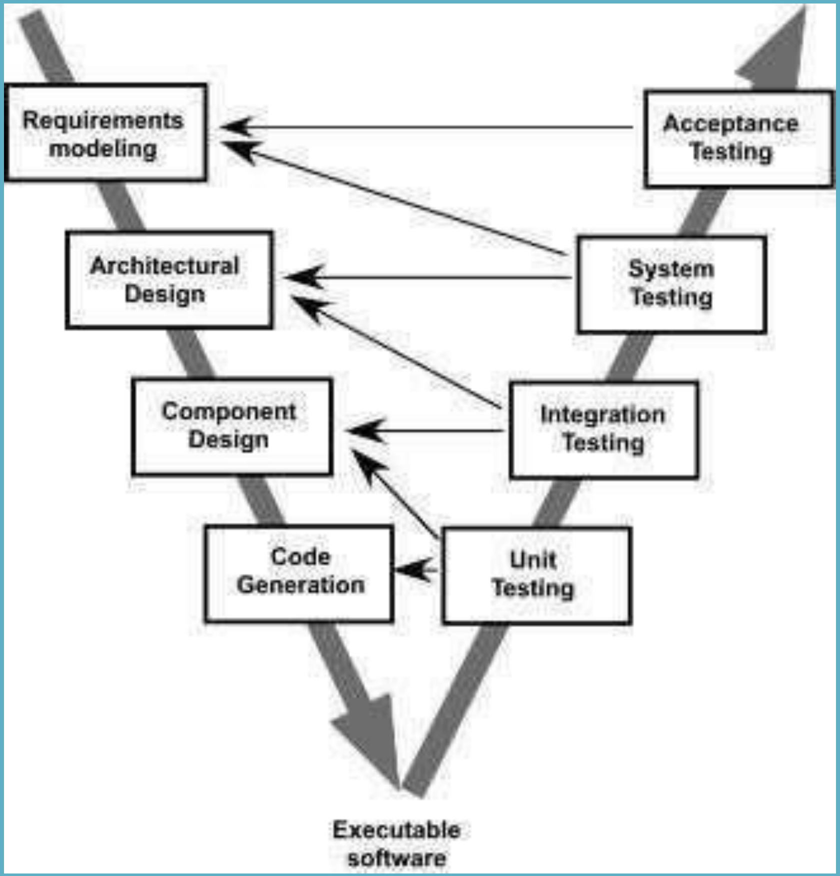
That leads to a few questions…

- If prescriptive process models strive for structure and order (prescribe a set of process elements and process flow), are they inappropriate for a software world that thrives on change?
- Yet, if we reject traditional process models (and the order they imply) and replace them with something less structured, do we make it impossible to achieve coordination and coherence in software work?

S/W Process Models

- Classic Process Models
  - Waterfall Model (Linear SequentialModel)
- Incremental Process Models
  - Incremental Model
- Evolutionary Software Process Models
  - Prototyping
  - Spiral Model
  - Concurrent Development Model

The V-Model



A variation of waterfall model depicts the relationship of quality assurance actions to the actions associated with communication, modeling and early code construction activates.

Team first moves down the left side of the V to refine the problem requirements. Once code is generated, the team moves up the right side of the V, performing a series of tests that validate each of the models created as the team moved down the left side.

The V-model provides a way of visualizing how verification and validation actions are applied to earlier engineering work.

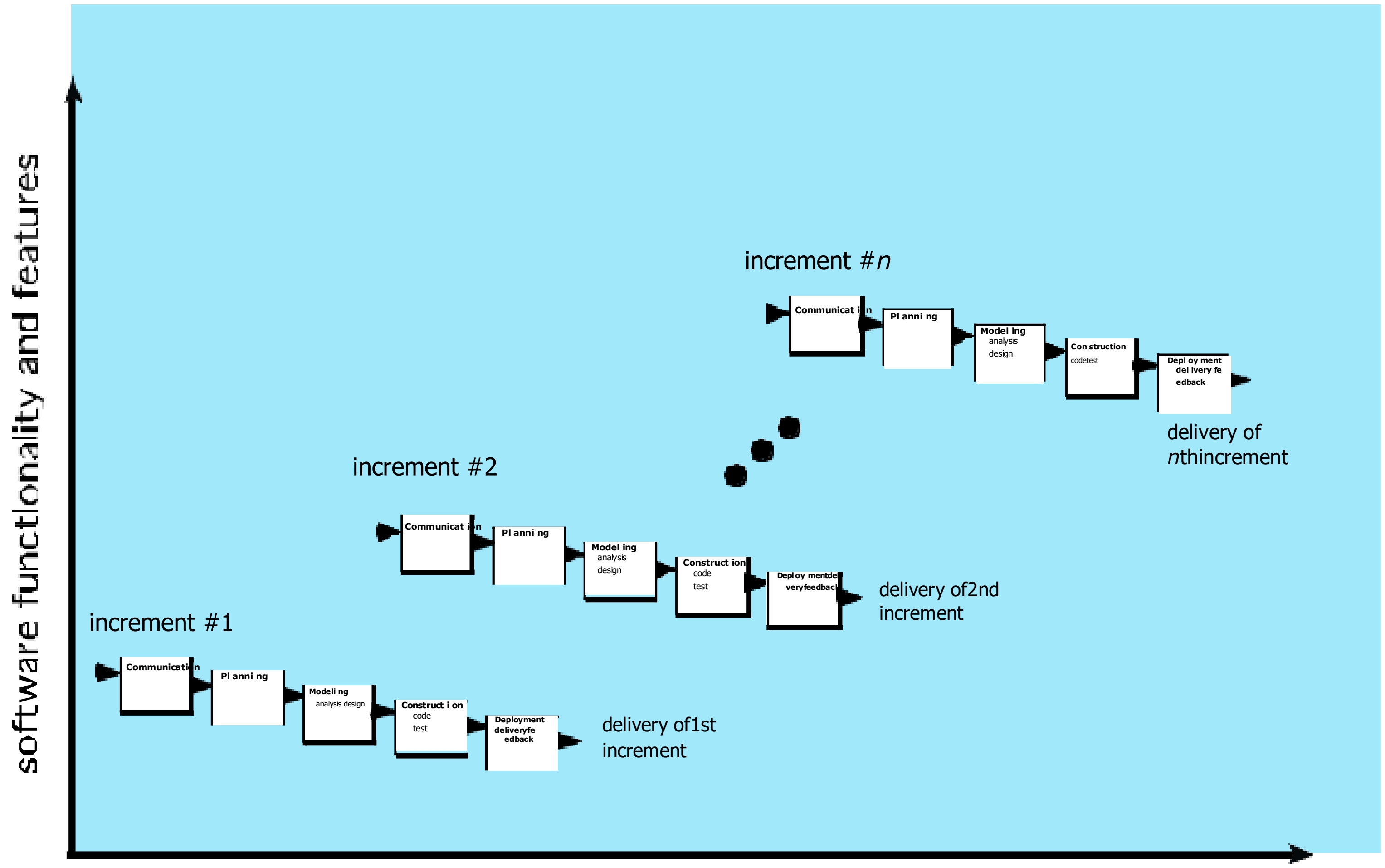
The problems that are sometimes encountered when the waterfall model is applied are:

- Real projects rarely follow the sequential flow that the model proposes. Although the linear model can accommodate iteration, it does so indirectly. As a result, changes can cause confusion as the project team proceeds.

- It is often difficult for the customer to state all requirements explicitly. The waterfall model requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects.

- The customer must have patience. A working version of the program(s) will not be available until late in the project time span. A major blunder, if undetected until the working program is reviewed, can be disastrous.

Incremental Model



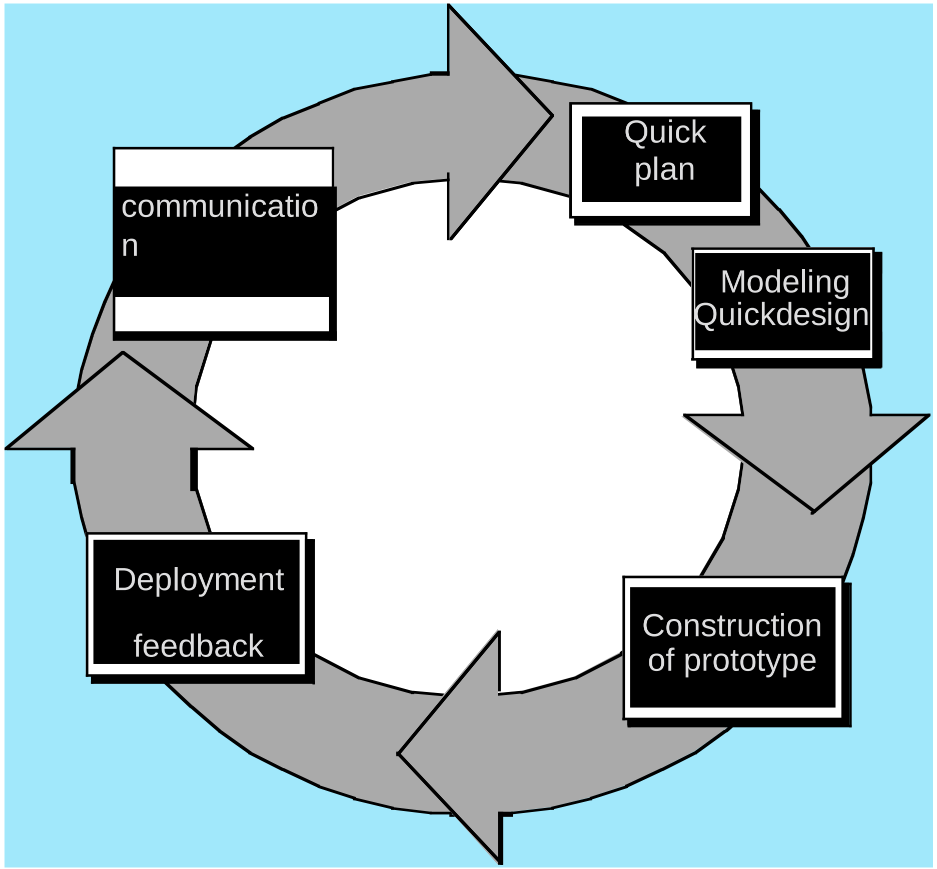
Project Calendartime

- When initial requirements are reasonably well defined, but the overall scope of the development effort precludes a purely linear process. A compelling need to expand a limited set of new functions to a later system release.
- It combines elements of linear and parallel process flows. Each linear sequence produces deliverable increments of the software.
- The first increment is often a core product with many supplementary features. Users use it and evaluate it with more modifications to better meet the needs.
- The incremental process model focuses on the delivery of an operational product with each increment. Early increments are stripped-down versions of the final product, but they do provide capability that serves the user and also provide a platform for evaluation by the user.
- Incremental development is particularly useful when staffing is unavailable for a complete implementation by the business deadline that has been established for the project

Evolutionary Models:Prototyping

- When to use: Customer defines a set of general objectives but does not identify detailed requirements for functions and features. or Developer may be unsure of the efficiency of an algorithm, the form that human computer interaction should take.
- What step: Begins with communication by meeting with stakeholders to define the objective, identify whatever requirements are known, outline areas where further definition is mandatory. A quick plan for prototyping and modeling (quick design) occur. Quick design focuses on a representation of those aspects the software that will be visible to end users. (interface and output). Design leads to the construction of a prototype which will be deployed and evaluated. Stakeholder's comments will be used to refine requirements.
- Both stakeholders and software engineers like the prototyping paradigm.
  Users get a feel for the actual system, and developers get to build something immediately. However, engineers may make compromises in order to get a prototype working quickly. The less-than-ideal choice may be adopted forever after you get used to it.

- When to use: Customer defines a set of general objectives but does not identify detailed requirements for functions and features. or Developer may be unsure of the efficiency of an algorithm, the form that human computer interaction should take.
- What step: Begins with communication by meeting with stakeholders to define the objective, identify whatever requirements are known, outline areas where further definition is mandatory. A quick plan for prototyping and modeling (quick design) occur. Quick design focuses on a representation of those aspects the software that will be visible to end users. (interface and output). Design leads to the construction of a prototype which will be deployed and evaluated. Stakeholder's comments will be used to refine requirements.
- Both stakeholders and software engineers like the prototyping paradigm.
- Users get a feel for the actual system, and developers get to build something immediately. However, engineers may make compromises in order to get a prototype working quickly. The less-than-ideal choice may be adopted forever after you get used to it.


communicatio
n
Quick plan
Modeling Quickdesign
Construction of prototype
Deployment feedback

Prototypingcanbeproblematicforthefolowing reasons:

- software, unaware that the prototype is held together haphazardly, unaware that in the rush to get it working you haveŶ't considered overall software quality or long-term maintainability.

- As a software engineer, you often make implementation compromises in order to get a prototype working quickly.

- An inappropriate operating system or programming language may be used simply because it is available and known;

- An inefficient algorithm may be implemented simply to demonstrate capability. After a time, you may become comfortable with these choices and forget all the reasons why they were inappropriate. The less-than-ideal choice has now become an integral part of the system

EvolutionaryModels:TheSpiral

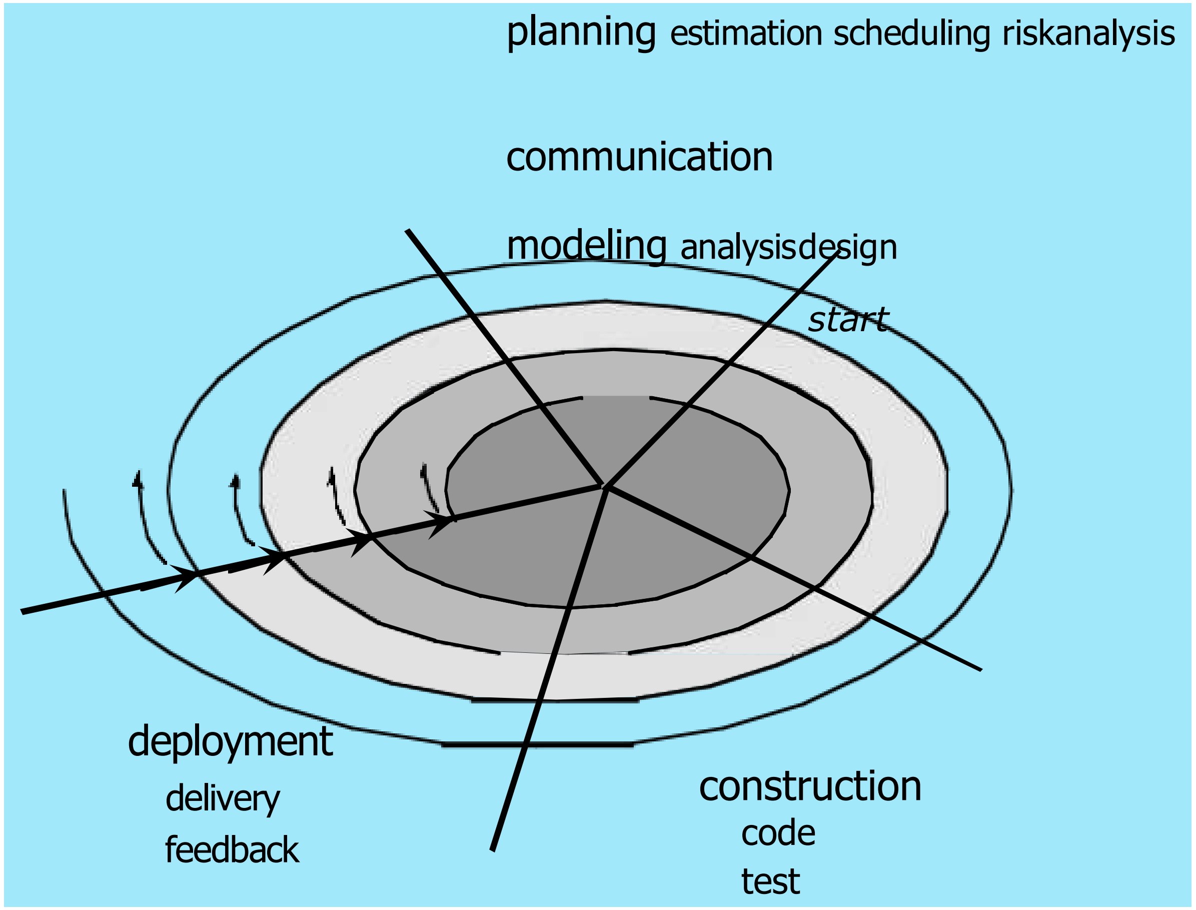
- It couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model and is a risk-driven process model generator that is used to guide multi-stakeholder concurrent engineering of software intensive systems.
- Two main distinguishing features: one is cyclic approach for incrementally growing a systeŵ's degree of definition and implementation while decreasing its degree of risk. The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions.
- A series of evolutionary releases are delivered. During the early iterations, the release might be a model or prototype. During later iterations, increasingly more complete version of the engineered system are produced.
- The first circuit in the clockwise direction might result in the product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software.

- Each pass results in adjustments to the project plan. Cost and schedule are adjusted based on feedback. Also, the number of iterations will be adjusted by project manager.

- Good to develop large-scale system as software evolves as the process progresses and risk should be understood and properly reacted to. Prototyping is used to reduce risk.

- However, it may be difficult to convince customers that it is controllable as it demands considerable risk assessment expertise.

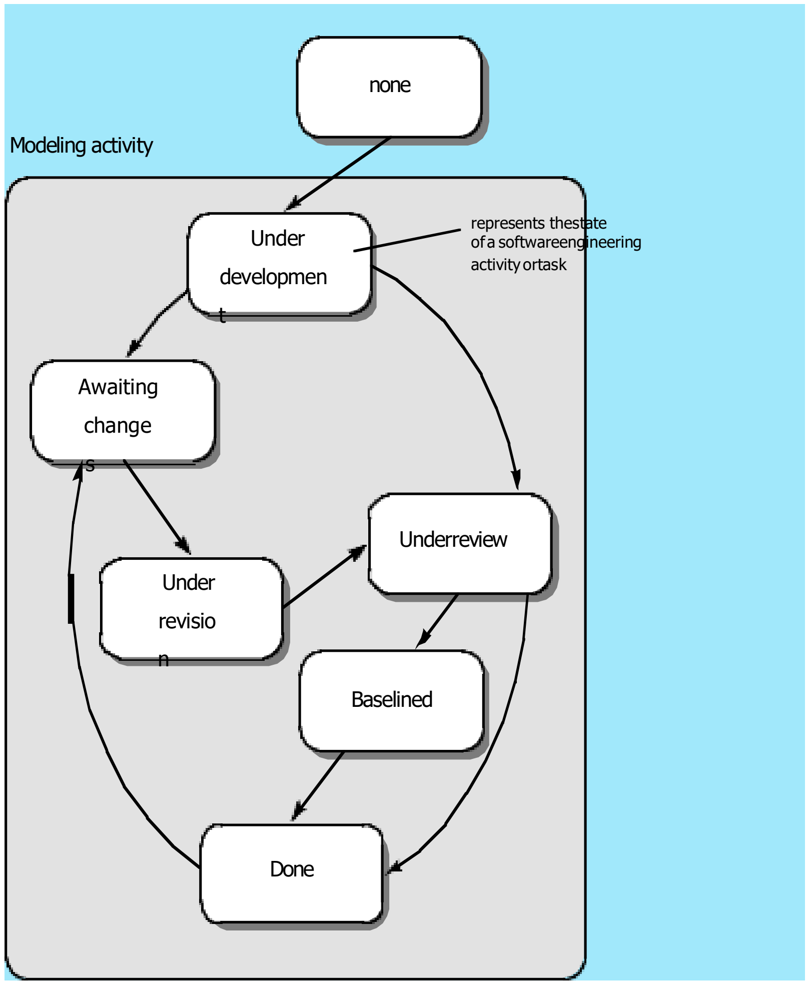
Spiral Mode


planning estimation scheduling riskanalysis
communication
modeling analysisdesign
start
deployment
delivery
feedback
construction
code
test

Concurrent Model

- Allow a software team to represent iterative and concurrent elements of any of the process models. For example, the modeling activity defined for the spiral model is accomplished by invoking one or more of the following actions: prototyping, analysis and design.
- The Figure shows modeling may be in any one of the states at any given time. For example, communication activity has completed its first iteration and in the awaiting changes state. The modeling activity was in inactive state, now makes a transition into the under development state. If customer indicates changes in requirements, the modeling activity moves from the under development state into the awaiting changes state.
- Concurrent modeling is applicable to all types of software development and provides an accurate picture of the current state of a project. Rather than confining software engineering activities, actions and tasks to a sequence of events, it defines a process network. Each activity, action or task on the network exists simultaneously with other activities, actions or tasks. Events generated at one point trigger transitions among the state.

Concurrent Model



Specialized Process Models

- Component-Based Development
- The Formal Methods Model
- Aspect-Oriented Software Development

Specialized process models

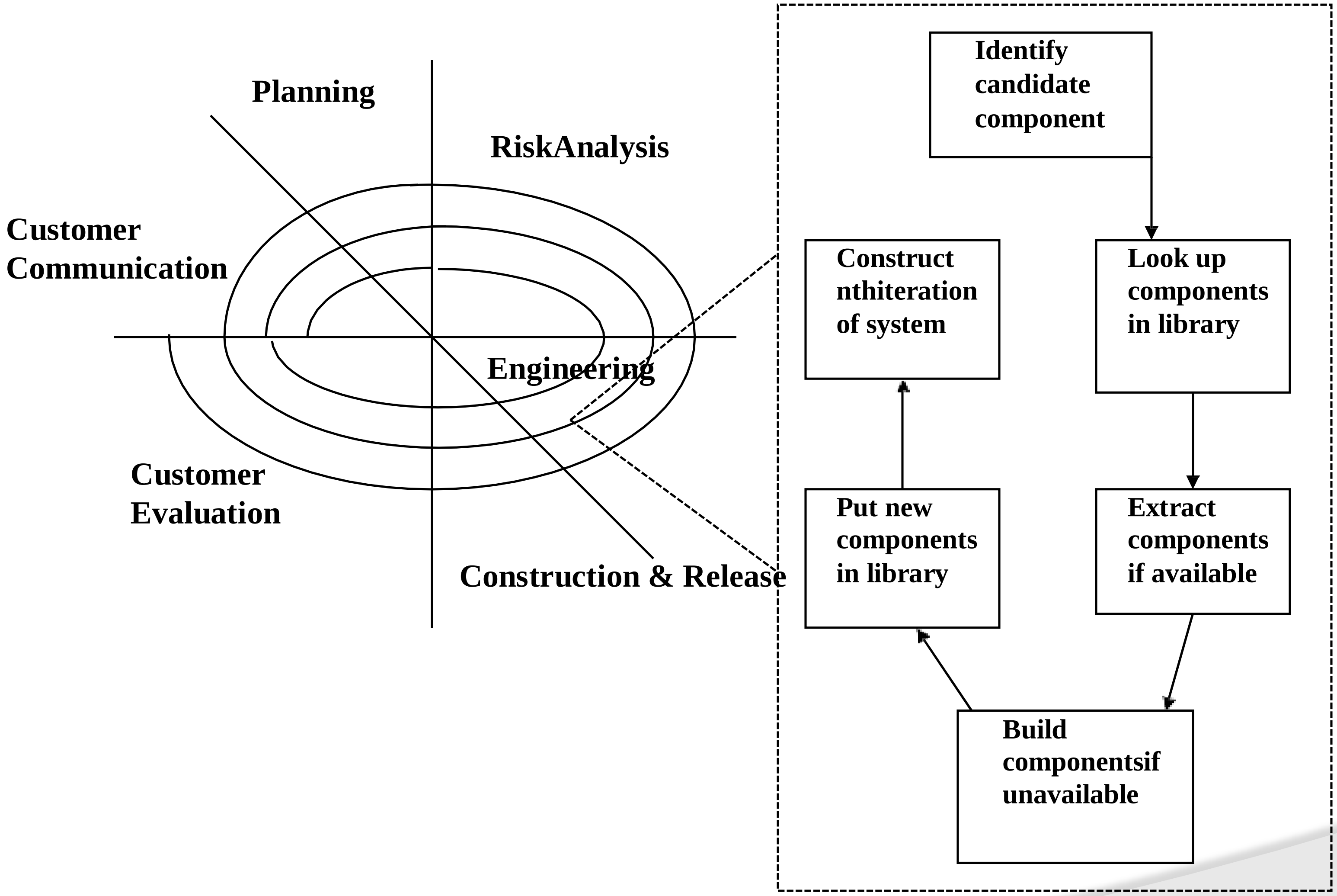
Component-Based Development:

Commercial off-the-shelf (COTS) software components, developed by vendors who offer them as products, provide targeted functionality with well-defined interfaces that enable the component to be integrated into the software that is to be built.

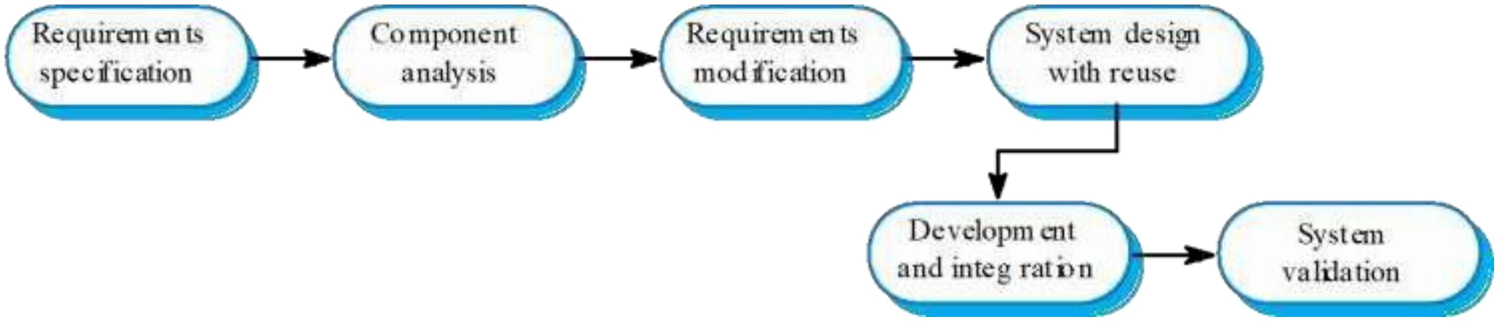
These components can be as either conventional software modules or object-oriented packages or packages of classes

- Steps involved in CBS are
- Available component-based researched products are and evaluated for the application domain in question.
- Components are integrated into the architecture
- Comprehensive testing is conducted ensure proper functionality.

Component Assembly Model


Planning
RiskAnalysis
Customer Communication
Engineering
Customer Evaluation
Construction & Release
Identify candidate component
Look up components in library
Extract components if available
Build componentsif unavailable
Put new components in library
Construct nthiteration of system

Re-use Oriented Development


Requirements specification
Component analysis
Requirements modification
System design with reuse
Development and integration
System validation

Specialized process model

Formal Methods Model

- Formal methods model encompasses a set of activities that leads to formal mathematical specification of computer software
- They enable software engineers to specify, develop and verify a computer based system by applying a rigorous mathematical notation
- Development of formal models is quite time consuming and expensive
- Extensive training is needed in applying formal methods
- Difficult to use the model as a communication mechanism for technically unsophisticated customers

Specialized Process model

Aspect-oriented Software Development

•The aspect-oriented approach is based on the principle of identifying common program code within certain aspects and placing the common procedures outside the main business logic

•The process of aspect orientation and software development may include modeling, design, programming, reverse engineering and re-engineering;

•The domain of AOSD includes applications, components and databases;

•Interaction with and integration into other paradigms is carried out with the help of frameworks, generators, program languages and architecture-description languages (ADL).

Unified Process

- The Unified Process is an iterative and incremental development process. Unified Process divides the project into four phases
  1. Inception 2. Elaboration 3. Construction 4. Transition
- The Inception, Elaboration, Construction and Transition phases are divided into a series of time boxed iterations. (The Inception phase may also be divided into iterations for a large project.)
- Each iteration results in an *increment*, which is a release of the system that contains added or improved functionality compared with the previous release.
- Although most iterations will include work in most of the process disciplines (*e.g.* Requirements, Design, Implementation, Testing) the relative effort and emphasis will change over the course of the project.
- **Risk Focused**

  The Unified Process requires the project team to focus on addressing the most critical risks early in the project life cycle. The deliverables of each iteration, especially in the Elaboration phase, must be selected in order to ensure that the greatest risks are addressed first. Risk Focused

Unified Process Model

- **Inception Phase**
  - Inception is the smallest phase in the project, and ideally it should be quite short. If the Inception Phase is long then it is usually an indication of excessive up-front specification, which is contrary to the spirit of the Unified Process.
  - The following are typical goals for the Inception phase.
    - Establish a justification or business case for theproject
    - Establish the project scope and boundaryconditions
    - Outline the use cases and key requirements that will drive the design tradeoffs
    - Outline one or more candidate architectures
    - Identify risks
    - Prepare a preliminary project schedule and costestimate
  - The Lifecycle Objective Milestone marks the end of the Inception phase.

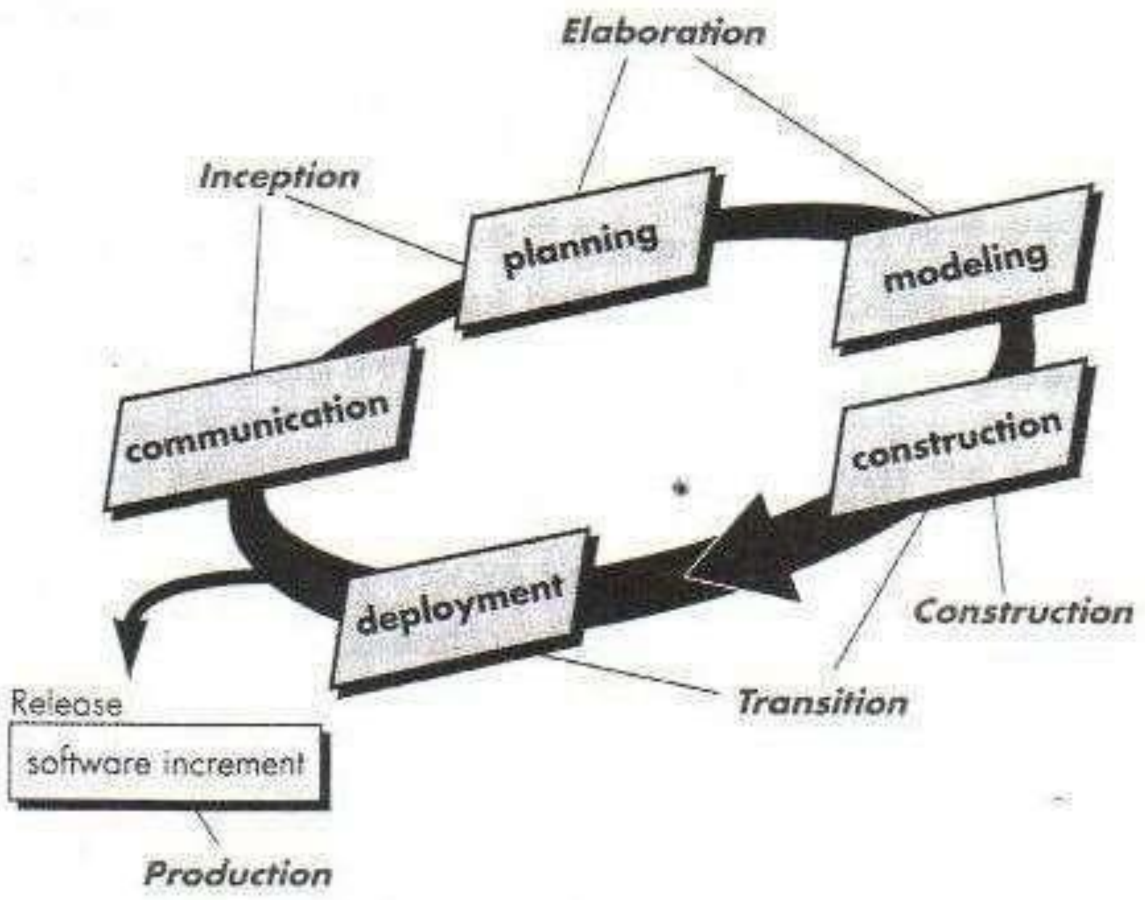
Unified Process

- **Elaboration Phase**
  - During the Elaboration phase the project team is expected to capture a majority of the system requirements. The primary goals of Elaboration are to address known risk factors and to establish and validate the system architecture.
  - Common processes undertaken in this phase include the creation of use case diagrams, conceptual diagrams (class diagrams with only basic notation) and package diagrams (architectural diagrams).
  - The architecture is validated primarily through the implementation of an Executable Architectural Baseline. This is a partial implementation of the system which includes the core, most architecturally significant, components. It is built in a series of small, timeboxed iterations.

- **Elaboration Phase**
  - By the end of the Elaboration phase the system architecture must have stabilized and the executable architecture baseline must demonstrate that the architecture will support the key system functionality and exhibit the right behavior in terms of performance, scalability and cost.
  - The final Elaboration phase deliverable is a plan (including cost and schedule estimates) for the Construction phase. At this point the plan should be accurate and credible, since it should be based on the Elaboration phase experience and since significant risk factors should have been addressed during the Elaboration phase.
  - The Lifecycle Architecture Milestone marks the end of the Elaboration phase.

- **Construction Phase**
  - Construction is the largest phase in the project. In this phase the remainder of the system is built on the foundation laid in Elaboration. System features are implemented in a series of short, timeboxed iterations. Each iteration results in an executable release of the software. It is customary to write full text use cases during the construction phase and each one becomes the start of a new iteration.
  - Common UML (Unified Modeling Language) diagrams used during this phase include Activity, Sequence, Collaboration, State (Transition) and Interaction Overview diagrams.
  - The Initial Operational Capability Milestone marks the end of the Construction phase.

- **Transition Phase**
  - The final project phase is Transition. In this phase the system is deployed to the target users. Feedback received from an initial release (or initial releases) may result in further refinements to be incorporated over the course of several Transition phase iterations. The Transition phase also includes system conversions and user training.
  - The Product Release Milestone marks the end of the Transition phase.


Elaboration
Inception
planning
modeling
communication
construction
deployment
Construction
Transition
Release
software increment
Production

- **Advantages of UP SoftwareDevelopment**
  - This is a complete methodology in itself with an emphasis on accurate documentation
  - It is proactively able to resolve the project risks associated with other projects.
  - Less time is required for integration as the process of integration goes on throughout .
  - The development time required is less due to reuse of components.

- **Disadvantages of RUP SoftwareDevelopment**
  - The team members need to be expert in their field to develop a software under this methodology.
  - On cutting edge projects which utilise new technology, the reuse of components will not be possible. Hence the time saving one could have made will be impossible to fulfill.
  - Integration throughout the process of software development, in theory sounds a good thing. But on particularly big projects with multiple development streams it will only add to the confusion and cause more issues during the stages of testing

Personal Team and Process Models

- The best software process is one that is close to the people who will be doing the work. The PSP model defines five framework activities.

1. Personal Software Process (PSP)

Planning: This activity isolates requirements and develops both size and resource estimates. In addition, a defect estimate is made. All metrics are recorded on worksheets or templates. Finally, development tasks are identified and a project schedule is created.

High-level design: External specifications 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 review: Formal verification methods (Chapter 21) are applied to uncover errors in the design. Metrics are maintained for all important tasks and work results.

Personal team and process models

- **Development.** The component-level design is refined and reviewed. Code is generated, reviewed, compiled, and tested. Metrics are maintained for all important tasks and work results.
- **Postmortem.** Using the measures and metrics collected, the effectiveness of the process is determined. Measures and metrics should provide guidance for modifying the process to improve its effectiveness.

2. **Team Software Process (TSP):** The goal of TSP is to build a self directed project team that organizes itself to produce high-qualitysoftware. TSP objectivesare,
   - Build self-directed teams that plan and track their work, establish goals, and own their processes and plans. These can be pure software teams or integrated product teams (IPTs) of 3 to about 20 engineers.
   - Show managers how to coach and motivate their teams and how to help them sustain peak performance.
   - Accelerate software process improvement by making CMM23 Level 5 behavior normal and expected.
   - Provide improvement guidance to high-maturityorganizations.
   - Facilitate university teaching of industrial-grade team skills.

What is Estimation ?

Estimation is attempt to determine how much money, effort, resources & time it will take to build a specific software based system or project.

Estimation involves answering the following questions:

1. How much effort is required to complete each activity?
2. How much calendar time is needed to complete each activity?
3. What is the total cost of each activity?

Project cost estimation and project scheduling are normally carried out together. The costs of development are primarily the costs of the effort involved, so the effort computation is used in both the cost and the schedule estimate.

Do some cost estimation before detailed schedules are drawn up. These initial estimates may be used to establish a budget for the project or to set a price for the software for a customer.

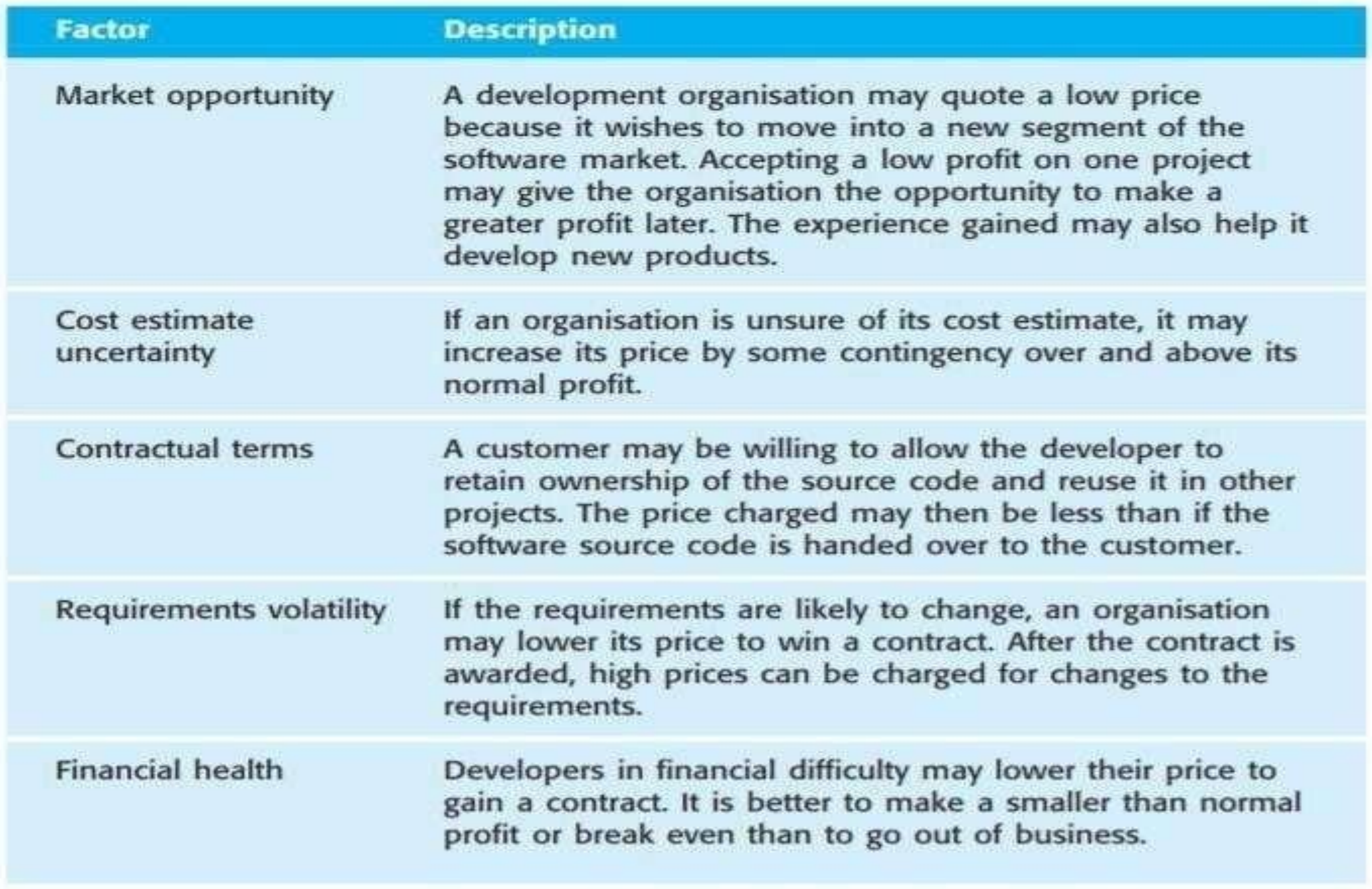
There are three parameters involved in computing the total cost of asoftware development project:

- Hardware and software costs including maintenance
- Travel and training costs
- Effort costs (the costs of paying software engineers).

The following costs are all part of the total effort cost:

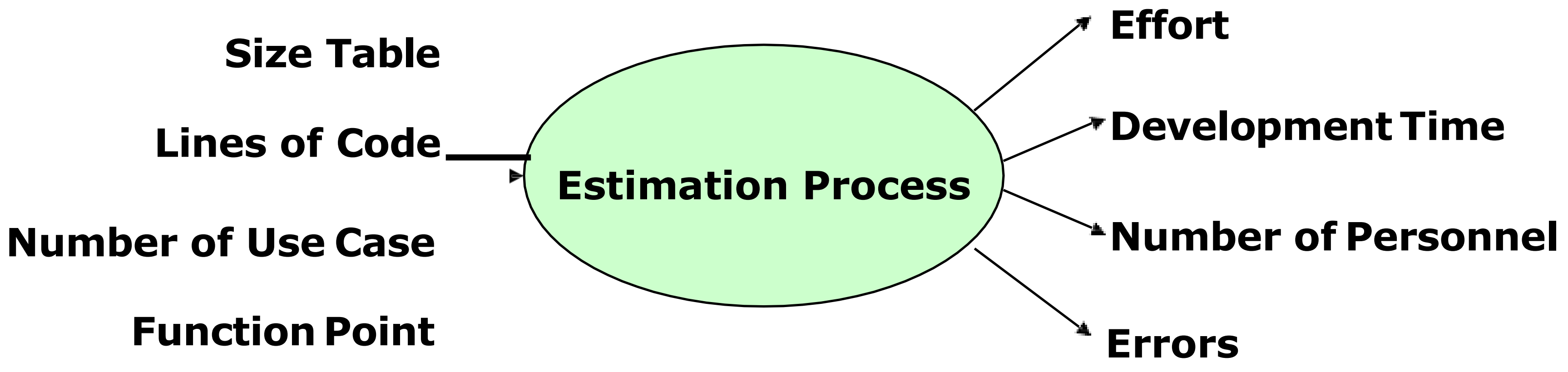
1. Costs of providing, heating and lighting office space
2. Costs of support staff such as accountants, administrators, system managers, cleaners and technicians
3. Costs of networking and communications
4. Costs of central facilities such as a library or recreationalfacilities
5. Costs of Social Security and employee benefits such as pensions and health insurance.

Factors affecting software pricing



Factor	Description
Market opportunity	A development organisation may quote a low price because it wishes to move into a new segment of the software market. Accepting a low profit on one project may give the organisation the opportunity to make a greater profit later. The experience gained may also help it develop new products.
Cost estimate uncertainty	If an organisation is unsure of its cost estimate, it may increase its price by some contingency over and above its normal profit.
Contractual terms	A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer.
Requirements volatility	If the requirements are likely to change, an organisation may lower its price to win a contract. After the contract is awarded, high prices can be charged for changes to the requirements.
Financial health	Developers in financial difficulty may lower their price to gain a contract. It is better to make a smaller than normal profit or break even than to go out of business.

Cost Estimation Process



Function points..

- **STEP 2:** Multiply each number by a weight factor, according to complexity (**simple**, **average** or **complex**) of the parameter, associated with that number. The value is given by a table:

Parameter	simple	average	complex
users inputs	3	4	6
users outputs	4	5	7
users requests	3	4	6
files	7	10	15
external interfaces	5	7	10

- **STEP 3:** Calculate the total **UFP**(Unadjusted Function Points)

- **STEP 4:** Calculate the total **TCF** (Technical Complexity Factor) by givinga value
between 0 and 5 accordingto the importance of the following points:

- **<u>Technical Complexity Factors:</u>**

  - 1. Data Communication
  - 2. Distributed DataProcessing
  - 3. Performance Criteria
  - 4. Heavily Utilized Hardware
  - 5. High Transaction Rates
  - 6. Online Data Entry
  - 7. Online Updating End-user
  - 8.
  - 9. Efficiency
  - 10. Complex Computations
  - 11. Reusability
  - 12. Ease of InstallationEase of
  - 13. OperationPortability
  - 14. Maintainability

- **STEP 5:** Sum the resulting numbers too obtain **DI** (degree of influence)
- **STEP 6: TCF** (Technical Complexity Factor) by given by theformula
  - ***TCF=0.65+0.01*DI***
- **STEP 6:** Function Points are by given by theformula
  - ***FP=UFP*TCF***

Relation between LOC andFP

  - *LOC = LanguageFactor * FP*
  - where
    - LOC (Lines of Code)
    - FP (FunctionPoints)

Effort Computation

- The **Basic COCOMO model** computes effort as a function of program size. The Basic COCOMO equation is:

– ***E = aKLOC^b***

- Effort for three modes of Basic COCOMO.

Mode	**a**	**b**
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.20

- The **intermediate COCOMO model** computes effort as a function of program size and a set of cost drivers. The Intermediate COCOMO equation is:

– ***E = aKLOC^b*EAF***

- Effort for three modes of intermediate COCOMO.

Mode	**a**	**b**
Organic	3.2	1.05
Semi-detached	3.0	1.12
Embedded	2.8	1.20

TotalEAF = Product of the selected factors

Adjusted value of Effort:Adjusted PersonMonths:

APM = (Total EAF) * PM

S/W Development Time

- **DevelopmentTimeEquationParameterTable:**

Parameter	Organic	Semi-detached	Embedded
C	2.5	2.5	2.5
D	0.38	0.35	0.32

DevelopmentTime, **TDEV=C* (APM**D)**

NumberofPersonnel, **NP=APM / TDEV**

Distribution of Effort

- A development process typically consists ofthe following stages:
- Requirements Analysis
- Design (High Level +Detailed)
- Implementation & Coding
- Testing (Unit + Integration)

Error Estimatio

- Calculatetheestimatednumberoferrorsinyourdesign,i.e.total errors found inrequirements,specifications,code,usermanuals,andbad fixes:
  - AdjusttheFunctionPointcalculatedinstep1
    AFP=FP** 1.25

Usethefollowingtableforcalculatingerror estimates

Error Type	Error / AFP
Requirements	1
Design	1.25
Implementation	1.75
Documentation	0.6
Due to Bug Fixes	0.4

Estimation

- **LOC based estimation**
- **Source lines of code** (**SLOC**), also known as **lines of code** (**LOC**), is a software metric used to measure the size of a computer program by counting the numberof lines in the text of the program's source code.
- SLOC is typically used to predict the amount of effort that will be required to develop a program, as well as to estimate programming productivity or maintainability once the software is produced.
- Lines used for commenting the code and header file are ignored.
- **Two major types of LOC:**

1. **Physical LOC**
   - Physical LOC is the count of lines in the text of the program's source code including
     comment lines.
   - Blank lines are also included unless the lines of code in a section consists ofmore
     than 25% blank lines.

2. Logical LOC

- Logical LOC attempts to measure the number of executable statements, but their
  specific definitions are tied to specific computer languages.
- Ex: Logical LOC measure for C-like programming languages is the number of statement-terminating semicolons(;)

LOC-based Estimation

The problems of lines of code (LOC)

–Different languages lead to different lengths of code
–It is not clear how to count lines of code
– A report, screen, or GUI generator can generate thousands of lines of code in minutes
– Depending on the application, the complexity of code is different.

S/W Cost Estimation

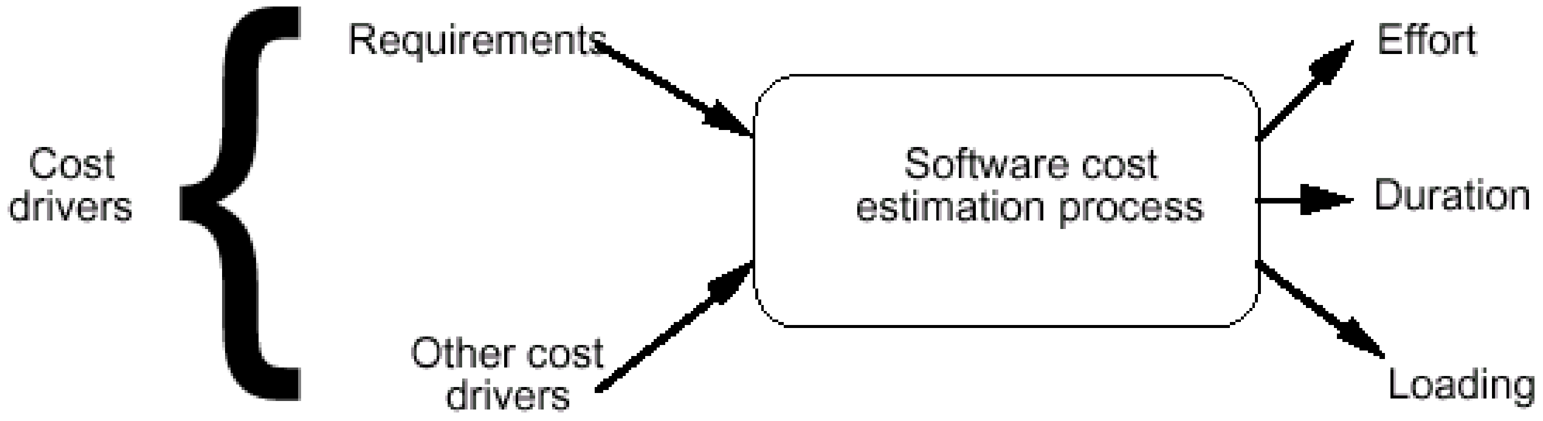


Figure 1. Classical view of software estimation process.

INTRODUCTION TO COCOMO MODELS

- The **Constructive Cost Model** (COCOMO) is the most widely used software estimation model in the world.
- The COCOMO model predicts the **effort** and **duration** of a project based on inputs relating to the size of the resulting systems and a number of "**cost drives**" that affect productivity.

Effort

- Effort Equation
  - **PM = C * (KDSI)^{n} (**person-months)
    - where **PM** = number of person-month (=152 working hours),
    - **C** = a constant,
    - **KDSI** = thousands of "delivered source instructions" (DSI) and
    - **n** = a constant.

INTRODUCTION TO COCOMO MODELS

- Productivityequation
  - **(DSI) /(PM)**
    - where **PM** = number of person-month (=152 working hours),
    - **DSI** = "delivered sourceinstructions—
- Schedule equation
  - **TDEV = C * (PM)^{n}(**months)
    - where TDEV = number of months estimated forsoftware development.
- Average Staffing Equation

  –
- **(PM) / (TDEV)**
- **(**FSP)
- whereFSPmeansFul-time⁾-equivalentSoftware Personnel.

COCOMO MODELS

- COCOMO is defined in terms of three different models:
  - the **Basic model**,
  - the **Intermediate model**, and
  - the **Detailed model**.
- The more complex models account for more factors that influence software projects, and make more accurate estimates.

The development model

- The most important factors contributing to a project's duration and cost is the Development Mode
- **Organic Mode:**The project is developed in a familiar,stable environment, and the product is similar to previously developed
- **Mode:** The project'scharacteristics are intermediate between Organic and Embedded.
- **Embedded Mode:** The project is characterized by tight, inflexible constraints and interface requirements. An embedded mode project will require a great deal of innovation.

Modes

Feature	Organic	Semidetached	Embedded
Organizational understanding of product and objectives	Thorough	Considerable	General
Experience inworking with relatedsoftware systems	Extensive	Considerable	Moderate
Need forsoftware conformance withpre-established requirements	Basic	Considerable	Full
Need for software conformance with external interface specifications	Basic	Considerable	Full

Feature	**Organic**	**Semidetached**	**Embedded**
Concurrent development of associated new hardware and operational procedures	Some	Moderate	Extensive
Need for innovative data processing architectures, algorithms	Minimal	Some	Considerable
Premium on early completion	Low	Medium	High
Product size range	<50 KDSI	<300KDSI	All

What Is project scheduling

write a computer program for an automated manufacturing application. The reason for his selection was simple. He was the only person in his technical group who had attended a computer programming seminar. He knew the ins and outs of assembly language and FORTRAN but nothing about software engineering and even less about project scheduling and tracking. His boss gave him the appropriate manuals and a verbal description of what had to be done. He was informed that the project must be completed in two months. He read the manuals, considered his approach, and began writing code. After two weeks, the boss called him into his office and asked how things were going.̄Really great, the young engineer with youthful enthusiasm.̄This was much simpler thought. I'ŵ probably close to 75 percent finished.

- ✓ You'leselected an appropriate process model.
- ✓ You'e identified the software engineering tasks that have to be performed.
- ✓ You estimated the amount of work and the number of people, you know the deadline, you'leeven considered the risks.
- ✓ Now it's time to connect the dots. That is, you have to create

What Is project scheduling

- **Why it’s Important?**
  - ✓ In order to build a complex system, many software engineering tasks occur in parallel.
  - ✓ The result of work performed during one task may have a profound effect on work to be conducted in another task.
  - ✓ These interdependencies are very difficult to understand withouta schedule.
  - ✓ It’s also virtually impossible to assess progress on a moderate or large software project without a detailedschedule
- **What are the steps?**
  - ✓ The software engineering tasks dictated by the software are refined for the functionality to be built.

Effort and duration are allocated to each task and a task network is created in a manner that enables the software team to meet the delivery deadline established.

What Is project scheduling

Basic Concept of Project Scheduling

✓An unrealistic deadline established by someone outside the software development group and forced on managers and practitioner's within the group.

✓Changing customer requirements that are not reflected in schedulechanges.

✓An honest underestimate of the amount of effort and/or the number of resourcesthat 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.

▪**Why should we do when the management demands that we make a dead line I impossible?**

✓ Perform a detailed estimate using historical data from past projects.

✓ Determine the estimated effort and duration for the project.

✓ Using an incremental process model, develop a software engineering strategy that will deliver critical functionality by the imposed deadline, but delay other functionality until later. Document the plan.

✓ Meet with the customer and (using the detailed estimate), explain why the imposed deadline is unrealistic.

Project scheduling

- Project Scheduling
- Basic Principles
- The Relationship Between People and Effort
- Effort Distribution
- Software project scheduling is an action that distributes estimated effort acrossthe planned project duration by allocating the effort to specific software engineering tasks.
- During early stages of project planning, a macroscopic schedule is developed. As the project gets under way, each entry on the macroscopic schedule is refined into a detailed schedule.

Project scheduling

- **Basic Principles of ProjectScheduling.**

1. **Compartmentalization:** The project must be compartmentalized into a number of manageable activities and tasks. To accomplish compartmentalization, both the product and the process are refined.
2. **Interdependency:** The interdependency of each compartmentalized activity or task must be determined. Some tasks must occur in sequence, while others can occur in parallel. Other activities can occur independently.
3. **Time allocation:** Each task to be scheduled must be allocated some number of work units (e.g., persoŶ-days of effort). In addition, each task must be assigned a start date and a completion date. whether work will be conducted on a full-time or part-time basis.
4. **Effort validation:** Every project has a defined number of people on the software team. The project manager must ensure that no more than the allocated number of people have been scheduled at any given time.
5. **Defined responsibilities**. Every task that is scheduled should be assigned to a specific team member.

Project scheduling

6. **Defined outcomes:** Every task that is scheduled should have a defined outcome. For software projects, the outcome is normally a work (e.g., the Product design of a component) or a part of a work product. Work products are often combined in deliverables.

7. **Defined milestones:** Every task or group of tasks should be associated with a project milestone. A milestone is accomplished when one or more work products has been reviewed for quality and has been approved. Each of these principles is applied as the project scheduleevolves.

UNIT –II
SOFTWARE REQUIREMENTS AND REQUIREMENTS ENGINEERING PROCESS

Contents

- Software requirements: Functional and nonfunctional, user requirements, system requirements
- Software requirements document
- Requirement engineering process
- Feasibility studies, requirements elicitation and analysis
- Requirements validation, requirements management
- Classical analysis: Structured system analysis, petri nets, data dictionary.

Requirement Validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Requirements error costs are high so validation is very important
- Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

•**Requirements Checking**

- Validity : Does the system provide the functions which best support the customers needs
- Consistency: Are there any requirements conflicts?
- Completeness: Are all functions required by the customer included?
- Realism: Can the requirements be implemented given available budget and technology
- Verifiability: Can the requirements be checked?

- Requirements reviews
  - Systematic manual analysis of the requirements.
- Prototyping
  - Using an executable model of the system to check requirements.
- Test-casegeneration
  - Developing tests for requirements to checktestability.

Requirements Reviews

- Regular reviews should be held while the requirements definition is being formulated.
- Both client and contractor staff should be involved in reviews.
- Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.
- Don't underestimate the problems involved in requirements validation. Ultimately, it is difficult to show that a set of requirements does in fact meet a user’s needs. Users need to picture the system in operation and imagine how that system would fit into their work.
- It is hard even for skilled computer professionals to perform this type of abstract analysis and harder still for system users. As a result, you rarely find all requirements problems during the requirements validation process. It is inevitable that there will be further requirements changes to correct omissions and misunderstandings after the requirements document has been agreed upon.

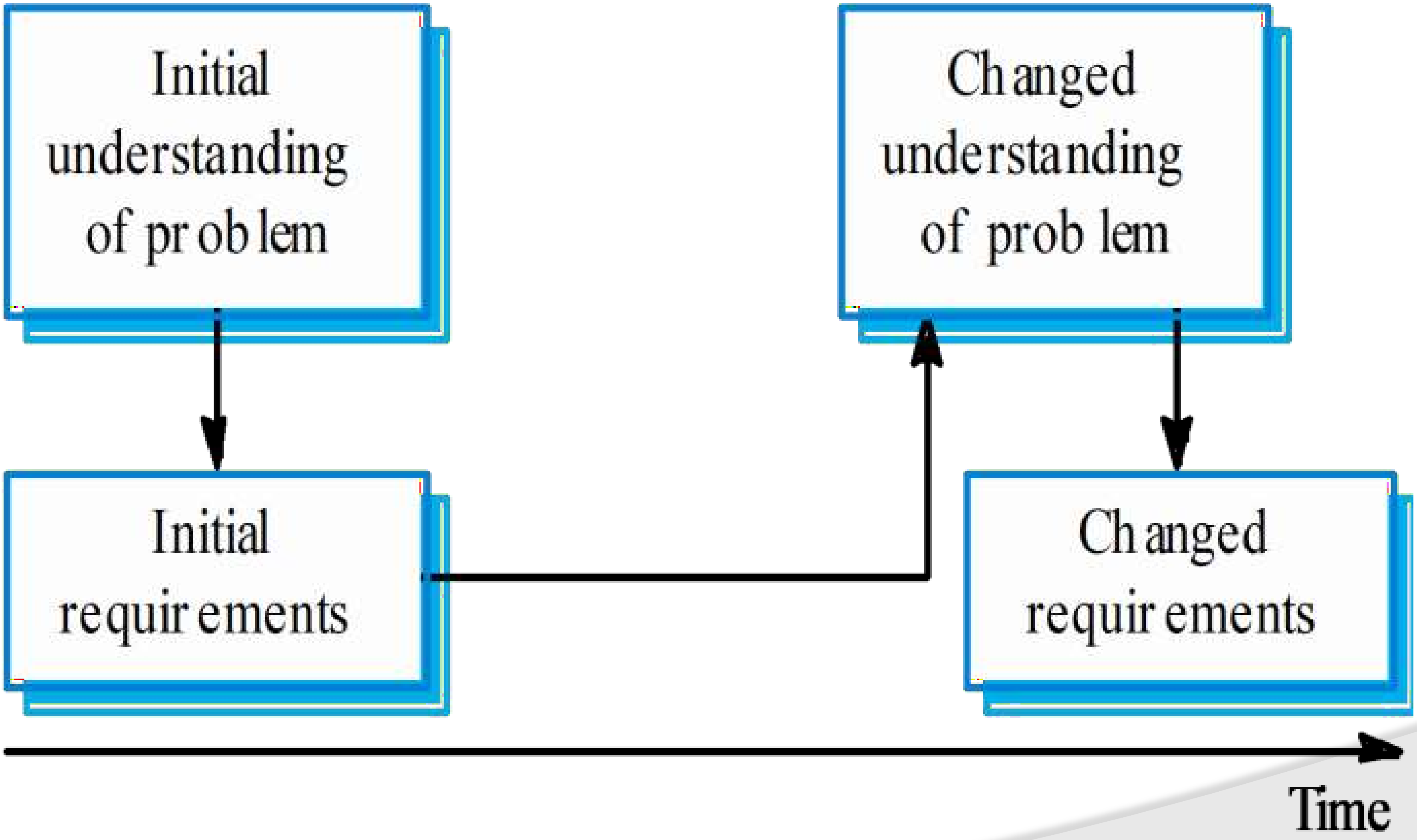
Requirement Management

- Requirements during the requirements engineering process and system development.
- Requirements are inevitably incomplete and inconsistent
  - ✓ New requirements emerge during the process as business needs change and a better understanding of the system is developed;
  - ✓ Different viewpoints have different requirements and these are often contradictory.

Requirements Change

- The priority of requirements from different viewpoints changes during the development process.
- System customers may specify requirements from a business perspective that conflict with end-user requirements.
- The business and technical environment of the system changes during its development.

Requirements Evaluation


Initial understanding of problem
Changed understanding of problem
Initial requirements
Changed requirements
Time

- **Enduring requirements**
  - These are relatively stable requirements that derive from the core activity of the organization
  - Relate directly to the domain of the system
  - These requirements may be derived from domain models that show the entities and relations which characterize an application domain
  - For example, in a hospital there will always be requirements concerned with patients, doctors, nurses, treatments, etc.
- **Volatile Requirements**
  - These are requirements that are likely to change during the system development process or after the system has been become operational.
  - Examples of volatile requirements are requirements resulting from government health-care policies or healthcare charging mechanisms.

- Volatile requirements
  - These are requirements that are likely to change during the system development process or after the system has been become operational.
  - Examples of volatile requirements are requirements resulting from government health-care policies or healthcare charging mechanisms

Traceability

- Traceability is concerned with the relationships between requirements, their sources and the system design
- Source traceability
  - Links from requirements to stakeholders who proposed these requirements;
- Requirements traceability
  - Links between dependent requirements;
- Design traceability
  - Links from the requirements to the design;

A Traceability Matrix

Req. id	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2
1.1		D	R					
1.2			D			D		D
1.3	R			R				
2.1			R		D			D
2.2								D
2.3		R		D				
3.1								R
3.2							R	

Case Tool Support

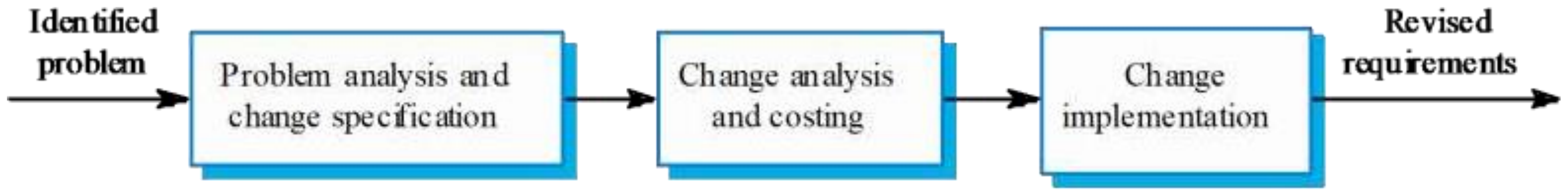
- Requirements storage
  - Requirements should be managed in a secure, managed data store.
- Change management
  - The process of change management is a workflow process whose stages can be defined and information flow between these stages partially automated.
- Traceability management
  - Automated retrieval of the links between requirements.

Requirements Management

- During the requirements engineering process, one has to plan:
  - Requirements identification
    - How requirements are individually identified;
  - A change management process
    - The process followed when analyzing a requirements change;
  - Traceability policies
    - The amount of information about requirements relationships that is maintained;
  - CASE tool support
    - The tool support required to help manage requirements change;

Requirements Change

- Should apply to all proposed changes to the requirements.
- Principal stages
  - Problem analysis. Discuss requirements problem and propose change;
  - Change analysis and costing. Assess effects of change on other requirements;
  - Change implementation. Modify requirements document and other documents to reflect change.



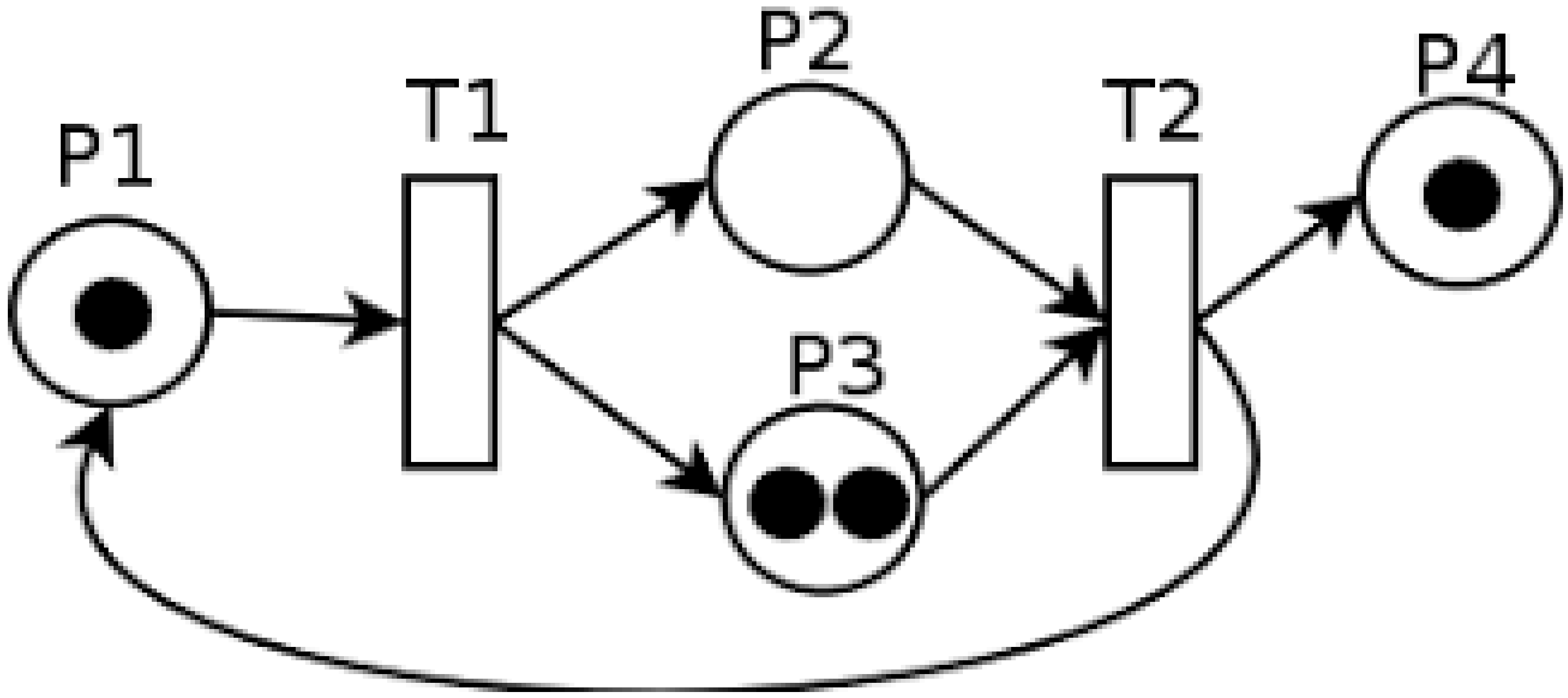
Identified problem
Problem analysis and change specification
Change analysis and costing
Change implementation
Revised requirements

Classical Analysis

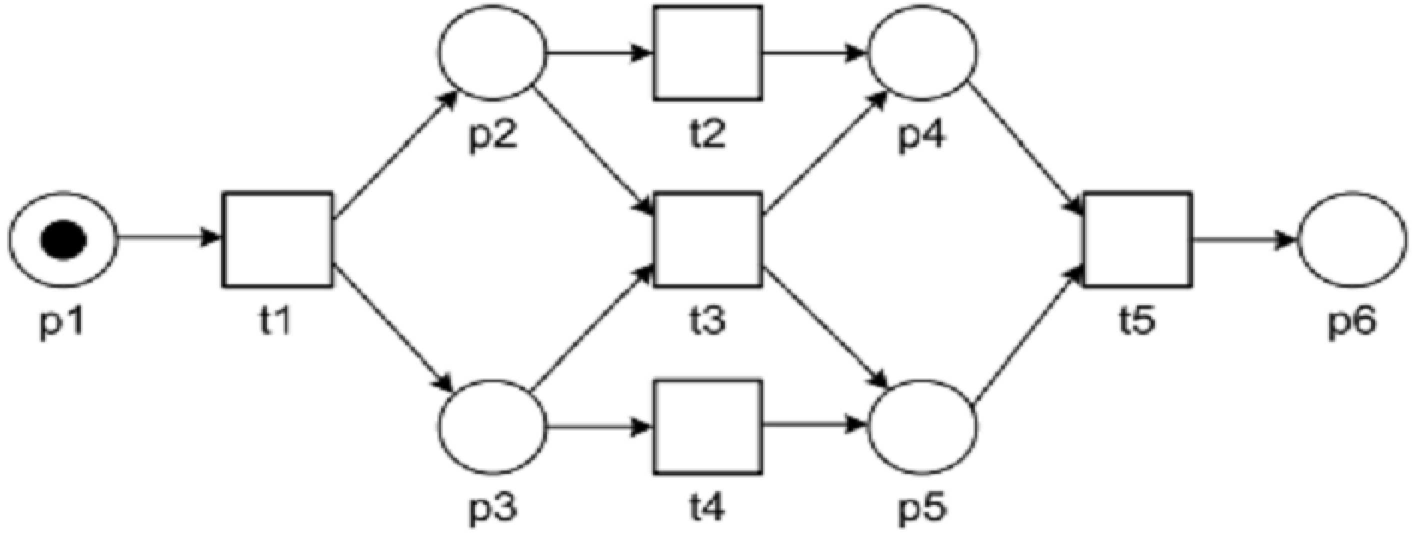
High-Level Petri Nets

- Theclassical Petri net was invented by CarlAdam Petri in 1962.
- A lot of research has been conducted (>10,000 publications). Until 1985 it was mainly used by theoreticians.
- Since the 80's their practical use has increased because of the introduction of high-level Petri nets and the availability ofmany tools.
- **High-level Petri nets** are Petri nets extended with color (for the modeling of attributes)
    - ✓time (for performance analysis)
    - ✓hierarchy (for the structuring of models, DFD's)
- Petri Nets can be used to rigorously define a system (reducing ambiguity,making the operationsofasystemclear,allowingustoprove propertiesofa systemetc.)
- They areoften used for distributed systems (with several subsystems acting independently) and forsystems with resourcesharing.

The Classical Petrint Model



- Petri Nets can be used to rigorously define a system (reducing ambiguity,making the operationsofasystemclear,lowingustoprove propertiesofa systemetc.)
- They areoften used for distributed systems (with several subsystems acting independently) and forsystems with resourcesharing.
- Sincethere maybe morethan onetransition inthe Petri Netactiveatthe
- sametime (andwe do notknowwhich will fade first), they arenon-deterministic.



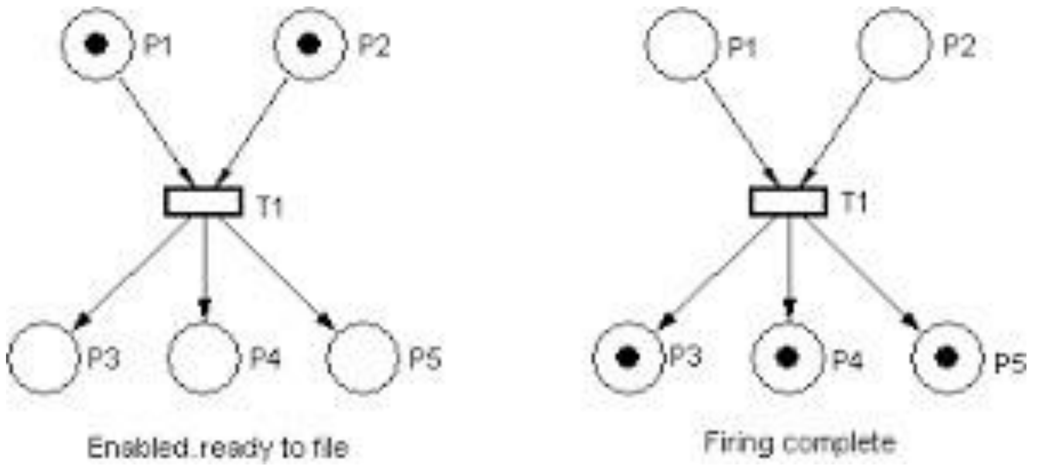
- Transition t1 has three input places (p1, p2 and p3) and two output places (p3 and p4).
- Place p3 is both an input and an output place of t1.

- Transition t1 has three input places (p1, p2 and p3) and two output places (p3 and p4).
- Place p3 is both an input and an output place of t1.

Transition with Multiple Input

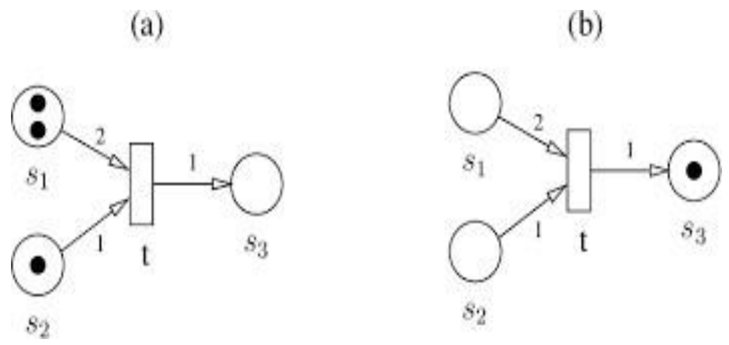
•Transition t1 has three **input places** (p1, p2 and p3) and two **output places** (p3 and p4).

•Place p3 is both an input and an output place of t1.



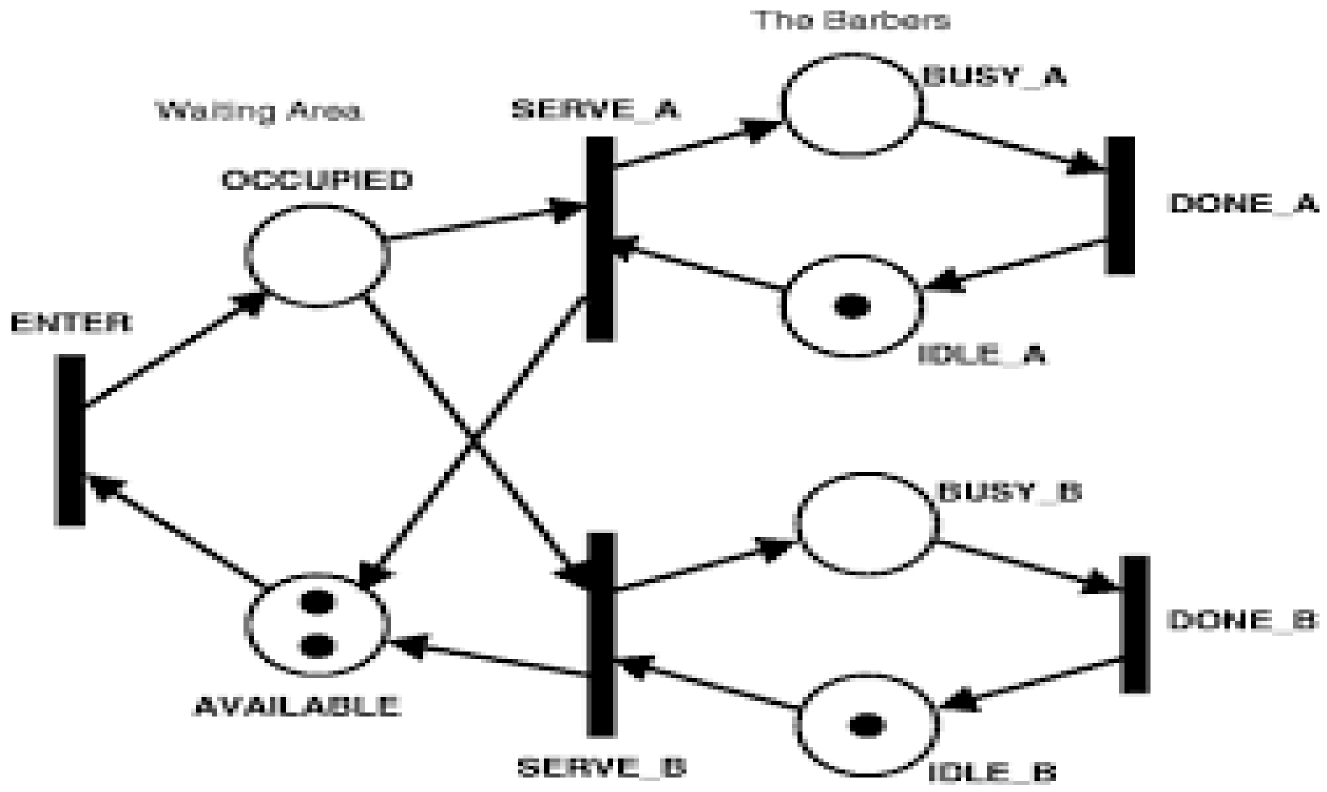
Transitions with Multiple Inputs

- An enabled transition may**fire**.
- Firing corresponds to **consuming** tokens from the input places and **producing** tokens for the outputplaces.



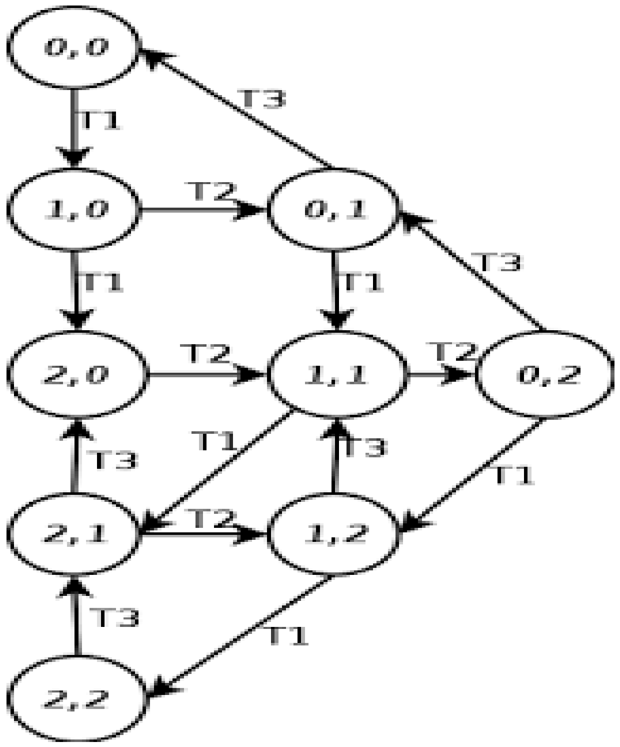
•Firing is **atomic** (only one transition fires at a time, even if more than one is enabled)

Example for Pertinet

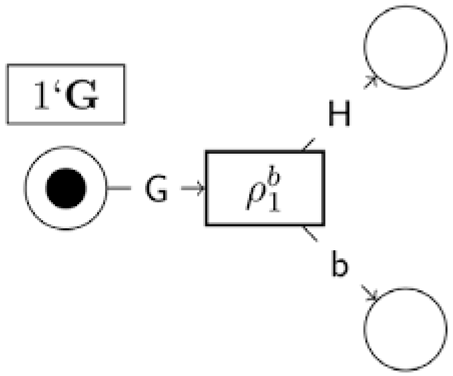


Creating / Consuming Token

- A transition without any input can fire at any time and produces tokens in the connected places:



Creating/ConsumingTokens



Two fight for the sametoken:conflict. Even if there are two tokens, there is stil a conflict.

Thenexttransitiontofire(t1 ort2) is arbitrary (non-deterministic).

Data Dictionary

- A tool for recording and processing information(metadata) organization uses.
- A central catalogue formetadata.
- Can be integrated within the DBMS or be separate
- May be referenced during system design, programming, and by actively executing programs.
- Can be used as a repository for common code (e.g. library routines).

Benefits of DDS

Benefits of a DDS are mainly due to the fact that it is a central store of information about the database.

Benefits include -

- Improved documentation and control
- Consistency in data use
- Easier data analysis
- Reduced data redundancy
- Simpler programming
- The enforcement of standards
- Better means of estimating the effect of change.

DD Management

- With so much detail held on the DDS, it is essential that an cross-referencing facility is provided by the DDS.
- The DDS can produce reports for use by the data administration staff (to investigate the efficiency of use and storage of data), systems analysts, programmers, and users.
- A query language is provided for ad-hoc queries. If the DD is the DBMS, then the query language will be that of the DBMS itself.

Management Objective

From an management point of view, the DDS should

- Provide facilities for documenting information collected during computer project.
- provide details of applications usage and their data usage once a system has been implemented, so that analysis and redesign may be facilitated as the environment changes.
- make access to the DD information easier than a paper-based approach by providing cross-referencing and indexing facilities.
- make extension of the DD information easier.
- Encourage systems analysts to follow structured methodologies.

Management Advantages

A number of possible benefits may come from using a DDS:

- Improve control and knowledge about the data resource.
- Allows accurate assessment of cost and time scale to effect any changes.
- Reduces the clerical load of database administration, and gives more control
- Over the design and use of the database.
- Accurate data definitions can be provided securely directly to programs.
- Aid the recording, processing, storage and destruction of data and associated documents.

A DDS is a useful management tool, but at a price.

- The DDS project may itself take two or three years.
- It needs careful planning, defining the exact requirements designing its contents, testing, implementation and evaluation.
- The cost of a DDS includes not only the initial price of its installation and any hardware requirements, but also the cost of collecting the information entering it into the DDS, keeping it up-to-date and enforcing standards.
- The use of a DDS requires management commitment, which is not easy to achieve, particularly where the benefits are intangible and long term.

UNIT-III
DESIGN ENGINEERING, CREATING AN ARCHITECTURAL DESIGN AND MODELING COMPONENT-LEVEL DESIGN

Design Process

- Quality Guidelines
  - A design should exhibit an architecturethat
    - (1) Has been created using recognizable architectural styles or patterns
    - (2) Is composed of components that exhibit good design characteristics (these are discussed later in this chapter)
    - (3) Can be implemented in an evolutionary fashion, thereby facilitating implementation and testing.
  - A design should be modular; that is, the software should be logically partitioned into elements or subsystems.
  - A design should contain distinct representations of data, architecture, interfaces, and components.
  - A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns.

- Quality Guidelines
- A design should lead to components that exhibit independent functional characteristics.
- A design should lead to interfaces that reduce the complexity of Connections between componentsand with the environment.
- A design should be derived using a repeatable method that is driven
- A design should be represented using a notation that effectively
- communicates its meaning.

Design Model

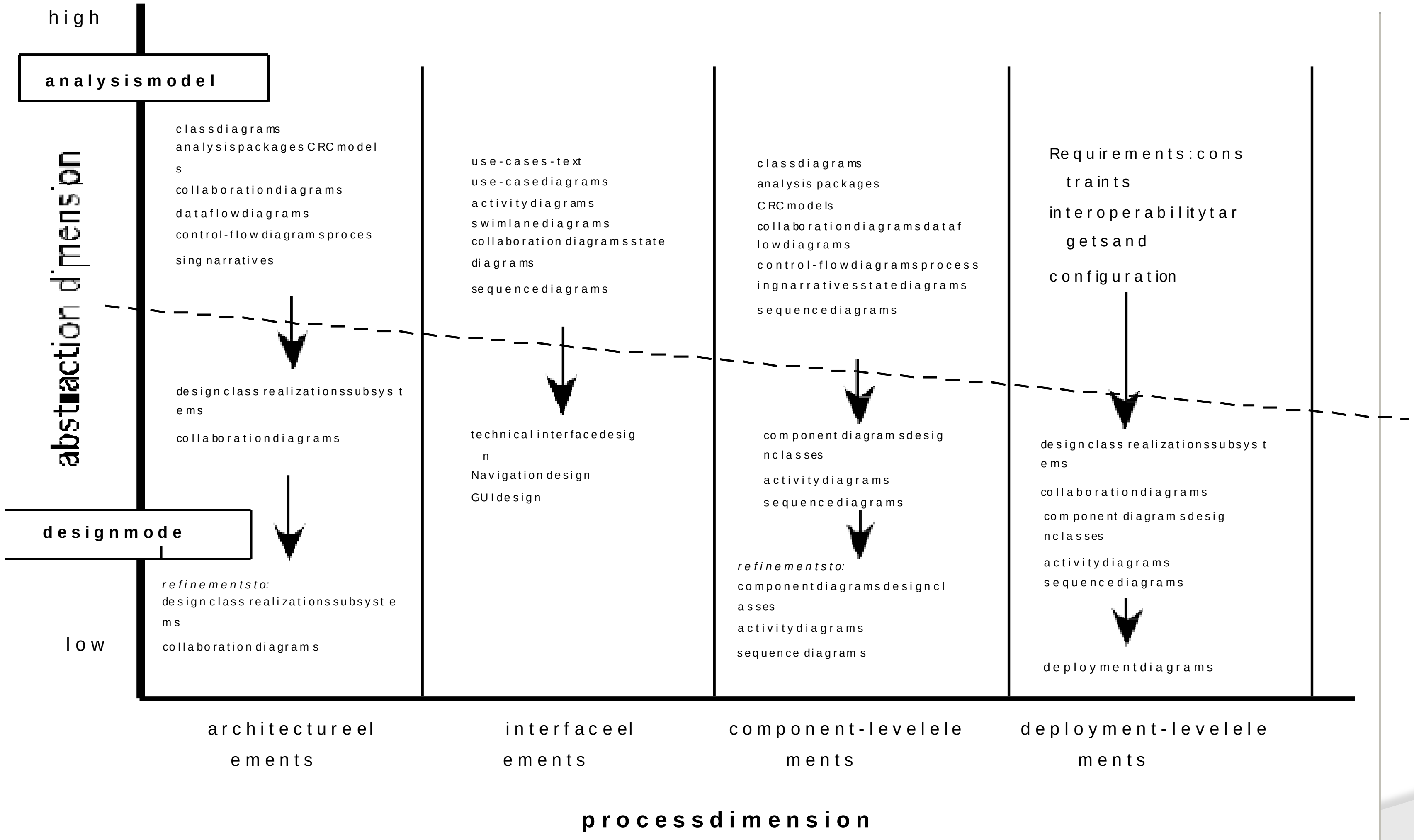
Design model can be viewed as

–Process dimension indicating the evolution of the design model as design tasks are executed as part of the software process.

–Abstraction dimension represents the level of detail as each element of the analysis model is transformed into a design equivalent and then refined iteratively

- Design Model Elements are as follows
- Data design elements
- Architectural design elements
- Interface design elements
- Component-level design elements
- Deployment-level design elements

The DesignModel


high
analysismodel
abstraction dimension
classdiagrams analysispackages CRCmodels collaborationdiagrams dataflowdiagrams control-flow diagram sprocessing narratives
use-cases-text use-casediagrams activitydiagrams swimlanediagrams collaboration diagramsstate diagrams sequencediagrams
classdiagrams analysis packages CRCmodels collaborationdiagramsdataflowdiagrams control-flowdiagramsprocessingnarrativesstatediagrams sequencediagrams
Requirements:constraints interoperabilitytargetsand configuration
design class realizationssubsystems collaborationdiagrams
technicalinterfacedesign Navigationdesign GUIdesign
component diagramsdesignclasses activitydiagrams sequencediagrams
design class realizationssubsystems collaborationdiagrams component diagramsdesignclasses activitydiagrams sequencediagrams
designmodel
refinementsto: design class realizationssubsystems collaboration diagrams
refinementsto: componentdiagramsdesignclasses activitydiagrams sequence diagrams
deploymentdiagrams
low
architectureelements
interfaceelements
component-levelelements
deployment-levelelements
processdimension

- Data design elements
  - Data design creates a model of data and/or information that is represented at a high level of abstraction.
  - Data model is then refined into progressively more implementation-specific representations that can be processed by thecomputer-based system
- Architectural level databases and files
- Component level data structures

Architectural design elements

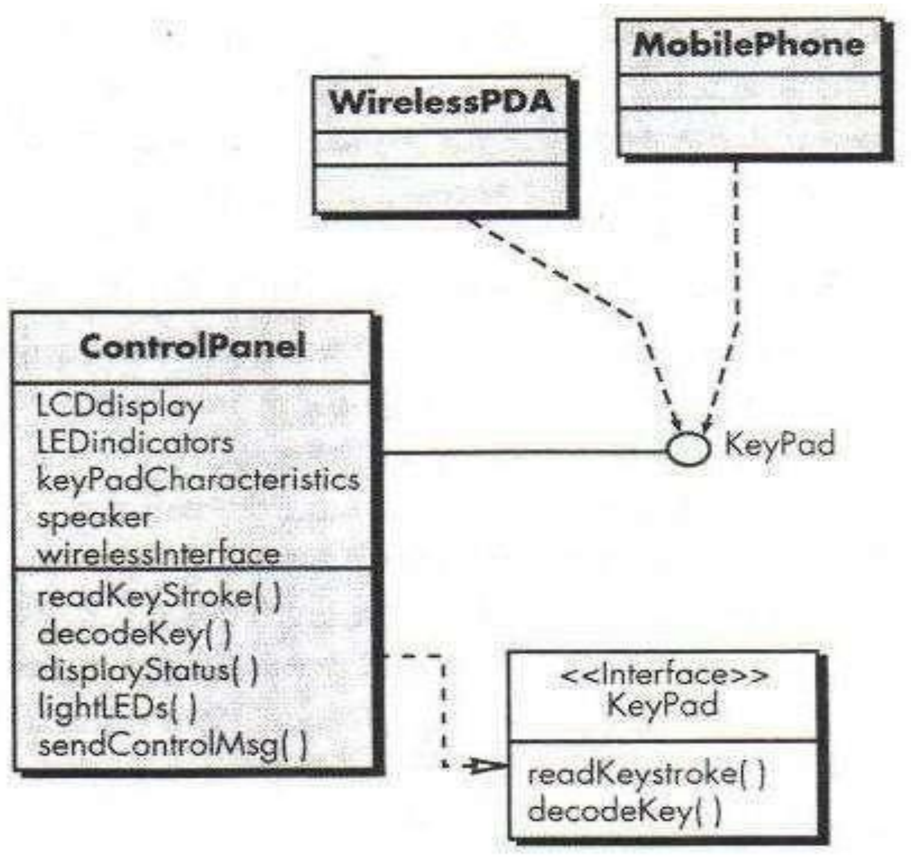
•The architectural design for software is the equivalent to the floor plan of a house. The floor plan depicts the overall layout of the rooms; their size, shape, and relationship to one another; and the doors and windows that allow movement into and out of the rooms. The floor plan gives us an overall view of the house. Architectural design elements give us an overall view of the software.

- The architectural model is derived from
  - Information about the application domain for the software to be built
  - Specific requirements model elements such as data flow diagrams or analysis classes, their relationships and collaborations for the problem at hand
  - The availability of architectural patterns and styles

- Interface design elements
  - The interface design elements for software tell how information flows into and out of the system and how it is communicated among the components defined as part of the architecture
  - Important elements of interface design
    - The user interface (UI): Usability design incorporates aesthetic elements (e.g., layout, color, graphics, interaction mechanisms), ergonomic elements (e.g., information layout and placement, metaphors, UI navigation), and technical elements (e.g., UI patterns, reusable components). In general, the UI is a unique subsystem within the overall application architecture.
    - External interfaces to other systems, devices, networks or other producers or consumers of informationThe design of external interfaces requires definitive information about the entity to which information is sent or received.
    - Internal interfaces between various design componentsThe design of internal interfaces is closely aligned with component-level design

Design Model

Design Model - Interface Elements



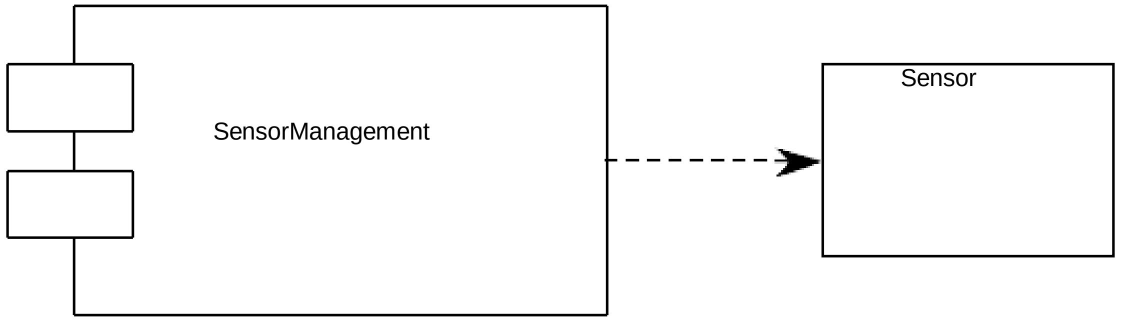
Design Model

Component-level design elements

•The component-level design for software is the equivalent to a set of detailed drawings (and specifications) for each room in a house. These drawings depict wiring and plumbing within each room, the location of electrical receptacles and wall switches, faucets, sinks, showers, tubs, drains, cabinets, and closets.

- Component-level design for software fully describes the internal detail of each software component.
- Component-level design defines data structures for all local data objects and algorithmic detail for all processing that occurs within a component and an interface that allows access to all component operations
- The design details of a component can be modelled at many different levels of abstraction.
- An UML activity diagram can be used to represent processing logic. Detailed procedural flow for a component can be represented using either pseudocode or diagrammatic form (e.g., flowchart or box diagram.

ComponentElements



- Deployment-level design elements
  - Deployment-level design elements indicate how software functionality and subsystems will be allocated within the physical computing environment that will support the software.
  - Deployment diagrams shows the computing environment but does not explicitly indicate configuration details

S/W Architecture

What is software Architecture

•When you consider the architecture of a building, many different attributes come to mind. At the most simplistic level, you think about the overall shape of the physical structure. But in reality, architecture is much more. It is the manner in which the various components of the building are integrated to form a cohesive whole.

•The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them

•Software architecture enables to

- Analyze the effectiveness of the design in meeting its stated requirements
- Consider architectural alternatives at a stage when making design changes is still relatively easy
- Reduce the risks associated with the construction of the software

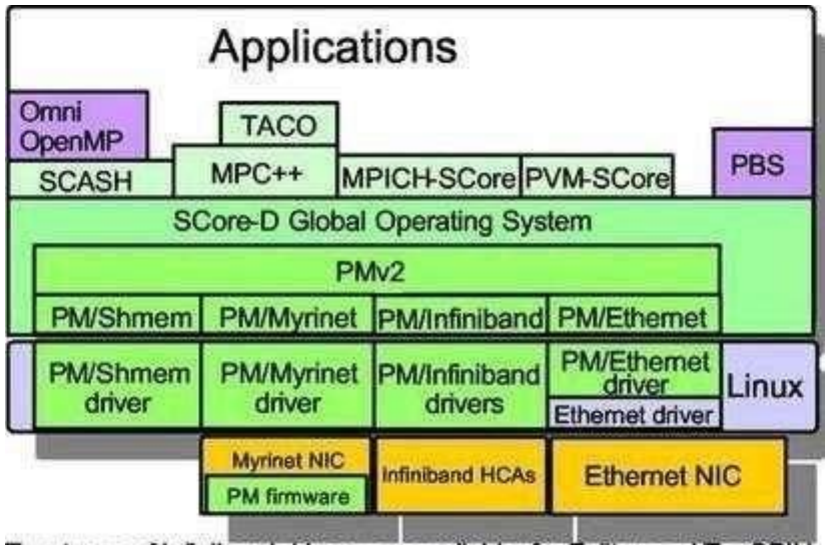
Why Is ArchitectureImportant?

•Representations of software architecture are an enabler for communication between all parties, interested in the development of a computer-based system.

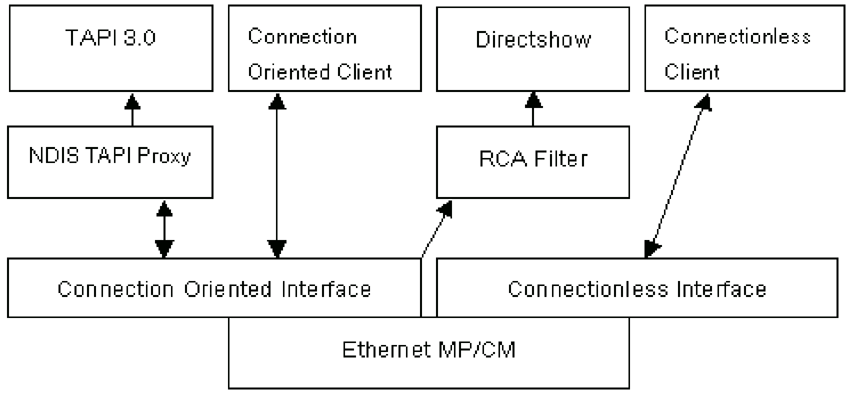
•The architecture highlights early design decisions that will have a profound impact on all software engineering work that follows and, as important, on ultimate success of the system as an operational entity.

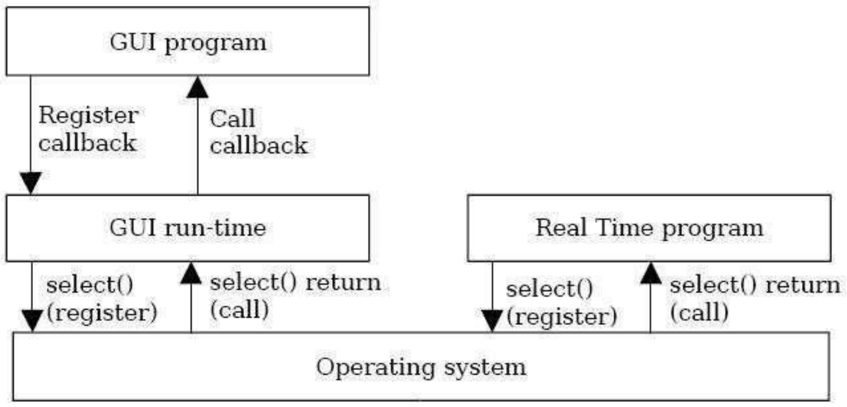
•Architecture constitutes a relatively small, intellectually graspable model of how the system is structured and how its components work together.

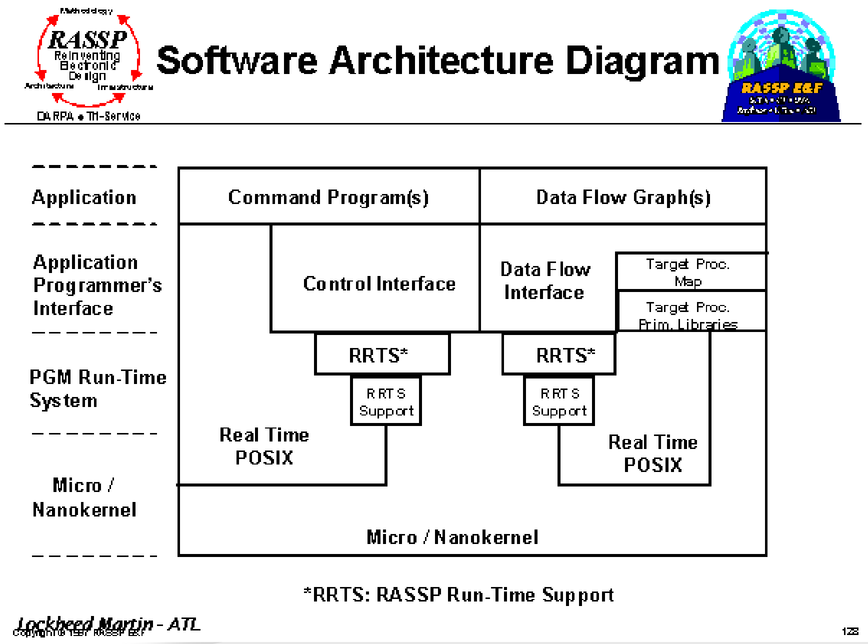
Ex: S/W Architecture Diagram



Two types of Infiniband drivers are available: for Fujitsu and TopSPIN







Software Architecture

Architectural Descriptions

- Each of us has a mental image of what the word architecture means. In reality, however, it means different things to different people.
- The implication is that different stakeholders will see an architecture from different viewpoints that are driven by different sets of concerns.
- An architectural description is actually a set of work products that reflect different views of the system.
- Developers want clear, decisive guidance on how to proceed with design.
- Customers want a clear understanding on the environmental changes that must occur and assurances that the architecture will meet their business needs.

Architectural Decisions

- Each view developed as part of an architectural description addresses a specific stakeholder concern.
- To develop each view (and the architectural description as a whole) the system architect considers a variety of alternatives and ultimately decides on the specific architectural features that best meet the concern.
- Therefore, architectural decisions themselves can be considered to be one view of the architecture.
- The reasons that decisions were made provide insight into the structure of a system and its conformance to stakeholder concerns.

S/W Architecture Styles

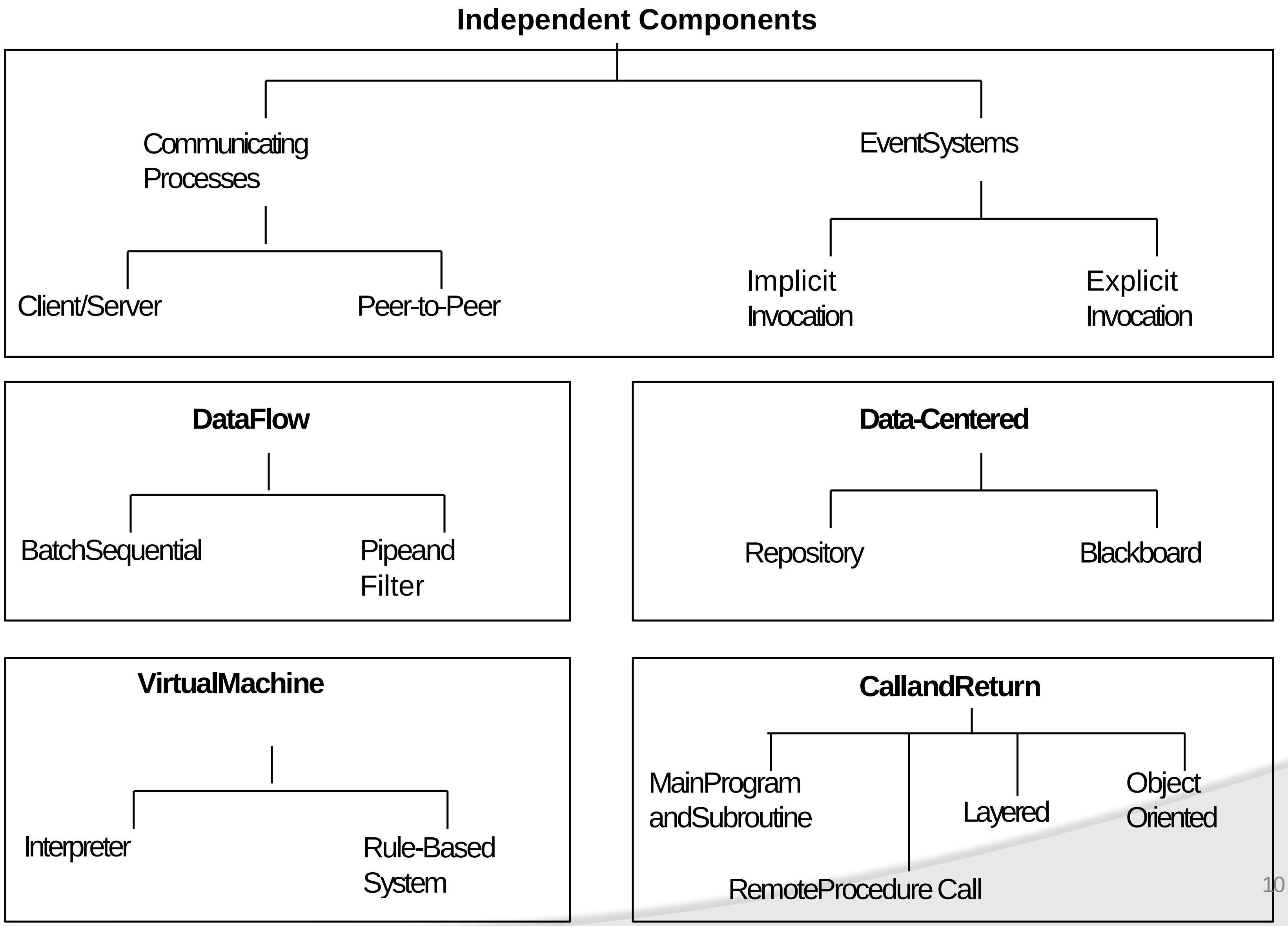
1. A Brief Taxonomy of Architectural Styles
2. Architectural Patterns
3. Organization and Refinement

S/W Architecture Styles of Home

S/W Architecture Style

- The software that is built for computer-based systems exhibit one of many architectural styles
- Each style describes a system category that encompasses
  - A set of component types that perform a function required by thesystem
  - A set of connectors (subroutine call, remote procedure call, data stream, socket) that enable communication, coordination, and cooperation among components
  - constraints that define how components can be integrated to form the system;
  - semantic models that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts

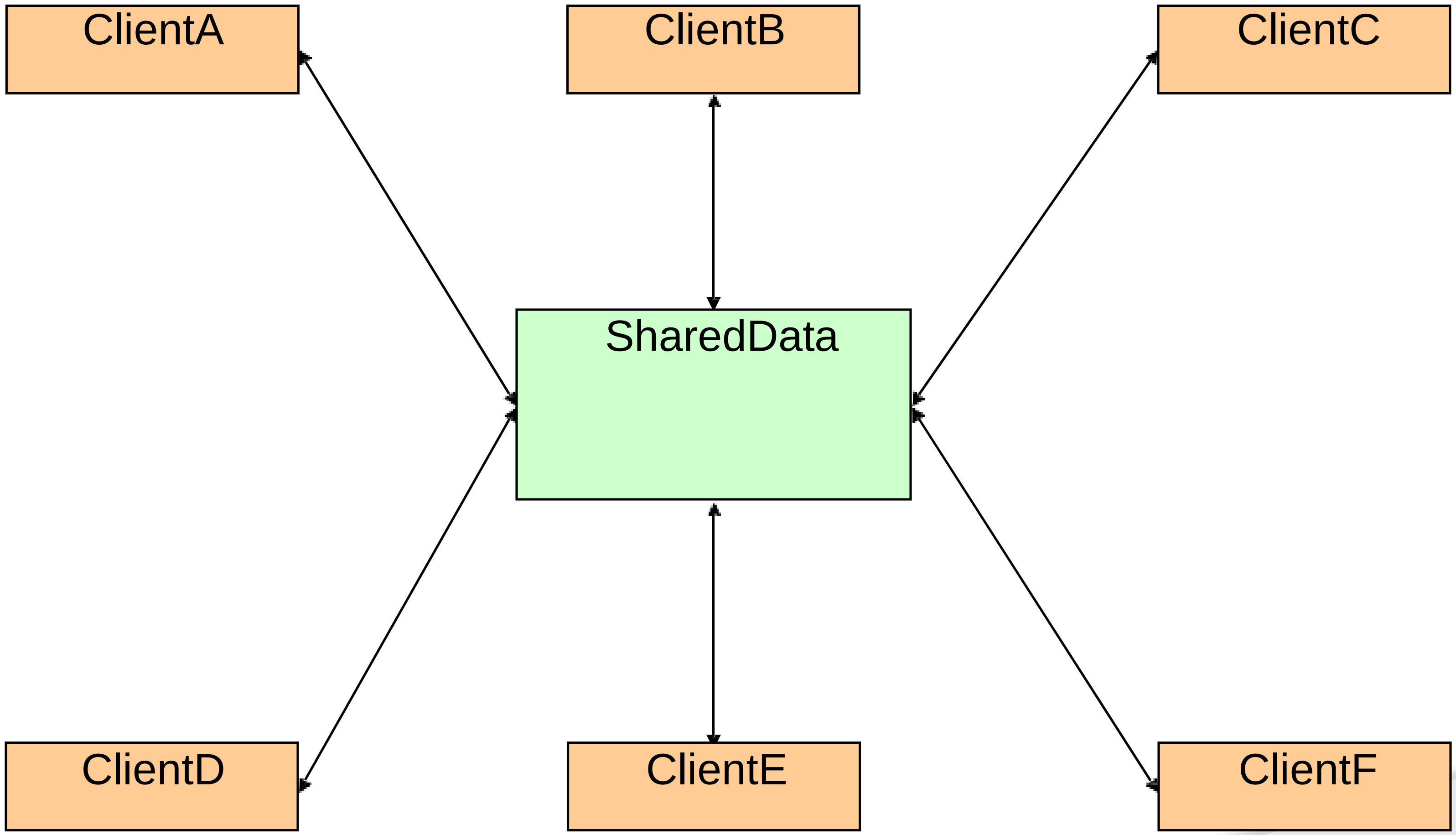
A Brief Taxonomy of Arct'l Style



A Brief Taxonomy of Architectural Styles

- Data-centered architectures.
- A data store (e.g., a file or database) resides at the center of this architecture and is accessed frequently by other components that update, add, delete, or otherwise modify data within the store.
- Illustrates a typical data-centered style. Client software accesses a central repository. In some cases the data repository is passive. That is, client software accesses the data independent of any changes to the data or the actions of other client software. A variation on this approach transforms the repository into a ̅ďlaĐkďoard

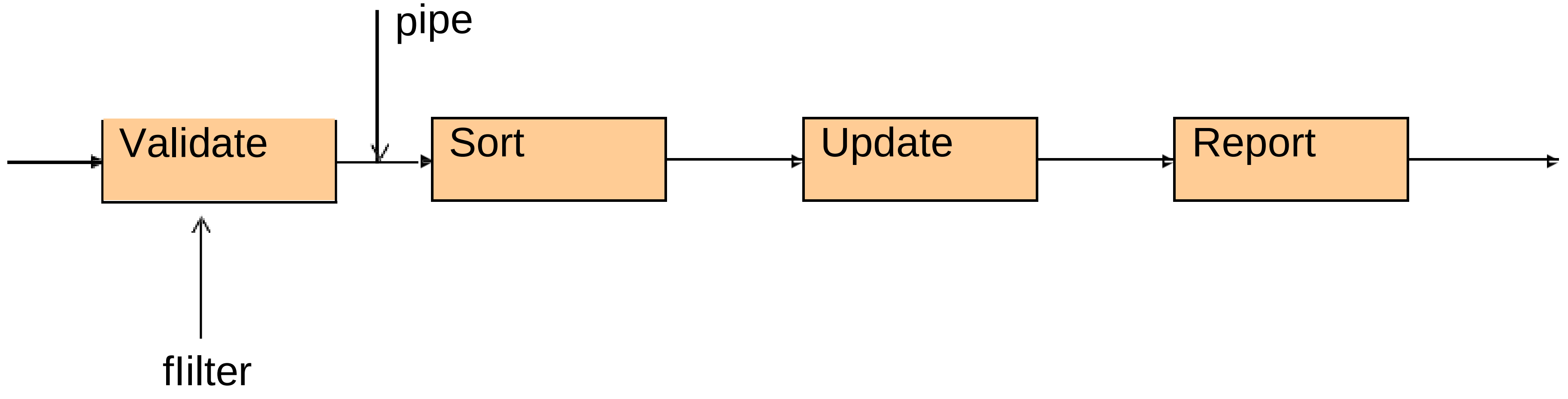
Data Centered Styles



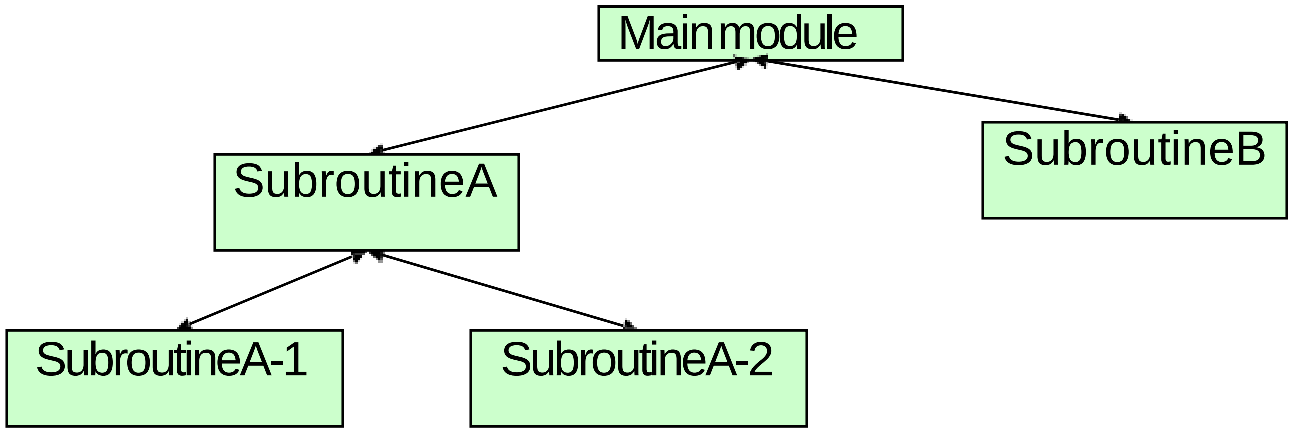
Data-flow architectures

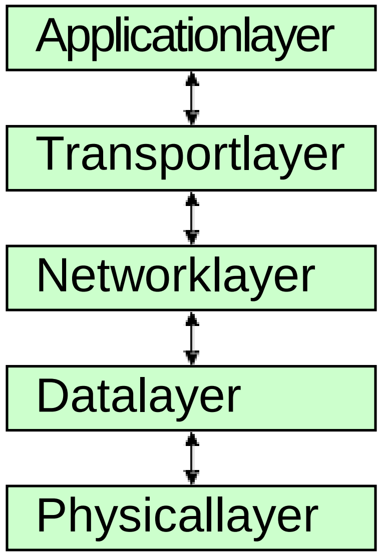
- This architecture is applied when input data are to be transformed througha series of computational or manipulative components into output data.
- A pipe-and-filter pattern shows has a set of components, called *filters, connected by pipes that transmit data from one component to thenext.*
- *Each* filter works independently of those components upstream and downstream, is designed to expect data input of a certain form, produces data output (to the next filter) of a specified form.
- However, the filter does not require knowledge of the workings of its neighboring filters.

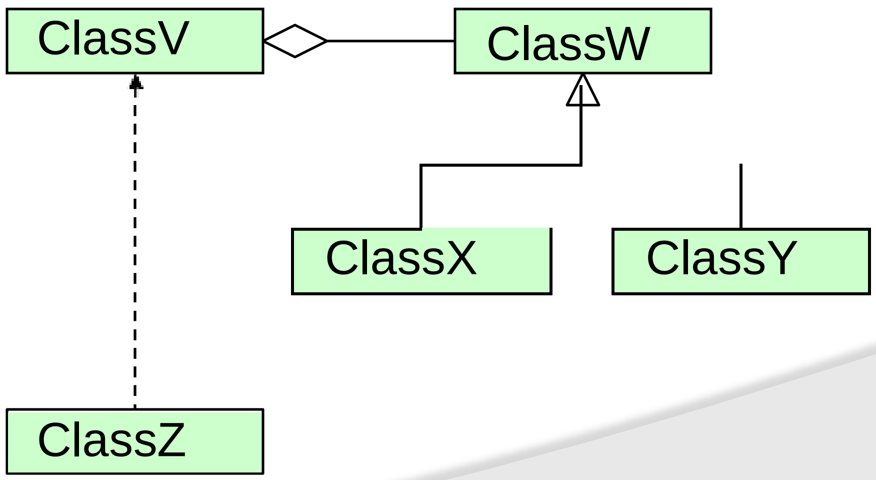
Data Flow Styles



Call & Return Styl







S/W Arct. Styles

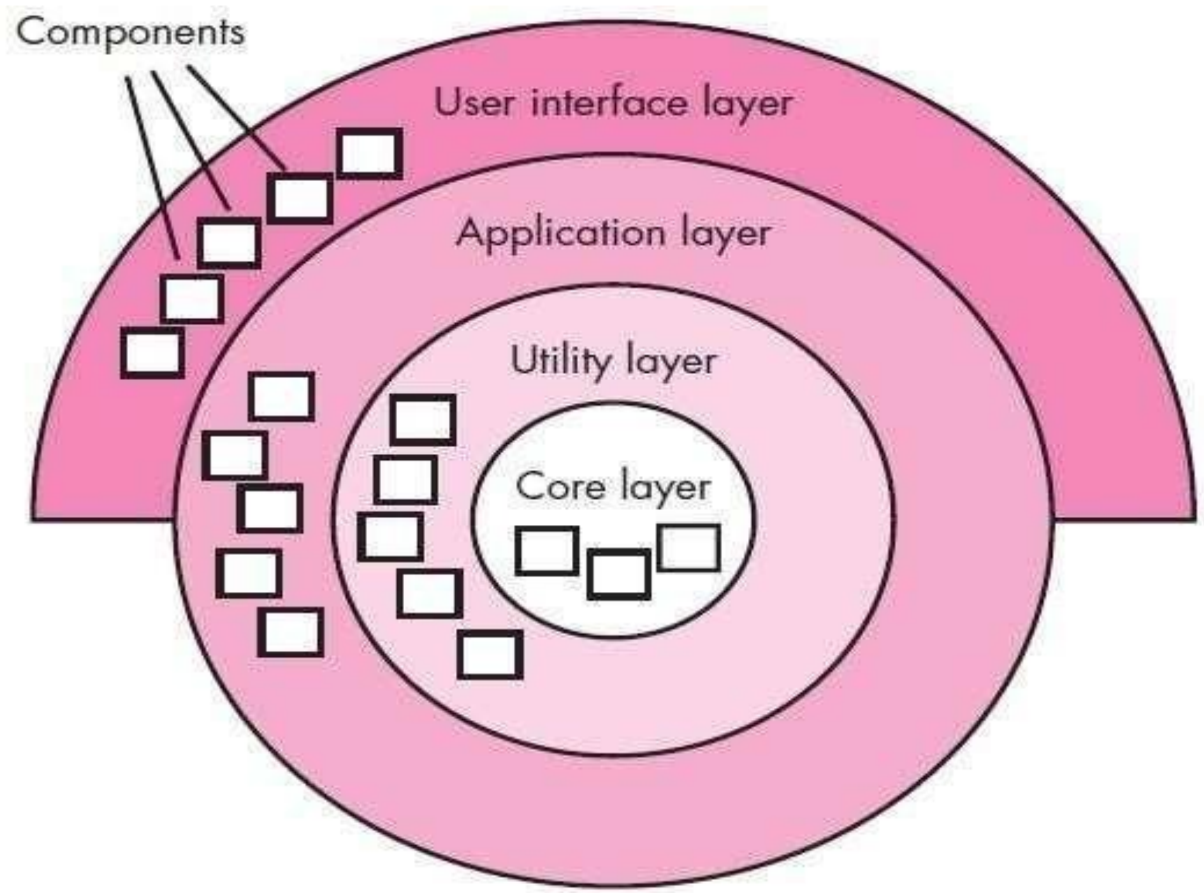
Object-oriented architectures

- The components of a system encapsulate data and the operations that must be applied to manipulate the data.
- Communication and coordination between components are accomplished via message passing.

Layered architectures.

- The basic structure of a layered architecture is illustrated in Figure.
- A number of different layers are defined, each accomplishing operations that progressively become closer to the machine instruction set.
- At the outer layer, components service user interface operations.
- At the inner layer, components perform operating system interfacing.
- Intermediate layers provide utility services and application S/W functions.

Layered Architecture



Architectural Patterns

- As the requirements model is developed, you'll notice that the software must address a number of broad problems that span the entire application.
- For example, the requirements model for virtually every e-commerce application is faced with the following problem: *How do we offer a broad array of goods to a broad array of customers and allow those customers to purchase our goods online?*
- Architectural patterns address an application-specific problem within a specific context and under a set of limitations and constraints. The pattern proposes an architectural solution that can serve as the basis for architectural design.

S/W Architecture

Organization and Refinement

- Because the design process often leaves you with a number of architectural alternatives, it is important to establish a set of design criteria that can be used to assess an architectural design that is derived.

Control.

- How is control managed within the architecture? Does a distinct control hierarchy exist, and if so, what is the role of components within this control hierarchy? How do components transfer control within the system? How is control shared among components? What is the control topology (i.e., the geometric form that the control takes)? Is control synchronized or do components operate asynchronously?

Architectural Design Steps

- As architectural design begins, the software to be developed must be put into context—that is, the design should define the external entities (other systems, devices, people) that the software interacts with and the nature of the interaction.
    1. Represent the system in context
    2. Define archetypes
    3. Refine the architecture into components
    4. Describe instantiations of thesystem

Architectural Design Steps

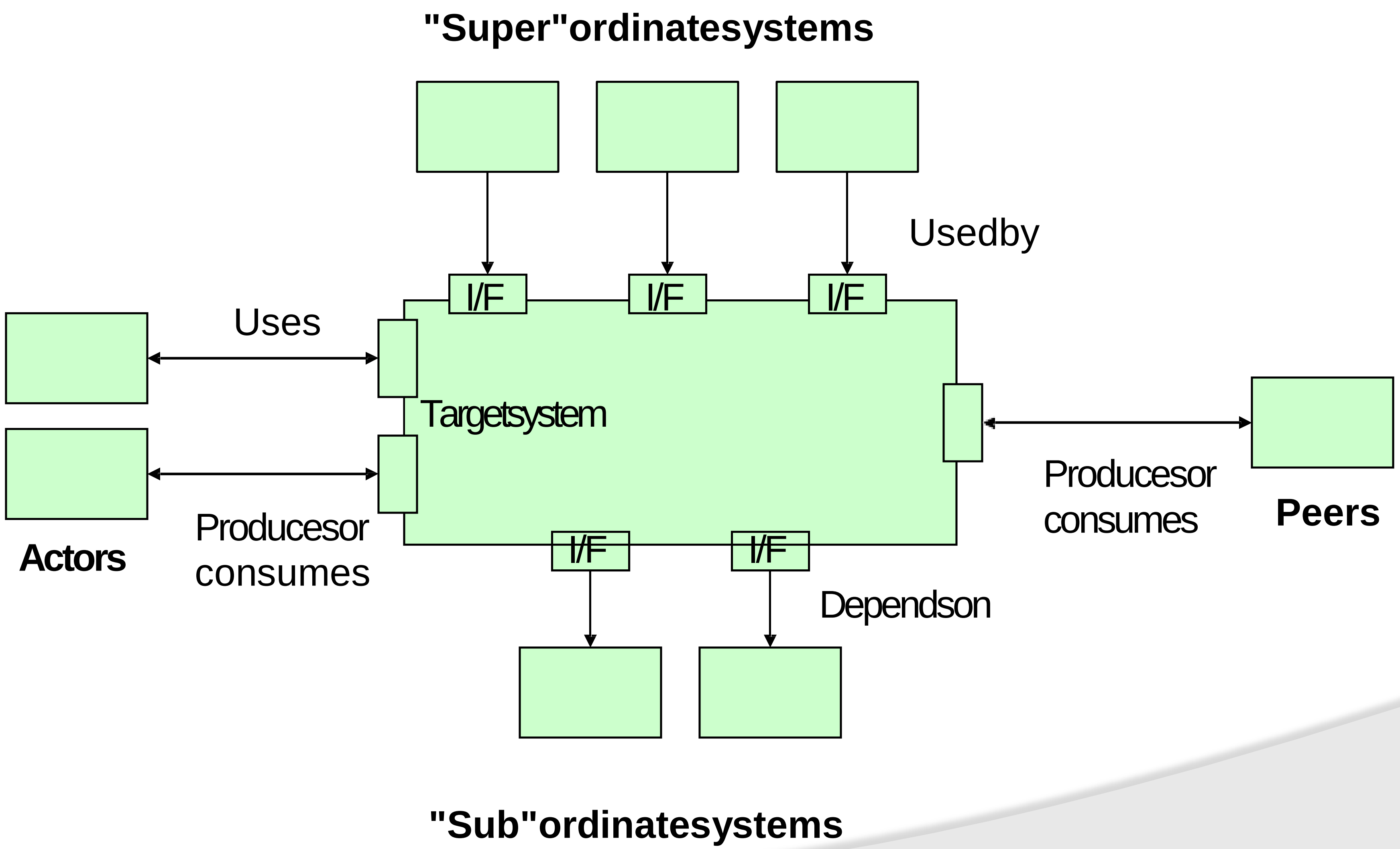
Represent the System in Context

•Use an architectural context diagram (ACD) that shows

- The identification and flow of all information into and out of a system
- The specification of all interfaces
- Any relevant support processing from/by other systems

An ACD models the manner in which software interacts with entities to its boundaries.

Architectural Design Steps



- An ACD identifies systems that interoperate with the target system
  - Super-ordinate systems
    - Use target system as part of some higher level processing scheme
  - Sub-ordinate systems
    - Those systems that are used by the target system and provide data or processing that are necessary to complete target system functionality
  - Peer-level systems
    - Interact on a peer-to-peer basis with target system to produced or consumed by peers and target system
  - Actors
    - People or devices that interact with target system to produce or consume data

Architectural Design Steps

Define Archetypes

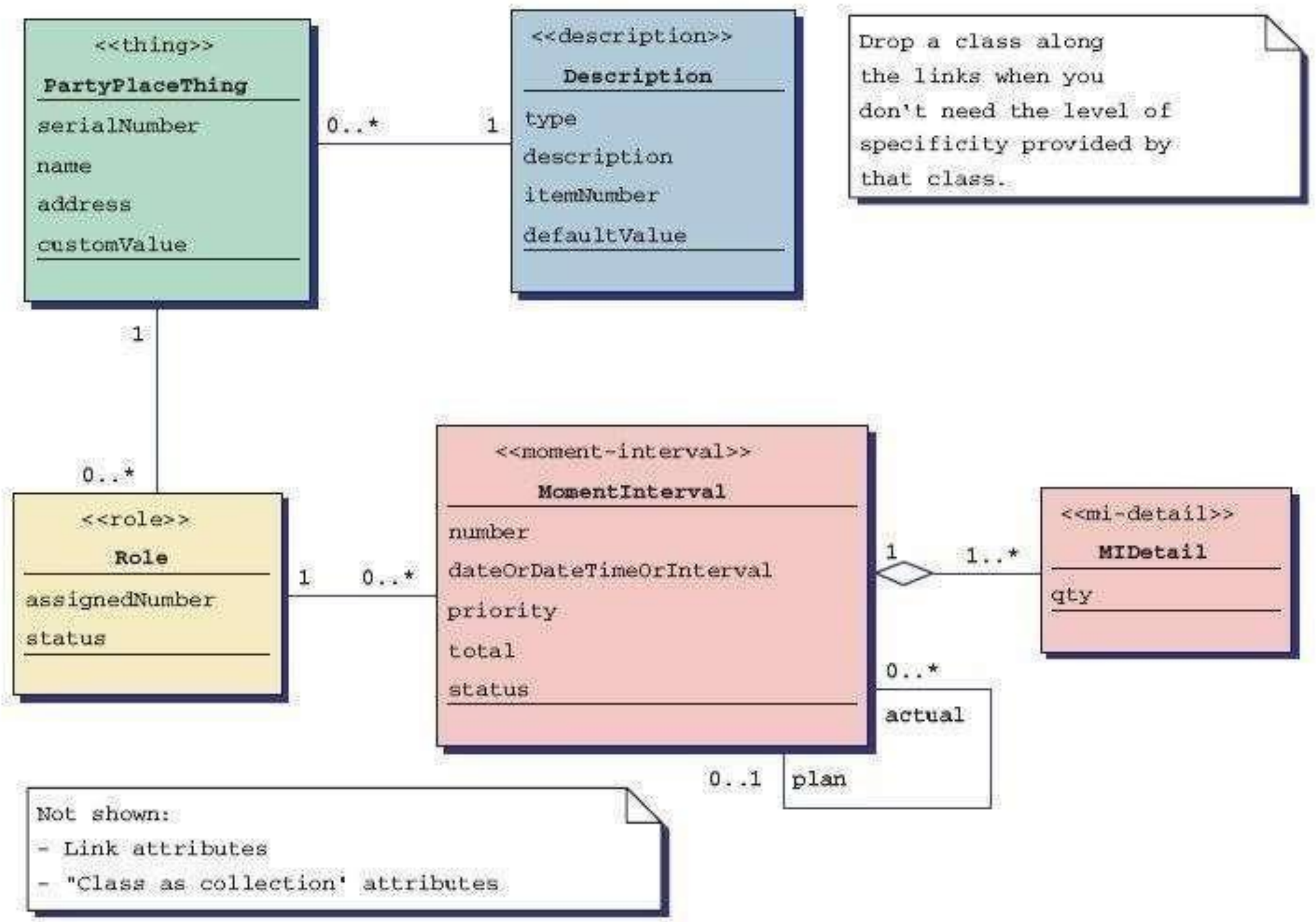
- Archetypes indicate the important abstractions within the problem domain (i.e., they model information).
- An archetype is a class or pattern that represents a core abstraction that is critical to the design of an architecture for the target system.
- Only a relatively small set of archetypes is required in order to design even relatively complex systems.
- The target system architecture is composed of these archetypes.
  - They represent stable elements of the architecture.
  - They may be instantiated in different ways based on the behavior of the system.

- **Archetypes in SoftwareArchitecture**
- **Node**

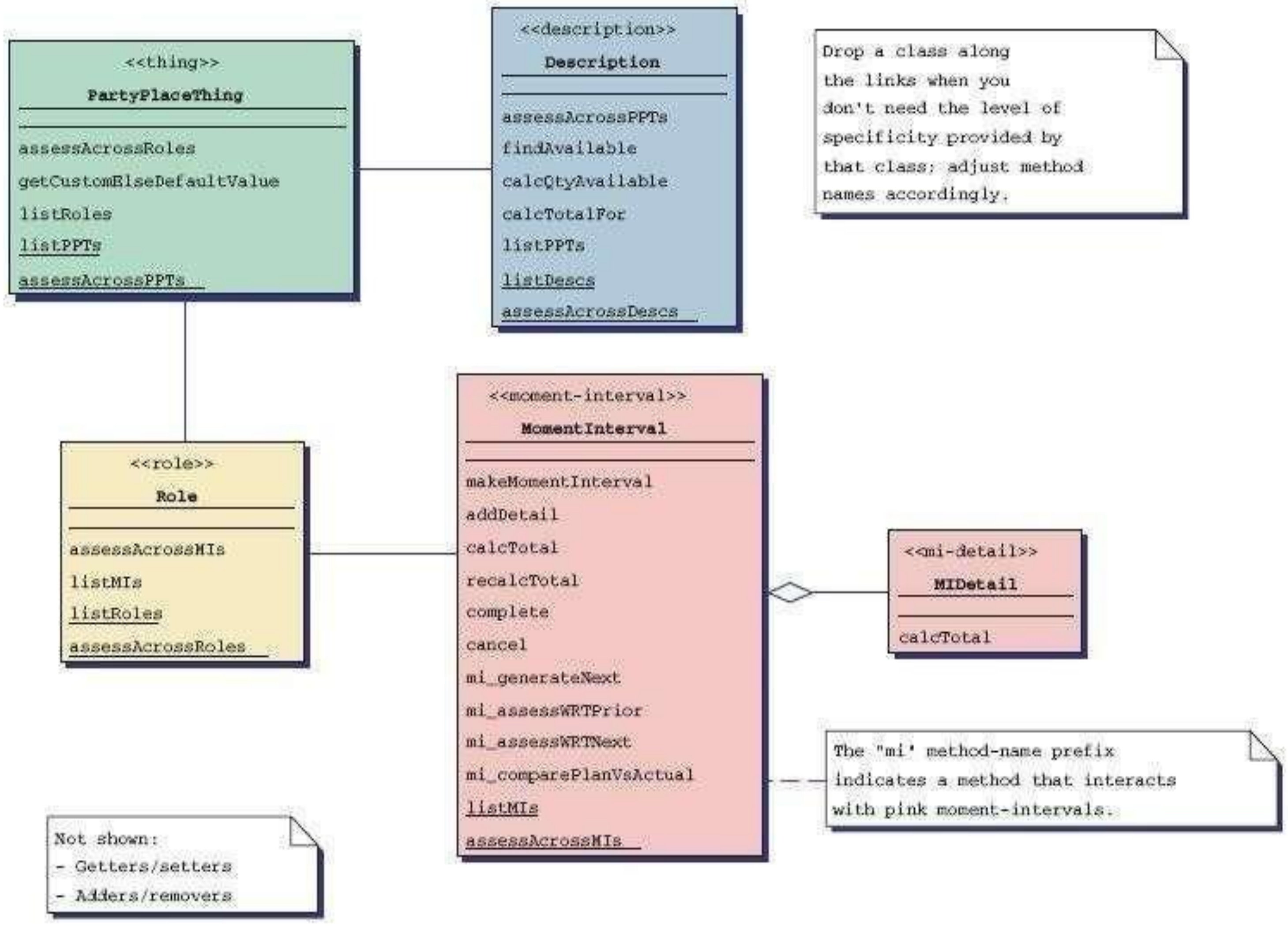
  - Represents a cohesive collection of input and output elements of the home security function

- **Detector/Sensor -** An abstraction that encompasses all sensingequipment that feeds information into the target system.
- **Indicator -** An abstraction that represents all mechanisms (e.g., alarm siren, flashing lights, bell) for indicating that an alarm condition is occurring.
- **Controller -** An abstraction that depicts the mechanism that allows the arming or disarming of a node. If controllers reside on a network, they have the ability to communicate with one another.

Arct. Types - Attributes

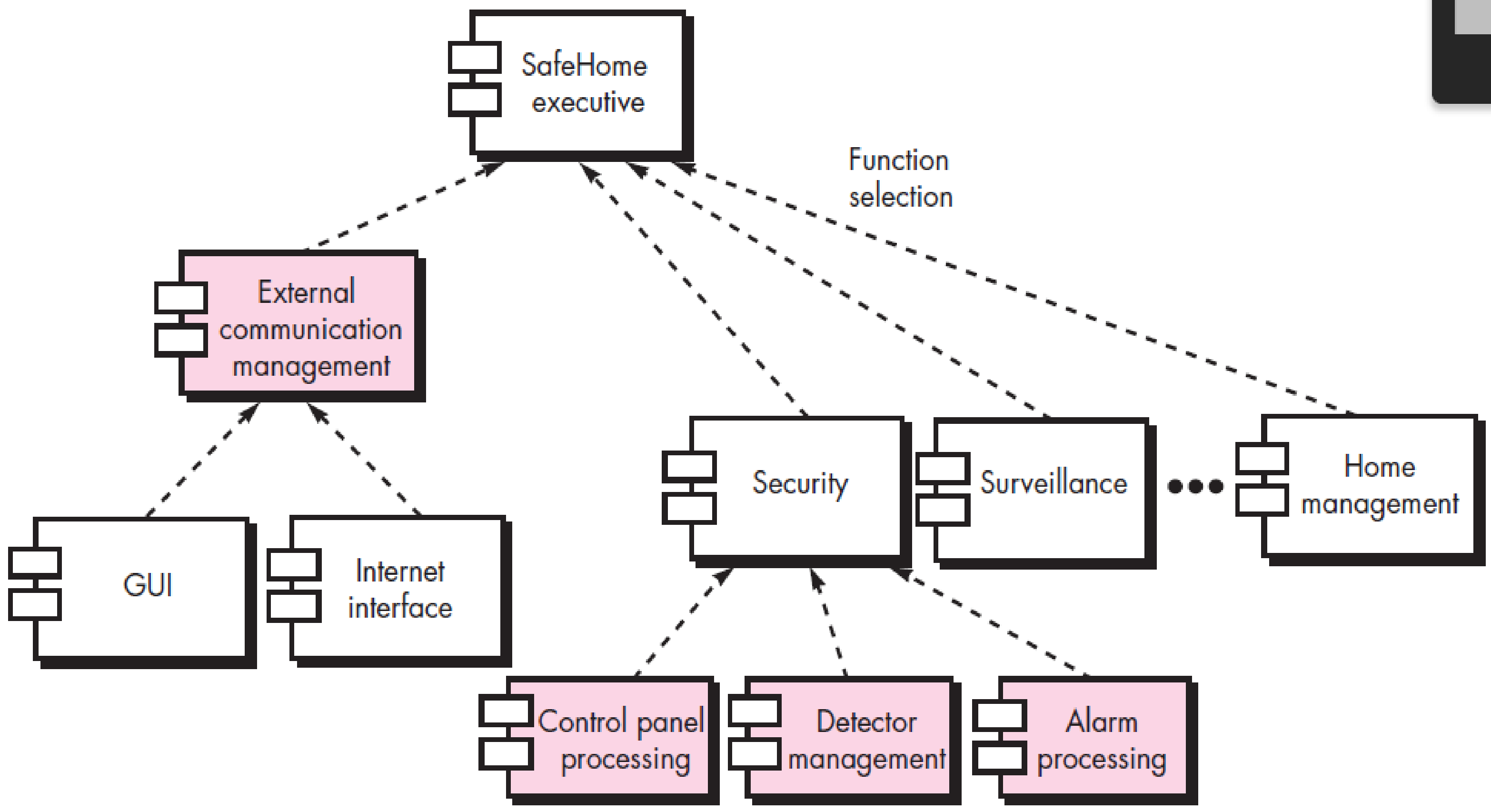


Arct. Types - Methods



- AMUL class diagram can represent the classes of the refined architecture and their relationships

FIGURE 9.8 Overall architectural structure for *SafeHome* with top-level components



Architectural Design Steps

Describe Instantiations of the System

- The architectural design that has been modeled to this point is still relatively high level.
- The context of the system has been represented, archetypes that indicate the important abstractions within the problem domain have been defined, the overall structure of the system is apparent, and the major softwarecomponents have been identified.
- However, further refinement (recall that all design is iterative) is still necessary.

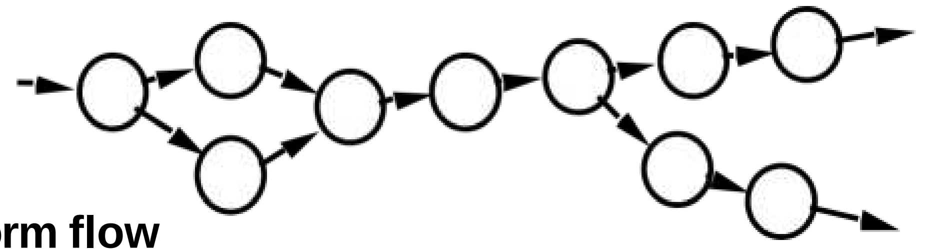
Architectural Mapping using Data

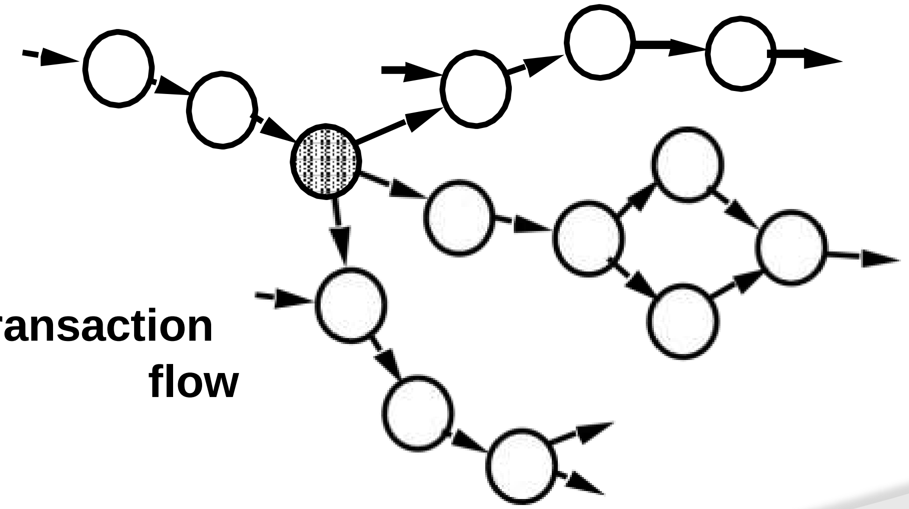
- **Transform mapping is a set of design steps that allows a DFD with transform flow characteristics to be mapped into a specific architectural style.**
  - Information must enter and exit software in an external world. The externalized data must be converted into an internal form for processing. Information enters along paths that transform external data into an internal form. These paths are identified are *Incomingflow*.
  - Incoming data are transformed through a transform center and move along the paths that now lead out of the software. Data moving along thesepaths are called *Outgoing flow.*

- **Transaction Flow**
  - Information flow Is often characterized by a Singledata item *called Transaction*, that triggers other data flow along one of many paths.
  - Transaction flow is characterized by data moving along an incoming path that converts external world information into a transaction
  - The transaction is evaluated and, based on its value, flow along one of many action paths is initiated. The hub of information from which many action paths emanate is called a *transaction center*

Architectural Mapping using Data Flow

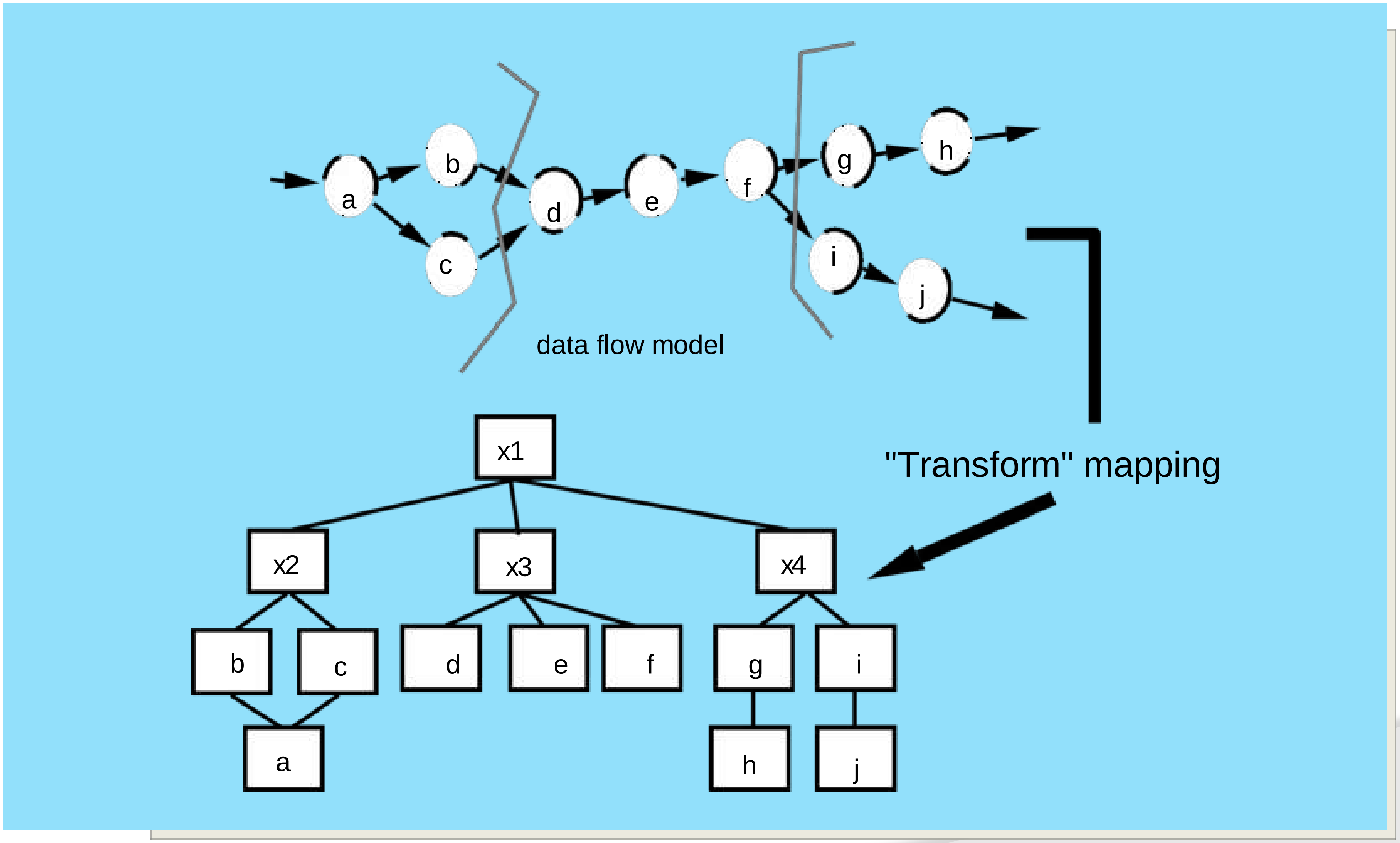
Flow characteristics





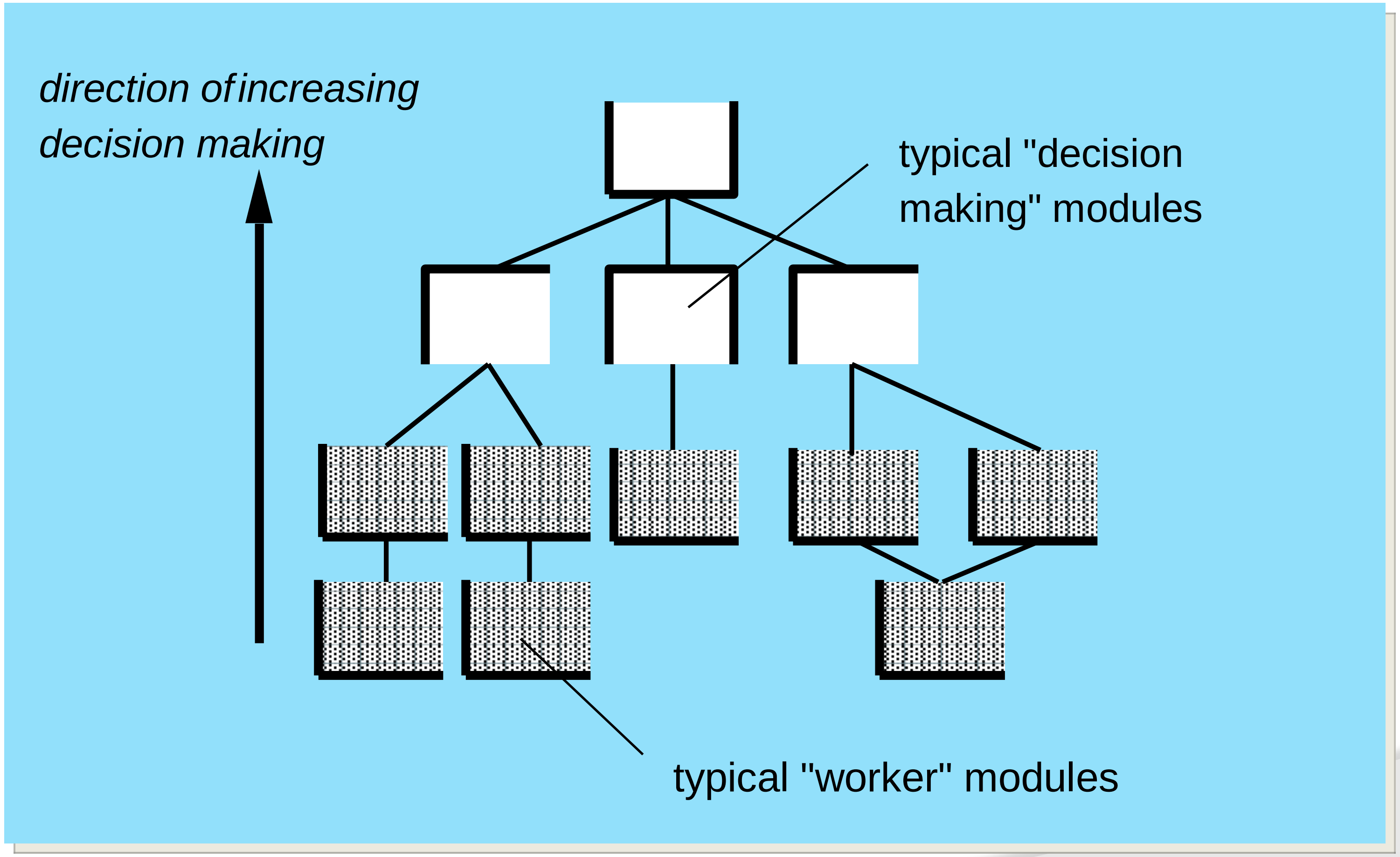
Architectural Mapping using Data Flow

Transform Mapping



Architectural Mapping using Data

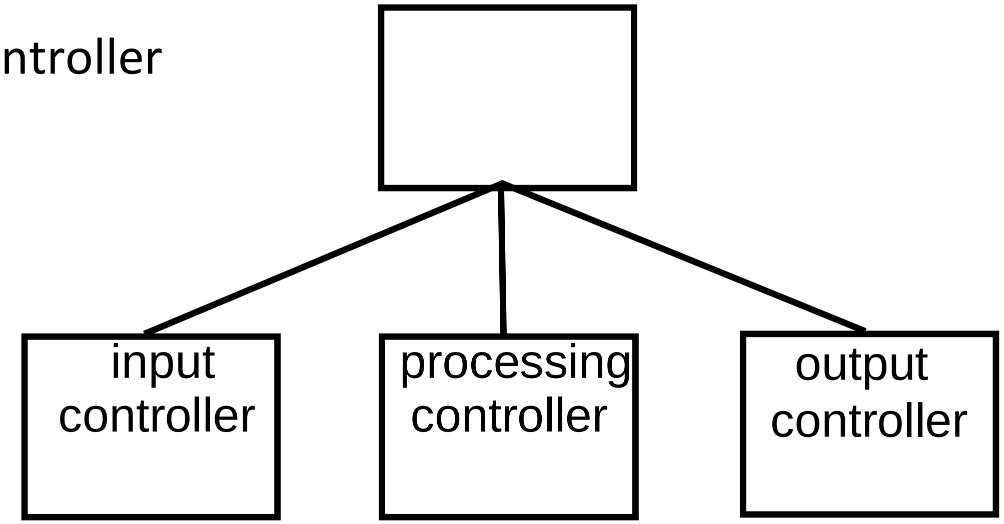
Factoring


direction of increasing decision making
typical "decision making" modules
typical "worker" modules

Architectural Mapping using Data Flow

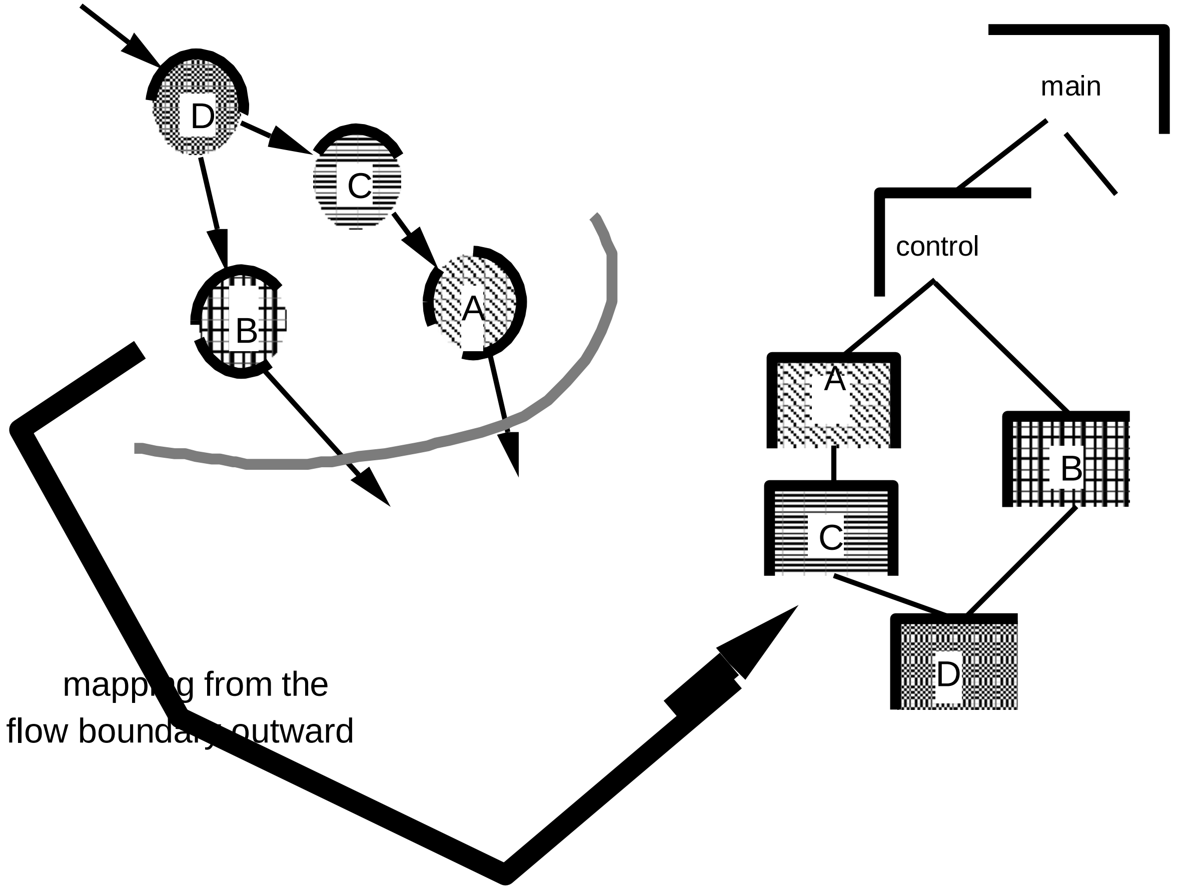
First Level Factoring

main programcontroller



Architectural Mapping using Data

SecondLevelFactoring

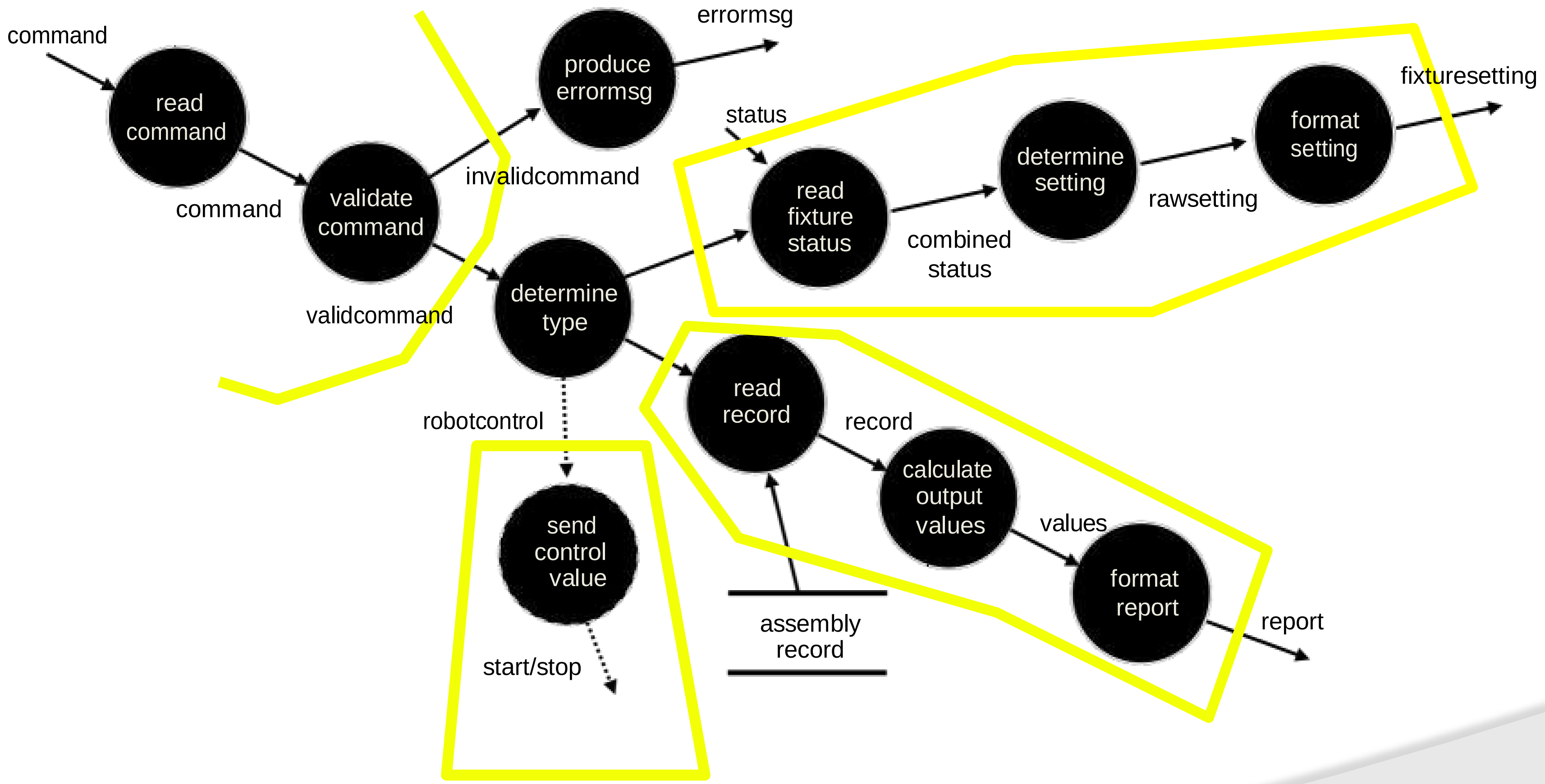


Architectural Mapping using Data

- **Transaction Mapping**
  1. Review the fundamental system model.
  2. Review and refine data flow diagrams for the software
  3. Determine whether the DFD has transform or transaction flow characteristics.
  4. Isolate the transaction center and the flow characteristics along each of the action paths.
  5. Map the DFD in a program structure amenable to transaction processing.
  6. Factor and refine the transaction structure and the structure of each action path.
  7. Refine the first-iteration architecture using design heuristics for improved software quality.

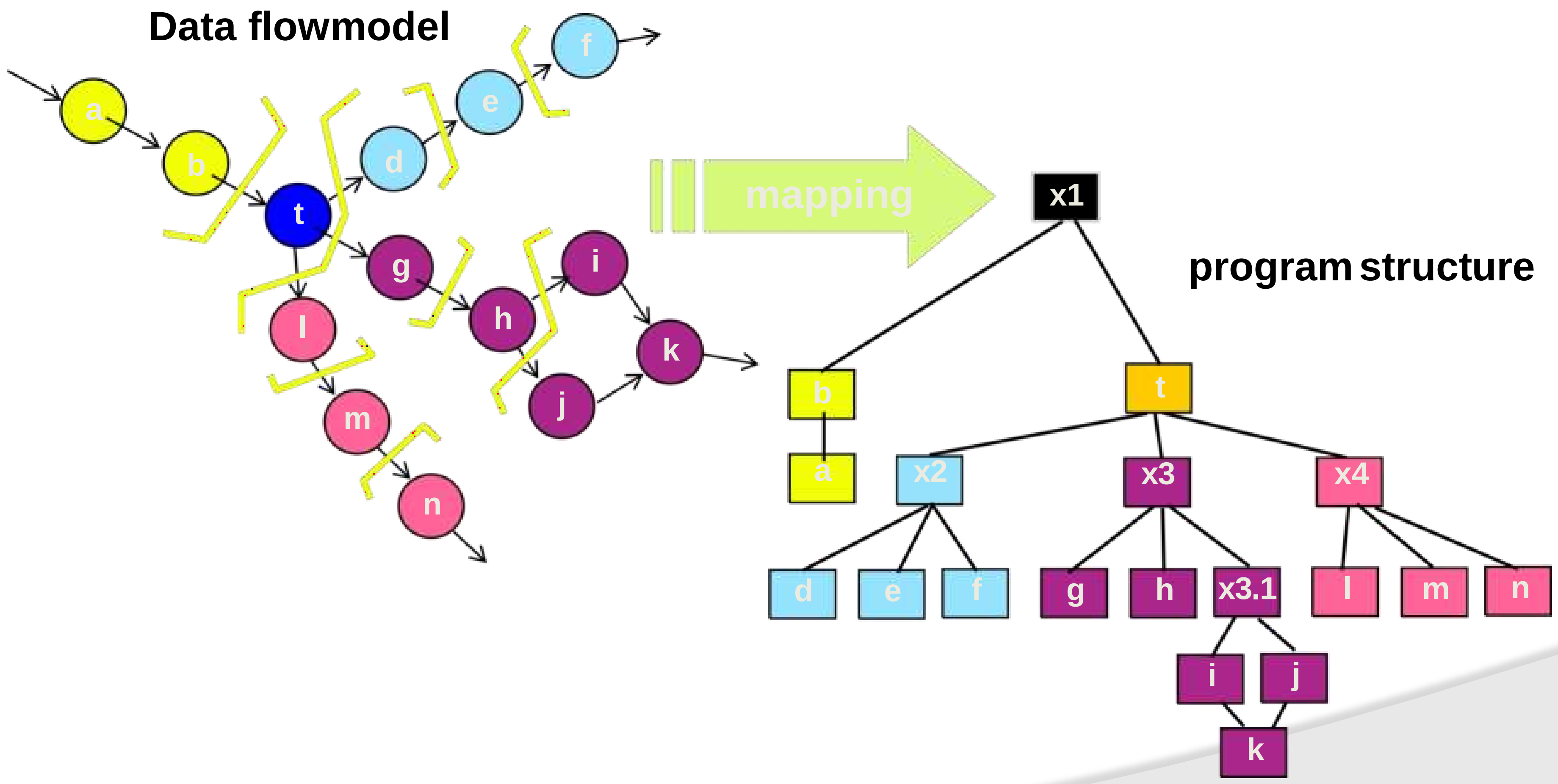
Architectural Mapping using Data Flow

Isolate Flow Paths



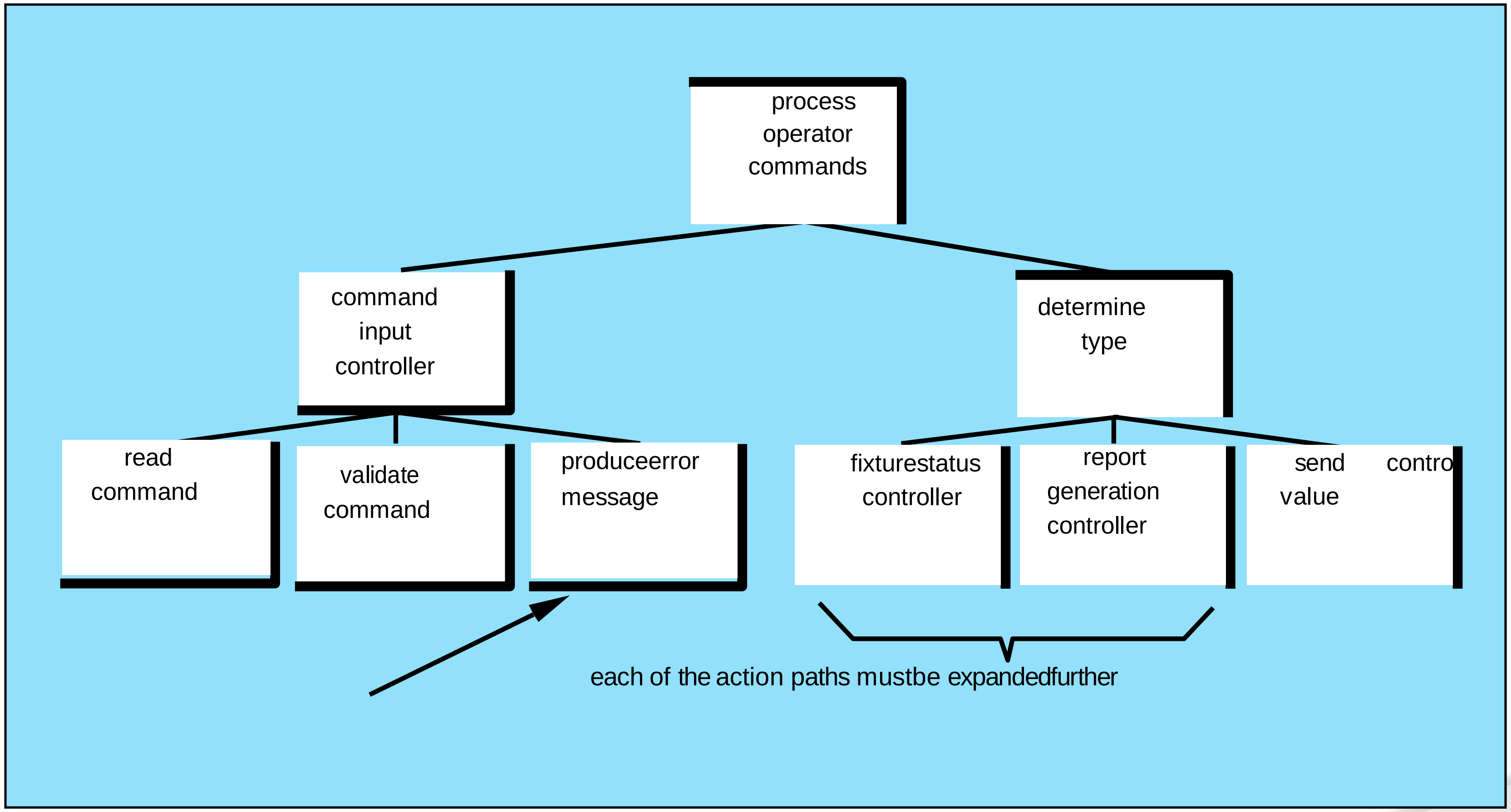
Architectural Mapping using Data Flow

Transaction Mapping



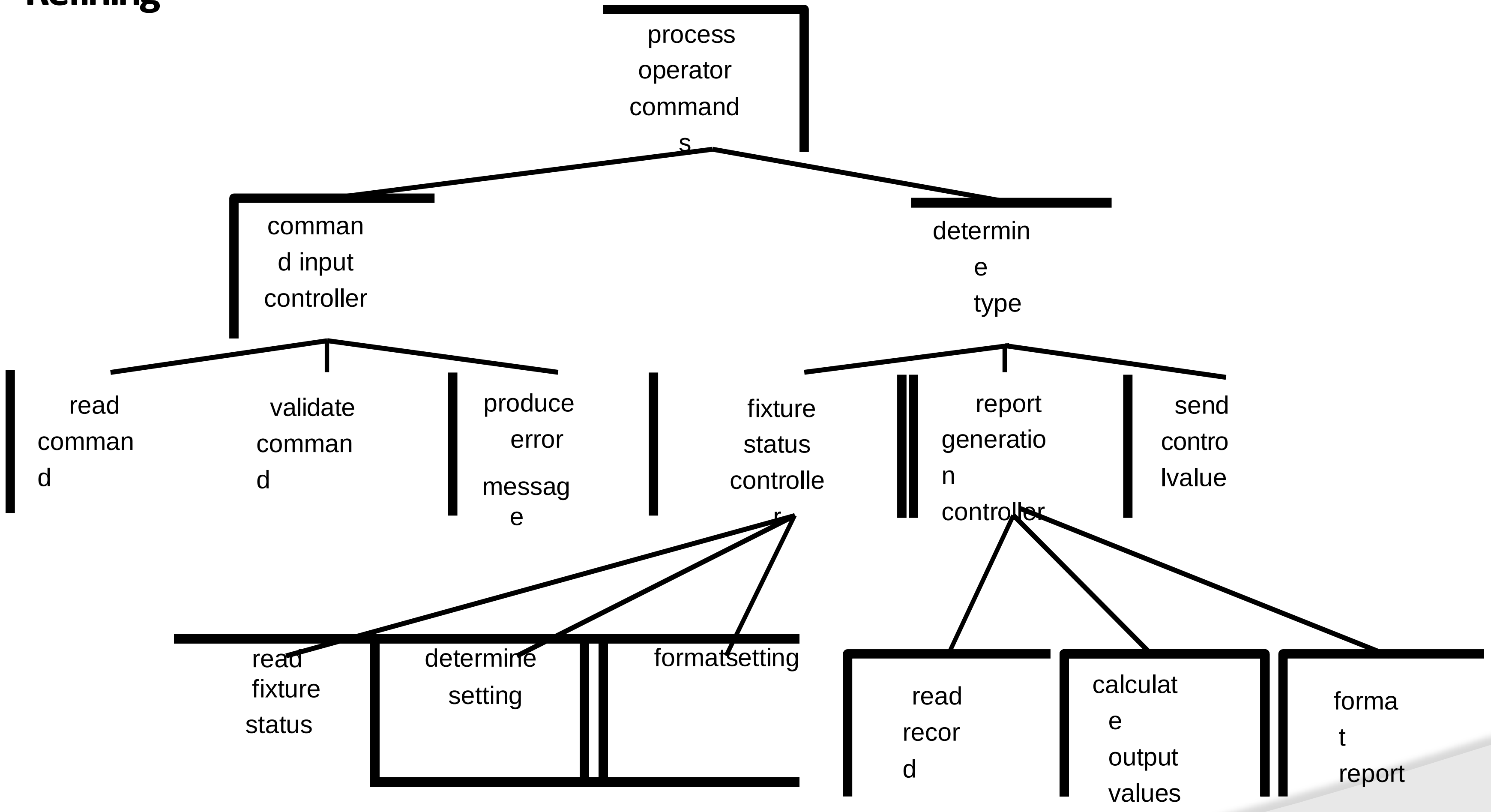
Architectural Mapping using Data Flow

Map theFlowModel



Architectural Mapping using Data Flow

Refining



Architectural Mapping Using Data Flow

- Refining the Architectural Design
- Any discussion of design refinement should be prefaced with the following comment:
- You should be concerned with developing a representation of software that will meet all functional and performance requirements and merit acceptance based on design measures and heuristics.
- Refinement of software architecture during early stages of design is to be encouraged.

UNIT–IV
TESTING STRATEGIES AND PRODUCT METRICS

Contents

Software testing fundamentals: Internal and external views of testing, white box testing, basis path testing, control structure testing, black box testing, regression testing, unit testing, integration testing, validation testing, system testing and debugging;

Software implementation techniques: Coding Practices, refactoring.

System Testing

- System testing is a series of different test whose primary purpose is to fully exercise the computer-based system.
- Each test has a different purpose, all work to verify that system elements have been properly integrated and perform allocated functions.

1. Recovery testing
   - Tests for recovery from system faults
   - Forces the software to fail in a variety of ways and verifies that recovery is properly performed
   - Tests reinitialization, checkpointing mechanisms, data recovery, and restart for correctness
   - If recovery is automatic, reinitialization, checkpointing mechanisms, data recovery, and restart are evaluated for correctness
   - If recovery requires human intervention, the mean-time-to-repair is evaluated to determine whether it is within acceptable limits

2. Security testing

- Verifies that protection mechanisms built into a system will, in fact, protect it from improper access
- During security testing, the tester plays the role of the individual who desires to penetrate the system.
- Anything goes! The tester may attempt to acquire passwords through external clerical means.
- may attack the system with custom software designed to break down any defenses that have been constructed

3. Stress testing

- Executes a system in a manner that demands resources in abnormal quantity, frequency, or volume.
- Stress tests are designed to confront programs with abnormal situations. In essence, the tester who performs stress testing asks: How high can we crank this up before it fails?
- A variation of stress testing is a technique called sensitivitytesting.

- A very small range of data contained within the bounds of valid data for a program may cause extreme and even erroneous processing or profound performance degradation.
- Sensitivity testing attempts to uncover data combinations within valid input classes that may cause instability or improper processing.

4. Performance Testing

   - Performance testing is designed to test the run-time performance of software within the context of an integrated system.
   - Performance testing occurs throughout all steps in the testing process. Even at the unit level, the performance of an individual module may be assessed as tests are conducted
   - Performance tests are often coupled with stress testing and usually requires both hardware and software instrumentation.
   - That is, it is often necessary to measure resource utilization (e.g., processor cycles) in an exacting fashion

The Art of Debugging

- Debugging occurs as a consequence of successful testing. When a test case uncovers an error, debugging is an action that results in the removal of the error.

1. The debugging process

   The debugging process attempts to match symptom with cause, thereby leading to error correction.

   The debugging process will usually have one of two outcomes:

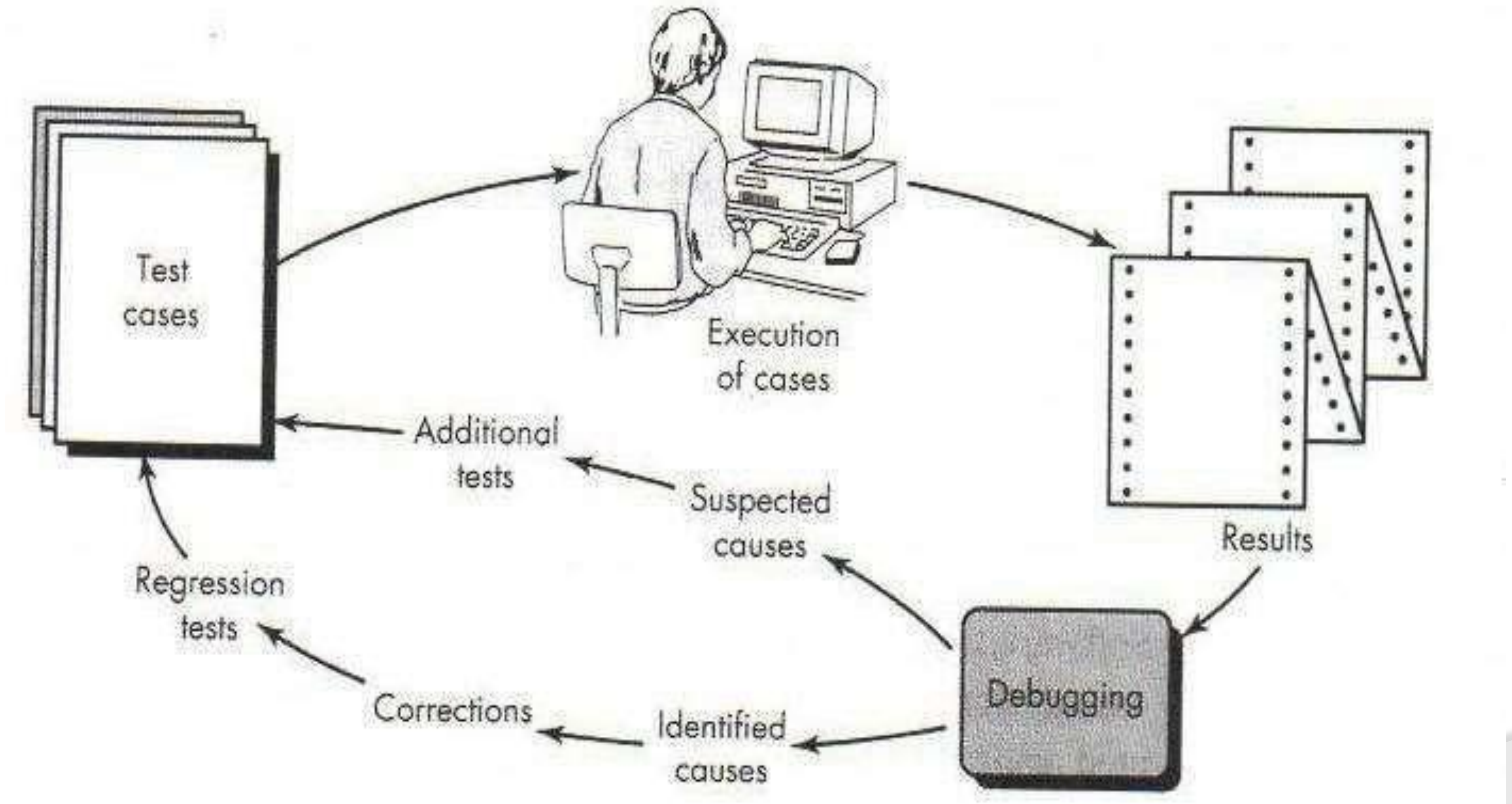
   (1) The cause will be found and corrected or

   (2) The cause will not be found. In the latter case, the person performing debugging may suspect a cause, design a test case to help validate that suspicion, and work toward error correction in an iterative fashion.

   – Debugging process beings with the execution of a test case. Results are assessed and a lack of correspondence between expected and actual performance is observed

   – Debugging attempts to match symptom with cause, thereby leading to error correction .

The Art of Debugging



- Characteristics of bugs
  - The symptom and the cause may be geographically remote.
  - The symptom may disappear (temporarily) when another error is corrected
  - The symptom may actually be caused by non-errors
  - The symptom may be caused by human error that is not easily traced
  - The symptom my be a result of timing problems, rather than processing problems
  - It may be difficult to accurately reproduce input conditions
  - The symptom may be intermittent.
  - The symptom may be due to causes that are distributed across a number of tasks running on different processors

2. Psychological Considerations

- Unfortunately, there appears to be some evidence that debugging prowess is an innate human trait. Some people are good at it and others are not.
- Although experimental evidence on debugging is open to many interpretations, large variances in debugging ability have been reported for programmers with the same education and experience.

3. Debugging Strategies

- Objective of debugging is to find and correct the cause of a software error or defect.
- Bugs are found by a combination of systematic evaluation, intuition, and luck.
- Debugging methods and tools are not a substitute for careful evaluation based on a complete design model and clear source code
- There are three main debugging strategies

1. Brute force 2. Backtracking 3. Cause elimination

- BruteForce
  - Most commonly used and least efficient method for isolating the causeof a software error
  - Used when all else fails
  - Involves the use of memory dumps, run-time traces, and output statements
  - Leadsmanytimestowastedeffortandtime
- Backtracking
  - Can be used successfully in small programs
  - The method starts at the location where a symptom has been uncovered
  - The source code is then traced backward (manually) until the locationof the cause is found
  - In large programs, the number of potential backward paths may become unmanageably large

4. Correcting the Error

- Once a bug has been found, it must becorrected.
- But the correction of a bug can introduce other errors and therefore do more harmthan good.
- Van Vleck suggests three simple questions that you should ask before making the correction that removes the cause of a bug.

Three Questions to ask Before Correcting the Error

- – Is the cause of the bug reproduced in another part of the program?

- Similar errors may be occurring in other parts of the program

- – What next bug might be introduced by the fix that I'ŵ about to make?
    - The source code (and even the design) should be studied to assess the coupling of logic and data structures related to the fix
- – What could we have done to prevent this bug in the first place?
    - This is the first step toward software quality assurance
    - By correcting the process as well as the product, the bug will be removed from the current program and may be eliminated from all future programs

Software implementation Techniques

Refactoring:

- Refactoring is usually motivated by noticing a codesmell.
- For example the method at hand may be very long, or it may be a near duplicate of another nearby method.
- Once recognized, such problems can be addressed by *refactoring* the source code, or transforming it into a new form that behaves the same as before but that no longer"smells".

There are two general categories of benefits to the activity ofrefactoring.

Maintainability. It is easier to fix bugs because the source code is easy to read and the intent of its author is easy to grasp. This might be achieved by reducing large monolithic routines into a set of individually concise, well-named, single-purpose methods. It might be achieved by moving a method to a more appropriate class, or by removing misleading comments.

Extensibility. It is easier to extend the capabilities of the application if it uses recognizable
design patterns, and it provides some flexibility where none before may have existed.

- Before applying a refactoring to a section of code, a solid set of automatic unit tests is needed. The tests are used to demonstrate that the behavior of the module is correct before the refactoring.
- The tests can never prove that there are no bugs, but the important point is that this process can be cost-effective: good unit tests can catch enough errors to make them worthwhile and to make refactoring safe enough.

Project Management

- Organising, planning and scheduling software projects
- Objectives
  - To introduce software project management and to describe its distinctive characteristics
  - To discuss project planning and the planning process
  - To show how graphical schedule representations are used by project management
  - To discuss the notion of risks and the risk management process

UNIT–V
RISK MANAGEMENT AND QUALITY MANAGEMENT

contents

Estimation: FP based, LOC based, make/buy decision COCOMO II: Planning, project plan, planning process, RFP risk management, identification, projection;

RMMM: Scheduling and tracking, relationship between people and effort, task set and network, scheduling

EVA: Process and project metrics.

Software Project

- Concerned with activities involved in ensuring that software is delivered
  - on time
  - within the budget
  - in accordance with the requirements
- Project management is needed because software development is always subject to budget and schedule constraints
  - Set by the development organisation or the customer

Software Management

- The product is intangible
- The product is uniquely flexible
- The product is uniquely complex
- Software engineering is not recognized as an engineering discipline with the same status as mechanical, electrical engineering, etc.
- The software development process is not standardised
- Many software projects are one-off projects

Management Activities

- Proposal writing
- Project planning andscheduling
- Projectcosting
- Project monitoring andreviews
- Personnel selection andevaluation
- Report writing andpresentations

Project Staffing

- May not be possible to appoint the ideal people to work on a project
  - Project budget may not allow for the use of highly-paid staff
  - Staff with the appropriate experience may not be available

An organization may wish to develop on a software project

Employee skills

- Here’s Bob. He’s a sophomore. He’ll be a member of your HazMat Rover team. He doesn’t know much yet, but he can brew a mean cup of coffee and has a great personality.
- Managers have to work within these constraints

Project Planning

- Probably the most time-consuming project management activity
- Continuous activity from initial concept through to system delivery
- Plans must be regularly revised as new information become available

–Beware of grumbling developers

- Various different types of plan may be developed to support the main Software project plan that is concerned with schedule and budget.

Types of Project Plan

Plan	Description
Quality plan	Describes the quality procedures and standards that will be used in a project
Validation plan	Describes the approach, resources and schedule used for system validation.
Configuration management plan	Describes the configuration management procedures and structures to be used.
Maintenance plan	Predicts the maintenance requirements of the system, maintenance costs and effort required.
Staff development plan	Describeshow the skill and experience of the project team members will be developed.

Estimation

- Software cost and effort estimation will never be an exact science. Too many variables—human, technical, environmental, political—can affect the ultimate cost of software and effort applied to develop it.
- **Toachieve reliable cost and effort estimates, a number of optionsarise:**
    1. Delay estimation until late in the project (obviously, we can achieve 100 percent accurate estimates after the project iscomplete!).
    2. Base estimates on similar projects that have already been completed.
    3. Use relatively simple decomposition techniques to generateproject cost and effort estimates.
    4. Use one or more empirical models for software cost and effortestimation.
- Unfortunately, the first option, however attractive, is not practical. Cost estimates must be provided up-front. However, recognize that the longer you wait, the more you know, and the more you know, the less likely you are to make serious errors in your estimates.
- The second option can work reasonably well, if the current project is quite similar to past efforts and other project influences (e.g., the customer, business conditions, the software engineering environment, deadlines) are roughly equivalent. Unfortunately, past experience has not always been a good indicator of future results.

Function point Based Estimation

Function Point based Estimation :

- A **Function Point** (FP) is a unit of measurement to express the amount of business functionality, an information system (as a product) provides to a user. FPs measure software size. They are widely accepted as an industry standard for functional sizing. Function point analysis is a method of quantifying the size and complexity of a software
- system in terms of the functions that the system delivers to the user
- It is Independent of the computer language, development methodology, technology or capability of the project team used to develop the application
- Function point analysis is designed to measure business applications (not scientific
- applications)
- Scientific applications generally deal with complex algorithms that the function point method is not designed to handle
- Function points are independent of the language, tools, or methodologies used for implementation (ex. Do not take into consideration programming languages, DBMS, or processing hardware)Function points can be estimated early in analysis and design

LOC Base Estimation

LOC based estimation

- **Source lines of code** (**SLOC**), also known as **lines of code** (**LOC**), is a software metric used to measure the size of a computer program by counting the number of lines in the text of the program's source code.
- SLOC is typically used to predict the amount of effort that will be required to develop a program, as well as to estimate programming productivity or maintainability once the software is produced.
- Lines used for commenting the code and header file areignored.

Two major types of LOC:

1. Physical LOC

- Physical LOC is the count of lines in the text of the program's source code including comment lines.
- Blank lines are also included unless the lines of code in a section consists of more than 25% blank lines.

2. Logical LOC

- Logical LOC attempts to measure the number of executable statements, but their specific definitions are tied to specific computer languages.
- Ex: Logical LOC measure for C-like programming languages is the number of statement-terminating semicolons(;)

The problems of lines of code (LOC)

– Different languages lead to different lengths ofcode

– It is not clear how to count lines of code

– Areport, screen, or GUI generator can generate thousands of lines of code in minutes

– Depending on the application, the complexity of code is different.

Make / Buy Decision

- In many software application areas, it is often more cost effective to acquire rather than develop computer software.
- Software engineering managers are faced with a make/ buy decision that canbe further complicated by a number of acquisitionoptions.
  (1) Software may be purchased (or licensed) off-the-shelf
  (2) full-experience or partial experience software components may be acquired and then modified and integrated to meet specific needs.
  (3) Software may be custom built by an outside contractor to meet the purchaser’s specifications.
- In the final analysis the make/buy decision is made based on thefollowing conditions:
  (1) Will the delivery date of the software product be sooner than that for internally developed software?
  (2) Will the cost of acquisition plus the cost of customization be less than the cost of developing the software internally?
  (3) Will the cost of outside support (e.g., a maintenance contract) be less than the cost of internal support?

Creating a Decision Tree:

- The steps just described can be augmented using statistical techniques such as decision treeanalysis.
- For example, considered the figure below it depicts a decision tree for a software based system X. In this case, the software engineering organization can

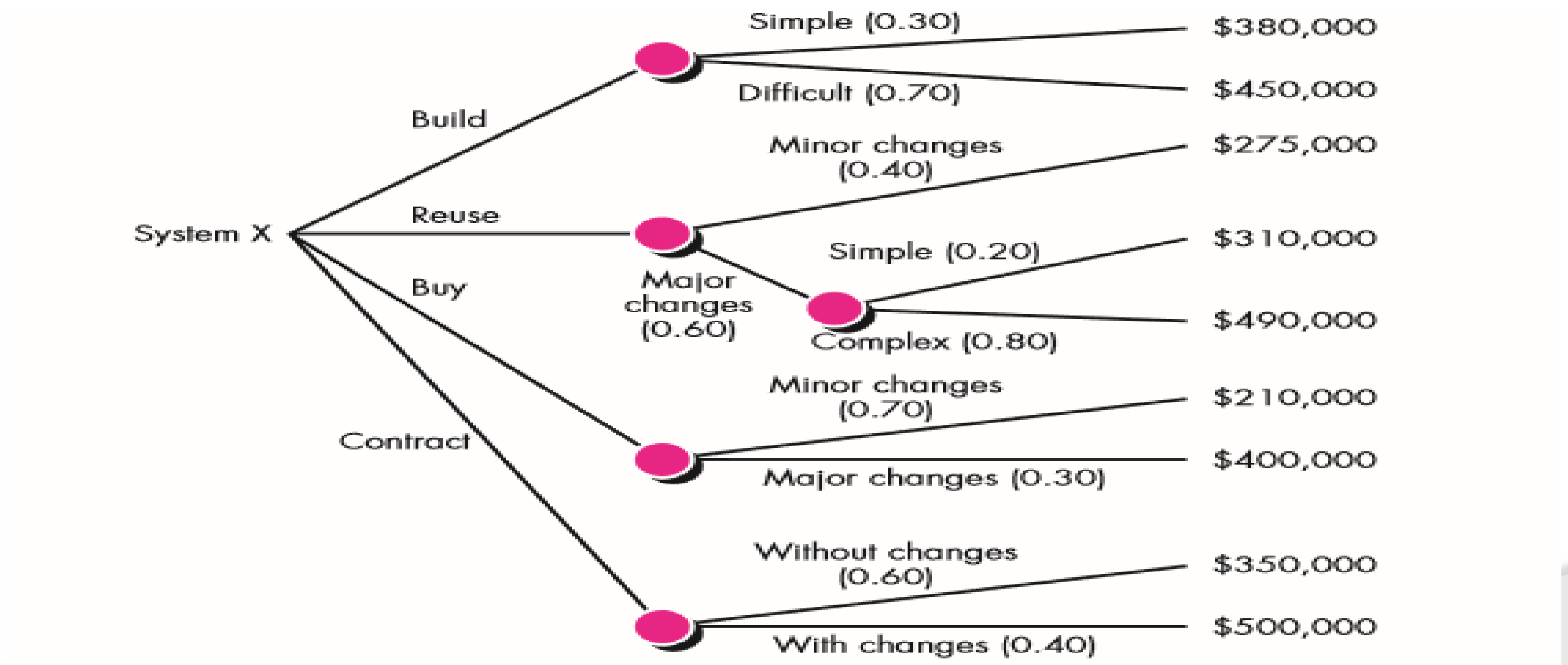
  (1) build system X from scratch

  (2) reuse existing partial-experience components to construct the system

  (3) buy an available software product and modify it to meet local needs, or
  (4) contract the software development to an outsidevendor.

If the system is to be built from scratch, there is a 70 percent probability that the job will be difficult.
The expected value for cost, computed along any branch of the decision tree,is:
where i is the decision tree path. For the build path.

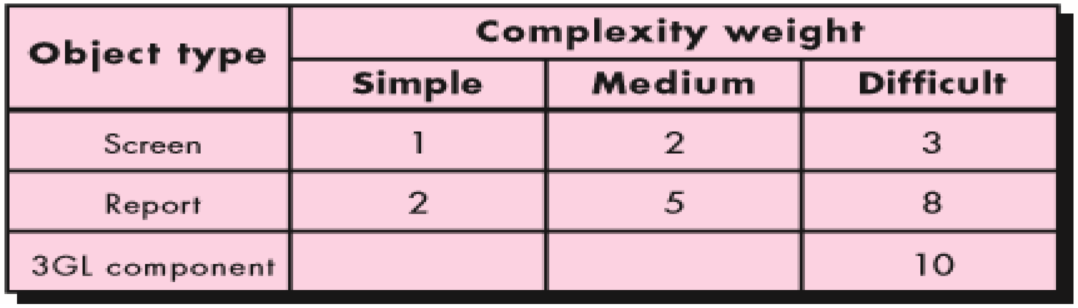
- It is important to note, however, that many criteria —not just cost—must be considered during the decision-making process. Availability, experience of the developer/ vendor/contractor, conformance to requirements, local politics and the likelihood of change are but a few of the criteria that may affect the decision.



Outsourcing

- Sooner or later, every company that develops computer software asks a fundamental question: Is there a way that we can get the software and systems we need at a lower price?
- The answer to this question is not a simple one, and the emotional discussions that occur in response to the question always lead to a single word: **outsourcing**. Regardless of the breadth of focus, the outsourcing decision is often a financial one.
- Outsourcing is extremely simple. Software engineering activities are contracted to a third party who does the work at lower cost and, hopefully, higher quality.
- The decision to outsource can be either **strategic** or **tactical**.
- At the **strategic level**, business managers consider whether a significant portion of all software work can be contracted to others.
- At the **tactical level**, a project manager determines whether part or all of a project can be best accomplished by subcontracting the software work.
- On the positive side, cost savings can usually be achieved by reducing the number of software people and the facilities (e.g., computers, infrastructure) that support them.
- On the negative side, a company loses some control over the software that it needs.

- The COCOMO II application composition model uses object points :
- The object point is an indirect software measure that is computed using counts of the number of
  (1) Screens (at the user interface),
  (2) Reports
  (3) Components likely to be required to build the application.
- Each object instance (e.g., a screen or report) is classified into one of three complexity levels (i.e., simple, medium, or difficult). Once complexity is determined, the number of screens, reports, and components are weighted according to the table given below



Object type	Complexity weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component			10

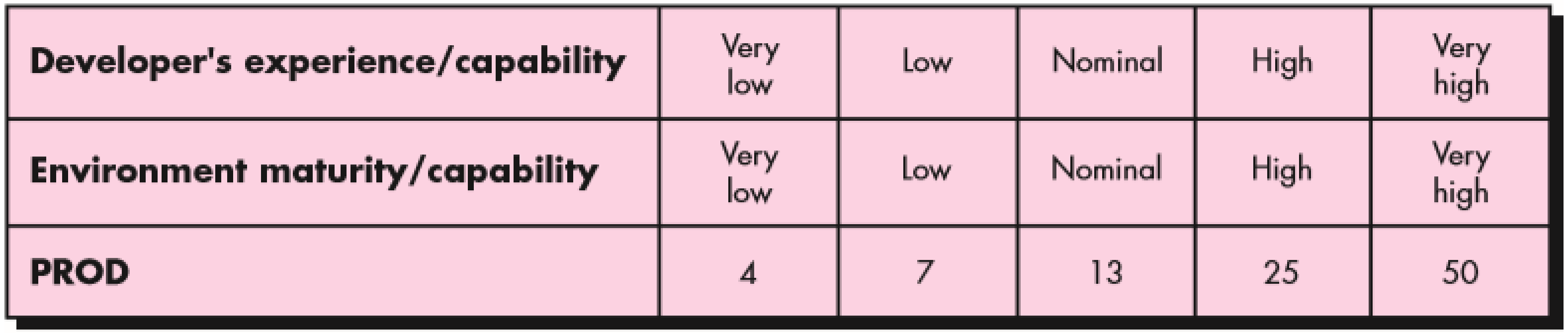
- When component-based development or general software reuse is to be applied, the percent of reuse (%reuse) is estimated and the object point count is adjusted: $\text{NOP} = (\text{object points}) \times [(100 - \%\text{reuse})/100]$

where NOP is defined as new object points.

- To derive an estimate of effort based on the computed NOP value, a productivity ratemust bederived.

$$\text{PROD} = \frac{\text{NOP}}{\text{person-month}}$$

$$\text{Estimated effort} = \frac{\text{NOP}}{\text{PROD}}$$



Developer's experience/capability	Very low	Low	Nominal	High	Very high
Environment maturity/capability	Very low	Low	Nominal	High	Very high
PROD	4	7	13	25	50

Risk Management

- A Hazardis

  Any real or potential condition that can cause injury, illness, or death to personnel; damage to or loss of a system, equipment or property; or damage to the environment. Simpler A threat of harm. A hazard can lead to one or several consequences.

- Risk is

  The expectation of a loss or damage (consequence) The combined severity and probability of a loss The long term rate of loss. A potential problem (leading to a loss) that may - or may not occur in thefuture.

  - Risk Management is A set of practices and support tools to identify, analyze, and treat risks explicitly.
  - Treating a risk means understanding it better, avoiding or reducing it (risk mitigation), or preparing for the risk tomaterialize.
  - Risk management tries to reduce the probability of a risk to occur and the impact (loss) caused byrisks.

- Reactive versus Proactive Risk Strategies
- Software risks

Reactive versus Proactive Risk Strategies

- The majority of software teams rely solely on reactive risk strategies. At best, a reactive strategy monitors the project for likely risks. Resources are set aside to deal with them, should they become actual problems.
- The software team does nothing about risks until something goes wrong. Then, the team flies into action in an attempt to correct the problem rapidly. This is often called a fire- fighting mode.
- A considerably more intelligent strategy for risk management is to be proactive.
- A proactive strategy begins long before technical work is initiated. Potential risks are identified, their probability and impact are assessed, and they are ranked by importance. Then,
- The software team establishes a plan for managing risk. The primary objective is to avoid risk, but because not all risks can be avoided, the team works to develop a contingency plan that will enable it to respond in a controlled and effective manner.

SoftwareRisks

Risk always involves two characteristics:

- Risk always involves two characteristics: uncertainty the risk may or may not happen; that is, there are no 100 percent probable risks and loss if the risk becomes a reality, unwanted consequences or losses will occur.
- When risks are analyzed, it is important to quantify the level of uncertainty and the degree of loss associated with each risk.

•Different categories of risks are follows:

1. Project risks

- ❖ Threaten the project plan. That is, if project risks become real, it is likely that the project schedule will slip and that costs will increase.
- ❖ Project risks identify potential budgetary, schedule, personnel (staffing and organization), resource, stakeholder, and requirements problems and their impact on a software project.

2. *Technical risks*

- ❖ Threaten the quality and timeliness of the software to be produced.
- ❖ If a technical risk becomes a reality, implementation may become difficult or impossible. Technical risks identify potential design, implementation, interface, verification, and maintenance problems.
- ❖ In addition, specification ambiguity, technical uncertainty, technical obsolescence, and leading-edge technology are also risk factors. Technical risks occur because the problem is harder to solve than you thought it would be.

3. *Business risks*

- ❖ Business risks threaten the viability of the software to be built and often jeopardize the project or the product.
- ❖ Candidates for the top five business risks are
  - (1) Building an excellent product or system that no one really wants (market risk)
  - (2) Building a product that no longer fits into the overall business strategy for the company (strategic risk)
  - (3) Building a product that the sales force doesn't understand how to sell (sales risk)
  - (4) Losing the support of senior management due to a change in focus or a change in people (management risk)
  - (5) Losing budgetary or personnel commitment (budget risks).

Software Risk

Another general categorization of risks has been proposed by Charette.

1. *Known risks* are those that can be uncovered after careful evaluation of the project plan, the business and technical environment in which the project is being developed, and other reliable information sources (e.g., unrealistic delivery date, lack of documented requirements or software scope, poor development environment).
2. *Predictable* risks are extrapolated from past project experience (e.g., staff turnover, poor communication with the customer, dilution of staff effort as ongoing maintenance requests are serviced).
3. *Unpredictable risks* are the joker in the deck. They can and do occur, but they are extremely difficult to identify in advance.

Risk Projection

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

(1) The likelihood or probability that the risk is realand

(2) The consequences of the problems associated with the risk, should it occur

Managers and technical staff to perform four risk projection steps:

1. Establish a scale that reflects the perceived likelihood of a risk.
2. Delineate the consequences of the risk.
3. Estimate the impact of the risk on the project and the product.
4. Assess the overall accuracy of the risk projection so that there will be no misunderstandings.

The intent of these steps is to consider risks in a manner that leads to prioritization. No software team has the resources to address every possible risk with the same degree of rigor.

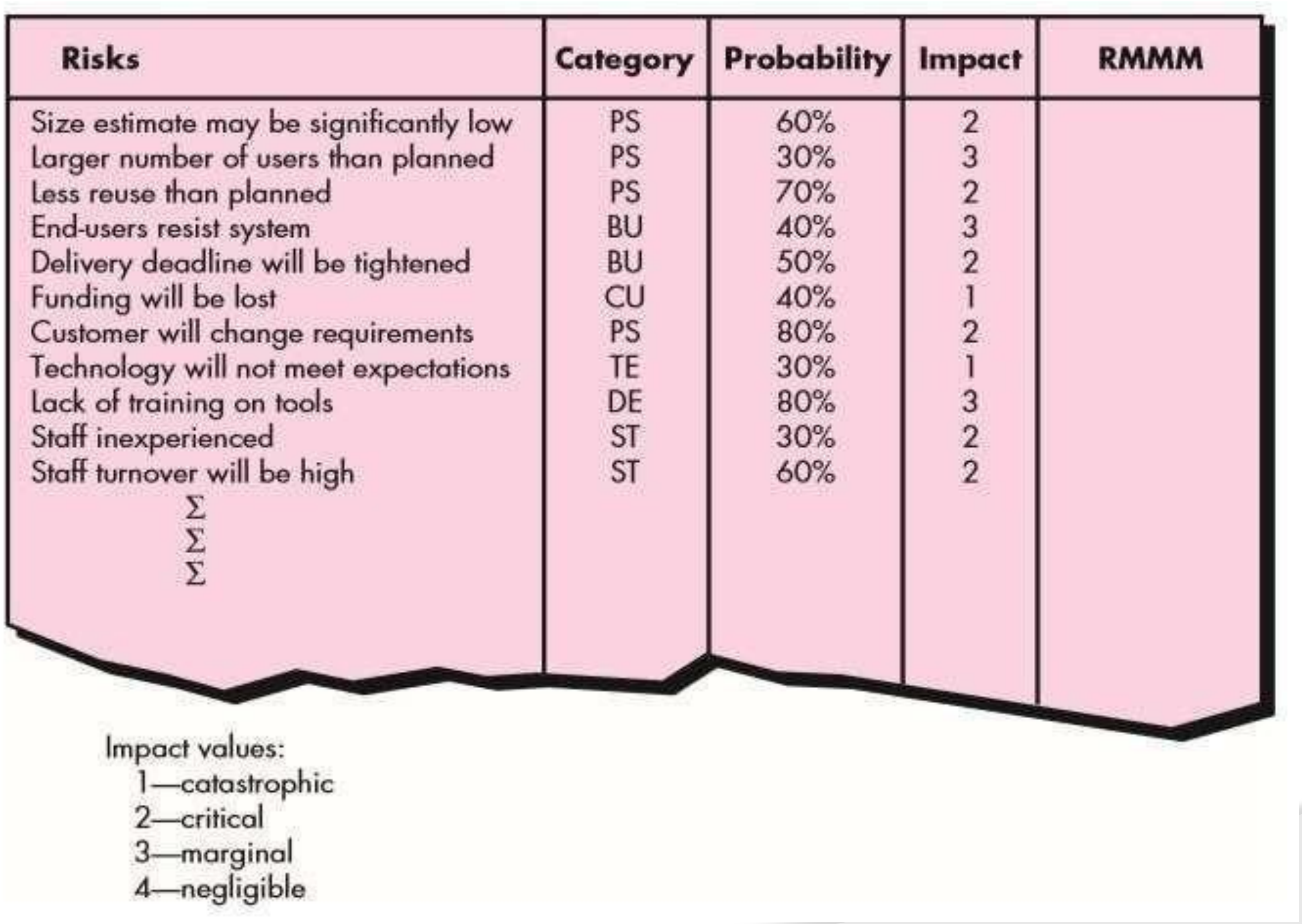
By prioritizing risks, you can allocate resources where they will have the most impact.

1. Developing a Risk Table

- A risk table provides you with a simple technique for risk projection. A sample risk table is illustrated in Figure.
- List all the risks (no matter how remote) in the first column of the table.
- Each risk is categorized in the second column (e.g., PS implies a project size risk, BU implies a business risk).
- The probability of occurrence of each risk is entered in the next column of the table. The probability value for each risk can be estimated by team members individually.
- Next, the impact of each risk is assessed. Each risk component is assessed, and an impact category is determined.
- The categories for each of the four risk components—performance, support, cost, and schedule—are averaged to determine an overall impact value.

  Once the first four columns of the risk table have been completed, the table is sorted by probability and by impact.
- High-probability, high-impact risks percolate to the top of the table, and low-probability risks drop to the bottom.

SampleRisk table priortosorting



Risks	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger number of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End-users resist system	BU	40%	3	
Delivery deadline will be tightened	BU	50%	2	
Funding will be lost	CU	40%	1	
Customer will change requirements	PS	80%	2	
Technology will not meet expectations	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover will be high	ST	60%	2	
Σ				
Σ				
Σ				

Impact values:
1—catastrophic
2—critical
3—marginal
4—negligible

What is Project Scheduling?

Basic Concept of Project Scheduling

- ✓ An unrealistic deadline established by someone outside the software development group and forced on managers and practitioner's within the 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.

▪**Why should we do when the management demands that we make a dead line I impossible?**

- ✓ Perform a detailed estimate using historical data from past projects.
- ✓ Determine the estimated effort and duration for theproject.
- ✓ Using an incremental process model, develop a software engineering strategy that will deliver critical functionality by the imposed deadline, but delay other functionality until later. Document the plan.
- ✓ Meet with the customer and (using the detailed estimate), explain why the imposed deadline is unrealistic.

- Project Scheduling
- Basic Principles
- The Relationship Between People and Effort
- Effort Distribution
- Software project scheduling is an action that distributes estimated efforts across the planned project duration by allocating the effort to specific software engineering tasks.
- During early stages of project planning, a macroscopic schedule is developed.
- As the project gets under way, each entry on the macroscopic schedule is refined into a detailed schedule.

- **Basic Principles of Project Scheduling.**

1. **Compartmentalization:** The project must be compartmentalized into a number of manageable activities and tasks. To accomplish compartmentalization, both the product and the process are refined.
2. **Interdependency:** The interdependency of each compartmentalized activity or task must be determined. Some tasks must occur in sequence, while others can occur in parallel. Other activities can occur independently.
3. **Time allocation:** Each task to be scheduled must be allocated some number of work units (e.g., persoŶ-days of effort). In addition, each task must be assigned a start date and a completion date. whether work will be conducted on a full-time or part-time basis.
4. **Effort validation:** Every project has a defined number of people on the software team. The project manager must ensure that no more than the allocated number of people have been scheduled at any given time.
5. **Defined responsibilities**. Every task that is scheduled should be assigned to a specific team member.

6. **Defined outcomes:** Every task that is scheduled should have a defined outcome. For software projects, the outcome is normally a work product (e.g., the design of a component) or a part of a work product. Work products are often combined in deliverables.
7. **Defined milestones:** Every task or group of tasks should be associated with a project milestone. A milestone is accomplished when one or more work products has been reviewed for quality and has been approved.

Each of these principles is applied as the project scheduleevolves.

The Relationship Between People and Effort

- The Relationship Between People and effort
- In a small software development project a single person can analyze requirements, perform design, generate code, and conduct tests. As the size of a project increases, more people must become involved.
- There is a common myth that is still believed by many managers who are responsible for software development projects: If we fall behind schedule, wecan always add more programmers and catch up later in the project.
- Unfortunately, adding people late in a project often has a disruptive effect on the project, causing schedules to slip even further. The people who are added must learn the system, and the people who teach them are the same people who were doing the work.
- While teaching, no work is done, and the project falls further behind. In addition to the time it takes to learn the system, more people.
- Although communication is absolutely essential to successful software development, every new communication path requires additional effort and therefore additional time.

Effort Distribution

- 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 frontend analysis and design. A similar percentage is applied to back-end testing. You can correctly infer that coding (20 percent of effort) is deemphasized.
- Work expended on project planning rarely accounts for more than 2 to 3 percent of effort, unless the plan commits an organization to large expenditures with high risk. Customer communication and requirements analysis may comprise 10 to 25 percent of project effort.
- Effort expended on analysis or prototyping should increase in direct proportion with project size and complexity.
- A range of 20 to 25 percentof effort is normally applied to software design. Time expended for design review and subsequent iteration must also be considered.
- Because of the effort applied to software design, code should follow with relatively little difficulty.
- A range of 15 to 20 percent of overall effort can be achieved. Testing and subsequent debugging can account for 30 to 40 percent of software development effort.
- The criticality of the software often dictates the amount of testing that is required. If software is human rated (i.e., software failure can result in loss of life), even higher percentages are typical.

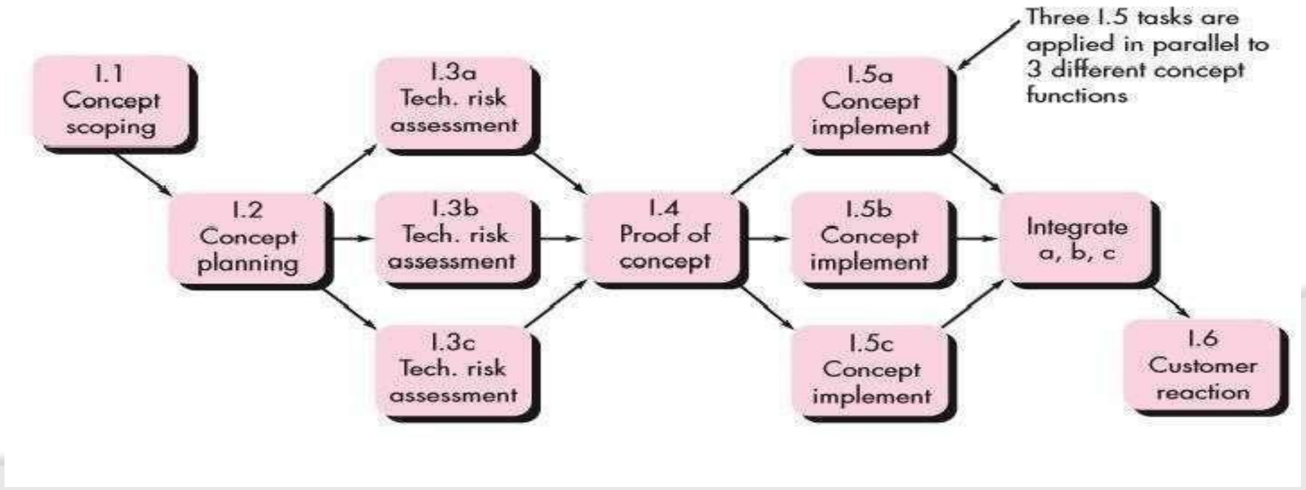
Defining a task for the Software

- A task set is a collection of software engineering work tasks, milestones, work products, and quality assurance filters that must be accomplished to complete a particular project.
- The task set must provide enough discipline to achieve high software quality. But, at the same time, it must not burden the project team with unnecessary work.
- Most software organizations encounter the followingprojects:
  1. Concept development projects that are initiated to explore some new business concept or application of some newtechnology.
  2. New application development projects that are undertaken as a consequence of a specific customerrequest.
  3. Application enhancement projects that occur when existing software undergoes major modifications to function, performance, or interfaces that are observable by the enduser.
  4. Application maintenance projects that correct, adapt, or extend existing software in ways that may not be immediately obvious to the end user.
  5. Reengineering projects that are undertaken with the intent of rebuilding an existing (legacy) system in whole or in part.

1. A Task SetExample
   - 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.
2. Refinement of Software Engineering Actions
   - The software engineering actions are used to define a macroscopic schedule for a project.
   - The macroscopic schedule must be refined to create a detailedproject schedule.
   - Refinement begins by taking each action and decomposing it into a setof tasks (with related work products and milestones).

Defining a task Network

- A task network, also called an activity network, is a graphic representation of the task flow for a project.
- It is sometimes used as the mechanism through which task sequence and dependencies are input to an automated project scheduling tool.
- In its simplest form (used when creating a macroscopic schedule), the task network depicts major software engineering actions. Figure below shows a schematic task network for a concept development project.
- It is important to note that the task network shown in Figure 27.2 is macroscopic. In a detailed task network (a precursor to a detailed schedule), each action shown in the figure would be expanded.



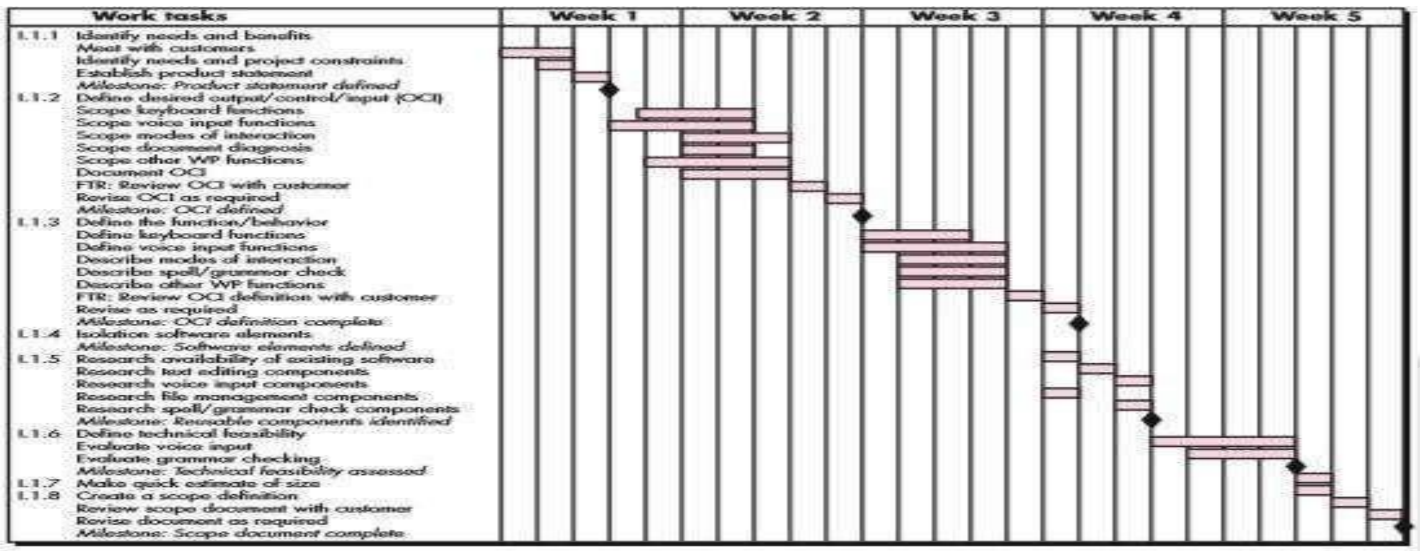
Scheduling

Scheduling of a software project does not differ greatly from scheduling of any multitask engineering effort. Therefore, generalized project scheduling tools and techniques can be applied with little modification for software projects.
Program evaluation and review technique and the critical path method (CPM) are two project scheduling methods that can be applied to software development.

1.Time-Line Charts:

- When creating a software project schedule, begin with a set of tasks.
- If automated tools are used, the work breakdown is input as a task network or task outline. Effort, duration, and start date are then input for each task. In addition, tasks may be assigned to specific individuals.
- As a consequence of this input, a time-line chart, also called a Gantt chart, is generated.
- A time-line chart can be developed for the entire project. Alternatively, separate charts can be developed for each project function or for each individual working on the project.

- All project tasks (for concept scoping) are listed in the left hand column. The horizontal bars indicate the duration of each task. When multiple bars occur at the same time on the calendar, task concurrency is implied. The diamonds indicate milestones.
- Once the information necessary for the generation of a time-line chart has been input, the majority of software project scheduling tools produce project tables. A tabular listing of all project tasks, their planned and actual start and end dates, and a variety of related information. Used in conjunction with the time-line chart, project tables enable you to track progress.



2. Tracking the Schedule

- If it has been properly developed, the project schedule becomes a road map that defines the tasks and milestones to be tracked and controlled as the project proceeds.
- Tracking can be accomplished in a number of different ways:
- Conducting periodic project status meetings in which each team member reports progress and problems.
- Evaluating the results of all reviews conducted throughout the software engineering process.
- Determining whether formal project milestones have been accomplished by the scheduled date.
- Comparing the actual start date to the planned start date for each project task listed in the resource table.
- Meeting informally with practitioners to obtain their practitioners assessment of progress to date and problems on the horizon.
- Using earned value analysis to assess progress quantitatively. In reality, al of these tracking techniques areused byexperienced project managers.

3. Tracking Progress for an OO Project

Technical milestone: OO analysis complete

- All hierarchy classes defined and reviewed
- Class attributes and operations are defined andreviewed
- Class relationships defined and reviewed o Behavioral model defined and reviewed and Reusable classed identified

Technical milestone: OO design complete

- Subsystems defined and reviewed
- Classes allocated to subsystems and reviewed
- Task allocation has been established andreviewed
- Responsibilities and collaborations have been identified Attributes and operations have been designed and reviewed o Communication model has been created and reviewed

- **Technical milestone: OO programming complete**
  - ✓Each new design model class has been implemented
  - ✓Classes extracted from the reuse library have been implemented o

  Prototype or increment has been built
- **Technical milestone: OO testing**
  - ✓The correctness and completeness of the OOA and OOD models
  - ✓has been reviewed
  - ✓Class-responsibility-collaboration network has been Developed and reviewed
  - ✓Test cases are designed and class-level tests have been conducted for
  - ✓each class
  - ✓Test cases are designed, cluster testing is completed, and classes have
  - ✓been integrated
  - ✓System level tests are complete

Scheduling for WebApp Projects

- WebApp project scheduling distributes estimated effort across the planned time line (duration) for building each WebAppincrement.
- This is accomplished by allocating the effort to specific tasks.
- The overall WebApp schedule evolves over time.
- During the first iteration, a macroscopic schedule is developed.
- This type of schedule identifies all WebApp increments and projects the dates on which each will be deployed.
- As the development of an increment gets under way, the entry for the increment on the macroscopic schedule is refined into a detailed schedule.
- Here, specific development tasks (required to accomplish an activity) are identified and scheduled.

Earned Value Analysis

- It is reasonable to ask whether there is a quantitative technique for assessing progress as the software team progresses through the work tasks allocated to the project schedule.
- A Technique for performing quantitative analysis of progress does exist. It is called earned value analysis (EVA).
- To determine the earned value, the following steps are performed:
  1. The budgeted cost of work scheduled (BCWS) is determined for each work task represented in the schedule. During estimation, the work (in person-hours or person-days) of each software engineering task is planned. Hence, BCWSi 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 BCWSi values for all work tasks that should have been completed by that point in time on the project schedule.
  2. The BCWS values for all work tasks are summed to derive the budget at completion (BAC). Hence, BAC (BCWSk) for all tasks k
  3. 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.

- 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. An SPI value close to 1.0 indicates efficient execution of the project schedule. SV is simply an absolute indication of variance from the planned schedule.

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. It is also possible to compute the actual cost of work performed (ACWP). The value for ACWP is the sum of the effort actual y 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

A CPI value close to 1.0 provides a strong indication that the project is within its defined budget. CV is an absolute indication of cost savings (against planned costs) or shortfall at a particular stage of a project.

Process and Project Metrics

What are Metrics?

- Software process and project metrics are quantitative measures
- They are a management tool
- They offer insight into the effectiveness of the software Process and the projects that are conducted using the process as aframework
- Basic quality and productivity data arecollected
- These data are analyzed, compared against past averages, and assessed
- The goal is to determine whether quality and productivity improvements have occurred
- The data can also be used to pinpoint problemareas
- Remedies can then be developed and the softwareprocess can be Improved

Reasons to Measure

- To <u>characterize</u> in order to
  - Gain an understanding of processes,products, resources, and environments
- Establish baselines for comparisons withfuture

assessments

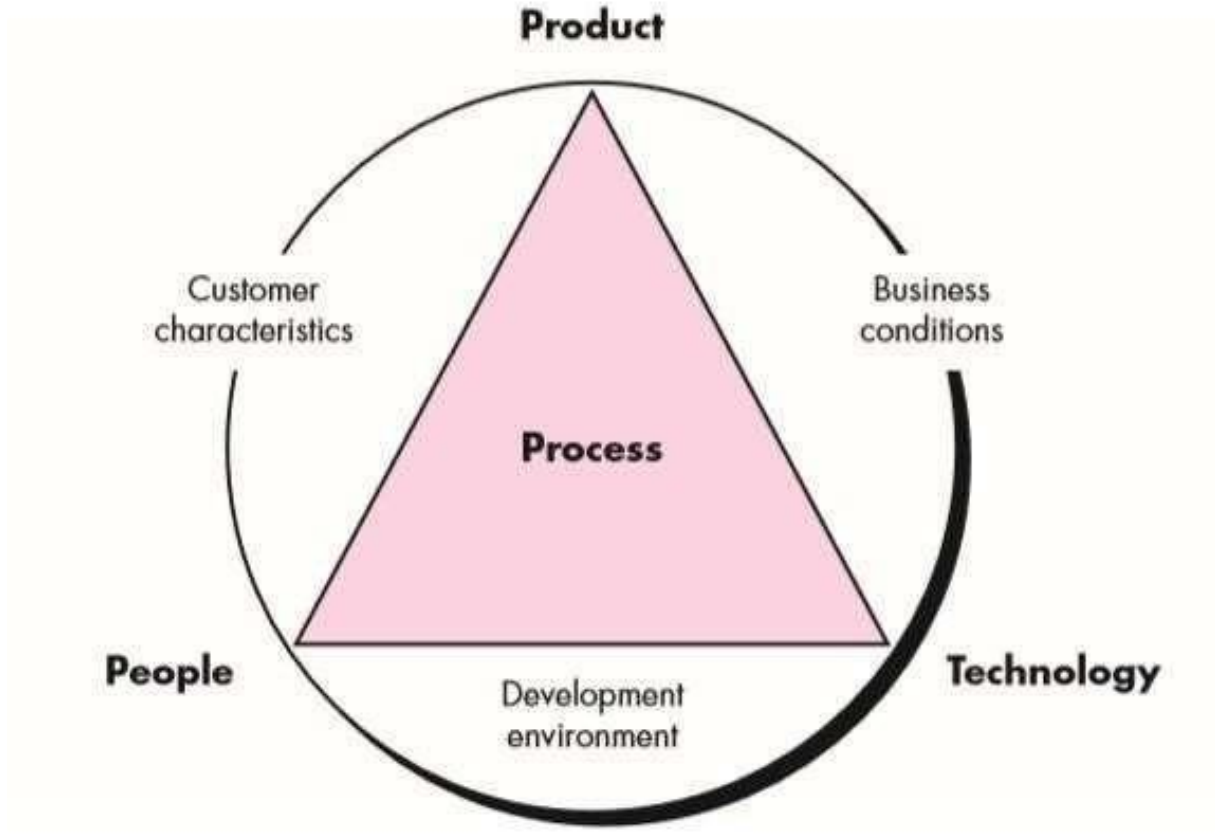
- To <u>evaluate</u> in orderto
  - Determine statuswith respect to plans
- To <u>predict</u> in order to
  - Gain understanding of relationships amongprocesses and products
- Build models of these relationships
- To <u>improve</u> in order to
  - Identify roadblocks, root causes, inefficiencies, and other opportunities for
    improving product quality and process performance

Metrics In The Process and Project Domains

- *Process metrics are collected overlong periods of time.*
- Their intent is to provide a set of process indicators that lead to long-term software processimprovement.
- *Project metrics enable a software project manager to*
  - assess the status of an ongoing project,
  - track potential risks,
  - uncover problem areas before they go
  - adjust work flow ortasks,
  - evaluate the project teaŵ's ability to control quality of software work products

Metrics In The Process and Project Domain

Determinant software quality and organizational effectiveness



MetricsIn TheProcessandProject

- **Measure the effectivenessof a process should be a set of metrics based on <u>outcomes</u> of the process such as**
  - Errors uncovered before release of thesoftware
  - Defects delivered to and reported by the end users
  - Work products delivered
  - Human effortexpended
  - Calendar timeexpended
  - Conformance to theschedule
  - Time and effort to complete each generic activity.
- **Etiquette(good manners) of Process Metrics**:
- Use common sense and organizationalsensitivity when interpreting metrics data
- Provide regular feedback to the individuals and teams who collect measures andmetrics
- DoŶ’t use metrics to evaluate individuals

Project Metrics:-

- Many of the same metrics are used in both the process and project domain Project metrics are used for making tactical decisions
- They are used to adapt project workflow and technical activities .
- The first application of project metrics occurs during estimation
- Metrics from past projects are used as a basis for estimating time and effort.
- As a project proceeds, the amount of time and effort expended are compared to original estimates.
- As technical work commences, other project metrics become important
- Production rates are measured (represented in terms of models created, review hours, function points, and delivered source lines of code)
- Error uncovered during each generic framework activity (i.e, communication, planning, modeling, construction, deployment) are measured

Software Measurement

Reconciling LOC and FP Metrics:-

- Relationship between LOC and FP dependsupon
  - The programming language that is used to implement thesoftware. The quality of thedesign
  - FP and LOC have been found to be relatively accurate predictors of software development effort and cost
  - However, a historical baseline of informationmust first beestablished. LOC and FP can be used to estimate object-orientedsoftwareprojects
  - However, they do not provide enough granularity for the schedule and effort adjustments required in the iterations of an evolutionary or incremental process The table on the next slide provides a rough estimate of the average LOC to one FP in various programming languages.

Software Measurement

LOC per function point

Language	Average	Median	Low	High
Ada	154	--	104	205
Assembler	337	315	91	694
C	162	109	33	704
C++	66	53	29	178
COBOL	77	77	14	400
Java	55	53	9	214
PL/1	78	67	22	263
Visual Basic	47	42	16	158

Object-oriented Metrics:-

Following set of metrics for OOprojects:

- **Number of scenario scripts:-** A scenario script isa detailed sequence of steps that describe the interaction between the user and the application.
- Each script is organized into triplets of theform {**initiator,** ***action, participant}***
- where initiator is the object that requests some service, *action is theresult of the request, and participant is the server object that satisfies* the request.

Number of key classes.:- Key classes are the ħ ighly independent components that are defined early in object-oriented analysis

- Because key classes are central to the problem domain, the number of such classes is an indication of the amount of effort required to develop the software.
- Also an indication of the potential amount of reuse to be applied during system development.

Numberof support classes:- Support classes are required to implement the system but are not immediately related to the problemdomain.

- The number of support classes is an indication of the amount of effort required to develop the software and also an indication of the potential amount of reuse to be applied during systemdevelopment.

- **Number of subsystems**
- A subsystem is an aggregation of classes that support a function that is visible to the end user of a system.

Average number of support classes per key class

- Key classes are identified early in a project (e.g., at requirements analysis)
- Estimation of the number of support classes can be made from the number of keyclasses
- GUI applications have between <u>two and three times</u> more support classes as key classes
- Non-GUI applications have between <u>oneand two times</u> more support classes as key classes

Use-Case–OrientedMetrics:-

Use cases describe user-visible functions and features that are basic requirements for a system.

The number of use cases is directly proportional to the size of the application in LOC and to the number of test cases that will have to be designed to fully exercise the application.

WebApp Project Metrics:-

The objective of all WebApp projects is to deliver a combination of content and functionality to the end user.

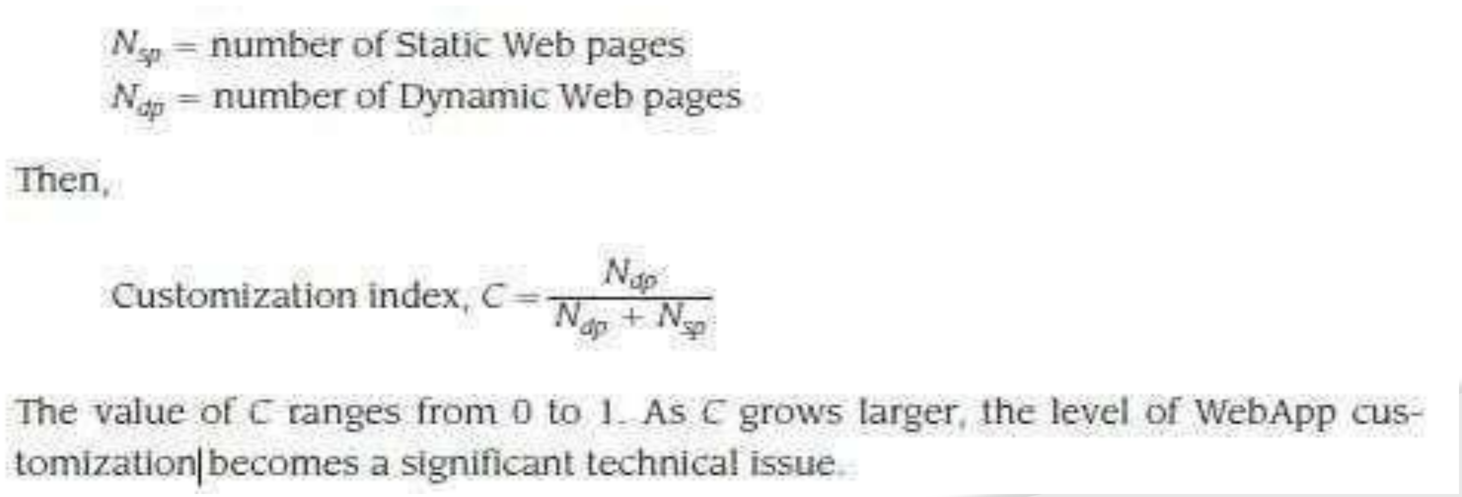
The measures that can be collected are:

- Number of static Webpages.
- Number of dynamic Webpages.
- Number of internal page links.:-Internal page linksare pointers that provide a hyperlink to some other Web page within theWebApp

SoftwareMeasurement

Number of persistent dataobjects.

- Number of external systems interfaced:- WebApps must often interface with ďaĐkrooŵbusiness applications.
- Number of static content objects:-Static content objects encompass static text-based, graphical, video, animation, and audio information that are incorporated within the WebApp.
- Number of dynamic contentobjects.
- Number of executablefunctions



N_{sp} = number of Static Web pages
N_{dp} = number of Dynamic Web pages

Then,

$$\text{Customization index, } C = \frac{N_{dp}}{N_{dp} + N_{sp}}$$

The value of C ranges from 0 to 1. As C grows larger, the level of WebApp customization becomes a significant technical issue.

- Measurements in the physical world can cartelize in two ways. direct measures andindirect measures.
- Direct measures of the software process include cost andeffortapplied.
- Direct measures of the product include lines of code (LOC) produced, execution speed, memory size, and defects reported over some set period of time.
- Indirect measures of the product include functionality, quality, efficiency, reliability, maintainability.
- Project metrics can be consolidated to create process metricsforan organization.

- Size-Oriented Metrics

- Size-oriented metrics are not universallyaccepted as thebest wayto measure thesoftwareprocess.
- Opponents argue that KLOCmeasurements Are
  - dependent on the programminglanguage
  - Penalize well-designed but shortprograms
  - Cannot easily accommodatenonprocedural languages
  - Requirea level of detail that may be difficult to achieve.

- Function-Oriented Metrics:-

- Function-oriented metrics use a measure of the functionalitydelivered
- by the application as anormalizationvalue
- Mostwidely used metric of this type is the function point
- Computationof the function point is based on characteristics of the software's information domain andcomplexity.

MetricsForSoftwareQuality

- The overriding goal of software engineering is to produce a high-quality system, application, or product within a time frame that satisfies a marketneed.
- The quality of a system, application, or product is only as good as the requirements that describe the problem, the design that models the solution, the code that leads to an executable program, and the tests that exercise the software to uncover errors.

Measuring Quality

- There are many measures of software quality,8 correctness, maintainability, integrity, and usability provide useful indicators for the projectteam

Correctness:

- Correctness is the degree to which the software performs its requiredfunction.
- The most common measure for correctness is defects per KLOC, where a defect is defined as a verified lack of conformance torequirements.
- Defects are those problems reported by a userof the program after the program has been released forgeneraluse.

Metric for Software Quality

Maintainability:

- Maintainability is the ease with which a program can be corrected if an error is encountered, adapted if its environment changes, or enhanced if the customer desires a change inrequirements.
- *Mean -time-to-change (MTTC), the time it takes to analyze the change* request, design an appropriate modification, implement the change, test it, and distribute the change to allusers.

- **Integrity:**
  - Software integrity has become increasingly important in the age of cyberterrorists and hackers.
  - Attackscan be made on all three components of software: programs, data, and documentation.
  - Tomeasure integrity, two attributes mustbedefined:
    - threat andsecurity.
- **Usability:**
  - If a program is not easy to use, it is often doomed to failure, even if the functions that it performs arevaluable

IntegratingMetricsWithin TheSoftwareProcess

Establishing aBaseline:-

- By establishing a metrics baseline, benefits can be obtained at thesoftware process, product, and project levels
- Thesame metrics can serve many masters
- The baselineconsists of data collected from past software development projects.

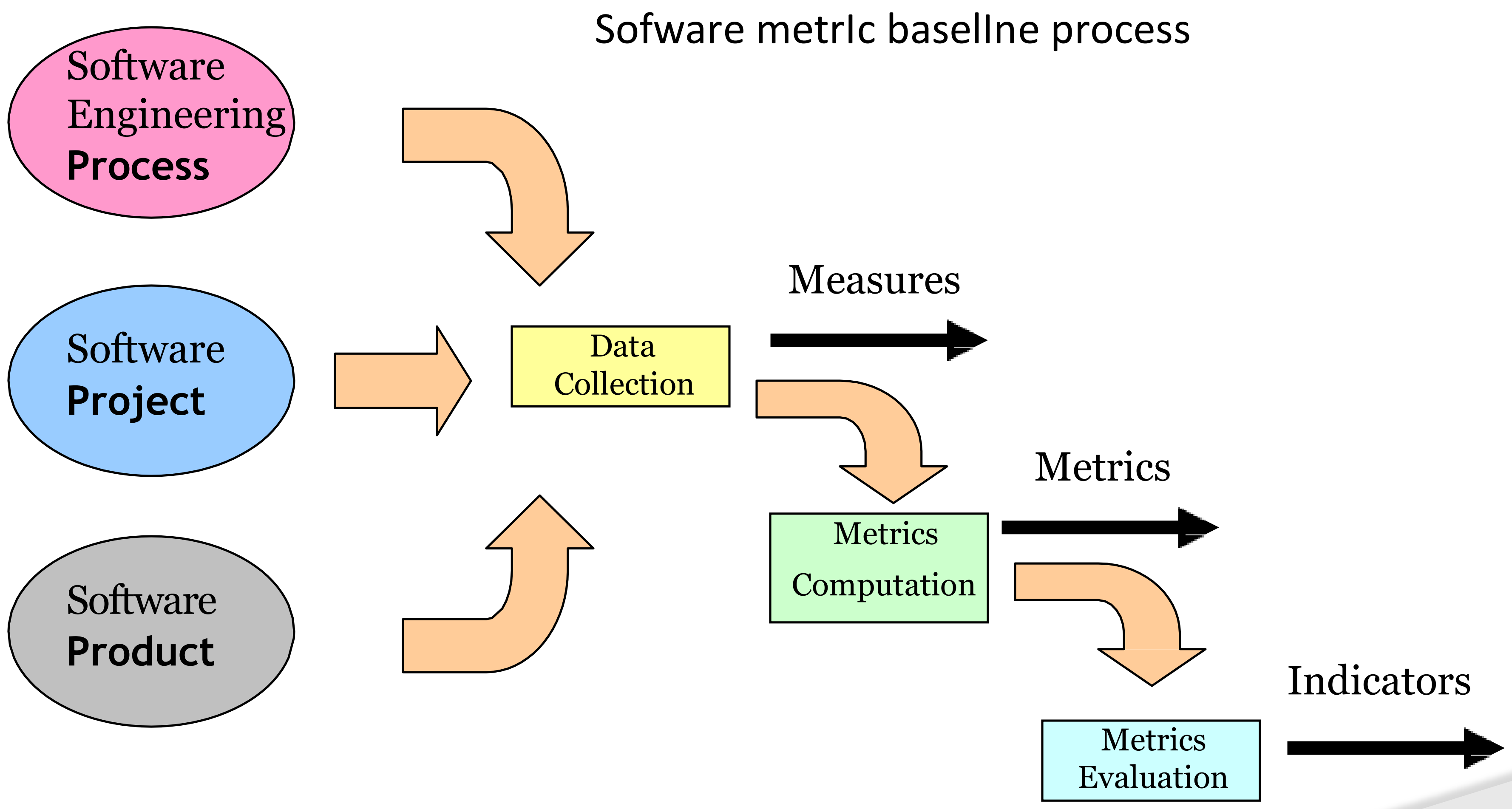
 Baselinedata must havethe following attributes

 Data must be reasonably accurate (guesses should be avoided)
 Measures must be consistent (e.g., a line of code must beinterpretedconsistentlyacross allprojects) Pastapplicationsshould be similar to thework that is to beestimated.

Metrics Collection, Computation, and Evaluation

- Data collection requires an historical investigation of past projects to reconstruct required data

Integrating Metrics Within The Software Process



Metric for Small Organization

- Mostsoftware organizations have fewer than 20 software engineers.
- It is reasonable to suggest that software organizations of all sizes measure and then use the resultant metrics to help improve their local software process and the quality and timeliness of the products they produce.
- A commonsense approach to the implementation of any software process- related activity is: keep it simple, customize to meet local needs, and be sure it adds value.

Asmall organizationmight select the followingset of easily collectedmeasures:

- Time (hours or days) elapsed from the time a request is made until evaluation is complete, *tqueue.*
- Effort (person-hours) to performtheevaluation,*Weval.*
- Time (hours or days) elapsed from completion of evaluation to assignmentof change order to personnel,*teval.*

Establishing a S/W Metrics

The Software Engineering Institute has developed a comprehensive guidebook for establishing a —goal-driven software metricsprogram.

The guidebook suggests the followingsteps:

- Identify businessgoal
- Identify what you wantto know
- Identifysubgoals
- Identify subgoal entitiesandattributes
- Formalize measurementgoals
- Identify quantifiablequestions and indicators related to subgoals
- Identifydata elements needed to be collected to construct the indicators
- Define measures to be used and create operational definitions for them.
- Identify actions needed to implementthemeasures
- Prepareaplan to implement the measures

- For example,consider the SafeHome product. Working as a team,
- software engineering and business managers develop a list of prioritized business goals:

1. Improve our customers' satisfaction with our products.
2. Make our products easier touse.
3. Reduce the time it takes us to get a new productto market.
4. Make support for our productseasier.
5. Improve our overall profitability.