UNIT - I

Introduction

Topics

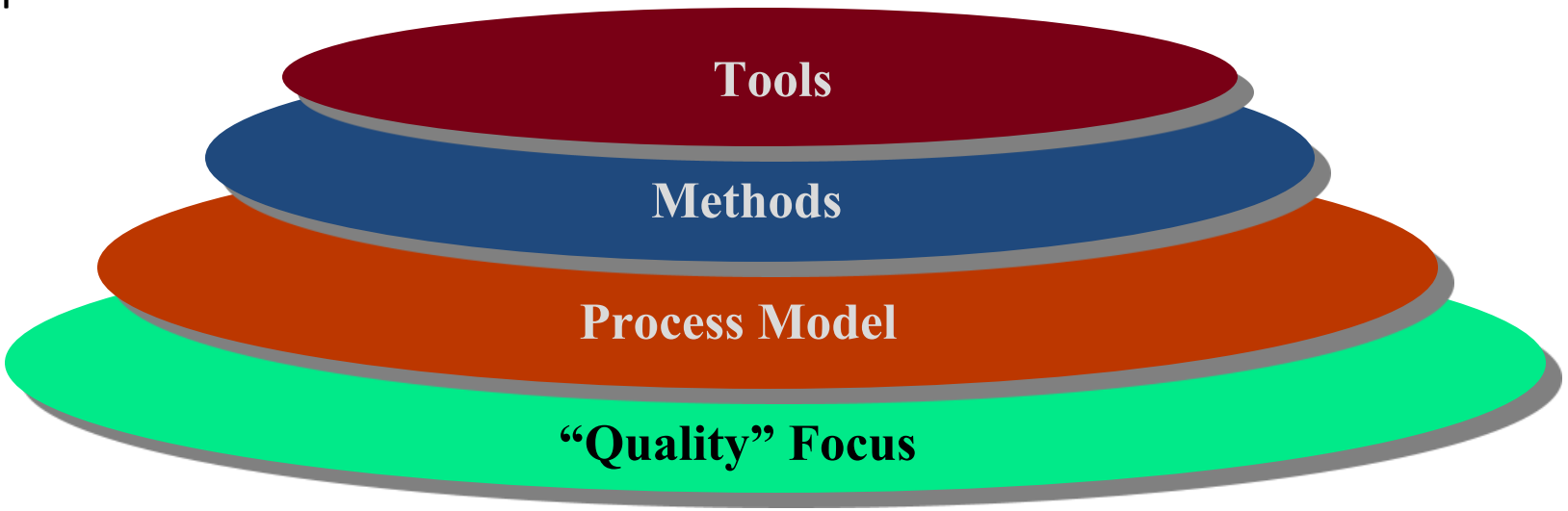
- Introduction to Software Engineering
- Software Project Management – Lifecycle Activities
- Traditional – Waterfall, V Model, Prototype, Spiral, RAD
- Conventional – Agile, XP, Scrum
- Introduction to Requirements Engineering
- Requirements Elicitation
- Software Project Effort and Cost Estimation
- Cocomo 1 and 2
- Risk Management
- Configuration Management
- Project Planning – WBC, Planning, Scope, Risk

Introduction To Software Engineering

- The IEEE definition:
  - Software Engineering: The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.
- Some realities:
  - To understand the problem before a software solution is developed
  - Design becomes a essential activity
  - Software should exhibit high quality
  - Software should be maintainable

Layered Technology

- Software engineering is a layered technology.
- The bedrock that supports software engineering is a quality focus
- The foundation for software engineering is the process layer.
- Process defines a framework that must be established for effective delivery of software.
- Software engineering methods provide the technical how-to's for building software.
- Methods encompass a broad array of tasks that include communication, requirements analysis, design modeling, program construction, testing, and support.
- Software engineering tools provide automated or semi automated support for the process and the methods.



Software Process

- A process is a collection of activities, actions, and tasks that are performed when some work product is to be created.

- An **activity** strives to achieve a broad objective
- (e.g., communication with stakeholders) and is applied regardless of the application domain, size of the project, complexity of the effort, or degree of rigor with which software engineering is to be applied.

- An **action** (e.g., architectural design) encompasses a set of tasks that produce a major work product (e.g., an architectural design
- model).

- A **task** focuses on a small, but well-defined objective (e.g., conducting a unit test) that produces a tangible outcome

A generic process framework

- **Communication**
  - The intent is to understand stakeholders' objectives for the project and to gather requirements that help define software features and functions.
- **Planning**
  - Any complicated journey can be simplified if a map exists.
  - A software project is a complicated journey, and the planning activity creates a "map" that helps guide the team as it makes the journey.
  - The map—called a software project plan—defines the software engineering work by describing the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.
- **Modeling**
  - A software engineer does the same thing by creating models to better understand software requirements and the design that will achieve those requirements.
- **Construction**
  - This activity combines code generation (either manual or automated) and the testing that is required to uncover errors in the code.
- **Deployment**
  - The software (as a complete entity or as a partially completed increment) is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation

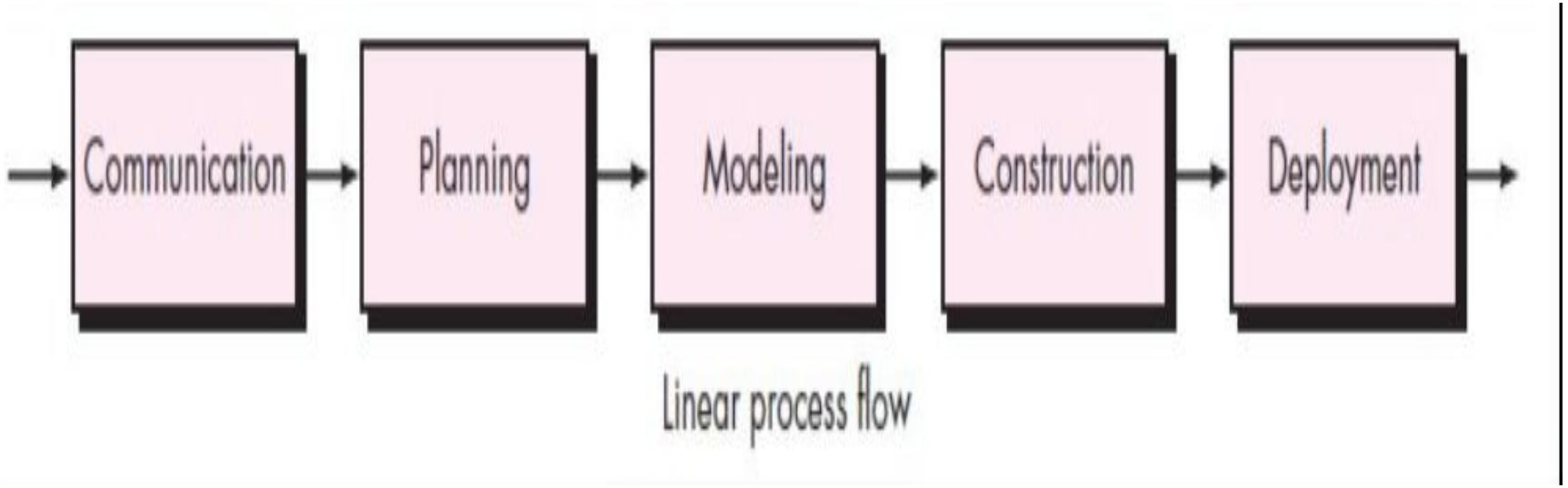
Umbrella Activities

- **Software project tracking and control**
  - Allows the software team to assess progress against the project plan and take any necessary action to maintain the schedule.
- **Risk management**
  - assesses risks that may affect the outcome of the project or the quality of the product.
- **Software quality assurance**
  - defines and conducts the activities required to ensure software quality.
- **Technical reviews**
  - assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.
- **Measurement**
  - defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders' needs;
- **Software configuration management**
  - manages the effects of change throughout the software process.
- **Reusability management**
  - defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.
- **Work product preparation and production**
  - encompasses the activities required to create work products such as models, documents, logs, forms, and lists.

Process Flow

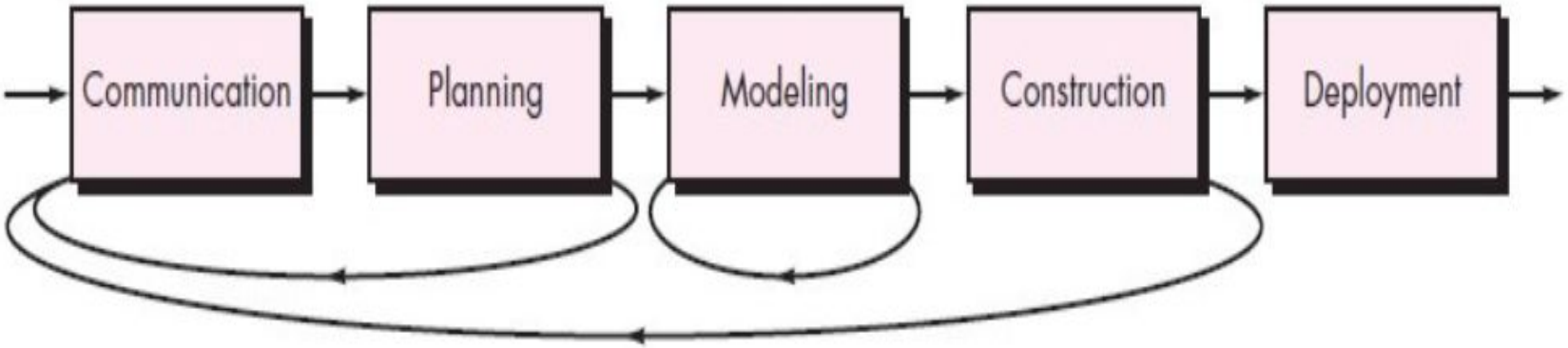
- Linear Process flow
- Iterative Process flow
- Evolutionary Process Flow
- Parallel Process Flow

Linear Process flow



- A **linear process flow** executes each of the five framework activities in sequence, beginning with communication and culminating with deployment.

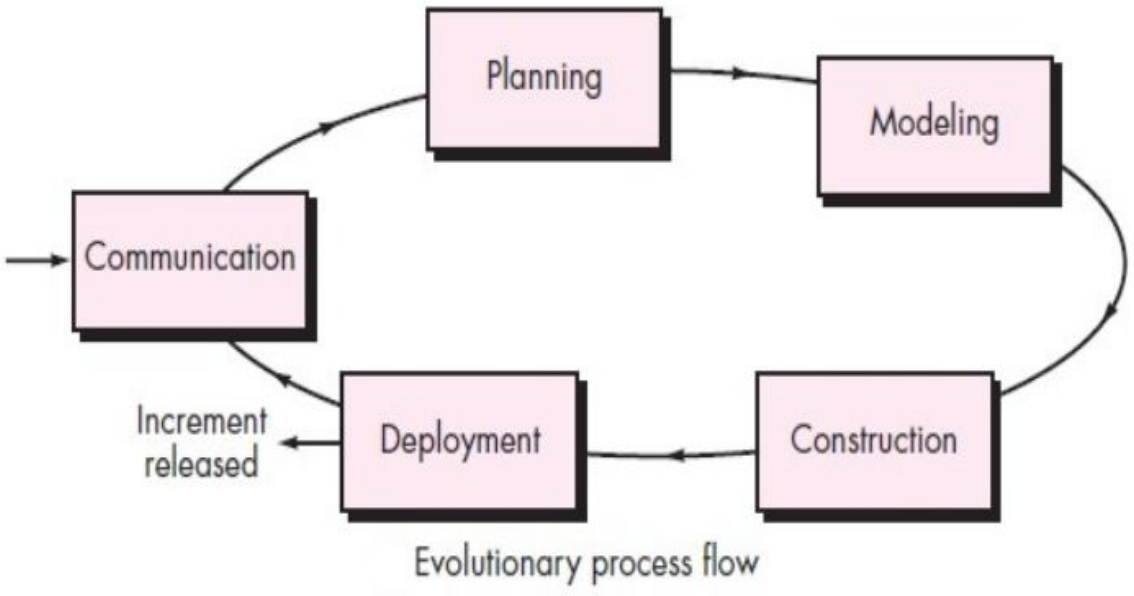
Iterative Process Flow



Iterative process flow

- An **iterative process flow** repeats one or more of the activities before proceeding to the next.

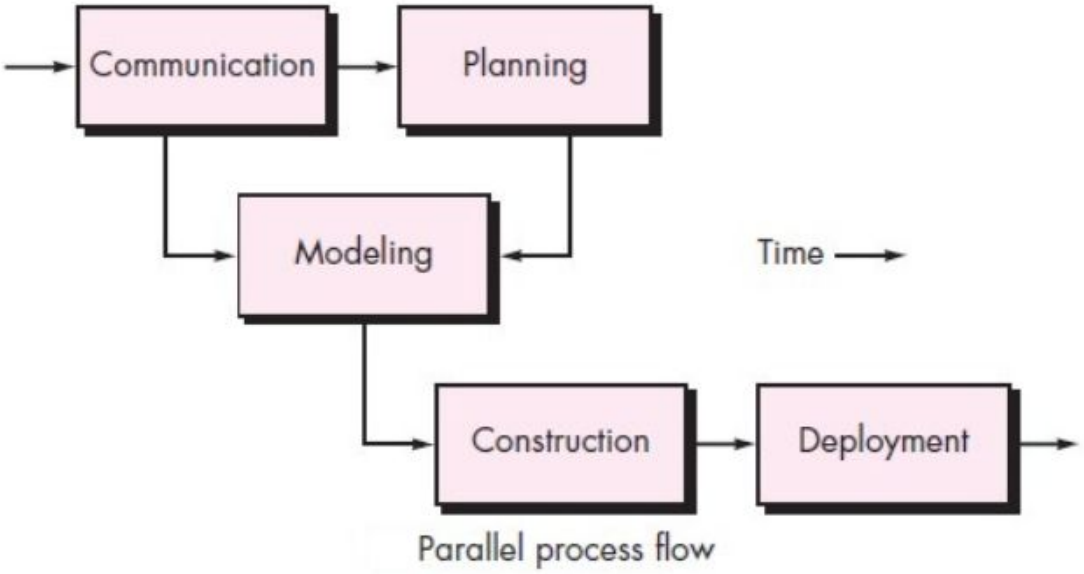
Evolutionary Process Flow



Evolutionary process flow

- An **evolutionary process flow** executes the activities in a "circular" manner.

Parallel Process flow



Parallel process flow

- A **parallel process flow** executes one or more activities in parallel with other activities.

Essence of Practice

- Polya suggests:
    1. Understand the problem (communication and analysis).
    2. Plan a solution (modeling and software design).
    3. Carry out the plan (code generation).
    4. Examine the result for accuracy (testing and quality assurance).

- **Understand the problem (Communication and Analysis)**

  - Who are the stakeholders?
  - What are the unknowns?
  - What data, functions, and features are required to properly solve the problem?
  - Can the problem be compartmentalized?
  - Is it possible to represent smaller problems that may be easier to understand?
  - Can the problem be represented graphically?
  - Can an analysis model be created?

Essence of Practice

- **Plan the Solution**

  - Have you seen similar problems before? Are there patterns that are familiar 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 sub problems be defined? If so, are solutions readily apparent for the sub problems?

  - Can you represent a solution in a manner that leads to effective implementation? Can a design model be created? Can an analysis model be created?

- **Carry Out the Plan**

  - Does the solution 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?

Essence of Practice

- **Examine the Result**

  - Is it possible to test each component part of the solution? Has a reasonable testing strategy been implemented?

  - Does the solution produce results that conform to the data, functions, and features that are required? Has the software been validated against all stakeholder requirements?

- **How it all Starts –** Safe Home:
  - Every software project is precipitated by some business need:
    - The need to correct a defect in an existing application;
    - The need to adapt a 'legacy system' to a changing business environment;
    - The need to extend the functions and features of an existing application, or
    - The need to create a new product, service, or system

Software Project Management

Introduction

- Concerned with activities involved in ensuring that software is delivered on time and on schedule and in accordance with the requirements of the organisations developing and procuring the software.
- Project management is needed because software development is always subject to budget and schedule constraints that are set by the organisation developing the software.

Success Criteria

- Deliver the software to the customer at the agreed time.
- Keep overall costs within budget.
- Deliver software that meets the customer's expectations.
- Maintain a coherent and well-functioning development team.

Software Project Management

Software Management Distinctions

- The product is intangible.
  - Software cannot be seen or touched.
  - Software project managers cannot see progress by simply looking at the artefact that is being constructed.
- Many software projects are 'one-off' projects.
  - Large software projects are usually different in some ways from previous projects.
  - Even managers who have lots of previous experience may find it difficult to anticipate problems.
- Software processes are variable and organization specific.
  - We still cannot reliably predict when a particular software process is likely to lead to development problems.

Factors Influencing Project Management

- Company size
- Software customers

Software Project Management

Factors Influencing Project Management

- Software size
- Software type
- Organizational culture
- Software development processes
- These factors mean that project managers in different organizations may work in quite different ways.

Software Management Lifecycle Activities

- Proposal writing
  - The first stage in a software project may involve writing a proposal to win a contract to carry out an item of work.
  - The proposal describes the objectives of the project and how it will be carried out.

- Project planning
  - Project managers are responsible for planning, estimating and scheduling project development and assigning people to tasks.

Software Project Management

Software Management Lifecycle Activities

- Risk management
  - Project managers assess the risks that may affect a project, monitor these risks and take action when problems arise.

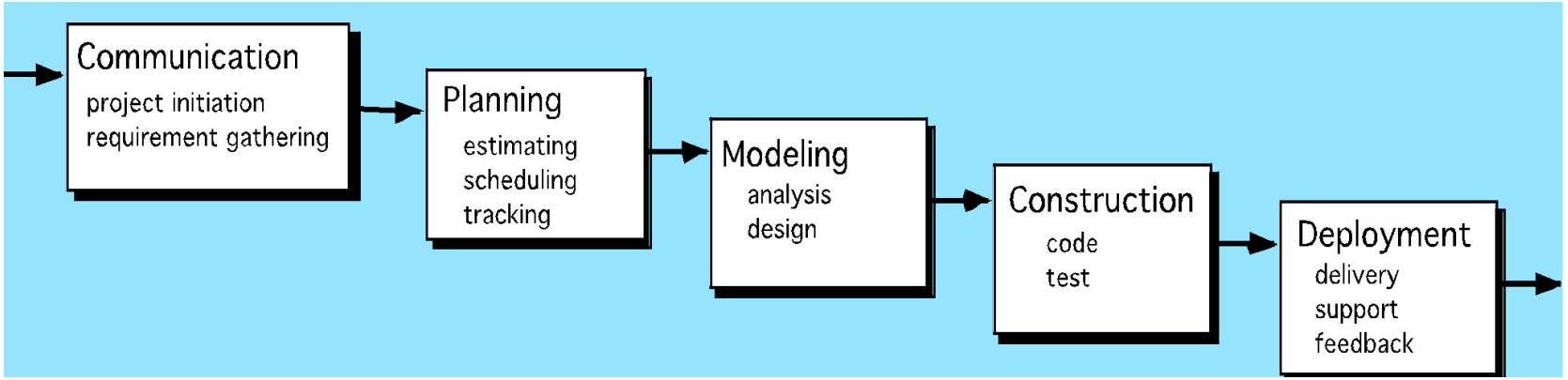
- People management
  - Project managers have to choose people for their team and establish ways of working that leads to effective team performance.

- Reporting
  - Project managers are usually responsible for reporting on the progress of a project to customers and to the managers of the company developing the software.

Traditional Software Process Models

Waterfall

- It is the oldest paradigm for Software Engineering.
- When requirements are well defined and reasonably stable, it leads to a linear fashion.
- The waterfall model, sometimes called the classic life cycle, suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction and deployment, culminating in ongoing support of the completed software.
- When to select?
- There are times when the requirements for a problem are well understood—when work flows from communication through deployment in a reasonably linear fashion.



Traditional Software Process Models

Waterfall - Problems

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
  - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
  - Few business systems have stable requirements.
- The waterfall model is mostly used for large system engineering projects where a system is developed at several sites.
  - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

Waterfall - Advantages

1. Simple
2. Easy to understand even for non technical customers
3. Oldest, widely used
4. Base for all other models by including feed back loops, iterations etc.

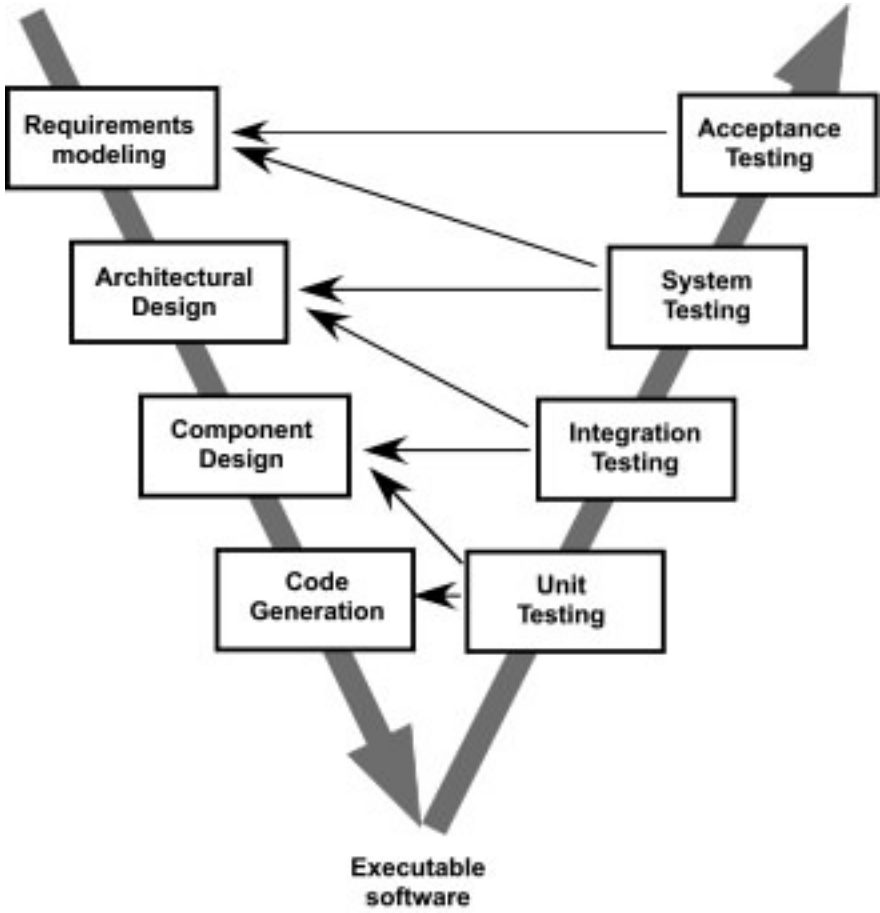
Traditional Software Process Models

Waterfall - Disadvantages

1. Real projects rarely follow this linear sequence.
2. Difficult for customer to state all requirements at one shot
3. Customer must have patience.

V-Model

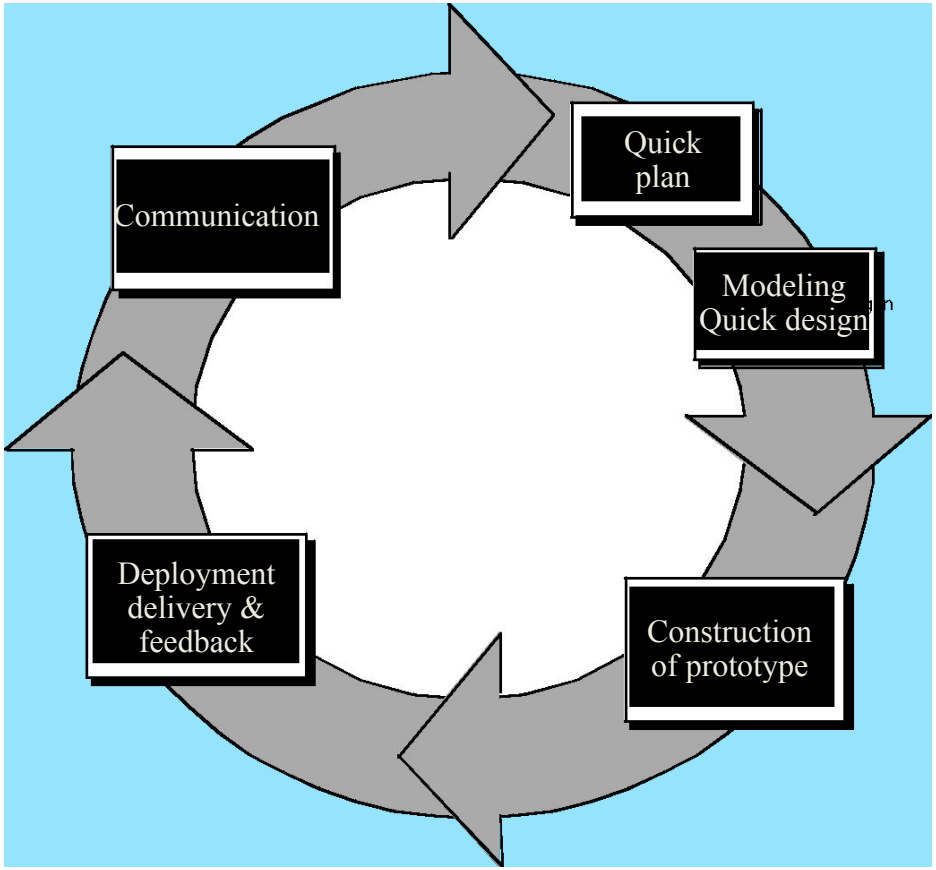
- A variation of waterfall model depicts the relationship of quality assurance actions to the actions associated with communication, modeling and early code construction activities.
- Team first moves down the left side of the V to refine the problem requirements.
- Once code is generated, the team moves 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.



Traditional Software Process Models

Prototyping

- Begins with communication
- A quick plan for prototyping and modeling occur.
- Quick design focuses on the representation of those aspects the software 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.

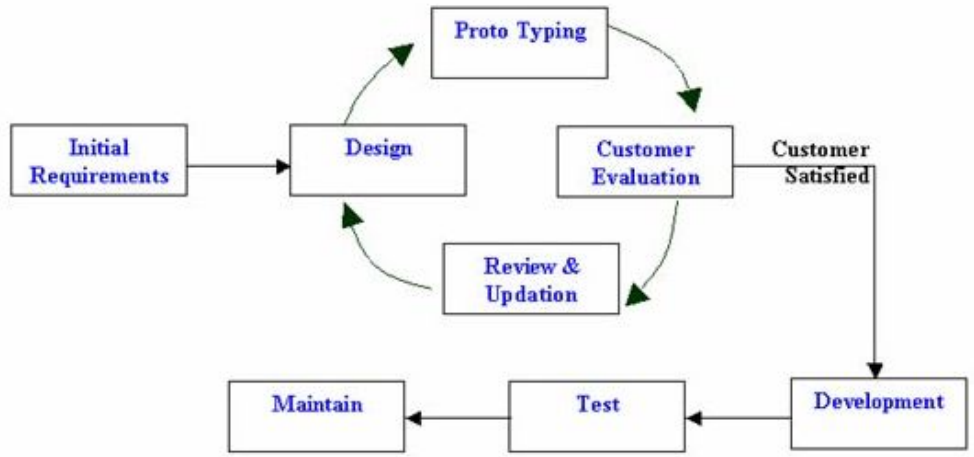


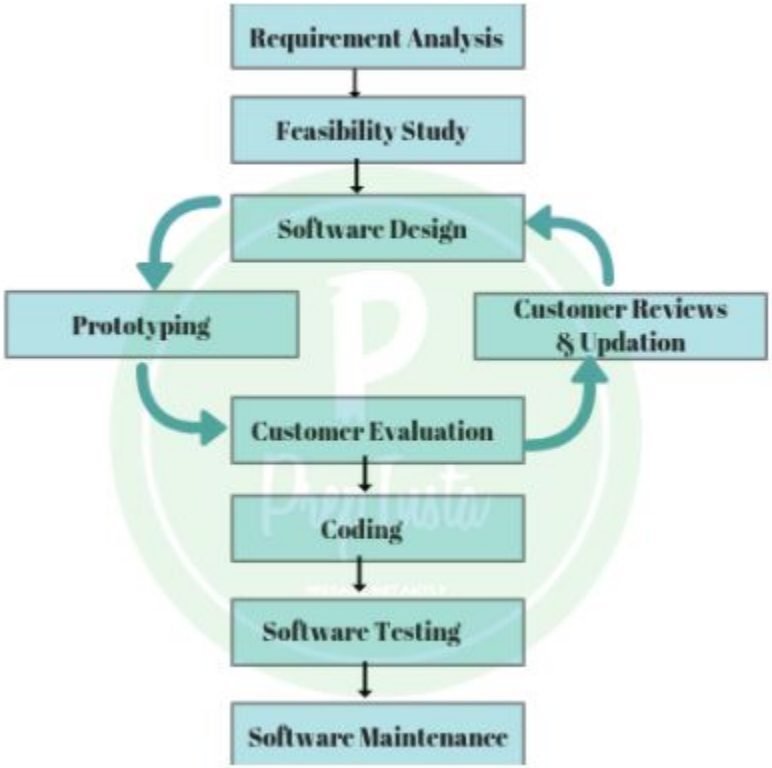
Prototype Model

- A Prototype is a Basic Working Model, Mock-up or a Simple Simulation of the Product

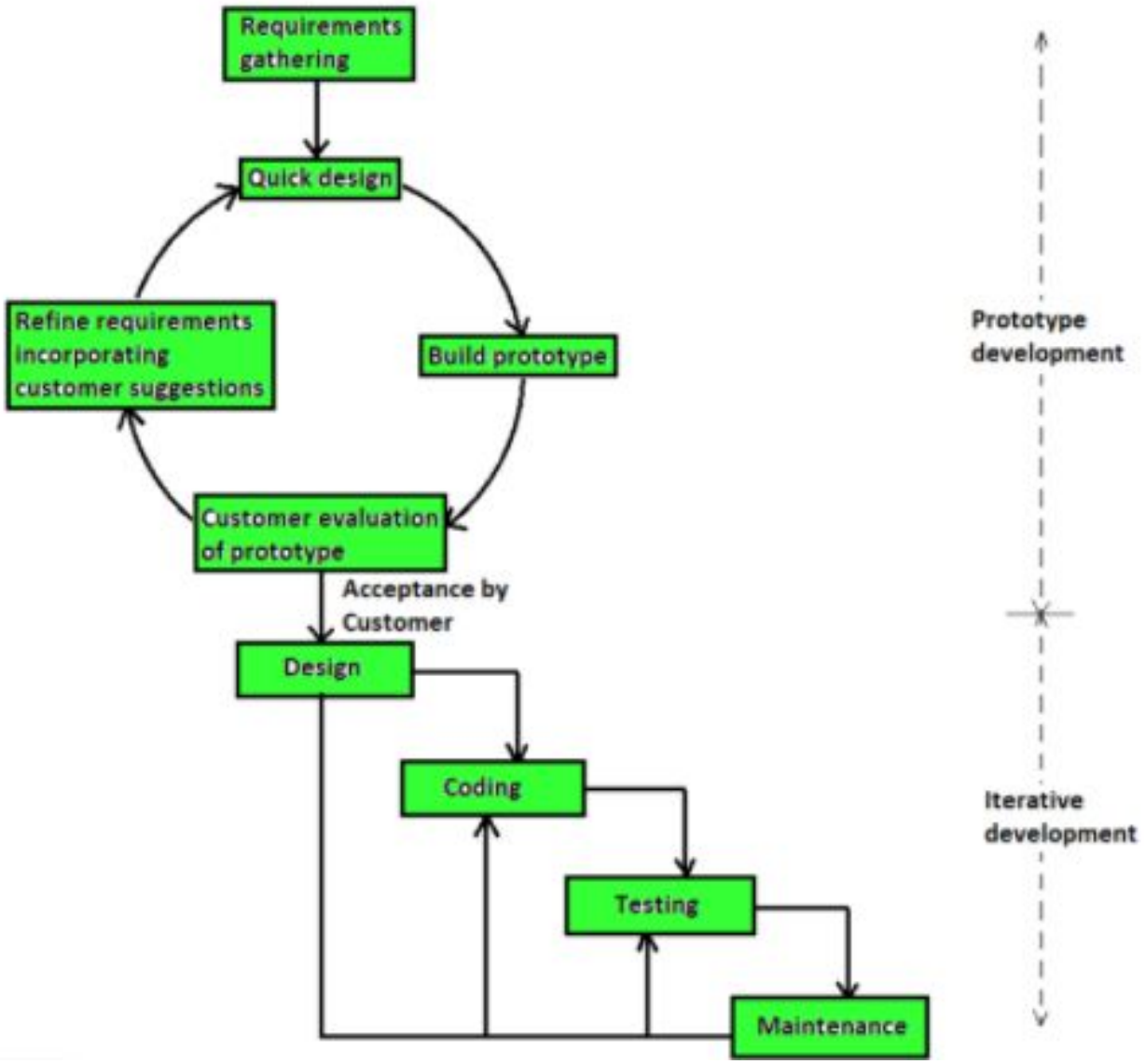
- The main reason for prototyping is to validate the idea and this is the step in converting an idea to a real product

" A Prototype is worth a Thousand Words"





Prototype Model



Traditional Software Process Models

Prototyping – When to Select

- Customer defines a set of general objectives
- Does not identify detailed requirements
- Developer may be unsure of the efficiency of an algorithm, the form that human computer interaction should take.
- When your customer has a legitimate need, but is clueless about the details, develop a prototype as a first step.

Prototyping - Advantages

1. Provides working model.
2. Customer is highly satisfied with such a modeling at initial stages
3. Developer gains business insight, reducing ambiguity
4. Great involvement of users
5. Reduce risks

Traditional Software Process Models

Prototyping - Disadvantages

1. Customer - not aware that only interface or appearance is concentrated much and long term quality is at stake.
2. False expectations from customer that end software modelling is finished or will have the same behavior/pace of the prototype.
3. Inappropriate choices of technology
4. Various iterations to a prototype that is to be discarded is expensive

Traditional Software Process Models

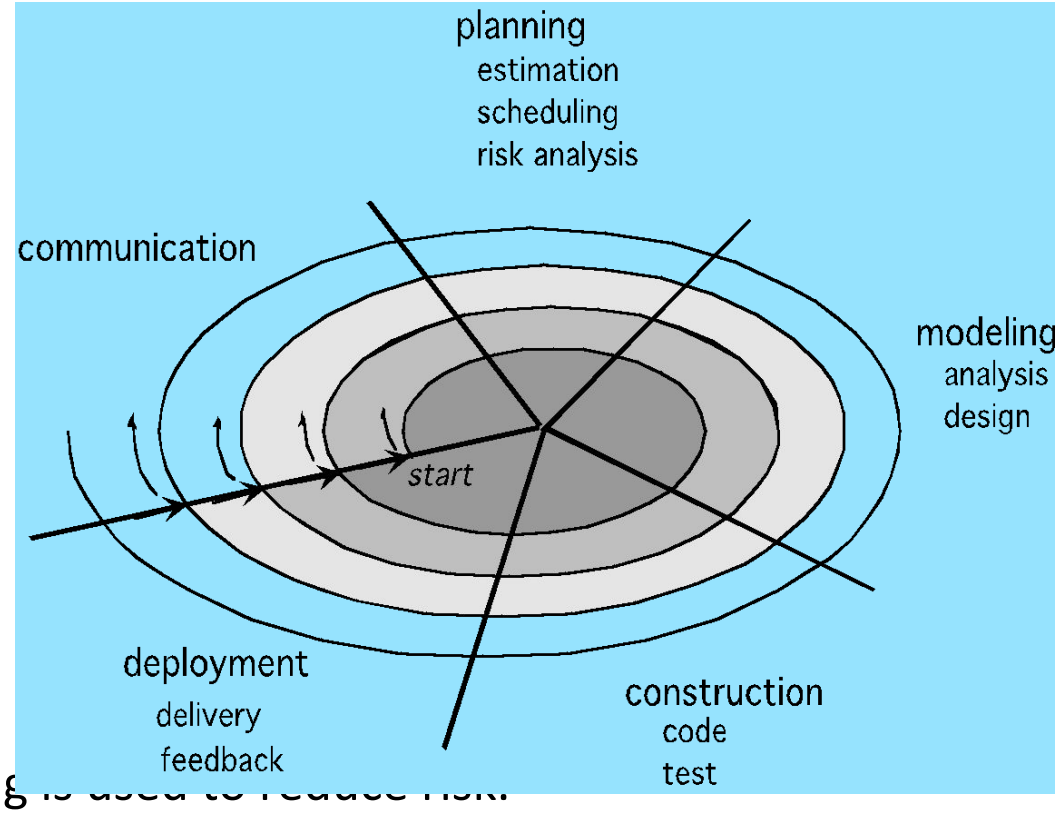
Spiral

- 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 system'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.

Traditional Software Process Models

Spiral

- 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.

Risk Analysis Model / Meta Model

- Risk Analysis, which seeks to identify situations that might cause a development effort to fail or go over budget / schedule, occurs during each spiral cycle
- Displaced Sweep denotes increasing levels of effort required for risk analysis

- Model appears like a spiral with many loops.
- The exact number of loops in the spiral is not fixed.
- The Spiral Model is called a **Meta Model** since it **encompasses all other life cycle models**.

The development spiral consists of four quadrants as shown in t

Quadrant 1: Determine objectives, alternatives, and constraints.

Quadrant 2: Evaluate alternatives, identify, resolve risks.

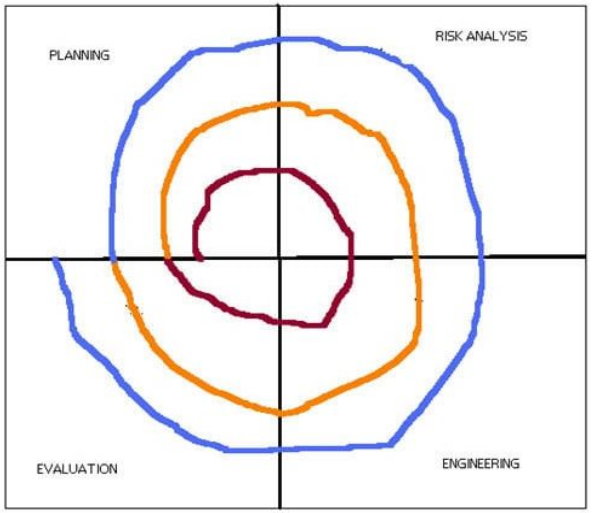
Quadrant 3: Develop, verify, next-level product.

Quadrant 4: Plan next phases.

4 Phases of Spiral Model

The Spiral Model has Four Phases:

- Planning,
- Risk Analysis,
- Engineering &
- Evaluation.



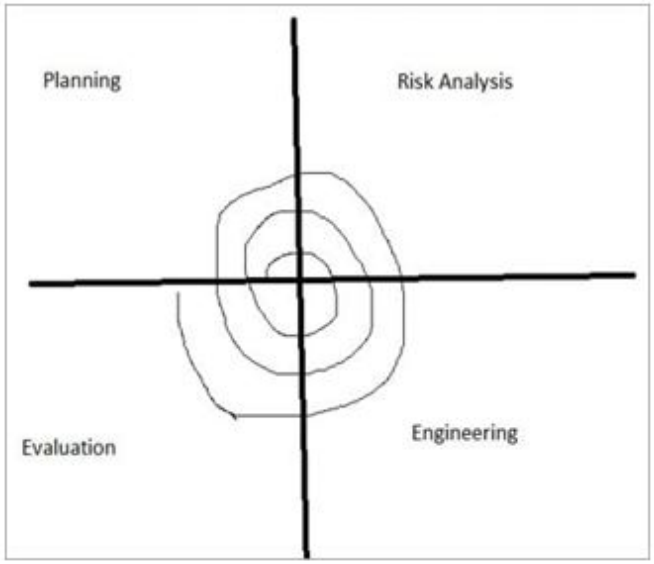
Planning Phase: Requirements are gathered during the planning phas Requirements like 'BRS' that is 'Bussiness Requirement Specificatio and 'SRS' that is 'System Requirement specifications'.

Risk Analysis: In the **risk analysis phase**, a process is undertaken to identify risk and alternate solutions. A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.

Engineering Phase: In this phase software is **developed**, along with testing at the end of the phase. Hence in this phase the development and testing is done.

Evaluation phase: This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

PHASES OF SPIRAL MODEL / TASK REGIONS



Traditional Software Process Models

Spiral - Advantages

1. Applies throughout lifecycle
    - Concept Development
    - New Product Development
    - Product Enhancement
2. Risk is considered at each pass
3. Uses prototyping as risk reduction mechanism
4. Customer and developer understand and better react to risks

Spiral - Disadvantages

1. Difficult to convince customers that it is controllable
2. Demands considerable risk assessment expertise
3. Major risk is not uncovered/managed, problem will occur

Traditional Software Process Models

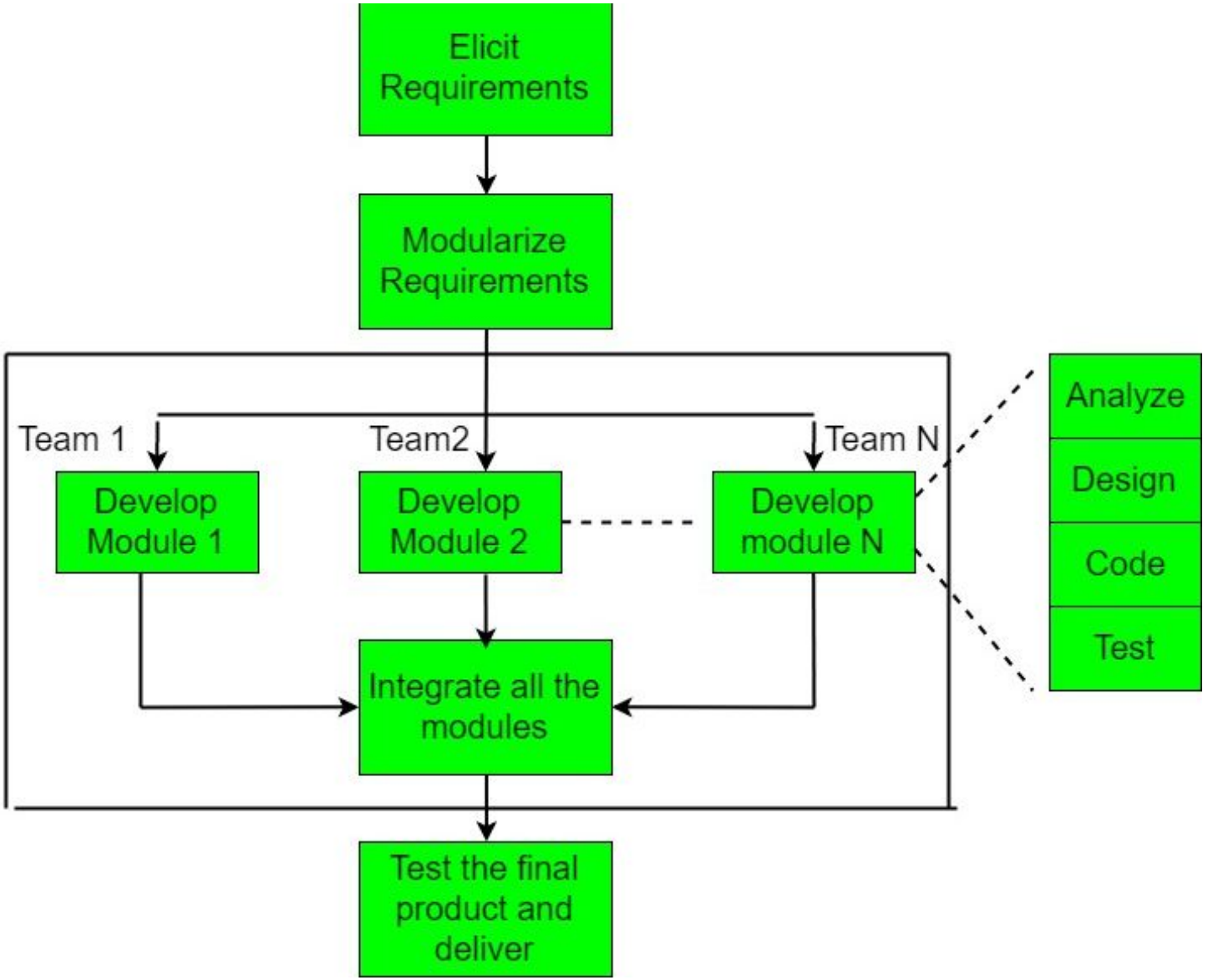
Rapid Application Development

- An approach to software development aimed at rapid delivery of the software.
- It often involves the use of database programming and development support tools such as screen and report generators.
- There are three broad phases to RAD:
  - Requirements planning
  - RAD design workshop
  - Implementation
- Requirements planning

  - Users and analysts meet to identify objectives of the application or system

  - Oriented toward solving business problems
- RAD Design Workshop
  - Design and refine phase
  - Use group decision support systems to help users agree on designs
  - Programmers and analysts can build and show visual representations of the designs and workflow to users

RAD Process Model



Traditional Software Process Models

Rapid Application Development

- RAD Design Workshop
  - Users respond to actual working prototypes
  - Analysts refine designed modules based on user responses
- Implementation Phase

  - As the systems are built and refined, the new systems or partial systems are tested and introduced to the organization

  - When creating new systems, there is no need to run old systems in parallel

- The Martin approach to RAD includes four phases:
  - Requirements planning
  - User design
  - Construction
  - Cutover

Traditional Software Process Models

Rapid Application Development – When to Select

- The team includes programmers and analysts who are experienced with it
- There are pressing reasons for speeding up application development
- The project involves a novel ecommerce application and needs quick results
- Users are sophisticated and highly engaged with the goals of the company

Rapid Application Development – Advantages

- RAD is very powerful when used within the SDLC
- It can be used as a tool to update, improve or innovate selected portions of the system

Rapid Application Development – Disadvantages

- May try and hurry the project too much
- Loosely documented
- May not address pressing business problems
- Potentially steep learning curve for programmers inexperienced with RAD tools

S.No.	Spiral Model	Prototype Model
1	Spiral model is a risk-driven software development model and is made with features of incremental, waterfall or evolutionary prototyping models.	Prototype model is a software development model in which a prototype is built, tested and then refined as per customer needs.
2	It is also referred to as meta mode	It is also referred to as rapid or closed ended prototype
3	It takes special care about risk analysis and alternative solution is undertaken	It does not give emphasis on risk analysis.
4	In spiral model, there is no continuous customer interaction	In prototype model, customer interaction is continuous until the final prototype is approved
5	It is best suited when the customer requirements are clear.	It is best suited when the requirement of the client is not clear and supposed to be changed

S.No	Spiral Model	Prototype Model
6	Cost effective quality improvement is not possible.	Cost effective quality improvement is very much possible
7	In Spiral model, improvement of quality can increase the cost of product.	In Prototype model, improvement of quality does not increases the cost of product

Differences between Various Lifecycle Models

Properties of Model	Water-Fall Model	Incremental Model	Spiral Model	Rad Model
Planning in early stage	Yes	Yes	Yes	No
Returning to an earlier phase	No	Yes	Yes	Yes
Handle Large-Project	Not Appropriate	Not Appropriate	Appropriate	Not Appropriate
Detailed Documentation	Necessary	Yes but not much	Yes	Limited
Cost	Low	Low	Expensive	Low
Requirement Specifications	Beginning	Beginning	Beginning	Time boxed release

Properties of Model	Water-Fall Model	Incremental Model	Spiral Model	Rad Model
Flexibility to change	Difficult	Easy	Easy	Easy
User Involvement	Only at beginning	Intermediate	High	Only at the beginning
Maintenance	Least	Promotes Maintainability	Typical	Easily Maintained
Duration	Long	Very long	Long	Short
Risk Involvement	High	Low	Medium to high risk	Low
Framework Type	Linear	Linear + Iterative	Linear + Iterative	Linear
Testing	After completion of coding phase	After every iteration	At the end of the engineering phase	After completion of coding
Overlapping Phases	No	Yes (As parallel development is there)	No	Yes
Maintenance	Least Maintainable	Maintainable	Yes	Easily Maintainable
Re-usability	Least possible	To some extent	To some extent	Yes

Properties of Model	Water-Fall Model	Incremental Model	Spiral Model	Rad Model
Time-Frame	Very Long	Long	Long	Short
Working software availability	At the end of the life-cycle	At the end of every iteration	At the end of every iteration	At the end of the life cycle
Objective	High Assurance	Rapid Development	High Assurance	Rapid development
Team size	Large Team	Not Large Team	Large Team	Small Team
Customer control over administrator	Very Low	Yes	Yes	Yes

Agile Methodology

- **Customer Satisfaction** through **Early and Continuous Software Delivery** – **Customers** are Happier when they receive Working Software at regular intervals, rather than waiting extended periods of time between releases.
- The Agile Manifesto and the Twelve Principles of Agile Software were the consequences of Industry Frustration in the 1990's.
- The **Enormous Time Lag** between **Business Requirements** (the Applications and Features customers were Requesting) and the **Delivery of Software** that answered those Needs, led to the cancelling of many projects

History of Agile Model

- Business Requirements and Customer Requisites **Changed** during this Lag Time, and the Final Product **did not meet** the then **Current Needs**.
- The Software Development Models available, did not meet the demand for speed and did not take advantage of just how quickly software could be altered.
- **In 2000**, a **Group of Seventeen "Thought Leaders"** including **Jon Kern, Kent Beck, Ward Cunningham, Arie van Bennekum, and Alistair Cockburn**, met where they Published the Agile Manifesto and the Twelve Principles of Agile Development.

Four Values of The Agile Manifesto

- The Agile Manifesto is comprised of **4 Foundational Values** and **12 Supporting Principles** which lead the Agile Approach to Software Development.
  - **Individuals and Interactions Over Processes and Tools**
  - **Working Software Over Comprehensive Documentation**
  - **Customer Collaboration Over Contract Negotiation**
  - **Responding to Change Over Following a Plan**

Conventional Software Process Models

Agile

- Agile software development is a group of software development methods in which requirements and solutions evolve through collaboration between self-organizing, cross-functional teams.
- It promotes adaptive planning, evolutionary development, early delivery, continuous improvement, and encourages rapid and flexible response to change.
- What is Agility?
  - Effective (rapid and adaptive) response to change
  - Effective communication among all stakeholders
  - Drawing the customer onto the team
  - Organizing a team so that it is in control of the work performed
  - Yielding …Rapid, incremental delivery of software
- Agile Process is driven by customer descriptions of what is required (scenarios)
- Recognizes that plans are short-lived
- Develops software iteratively with a heavy emphasis on construction activities
- Delivers multiple ‘software increments’
- Adapts as changes occur

Conventional Software Process Models

Agile - Principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face–to–face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.

Conventional Software Process Models

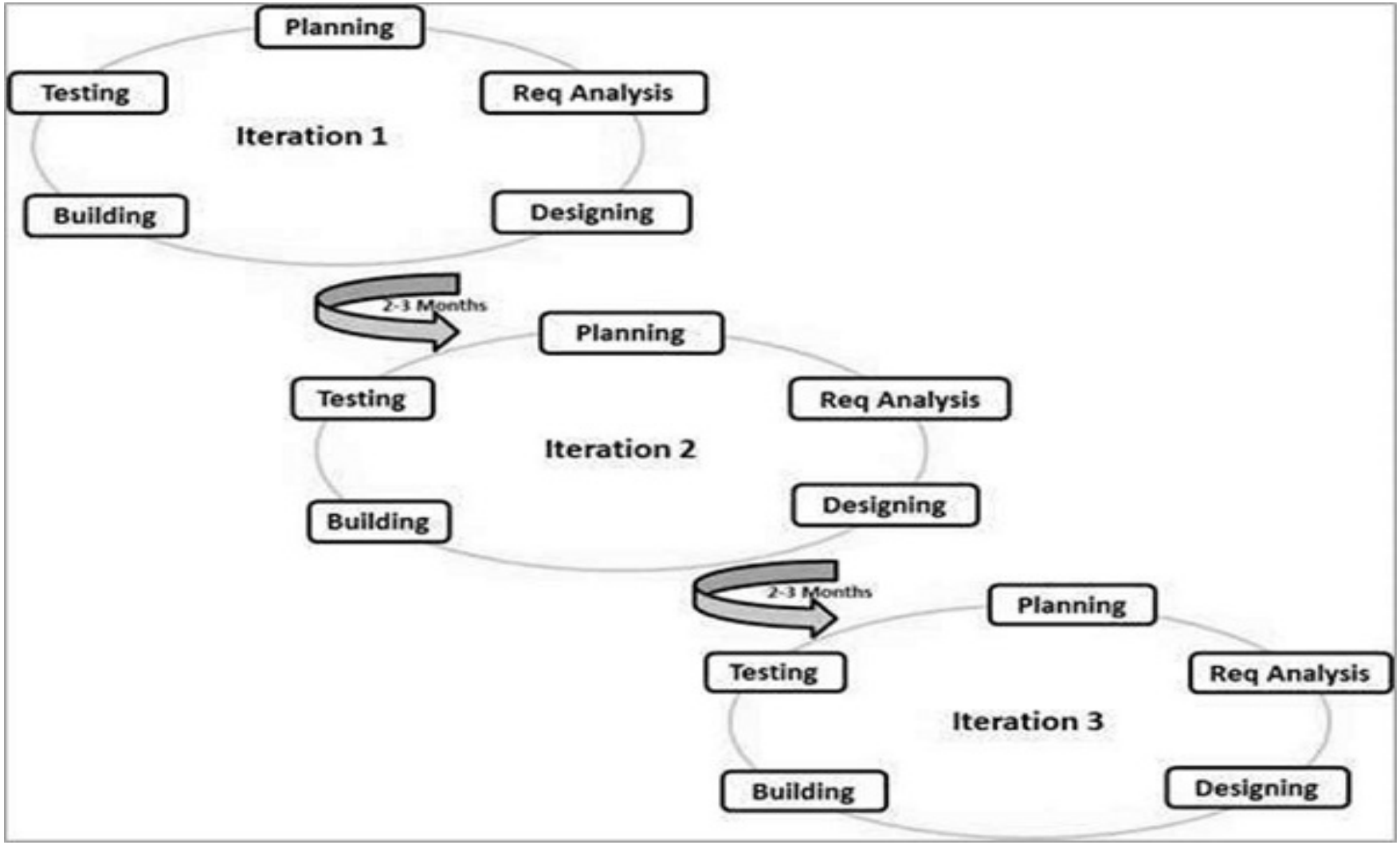
Agile – Principles

11. The best architectures, requirements, and designs emerge from self–organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Human Factors

- The process moulds to the needs of the people and team, not the other way around
- Key traits must exist among the people on an agile team and the team itself:
  - Competence
  - Common focus
  - Collaboration
  - Decision-making ability
  - Fuzzy problem-solving ability
  - Mutual trust and respect
  - Self-organization

Graphical illustration of the Agile Model



Conventional Software Process Models

Agile – Extreme Programming (XP)

- The most widely used agile process, originally proposed by Kent Beck
- XP Planning
  - Begins with the creation of "user stories (Listening leads to creation of set of "stories" (also called user stories) that describe required output, features and functionality for software to be built)."
  - Agile team assesses each story and assigns a cost
  - Stories are grouped, for a deliverable increment
  - A commitment is made on delivery date
  - After the first increment "project velocity" is used to help define subsequent delivery dates for other increments

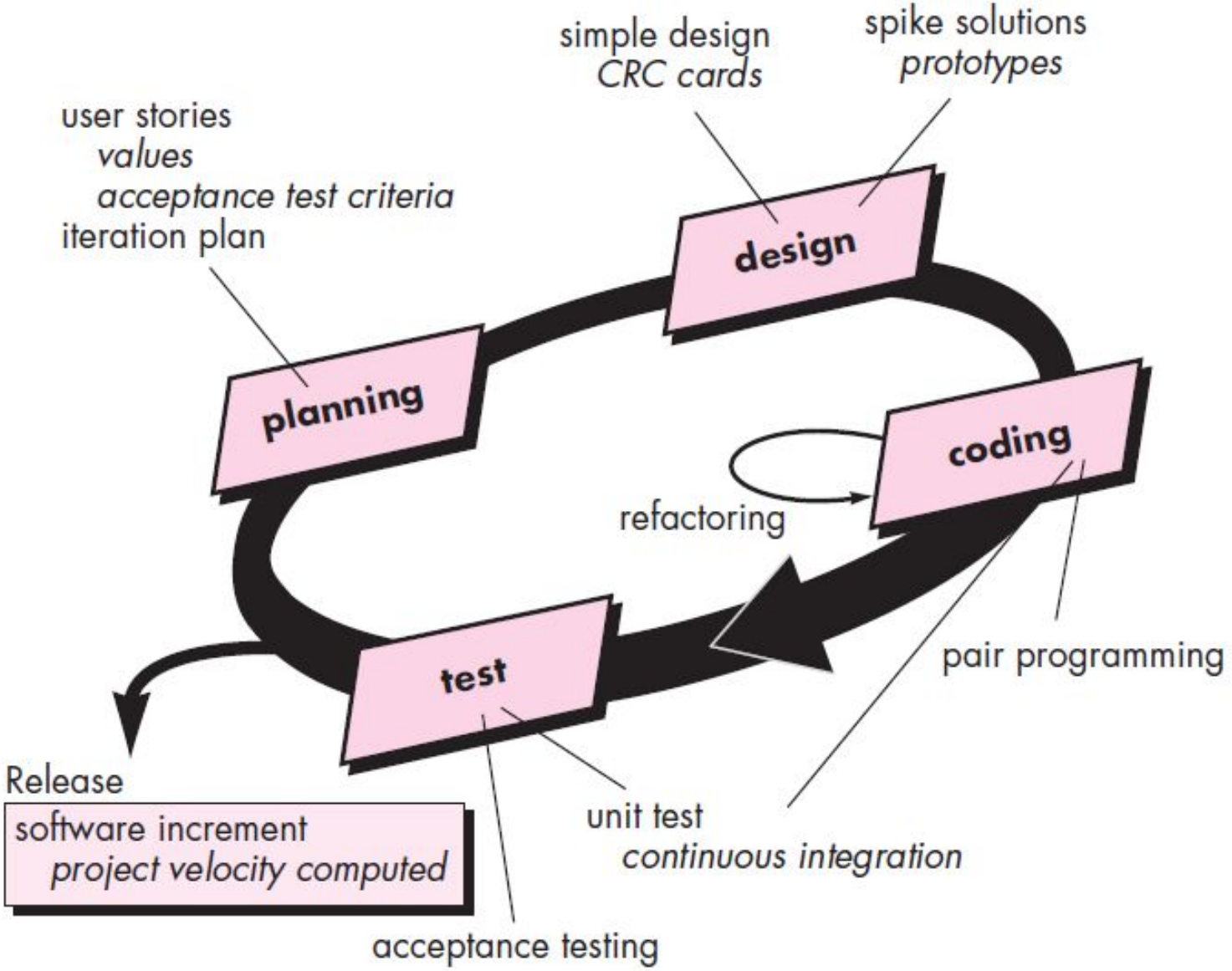
- XP Design
  - Follows the KIS principle
  - Encourage the use of CRC cards
  - For difficult design problems, suggests the creation of "spike solutions"—a design prototype
  - Encourages "refactoring"—an iterative refinement of the internal program design

Extreme Programming is based on the following values –

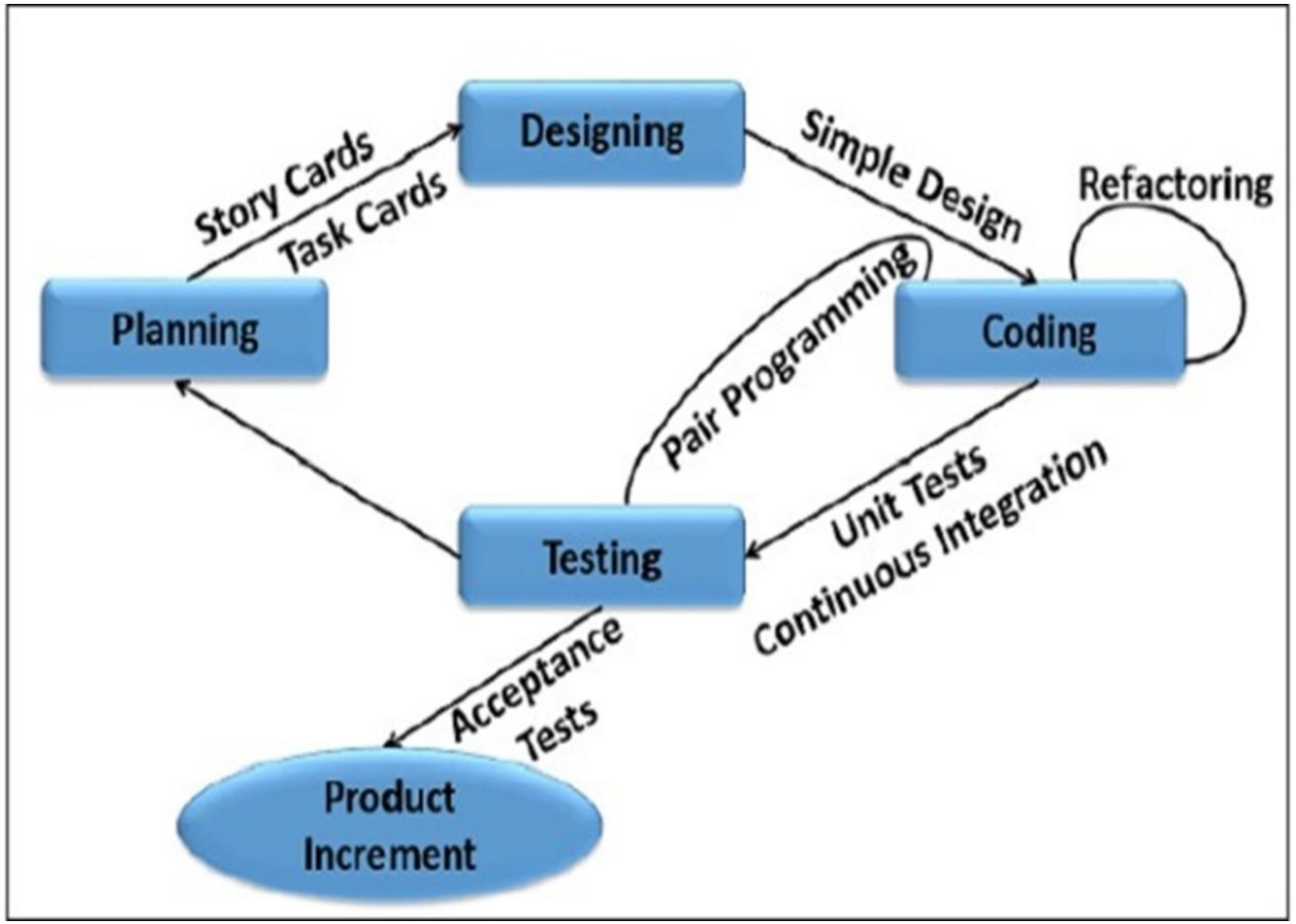
- Communication
- Simplicity
- Feedback
- Courage
- Respect

Conventional Software Process Models

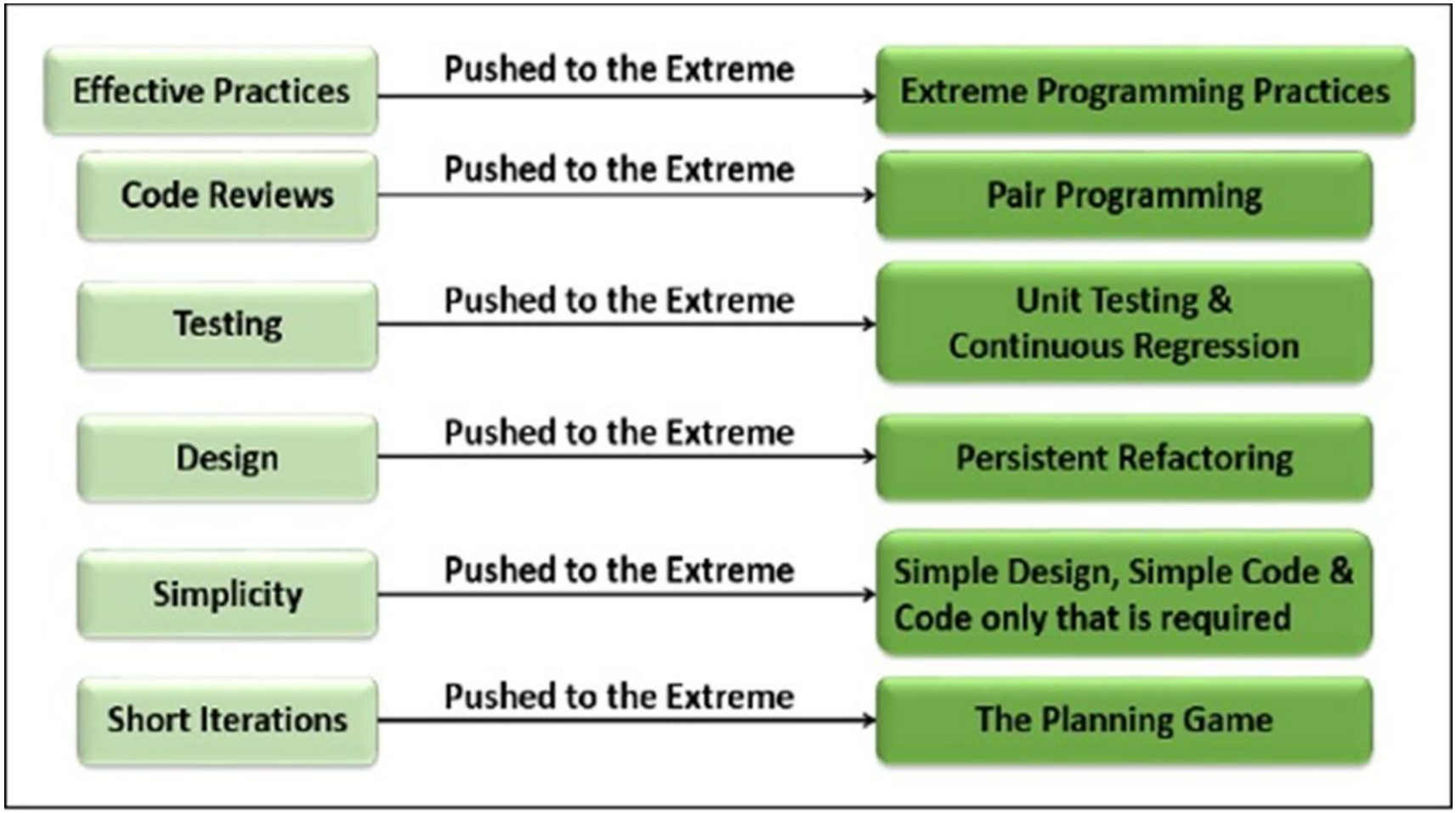
Agile – Extreme Programming (XP)



Agile – Extreme Programming (XP)



Agile – Extreme Programming (XP)



Conventional Software Process Models

Agile – Extreme Programming (XP)

- XP Coding
  - Recommends the construction of a unit test for a store before coding commences
  - Encourages “pair programming” (XP recommends that two people work together at one computer workstation to create code for a story)

- XP Testing
  - All unit tests are executed daily
  - “Acceptance tests” are defined by the customer and executed to assess customer visible functionality

Class CardReader	
Responsibilities	**Collaborators**
Tell ATM when card is inserted	ATM
Read information from card	Card
Eject card	
Retain card	

Conventional Software Process Models

Agile – Scrum

- Scrum is an agile software development method that was conceived by Jeff Sutherland and his development team in the early 1990s.
- In recent years, further development on the scrum methods has been performed by Schwaber and Beedle.
- Scrum principles are consistent with the agile manifesto and are used to guide development activities within a process that incorporates the following framework activities: Requirements, Analysis, Design, Evolution and delivery.
- Within each framework activity, work tasks occur within a process pattern called a sprint.
- The work conducted within a sprint is adapted to the problem at hand and is defined and often modified in real time by the Scrum team.
- The overall flow of the Scrum process is illustrated in figure below.
- Scrum emphasizes the use of a set of software process patterns that have proven effective for projects with tight timelines, changing requirements and business criticality.
- Each of these process patterns defines a set of development actions:

  1. Backlog – a prioritized list of project managements or features that provide business value for the customer. Items can be added to the backlog at any time. The product manager assess the backlog and updates priorities as required.

Conventional Software Process Models

Agile – Scrum

- Each of these process patterns defines a set of development actions:

  2. Sprints – consist of work units that are required to achieve a requirement defined in the backlog that must be fit into a predefined time-box (typically 30 days). Changes (e.g., backlog work items) are not introduced during the sprint. Hence, the sprint allows team members to work in a short term, but stable environment.

  3. Scrum meetings – are short (typically 15 minutes) meetings held daily by the Scrum team. Three key questions are asked and answered by all team members:

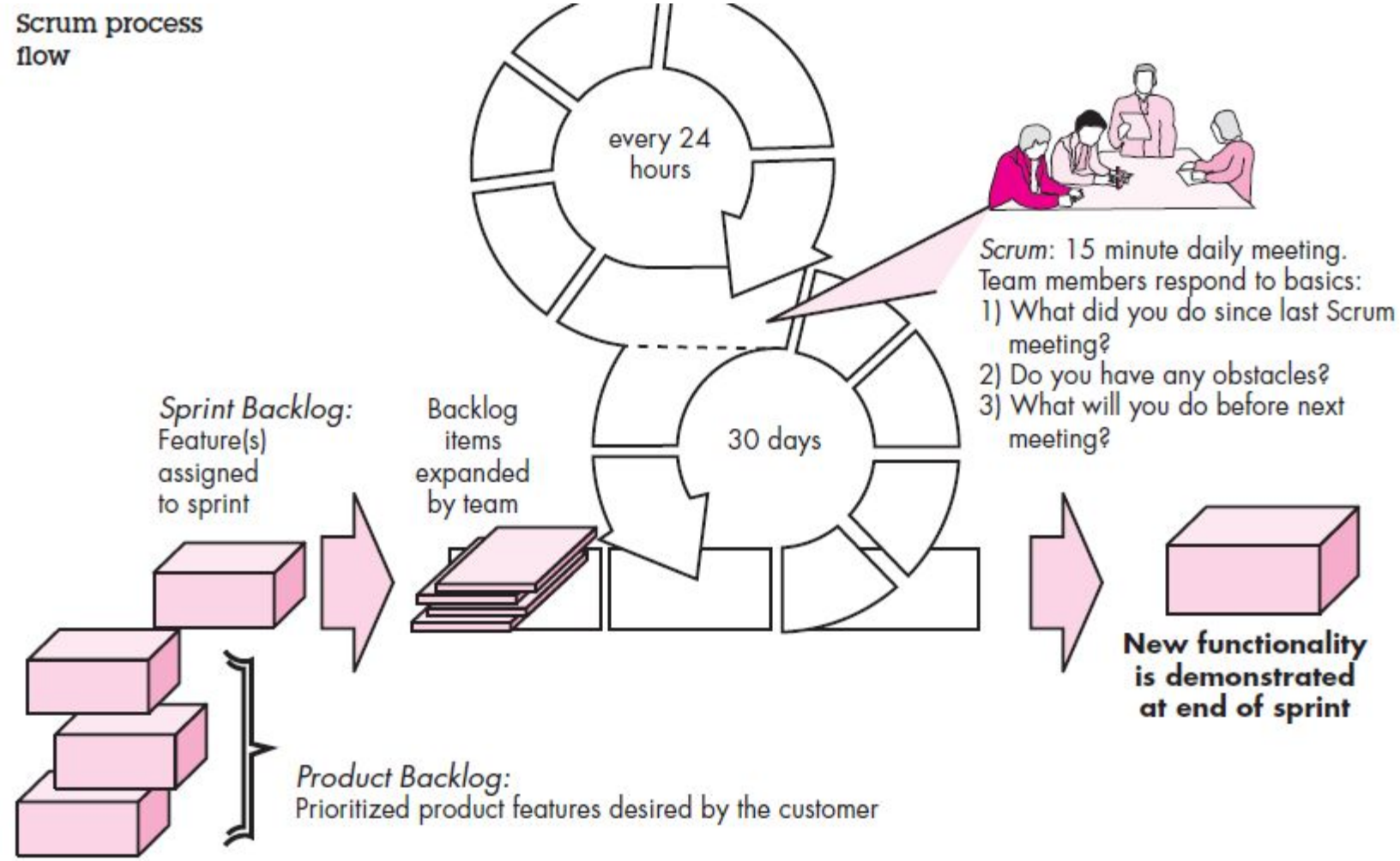
  a. What did you do since the last team meeting?

  b. What obstacles are you encountering?

  c. What do you plan to accomplish by the next team meeting?
- A team leader, called a Scrum master, leads the meeting and assesses the responses from each person.
- The Scrum meeting helps the team to uncover potential problems as early as possible.
- Also, these daily meetings lead to “knowledge socialization” and thereby promote a self-organizing team structure.

Conventional Software Process Models

Agile – Scrum



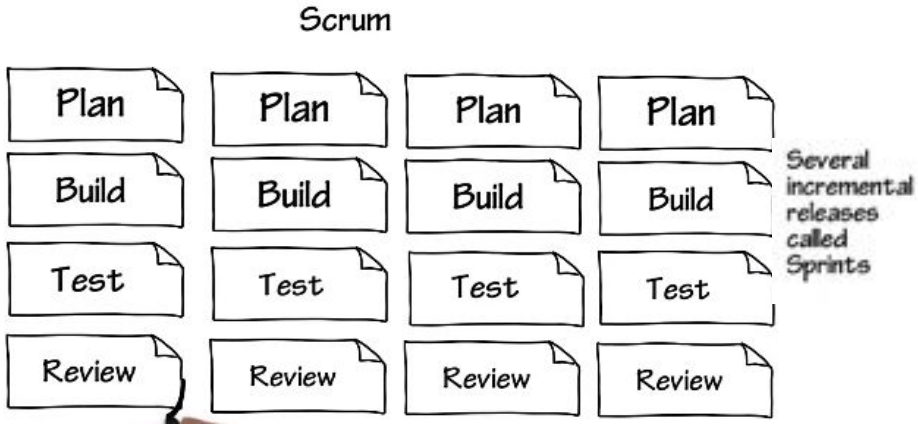
Scrum Implementation of Agile

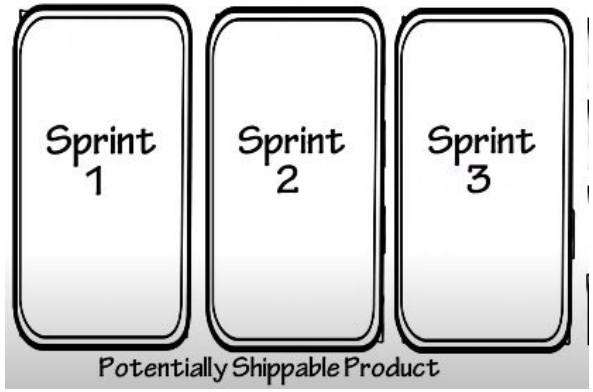
- *A Process is broken down into Smaller Pieces.*
- *Firstly, we do just enough Planning to get started with buliding with the Minimum Features*
- *We Build those Planned*
- *Next we Test & Review the small fetaure set and get it ready to Ship.*

- *When that Cycle is Completed , we end up with a Potentially Shippable Work Product*
- *This Process Usually Occurs in a Time peroid of 1- 3 weeks.*
- *This is repeated Time & Time again – Reducing the time from Planning , Development , Testing & Review*

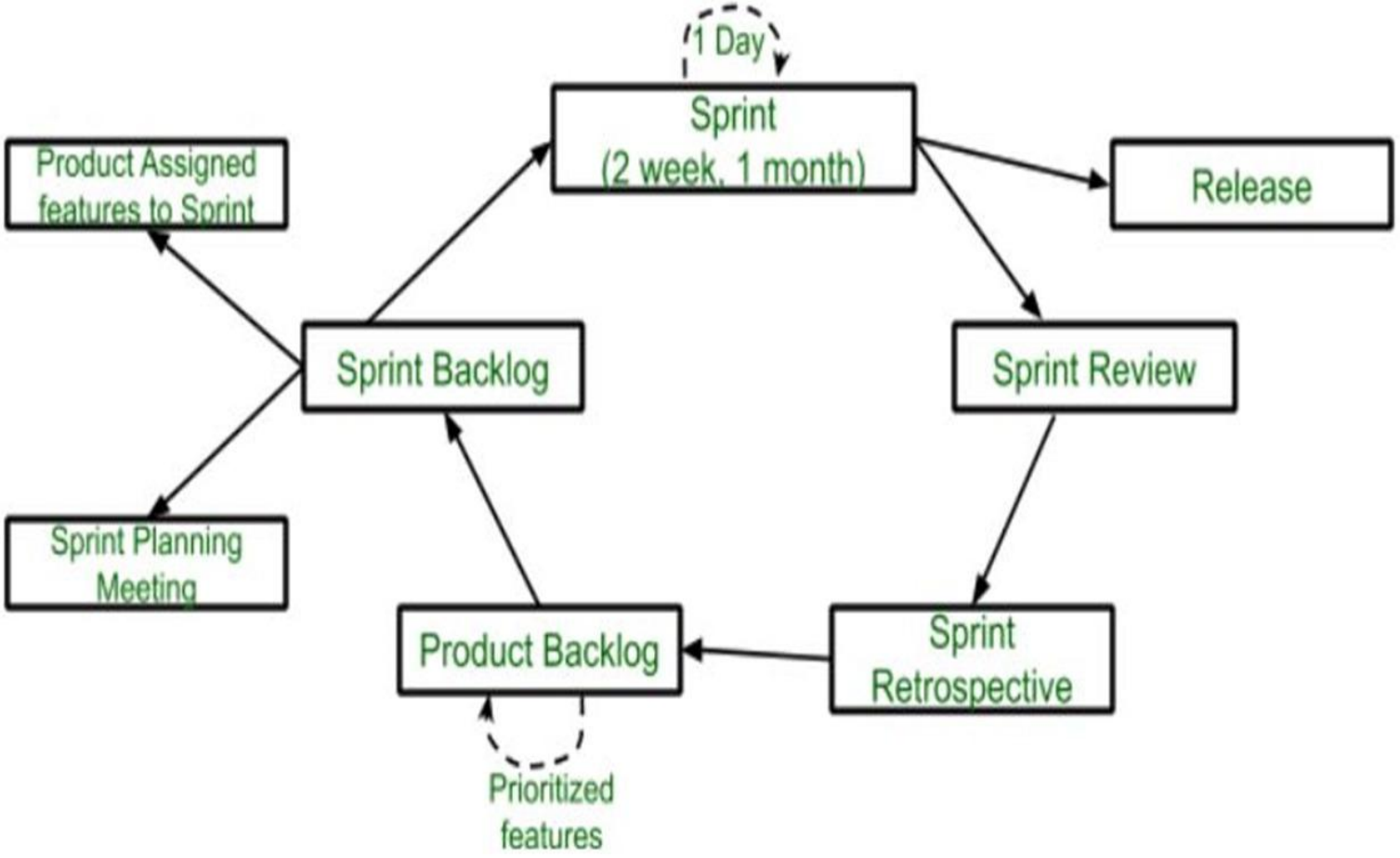
- *We try to reduce the time between the Incremental Releases.*
- *We end up with several Incremenral Releases called as **Sprints**.*

- *Sprints Usually takes from 1- 3 weeks*
- *We keep Repeating these Sprints Until our Product is Feature Complete*
- *Sometime we may endup shipping the Product with Second Sprint / Third Sprint / Fourth Sprint... But we Always endup with Shipping an Working Product*





Agile – Scrum


1 Day
Sprint
(2 week, 1 month)
Release
Product Assigned
features to Sprint
Sprint Backlog
Sprint Review
Sprint Planning
Meeting
Product Backlog
Sprint
Retrospective
Prioritized
features

- **Sprint:**
  A Sprint is a time-box of one month or less. A new Sprint starts immediately after the completion of the previous Sprint.
- **Release:**
  When the product is completed then it goes to the Release stage.
- **Sprint Review:**
  If the product still have some non-achievable features then it will be checked in this stage and then the product is passed to the Sprint Retrospective stage.
- **Sprint Retrospective:**
  In this stage quality or status of the product is checked.
- **Product Backlog:**
  According to the prioritize features the product is organized.
- **Sprint Backlog:**
  Sprint Backlog is divided into two parts Product assigned features to sprint and Sprint planning meeting.

Conventional Software Process Models

Agile – Scrum

- Demos – deliver the software increment to the customer so that functionality that has been implemented can be demonstrated and evaluated by the customer.
- It is important to note that the demo may not contain all planned functionality, but rather those functions that can be delivered within the time-box that was established.
- Beedle and his colleagues present a comprehensive discussion of these patterns in which they state: “Scrum assumes up-front the existence of chaos…”
- The Scrum process patterns enable a software team to work successfully in a world where the elimination of uncertainty is impossible.

Scrum has been used by:

- Microsoft
- Yahoo
- Google
- Electronic Arts
- Lockheed Martin
- Philips
- Siemens
- Nokia
- IBM
- Capital One
- BBC
- Intuit
- Nielsen Media
- First American Real Estate
- BMC Software
- Ipswitch
- John Deere
- Lexis Nexis
- Sabre
- Salesforce.com
- Time Warner
- Turner Broadcasting
- Oce

Introduction to Requirement Engineering

What is a requirement?

- A condition or capability needed by a user to solve a problem or achieve an objective.
- A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
- It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.
- A requirement should not be just a single word nor a whole story.
- A good business requirement should contain a specific structure

Introduction to Requirement Engineering

Functional and Non functional Requirements

- **User must enter login credential to do transaction-functional requirement**
- **Website should withstand 500 simultaneous users to the bank website without web server going down or performance degrade-non functional requirement**

Types of requirement

- **Functional Requirement(Needs)**
- **Non Functional Requirement(Wants)**

- Functional requirements are those which are related to the technical functionality of the system.
- Non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system in particular conditions, rather than specific behaviors.

Introduction to Requirement Engineering

Documenting the Functional Requirement

Example: - Withdraw Cash from ATM

R1: withdraw cash

Description: The withdraw cash function first determines the type of account that the user has and the account number from which the user wishes to withdraw cash. It checks the balance to determine whether the requested amount is available in the account. If enough balance is available, it outputs the required cash; otherwise it generates an error message.

R1.1 select withdraw amount option

Input: "withdraw amount" option

Output: user prompted to enter the account type

R1.2: select account type

Input: user option

Output: prompt to enter amount

R1.3: get required amount

Input: amount to be withdrawn in integer values greater than 100 and less than 10,000 in multiples of 100.

Output: The requested cash and printed transaction statement.

Processing: the amount is debited from the user's account if sufficient balance is available, otherwise an error message displayed.

Introduction to Requirement Engineering

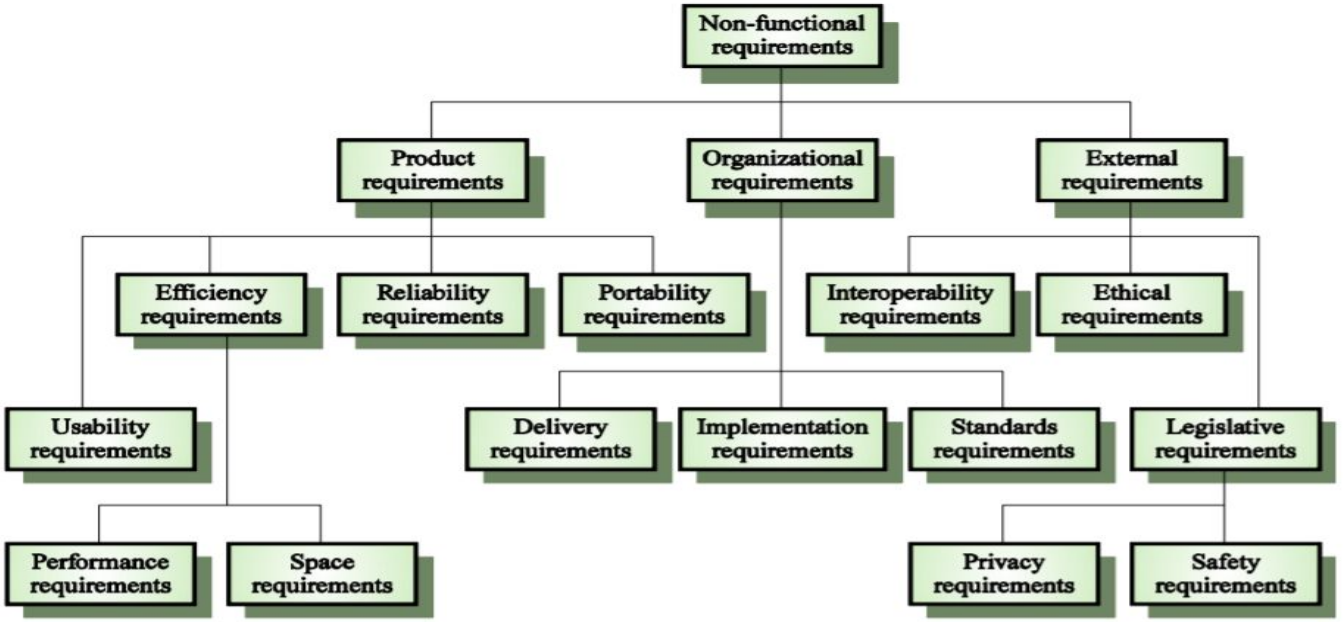
Advantages of Functional Requirement

- Helps you to check whether the application is providing all the functionalities that were mentioned in the functional requirement of that application
- A functional requirement document helps you to define the functionality of a system or one of its subsystems.
- Functional requirements along with requirement analysis help identify missing requirements. They help clearly define the expected system service and behavior.
- Errors caught in the Functional requirement gathering stage are the cheapest to fix.
- Support user goals, tasks, or activities for easy project management
- Functional requirement can be expressed in Use Case form or user story as they exhibit externally visible functional behavior.

Introduction to Requirement Engineering

What is Non-Functional Requirement?

- A non-functional requirement defines the quality attribute of a software system.
- They represent a set of standards used to judge the specific operation of a system. Example, how fast does the website load?
- A non-functional requirement is essential to ensure the usability and effectiveness of the entire software system.
- Failing to meet non-functional requirements can result in systems that fail to satisfy user needs.



Introduction to Requirement Engineering

Product Requirement

These requirements specify how software product performs. Product requirements comprise the following.

- **Usability requirements:** Describe the ease with which users are able to operate the software. For example, the software should be able to provide access to functionality with fewer keystrokes and mouse clicks.
- **Efficiency requirements:** Describe the extent to which the software makes optimal use of resources, the speed with which the system executes, and the memory it consumes for its operation. For example, the system should be able to operate at least three times faster than the existing system.
- **Reliability requirements:** Describe the acceptable failure rate of the software. For example, the software should be able to operate even if a hazard occurs.
- **Portability requirements:** Describe the ease with which the software can be transferred from one platform to another. For example, it should be easy to port the software to a different operating system without the need to redesign the entire software.

Organizational Requirement

These requirements are derived from the policies and procedures of an organization. Organizational requirements comprise the following.

- **Delivery requirements:** Specify when the software and its documentation are to be delivered to the user.
- **Implementation requirements:** Describe requirements such as programming language and design method.
- **Standards requirements:** Describe the process standards to be used during software development. For example, the software should be developed using standards specified by the ISO and IEEE standards.

Introduction to Requirement Engineering

External Requirement

These requirements include all the requirements that affect the software or its development process externally. External requirements comprise the following.

- **Interoperability requirements:** Define the way in which different computer based systems will interact with each other in one or more organizations.
- **Ethical requirements:** Specify the rules and regulations of the software so that they are acceptable to users.
- **Legislative requirements:** Ensure that the software operates within the legal jurisdiction. For example, pirated software should not be sold.

Examples of Non-functional requirements

- A website should be capable enough to handle 20 million users with affecting its performance
- The software should be portable. So moving from one OS to other OS does not create any problem.
- Privacy of information, the export of restricted technologies, intellectual property rights, etc. should be audited.

Introduction to Requirement Engineering

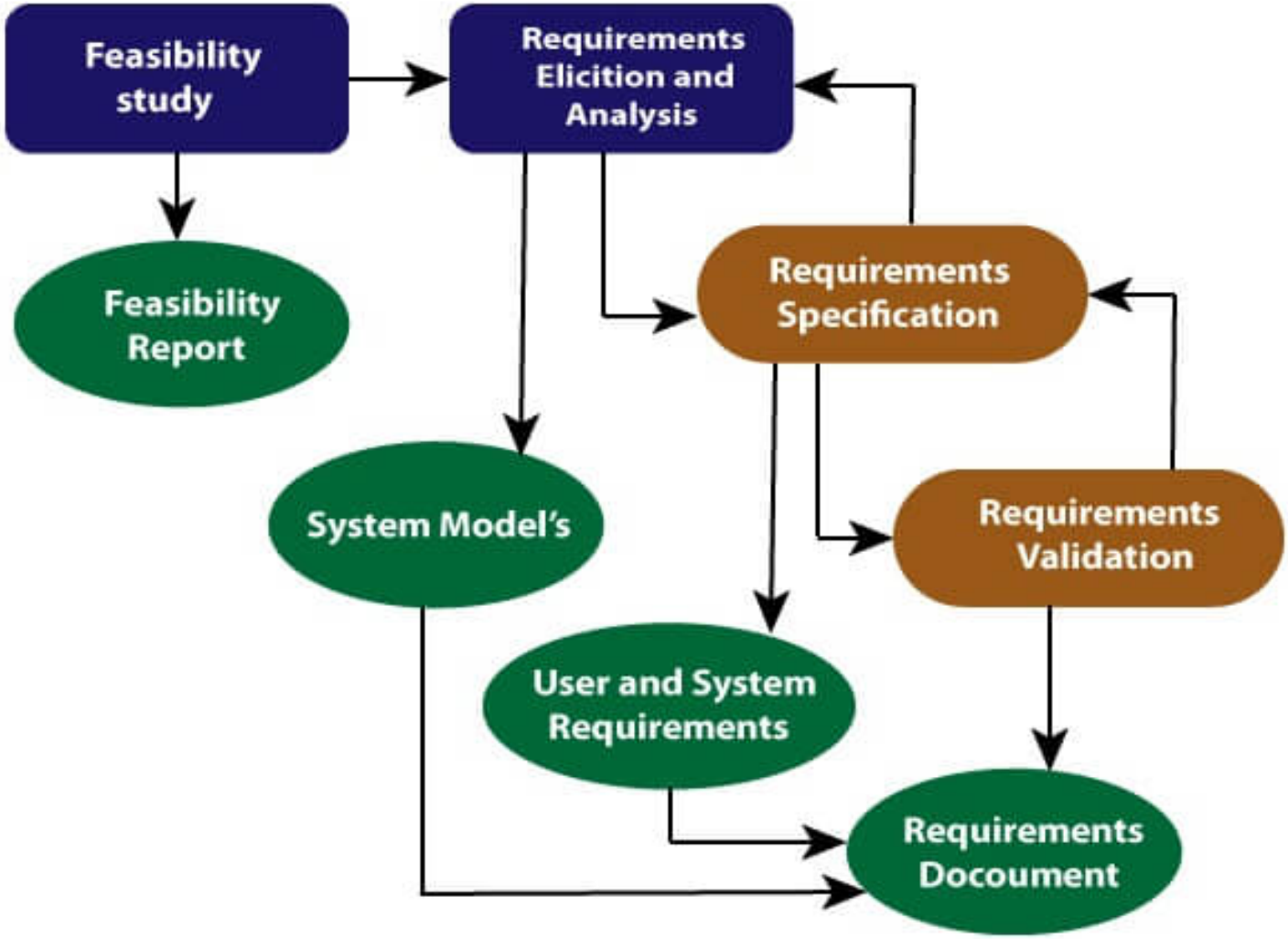
Advantages of Non-Functional Requirement

- The nonfunctional requirements ensure the software system follow legal and compliance rules.
- They ensure the reliability, availability, and performance of the software system
- They ensure good user experience and ease of operating the software.
- They help in formulating security policy of the software system.

Functional vs. Non-Functional Requirements

Functional Requirements	Non-functional Requirements
• **Products** The system **shall display** a list of all products offered by the shop. *MustHave* The system **shall organise** the list of products by product category. *MustHave* The system **shall display** detailed product descriptions consisting of name, photograph, price and text of description on demand. *MustHave* The system shall allow the items in the catalogue to be **searched**. *ShouldHave.* The system **shall display** the number of items currently in the shopping basket on each page of the catalogue. *CouldHave* • **Payment** The system **shall accept** all major credit cards. *MustHave* The system **shall validate** payment with the credit card processing company. *MustHave*	• **Capacity** The system shall support 1000 transactions per day. *ShouldHave* The system shall support a peak transaction rate of 10 transactions per second. *ShouldHave* The system shall support 5000 concurrent sessions. *MustHave* • **Availability** The system shall be available 24 hours per day, 360 days per year. *MustHave* The system shall not lose any transaction data. *MustHave* The system shall accept payment and raise an order within 5 seconds in 95% of the cases. *ShouldHave* The system shall log in a customer within 5 seconds. *ShouldHave*

The requirements engineering process



Requirement Engineering Process

Introduction to Requirement Engineering

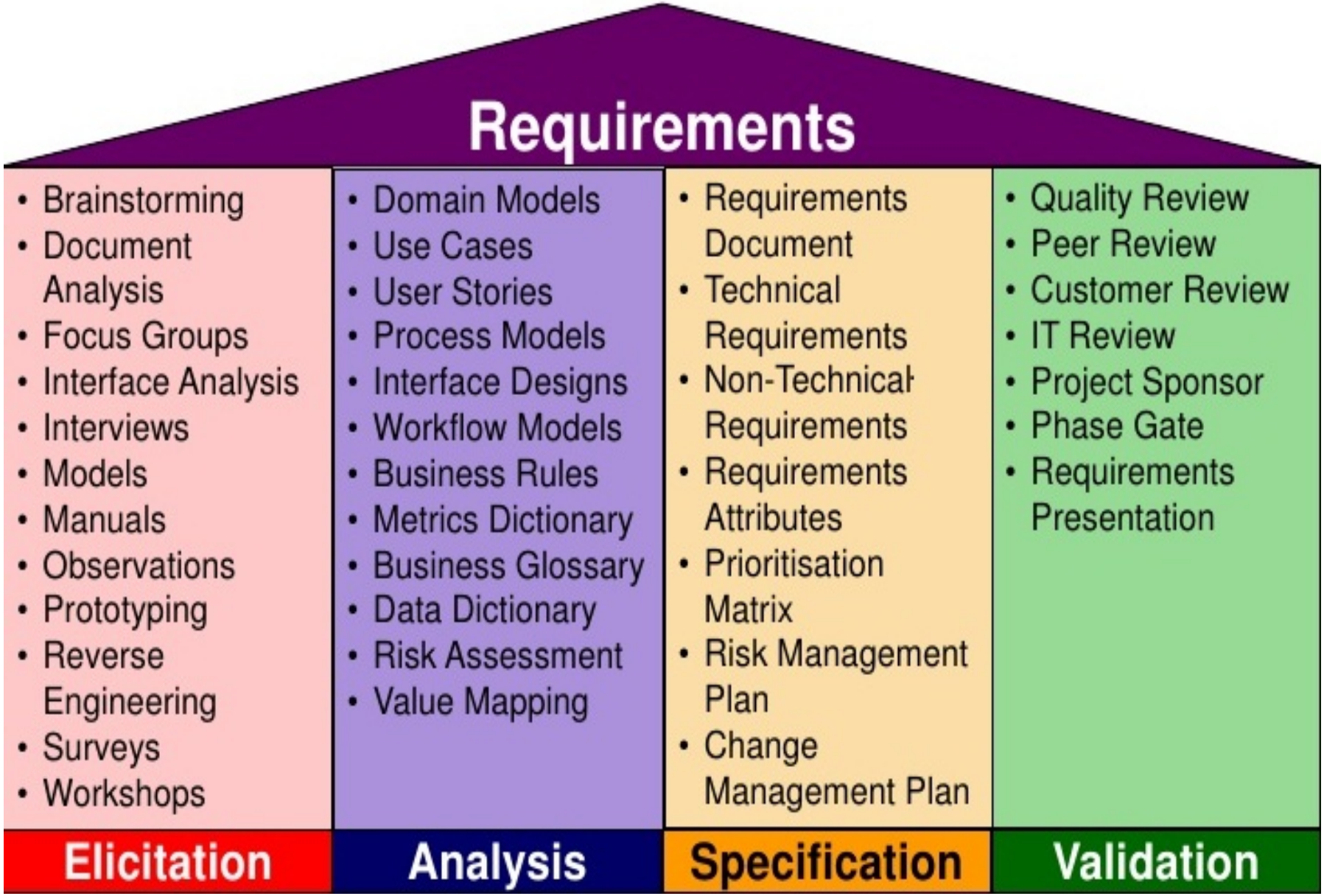
- Inception—ask a set of questions that establish …
  - Basic understanding of the problem
  - The people who want a solution
  - The nature of the solution that is desired, and
  - The effectiveness of preliminary communication and collaboration between the customer and the developer
- Elicitation—elicit requirements from all stakeholders
- Elaboration—create an analysis model that identifies data, function and behavioral requirements
- Negotiation—agree on a deliverable system that is realistic for developers and customers
- Specification—can be any one (or more) of the following:
  - A written document
  - A set of models
  - A formal mathematical
  - A collection of user scenarios (use-cases)
  - A prototype

Introduction to Requirement Engineering

- Validation—a review mechanism that looks for
  - errors in content or interpretation
  - areas where clarification may be required
  - missing information
  - inconsistencies (a major problem when large products or systems are engineered)
  - conflicting or unrealistic (unachievable) requirements.
- Requirements management

Inception

- Identify stakeholders
  - “who else do you think I should talk to?”
- Recognize multiple points of view
- Work toward collaboration
- The first questions
  - Who is behind the request for this work?
  - Who will use the solution?
  - What will be the economic benefit of a successful solution?
  - Is there another source for the solution that you need?


Requirements
• Brainstorming
• Document Analysis
• Focus Groups
• Interface Analysis
• Interviews
• Models
• Manuals
• Observations
• Prototyping
• Reverse Engineering
• Surveys
• Workshops
Elicitation
• Domain Models
• Use Cases
• User Stories
• Process Models
• Interface Designs
• Workflow Models
• Business Rules
• Metrics Dictionary
• Business Glossary
• Data Dictionary
• Risk Assessment
• Value Mapping
Analysis
• Requirements Document
• Technical Requirements
• Non-Technical Requirements
• Requirements Attributes
• Prioritisation Matrix
• Risk Management Plan
• Change Management Plan
Specification
• Quality Review
• Peer Review
• Customer Review
• IT Review
• Project Sponsor
• Phase Gate
• Requirements Presentation
Validation

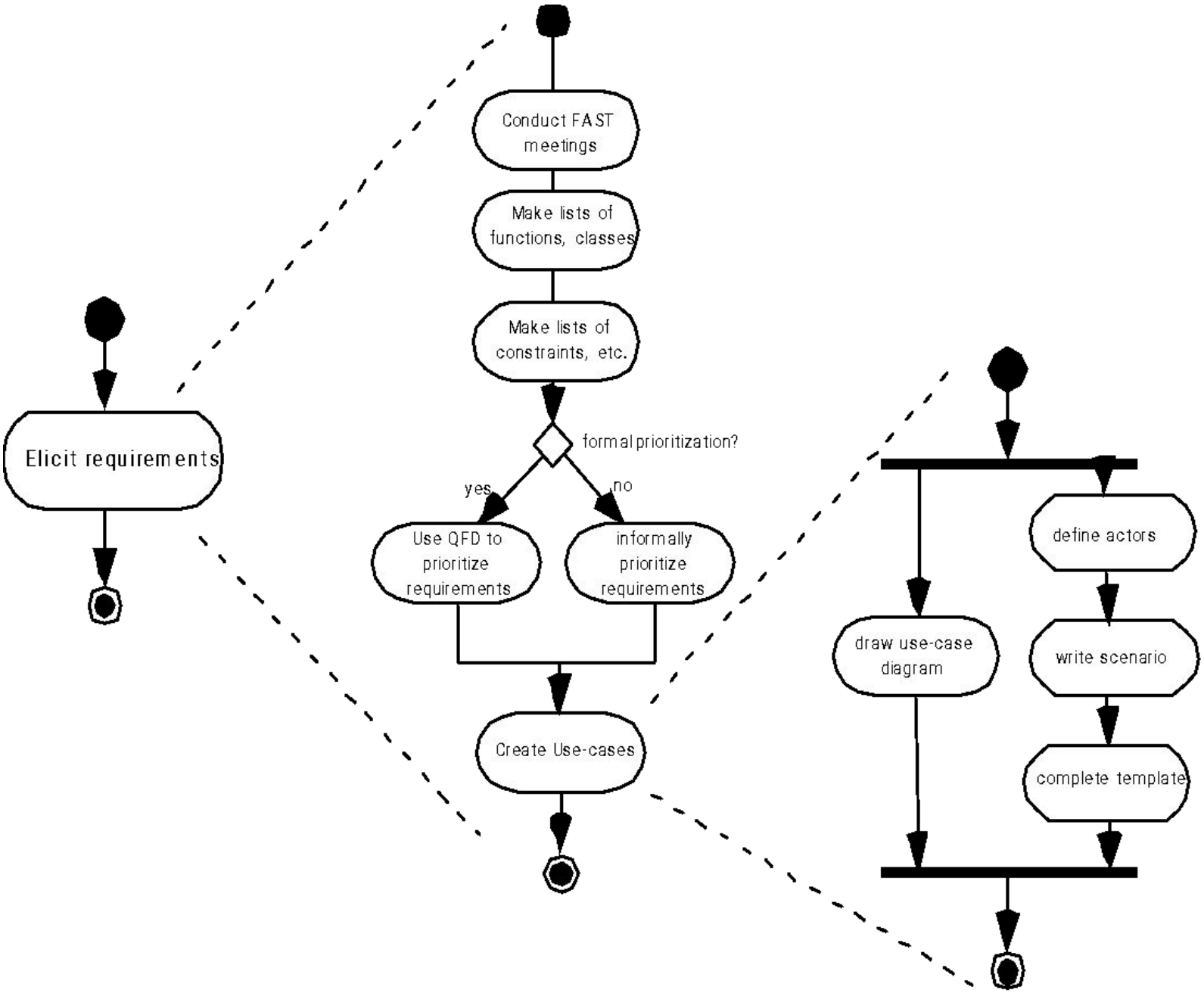
Data-Gathering Techniques[1]

Technique	Good for	Kind of data	Plus	Min
Questionnaires	Answering specific questions	Quantitative and qualitative data	Can reach many people with low resource	The design is crucial. Response rate may be low. Responses may not be what you want
Interviews	Exploring issues	Some quantitative but mostly qualitative data	Interviewer can guide interviewee. Encourages contact between developers and users	Time consuming. Artificial environment may intimidate interviewee
Focus groups and workshops	Collecting multiple viewpoints	Some quantitative but mostly qualitative data	Highlights areas of consensus and conflict. Encourages contact between developers and users	Possibility of dominant characters
Naturalistic observation	Understanding context of user activity	Qualitative	Observing actual work gives insight that other techniques cannot give	Very time consuming. Huge amounts of data
Studying documentation	Learning about procedures, regulations, and standards	Quantitative	No time commitment from users required	Day-to-day work will differ from documented procedures

Requirements Elicitation

- Meetings are conducted and attended by both software engineers and customers.
- Rules for preparation and participation are established
- An agenda is suggested
- A "facilitator" (can be a customer, a developer, or an outsider) controls the meeting.
- A "definition mechanism" (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room or virtual forum) is used.
- The goal is
  - To identify the problem
  - Propose elements of the solution
  - Negotiate different approaches, and
  - Specify a preliminary set of solution requirements

Requirements Elicitation



Quality Function Deployment

- Function deployment determines the “value” (as perceived by the customer) of each function required of the system
- Information deployment identifies data objects and events
- Task deployment examines the behavior of the system
- Value analysis determines the relative priority of requirements

Elicitation Work Products

- A statement of need and feasibility.
- A bounded statement of scope for the system or product.
- A list of customers, users, and other stakeholders who participated in requirements elicitation.
- A description of the system's technical environment.
- A list of requirements (preferably organized by function) and the domain constraints that apply to each.
- A set of usage scenarios that provide insight into the use of the system or product under different operating conditions.
- Any prototypes developed to better define requirements.

Elicitation Techniques

Elicitation techniques

- Stakeholder analysis
- Analysis of existing systems or documentation, background reading
- Task observation, ethnography
- Questionnaires
- Interviewing
- Brainstorming
- Joint Application Design (JAD)
- Prototyping
- Pilot system
- Use cases and scenarios
- Risk analysis

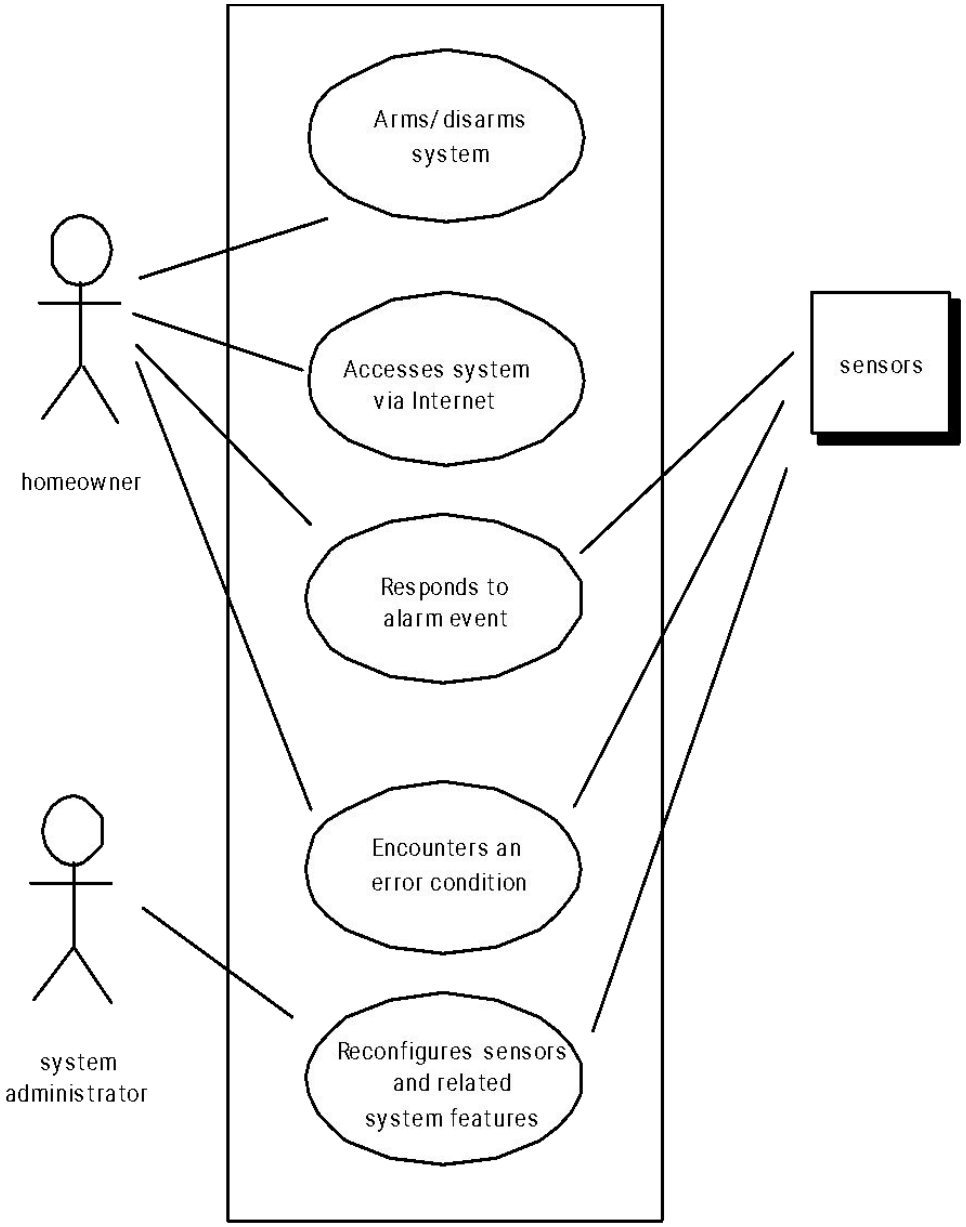
Building Analysis Model

- Elements of the analysis model
  - Scenario-based elements
    - Functional—processing narratives for software functions
    - Use-case—descriptions of the interaction between an “actor” and the system
  - Class-based elements
    - Implied by scenarios
  - Behavioral elements
    - State diagram
  - Flow-oriented elements
    - Data flow diagram

Scenario-based Elements - Use Cases

- A collection of user scenarios that describe the thread of usage of a system.
- Each scenario is described from the point-of-view of an “actor”—a person or device that interacts with the software in some way.
- Each scenario answers the following questions:
  - Who is the primary actor, the secondary actor(s)?
  - What are the actor’s goals?
  - What preconditions should exist before the story begins?
  - What main tasks or functions are performed by the actor?
  - What extensions might be considered as the story is described?
  - What variations in the actor’s interaction are possible?
  - What system information will the actor acquire, produce, or change?
  - Will the actor have to inform the system about changes in the external environment?
  - What information does the actor desire from the system?
  - Does the actor wish to be informed about unexpected changes?

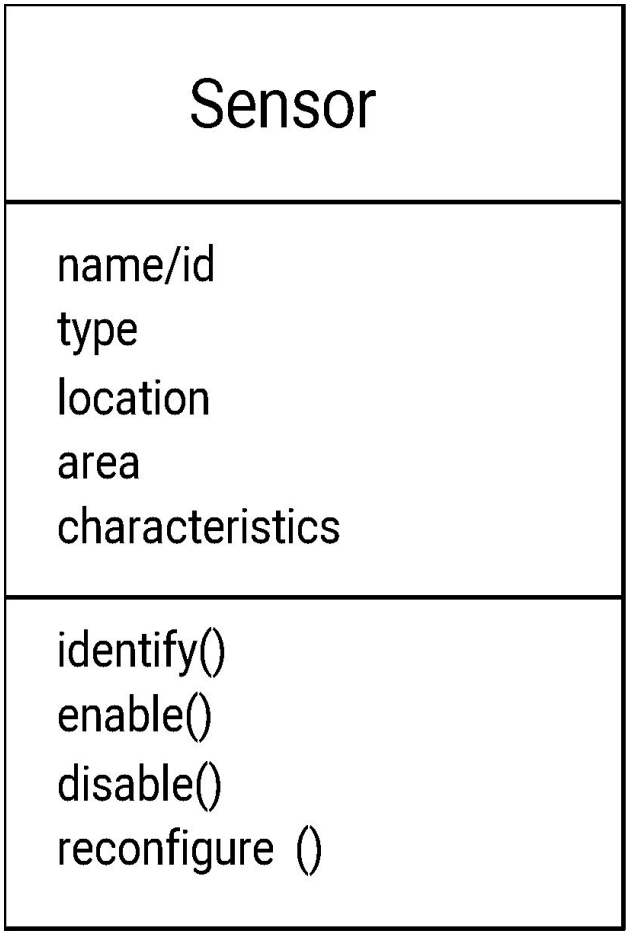
Use Case Diagram



Class-based Elements - Class Diagram

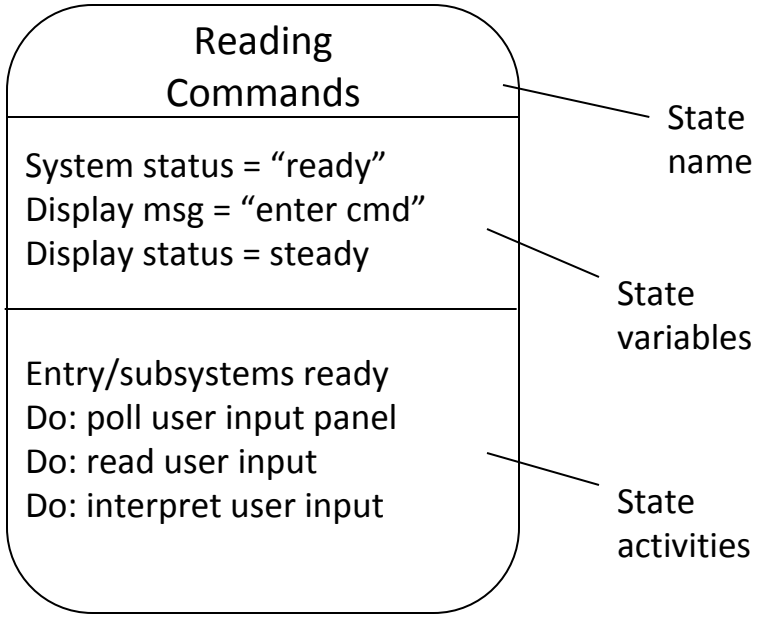
- Each usage scenario implies a set of objects that are manipulated as an actor interacts with the system.
- These objects are categorized into classes – a collection of things that have similar attributes and common behaviors.
- For example, a UML class diagram can be used to depict a Sensor class for the SafeHome security function.
- Note that the diagram lists the attributes of sensors (name, type) and the operations (e.g., identify, enable) that can be applied to modify these attributes.
- In addition to class diagrams, other analysis modeling elements depict the manner in which classes collaborate with one another and the relationships and interactions between classes.

From the *SafeHome* system …



Behavioral Elements - State Diagram

- The behavior of a computer-based system can have a profound effect on the design that is chosen and the implementation approach that is applied.
- Therefore, the requirements model must provide modeling elements that depict behavior.
- The state diagram is one method for representing the behavior of a system by depicting its states and the events that cause the system to change state.
- A state is any externally observable mode of behavior.
- In addition, the state diagram indicates actions (process activation) taken as a consequence of a particular event.
- To illustrate the use of a state diagram, consider software embedded within the SafeHome control panel that is responsible for reading user input.
- A simplified UML state diagram is shown.



Flow-Oriented Elements

- Information is transformed as it flows through a computer-based system.
- The system accepts input in a variety of forms, applies functions to transform it and produces output in a variety of forms.
- Input may be a control signal transmitted by a transducer, a series of numbers typed by a human operator, a packet of information transmitted on a network link, or a voluminous data file retrieved from secondary storage.
- The transform(s) may comprise a single logical comparison, a complex numerical algorithm, or a rule-inference approach of an expert system.
- Output may light a single LED or produce a 200-page report.
- In effect, we can create a flow model for any computer-based system, regardless of size and complexity.

Analysis Patterns

Pattern name: A descriptor that captures the essence of the pattern.

Intent: Describes what the pattern accomplishes or represents

Motivation: A scenario that illustrates how the pattern can be used to address the problem.

Forces and context: A description of external issues (forces) that can affect how the pattern is used and also the external issues that will be resolved when the pattern is applied.

Solution: A description of how the pattern is applied to solve the problem with an emphasis on structural and behavioral issues.

Consequences: Addresses what happens when the pattern is applied and what trade-offs exist during its application.

Design: Discusses how the analysis pattern can be achieved through the use of known design patterns.

Known uses: Examples of uses within actual systems.

Related patterns: One or more analysis patterns that are related to the named pattern because (1) it is commonly used with the named pattern;

(2) it is structurally similar to the named pattern;

(3) it is a variation of the named pattern.

Negotiating Requirements

- Identify the key stakeholders
  - These are the people who will be involved in the negotiation
- Determine each of the stakeholders “win conditions”
  - Win conditions are not always obvious
- Negotiate
  - Work toward a set of requirements that lead to “win-win”

Requirements Validation Techniques

- **Requirements reviews/inspections:** systematic manual analysis of the requirements.
- **Prototyping:** Using an executable model of the system to check requirements.
- **Test-case generation:** Developing tests for requirements to check testability.
- **Automated consistency analysis:** checking for the consistency of structured requirements descriptions.

Validating Requirements

- Is each requirement consistent with the overall objective for the system/product?
- Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
- Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
- Is each requirement bounded and unambiguous?
- Does each requirement have attribution? That is, is a source (generally, a specific individual) noted for each requirement?
- Do any requirements conflict with other requirements?

Validating Requirements

- Is each requirement achievable in the technical environment that will house the system or product?
- Is each requirement testable, once implemented?
- Does the requirements model properly reflect the information, function and behavior of the system to be built.
- Has the requirements model been “partitioned” in a way that exposes progressively more detailed information about the system.
- Have requirements patterns been used to simplify the requirements model.
- Have all patterns been properly validated? Are all patterns consistent with customer requirements?

Software Project Effort and Cost Estimation

- Software projects are inherently effort driven because most of work involves human effort to build the software product.
- Estimating the effort required to build the software product is difficult as the result of effort is intangible and difficult to make effort estimation in building software artifacts.
- There are many techniques like Function point analysis, wide band Delphi, COCOMO etc. for making effort estimation on software projects.
- Depending on requirement, a suitable effort estimation technique is chosen for any software project.
- Since effort estimation techniques are not foolproof, effort estimates need to be revised as the project progresses.
- Once effort estimates are made for the project, cost estimates are calculated based on the effort estimate and cost parameters like hourly salary of individual employees.
- Cost estimates are done using techniques like activity based costing or cost factor analysis.

Software Project Effort and Cost Estimation

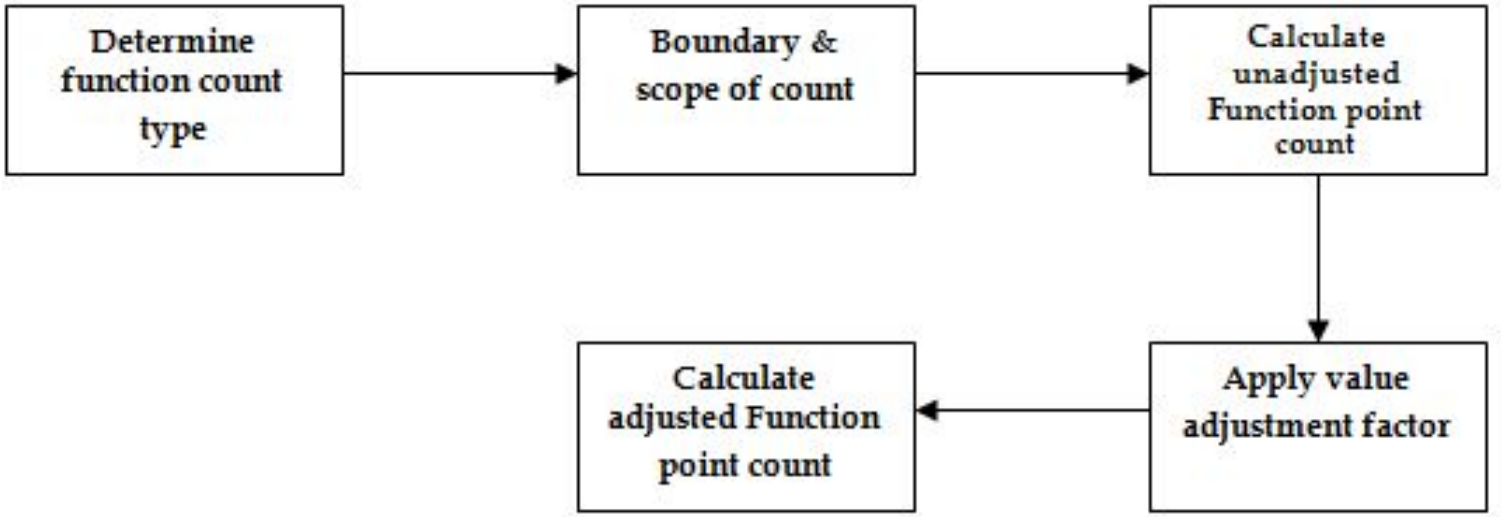
- Estimation of resources, cost, and schedule for a software engineering effort requires
  - Experience
  - Access to good historical information (metrics)
  - The courage to commit to quantitative predictions when qualitative information is all that exists
- Estimation carries inherent risk and this risk leads to uncertainty.
- Project scope must be understood
- Elaboration (decomposition) is necessary
- Historical metrics are very helpful
- At least two different techniques should be used
- Uncertainty is inherent in the process
- Estimation Techniques:
  - Past (similar) project experience
  - Conventional estimation techniques
    - Task breakdown and effort estimates
    - Size (e.g., FP) estimates
  - Empirical models
  - Automated tools

Software Project Effort and Cost Estimation

- Software projects use different kinds of software development life cycle models like waterfall model, iterative model etc.
- Effort estimation for each type of software development lifecycle model requires understanding the difference in the way the software product is built.
- In iterative models, software products are built in small incremental cycles.
- On the contrary in the waterfall model, software products are built in one go and thus all product features are fully built in the same one cycle.
- This fundamental difference necessitates a different approach to effort estimation for each type of software development lifecycle model projects.
- Software products are made manually by software engineers.
- How many of these people are needed on the project and for how long, is determined by the effort estimate and project duration.
- Resource estimation also needs to take into consideration the skill set required on the project.
- Of course, speed with which a software engineer can build a software product varies and thus this factor can affect resource estimation on the project.

Software Project Effort and Cost Estimation

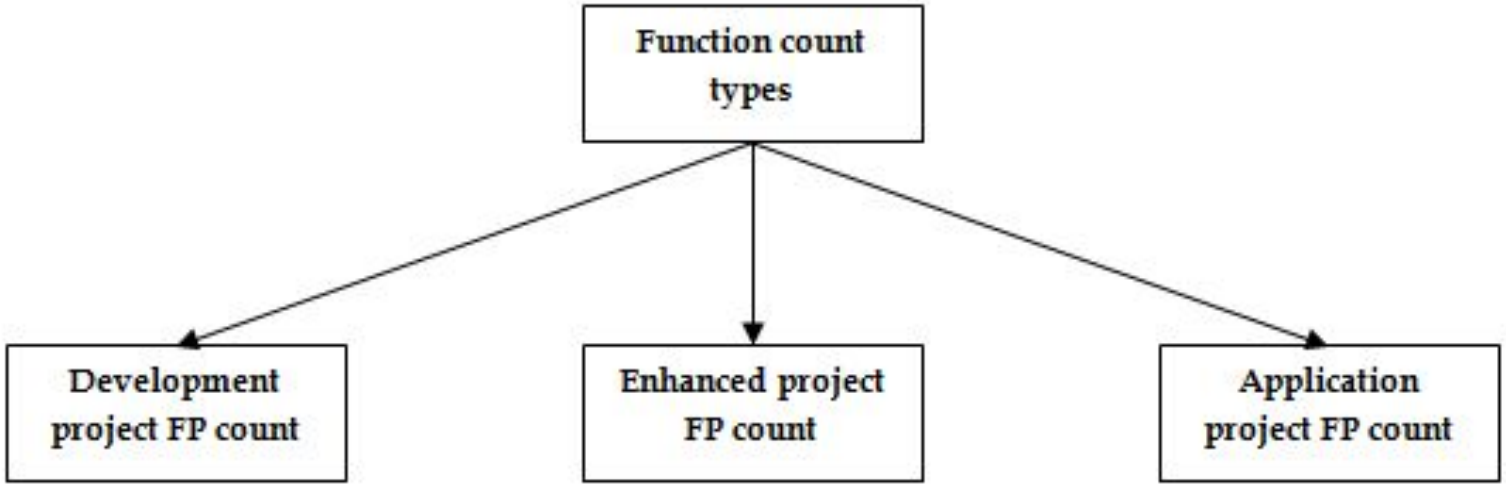
Function point count for effort estimate (function point analysis technique)

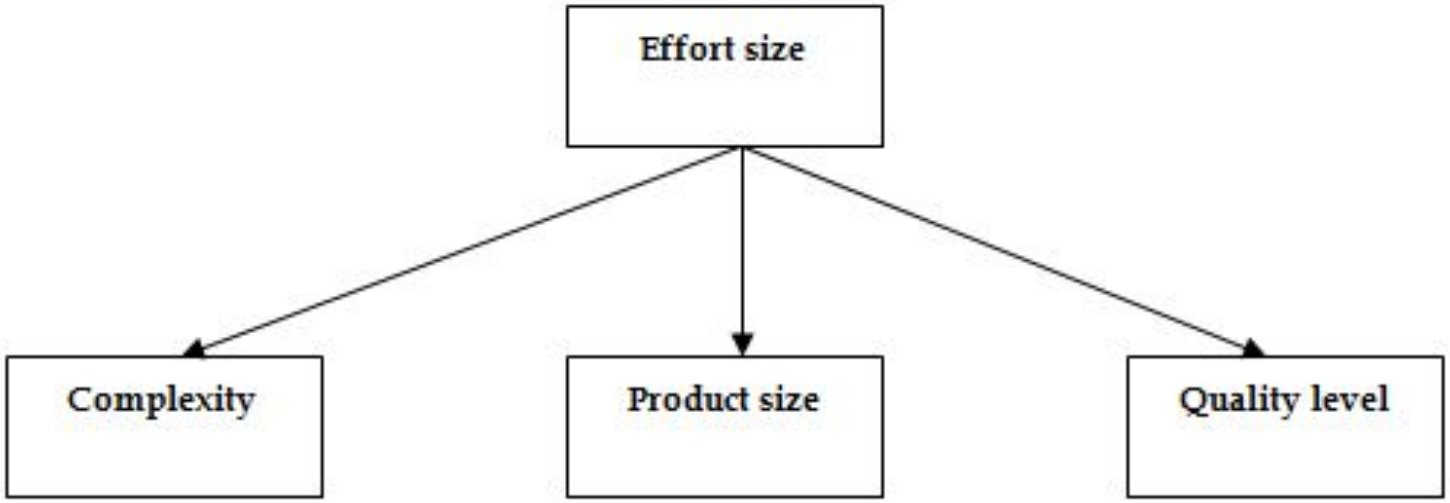


- Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and qualitative assessments of software complexity.
- The function point (FP) metric can be used effectively as a means for measuring the functionality delivered by a system.
- Using historical data, the FP metric can then be used to – a) estimate the cost or effort required to design, code and test the software, b) predict the number of errors that will be encountered during testing, c) forecast the number of components and/or the number of projected source lines in the implemented system.

Software Project Effort and Cost Estimation

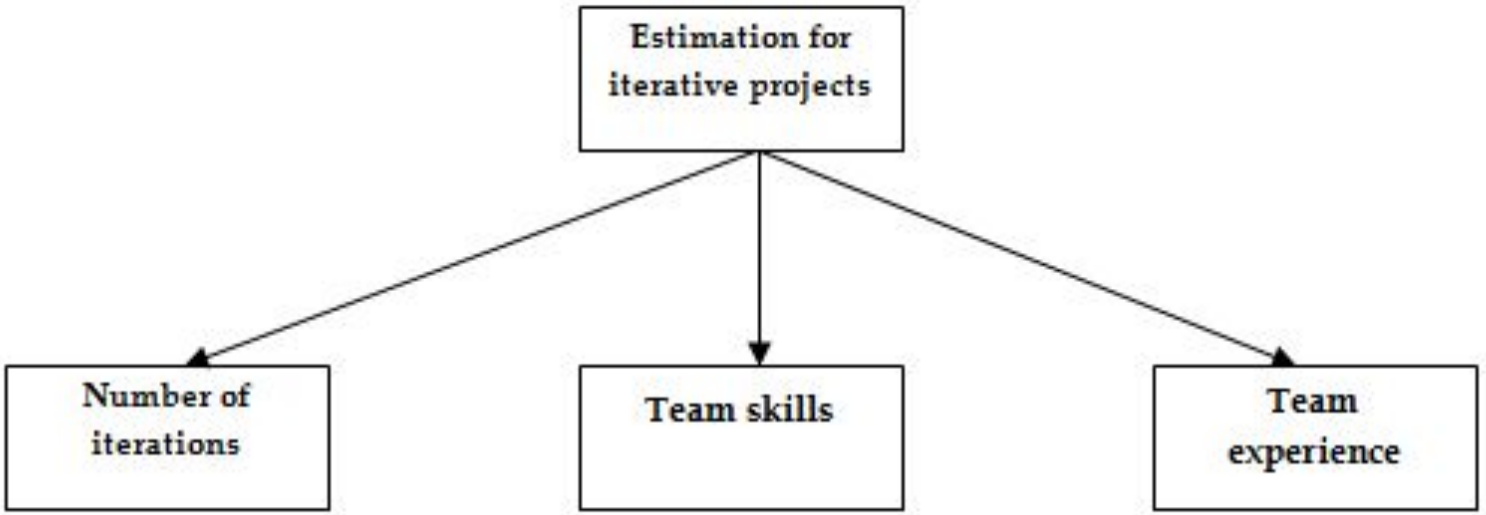
Function count type for effort estimate (function point analysis technique)





Effort sizing for effort estimate (function point analysis technique)

Software Project Effort and Cost Estimation



Effort estimate for iterative projects

Software Project Effort and Cost Estimation

Comparison of effort estimate techniques

	Project details	Estimation technique
1	Historical project data + current project data	FPA
2	Current project data	COCOMO, Wide Band Delphi
3	No data	No technique can be used

Effort and cost for various project tasks

Effort type	Hours	Costs/hour	Costs
development effort	500	70	35000
test effort	300	60	18000
project management	50	90	4500
test management	30	80	2400
Total Cost			59900

Software Pricing

- Estimates are made to discover the cost, to the developer, of producing a software system.
  - You take into account, hardware, software, travel, training and effort costs.
- There is not a simple relationship between the development cost and the price charged to the customer.
- Broader organisational, economic, political and business considerations influence the price charged.
- Factors affecting Software Pricing

Factor	Description
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.
Cost estimate uncertainty	If an organization is unsure of its cost estimate, it may increase its price by a contingency over and above its normal profit.
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. Cash flow is more important than profit in difficult economic times.

Software Pricing

- Factors affecting Software Pricing

Factor	Description
Market opportunity	A development organization 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 organization the opportunity to make a greater profit later. The experience gained may also help it develop new products.
Requirements volatility	If the requirements are likely to change, an organization may lower its price to win a contract. After the contract is awarded, high prices can be charged for changes to the requirements.

- Pricing strategies
- Under pricing
  - A company may under price a system in order to gain a contract that allows them to retain staff for future opportunities
  - A company may under price a system to gain access to a new market area
- Increased pricing
  - The price may be increased when a buyer wishes a fixed-price contract and so the seller increases the price to allow for unexpected risks

Pricing to Win

- The software is priced according to what the software developer believes the buyer is willing to pay.
- If this is less that the development costs, the software functionality may be reduced accordingly with a view to extra functionality being added in a later release.
- Additional costs may be added as the requirements change and these may be priced at a higher level to make up the shortfall in the original price.

Software Project Effort and Cost Estimation

Project Planning – Task Set

- Estimate cost and effort
  - Decompose the problem
  - Develop two or more estimates using size, function points, process tasks or use-cases
  - Reconcile the estimates

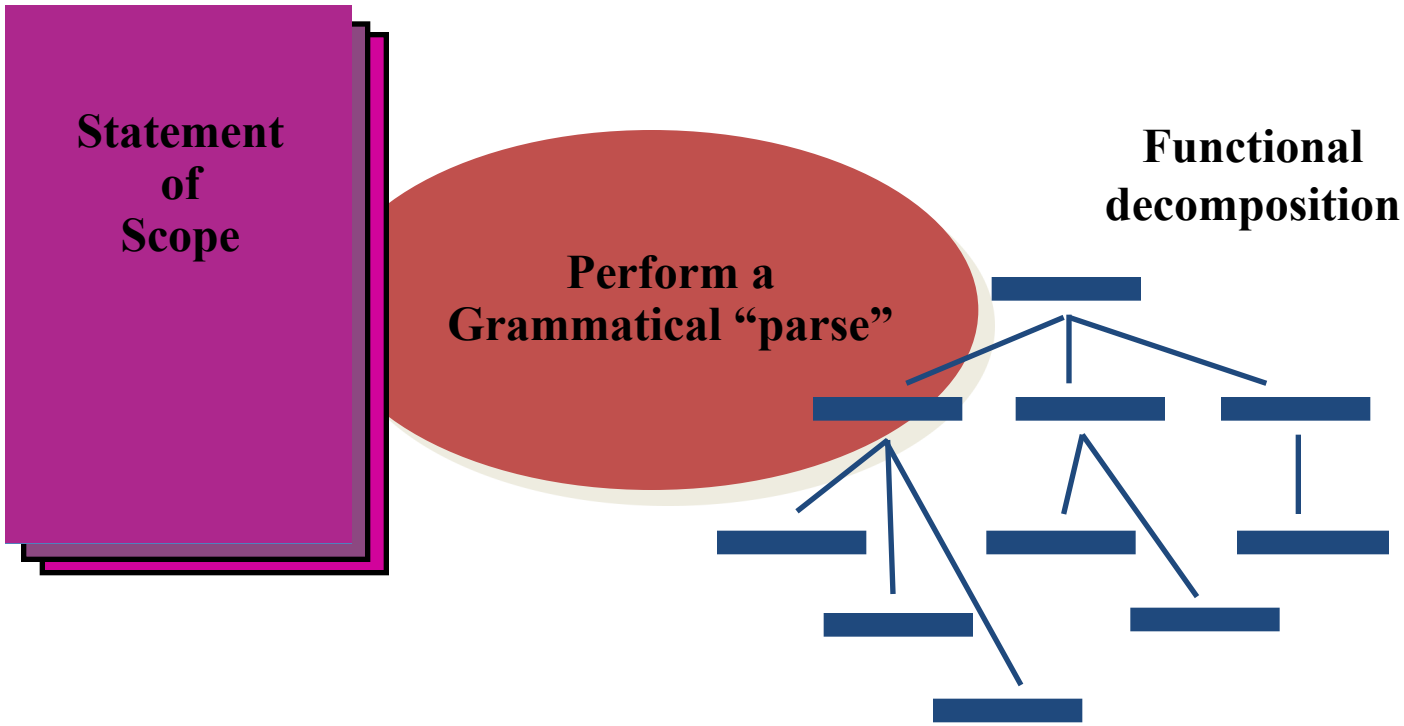
- Develop a project schedule
  - Establish a meaningful task set
  - Define a task network
  - Use scheduling tools to develop a timeline chart
  - Define schedule tracking mechanisms

Software Project Effort and Cost Estimation – Software Sizing

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

Software Project Effort and Cost Estimation - Techniques

Functional Decomposition



Software Project Effort and Cost Estimation - Techniques

Conventional Methods – LOC/FP Approach

- Compute LOC/FP using estimates of information domain values
- Use historical data to build estimates for the project
- Example – LOC Approach

Function	Estimated LOC
user interface and control facilities (UICF)	2,300
two-dimensional geometric analysis (2DGA)	5,300
three-dimensional geometric analysis (3DGA)	6,800
database management (DBM)	3,350
computer graphics display facilities (CGDF)	4,950
peripheral control (PC)	2,100
design analysis modules (DAM)	8,400
estimated lines of code	**33,200**

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

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

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

Software Project Effort and Cost Estimation - Techniques

Conventional Methods – LOC/FP Approach

Example: FP Approach

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

The estimated number of FP is derived:

$$FP_{estimated} = \text{count-total} * [0.65 + 0.01 * \sum(F_i)]$$

$$FP_{estimated} = 375$$

organizational average productivity = 6.5 FP/pm.

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

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

- Example: Compute the function point, productivity, documentation, cost per function for the following data:
- Number of user inputs = 24
- Number of user outputs = 46
- Number of inquiries = 8
- Number of files = 4
- Number of external interfaces = 2
- Effort = 36.9 p-m
- Technical documents = 265 pages
- User documents = 122 pages
- Cost = $7744/ month
- Various processing complexity factors are: 4, 1, 0, 3, 3, 5, 4, 4, 3, 3, 2, 2, 4, 5.

Weights of 5-FP Attributes

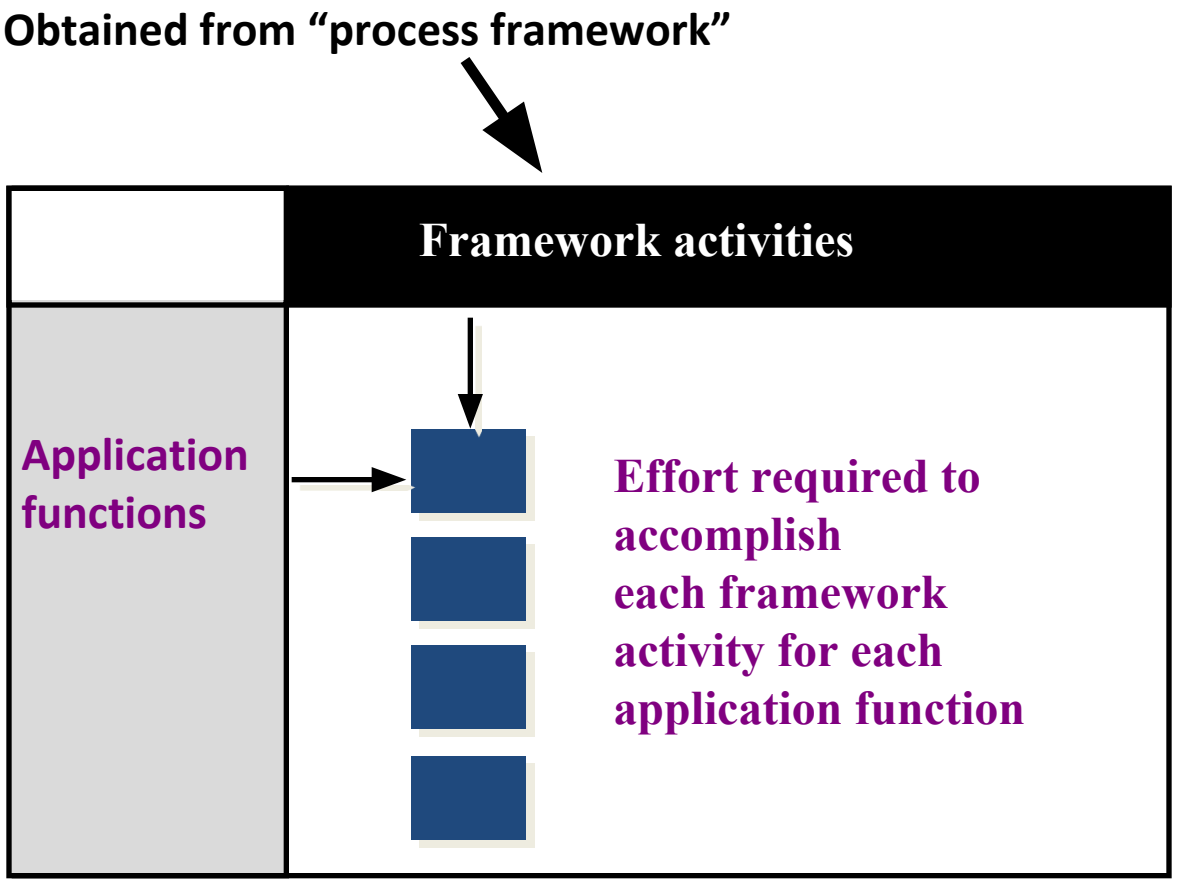
Measurement Parameter	Low	Average	High
1. Number of external inputs (EI)	7	10	15
2. Number of external outputs (EO)	5	7	10
3. Number of external inquiries (EQ)	3	4	6
4. Number of internal files (ILF)	4	5	7
5. Number of external interfaces (EIF)	3	4	6

- So sum of all fi (i ← 1 to 14) = 4 + 1 + 0 + 3 + 5 + 4 + 4 + 3 + 3 + 2 + 2 + 4 + 5 = 43
- FP = Count-total * [0.65 + 0.01 *∑(fi)]
- = 378 * [0.65 + 0.01 * 43]
- = 378 * [0.65 + 0.43]
- = 378 * 1.08 = 408
- Functional Point (FP) Analysis
- Total pages of documentation = technical document + user document
- = 265 + 122 = 387pages
- Documentation = Pages of documentation/FP
- = 387/408 = 0.94

Measurement Parameter	Count		Weighing factor
1. Number of external inputs (EI)	24	*	4 = 96
2. Number of external outputs (EO)	46	*	4 = 184
3. Number of external inquiries (EQ)	8	*	6 = 48
4. Number of internal files (ILF)	4	*	10 = 40
5. Number of external interfaces (EIF) Count-total →	2	*	5 = 10 378

Software Project Effort and Cost Estimation - Techniques

Process Based Estimation



Software Project Effort and Cost Estimation - Techniques

Process Based Estimation - Example

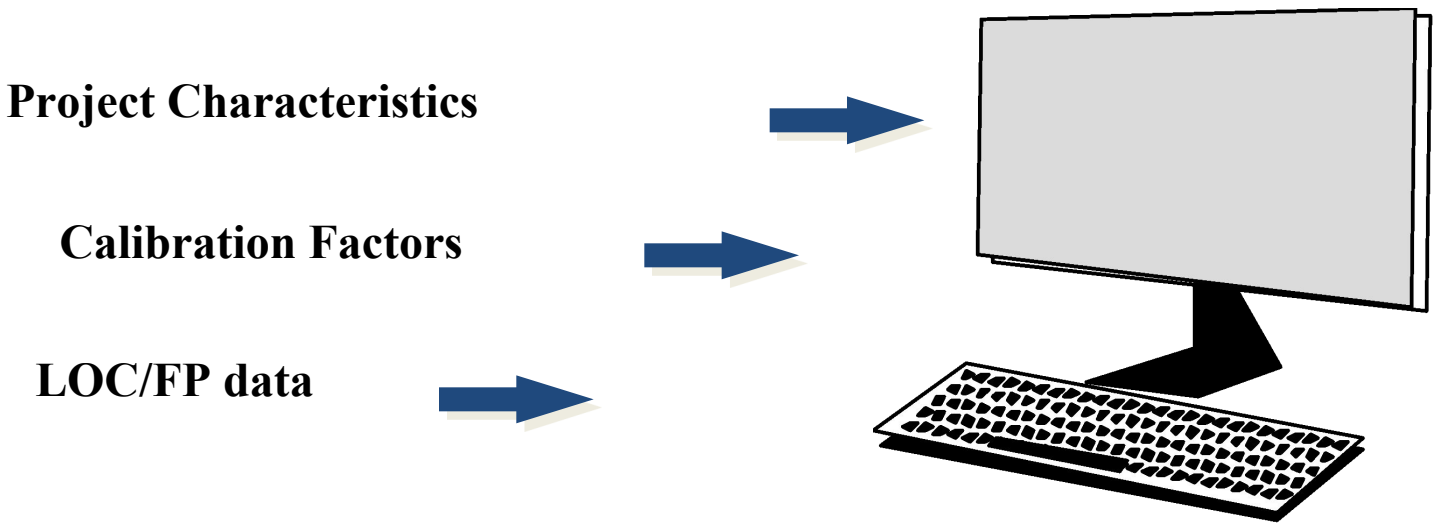
Activity →	CC	Planning	Risk Analysis	Engineering		Construction Release		CE	Totals
Task →				analysis	design	code	test		
Function									
▼									
UICF				0.50	2.50	0.40	5.00	n/a	8.40
2DGA				0.75	4.00	0.60	2.00	n/a	7.35
3DGA				0.50	4.00	1.00	3.00	n/a	8.50
CGDF				0.50	3.00	1.00	1.50	n/a	6.00
DSM				0.50	3.00	0.75	1.50	n/a	5.75
PCF				0.25	2.00	0.50	1.50	n/a	4.25
DAM				0.50	2.00	0.50	2.00	n/a	5.00
Totals	0.25	0.25	0.25	3.50	20.50	4.50	16.50		46.00
% effort	1%	1%	1%	8%	45%	10%	36%		

CC = customer communication CE = customer evaluation

Based on an average burdened labor rate of $8,000 per month, **the total estimated project cost is $368,000 and the estimated effort is 46 person-months.**

Software Project Effort and Cost Estimation - Techniques

Tool Based Estimation



Software Project Effort and Cost Estimation - Techniques

Estimation with Use Cases

	use cases	scenarios	pages	scenarios	pages	LOC	LOC estimate
User interface subsystem	6	10	6	12	5	560	3,366
Engineering subsystem group	10	20	8	16	8	3100	31,233
Infrastructure subsystem group	5	6	5	10	6	1650	7,970
Total LOC estimate							42,568

n =30 percent

$$\text{LOC estimate} = N \times \text{LOC}_{\text{avg}} + [(S_a/S_h - 1) + (P_a/P_h - 1)] \times \text{LOC}_{\text{adjust}} \qquad (26.2)$$

where

N = actual number of use cases
LOC_{avg} = historical average LOC per use case for this type of subsystem
$\text{LOC}_{\text{adjust}}$ = represents an adjustment based on n percent of LOC_{avg} where n is defined locally and represents the difference between this project and "average" projects
S_a = actual scenarios per use case
S_h = average scenarios per use case for this type of subsystem
P_a = actual pages per use case
P_h = average pages per use case for this type of subsystem

Using 620 LOC/pm as the average productivity for systems of this type and a burdened labor rate of $8000 per month, the cost per line of code is approximately $13. Based on the use-case estimate and the historical productivity data, **the total estimated project cost is $552,000 and the estimated effort is 68 person-months.**

Software Project Effort and Cost Estimation - Problems

- Estimated LOC count is 56,100 . Assuming that your organization produces 450 LOC/pm with a burdened labor rate of $7000 per person-month, find the cost /LOC, total estimated project cost and estimated effort in person months.

To Compute:

- Cost per LOC = Labor rate per month/LOC per pm
- Total Estimated Project Cost = Estimated LOC * Cost per LOC
- Estimated Effort in pm = Total Estimated Project Cost/ Labor rate per month

COCOMO I

- The constructive cost model was developed by Barry W. Boehm in the late 1970s and published in Boehm's 1981 as a model for estimating effort, cost and schedule for software projects.
- Basic model - It estimates the software in a rough and quick manner.
- Mostly used in small and medium sized projects.
- 3 modes of development:

  a) Organic,

  b) Semi Detached,

  c) Embedded

	Organic	Semi Detached	Embedded
Size	2-50 KLOC	50-300 KLOC	300 KLOC or above
Team size	Small size	Medium Size	Large size
Developer Experience	Experienced developers needed	Average experienced developers	Very little experienced people
Environment	Familiar Environment	Less Familiar Environment	Significant Environment (almost new)
Innovation	Little	Middle	Major
Deadline	Not Tight	Medium	Tight deadlines
Examples	Payroll system	Utility Systems (compiler)	Air System

COCOMO - I

- Equation & Example

Equations

NAME	Equation	Unit
Efforts	$=a(KLOC)^b$	Persons month
Development Time	$c(effort)^d$	Months
Effort Staff Size	effort/dev time	persons
Productivity	KLOC/Effort	KLOC/PM

VALUES

Software project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

COCOMO I

Numerical

- Suppose a project was estimated to be 400 KLOC. Calculate effort and time for organic mode.
- Organic:
  Effort = a $(KLOC)^b$
  Effort = 2.4 $(400)^{1.05}$
  Effort = 1295PM
  Development Time = c $(Effort)^d$
  Development Time = 2.5 $(1295)^{0.38}$
  Development Time = 38 Months, Find Effort Staff Size, Productivity??
- **Semi-detached:**
  Effort = a $(KLOC)^b$
  Effort = 3.0 $(400)^{1.12}$
  Person Months = 2462PM, Find Development Time, Effort Staff Size, Productivity??
- **Embedded:**
  Effort = a $(KLOC)^b$
  Effort = 3.6 $(400)^{1.20}$
  Person Months = 4772PM , Find Development Time , Effort Staff Size, Productivity??

COCOMO II

- Barry Boehm introduced a hierarchy of software estimation models bearing the name COCOMO, for Constructive Cost Model.
- The original COCOMO model became one of the most widely used and discussed software cost estimation models in the industry.
- It has evolved into a more comprehensive estimation model, called COCOMO II.
- Like its predecessor, COCOMO II is actually a hierarchy of estimation models that address the following areas:
  - **Application composition model:** Used during the early stages of software engineering, when prototyping of user interfaces, consideration of software and system interaction, assessment of performance, and evaluation of technology maturity are paramount.
  - **Early design stage model:** Used once requirements have been stabilized and basic software architecture has been established.
  - **Post-architecture-stage model:** Used during the construction of the software.
- Like all estimation models for software, the COCOMO II models require sizing information.
- Three different sizing options are available as part of the model hierarchy: object points, function points and lines of source code.

COCOMO II

- The COCOMO II application composition model uses object points and is illustrated below.
- It should be noted that other, more sophisticated estimation models (using FP and KLOC) are also available as part of COCOMO II.

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

- Like function points, the object point is an indirect software measure that is computed using counts of the number of (a) screens (at the user interface), (b) reports, and (c) 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) using criteria suggested by Boehm.
- In essence, complexity is a function of the number and source of the client and server data tables that are required to generate the screen or report and the number of views or sections presented as part of the screen or report.

COCOMO II

- Once complexity is determined, the number of screens, reports and components are weighted according to the table illustrated in above figure.
- The object point count is then determined by multiplying the original number of object instances by the weighting factor in the figure and summing to obtain a total object point count.
- 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:

  NOP = (object points) * [(100 - % reuse) / 100], where NOP is defined as new object points
- To derive an estimate of effort based on the computed NOP value, a "productivity rate" must be derived.

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

COCOMO II

- The figure above presents the productivity rate,

  PROD = NOP / person-month, for different levels of developer experience and development environment maturity.
- Once the productivity rate has been determined, an estimate of project effort is computing using,

  Estimated effort = NOP / PROD
- In more advanced COCOMO II models (these models use FP and KLOC counts of the size variable), a variety of scale factors, cost drivers, and adjustment procedures are required.

COCOMO II - Problems

- Use the COCOMO II model to estimate the effort required to build software for a simple ATM that produces 12 screens, 10 reports, and will require approximately 80 software components, Percentage of reuse is 20%, Value of Prod=9. Use the application composition model with object points.

To Compute:

- Object points = screen + report + components
- NOP = Object Points * [(100 - % reuse)/100]
- Estimated Effort = NOP/PROD

Risk Management

Introduction

- A risk is a potential problem – it might happen or it might not
- Conceptual definition of risk
  - Risk concerns future happenings
  - Risk involves change in mind, opinion, actions, places, etc.
  - Risk involves choice and the uncertainty that choice entails

- Two characteristics of risk
  - Uncertainty – the risk may or may not happen, that is, there are no 100% risks. (those, instead are called constraints)
  - Loss – the risk becomes a reality and unwanted consequences or losses occur

Risk Management

Risk Categorization – Approach #1

- Project risks
  - They threaten the project plan
  - If they become real, it is likely that the project schedule will slip and that costs will increase
- Technical risks
  - They threaten the quality and timeliness of the software to be produced
  - If they become real, implementation may become difficult or impossible
- Business risks
  - They threaten the viability of the software to be built
  - If they become real, they jeopardize the project or the product
- Sub-categories of Business risks
  - Market risk – building an excellent product or system that no one really wants
  - Strategic risk – building a product that no longer fits into the overall business strategy for the company
  - Sales risk – building a product that the sales force doesn't understand how to sell
  - Management risk – losing the support of senior management due to a change in focus or a change in people
  - Budget risk – losing budgetary or personnel commitment

Risk Management

Risk Categorization – Approach #2

- Known risks
  - Those risks 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).
- Predictable risks
  - Those risks that are extrapolated from past project experience (e.g., past turnover).
- Unpredictable risks
  - Those risks that can and do occur, but are extremely difficult to identify in advance.

Risk Strategies

- Reactive risk strategies
  - "Don't worry, I'll think of something"
  - The majority of software teams and managers rely on this approach
  - Nothing is done about risks until something goes wrong
    - The team then flies into action in an attempt to correct the problem rapidly
    - (Fire fighting)
  - Crisis management is the choice of management techniques

- Proactive risk strategies
  - Steps for risk management are followed (see next slide)
  - Primary objective is to avoid risk and to have a contingency plan in place to handle unavoidable risks in a controlled and effective manner

Steps for Risk Management

- Identify possible risks; recognize what can go wrong.
- Analyze each risk to estimate the probability that it will occur and the impact (i.e., damage) that it will do if it does occur.
- Rank the risks by probability and impact

  - Impact may be negligible, marginal, critical, and catastrophic
- Develop a contingency plan to manage those risks having high probability and high impact.

Risk Identification

Background

- Risk identification is a systematic attempt to specify threats to the project plan
- By identifying known and predictable risks, the project manager takes a first step toward avoiding them when possible and controlling them when necessary
- Generic risks
  - Risks that are a potential threat to every software project
- Product-specific risks
  - Risks that can be identified only by those a with a clear understanding of the technology, the people, and the environment that is specific to the software that is to be built
  - This requires examination of the project plan and the statement of scope
  - "What special characteristics of this product may threaten our project plan?“

Risk Item Checklist

- Used as one way to identify risks
- Focuses on known and predictable risks in specific subcategories (see next slide)
- Can be organized in several ways
  - A list of characteristics relevant to each risk subcategory
  - Questionnaire that leads to an estimate on the impact of each risk
  - A list containing a set of risk component and drivers along with their probability of occurrence

Known & Predictable Risk Categories

- **Product size** – risks associated with overall size of the software to be built
- **Business impact** – risks associated with constraints imposed by management or the marketplace
- **Customer characteristics** – risks associated with sophistication of the customer and the developer's ability to communicate with the customer in a timely manner
- **Process definition** – risks associated with the degree to which the software process has been defined and is followed
- **Development environment** – risks associated with availability and quality of the tools to be used to build the project
- **Technology to be built** – risks associated with complexity of the system to be built and the "newness" of the technology in the system
- **Staff size and experience** – risks associated with overall technical and project experience of the software engineers who will do the work

Questionnaire on Project Risk

(Questions are ordered by their relative importance to project success)

1) Have top software and customer managers formally committed to support the project?
2) Are end-users enthusiastically committed to the project and the system/product to be built?
3) Are requirements fully understood by the software engineering team and its customers?
4) Have customers been involved fully in the definition of requirements?
5) Do end-users have realistic expectations?
6) Is the project scope stable?
7) Does the software engineering team have the right mix of skills?
8) Are project requirements stable?
9) Does the project team have experience with the technology to be implemented?
10) Is the number of people on the project team adequate to do the job?
11) Do all customer/user constituencies agree on the importance of the project and on the requirements for the system/product to be built?

Risk Components and Drivers

- The project manager identifies the risk drivers that affect the following risk components
  - **Performance risk** - the degree of uncertainty that the product will meet its requirements and be fit for its intended use
  - **Cost risk** - the degree of uncertainty that the project budget will be maintained
  - **Support risk** - the degree of uncertainty that the resultant software will be easy to correct, adapt, and enhance
  - **Schedule risk** - the degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time
- The impact of each risk driver on the risk component is divided into one of four impact levels
  - Negligible, marginal, critical, and catastrophic
- Risk drivers can be assessed as impossible, improbable, probable, and frequent

Risk Projection (Estimation)

- Risk projection (or estimation) attempts to rate each risk in two ways
  - The probability that the risk is real
  - The consequence of the problems associated with the risk, should it occur
- The project planner, managers, and technical staff perform four risk projection steps
- The intent of these steps is to consider risks in a manner that leads to prioritization
- Be prioritizing risks, the software team can allocate limited resources where they will have the most impact
- Risk Projection / Estimation Steps

  1. Establish a scale that reflects the perceived likelihood of a risk (e.g., 1-low, 10-high)

  2. Delineate the consequences of the risk

  3. Estimate the impact of the risk on the project and product

  4. Note the overall accuracy of the risk projection so that there will be no misunderstandings

Risk Table

- A risk table provides a project manager with a simple technique for risk projection.
- It consists of five columns
  - Risk Summary – short description of the risk
  - Risk Category – one of seven risk categories
  - Probability – estimation of risk occurrence based on group input
  - Impact – (1) catastrophic (2) critical (3) marginal (4) negligible
  - RMMM – Pointer to a paragraph in the Risk Mitigation, Monitoring, and Management Plan

Risk Summary	Risk Category	Probability	Impact (1-4)	RMMM

Developing a Risk Table

- List all risks in the first column (by way of the help of the risk item checklists)
- Mark the category of each risk
- Estimate the probability of each risk occurring
- Assess the impact of each risk based on an averaging of the four risk components to determine an overall impact value
- Sort the rows by probability and impact in descending order
- Draw a horizontal cutoff line in the table that indicates the risks that will be given further attention

Assessing Risk Impact

- Three factors affect the consequences that are likely if a risk does occur
  - Its nature – This indicates the problems that are likely if the risk occurs
  - Its scope – This combines the severity of the risk (how serious was it) with its overall distribution (how much was affected)
  - Its timing – This considers when and for how long the impact will be felt
- The overall risk exposure formula is RE = P x C
  - P = the probability of occurrence for a risk
  - C = the cost to the project should the risk actually occur
- Example
  - P = 80% probability that 18 of 60 software components will have to be developed
  - C = Total cost of developing 18 components is $25,000
  - RE = .80 x $25,000 = $20,000

Risk Mitigation, Monitoring and Management

- An effective strategy for dealing with risk must consider three issues
  - (Note: these are not mutually exclusive)
  - Risk mitigation (i.e., avoidance)
  - Risk monitoring
  - Risk management and contingency planning
- Risk mitigation (avoidance) is the primary strategy and is achieved through a plan.

  - Example: Risk of high staff turnover

Risk Mitigation, Monitoring and Management

- Strategy for Reducing Staff Turnover

❑ Meet with current staff to determine causes for turnover (e.g., poor working conditions, low pay, competitive job market)

❑ Mitigate those causes that are under our control before the project starts

❑ Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave

❑ Organize project teams so that information about each development activity is widely dispersed

❑ Define documentation standards and establish mechanisms to ensure that documents are developed in a timely manner

❑ Conduct peer reviews of all work (so that more than one person is "up to speed")

❑ Assign a backup staff member for every critical technologist

Risk Mitigation, Monitoring and Management

- During risk monitoring, the project manager monitors factors that may provide an indication of whether a risk is becoming more or less likely.
- Risk management and contingency planning assume that mitigation efforts have failed and that the risk has become a reality.
- RMMM steps incur additional project cost
  - Large projects may have identified 30 – 40 risks
- Risk is not limited to the software project itself
  - Risks can occur after the software has been delivered to the user
- Software safety and hazard analysis
  - These are software quality assurance activities that focus on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail.
  - If hazards can be identified early in the software process, software design features can be specified that will either eliminate or control potential hazards.

RMMM Plan

- The RMMM plan may be a part of the software development plan or may be a separate document.
- Once RMMM has been documented and the project has begun, the risk mitigation, and monitoring steps begin
  - Risk mitigation is a problem avoidance activity
  - Risk monitoring is a project tracking activity
- Risk monitoring has three objectives
  - To assess whether predicted risks do, in fact, occur
  - To ensure that risk aversion steps defined for the risk are being properly applied
  - To collect information that can be used for future risk analysis
- The findings from risk monitoring may allow the project manager to ascertain what risks caused which problems throughout the project.

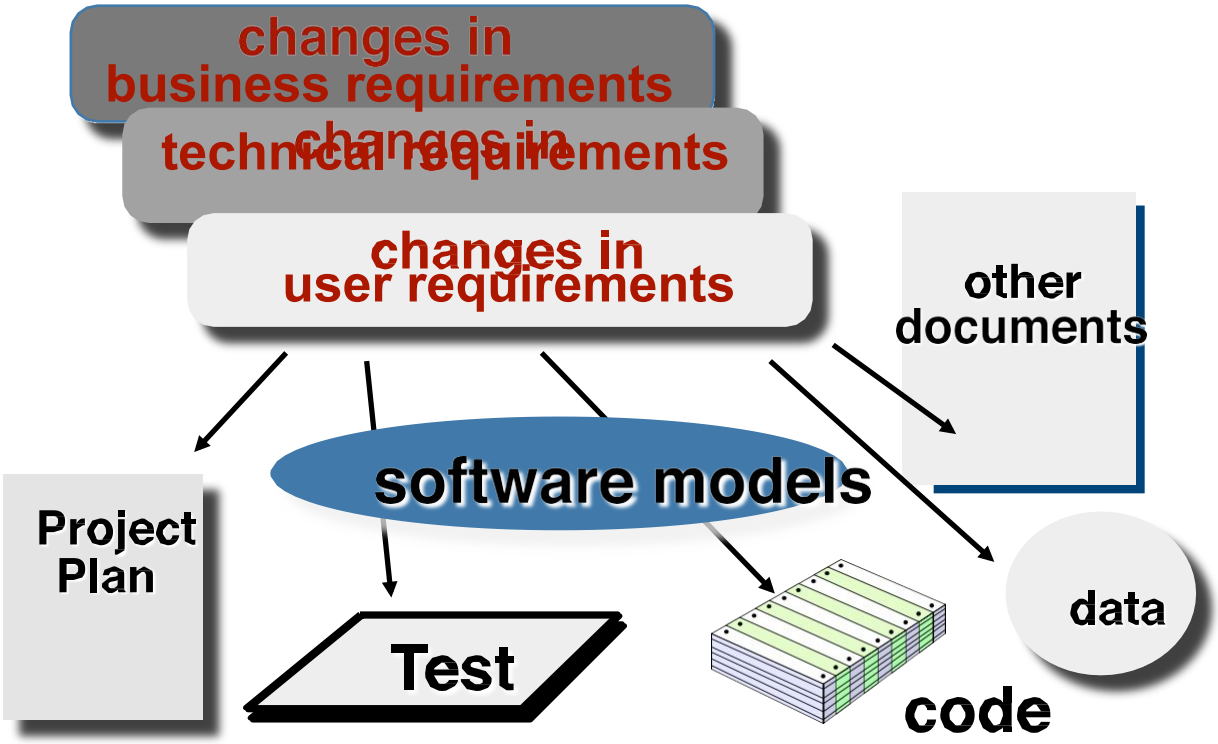
Seven Principles of Risk Management

- Maintain a global perspective
  - View software risks within the context of a system and the business problem that is intended to solve
- Take a forward-looking view
  - Think about risks that may arise in the future; establish contingency plans
- Encourage open communication
  - Encourage all stakeholders and users to point out risks at any time
- Integrate risk management
  - Integrate the consideration of risk into the software process
- Emphasize a continuous process of risk management
  - Modify identified risks as more becomes known and add new risks as better insight is achieved
- Develop a shared product vision
  - A shared vision by all stakeholders facilitates better risk identification and assessment.
- Encourage teamwork when managing risk
  - Pool the skills and experience of all stakeholders when conducting risk management activities

Configuration Management

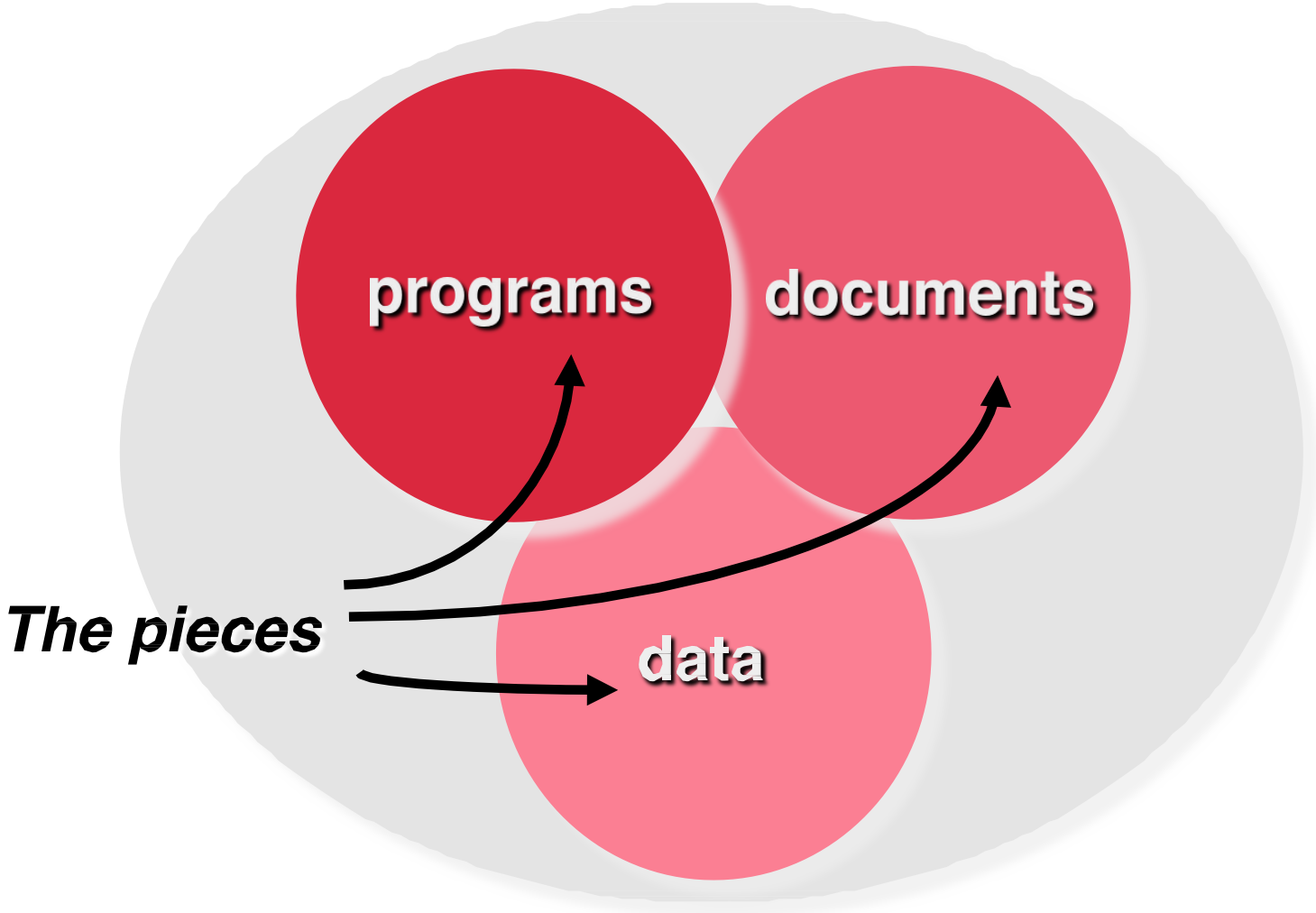
Introduction

- The First Law: No matter where you are in the system lifecycle, the system will change and the desire to change it will persist throughout the lifecycle.
- What are these changes?



Configuration Management

Software Configuration



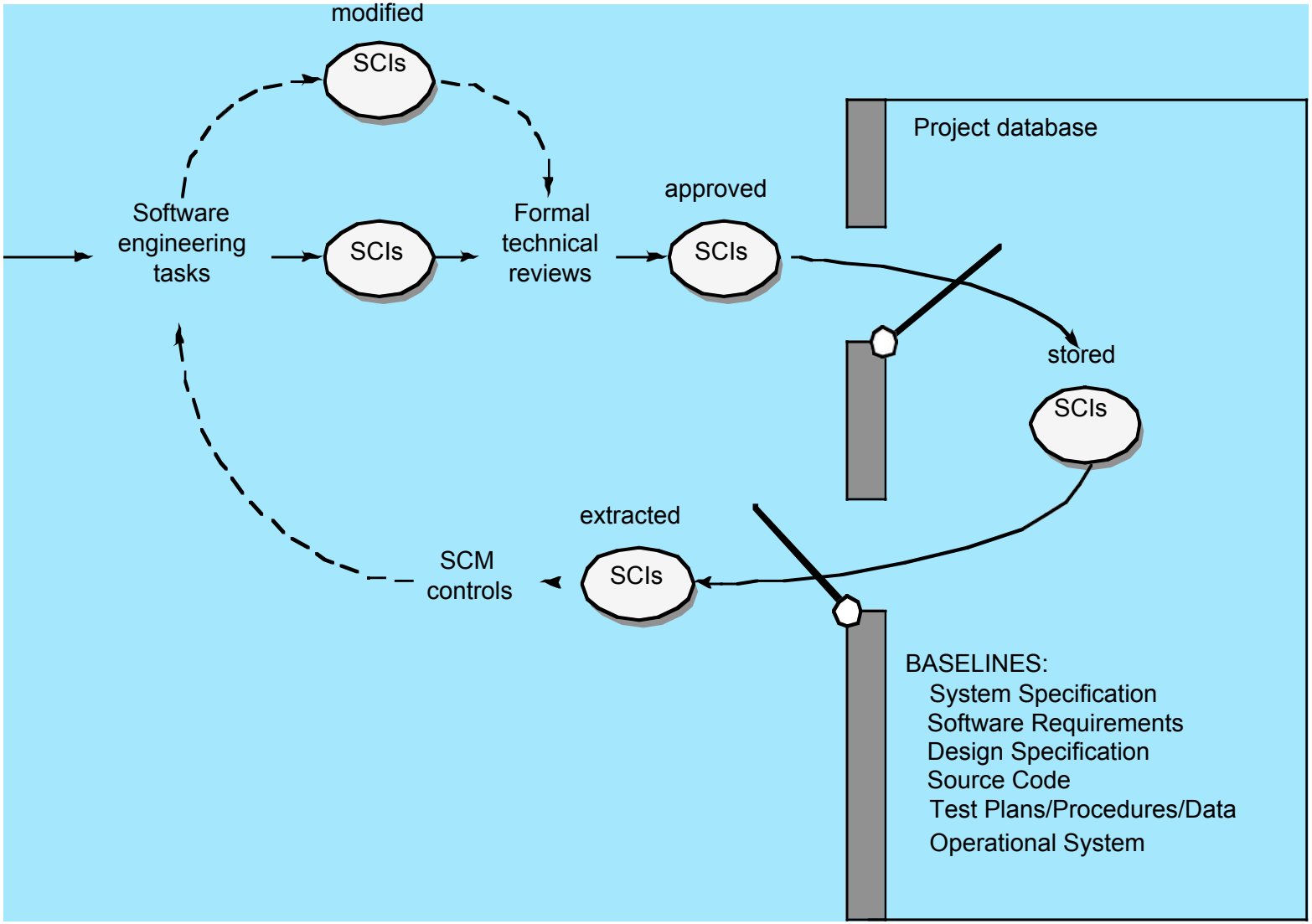
Configuration Management

Baselines

- The IEEE(IEEE Std. No. 610.12-1990) defines a baseline as:
  - A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures.
- A baseline is a milestone in the development of software that is marked by the delivery of one or more software configuration items and the approval of these SCIs that is obtained through a formal technical review.

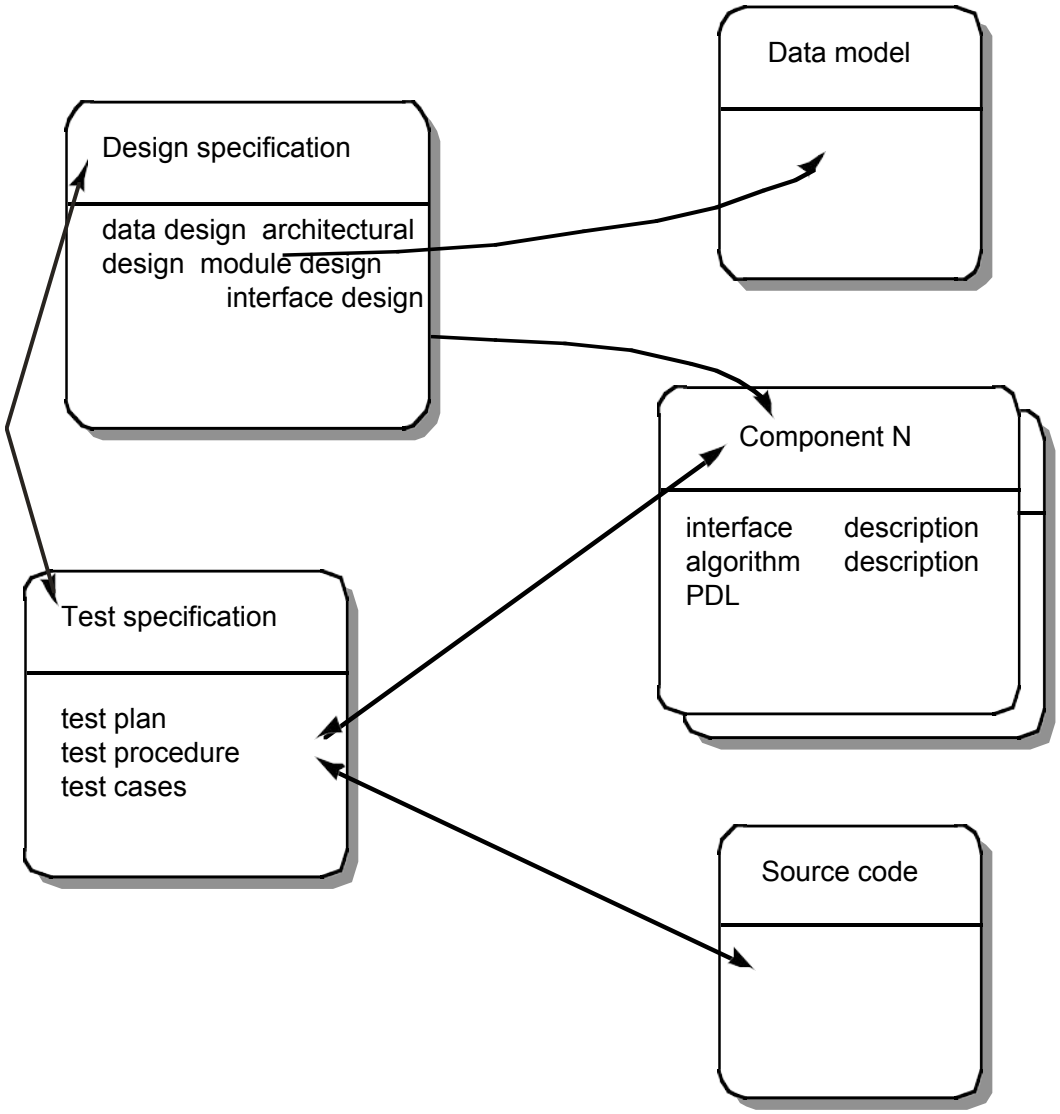
Configuration Management

Baselines



Configuration Management

Software Configuration Objects



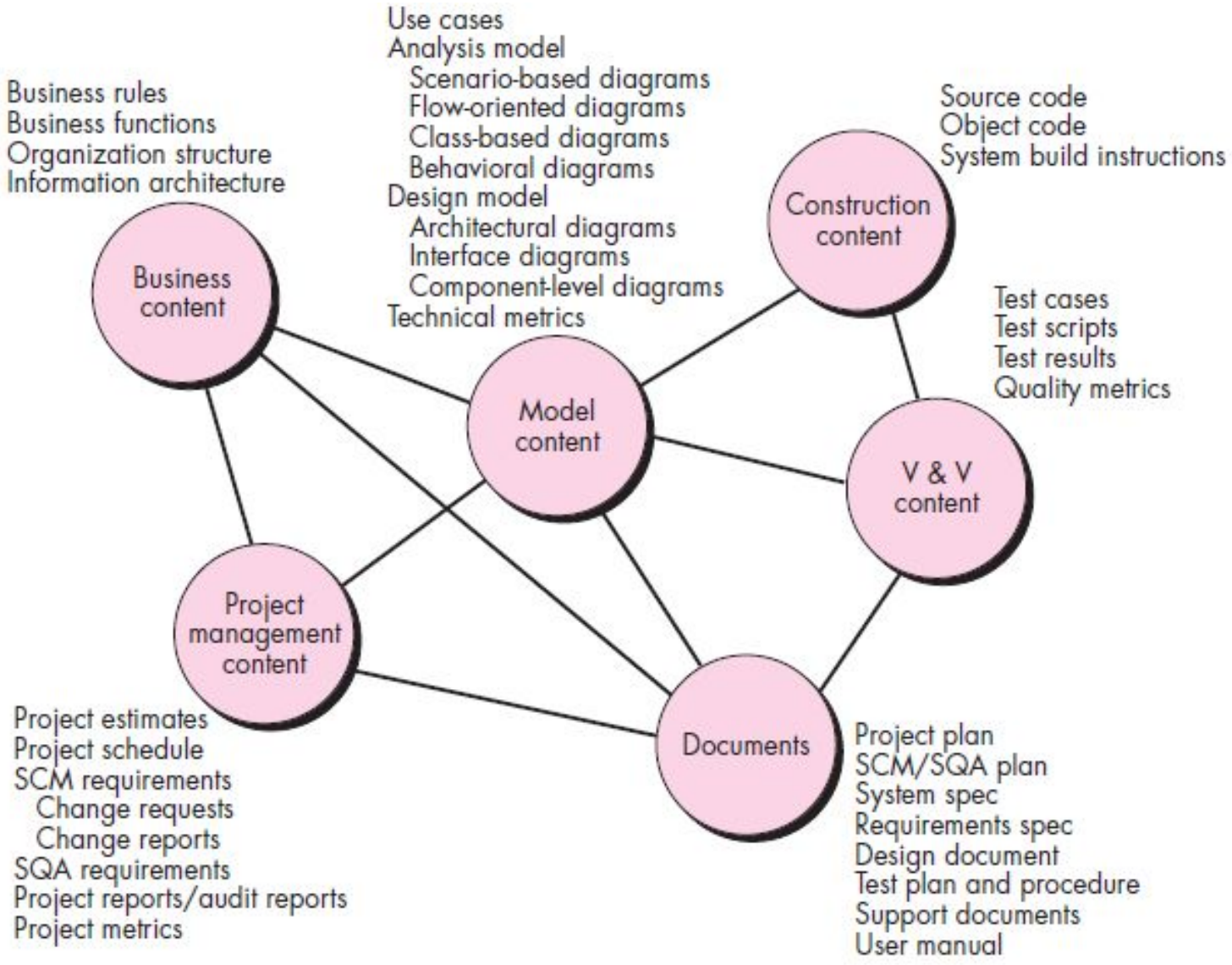
Configuration Management

SCM Repository

- The SCM repository is the set of mechanisms and data structures that allow a software team to manage change in an effective manner.
- The repository performs or precipitates the following functions:
  - Data integrity
  - Information sharing
  - Tool integration
  - Data integration
  - Methodology enforcement
  - Document standardization
- Repository Content

Configuration Management

Repository Content



Configuration Management

Repository Features

- Versioning
  - Saves all of these versions to enable effective management of product releases and to permit developers to go back to previous versions.
- Dependency tracking and change management
  - The repository manages a wide variety of relationships among the data elements stored in it.
- Requirements tracing
  - Provides the ability to track all the design and construction components and deliverables that result from a specific requirement specification
- Configuration management
  - Keeps track of a series of configurations representing specific project milestones or production releases.
  - Version management provides the needed versions and link management keeps track of interdependencies.
- Audit trails
  - Establishes additional information about when, why and by whom changes are made.

Configuration Management

SCM Elements

- Component elements—a set of tools coupled within a file management system (e.g., a database) that enables access to and management of each software configuration item.

- Process elements—a collection of procedures and tasks that define an effective approach to change management (and related activities) for all constituencies involved in the management, engineering and use of computer software.

- Construction elements—a set of tools that automate the construction of software by ensuring that the proper set of validated components (i.e., the correct version) have been assembled.

- Human elements—to implement effective SCM, the software team uses a set of tools and process features (encompassing other CM elements).

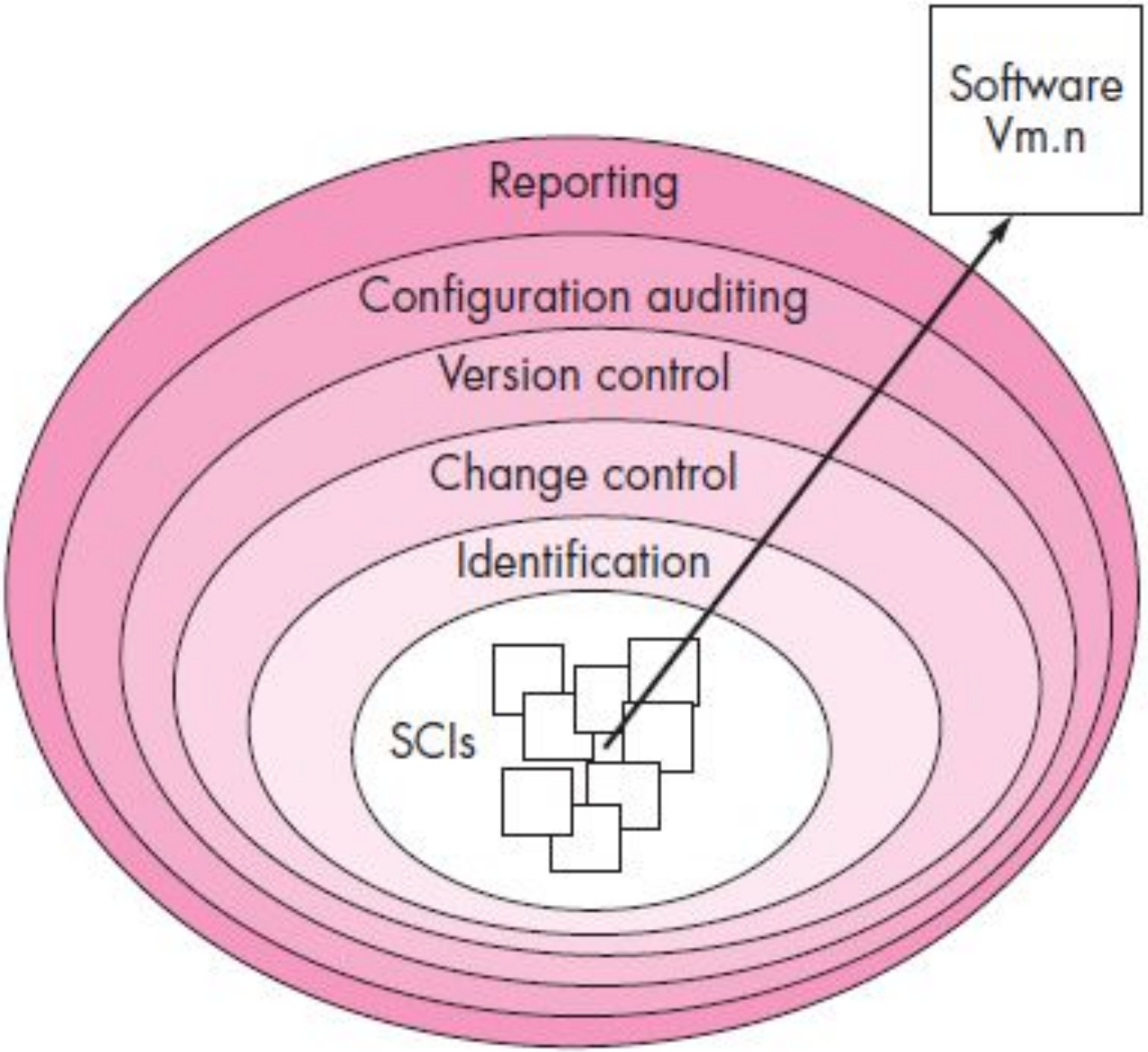
Configuration Management

SCM Process - Addresses the following questions…

- How does a software team identify the discrete elements of a software configuration?
- How does an organization manage the many existing versions of a program (and its documentation) in a manner that will enable change to be accommodated efficiently?
- How does an organization control changes before and after software is released to a customer?
- Who has responsibility for approving and ranking changes?
- How can we ensure that changes have been made properly?
- What mechanism is used to appraise others of changes that are made?

Configuration Management

SCM Process



Configuration Management

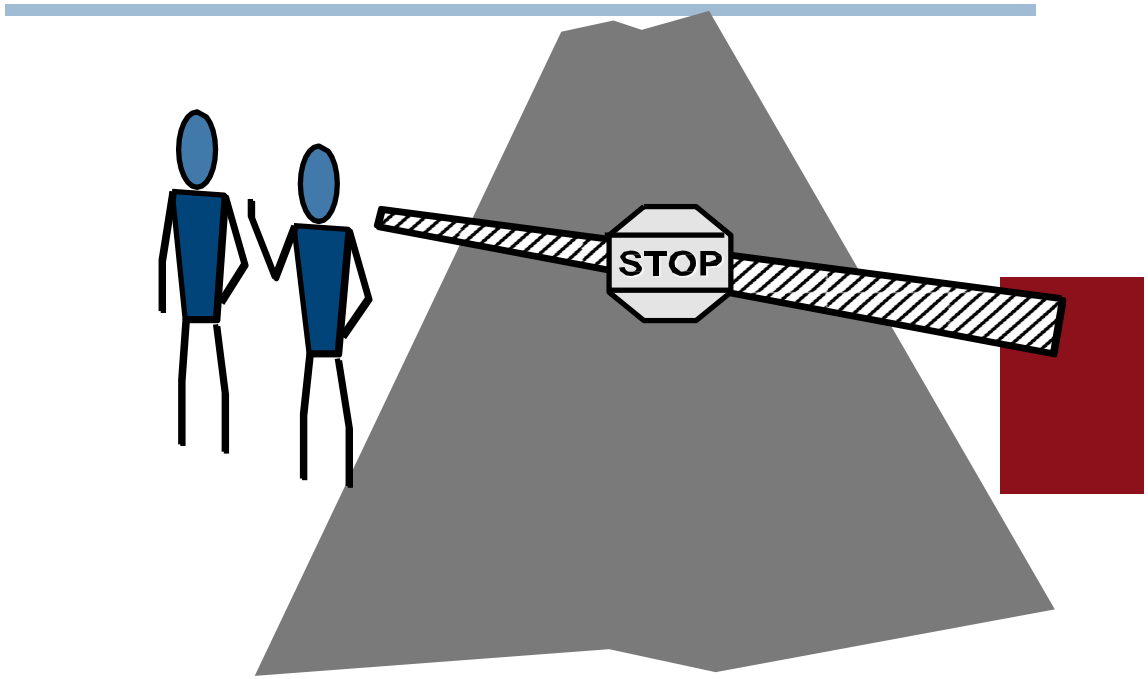
Version Control

- Version control combines procedures and tools to manage different versions of configuration objects that are created during the software process.
- A version control system implements or is directly integrated with four major capabilities:
  - A Project database (repository) that stores all relevant configuration objects.
  - A Version management capability that stores all versions of a configuration object (or enables any version to be constructed using differences from past versions);
  - A Make facility that enables the software engineer to collect all relevant configuration objects and construct a specific version of the software.
  - An Issues tracking (also called bug tracking) capability that enables the team to record and track the status of all outstanding issues associated with each configuration object.

Configuration Management

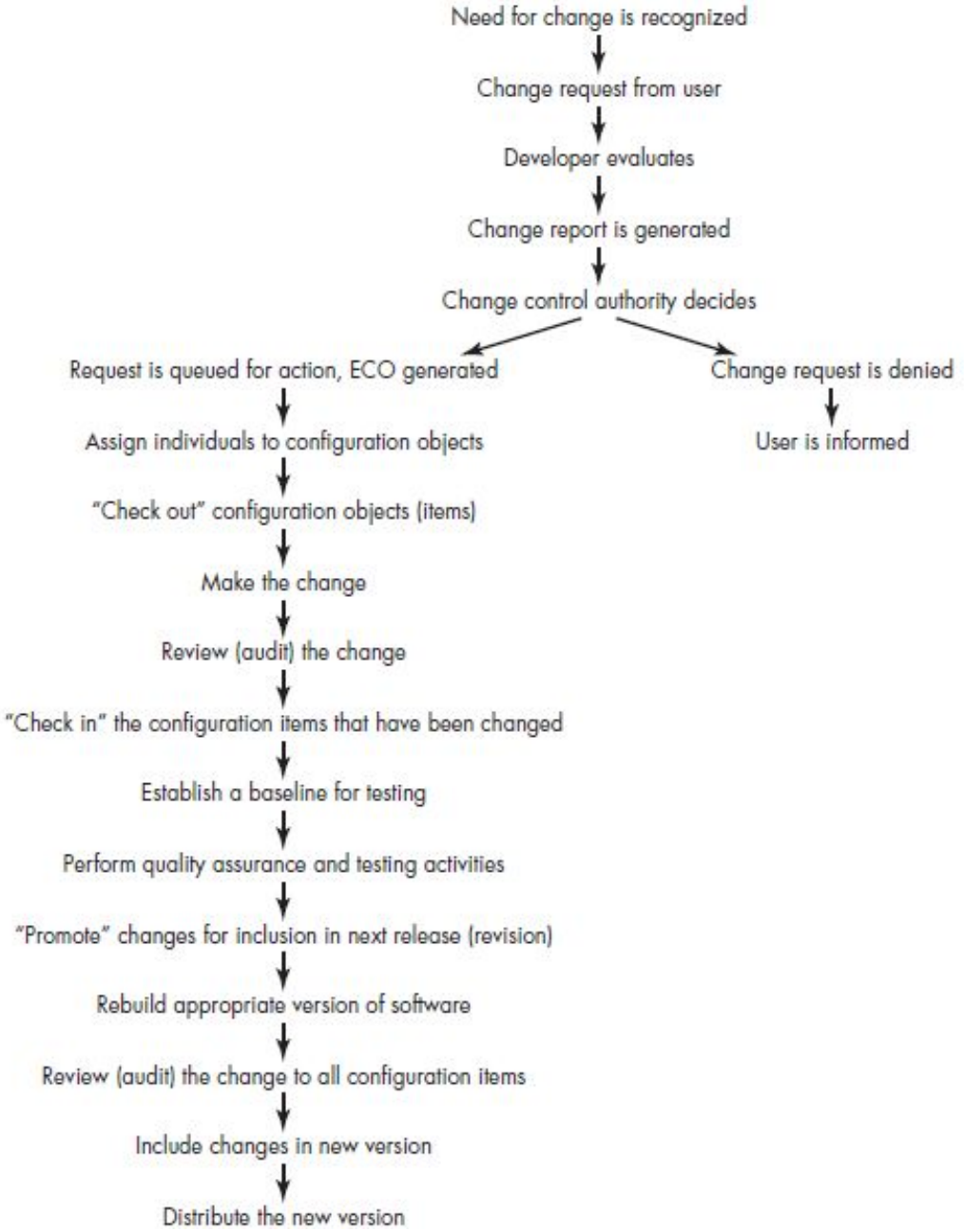
Change Control

- The reality of change control in a modern software engineering context has been summed up beautifully by James Bach.
- Change control is vital. But the forces that make it necessary also make it annoying.
- Bach recognizes that we face a balancing act. Too much change control and we create problems. Too little, and we create other problems.
- The change control process is illustrated schematically in below figure.



Configuration Management

Change Control Process



Configuration Management

Auditing

- Identification, version control and change control help you to maintain order in what would otherwise be a chaotic and fluid situation.
- How can a software team ensure that the change has been properly implemented? The answer is twofold: (1) Technical reviews, (2) Software configuration audit

- The technical review focuses on the technical correctness of the configuration object that has been modified.
- A software configuration audit complements the technical review by assessing a configuration object for characteristics that are generally not considered during review.

Configuration Management

Auditing

- The review asks and answers the following questions:

  1. Has the change specified in the ECO (Engineering Change Order) been made? Have any additional modifications been incorporated?

  2. Has a technical review been conducted to assess technical correctness?

  3. Has the software process been followed and have software engineering standards been properly applied?

  4. Has the change been "highlighted" in the SCI? Have the change date and change author been specified? Do the attributes of the configuration object reflect the change?

  5. Have SCM procedures for noting the change, recording it and reporting it been followed?

  6. Have all related SCIs been properly updated?

Configuration Management

Status Accounting

- Configuration status reporting (sometimes called status accounting) is an SCM task that answers the following questions:

  (1) What happened?

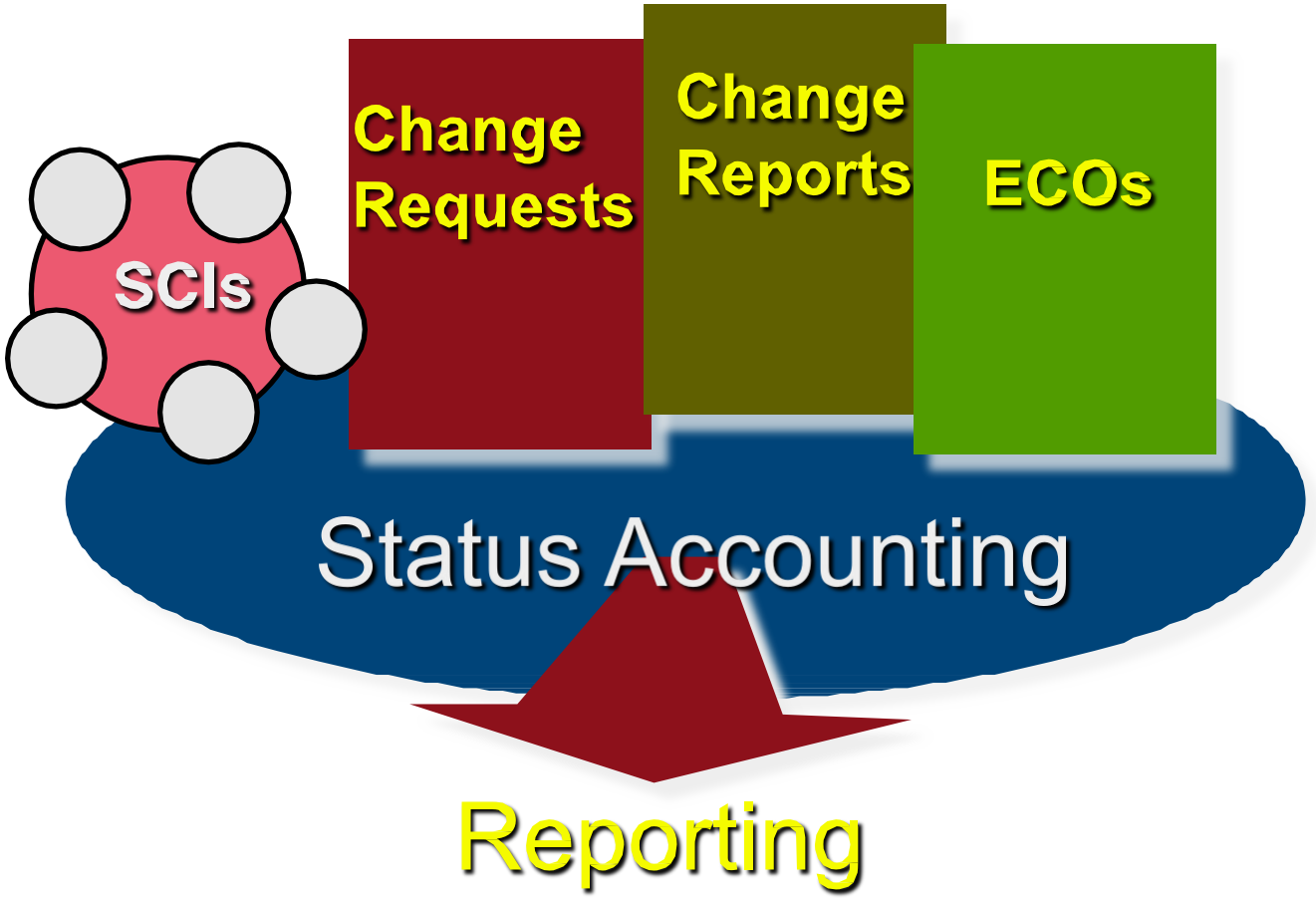
  (2) Who did it?

  (3) When did it happen?

  (4) What else will be affected?
- The flow of information for configuration status reporting (CSR) is illustrated in figure - The Change control process.
- Each time an SCI is assigned new or updated identification, a CSR entry is made.
- Each time a change is approved by the CCA (i.e., an ECO is issued), a CSR entry is made. (Change Control Authority)
- Each time a configuration audit is conducted, the results are reported as part of the CSR task.
- Output from CSR may be placed in an online database or website, so that software developers or support staff can access change information by keyword category.
- In addition, a CSR report is generated on a regular basis and is intended to keep management and practitioners appraised of important changes.

Configuration Management

Status Accounting



Configuration Management

SCM for Web Engineering

- Content
  - A typical WebApp contains a vast array of content—text, graphics, applets, scripts, audio/video files, forms, active page elements, tables, streaming data, and many others.
  - The challenge is to organize this sea of content into a rational set of configuration objects and then establish appropriate configuration control mechanisms for these objects.

- People
  - Because a significant percentage of WebApp development continues to be conducted in an ad hoc manner, any person involved in the WebApp can (and often does) create content.

- Scalability
  - As size and complexity grow, small changes can have far-reaching and unintended affects that can be problematic.
  - Therefore, the rigor of configuration control mechanisms should be directly proportional to application scale.

Configuration Management

SCM for Web Engineering

- Politics
  - Who owns a WebApp?
  - Who assumes responsibility for the accuracy of the information on the website?
  - Who assures that quality control processes have been followed before information is published to the site?
  - Who is responsible for making changes?
  - Who assumes the cost of change?

SCM Audit

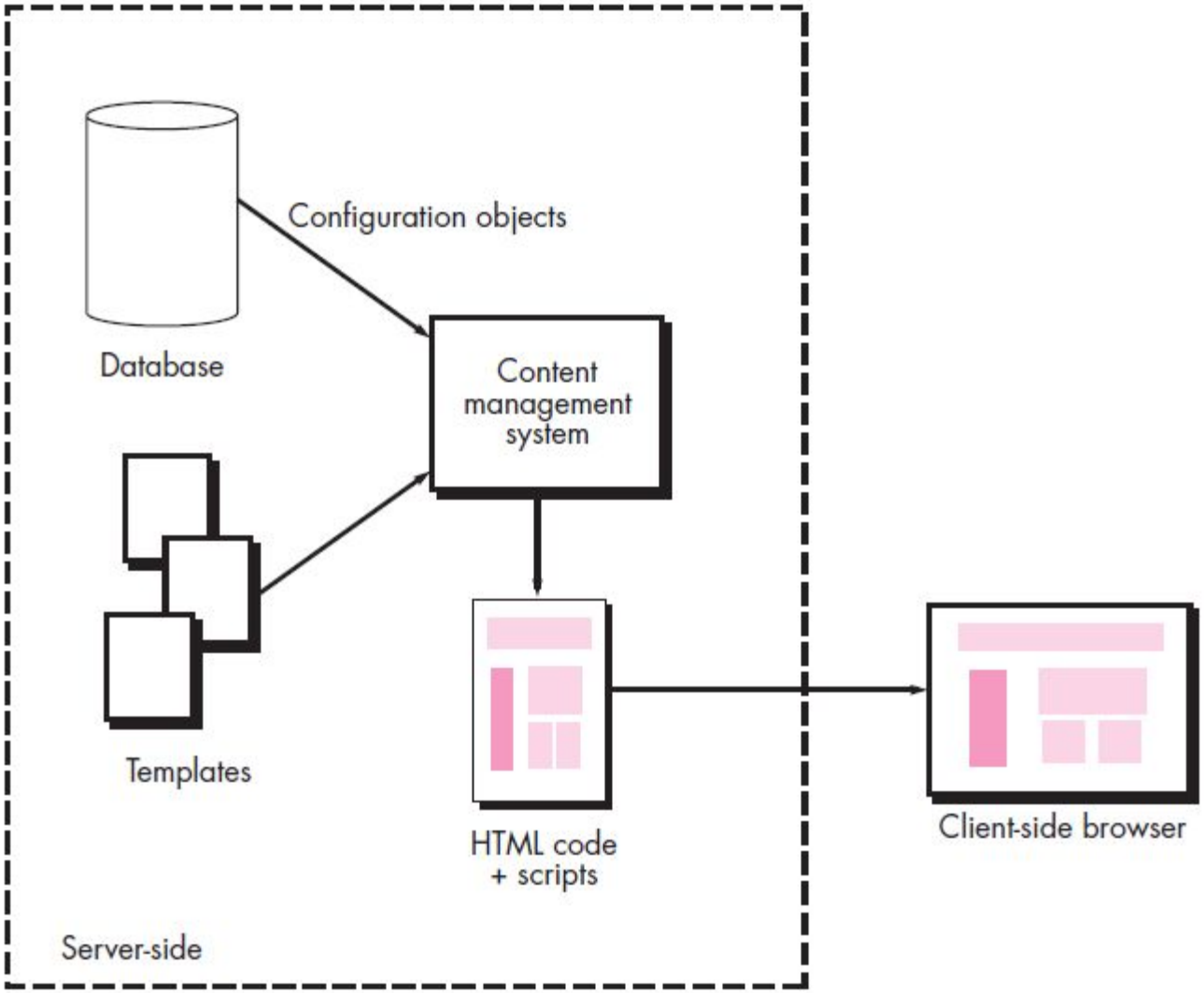
Configuration Management

Content Management

- The collection subsystem encompasses all actions required to create and/or acquire content and the technical functions that are necessary to
  - Convert content into a form that can be represented by a mark-up language (e.g., HTML, XML)
  - Organize content into packets that can be displayed effectively on the client-side.
- The management subsystem implements a repository that encompasses the following elements:
  - **Content database** – the information structure that has been established to store all content objects.
  - **Database capabilities**—functions that enable the CMS to search for specific content objects (or categories of objects), store and retrieve objects and manage the file structure that has been established for the content.
  - **Configuration management functions**—the functional elements and associated workflow that support content object identification, version control, change management, change auditing and reporting.

Configuration Management

Content Management



Configuration Management

Content Management

- The publishing subsystem extracts from the repository, converts it to a form that is amenable to publication and formats it so that it can be transmitted to client-side browsers.
- The publishing subsystem accomplishes these tasks using a series of templates.
- Each template is a function that builds a publication using one of three different components
    - Static elements—Text, graphics, media and scripts that require no further processing are transmitted directly to the client-side
    - Publication services—Function calls to specific retrieval and formatting services that personalize content (using predefined rules), perform data conversion, and build appropriate navigation links.
    - External services—Provide access to external corporate information infrastructure such as enterprise data or “back- room” applications.

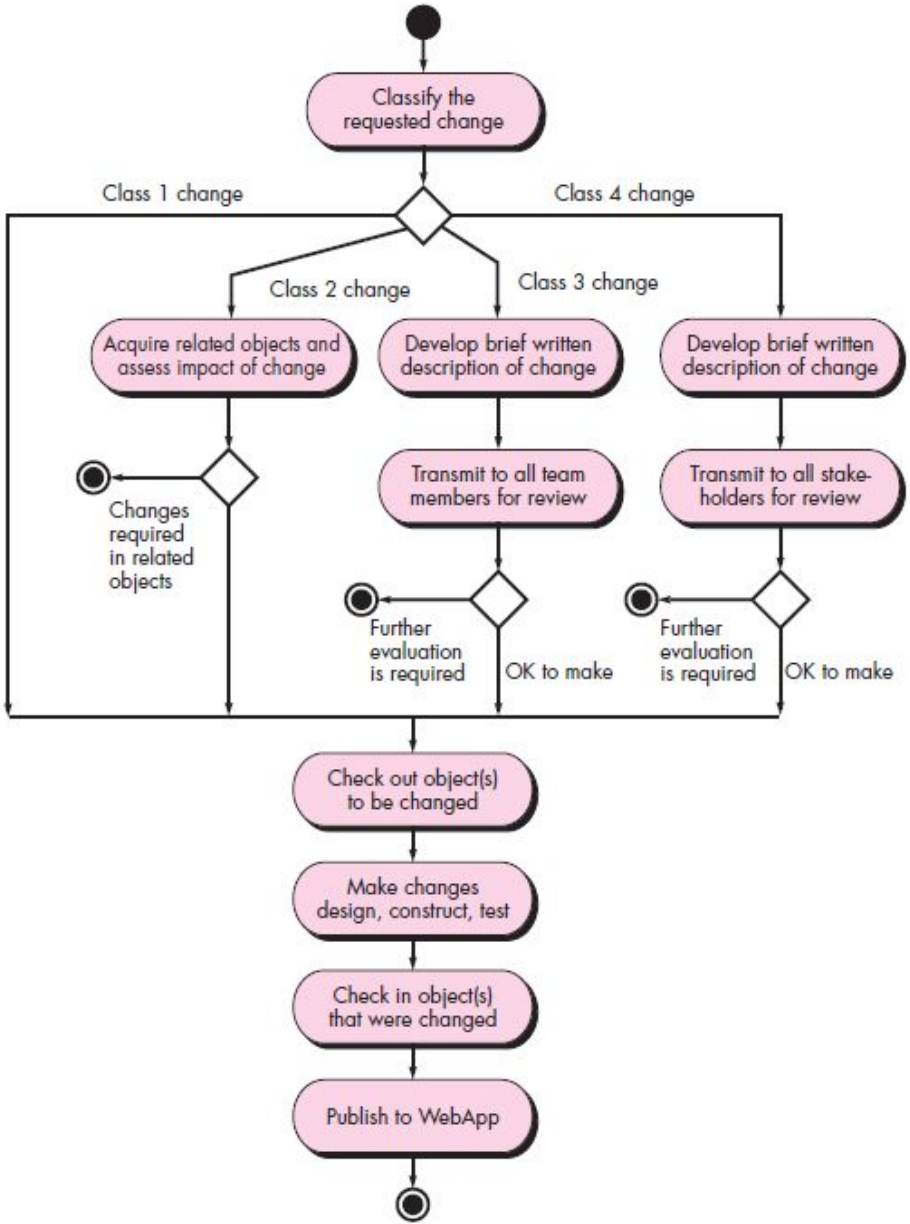
Configuration Management

Change Management for WebApps

- The workflow associated with change control for conventional software is generally too ponderous for WebApp development.
- It is unlikely that the change request, change report, and engineering change order sequence can be achieved in an agile manner that is acceptable for most WebApp development projects.
- To implement effective change management within the "code and go" philosophy that continues to dominate WebApp development, the conventional change control process must be modified.

Configuration Management

Change Management for WebApp



Configuration Management

Change Management for WebApps

- Each change should be categorized into one of four classes:

  1. Class 1 - A content or function change that corrects an error or enhances local content or functionality

  2. Class 2 - A content or function change that has an impact on other content objects or functional components

  3. Class 3 - A content or function change that has a broad impact across a WebApp (e.g., major extension of functionality, significant enhancement or reduction in content, major required changes in navigation)

  4. Class 4 - A major design change (e.g., a change in interface design or navigation approach) that will be immediately noticeable to one or more categories of user.

- Once the requested change has been categorized, it can be processed according to the algorithm shown in above figure - Managing changes for WebApps

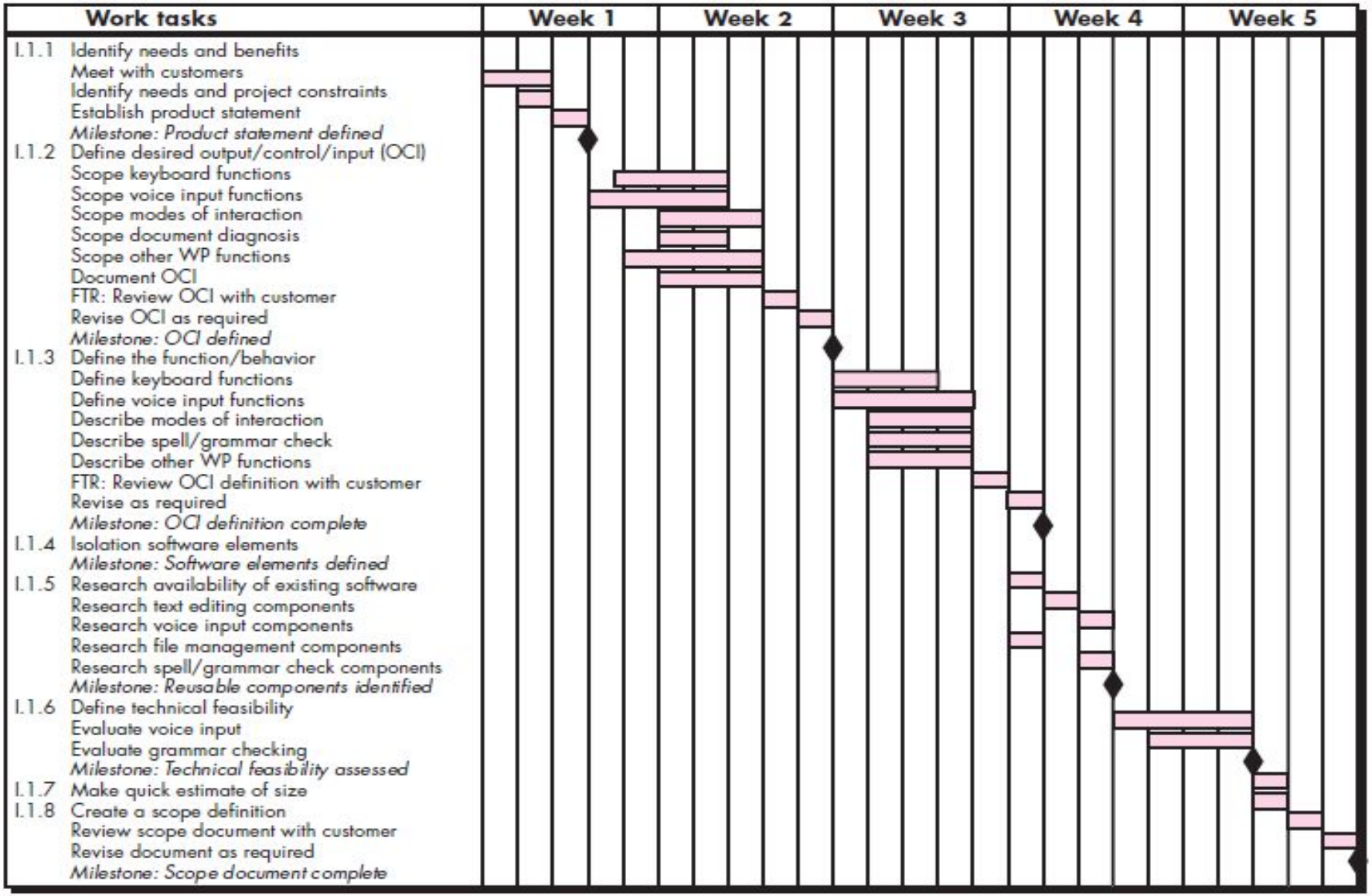
Project Planning

WBC

- 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 (PERT) and the Critical Path Method (CPM) are two project scheduling methods that can be applied to software development.
- Both techniques are driven by information already developed in earlier project planning activities: estimates of effort, a decomposition of the product function, the selection of the appropriate process model and task set and decomposition of the tasks that are selected.
- Interdependencies among tasks may be defined using a task network. Tasks, sometimes called the project work breakdown structure (WBS) are defined for the product as a whole or for individual functions.
- Both PERT and CPM provide quantitative tools that allow you to (1) determine the critical path - the chain of tasks that determines the duration of the project, (2) establish "most likely" time estimates for individual tasks by applying statistical methods, and (3) calculate "boundary times" that define a time "window" for a particular task.

Project Planning

- An example time-line chart



Project Planning

WBC - Time-Line Charts

- When creating a software project schedule, you begin with a set of tasks (the work breakdown structure).
- 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.
- Figure - An example time-line chart illustrates the format of a time-line chart.
- It depicts a part of a software project schedule that emphasizes the concept scoping task for a word-processing (WP) software product.

Project Planning

- An example project table

Work tasks	Planned start	Actual start	Planned complete	Actual complete	Assigned person	Effort allocated	Notes
I.1.1 Identify needs and benefits							Scoping will require more effort/time
Meet with customers	wk1, d1	wk1, d1	wk1, d2	wk1, d2	BLS	2 p-d	
Identify needs and project constraints	wk1, d2	wk1, d2	wk1, d2	wk1, d2	JPP	1 p-d	
Establish product statement	wk1, d3	wk1, d3	wk1, d3	wk1, d3	BLS/JPP	1 p-d	
Milestone: Product statement defined	wk1, d3	wk1, d3	wk1, d3	wk1, d3			
I.1.2 Define desired output/control/input (OCI)							
Scope keyboard functions	wk1, d4	wk1, d4	wk2, d2		BLS	1.5 p-d	
Scope voice input functions	wk1, d3	wk1, d3	wk2, d2		JPP	2 p-d	
Scope modes of interaction	wk2, d1		wk2, d3		MLL	1 p-d	
Scope document diagnostics	wk2, d1		wk2, d2		BLS	1.5 p-d	
Scope other WP functions	wk1, d4	wk1, d4	wk2, d3		JPP	2 p-d	
Document OCI	wk2, d1		wk2, d3		MLL	3 p-d	
FTR: Review OCI with customer	wk2, d3		wk2, d3		all	3 p-d	
Revise OCI as required	wk2, d4		wk2, d4		all	3 p-d	
Milestone: OCI defined	wk2, d5		wk2, d5				
I.1.3 Define the function/behavior							

Project Planning

WBC - Time-Line Charts

- 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 (Figure - An example project table).
- Used in conjunction with the time-line chart, project tables enable you to track progress.

Project Planning

Why are Projects Late?

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

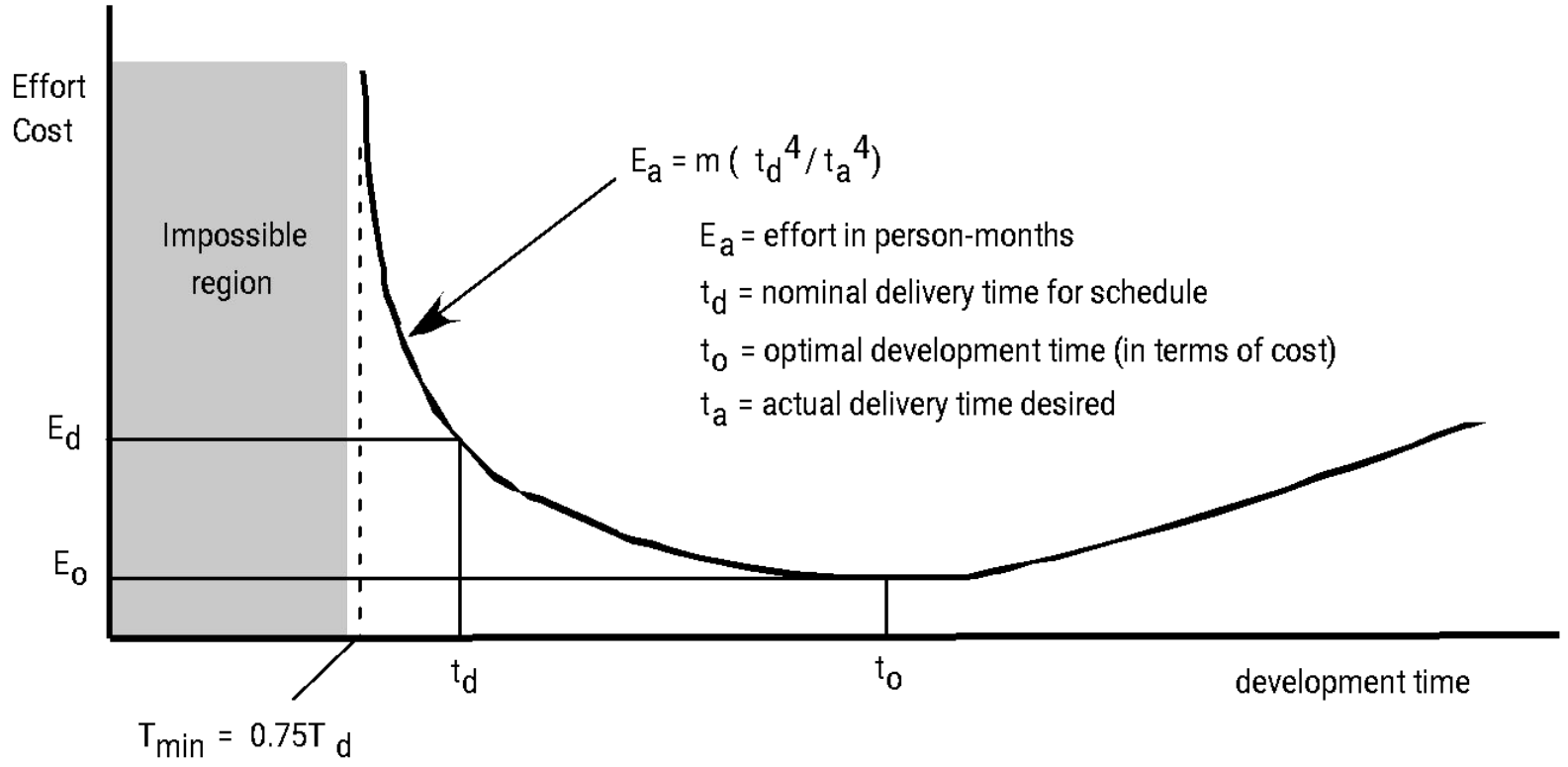
Project Planning

Scheduling Principles

- Compartmentalization—Define distinct tasks
- Interdependency—Indicate task interrelationship
- Effort validation—Be sure resources are available
- Defined responsibilities—People must be assigned
- Defined outcomes—Each task must have an output
- Defined milestones—Review for quality

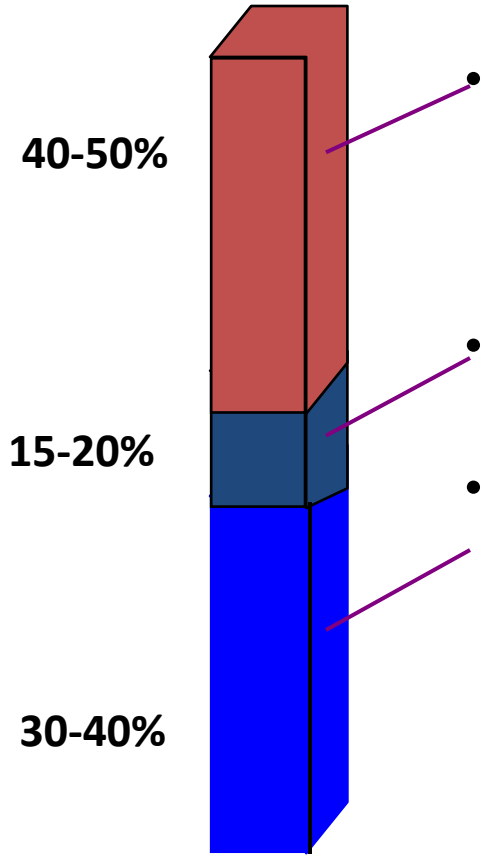
Project Planning

Effort and Delivery Time



Project Planning

Effort Allocation



- "Front end" activities
  - Customer communication
  - Analysis
  - Design
  - Review and modification
- Construction activities
  - Coding or code generation
- Testing and installation
  - Unit, integration
  - White-box, black box
  - Regression

Project Planning

- The overall goal of project planning is to establish a pragmatic strategy for controlling, tracking and monitoring a complex technical project.
- Why? So the end result gets done on time, with quality!

Project Planning Task Set – I

- Establish project scope
- Determine feasibility
- Analyze risks
- Define required resources
  - Determine require human resources
  - Define reusable software resources
  - Identify environmental resources

Project Planning

Project Planning Task Set – II

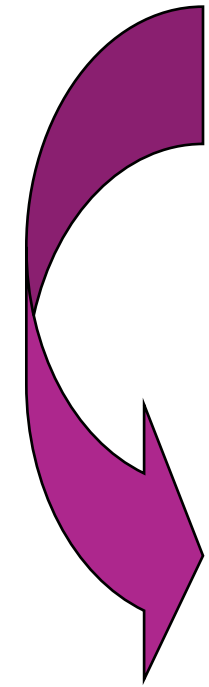
- Estimate cost and effort
  - Decompose the problem
  - Develop two or more estimates using size, function points, process tasks or use-cases
  - Reconcile the estimates
- Develop a project schedule
  - Scheduling
    - Establish a meaningful task set
    - Define a task network
    - Use scheduling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

Project Planning

- Defining Task Sets
  - Determine type of project
  - Assess the degree of rigor required
  - Identify adaptation criteria
  - Select appropriate software engineering tasks

Project Planning

Task Set Refinement



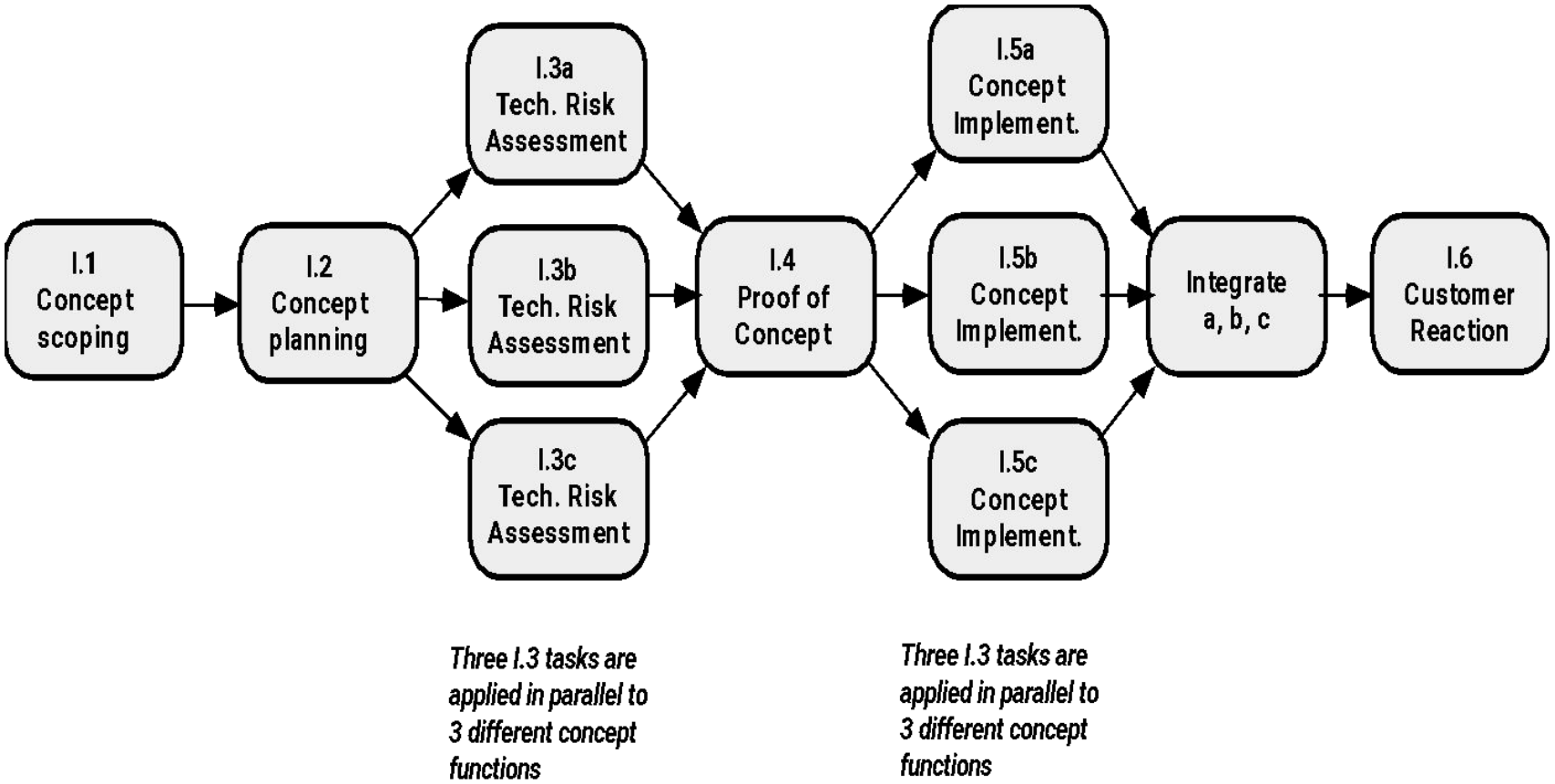
is refined to

1.1 **Concept scoping** determines the overall scope of the project.

Task definition: Task 1.1 Concept Scoping
1.1.1 Identify need, benefits and potential customers;
1.1.2 Define desired output/control and input events that drive the application;
Begin Task 1.1.2
1.1.2.1 FTR: Review written description of need
FTR indicates that a formal technical review is to be conducted.
1.1.2.2 Derive a list of customer visible outputs/inputs
1.1.2.3 FTR: Review outputs/inputs with customer and revise as required;
endtask Task 1.1.2
1.1.3 Define the functionality/behavior for each major function;
Begin Task 1.1.3
1.1.3.1 FTR: Review output and input data objects derived in task 1.1.2;
1.1.3.2 Derive a model of functions/behaviors;
1.1.3.3 FTR: Review functions/behaviors with customer and revise as required;
endtask Task 1.1.3
1.1.4 Isolate those elements of the technology to be implemented in software;
1.1.5 Research availability of existing software;
1.1.6 Define technical feasibility;
1.1.7 Make quick estimate of size;
1.1.8 Create a Scope Definition;
endTask definition: Task 1.1

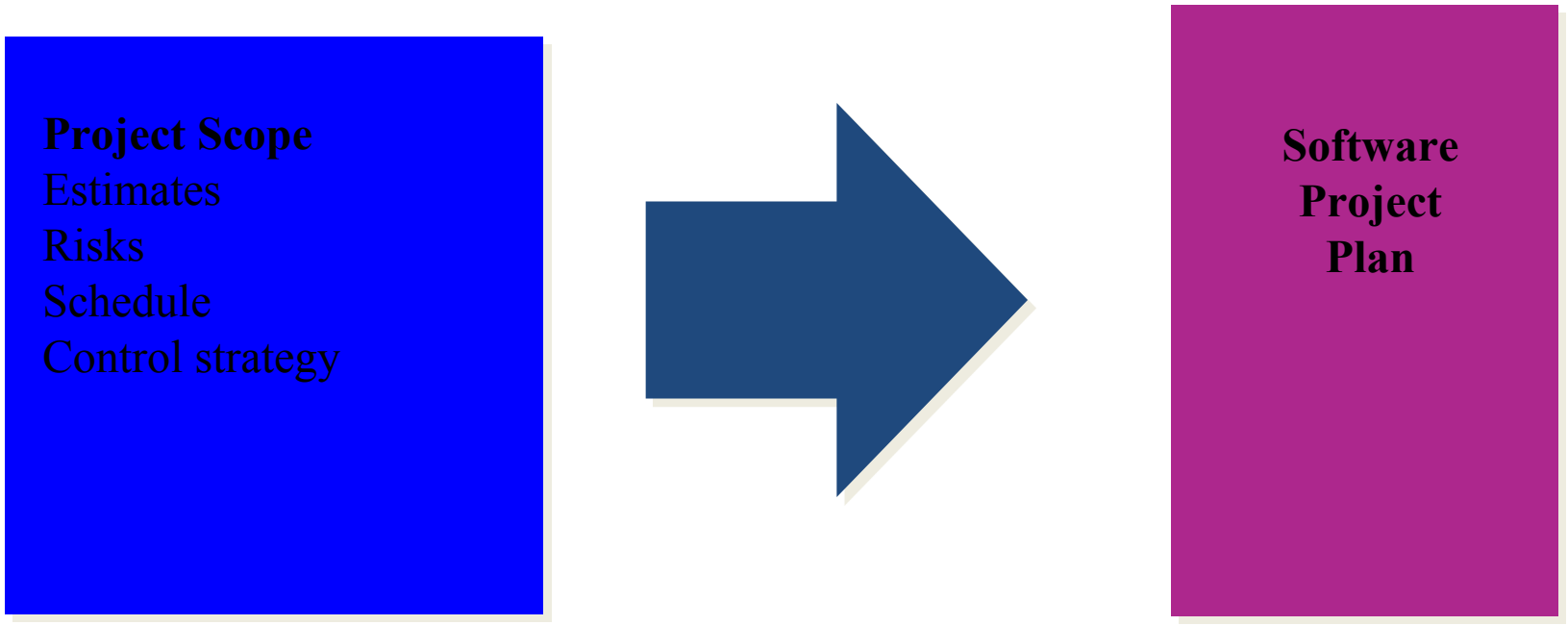
Project Planning

Define a Task Network



Estimation

- Estimation of resources, cost, and schedule for a software engineering effort requires
  - Experience
  - Access to good historical information (metrics)
  - The courage to commit to quantitative predictions when qualitative information is all that exists
- Estimation carries inherent risk and this risk leads to uncertainty
- Write it Down!



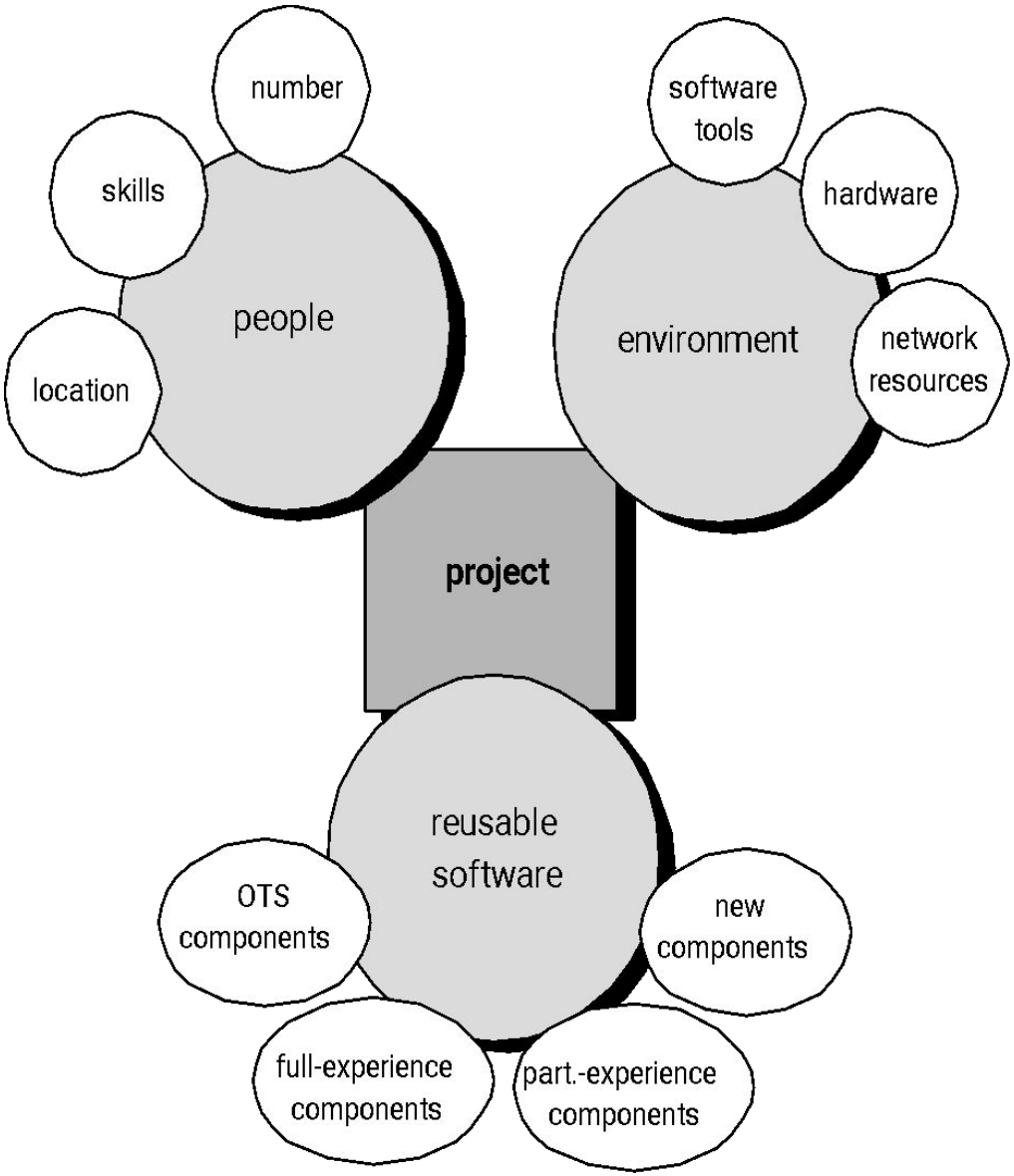
Scope

What is Scope?

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

  - Understand the customers needs
  - Understand the business context
  - Understand the project boundaries
  - Understand the customer's motivation
  - Understand the likely paths for change
  - Understand that ...Even when you understand, nothing is guaranteed!

Scope


number
skills
people
location
software tools
hardware
environment
network resources
project
reusable software
OTS components
new components
full-experience components
part.-experience components

REFERENCES

- Roger S. Pressman, Software Engineering – A Practitioner Approach, 6th ed., McGraw Hill, 2005
- Ian Somerville, Software Engineering, 8th ed., Pearson Education, 2010
- Kendall & Kendall, System Analysis and Design, 9th ed., Pearson Education, 2014
- Education, https://www.slideshare.net/AwaisSiddique8/cocomo-model-1-and-2

THANK YOU