Chapter 1

1.1	Software life cycle models: Waterfall, RAD, Spiral, Open-source, Agile process.
1.2	Understanding software process, Process metric,CMM levels,
1.3	Planning & Estimation: Product metrics Estimation- LOC, FP, COCOMO models.
1.4	Project Management activities : Planning, Scheduling and Tracking

Definition of Software Engineering

- **Software** is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product.**

- **Engineering** on the other hand, is all about developing products, using well-defined, scientific principles and methods.


Software Product
Requirements
System Analysis
System Design
Code Design
Testing
Deployment
Maintenance
Updates

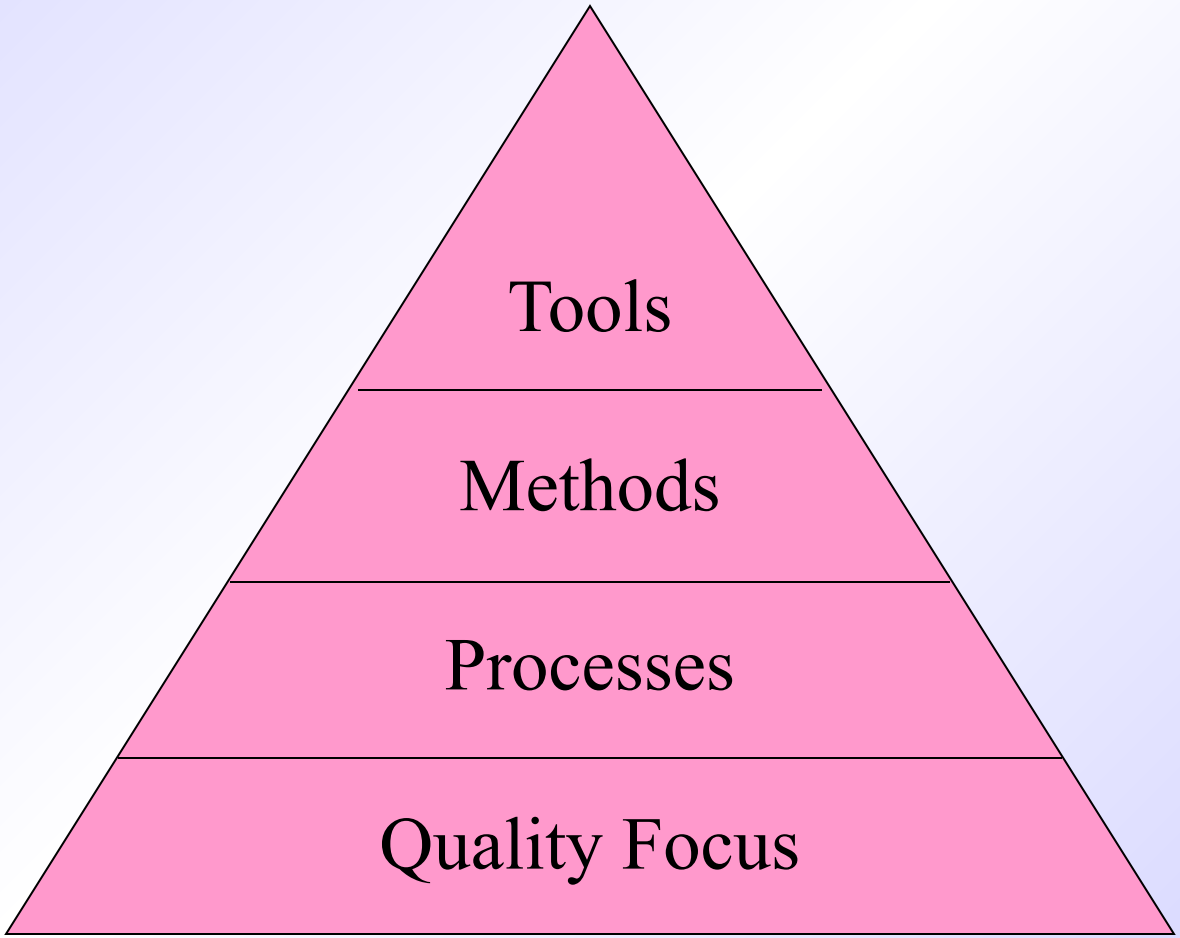
Software Engineering - Defined

- (1969) Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is **reliable and works efficiently on real machines**

- **Software engineering** is an engineering branch associated with **development of software** product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an **efficient and reliable software product.**

- (IEEE) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

Software Engineering is a Layered Technology



Process, Methods, and Tools

- **Process**
  - Holds technology and time for development of software.
  - Provides the **glue** that holds the layers together; enables rational and timely development; provides a framework for effective delivery of technology; forms the basis for management; provides the context for technical methods, work products, milestones, quality measures, and change management
- **Quality**
  - Software Engineering rests **on solid quality foundation**.
  - Quality assurance leads to more effective approach for software development.
- **Methods**
  - Provide the **technical "how to"** for building software; rely on a set of basic principles; encompass a broad array of tasks; include modeling activities
- **Tools**
  - Provide **automated or semi-automated** support for the process and methods.
  - Tools can also be integrated so that they can also be used for other system software development.

Prescriptive Process Model

- Prescriptive process models advocate an **orderly approach** to software engineering.
- Defines a distinct **set of activities**, actions, tasks, milestones, and work products that are required to engineer high-quality software.
- The activities may be **linear, incremental, or evolutionary.**
- Software process models prescribe a **workflow** in which the software development process elements are interrelated to one another.

Generic Process Framework

- **Communication**
  - Involves communication among the customer and other stake holders; encompasses requirements gathering.
  - All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.
- **Planning**
  - Establishes a plan for software engineering work; addresses technical tasks, resources, work products, and work schedule
- **Modeling (Analyze, Design)**
  - Encompasses the creation of models to better understand the requirements and the design.
  - The requirement specifications are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.

Generic Process Framework

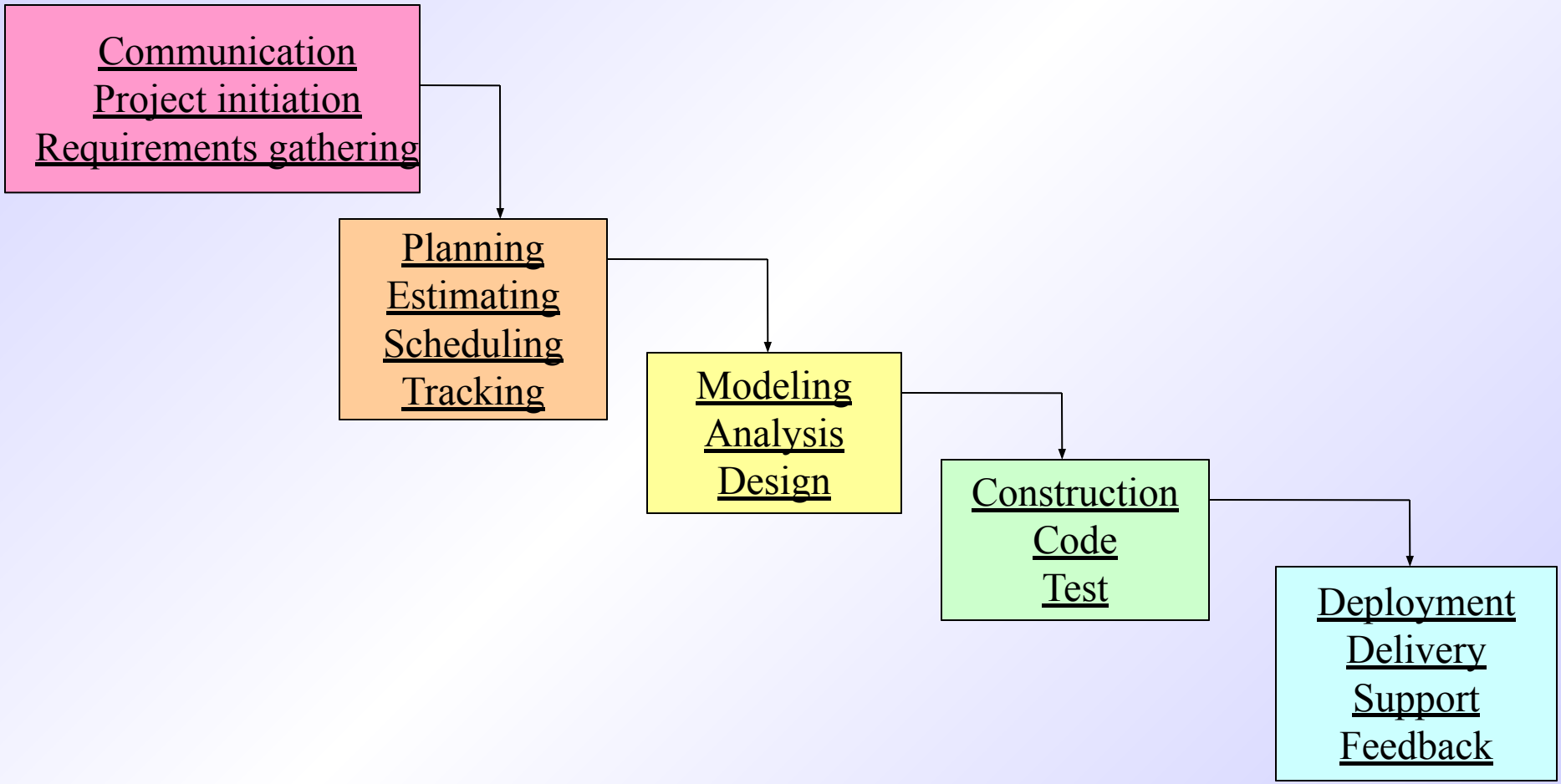
- **Construction (Code, Test)**
  - Combines code generation and testing to uncover errors.
  - With inputs from system design, the system is first developed in small programs called units and integrated. Each unit is developed and tested for its functionality which is referred to as Unit Testing.
  - Post integration the entire system is tested for any faults and failures.

- **Deployment**
  - Involves delivery of software to the customer for evaluation and feedback

Waterfall Model (Description)

- Oldest software lifecycle model and best understood by upper management.
- Used when requirements are well understood and risk is low.
- Work flow is in a linear (i.e., sequential) fashion.
- Used often with well-defined adaptations or enhancements to current software.
- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- The project is short.

Waterfall Model (Diagram)



Advantages

- Simple and easy to understand and use.
- Easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Process and results are well documented.

Disadvantages

- No working software is produced until late during the life cycle.
- Doesn't support iteration, so changes can cause confusion.
- Difficult for customers to state all requirements explicitly and up front
- Requires customer patience because a working version of the program doesn't occur until the final phase.
- High amounts of risk and uncertainty.

RAD Rapid Application Development

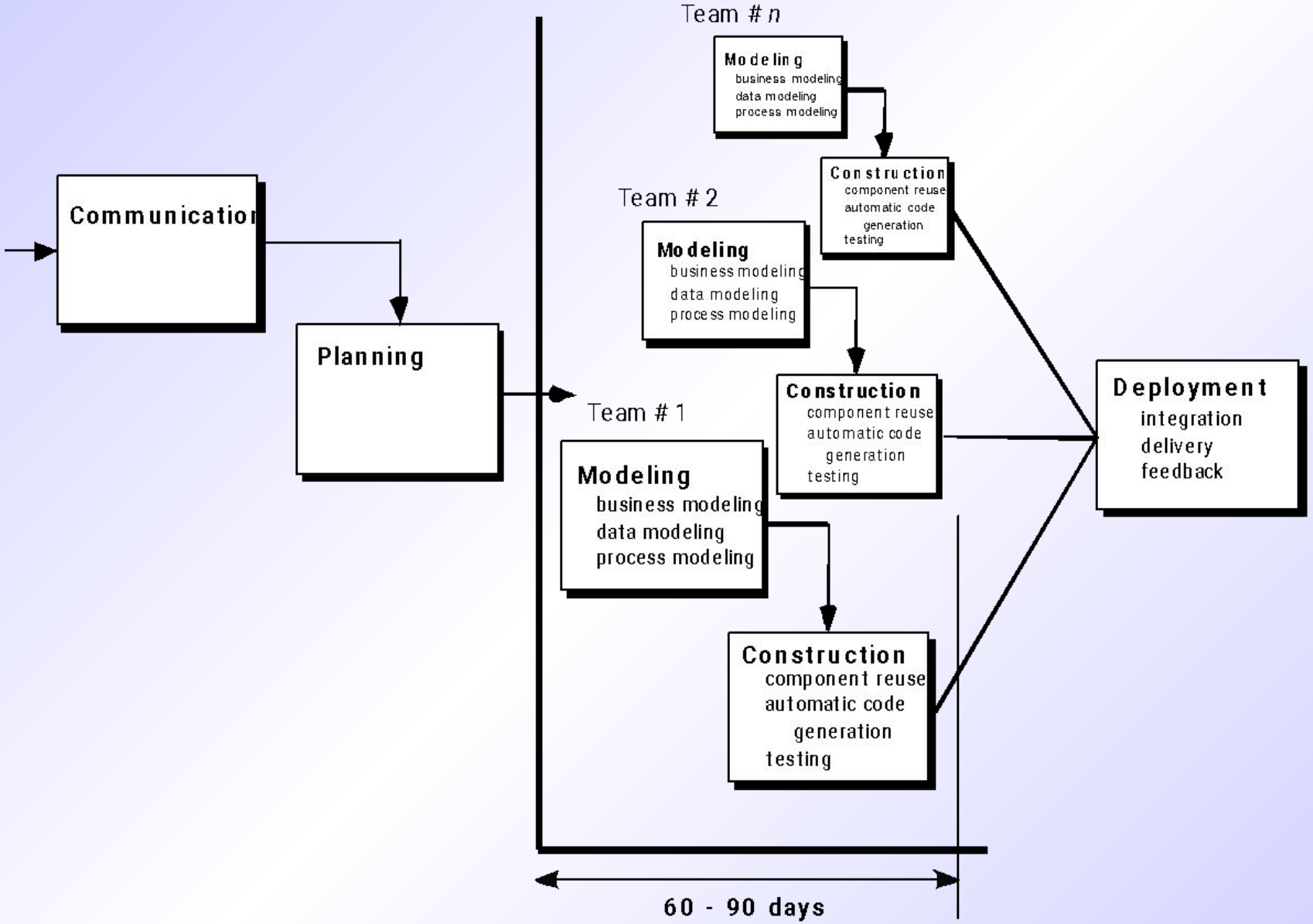
- RAD refers to a development life cycle designed to give much faster development and higher quality systems than the traditional life cycle.

- In RAD model the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery.

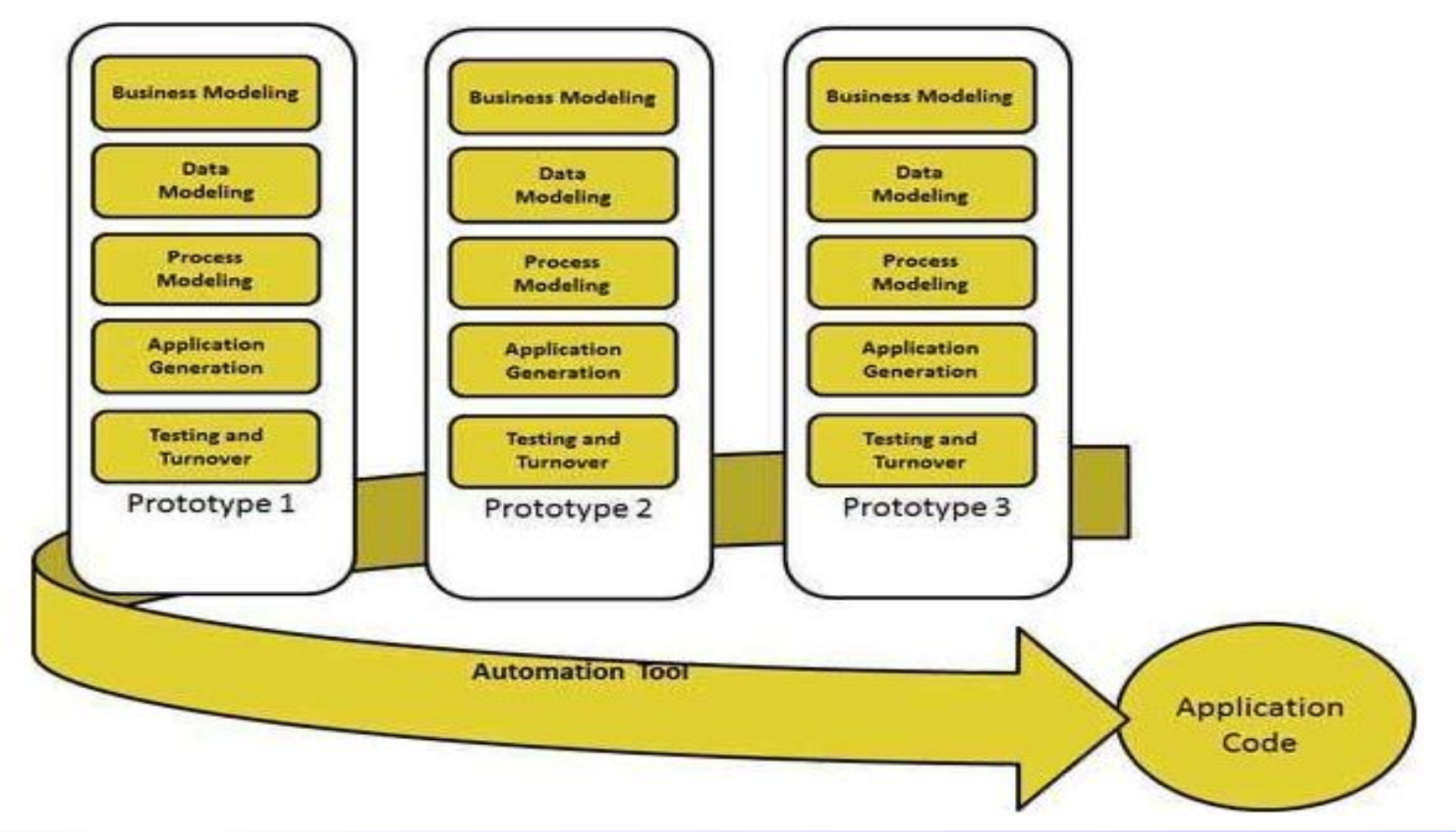
- RAD projects follow iterative and incremental model and have small teams comprising of developers, domain experts, customer representatives and other IT resources working progressively on their component or prototype.

RAD Rapid Application Development

- Since there is no detailed preplanning, it makes it easier to incorporate the changes within the development process.
- Small working teams.
- The most important aspect for this model to be successful is to make sure that the prototypes developed are reusable.


Team # n
Modeling
business modeling
data modeling
process modeling
Construction
component reuse
automatic code
generation
testing
Team # 2
Modeling
business modeling
data modeling
process modeling
Communication
Planning
Construction
component reuse
automatic code
generation
testing
Deployment
integration
delivery
feedback
Team # 1
Modeling
business modeling
data modeling
process modeling
Construction
component reuse
automatic code
generation
testing
60 - 90 days

RAD Rapid Application Development



Advantages

- Changing requirements can be accommodated.
- Progress can be measured.
- Iteration time can be short with use of powerful RAD tools.
- Increases reusability of components.
- Reduced development time.
- Encourages customer feedback.
- Integration from very beginning solves a lot of integration issues.

Disadvantages

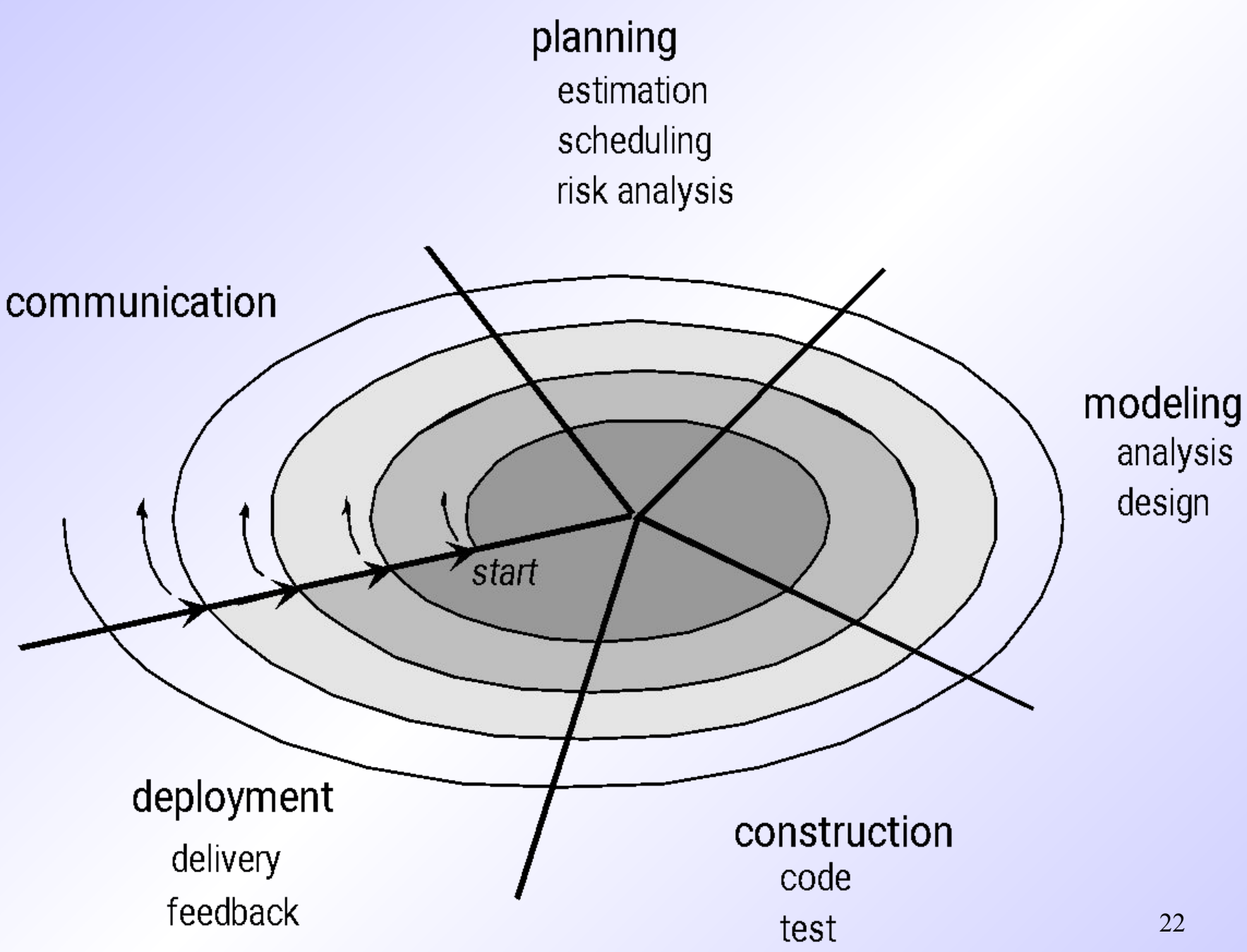
- Dependency on technically strong team members for identifying business requirements.
- Only system that can be modularized can be built using RAD.
- Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.
- Requires user involvement throughout the life cycle.
- Suitable for project requiring shorter development times.
- Management complexity is more.

Spiral Model

- Used when requirements are not well understood and risks are high.
- Inner spirals focus on identifying software requirements and project risks; may also incorporate prototyping.
- Outer spirals take on a classical waterfall approach after requirements have been defined, but permit iterative growth of the software.

Spiral Model

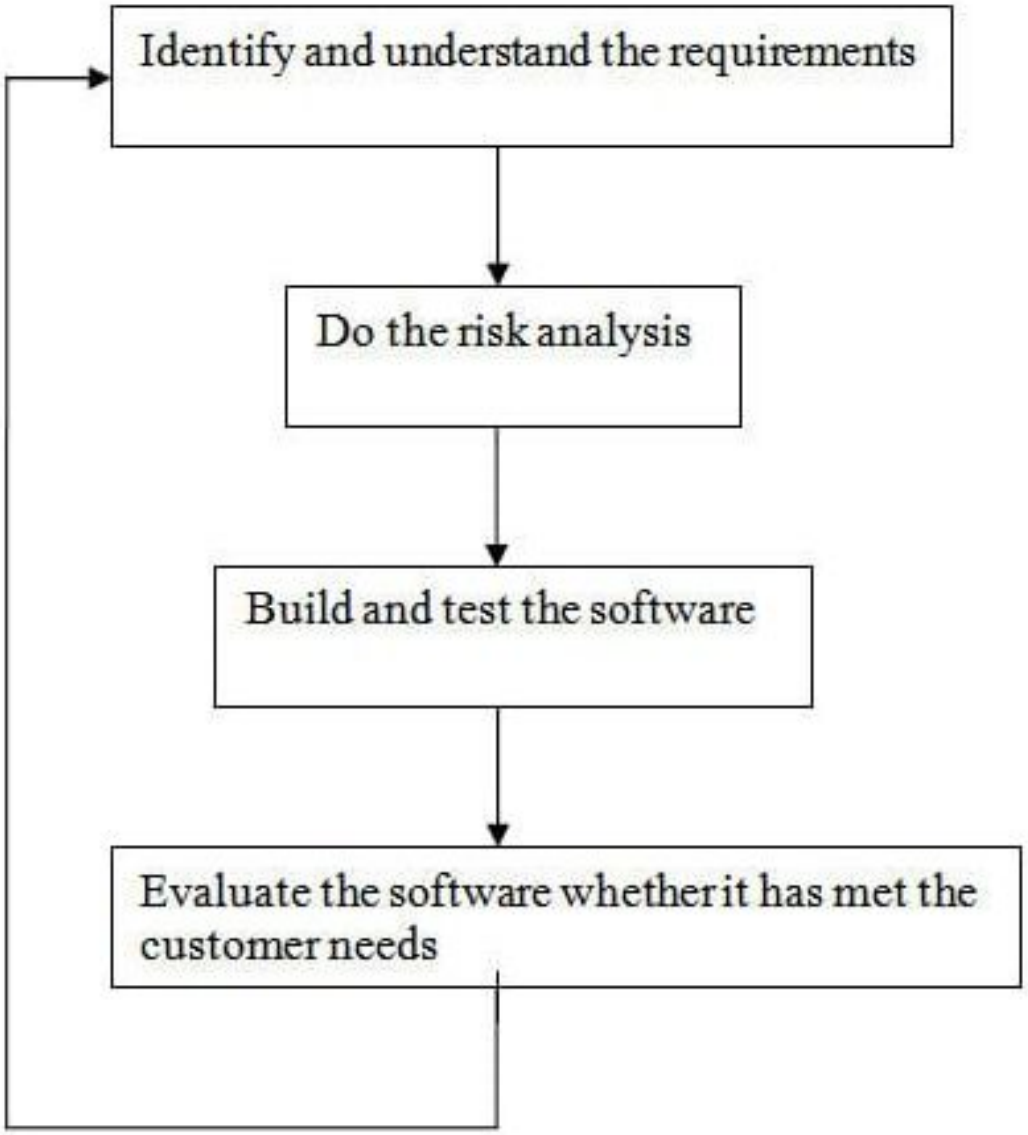
- Requires considerable expertise in risk assessment.
- Extends waterfall model by adding iteration to explore /manage risk.
- **Key idea:** on each iteration identify and solve the sub-problems with the highest risk.


planning
estimation
scheduling
risk analysis
communication
modeling
analysis
design
start
deployment
delivery
feedback
construction
code
test

Spiral Model

- New product line which should be released in phases to get enough customer feedback.
- Significant changes are expected in the product during the development cycle

To explain in simpler terms, the steps involved in the spiral model are:



Advantages

- **Realism:** the model accurately reflects the iterative nature of software development on projects with unclear requirements Allows for extensive use of prototypes.
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and more risky parts can be developed earlier which helps better risk management.
- Flexible: incorporates the advantages of the waterfall and evolutionary methods

Disadvantages

- Management is more complex.
- End of project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.
- Spiral may go indefinitely.
- Large number of intermediate stages requires excessive documentation.

Open Source Model

- All successful open-source software projects go through **two informal phases**.

- Single individual has an **idea for a program**, such as an operating system (Linux), a Net browser (Firefox), or a Web server (Apache) , initial version of that program is build.

- Software is made available to users over internet as it is free.

Open Source Model

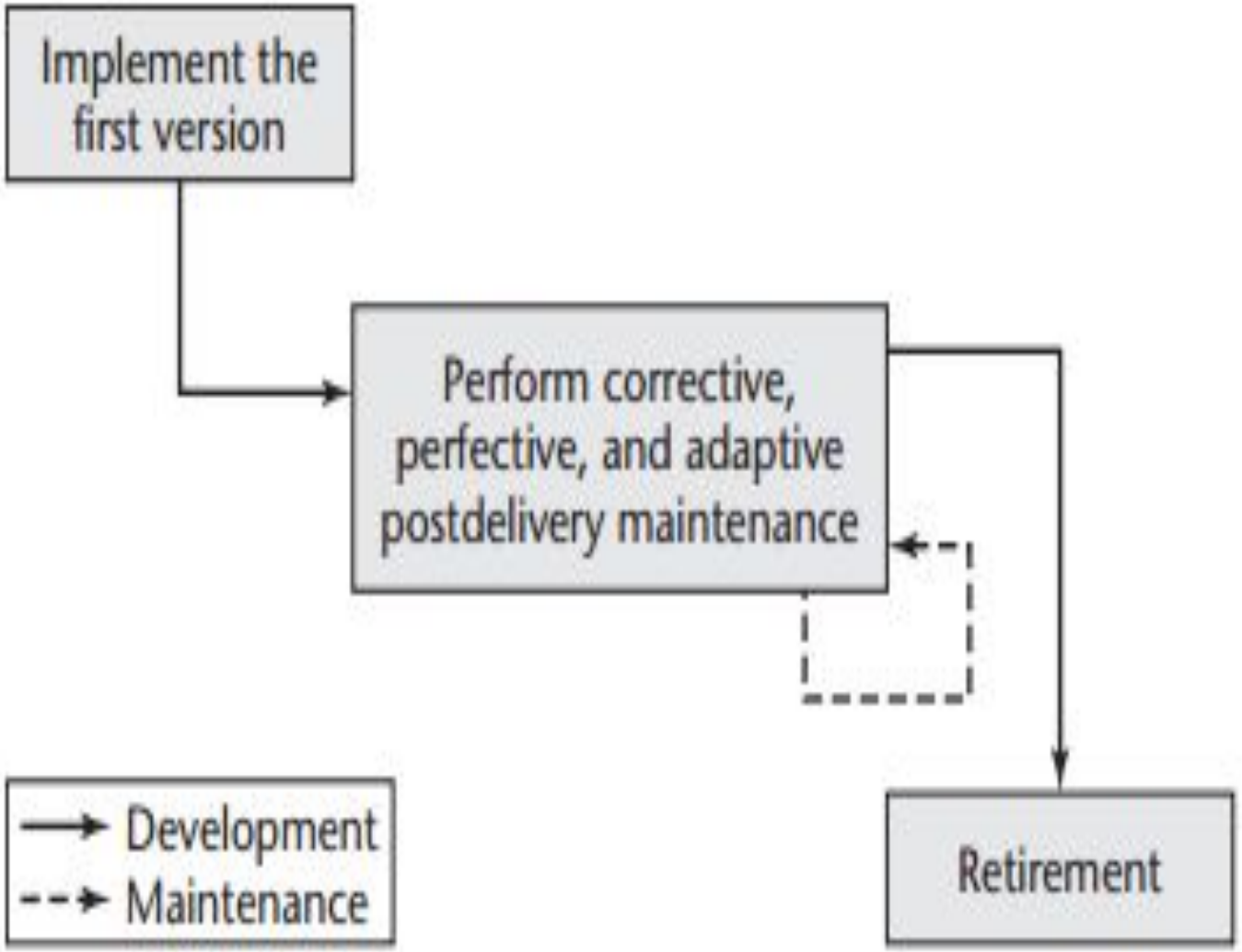
- Once the users start using the program second informal phase of the project begins.
- Users become co-developers, in that some users report defects and others suggest ways of fixing those defects.
- Some users put forward ideas for extending the program, and others implement those ideas.
- As the program expands in functionality, yet other users port the program so that it can run on additional operating system/hardware combinations.

Open Source Model

- A key aspect is that individuals work on an open-source project in their spare time on a voluntary basis; they are not paid to participate.

- The second informal phase of the open-source life-cycle model consists solely of **post delivery maintenance.**

Figure 2.11
The open-source life-cycle model.



Agile Process Model

- Agile SDLC model is a combination of **iterative and incremental process models** with focus on process adaptability and customer satisfaction by rapid delivery of working software product.

- Agile Methods **break the product into small incremental builds.** These builds are **provided in iterations.** Each iteration typically lasts from about one to three weeks. every iteration involves cross functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing.

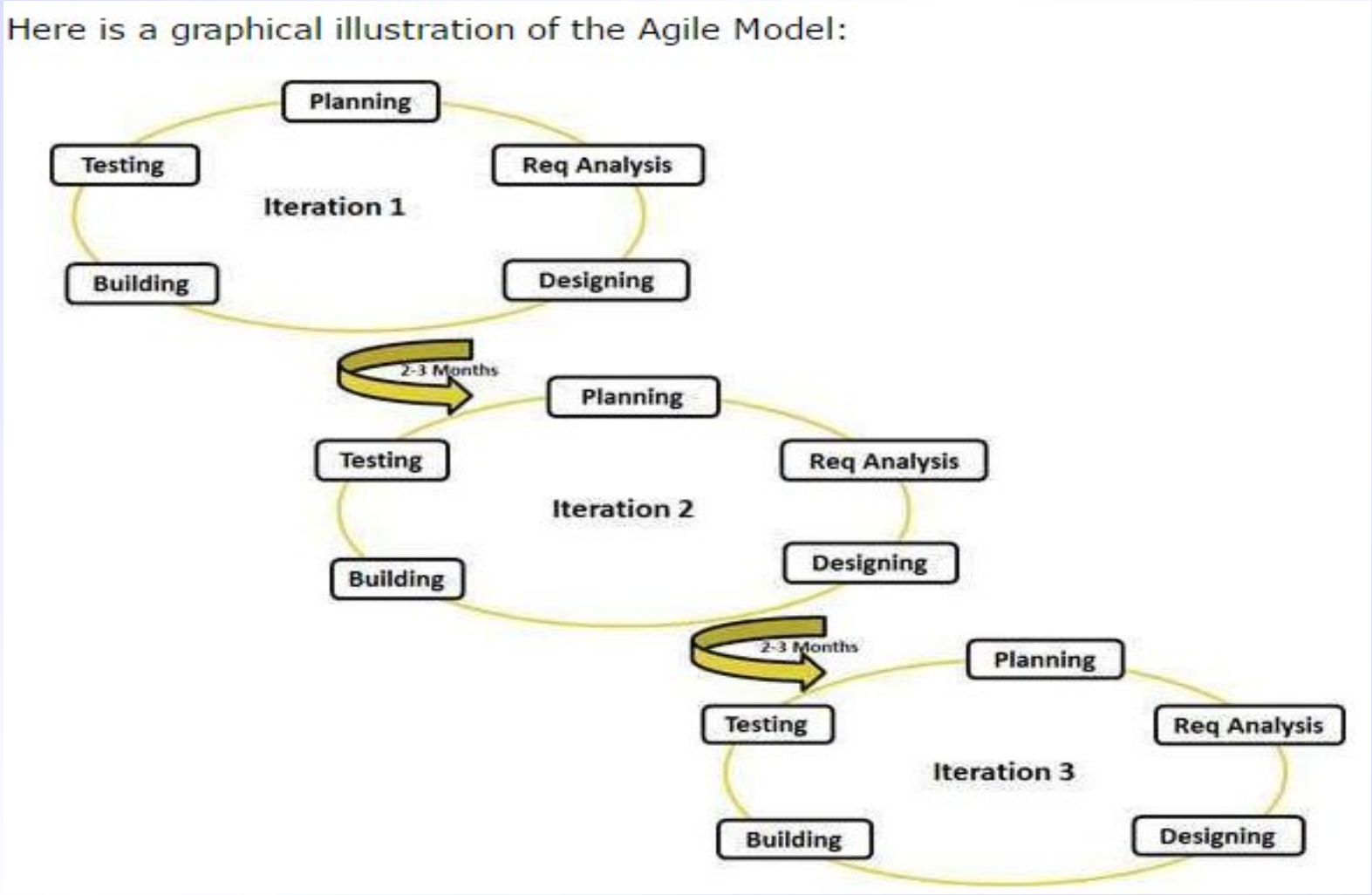
Agile Process Model

- At the end of the iteration a working product is displayed to the customer and important stakeholders.

- Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

- Agile processes must be adapted incrementally to manage unpredictability

Agile Process Model



Advantages

- Is a very realistic approach to software development.
- Promotes teamwork and cross training.
- Functionality can be developed rapidly and demonstrated.
- Suitable for fixed or changing requirements.
- Delivers early partial working solutions.
- Good model for environments that change steadily.

Advantages

- Easy to manage.
- Little or no planning required.
- Highest priority is to satisfy the customer.

Disadvantages

- In case of some software deliverables, especially the large ones, it is **difficult to assess the effort** required at the **beginning** of the software development life cycle.
- For Designing and Documentation, agile methodology **pays less importance.**
- The project can easily get taken **off track** if the customer representative is not clear what final outcome that they want.
- For Agile methodology, **experience resource** will be needed.

Agile Process Models

- Extreme Programming (XP)
- Adaptive Software Development (ASD)
- Dynamic Systems Development Method (DSDM)
- Scrum
- Crystal
- Feature Driven Development (FDD)
- Agile Modeling (AM)

Extreme Programming (XP)

- Relies on object-oriented approach as it is based on **set rules and practices** that occur within the activities defined.

- Agile processes must be **adapted incrementally to manage unpredictability**

Extreme Programming (XP)

XP Planning

- Begins with the creation of "user stories"
- Agile team assesses each story and assigns a cost
- Stories are grouped to 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

Extreme Programming (XP)

XP Design

- Follows the KIS (Keep it simple) principle
- Encourage the use of CRC (class responsibility collaborator-effective mechanism for problem solving as uses object oriented approach) cards for difficult design problems, suggests the creation of "spike solutions"—a design prototype
- Encourages "refactoring" or "restructuring"—an iterative refinement of the internal program design

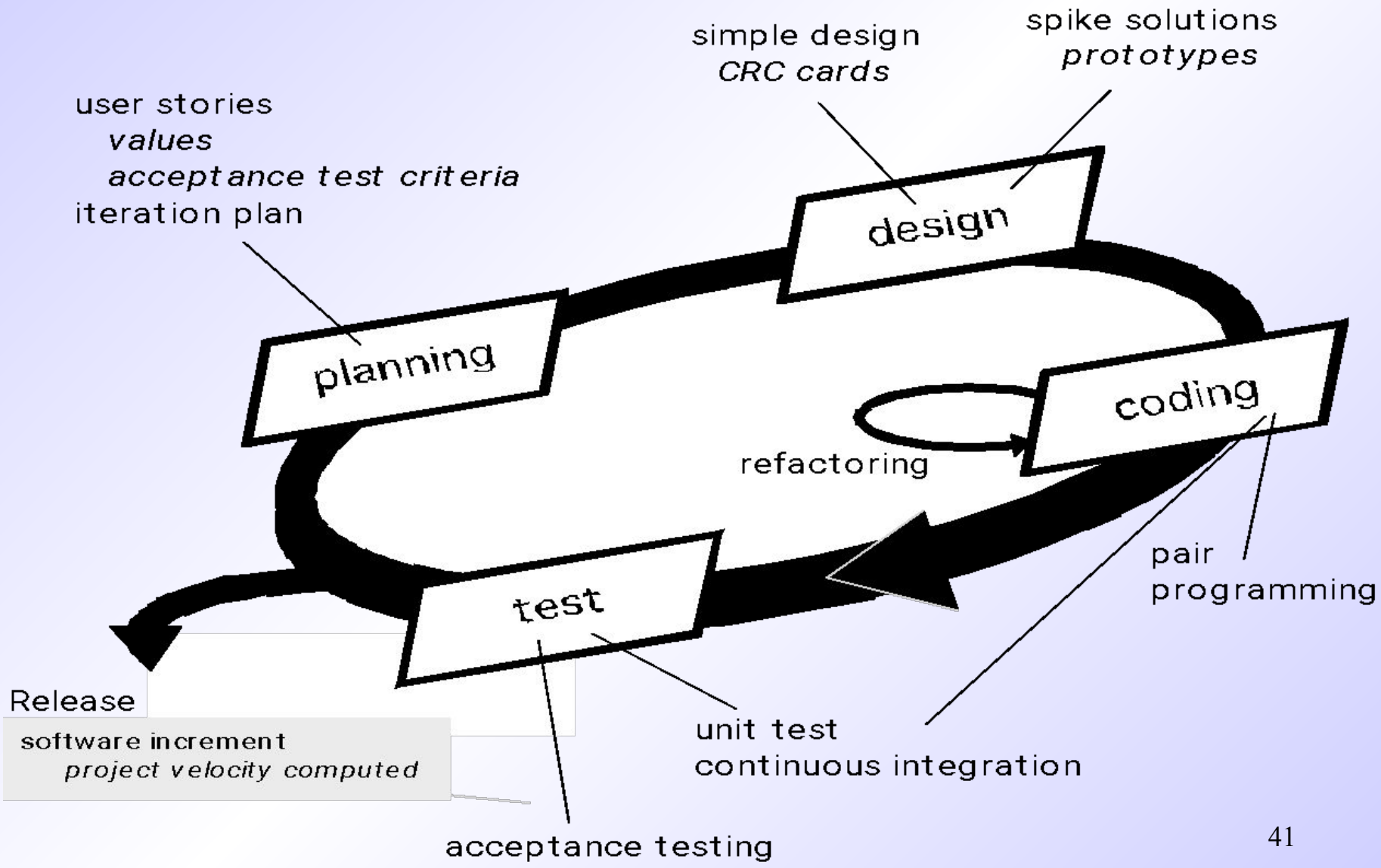
XP Coding

- Recommends the construction of a unit test for a store *before* coding commences
- Encourages "pair programming-two people working on same program"

XP Testing

- All unit tests are executed daily
- "Acceptance tests" are defined by the customer and executed to assess customer visible functionality

Extreme Programming (XP)



Adaptive Software Development

- Technique for building complex software and systems.
- Self-organization arises when independent agents cooperate to create a solution to a problem that is beyond the capability of any individual agent.
- Emphasizes self-organizing teams, interpersonal collaboration, and both individual and team learning

Adaptive Software Development

- Adaptive Software Development practices provide ability to accommodate change and are **adaptable in turbulent environments** with products evolving with little planning and learning.
- The Adaptive Software Development Lifecycle **focuses on results**, not tasks, and the results are identified as application features.

ASD — distinguishing features

- 1. Mission Driven
  The activities in the each development cycle must be justified against the overall project mission.
- 2. Component Based
  Development activities should not be task oriented but rather focus on developing working software. It focuses on results.
- 3. Iterative
  Redoing of development instead of doing it right the first time

ASD — distinguishing features

- 4. Time Boxed
  Setting fixed delivery times for projects
- 5. Change Tolerant
  Able to incorporate change is viewed as a competitive advantage (not as a problem).
- 6. Risk Driven
  The development of high risk items should begin as early as possible.

ASD phases

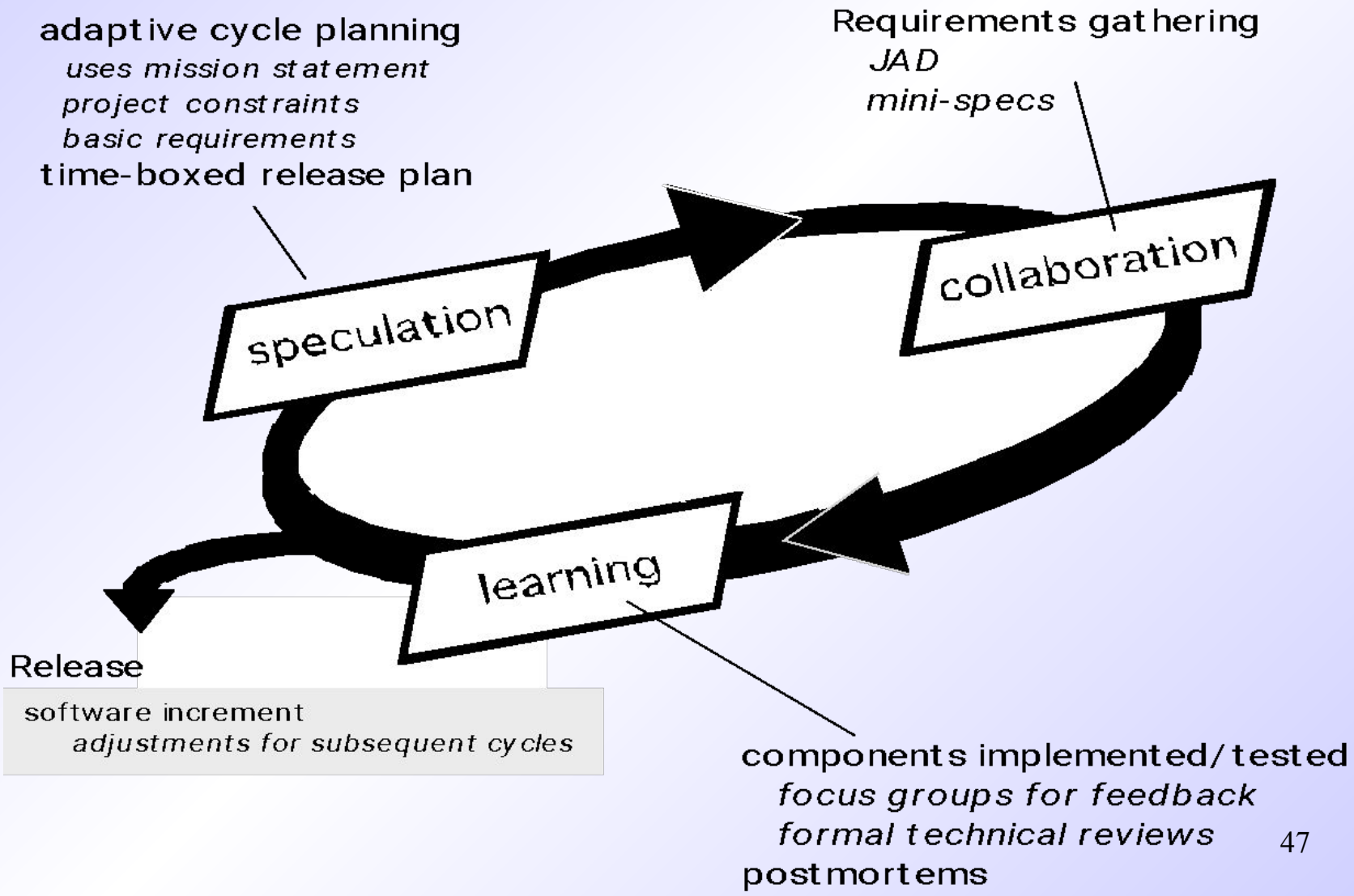
- **Speculation**

✔ project initiated- customer mission statement , delivery dates , requirements specified

✔ adaptive cycle planning takes place- requirements will keep changing and those changes to be adapted.

- **Collaboration** (requires teamwork from a jelled team, **joint application development** is preferred requirements gathering approach)
- **Learning** (components implemented and testes, focus groups provide feedback, formal technical reviews, postmortems)

Adaptive Software Development



Dynamic Systems Development Method

- Provides a framework for building and maintaining systems which meet **tight time constraints** using incremental prototyping in a controlled environment.
- Uses Pareto principle (80% of project can be delivered in 20% required to deliver the entire project)
- Each increment only delivers enough functionality to move to the next increment.
- Uses time boxes to fix time and resources to determine how much functionality will be delivered in each increment

DSDM- distinguishing features

- Active user Involvement is Imperative
- Teams Must be Empowered to Make Decisions
- Focus on Frequent Delivery
- Fitness for Business is Criterion for Accepted Deliverables
- Iterative and Incremental Development is Mandatory
- All Changes During Development Must Be Reversible

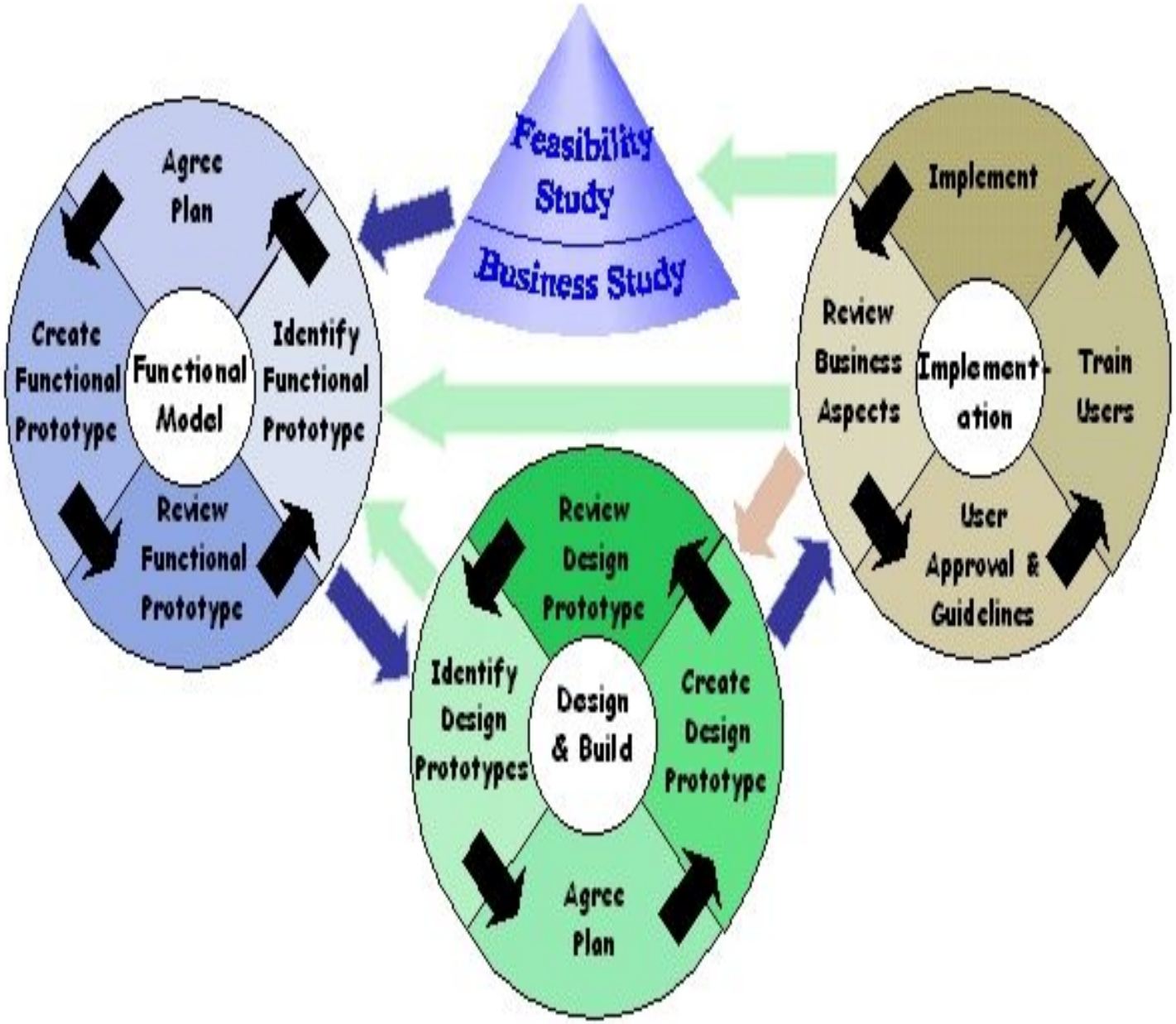
DSDM- distinguishing features

- **Requirements are Baselined at High-Level-** to **limit the degree of freedom** to which requirements can be altered during the development process, some high-level requirements need to be established.

- Testing is Integrated Throughout the Lifecycle

- **Collaborative and Co-operative Approach-**encouraging collaboration of technical staff and business staff in a project is mandatory during DSDM projects, because co-operation is crucial to succeed in a DSDM project.

The DSDM Development Process



DSDM life cycle activities

- **Feasibility study**—establishes the **basic business requirements** and constraints associated with the application to be built and then **assesses** whether the application is a viable candidate for the DSDM process.

- **Business study**—establishes the **functional and information requirements** that will allow the application to provide **business value**

DSDM life cycle activities

- **Functional model iteration**

✔ produces a set of **incremental prototypes** that demonstrate functionality for the customer.

✔ Additional requirements

- **Design and build iteration—revisits prototypes** built during functional model iteration to ensure that each has been engineered in a manner that will enable it to provide operational business value for end users

DSDM life cycle activities

- **Implementation**—places the **latest software increment** (an "operationalized" prototype) into the operational environment.

Scrum

- Scrum Principles
- Scrum's goal is facilitating the self-organization of the team so that it can adapt to

✔ the specifics of the project and

✔ their changes over time

- Scrum is an agile software development process model based on multiple **small teams** working in an **intensive and interdependent manner.**

- Process must be **adaptable to both technical and business challenges** to ensure best product produced.

- Process yields **frequent increments** that can be inspected, adjusted, tested, documented and built on

Scrum

- Development work is partitioned into “packets”
- Testing and documentation are on-going as the product is constructed.
- Work occurs in “sprints” and is derived from a “backlog” of existing requirements.

Scrum

- **Backlog**

✔ prioritized list of project requirements.

✔ Items can be added to the backlog.

✔ Updates priorities.

- 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.

Scrum

- Sprint

✔ Changes are not allowed during sprint.

✔ Allows team members to work in short and stable environment.

- Scrum meetings

✔ are short (typically 15 minutes) meetings held daily by the Scrum team to check the progress of ongoing work.

Scrum

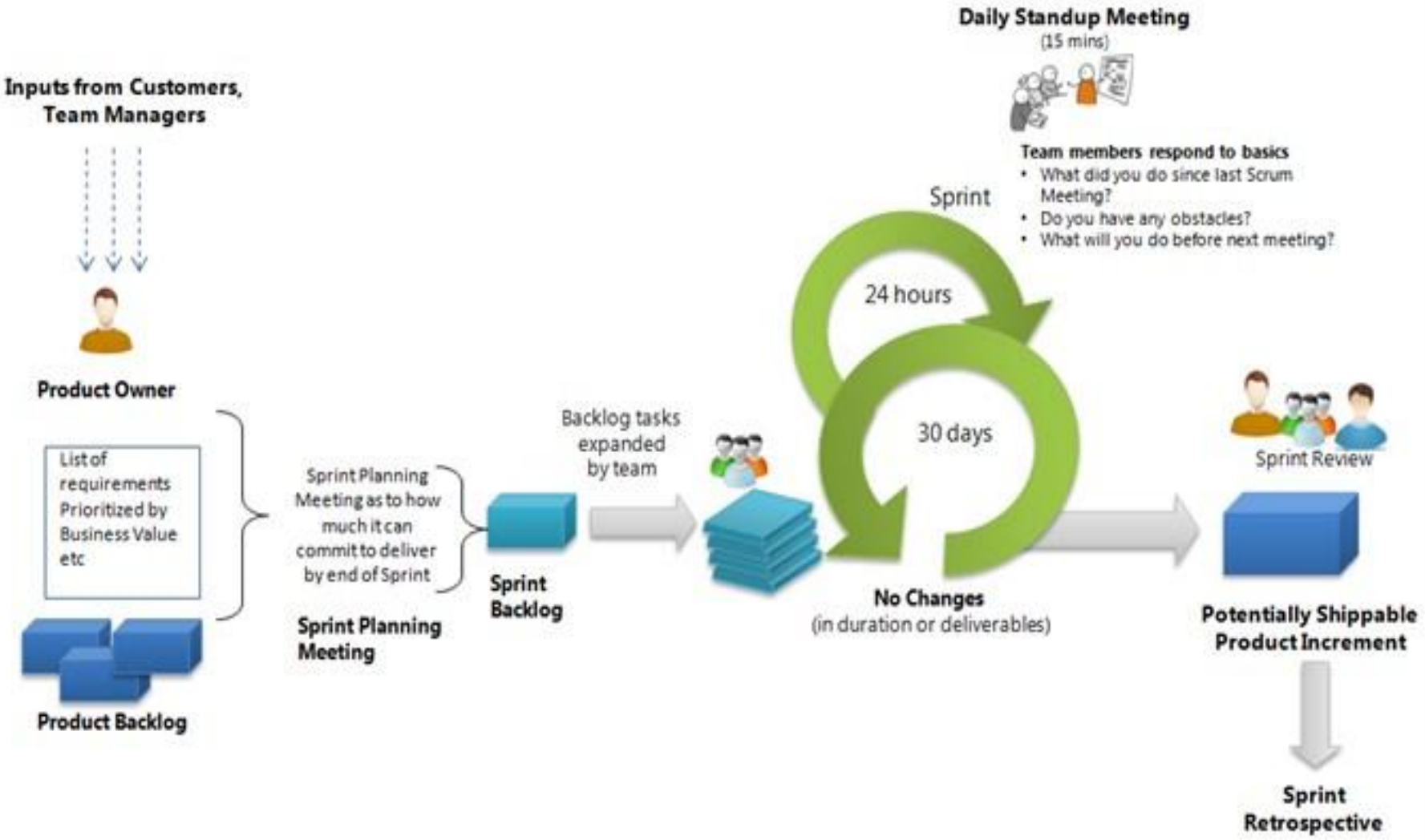
- Demos

✔ deliver the software increment to the customer so that functionality that has been implemented can be **demonstrated and evaluated by the customer.**

✔ May not contain all planned functions but delivery within time-box.

- **Scrum process patterns enable a software team to work successfully in a world where the elimination of uncertainty is impossible.**

Agile Scrum Methodology



Crystal

- Crystal is meant to be a **family of methods** for **different project sizes** and criticalities.

- The Crystal methodology is one of the most **lightweight, adaptable** approaches to software development.

- Crystal is actually comprised of a **family of agile methodologies**, whose unique characteristics are driven by several factors such as **team size, system criticality, and project priorities.**

Crystal

- Key characteristics of Crystal include **teamwork, communication, and simplicity**, as well as reflection to frequently adjust and improve the process.

- primary goal of **delivering useful**, working software and a secondary goal of setting up for the next game

		Crystal Methodologies				
		Clear	Yellow	Orange	Red	Maroon
Criticality of the Project	Life (L)	L6	L20	L40	L80	L200
	Essential Money (E)	E6	E20	E40	E80	E200
	Discretionary Money (D)	D6	D20	D40	D80	D200
	Comfort (C)	C6	C20	C40	C80	C200
		1 to 6	7 to 20	21 to 40	41 to 80	81 to 200
		Number of People involved in the Project				

Feature Driven Development

- FDD—distinguishing features
- Emphasis is on defining “features”
- a feature “is a client-valued function that can be implemented in two weeks or less.”
- A feature is a small, client-valued function expressed in the form <action><result><object>.
- A features list is created and “plan by feature” is conducted
- Design and construction merge in FDD
- Emphasizes collaboration among team members

Feature Driven Development

- Manages problem and project complexity using feature-based decomposition followed integration of software increments.

- Technical communication using verbal, graphical, and textual means.

- Software quality encouraged by using incremental development, design and code inspections, SQA audits, metric collection, and use of patterns (analysis, design, construction)

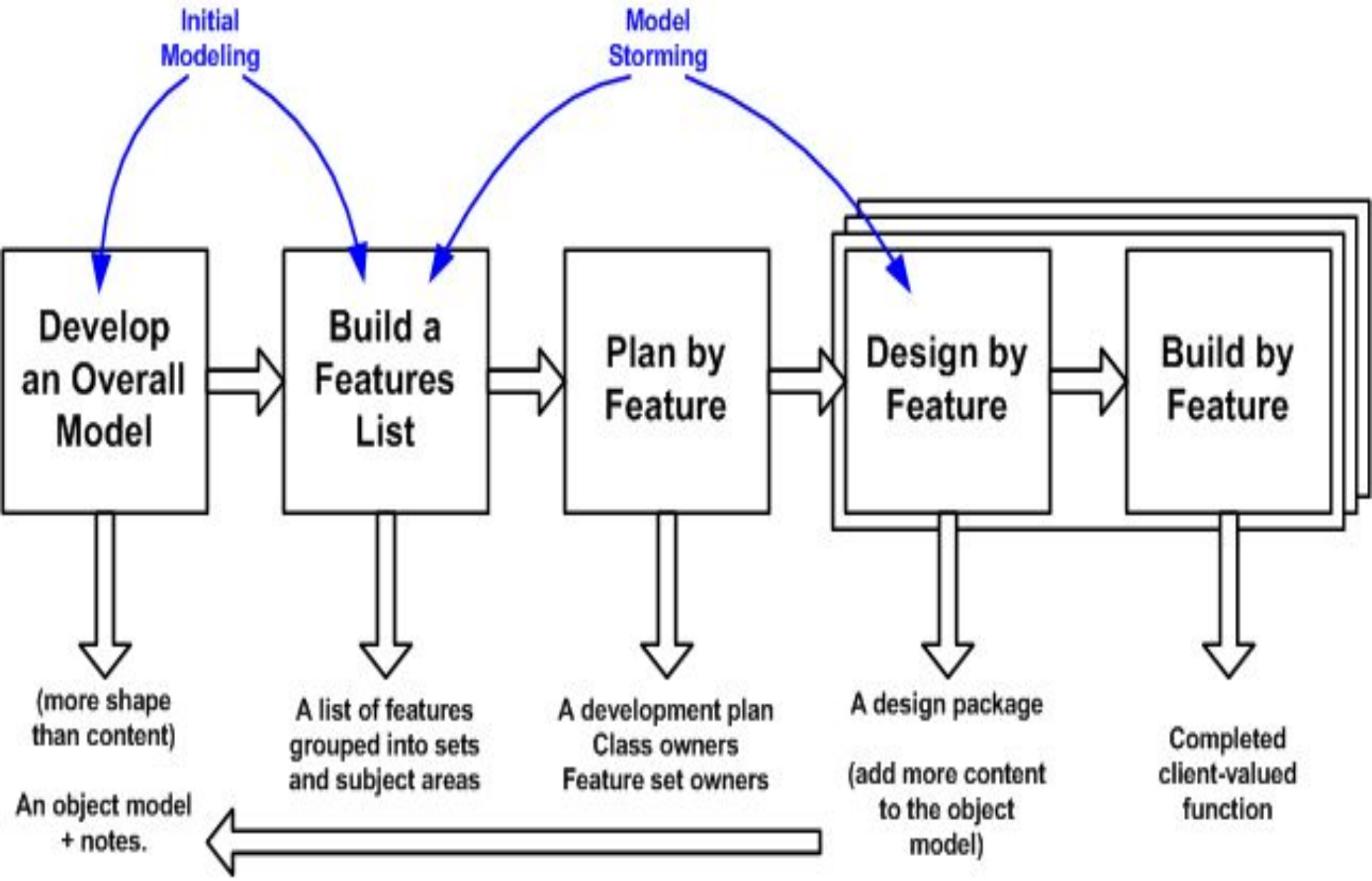
Feature Driven Development

- **Develop overall model** (contains set of classes depicting business model of application to be built)

- **Build features list** (features extracted from domain model, features are categorized and prioritized, work is broken up into two week chunks)

Feature Driven Development

- **Plan by feature** (features assessed based on priority, effort, technical issues, schedule dependencies)
- **Design by feature** (classes relevant to feature are choosen, class and method prologs are written, preliminary design detail developed, owner assigned to each class, owner responsible for maintaining design document for his or her own work packages)
- **Build by feature** (class owner translates design into source code and performs unit testing, integration performed by chief programmer)



Agile Modeling

- Agile Modeling (AM) is a practice-based methodology for effective modeling and documentation of software-based systems.

Agile Modeling

- **Modeling principles**
- **Model with a purpose:** A developer who uses AM should have a specific goal in mind before creating the model.
- **Use multiple models:**

✔ There are many different models and notations that can be used to describe software.

✔ Only a small subset is essential for most projects.

- **Travel light.** As software engineering work proceeds, keep only those models that will provide long-term value for designing various applications.

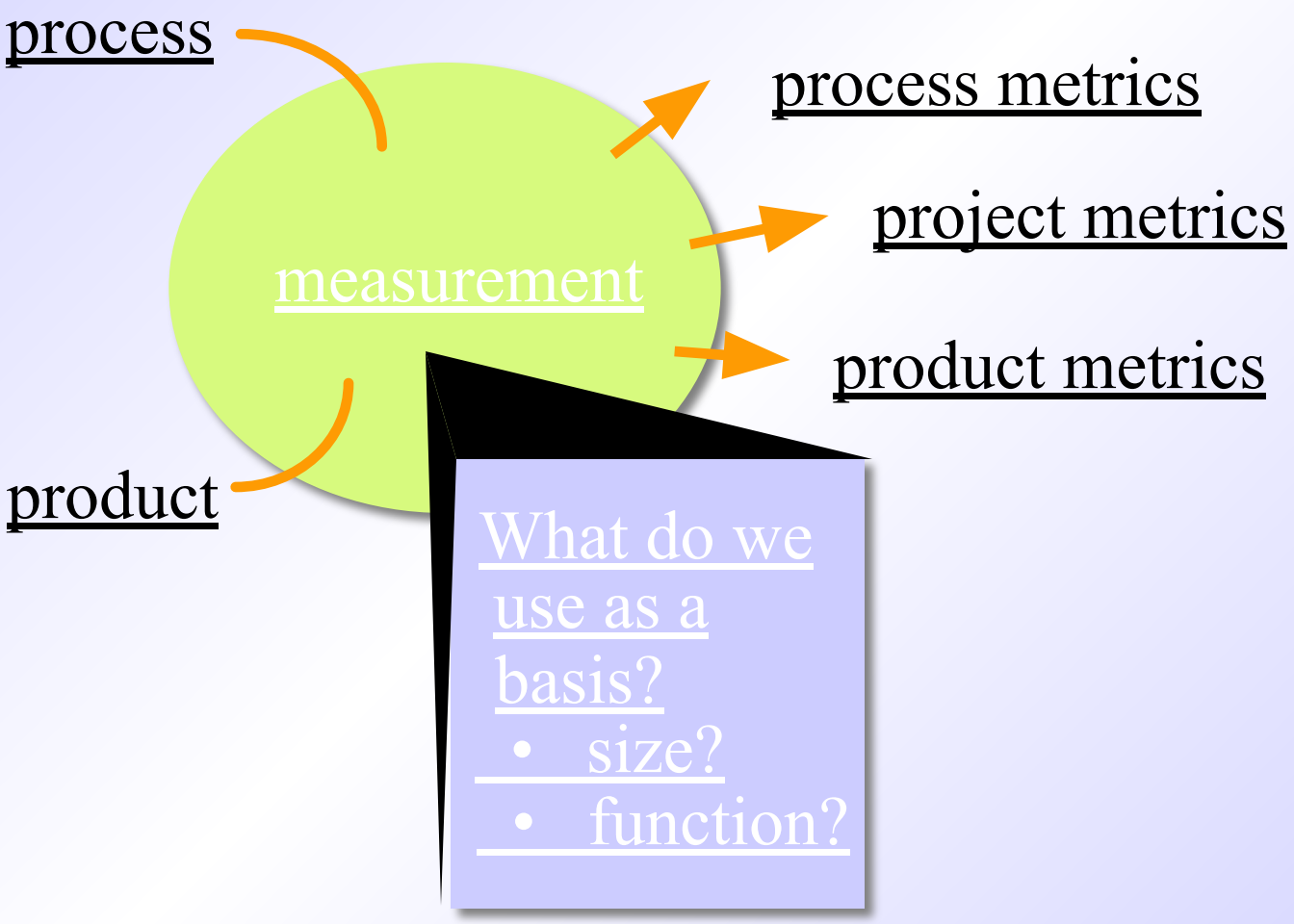
Agile Modeling

- **Content is more important than representation.** Modeling should impart information to its intended audience.

- **Know the models and the tools you use to create them.** Understand the strengths and weaknesses of each model and the tools that are used to create it.

- **Adapt locally.** The modeling approach should be adapted to the needs of the agile team.

A Good Manager Measures



What is a Process?

- A **system of operations** in producing something; a series of actions, changes, or functions that achieve an end or a result.
- A **sequence of steps** performed for a given purpose.
- Process measurement measures the efficacy of a software process indirectly.
  - That is, we derive a set of metrics based on the outcomes that can be derived from the process.
  - Outcomes include
    - measures of errors uncovered before release of the software
    - defects delivered to and reported by end-users
    - work products delivered (productivity)
    - human effort expended
    - calendar time expended
    - schedule conformance
    - other measures.

What is a Software Process?

- A **set of activities, methods, practices, and transformations** that people use to **develop and maintain software** and the associated products (e.g., project plans, design documents, code, test cases, and user manuals) is a software process.

- As an **organization matures**, the **software process becomes better defined** and **more consistently implemented** throughout the organization.

- Software process maturity is the extent (degree) to which a specific process is explicitly defined, managed, measured, controlled, and effective.

- Software process metrics can provide **significant benefit** as an organization works to **improve its overall level of process maturity**.

Process Metrics

- **Time taken** for a particular process to be completed.

✔ This can be total time devoted to the process , calendar time , time spent on process by engineers etc.

- The **resources required** for a particular for particular process.

✔ Resources might include efforts in person-days , travel costs , or computer resources.

- The **number of occurrences** of a particular event.

✔ Events might be monitored include the number of **defects discovered** during code inspection , number of **requirement changes** requested , average number of lines of **code modified** in response to requirement change

Process Metrics

- **Reuse data**
  - ✔ The number of components produced and their degree of reusability.

- **Quality-related**
  - ✔ focus on quality of work products and deliverables

Immature Software Organizations

- Software processes are generally improvised
- If a process is specified, it is not rigorously followed or enforced
- The software organization is reactionary
- Managers only focus on solving immediate (crisis) problems
- Schedules and budgets are routinely exceeded because they are not based on realistic estimates
- When hard deadlines are imposed, product functionality and quality are often compromised
- There is no basis for judging process quality or for solving product or process problems
- Activities such as reviews and testing are curtailed or eliminated when projects fall behind schedule

Capability Maturity Model (SW-CMM)

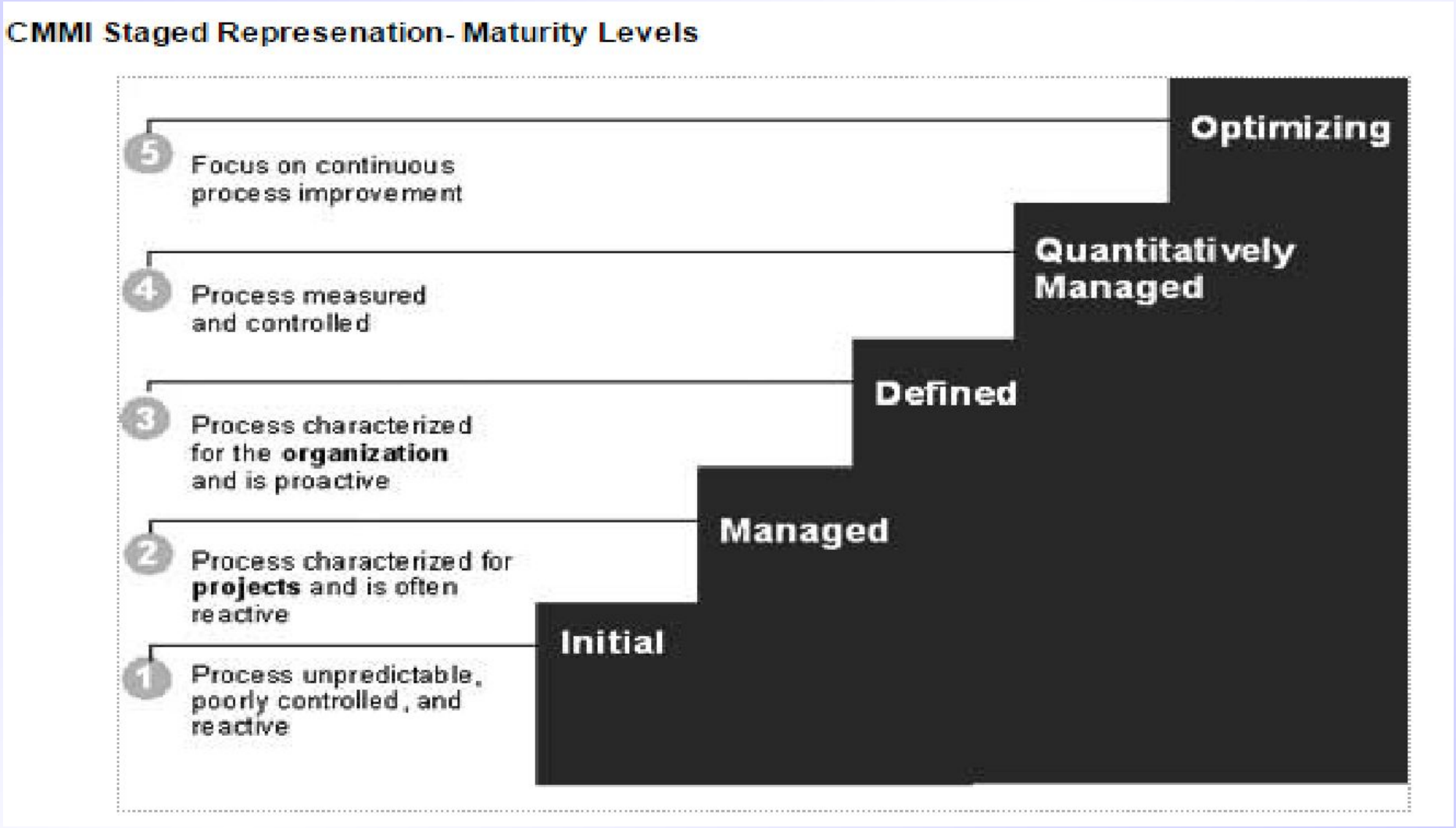
- CMMI (Capability Maturity Model Integration) is a proven industry framework to improve product quality and development efficiency for both hardware and software.
- CMMI, staged, uses 5 levels to describe the maturity of the organization.
- Each maturity level has an associated set of process area and generic goals.
- The maturity levels are measured by the achievement of the **specific** and **generic goals** that apply to each predefined set of process areas.

Capability - Maturity

- A **maturity level** is a well-defined evolutionary plateau toward achieving a mature software process. Each maturity level provides a layer in the foundation for continuous process improvement.

- A **capability level** is a well-defined evolutionary plateau describing the organization's capability relative to a process area. A capability level consists of related specific and generic practices for a process area that can improve the organization's processes associated with that process area. Each level is a layer in the foundation for continuous process improvement.

Five Levels of Software Process Maturity



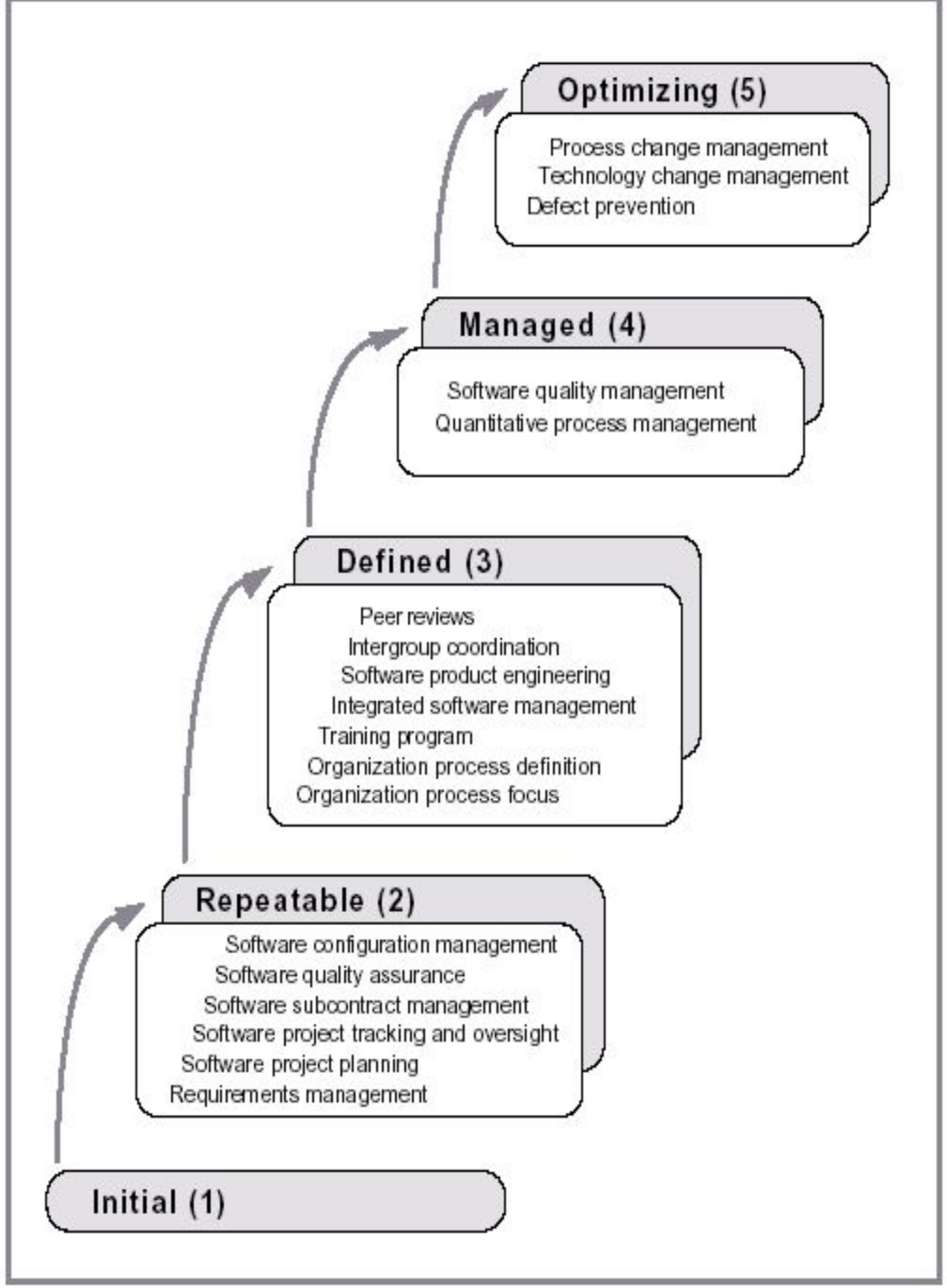


Figure 3.2 The Key Process Areas by Maturity Level

Characteristics of Each Level

- **Initial Level (Level 1)**
  - Characterized as **ad-hoc**, and occasionally even **chaotic** processes.
  - **Few processes** are defined, and **success** depends on **individual effort**.
  - Maturity level 1 organizations often produce products and services that work; however, they frequently exceed the budget and schedule of their projects.
  - Maturity level 1 organizations are characterized by a tendency to over commit, abandon processes in the time of crisis, and not be able to repeat their past successes.

Characteristics of Each Level (continued)

- **Managed Level (level 2)**
- At maturity level 2, an organization has achieved all the **specific** and **generic goals** of the maturity level 2 process areas.

- The process discipline reflected by maturity level 2 helps to ensure that **existing practices** are retained during times of stress. When these practices are in place, projects are performed and managed according to their documented plans.

- At maturity level 2, requirements, processes, work products, and services are managed. The status of the work products and the delivery of services are visible to management at defined points.

- The **work products and services satisfy their specified requirements**, standards, and objectives.

Characteristics of Each Level (continued)

- **Defined (Level 3)**
  - The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization.
  - All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.
  - A critical distinction between maturity level 2 and maturity level 3 is the scope of standards, process descriptions, and procedures.
  - At maturity level 2, the standards, process descriptions, and procedures may be quite different in each specific instance of the process (for example, on a particular project).

Characteristics of Each Level (continued)

- At maturity level 3, the standards, process descriptions, and procedures for a project are **tailored** from the **organization's set of standard processes** to suit a particular project or organizational unit.

- The organization's set of standard processes includes the processes addressed at maturity level 2 and maturity level 3.

- Another critical distinction is that at maturity level 3, processes are typically described in more detail and more rigorously than at maturity level 2

Characteristics of Each Level (continued)

- **Quantitatively Managed (Level 4)**

✔ At maturity level 4, an organization has achieved all the **specific goals** of the process areas assigned to maturity levels 2, 3, and 4 and the **generic goals** assigned to maturity levels 2 and 3.

✔ At maturity level 4 Sub processes are selected that significantly contribute to overall process performance. These selected sub processes are controlled using statistical and other quantitative techniques.

✔ Quantitative objectives for quality and process performance are established and used as criteria in managing processes.

✔ Quantitative objectives are based on the needs of the customer, end users, organization, and process implementers.

Characteristics of Each Level (continued)

- ✔ detailed measures of process performance are collected and statistically analyzed.
- ✔ Special causes of process variation are identified and, where appropriate, the sources of special causes are corrected to prevent future occurrences.
- ✔ A critical distinction between maturity level 3 and maturity level 4 is the predictability of process performance.
- ✔ At maturity level 4, the performance of processes is controlled using statistical and other quantitative techniques, and is quantitatively predictable. At maturity level 3, processes are only qualitatively predictable.

Characteristics of Each Level (continued)

- Optimizing **(Level 5)**

✔ At maturity level 5, an organization has achieved all the **specific goals** of the process areas assigned to maturity levels 2, 3, 4, and 5 and the **generic goals** assigned to maturity levels 2 and 3.

✔ Maturity level 5 focuses on continually improving process performance through both incremental and innovative technological improvements.

✔ At maturity level 4, processes are concerned with addressing special causes.

✔ At maturity level 5, processes are concerned with addressing common causes.

CMMI- Capability Levels

- **Level 0 (Incomplete)**

Process area is either not formed or does not achieve goals and objectives defined by CMMI.

- **Level 1(Performed)**

✔ All specific goals of CMMI satisfied.

✔ Work tasks completed for defined work.

- **Level 2 (Managed)**

✔ Level 1 criteria satisfied.

✔ Work done on process area conforms to organization's defined policy

CMMI- Capability Levels

- ✔ Availability of adequate resources.
- ✔ Stakeholders are actively involved.
- ✔ All work tasks are evaluated and adhere to CMMI standards.

Level 3 (Defined)

- ✔ All level 2 criteria satisfied.
- ✔ Processes are tailored from organization's set of standard guidelines.

Level 4 (Quantitatively managed)

- ✔ All level 3 criteria satisfied.
- ✔ Quantitative objectives for quality and process performance are established and used as criteria in managing process.

CMMI- Capability Levels

Level 5 (Optimized)

✔All level 4 criteria satisfied.

✔Process area is optimized to meet changing customer requirements and continually improve the efficiency of the software product

The CMM Structure

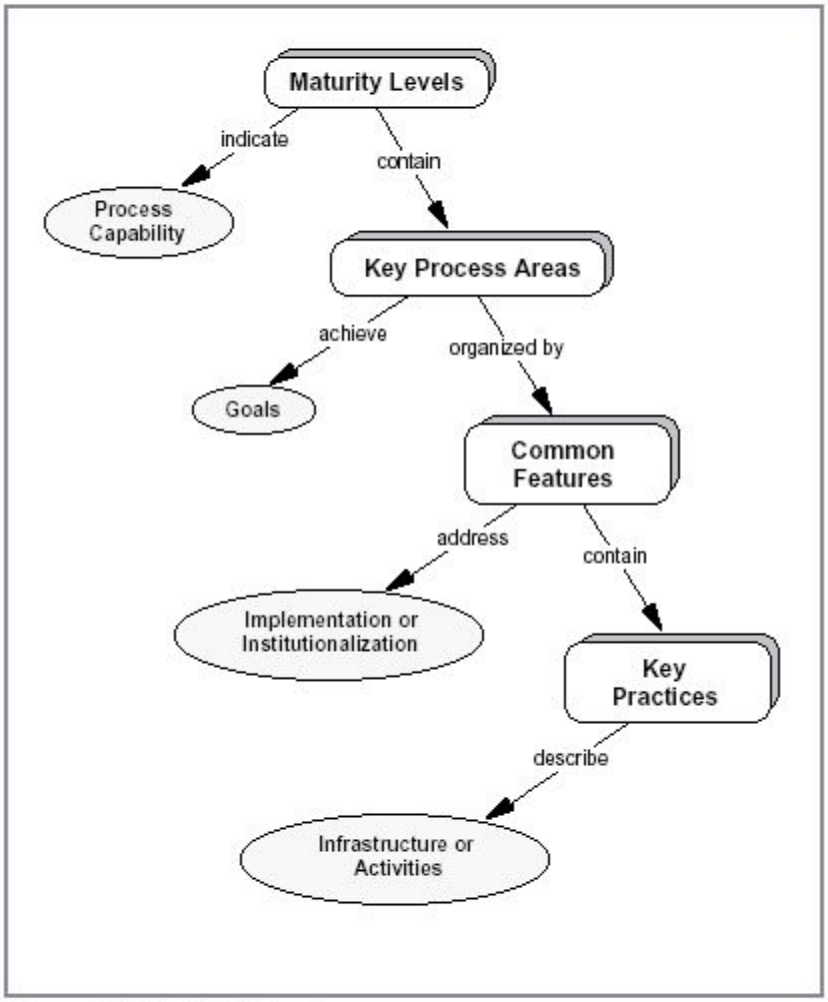


Figure 3.1 The CMM Structure

Software Project Planning

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

- Why?

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

Project Planning-Estimation

- Objective of software **project planning** is to provide a **framework** that enables the manager to make reasonable **estimates** of resources, cost, and schedule.

- Estimates should attempt to define best-case and worst-case scenarios so that project outcomes can be bounded.

- Plan must be adapted and updated as the project proceeds.

Project Estimation

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

Product Metrics

- Product metrics of computer software provide us with a systematic way to assess quality based on a set of clearly defined rules.

- Product metrics help in assessing efficiency, reliability, complexity, understandability and maintainability of sot ware product.

Function Point (FP)

- The function point (FP) metric can be used effectively as a means for **measuring the functionality** delivered by a system.

- The **Function Point (FP)** metric can be used effectively as a means for **measuring the functionality** delivered by a **system.**

- Function point metrics, **measure functionality** from the **users point of view**, that is, on the basis of what the user requests and receives in return

Function Point (FP)

- FP metric can then be used to:

(1) estimate the **cost or effort** required to design code and test the software.

(2) predict the **number of errors** that will be encountered during testing.

(3) **forecast** the number of components and/or the number of projected source lines in the implemented system.

Function Point (FP)

- Information domain values for measurement of FP:

1. **Number of external inputs (EIs)**

✔ An EI processes data or control information that comes from outside the application's boundary.

✔ Inputs should be distinguished from inquiries, which are counted separately.

2. **Number of external outputs (EOs).**

✔ An EO is an elementary process that generates data or control information sent outside the application's boundary

✔ Each external output is derived data within the application that provides information to the user.

Function Point (FP)

- external output refers to reports, screens, error messages, etc.

3. Number of external inquiries (EQs).

✔ External Inquiry (EQ) is a transaction function with both input and output components that result in data retrieval.

✔ An external inquiry is defined as an online input that results in the generation of some immediate software response in the form of an online output.

Function Point (FP)

4. Number of internal logical files (ILFs).

✔Internal Logical File (ILF) is a user identifiable group of logically related data or control information that resides entirely within the application boundary.

✔Maintained via external inputs (EI).

5. Number of external interface files (EIFs).

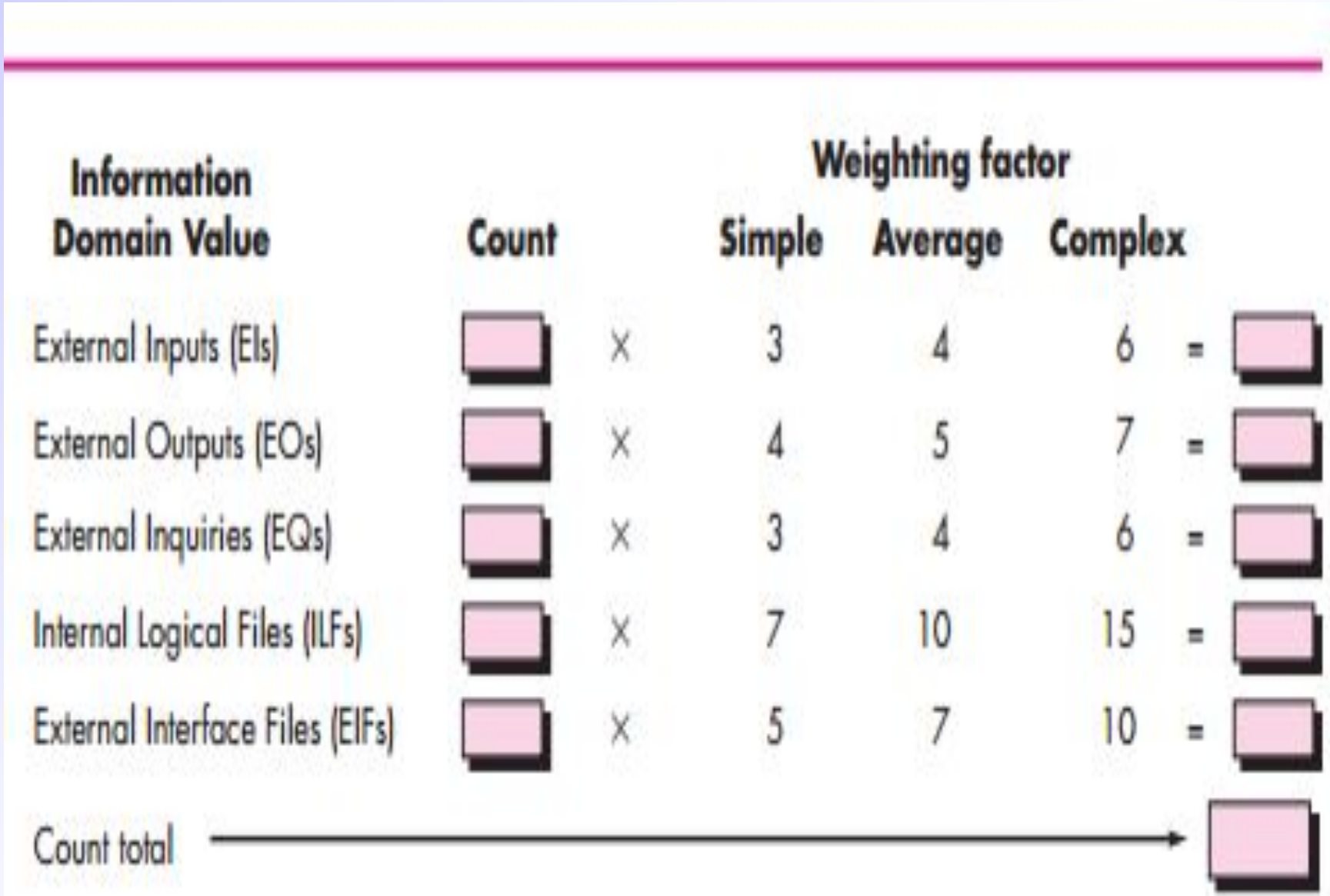
✔the data resides entirely outside the application and is maintained by another application. The external interface file is an internal logical file for another application.

Function Point (FP)

- To compute function points (FP), the following relationship is used:

$$FP = \text{count total} \times [0.65 + 0.01 \times \Sigma (F_i)]$$

- where count total is the sum of all FP entries

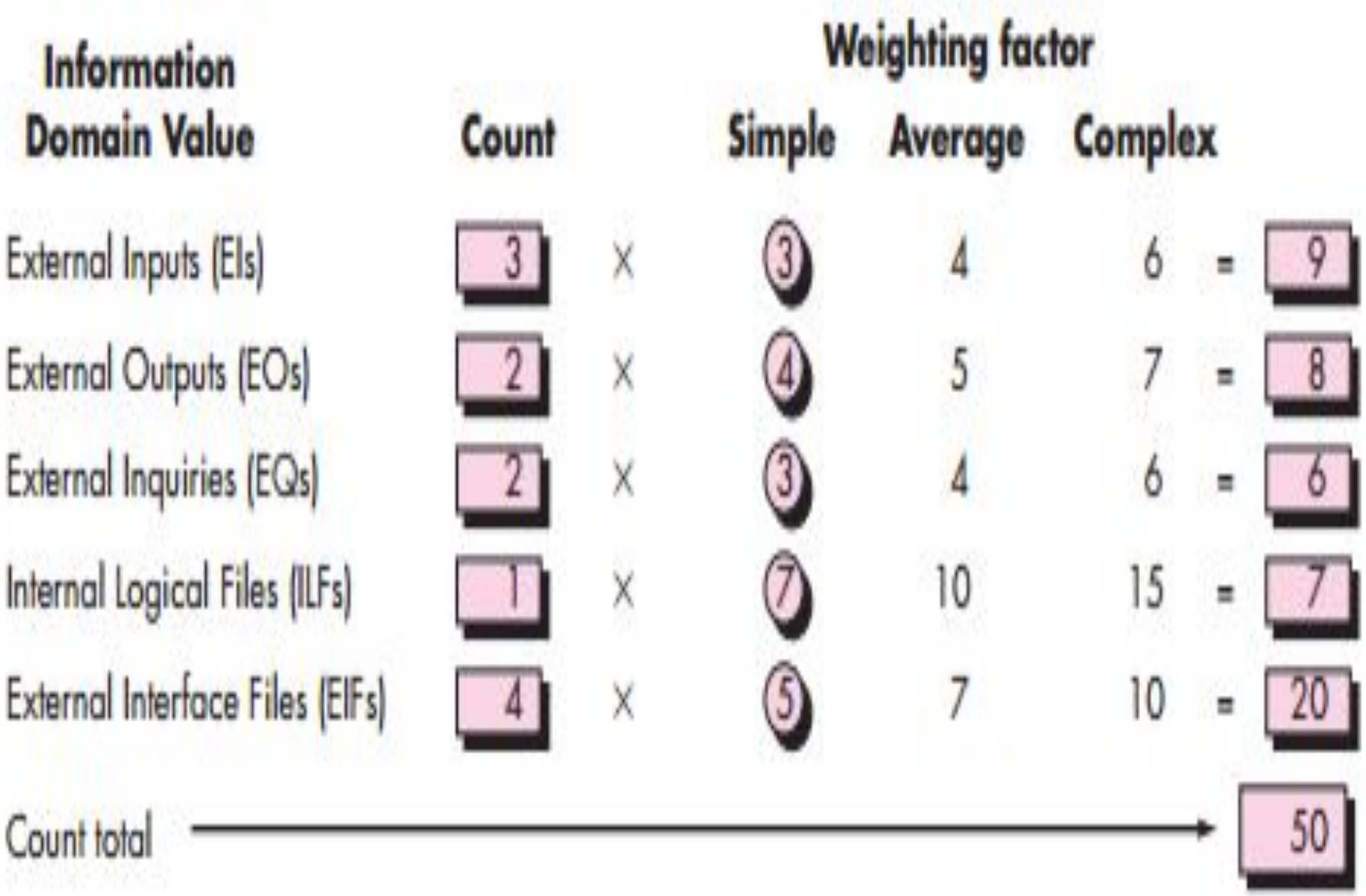

Information Domain Value
Count
Weighting factor
Simple
Average
Complex
External Inputs (EIs)
X
3
4
6
=
External Outputs (EOs)
X
4
5
7
=
External Inquiries (EQs)
X
3
4
6
=
Internal Logical Files (ILFs)
X
7
10
15
=
External Interface Files (EIFs)
X
5
7
10
=
Count total

The F_i (i = 1 to 14) are *value adjustment factors* (VAF) based on responses to the following questions [Lon02]:

1. Does the system require reliable backup and recovery?
2. Are specialized data communications required to transfer information to or from the application?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require online data entry?
7. Does the online data entry require the input transaction to be built over multiple screens or operations?
8. Are the ILFs updated online?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

Function Point (FP)

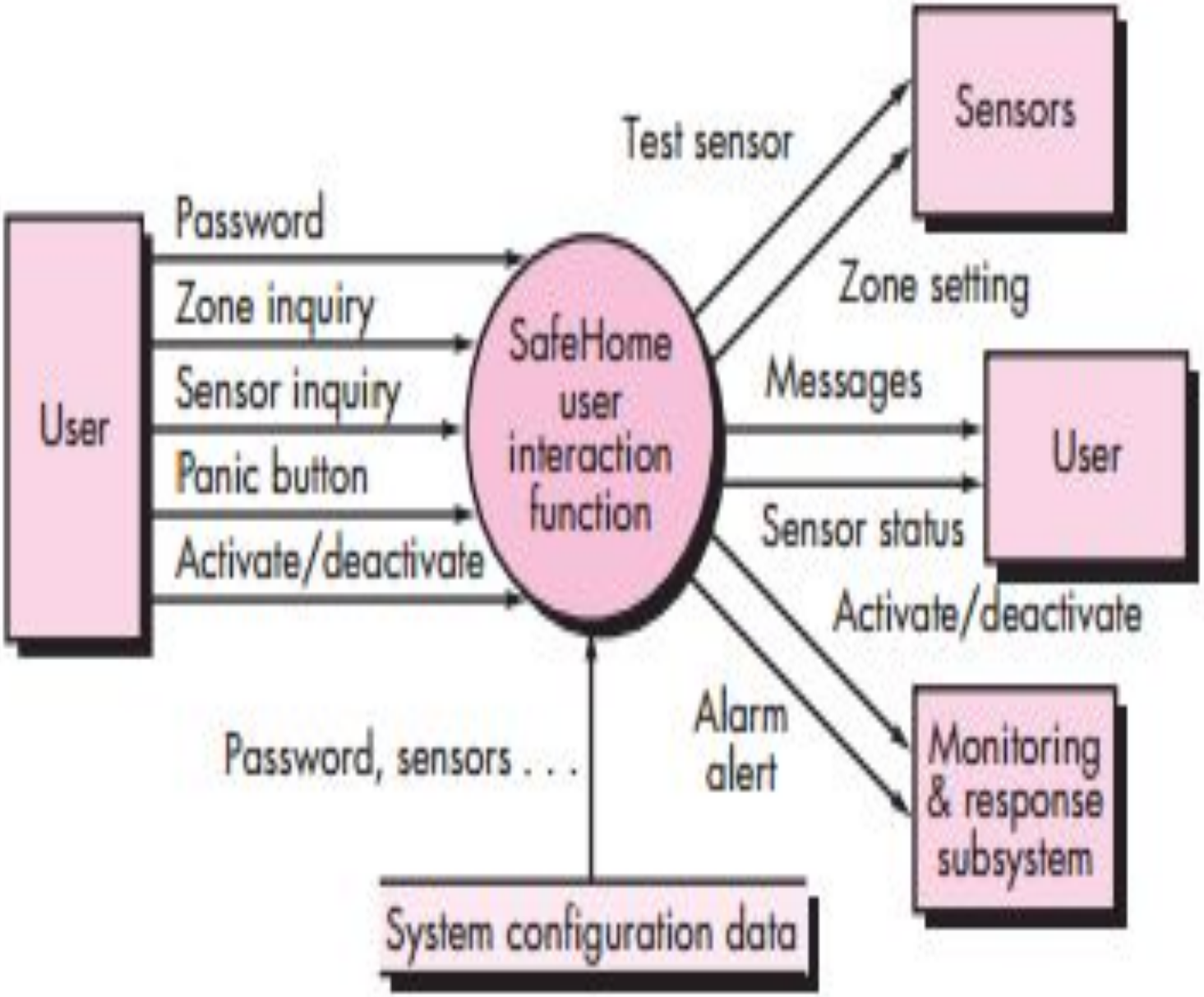
- Each of these questions is answered for VAF using a scale that ranges from 0 (not important or applicable) to 5 (absolutely essential).


Information Domain Value
Count
Weighting factor
Simple
Average
Complex
External Inputs (EIs)
3
x
3
4
6
=
9
External Outputs (EOs)
2
x
4
5
7
=
8
External Inquiries (EQs)
2
x
3
4
6
=
6
Internal Logical Files (ILFs)
1
x
7
10
15
=
7
External Interface Files (EIFs)
4
x
5
7
10
=
20
Count total
50

Function Point (FP)

$$FP = 50 \times [0.65 + (0.01 \times 46)] = 56$$

- Example FP: ..\..\Desktop\Example FP.docx


Sensors
Test sensor
Password
Zone setting
Zone inquiry
SafeHome
user
interaction
function
Messages
User
Sensor inquiry
User
Panic button
Sensor status
Activate/deactivate
Activate/deactivate
Alarm
alert
Monitoring
& response
subsystem
Password, sensors . . .
System configuration data

Line Of Code (LOC)

- LOC is a software metric used to measure the size of computer program by counting number of lines in the text of program's source code.
- Lines used for commenting code and header files are ignored.
- LOC is typically used to predict the amount of effort that will be required to develop a program as well as to estimate programming productivity or maintainability once the software product is developed.

Line Of Code (LOC)

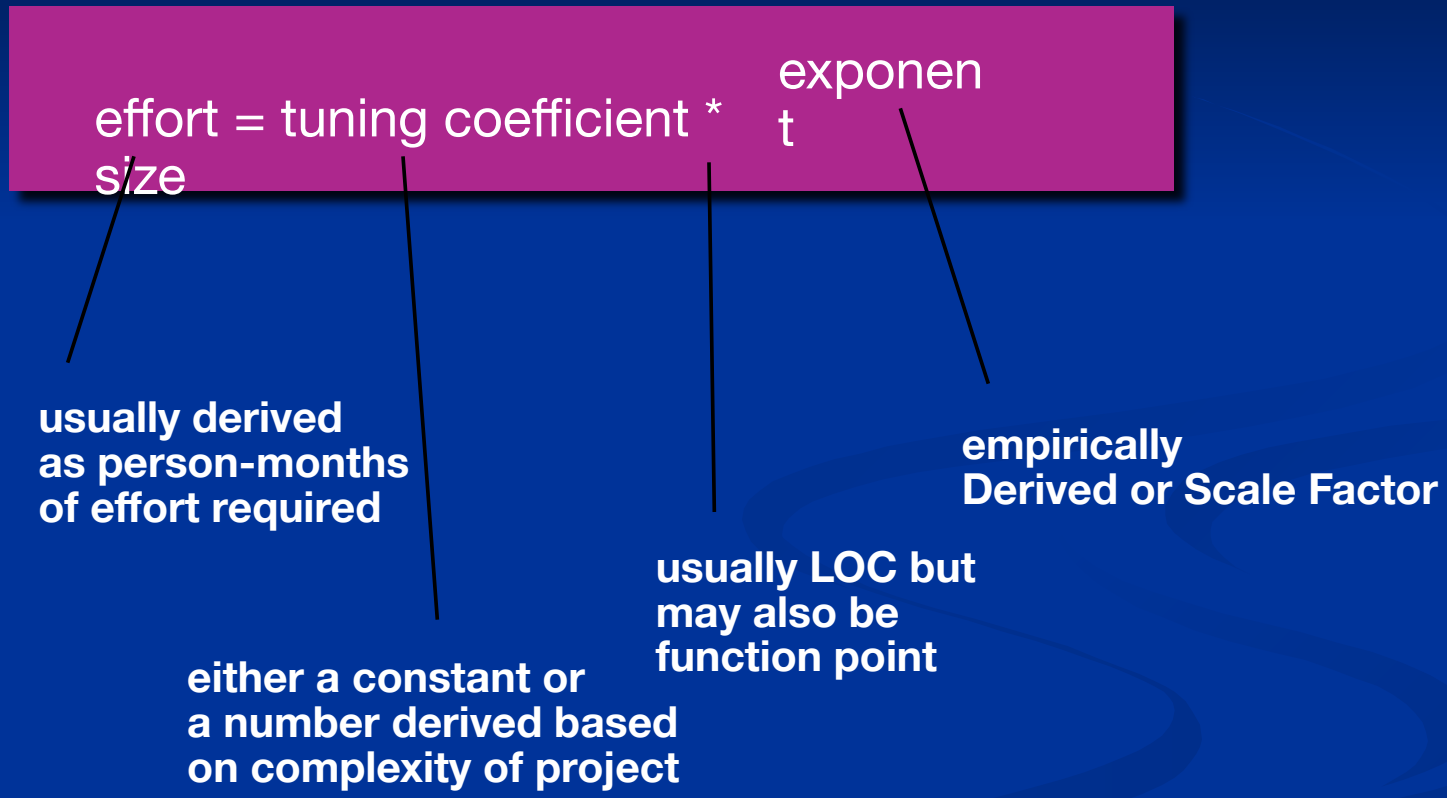
- Two major types of LOC:

1. Physical LOC: count text
2. Logical LOC: count number of executable statements

LOC example: ..\..\Desktop\Example LOC.docx

Empirical Estimation Models

General form:



COCOMO(Constructive Cost Model)

- 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.

COCOMO(Constructive Cost Model)

- 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**
- **lines of source code**

COCOMO(Constructive Cost Model)

- **object point** is an indirect software measure that is computed using counts of the number

(1) screens (at the user interface)

(2) reports

(3) components likely to be required to build the application

- Each object instance (e.g., a screen or report) is classified into one of three complexity levels (i.e. Simple, medium, or difficult) using criteria suggested

COCOMO(Constructive Cost Model)

- **New Object Point (NOP)**

$$NOP = (\text{object points}) \times [(100 - \%\text{reuse})/100]$$

COCOMO(Constructive Cost Model)

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

- The **object point count** is then determined by multiplying the original number of object instances by the weighting factor and summing to obtain a total object point count.

COCOMO (Constructive Cost Model)

- estimate of effort is based on the computed NOP value, a “productivity rate” s derived

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

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

The Software Equation

A dynamic multivariable model

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

where

E = effort in person-months or person-years

t = project duration in months or years

B = "special skills factor"

P = "productivity parameter"

The Software Equation

- P = 2,000 for development of real-time embedded software.
- P = 10,000 for telecommunication and systems software.
- P = 28,000 for business systems applications

The Software Equation

- B=0.16 for programs with LOC 5 to 15
- B=0.35 for programs with LOC greater than 70

Estimation

- To simplify the estimation process set of equations derived from the software equation are stated:
- Minimum development time

$$t_{min} = 8.14 \frac{LOC}{p^{0.43}} \text{ in months for } t_{min} > 6 \text{ months}$$

Estimation

- Estimated Effort

$$E = 180\,Bt^3 \text{ in person-months for } E \geq 20 \text{ person-months}$$

$$t_{min} = 8.14\ (33200/12000)^{0.43}$$
$$t_{min} = 12.6 \text{ calendar months}$$
$$E = 180 \times 0.28 \times (1.05)^3$$
$$E = 58 \text{ person-months}$$

Why Are Projects Late?

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

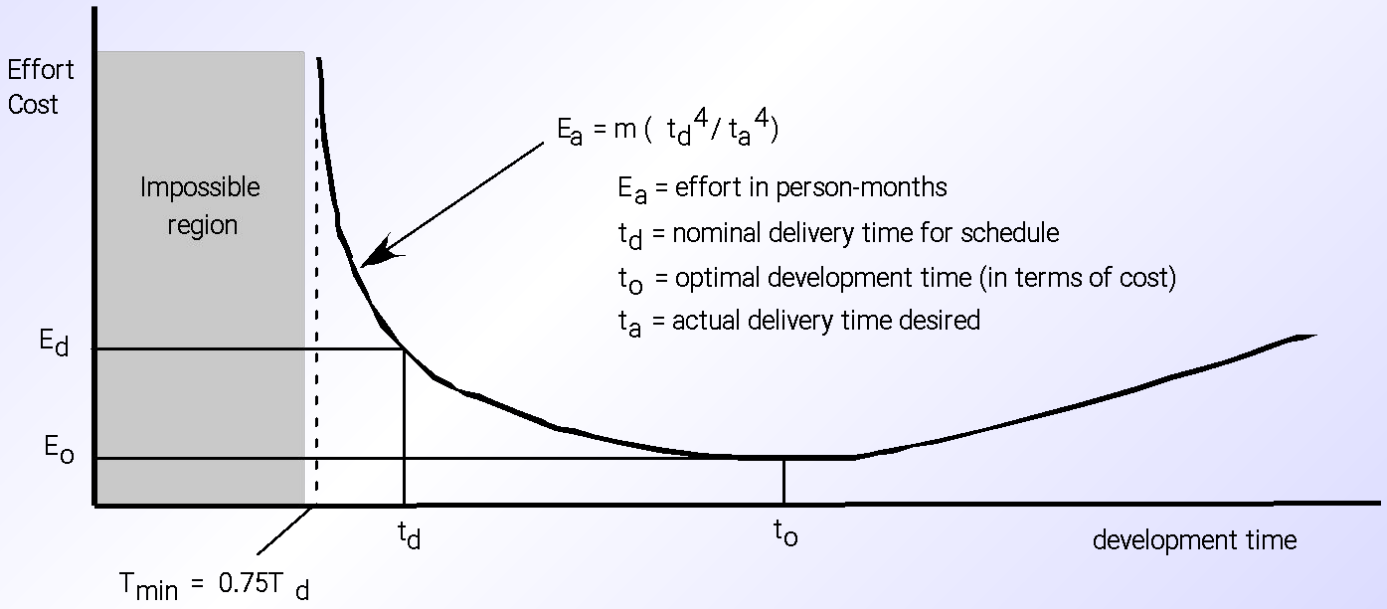
Scheduling

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

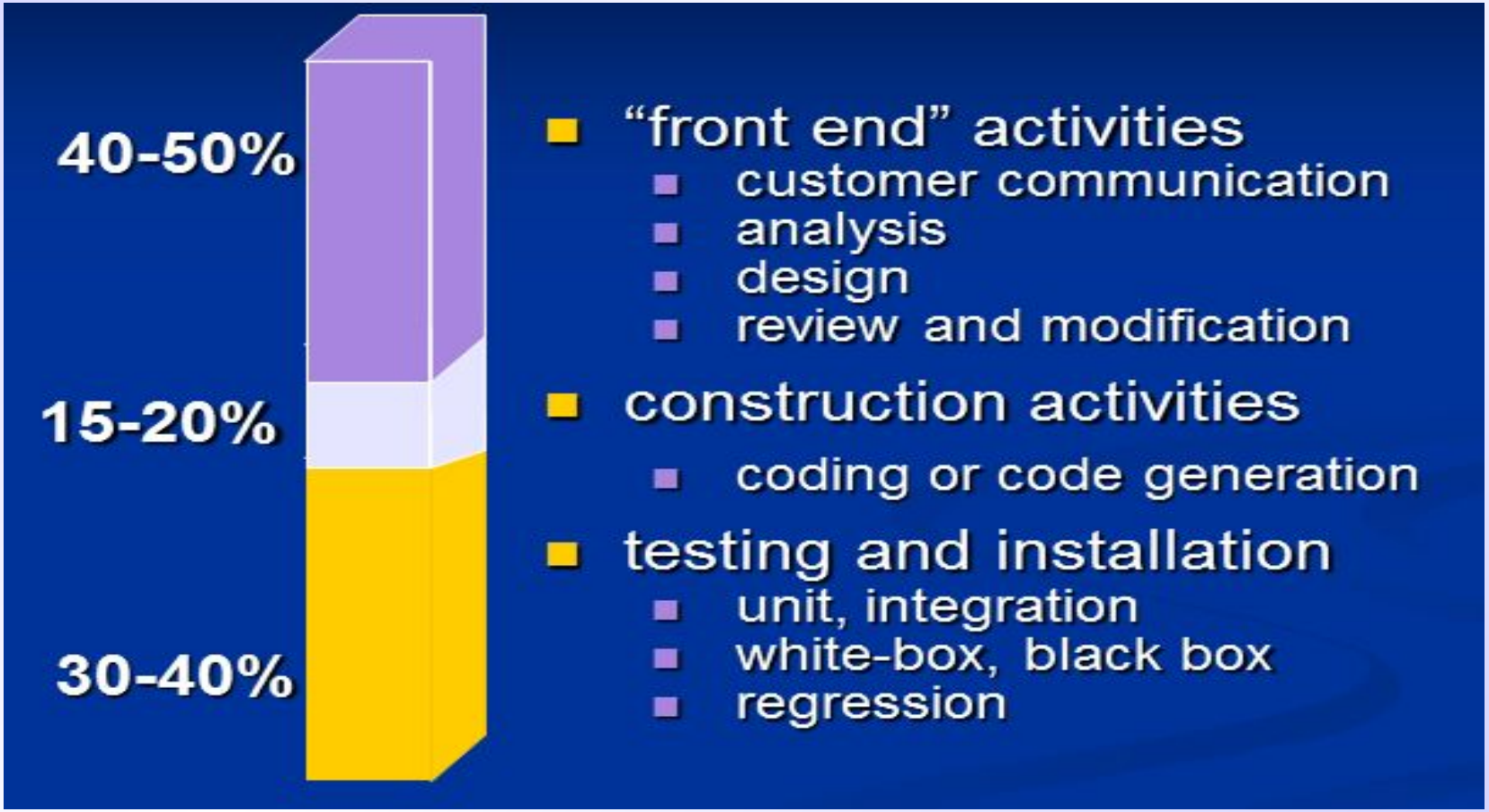
Scheduling Principles

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

Effort and Delivery Time



Effort Allocation (40-20-40 rule)



Defining Task Network

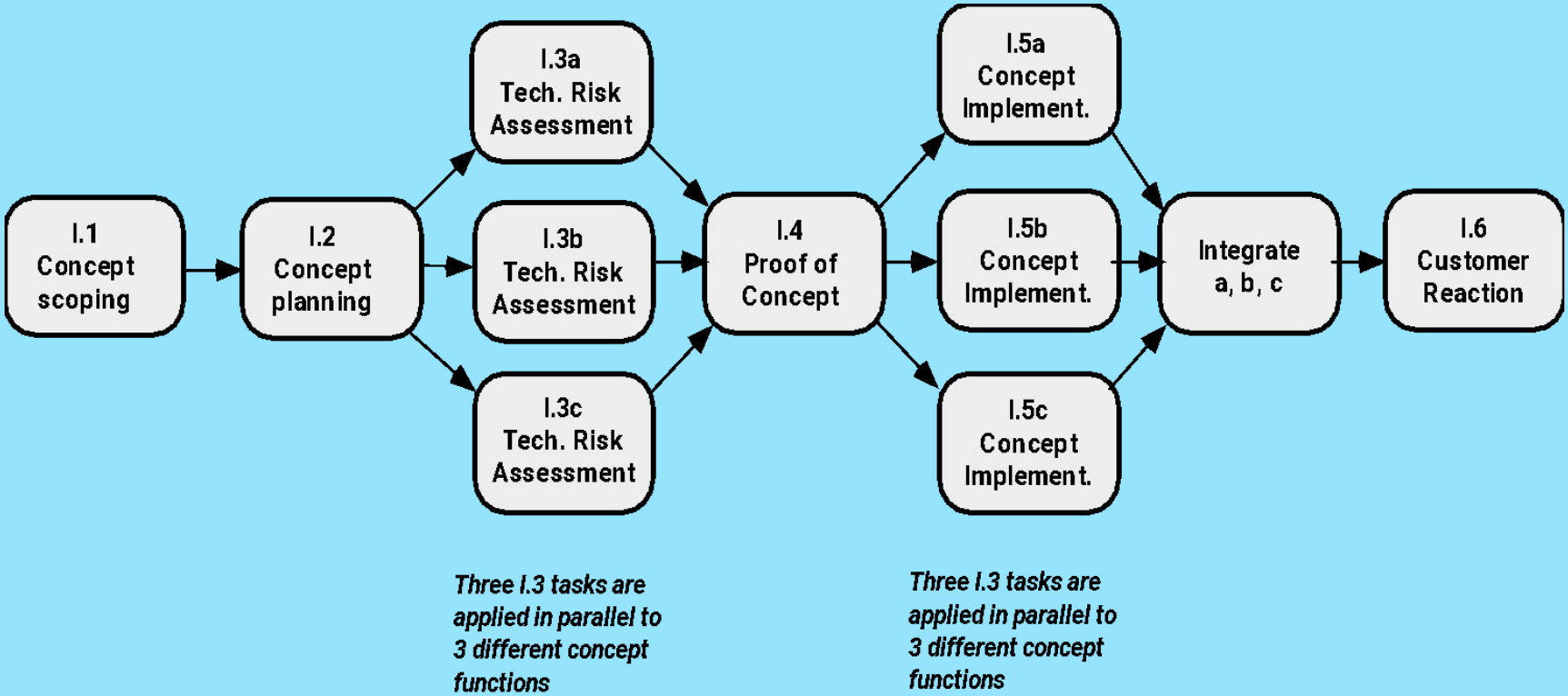
- **Task Network:**

✔ also called an activity network.

✔ Graphic representation of the task flow for the project.

✔ It is also used as a mechanism through which task sequence and project dependencies are input to an automated project scheduling tool

Define a Task Network



Task Set Refinement

1.1 Concept scoping determines the overall scope of the project.

is refined to

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 (Chapter 26) 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

Scheduling

- **Objective of project scheduling** tool is to enable a project manager to define work tasks , establish their dependencies , assign human resources, develop of variety of graphs and charts that aid in tracking and control of the software project.

Scheduling

- Program evaluation and review technique (PERT) and Critical path method (CPM) are two project scheduling methods that can be applied to software development.

- PERT and CPM are driven by information from earlier projects:

✔ Estimates effort

✔ A decomposition of product function

✔ Selection of appropriate process model and task set

✔ Decomposition of tasks

Scheduling

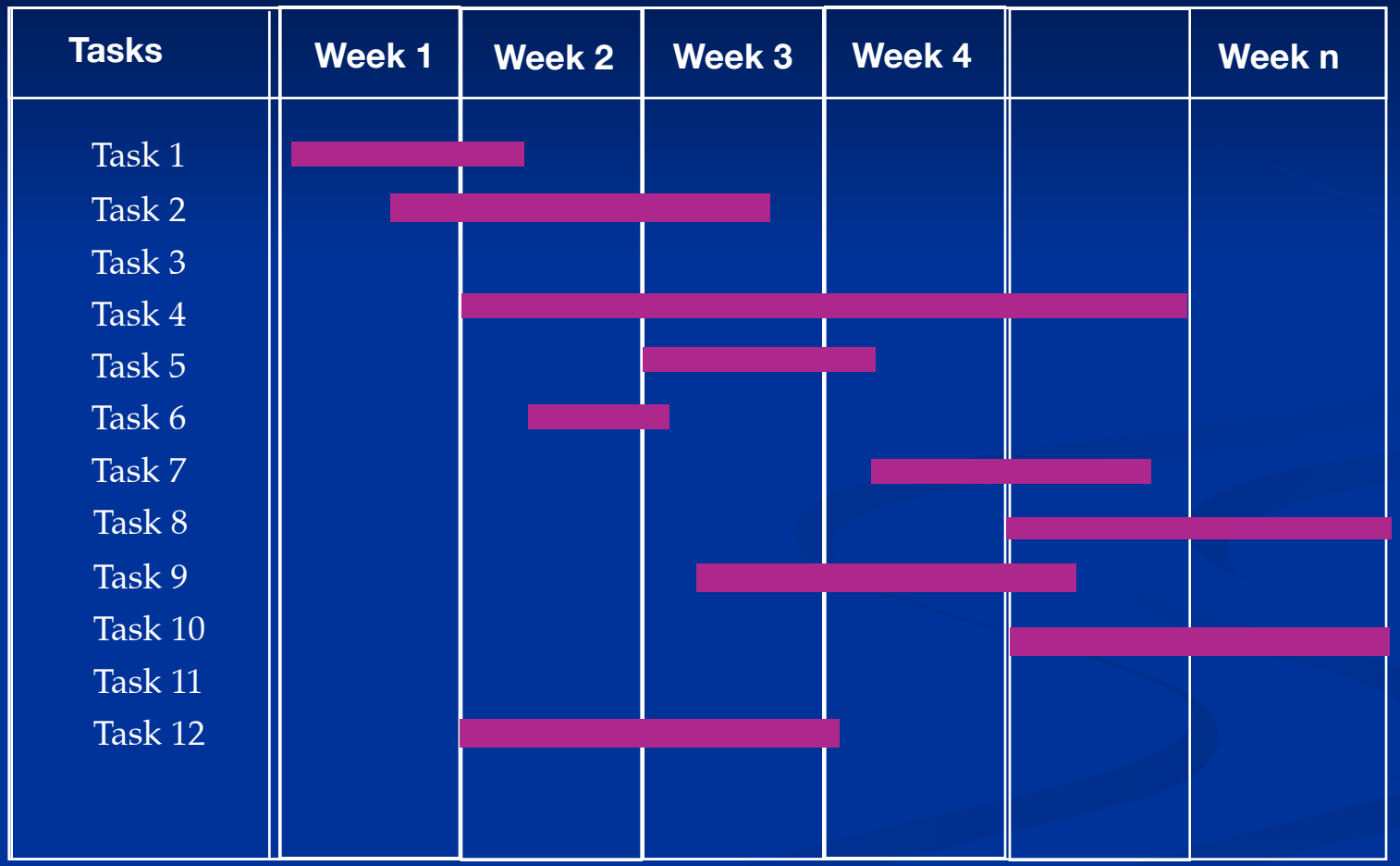
- PERT and CPM are quantitative tools which allow planner to :

✔ determine **critical path-** chain of tasks that determine duration of tasks.

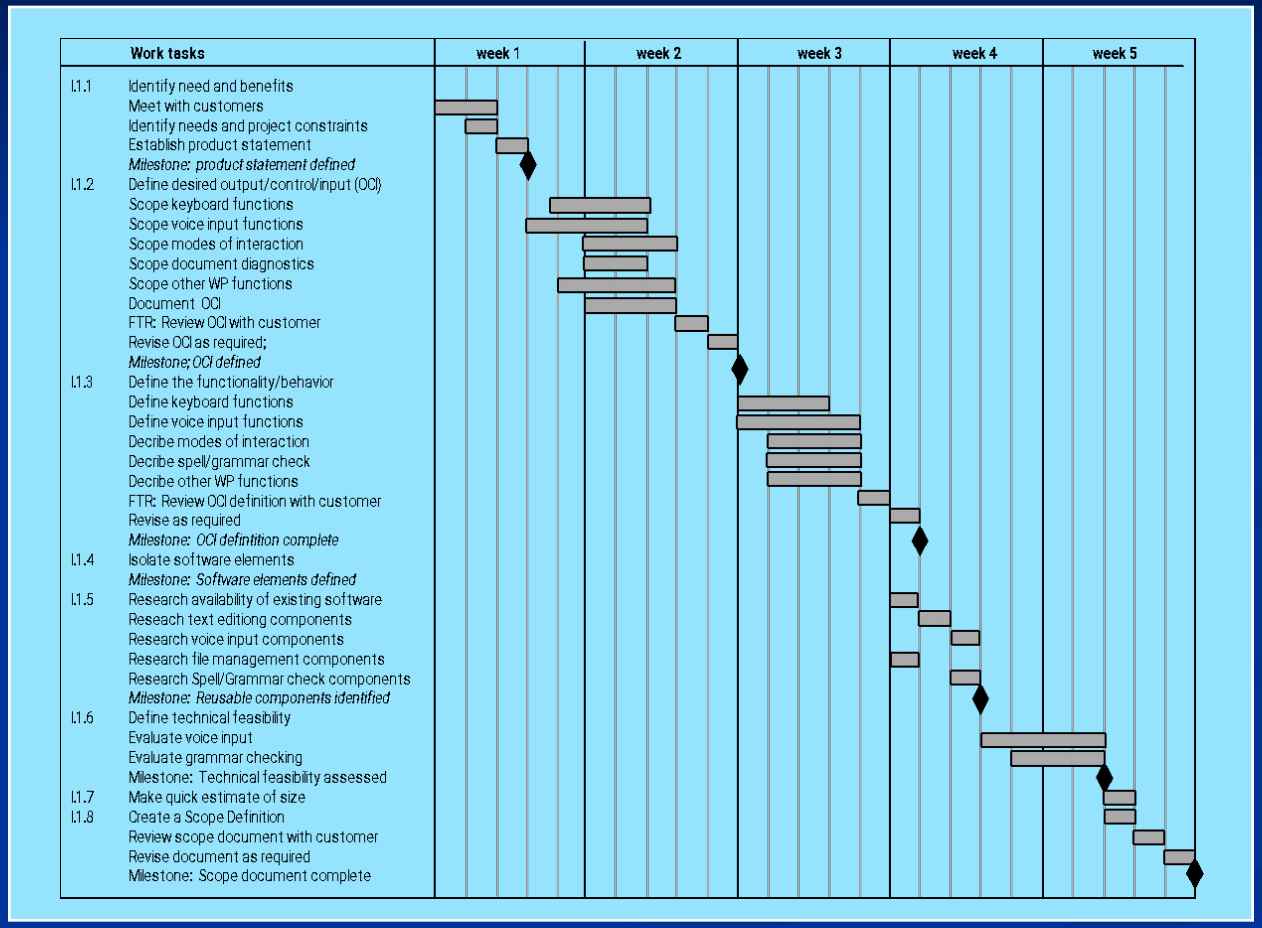
✔ Most likely **time estimates** for individual tasks by applying statistical models.

✔ **Boundary window**-time window for a particular task.

Timeline Charts



Use Automated Tools to Derive a Timeline Chart



Schedule Tracking

- conduct **periodic project status meetings** in which each team member reports progress and problems.
- evaluate the **results of all reviews conducted** throughout the software engineering process.
- determine whether formal **project milestones** have been **accomplished** by the scheduled date.

Schedule Tracking

- **compare actual start-date to planned start-date** for each project task listed in the resource table
- **meet informally** with **practitioners** to obtain their subjective **assessment** of progress to date and problems on the horizon.
- use **earned value analysis** to assess progress quantitatively.

Progress on an OO Project-I

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

Progress on an OO Project-II

- *Technical milestone: OO programming completed*
  - Each new class has been implemented in code from the design model.
  - Extracted classes (from a reuse library) have been implemented.
  - Prototype or increment has been built.
- *Technical milestone: OO testing*
  - The correctness and completeness of OO analysis and design models has been reviewed.
  - A class-responsibility-collaboration network has been developed and reviewed.
  - Test cases are designed and class-level tests have been conducted for each class.
  - Test cases are designed and cluster testing is completed and the classes are integrated.
  - System level tests have been completed.

Earned Value Analysis (EVA)

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