

SOFTWARE ENGINEERING

-SYLLABUS

- Business software
 - Accounting software
- Analytics
- Data mining closely related to database
- Decision support systems
- Airline reservations
- Banking
- Automated teller machines
- Cheque processing
- Credit cards
- Commerce

- Trade
- Auctions (e.g. eBay)
- Reverse auctions (procurement)
- Bar code scanners
- Compilers
- Parsers
- Compiler optimization
- Interpreters
- Linkers
- Loaders
- Communication
- E-mail
- Instant messengers

- System Software
 - Operating systems , Compilers , Editors , Device drivers
- Real-Time Systems and Software
 - Hard Real-time systems(Robot, Railway level crossing)
 - Soft Real-time Systems(Elevator Controller, Cash dispenser
- Business Software
 - Information systems , Databases , Payroll
- Engineering and Scientific Software
 - Astronomy , Earthquake Prediction
- Embedded software
 - Microcontroller , Intel 8051
- Web Based (Client-Server) Software
 - File servers (i.e. mapped directory drives),Print servers,Email,Databases
- Artificial Intelligence (AI) Software
- Safety Critical Systems/Software

The London Ambulance Service

The London Ambulance Service introduced a new computer-aided despatch system in 1992 which was intended to automate the system that despatched ambulances in response to calls from the public and the emergency services. This new system was extremely inefficient and ambulance response times increased markedly. Shortly after its introduction, it failed completely and LAS reverted to the previous manual system. The systems failure was not just due to technical issues but to a failure to consider human and organisational factors in the design of the system.

Project #1: Personal Health Monitoring

- Develops software solutions for continuous monitoring of person's physical activities and vital signs, and motivating people to be more physically active.

Project # 2: Parking Garage Automation

- Develops a computerized system to manage parking usage and online reservations for a parking garage.

Project # 3 : Traffic Monitoring

- Develops a system to inform the user of road and traffic conditions in the area of interest.

Project # 4: Virtual Instructional Laboratories

- Develop a step-by-step visual simulation of engineering topics to supplement actual labs.

Project # 5 : Restaurant Automation

- Develops a computerized system to help restaurant personnel coordinate their activities and improve their services, and for the management to track business growth and create future plans.

MODULE 1

1.1	Software life cycle models: Waterfall, RAD, Spiral, Open-source, Agile process. Chapter 1
1.2	Understanding software process, Process metric , CMM levels, Chapter 25
1.3	Planning & Estimation: Product metrics Estimation- LOC, FP, COCOMO models. Chapter 26
1.4	Project Management activities : Planning, Scheduling and Tracking. Chapter 27

Definition of SOFTWARE ENGINEERING

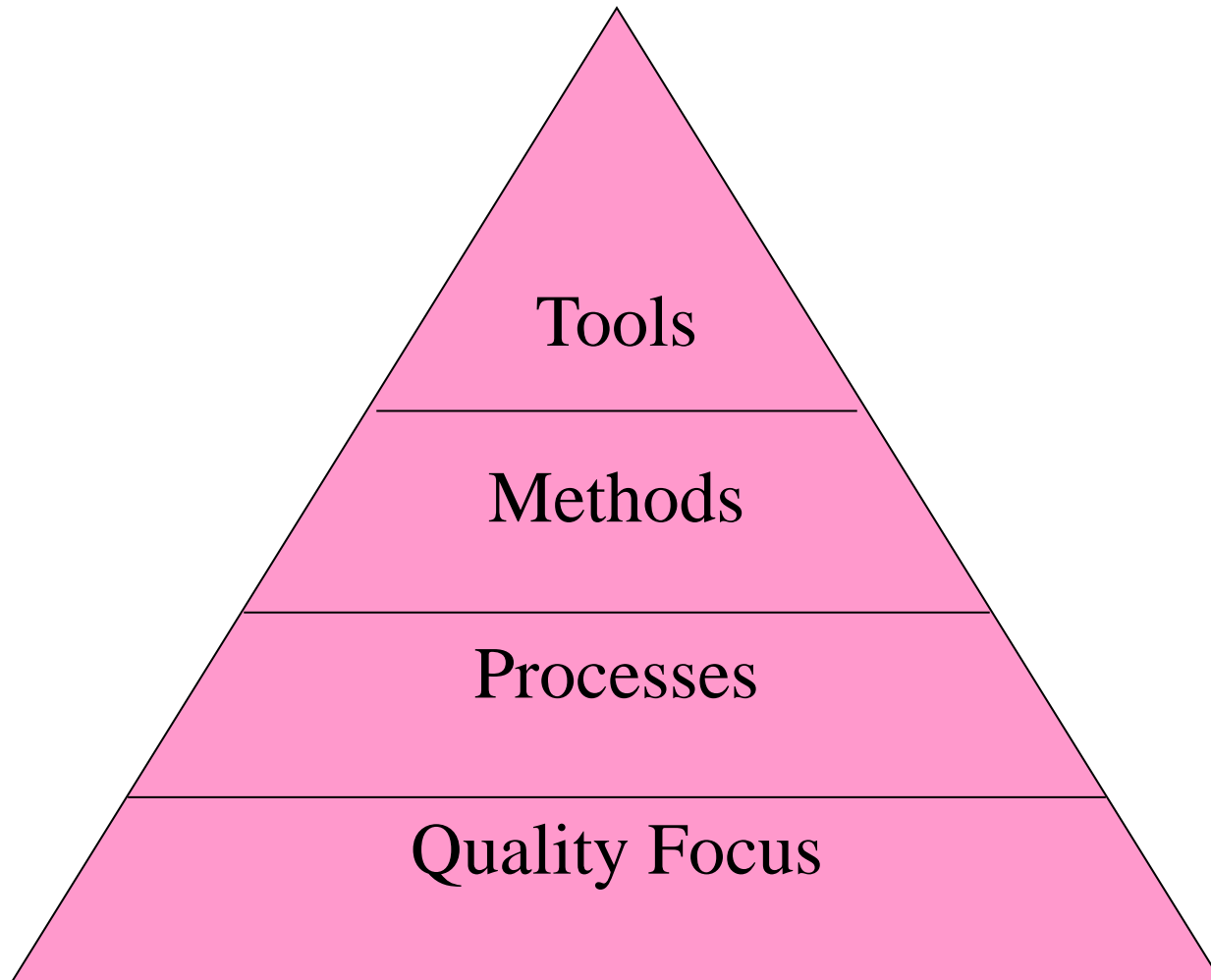
- **Software** is more than just a **program code**.
 - A program is an **executable code**, which serves some computational purpose.
 - Software is considered to be collection of executable programming code, associated libraries and documentations.
 - Software, when made for a specific requirement is called **software product**.
- **Engineering** on the other hand, is all about
 - **developing products,**
 - **using well-defined, scientific principles and methods.**



Software Engineering - Defined

- (1969) Software engineering is the establishment and use of sound engineering principles in order to obtain economical 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

1. Quality Focus

- Software Engineering rests **on solid quality foundation**.
- Quality assurance leads to more effective approach for software development

→ **CORRECTNESS, MAINTAINABILITY, INTEGRITY, USABILITY**

2. 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.

→ **WHAT ACTIVITIES, ACTIONS, TASKS ?**

3. 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

COMMUNICATION, REQUIREMENT ANALYSIS, ANALYSIS & DESIGN

MODELLING, PROGRAM CONSTRUCTION, TESTING & SUPPORT

4. 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.

Eg: RATIONAL ROSE

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

1. Communication
2. Planning
3. Modelling (Analyse ,Design)
4. Construction (Code, Test)
5. Deployment

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.

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

➤ RAD MODEL

➤ SPIRAL MODEL

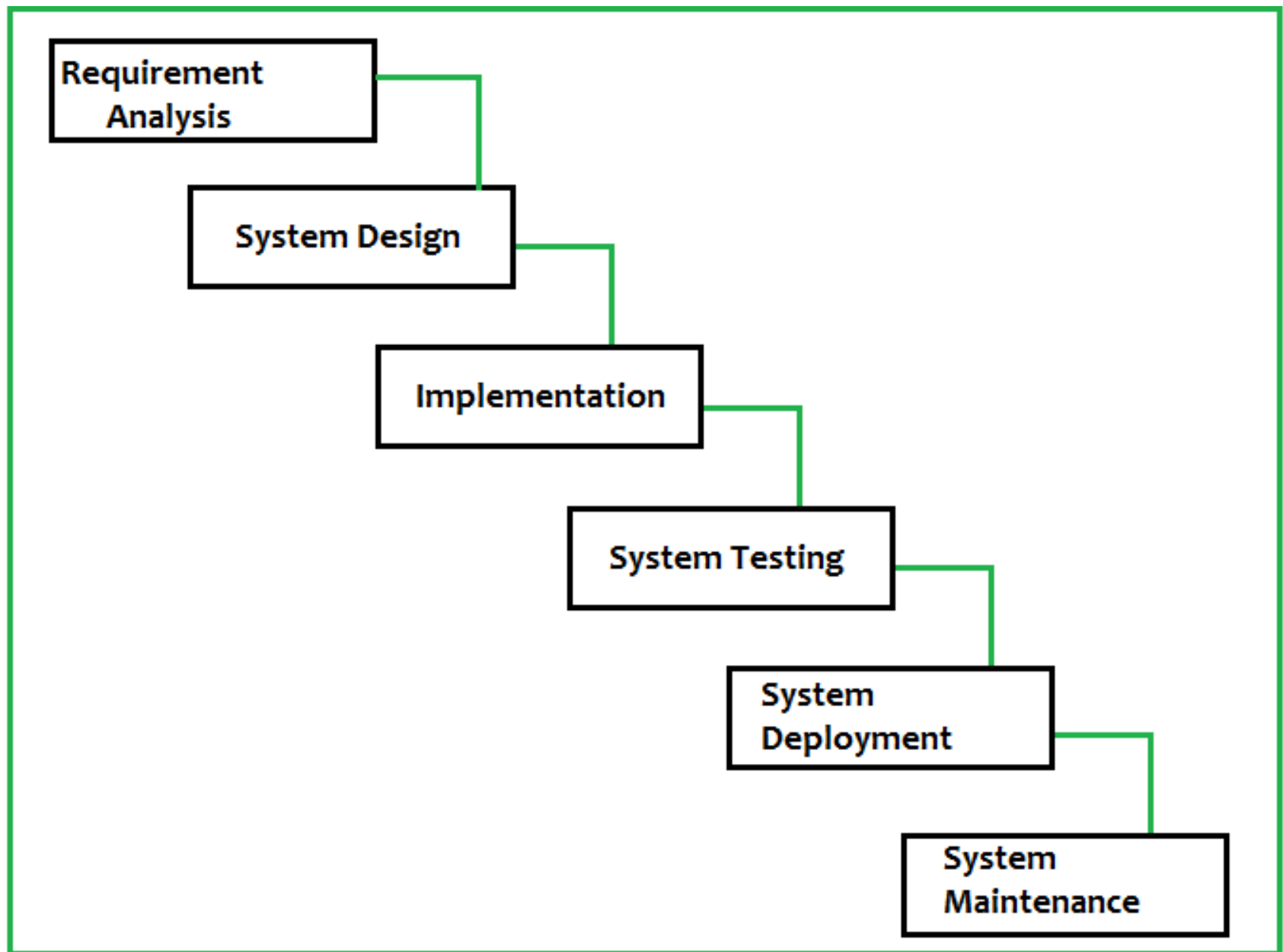
➤ AGILE PROCESS MODEL

➤ OPEN SOURCE MODEL

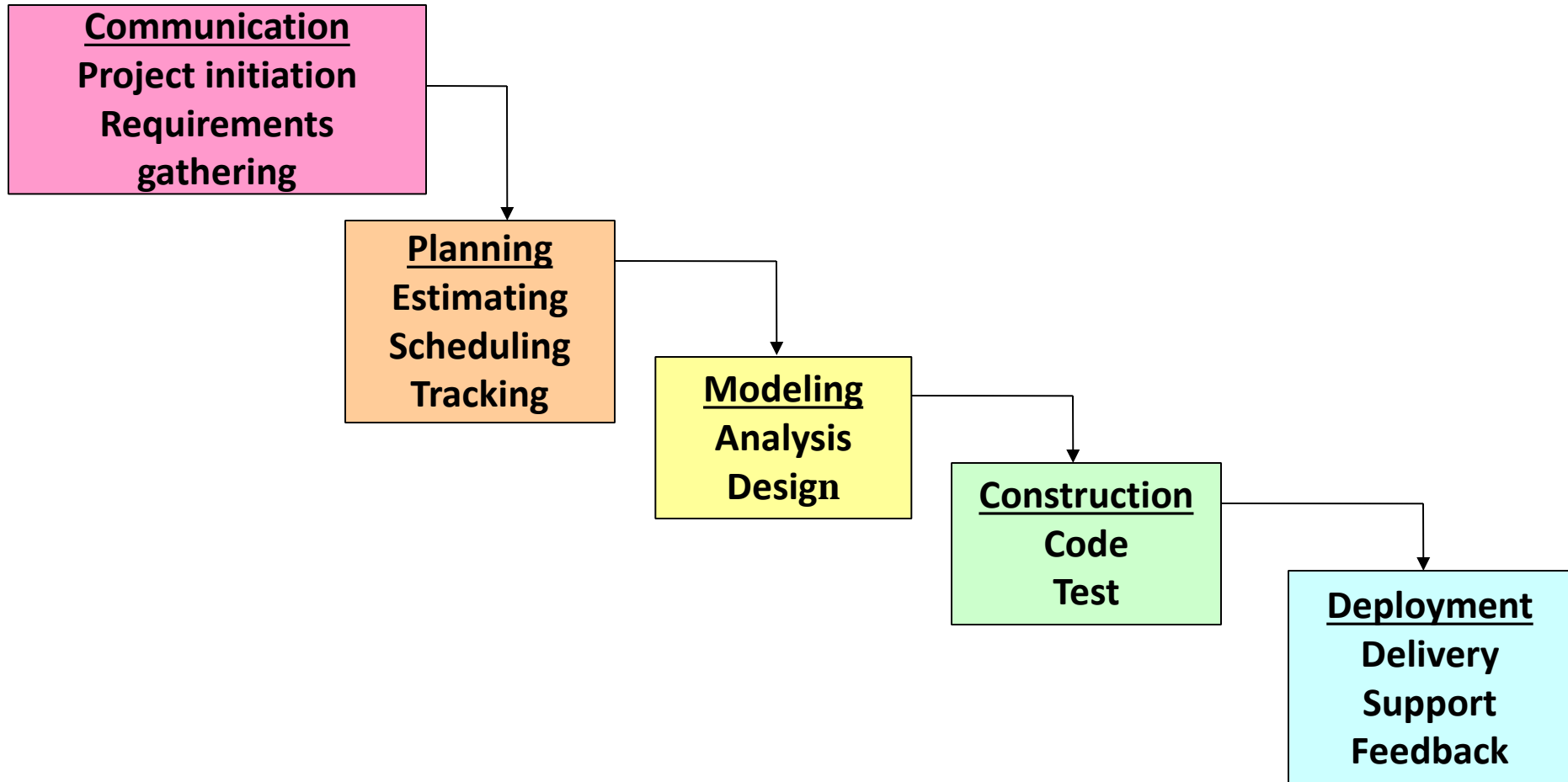


WATERFALL MODEL (Description)

- **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

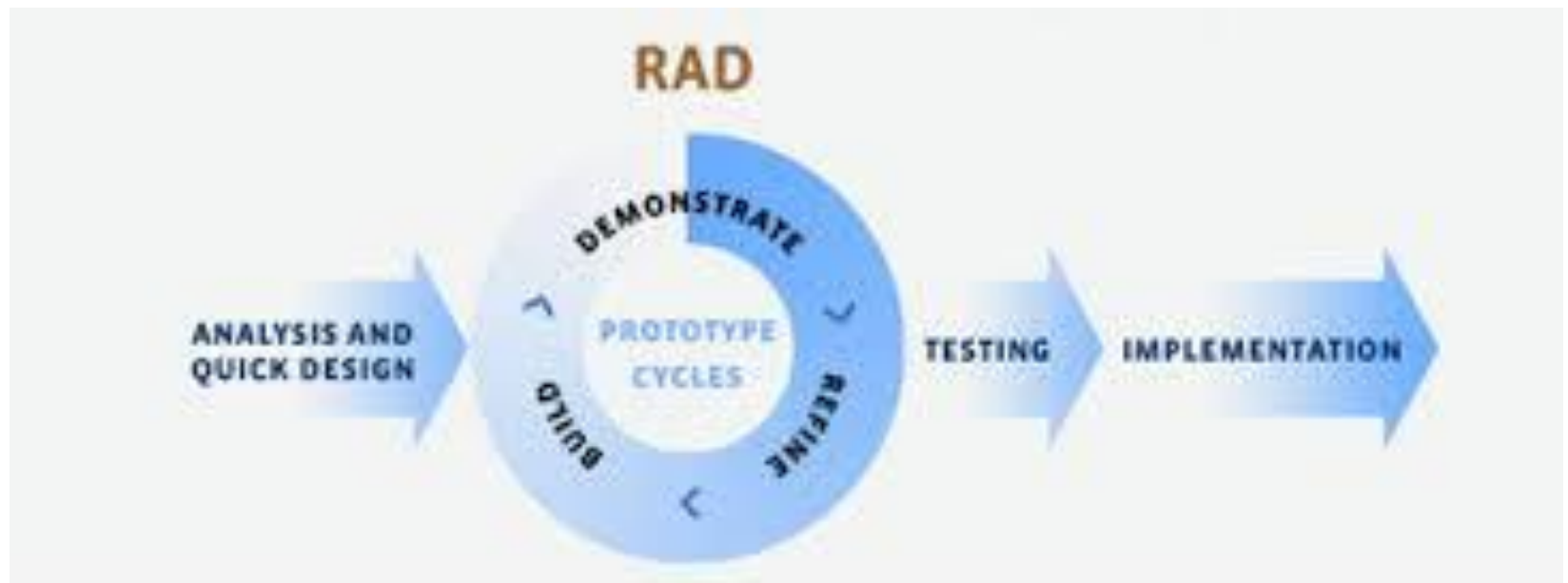
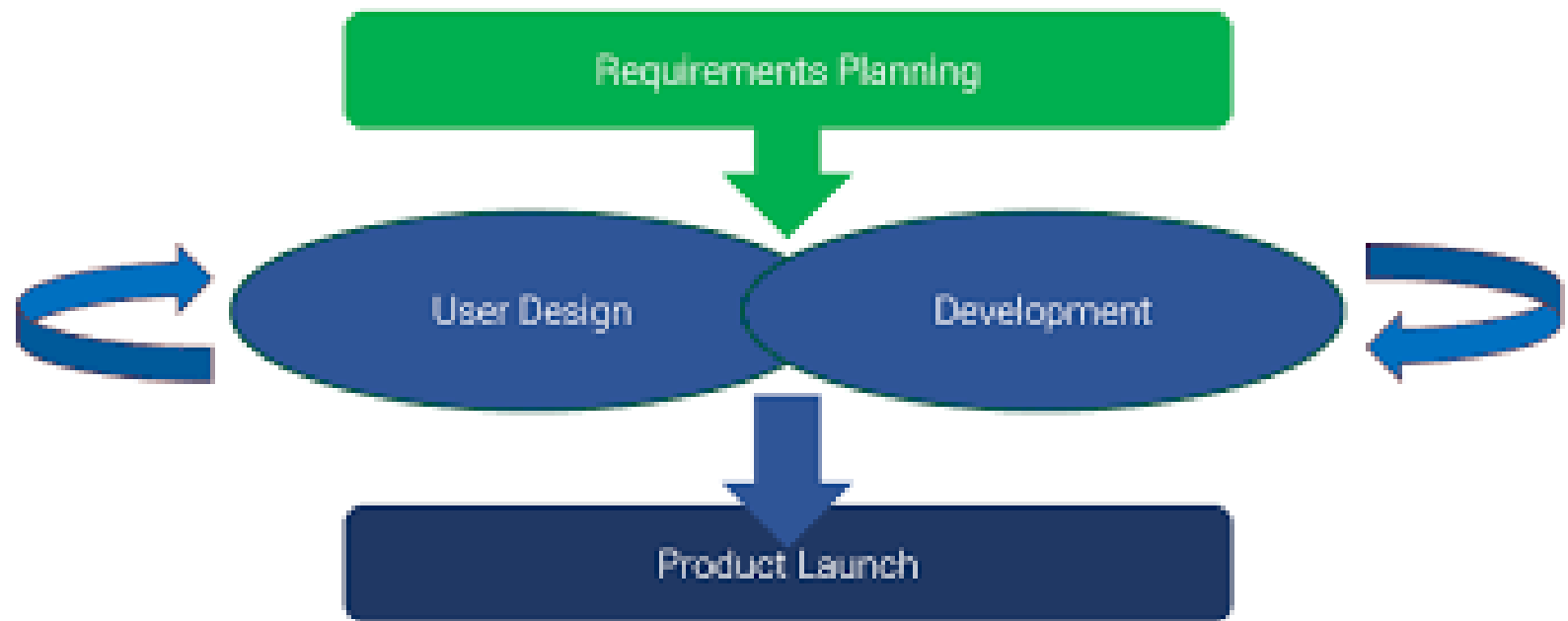
- Simple and easy to understand and use.
- Easy to manage due to the **rigidity of the model** – each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for **smaller projects** where requirements are very well understood.
- **Clearly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

APPLICATIONS

- Banking
- Healthcare
- Control system for nuclear facilities
- Space shuttles
- Aircraft Systems
- Military
- DOD

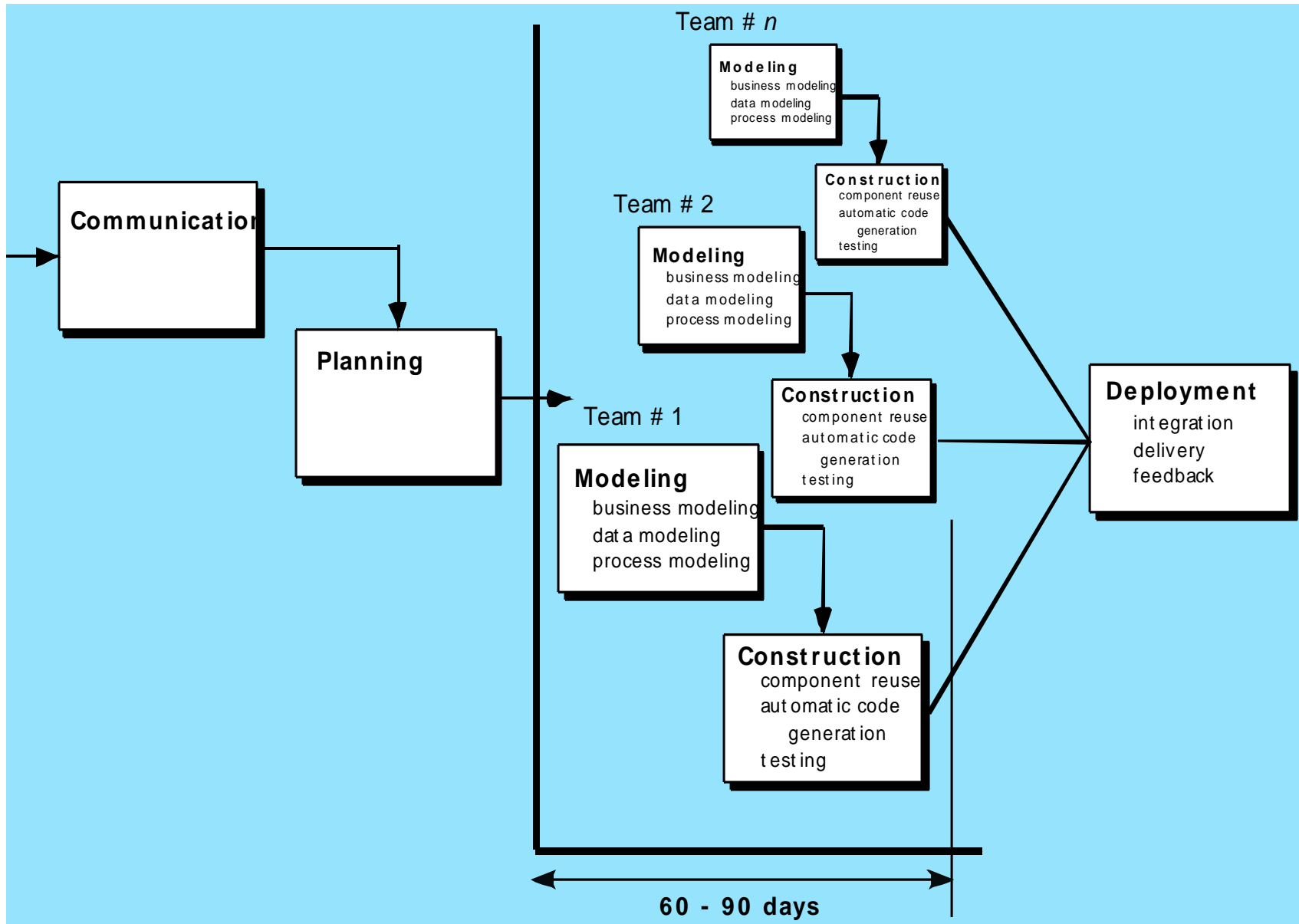


RAD-RAPID APPLICATION DEVELOPMENT

- 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

- Since there is no detailed pre-planning, 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”**.



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

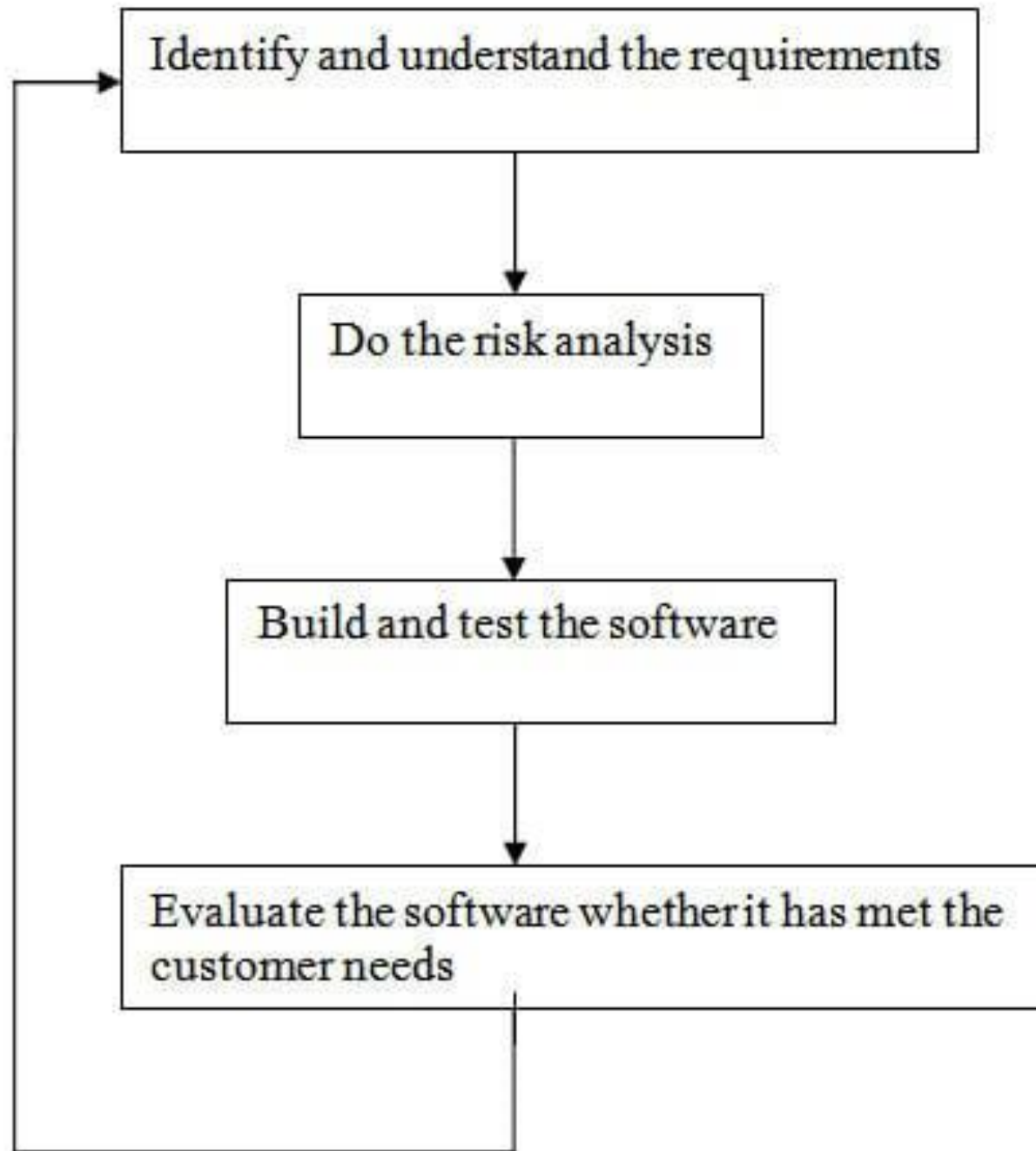
- **“Dependency”** on technically strong team members for identifying business requirements.
- Only system that can be **modularized** can be built using RAD.
- Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.
- Requires **user involvement** throughout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

APPLICATIONS

Sub Modules within modules

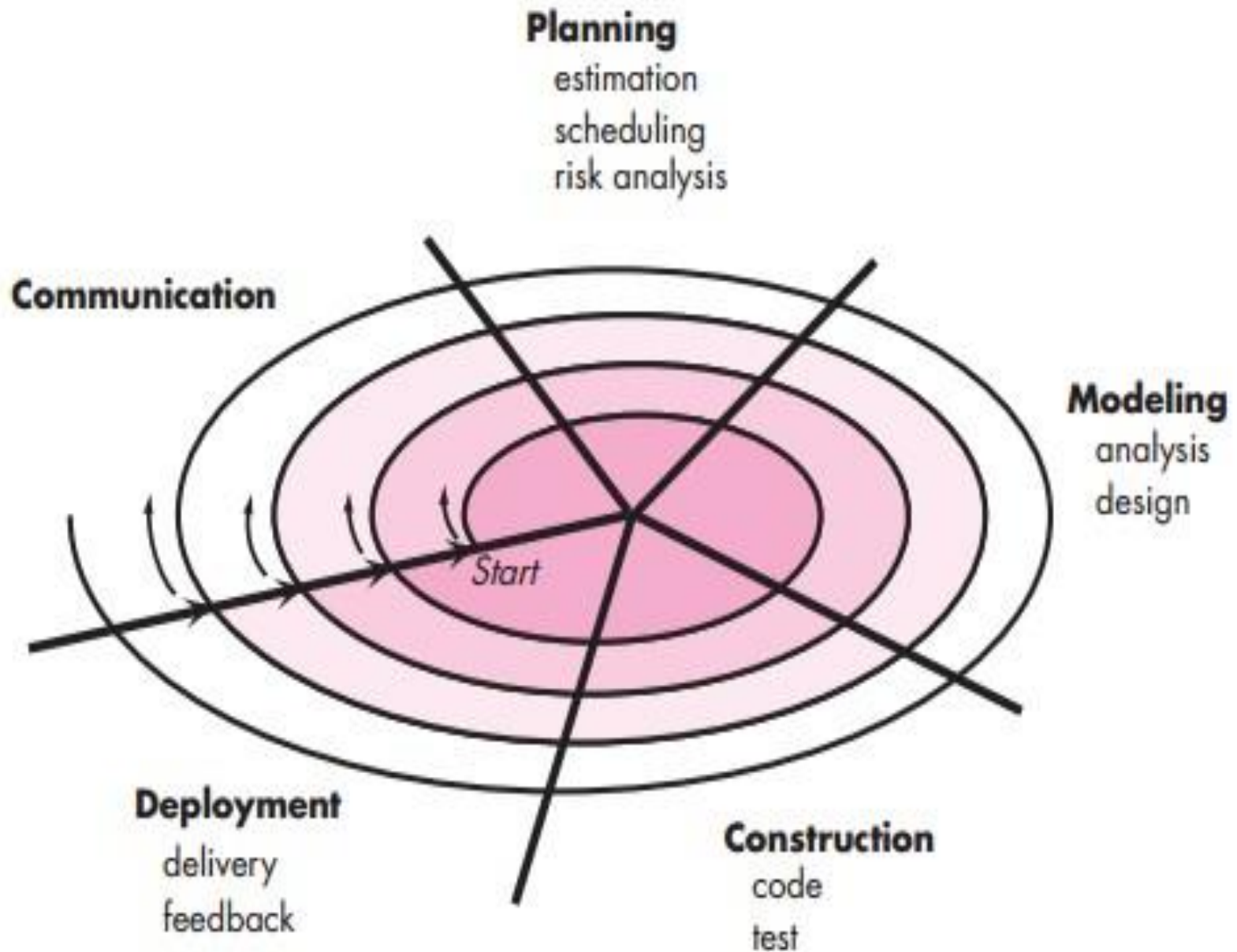
Cloud

- Spiral model is a combination of **sequential and prototype** model.
- This model is best used for **large projects which involves continuous enhancements.**
- There are specific activities which are done in one iteration (spiral) where the output is a small prototype of the large software.
- The same activities are then **repeated** for all the spirals till the entire software is build.



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.

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

Advantages

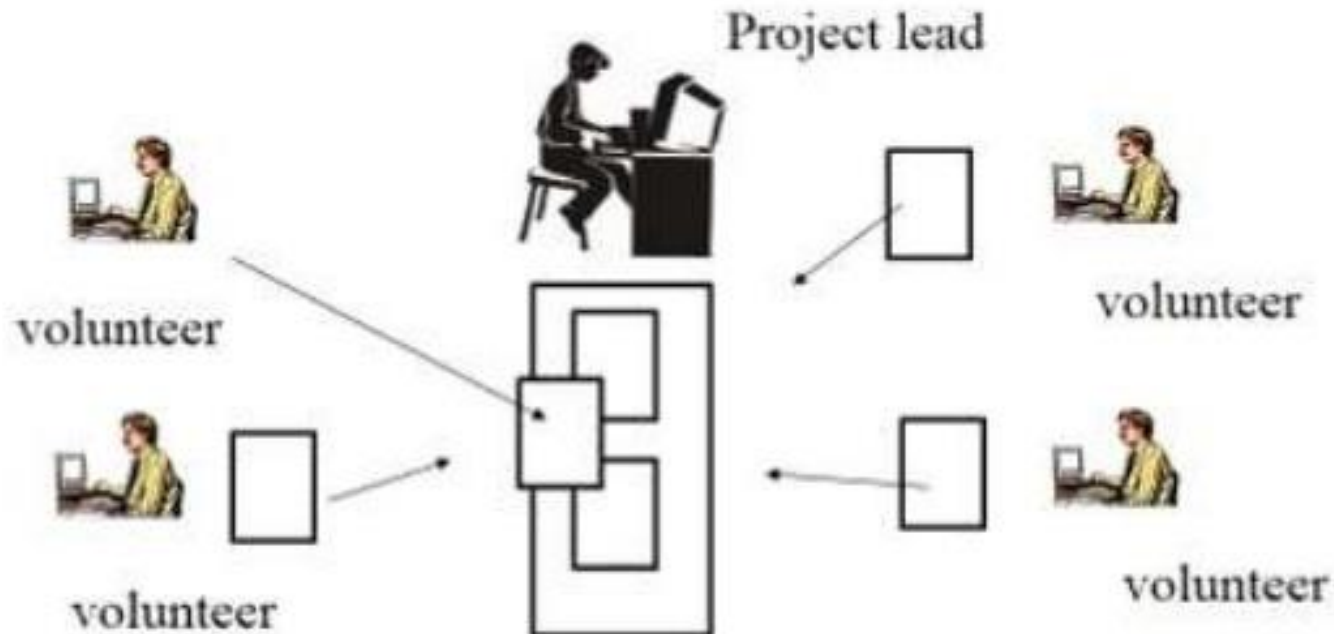
- **Realism:** the model accurately reflects the iterative nature of software development on projects with unclear requirements Allows for extensive use of prototypes.
- **Requirements** can be captured more **accurately**.
- **Users** see the system early.
- Development can be divided into smaller parts 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.**

PROPERTIES OF MODEL	WATER-FALL MODEL	SPIRAL MODEL	RAD MODEL
Planning in early stage	Yes	Yes	No
Returning to an earlier phase	No	Yes	Yes
Handle Large-Project	Not Appropriate	Appropriate	Not Appropriate
Detailed Documentation	Necessary	Yes	Limited
Cost	Low	Expensive	Low
Requirement Specifications	Beginning	Beginning	Time boxed release
Flexibility to change	Difficult	Easy	Easy
User Involvement	Only at beginning	High	Only at the beginning
Maintenance	Least	Typical	Easily Maintained
Duration	Long	Long	Short
Risk Involvement	High	Medium to high risk	Low
Framework Type	Linear	Linear + Iterative	Linear
Testing	After completion of coding phase	At the end of the engineering phase	After completion of coding
Overlapping Phases	No	No	Yes
Maintenance	Least Maintainable	Yes	Easily Maintainable
Re-usability	Least possible	To some extent	Yes
Time-Frame	Very Long	Long	Short
Working software availability	At the end of the life-cycle	At the end of every iteration	At the end of the life cycle
Objective	High Assurance	High Assurance	Rapid development
Team size	Large Team	Large Team	Small Team

OPEN SOURCE DEVELOPMENT MODEL



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

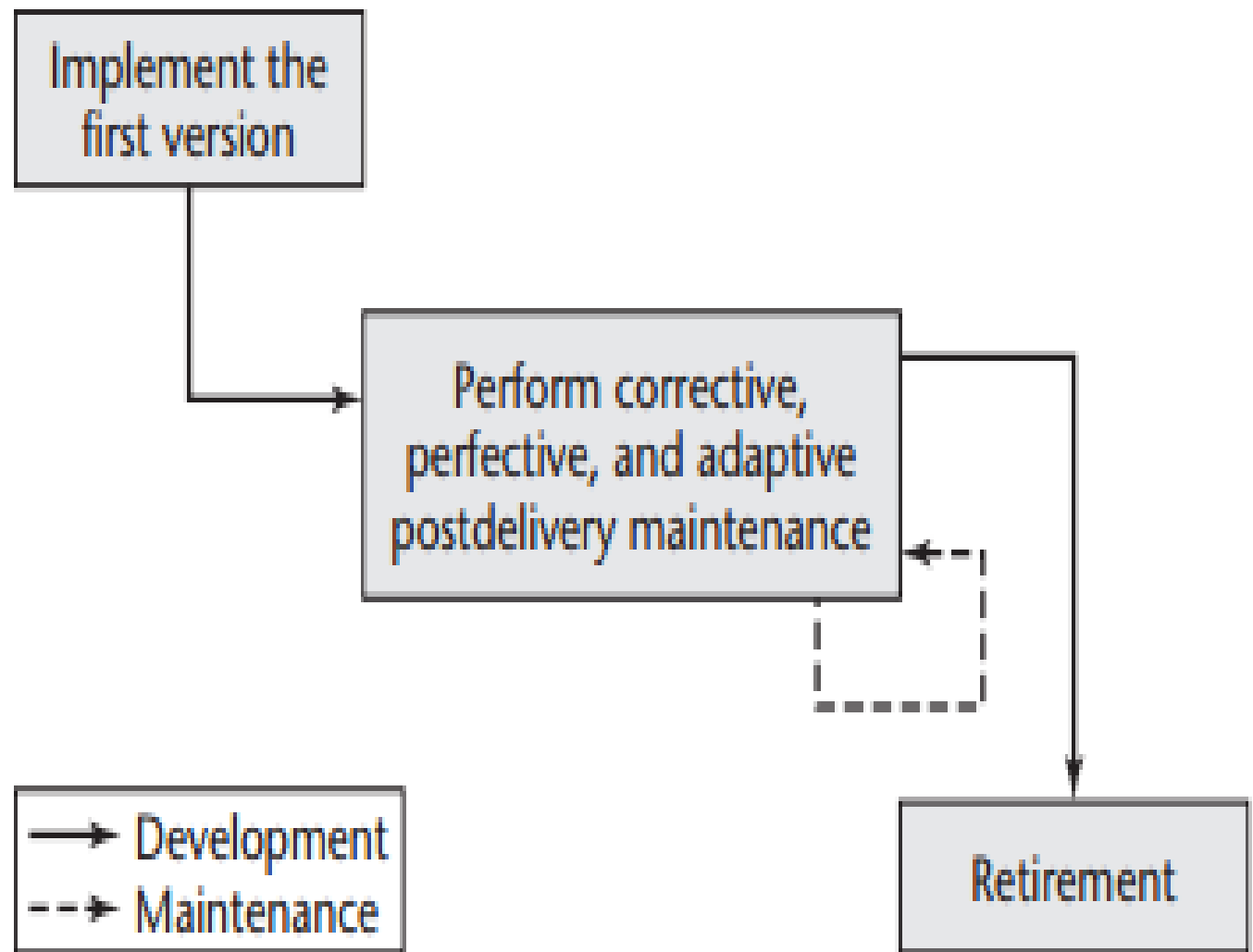
- Once the users start using the program second informal phase of the project begins.
- **Users become co-developers**-some users report defects and others suggest ways of **fixing those defects**.
- Some users **put forward ideas** for extending the program, and others implement those ideas.
- As the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

Open Source Model

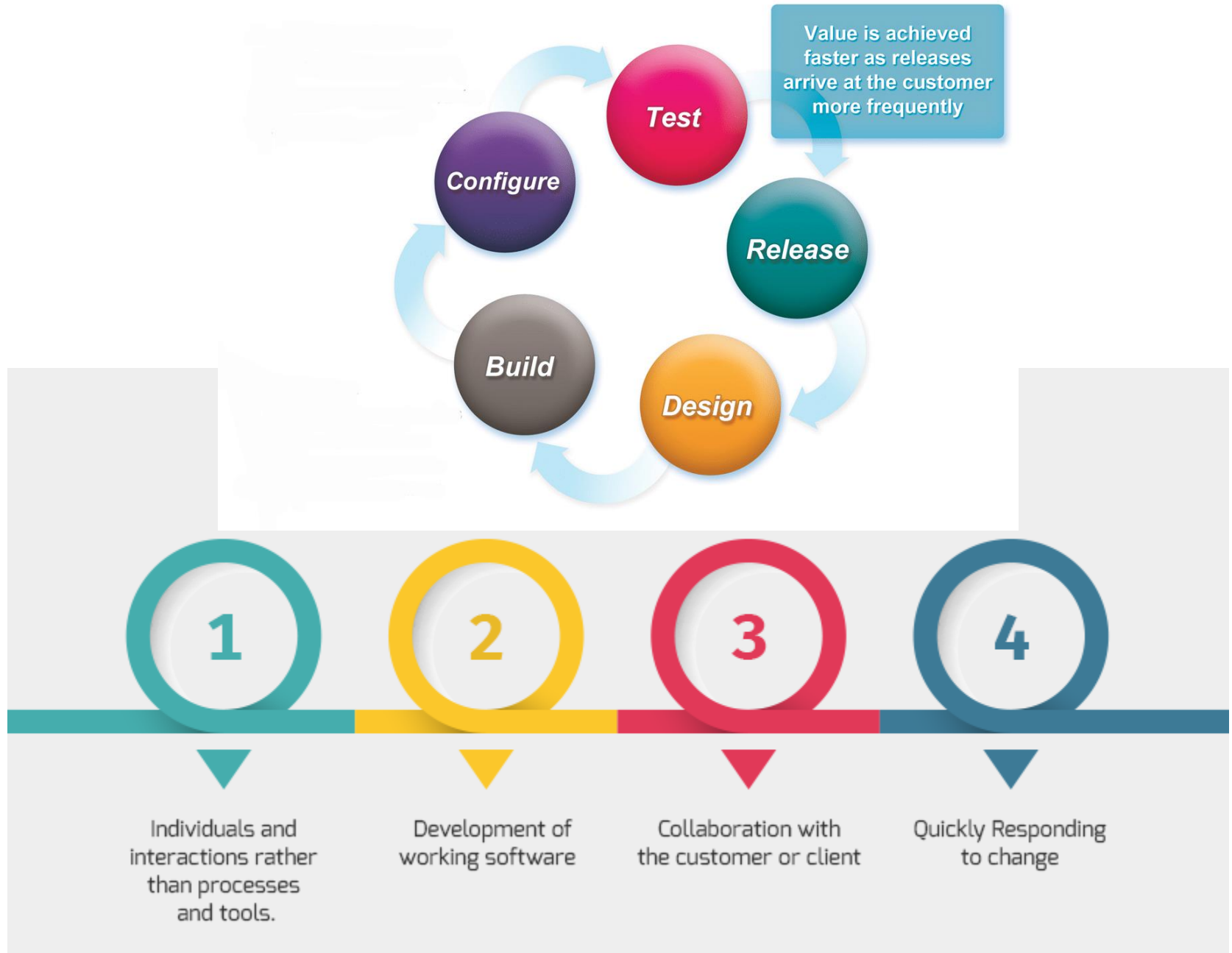
- A key aspect is that individuals work on an open-source project in their spare time on a “**voluntary basis**”; they are not paid to participate.
- The second informal phase of the open-source life-cycle model consists solely of **post delivery maintenance**.

Figure 2.11

The open-source life-cycle model.



Agile Development Process



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.

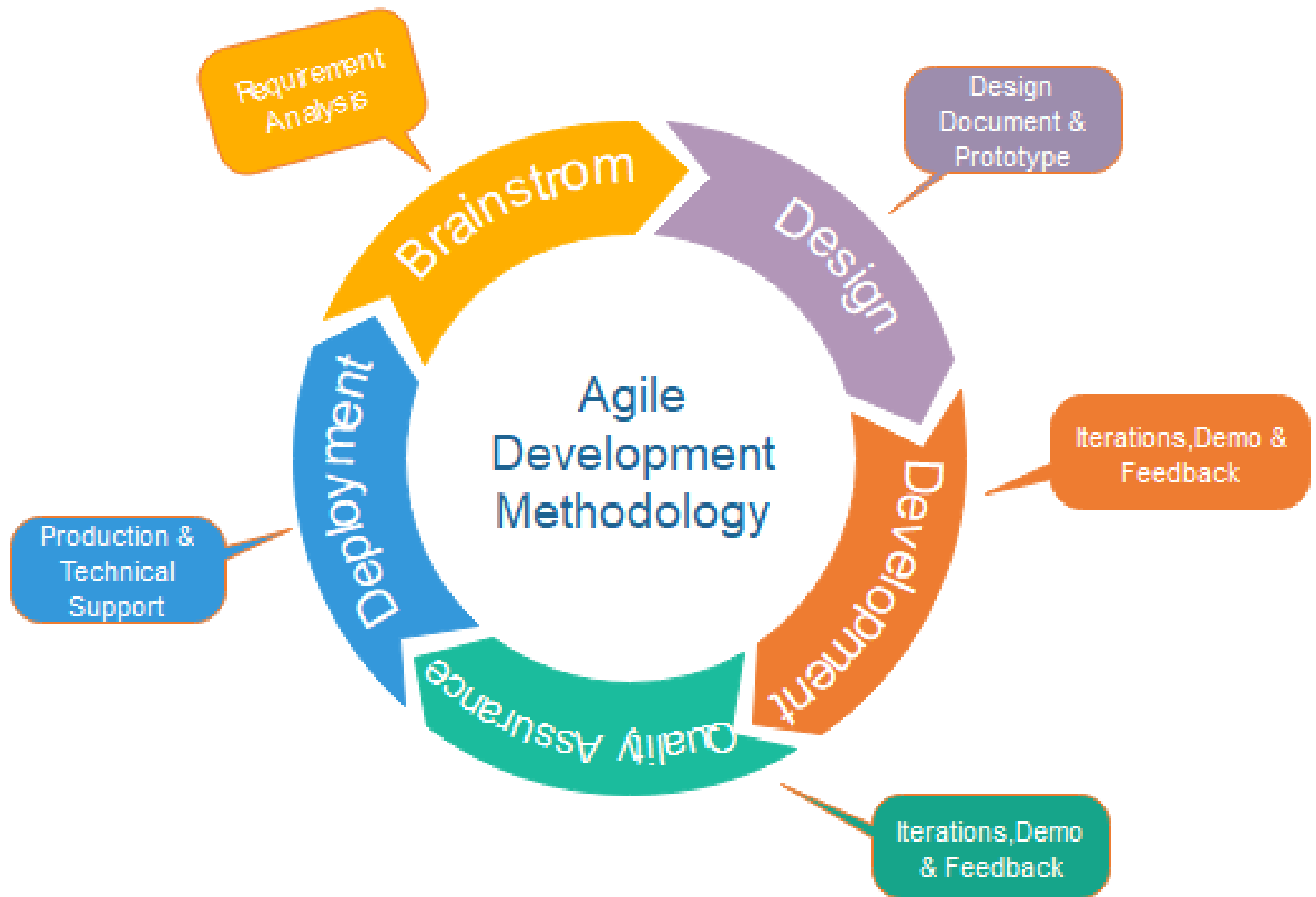


Fig. Agile Model

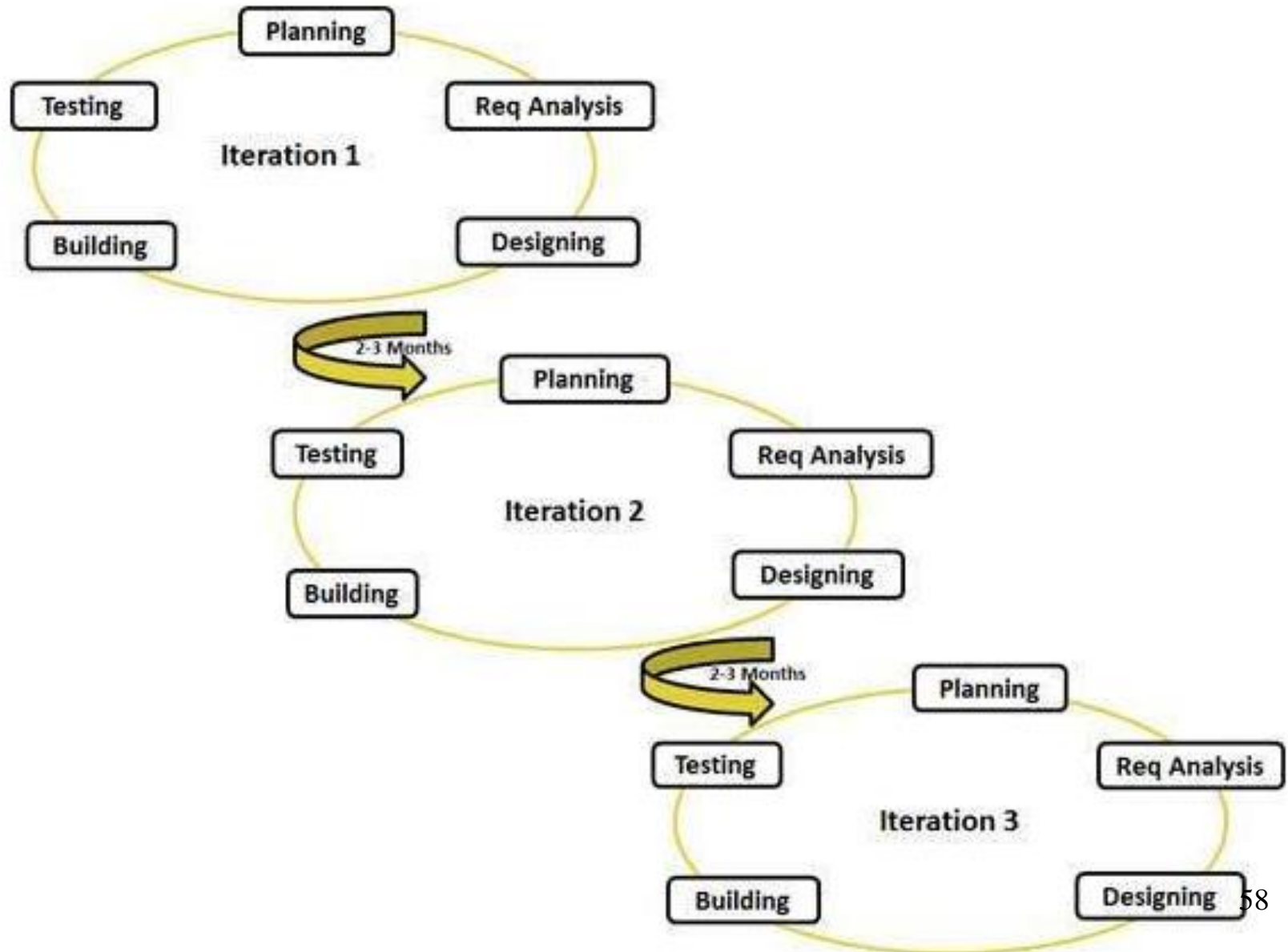
Agile Process Model

- 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

Here is a graphical illustration of the Agile 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.**
- 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.
- Agile methodology **pays less importance to** designing and documentation
- 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

1. Extreme Programming (XP)
2. Adaptive Software Development (ASD)
3. Dynamic Systems Development Method (DSDM)
4. Scrum
5. Crystal
6. Feature Driven Development (FDD)
7. Agile Modeling (AM)

1. Extreme Programming (XP)

- An agile software development framework that aims to produce **higher quality software, and higher quality of life for the development team.**
- XP is the most specific of the agile frameworks regarding **appropriate engineering practices** for software development.

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”**

Features of 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

XP Design

- Follows the KIS (Keep it simple) principle
- Encourage the **use of CRC (class responsibility collaborator cards)**
 - effective mechanism for problem solving as it 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

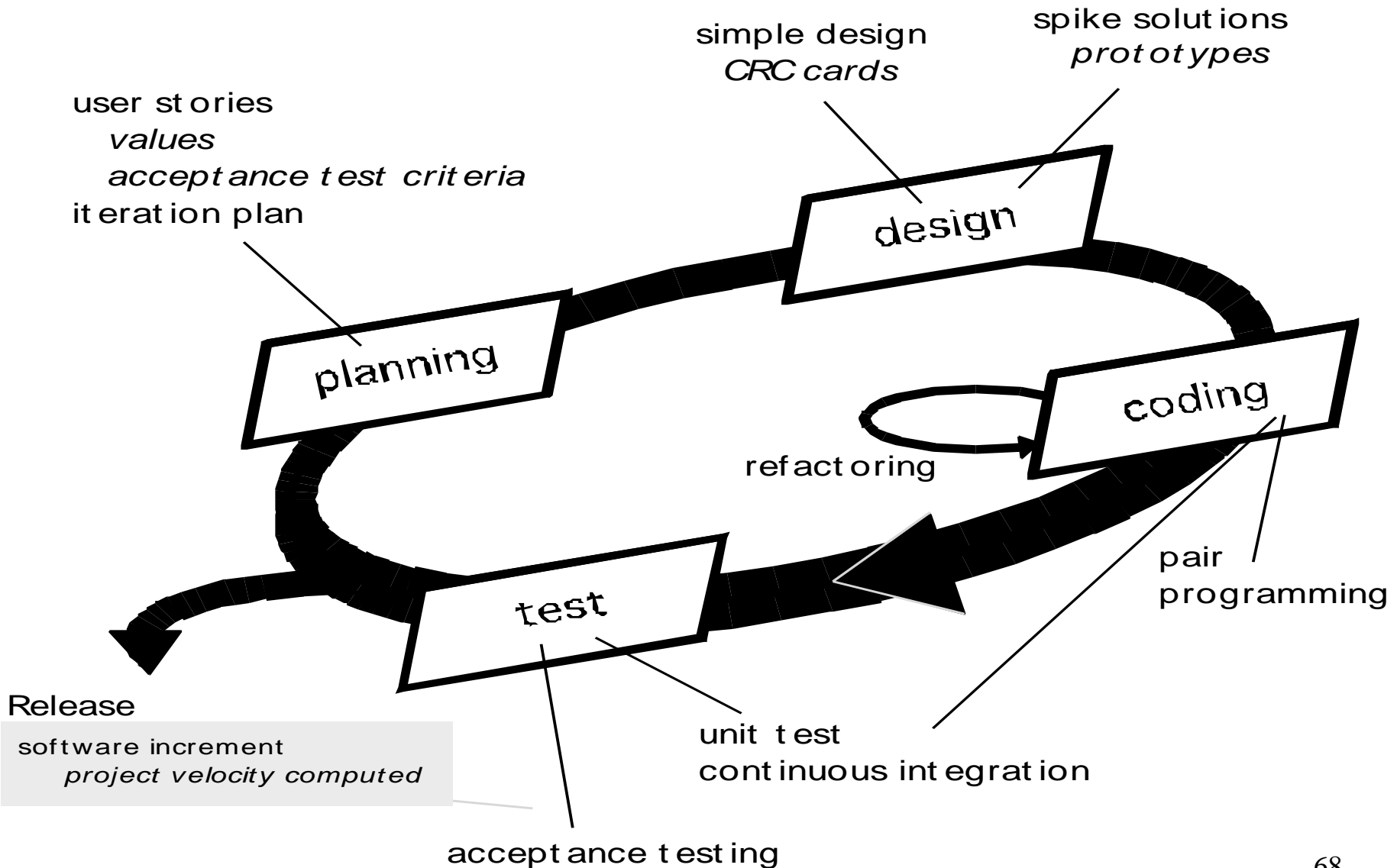
XP Coding

- 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)

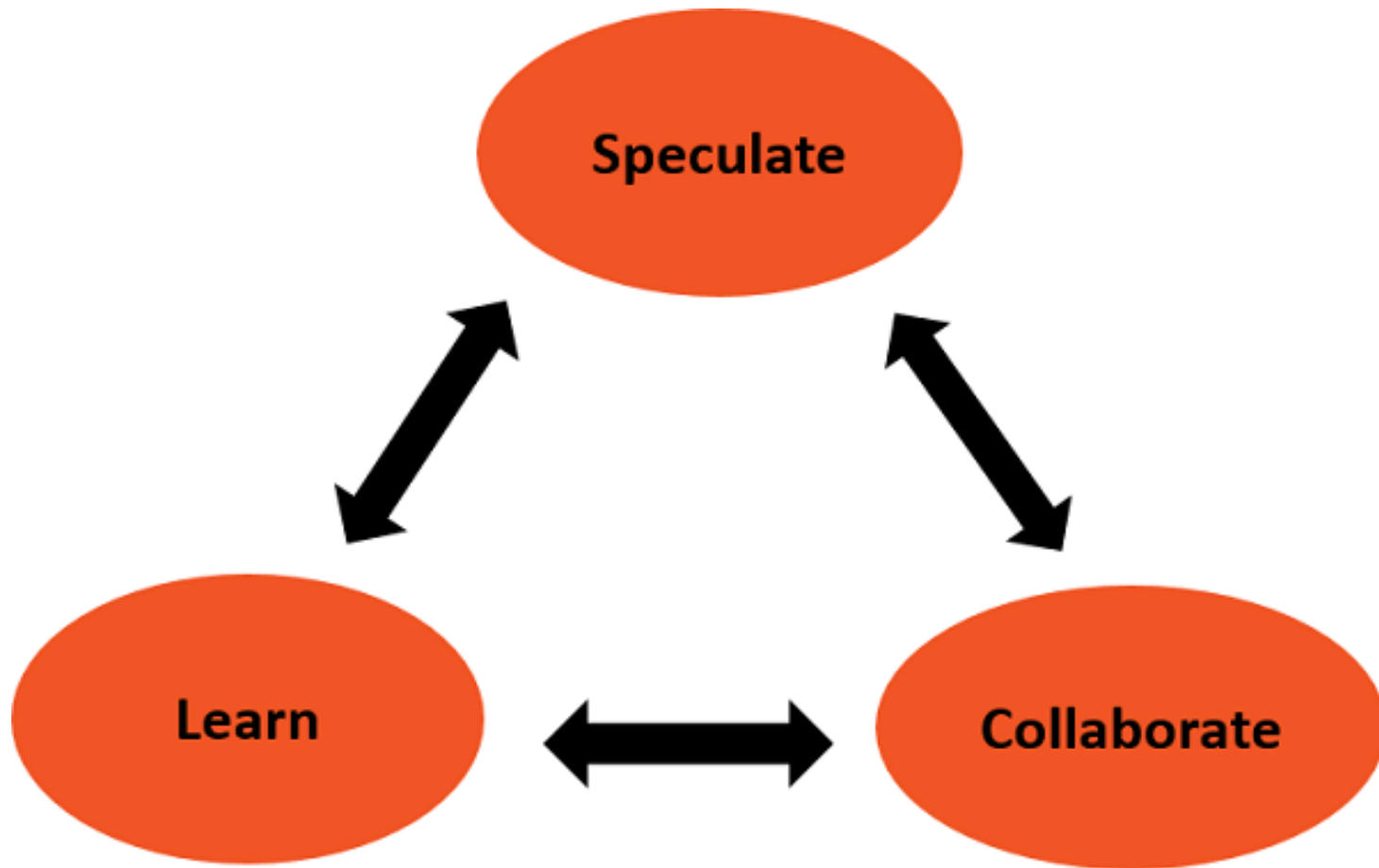


2. Adaptive Software Development

Aims to enable teams to **quickly and effectively adapt to changing requirements** or market needs by **evolving their products with lightweight planning and continuous learning.**

The ASD approach encourages teams to develop according to a three-phase process: **speculate, collaborate, learn.**

- 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



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 tested, “**focus groups**” provide feedback, formal technical reviews, postmortems)

3. Dynamic Systems Development Method(DSDM)

- Agile framework that addresses the “entire project **lifecycle** and its **impact** on the business”
- The framework is built on four principles:
 - FEASIBILITY AND BUSINESS STUDY,
 - FUNCTIONAL MODEL AND PROTOTYPE ITERATION,
 - DESIGN AND BUILD ITERATION, AND
 - IMPLEMENTATION.

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.

- Provides a **framework** for building and maintaining systems which meet **tight time constraints** using incremental prototyping in a controlled environment.
- **Uses Pareto principle –“roughly 80% of output is a direct result of about 20% of the input.”**
 - (80% of project can be delivered in 20% resources 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

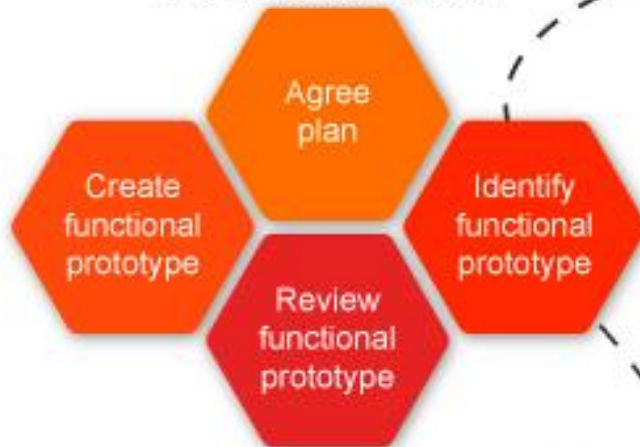
DSDM- distinguishing features

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

Feasibility study & business study



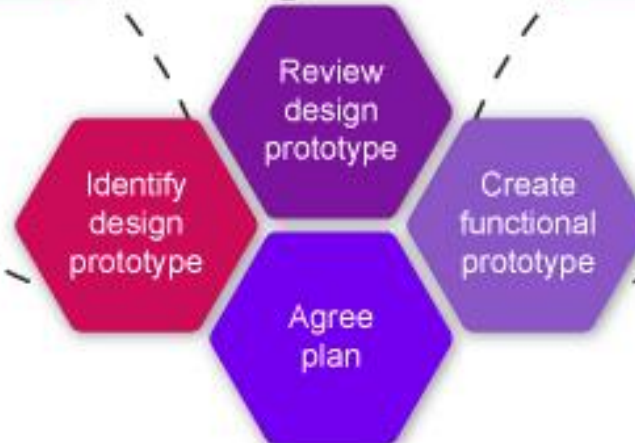
Functional model

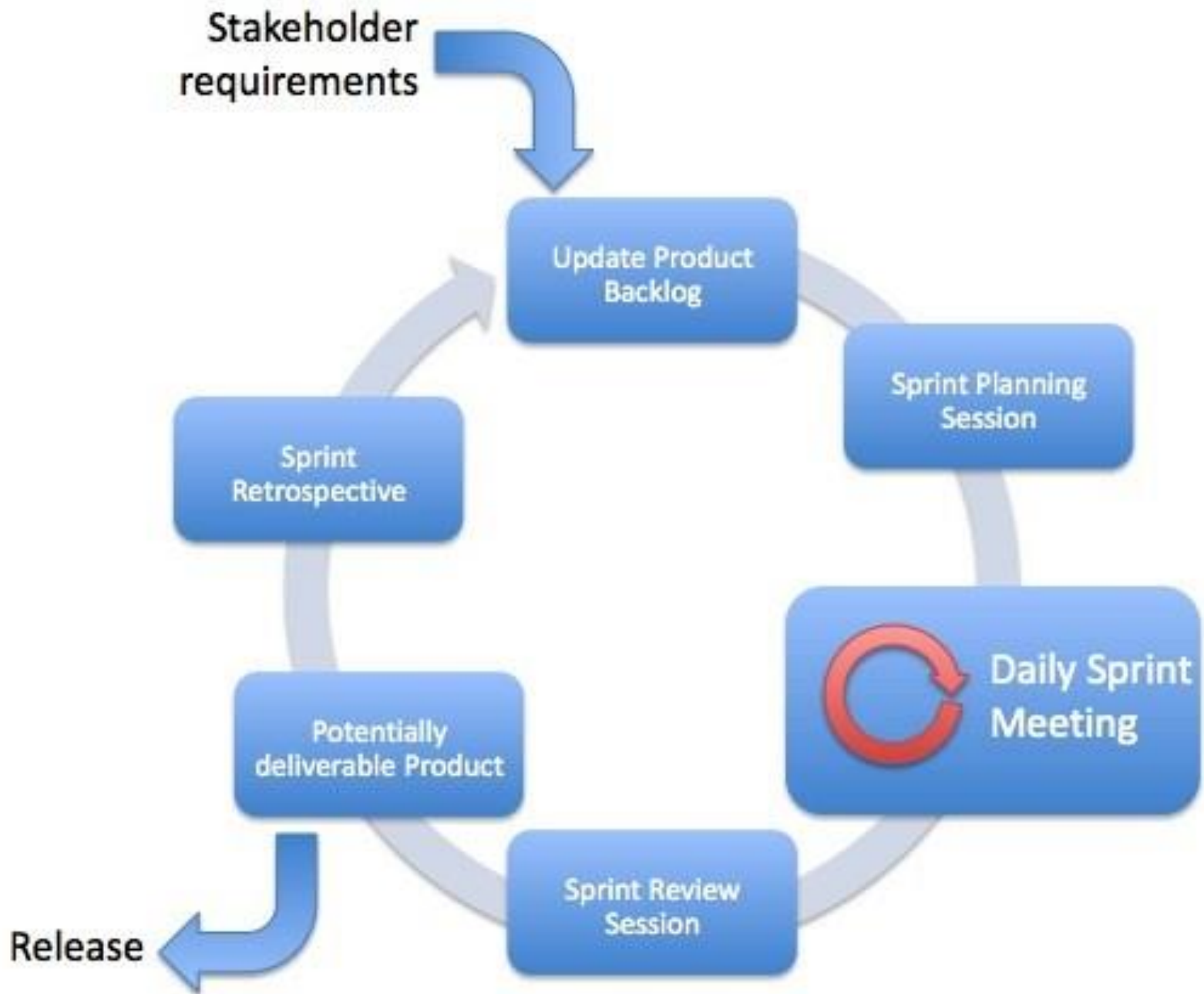


Implementation



Design & Build





4.SCRUM

- SCRUM IS A **FRAMEWORK** WITHIN WHICH PEOPLE CAN ADDRESS **COMPLEX ADAPTIVE PROBLEMS**, WHILE PRODUCTIVELY AND CREATIVELY DELIVERING PRODUCTS OF THE **HIGHEST POSSIBLE VALUE.**
- Scrum itself is a simple framework for “**effective team collaboration on complex product**”

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.

- **Product backlog**: List of priority wise requirement
- **Sprint**: Period in which development occurs
- **Sprint backlog**: List of task to be completed which are prioritized

SPRINT

- ✓ Changes are not allowed during sprint.
- ✓ Allows team members to work in short and stable environment.

SCRUM MEETINGS

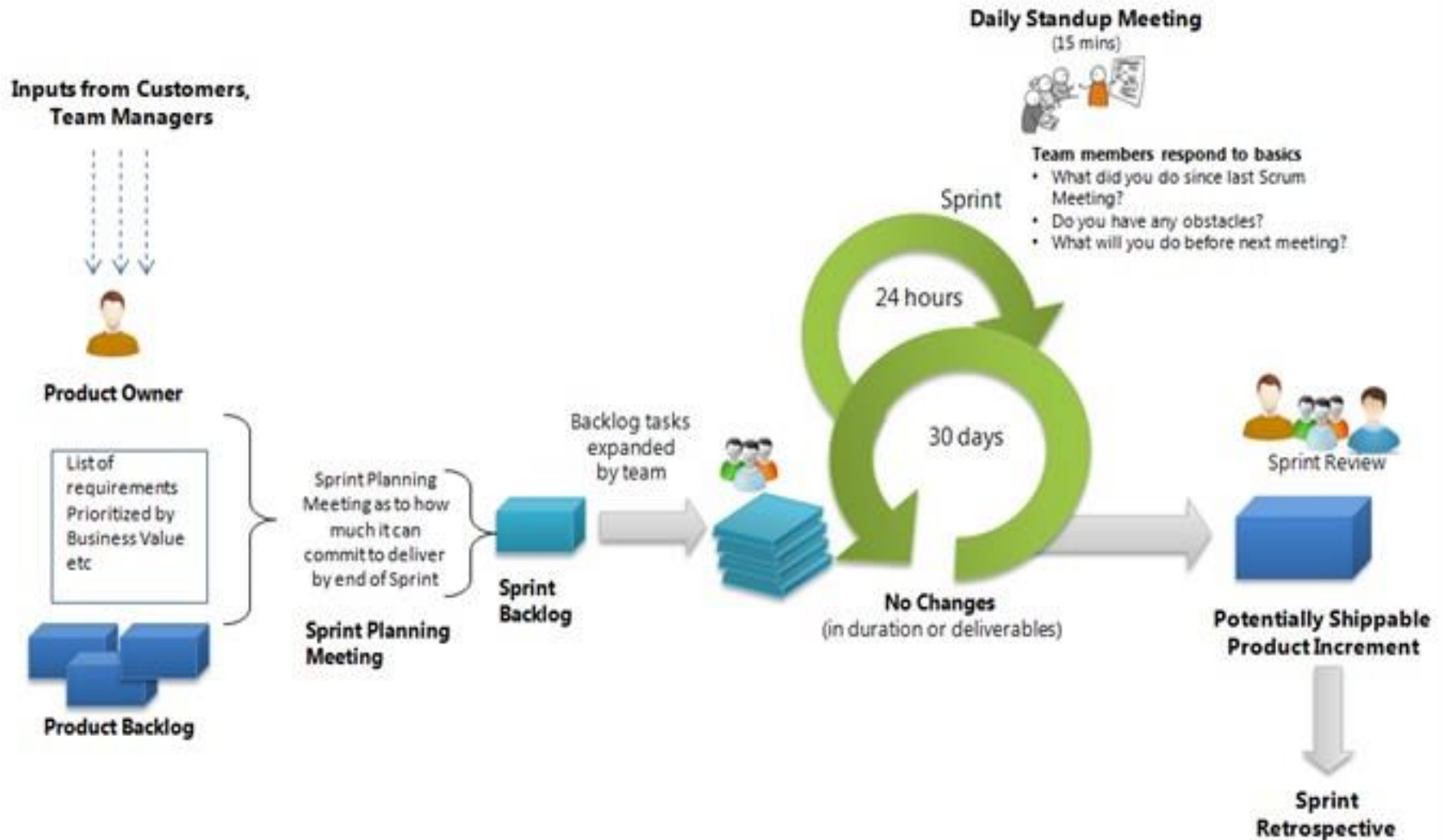
- ✓ are short (typically 15 minutes) meetings held daily by the Scrum team to check the progress of ongoing work.

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



5. 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

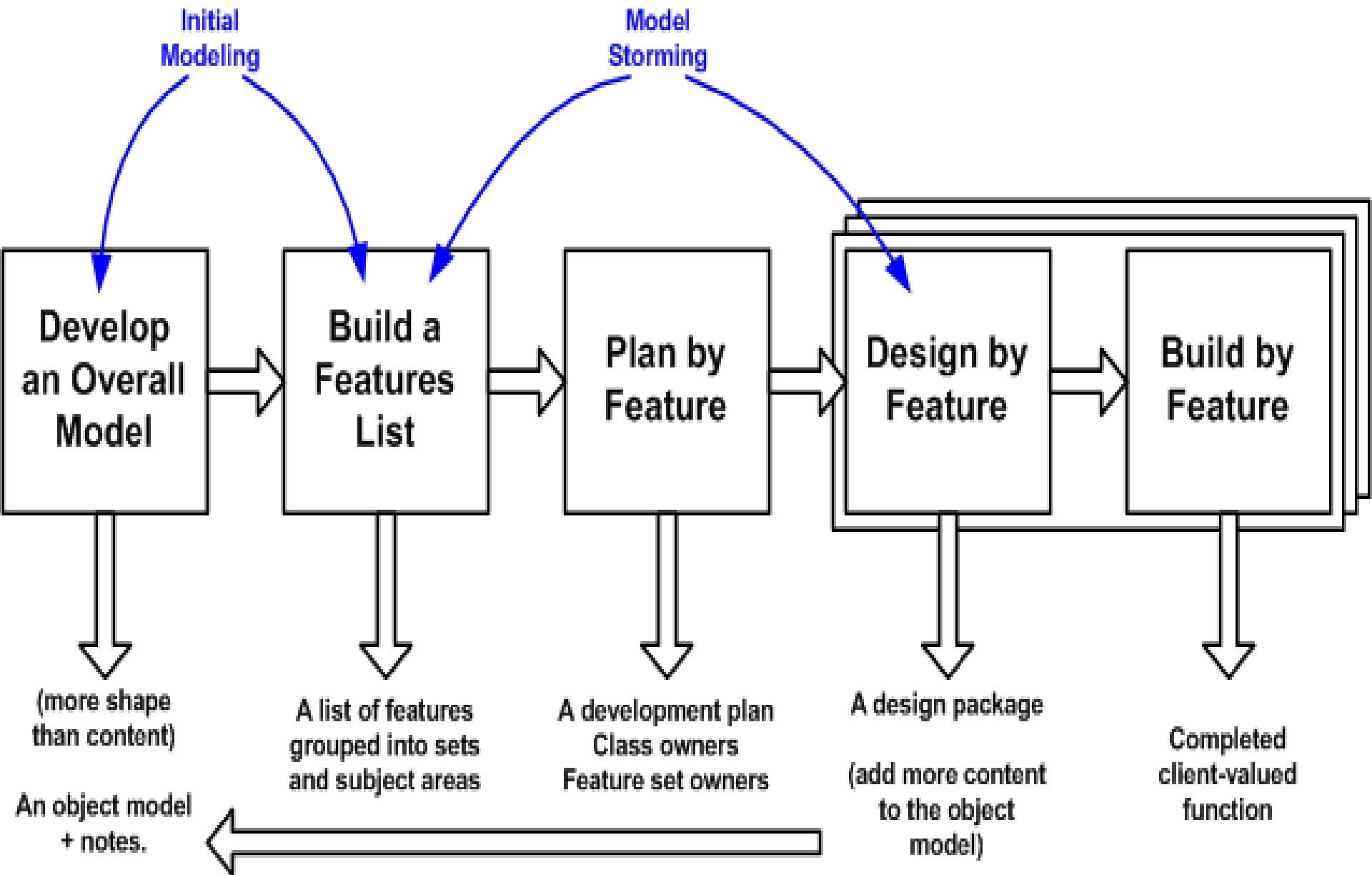
6. FEATURE DRIVEN DEVELOPMENT

- 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

- **Manages “problem and project complexity”** using feature-based decomposition followed by 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)

- **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)

- **Plan by feature** (features assessed based on priority, effort, technical issues, schedule dependencies)
- **Design by feature** (classes relevant to feature are chosen, 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)



XP	Main limitations of XP are poor documentation. It is also inappropriate for distributed teams, reuse and subcontracting and development of average and complex software. The success of XP mainly depends on the support of its stakeholders.
SCRUM	Scrum is mainly a management framework than a methodology. It doesn't support for the development of large software and sizeable teams. Scrum also does not assist that how to complete an iteration in one month.
DSDM	DSDM does not handle the engineering of average and complex projects. There is no support for the management of sizeable teams. There is a provision to compose DSDM with XP methodology but at the expense to manage limitations of XP that are infused into DSDM.
CRYSTAL	Geographically distributed teams are not supported by Crystal family of models. More work is needed to measure usefulness of Crystal family of methodologies for the engineering of all types of projects
FDD	FDD is the most recently proposed methodology as compared to other agile methodologies. More validation is required to be widely practiced in software industry.

Table 2-The Data of Two Controlled Case Studies

Items	XP	Scrum
Type of Information System	Library	Payroll
Team size	6	6
Calendar Time (weeks)	5	5
Releases	4	4
Total Tasks defined	96	82
Total work effort (h)	1260	1000
Team Productivity	41	46
Post release defects	20	16
Customer Satisfaction	82%	85%

AGILE MODELING

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

- ✓ **Agile Modeling** is a collection of **VALUES, PRINCIPLES, AND PRACTICES** for **modeling** software that can be applied on a software development project in an effective and light-weight manner.

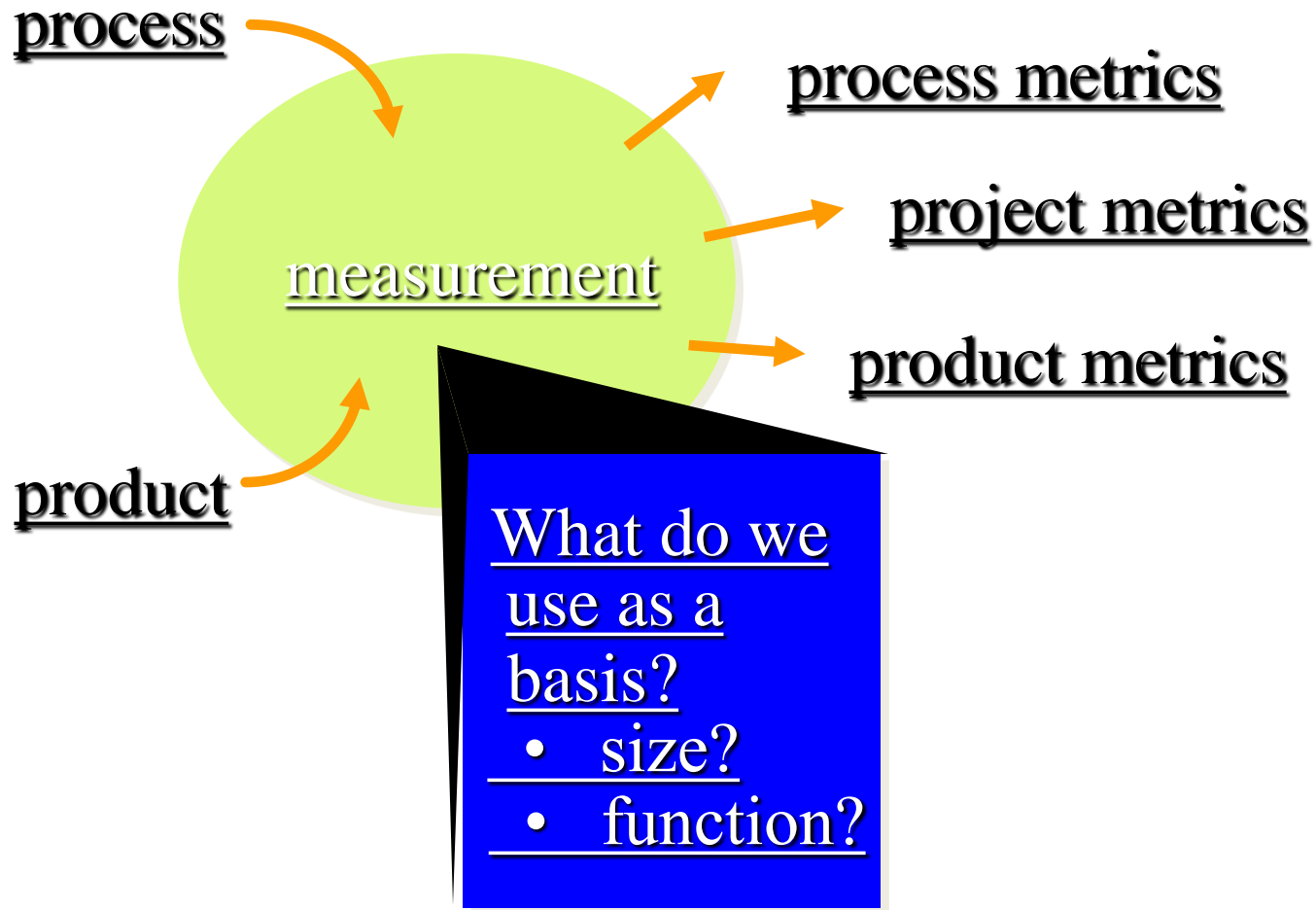
MODELING PRINCIPLES

- **Model with a PURPOSE** : A developer who uses AM should have a **specific goal in mind** before creating the model.
- **Use multiple models:**
 - i. There are many different models and notations that can be used to describe software.
 - ii. Only a **small subset is essential for most projects**.
- **Travel light.** As software engineering work proceeds, keeps only those models that will **provide long-term value for designing various applications**.

MODELING PRINCIPLES

- **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 ; 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 **efficiency of a software process** **“indirectly”**
 - 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 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)

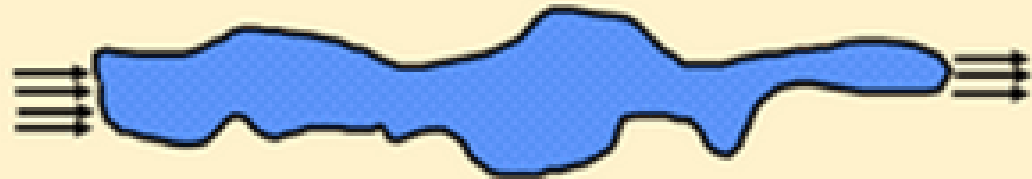
- CMMI (Capability Maturity Model Integration) is a proven industry framework to **improve product quality** and development efficiency for both hardware and software.
- CMMI, uses 5 levels to describe the maturity of the organization.
- Each maturity level has an associated set of **process area and 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

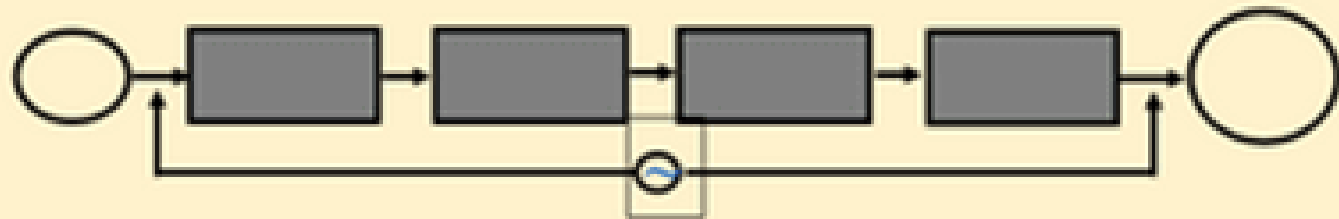
- A **maturity level** is a well-defined evolutionary plateau toward achieving a **mature software process**.
 - Each maturity level provides a layer in the foundation for continuous process improvement.
- A **capability level** is a well-defined evolutionary plateau describing the **organization's capability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

The CMM Maturity Levels

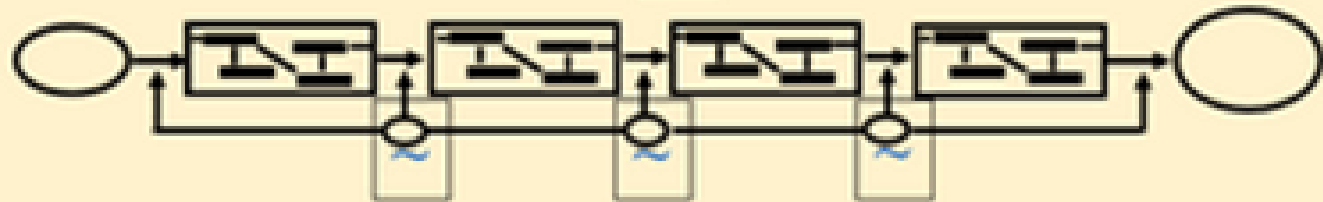
Maturity Level 1



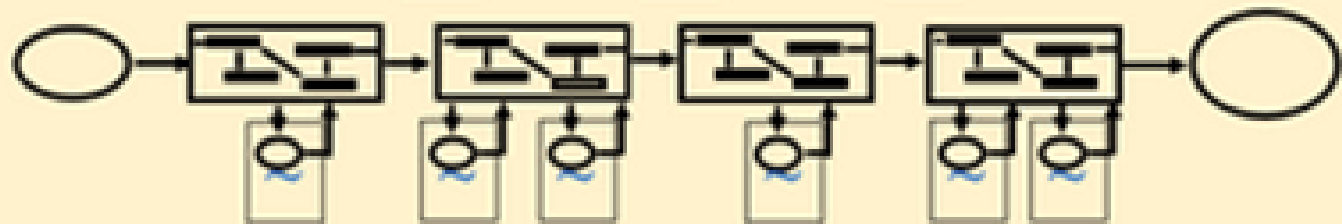
Maturity Level 2



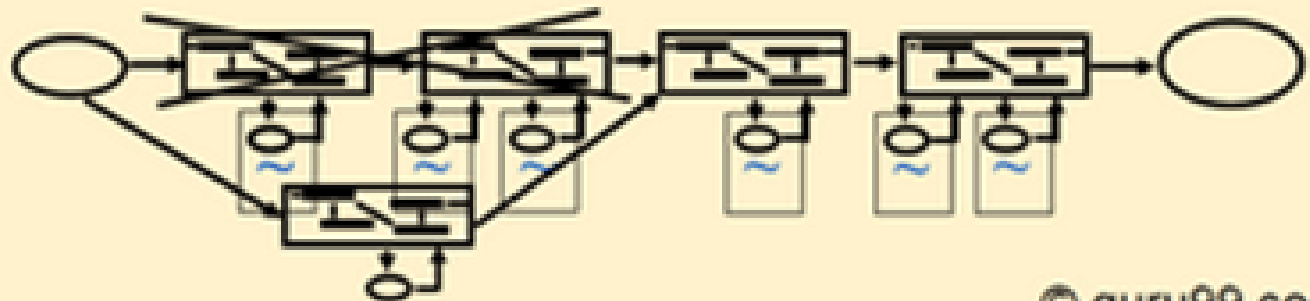
Maturity Level 3



Maturity Level 4



Maturity Level 5





Larsen &
Toubro



CGI Inc.



Cognizant



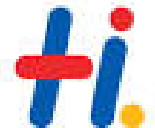
CSC
Covansys
Corporation



FCG
Software
Services...



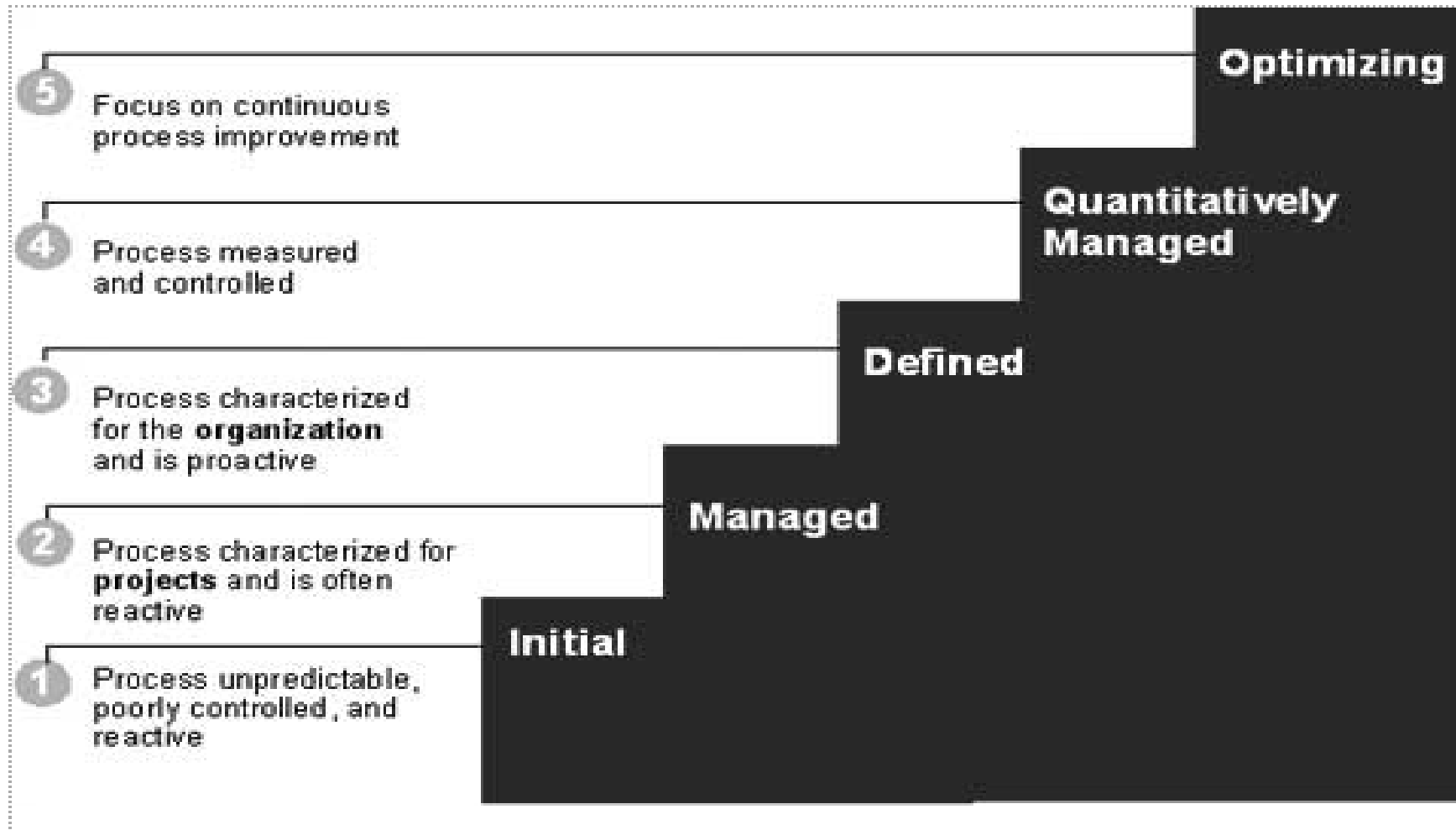
HCL
Technolo...



Hexaware
Technolo...

Five Levels of Software Process Maturity

CMML Staged Representation- Maturity Levels



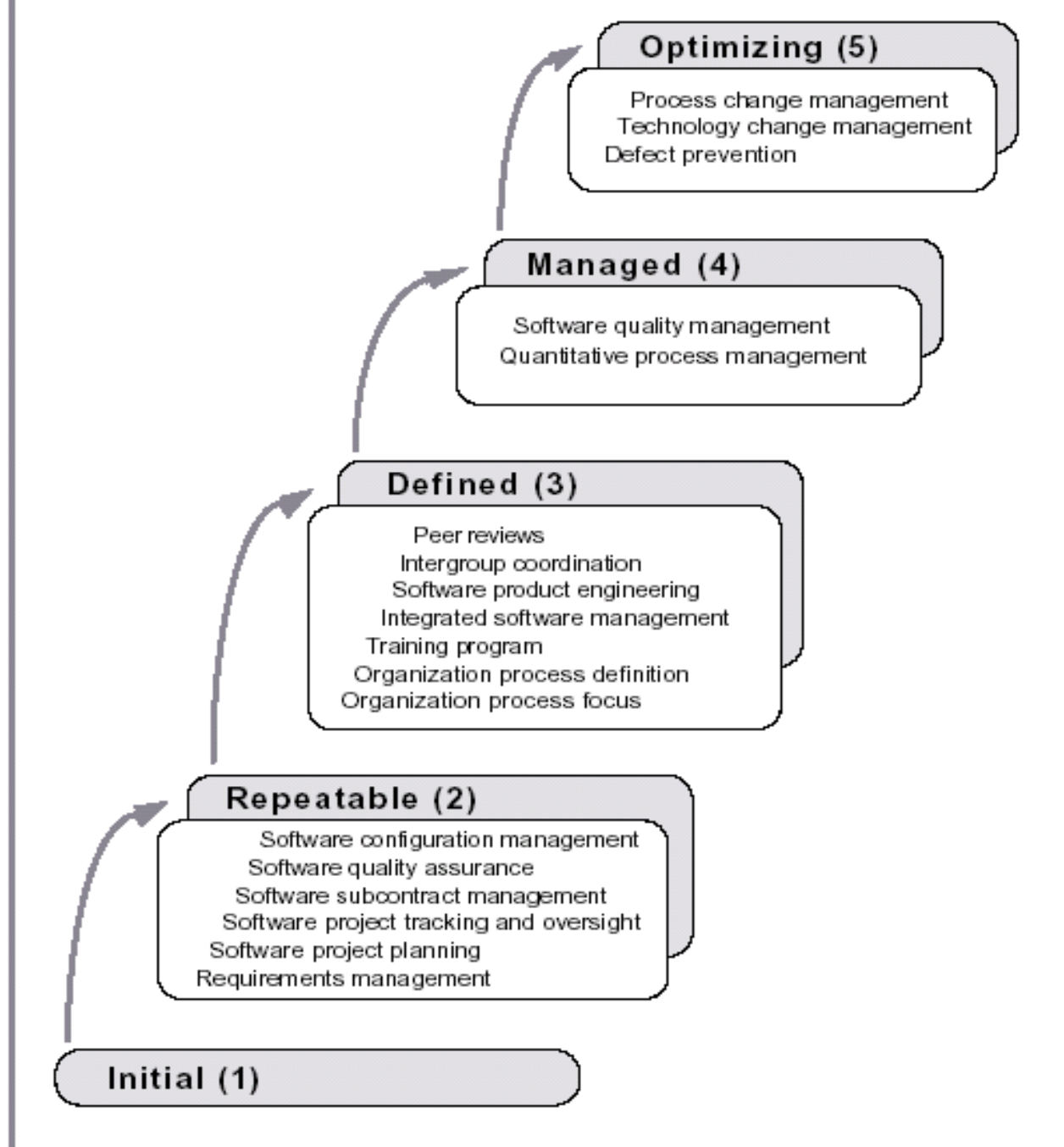


Figure 3.2 The Key Process Areas by Maturity Level

CHARACTERISTICS OF INITIAL (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.
- 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 MANAGED (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 DEFINED (LEVEL 3)

- 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).

(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 QUANTATIVELY MANAGED

(LEVEL 4)

- 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.
- 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.

(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**.
- 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 OPTIMIZING (LEVEL 5)

- 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.
- Focuses on **continually improving process** performance through both incremental and innovative technological improvements.
- Processes are concerned with **addressing special 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)

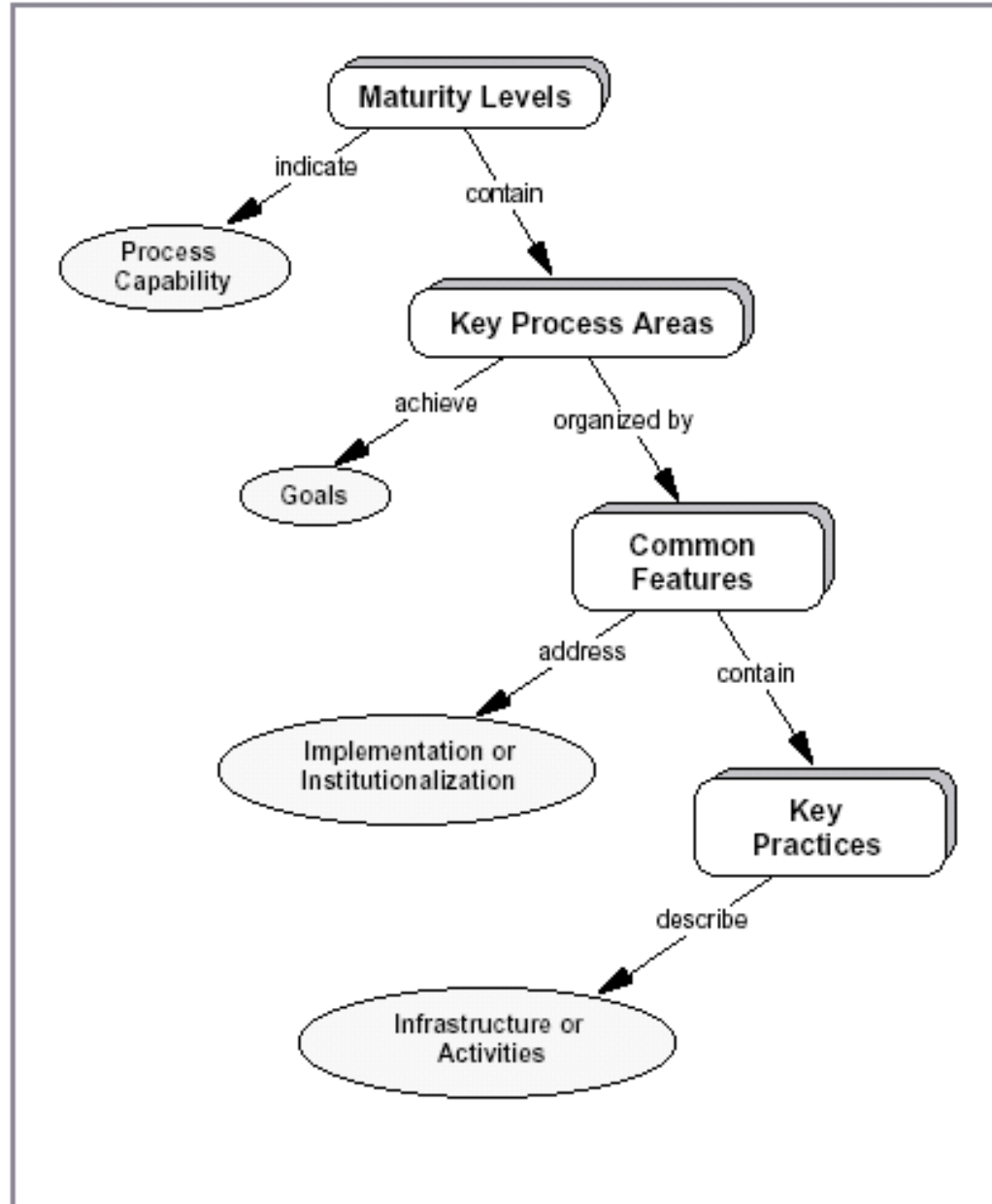
- All level 3 criteria satisfied.
- **Quantitative objectives** for quality and process performance are established and used as criteria in managing process.

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



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

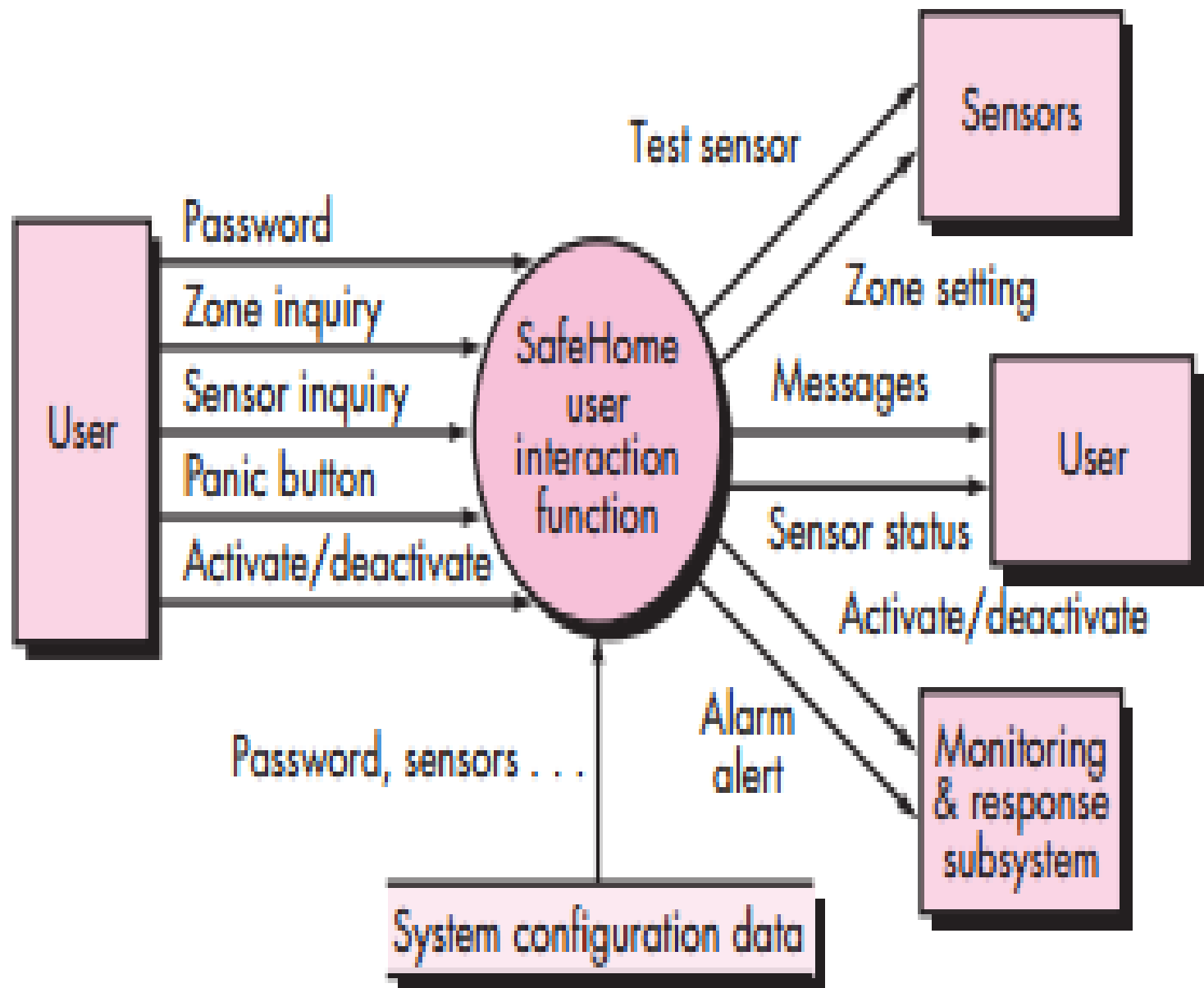
- Project **scope** must be understood.
- **Elaboration** (decomposition) is necessary.
- **Historical metrics** are very helpful.
- At least **two different techniques** should be used.
- **Uncertainty** is inherent in the process.

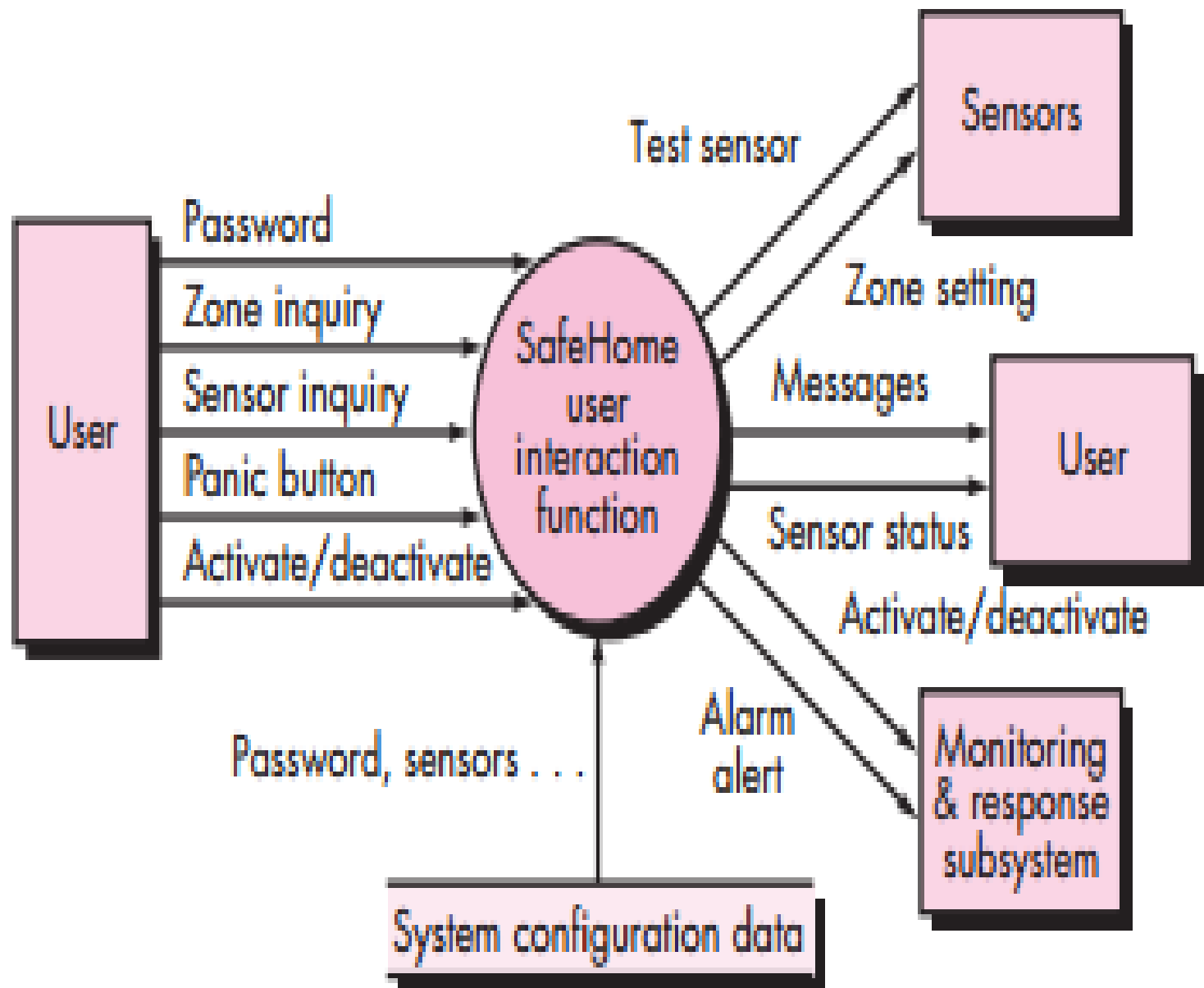
Product Metrics

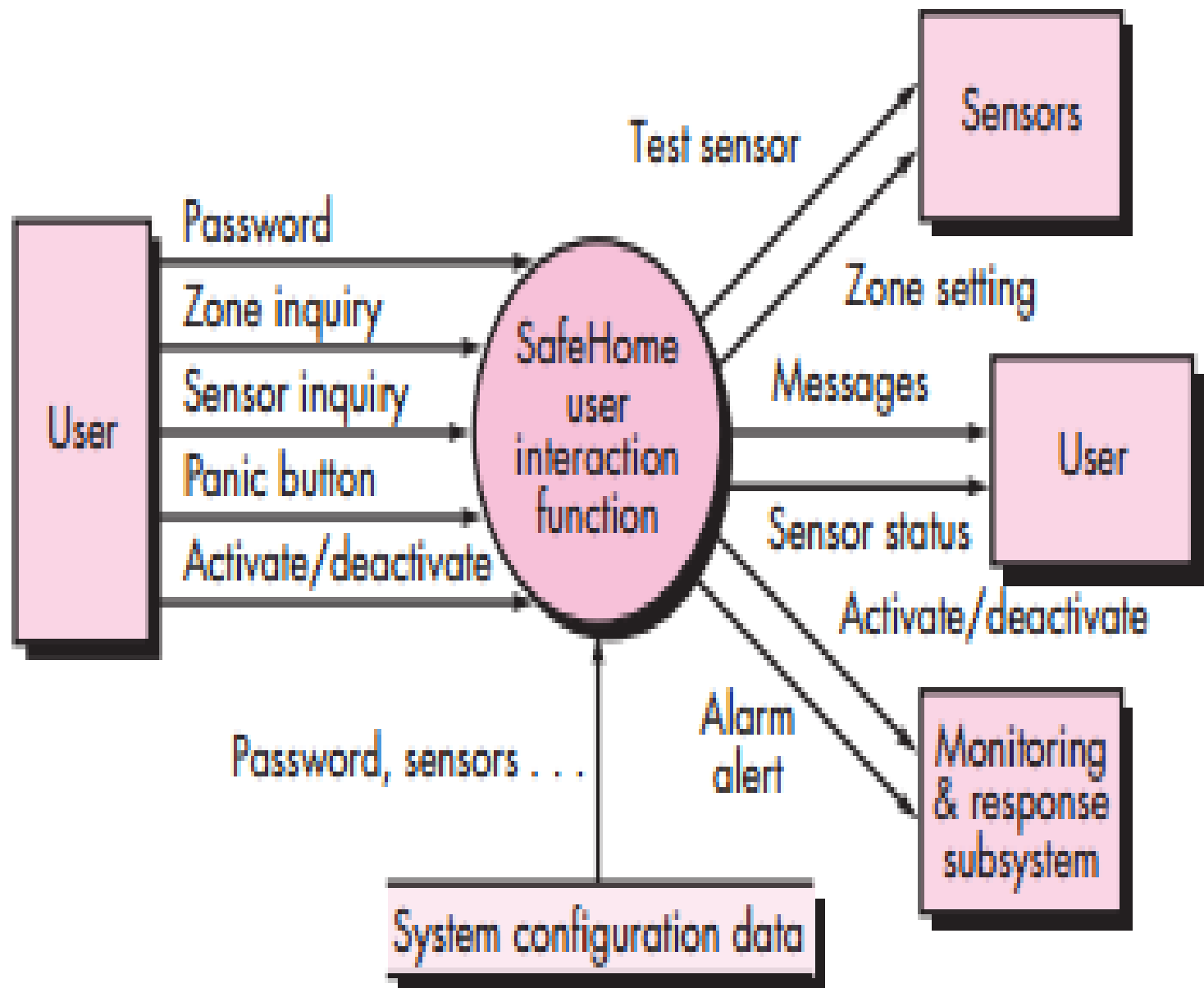
- Product metrics of computer software provides us with a **systematic way to assess quality** based on a set of clearly defined rules.
- Product metrics help in assessing **efficiency, reliability, complexity, understandability and maintainability** of software product.

FUNCTION POINT (FP)

- 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)

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

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

- 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)

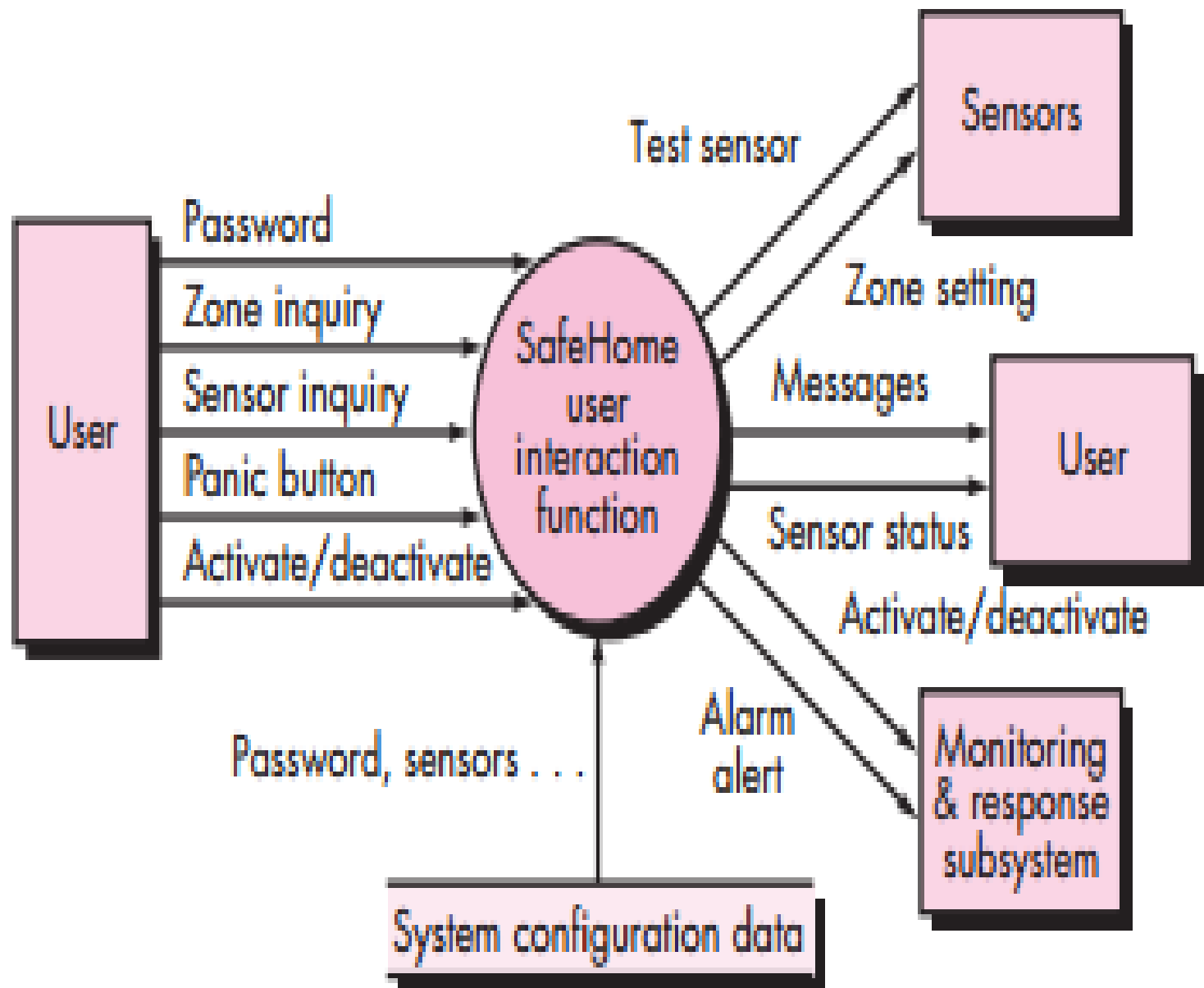
- Each of these questions is answered for VAF using a scale that ranges from 0 (not important or applicable) to 5 (absolutely essential).

Information Domain Value	Weighting factor				
	Count		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



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.

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)

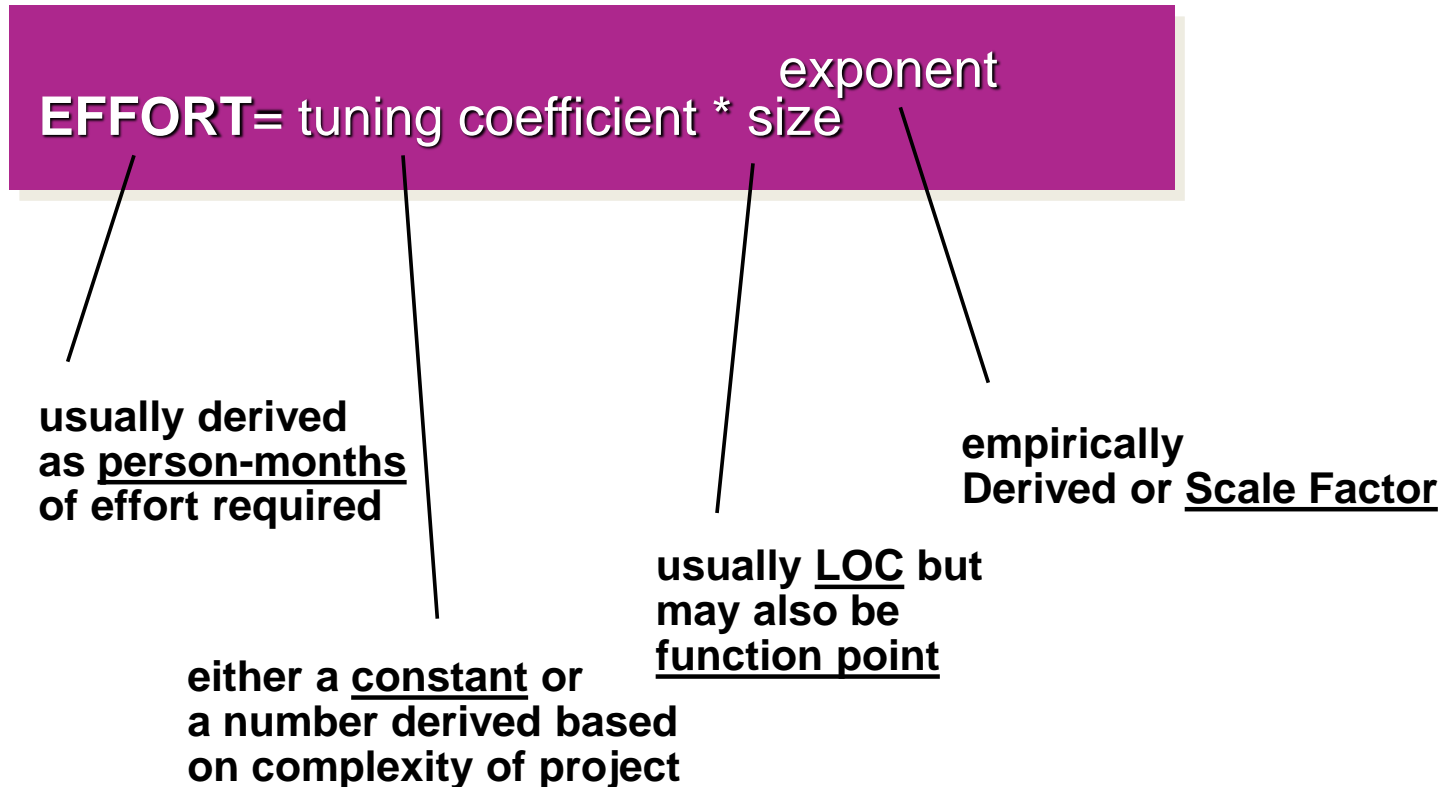
Two major types of LOC

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

[LOC example](#)

EMPIRICAL ESTIMATION MODELS

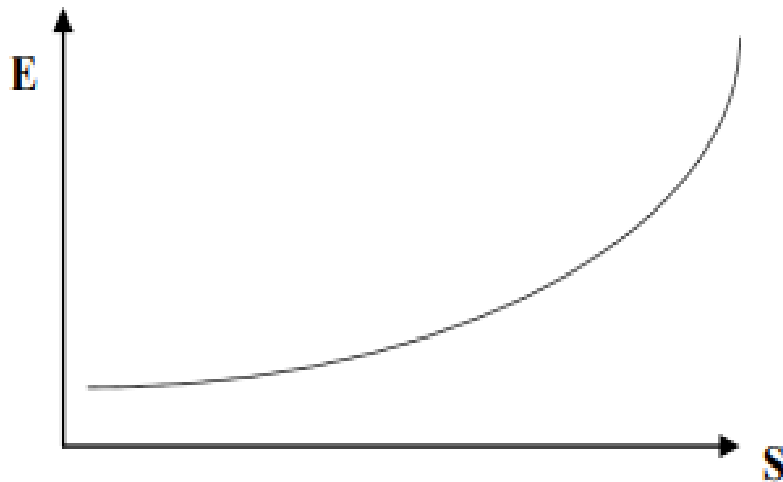
General form:



Empirical Estimation Models

- Graph demonstrates that the amount of effort accelerates as size increases

The relationship seen between development effort and software size is generally:



COCOMO

- Cocomo (Constructive Cost Model) is a regression model based on LOC, i.e **number of Lines of Code**.
- It is a procedural cost estimate model for software projects and often used as a process of reliably predicting the various parameters associated with making a project such as
 - size, effort, cost, time and quality.
- It was proposed by **Barry Boehm** in 1970 and is based on the study of 63 projects, which make it one of the best-documented models.

- The key parameters which define the quality of any software products, which are also an outcome of the COCOMO are primarily

EFFORT & SCHEDULE

- **Effort:** Amount of labor that will be required to complete a task. It is measured in person-months units.
- **Schedule:** Simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put. It is measured in the units of time such as weeks, months.

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)**

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

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

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

where

E = effort in person-months or person-years

t = project duration in months or years

B = “special skills factor”

P = “productivity parameter”

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

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

Estimated Effort Estimation

$E = 180 Bt^3$ in person-months for $E \geq 20$ person-months

$B \rightarrow$ Special skills factor

$t \rightarrow$ project duration

$$t_{\min} = 8.14 (33200/12000)^{0.43}$$

$$t_{\min} = 12.6 \text{ calendar months}$$

$$E = 180 \times 0.28 \times (1.05)^3$$

$$E = 58 \text{ person-months}$$

Why Are Projects Late?

- **Unrealistic deadline** established by someone outside the software development group
- **Changing customer requirements** that are not reflected in schedule changes
- 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
- 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**

Compartmentalization

- The project must be compartmentalized into a number of **manageable activities, actions and tasks.**
- To accomplish compartmentalization, both the **product and the process are decomposed.**

Interdependency

The **interdependency** of each compartmentalized activity, action, or task must be determined.

Some tasks must occur in **sequence** while others can occur in **parallel**.

Time allocation

- Each task to be scheduled must be allocated some number of **work units**.
- In addition, each task must be assigned a **start date and a completion date** that are a function of the inter dependencies and whether work will be conducted on a full-time or part-time basis.

Effort validation

As time allocation occurs, the project manager must ensure that **no more than the allocated number of people** have been scheduled at any given time.

Defined responsibilities

Every task that is scheduled should be assigned to a **specific team member**.

Defined outcomes

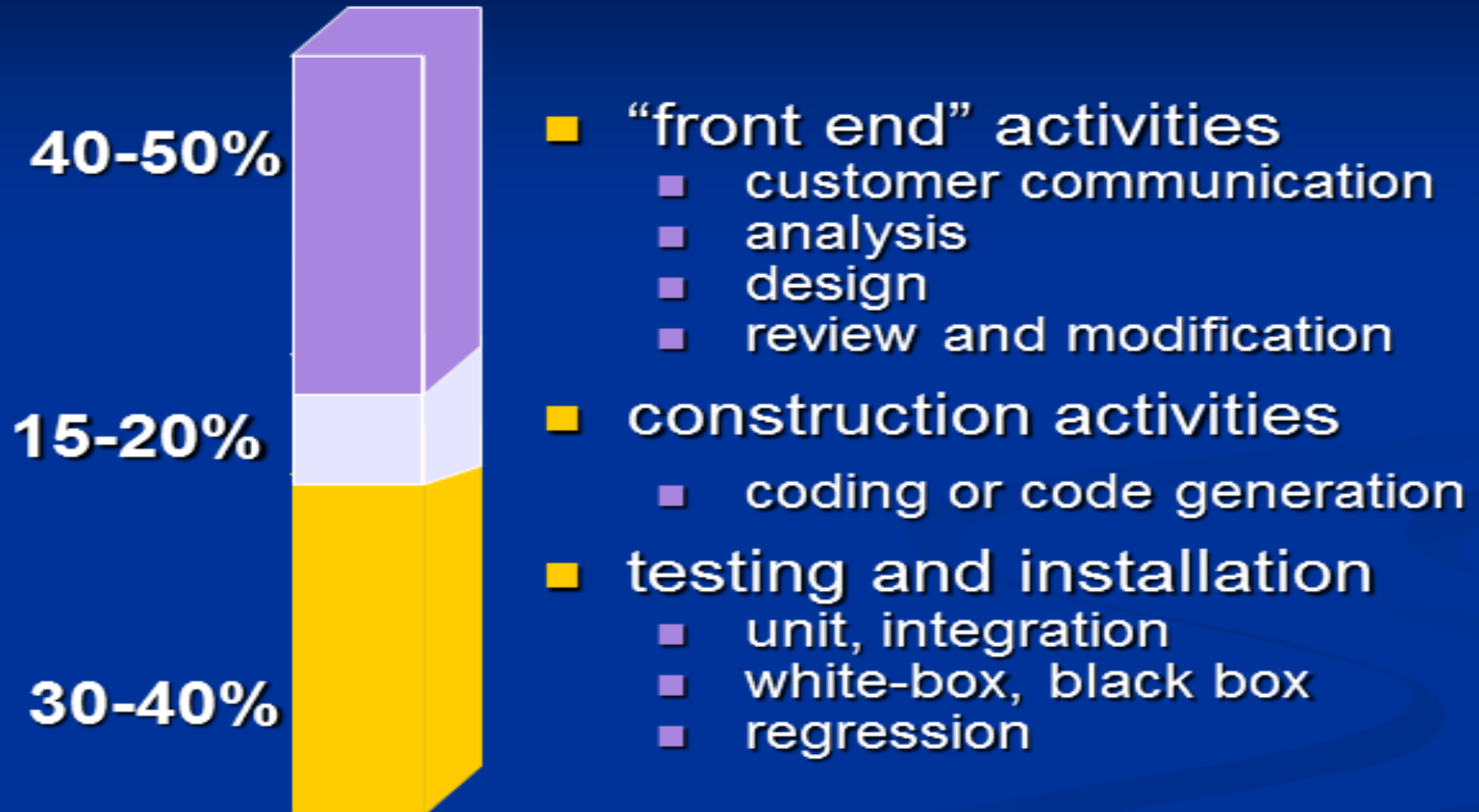
Every task that is scheduled should have a **defined outcome.**

For software projects, the outcome is normally a work product or a part of work product.

Defined milestones

- Every task or group of tasks should be associated with a project milestone.
- A milestone is accomplished when one or more work products has been reviewed for quality.

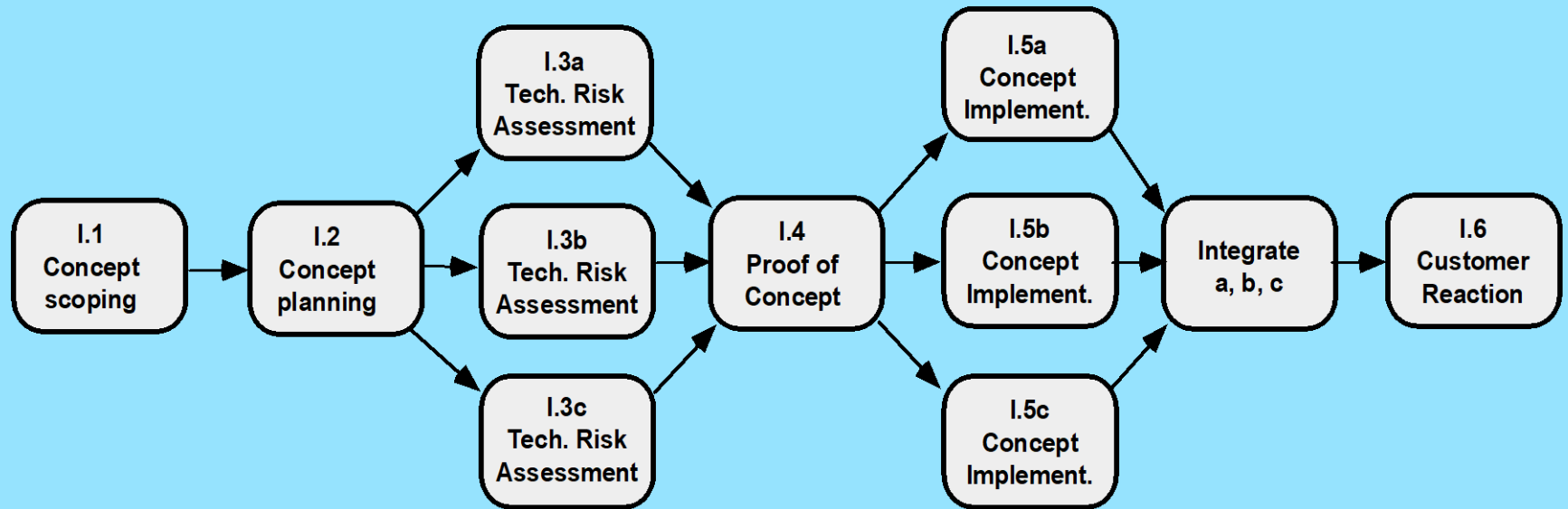
Effort Allocation (40-20-40 rule)



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

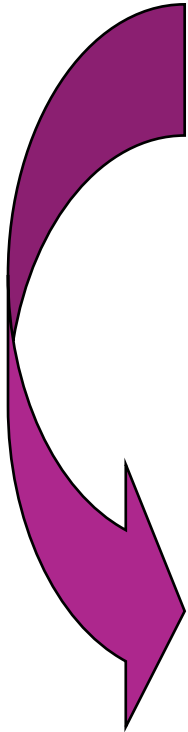


Three I.3 tasks are applied in parallel to 3 different concept functions

Three I.5 tasks are applied in parallel to 3 different concept functions

Task Set Refinement

Concept scoping



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.

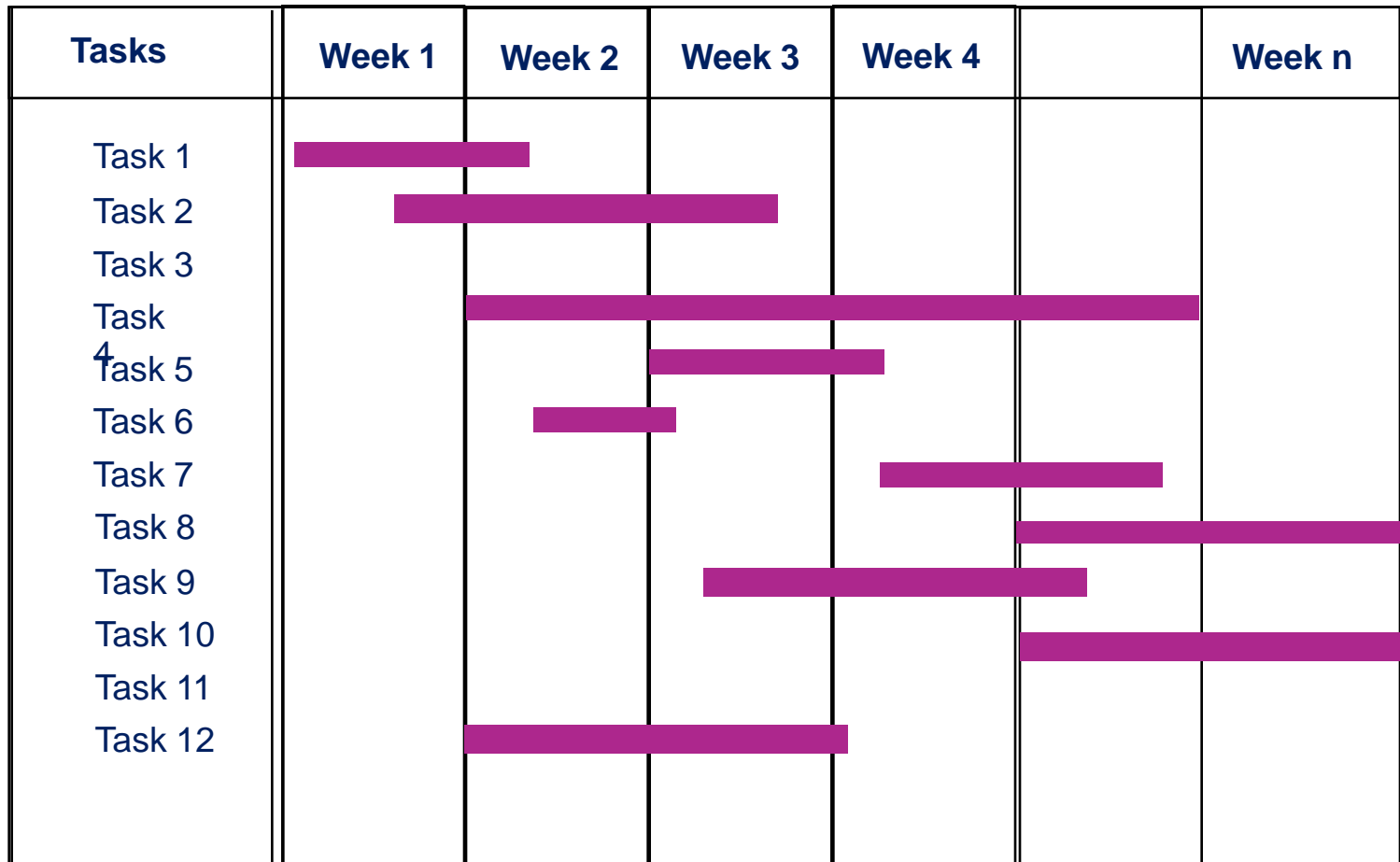
Scheduling

- 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**

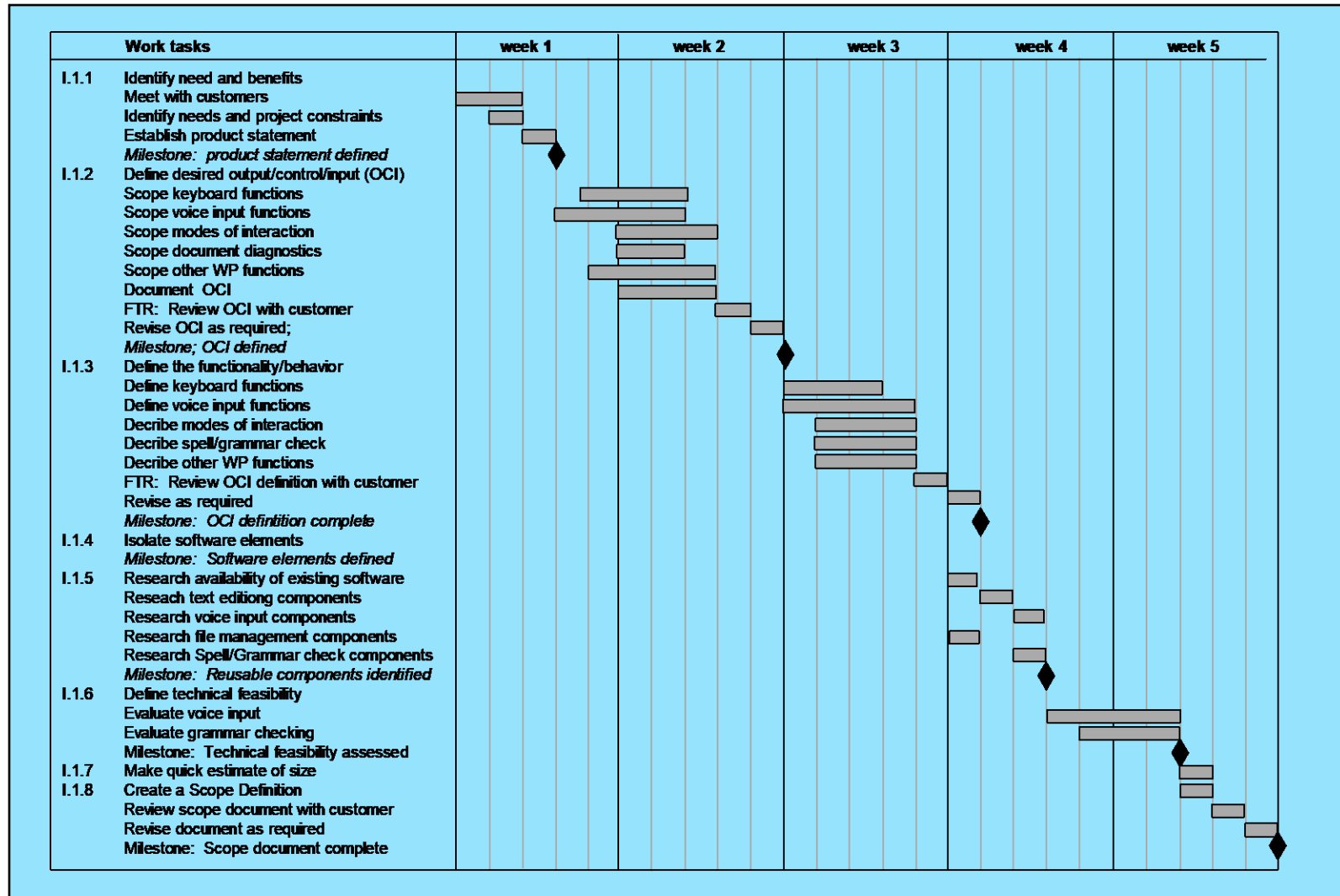
Scheduling

- 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.”

Earned Value Analysis (EVA)

- The earned value system provides a common value scale for every [software project] task, regardless of the type of work being performed.

STEPS TO DETERMINE EVA

1. The *budgeted cost of work scheduled (BCWS)* is determined for each work task represented in the schedule.
 - During estimation, the work (in person-hours or person-days) of each software engineering task is planned.
 - $BCWS_i$ is the effort planned for work task i .

Steps to determine EVA

2. The BCWS values for all work tasks are summed to derive the ***budget at completion (BAC)***.

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

Steps to determine EVA

3. value for ***budgeted cost of work performed (BCWP)*** is computed.

BCWP = sum of the BCWS values for all work tasks that have **actually been completed**

- Distinction between BCWS and BCWP

Steps to determine EVA

- Given values for BCWS, BAC, and BCWP, important progress indicators can be computed:

$$\text{Schedule performance index, SPI} = \frac{\text{BCWP}}{\text{BCWS}}$$

$$\text{Schedule variance, SV} = \text{BCWP} - \text{BCWS}$$

- -

$$\text{Percent scheduled for completion} = \frac{\text{BCWS}}{\text{BAC}}$$

Earned Value Analysis (EVA)

$$\text{Percent complete} = \frac{\text{BCWP}}{\text{BAC}}$$

- Actual Cost of Work Performed (ACWP)

$$\text{Cost performance index, CPI} = \frac{\text{BCWP}}{\text{ACWP}}$$

$$\text{Cost variance, CV} = \text{BCWP} - \text{ACWP}$$