

Unit-1

Introduction to **Software & Software Engineering**

Reference Book:
Software Engineering -A Practitioner's Approach (Seventh Edition) -
Roger S. Pressman.

Chapter 1: Software & Software Engineering
Chapter 2: Process Models(till 2.3)



Outline

- Software, Characteristics of Software, Software Application Domains
- Software Engineering
- Software Myths
 - Management Myth
 - Customer Myth
 - Practitioner's/Developer Myth)
- Software Engineering Layered Approach
- Software Process, Process Framework Activities , Umbrella Activities
- Software Process Models
 - The Waterfall Model
 - Incremental Process Model
 - Prototyping Model, Spiral Model
 - Spiral Model
 - Rapid Application Development Model (RAD)
- Component based Development

What is Software?

Software is

- 1) **Computer program** that when executed provide desired features, function & performance
- 2) **Data Structure** that enable programs to easily manipulate information
- 3) **Descriptive information** in both hard and soft copy that describes the operation and use of programs



**Computer
Program**



**Data
Structur
e**



**Documents
Soft &
Hard**

Program vs Software:

- A program is a **set of instructions that are given to a computer** in order to achieve a specific task
- A program is **one of the stages** involved in the development of the software.
- Software is when a program is made available for **commercial business and is properly documented along with its licensing.**
- Software development usually follows **a life cycle**
 - which involves the feasibility study of the project, requirement gathering, development of a prototype, system design, coding, and testing.

Software=Program+documentation+licensing.

Dual Role of Software: It is both a product and a vehicle for delivering a product

Software is a **product**

- **Delivers computing potential**
- **Produces, manages, acquires, modifies, displays, or transmits information**
- **Modern software is developed by teams of software specialists**

Software is a **vehicle for delivering a product**

- **Supports or directly provides system functionality**
- **Controls other programs (e.g., an operating system)**
- **Effects communications (e.g., networking software)**
- **Helps build other software (e.g., software tools)**

Software is dead.....!

The **old School view of Software**

- You buy it
- You own it &
- It's your job to manage it
- That is coming to an end



Because of **web 2.0** & extensive **computing power**, there is a different generation of software

- It is delivered via Internet
- It looks exactly like it's residing on each user's computing device
- Actually it reside on far away server



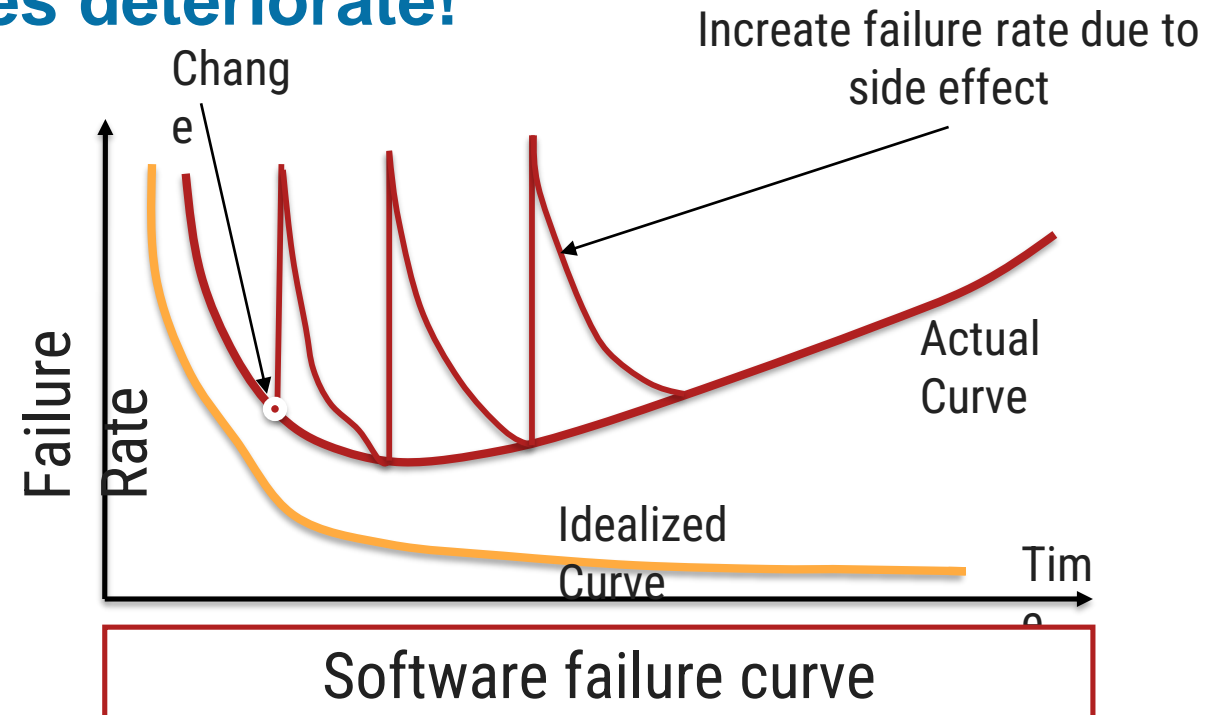
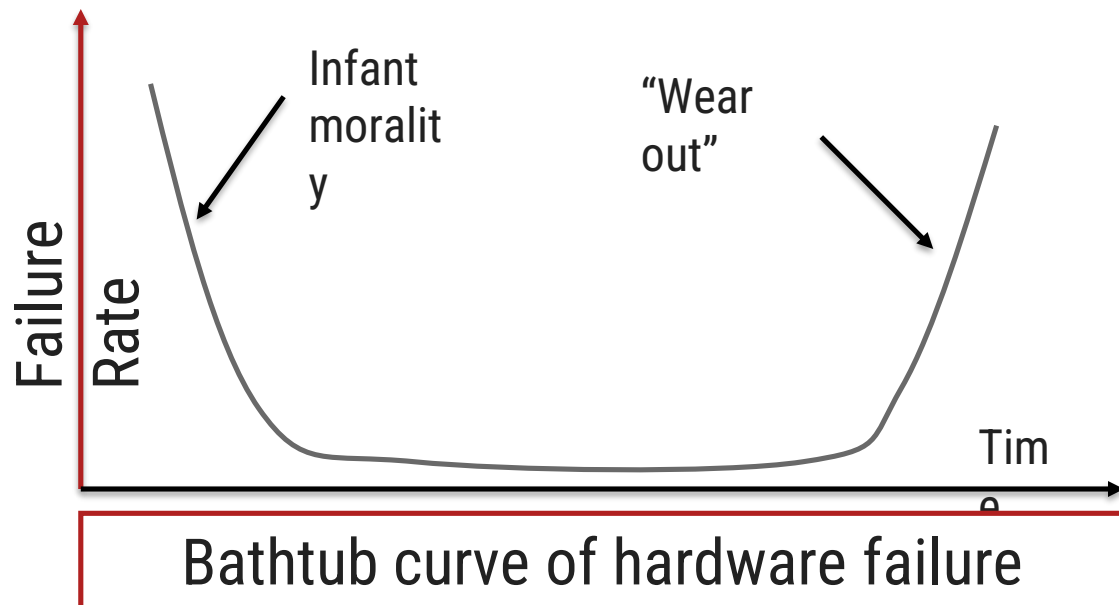
Characteristics of Software

Software is developed or engineered

→ It is not manufactured like hardware

- Manufacturing phase can introduce quality problem that are nonexistent (or easily corrected) for software
- Both requires construction of “product” but approaches are different

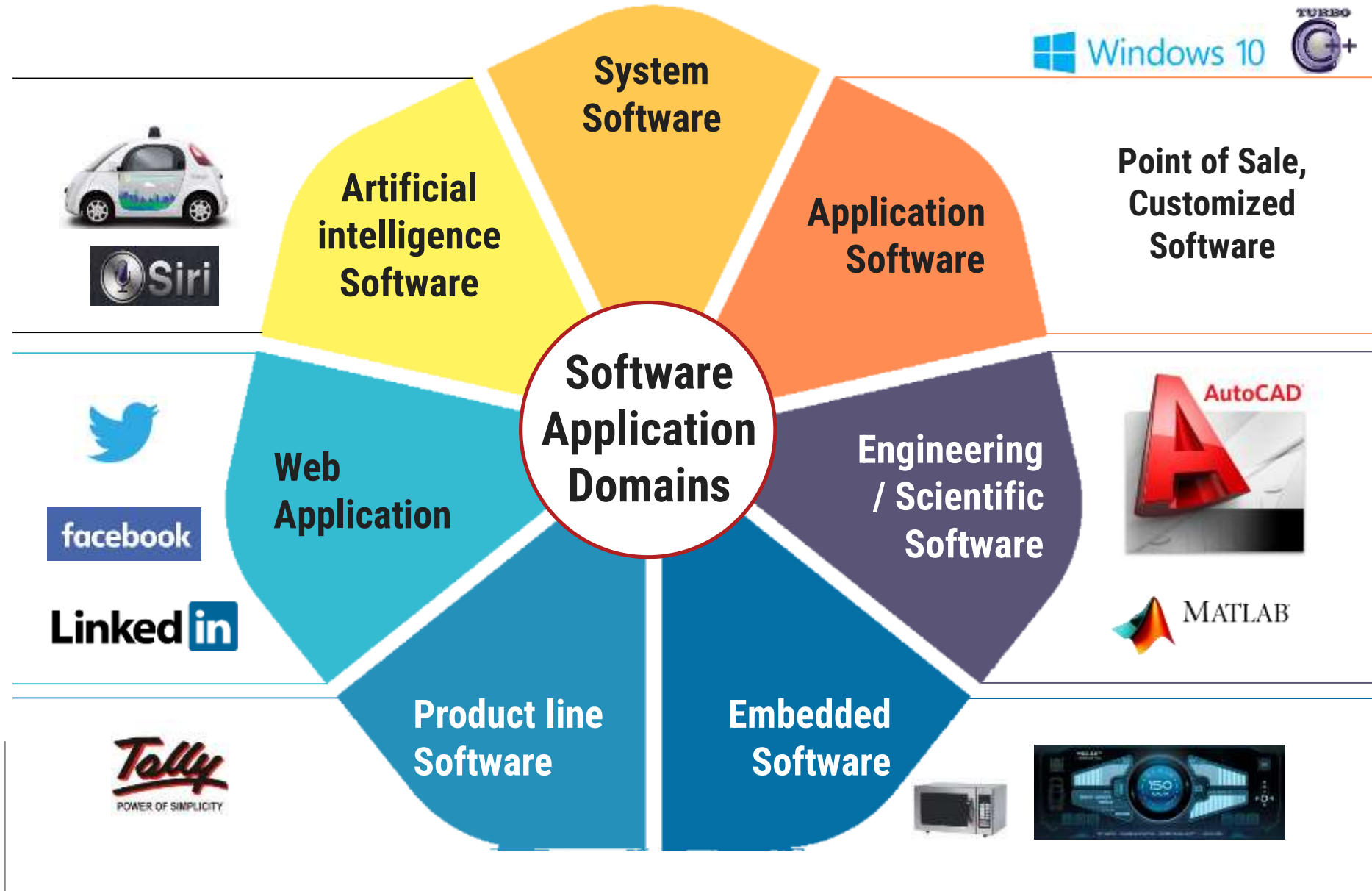
Software doesn't “wear-out”- But it does deteriorate!



Characterics of Software

- Although the industry is moving toward component-based construction, most software continues to be **custom built**.
- It can be **complex**, meaning it can have many interrelated parts and features.
- It can be affected by changing requirements, meaning it may need to be **updated or modified as the needs of** users change.
- It can be affected by bugs and other issues, meaning it may need to be **tested and debugged** to ensure it works as intended.

Software Application Domains



Software Application Domains

System software—a collection of programs written to service other programs. The systems software area is characterized by heavy interaction with computer hardware; heavy usage by multiple users; concurrent operation that requires scheduling, resource sharing, and sophisticated process management; complex data structures; and multiple external interfaces. (it is a program used to manage the other program. Ex: operating system, compiler, different device drivers)

Application software—stand-alone programs that solve a specific business need. Applications in this area process business or technical data in a way that facilitates business operations or management/technical decision

Engineering/scientific software—has been characterized by “number crunching” algorithms. Applications range from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing.

Embedded software—resides within a product or system and is used to implement and control features and functions for the end user and for the system itself. Embedded software can perform limited and esoteric functions (e.g., keypad control for a microwave oven) or provide significant function and control capability (e.g., digital functions in an automobile such as fuel control).

Product-line software—designed to provide a specific capability for use by many different customers. Product-line software can focus on a limited and esoteric marketplace (e.g., inventory control products) or address mass consumer markets (e.g., word processing, spreadsheets, computer graphics).

Artificial intelligence software—makes use of non-numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis. Applications within this area include robotics, expert systems, pattern recognition (image and voice), artificial neural networks.

Web applications—called “WebApps,” s can be little more than a set of linked hypertext files that present information using text and limited graphics. WebApps are evolving into sophisticated computing environments that not only provide stand-alone features, and content to the end user, but also are integrated with corporate databases and business applications.

Software engineering is the establishment and use of **sound engineering principles** in order to obtain **economically software** that is **reliable and works** efficiently in **real machines**.

Software Engineering is the science and art of building (designing and writing programs) a software systems that are:

- 1) on **time**
- 2) on **budget**
- 3) with acceptable **performance**
- 4) with **correct operation**

The IEEE has developed a more comprehensive definition when it states:

Engineering is the process of **designing and building** something that serves a **particular purpose** and finds a **cost-effective solution** to problems.

*Software Engineering is a **systematic, disciplined, quantifiable study and approach to the design, development, operation, and maintenance** of a software system.*

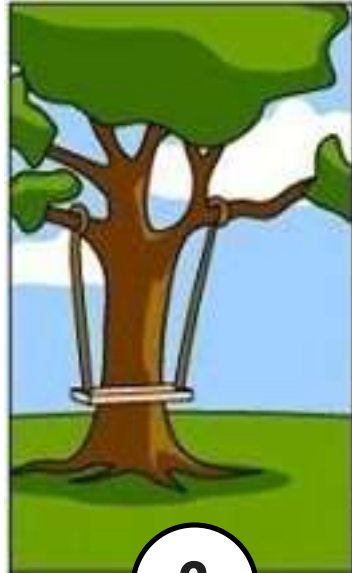
Why to Study Software Engineering?

Software Development Life Cycle **without** Software Engineering



1

How the
Customer
Explains
Requirement



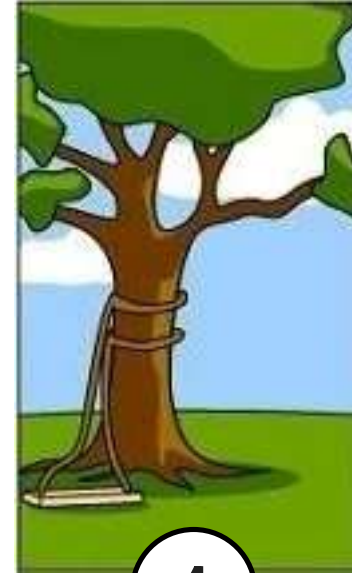
2

How the
Project
Leader
understand it



3

How the
System
Analyst
design it



4

How the
Programmer
Works
on it

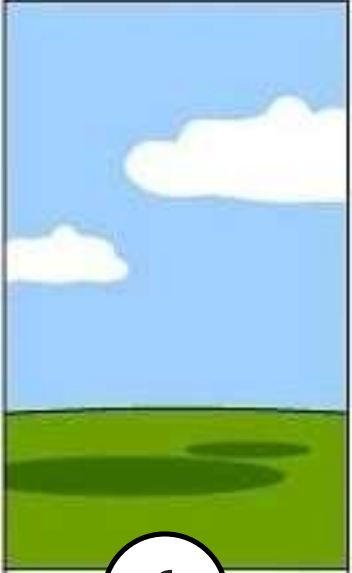


5

How the
Business
Consultant
describe it

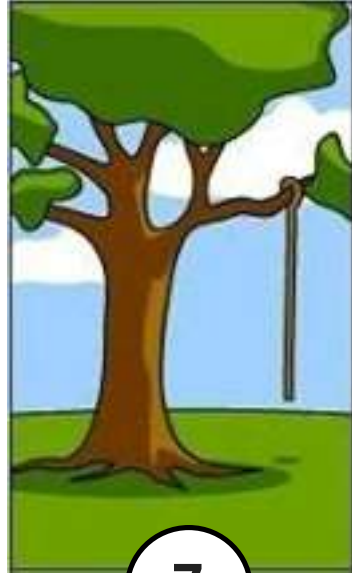
Why to Study Software Engineering?

Software Development Life Cycle **without** Software Engineering



6

How the
Project
documented



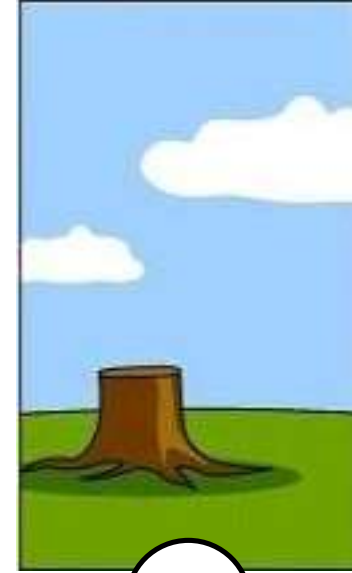
7

What
Operations
Installed



8

How the
Customer is
billed



9

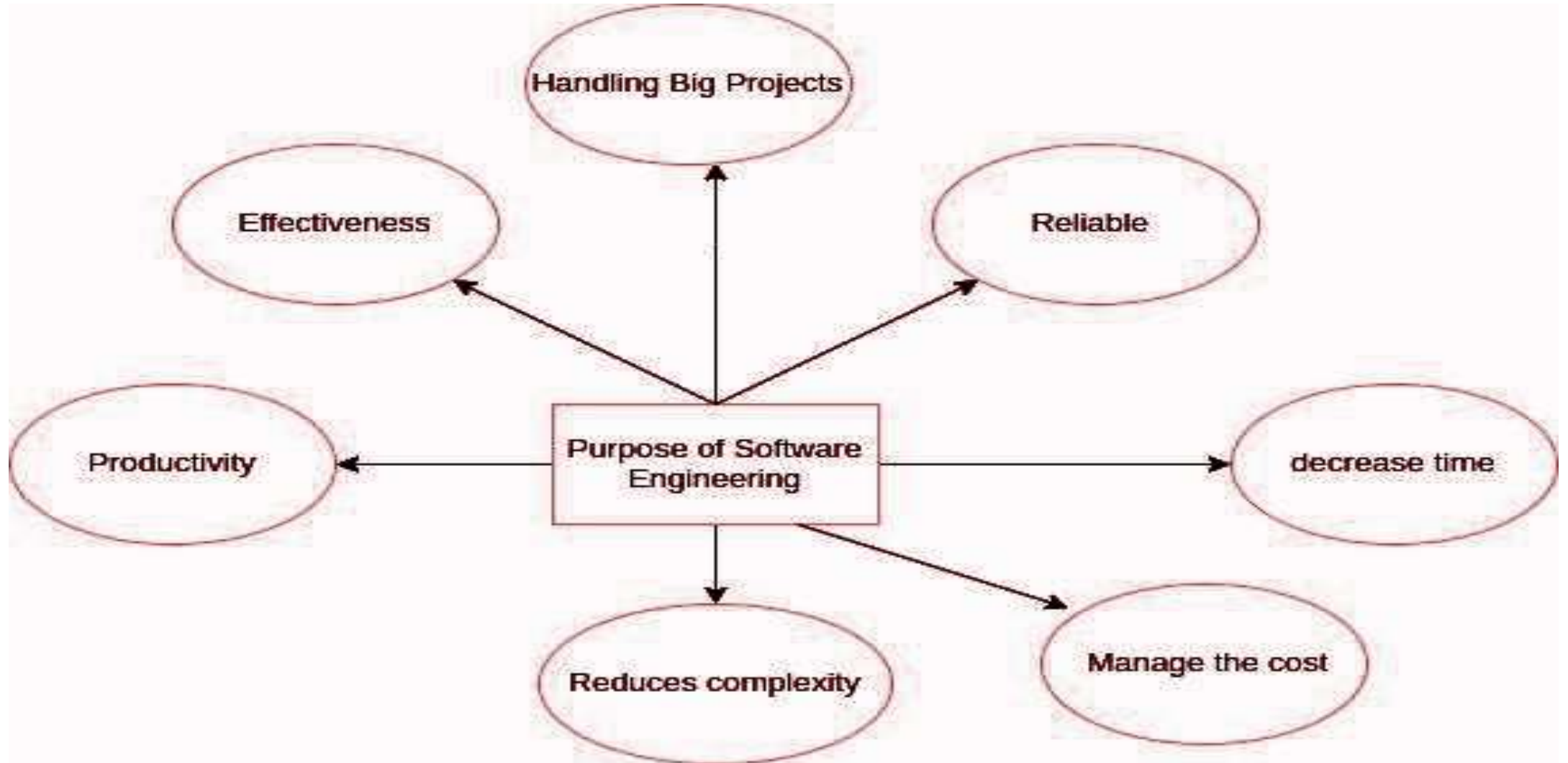
How it
was
supported



10

What the
customer
really needed

Why to Study Software Engineering?



SDLC **without** Software Engineering

Customer Requirement

- Have one trunk
- Have four legs
- Should carry load both passenger & cargo
- Black in color
- Should be herbivorous



Solution

- Have one trunk
- Have four legs
- Should carry load both passenger & cargo
- Black in color
- Should be herbivorous



Our value added,
also gives
milk

The software development **process** needs to be **engineered** to avoid the **communication gap** & to **meet the actual requirements** of customer within **stipulated budget** & **time**

Objectives of Software Engineering:

Functionality:

It refers to the degree of performance of the software against its intended purpose.

Reliability:

A set of attributes that bears on the capability of software to maintain its level of performance under the given condition for a stated period of time.

Efficiency:

It refers to the ability of the software to use system resources in the most effective and efficient manner. The software should make effective use of storage space and executive command as per desired timing requirements.

Usability:

It refers to the extent to which the software can be used with ease. the amount of effort or time required to learn how to use the software.

Maintainability:

It refers to the ease with which the modifications can be made in a software system to extend its functionality, improve its performance, or correct errors.

Portability:

A set of attributes that bears on the ability of software to be transferred from one environment to another, without or minimum changes.

- **Adaptability :**

In this case, the software allows differing system constraints and the user needs to be satisfied by making changes to the software.

- **Interoperability :**

Capability of 2 or more functional units to process data cooperatively.

- **Correctness :**

A software product is correct if the different requirements as specified in the SRS document have been correctly implemented.

Advantages of Software Engineering

Improved quality: By following established software engineering principles and techniques, software can be developed with fewer bugs and higher reliability.

Increased productivity: Using modern tools and methodologies can streamline the development process, allowing developers to be more productive and complete projects faster.

Better maintainability: Software that is designed and developed using sound software engineering practices is easier to maintain and update over time.

Reduced costs: By identifying and addressing potential problems early in the development process, software engineering can help to reduce the cost of fixing bugs and adding new features later on.

Increased customer satisfaction: By involving customers in the development process and developing software that meets their needs, software engineering can help to increase customer satisfaction.

Better team collaboration: By using Agile methodologies and continuous integration, software engineering allows for better collaboration among development teams.

Better scalability: By designing software with scalability in mind, software engineering can help to ensure that software can handle an increasing number of users and transactions.

Better Security: By following the software development life cycle (SDLC) and performing security testing, software engineering can help to prevent security breaches and protect sensitive data.

Disadvantages of Software Engineering

- **High upfront costs:** Implementing a systematic and disciplined approach to software development can be resource-intensive and require a significant investment in tools and training.
- **Limited flexibility:** Following established software engineering principles and methodologies can be rigid and may limit the ability to quickly adapt to changing requirements.
- **Bureaucratic:** Software engineering can create an environment that is bureaucratic, with a lot of process and paperwork, which may slow down the development process.
- **Complexity:** With the increase in the number of tools and methodologies, software engineering can be complex and difficult to navigate.

Limited creativity: The focus on structure and process can stifle creativity and innovation among developers.

High learning curve: The development process can be complex, and it requires a lot of learning and training, which can be challenging for new developers.

High dependence on tools: Software engineering heavily depends on the tools, and if the tools are not properly configured or are not compatible with the software, it can cause issues.

High maintenance: The software engineering process requires regular maintenance to ensure that the software is running efficiently, which can be costly and time-consuming.

Legacy Software

Some of these are state-of-the-art software—just released to individuals, industry, and government. But other programs are **older**, in some cases much older. These older programs—often referred to as legacy software—have been the focus of continuous attention and concern since the 1960s.

“Legacy software systems . . . were developed decades ago and have been continually modified to meet changes in business requirements and computing platforms. The proliferation of such systems is causing headaches for large organizations who find them costly to maintain and risky to evolve”

Many different reasons why a system might earn the "legacy" label

- End of life
- Outdated architecture
- Lack of internal system knowledge
- Lack of internal system skills
- Scalability
- Challenging to update and innovate

Most organizations **keep** their legacy systems because of at least one of the following reasons:

- **High migration costs:** the IT system is running aging or end-of-life technology that may lack current documentation, making migration complex and, usually, expensive.
- **Skills gap:** the technology is supported by “mature” developers with hard-to-find skills, or, in case of migration to another technology, organizations lack enough manpower to focus on the migration while keeping the business running as usual.
- **Fear:** legacy systems are often a mission-critical technology, and the organization is afraid of the impact changing it or replacing it can have on the business.

- **Familiarity:** In some cases, a company may keep its older software because its staff is familiar with it. Introducing new technology can disrupt the pace of operations and may require **extensive training**. It can also be expensive to invest in new software and devices.
- **Data loss:** When transferring to a new system, companies risk losing valuable data. Although it's uncommon, data may download incorrectly to the new software or become lost during data transfer. To avoid losing data, companies may perform a system backup or archive important information.

What to do with Legacy Software?

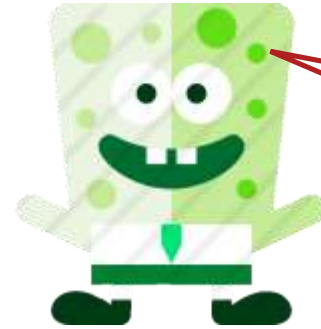
- **Software should be adapted to meet new computing environment and technology.**
- **Enhanced for new business requirements.**
- **Ensure new design is extensible and interoperable with other system.**

Software Myths

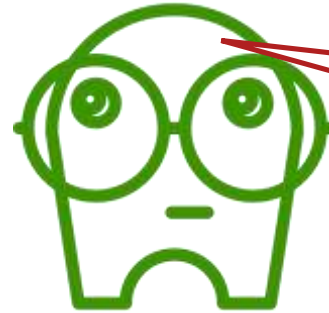
Beliefs about software and the process used to build it.



“Misleading Attitudes that cause serious problem” are myths.



Management Myths



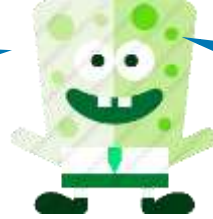
Customer Myths



Practitioner's
(Developer) Myths

Management myth - 1 & 2

We **have standards and procedures** to build a system, which is enough.



We have **the newest computers and development tools**.

Reality

Are software **practitioners** aware of standard's existence?

Does it **reflect modern software engineering** practice?

Is it **complete**?

Is it streamlined to **improve time to delivery** while **still maintaining a focus on quality**?

In many cases, the answer to all of these questions is "no."

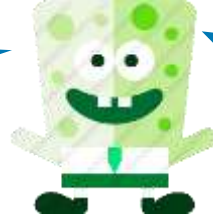
Reality

It **takes much more than the latest model** computers to do high-quality software development.

Computer-aided software engineering (CASE) tools are more important than hardware.

Management myth - 3 & 4

We **can add more programmers** and can catch up the schedule.



I **outsourced the development** activity, now I **can relax** and **can wait** for the **final** working **product**.

Reality

Software **development is not a mechanistic process** like manufacturing.

In the words of Fred Brooks : "**adding people to a late software project makes it later.**"

People who were **working** must **spend time educating** the newcomers.

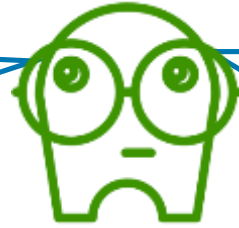
People can be added but only **in a planned and well-coordinated** manner.

Reality

If an **organization** does **not understand how to manage** and **control** software projects internally, it will invariably struggle when it outsources software projects.

Customer myth - 1 & 2

A **general statement of objectives** (requirements) is **sufficient** to start a development.



Requirement Changes can be **easily accommodated** because software is very flexible.

Reality

Comprehensive (**detailed**) **statements** of requirements is not always possible, an **ambiguous** (unclear) "**statement of objectives**" can lead to disaster.

Unambiguous (clear) requirements can be gathered only through effective and continuous communication between customer and developer.

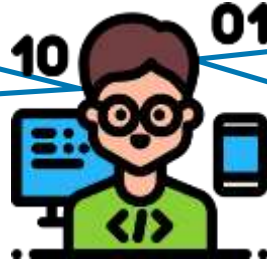
Reality

It is true that software **requirements change**, but the **impact** of change **varies with the time** at which it is introduced.

When requirements changes are requested early the cost impact is relatively small.

Practitioner's (Developer) myth – 1 & 2

Once we **write** the **program**, our **job is done**.



I **can't** access **quality until** it is **running**.

Reality

Experts say "**the sooner you begin 'writing code', the longer it will take you to get done.**"

Industry data indicates that 60 to 80 % effort expended on software will be after it is delivered to the customer for the first time.

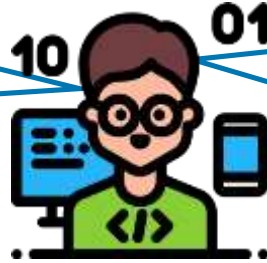
Reality

One of the most effective software **quality assurance mechanisms** can be **applied from the beginning** of a project - **the technical review**.

Software reviews are more effective "quality filter" than testing for finding software defects.

Practitioner's (Developer) myth – 3 & 4

Working **program** is the **only deliverable** work **product**.



Software engineering is about **unnecessary** documentation.

Reality

A working program is only one **part of a software configuration**.

A variety of work products (e.g., **models, documents, plans**) provide a foundation for successful engineering and, more important, guidance for software support.

Reality

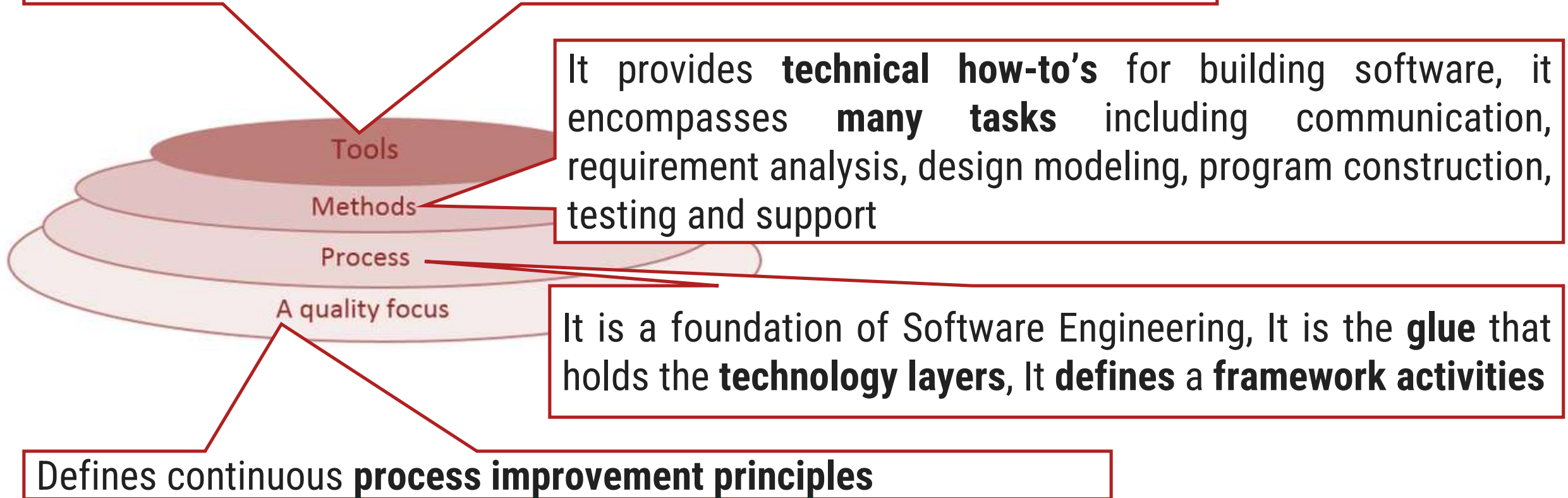
Software engineering is not about creating documents. It is about **creating a quality product**.

Better quality leads to reduced rework. And reduced rework results in faster delivery times.

Software Engineering Layered Approach

Software Engineering Tools **allows automation of activities** which helps to perform systematic activities. A system for the support of software development, called **computer-aided software engineering** (CASE).

Examples: Testing Tools, Bug/Issue Tracking Tools etc...



Software Engineering Layered Approach Cont.

Software Engineering is a layered technology

Quality

Main principle of Software Engineering is Quality Focus.

An **engineering approach** must have a **focus on quality**.

Total Quality Management (**TQM**), **Six Sigma**, **ISO 9001**, **ISO 9000-3**, **CAPABILITY MATURITY MODEL (CMM)**, **CMMI** & similar approaches encourages a continuous process improvement culture

Process Layer

It is a foundation of Software Engineering, It is the glue the holds the technology layers together and enables logical and timely development of computer software.

It **defines** a **framework** with activities for effective delivery of software engineering technology

It establish the context in which technical methods are applied, work products (models, documents, data, reports, forms, etc.) are produced, milestones are established, quality is ensured, and change is properly managed.

Software Engineering Layered Approach Cont.

Method

It provides **technical how-to's** for building software

It **encompasses many tasks** including communication, requirement analysis, design modeling, program construction, testing and support

Tools

Software engineering tools provide automated or semi-automated support for the process and the methods

Computer-aided software engineering (**CASE**) is the scientific application of a **set of tools** and **methods** to a software system which is meant to **result in high-quality, defect-free, and maintainable software products**.

CASE tools automate many of the activities involved in various life cycle phases.

Software Process

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

A process is not a **rigid prescription** for how to build the software, rather it is **adaptable approach** that enables the people doing the work to **pick** and **choose** the **appropriate set of work actions** and tasks

An **activity** try to **achieve** a **broad objective** (e.g., communication with stakeholders)

An **activity** is **applied** regardless of the **application domain**, **size of the project**, **complexity of the effort**, or **degree of accuracy** with which software engineering is to be applied.

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

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

Each of these activities, actions & tasks **reside within** a **framework** or model

Software Process

Figure represents “The Software Process”

Each framework **activity is populated** by **set of** software engineering **actions**

Each software engineering **action is defined by** a **task set** that identifies work to be completed, product to be produced, quality assurance points & milestones to indicate progress

The **purpose** of software process is

to **deliver** software in **timely** manner and within sufficient **quality to satisfy** those

- ➔ Who has given proposal for software development and
- ➔ Those who will use software

Software Process Framework

Process

framework

Umbrella activities

framework activity #1

Software Engineering action

#1 Task

Sets

...

Software Engineering action

#1 Task

Sets

...

Work tasks

Work products

Quality assurance points

Work tasks






Work products

Quality assurance points

framework activity #n

Process Framework Activities (CPMCD)

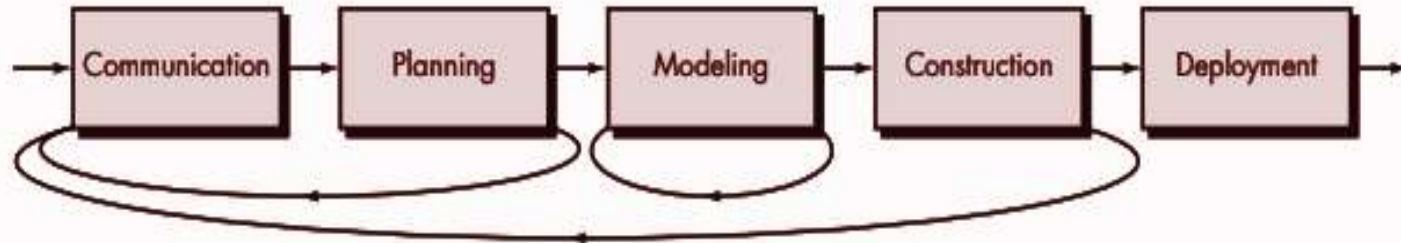
A process framework establishes the foundation for complete software engineering process, it encompasses five activities

Communication		Communication with Customers / stockholders to understand project requirements for defining software features	Planning		Software Project Plan which defines workflow that is to follow. It describes technical task, risks, resources, product to be produced & work schedule
Modeling		Creating models to understand requirements and shows design of software to achieve requirements	Construction		Code Generation (manual or automated) & Testing (to uncover errors in the code)
Deployment		Deliver Software to Customer Collect feedback from customer based on evaluation Software Support			

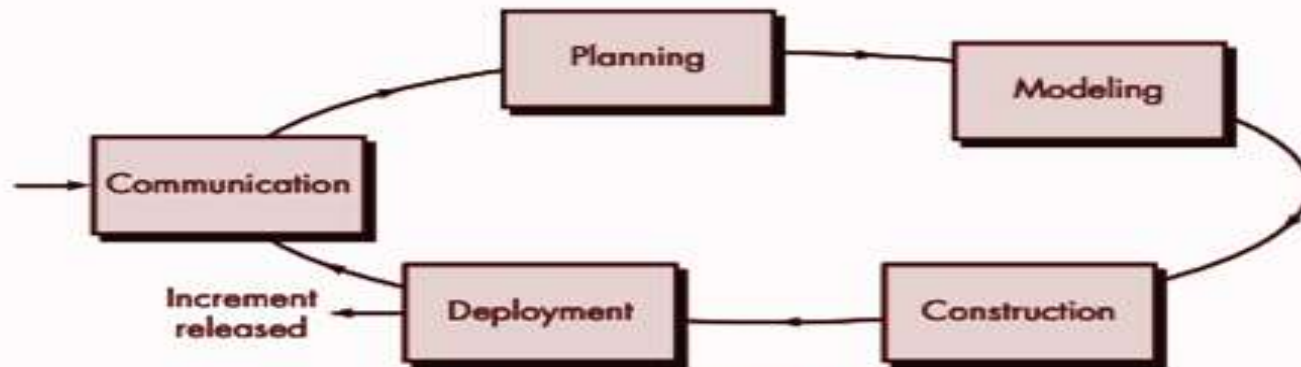
PROCESS FLOWS



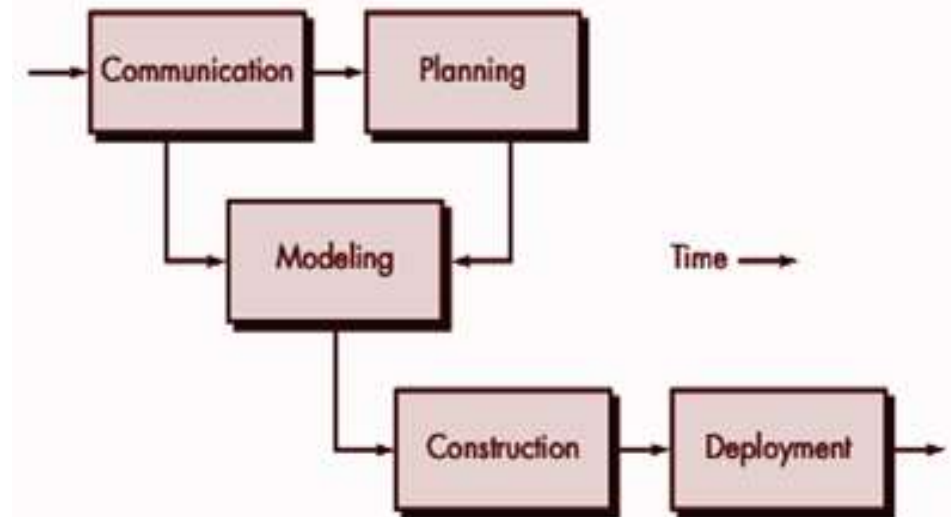
(a) Linear process flow



(b) Iterative process flow



(c) Evolutionary process flow

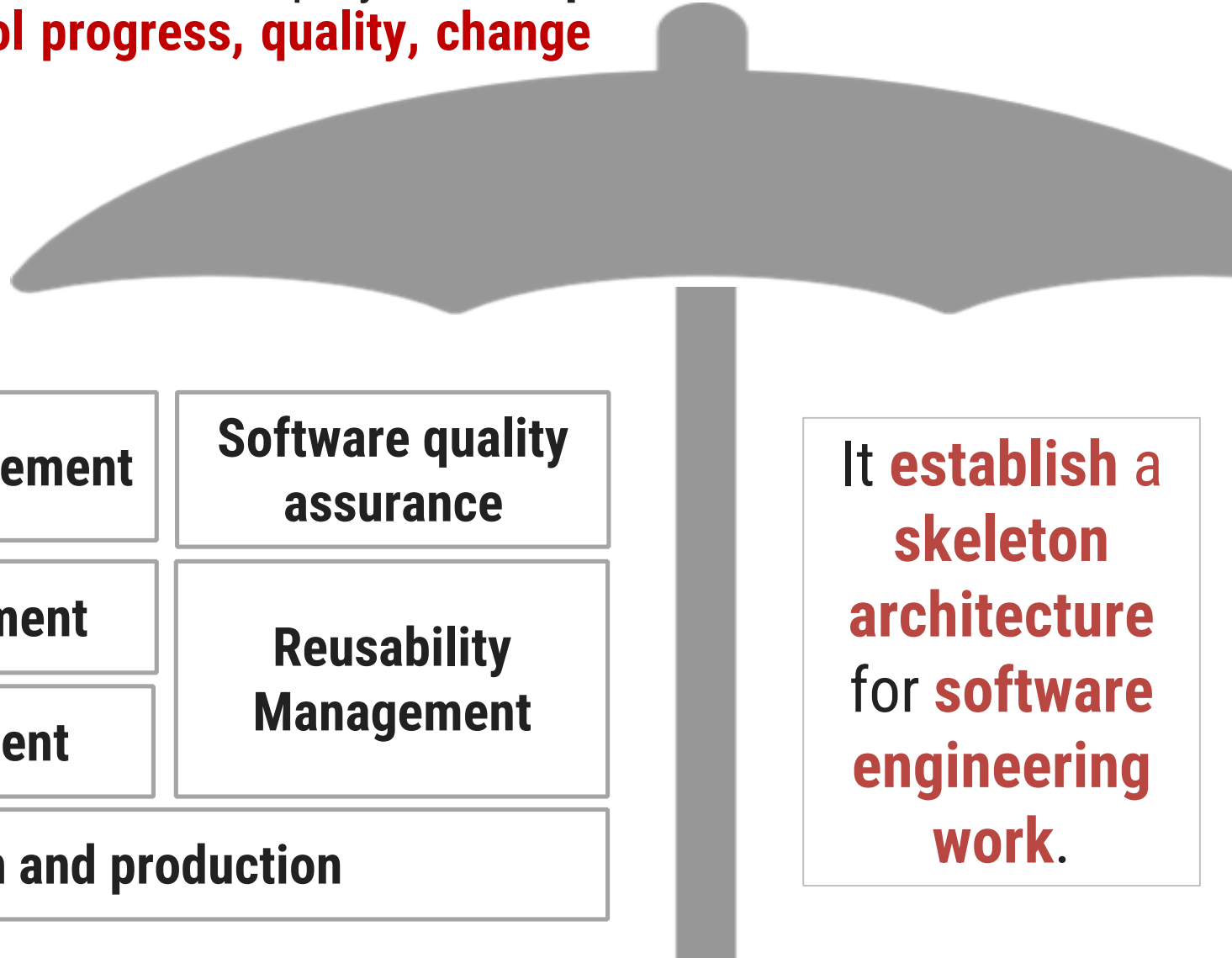


(d) Parallel process flow

Umbrella Activities

Umbrella activities applied throughout the software project & help a software team to manage and **control progress, quality, change & risks**

Umbrella activities are those which keep running in the background throughout the software development



**Software project
Tracking & Control**

Risk Management

**Software quality
assurance**

Technical Reviews

Measurement

**Reusability
Management**

Software Configuration Management

Work product preparation and production

It **establish a
skeleton
architecture
for software
engineering
work.**

Umbrella Activities Cont.

Software project tracking and control: allows the software team to **assess progress against the project plan** and take any necessary action to maintain the schedule.

Risk management: assesses (**evaluates**) **risks** that may affect the outcome of the project or the quality of the product.

Software quality assurance: defines and conducts the activities required to **ensure software quality**.

Technical reviews: **assesses** software engineering **work** products in an effort **to uncover** and remove **errors** before they are propagated to the next activity.

Measurement: defines and collects process, project and product measures that assist the team in delivering software that meets stakeholders' needs.

Software configuration management: it manages the effects of change throughout the software process.

Umbrella Activities Cont.

Reusability management: it defines criteria for work product reuse (including software components) and establishes **mechanisms to achieve reusable components**.

Work product preparation and production: it encompasses (includes) the activities required to create work products such as **models, documents, logs, forms and lists**.

7. Principles of Software Engineering

- **A software system exists for one reason: to provide value to its users.**
- **All design should be as simple as possible, but no simpler.**
- **A clear vision is essential to the success of a software project.**
- **Always specify, design, and implement knowing someone else will have to understand what you are doing.**
- **Never design yourself into a corner- Be Open to the Future.**
- **Planning ahead for reuse reduces the cost and increases the value of both the reusable components and the systems into which they are incorporated.**
- **Placing clear, complete thought before action almost always produces better results.**

Software Process Models

The **process model** is the abstract representation of process.

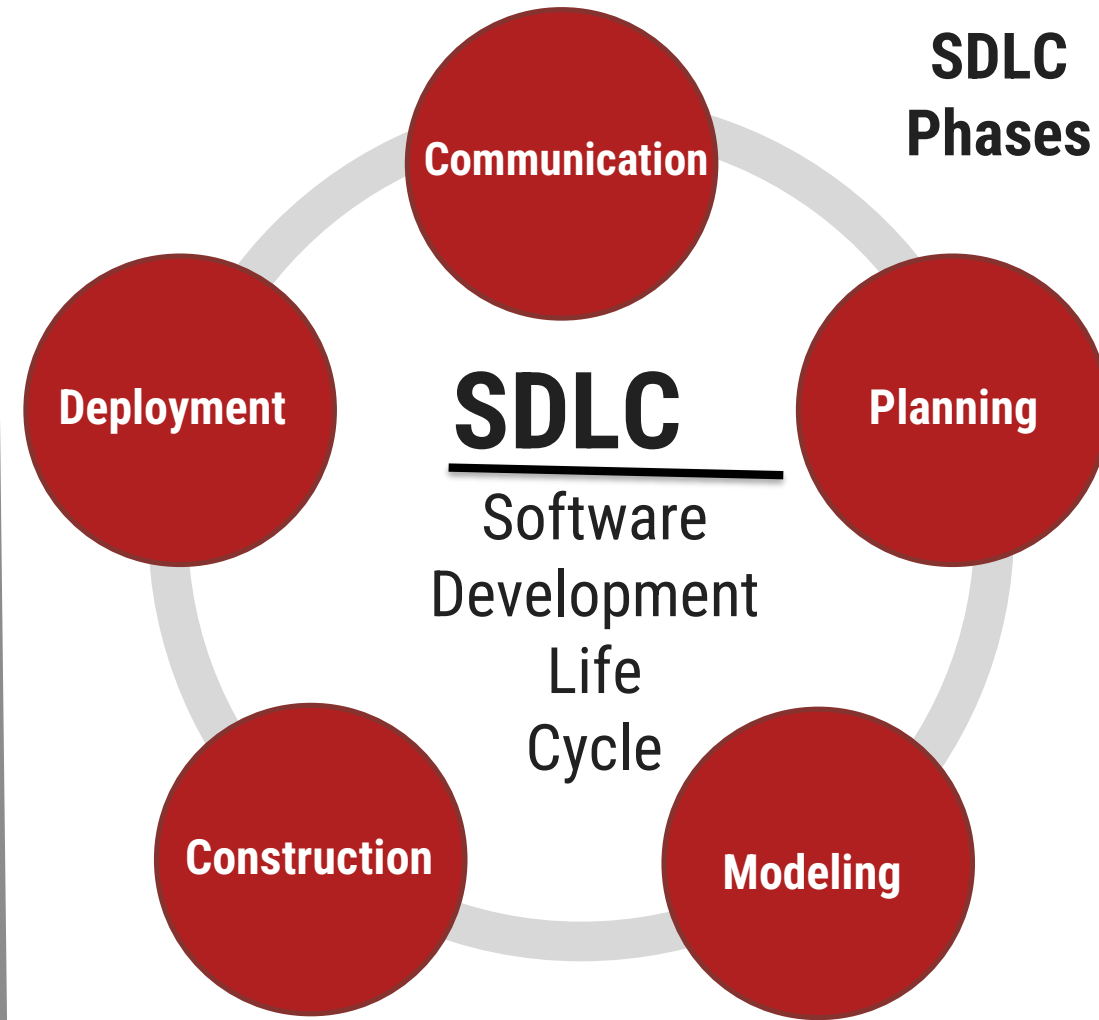
Also known as **Software development life cycle (SDLC)** or Application development life cycle Models

Process models **prescribe** a distinct set of **activities, actions, tasks and milestones (deliverables)** required to engineer high quality software.

Process **models are not perfect** but **provide roadmap** for software engineering work.

Software models provide stability, control and organization to a process that if not managed can easily get out of control.

Software process models are adapted (adjusted) to meet the needs of software engineers and managers for a specific project.



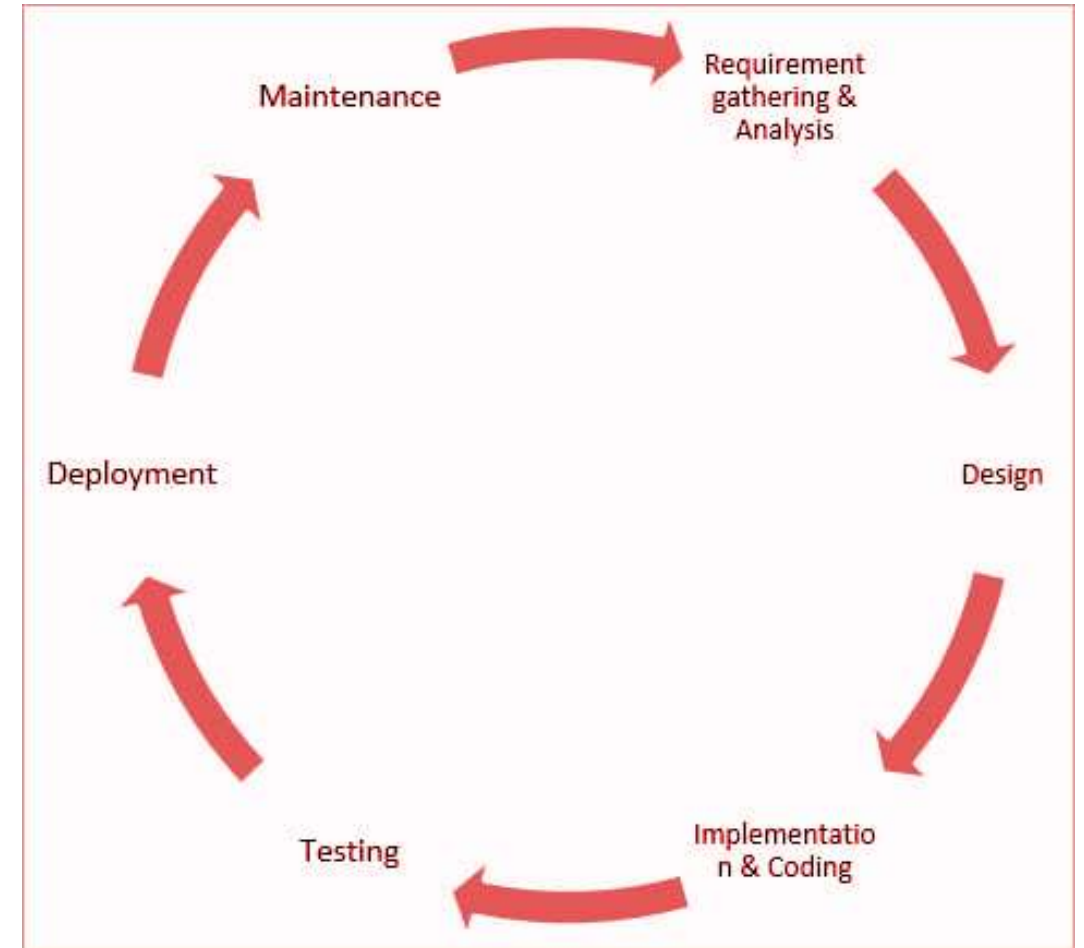
Software Development Life Cycles

A generic process framework for software engineering defines five framework activities

communication, planning, modeling, construction, and deployment.

There are many different software processes but all must include four activities that are fundamental to software engineering:

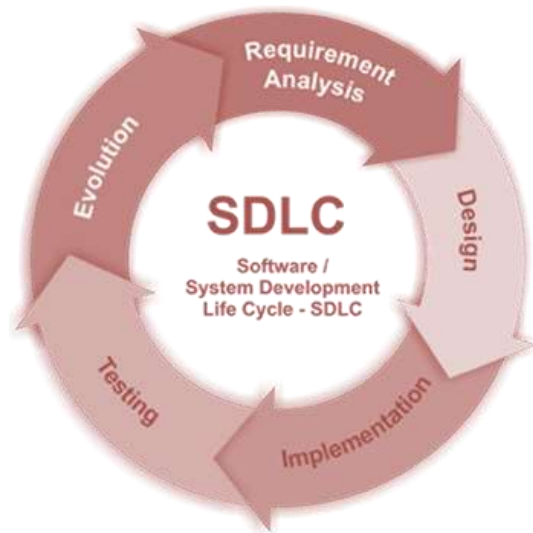
1. Software specification
2. Software design and implementation
3. Software validation
4. Software evolution



Different Process Models/ Prescriptive process models

Process model is selected based on different parameters

- ➔ Type of the project & people
- ➔ Complexity of the project
- ➔ Size of team
- ➔ Expertise of people in team
- ➔ Working environment of team
- ➔ Software delivery deadline



Process Models

Waterfall Model (Linear Sequential Model)

Incremental Process Model

Prototyping Model

The Spiral Model

Rapid Application Development Model

Agile Model

The Waterfall Model

When to use ?

Requirements are very well **known, clear** and **fixed**

Product **definition** is **stable**

Technology is **understood**

There are **no ambiguous** (unclear) **requirements**

Ample (**sufficient**) **resources** with required **expertise** are **available** freely

The **project** is **short**

Advantages

Simple to implement and manage

Drawbacks

Unable to accommodate changes at **later stages**, that is required in most of the cases.

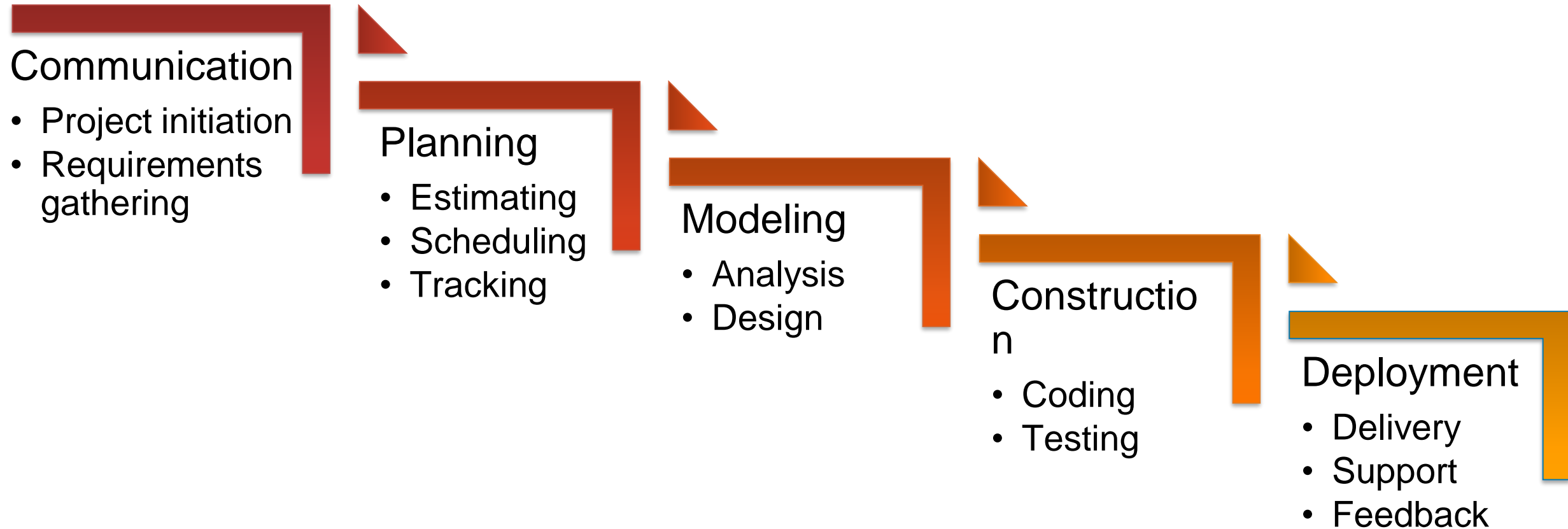
Working version is **not available** during development. Which can lead the development with major mistakes.

Deadlock can occur due to delay in any step.

Not suitable for **large projects**.

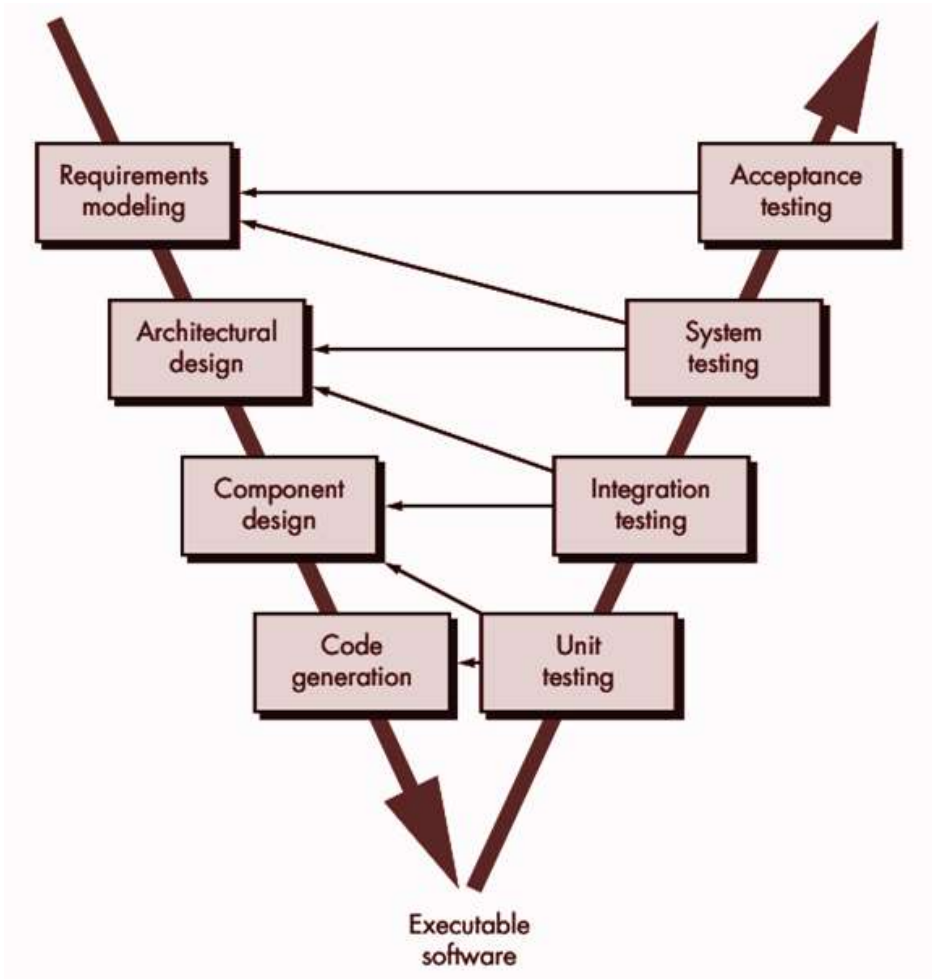
The Waterfall Model

Classic life cycle or linear sequential model



When **requirements** for a problems are **well understood** then this model is used in which **workflow** from communication to deployment is **linear**

V- Process Model



The V-model, or **Validation and Verification model**, expands on the Waterfall model with the addition of **early test planning**.

It follows a **sequential design process** same as the waterfall model.

Testing of the device is **planned in parallel** with a corresponding stage of development.

The V-shaped model should be used for **small to medium-sized projects** where requirements are clearly defined and fixed.

V- Process Model

Advantages

- Each development stage has a **corresponding testing activity**.
- This allows the team to detect errors in requirement specifications, code, and architecture **early in the development** of the project.
- The addition of early test planning gives the V-Model a **greater chance of success than that of the Waterfall model**.

Disadvantages

- **Inflexible.**
- A team can only **start the next stage when the current stage is complete**.
- Adjustments are **difficult, expensive, and time-intensive**.

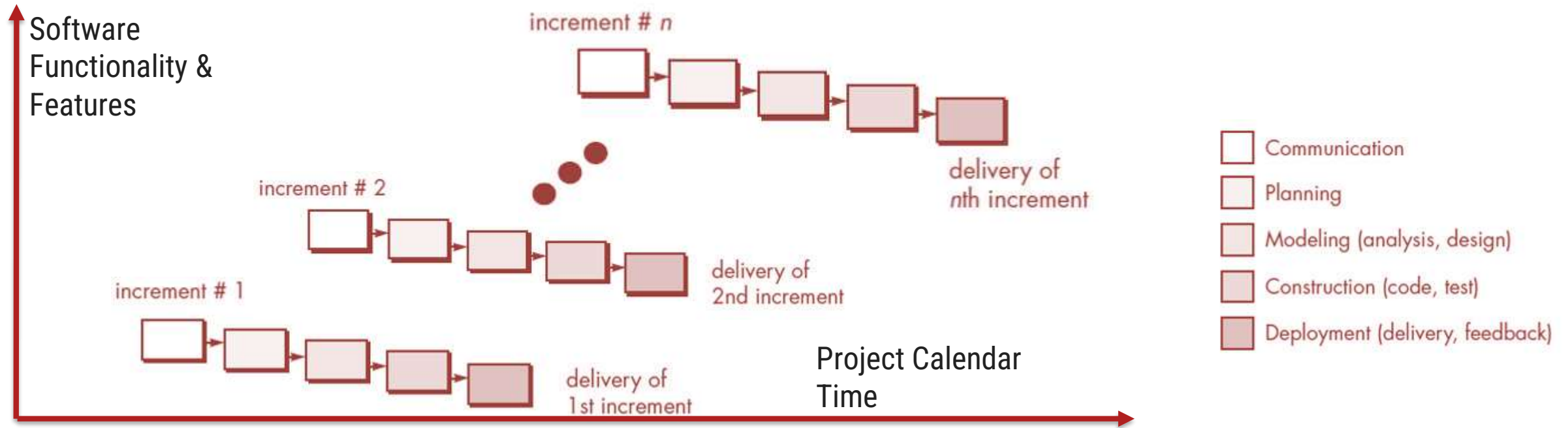
Incremental Process Model

There are many situations in which **initial software requirements** are reasonably **well defined**, but the **overall scope of the development** effort prevent a purely linear process.

In addition, there may be a **compelling need** to provide a **limited set of software functionality** to users **quickly** and then **refine and expand on that functionality** in later software releases.

In such cases, there is a need of a process model that is designed to produce the software in increments.

Incremental Process Model



The incremental model **combines** elements of **linear** and **parallel** process flows.

This model applies linear sequence in an iterative manner.

Initially **core working product** is **delivered**.

Each linear **sequence** produces deliverable **“increments”** of the software.

Incremental Process Model

e.g., **word-processing software** developed **using** the **incremental model**

It might deliver basic file management, editing and document production functions in the first increment
more sophisticated editing in the second increment;
spelling and grammar checking in the third increment;
and
advanced page layout capability in the fourth increment.

When to use ?

When the **requirements** of the **complete** system are clearly **defined** and understood but **staffing is unavailable** for a **complete implementation** by the business deadline.

Advantages

Generates **working software quickly** and early during the software life cycle.

It is **easier to test** and debug during a smaller iteration.

Customer can **respond** to each built.

Lowers initial delivery **cost**.

Easier to **manage risk** because risky pieces are identified and handled during iteration.

Prototyping model

When to use ?

Customers have general **objectives of software** but **do not have detailed requirements** for functions & features.

Developers are **not sure** about **efficiency of an algorithm & technical feasibilities**.

It serves as a **mechanism** for **identifying software requirements**.

Prototype can be served as “**the first system**”.

“**quick and dirty**” – quality not important, scripting etc. can be used

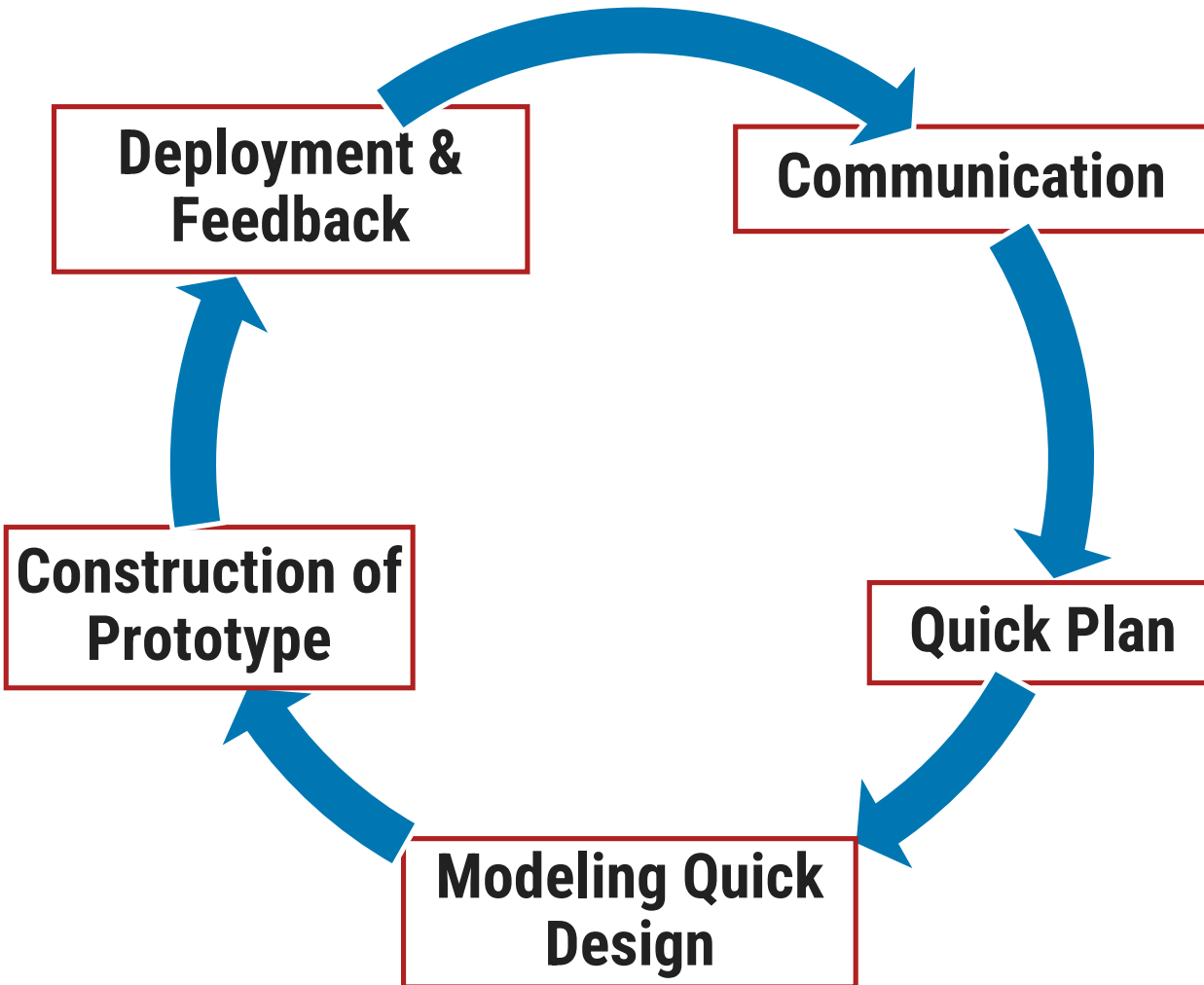
Things like exception handling, recovery, standards are **omitted**

Both stakeholders and software engineers like prototyping model

- ➔ Users get feel for the actual system
- ➔ Developers get to build something immediately

Prototyping model cont.

It works as follow



Communicate with stockholders & **define objective** of Software

Identify requirements & design **quick plan**

Model a quick **design** (focuses on visible part of software)

Construct Prototype & deploy

Stakeholders **evaluate** this **prototype** and provides **feedback**

Iteration occurs and **prototype** is **tuned** based on **feedback**

Prototyping model cont.

Problem Areas

Potential hit on **cost** and **schedule**

Customer demand that “**a few fixes**” be applied to **make** the **prototype a working product**, due to that software quality suffers as a result

Developer often makes **implementation** in order to get a prototype working quickly; **without considering other factors** in mind like OS, Programming language, etc.

Potential false sense of security if prototype does not focus on key (**high risk**) issues

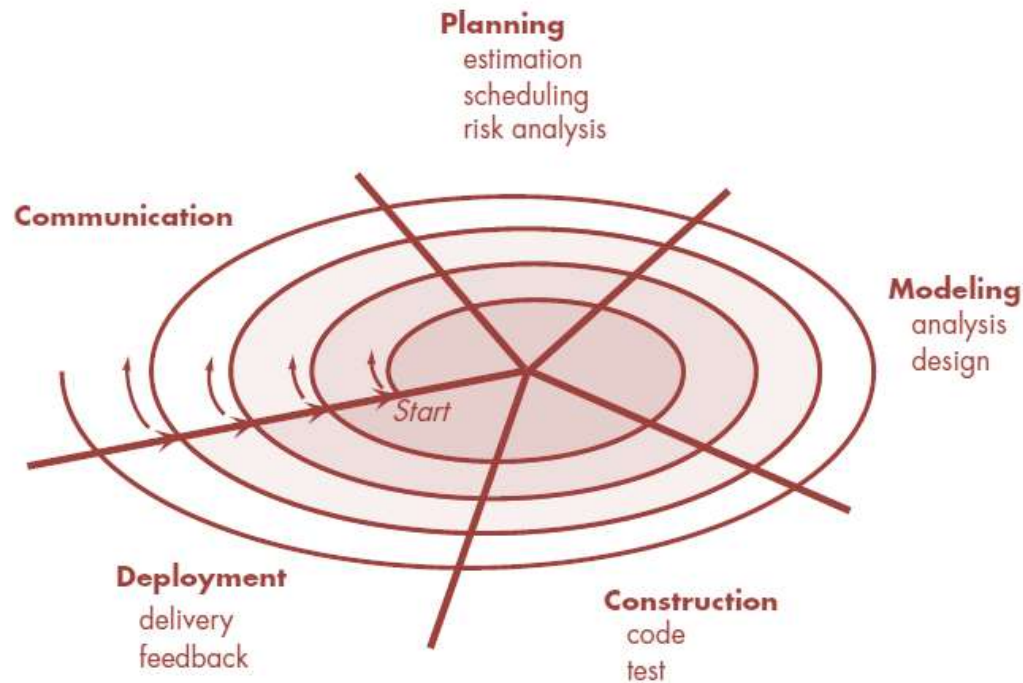
Advantages

Users are actively **involved** in the **development**

Since in this methodology a working model of the system is provided, the **users get a better understanding** of the **system** being developed

Errors can be **detected** much **earlier**

The Spiral Model



It provides the **potential** for **rapid development**.

Software is developed in a series of evolutionary releases.

Early iteration release might be **prototype** but **later iterations** provides more **complete version of software**.

It is divided into framework activities (C,P,M,C,D). Each activity represent one segment of the spiral

Each pass through the **planning** region results in **adjustments** to

- the **project plan**
- **Cost & schedule** based on feedback

The Spiral Model Cont.

When to use Spiral Model?

For development of **large scale / high-risk projects**.

When costs and **risk evaluation is important**.

Users are **unsure** of their **needs**.

Requirements are **complex**.

New product line.

Significant **(considerable)** **changes** are expected.

Advantages

High amount of risk analysis hence, **avoidance of Risk** is enhanced.

Strong approval and **documentation** control.

Additional functionality can be **added** later.

Software is **produced early** in the Software Life Cycle.

Disadvantages

Can be **a costly model** to use.

Risk analysis **requires highly specific expertise**.

Project's success is highly dependent on the risk analysis phase.

Doesn't work well for smaller projects.

Rapid Application Development Model (RAD)

It is a type of **incremental model** in which; **components** or functions are **developed in parallel**.

Rapid development is **achieved** by **component based construction**

This can **quickly give** the customer **something to see** and use and to provide feedback.

Communication

Planning

Team - 1

Modeling

Construction

Team - 2

Modeling

Construction

Team - 3

Modeling

Construction

- Integration
- Delivery
- Feedback

Deployment

- Component Reuse
- Automatic Code Generation
- Testing

- Business Modeling
- Data Modeling
- Process Modeling

Rapid Application Development Model (RAD) Cont.

Communication

This phase is used to understand business problem.

Planning

Multiple software teams work in parallel on different systems/modules.

Modeling

Business Modeling: *Information flow* among the business.

- Ex. What kind of information drives (moves)?
- Who is going to generate information?
- From where information comes and goes?

Data Modeling: Information refine into set of *data objects* that are *needed* to support business.

Process Modeling: *Data object* transforms to *information flow* necessary to implement business.

Construction

It highlighting the *use of pre-existing software component*.

Deployment

Integration of modules developed by parallel teams, **delivery** of integrated software and **feedback** comes under deployment phase.

Rapid Application Development Model (RAD) Cont.

When to Use?

There is a need to create a **system** that can be **modularized in 2-3 months** of time.

High availability of **designers** and **budget** for modeling along with the cost of automated code generating tools.

Resources with **high** business **knowledge** are available.

Advantages

Reduced development **time**.

Increases reusability of components.

Quick initial **reviews** occur.

Encourages customer **feedback**.

Integration from very beginning **solves** a lot of **integration issues**.

Drawback

For **large** but scalable **projects**, RAD **requires sufficient human resources**.

Projects **fail if developers** and **customers** are **not committed** in a much-shortened time-frame.

Problematic if system **can not be modularized**.

Not appropriate when technical risks are high (heavy use of new technology).

Component based Development

- ▶ **Commercial off the shelf (COTS)** software **components** are offered **as product**.
- ▶ **COTS** provides **set of functionality** with **well defined interfaces** that enables component to be integrated into software.
- ▶ The component based development model **incorporates** many **characteristics** of the **spiral model**.
- ▶ It is **evolutionary** in **nature**.
- ▶ Component based development model **constructs** applications from **prepackaged** software **components**.
- ▶ **Modeling** and **construction** activities begin with the **identification of components**.

Component based Development

Component based development incorporates the following steps

1. Available **component-based products** are **researched** & **evaluated** for software development.
2. Component **integration issues** are **considered**.
3. A **software architecture** is **designed** to accommodate the components.
4. Components are **integrated** into the **architecture**.
5. **Testing** is conducted to insure proper functionality.

Advantages

- ▶ It leads to **software reuse**.
- ▶ It **reduces development** cycle **time**.
- ▶ **Reduction** in project **cost**.

Typical Effort Distribution

Req. - ?

Design - ?

Coding - ?

Testing - ?

Req. - 15-20%

Design - 25-30%

Coding - 25-30%

Testing - 20-30%

Coding is not **the most** expensive!

How programmers spend their time?

Writing programs - ?

Reading programs and manuals - ?

Job communication - ?

Others - ?

Writing programs - 13%

Reading programs and manuals - 16%

Job communication - 32%

Others - 39%

- Programmers spend **more time in reading programs** than in writing them.
- Writing programs is a small part of their lives.

When are defects introduced?

Requirement - ?

Design - ?

Coding - ?

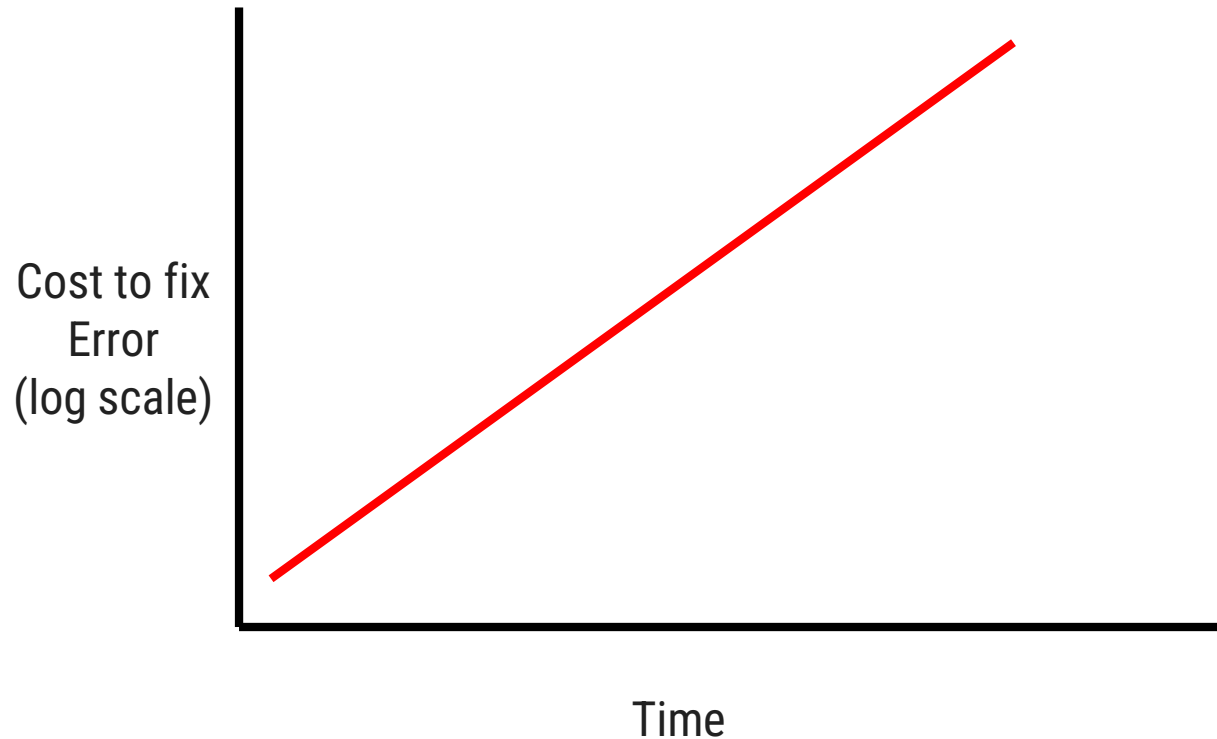
Requirement - 20%

Design - 30%

Coding - 50%

- Defects can be injected at any of the major phases.
- **Cost of latency:** Cost of defect removal increases exponentially with latency time.

When are defects introduced?



- Cheapest way to detect and remove defects close to where it is injected.
- Hence must check for defects after every phase.

Product & Process

If the **process is weak**, the end **product** will **suffer**. But **more confidence** on **process** is also **dangerous**.

People **gain more satisfaction** from the **creative process** as they do from the end product.

- Like an artist enjoys the brush strokes as much as the framed result.
- A writer enjoys the search for the proper metaphor (comparison) as much as the finished book.

As **software professional**, you should also **derive** as much **satisfaction** from the **process** as the end product.

The duality (contrast) of product and process is one important element in keeping creative people engaged as software engineering continues to evolve.

Thank You!