Chapter 1

- **Software & Software Engineering**

SE-IT

Ms. Sakhare Yashoda N.
VPKBIET , Baramati

Unit I

Introduction To Software Engineering

06 hrs

- **Software Engineering Fundamentals**: Nature of Software, Software Engineering Practice, Software Process, Software Myths.
- **Process Models** : A Generic Process Model, Linear Sequential Development Model, Iterative Development Model, The incremental Development Model
- **Agile software development:** Agile manifesto, agility principles, Agile methods, myth of planned development, Introduction to Extreme programming and Scrum.
- **Agile Practices:** test driven development, pair programming, continuous integration in DevOps , Refactoring

WHAT IS SOFTWARE?

Software is:

(1) instructions (computer programs) that when executed provide desired features, function, and performance;

(2) data structures that enable the programs to adequately manipulate information and

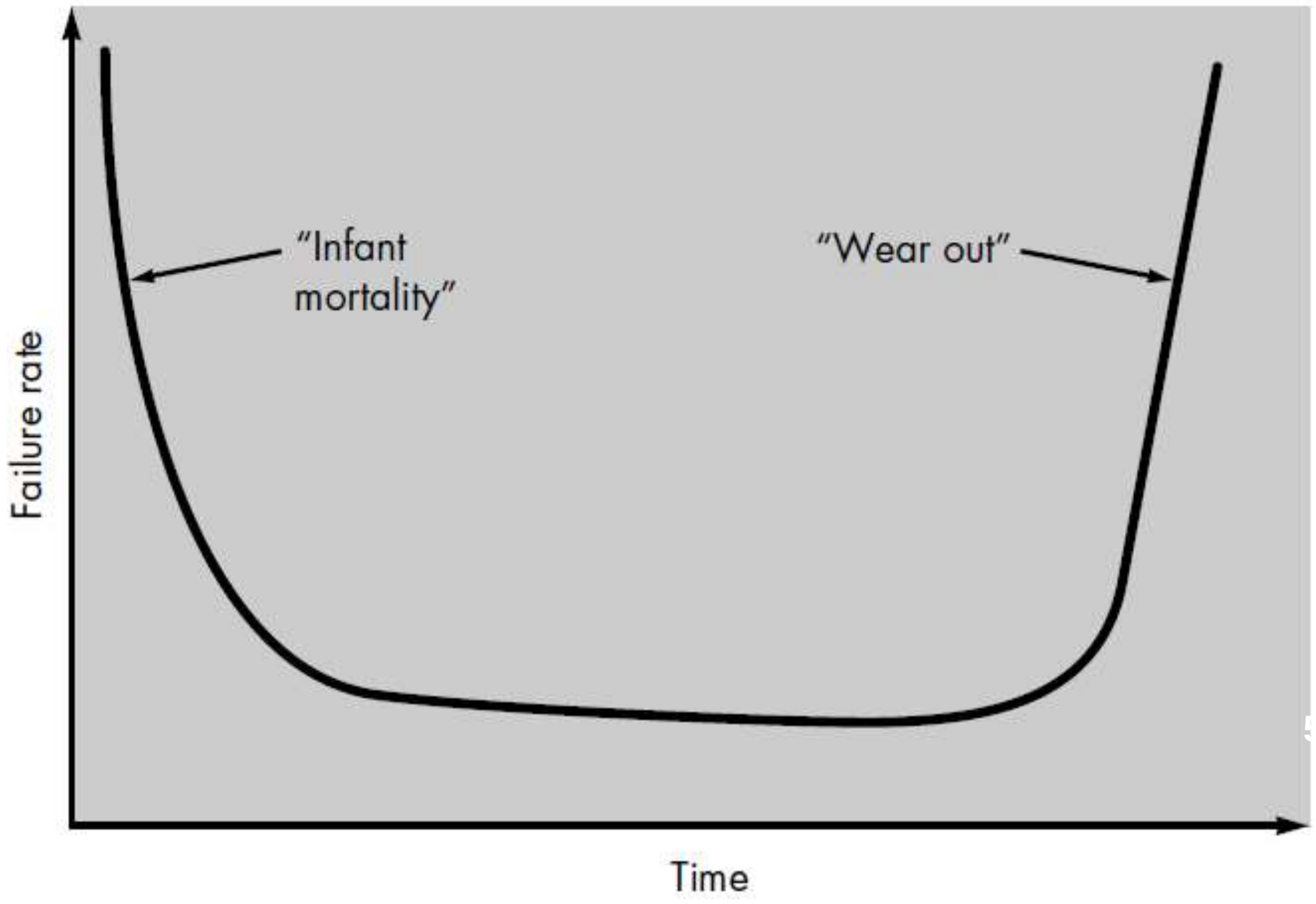
(3) documentation that describes the operation and use of the programs.

Software characteristics

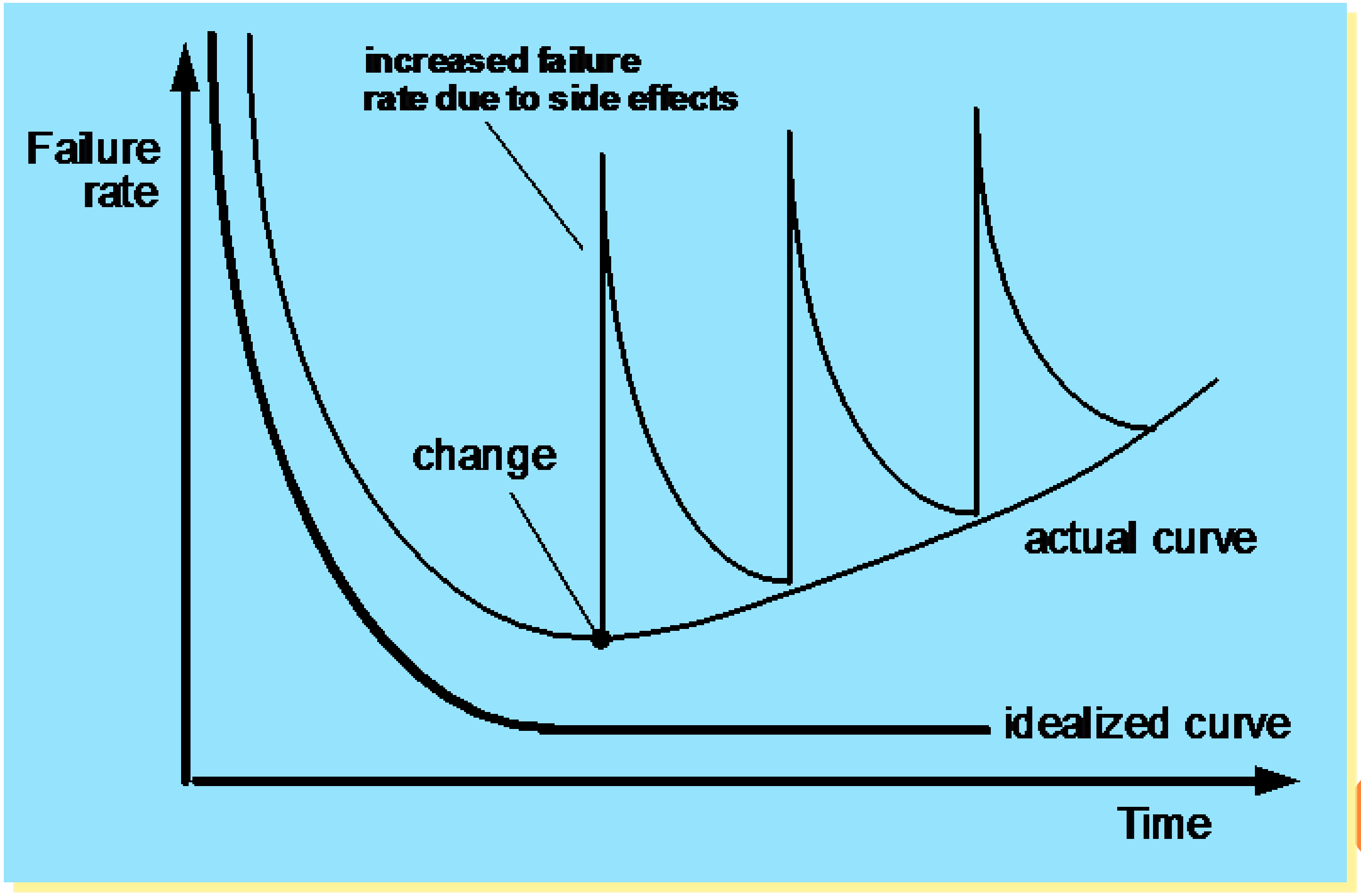
- ***Software is developed or engineered, it is not manufactured in the classical sense.***
- ***Software doesn't "wear out."***
- ***Although the industry is moving toward component-based construction, most software continues to be custom-built.***

Bath Tub Curve : Failure Curve for Hardware

FIGURE 1.1
Failure curve for hardware



Wear vs. Deterioration



Nature of a software

- Duel role of software
  - As a Product
  - As a vehicle (deliver Information)
- Delivery products of software
- Changes in Computer software
- Software industry as a dominant factor
- List of Questionnaires arises at the time of building computer-based software's
  - Software finished time
  - Development cost
  - Error detection and correction before delivery
  - Maintaining existing programs
  - Difficulty in measuring progress

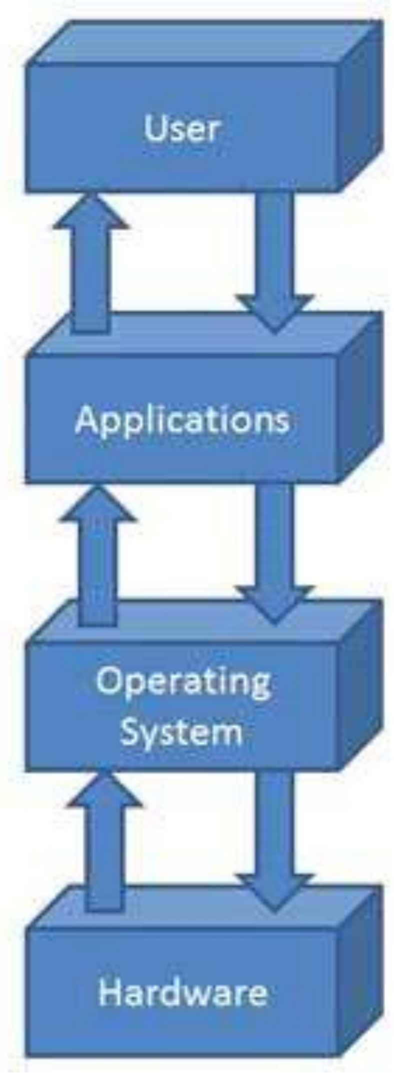
Software Application Domains

- System software
- Application software
- Engineering/scientific software
- Embedded software
- Product-line software
- WebApps (Web applications)
- Artificial Intelligence software

System Software Application Software

System Software	Application Software
The system software is collection of programs designed to operate, control, and extend the processing capabilities of the computer itself	Application software products are designed to satisfy a particular need of a particular environment.
System software are generally prepared by computer manufactures.	All software applications prepared in the computer lab can come under the category of Application software
These software products comprise of programs written in low-level languages which interact with the hardware at a very basic level.	Application software may consist of a single program, such as a Microsoft's notepad for writing and editing simple text. It may also consist of a collection of programs, often called a software package, which work together to accomplish a task, such as a spreadsheet package
Eg: Operating System, Compilers, Interpreter, Assemblers etc.	Eg: Microsoft Word , Microsoft Excel, Microsoft Powerpoint

Operating System



Scientific /Engineering software

- **scientific software** broadly as **software** used for **scientific** purposes.
- **Scientific software** is mainly developed to better understand or make predictions about real world processes.
- To develop **scientific software**, **scientists** first develop discretized models
-

Scientific and engineering software

- It satisfies the needs of **scientific** or **engineering** user to perform enterprise specific tasks.
- These **software** is written for specific applications using principles, techniques and formulae specific to that field.
- Examples are **software** like
  - MATLAB, AUTOCAD,ORCAD,

Basics of Embedded System

- An embedded system is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints.

- It is embedded as part of a complete device often including hardware and mechanical parts. Embedded systems control many devices in common use today.

- They are used in transportation, fire safety, safety and security, medical applications.

Embedded software

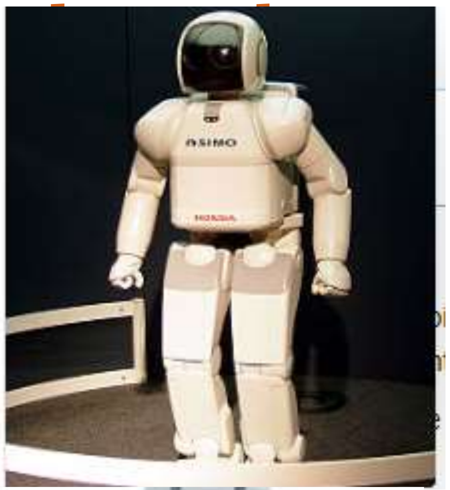
- It is computer **software**, written to control machines or devices that are not typically thought of as computers, commonly known as **embedded** systems.
- It is typically specialized for the particular hardware that it runs on and has time and memory constraints.
- **EX**-Anti-lock braking systems found in automobiles,
- Image processing systems found in medical imaging equipment,
- Fly-by-wire control systems found in aircraft,
- Motion detection systems in security cameras.
- Traffic control systems found in traffic lights.

EMBEDDED SOFTWARE

- "Embedded software" is specialized programming within non-PC devices – either as part of a microchip or as part of another application that sits on top of the chip – to control specific functions of the device.
- Embedded software has fixed hardware requirements and capabilities.
- Processing and memory restrictions.

- Manufacturers build embedded software into the electronics of cars, telephones, modems, robots, appliances, toys, security systems, televisions and set-top boxes,

A **robot** is a machine—especially one programmable by a computer— capable of carrying out a complex series of actions automatically. Robots can be guided by an external control device or the control may be embedded within. Robots may be constructed on the



Car

A **car** is a wheeled motor vehicle used for transportation. Most definitions of cars say that they run primarily on roads, seat one to eight people, have four wheels, and mainly transport people rather than goods.

A **watch** is a portable timepiece intended to be carried or worn by a person. It is designed to keep a consistent movement despite the motions caused by the person's activities. A **wristwatch** is designed to be worn around the wrist, attached by a watch strap or other ty

PRODUCT LINES SOFTWARE(SPLS),

- It refers to software engineering methods, tools and
- techniques

for creating a collection of similar software systems from a shared set of software assets using a common means of production.

Benefits of SPLs

- Improved productivity
- Increased quality
- Decreased cost
- Decreased labor needs
- Decreased time to market
- Ability to move into new markets in months, not years

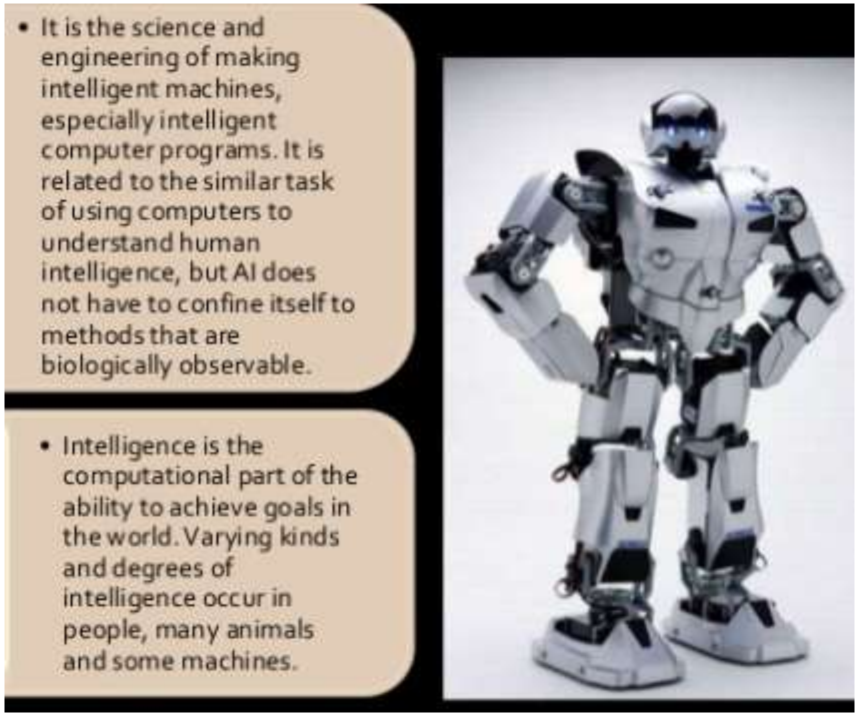
WEB APPLICATION(WEB APP)

- It is application software that runs on a web server, unlike computer-based software programs that are run locally on the operating system (OS) of the device.
- Web applications are accessed by the user through a web browser with an active internet connection.
- These applications are programmed using a client–server modeled structure—the user ("*client*") is provided *services* through an *off-site server* that is hosted by a third-party.
- Examples : web-mail, online retail sales, online banking, and online auctions.

Web applications

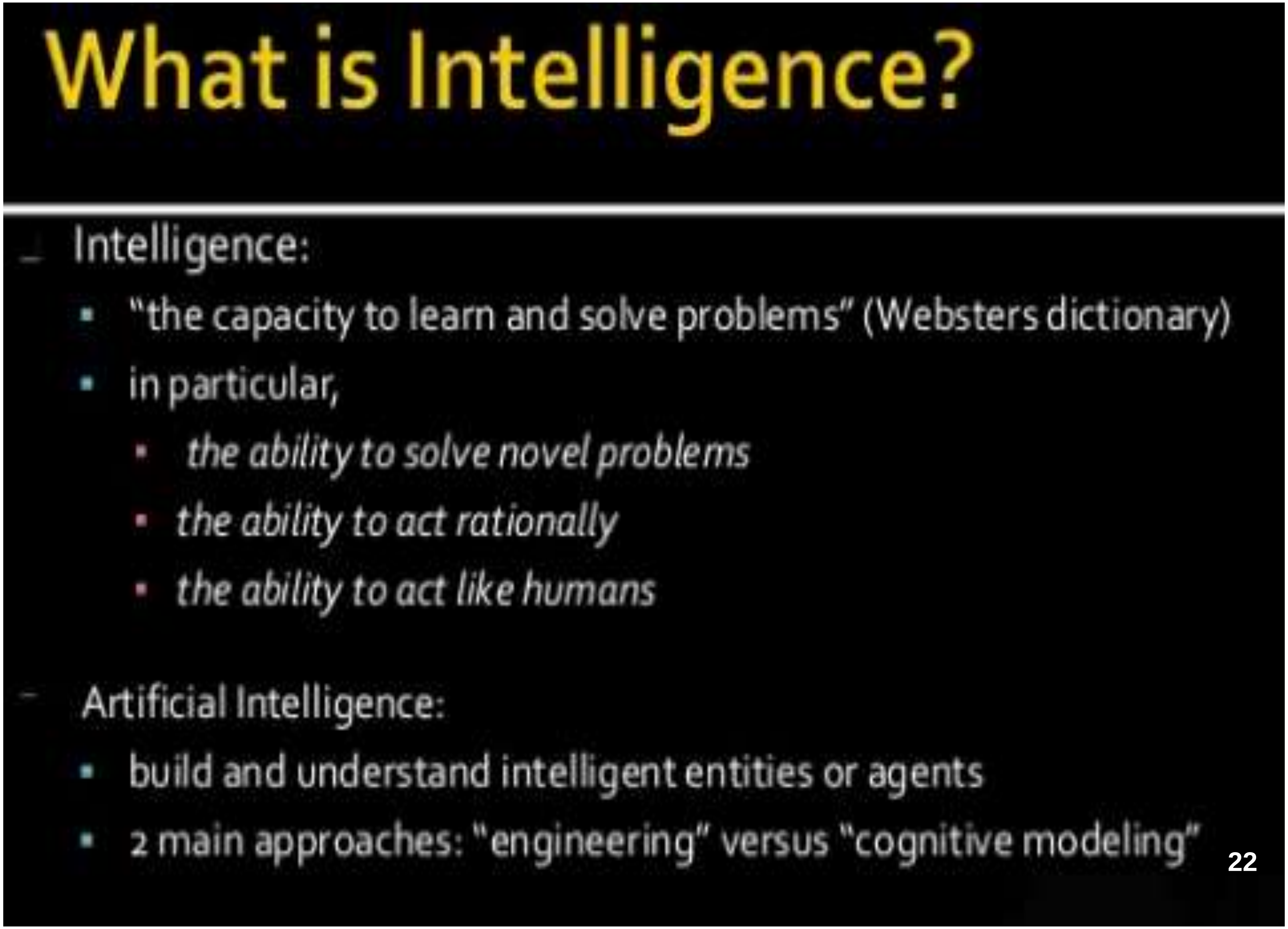
- It include online forms,
- shopping carts,
- word processors,
- spreadsheets,
- video and photo editing,
- file conversion,
- file scanning, and
- email programs such as Gmail, Yahoo and AOL.
- Popular **applications** include Google **Apps** and Microsoft 365

ARTIFICIAL INTELLIGENCE



- It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable.

- Intelligence is the computational part of the ability to achieve goals in the world. Varying kinds and degrees of intelligence occur in people, many animals and some machines.



What is Intelligence?

- Intelligence:
  - "the capacity to learn and solve problems" (Websters dictionary)
  - in particular,
    - *the ability to solve novel problems*
    - *the ability to act rationally*
    - *the ability to act like humans*
- Artificial Intelligence:
  - build and understand intelligent entities or agents
  - 2 main approaches: "engineering" versus "cognitive modeling"

Artificial intelligence Software

- “**Software** that is capable of **intelligent** behavior.”
- In creating **intelligent software**, this involves simulating a number of capabilities, including
- Reasoning,
- learning,
- problem solving,
- perception,
- knowledge representation.

EXAMPLES

- **GOOGLE ASSISTANT**
- **MASDIMA**
- **CORTANA**
- **SISENSE**
- **H2O.AI**

Software—New Categories

- Open world computing—pervasive, distributed computing
- Ubiquitous computing—wireless networks
- Net sourcing—the Web as a computing engine
- Open source—”free” source code open to the computing community (a blessing, but also a potential curse!)
- Legacy Software

Legacy Software

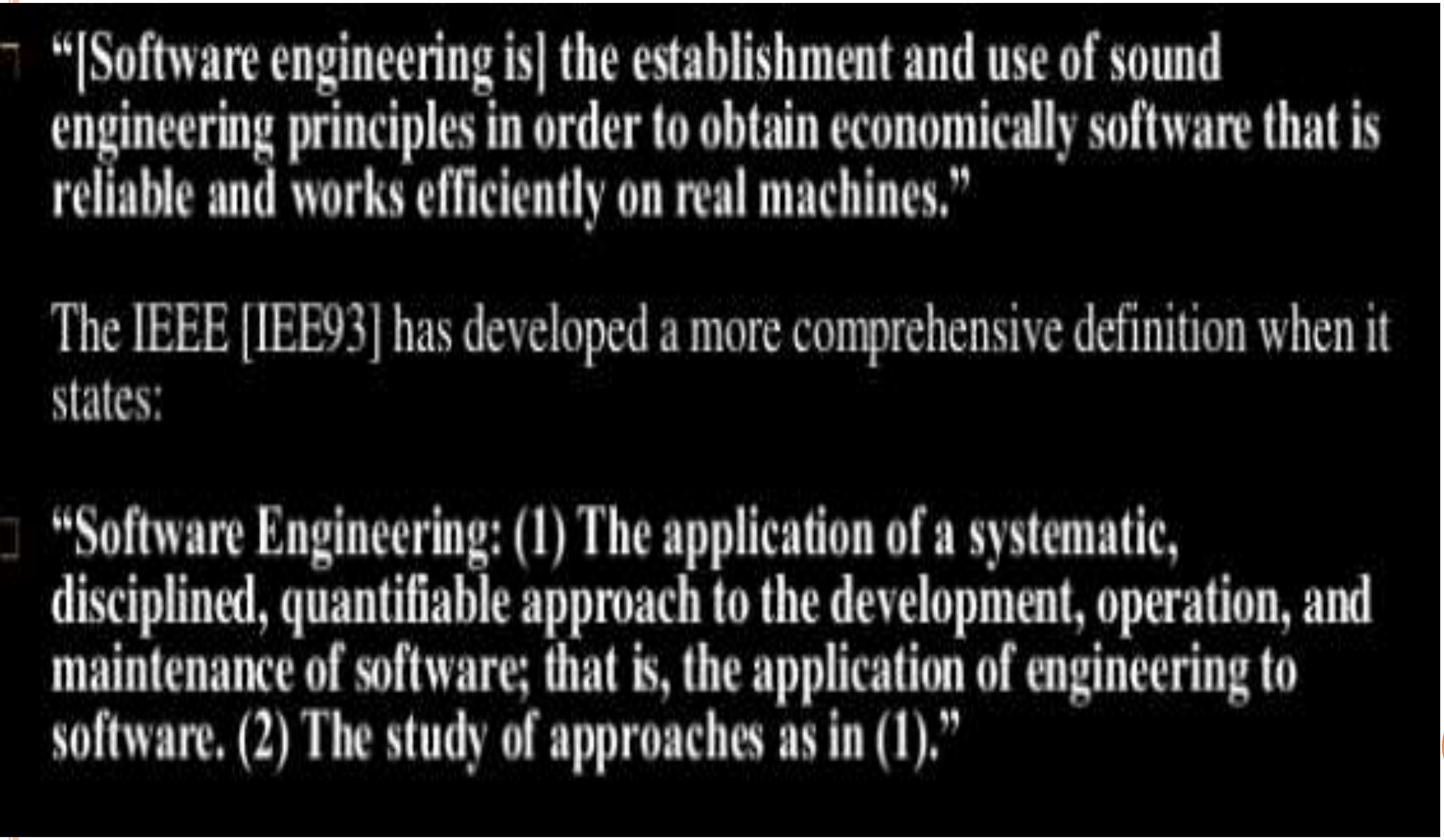
Why must it change?

- software must be adapted
- software must be enhanced.
- software must be extended to make it interoperable.
- software must be re-architected.

Characteristics of WebApps

- Network intensiveness
- Concurrency
- Unpredictable load
- Performance
- Availability
- Data driven
- Content sensitive
- Continuous evolution..
- Immediacy.
- Security.
- Aesthetics.

Software Engineering



“[Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.”

The IEEE [IEE93] has developed a more comprehensive definition when it states:

“Software Engineering: (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1).”

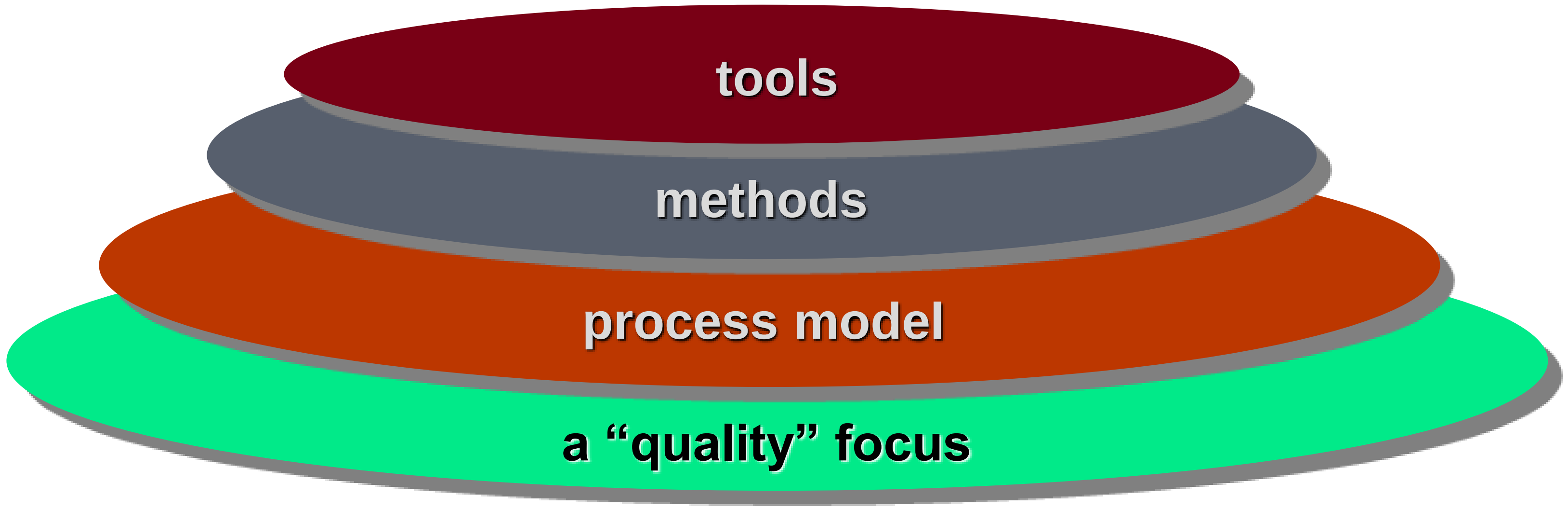
Software Engineering

- Some realities:
  - *a concerted effort should be made to understand the problem before a software solution is developed*
  - *design becomes a pivotal activity*
  - *software should exhibit high quality*
  - *software should be maintainable*
- The seminal definition:
  - *[Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.*

Software Engineering

- The IEEE definition:
  - *Software Engineering is :*
  - *(1) The application of a*
    - *systematic,*
    - *disciplined,*
    - *quantifiable approach(sdq) to the*
    - *development,*
    - *operation, and*
    - *Maintenance(DOM)of software; that is, the application of engineering to software.*
  - *(2) The study of approaches as in (1).*

A Layered Technology



Software Engineering

The Software Process

- A *process* is a collection of activities, actions, and tasks that are performed when some work product is to be created.
- An *activity* strives to achieve a broad objective (e.g., communication with stakeholders) and is applied regardless of the application domain, size of the project, complexity of the effort, or degree of rigor with which software engineering is to be applied.
- An *action* (e.g., architectural design) encompasses a set of tasks that produce a major work product (e.g., an architectural design model).
- A task focuses on a small, but well-defined objective (e.g., conducting a unit test) that produces a tangible outcome.

Framework Activities

- Communication
- Planning
- Modeling
  - Analysis of requirements
  - Design
- Construction
  - Code generation
  - Testing
- Deployment(CPMCD)

Umbrella Activities

- Software project management
- Formal technical reviews
- Software quality assurance
- Software configuration management
- Work product preparation and production
- Reusability management
- Measurement
- Risk management

Adapting a Process Model

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

Software Engineering Practice

- **The Essence of Practice**
- Polya suggests:

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

UNDERSTAND THE PROBLEM

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

Plan the Solution

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

Carry Out the Plan

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

Examine the Result

- *Is it possible to test each component part of the solution?* Has a reasonable testing strategy been implemented?
- *Does the solution produce results that conform to the data, functions, and features that are required?* Has the software been validated against all stakeholder requirements?

David Hooker's General Principles [Laws/Assumption]

1: ***The Reason It All Exists***

-To provide value to its users

2: ***KISS (Keep It Simple, Stupid!)***

-All design should be as simple as possible , but no simpler(must cover all requirements)

3: ***Maintain the Vision(Architectural vision)***

-Clear vision is essential to success of a software project

David Hooker's General Principles [Laws/Assumption]

4: ***What You Produce, Others Will Consume***

Specify ,design and implement knowing someone else will have to understand what you are doing.

5: ***Be Open to the Future***

Never design yourself into a corner

6: ***Plan Ahead for Reuse***

- *Planning ahead for reuse reduces the cost and increases the value of both the reusable components and the systems into which they are incorporated.*

.

7: ***Think! -*** *Placing clear ,complete thought before action almost always produces better results*

Software Myths

- Management myths
- Customer Myths
- Practitioner’s Myths

Management myths

- **Myth-We already have a book that's full of standards and procedures for building software,** won't that provide my people with everything they need to know?
- ***Myth: My people have state-of-the-art software development tools, after all, we*** buy them the newest computers.
- ***Myth: If I decide to outsource3 the software project to a third party, I can just relax*** and let that firm build it.

Customer Myths

- ***Myth: A general statement of objectives is sufficient to begin writing programs***—
- we can fill in the details later.
- ***Myth: Project requirements continually change, but change can be easily accommodated*** because software is flexible

Practitioner's Myths

- ***Myth:*** *Once we write the program and get it to work, our job is done.*
- ***Myth: Until I get the program "running" I have no*** *way of assessing its quality.*
- ***Myth: The only deliverable work product for a*** *successful project is the working* Program

Software Myths

- Affect managers, customers (and other non-technical stakeholders) and practitioners
- Are believable because they often have elements of truth,

but …

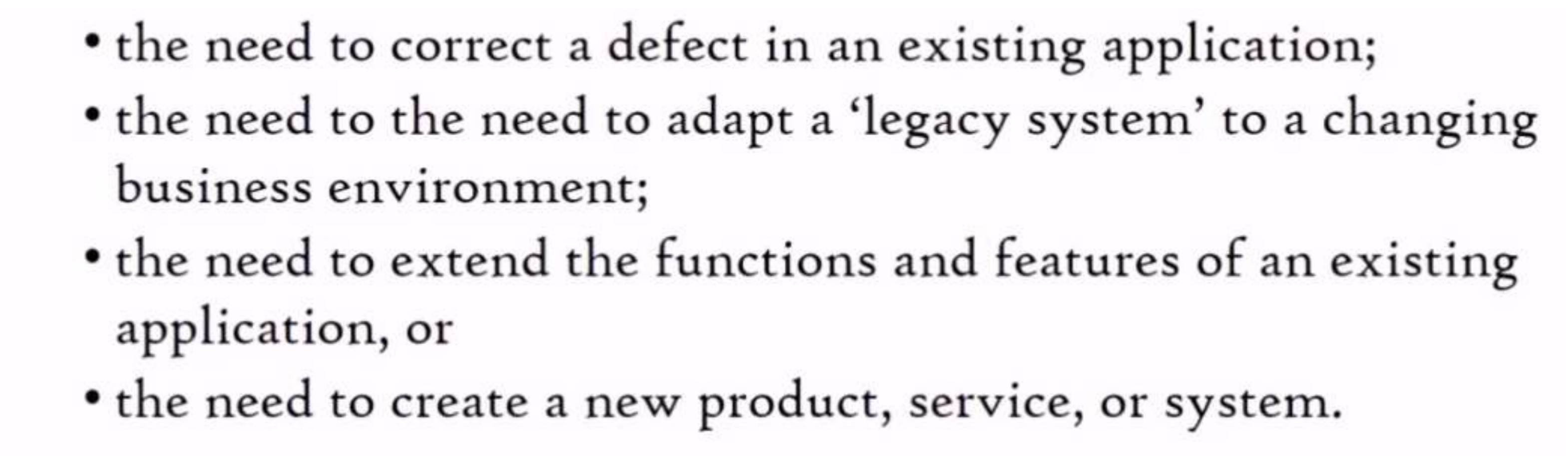
- Invariably lead to bad decisions,

therefore …

- Insist on reality as you navigate your way through software engineering

How it all should Start

Business needs



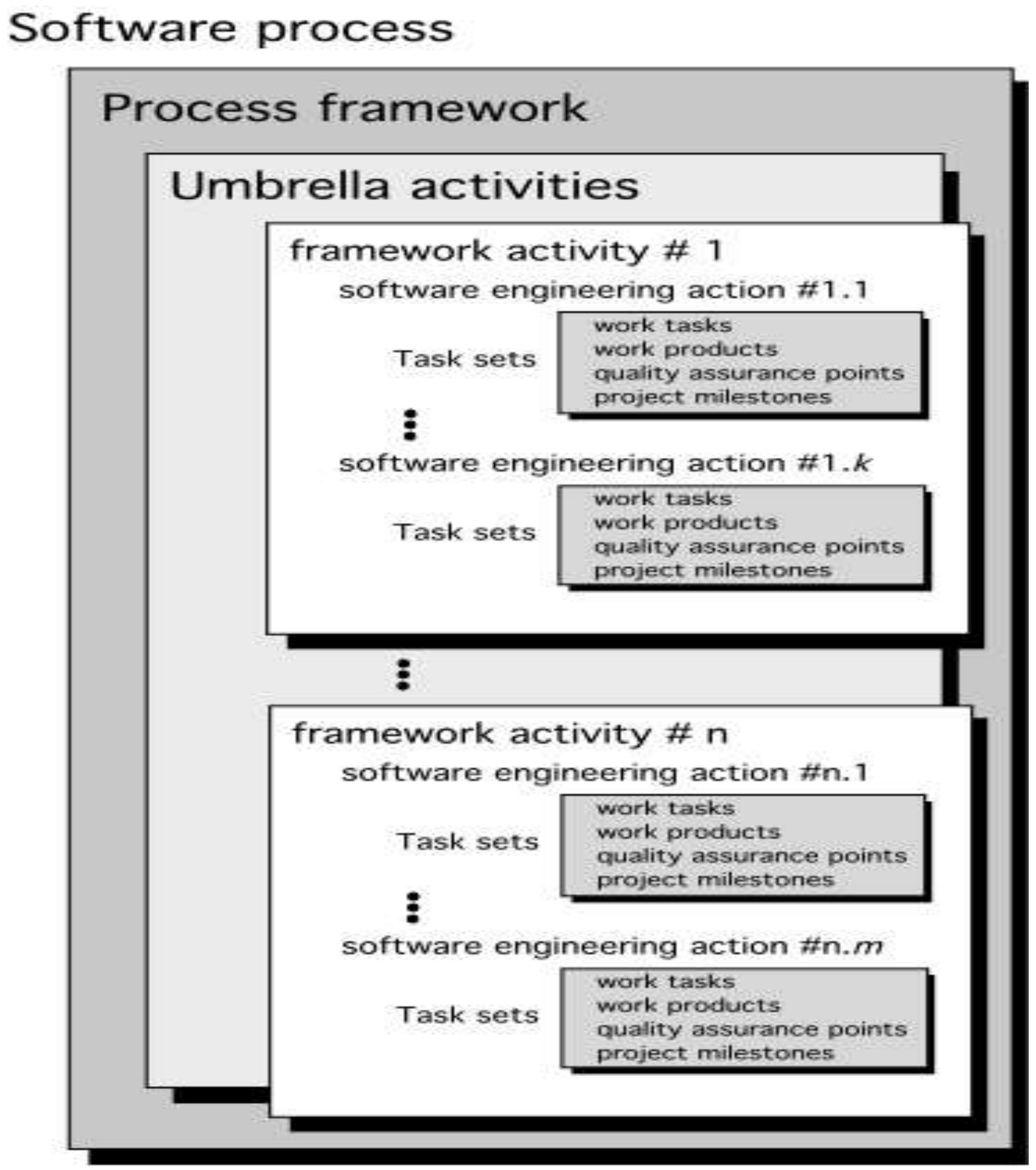
- the need to correct a defect in an existing application;
- the need to the need to adapt a 'legacy system' to a changing business environment;
- the need to extend the functions and features of an existing application, or
- the need to create a new product, service, or system.

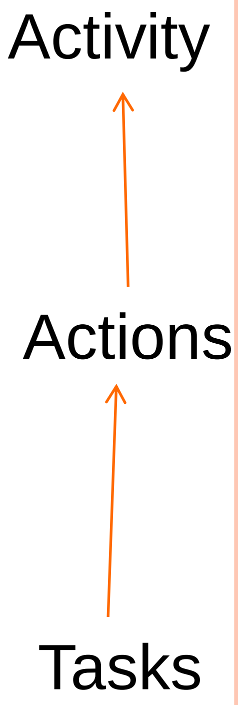
- A Generic Process Model,
- Linear Sequential Development Model,
- Iterative Development Model,
- The incremental Development Model

Framework Activities

- Communication
- Planning
- Modeling
  - Analysis of requirements
  - Design
- Construction
  - Code generation
  - Testing
- Deployment(CPMCD)

MODEL(FRAMEWORK)

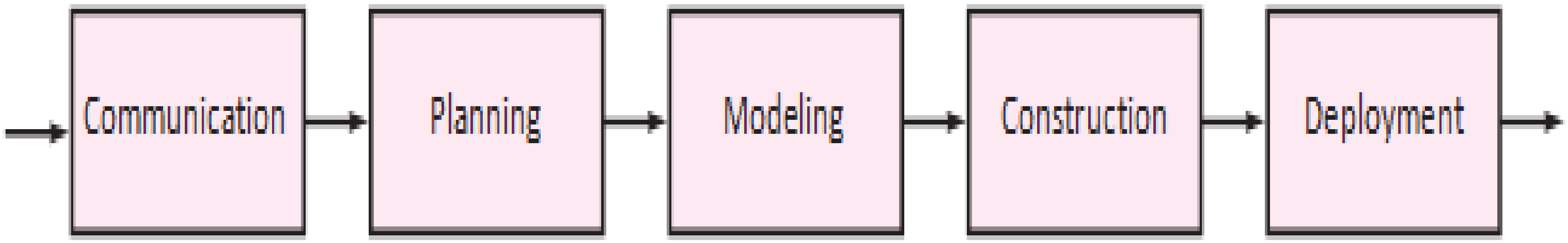




Umbrella Activities

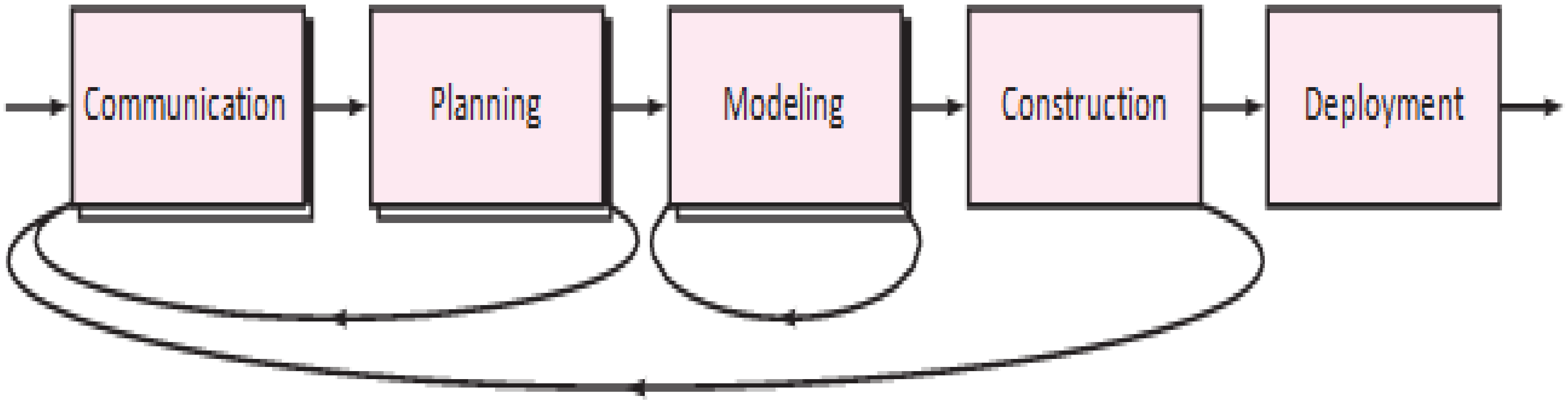
- Software project management
- Formal technical reviews
- Software quality assurance
- Software configuration management
- Work product preparation and production
- Reusability management
- Measurement
- Risk management

LINEAR PROCESS FLOW

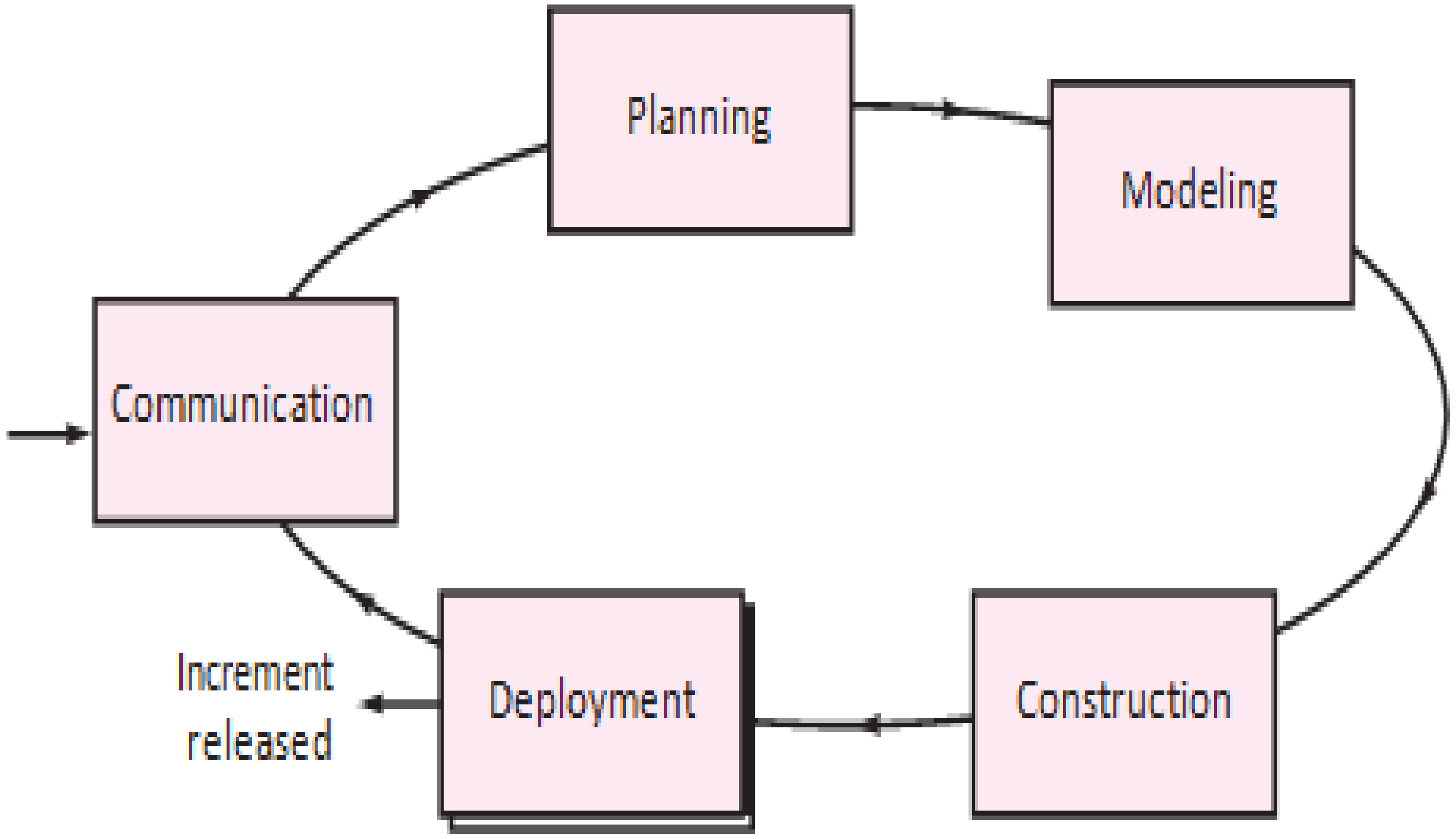


(a) Linear process flow

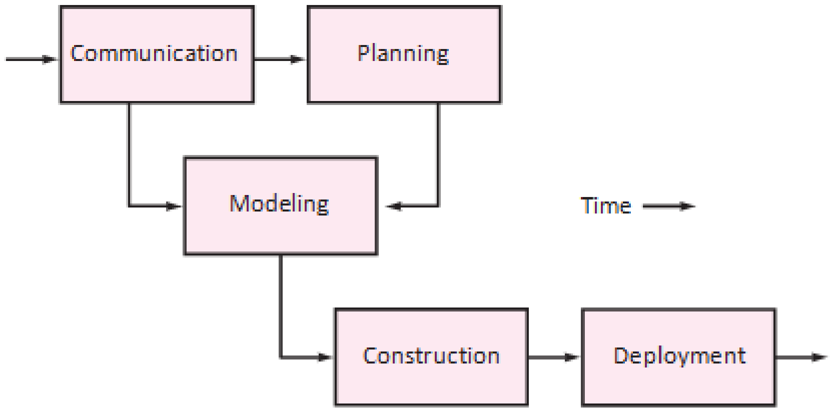
Iterative process flow



Evolutionary process flow



Parallel Process flow



Identifying a Task Set

- A task set defines the actual work to be done to accomplish the objectives of a software engineering action.
  - A list of the task to be accomplished
  - A list of the work products to be produced
  - A list of the quality assurance filters to be applied

Process Patterns

- A *process pattern*
  - describes a process-related problem that is encountered during software engineering work,
  - identifies the environment in which the problem has been encountered, and
  - suggests one or more proven solutions to the problem.

- Stated in more general terms, a process pattern provides you with a *template* [Amb98]—a consistent method for describing problem solutions within the context of the software process.

Ambler has proposed a template for describing a Process Pattern

- Pattern Name
- Forces
- Type
- Initial Context
- Problem
- Solution
- Resulting Context
- Related Patterns
- Known Uses and Examples

Process Pattern Types

- *Stage patterns*—defines a problem associated with a framework activity for the process.
- *Task patterns*—defines a problem associated with a software engineering action or work task and relevant to successful software engineering practice
- *Phase patterns*—define the sequence of framework activities that occur with the process, even when the overall flow of activities is iterative in nature.

PROCESS ASSESSMENT AND IMPROVEMENT

- **Standard CMMI Assessment Method for Process Improvement (SCAMPI)** — provides a five step process assessment model that incorporates five phases: initiating, diagnosing, establishing, acting and learning.

- **CMM-Based Appraisal for Internal Process Improvement (CBA IPI)**—provides a diagnostic technique for assessing the relative maturity of a software organization; uses the SEI CMM as the basis for the assessment [Dun01]

CMMI STEPS

- Initiate
- Managed
- Defined
- Qualitatively Managed
- Optimize

Process Assessment and Improvement

- **SPICE—The SPICE (ISO/IEC15504)** standard defines a set of requirements for software process assessment. The intent of the standard is to assist organizations in developing an objective evaluation of the efficacy of any defined software process. [ISO08]
- **ISO 9001:2000 for Software—**a generic standard that applies to any organization that wants to improve the overall quality of the products, systems, or services that it provides. Therefore, the standard is directly applicable to software organizations and companies. [Ant06]

- **Prescriptive Models**
  - The Waterfall Model(Linear Sequential Model/Classic Life cycle Model)
- **Incremental Models**

  -Incremental Model

  -Rapid Action Development(RAD)Model
- **Iterative Development Model(Evolutionary Process Models)**
  - Prototyping Model(Paradigm)
  - Spiral Model
  - Concurrent Model

Prescriptive Models

- Prescriptive process models advocate an orderly approach to software engineering

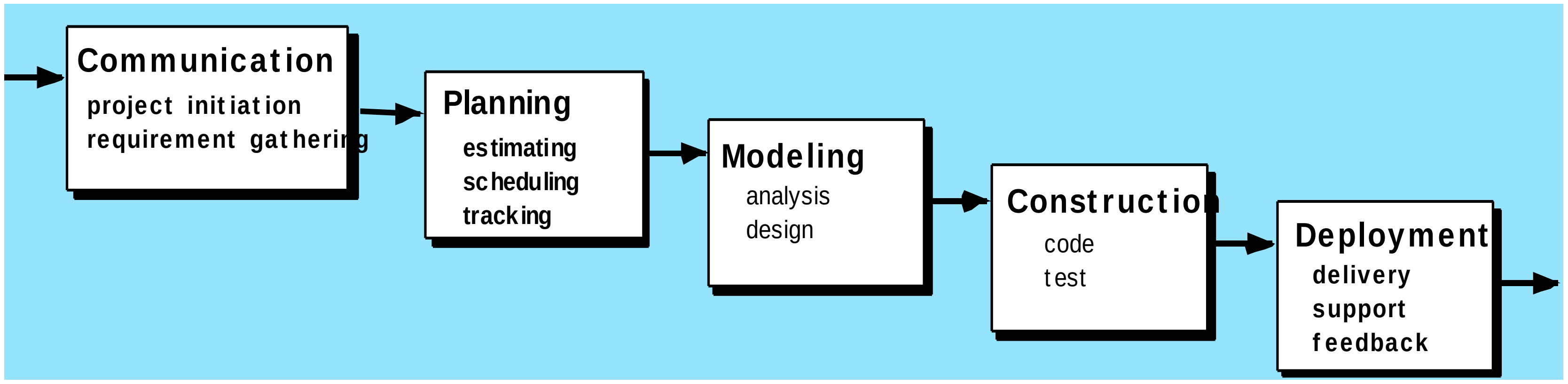
That leads to a few questions …

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

The Waterfall Model

- Sometimes called the *classic life cycle or the waterfall model, the linear sequential model* suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment, culminating in ongoing support of the completed software as shown in Fig.
- There are times when the requirements for a problem are well understood—when work flows from **communication** through **deployment** in a

The Waterfall Model



Advantages of Waterfall Model

- Base model
- Simple and easy
- Small projects
- It is systematic sequential approach for software development.
- Hence, it is most widely used

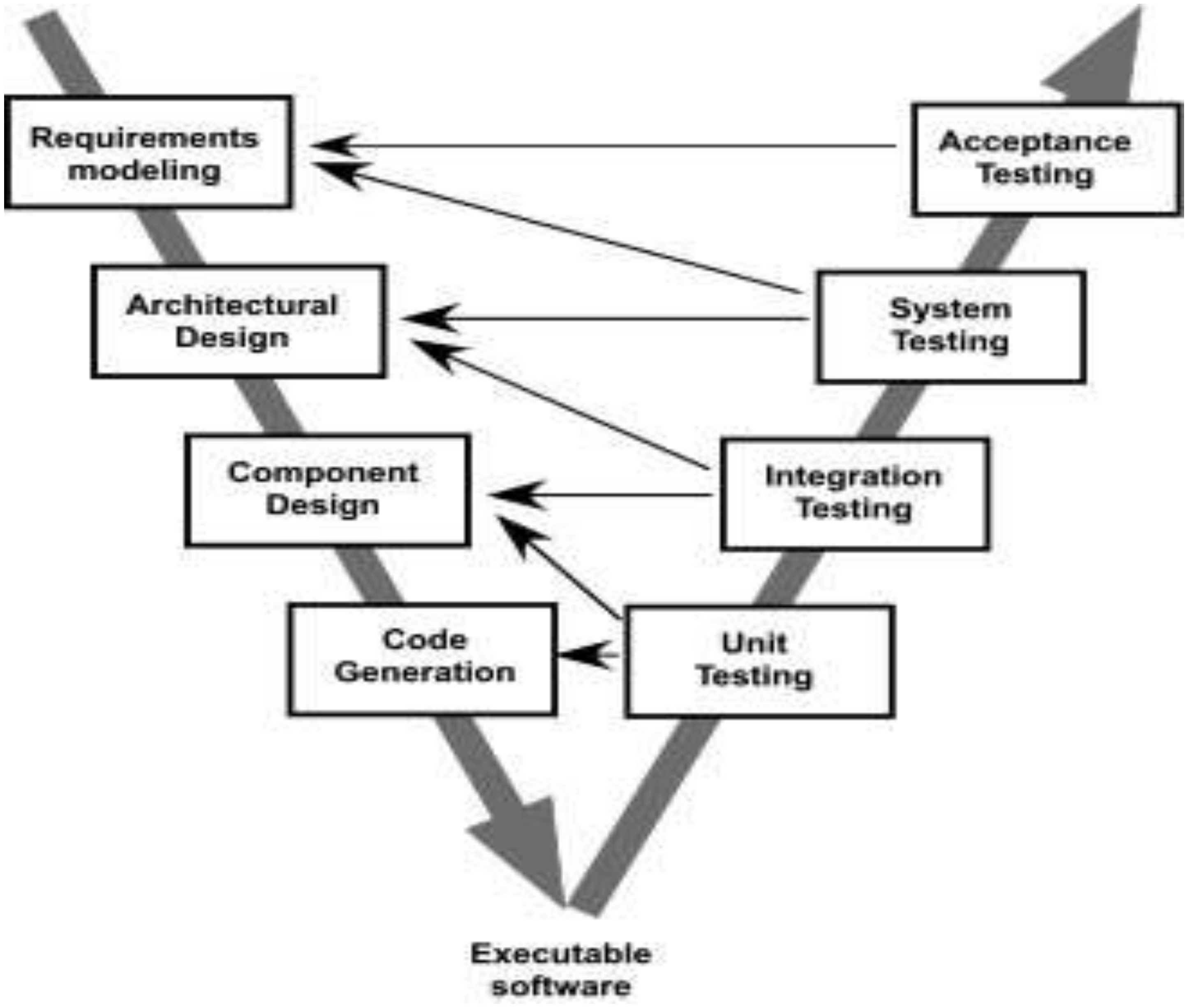
Limitations of Waterfall Model

- Changes can cause confusion as the project team proceeds.
- No feedback
- No Experiment
- No Parallelism
- High risk
- 60% efforts need in maintenance
- Customer must have a patience

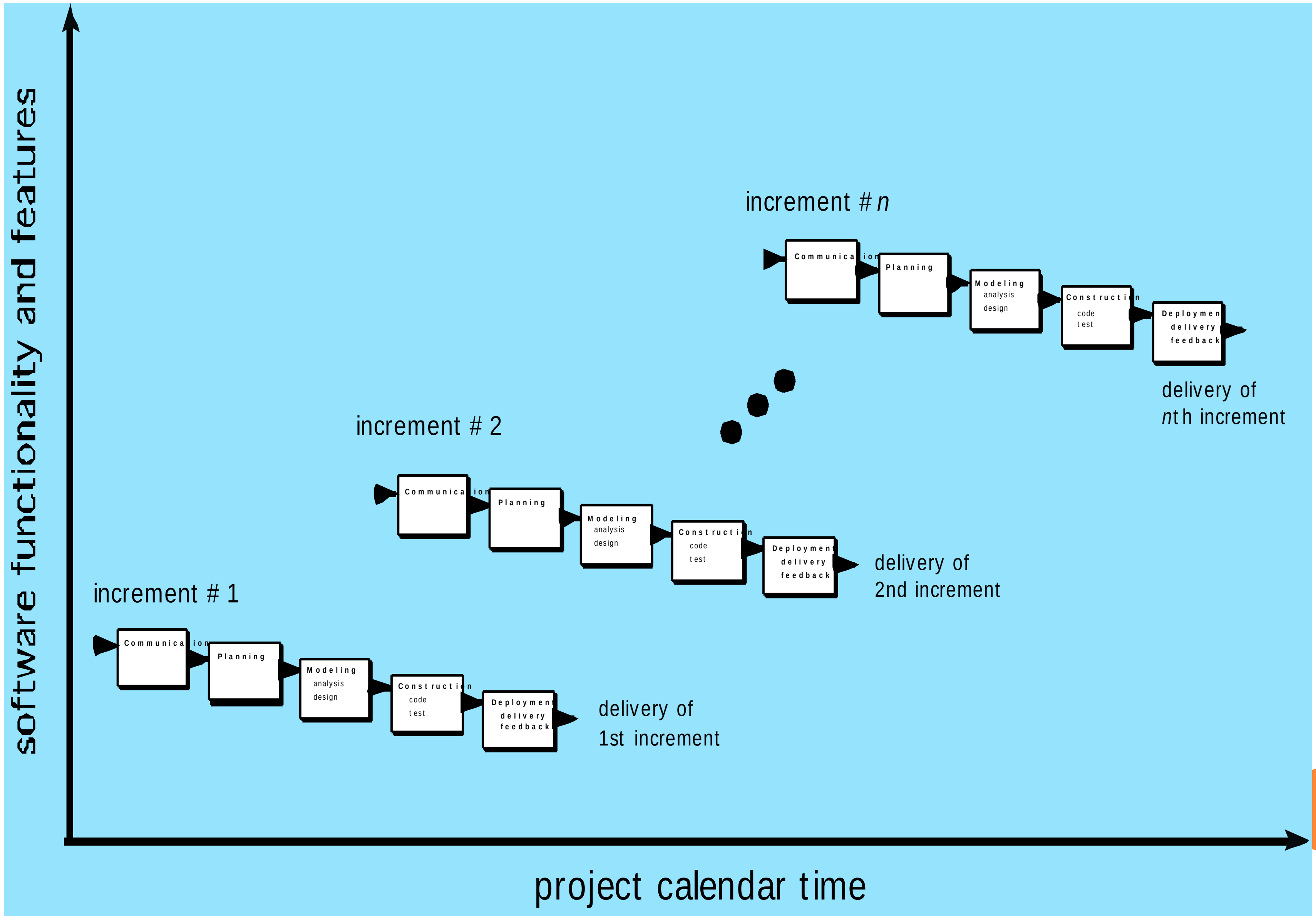
Concluding Remarks

- In an interesting analysis of actual projects, Bradac [Bra94] found that the linear nature of the classic life cycle leads to “blocking states” in which some project team members must wait for other members of the team to complete dependent tasks.
- However, it can serve as a useful process model in situations where requirements are fixed and work is to proceed to completion in a linear manner.

The V-Model


Requirements modeling
Acceptance Testing
Architectural Design
System Testing
Component Design
Integration Testing
Code Generation
Unit Testing
Executable software

The Incremental Model



THE INCREMENTAL MODEL

- There are many situations in which initial software requirements are reasonably well defined, but the overall scope of the development effort precludes a purely linear process.
- In addition, there may be a compelling need to provide a limited set of soft- ware functionality to users quickly and then refine and expand on that functionality in later software releases.
- In such cases, you can choose a process model that is designed to produce the software in increments.
- The *incremental* model combines elements of

The Incremental Model

- The incremental model applies linear sequences in a staggered fashion as calendar time progresses.

- The incremental model delivers a series of releases, called increments , that provide progressively more functionality for the customer as each increment is delivered. E.g. Word Processing Software.

- When an incremental model is used, the first increment is often a *core product*. That is, basic requirements are addressed but many supplementary features (some known, others unknown) remain undelivered. The core product is used by the customer (or undergoes detailed evaluation). As a result of use and/or evaluation, a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality. This process is repeated following the delivery of each increment, until the complete product is produced.

Advantages of Incremental Model

- Development team is small
- Handle technical risks
- Initial product delivery is faster
- Customer response

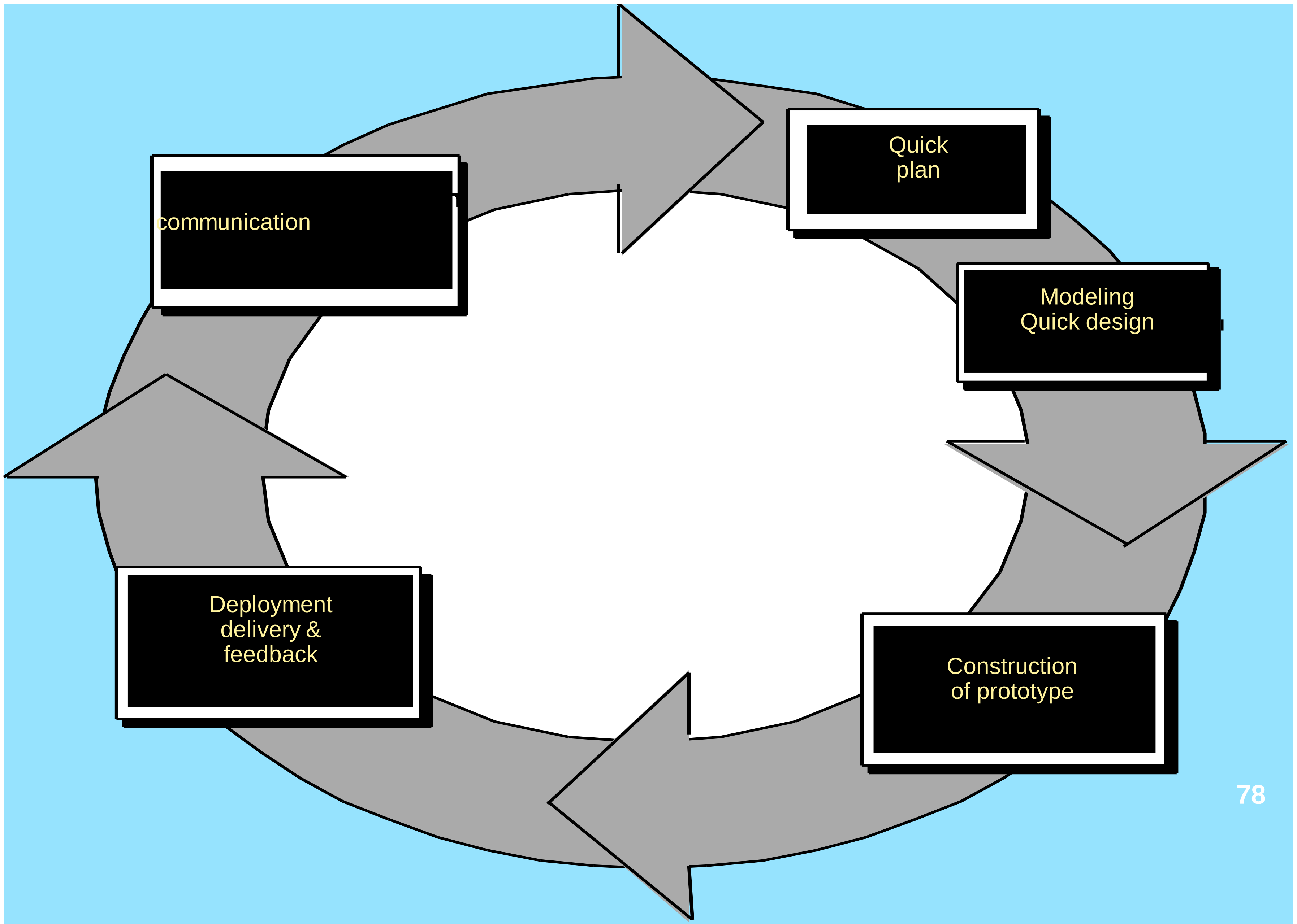
LIMITATIONS

- Increases product cost
- Customer demand after each increment
- Needs clear and complete planning and design.

Iterative Development Model (Evolutionary Process Models)

- Prototyping Model
- Spiral Model

Evolutionary Models: Prototyping



EVOLUTIONARY PROCESS MODELS

- Evolutionary models are iterative.
- Software, like all complex systems, evolves over a period of time. Business and product requirements often change as development proceeds, making a straight line path to an end product unrealistic;
- tight market deadlines make completion of a comprehensive software product impossible, but a limited version must be introduced to meet competitive or business pressure;
- a set of core product or system requirements is well understood, but the details of product or system extensions have yet to be defined.
- In these and similar situations, you need a process model that has been explicitly designed to accommodate a

Prototyping

- When requirements are **fuzzy** and the developer is unsure of the **efficiency** of **an algorithm**, the adaptability of an operating system etc. In these and many other situations, a prototyping paradigm may offer the best approach.

PROTOTYPING

- The prototyping paradigm begins with communication.
- You meet with other stakeholders to define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory.
- A prototyping iteration is planned quickly, and modeling (in the form of a "quick de- sign") occurs.
- A quick design focuses on a representation of those aspects of the soft- ware that will be visible to end users (e.g., human interface layout or output display formats).
- The quick design leads to the construction of a prototype.
- The prototype is deployed and evaluated by stakeholders, who provide feedback that is used to further refine requirements.
- Iteration occurs as the prototype is tuned to satisfy the needs of various stakeholders, while at the same time enabling you to better under- stand what needs to be done.

Prototyping

- Ideally the prototype serves as a mechanism for **identifying software requirements.**
- The Prototype can serve as **"the first system",** the one that **Brooks** recommends we **throw** away. But this may be an idealized view.
- It is true that both customers and developers like the prototyping paradigm.
- Users get a feel for the actual system and developers get to build something immediately.

Advantages of Prototyping model

- Effective paradigm for software engineering.
- Requirements are more clear
- System is transparent

DRAWBACKS:-

- Stakeholders see what appears to be a **working version** of the software, unaware that the prototype is held together haphazardly, unaware that in the rush to get it working you haven't considered overall **software quality** or **long-term maintainability.**

- As a software engineer, you often make implementation compromises in order to get a prototype working quickly. An inappropriate operating system or programming language may be used simply because it is available and known; an inefficient algorithm may be implemented simply to demonstrate capability.

Disadvantages of Prototyping model

- Software developer haven’t Considered Software quality to get a prototype working quickly.
- Uses inappropriate operating system and programming languages.
- Implement inefficient algorithms.

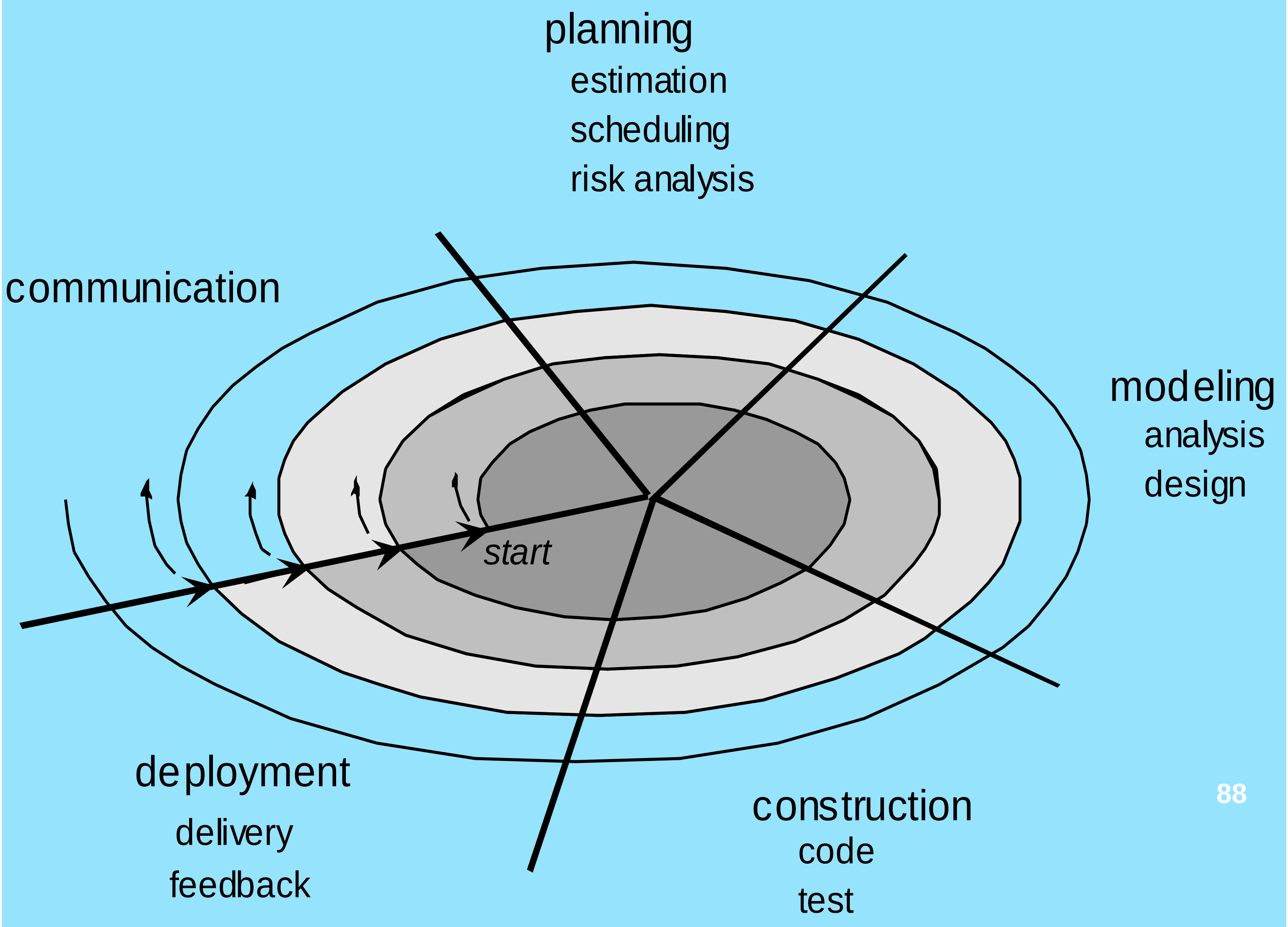
Concluding Remarks

- Although problems can occur, prototyping can be an **effective paradigm** for soft- ware engineering. The **key** is to define the **rules of the game** at the beginning;
- that is, all stakeholders should agree that the prototype is built to serve as a mechanism for defining requirements.
- It is then discarded (at least in part), and the actual software is engineered with an eye toward quality.

Evolutionary Models : The Spiral Model

- Originally proposed by **Barry Boehm** [Boe88], the *spiral model* is an evolutionary software process model.
- It **encompasses** the best features of both prototyping and waterfall model.
- It provides the **potential** for rapid development of increasingly more complete versions of the software.
- Using the spiral model, software is developed in a series of **evolutionary releases.**

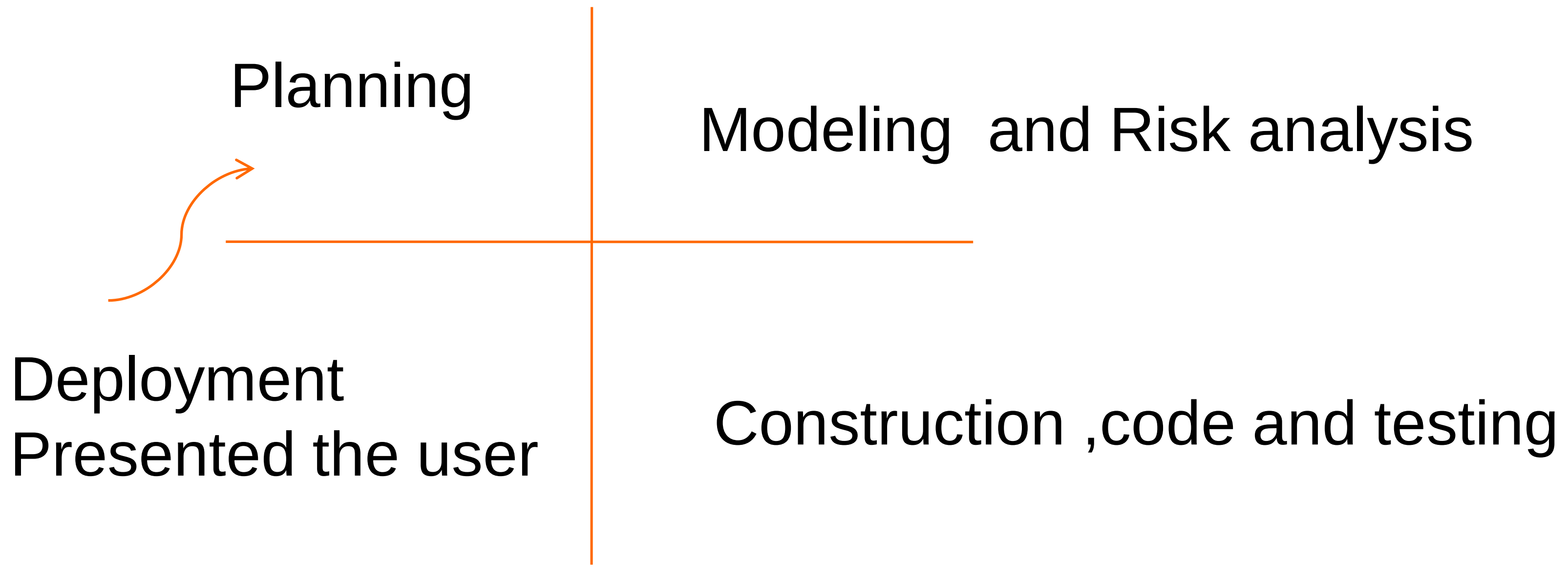
Evolutionary Models: The Spiral



FEATURES

- Cyclic Approach
- Large Projects
- Risk driven(Find out the risk and mitigate /remove the risk)
- Anchor point
- Apply Model throughout the life of computer software.

Spiral Model



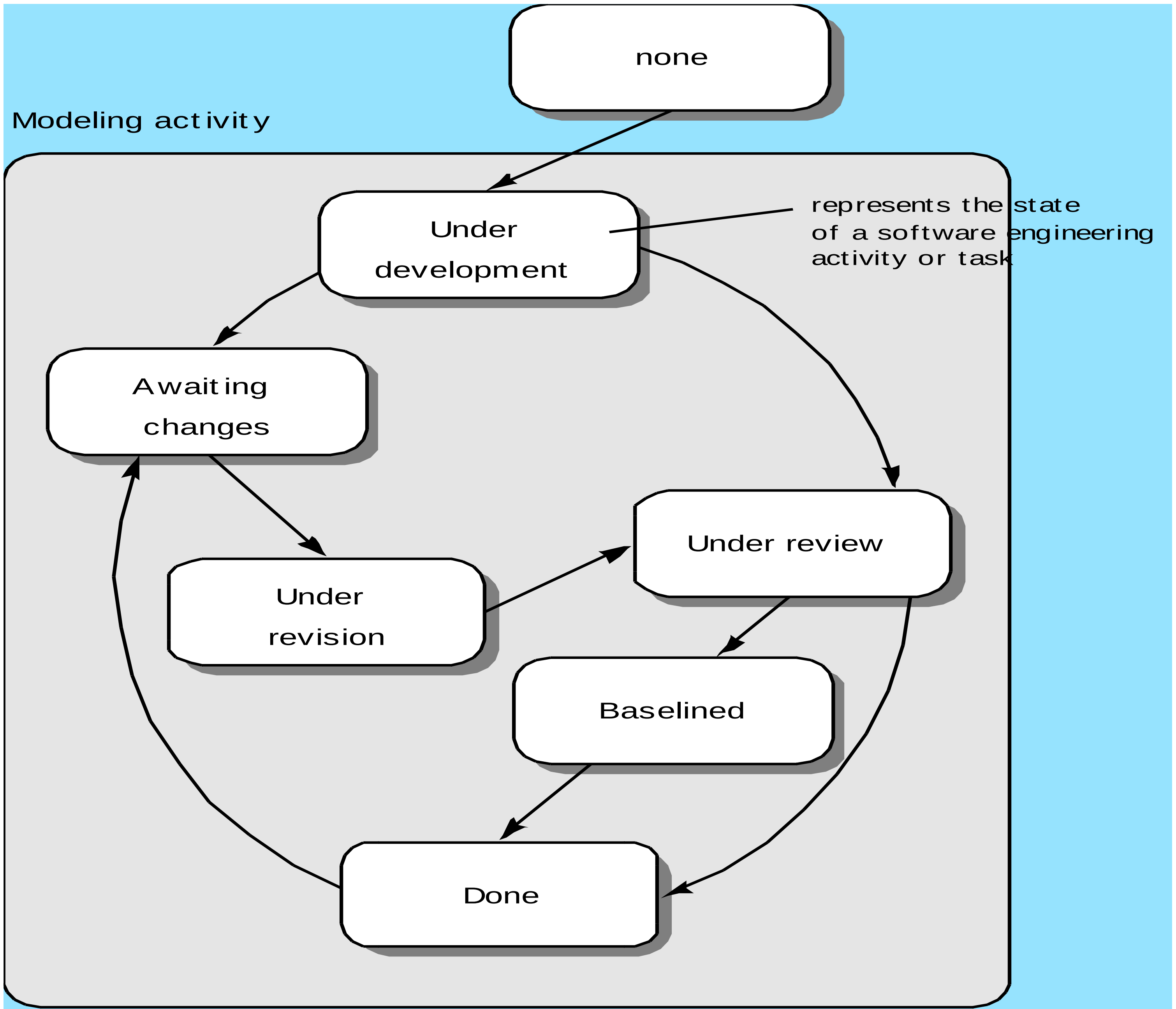
Advantages of Spiral model

- Realistic approach
- Project monitoring easy
- Reduces number of risks in software development.
- Suitable for very high risk projects.

DISADVANTAGES OF SPIRAL MODEL

- High cost
- Strict rules and regulations.
- If risk is not discovered in early stage ,it becomes a major risk in later days.
- Difficult to convince customer as controllable evolutionary approach.

Evolutionary Models: Concurrent



ADVANTAGES OF CONCURRENT MODEL

- Applicable for all types of software development process.
- Gives Systematic representation of current state of the project.
- All activities exist concurrently with different states.

DISADVANTAGES OF CONCURRENT MODEL

- Any change in requirement may halt the progress due to concurrency.
- Require clear and updated communication among team members.
- Software Requirement Specification(SRS)must be updated regularly to reflect the changes

Agile software development

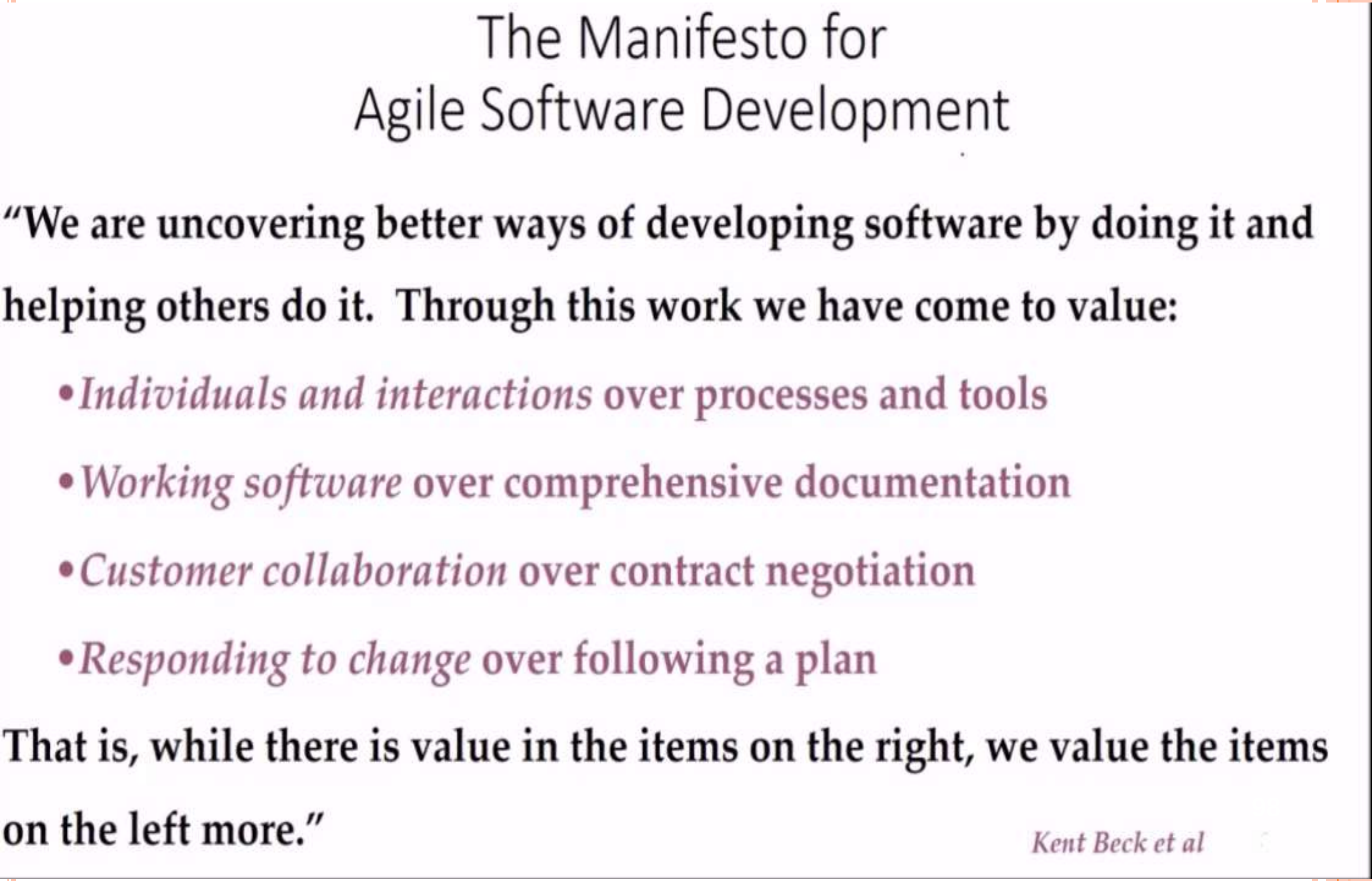
- Agile manifesto,
- agility principles,
- Agile methods,
- myth of planned development,
- Introduction to Extreme programming and Scrum.

What is “Agility”?

- Effective (rapid and adaptive) response to change
- Effective communication among all stakeholders
- Drawing the customer onto the team
- Organizing a team so that it is in control of the work performed

Yielding …

- Rapid, incremental delivery of software



The Manifesto for Agile Software Development

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- ***Individuals and interactions*** **over processes and tools**
- ***Working software*** **over comprehensive documentation**
- ***Customer collaboration*** **over contract negotiation**
- ***Responding to change*** **over following a plan**

That is, while there is value in the items on the right, we value the items on the left more."

Kent Beck et al

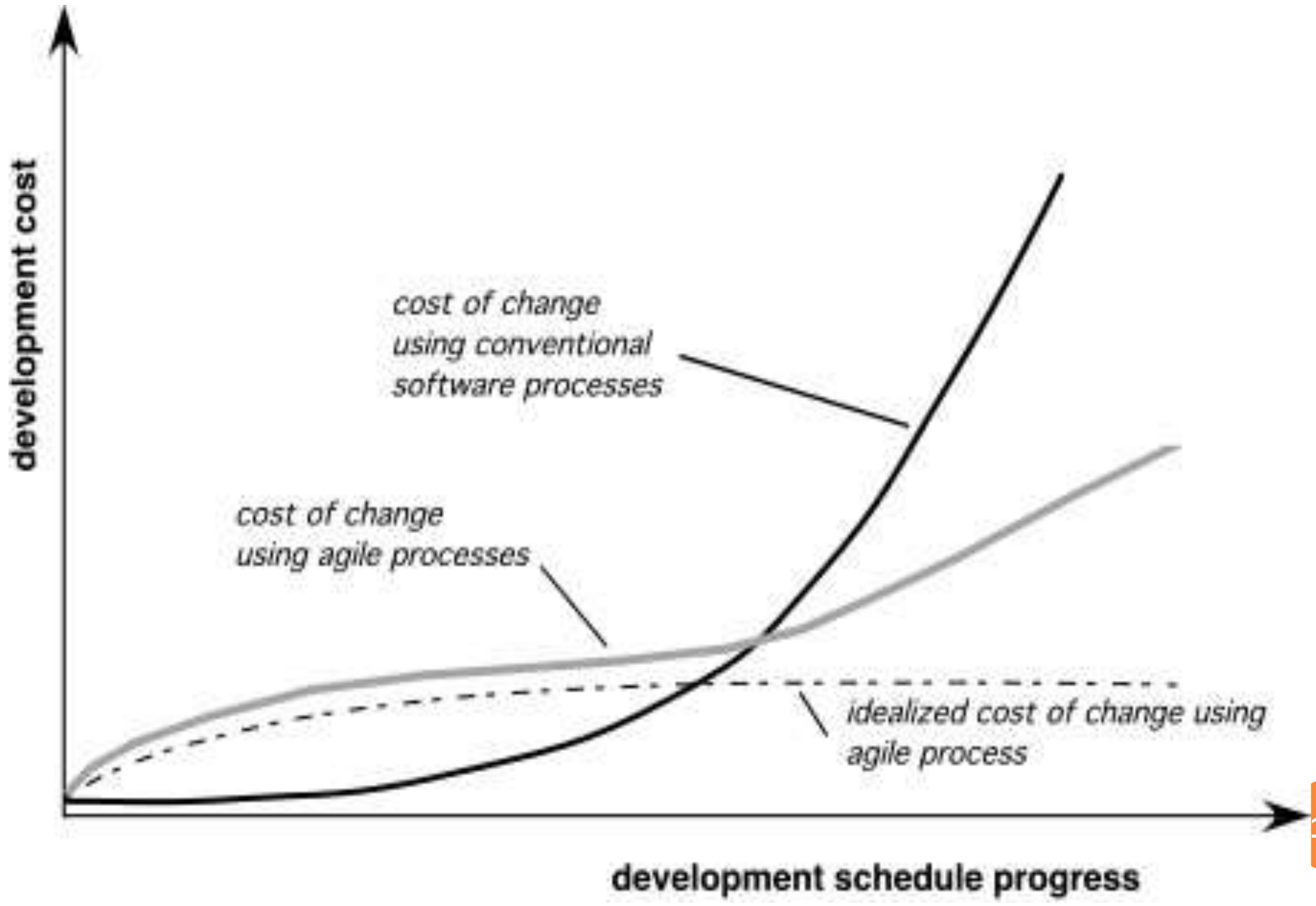
What is "Agility"?

- Effective (rapid and adaptive) response to change
- Effective communication among all stakeholders
- Drawing the customer onto the team
- Organizing a team so that it is in control of the work performed

Yielding …

- Rapid, incremental delivery of software

Agility and the Cost of Change



An Agile Process

- Is driven by customer descriptions of what is required (scenarios)
- Recognizes that plans are short-lived
- Develops software iteratively with a heavy emphasis on construction activities
- Delivers multiple ‘software increments’
- Adapts as changes occur

Agility Principles - I

1. Our highest priority is to **satisfy the customer** through early and continuous delivery of valuable software.
2. Welcome **changing requirements**, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the **shorter timescale.**
4. Business people and developers must work together daily throughout the project.
5. Build projects around **motivated individuals**. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face–to–face conversation.

Agility Principles - II

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work done – is essential.
11. The best architectures, requirements, and designs emerge from self–organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Human Factors

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

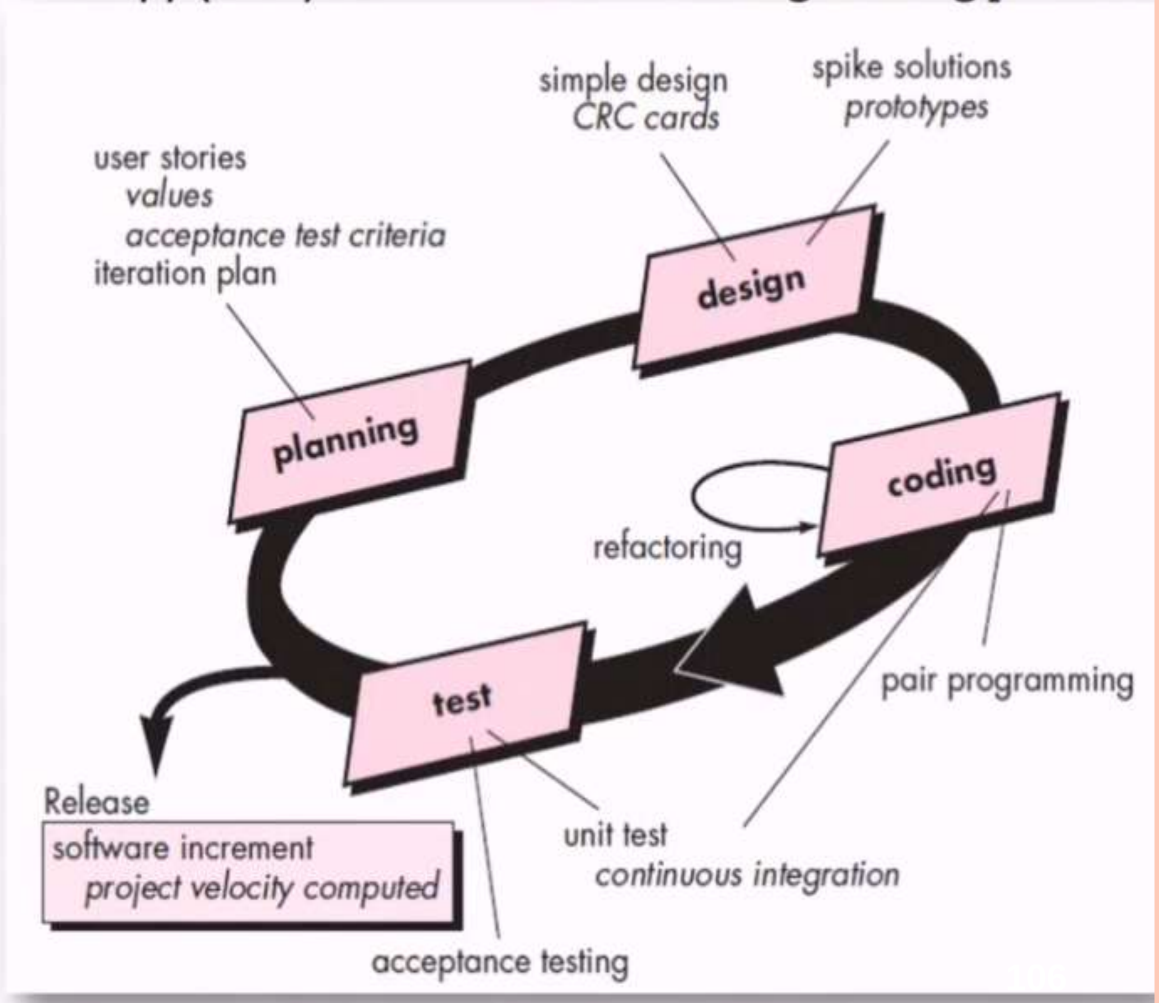
Myth of Planned Development

- Agile development is a methodology.
- Agile is Temporary
- Agile does not scale
- Agile is undisciplined
- Agile has no planning
- Agile has no documentation
- Agile has no upfront design

Extreme Programming (XP)

Extreme Programming uses an object-oriented approach as its preferred development paradigm and encompasses a set of rules and practices that occur within the context of four framework activities: planning, design, coding, and testing.

Extreme Programming process



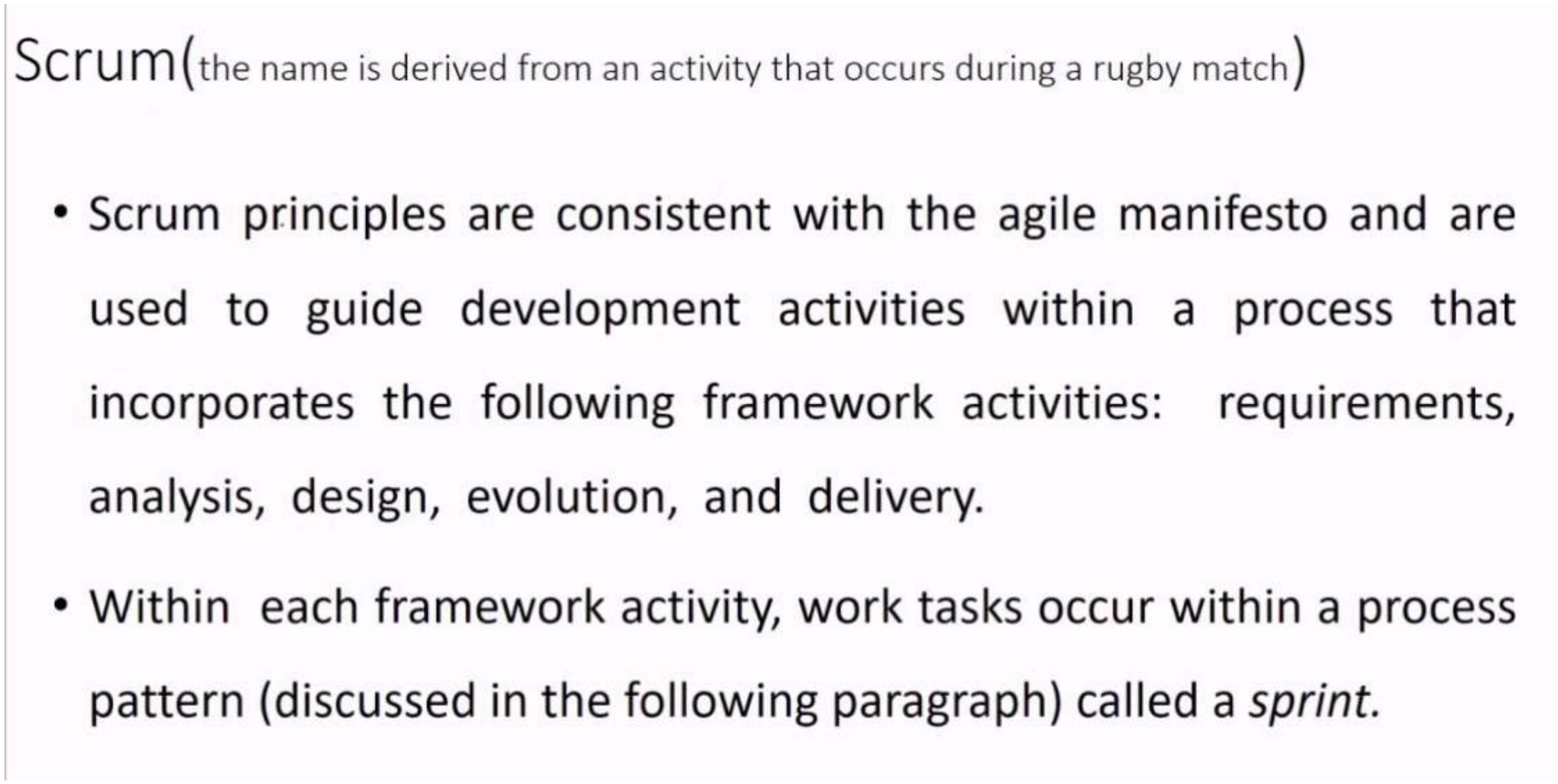
Extreme Programming (XP)

- The most widely used agile process, originally proposed by Kent Beck
- XP Planning
  - Begins with the creation of “user stories”
  - Agile team assesses each story and assigns a cost
  - Stories are grouped to for a deliverable increment
  - A commitment is made on delivery date
  - After the first increment “project velocity” is used to help define subsequent delivery dates for other increments

Extreme Programming (XP)

- XP Design
  - Follows the KIS principle
  - Encourage the use of CRC cards (Class-responsibility-Collaboration card are used in design object oriented software)
  - For difficult design problems, suggests the creation of "spike solutions"—a design prototype
  - Encourages "refactoring"—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"
- XP Testing
  - All unit tests are executed daily
  - "Acceptance tests" are defined by the customer and executed to assess customer visible functionality

SCRUM Principle

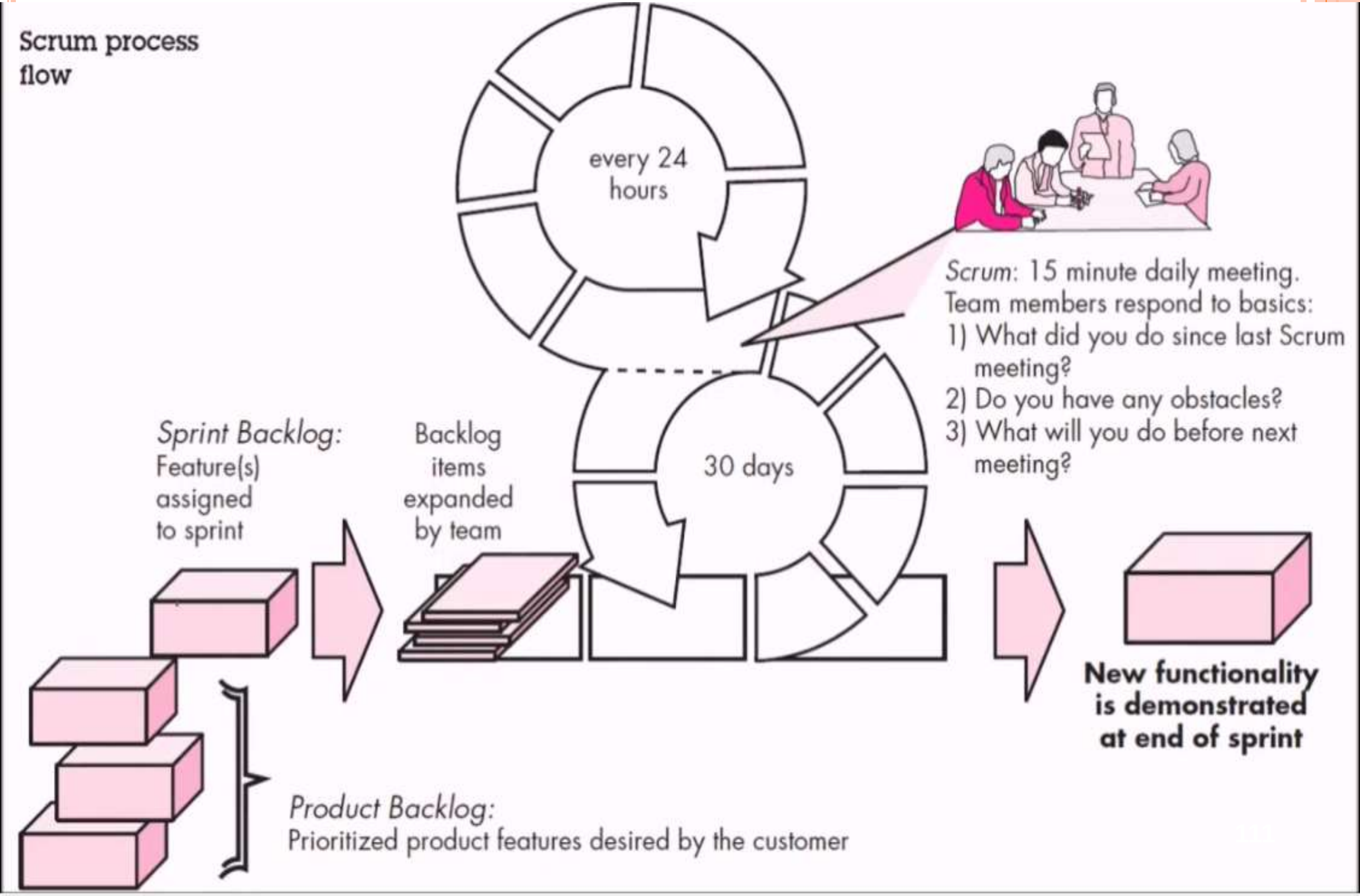


Scrum(the name is derived from an activity that occurs during a rugby match)

- Scrum principles are consistent with the agile manifesto and are used to guide development activities within a process that incorporates the following framework activities: requirements, analysis, design, evolution, and delivery.
- Within each framework activity, work tasks occur within a process pattern (discussed in the following paragraph) called a *sprint*.

Scrum

- Originally proposed by Schwaber and Beedle
- Scrum—distinguishing features
  - 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
  - Meetings are very short and sometimes conducted without chairs
  - "demos" are delivered to the customer with the time-box allocated


Scrum process flow
every 24 hours
30 days
Scrum: 15 minute daily meeting.
Team members respond to basics:
1) What did you do since last Scrum meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?
Sprint Backlog:
Feature(s) assigned to sprint
Backlog items expanded by team
New functionality is demonstrated at end of sprint
Product Backlog:
Prioritized product features desired by the customer

Agile Modeling

- Originally proposed by Scott Ambler
- Suggests a set of agile modeling principles
  - Model with a purpose
  - Use multiple models
  - Travel light
  - Content is more important than representation
  - Know the models and the tools you use to create them
  - Adapt locally

Agile Practices

- Pair programming,
- Test driven development,
- Continuous integration
- Refactoring

Pair Programming

- Pair programming is a style of programming in which two programmers **work side-by-side at one computer**, sharing one screen, keyboard and mouse, continuously collaborating on the same design, algorithm, code or test.
- One programmer, termed as the **driver**, has control of the keyboard/mouse and actively implements the code or writes a test.
- The other programmer, termed as the **navigator**, continuously observes the work of the driver to identify defects and also thinks strategically about the direction of the work.
- When necessary, the two programmers brainstorm on any challenging problem.
- The two programmers periodically switch roles and work together as equals to develop a software.

Pair Programming – Advantages

The significant advantages of Pair Programming are –

- Many mistakes are detected at the time they are typed, rather than in QA Testing or in the field.
- The end defect content is statistically lower.
- The designs are better and code length shorter.
- The team solves problems faster.
- People learn significantly more about the system and about software development.
- The project ends up with multiple people understanding each piece of the system.
- People learn to work together and talk more often together, giving better information flow and team dynamics.

Pair Programming research reveals that –

- Pairs use no more man-hours than singles.
- Pairs create fewer defects.
- Pairs create fewer lines of code.
- Pairs enjoy their work more.

Test Driven Development (TDD)

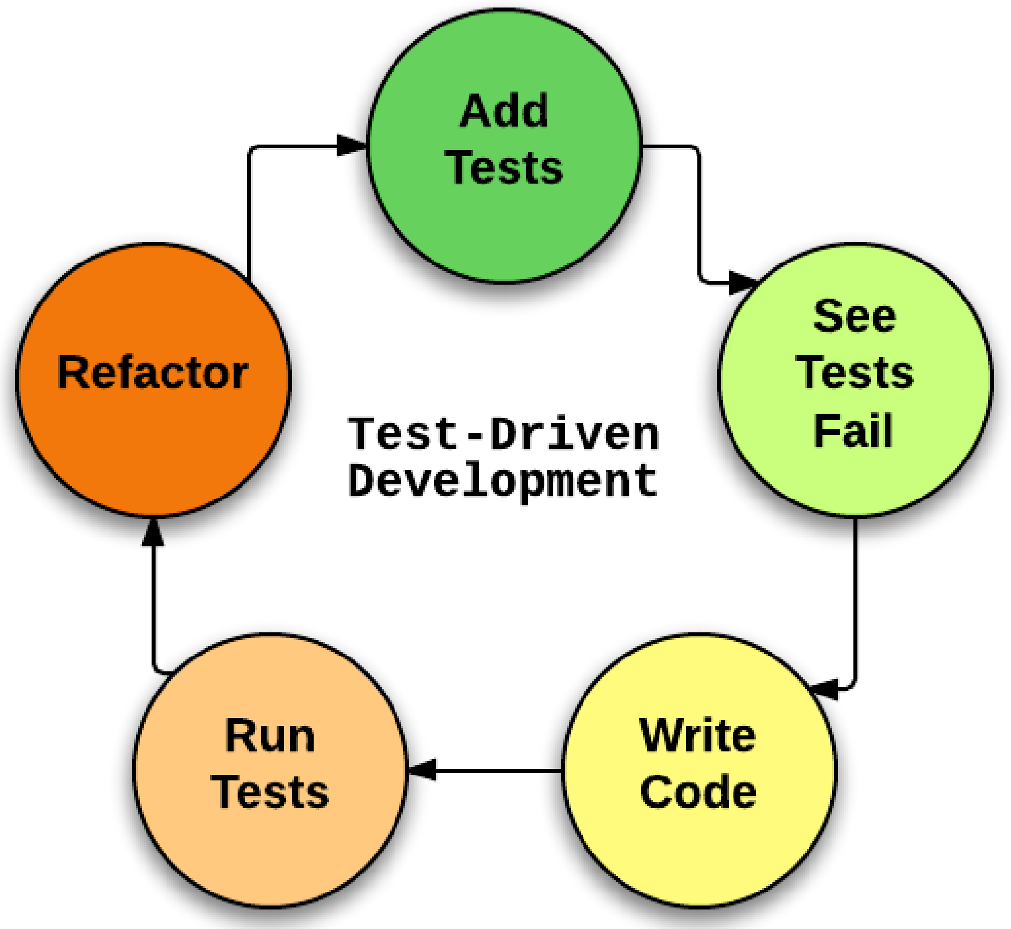
TEST DRIVEN DEVELOPMENT

- **Test-driven development (TDD)**, also called test-driven design, is a method of implementing software programming that interlaces unit **testing, programming and refactoring on source code.**
- Test-driven development was introduced as part of a larger software design paradigm known as **Extreme Programming (XP),** which is part of the Agile software development methodology.
- Test-driven development starts with developing test for each one of the features.
- The test might fail as the tests are developed even before the development.
- Development team then develops and refactors the code to pass the test. Test-driven development is related to the test-first programming evolved as part of extreme programming concepts.

Steps of the test-driven development approach:-

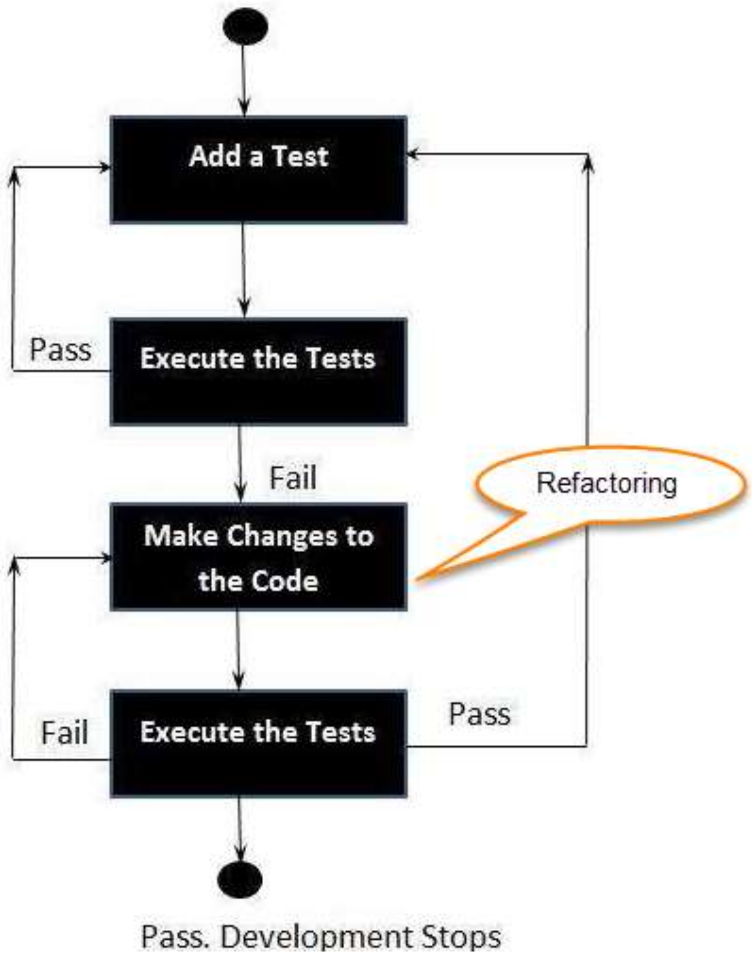
- Before any new code is written, the programmer must first create a failing unit test. Then,
- the programmer -- or pair, or mob -- creates just enough code to satisfy that requirement.
- Once the test is passing, the programmer may re-factor the design, making improvements without changing the behaviour.

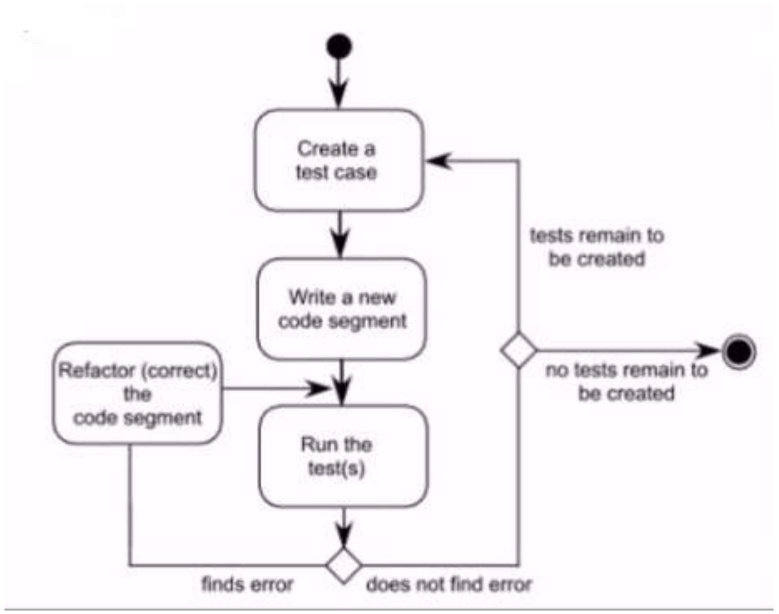
Test Driven Devolpment



- **Test-Driven Development Process**:
- Add a Test
- Run all tests and see if the new one fails
- Write some code
- Run tests and Refactor code
- Repeat

EXAMPLE


Add a Test
Pass
Execute the Tests
Fail
Refactoring
Make Changes to the Code
Fail
Execute the Tests
Pass
Pass. Development Stops


Create a test case
Write a new code segment
Refactor (correct) the code segment
Run the test(s)
tests remain to be created
no tests remain to be created
finds error
does not find error

Context of Testing:

- Valid inputs
- Invalid inputs
- Errors, exceptions, and events
- Boundary conditions
- Everything that might break

Benefits of TDD:

- Much less debug time
- Code proven to meet requirements
- Tests become Safety Net
- Near zero defects
- Shorter development cycles

Refactoring

REFACTORING

- Due to change in requirements or due to addition of new functionality the changes occur in coding.
- For accommodating those changes in the code we need to change the design.
- If we plan to code as per the changed design, then the code becomes very complex.
- Refactoring is the technique used to improve the code & avoid the design decay with time.
- Refactoring is done during coding.

Definition

- Refactoring consists of improving the internal structure of an existing program's source code, while preserving its external behaviour.
- The noun "refactoring" refers to one particular behaviour-preserving transformation, such as "Extract Method" or "Introduce Parameter.

Refactoring-Common Pitfalls

Refactoring does "not" mean:

- rewriting code
- fixing bugs
- improve observable aspects of software such as its interface
- Refactoring in the absence of safeguards against introducing defects (i.e. violating the "behaviour preserving" condition) is risky.
- Safeguards include aids to regression testing including automated unit tests or automated acceptance tests, and aids to formal reasoning such as type systems

EXPECTED BENEFITS

The following are claimed benefits of refactoring:

- refactoring improves objective attributes of code (length, duplication, coupling and cohesion, cyclomatic complexity) that correlate with ease of maintenance
- refactoring helps code understanding
- refactoring encourages each developer to think about and understand design decisions, in particular in the context of collective ownership / collective code ownership
- refactoring favours the emergence of reusable design elements (such as design patterns) and code modules

PURPOSE

- The purpose of code refactoring is to improve some of the non-functional properties of the code, such as readability, complexity, maintainability and extensibility.
- Refactoring can extend the life of source code, preventing it from becoming legacy code. The refactoring process makes future enhancements to such code a more pleasant experience.

Continuous Integration

- Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day.
- Each check-in is then verified by an automated build, allowing teams to detect problems early.
  By integrating regularly, you can detect errors quickly, and locate them more easily.

SOLVE PROBLEMS QUICKLY

- Because you're integrating so frequently, there is significantly less back-tracking to discover where things went wrong, so you can spend more time building features.
- Continuous Integration is cheap.
- Not integrating continuously is expensive.
- If you don't follow a continuous approach, you'll have longer periods between integrations.
- This makes it exponentially more difficult to find and fix problems.
- Such integration problems can easily knock a project off-schedule, or cause it to fail altogether.

Continuous Integration-Benefits

Continuous Integration brings multiple benefits to your organization:

- Say goodbye to long and tense integrations
- Increase visibility enabling greater communication
- Catch issues early and nip them in the bud
- Spend less time debugging and more time adding features
- Build a solid foundation
- Stop waiting to find out if your code's going to work
- Reduce integration problems allowing you to deliver software more rapidly

“**Continuous Integration** doesn’t get rid of bugs, but it does make them dramatically easier to find and remove.”

— Martin Fowler, Chief Scientist, ThoughtWorks

Reference

- *Software Engineering: A Practitioner’s Approach, 7/e*
- **by Roger S. Pressman**

THANK YOU