Introduction to Software Engineering

Dr.Sangeetha Yalamanchili
Department of IT
VRSEC

COURSE OUTCOMES

- **CO1-** Identify an appropriate software model that would implement the customer requirements.
- **CO2-** Analyze the requirements and identify the suitable architecture for the problem.
- **CO3-**Discriminate the specifications at each stage of Software Development Life Cycle.
- **CO4-** Implement various software testing strategies for verification and validation of the software products

SE


VALUE
QUALITY
SLICING
BUILDING
PLANNING
ORGANIZING
GUIDING
Bug-Free
Small
Value First
Continuous
Teams
What
Well-Designed
Complete
Grow the Product
What Next?
People ≠ Skills
When

Software Engineering Roles

Some of the roles include:

- ✓ Client
- ✓ Marketing and sales
- ✓ Product manager
- ✓ Program manager
- ✓ Business analyst
- ✓ Quality assurance personnel
- ✓ Software engineer (or developer)

What is Software ?

A Software is a….

Software can define as:

1. Instruction – executed provide desire features, function & performance.
2. Data structure that enable program to adequately manipulate operation.
3. Documents – operation and use of the program.

Software products may be developed for a particular customer or may be developed for a general market.

- Software products may be
  - **Generic** - developed to be sold to a range of different customers e.g. PC software such as Excel or Word.
  - **Bespoke** (**custom**) - developed for a single customer according to their specification.

S.E

Software Engineering

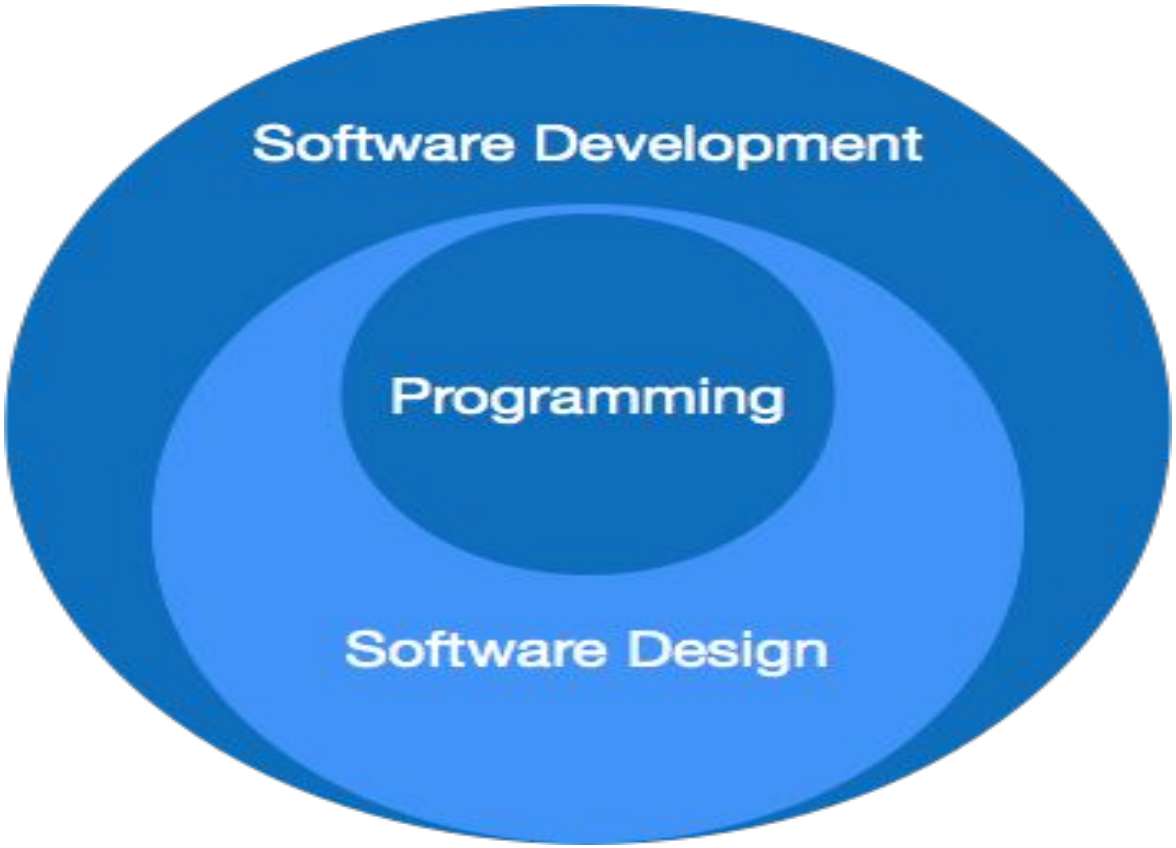
- **The IEEE definition:**
  - *Software Engineering: (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.*

16

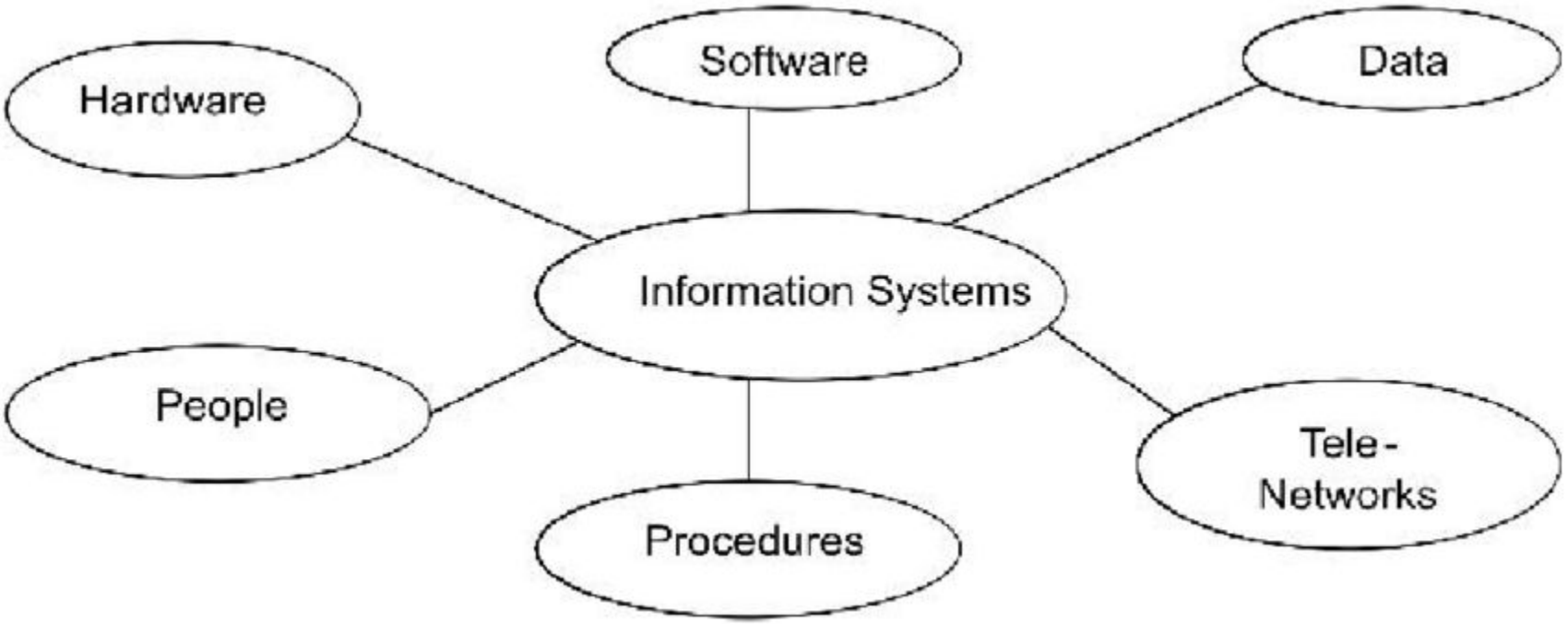
SE

Nature	Means	Aims
A computing discipline	Life cycle methods: - specification - design - implementation - evolution	Document and Programming
An engineering discipline	Engineering approaches: - standards - methodologies - tools - processes - organizational methods - management methods - quality assurance systems	Large-scale software

S.E



SE


Hardware
Software
Data
Information Systems
People
Tele-
Networks
Procedures

Difference between them?

TYPES

THE CHANGING NATURE OF SOFTWARE

- Seven Broad Categories of software are challenges for software engineers
  - System software
  - Application software
  - Engineering and scientific software
  - Embedded software
  - Product-line software
  - Web-applications
  - Artificial intelligence software

HARDWARE

- Mouse, Printer, Monitor, Keyboard

SOFTWARE

- Facebook, Email, YouTube, Word

Hardware vs. Software

Hardware

- Manufactured
- wear out
- Built using components
- Relatively simple

Software

- Developed/ engineered
- deteriorate
- Custom built
- Complex

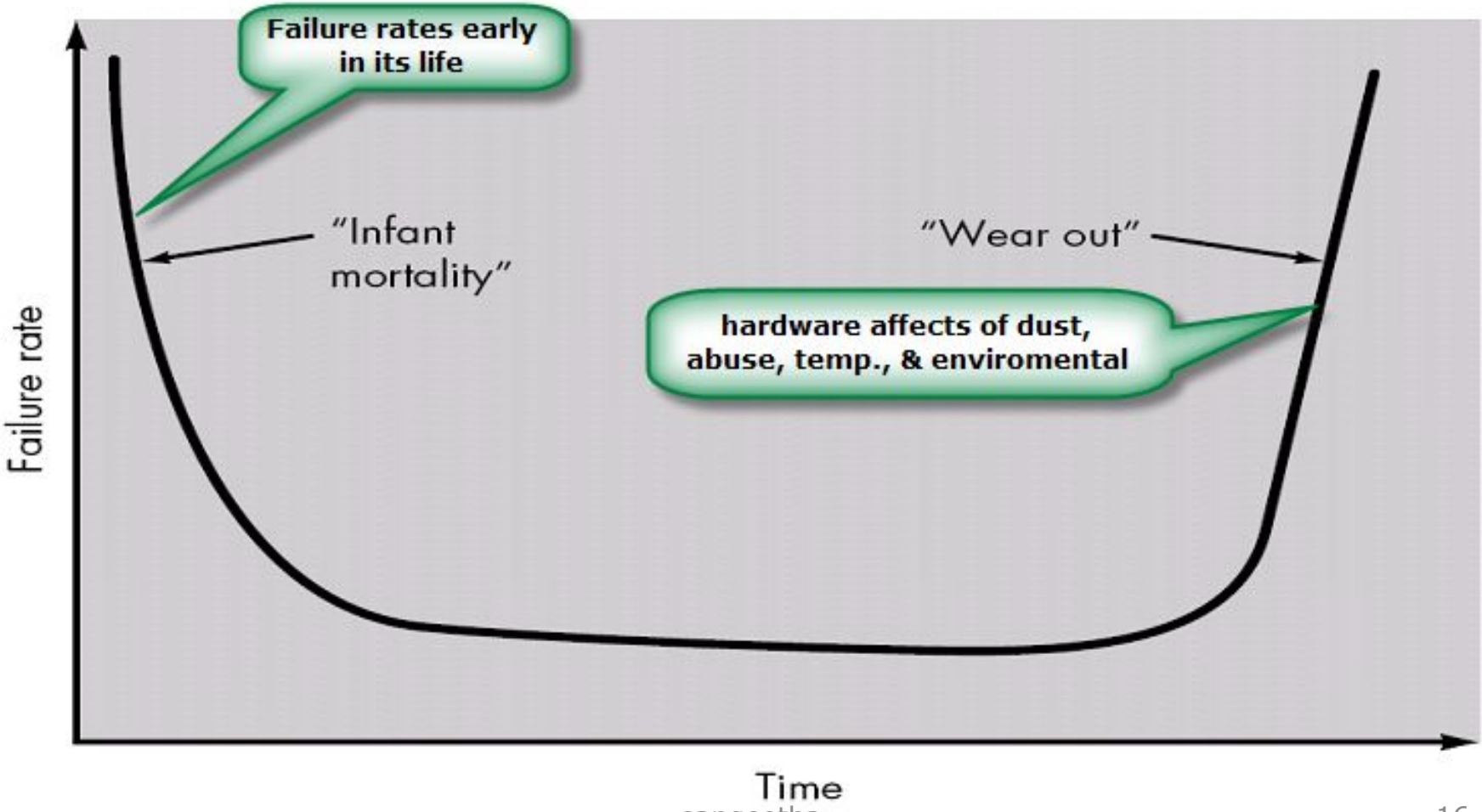
Manufacturing vs. Development

- Once a hardware product has been manufactured, it is difficult or impossible to modify. In contrast, software products are routinely modified and upgraded.

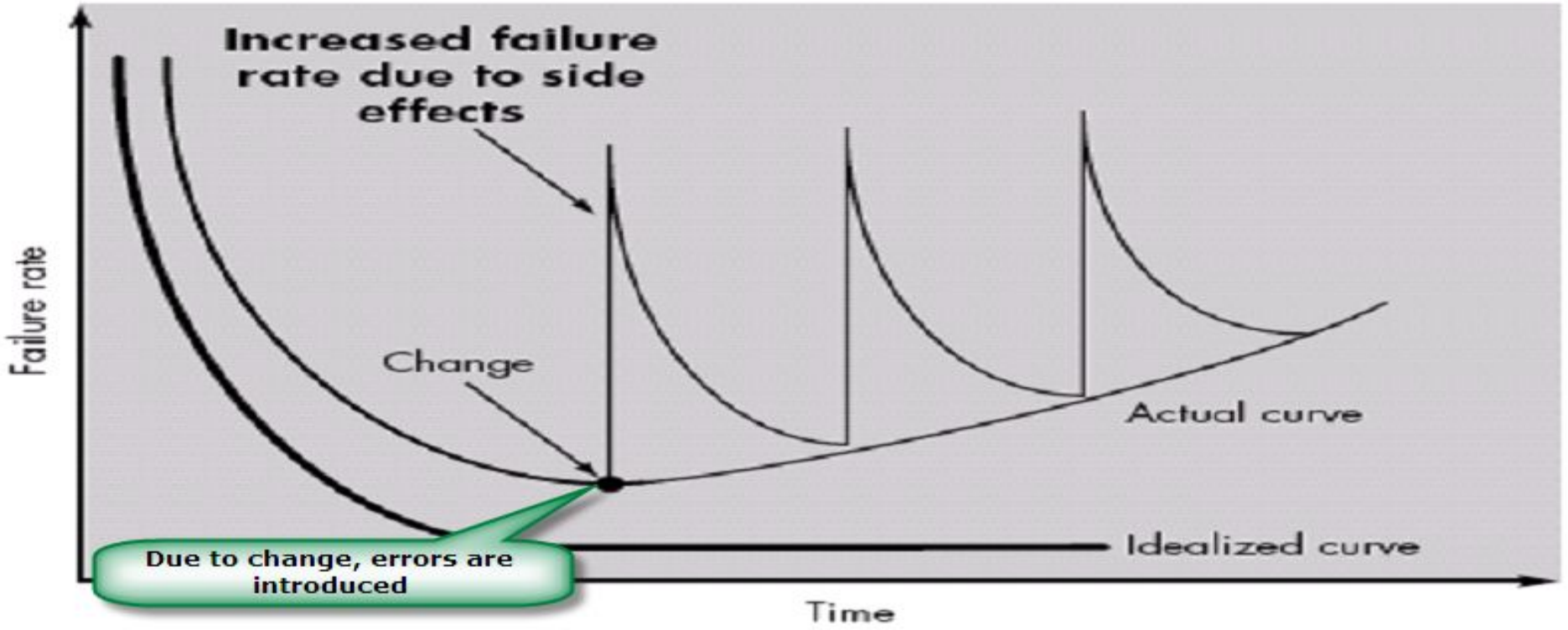
- In hardware, hiring more people allows you to accomplish more work, but the same does not necessarily hold true in software engineering.

- Unlike hardware, software costs are concentrated in design rather than production.

Failure curve for Hardware



Failure curve for Software



When a hardware component wears out, it is replaced by a spare part. There are no software spare parts. Every software failure indicates an error in design or in the process through which design was translated into machine executable code. Therefore, software maintenance involves considerably more complexity

Software characteristics

- Software is developed or engineered; it is not manufactured.

h/w vs s/w

- Software does not “wear out” but it does deteriorate.

failure curve

- Software continues to be custom built, as industry is moving toward component based construction.

Software Myths

Definition: Beliefs about software and the process used to build it. Myths have number of attributes that have made them insidious (i.e. dangerous).

- Misleading Attitudes - caused serious problem for managers and technical people.

1. **Management myths**
2. **Customer Myths**
3. **Practitioner's myths**

Management myths

Myth1: 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?

Myth2*:* If we get behind schedule, we can add more programmers and catch up

Myth3*:* If I decide to outsource the software project to a third party, I can just relax and let that firm build it.

Customer Myths

Customer may be a person from inside or outside the company that has requested software under contract.

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

Myth1*:* Once we write the program and get it to work, our job is done.

Myth2*:* Until I get the program "running" I have no way of assessing its quality.

Myth3: The only deliverable work product for a successful project is the working program.

Capability Maturity Model Integration (CMMI)

- The Software Engineering Institute (SEI) has developed process model to measure organization different level of process capability and maturity.
- CMMI – developed by SEI
- The CMMI defines each process area in terms of “specific goals” and the “specific practices” required to achieve these goals.`

CMMI Level

Level 0 (Incomplete) –

- Process are not perform or not achieve all the goals and objectives defined by the CMMI for Level I capability.

Level 1 (Performed) – All specific goals are performed as per defined

Level 2 (Managed) –

- All level 1 criteria have been satisfied
- In addition to Level I;
  - Stakeholders are actively involved,
  - Work tasks and products are monitored, controlled, reviewed, and evaluated .

Level 3 (Defined) –

- All level 2 criteria have been achieved.
- In addition;
  - management and engineering processes documented
  - standardized and integrated into organization-wide software process

CMMI Level (cont.)

Level 4 (Quantitatively Managed) -

- All level 3 criteria have been satisfied.
- Software process and products are quantitatively understood
- Controlled using detailed measures and assessment.

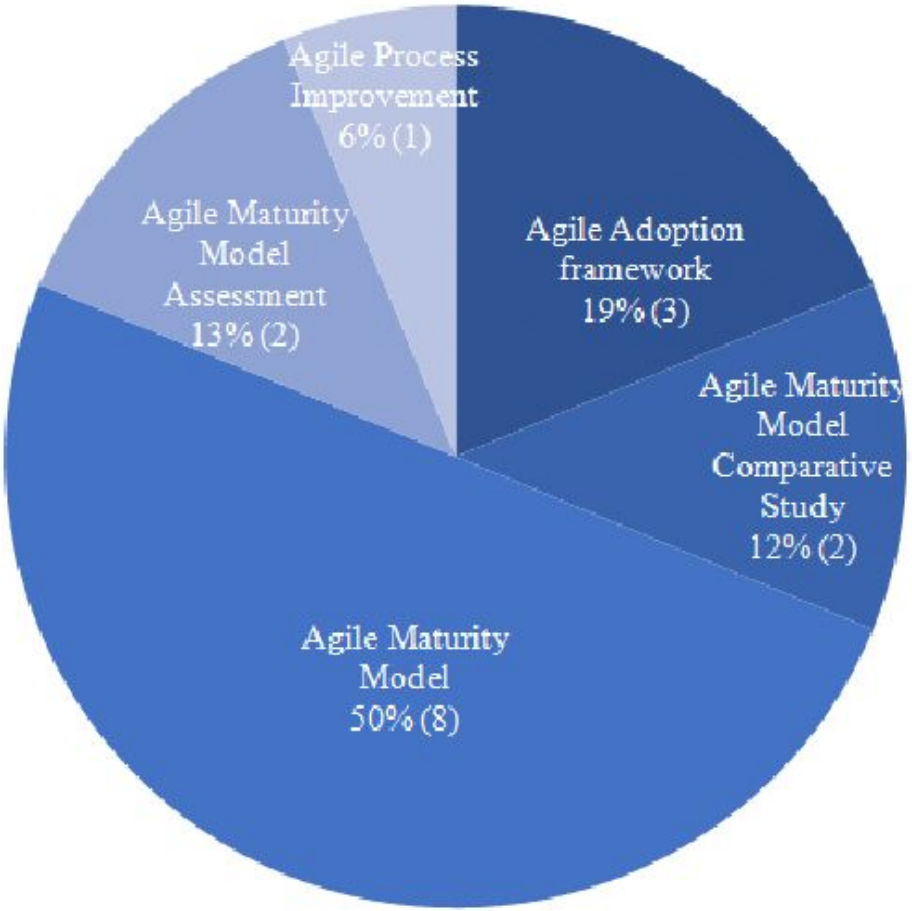
Level 5 (Optimized) –

- Continuous process improvement is enabled by quantitative feedback from the process and testing innovative ideas.

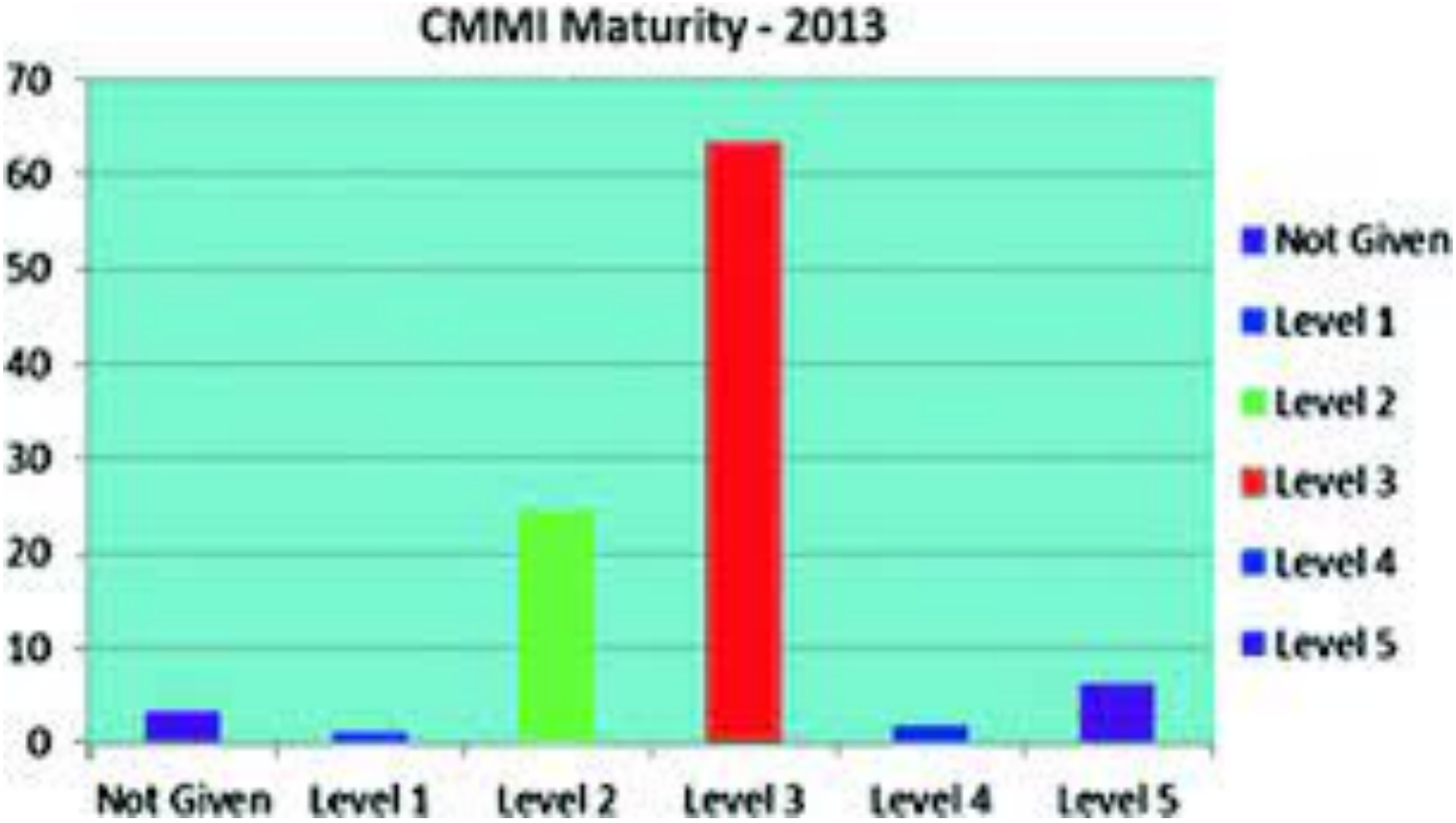
CMMI


Characteristics of the Maturity levels
Level 5
Optimizing
Focus on process improvement
Level 4
Quantitatively Managed
Processes measured and controlled
Level 3
Defined
Processes characterized for the organization and is proactive.
(Projects tailor their processes from organization's standards)
Level 2
Managed
Processes characterized for projects and is often reactive.
Level 1
Initial
Processes unpredictable, poorly controlled and reactive

Level	Focus	Process Areas
Optimizing	*Continuous process improvement*	Organizational Innovation and Deployment Causal Analysis and Resolution
Quantitatively managed	*Quantitative management*	Organizational Process Performance Quantitative Project Management
Defined	*Process standardization*	Requirements Development Technical Solution Product Integration Verification Validation Organizational Process Focus Organizational Process Definition Organizational Training Integrated Project Management Integrated Supplier Management Risk Management Decision Analysis and Resolution Organizational Environment for Integration Integrated Teaming
Managed	*Basic project management*	Requirements Management Project Planning Project Monitoring and Control Supplier Agreement Management Measurement and Analysis Process and Product Quality Assurance Configuration Management
Performed		


Agile Process Improvement 6% (1)
Agile Adoption framework 19% (3)
Agile Maturity Model Assessment 13% (2)
Agile Maturity Model Comparative Study 12% (2)
Agile Maturity Model 50% (8)

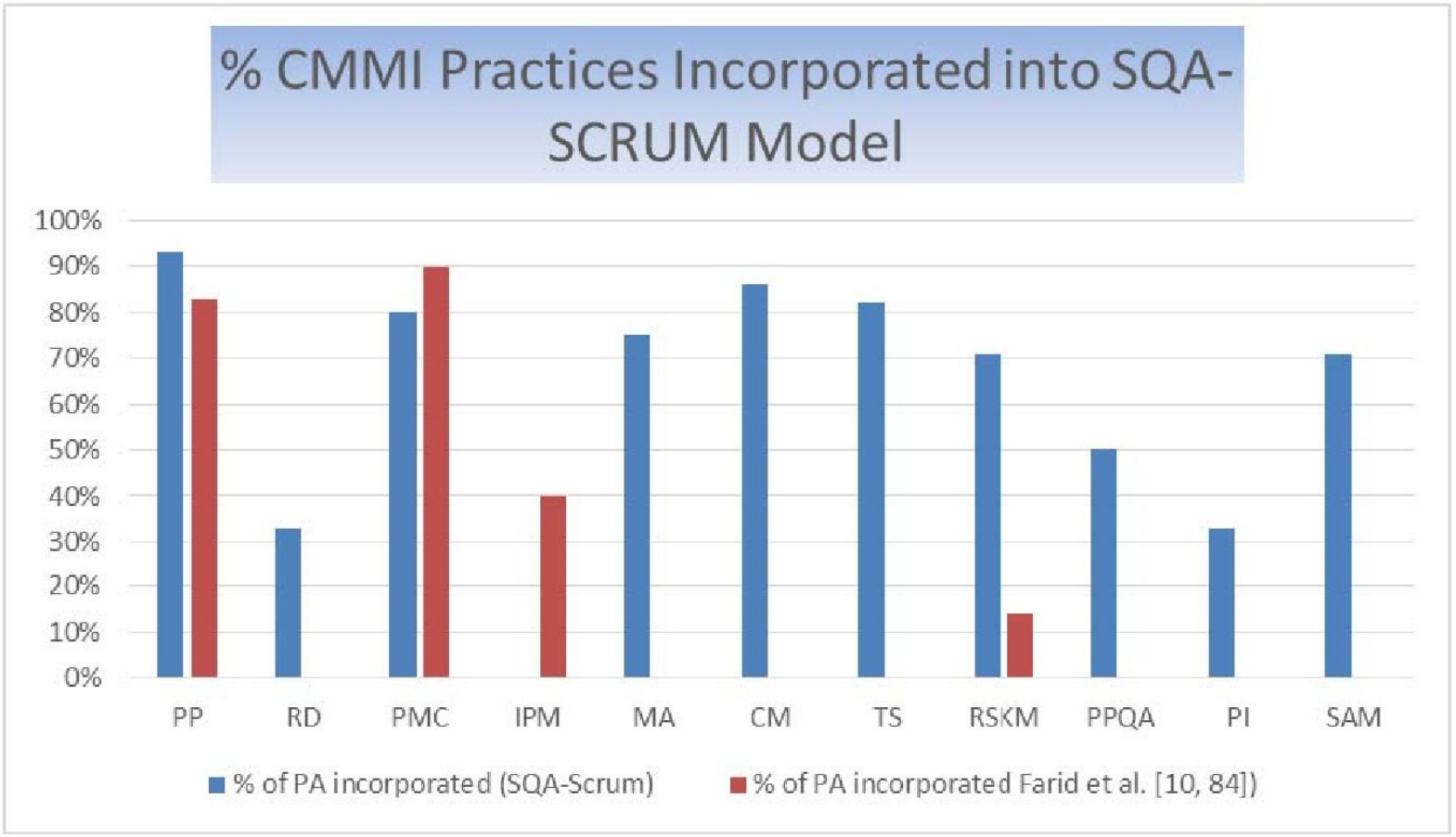
CMMI


CMMI Maturity - 2013
70
60
50
40
30
20
10
0
Not Given
Level 1
Level 2
Level 3
Level 4
Level 5
Not Given
Level 1
Level 2
Level 3
Level 4
Level 5

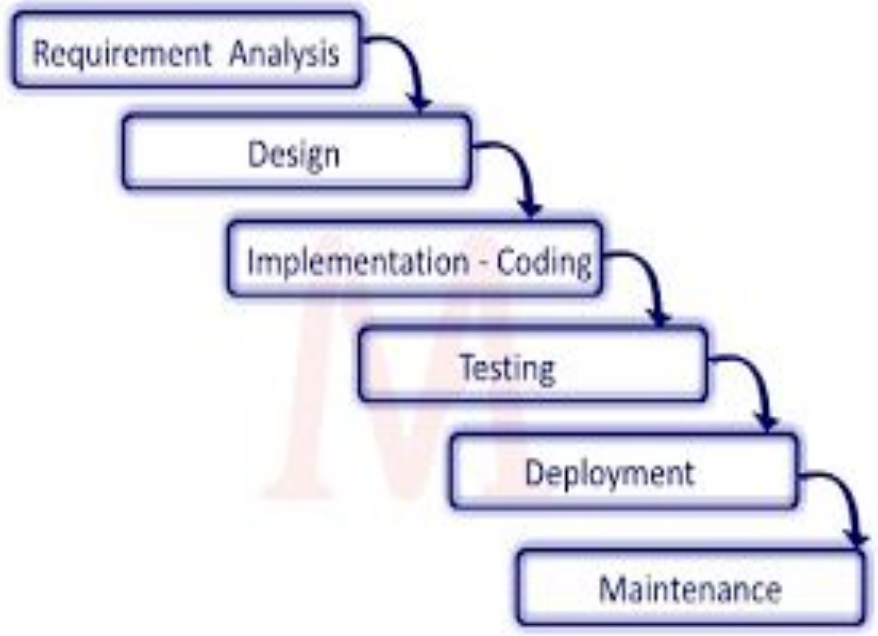
CMMI

15th April 2020. Vol.98. No 07

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195



Software Process



Software process model

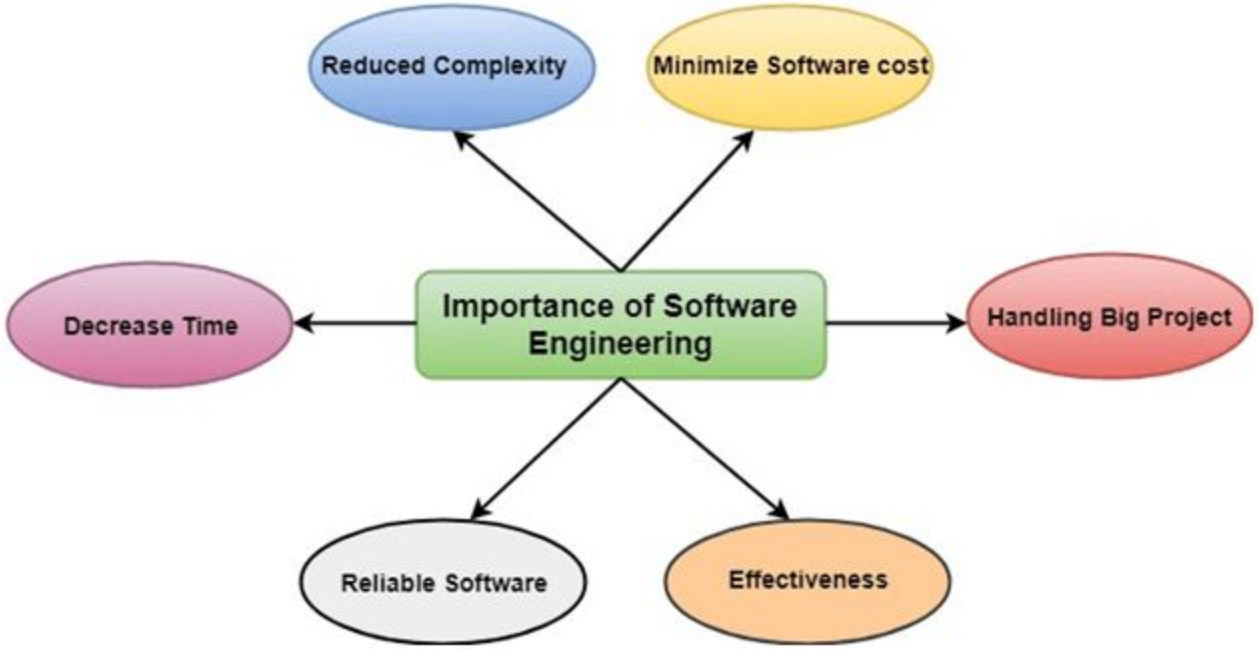
- Process models prescribe a distinct set of activities, actions, tasks, milestones, and work products required to engineer high quality software.
- Process models are not perfect, but provide roadmap for software engineering work.
- Software models provide stability, control, and organization to a process that if not managed can easily get out of control
- Software process models are developed to meet the needs of software engineers and managers for a specific project.

Software Process Models

- **Prescriptive Model**
- About software process model
- **Waterfall Model or Linear Sequential**
- **Incremental Process Models**
  - Incremental Model
  - RAD Model
- **Evolutionary Process Models**
  - Prototyping
  - Spiral Model
  - Concurrent Development Model
- **Unified process model**

Why Models are needed?

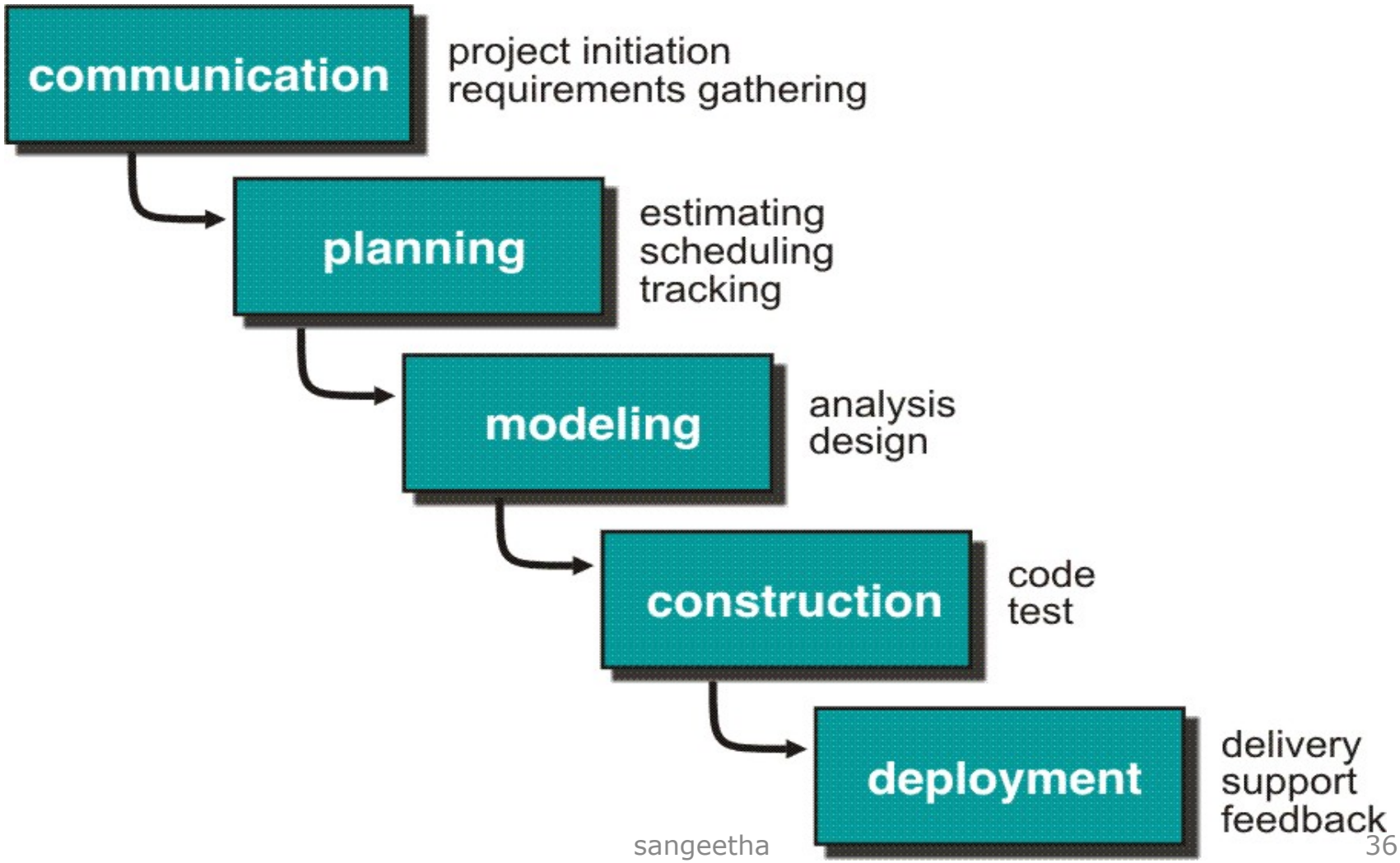
- Symptoms of inadequacy: the software crisis
  - scheduled time and cost exceeded
  - user expectations not met
  - poor quality



Prescriptive Model

- Prescriptive process models advocate an orderly approach to software engineering
  - Organize framework activities in a certain order
- Process framework activity with set of software engineering actions.
- Each action in terms of a task set that identifies the work to be accomplished to meet the goals.
- Software engineer choose process framework that includes activities like;
  - Communication
  - Planning
  - Modeling
  - Construction
  - Deployment

Waterfall Model or Classic Life Cycle



Waterfall Model or Classic Life Cycle

- Requirement Analysis and Definition: What - The systems services, constraints and goals are defined by customers with system users.
- Scheduling tracking -
  - Assessing progress against the project plan.
  - Require action to maintain schedule.
- System and Software Design: How –It establishes and overall system architecture. Software design involves fundamental system abstractions and their relationships.
- Integration and system testing: The individual program unit or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.
- Operation and Maintenance: Normally this is the longest phase of the software life cycle. The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life-cycle.

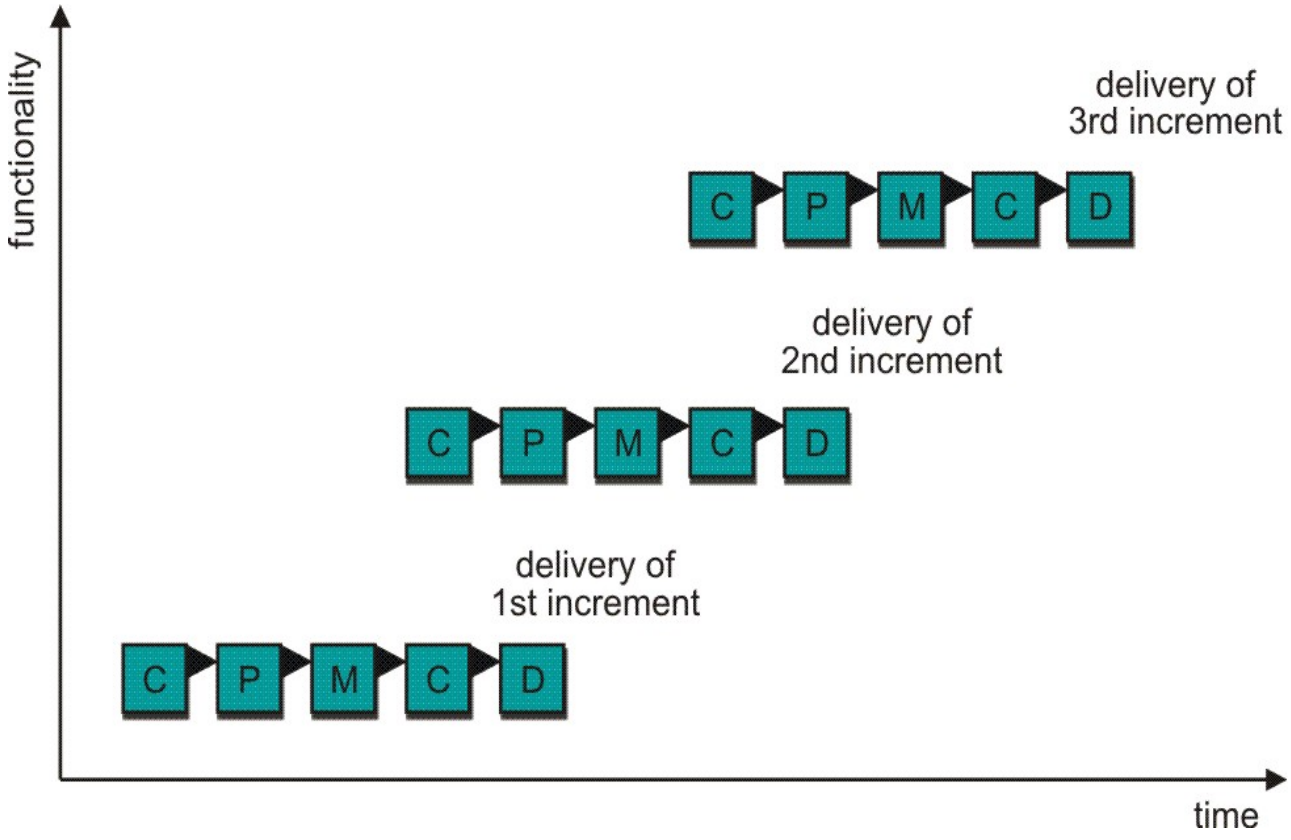
Limitations of the waterfall model

- The nature of the requirements will not change very much During development; during evolution
- The model implies that you should attempt to complete a given stage before moving on to the next stage
  - Does not account for the fact that requirements constantly change.
  - It also means that customers can not use anything until the entire system is complete.
- The model implies that once the product is finished, everything else is maintenance.
- Some teams sit ideal for other teams to finish
- Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.

Problems:

1. Real projects are rarely follow the sequential model.
2. Difficult for the customer to state all the requirement explicitly.
3. Assumes patience from customer - working version of program will not available until programs not getting change fully.

Incremental Process Model



C- Communication
P - Planning
M – Modeling
C - Construction
D - Deployment

Delivers software in small but usable pieces, each piece builds on pieces

The Incremental Model

- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.
- First Increment is often core product
  - Includes basic requirement
  - Many supplementary features (known & unknown) remain undelivered
- A plan of next increment is prepared
  - Modifications of the first increment
  - Additional features of the first increment
- It is particularly useful when enough staffing is not available for the whole project
- Increment can be planned to manage technical risks.
- Incremental model focus more on delivery of operation product with each increment.

The Incremental Model

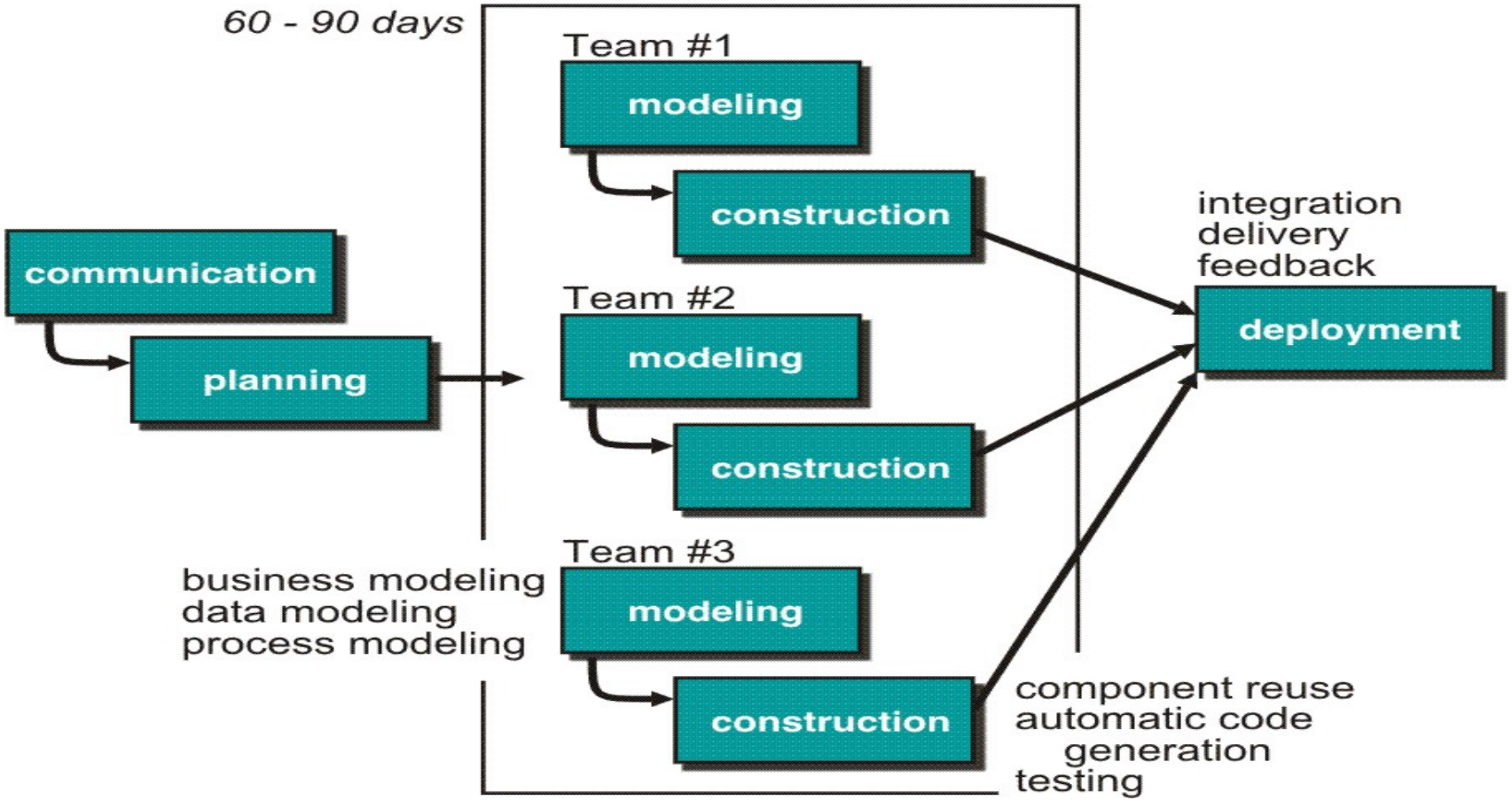
- User requirements are prioritised and the highest priority requirements are included in early increments.
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.
- Customer value can be delivered with each increment so system functionality is available earlier.
- Early increments act as a prototype to help elicit requirements for later increments.
- Lower risk of overall project failure.

For example word processing software developed using incremental model:

Basic file management editing and document production functions in the first increment.

More sophisticated document production capabilities in the second increment;spelling and grammer checking in the third increment;advanced page layout capability in the in the fourth increment.

Rapid Application Development (RAD) Model



Makes heavy use of reusable software

Rad Model:

- The Rad model emphasizes on the delivery of the projects in to small pieces.
- In case, if the project is large,it is divided into a series of smaller projects.each of these smaller projects are planned and delivered individually.therefore with a series of smaller projects,this model is delivered quickly.

RAD model

- **Communication** – to understand business problem.
- **Planning** – multiple s/w teams works in parallel on diff. system.
- **Modeling** –
  - **Business modeling** – Information flow among business is working.

  Ex. What kind of information drives?

  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** – Deliver to customer basis for subsequent iteration.
- RAD model emphasize a short development cycle.
- “High speed” edition of linear sequential model.
- If requirement are well understood and project scope is constrained then it enable development team to create “ fully functional system” within a very short time period.

RAD Model

- If application is modularized (“Scalable Scope”), each major function to be completed in less than three months.
- Each major function can be addressed by a separate team and then integrated to form a whole.

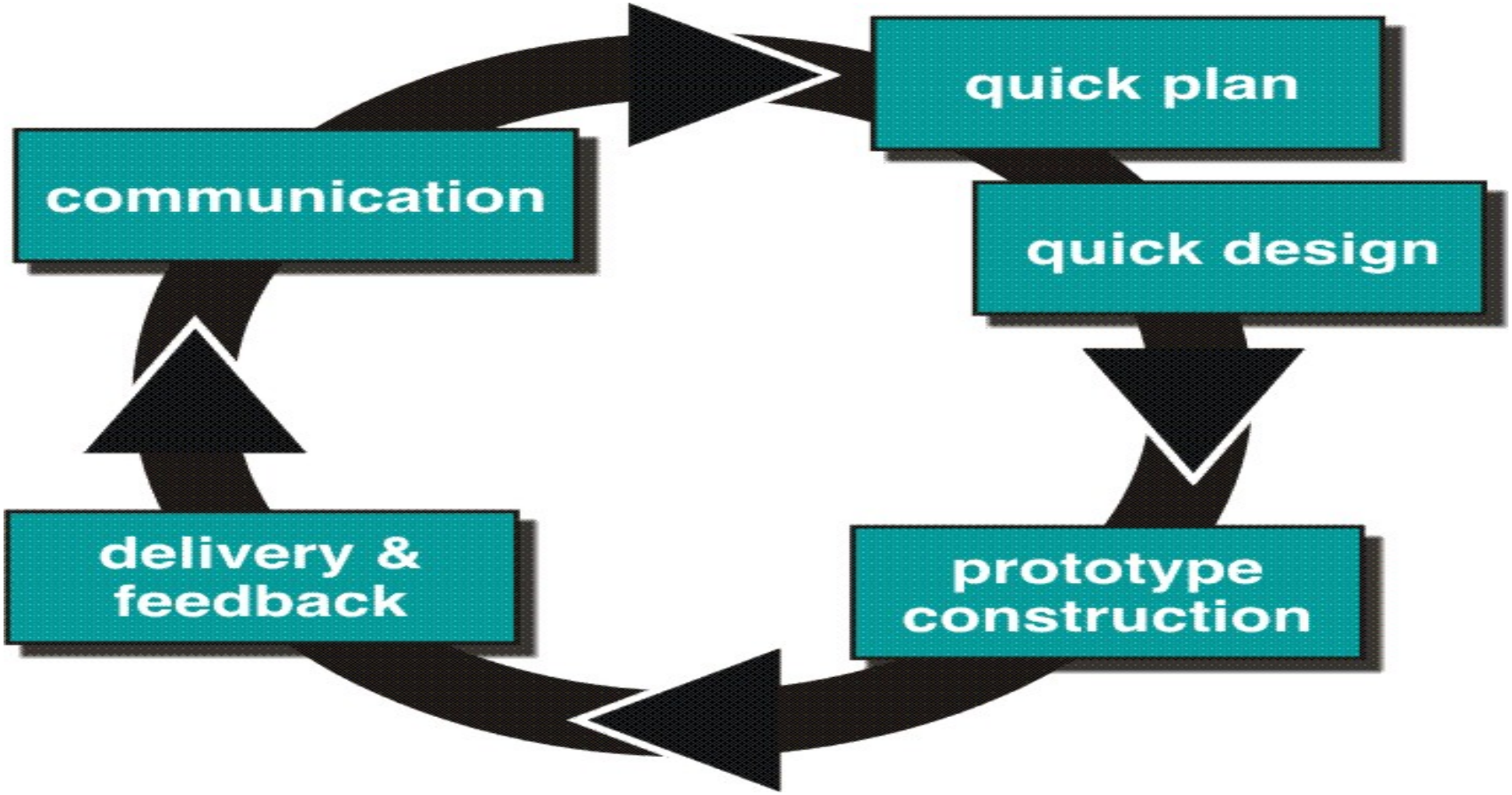
Drawback:

- 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)

Evolutionary Process Model

- Produce an increasingly more complete version of the software with each iteration.
- Evolutionary Models are iterative.
- Evolutionary models are:
  - Prototyping
  - Spiral Model
  - Concurrent Development Model

Evolutionary Process Models : Prototyping



Best approach when

- This model is applied when there is an absence of detailed information regarding input and output requirements in software.
- This model is also developed on the assumption that it is often difficult to know all the requirements at the beginning of a project.
- It is usually used when a system does not exist or there is no manual process to determine the requirements in case of a large and complex system.
- At any stage if the user is not satisfied with the prototype ,the model can be thrown away and an entirely new system can be developed.

- **Best approach when**:
  - Objectives defines by customer are general but does not have details like input, processing, or output requirement.
  - Developer may be unsure of the efficiency of an algorithm, O.S., or the form that human machine interaction should take.
- It can be used as standalone process model.
- Model assist software engineer and customer to better understand what is to be built when requirement are fuzzy.
- Prototyping start with communication, between a customer and software engineer to define overall objective, identify requirements and make a boundary.
- Going ahead, planned quickly and modeling (software layout visible to the customers/end-user) occurs.
- Quick design leads to prototype construction.
- Prototype is deployed and evaluated by the customer/user.
- Feedback from customer/end user will refine requirement and that is how iteration occurs during prototype to satisfy the needs of the customer.

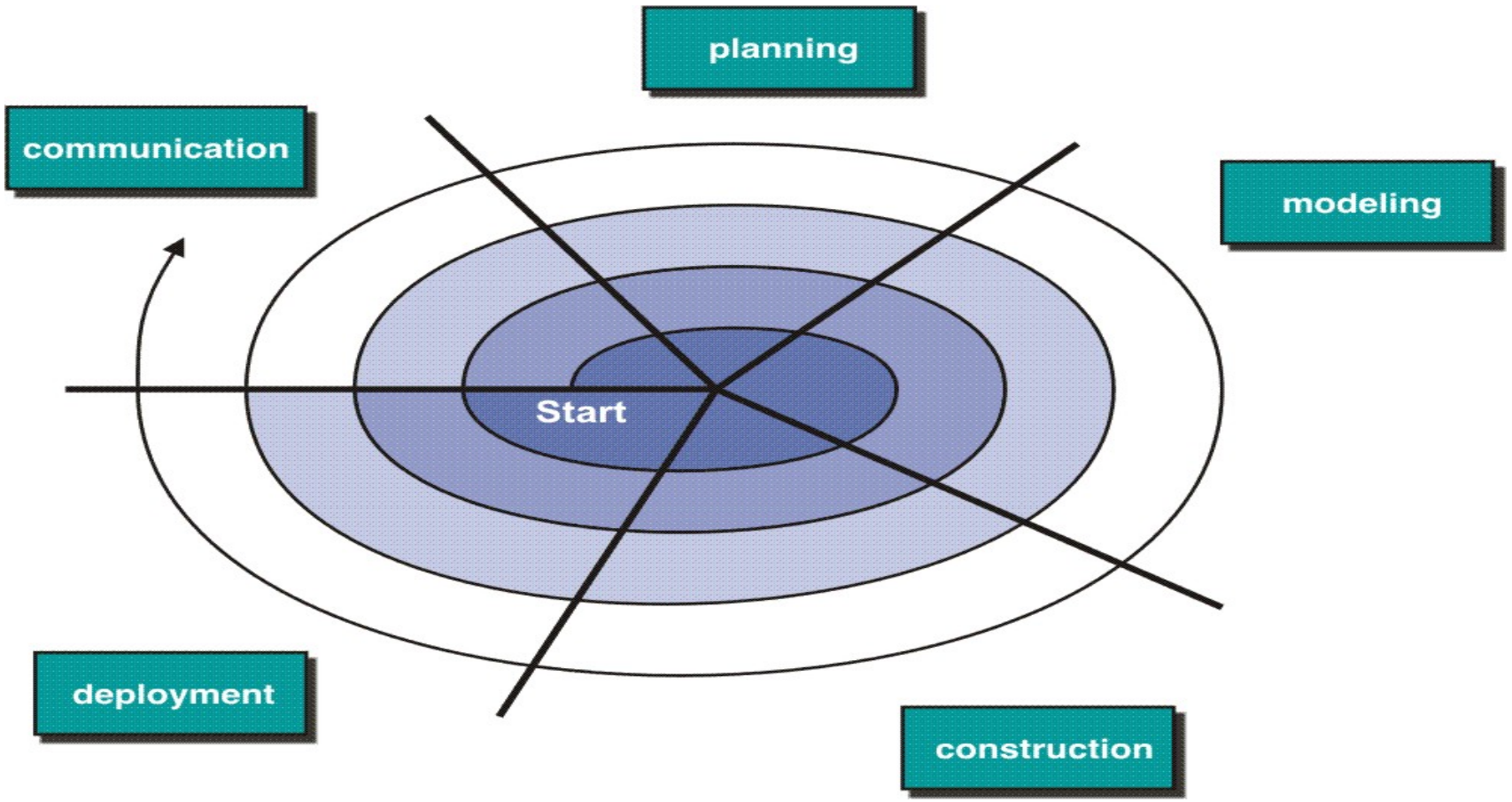
Prototyping (cont..)

- Both customers and developers like the prototyping paradigm.
  - Customer/End user gets a feel for the actual system
  - Developer get to build something immediately.

disadvantages:

- If the user is not satisfied by the developed prototype, then a new prototype is developed. this process goes on until a perfect prototype is developed. thus this model is time consuming and expensive.

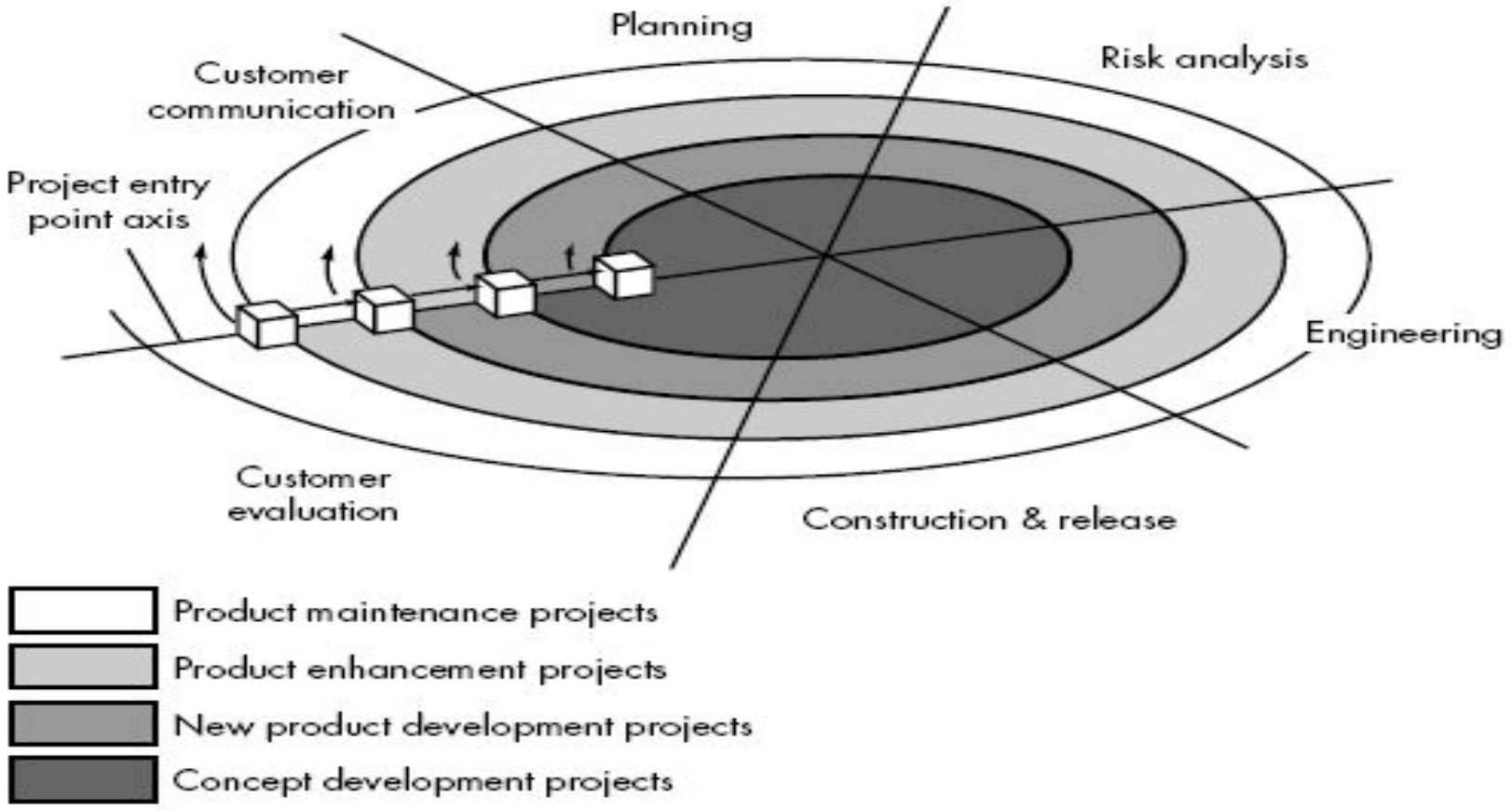
Evolutionary Model: Spiral Model



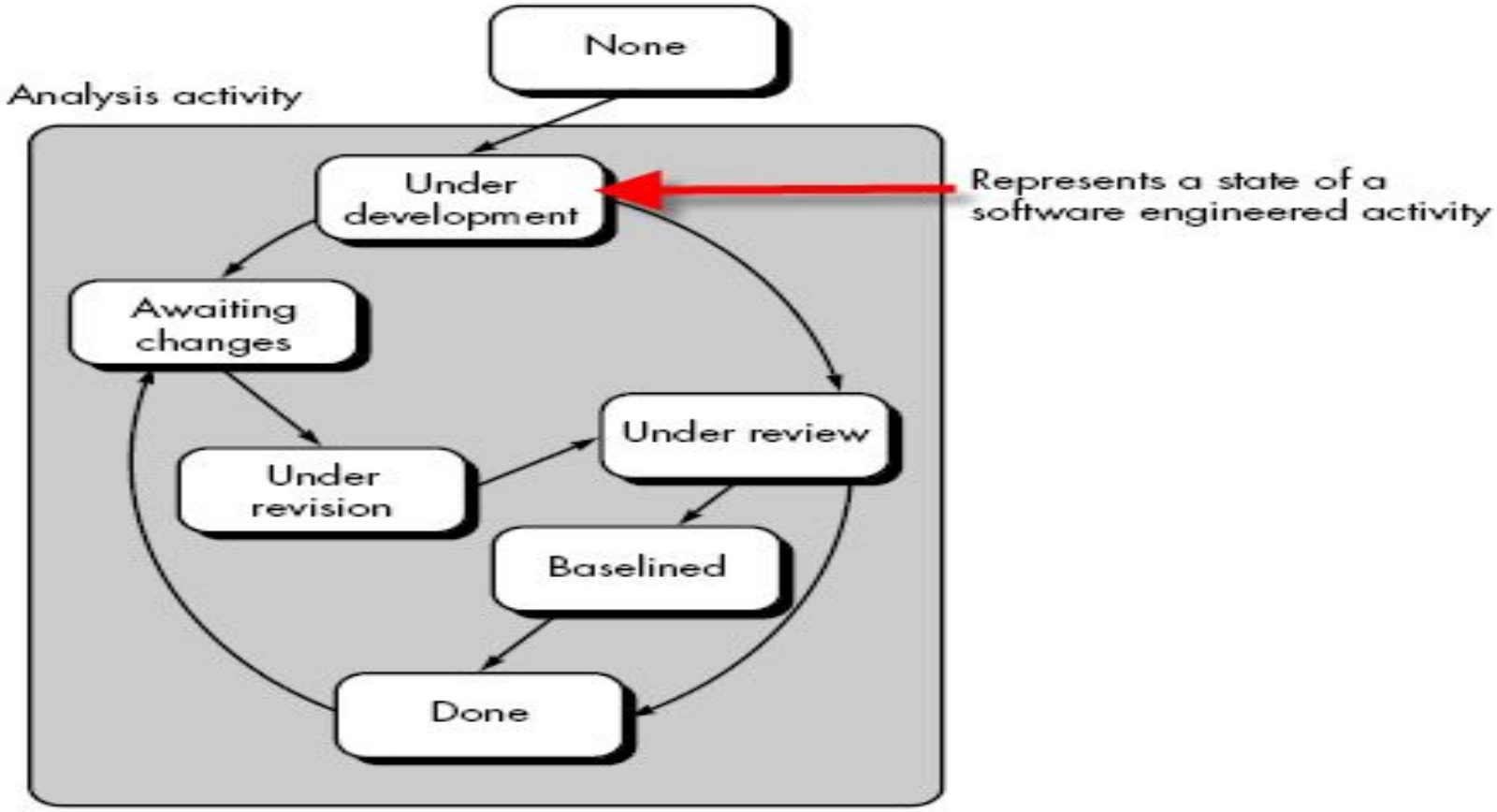
Spiral Model

- In 1980, a french mathematician introduces a process model known as the spiral model.the spiral model consist of activities which are organized in a spiral form that has several layers.
- This model combines the features of the prototype and classical waterfall model and it is beneficial for large,complex and expensive projects.
- The objective of the spiral model is to help the management in evaluating and resolving the risks which are involved in the software projects.
- The different areas of the risks in the software projects are project over runs, changing the requirements, delay of developing components.

Spiral Model



Concurrent Development Model



Concurrent Development Model

- It represented schematically as series of major technical activities, tasks, and their associated states.
- It is often more appropriate for system engineering projects where different engineering teams are involved.
- The activity-modeling may be in any one of the states for a given time.
- All activities exist concurrently but reside in different states.

E.g.

- The *analysis* activity (existed in the **none** state while initial customer communication was completed) now makes a transition into the **under development** state.
- *Analysis* activity moves from the **under development** state into the **awaiting changes** state only if customer indicates changes in requirements.
- Series of event will trigger transition from state to state.

E.g. During initial stage there was inconsistency in design which was uncovered. This will triggers the analysis action from the **Done** state into **Awaiting Changes** state.

Concurrent Development (Cont.)

- Visibility of current state of project
- It define network of activities
- Each activities, actions and tasks on the network exists simultaneously with other activities ,actions and tasks.

The unified process Model

- Background:

- Birthed during the late 1980's and early 1990s when object-oriented languages were gaining wide-spread use
- Many object-oriented analysis and design methods were proposed; three top authors were Grady Booch, Ivar Jacobson, and James Rumbaugh
- They eventually worked together on a unified method, called the Unified Modeling Language (UML)
  - UML is a robust notation for the modeling and development of object-oriented systems
  - UML became an industry standard in 1997
  - However, UML does not provide the process framework, only the necessary technology for object-oriented development

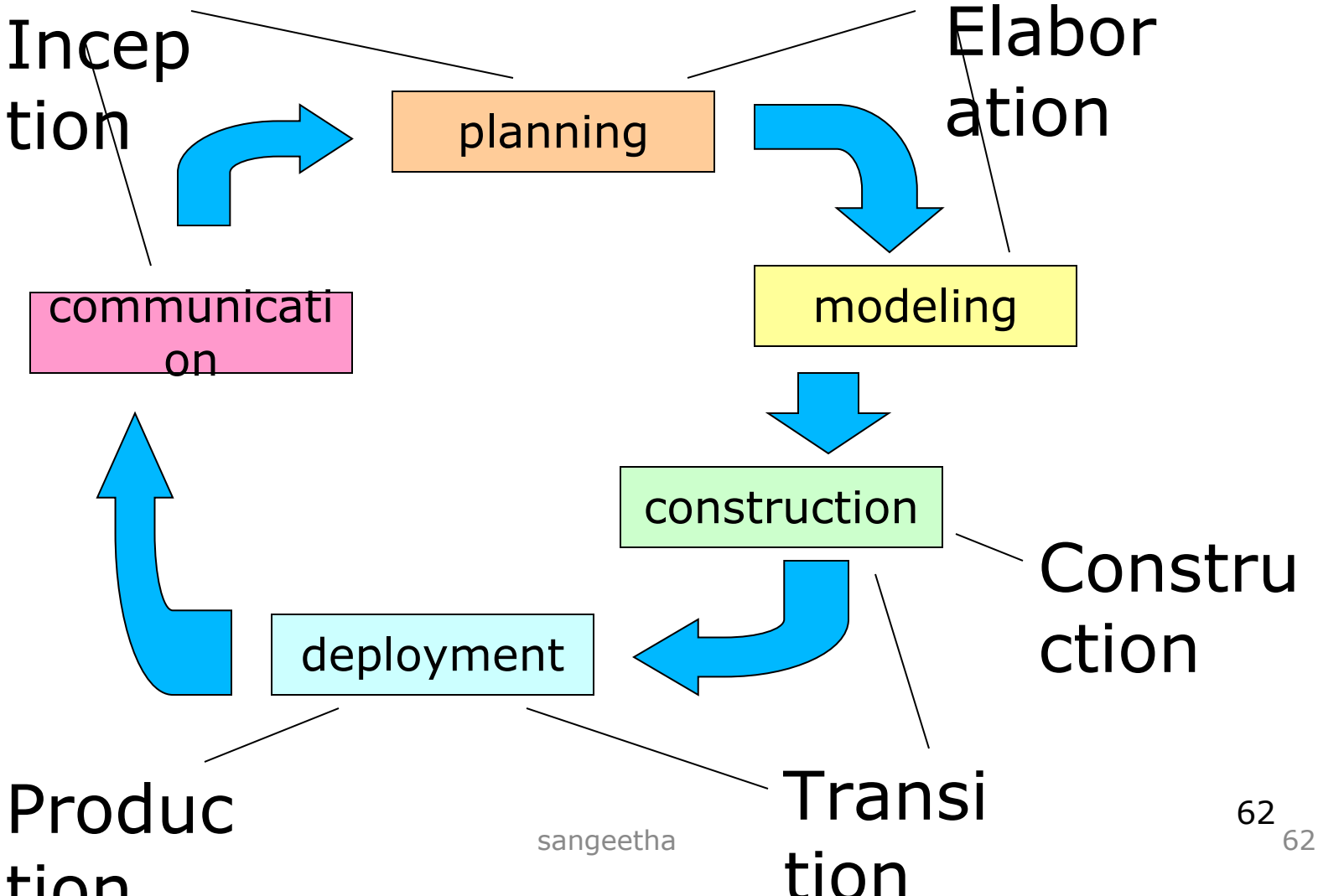
- Booch, Jacobson, and Rumbaugh later developed the unified process, which is a framework for object-oriented software engineering using UML
  - Draws on the best features and characteristics of conventional software process models
  - Emphasizes the important role of software architecture
  - Consists of a process flow that is iterative and incremental, thereby providing an evolutionary feel
- Consists of five phases: inception, elaboration, construction, transition, and production

GONNA & I'MMA

We shorten **going to** to **gonna**, but some people shorten it further, so that **I'm going to** becomes "**I'mma**".

I am going to take a break.
I'm going to take a break.
I'm **gonna** take a break.
I'mma take a break.

Phases of the Unified Process



Inception Phase

- Encompasses both customer communication and planning activities of the generic process
- Business requirements for the software are identified
- A rough architecture for the system is proposed
- A plan is created for an incremental, iterative development
- Fundamental business requirements are described through preliminary use cases
  - A use case describes a sequence of actions that are performed by a user

Elaboration Phase

- Encompasses both the planning and modeling activities of the generic process
- Refines and expands the preliminary use cases
- Expands the architectural representation to include five views
  - Use-case model
  - Analysis model
  - Design model
  - Implementation model
  - Deployment model
- Often results in an executable architectural baseline that represents a first cut executable system

Construction Phase

- Encompasses the construction activity of the generic process
- Uses the architectural model from the elaboration phase as input
- Develops or acquires the software components that make each use-case operational
- Analysis and design models from the previous phase are completed to reflect the final version of the increment
- Use cases are used to derive a set of acceptance tests that are executed prior to the next phase

Transition Phase

- Encompasses the last part of the construction activity and the first part of the deployment activity of the generic process
- Software is given to end users for beta testing and user feedback reports on defects and necessary changes
- The software teams create necessary support documentation (user manuals, trouble-shooting guides, installation procedures)
- At the conclusion of this phase, the software increment becomes a usable software release

Production Phase

- Encompasses the last part of the deployment activity of the generic process
- On-going use of the software is monitored
- Support for the operating environment (infrastructure) is provided
- Defect reports and requests for changes are submitted and evaluated

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

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 not 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.**

- **Competence:** In an agile development (as well as software engineering) context, "competence" encompasses innovate talent, specific software-related skills, and overall knowledge of the process. Skill and knowledge of process can and should be taught to all people who serve as agile team members.
- **Common focus:** Although members of the agile team may perform different tasks and bring different skills to the project, all should be focused on one goal—to deliver a working software increment to the customer within the time promised.
- **Collaboration:** Software engineering (regardless of process) is about assessing, analysing, and using information that is communicated to the software team; creating information that will help all stakeholders understand the work of the team; and building information (computer software and relevant databases) that provides business value for the customer. To accomplish these tasks, team members must collaborate—with one another and all other stakeholders.

- **Decision-making ability.** Any good software team (including agile teams) must be allowed the freedom to control its own destiny. This implies that the team is given autonomy—decision-making authority for both technical and project issues.
- **Fuzzy problem-solving ability**. Software managers must recognize that the agile team will continually have to deal with ambiguity problems. In some cases, the team must accept the fact that the problem they are solving today may not be the problem that needs to be solved tomorrow. However, lessons learned from any problem-solving activity (including those that solve the wrong problem) may be of benefit to the team later in the project.
- **Mutual trust and respect**. The agile team exhibits the trust and respect that are necessary to make the build software so strongly.

Agile Process Models

- Extreme Programming (XP)
- Adaptive Software Development(ASD)
- Dynamic Systems Development Method
- SCRUM
- Crystal
- Feature Driven Development
- Agile Modelling

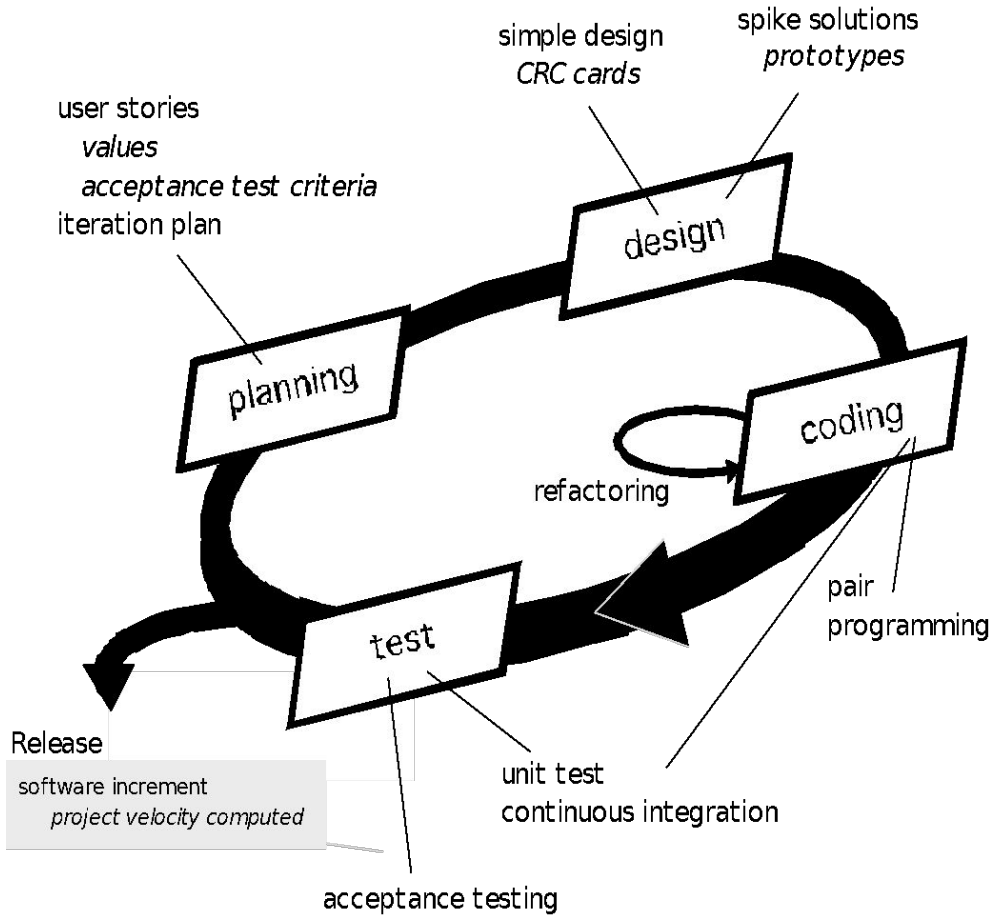
Extreme Programming (XP)

- 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 the team defines subsequent delivery dates for other increments.

Extreme Programming (XP)

- XP Design
  - Follows the KIS principle
  - Encourage the use of CRC cards (see Chapter 8)
  - 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".(**Pair programming** is an agile software development technique in which two **programmers** work together at one workstation. One, the driver, writes code while the other, the observer or navigator, reviews each line of code as it is typed in. The two **programmers** switch roles frequently.)
- XP Testing
  - All unit tests are executed daily
  - "Acceptance tests" are defined by the customer and excuted to assess customer visible functionality

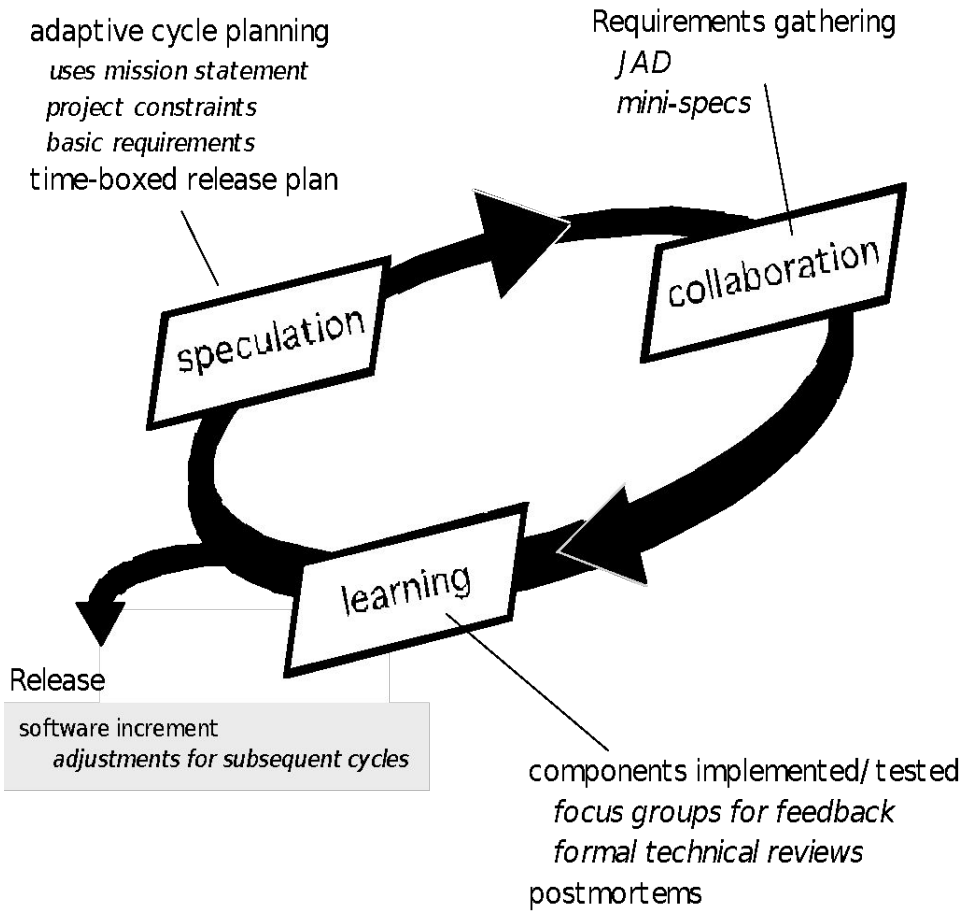
Extreme Programming (XP)



Adaptive Software Development

- Originally proposed by Jim Highsmith
- ASD — distinguishing features
  - Mission-driven speculation(planning)
  - Component-based focus
  - Emphasizes collaboration for requirements gathering
  - Emphasizes “learning” throughout the process

Adaptive Software Development



speculation

- During *speculation,* the project is initiated and *adaptive cycle planning* is conducted.
- Adaptive cycle planning uses project initiation information—the customer's mission statement, project constraints (e.g., delivery dates or user descriptions), and basic requirements—to define the set of release cycles (software increments) that will be required for the project.

collaboration

- Motivated people use *collaboration* in a way that It encompasses communication and teamwork, but it also emphasizes individualism, because individual creativity plays an important role in collaborative thinking. It is, above all, a matter of trust. People working together must trust one another to

(1) criticize without animosity,

(2) work as hard as or harder than they do,

(3) communicate problems or concerns in a way that leads to effective action.

learning

- ASD teams learn in three ways:
- Focus Groups
- Formal Technical Reviews
- Postmortems

Dynamic Systems Development Method

- Promoted by the DSDM Consortium (www.dsdm.org)
- The DSDM philosophy is borrowed from a modified version of the Pareto principle—80 percent of an application can be delivered in 20 percent of the time it would take to deliver the complete (100 percent) application.
- The DSDM Consortium has defined an agile process model, called the *DSDM life cycle that defines three different* iterative cycles, preceded by two additional life cycle activities

- *Feasibility study*—establishes the basic business requirements and constraints associated with the application to be built and then assesses whether the application is a viable candidate for the DSDM process.
- *Business study*—establishes the functional and information requirements that will allow the application to provide business value; also, defines the basic application architecture and identifies the maintainability requirements for the application.
- *Functional model iteration*—produces a set of incremental prototypes that demonstrate functionality for the customer

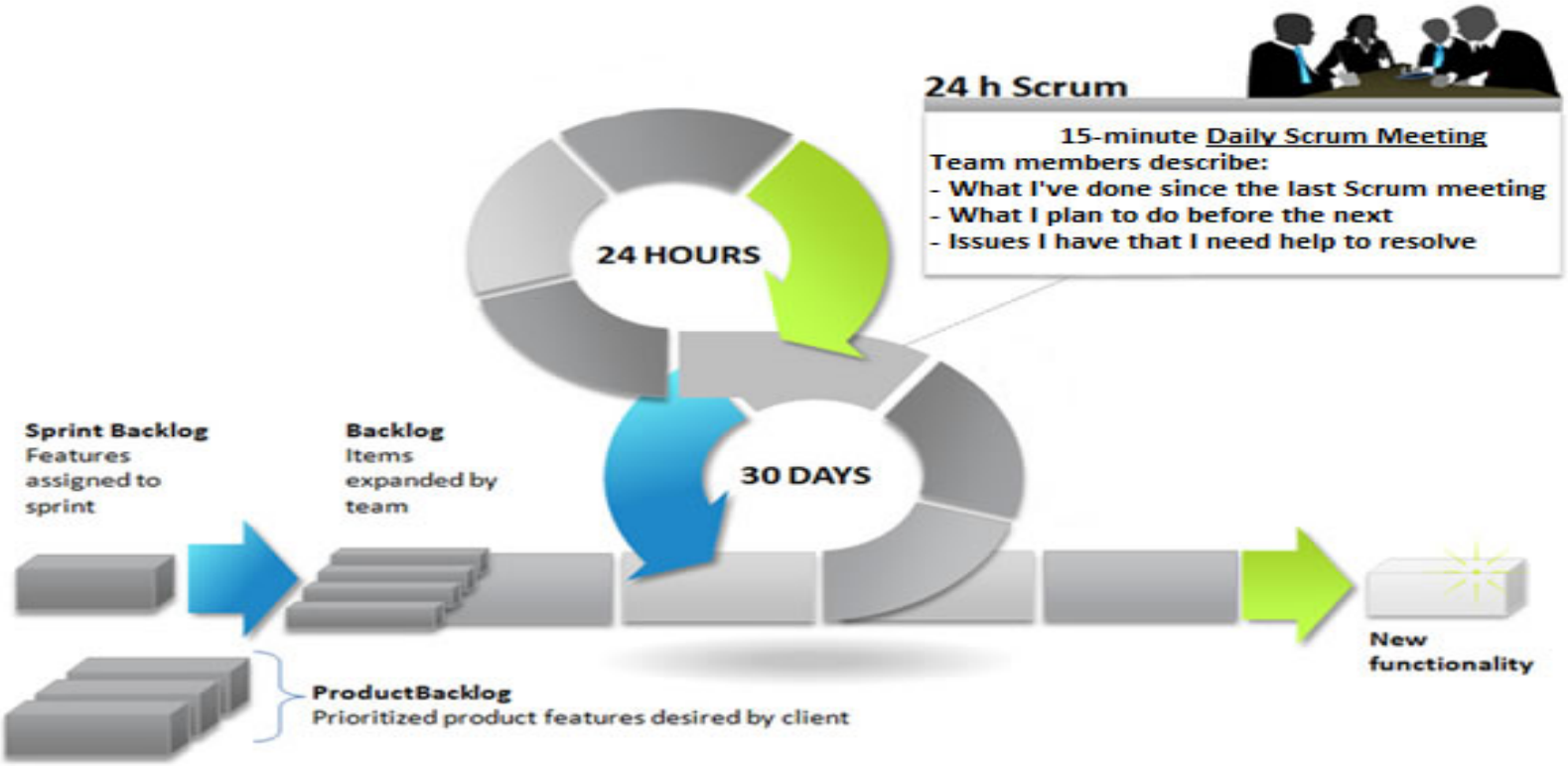
- *Design and build iteration*—revisits prototypes built during *functional model iteration* to ensure that each has been engineered in a manner that will enable it to provide operational business value for end users.
- *Implementation*—places the latest software increment into the operational environment. It should be noted that (1) the increment may not be 100 percent complete or (2) changes may be requested as the increment is put into place. In either case, DSDM development work continues by returning to the functional model iteration activity.

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

SCRUM PROCESS



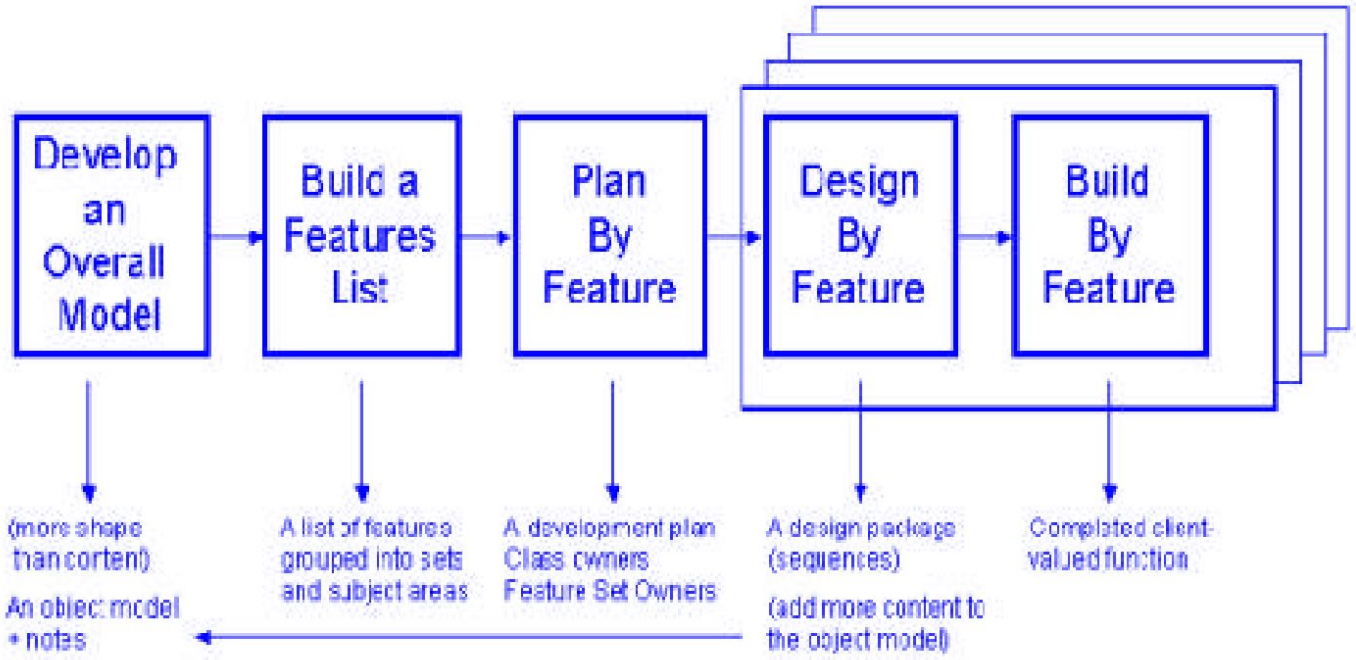
Crystal

- Proposed by Cockburn and Highsmith
- Crystal—distinguishing features
  - Suggests the use of "reflection workshops" to review the work habits of the team.
  - Face-to-face communication is emphasized

Feature Driven Development

- Originally proposed by Peter Coad et al
- FDD—distinguishing features
  - Emphasis is on defining "features"
    - a *feature* "is a client-valued function that can be implemented in two weeks or less."
  - Uses a feature template
    - <action> the <result> <by | for | of | to> a(n) <object>
  - A features list is created and "plan by feature" is conducted
  - Design and construction merge in FDD

Feature Driven Development



Reprinted with permission of Peter Coad

Agile Modeling

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

- **Model with a purpose.** A developer who uses AM should have a specific goal (e.g., to communicate information to the customer or to help better understand some aspect of the software) in mind before creating the model.
- **Use multiple models.** There are many different models and notations that can be used to describe software. Only a small subset is essential for most projects.

- **Content is more important than representation.** Modeling should impart information to its intended audience. A syntactically perfect model that imparts little useful content is not as valuable as a model with flawed notation that nevertheless provides valuable content for its audience.
- **Know the models and the tools you use to create them.** Understand the strengths and weaknesses of each model and the tools that are used to create it.
- **Adapt locally.** The modeling approach should be adapted to the needs of the agile team.