Software Engineering: A Practitioner's Approach, 6/e

Dr. Kiran Bhowmick

Unit 1:

Introduction to Software Engineering and Process Models:

Nature of Software, Software Engineering, Software Process, CMM, Generic Process Model.

Prescriptive Process Models: The Waterfall Model, V Model.

Incremental Process Model: Incremental Model

Evolutionary Process Models: Prototyping Paradigm, The Spiral Model

Concurrent Process Models: Concurrent Process Model

The Unified Process

Agile Methodology: Agility Principals, Agile Process Models: Extreme Programming (XP), Adaptive Software Development (ASD), Dynamic Systems Development Method (DSDM), Scrum, Crystal, Feature Driven Development (FDD), Agile Modeling (AM), Kanban Model

Chapter 2 2.1, 2.2, 2.3,
Chapter 3 (except 3.5 everything else)
Chapter 4; 4.1 and 4.3

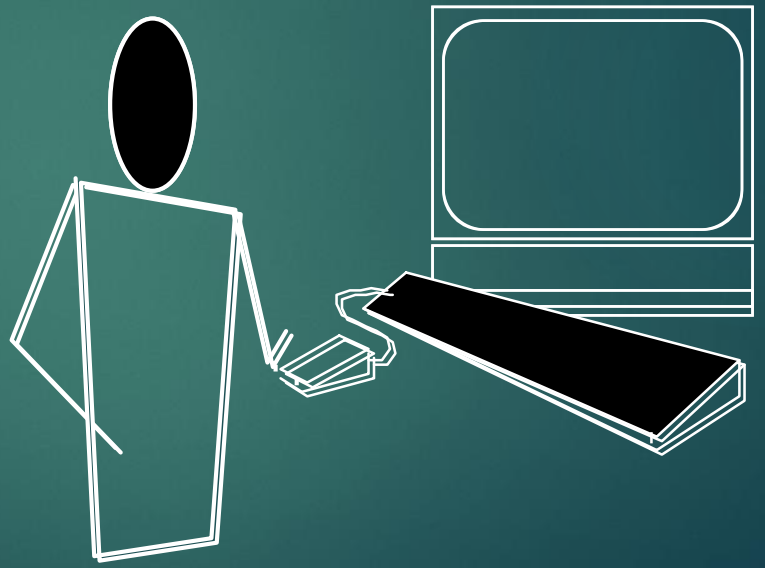
Software's Dual Role

- Software is a product
  - Delivers computing potential
  - Information transformer - Produces, manages, acquires, modifies, displays, or transmits information
- 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)

What is Software?

Software is a set of items or objects that form a “configuration” that includes

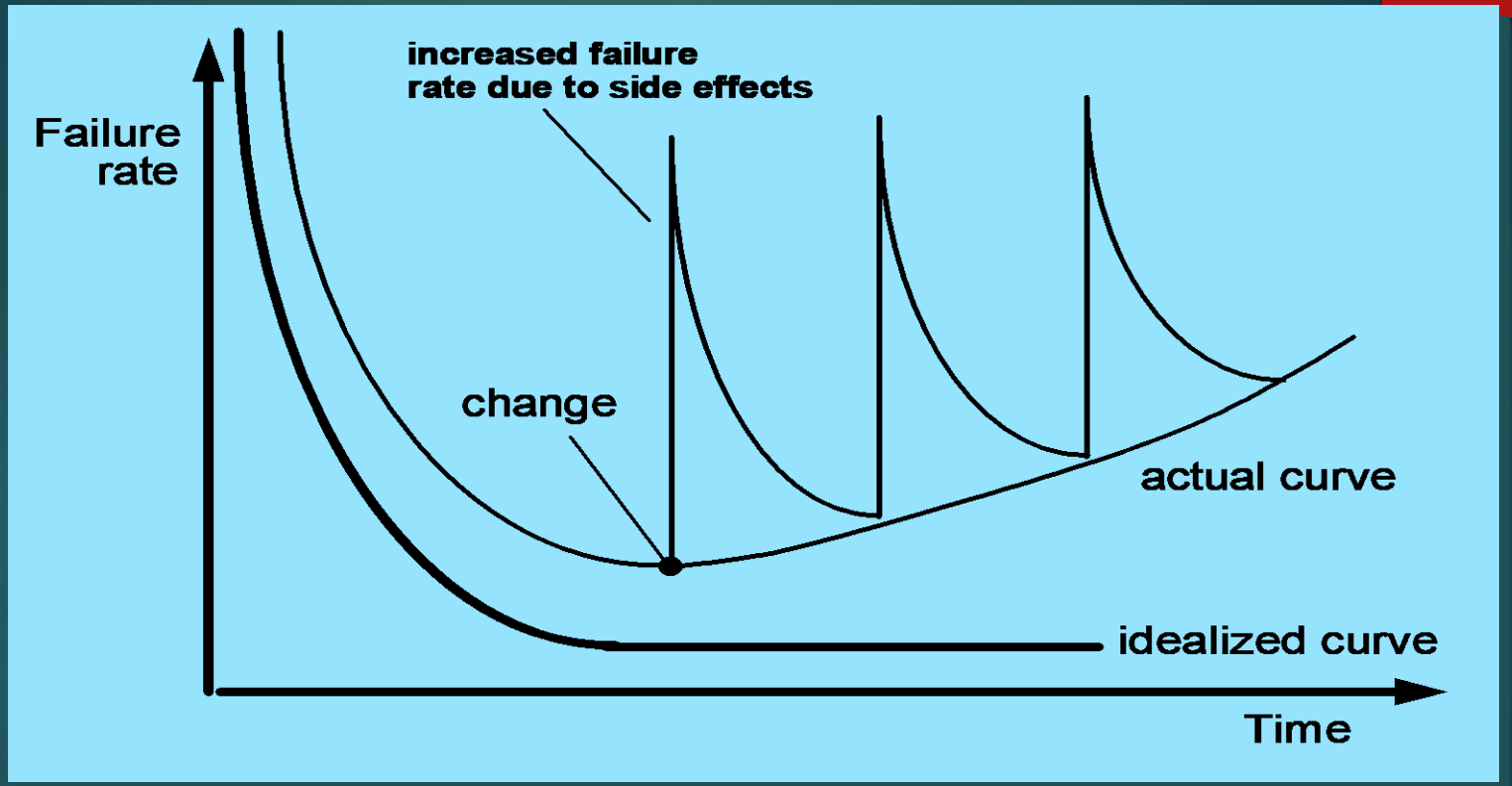
- programs
- data
- documents



What is Software?

- software is engineered or developed
- software doesn't wear out
- software is custom built

Wear vs. Deterioration



Nature of Software

- system software – compilers, editors, OS components, drivers
- application software – real time manufacturing process control, data processing applications
- engineering/scientific software – astronomy to volcanology
- embedded software – control in microwave
- product-line software – word processing, spreadsheets
- Web applications – WebApps, e-commerce and B2B applications
- AI software – pattern recognition, robotics

Software—New Categories

- Ubiquitous computing – wireless networks
- Netsourcing – personal financial planning
- Open source – "free" source code open to the computing community
- Also
  - Data mining
  - Grid computing
  - Cognitive machines
  - Software for nanotechnologies

Legacy Software

Why must it change?

- software must be adapted to meet the needs of new computing environments or technology.
- software must be enhanced to implement new business requirements.
- software must be extended to make it interoperable with other more modern systems or databases.
- software must be re-architected to make it viable within a network environment.

- **Goal of software engineering: devise methodologies that are founded on the notion of evolution.**
- **Systems should continually change ….. Interoperate and cooperate with each other.**

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

Software Myths

- Management Myths
  - We already have a book of standards and procedures for building software, that will provide my people with everything they need to know
  - If we get behind schedule, we can add more programmers and catch up
  - If I decide to outsource the software project to a third party I can just relax and let that firm build it.

Software Myths

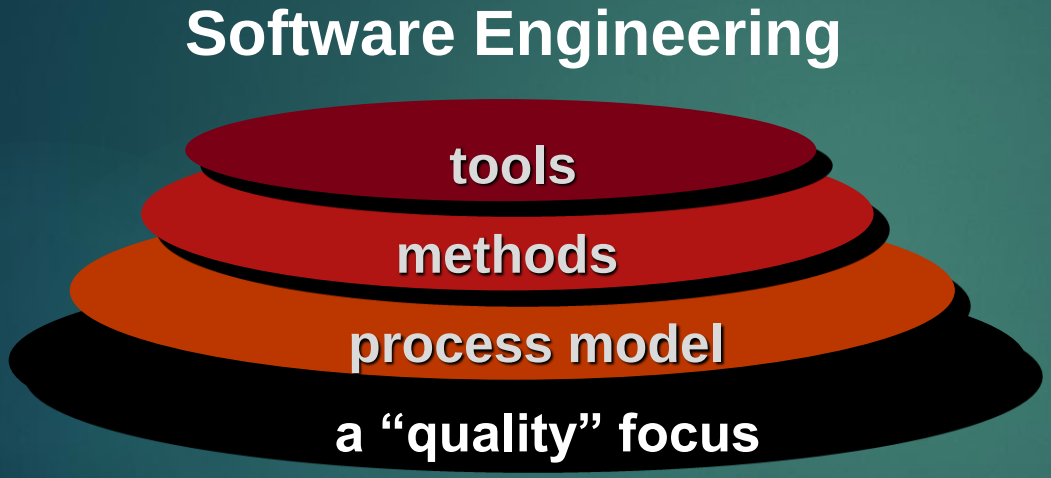
- Customer Myths
  - A general statement of objectives is sufficient to begin writing programs – we can fill in the details later
  - Project requirements continually change but change can easily be accommodated because software is flexible
- Practitioners Myths
  - Once we write the program and get it to work, our job is done.
  - Until I get the program running, I have no way of assessing its quality
  - The only deliverable work product for a successful projects is the working program
  - Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down

Definitions

- 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 ……Fritz Bauer

- Software Engineering is application of systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software ; that is , the application of engineering to the software. It is also study of approaches…. IEEE

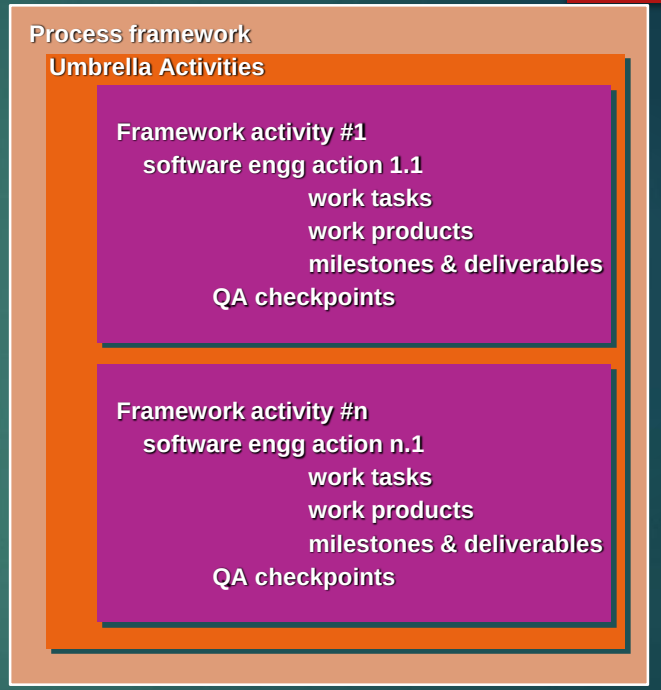
A Layered Technology



- Process defines framework
- Process forms basis for management control
- Process establishes context for methods to be applied, Work product produced, milestones, quality ensured, Change Managed
- Method provide Technical “How To”
- Methods include tasks: communication, req. analysis, design modeling, program construction, testing and support.
- Tools provide support for methods & processes

A Process Framework

- Framework establishes the foundation for a complete software process
- It identifies a small number of framework activities that are applicable to all software projects
- It encompasses a set of umbrella activities that are applicable across the entire software



A Process Framework

Framework Activities

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

Umbrella Activities

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

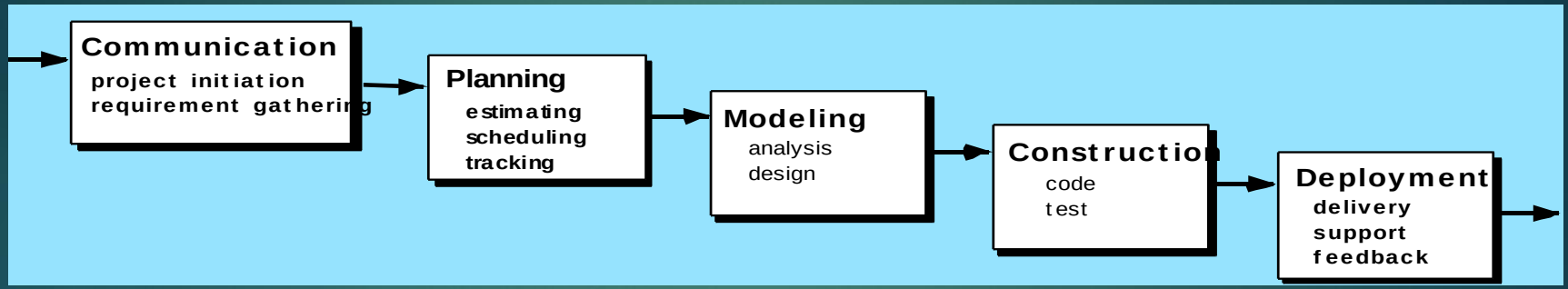
The Process Model: Adaptability

- the framework activities will always be applied on every project … BUT
- the tasks (and degree of rigor) for each activity will vary based on:
  - the type of project
  - characteristics of the project
  - common sense judgment; concurrence of the project team

Paradigms of software engineering process

- Prescriptive Models
  - Waterfall model
- Incremental Process Models
  - The incremental model
  - The RAD model
- Evolutionary Process models
  - Prototyping model
  - Spiral model
  - Concurrent Development model
- Specialized Process models
  - Component based model
  - Formal methods model
  - Aspect oriented s/w development model
- The Unified process model
- Agile process models
  - XP
  - Adaptive software development
  - Dynamic systems development method
  - Scrum
  - Crystal
  - Feature Driven Development

The Waterfall Model



Advantages of waterfall model:

- This model is simple and easy to understand and use.
- It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- In this model, phases are processed and completed one at a time. Phases do not overlap.
- Waterfall model works well for smaller projects where requirements are very well understood.

Disadvantages of waterfall model:

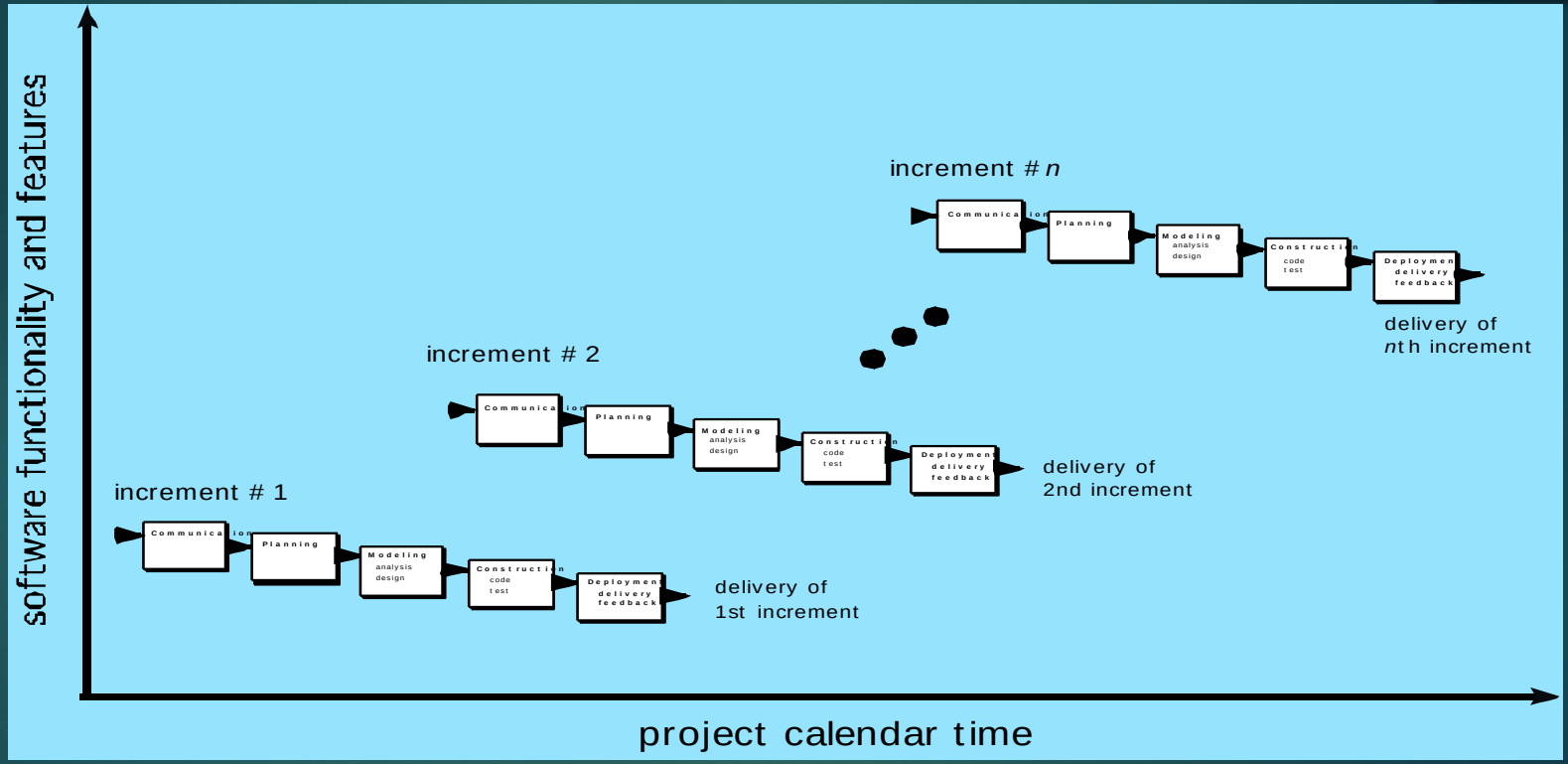
- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.
- Not a good model for complex and object-oriented projects.
- No working software is produced until late during the life cycle.
- Poor model for long and ongoing projects.
- High amounts of risk and uncertainty.

- When to use the waterfall model:
  - This model is used only when the requirements are very well known, clear and fixed.
  - Product definition is stable.
  - Technology is understood.
  - There are no ambiguous requirements
  - Ample resources with required expertise are available freely
  - The project is short.

Paradigms of software engineering process

- Prescriptive Models
  - Waterfall model
- Incremental Process Models
  - The incremental model
  - The RAD model
- Evolutionary Process models
  - Prototyping model
  - Spiral model
  - Concurrent Development model
- Specialized Process models
  - Component based model
  - Formal methods model
  - Aspect oriented s/w development model
- The Unified process model
- Agile process models
  - XP
  - Adaptive software development
  - Dynamic systems development method
  - Scrum
  - Crystal
  - Feature Driven Development

The Incremental Model

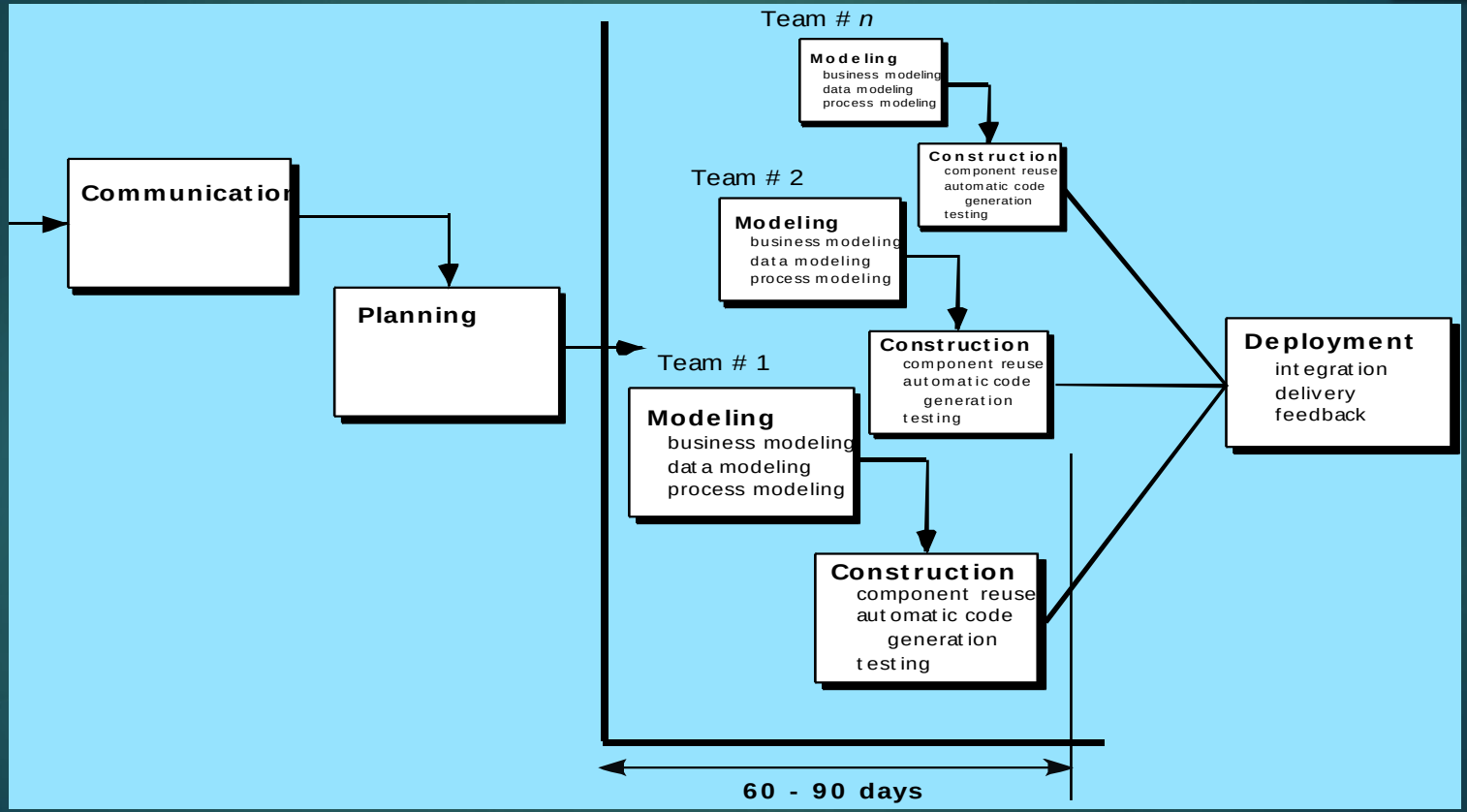


- **Advantages of Incremental model:**
  - Generates working software quickly and early during the software life cycle.
  - This model is more flexible – less costly to change scope and requirements.
  - It is easier to test and debug during a smaller iteration.
  - In this model customer can respond to each built.
  - Lowers initial delivery cost.
  - Easier to manage risk because risky pieces are identified and handled during iteration.
  - Implemented when staffing is unavailable.
  - Increments can be planned to manage technical risks

- **Disadvantages of Incremental model:**
  - Needs good planning and design.
  - Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
  - Total cost is higher than waterfall.

- **When to use the Incremental model:**
  - This model can be used when the requirements of the complete system are clearly defined and understood.
  - Major requirements must be defined; however, some details can evolve with time.
  - There is a need to get a product to the market early.
  - A new technology is being used
  - Resources with needed skill set are not available
  - There are some high risk features and goals.

The RAD Model



- **Advantages of the RAD model:**
  - Reduced development time.
  - Increases reusability of components
  - Quick initial reviews occur
  - Encourages customer feedback
  - Integration from very beginning solves a lot of **integration issues**.

- **Disadvantages of RAD model:**
  - Depends on strong team and individual performances for identifying business requirements.
  - Only system that can be modularized can be built using RAD
  - Requires highly skilled developers/designers.
  - High dependency on modeling skills
  - Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.
  - Not applicable for large scalable projects due to requirement of sufficient human resources
  - Developers and customers must be committed to the rapid development, else RAD project fails
  - Not applicable when technology risks are high

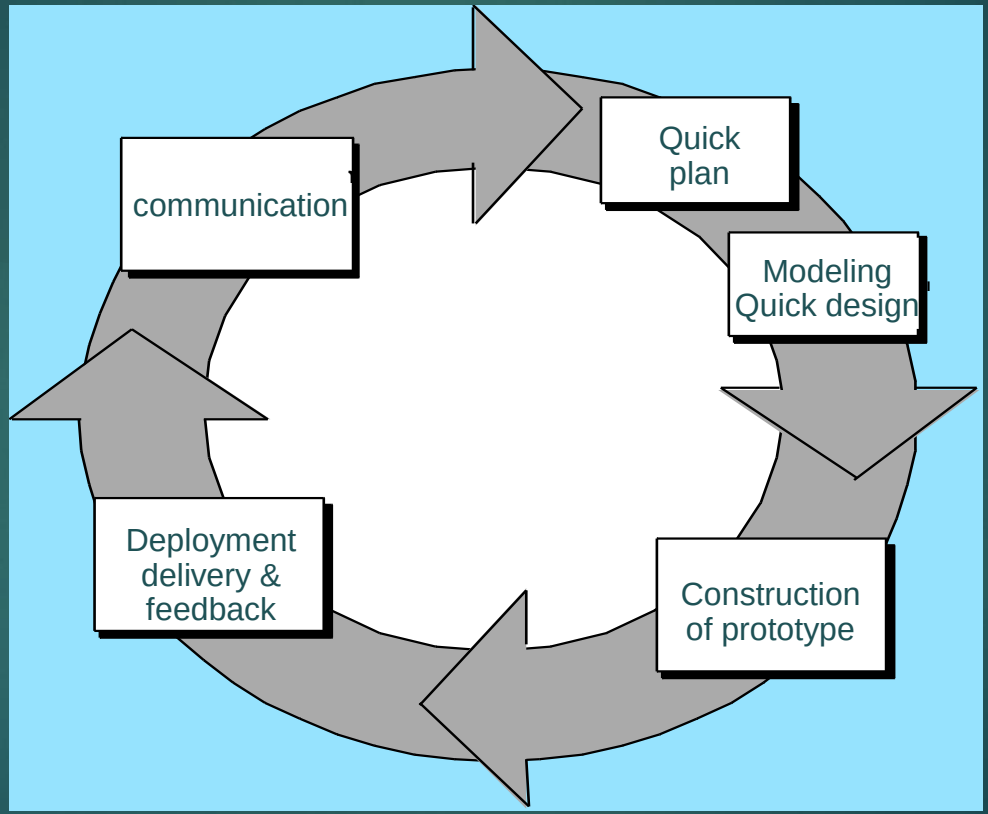
RAD

- **When to use RAD model:**
  - RAD should be used when there is a need to create a system that can be modularized in 2-3 months of time.
  - It should be used if there's high availability of designers for modeling and the budget is high enough to afford their cost along with the cost of automated code generating tools.
  - RAD **SDLC model** should be chosen only if resources with high business knowledge are available and there is a need to produce the system in a short span of time (2-3 months).

Paradigms of software engineering process

- Prescriptive Models
  - Waterfall model
- Incremental Process Models
  - The incremental model
  - The RAD model
- Evolutionary Process models
  - Prototyping model
  - Spiral model
  - Concurrent Development model
- Specialized Process models
  - Component based model
  - Formal methods model
  - Aspect oriented s/w development model
- The Unified process model
- Agile process models
  - XP
  - Adaptive software development
  - Dynamic systems development method
  - Scrum
  - Crystal
  - Feature Driven Development

Evolutionary Models: Prototyping

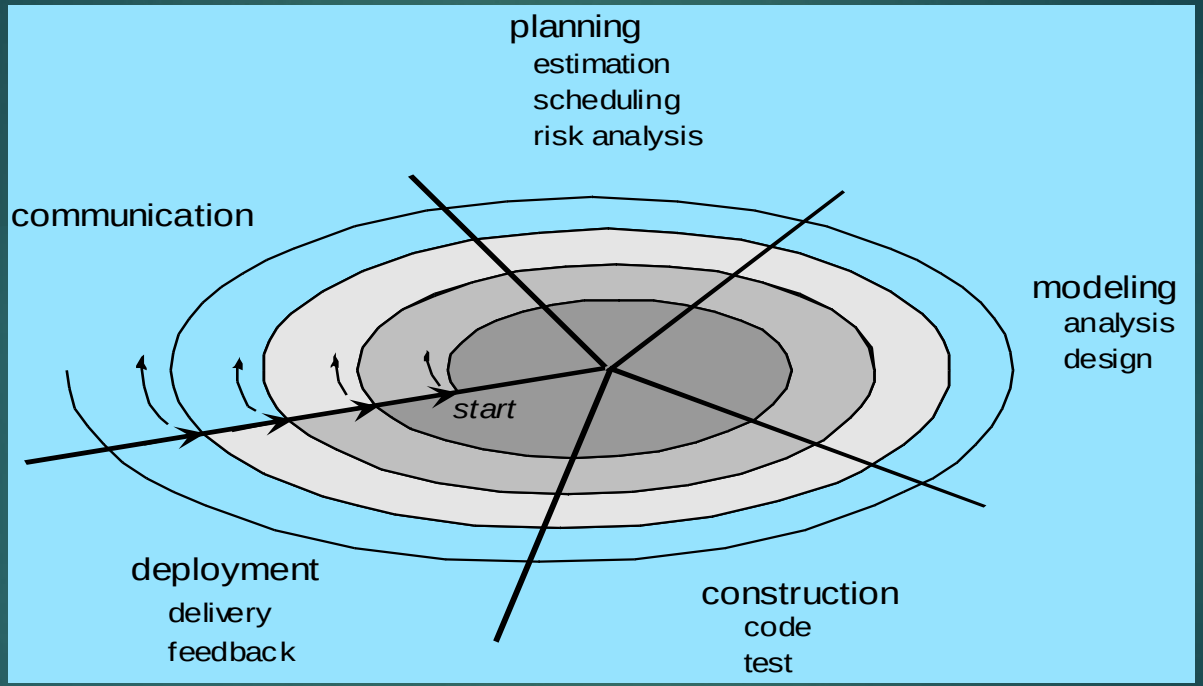


- **Advantages of Prototype model:**
  - Users are actively involved in the development
  - Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.
  - Errors can be detected much earlier.
  - Quicker user feedback is available leading to better solutions.
  - Missing functionality can be identified easily
  - Confusing or difficult functions can be identified
  - Requirements validation, Quick implementation of, incomplete, but functional, application.

- **Disadvantages of Prototype model:**
  - Leads to implementing and then repairing way of building systems.
  - Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
  - Incomplete application may cause application not to be used as the full system was designed
  - Incomplete or inadequate problem analysis.

- **When to use Prototype model:**
  - Prototype model should be used when the desired system needs to have a lot of interaction with the end users.
  - Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model. It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.
  - Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system. They are excellent for designing good human computer interface systems.

Evolutionary Models: The Spiral

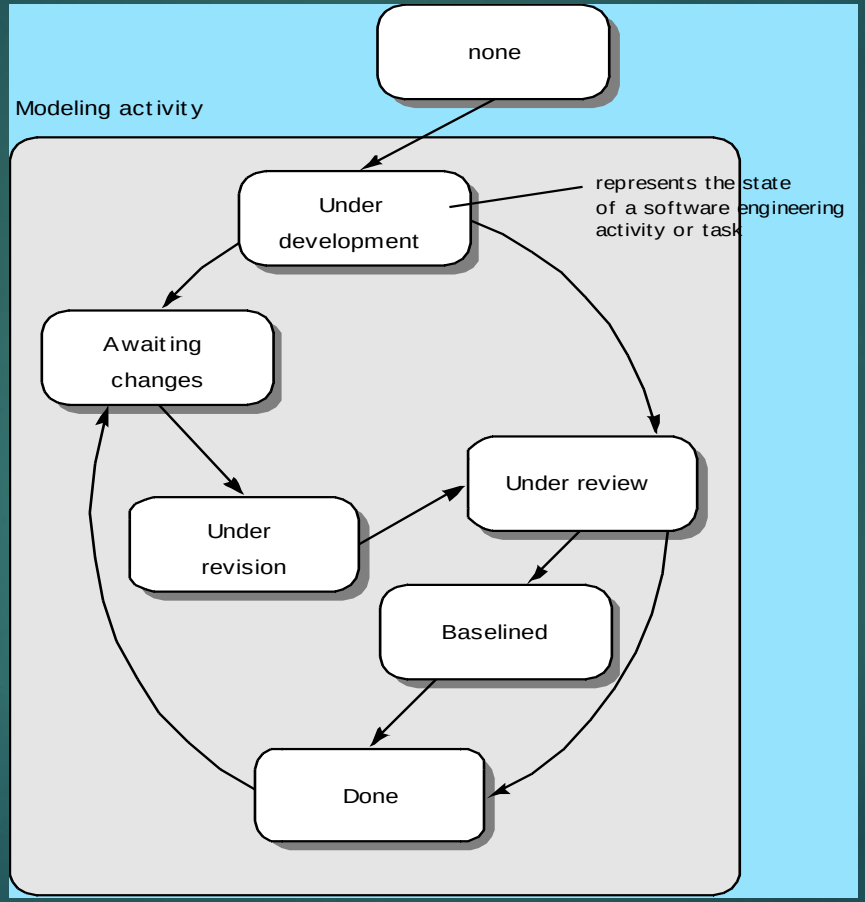


- **Advantages of Spiral model:**
  - High amount of risk analysis hence, avoidance of Risk is enhanced.
  - Good for large and mission-critical projects.
  - Strong approval and documentation control.
  - Additional Functionality can be added at a later date.
  - Software is produced early in the **software life cycle**.

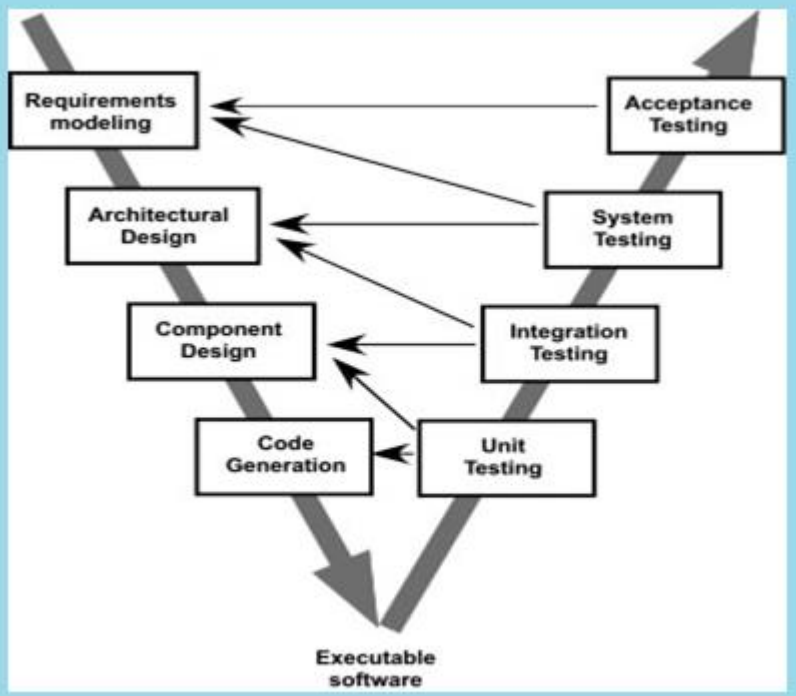
- **Disadvantages of Spiral model:**
  - Can be a costly model to use.
  - Risk analysis requires highly specific expertise.
  - Project's success is highly dependent on the risk analysis phase.
  - Doesn't work well for smaller projects.
  - Difficult to convince customer that the evolutionary approach is controllable.

- **When to use Spiral model:**
  - When costs and risk evaluation is important
  - For medium to high-risk projects
  - Long-term project commitment unwise because of potential changes to economic priorities
  - Users are unsure of their needs
  - Requirements are complex
  - New product line
  - Significant changes are expected (research and exploration)

Evolutionary Models: Concurrent model



The V-Model


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

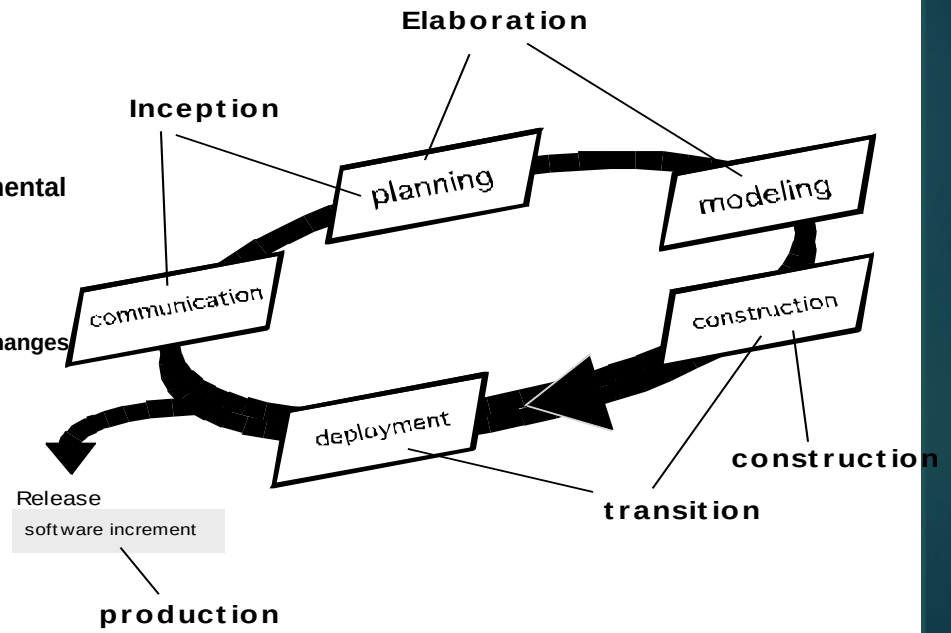
- **Why preferred?**
  - It is easy to manage due to the rigidity of the model. Each phase of V-Model has specific deliverables and a review process.
  - Proactive defect tracking – that is defects are found at early stage.
- **When to use?**
  - Where requirements are clearly defined and fixed.
  - The V-Model is used when ample technical resources are available with technical expertise.
- **Advantages:**
  - This is a highly disciplined model and Phases are completed one at a time.
  - V-Model is used for small projects where project requirements are clear.
  - Simple and easy to understand and use.
- **Disadvantages:**
  - High risk and uncertainty.
  - It is not a good for complex and object-oriented projects.
  - It is not suitable for projects where requirements are not clear and contains high risk of changing.

Still Other Process Models

- Component based development—the process to apply when reuse is a development objective
- Formal methods model—emphasizes the mathematical specification of requirements
- Aspect Oriented Software Development—provides a process and methodological approach for defining, specifying, designing, and constructing *aspects*
- Unified Process—a "use-case driven, architecture-centric, iterative and incremental" software process closely aligned with the Unified Modeling Language (UML)

The Unified Process (UP)

- Features
- Use Case Driven, Architecture Centric, Iterative & Incremental
- Draws Best Features from Conventional Process Models
- Implements Best Principles of Agile S/W Development
- Costumer Communication is emphasized ---- Use Case
- Architecture Focus on Understandability, Reliance to future Changes



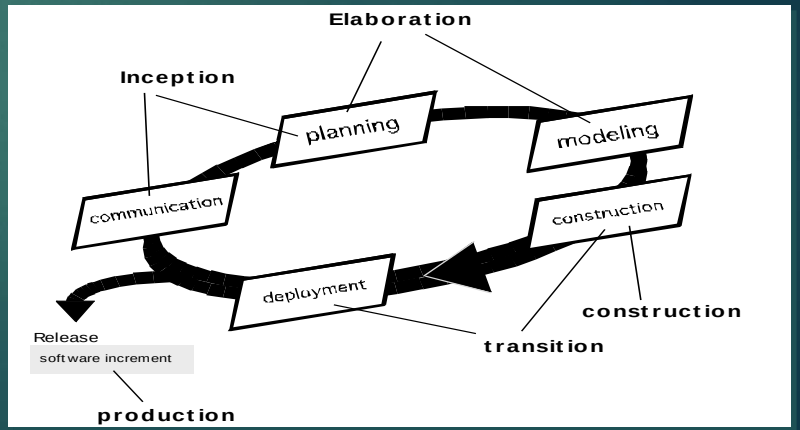
- Features
  - Use Case Driven, Architecture Centric, Iterative & Incremental
  - Draws Best Features from Conventional Process Models
  - Implements Best Principles of Agile S/W Development
  - Costumer Communication is emphasized ---- Use Case
  - Architecture Focus on
    - Understandability, Reliance to future Changes & Reuse

Inception phase

- Customer Commn & Planning Business Requirements Identified
- Rough Architecture Proposed, Plan for iterative, incremental project
- Requirements are described by use cases : features, functions, sequence of actions that are performed by actor, scope
- Major Subsystem and model represent different view of system
- Planning identifies resources, assess major risks, defines schedule, establishes bases to be applied as s/w increments

Elaboration phase

- Customer Commn & Modelling Activities
- Elaborate Use cases, Expands Architecture to include views i.e. Use Case Model, Analysis Model, Design Model, Implementation Model, Deployment Model
- Architecture Baselines are created but not all features & functions
- Plan carefully reviewed to ensure scope, risks, delivery dates remain reasonable. Plan may be modified



Construction phase

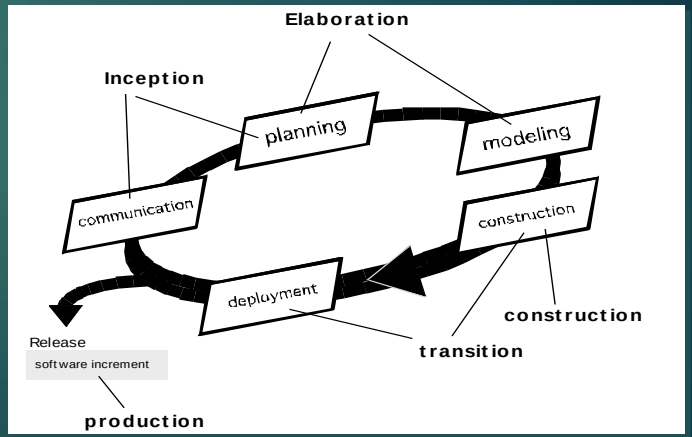
- Using s/w architecture Model develops or acquires components
- Analysis & Design model are completed to reflect final version of s/w increments
- Features & Functions implemented in code
- Tests are designed

Transition phase

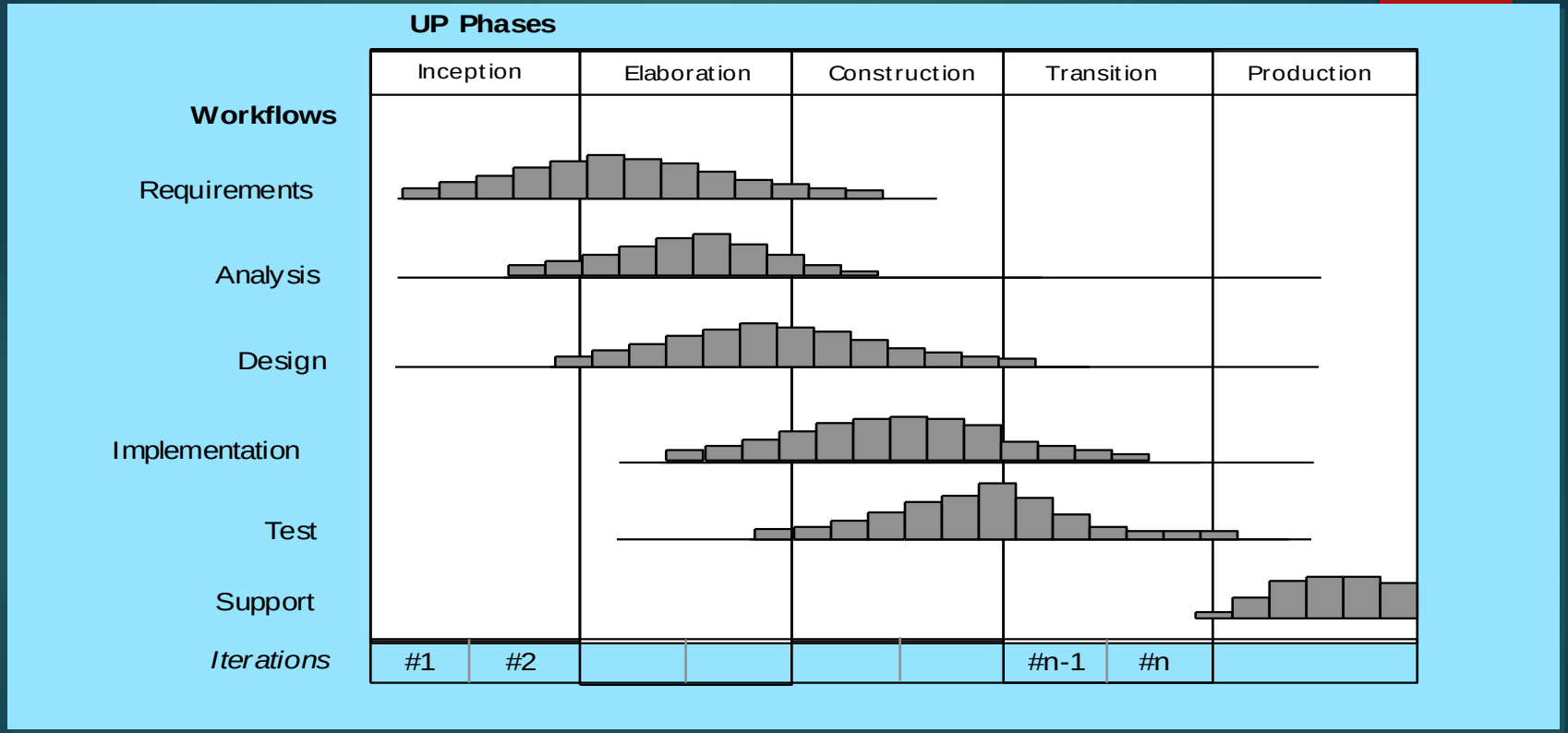
- Encompass Construction & Deployment
- S/W is given to end user for beta testing
- s/w team creates user manual, trouble shooting guides, installation procedures
- S/W increment becomes useable software release

Production phase

- Coincides deployment
- Ongoing s/w phase is monitored
- Support for operating environment is provided
- Defect report & request for changes are submitted and evaluated



UP Phases


UP Phases
Inception
Elaboration
Construction
Transition
Production
Workflows
Requirements
Analysis
Design
Implementation
Test
Support
Iterations
#1
#2
#n-1
#n

UP Work Products

Inception phase

- Vision document
- Initial use-case model
- Initial project glossary
- Initial business case
- Initial risk assessment.
- Project plan,
  - phases and iterations.
- Business model,
  - if necessary.
- One or more prototypes

Elaboration phase

- Use-case model
- Supplementary requirements
  - including non-functional
- Analysis model
- Software architecture
  - Description.
- Executable architectural
  - prototype.
- Preliminary design model
- Revised risk list
- Project plan including
  - iteration plan
  - adapted workflows
  - milestones
  - technical work products
- Preliminary user manual

Construction phase

- Design model
- Software components
- Integrated software
  - increment
- Test plan and procedure
- Test cases
- Support documentation
  - user manuals
  - installation manuals
  - description of current
    - increment

Transition phase

- Delivered software increment
- Beta test reports
- General user feedback

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

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."

What is “Agility”?

- Effective (rapid and adaptive) response to change (team members, new technology, requirements)
- Effective communication in structure and attitudes among all team members, technological and business people, software engineers and managers。
- Drawing the customer into the team. Eliminate “us and them” attitude. Planning in an uncertain world has its limits and plan must be flexible.
- Organizing a team so that it is in control of the work performed
- Eliminate all but the most essential work products and keep them lean.
- Emphasize an incremental delivery strategy as opposed to intermediate products that gets working software to the customer as rapidly as feasible.

What is “Agility”?

Yielding …

- Rapid, incremental delivery of software
- The development guidelines stress delivery over analysis and design although these activates are not discouraged, and active and continuous communication between developers and customers.

Why and What Steps are“Agility” important?

- **Why?** The modern business environment is fast-paced and ever-changing. It represents a reasonable alternative to conventional software engineering for certain classes of software projects. It has been demonstrated to deliver successful systems quickly.
- **What?** May be termed as “software engineering lite” The basic activities- communication, planning, modeling, construction and deployment remain. But they morph into a minimal task set that push the team toward construction and delivery sooner.
- The only really important work product is an operational “software increment” that is delivered.

An Agile Process

- Is driven by customer descriptions of what is required (scenarios). Some assumptions:
  - Recognizes that plans are short-lived (some requirements will persist, some will change. Customer priorities will change)
  - Develops software iteratively with a heavy emphasis on construction activities (design and construction are interleaved, hard to say how much design is necessary before construction. Design models are proven as they are created.)
  - Analysis, design, construction and testing are not predictable.
- Thus has to Adapt as changes occur due to unpredictability
- Delivers multiple 'software increments', deliver an operational prototype or portion of an OS to collect customer feedback for adaption.

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

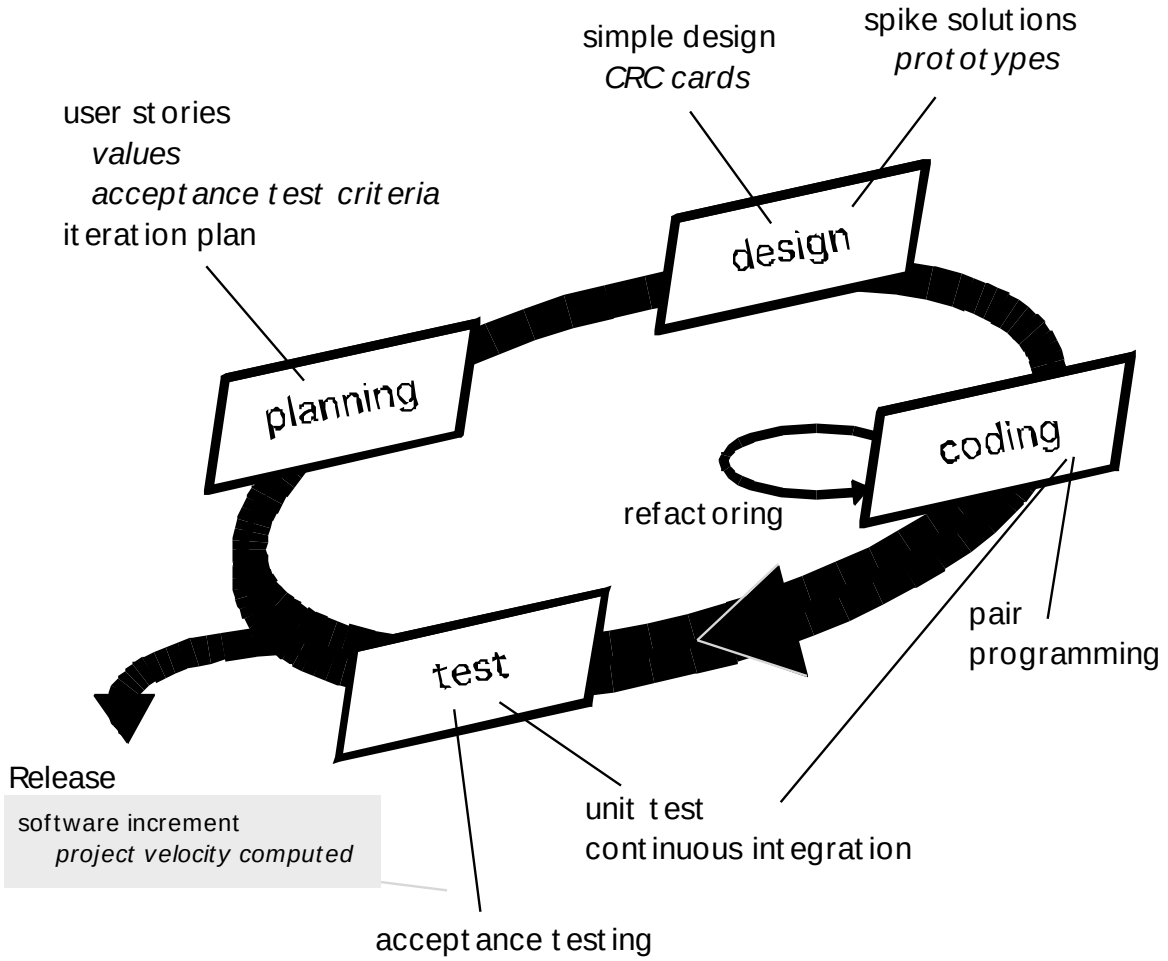
- *the process molds to the needs of the people and team,* not the other way around
- key traits must exist among the people on an agile team and the team itself:
  - **Competence. (talent, skills, knowledge)**
  - **Common focus. (deliver a working software increment)**
  - **Collaboration. (peers and stakeholders)**
  - **Decision-making ability. (freedom to control its own destiny)**
  - **Fuzzy problem-solving ability.(ambiguity and constant changes, today problem may not be tomorrow's problem)**
  - **Mutual trust and respect.**
  - **Self-organization. (themselves for the work done, process for its local environment, the work schedule)**

Extreme Programming (XP)

- The most widely used agile process, originally proposed by Kent Beck in 2004. It uses an object-oriented approach.
- XP Planning
  - Begins with the listening, leads to creation of "user stories" that describes required output, features, and functionality. Customer assigns a value(i.e., a priority) to each story.
  - Agile team assesses each story and assigns a cost (development weeks. If more than 3 weeks, customer asked to split into smaller stories)
  - Working together, stories are grouped for a deliverable increment next release.
  - A commitment (stories to be included, delivery date and other project matters) is made. Three ways: 1. Either all stories will be implemented in a few weeks, 2. high priority stories first, or 3. the riskiest stories will be implemented first.
  - After the first increment "project velocity", namely number of stories implemented during the first release is used to help define subsequent delivery dates for other increments. Customers can add stories, delete existing stories, change values of an existing story, split stories as development work proceeds.

Extreme Programming (XP)

- XP Design (occurs both before and after coding as refactoring is encouraged)
  - Follows the KIS principle (keep it simple) Nothing more nothing less than the story.
  - Encourage the use of CRC (class-responsibility-collaborator) cards in an object-oriented context. The only design work product of XP. They identify and organize the classes that are relevant to the current software increment. (see Chapter 8)
  - For difficult design problems, suggests the creation of "spike solutions"—a design prototype for that portion is implemented and evaluated.
  - Encourages "refactoring"—an iterative refinement of the internal program design. Does not alter the external behavior yet improve the internal structure. Minimize chances of bugs. More efficient, easy to read.
- XP Coding
  - Recommends the construction of a unit test for a story *before* coding commences. So implementer can focus on what must be implemented to pass the test.
  - Encourages "pair programming". Two people work together at one workstation. Real time problem solving, real time review for quality assurance. Take slightly different roles.
- XP Testing
  - All unit tests are executed daily and ideally should be automated. Regression tests are conducted to test current and previous components.
  - "Acceptance tests" are defined by the customer and executed to assess customer visible functionality


simple design
CRC cards
spike solutions
prototypes
user stories
values
acceptance test criteria
iteration plan
design
planning
coding
refactoring
pair
programming
test
Release
software increment
project velocity computed
unit test
continuous integration
acceptance testing

Class: FloorPlan

Description:

Responsibility:	Collaborator:
defines floor plan name/type	
manages floor plan positioning	
scales floor plan for display	
scales floor plan for display	
incorporates walls, doors and windows	Wall
shows position of video cameras	Camera

Type of Class:
Entity
Boundary
Controller

Responsibility:
Attributes
Operations

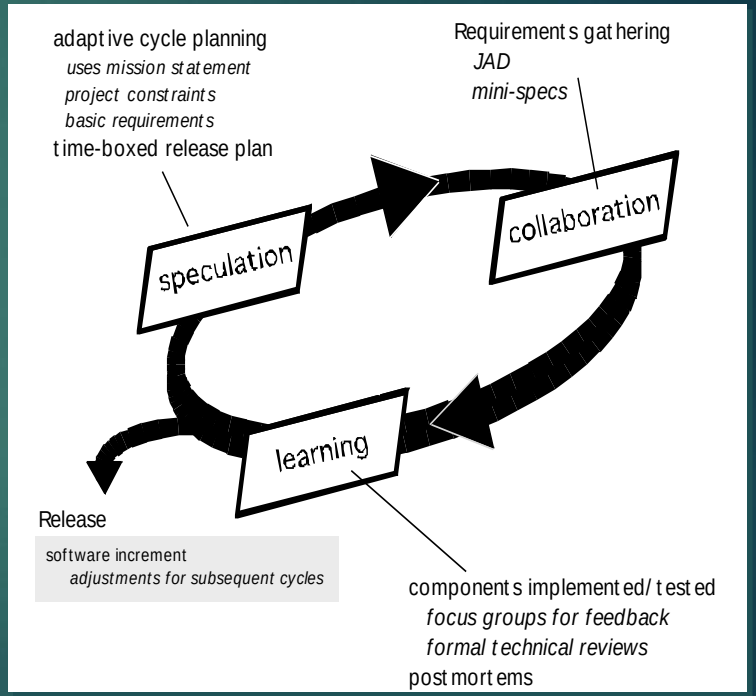
Collaborator:
Stand-alone
Or Collaborate

The XP Debate

- Requirements volatility: customer is an active member of XP team, changes to requirements are requested informally and frequently.
- Conflicting customer needs: different customers' needs need to be assimilated. Different vision or beyond their authority.
- Requirements are expressed informally: Use stories and acceptance tests are the only explicit manifestation of requirements. Formal models may avoid inconsistencies and errors before the system is built. Proponents said changing nature makes such models obsolete as soon as they are developed.
- Lack of formal design: XP deemphasizes the need for architectural design. Complex systems need overall structure to exhibit quality and maintainability. Proponents said incremental nature limits complexity as simplicity is a core value.

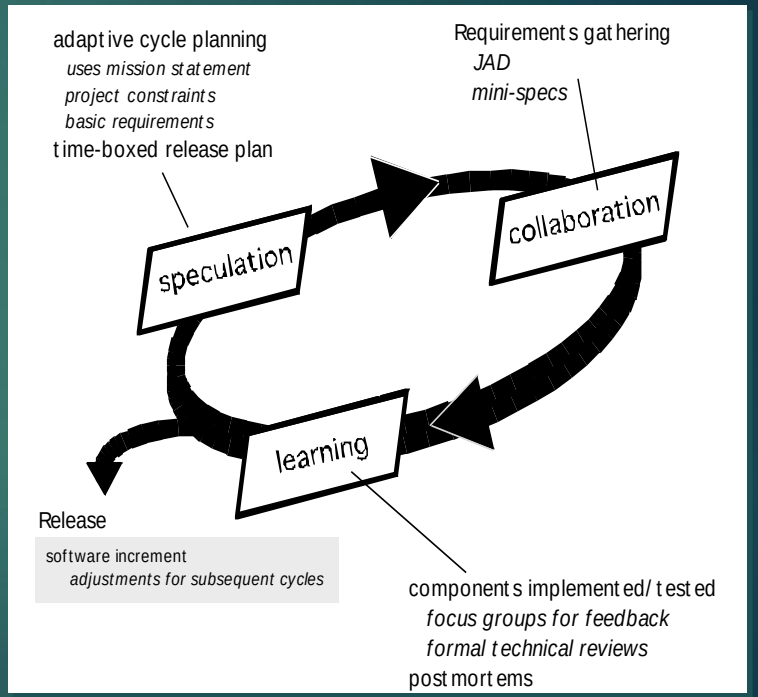
Adaptive Software Development (ASD)

- Originally proposed by Jim Highsmith (2000)focusing on human collaboration and team self-organization as a technique to build complex software and system.
- ASD — distinguishing features
  - Mission-driven planning
  - Component-based focus
  - Uses "time-boxing"
  - Explicit consideration of risks
  - Emphasizes collaboration for requirements gathering
  - Emphasizes "learning" throughout the process



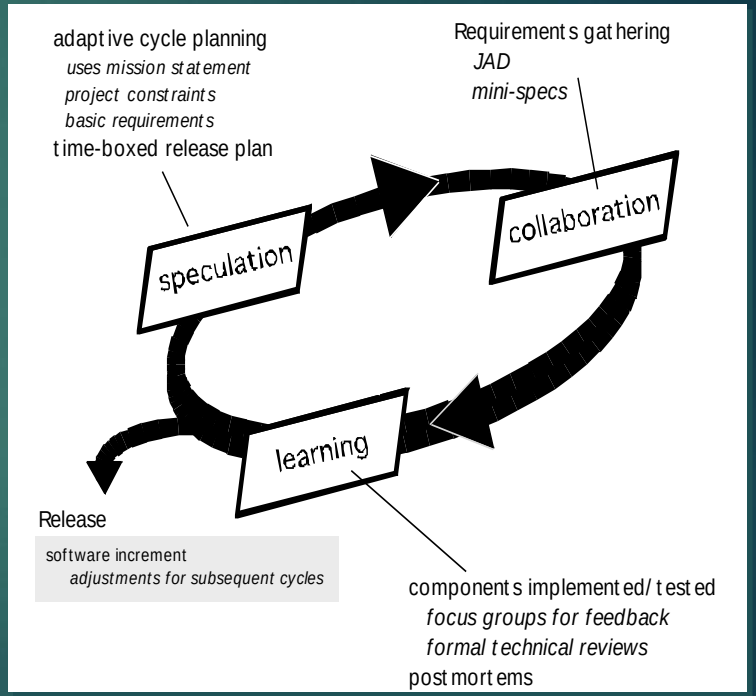
Three Phases of ASD

- 1. Speculation: 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 date), and basic requirements to define the set of release cycles (increments) that will be required for the project. Based on the information obtained at the completion of the first cycle, the plan is reviewed and adjusted so that planned work better fits the reality.
- 2. Collaborations are used to multiply their talent and creative output beyond absolute number (1+1>2). It encompasses communication and teamwork, but it also emphasizes individualism, because individual creativity plays an important role in collaborative thinking.
- It is a matter of trust. 1) criticize without animosity, 2) assist without resentments, 3) work as hard as or harder than they do. 4) have the skill set to contribute to the work at hand, 5) communicate problems or concerns in a way that leads to effective action.



Three Phases of ASD

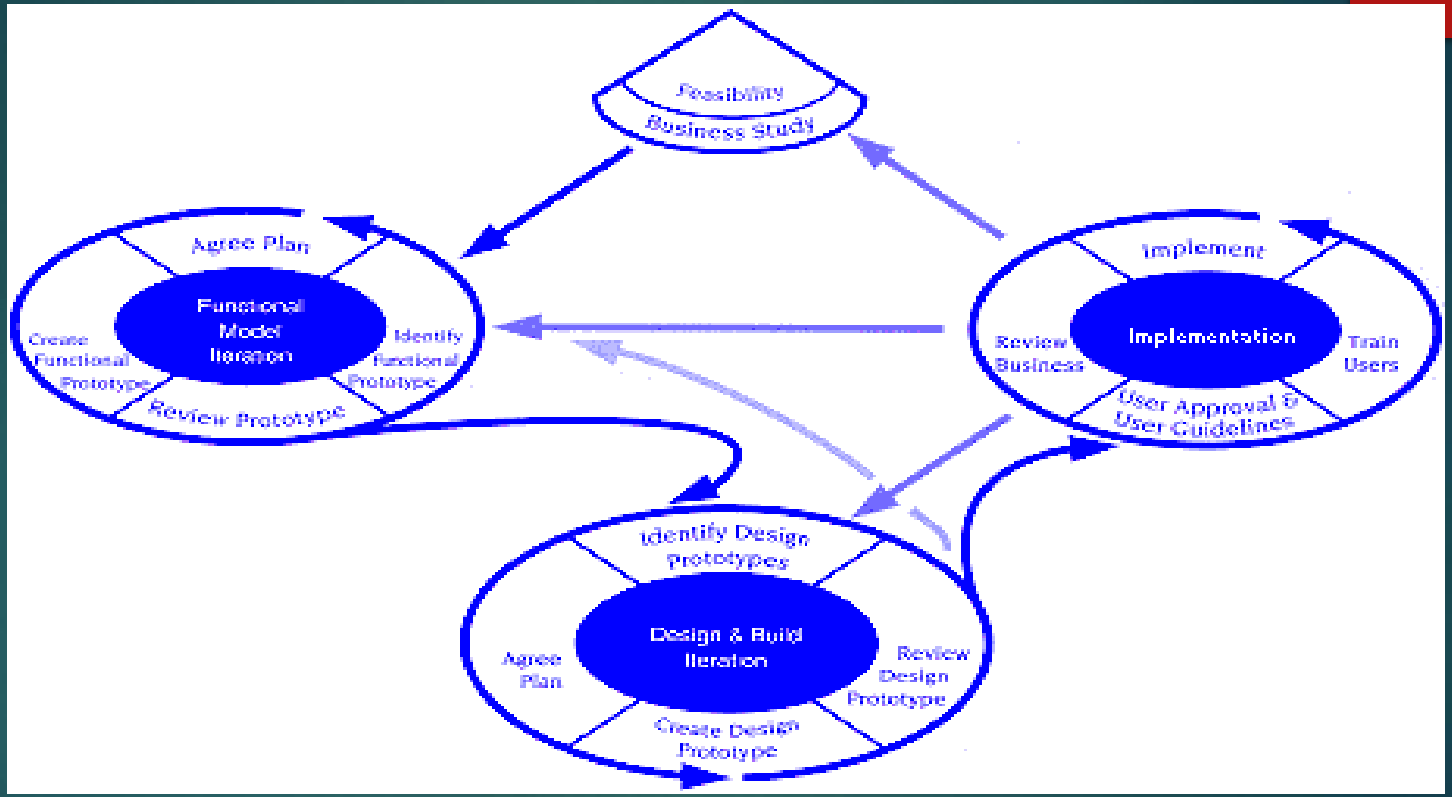
- 3. Learning: As members of ASD team begin to develop the components, the emphasis is on "learning". Highsmith argues that software developers often overestimate their own understanding of the technology, the process, and the project and that learning will help them to improve their level of real understanding.
- Three ways:
  - focus groups: customer provide feedback on software increments that are being delivered.
  - Formal technical reviews: Reviews improve quality and provide learning as they progress.
  - project postmortems: ASD team becomes introspective, addressing its own performance and process.



Dynamic Systems Development Method

- It is an agile software development approach that provides a framework for building and maintaining systems which meet tight time constraints through the use of incremental prototyping in a controlled project environment.
- Promoted by the DSDM Consortium (www.dsdm.org)
- DSDM—distinguishing features
  - Similar in most respects to XP and/or ASD
  - Nine guiding principles
    - Active user involvement is imperative.
    - DSDM teams must be empowered to make decisions.
    - The focus is on frequent delivery of products.
    - Fitness for business purpose is the essential criterion for acceptance of deliverables.
    - Iterative and incremental development is necessary to converge on an accurate business solution.
    - All changes during development are reversible.
    - Requirements are baselined at a high level
    - Testing is integrated throughout the life-cycle.

Dynamic Systems Development Method



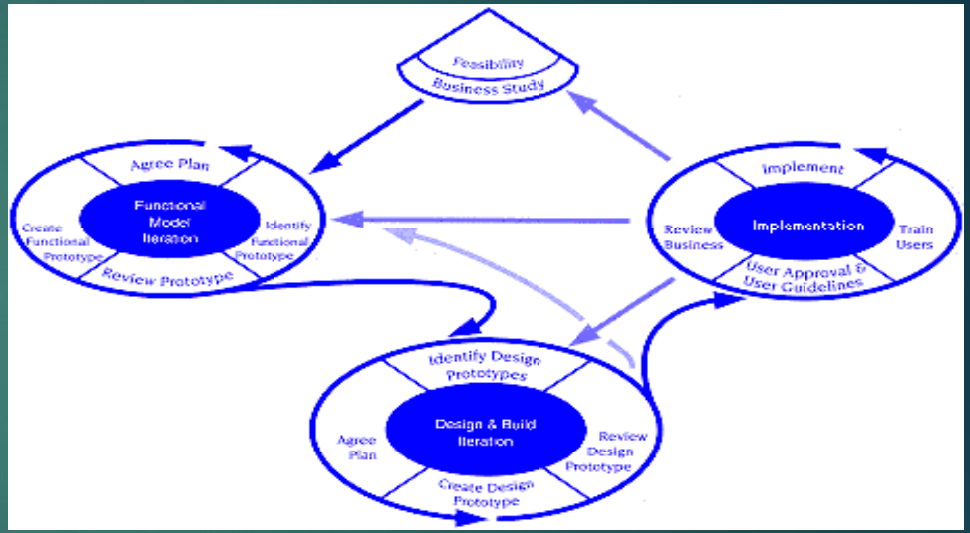
DSDM Life Cycle (with permission of the DSDM consortium)

Dynamic Systems Development Method

- Feasibility study
  - Establishes basic business requirements and constraints
  - Assess if the application is viable candidate for DSDM process
- Business study
  - Establish functional and information requirements that provide business value
  - Define basic architecture and identifies maintainability requirements
- Functional model iteration
- Design and build iteration
- Implementation

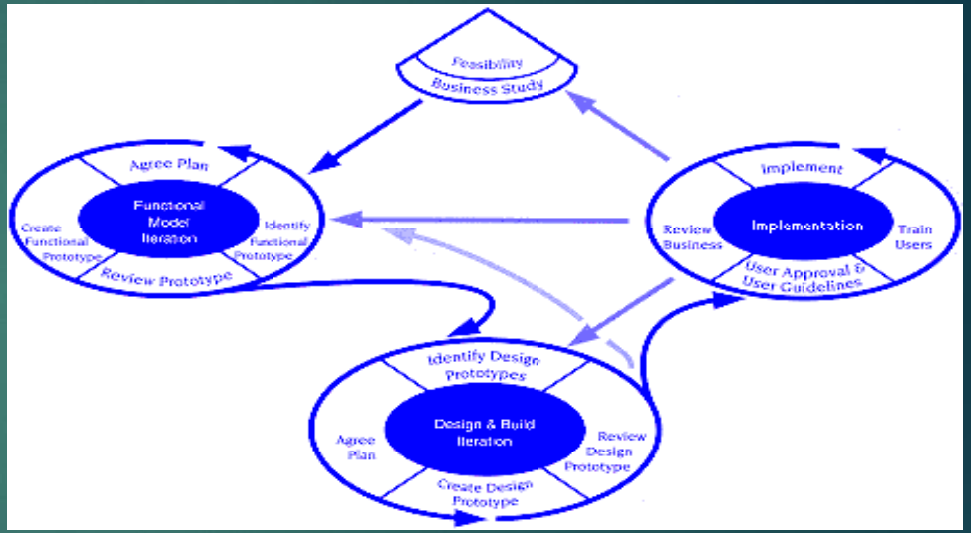
Dynamic Systems Development Method

- Functional model iteration
  - Incremental prototypes that demonstrate functionality to customer
  - Intent is to gather additional requirements by eliciting feedback from users as they exercise the prototype
- Design and build iteration
  - Revisit prototypes to ensure functionalities provide operational business value to end-users
  - Occurs concurrently with the functional model iteration



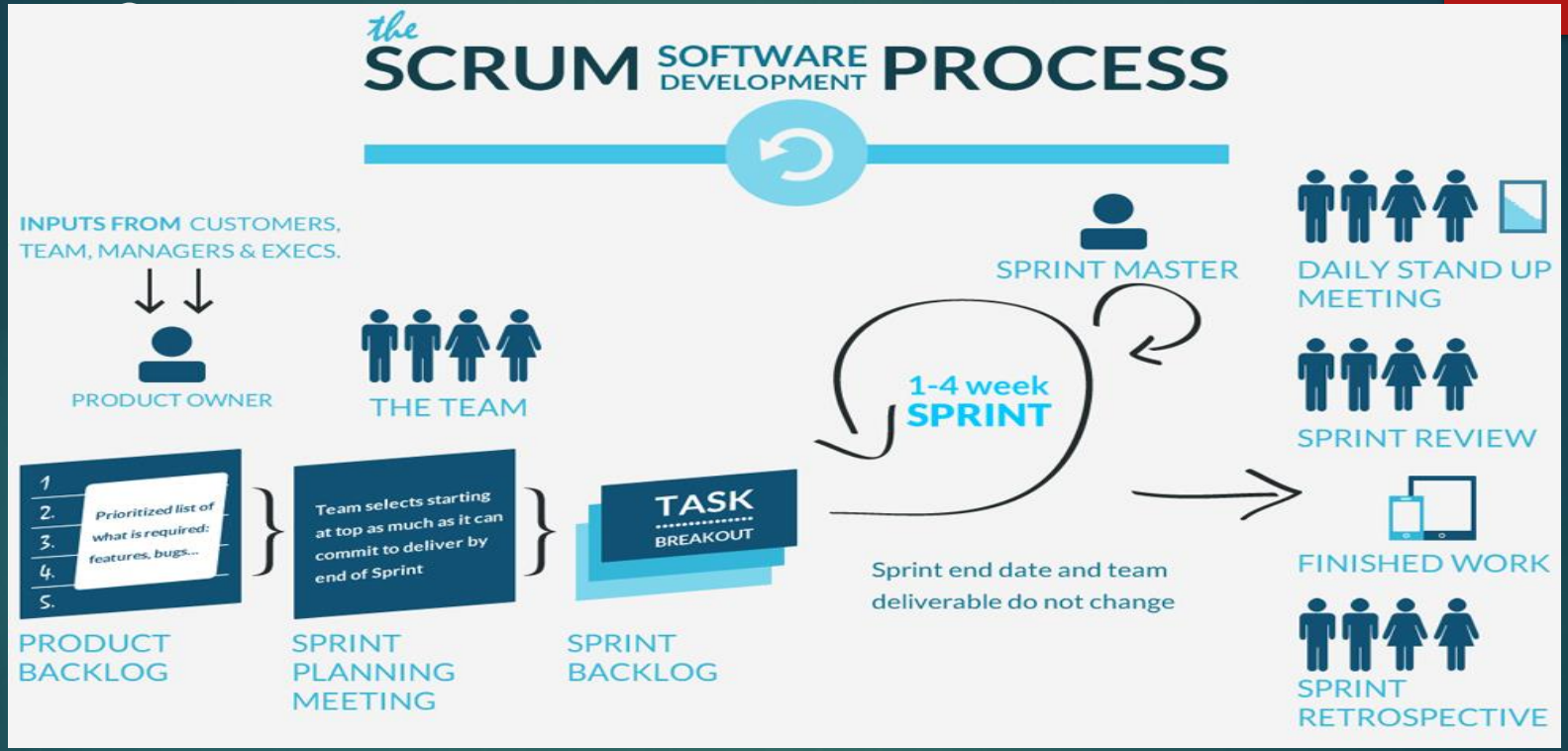
Dynamic Systems Development Method

- Implementation
  - Places latest software increment in operational environment
    - 1. increment may not be 100 percent complete
    - 2. changes may be requested as the increment is put into place
- DSDM continues by returning to the functional model iteration activity



Scrum

- A software development method Originally proposed by Schwaber and Beedle (an activity occurs during a rugby match) in early 1990.
- Scrum—distinguishing features
  - Development work is partitioned into "packets"
  - Testing and documentation are on-going as the product is constructed
  - Work units occurs in "sprints" and is derived from a "backlog" of existing changing prioritized requirements
  - Changes are not introduced in sprints (short term but stable) but in backlog.
  - Meetings are very short (15 minutes daily) and sometimes conducted without chairs (what did you do since last meeting? What obstacles are you encountering? What do you plan to accomplish by next meeting?)
  - "demos" are delivered to the customer with the time-box allocated. May not contain all functionalities. So customers can evaluate and give feedbacks.


the
SCRUM SOFTWARE DEVELOPMENT PROCESS
INPUTS FROM CUSTOMERS, TEAM, MANAGERS & EXECS.
PRODUCT OWNER
THE TEAM
SPRINT MASTER
DAILY STAND UP MEETING
1-4 week SPRINT
SPRINT REVIEW
1
2.
3.
4.
5.
Prioritized list of what is required: features, bugs...
Team selects starting at top as much as it can commit to deliver by end of Sprint
TASK BREAKOUT
Sprint end date and team deliverable do not change
FINISHED WORK
PRODUCT BACKLOG
SPRINT PLANNING MEETING
SPRINT BACKLOG
SPRINT RETROSPECTIVE

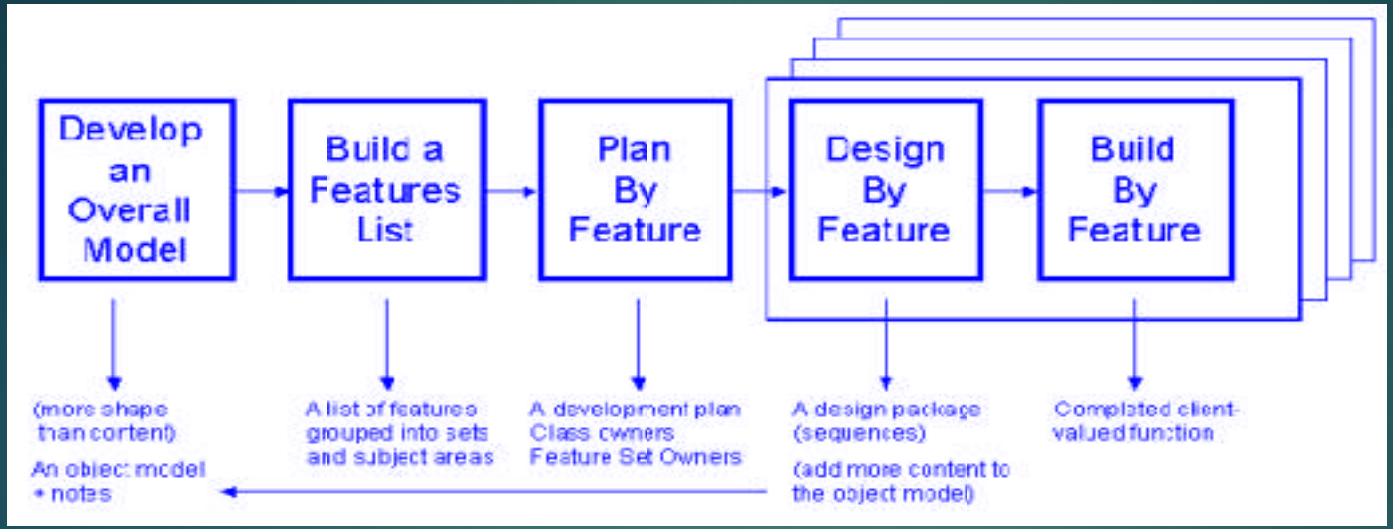
Crystal

- Proposed by Cockburn and Highsmith
- Crystal—distinguishing features
  - Actually a family of process models that allow "maneuverability" based on problem characteristics
    - Maneuverability – a resource limited corporate game of invention and communication with a primary goal of delivering useful, working software and a secondary goal of setting up for the next game
  - Face-to-face communication is emphasized
  - Suggests the use of "reflection workshops" to review the work habits of the team

Feature Driven Development

- Originally proposed by Peter Coad et al as a practical process model for object-oriented software engineering process model.
- FDD—distinguishing features
  - Emphasis is on defining "features" which can be organized hierarchically.
    - 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>**
    - E.g. Add the product to shopping cart.
    - Display the technical-specifications of the product.
    - Store the shipping-information for the customer.
  - 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

Kanban model

The Kanban Method is a means to design, manage, and improve flow systems for knowledge work. The method also allows organizations to start with their existing workflow and drive evolutionary change. They can do this by visualizing their flow of work, limit work in progress (WIP) and stop starting and start finishing.

The Kanban Method gets its name from the use of Kanban – visual signaling mechanisms to control work in progress for intangible work products.

When Applicable: Kanban can be used in any knowledge work setting, and is particularly applicable in situations where work arrives in an unpredictable fashion and/or when you want to deploy work as soon as it is ready, rather than waiting for other work items.

Principles

Change Management Principles: Kanban is structured to address the human tendency to resist change.

- **Start with what you do now** – Understand current processes as they are actually practiced and respect existing roles, responsibilities and job titles.
- **Agree to pursue improvement through evolutionary change**
- **Encourage acts of leadership at every level**

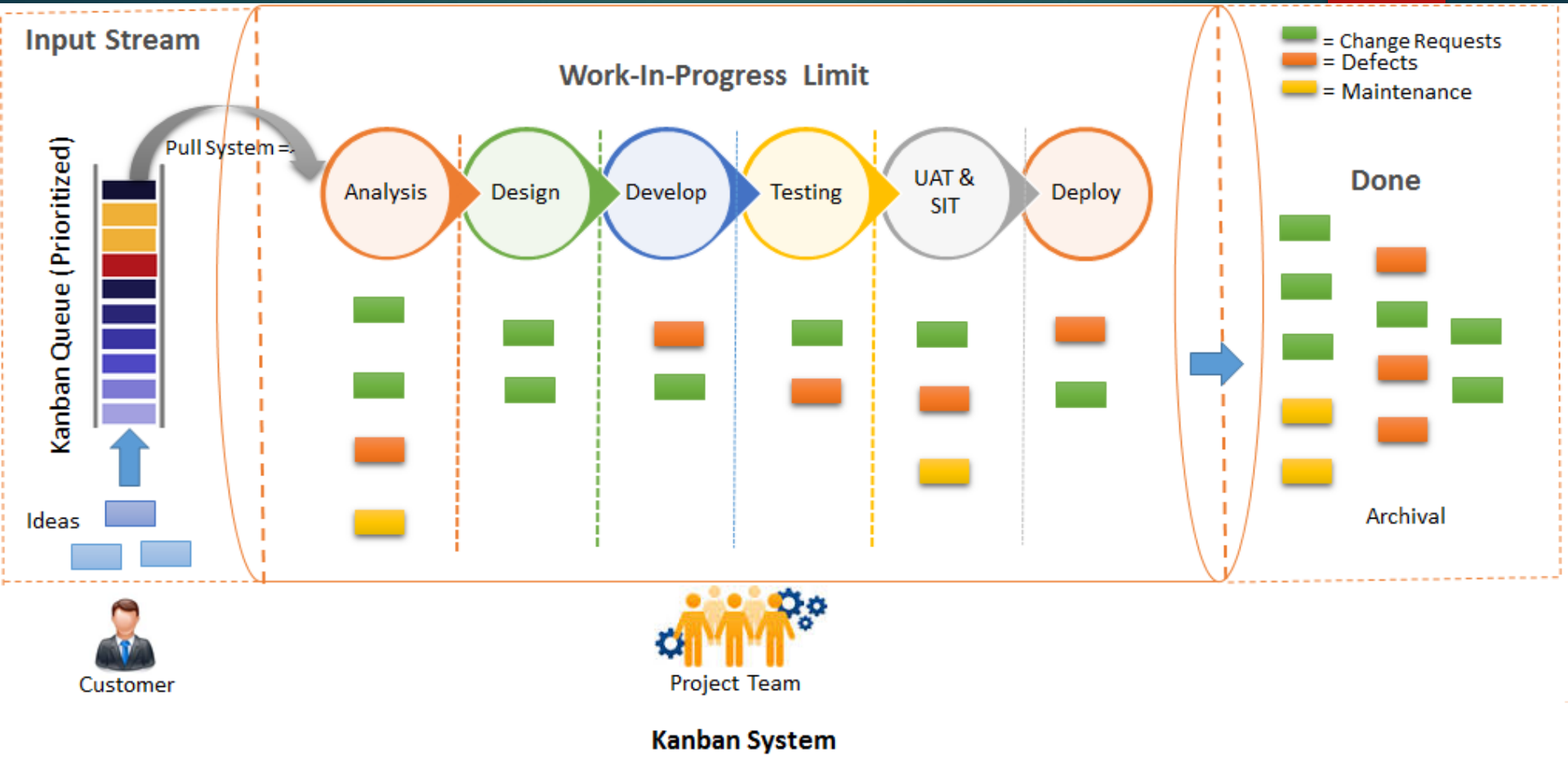
Service Delivery Principles: These principles acknowledge that organizations are a collection of interdependent services, and to place the focus on the work, not the people doing the work.

- Understand and focus on your customers' needs and expectations
- Manage the work; let people self-organize around it
- Evolve policies to improve customer and business outcomes

Kanban model

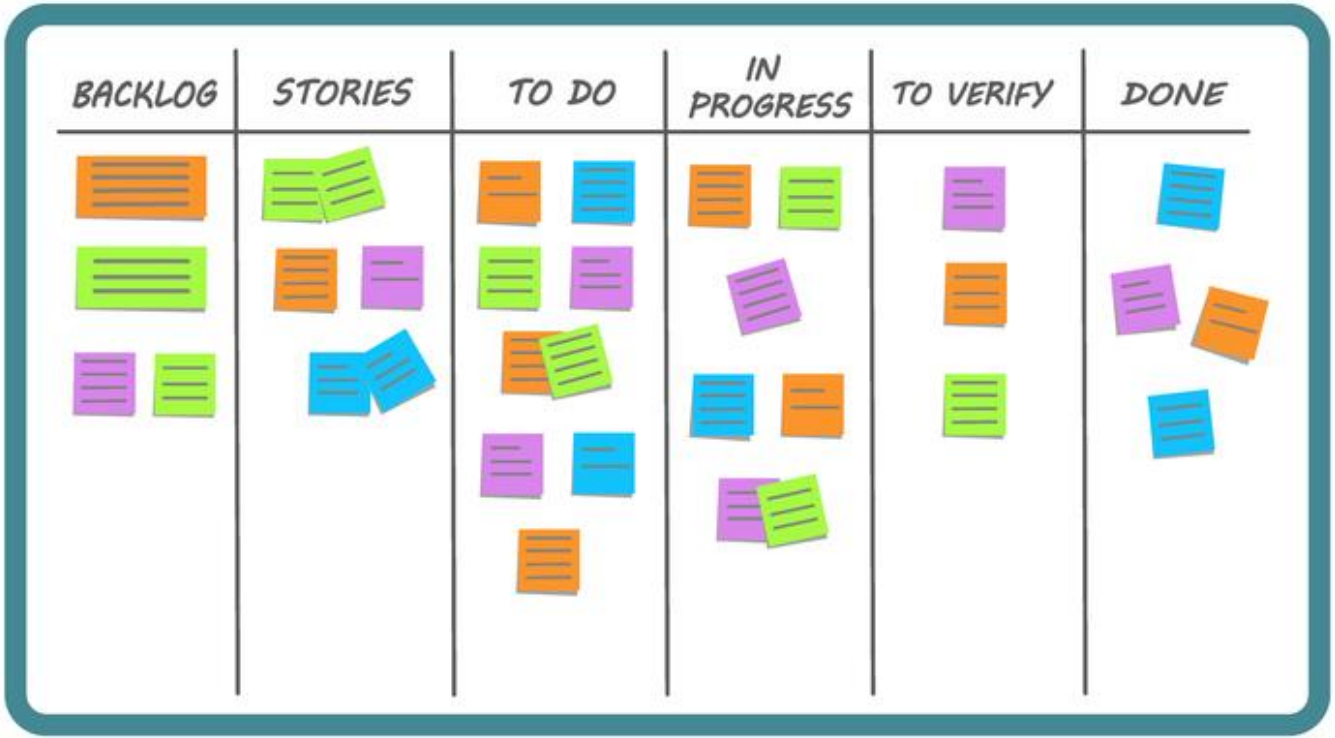
Values

Transparency – sharing information openly using clear and straightforward language improves the flow of business value.
Balance – different aspects, viewpoints, and capabilities must be balanced in order to achieve effectiveness.
Collaboration – Kanban was created to improve the way people work together.
Customer Focus – Kanban systems aim to optimize the flow of value to customers that are external from the system but may be internal or external to the organization in which the system exists.
Flow – Work is a continuous or episodic flow of value.
Leadership – Leadership (the ability to inspire others to act via example, words, and reflection) is needed at all levels in order to realize continuous improvement and deliver value.
Understanding – Individual and organizational self-knowledge of the starting point is necessary to move forward and improve.
Agreement – Everyone involved with a system are committed to improvement and agree to jointly move toward goals while respecting and accommodating differences of opinion and approach.
Respect – Value, understand, and show consideration for people.

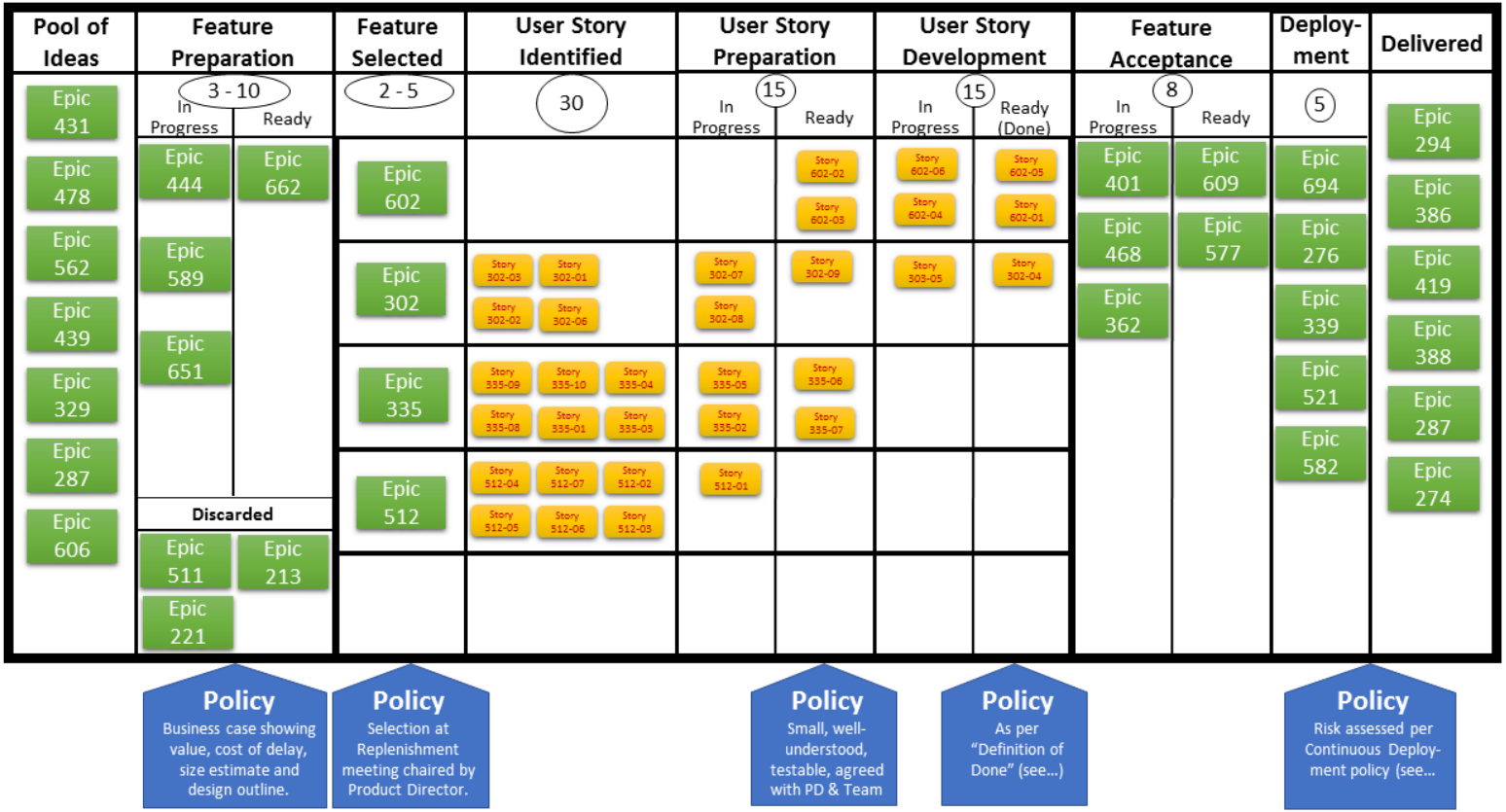


Kanban System

Sample Kanban



Sample Kanban



Kanban model

The following practices are activities essential to manage a kanban system.

Visualize

Kanban systems use mechanisms such as a kanban board to visualize work and the process it goes through. In order for the visualization to be the most effective, it should show

- where in the process a team working on a service agrees to do a specific work item (commitment point)
- Where the team delivers the work item to a customer (delivery point)
- Policies that determine what work should exist in a particular stage
- WIP Limits

Limit work in progress

When you establish limits to the amount of work you have in progress in a system and use those limits to guide when to start new items, you can smooth out the flow of work and reduce lead times, improve quality, and deliver more frequently.

Kanban model

Manage flow

The flow of work in a service should maximize value delivery, minimize lead times and be as predictable as possible. Teams use empirical control through transparency, inspection and adaption in order to balance these potentially conflicting goals. A key aspect of managing flow is identifying and addressing bottlenecks and blockers.

Make policies explicit

Explicit policies help explain a process beyond just the listing of different stages in the workflow. Policies should be sparse, simple, well-defined, visible, always applied, and readily changeable by the people working on the service. Examples of policies include: WIP Limits, capacity allocation, definition of done, and other rules for work items existing various stages in the process.

Implement feedback loops

Feedback loops are an essential element in any system looking to provide evolutionary change. The Feedback loops used in Kanban are described in the Lifecycle section.

Improve collaboratively, evolve experimentally

Kanban starts with the process as it currently exists and applies continuous and incremental improvement instead of trying to reach a predefined finished goal.

Kanban model

Lifecycle

Because work items tend to flow through a kanban system in single piece flow, and each system is different with respect to stages in its workflow, the best way to describe the lifecycle of the Kanban method is via the feedback loops involved.

Those feedback loops (cadences) are:

Strategy Review (Quarterly)

Select the services to provide and the context in which those services are appropriate.

Operations Review (Monthly)

Understand the balance between and across services, including deploying people and resources to maximize value delivery

Risk Review (Monthly)

Understand and respond to delivery risks in services

Service Delivery Review (Bi-Weekly)

Examine and improve the effectiveness of a service. This is similar to a retrospective that is focused on improving the kanban system.

Replenishment Meeting (Weekly)

Identify items that the team will work on and determine which work items may be selected next. This is analogous to a planning meeting for a sprint or iteration.

The Kanban Meeting (Daily)

A team working on a service coordinates their activities for the day. This is analogous to a daily standup.

Delivery Planning Meeting (Per Delivery Cadence)

Monitor and plan deliveries to customers.

Kanban model

	Scrum	Kanban
Origin	Software development	Lean manufacturing
Ideology	Learn through experiences, self-organize and prioritize, and reflect on wins and losses to continuously improve.	Use visuals to improve work-in-progress
Cadence	Regular, fixed-length sprints (i.e. two weeks)	Continuous flow
Practices	Sprint planning, sprint, daily scrum, sprint review, sprint retrospective	Visualize the flow of work, limit work-in-progress, manage flow, incorporate feedback loops
Roles	Product owner, scrum master, development team	No required roles

The CMMI

- The CMMI - Capability Maturity Model Integration – represents a process meta-model in 2 different ways
- Continuous model and staged model
- defines each process area in terms of "specific goals" and the "specific practices" required to achieve these goals.
- *Specific goals* establish the characteristics that must exist if the activities implied by a process area are to be effective.
- *Specific practices* refine a goal into a set of process-related activities.
- CMMI Levels
  - Level 0: Incomplete
  - Level 1: Performed
  - Level 2: Managed
  - Level 3: Defined
  - Level 4: Quantitatively managed
  - Level 5: Optimized

The CMMI


MATURITY LEVEL 5
Optimizing
Stable and flexible. Organization is focused on continuous improvement and is built to pivot and respond to opportunity and change. The organization's stability provides a platform for agility and innovation.
MATURITY LEVEL 4
Quantitatively Managed
Measured and controlled. Organization is data-driven with quantitative performance improvement objectives that are predictable and align to meet the needs of internal and external stakeholders.
MATURITY LEVEL 3
Defined
Proactive, rather than reactive. Organization-wide standards provide guidance across projects, programs and portfolios.
MATURITY LEVEL 2
Managed
Managed on the project level. Projects are planned, performed, measured, and controlled.
MATURITY LEVEL 1
Initial
Unpredictable and reactive. Work gets completed but is often delayed and over budget.
MATURITY LEVEL 0
Incomplete
Ad hoc and unknown. Work may or may not get completed.