SS ZG622: Software Project Management (Lecture #2)

BITS Pilani
Pilani|Dubai|Goa|Hyderabad

T V Rao, BITS-Pilani Off-campus Centre, Hyderabad

Text Books

T1: Bob Hughes, Mike Cotterel, and Rajib Mall, Software Project Management, 5th Edition, McGraw Hill, 2011

T2: Pressman, R.S. Software Engineering : A Practitioner's Approach, 7th Edition, McGraw Hill, 2010

R1: Sommerville, I., Software Engineering, Pearson Education, 9th Ed., 2010

R2: "Internationally Agile" by Matt Simons © Pearson Education

R3: Robert K. Wysocki, Effective Project Management: Traditional, Agile, Extreme, John Wiley & Sons © 2014

R4: George Stepanek, Software Project Secrets : Why Software Projects Fail, Apress ©2012

R5: A Guide to the Project Management Body of Knowledge (PMBOK® Guide), Fifth Edition by Project Management Institute Project Management Institute © 2013

R6: Jake Kouns and Daniel Minoli, Information Technology Risk Management in Enterprise Environments. John Wiley & Sons © 2010

innovate achieve lead

BITS Pilani

Pilani|Dubai|Goa|Hyderabad

L2: Sotware Processes

— Process concepts, Plan-driven Process Models, Agile Process Models

Source Courtesy: Some of the contents of this PPT are sourced from materials provided by publishers of prescribed books

Framework Activities

- Communication
  - Involves communication among the customer and other stake holders; encompasses requirements gathering
- Planning
  - Establishes a plan for software engineering work; addresses technical tasks, resources, work products, and work schedule
- Modeling (Analyze, Design)
  - Encompasses the creation of models to better understand the requirements and the design
- Construction (Code, Test)
  - Combines code generation and testing to uncover errors
- Deployment
  - Involves delivery of software to the customer for evaluation and feedback

Umbrella Activities

- Software project tracking and control
  - Assess progress against the plan
- Software quality assurance
  - Activities required to ensure quality
- Software configuration management
  - Manage effects of change
- Technical Reviews
  - Uncover errors before going to next activity
- Formal technical reviews
  - Assess work products to uncover errors

Umbrella Activities (contd)

- Risk management
  - Assess risks that may affect quality
- Measurement – process, project, product
- Reusability management (component reuse)
- Work product preparation and production
  - Models, documents, logs, forms, lists…

etc.

How Process Models Differ?

While all Process Models take same framework and umbrella activities, they differ with regard to

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

A Simple Practical Definition

A Process defines who is doing what, when, and how to reach a certain goal

-Ivar Jacobson, Grady Booch, and James Rumbaugh

Prescriptive and agile processes

- Prescriptive processes are processes where all of the process activities are planned in advance and progress is measured against this plan.
- In agile processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.
- In practice, most practical processes may include elements of both plan-driven and agile approaches.

- ***There are NO right or wrong software processes.***

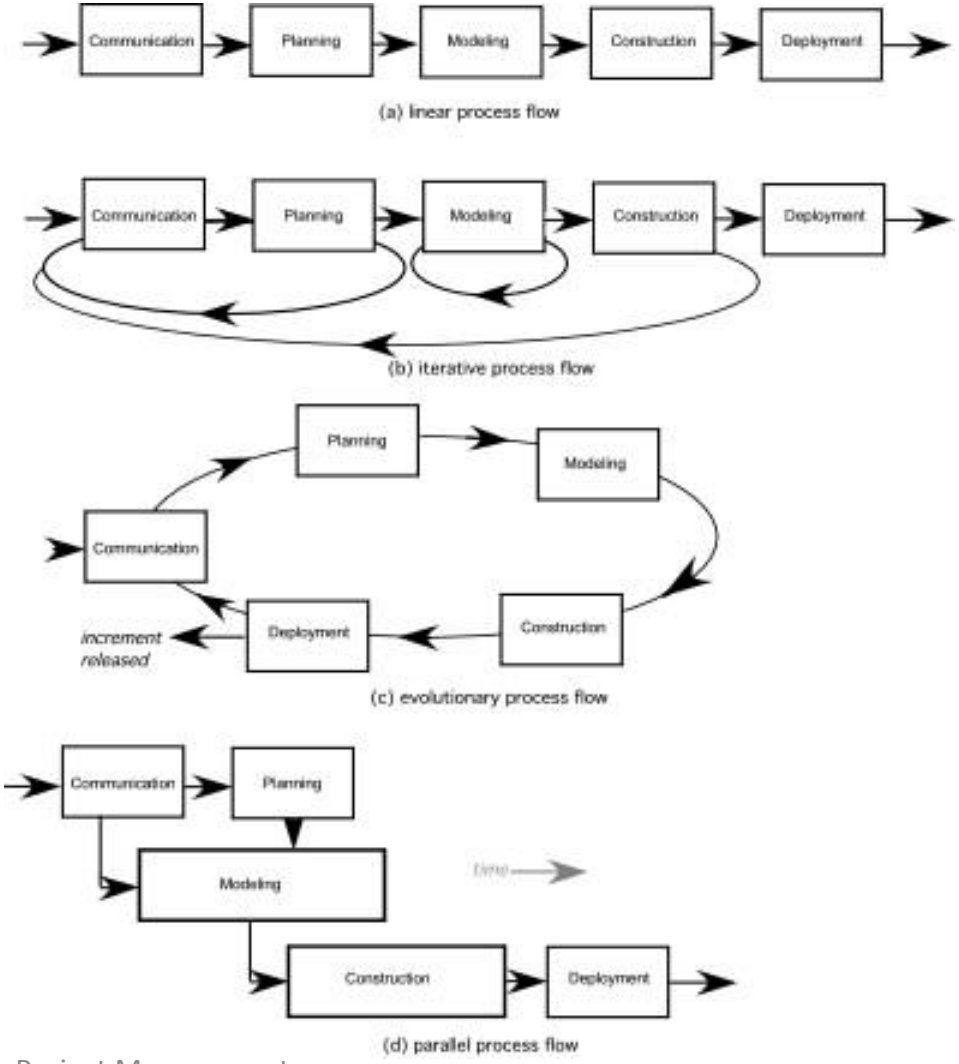
Prescriptive Process Model

- Defines a distinct set of activities, actions, tasks, milestones, and work products that are required to engineer high-quality software

- The activities may be linear, incremental, or evolutionary

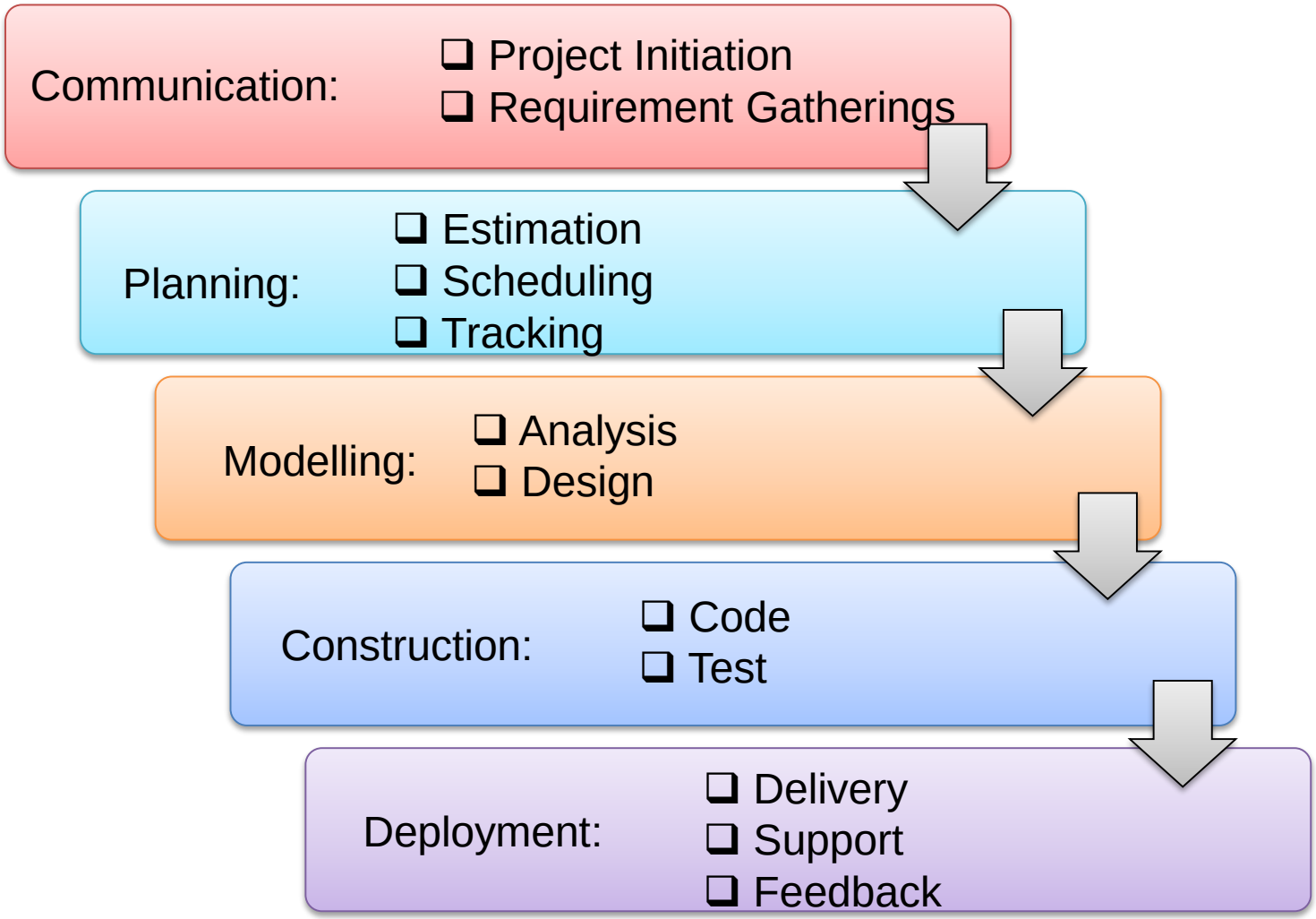
Process Models

An important variation among process models comes from flow of activities



Waterfall Model

(Diagram)



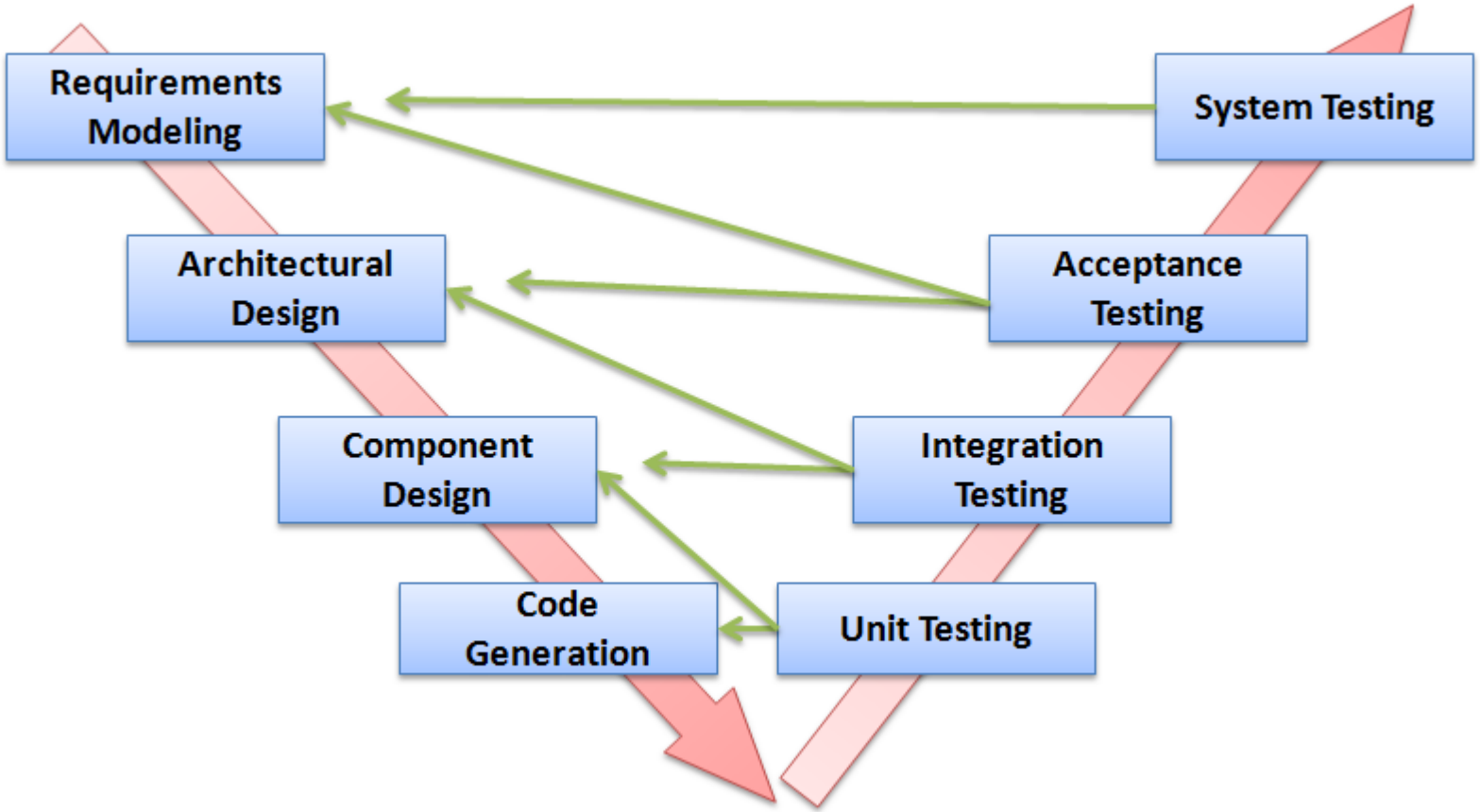
Critique of the waterfall model

- 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 makes no allowances for prototyping.
- Assumes understanding of problem and full requirements early on
- It implies that you can get the requirements right by simply writing them down and reviewing them.
- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.

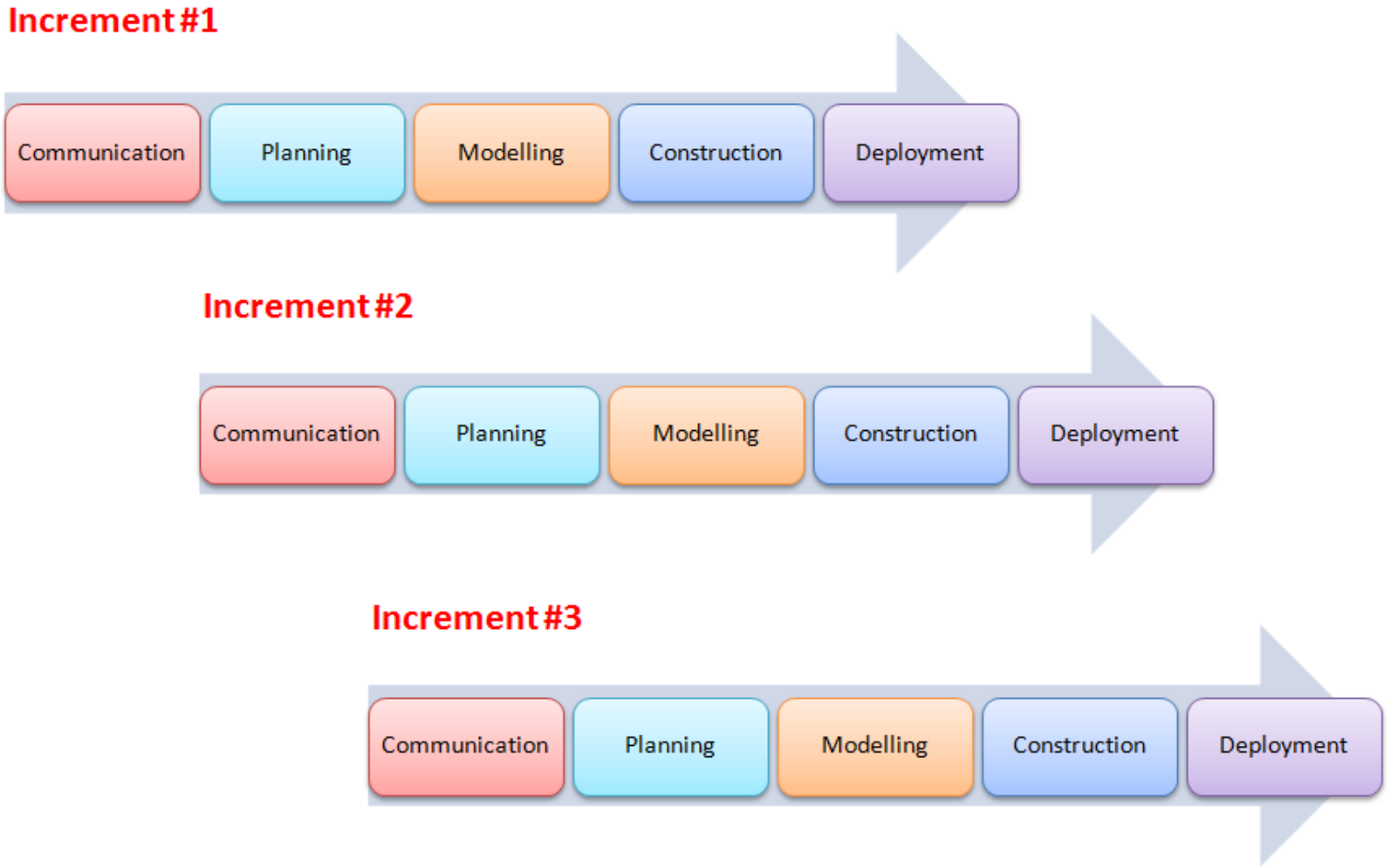
Critique of Waterfall Model continued

- Follows systematic approach to development
- The model implies that once the product is finished, everything else is maintenance.
- Assumes patience from customer
- Surprises at the end are very expensive
- Some teams sit idle 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.

V-Model


Requirements Modeling
System Testing
Architectural Design
Acceptance Testing
Component Design
Integration Testing
Code Generation
Unit Testing

Incremental Model



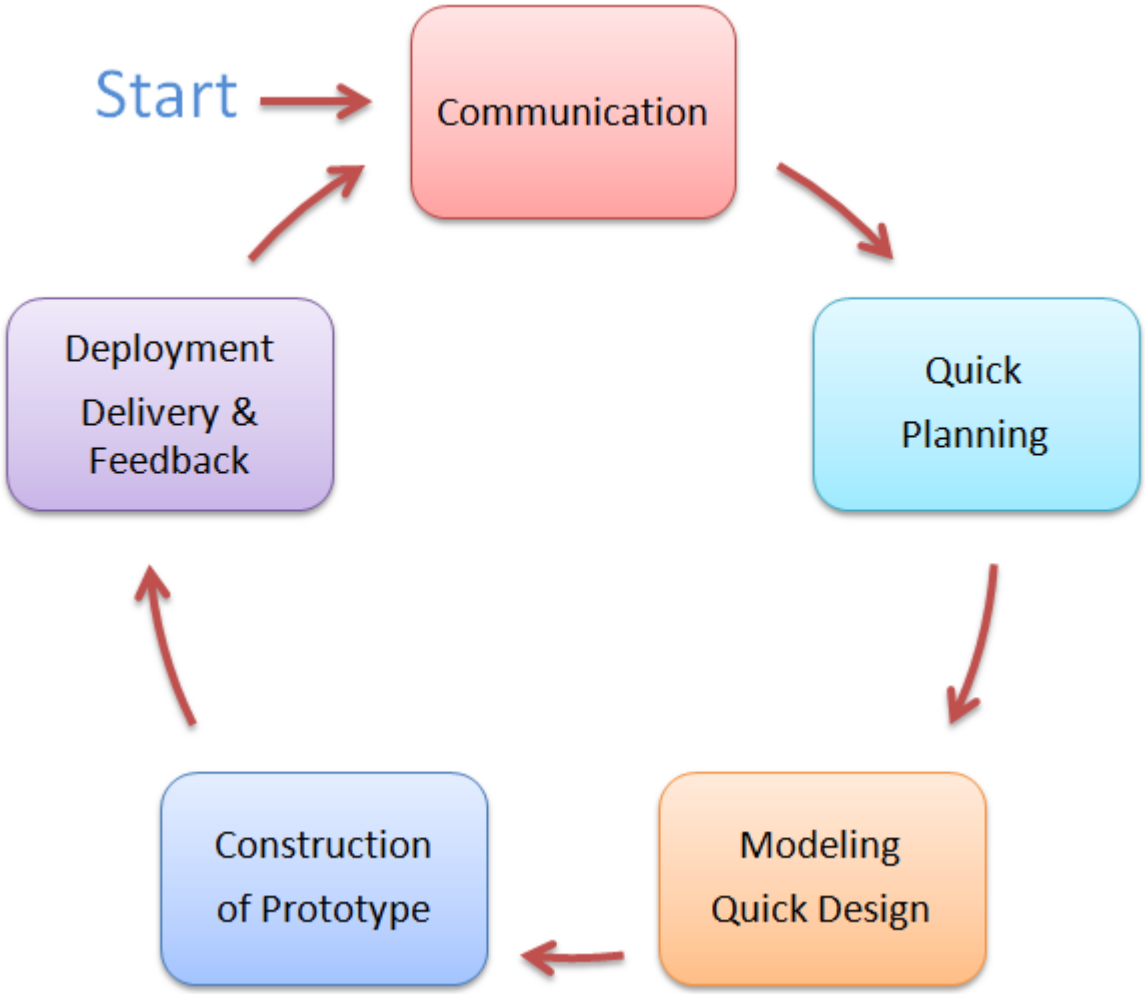
The Incremental Model

- 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
- First Increment is used or evaluated
- 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

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.
- The highest priority system services tend to receive the most testing.

Prototyping Model



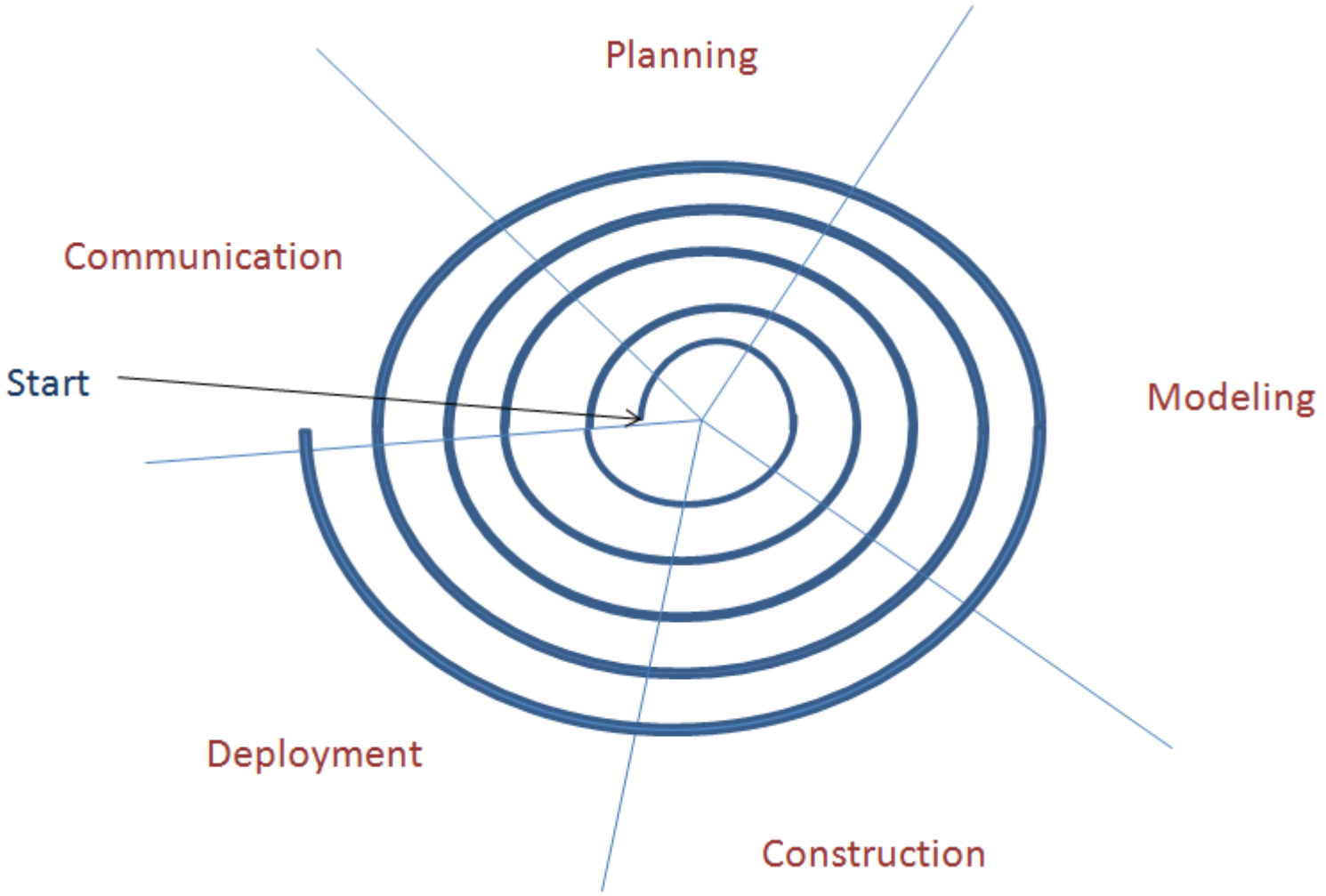
Prototyping Model

- Follows an evolutionary and iterative approach
- Used when requirements are not well understood
- Serves as a mechanism for identifying software requirements
- Focuses on those aspects of the software that are visible to the customer/user
- Feedback is used to refine the prototype

Prototyping Model (Potential Problems)

- The customer sees a "working version" of the software, wants to stop all development and then buy the prototype after a "few fixes" are made
- Developers often make implementation compromises to get the software running quickly (e.g., language choice, user interface, operating system choice, inefficient algorithms)
- Important Considerations:
  - Define the rules up front on the final disposition of the prototype before it is built
  - In most circumstances, plan to discard the prototype and engineer the actual production software with a goal toward quality

Spiral Model



Spiral Model

- Proposed by Dr. Barry Boehm in 1988 while working at TRW
- Follows an evolutionary approach
- Used when requirements are not well understood and risks are high
- Inner spirals focus on identifying software requirements and project risks; may also incorporate prototyping
- Outer spirals take on a classical waterfall approach after requirements have been defined, but permit iterative growth of the software
- Operates as a risk-driven model…a go/no-go decision occurs after each complete spiral in order to react to risk determinations
- Requires considerable expertise in risk assessment
- Serves as a realistic model for large-scale software development

Specialized Process Models

Component-based Development Model

- Consists of the following process steps
  - Available component-based products are researched and evaluated for the application domain in question
  - Component integration issues are considered
  - A software architecture is designed to accommodate the components
  - Components are integrated into the architecture
  - Comprehensive testing is conducted to ensure proper functionality
- Relies on a robust component library
- Capitalizes on software reuse, which leads to documented savings in project cost and time

Formal Methods Model

- Encompasses a set of activities that leads to formal mathematical specification of computer software
- Enables a software engineer to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation
- Ambiguity, incompleteness, and inconsistency can be discovered and corrected more easily through mathematical analysis
- Offers the promise of defect-free software
- Used often when building safety-critical systems

Formal Methods Model

- Development of formal methods is currently quite time-consuming and expensive
- Because few software developers have the necessary background to apply formal methods, extensive training is required
- It is difficult to use the models as a communication mechanism for technically unsophisticated customers

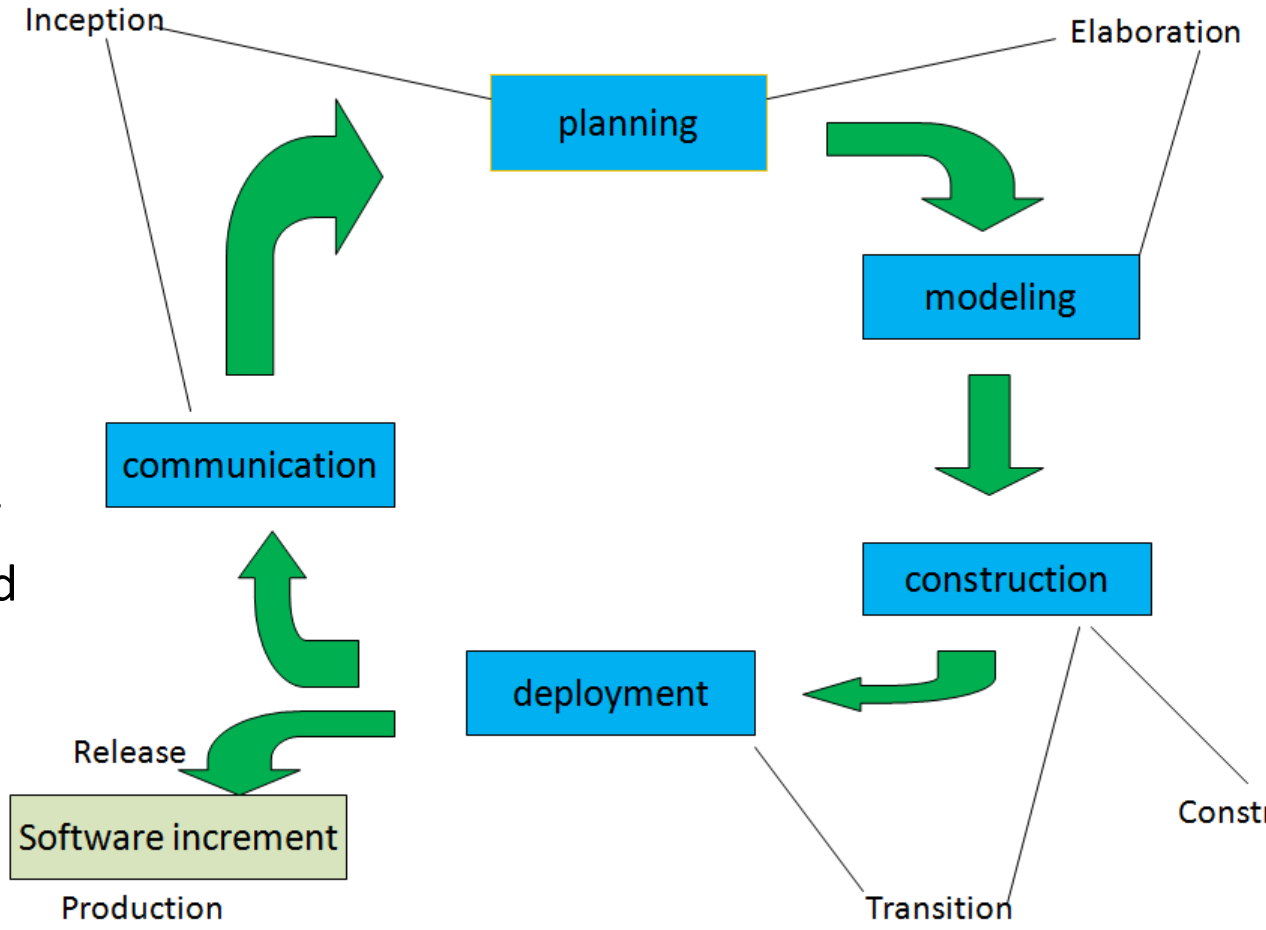
Aspect-Oriented Software Development

Provides a process and methodological approach for defining, specifying, designing, and constructing *aspects* such as

- user interfaces,
- security,
- memory management

that impact many parts of the system being developed

Unified Process



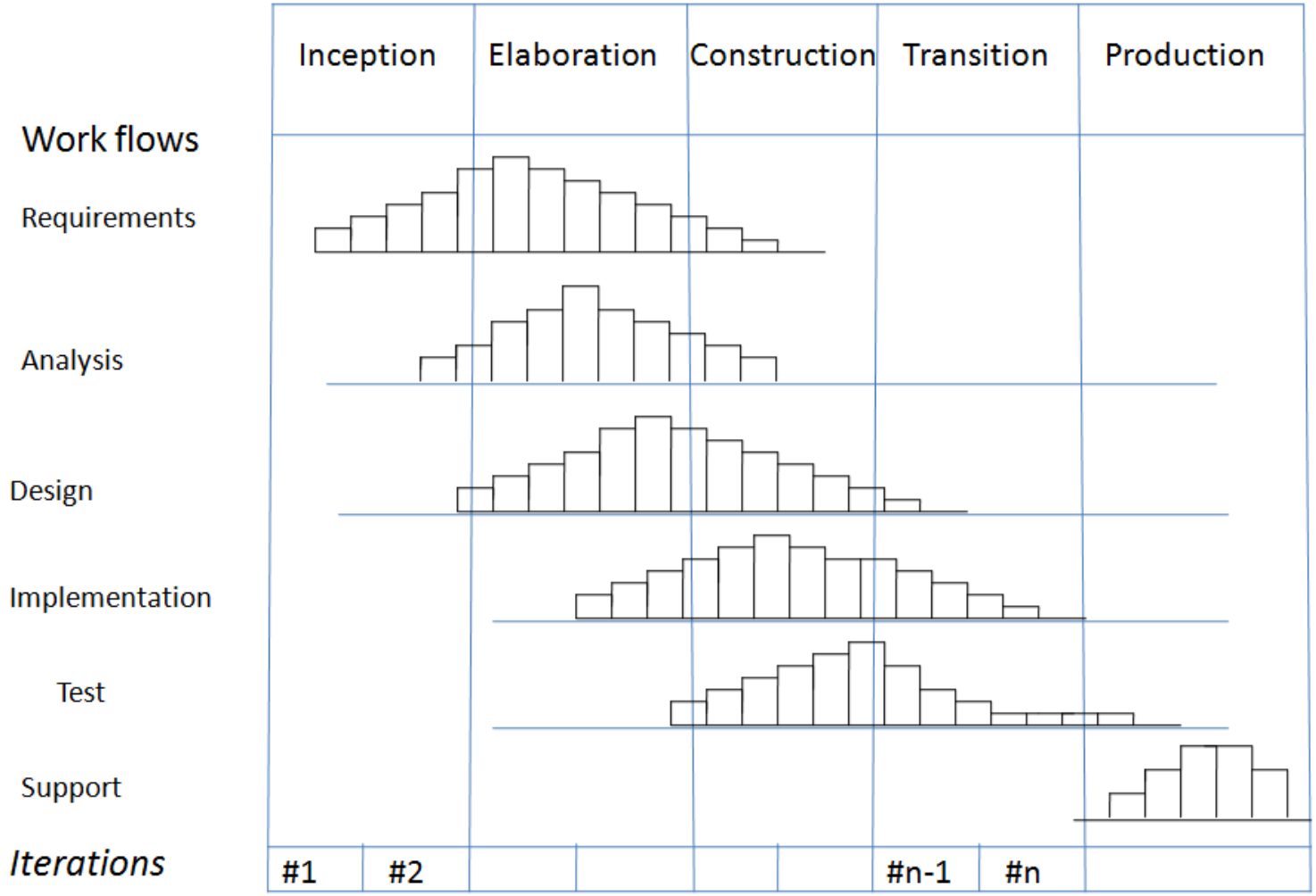
Phases of the Unified Process

The Unified Process is a framework for object-oriented software engineering using UML proposed by Booch, Jacobson, and Rumbaugh

Unified Process

Inception Phase	Elaboration Phase	Construction Phase	Transition Phase
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	Refines and expands the preliminary use cases Expands the architectural representation to include five views viz. Use-case, Analysis, Design, Implementation, Deployment Leads to an executable architectural baseline that demonstrates the viability of the architecture but does all features and functions.	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 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	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. At the conclusion of this phase, the software increment becomes a usable software release

UP Phases


Inception
Elaboration
Construction
Transition
Production
Work flows
Requirements
Analysis
Design
Implementation
Test
Support
Iterations
#1
#2
#n-1
#n

Unified Process Work Products

Inception phases

Vision document

Initial use case model
Initial project glossary
Initial business case
Initial risk assessment
Project plan, phase
And iteration
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 work flows
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.

Personal Software Process (PSP)

- Recommends five framework activities:
  - Planning
  - High-level design
  - High-level design review
  - Development
  - Postmortem
- stresses the need for each software engineer to identify errors early and as important, to understand the types of errors

Team Software Process (TSP)

- Each project is “launched” using a “script” that defines the tasks to be accomplished
- Teams are self-directed
- Measurement is encouraged
- Measures are analyzed with the intent of improving the team process

Team Software Process (TSP)

- Recommends five framework activities for TSP:
  - Project Launch
  - High-level design
  - Implementation
  - Integration and Testing
  - Postmortem

Another View on Software Process

- Because software, like all capital, is embodied knowledge, and because that knowledge is initially dispersed, tacit, latent, and incomplete in large measure, software development is a social learning process. The process is a dialogue in which the knowledge that must become software is brought together and embodied in the software.

-Howard Baetjer (economist)

Agile methods

- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
  - Focus on the code rather than the design
  - Are based on an iterative approach to software development
  - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.

- The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

Sommerville, I., Software Engineering, Pearson Education, 9th Ed., 2010

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

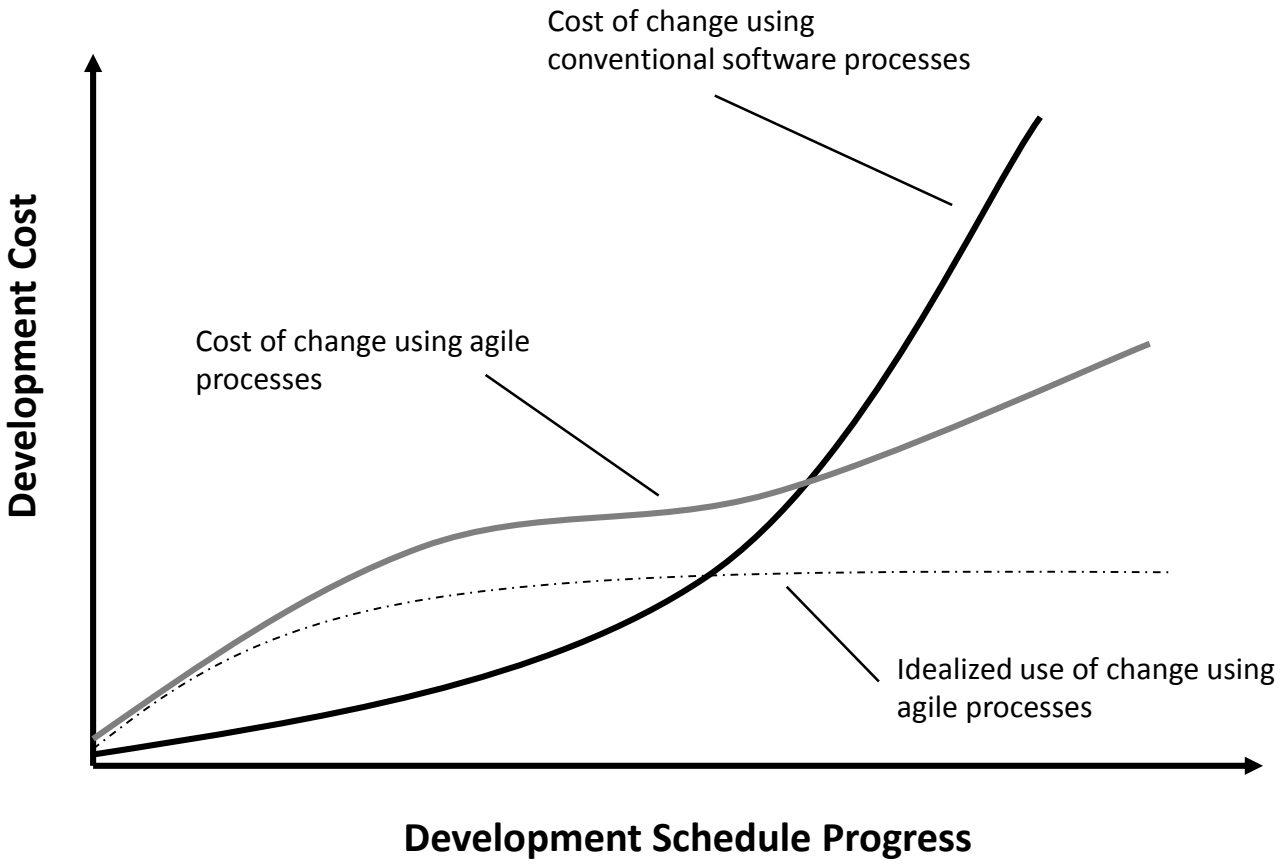
An Agile Process

- 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 Human Factors

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

Cost of Change



An Agile Concept - Time Boxing

... teams are managing the triple constraints that face any organisation - time, quality, scope. When using a fixed duration, we are telling everyone involved, 'time is urgent and we are going to include as much as we can within this time framework.' Since quality cannot be compromised, the only variable is scope. 'Time boxing' creates a sense of urgency and criticality for the entire organization

—Mark P. Dangelo, Author: *Innovative relevance*

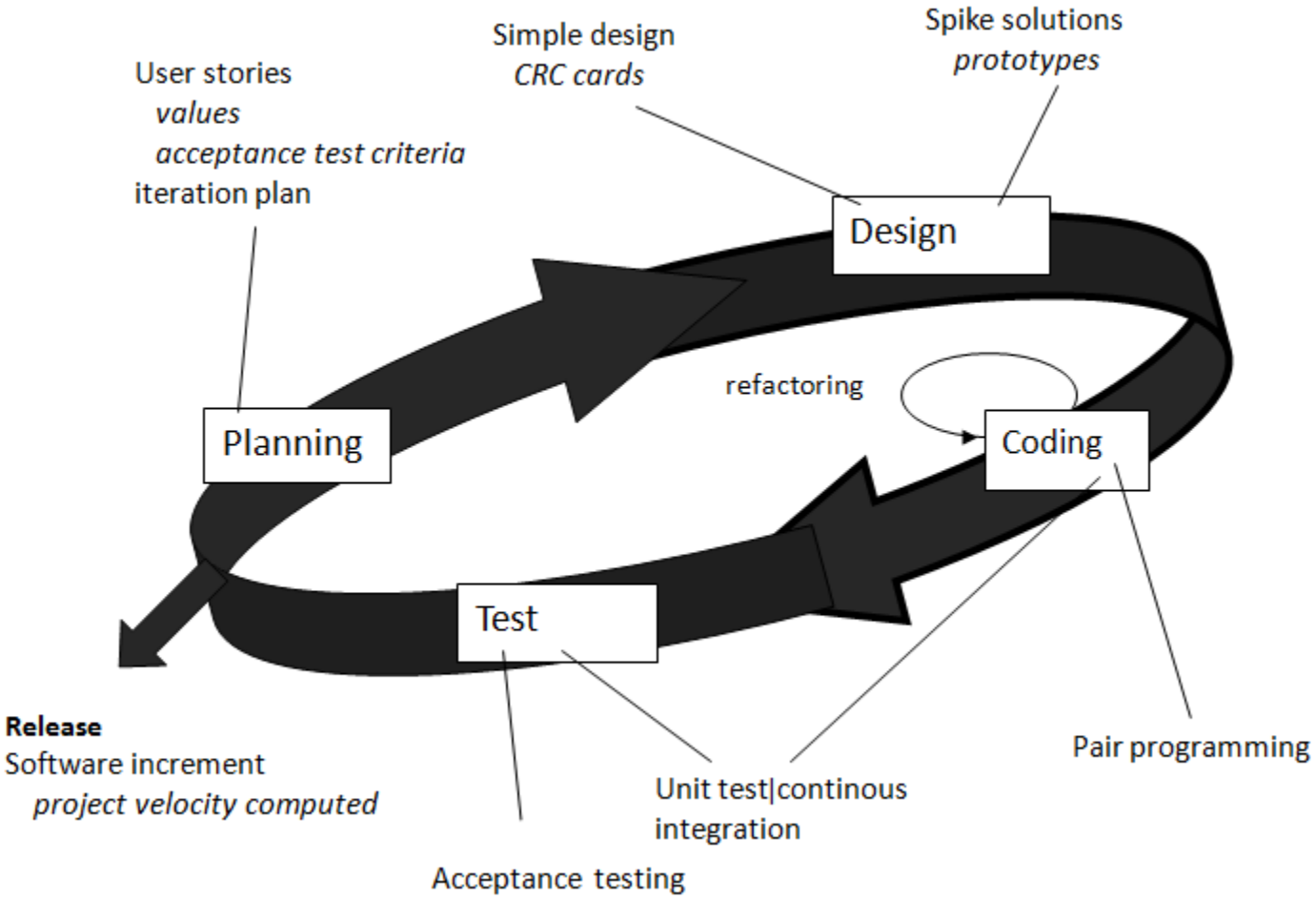
Extreme Programming (XP)

- 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 form a *deliverable increment*
  - A *commitment* is made on delivery date
  - After the first increment "*project velocity*" is used to help define subsequent delivery dates for other increments

Extreme Programming (XP)

- XP Design
  - Follows the *KIS principle*
  - Encourage the use of *CRC cards*
  - For difficult design problems, suggests the creation of "*spike solutions*"—a design prototype
  - Encourages "*refactoring*"—an iterative refinement of the internal program design
- XP Coding
  - Recommends the *construction of a unit test* for a store *before* coding commences
  - Encourages "*pair programming*"
- XP Testing
  - All *unit tests are executed daily*
  - "*Acceptance tests*" are defined by the customer and executed to assess customer visible functionality

Extreme Programming (XP)



Refactoring

- Programming team look for possible software improvements and make these improvements even where there is no immediate need for them.
- This improves the understandability of the software and so reduces the need for documentation.
- Changes are easier to make because the code is well-structured and clear.
- However, some changes requires architecture refactoring and this is much more expensive.

Examples of refactoring

- Re-organization of a class hierarchy to remove duplicate code.
- Tidying up and renaming attributes and methods to make them easier to understand.
- The replacement of inline code with calls to methods that have been included in a program library.

Pair programming

- In pair programming, programmers sit together at the same workstation to develop the software.
- Pairs are created dynamically so that all team members work with each other during the development process.
- The sharing of knowledge that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- Pair programming is not necessarily inefficient and there is evidence that a pair working together is more efficient than 2 programmers working separately.

Advantages of pair programming

- It supports the idea of collective ownership and responsibility for the system.
  - Individuals are not held responsible for problems with the code. Instead, the team has collective responsibility for resolving these problems.
- It acts as an informal review process because each line of code is looked at by at least two people.
- It helps support refactoring, which is a process of software improvement.
  - Where pair programming and collective ownership are used, others benefit immediately from the refactoring so they are likely to support the process.

Criticism of XP

Concerns expressed by critics w.r.t. XP include

- Requirements volatility
  - -rework may be unmanageable
- Conflicting customer needs
  - -team cannot reconcile conflicting demands
- Requirements are expressed informally
  - -omissions, inconsistencies, errors
- Lack of formal design
  - -Without architectural design, structure of the software lacks quality and maintainability

Scrum

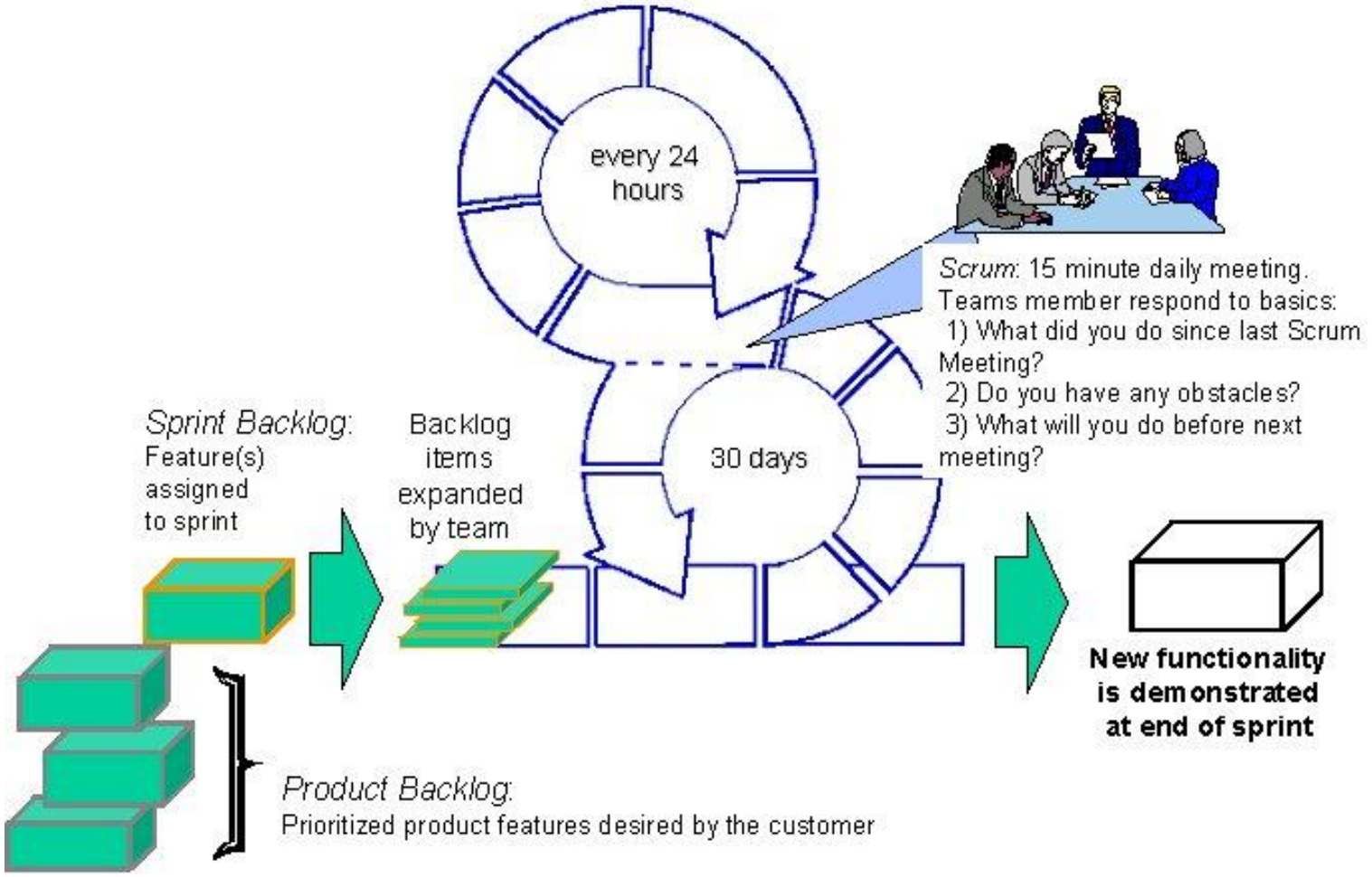
"Scrum assumes that the systems development process is an unpredictable, complicated process that can only be roughly described as an overall progression."

"Scrum is an enhancement of the commonly used iterative/incremental object-oriented development cycle"

-Jeff Sutherland

- Scrum—distinguishing features
  - Development work partitioned into packets makes up "Backlog"
  - 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



Lean software development

Adapts the principles of lean manufacturing to software engineering

Lean Principles	Software engineering actions
•Eliminate Waste	•Add no extraneous features
•Build Quality in	•Ensure effective testing
•Create Knowledge	•Improve mechanisms to find information
•Defer Commitment	•Assess cost & schedule impact for changes
•Deliver Fast	•Remove superfluous process steps
•Respect People	•Streamline stakeholder communication
•Optimize the Whole	•Reduce time required to process a request

Agile Unified Process

- Adopts “serial in the large” and “iterative in the small” philosophy suggested by Scott Ambler
- Adopts UP phased activities – inception, elaboration, construction, and transition
- Within each of the activity, team iterates to achieve agility.
- UML representations are used, though modeling is kept to bare minimum

Agile Modeling

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

Agile Process and Documentation

There are two keys to successful documentation on agile projects.

- Finding the point of "just enough" documentation. This is difficult to determine and will vary by project. Fortunately, the iterative nature of agile development allows you to experiment until you get it right.

- Not get attached to it or have unrealistic hopes of keeping it updated.

Documentation must be created to serve a specific purpose, and after it has served that purpose you'll all probably have more important things to do than keep updating the documents. It may seem counterintuitive, but it's often better to produce fresh documentation the next time some is clearly required. A side benefit of starting over each time you need to document part of your project is that it's great incentive to keep your documentation efficient!

Agile Process with Offshore Development

Martin Fowler suggests following successful practices:

- Use Continuous Integration to Avoid Integration Headaches
- Have Each Site Send Ambassadors to the Other Sites
- Use Contact Visits to build trust
- Don't Underestimate the Culture Change
- Use wikis to contain common information
- Use Test Scripts to Help Understand the Requirements
- Use Regular Builds to Get Feedback on Functionality

Agile Process with Offshore Development
(contd)

Martin Fowler suggests following successful practices:

- Use Regular Short Status Meetings
- Use Short Iterations
- Use an Iteration Planning Meeting that's Tailored for Remote Sites
- When Moving a Code Base, Bug Fixing Makes a Good Start
- Separate teams by functionality not activity
- Expect to need more documents.
- Get multiple communication modes working early

Article by Martin Fowler, Author and Member, Agile Alliance

Challenges with agile methods

- It can be difficult to keep the interest of customers who are involved in the process.
- Team members may be unsuited to the intense involvement that characterises agile methods.
- Prioritising changes can be difficult where there are multiple stakeholders.
- Maintaining simplicity requires extra work.
- Contracts may be a problem as with other approaches to iterative development.

Agile methods and software maintenance

- Most organizations spend more on maintaining existing software than they do on new software development. So, if agile methods are to be successful, they have to support maintenance as well as original development.
- Two key issues:
  - Are systems that are developed using an agile approach maintainable, given the emphasis in the development process of minimizing formal documentation?
  - Can agile methods be used effectively for evolving a system in response to customer change requests?
- Problems may arise if original development team cannot be maintained.

Large systems development

- Large systems are usually collections of separate, communicating systems, where separate teams develop each system. Frequently, these teams are working in different places, sometimes in different time zones.
- Large systems are ‘brownfield systems’, that is they include and interact with a number of existing systems. Many of the system requirements are concerned with this interaction and so don’t really lend themselves to flexibility and incremental development.
- Where several systems are integrated to create a system, a significant fraction of the development is concerned with system configuration rather than original code development.

Large systems development

- Large systems and their development processes are often constrained by external rules and regulations limiting the way that they can be developed.
- Large systems have a long procurement and development time. It is difficult to maintain coherent teams who know about the system over that period as, inevitably, people move on to other jobs and projects.
- Large systems usually have a diverse set of stakeholders.It is practically impossible to involve all of these different stakeholders in the development process.

Summary

- The roadmap to building high quality software products is software process.
- Software processes are adapted to meet the needs of software engineers and managers as they undertake the development of a software product.
- A software process provides a framework for managing activities that can very easily get out of control.
- Modern software processes must be agile, demanding only those activities, controls, and work products appropriate for team or product.
- Different types of projects require different software processes.
- The software engineer's work products (programs, documentation, data) are produced as consequences of the activities defined by the software process.
- The best indicators of how well a software process has worked are the quality, timeliness, and long-term viability of the resulting software product.

Thank You

Any Questions?