CSc 131 – Computer Software Engineering

What is a software process?

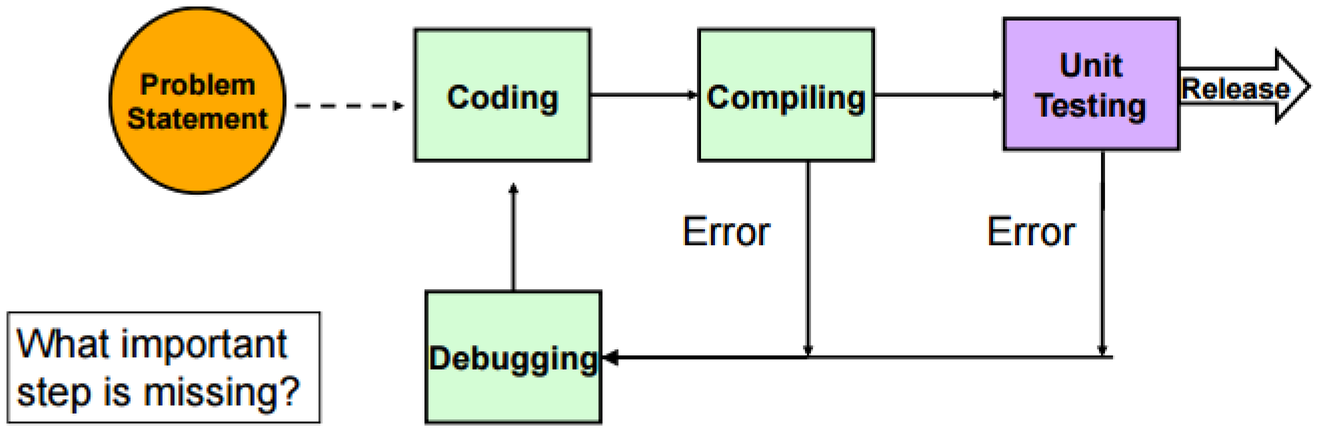
- Software Process: A structured set of activities used to develop a software system.

- It is a description of
  - what tasks need to be performed in
  - what sequence under
  - what conditions by
  - whom to

achieve the “desired results.”

- Desired results: high quality software product.

A Simple process (i.e CSC 20 or CSC 60)



- Suitable for student projects
- Students encounter problems when
  - some steps are skipped
  - problem statement is not well stated or understood

As projects get larger and more complex . . .

- The problem domain (also called application domain)/Solution domain becomes difficult

- We need more people and more coordination
  - Problem statement needs to be expanded and clarified (requirements/specifications)
  - Need a good, well-documented design
  - Need to make sure various developers can work together (tools, documentation)
  - Need to ensure adequate testing is done

- We need a more detailed process

Four primary software engineering activities

- There are many different software processes but all involve these activities:
  - **Specification** – defining what the system should do (stating the requirements)
  - **Development** – defining the organization of the system (aka the design) and implementing the system
  - **Validation** – checking that the system does what the customer wants
  - **Evolution** – changing the system in response to customer needs.

- Different software processes do the activities in different ways.

Software specification

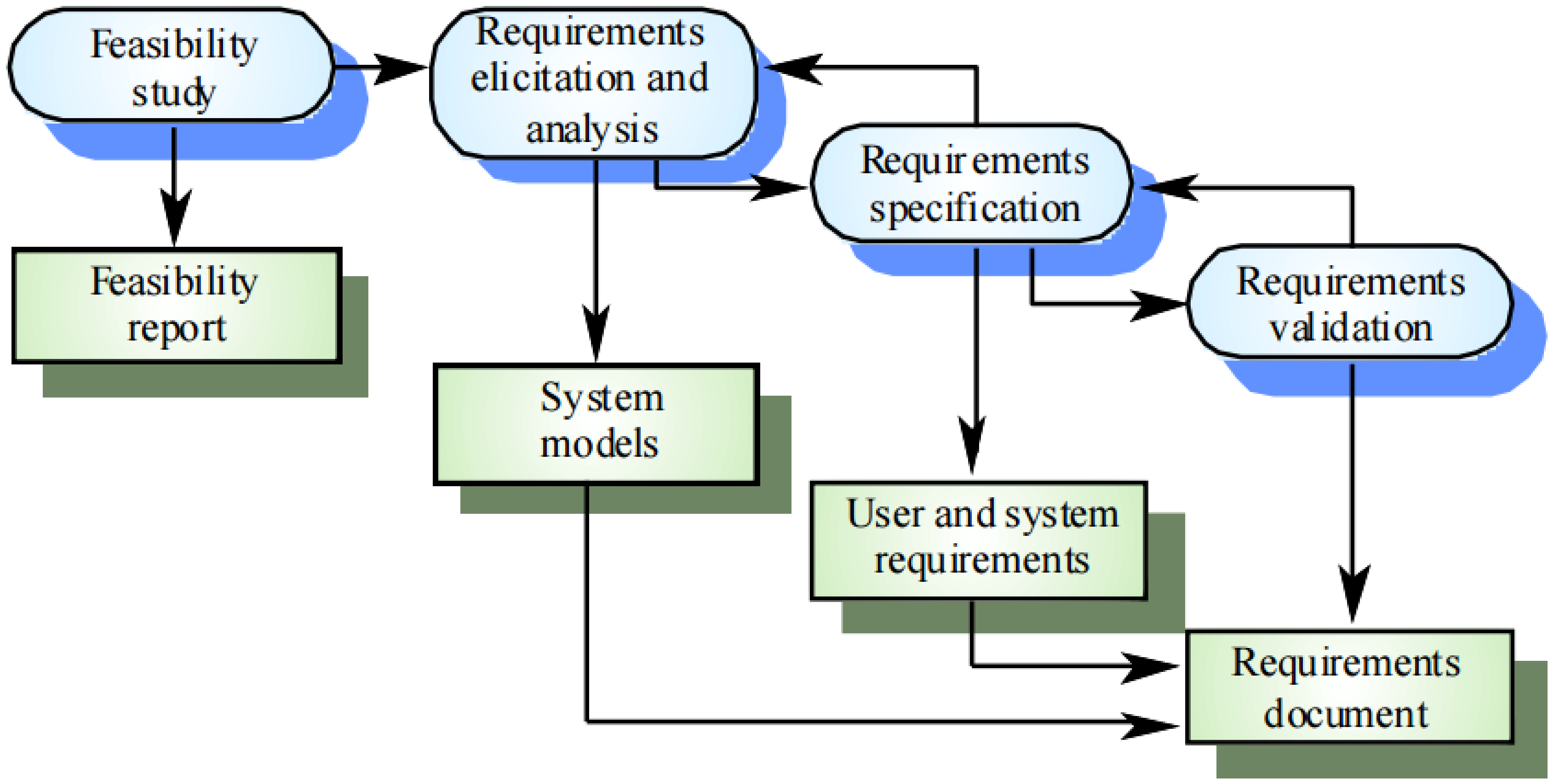
- The process of establishing the requirements:

- the **features/functions** that are required by the users
- the **constraints** on the system's operation and development.

- Requirements engineering process:

- Requirements elicitation and analysis
    - ❖ What do the customers/users require or expect from the system?
    - ❖ May observe existing systems, develop models or prototype
- Requirements specification
    - ❖ Defining the requirements in detail and documenting them.
- Requirements validation
    - ❖ Checking them for clarity, consistency, completeness, etc.

The Requirements Engineering Process



Software development: design and implementation

- Converting the requirements into an executable system.
- Software design

  - Description of the structure of the software using various models (describing the subcomponents, how they interact, etc)
- Implementation

  - Translate the design into an executable program
- Design and implementation are closely related and often interleaved.

Software validation

- Verification and validation (V & V) is intended to
  - show that a system conforms to its specification and
  - meets the needs of the system customer.
- Program testing:
  - executing the system over simulated data, ensuring the results are correct.
  - Automation verification
- Inspections and reviews:
  - humans analyze models and source code looking for errors or problems

Software Evolution

- After the software has been released, it must be kept up to date.
  - Customers require new functions
  - Defects must be repaired
  - Must adapt to new platforms and machines
- Activities include:
  - Modifying requirements/specifications (as needed)
  - Modifying design (as needed)
  - Modifying the implementation
  - Retesting, adding new test cases.

Traditional Software process models (or frameworks, or paradigms)

- A software process model:
  - is a simplified (or abstract) representation of a set of specific software processes.
  - must be "extended" with more detail to become an actual software process.

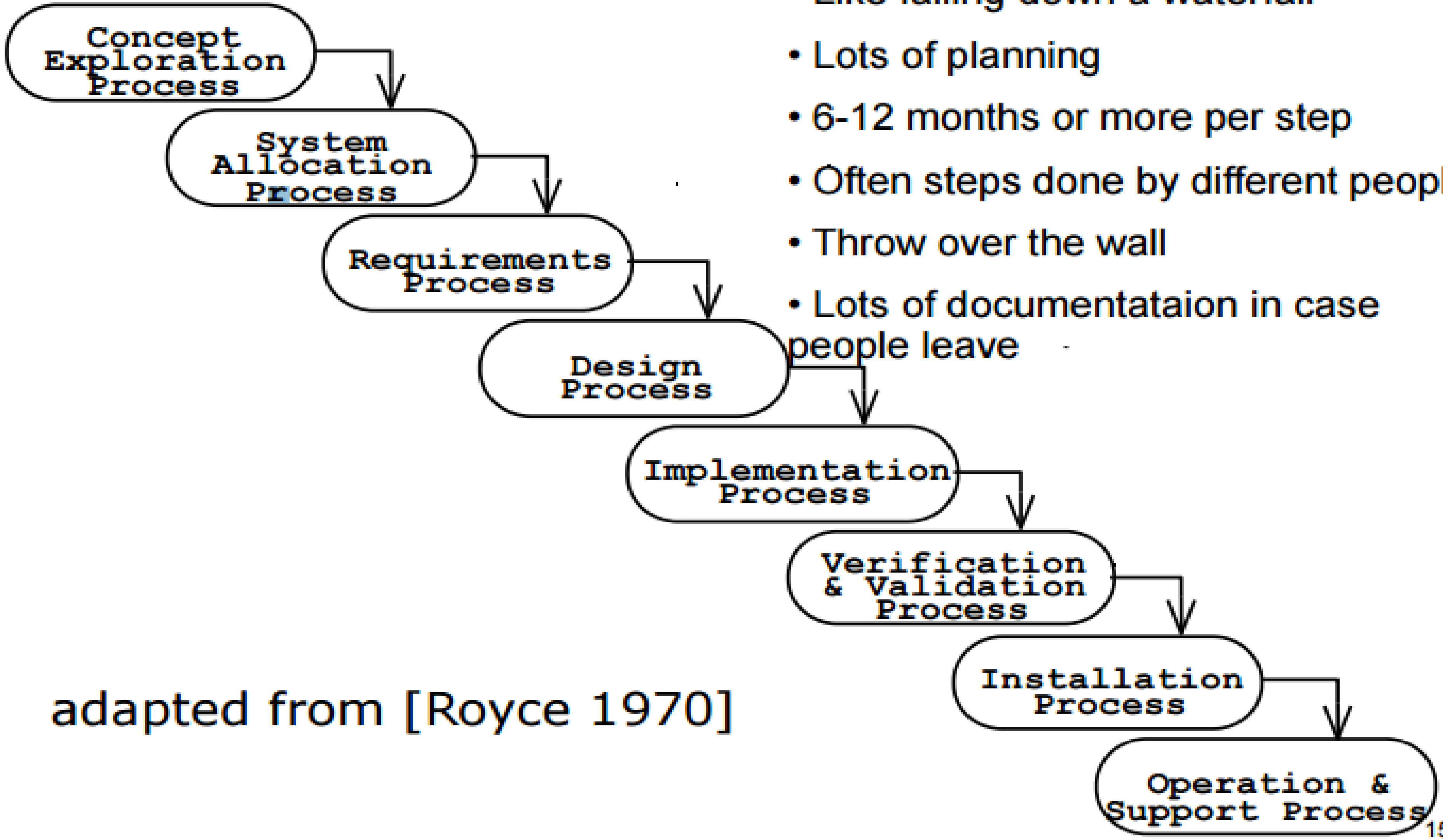
- Traditional software process models:
  A. Waterfall model
  B. Incremental development
  C. Spiral model
  D. Reuse-oriented software engineering

- There are others process modes

Waterfall model

- The waterfall model
  - One of the first published models
  - Separate and distinct phases are performed in sequence.
  - Planning occurs up front: “Plan-driven”
- The separate phases:
  - Requirements definition
  - Software design
  - Implementation
  - Testing
  - Maintenance
- The output of one stage is input to the next.
- Tends to require/generate much documentation.

Waterfall model (Cont)


Concept Exploration Process
System Allocation Process
Requirements Process
Design Process
Implementation Process
Verification & Validation Process
Installation Process
Operation & Support Process

- Like falling down a waterfall
- Lots of planning
- 6-12 months or more per step
- Often steps done by different people
- Throw over the wall
- Lots of documentataion in case people leave

adapted from [Royce 1970]

Waterfall Issues

- Good features:
  - Simple and easy to implement (better than no process)
  - Easy for managers to track the progress of the project
- Can be used for large projects when a system is developed at several sites.
  - Plan-driven nature of the this model helps coordinate the work.
- Main drawback: The difficulty of accommodating changes after the process is underway.
  - Change requires “backtracking”: revising previous step(s), re-work (costly)
  - This model is appropriate only when
    - a) the requirements are well-understood upfront and
    - b) changes will be fairly limited during the design process.
- Customers often need to change the requirements

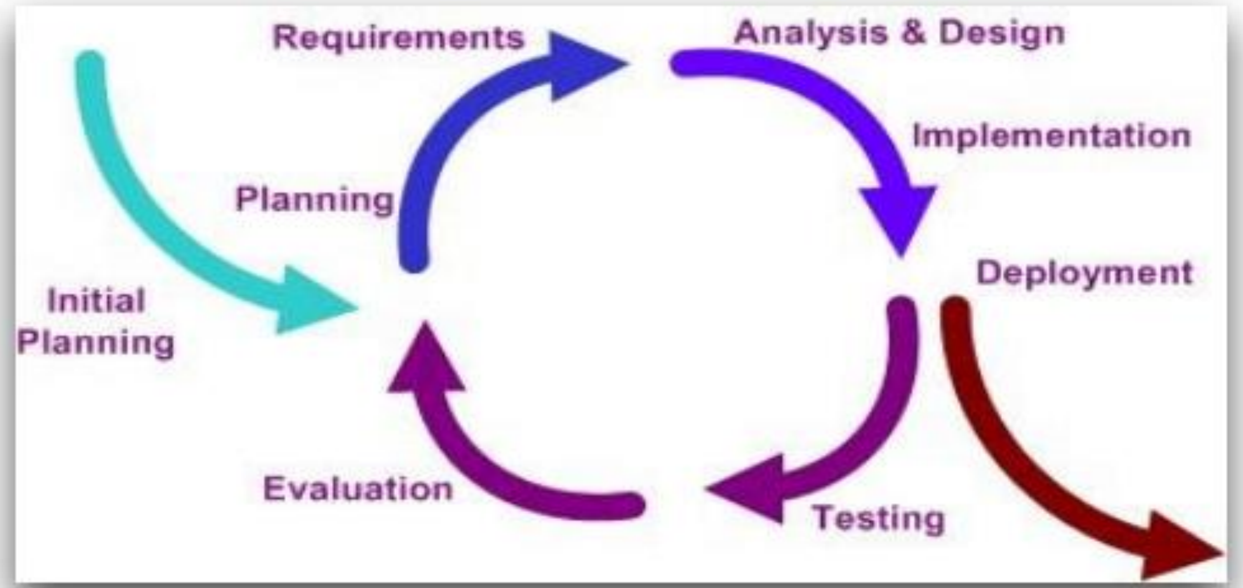
Incremental development

- Several development iterations are performed in sequence.
- Each iteration is a self-contained mini-project composed of activities such as requirements analysis, design, programming, and test

From: Craig Larman,
Agile and Iterative Development - A Manager's Guide

- Each iteration produces a new version (called an increment).
    - Each version adds functionality to the previous version.
    - Only the final version is a complete system.
- Each version is exposed to the user for feedback
    - The customer may come to the developers' site for demos/testing.
    - If the intermediate versions are given to the customer, it is called Incremental Delivery.

Incremental development (Cont)



Each time around the loop produces a new version of the software.

Incremental development benefits

- Reduces cost of accommodating changing customer requirements.
  - Early versions are incomplete, so less re-work to do.
  - May require no changes to current version (add to future version).
- It is easier to get customer feedback.
  - Users understand a working incremental release better than documents from the specification or design phase.
- Does not need to be planned entirely up front.
- Early versions can implement the most important, urgent, or risky features

Incremental development problems

- The process is not visible
  - there's less process documentation, so it's difficult to measure progress.
  - may not know how many more increments are required.
- Difficult to design and implement common facilities needed by all versions
- System structure tends to degrade as new increments are added.
  - this makes the code more difficult to modify each time.
  - UNLESS time and money are spent on refactoring to improve the software.
  - Refactoring: disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.
  - Modifying a program to improve its structure, reduce its complexity, or make it easier to understand.

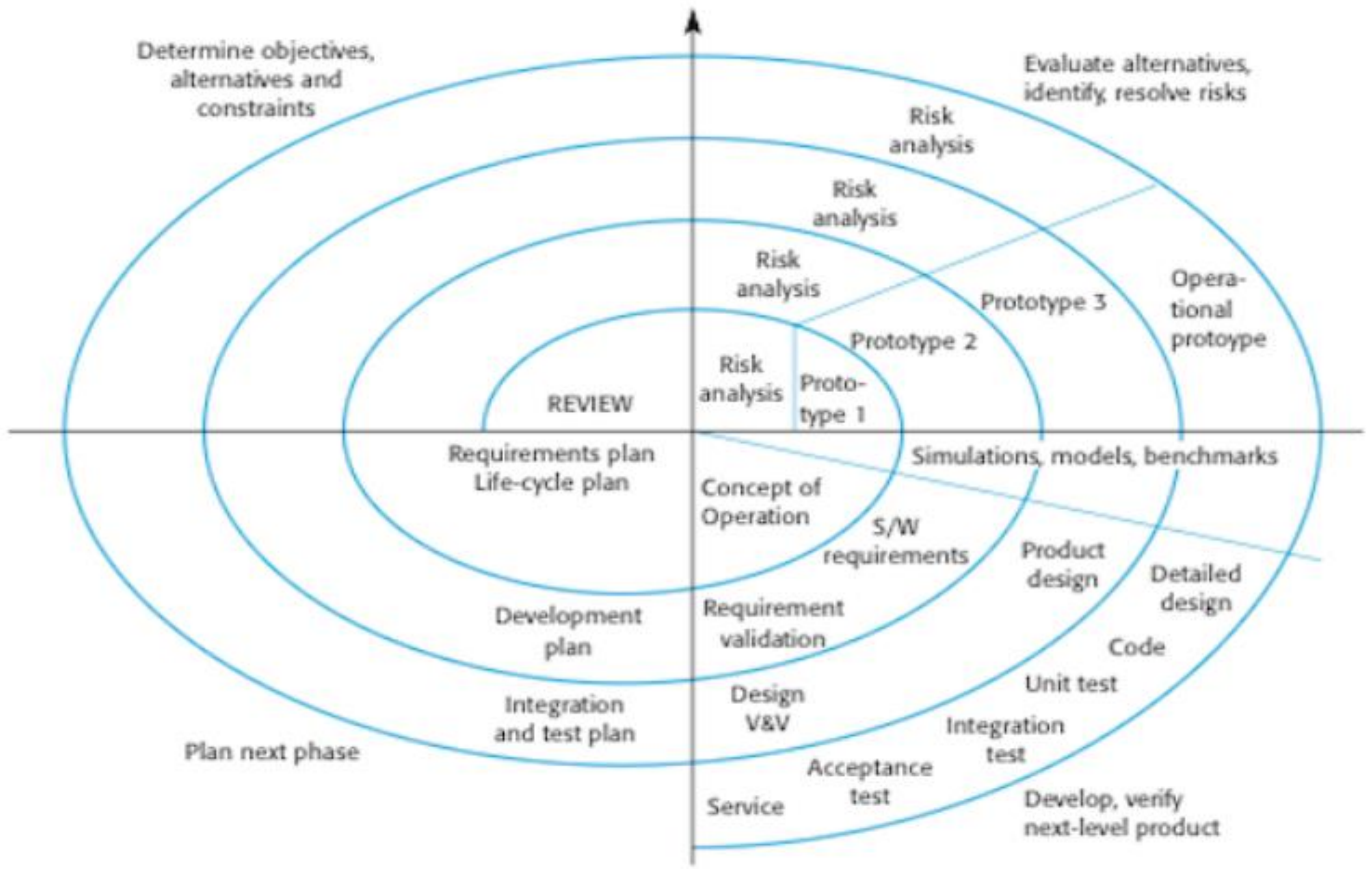
Spiral Model

Proposed by Barry Boehm in 1988.

- Process represented as a spiral
  - Each loop represents a phase in the process.
  - Content of each phase is not predetermined, plan as you go.
- Risks are explicitly assessed and resolved.
  - Assumes need for change are a result of project risks.
- Sectors of the model:
  - Identify objectives, alternatives and constraints.
  - Evaluate and reduce risk (may develop prototype).
  - Development and validation
  - Plan next phase (after review of current phase)

Spiral Model

Boehm's spiral model of the software process



Spiral Model Issues

- Good for high-risk projects.
  - Often used in combination with other process models.
- In practice, the model is rarely used as published.
- Somewhat similar to incremental development, but
  - Risk assessment is incorporated into the process
  - Development is not required to be incremental:
    - ✦ prototypes and results of previous loops can be discarded.
    - ✦ production development could be postponed until the last loop.

Reuse-oriented software engineering

- The system is assembled from existing components.
- Components may be in the form of
  - source code that must be compiled into the final product OR
  - already compiled code that can be accessed from other programs.
- Process stages:
  - Requirements specification (similar to other process models)
  - Component analysis: search for close matches to requirements
  - Requirements modification: to reflect available components
  - System design with reuse: organize framework around acceptable components (may require designing new code).
  - Development and integration: components are integrated along with new code
  - System validation (similar to other process models)

Types of software components for reuse

- Web services (or “API”)
  - Various “functions” available for remote invocation from apps
  - Examples: Weather API from Weather Channel, Endicia Label Server API (labels with USPS postage)
  - http://graphical.weather.gov/xml/SOAP_server/ndfdXML.htm
- Library of Classes: framework
  - Developed as a package to be integrated (compiled) with a component framework such as .NET or J2EE.
  - Example: parsekit for Mac OS X apps (scanners/parsers)
- Stand-alone software systems that are configured for use in a particular environment.
  - often called COTS: “Commercial off the Shelf” systems
  - Example: PeopleSoft, HR management for companies

Advantages and Disadvantages of Reuse-oriented Software Engineering

- Benefits

  - Reduces costs and risks (less code to write, already tested)
  - Usually leads to faster delivery.

- Disadvantages

  - Requirements may have to be compromised (no good matches found)
  - Control over evolution of system is lost (dependent on developers of the components).
  - Cost consideration when updating.

Coping with change

- Change is inevitable in all large software projects.
  - Changing business environments lead to changing requirements
    - ❖ New opportunities and technologies
    - ❖ Changing markets, new competitors
  - New technologies open up new possibilities for improving implementations
  - New platforms require application changes
- Change leads to rework:
  - new requirements lead to more requirements analysis
  - this may lead to redesign of the system or components
  - this may lead to changes to the implementation
  - this may lead to new tests, and re-testing the system

Reducing the costs of rework: two approaches

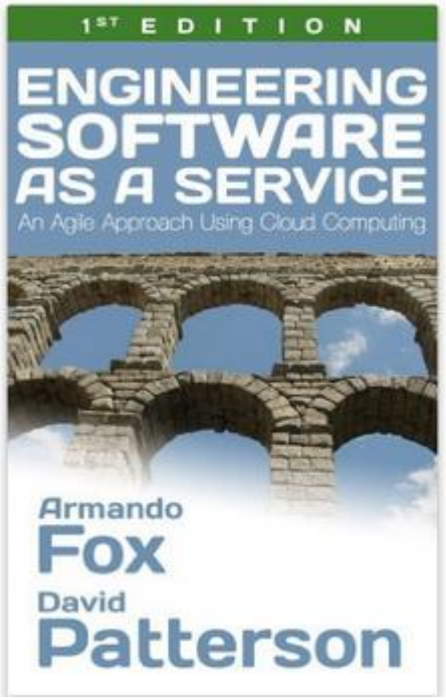
- Change avoidance: include process activities that anticipate possible changes before significant rework is required.
  - i.e. develop a prototype to show some key features of the system to users, let them refine requirements before committing to them.

- Change tolerance: design the process to accommodate change at low cost
  - Use incremental development, get feedback from users.
  - Changes likely apply to most recent increment only, OR
  - Can be incorporated into later increments.

Software Development Processes: The Agile Manifesto

Acknowledgement

- Slides have been based in-part upon original slides/notes of a number of books and people including:
- *Engineering Software as a Service: An Agile Approach Using Cloud Computing*, A. Fox, D. Patterson, Strawberry Canyon LLC, 2013.


1ST EDITION
ENGINEERING SOFTWARE AS A SERVICE
An Agile Approach Using Cloud Computing
Armando Fox
David Patterson

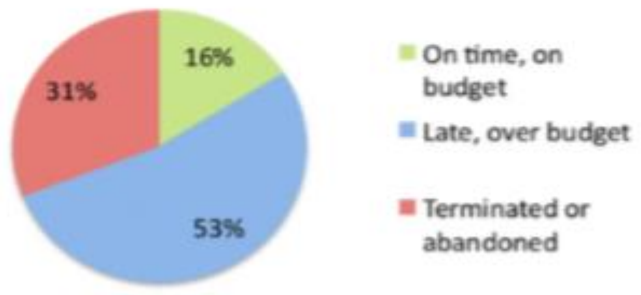
Alternative Process?

- How well can Plan-and-document hit the, schedule, cost, scope & quality targets?
- P&D requires extensive documentation and planning and depends on an experienced manager.
  - Can we build software effectively without careful planning and documentation?
  - How to avoid “just hacking”?

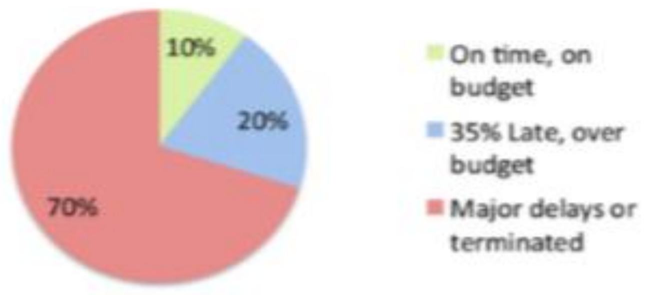
How Well do Plan-and-Document Processes Work?

- IEEE Spectrum "Software Wall of Shame"

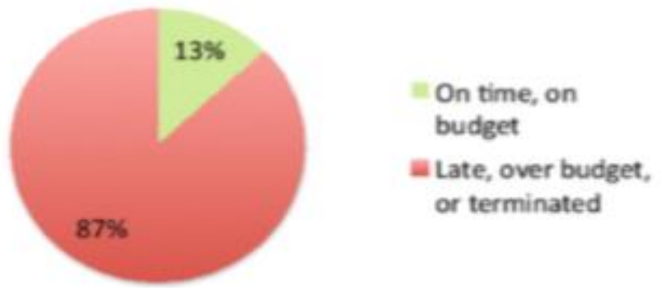
Software Projects (Johnson 1995)



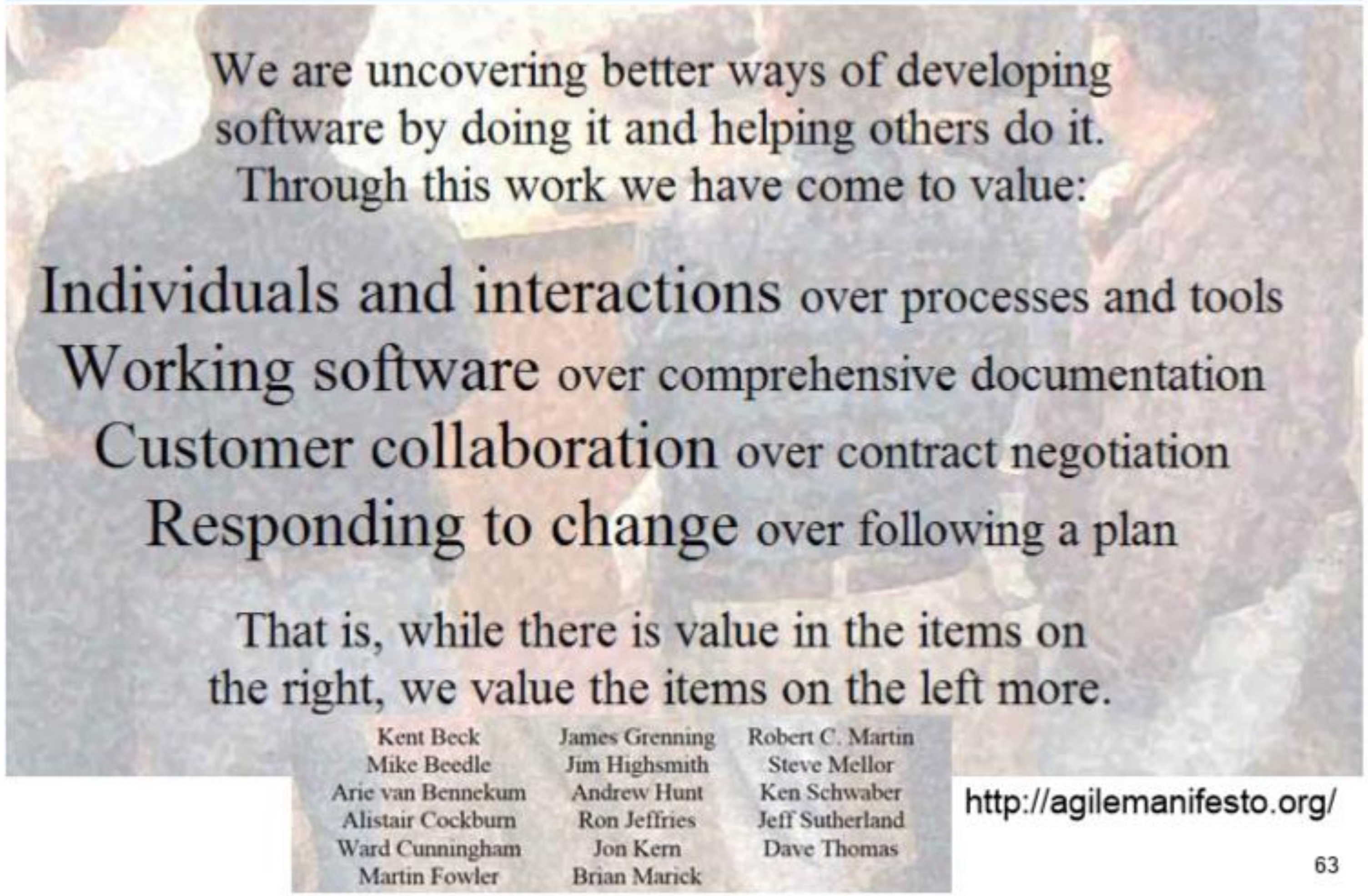
Software Projects (Jones 2004)



Software Projects (Taylor 2000)



Manifesto for Agile Software Development (2001)



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

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

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

Kent Beck	James Grenning	Robert C. Martin
Mike Beedle	Jim Highsmith	Steve Mellor
Arie van Bennekum	Andrew Hunt	Ken Schwaber
Alistair Cockburn	Ron Jeffries	Jeff Sutherland
Ward Cunningham	Jon Kern	Dave Thomas
Martin Fowler	Brian Marick	

http://agilemanifesto.org/

“Extreme Programming” (XP) version of Agile lifecycle

- If short iterations are good, make them as short as possible (weeks vs. years)
- If simplicity is good, always do the simplest thing that could possibly work
- If testing is good, test all the time. Write the test code before you write the code to test.
- If code reviews are good, review code continuously, by programming in pairs, taking turns looking over each other’ s shoulders.

“Extreme Programming” (XP) version of Agile lifecycle

- If short iterations are good, make them as short as possible (weeks vs. years)
- If simplicity is good, always do the simplest thing that could possibly work
- If testing is good, test all the time. Write the test code before you write the code to test.
- If code reviews are good, review code continuously, by programming in pairs, taking turns looking over each other’ s shoulders.

Agile Lifecycle

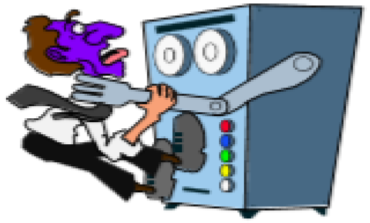
- Embraces change as a fact of life: continuous improvement vs. phases
- Developers continuously refine working but incomplete prototype until customers happy, with customer feedback on each Iteration (every ~1 to 2 weeks)
- Agile emphasizes
  - Test-Driven Development (TDD) to reduce mistakes,
  - written down User Stories to validate customer requirements,
  - Velocity to measure progress

Agile Then and Now

- Controversial in 2001
  - "… yet another attempt to undermine the discipline of software engineering… nothing more than an attempt to legitimize hacker behavior."

    – Steven Ratkin, "Manifesto Elicits Cynicism," *IEEE Computer*, 2001
- Accepted in 2013
  - 2012 study of 66 projects found majority using Agile, even for distributed teams

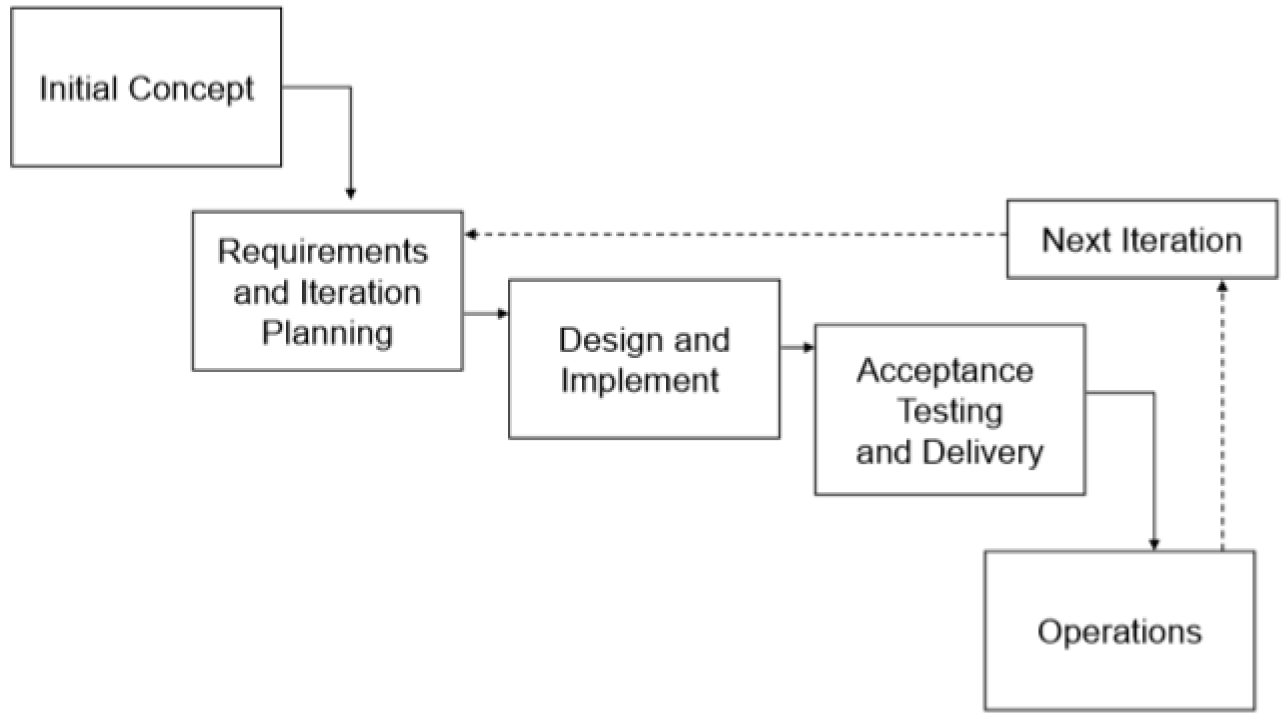
Yes: Plan-and-Document
No: Agile (Sommerville, 2010)



1. Is specification required?
2. Are customers unavailable?
3. Is the system to be built large?
4. Is the system to be built complex (e.g., real time)?
5. Will it have a long product lifetime?
6. Are you using poor software tools?
7. Is the project team geographically distributed?
8. Is team part of a documentation-oriented culture?
9. Does the team have poor programming skills?
10. Is the system to be built subject to regulation?

When asking these questions for projects done by small / student teams in a class, virtually / generally all answers point to Agile.

Agile Development


Initial Concept
Requirements and Iteration Planning
Design and Implement
Acceptance Testing and Delivery
Operations
Next Iteration

Discussion of Agile

- Each iteration a mini-project
  - Each iteration's deliverable is not a prototype, but an operational system
  - Understand risk vs. business value in planning iterations
  - Put some working functionality into user's hands as early as possible
- Timeboxing
  - Set the date for delivering an iteration
  - Date cannot change
  - Only functionality (scope) can change
  - Short duration iterations (weeks, not months)

Some Big Companies using Agile Lifecyle

- Frequent upgrades of SaaS—due to only having a single copy of the software—perfectly align with the Agile software lifecycle.
- Hence, Amazon, eBay, Facebook, Google, and other SaaS providers all rely on the Agile lifecycle, and traditional software companies like Microsoft are increasingly using Agile in their product development.