Group Project
Session 1

Dr Samia Oussena

Overview

- Aims / Objectives
  - Introduce to Agile development and their importance in the implementation of information systems
  - Consider the issues raised when following iterative and incremental development approaches.
  - Provide the ability to select an appropriate methodology for the development task at hand.

Resources

- Material is available week-by-week on Blackboard.
- Material will be introduced in the lecture/tutorial.
- Detailed information about the module is found in the MSG on BB
- Other material is available within the MSG and via referenced textbooks

Module Assessment

- In module assignment
  - Groups of developers working to develop a working system

Course Work Assignment

- 2 elements:
  - Element 1 Group work
    - 30%
  - Element 2 individual
    - Individual report
      - Submitted through Turnitin

Course Work Assignment

- Broad scenario presented in the module assignment document.
    - Team based development
    - Customer interaction
    - Individual report of development.
- Group work essential.

Inherent Problems with Software Development

Requirements are complex

– The client usually does not know all the functional requirements in advance

Requirements may be changing

– Technology enablers introduce new possibilities to deal with nonfunctional requirements

Frequent changes are difficult to manage

– Identifying milestones and cost estimation is difficult

Inherent Problems with Software Development

There is more than one software system

- New system must often be backward compatible with existing system ("legacy system")
- Phased development: Need to distinguish between the system under development and already released systems

Software Process

- A software process is more detailed than a life cycle model.
- A Software process says who does what when.
- A software process includes:
  - Roles
  - Workflows
  - Procedures
  - Standards
  - Templates

Characteristics of a Good Process

- A good software process is:
  - **Repeatable** – a process is repeatable if it can be performed again on a new project? If the process isn’t documented, it probably isn’t repeatable.
  - **Predictable** – Organizations like predictability. It means they know what to expect when repeating the process.
  - **Adaptable** - the process can be tailored for the unique aspects of a project. In general, the more adapted a process is, the less predictable it will be.
  - **Learnable** –
  - **Measurable** – Measurability is important for tracking, control, visibility and improvability.
  - **Improvable**

Terminology

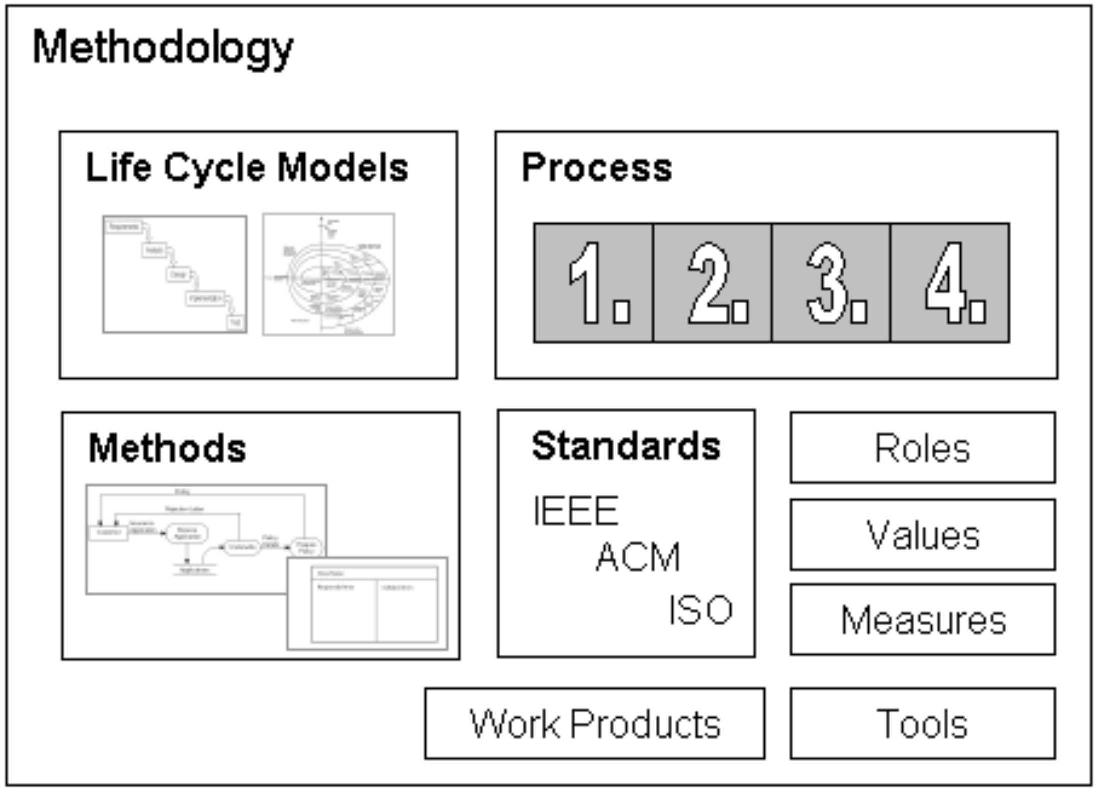
- **Software Life Cycle** – The high-level phases or stages that software goes through from the moment it is conceptualized until the last version is removed from the last machine. The software life cycle consists of new development followed by a number of maintenance updates and finally retirement. [IEEE 610]
- **Software Development Life Cycle (SDLC)** – The sequence of phases a project goes through from start to finish. The standard phases of the SDLC are: requirements specification, analysis, design, implementation and test. Inception and delivery may also be included. Note, these phases are not necessarily sequential. They may overlap or be performed iteratively. The SDLC is applied to new development and maintenance updates. [IEEE 610]

Terminology [cont]

- Software Life Cycle Models (aka Software Process Models) – abstract models that describe a class of development approaches with similar characteristics. Some of the criteria used to distinguish software life cycle models are: <u>timing between phases, entry and exit criteria between phases and the artifacts created during each phase</u>. Examples include: Waterfall, Spiral, Rapid Prototyping, Incremental Development, etc.

Terminology [cont]

- Software Process, Methods, and Methodologies

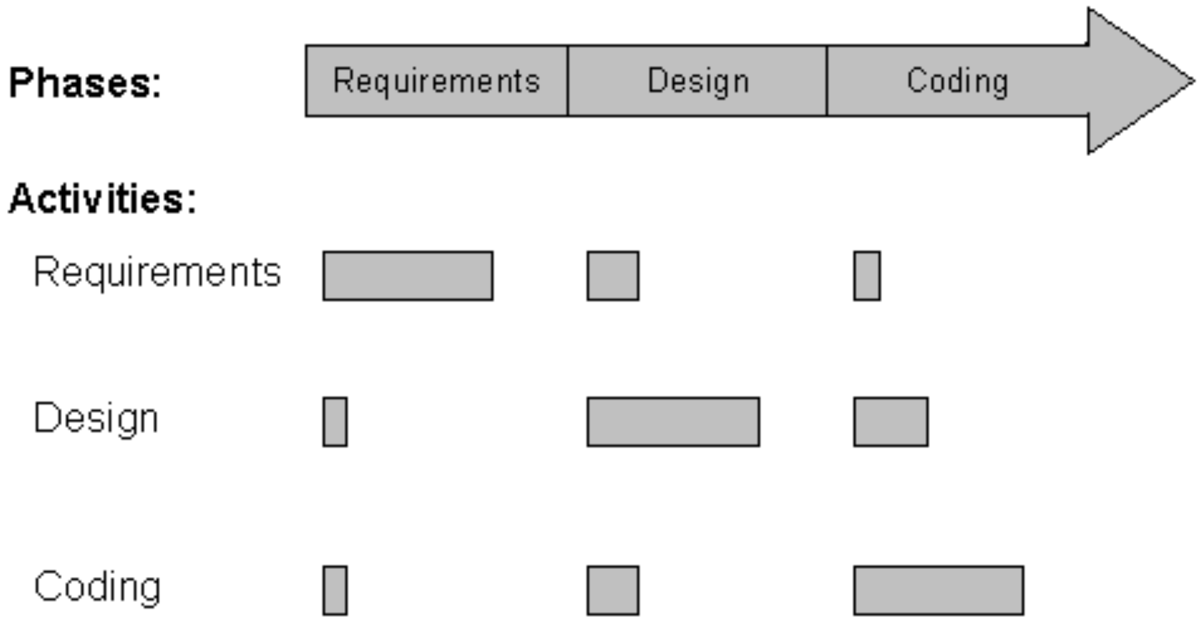


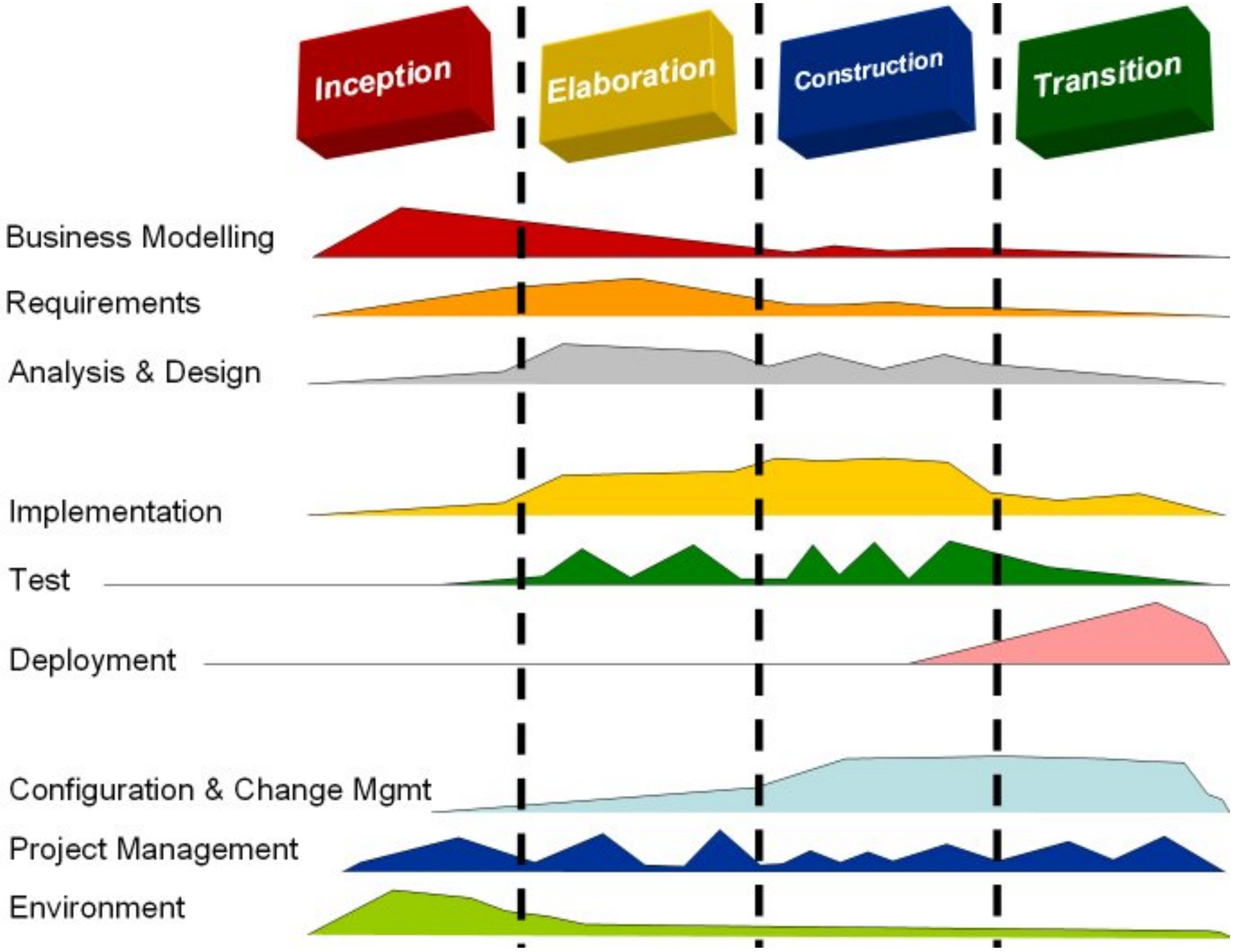
Phase Names vs. Activity Names

- Is requirements (or design, or coding, or etc.) a phase or an activity?
- Terms such as requirements, design, coding, etc. are used to characterize a phase (adjective) and also an activity (verb/adverb) that might occur during one or more phases.
- It's important to distinguish between a label given to a phase and the activity that occurs during that phase.

Phase Names vs. Activity Names [cont]

- The label on a phase is descriptive of the dominate activity performed during that phase. It becomes confusing when there isn't a one-to-one correspondence between phase name and activity within that phase.



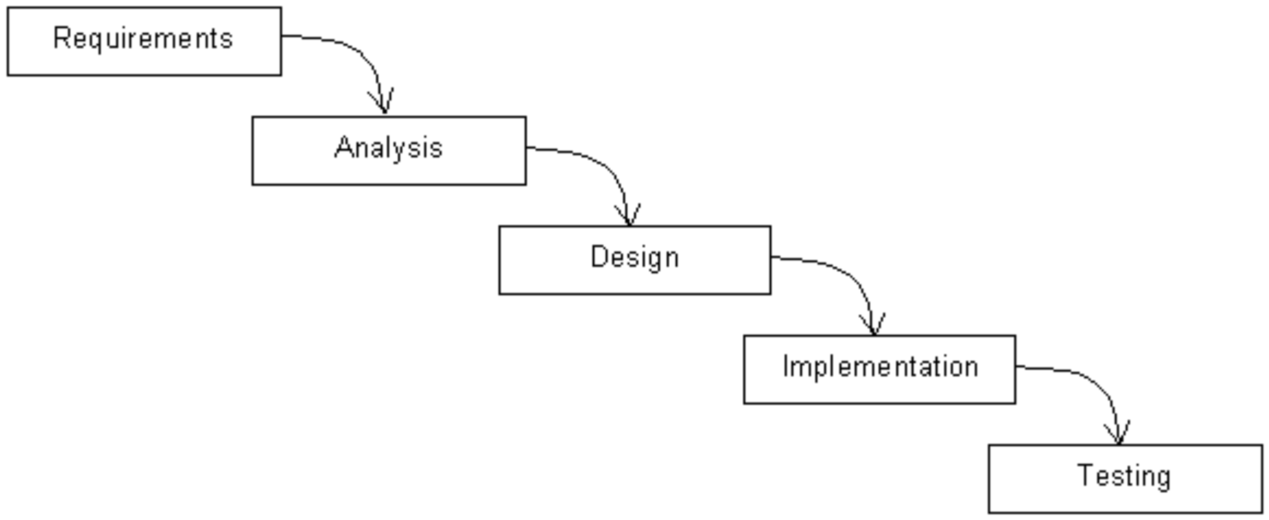

Inception
Elaboration
Construction
Transition
Business Modelling
Requirements
Analysis & Design
Implementation
Test
Deployment
Configuration & Change Mgmt
Project Management
Environment

Software Life Cycle Models

- Abstract models that define a few broad types of software development processes
- Life cycle models specify:
  - Major phases including milestones and deliverables
  - Timing between each phase
  - Entry and exit criteria for each phase
- Life cycle models are a starting point for project management

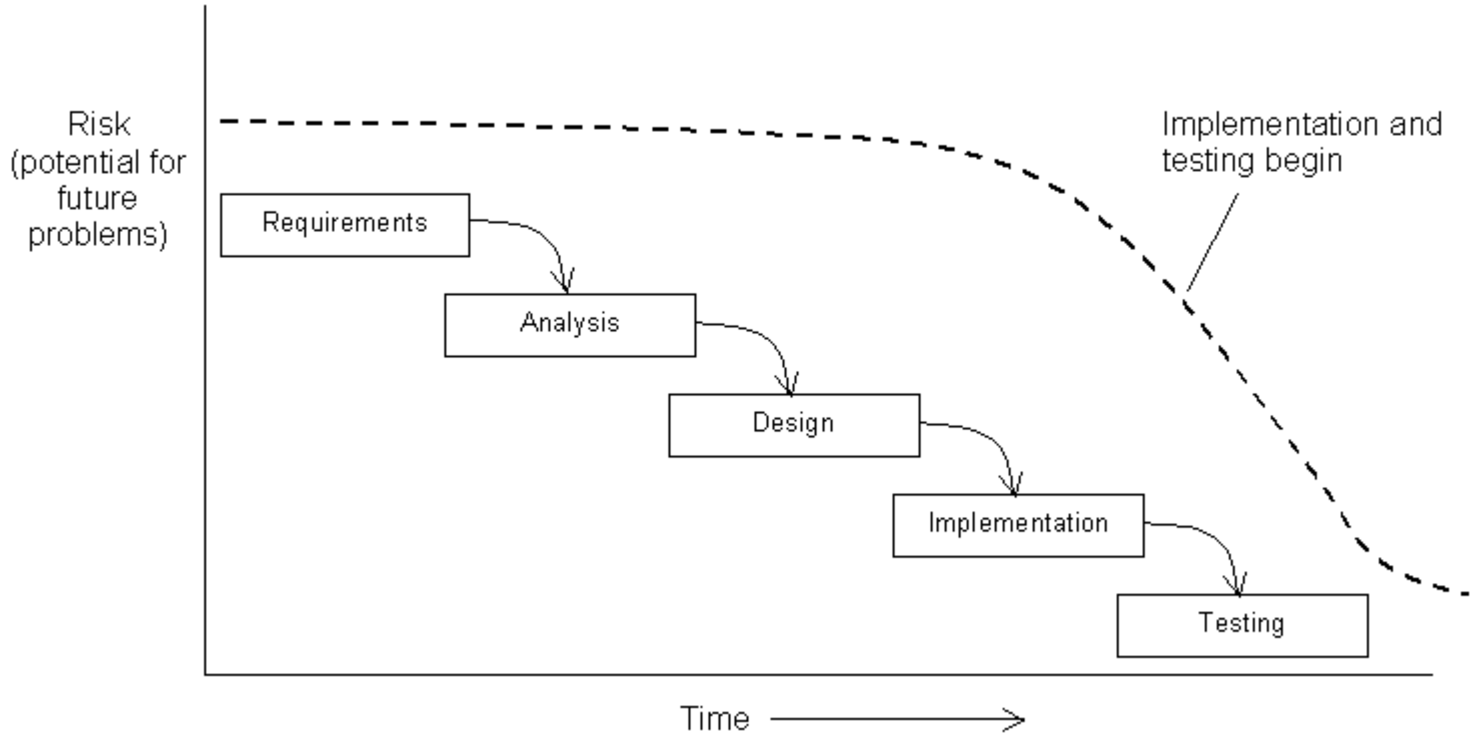
Waterfall

- Sequential stages with little or no backtracking



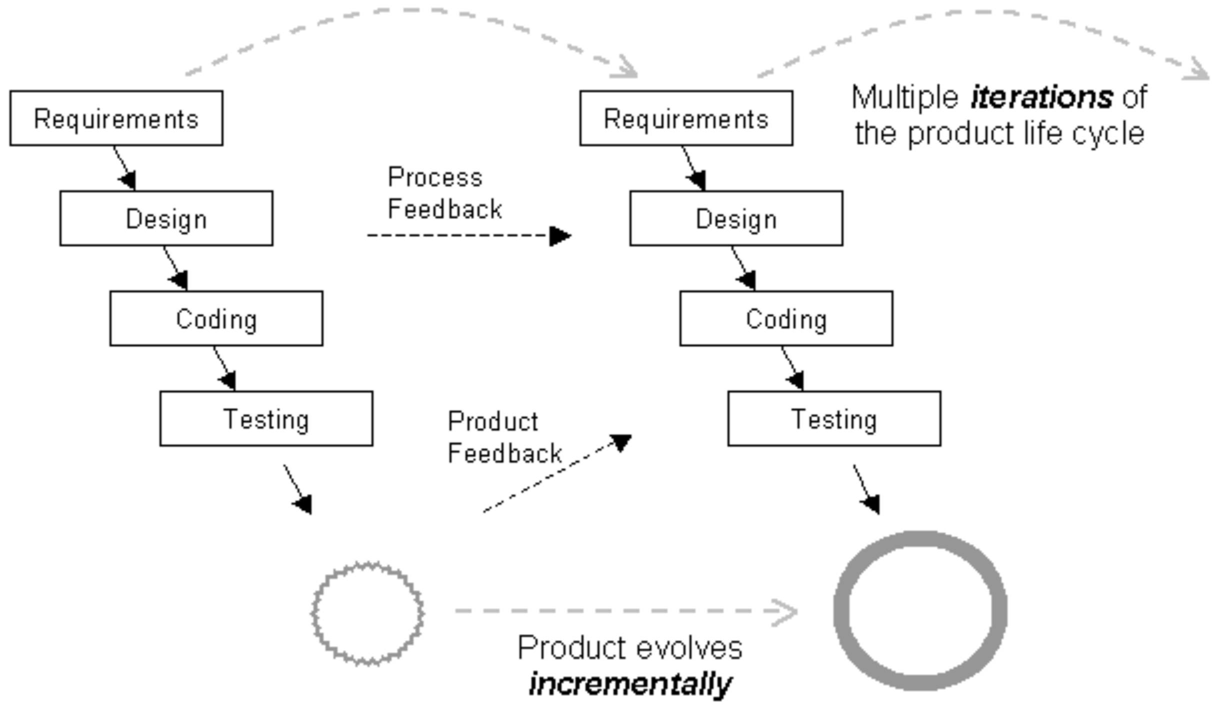
Risk Profile of Waterfall Project

- When following the waterfall process model risks aren’t resolved until late in the development cycle.



Iterative and Incremental

- A characteristic of modern life cycle models. The product evolves incrementally over a series of iterations.



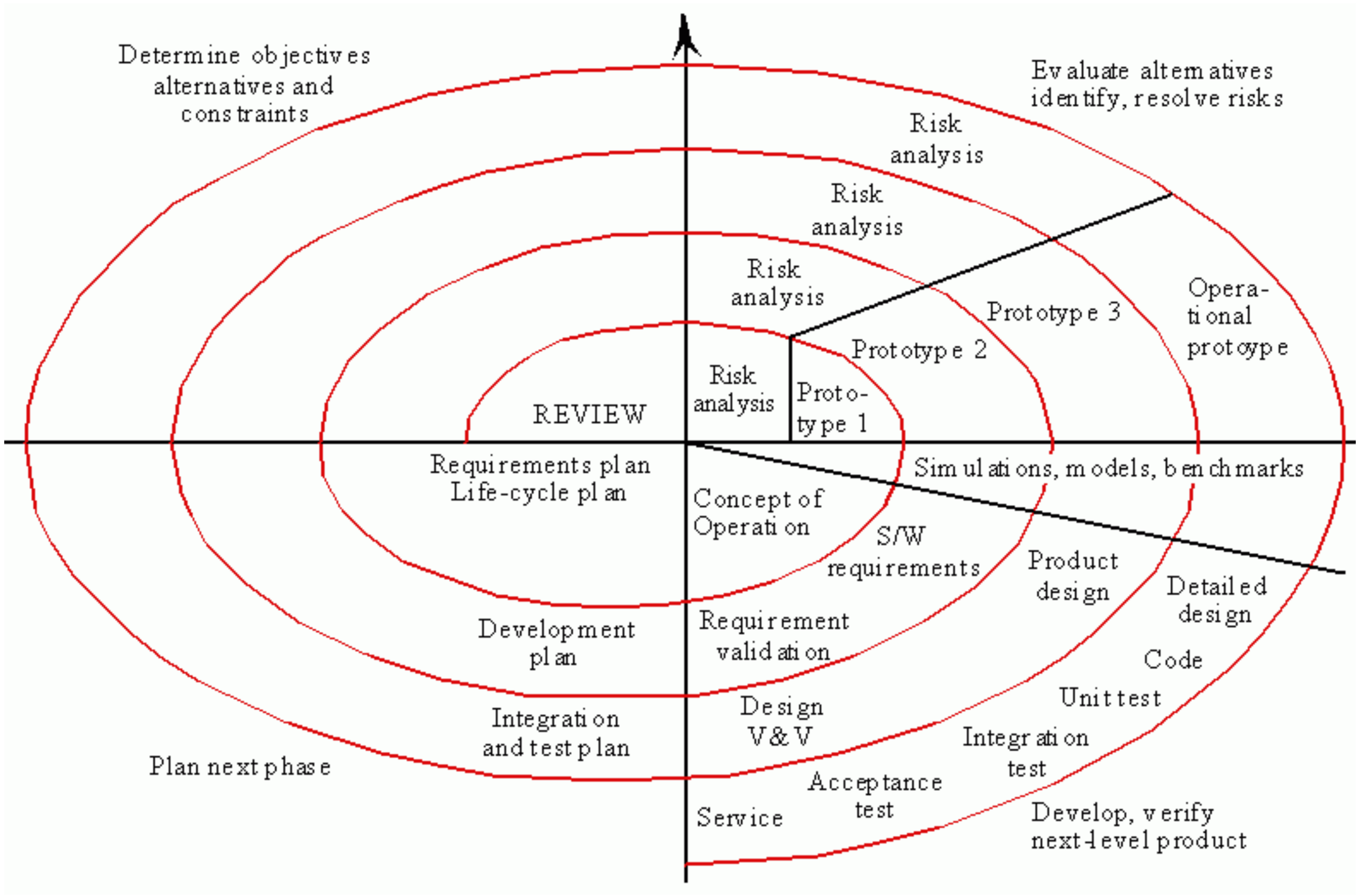
Benefits of an Iterative and Incremental Approach

- More appropriate for applications with <u>emergent requirements</u>. For many UI applications, requirements aren't well-known form the start but rather emerge over time based on experience with early increments and prototypes.
- Incremental functionality is a <u>more accurate measure of progress</u> than paper deliverables. This provides better visibility into project status.
- <u>Manages risks better </u>(technical risks with architecture and programming and project risks with meeting schedule and budget targets)

Modern Software Life Cycle Models

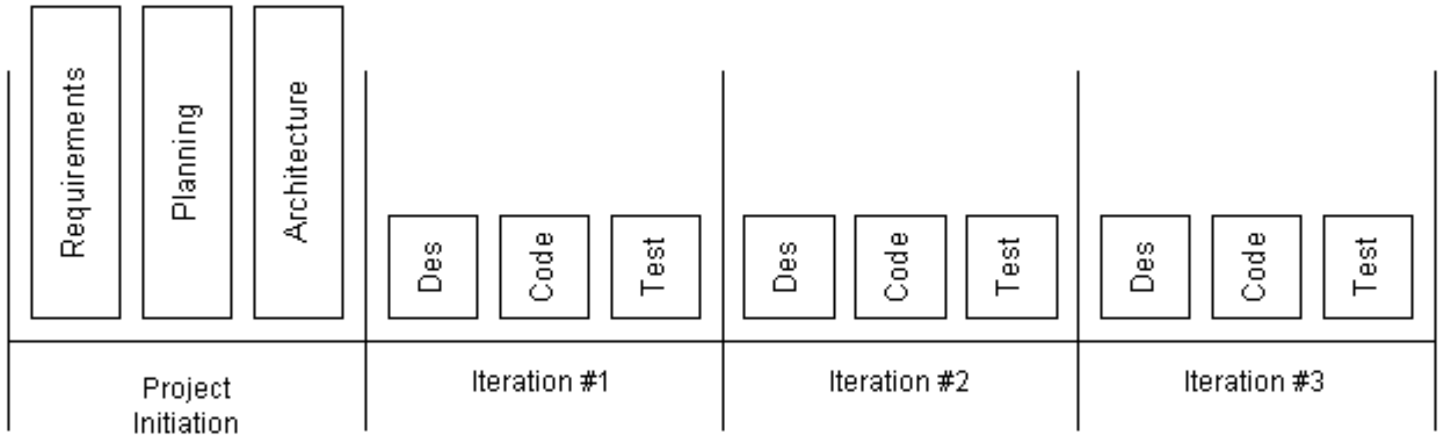
- Spiral
- Staged
- Evolutionary Prototyping
- Evolutionary Delivery

Spiral


Determine objectives alternatives and constraints
Evaluate alternatives identify, resolve risks
Risk analysis
Risk analysis
Risk analysis
Risk analysis
REVIEW
Proto-type 1
Prototype 2
Prototype 3
Opera-tional prototype
Requirements plan Life-cycle plan
Concept of Operation
Simulations, models, benchmarks
S/W requirements
Product design
Detailed design
Development plan
Requirement validation
Code
Unit test
Integration and test plan
Design V&V
Integration test
Plan next phase
Acceptance test
Service
Develop, verify next-level product

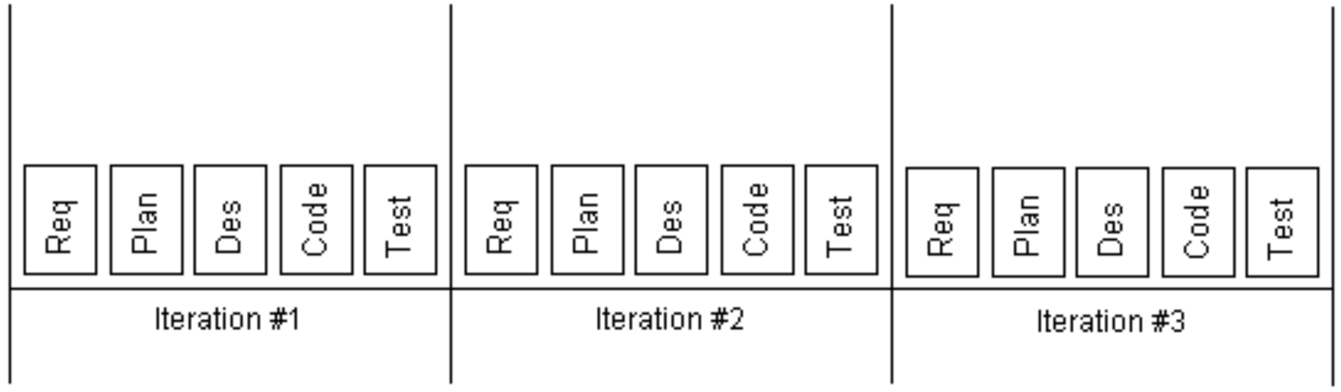
Staged

- Requirements and planning occur up-front but product is developed and delivered over a series of iterations.



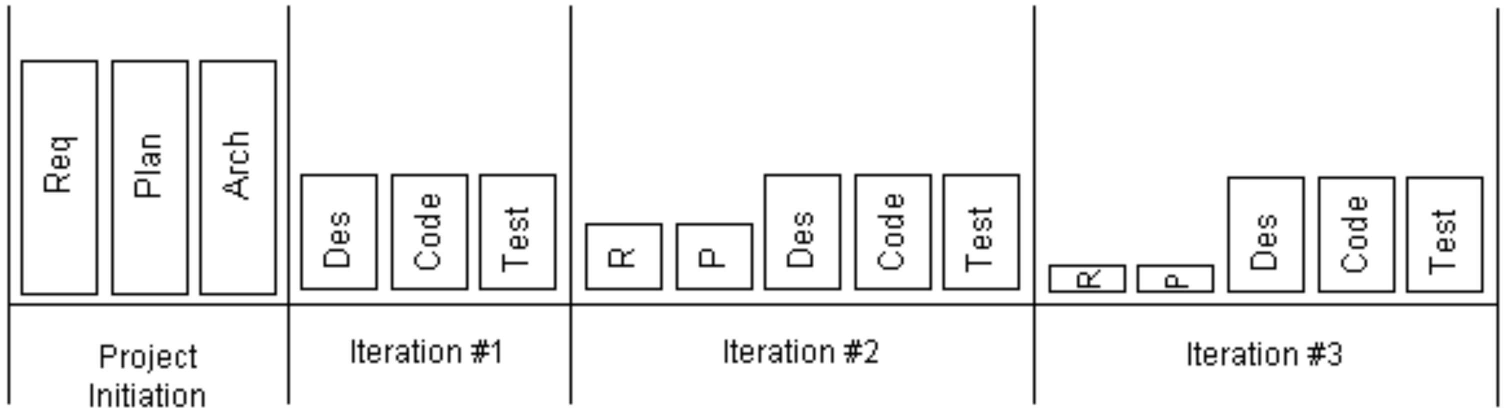
Evolutionary Prototyping

- The contents of each iteration is driven by feedback from previous iterations.



Evolutionary Delivery

- The middle ground. Considerable planning and product definition occur up-front, but the content of each iteration is also influenced by feedback from previous iterations.



Plan driven Methods

- Software development practices
  - Heavyweight
  - Heavily regulated, regimented i.e. constraining
- Problems

- Testing ignores method and budget
- Bureaucratic sign off
- No prioritizing and no flexibility
- Narrow skills
- Handover is disjointed
- Limited user involvement
- Change control madness

What is Agile Software Development?

- Collection of methods that share certain characteristics:
  - Iterative and incremental lifecycle model. Common length of an iteration is 1-4 weeks.
  - No major up-front effort to capture comprehensive detailed requirements. And consequently, no long-range detailed plans.
  - Flexibility is given priority over predictability (of cost, schedule and features). Adaptable. Able to respond to changes in requirements, environment, etc.
  - More code-focused than document-focused.
  - Human-friendly
  - Modern practices that work. In the words of Alistair Cockburn, “we cherry-picked the best practices at the time”.

Agile Methods

- Incremental approaches emerge as a response
  - Mid 1990's e.g. RAD, XP, etc
  - Increased flexibility
- Feb 2001
  - Agile Manifesto
  - Agile Alliance

The Agile Manifesto

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

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

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

Soure: The Agile Alliance

Beck, Kent; et al. (2001). "Manifesto for Agile Software Development". Agile Alliance. http://agilemanifesto.org/.

Principles Behind Agile Manifesto

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time-scale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Motivations

- The agile movement was in part motivated by a dissatisfaction with existing heavyweight processes.
- It was near the height of the .com era when time-to-market often took precedence over other desirable characteristics such as reliability, dependability, etc.
- Greater emphasis/recognition of the people factor. Software development isn’t manufacturing.