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 with SEGA 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

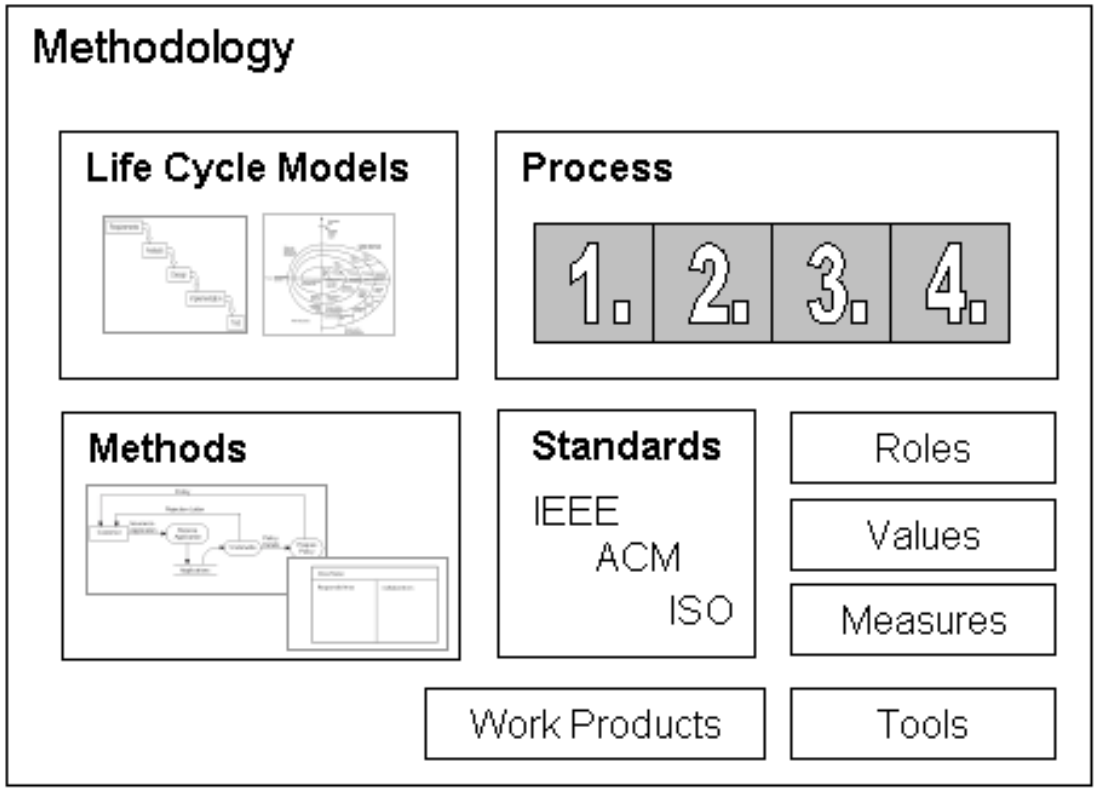
- **Software Life Cycle** – The high-level phases or stages that software goes through from the moment it is conceptualized until the last version is removed from the last machine. The software life cycle consists of new development followed by a number of maintenance updates and finally retirement. [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: timing between phases, entry and exit criteria between phases and the artifacts created during each phase. Examples include: Waterfall, Spiral, Rapid Prototyping, Incremental Development, etc.

Terminology [cont]

- Software Process, Methods, and Methodologies

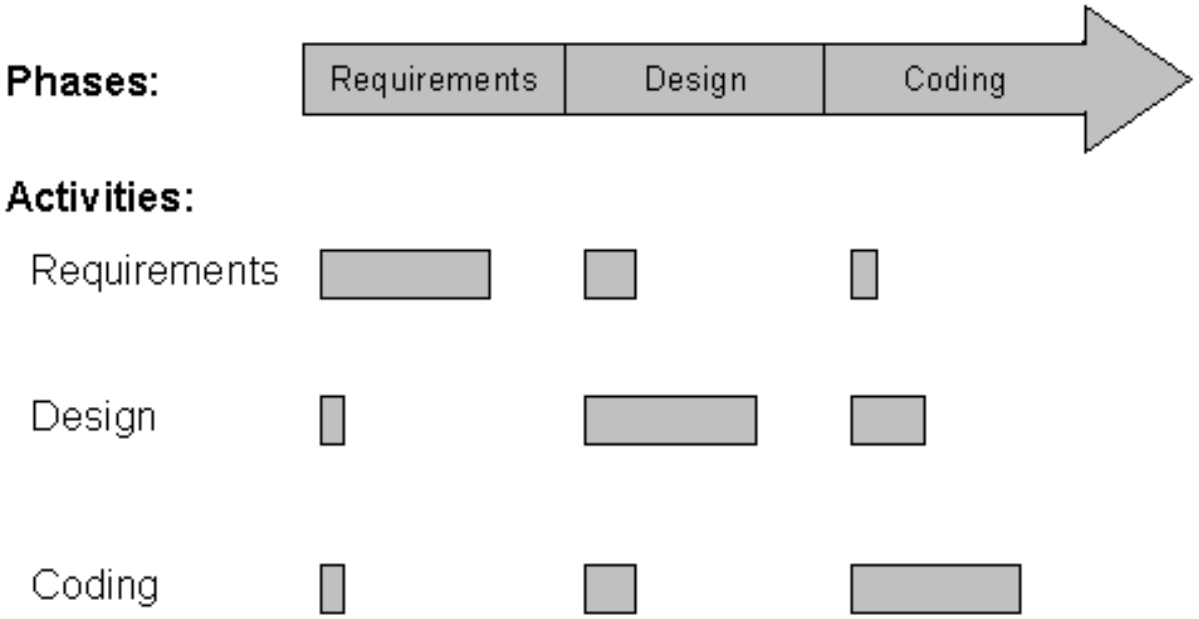


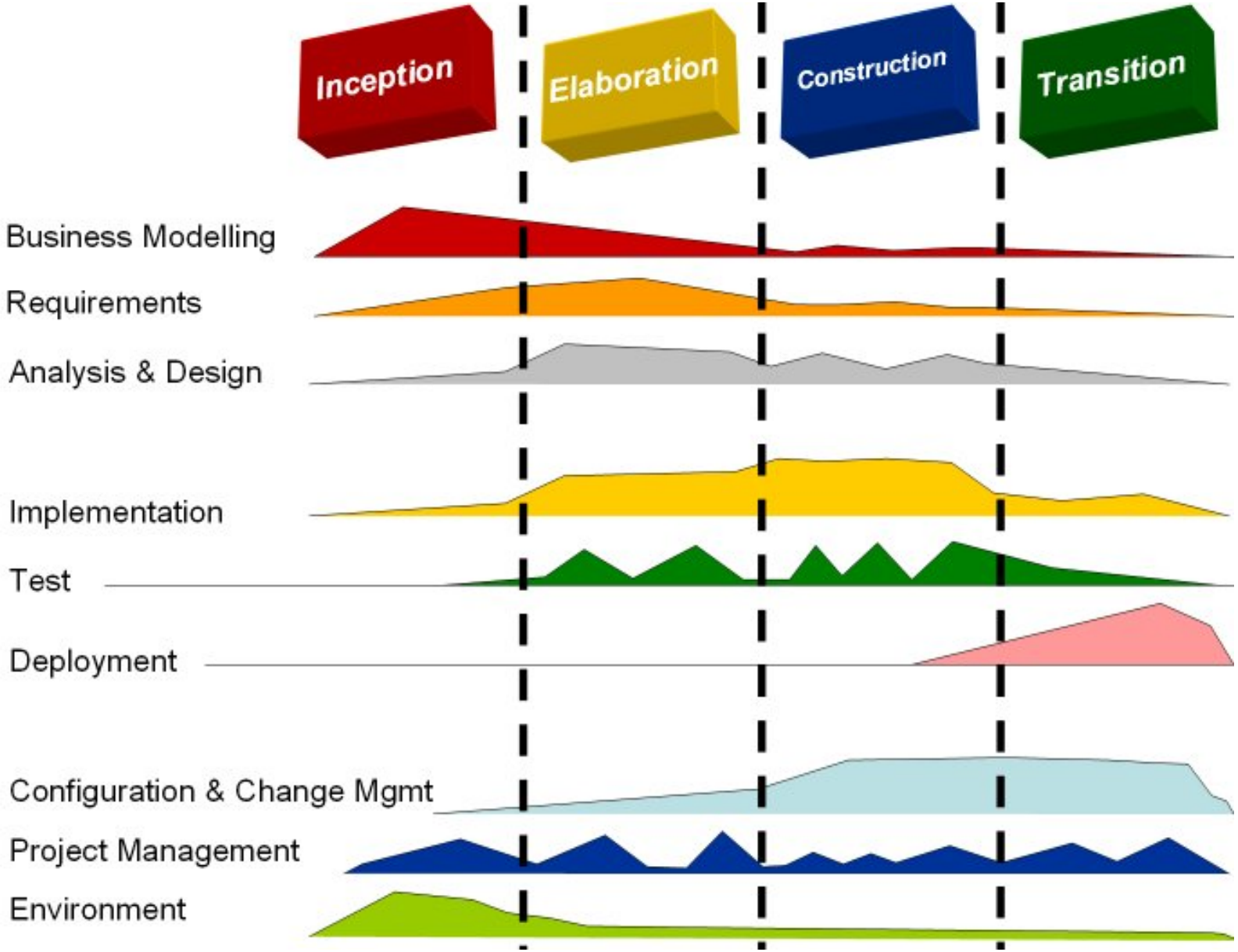
Phase Names vs. Activity Names

- 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.



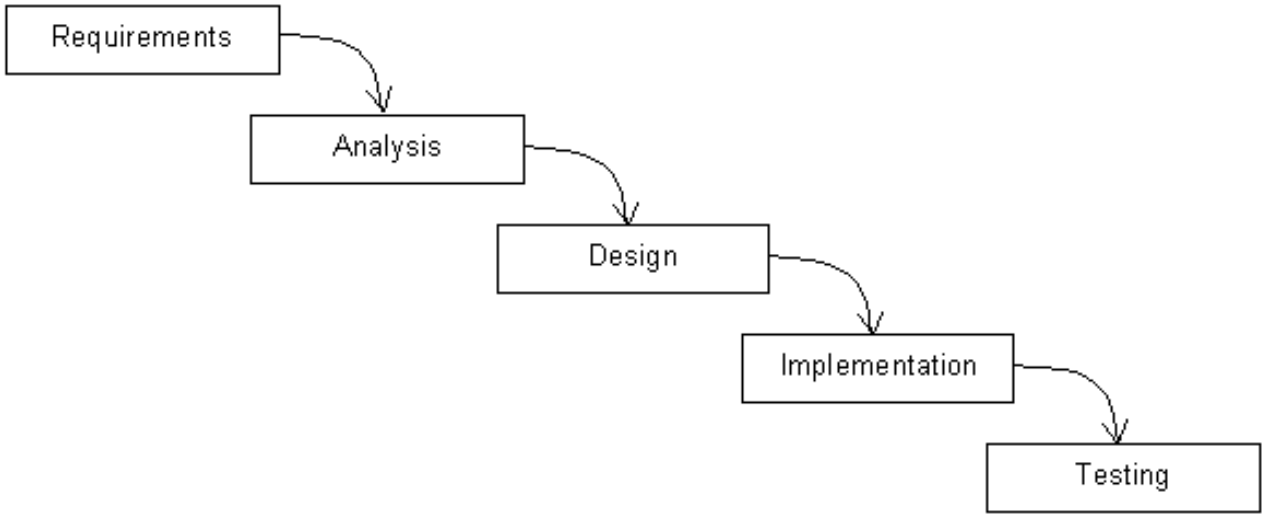

Inception
Elaboration
Construction
Transition
Business Modelling
Requirements
Analysis & Design
Implementation
Test
Deployment
Configuration & Change Mgmt
Project Management
Environment

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

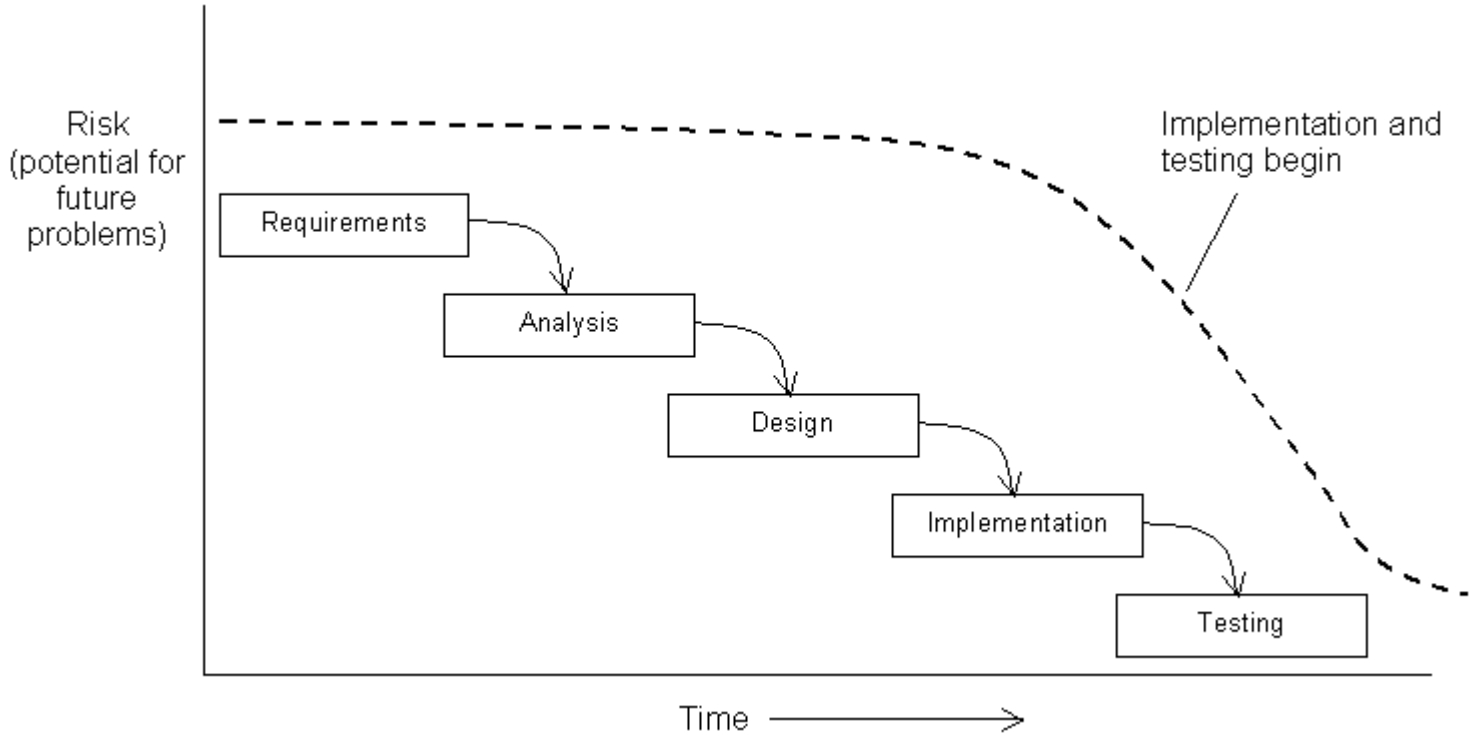
Waterfall

- Sequential stages with little or no backtracking



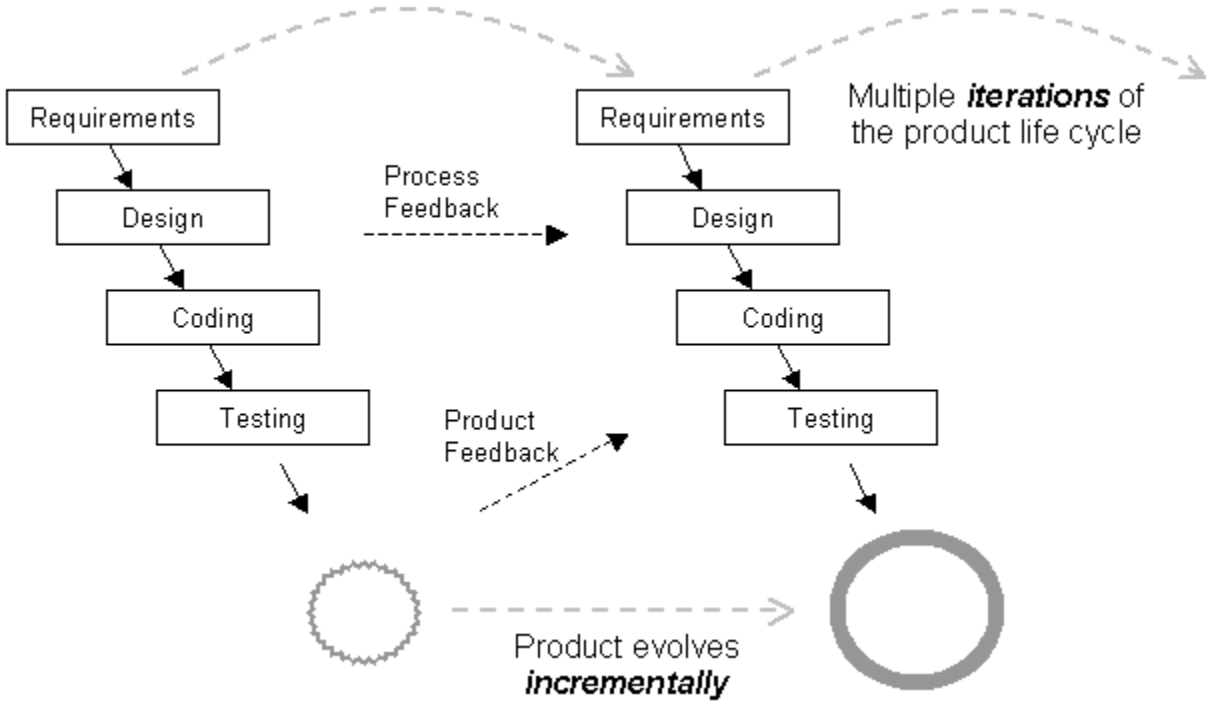
Risk Profile of Waterfall Project

- 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.



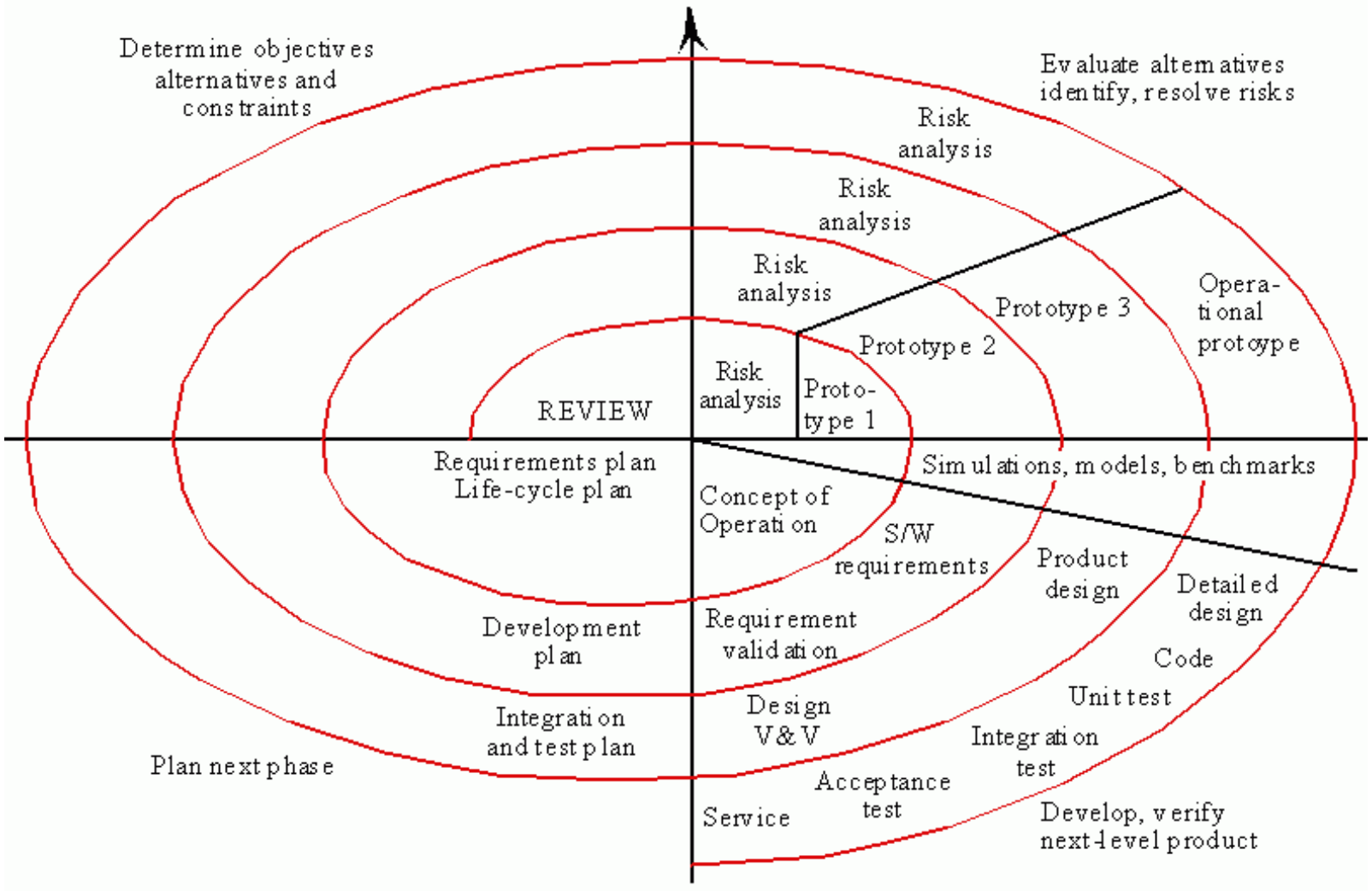
Benefits of an Iterative and Incremental Approach

- More appropriate for applications with emergent requirements. 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 more accurate measure of progress than paper deliverables. This provides better visibility into project status.
- Manages risks better (technical risks with architecture and programming and project risks with meeting schedule and budget targets)

Modern Software Life Cycle Models

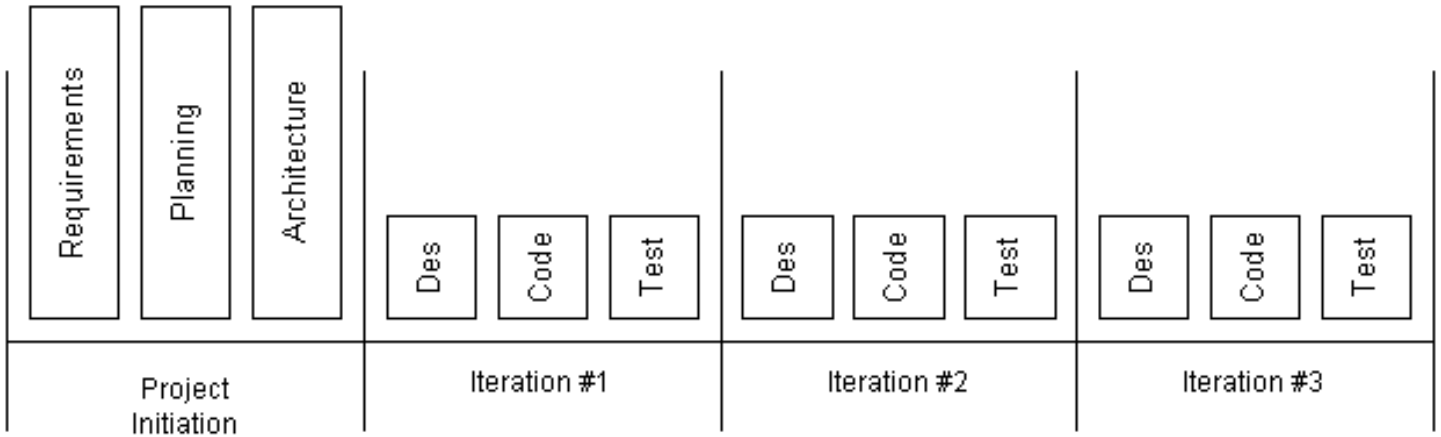
- Spiral
- Staged
- Evolutionary Prototyping
- Evolutionary Delivery

Spiral


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

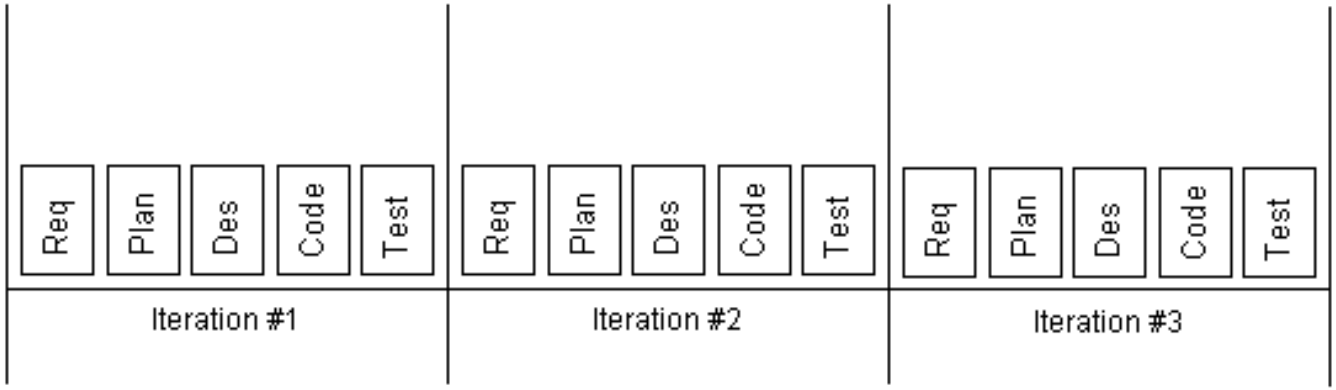
Staged

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



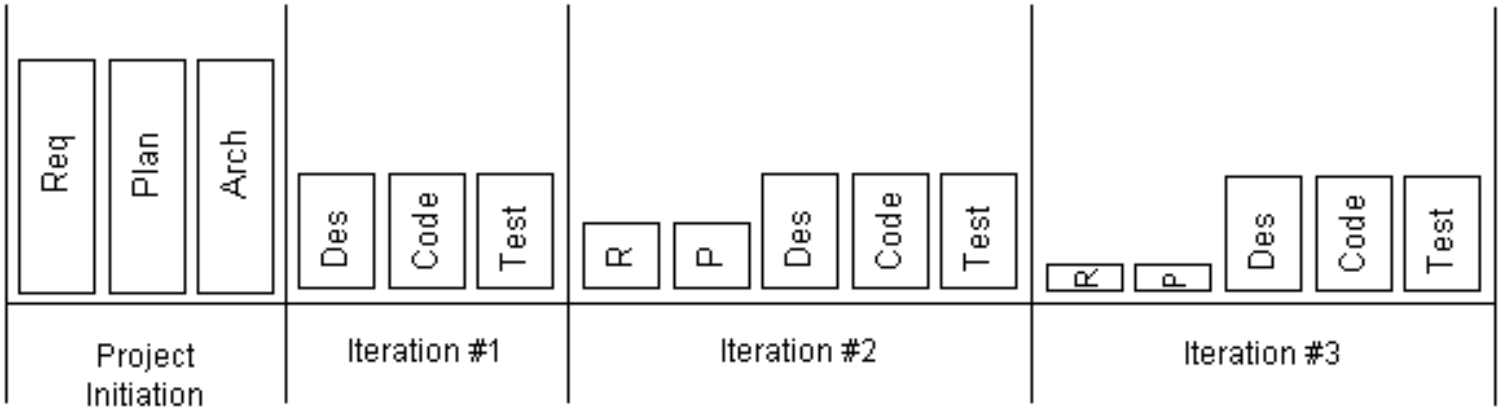
Evolutionary Prototyping

- 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:

- *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."

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.

Example Agile Methodologies

- Extreme Programming (Beck)
- SCRUM (Schwaber and Beedle)
- Lean Software Development
- Dynamic Systems Development Methodology
- Crystal Methodologies (Cockburn)
- Adaptive Software Development (Highsmith)
- Feature-Driven Development (Palmer and Felsing)

How Agile Differed

- Collaboration:
  - Less paperwork and more conversation
  - Stakeholders actively involved
- Focus on working software:
  - Greater feedback makes agile projects easier to manage
  - Less documentation is required
- Agilists are generalizing specialists:
  - Less hand offs between people
  - Less people required
- Agile is based on practice, not theory

Agile Software Development

- An iterative and incremental (evolutionary) approach performed in a highly collaborative manner with **just the right amount of ceremony** to produce high quality software in a cost effective and timely manner which meets the changing needs of its stakeholders.
- Core principles
  - "Fits just right" process
  - Continuous testing and validation
  - Consistent team collaboration
  - Rapid response to change
  - Ongoing customer involvement
  - Frequent delivery of working software

DSDM

- Originally based upon the RAD methodology
  - Developed here in the UK by the DSDM Consortium
- DSDM is an iterative and incremental approach
  - Emphasises continuous user involvement
- **Goal**
  - Deliver software systems on time and on budget while adjusting for changing requirements along the development process.

DSDM

- DSDM can be considered a framework of controls for rapid development
- DSDM Principles
  - Active user involvement is imperative
  - DSDM teams must be empowered
  - The focus is on frequent delivery of products
  - Fitness for business purpose
  - Iterative and incremental development
  - All changes during development are reversible
  - Requirements are baselined at a high level
  - Testing is integrated throughout the life-cycle
  - A collaborative and cooperative approach between all stakeholders is essential.

DSDM Project Lifecycle

- Three general phases
  - Each phase consists of a number of sub phases and steps

- Phase 1 – Pre Project
- Phase 2 – Project Life Cycle
- Phase 3 – Post Project

Key Techniques

- Timeboxing
- MoSCoW
  - Must, Should, Could, Would
- Prototyping
- Testing
- Modeling
- Configuration Management

DSDM Project Lifecycle

- Three general phases
  - Each phase consists of a number of sub phases and steps

- Phase 1 – Pre Project
- Phase 2 – Project Life Cycle
- Phase 3 – Post Project

Key Techniques

- Timeboxing
- MoSCoW
  - Must, Should, Could, Would
- Prototyping
- Testing
- Modeling
- Configuration Management

Extreme Programming

- Iterative methodology
  - Kent Beck, C3 project
- Premise:
  - Taking the beneficial elements of traditional software engineering practices to "extreme" levels will produce improvement
  - i.e. If some is good, more is better!

XP Project Lifecycle

- No one process fits every project
  - Practices should be tailored to the needs of individual projects.
- Multiple short development cycles
  - Exploration Phase
  - Planning Phase
  - Iterations to Release Phase
  - Productionizing Phase
  - Maintenance Phase

XP Activities & Values

- Values
  - Communication
  - Simplicity
  - Feedback
  - Courage
  - Respect
- Activities
  - Coding
    - The only truly important product of the system development process is code!!!
  - Testing
  - Listening
  - Designing

XP Principles

- Humanity
- Economics – (Can't accuse agile proponents of not taking an engineering perspective.)
- Mutual Benefit
- Self-Similarity – reusing solutions in new contexts.
- Improvement
- Diversity
- Reflection
- Flow – deliver a steady flow of valuable software
- Opportunity
- Redundancy
- Failure – taking risks is good. If you aren't crashing once in awhile, you aren't driving on the edge (as fast as you can).
- Quality – "quality is not a control variable"
- Baby Steps
- Accepted Responsibility

XP Practices

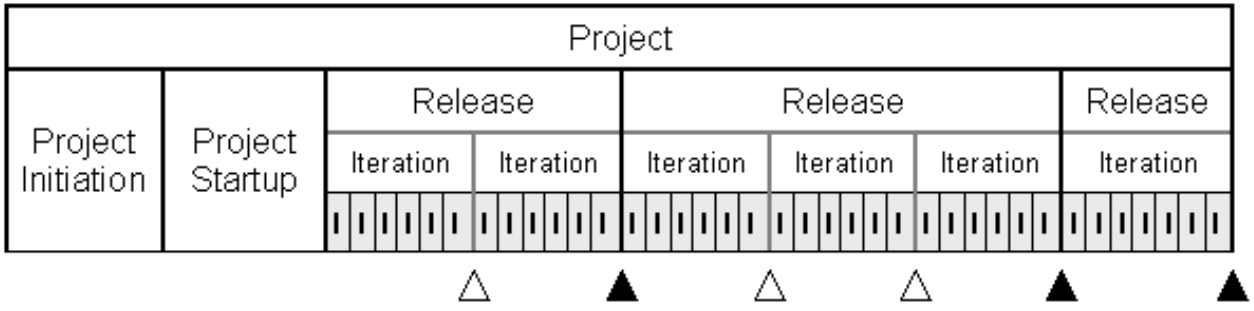
- Sit Together – More opportunities for communication
- Whole Team – all the skills you need are on one team (inc customer).
- Informative Workspace – Project info & status. Increased visibility.
- Energized Work – Working beyond your limits creates negative value.
- Pair Programming – write code in pairs. Strategic and tactical.
- Stories – Customer-visible units of work used for planning.
- Weekly Cycle – plan, pick stories, divide into tasks, working s/w.
- Quarterly Cycle – themes, reflection, big picture thinking/planning.
- Slack – expendable tasks; a form of management reserve.
- Ten-Minute Build – automatic builds.
- Continuous Integration – divide conquer and integration.
- Test-First Programming – tests substitute for requirements
- Incremental Design – ongoing and emergent design. Refactoring

Planning in XP

Work is planned in units called Stories. A story starts out as a brief description of some feature or functionality that is important to the user.
There is high-level planning each quarter and more detailed planning for each iteration (1-3 weeks).
Stores are estimated by developers and prioritized by the customer. Each iteration the customer picks another set of stories for implementation. Experience on earlier iterations is used as a guide regarding how many stories can be estimated in future iterations. Estimates on existing stories might be updated based on new information gained from doing past iterations. Stories are divided into tasks at the start of an iteration. Developers commit to completing certain tasks.

Internal and External Releases

- There are three important opportunities for feedback and control: during frequent integration, at the end of an iteration with an internal release and at the end of an iteration with an external release.



Agile Method Weaknesses

- There are some/many projects for which agile practices are infeasible (daily customer contact, whole team, etc.)
- Doesn't scale up to large teams. (Solution: Divide problem among independent agile projects.)
- Puts a premium on having top-notch talent. This can be a limiting factor (unless you work in Lake Wobegon where all the developers are above average.)
- Not appropriate for safety-critical software development

Table 1. Home ground for agile and plan-driven methods.

Home-ground area	Agile methods	Plan-driven methods
Developers	Agile, knowledgeable, collocated, and collaborative	Plan-oriented; adequate skills; access to external knowledge
Customers	Dedicated, knowledgeable, collocated, collaborative, representative, and empowered	Access to knowledgeable, collaborative, representative, and empowered customers
Requirements	Largely emergent; rapid change	Knowable early; largely stable
Architecture	Designed for current requirements	Designed for current and foreseeable requirements
Refactoring	Inexpensive	Expensive
Size	Smaller teams and products	Larger teams and products
Primary objective	Rapid value	High assurance