Lecture 4

The Software Process

Miguel Goulão
mgoul@fct.unl.pt

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA
Departamento de Informática

The software process

- The set of activities and associated results which produce a software product or System
- The sequence of steps required to develop and maintain software
- Sets out the technical and management framework for applying methods, tools and people to the software task
- The Software Process is a description of the process which guides software engineers as they work by identifying their roles and tasks.

The Software Process – Fundamental activities

Feasibility/marketing study (optional)

- Feasibility report

Software Specification*

- define functionality and constraints

Software Development

- produce software

Software Verification and Validation

- does what the customer wants
- matches the specification

Software Evolution

- to meet customer changing needs

Strong disclaimer!

There is NO UNIVERSAL PROCESS

Software Specification (aka Requirements Engineering)

- Understand and define **what services are required** from the system and **identify constraints** to the system's operation and development
- Leads to a **requirements document** which is a twofold specification of the system for:
  - Customers and end-users get high level statements
  - System developers get the detailed system specification

Software specification main activities

- **Requirements Elicitation** and **Analysis** (may develop System Model and prototypes)
- **Requirements Specification** (leads to User and system requirements document)
- **Requirements Validation** (checks realism, consistency and completeness)

Note: **Agile methods requirements specification** is not a separate activity from system development (specified before each increment and prioritized, clients are part of the team)

Design and implementation

Develop an executable system for delivery to the customer.

- Architectural Design: identify overall structure of the system (components, relations and distribution)
- Interface Design: Interfaces between components
- Component Selection and Design: look for reusable components or design new ones (can leave details to programmer)
- Database Design: Data Structures Design

Verification and Validation (V&V)

Verification:
are we building **the system right**?

- Look at the system's specification

Validation:
are we building **the right system**?

- Meets the expectation of the customer

Verification and Validation (V&V)

Stages:

- Component (or unit) Testing (programmer)
- System Testing (integration)
- Acceptance Testing

- If to be marketed, sometimes Beta testing is done (delivered the system to a controlled group of users – reporting problems to developers)

Software Evolution

(Historically separated from process development)

Software is continuously changed over its lifetime due to changing requirements and customer needs and other context constraints changes

Characteristics of a good process

- Understandability
- Visibility
- Supportability
- Acceptability
- Reliability
- Robustness
- Maintainability
- Rapidity

Steps in a Generic Software Process

- Project Definition
- Requirements Analysis
- Design
- Program Implementation
- Component Testing
- Integration Testing
- System Testing
- System Delivery
- Maintenance

Process activities

System Delivery

- Implementation of the system into the working environment and replacement of the existing system

Maintenance

- Corrective
- Adaptive
- Perfective

Software Process Model

Also known as **Software Development Life Cycle**

A simplified description of a software process that presents one view of that process.

Software development life cycles

Software development life cycles

Prescriptive Models

- Traditional/Waterfall
- Rapid Application Development (RAD)
- Incremental
- V-Model
- Prototyping

Evolutionary

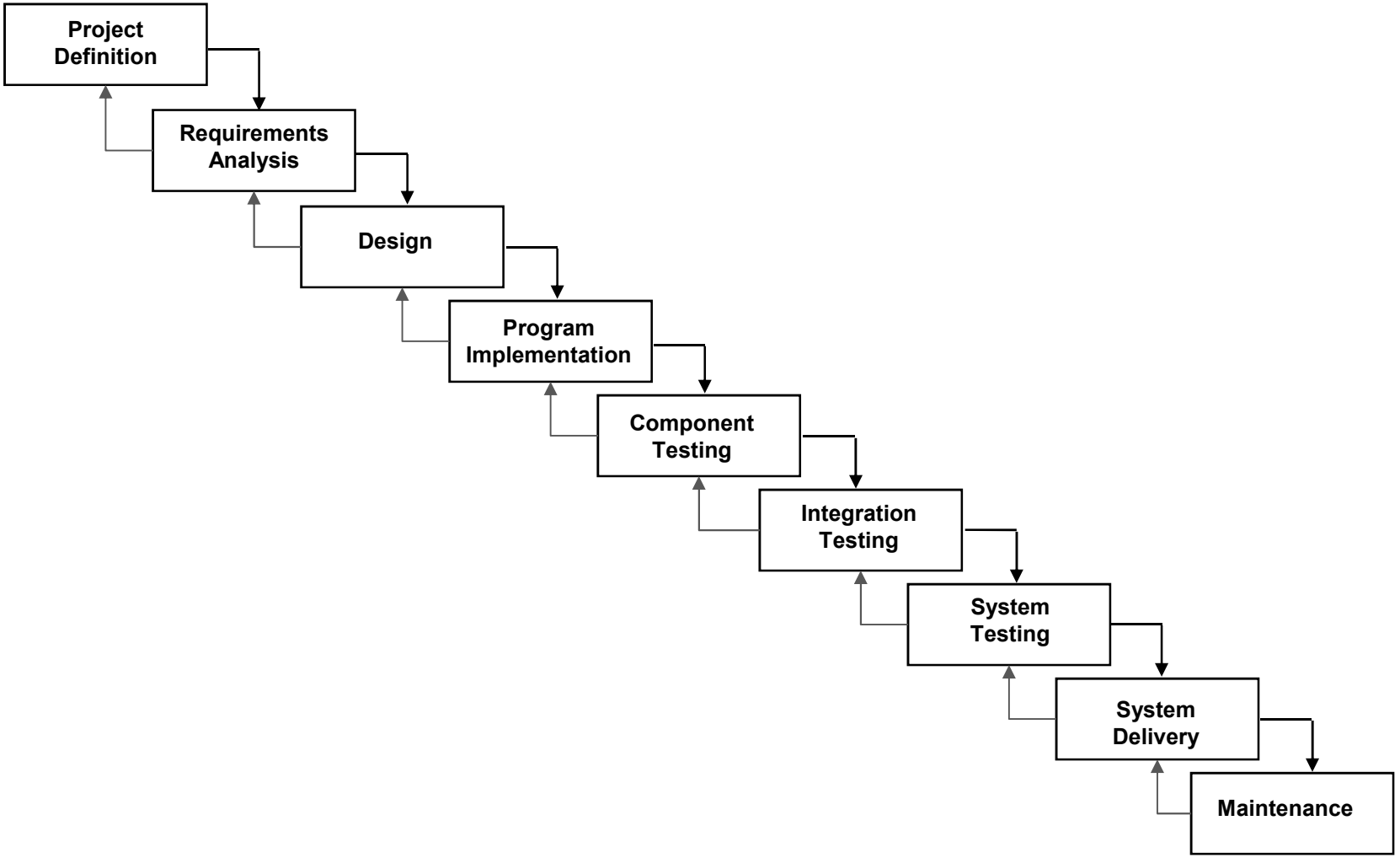
- Spiral
- Component Assembly

Agile & RUP

- Agile Methods (e.g. XP, Scrum)
- RUP

Prescriptive models

The Waterfall model



Waterfall Model - Plan driven

- Most widely used, though no longer state-of-the-art
- Each step results in documentation
- May be suitable for well-understood developments using familiar technology
- Not suited to brand new, and different systems from the common because of specification uncertainty
- Difficulty in accommodating change after the process has started
- Can accommodate iteration but indirectly
- **Working version not available till late in process**
- Often get blocking states

Waterfall Model

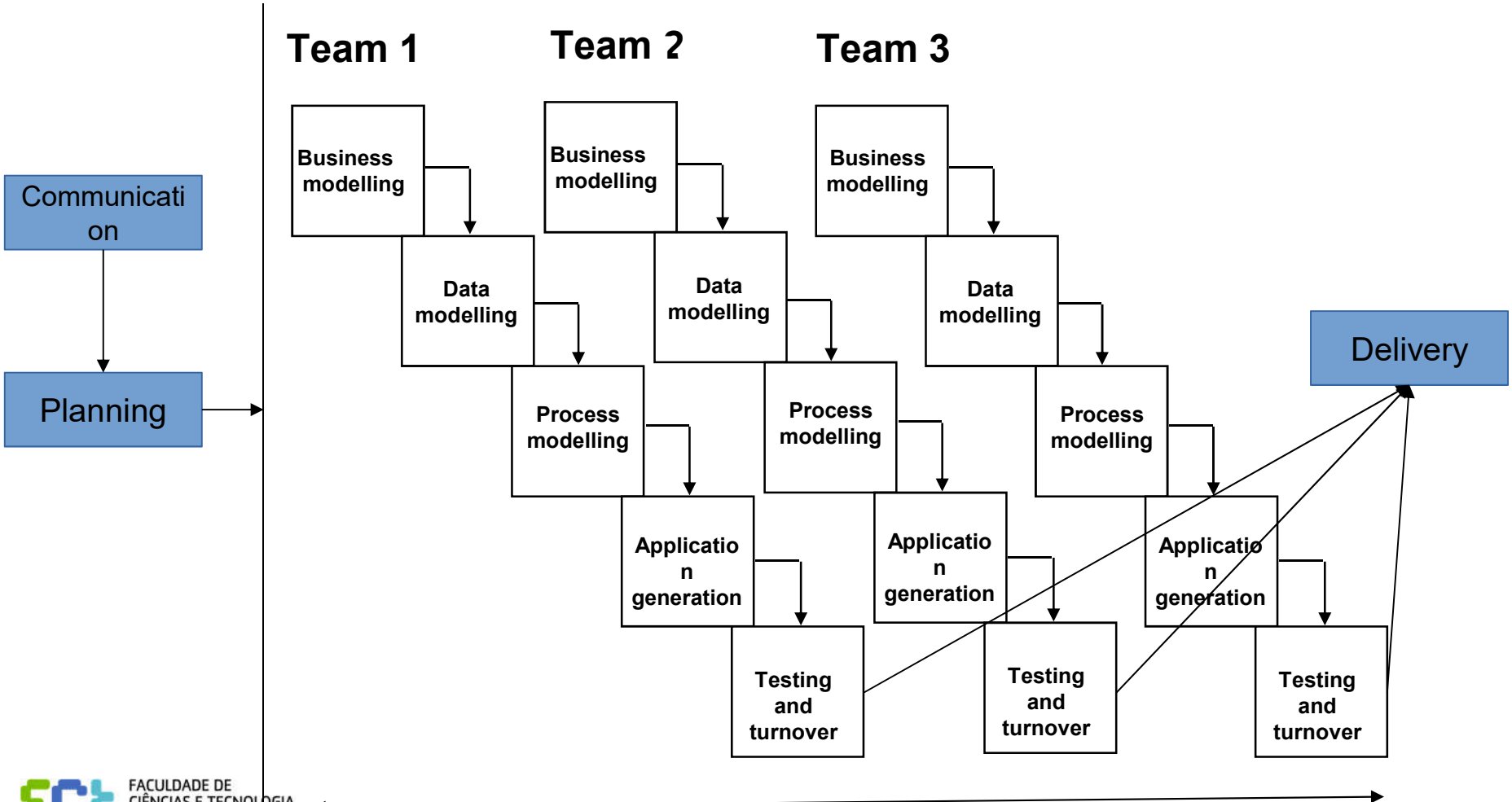
- Reflects the type of process model used in other engineering approaches
- Is still used when the software project is part of a larger system engineering project

- Adequate to Embedded, Critical and Large Systems
- Not adequate to informal team communication, or when requirements change quickly.

Waterfall Model variant: formal system development

- Mathematical model of a system specification is created
  - Model refined using mathematical transformations to executable code
  - Example: VDM, B method
- Adequate to critical systems engineering: systems with strong safety, reliability or security requirements
- Problems with high cost of formal specification

Rapid Application Development (RAD)



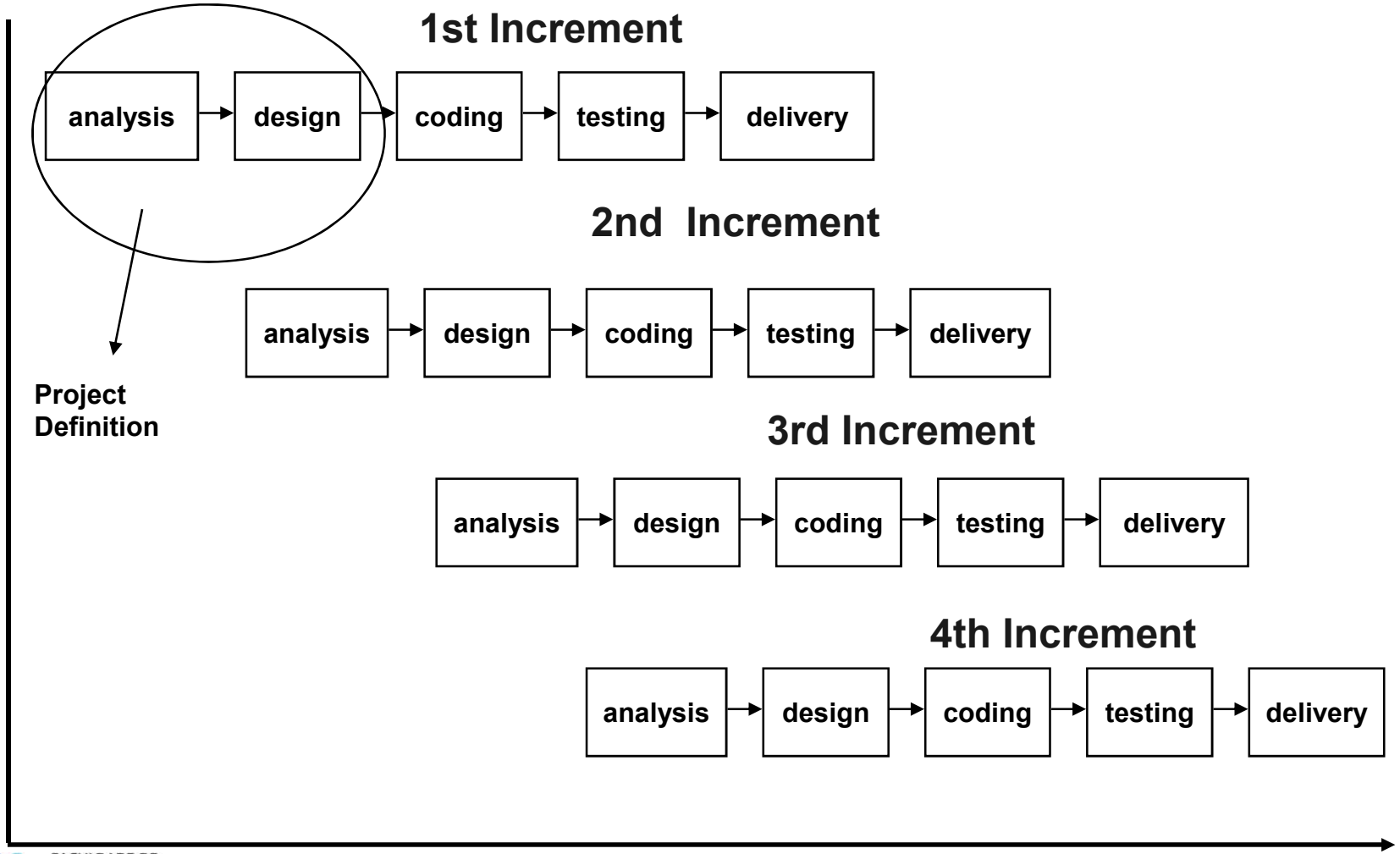
Rapid Application Development

- Similar to waterfall but uses a very short development cycle (60 to 90 days to completion)
- Uses component-based construction and emphasizes reuse and code generation
- Use multiple teams on scalable projects
- Requires heavy resources
- Requires developers and customers who are heavily committed
- Performance can be a problem
- Difficult to use with new technology

Rapid Application Development Problems

- For large projects **requires sufficient human resources** to create the right number of RAD teams
- **If the system cannot be modularized**, building the components necessary for RAD will be problematic
- **If high performance is an issue**, requiring to tune the several system components, it may not work
- It may not be appropriate when **technical risks** are high

Incremental Development



Incremental Development

- Applies an iterative philosophy to get feedback early with several versions
- Divide functionality of system into increments and use a linear sequence of development on each increment
- First increment delivered is usually the core product, *i.e.* only basic functionality
- Reviews of each increment impact on design of later increments
- Manages risk well

Incremental Development

Can be implemented with waterfall

- but needs to plan the increments in advance

Extreme Programming (XP), and other Agile Methods, are incremental

- but they do not implement the waterfall model steps in the standard order
- requirements are set only before each iteration

Incremental Process Model

- Ideal when the staff is unavailable for a complete implementation
- Increments can be planned to manage technical risks

Incremental Process Model

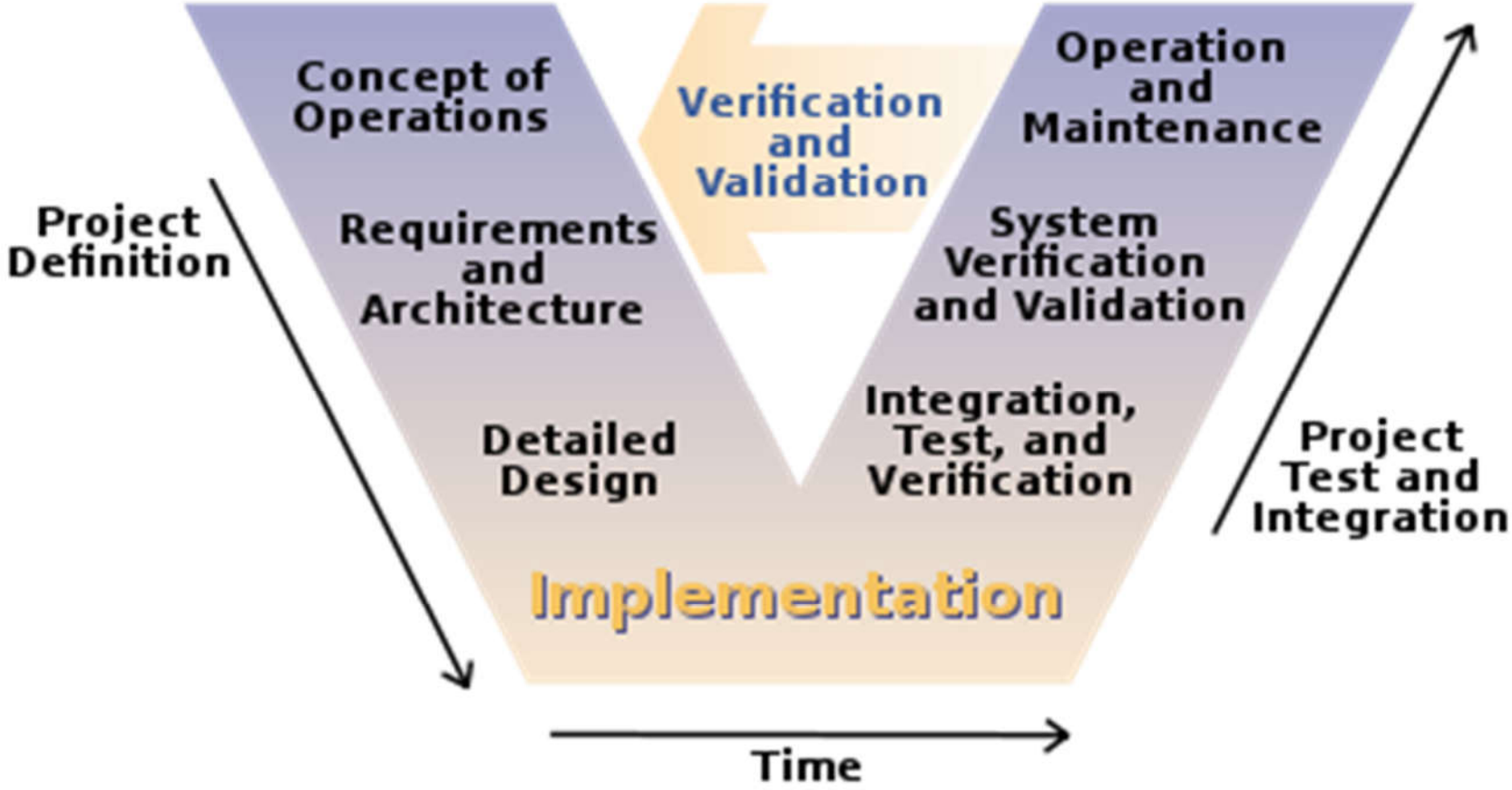
Positive:

- Cost of implementing is reduced
- easier to get customer feedback
- early delivery and deployment of useful software

Negative:

- The process is not visible (not cost effective to produce documentation for each iteration)
- system structure tends to degrade (Agile methods propose to often refactor)
- Might collide with organizations' bureaucracy

V-Model


Concept of Operations
Requirements and Architecture
Detailed Design
Implementation
Integration, Test, and Verification
System Verification and Validation
Operation and Maintenance
Verification and Validation
Project Definition
Project Test and Integration
Time

V-Model

Pros:

- Minimizes project risks - due to the explicit concerns:
  - Verification (Am I doing things right?)
  - Validation (Am I doing the right thing?)
- Improvement and guarantee of quality
- reduction of total cost over the entire project and systems life cycle

Cons:

- Apart from the mentioned benefits it suffers from the same problems of the waterfall model

Prototyping

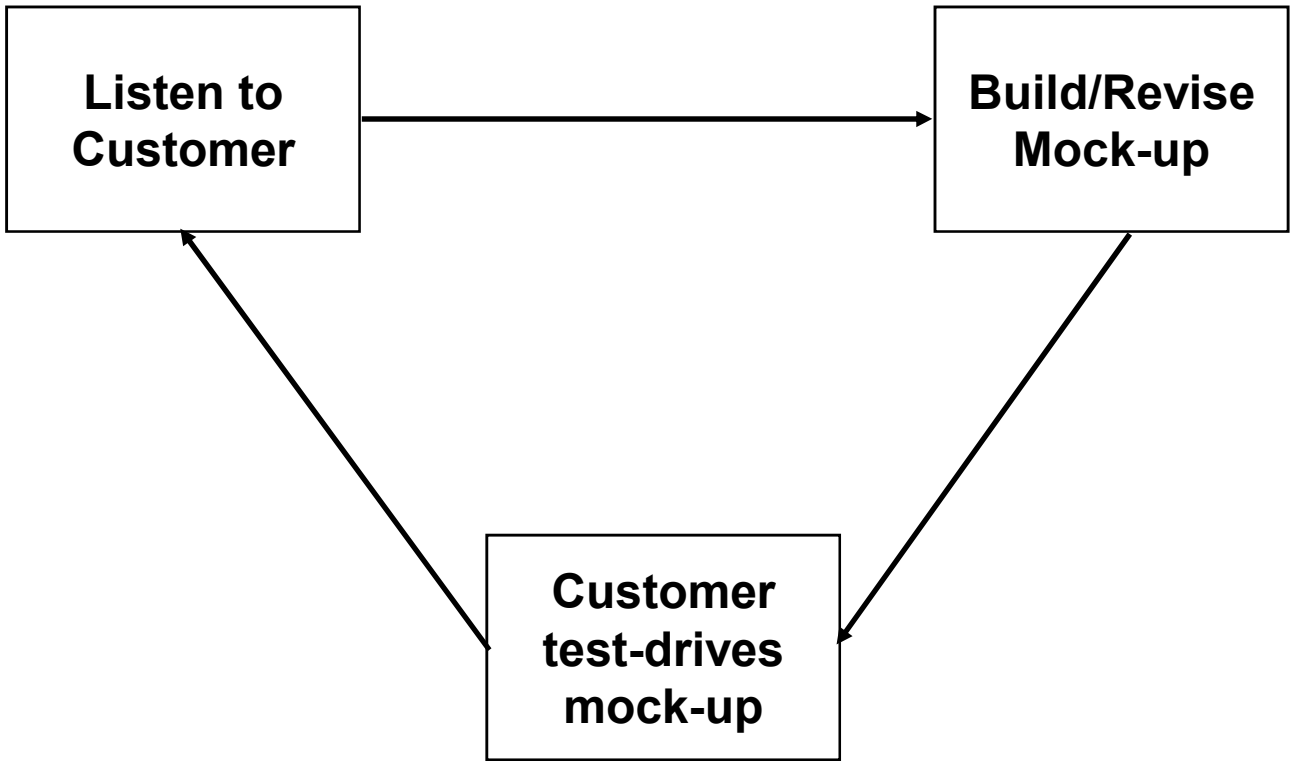
Exploratory development

- work with the customer to explore their requirements and deliver a final product, starting by the parts of the system that are understood. New proposed features are added.

Throwaway prototyping

- help on better understand the requirements and get a better requirements definition. The prototype concentrates on poorly understood requirements.

Prototyping



Prototyping

- Specifying requirements is often very difficult
- Users don’t know exactly what they want until they see it
- Prototyping involves building a mock-up of the system and using to obtain for user feedback
- Closely related to what are now called “Agile Methods”

Prototyping

Ideally mock-up serves as mechanism for identifying requirements

Users like the method, get a feeling for the actual system

Less ideally may be the basis for completed product

- prototypes often ignore quality/performance/maintenance issues
- may create pressure from users on deliver earlier
- may use a less-than-ideal platform to deliver e.g Visual Basic - excellent for prototyping, may not be as effective in actual operation

Prototyping – Shortcomings

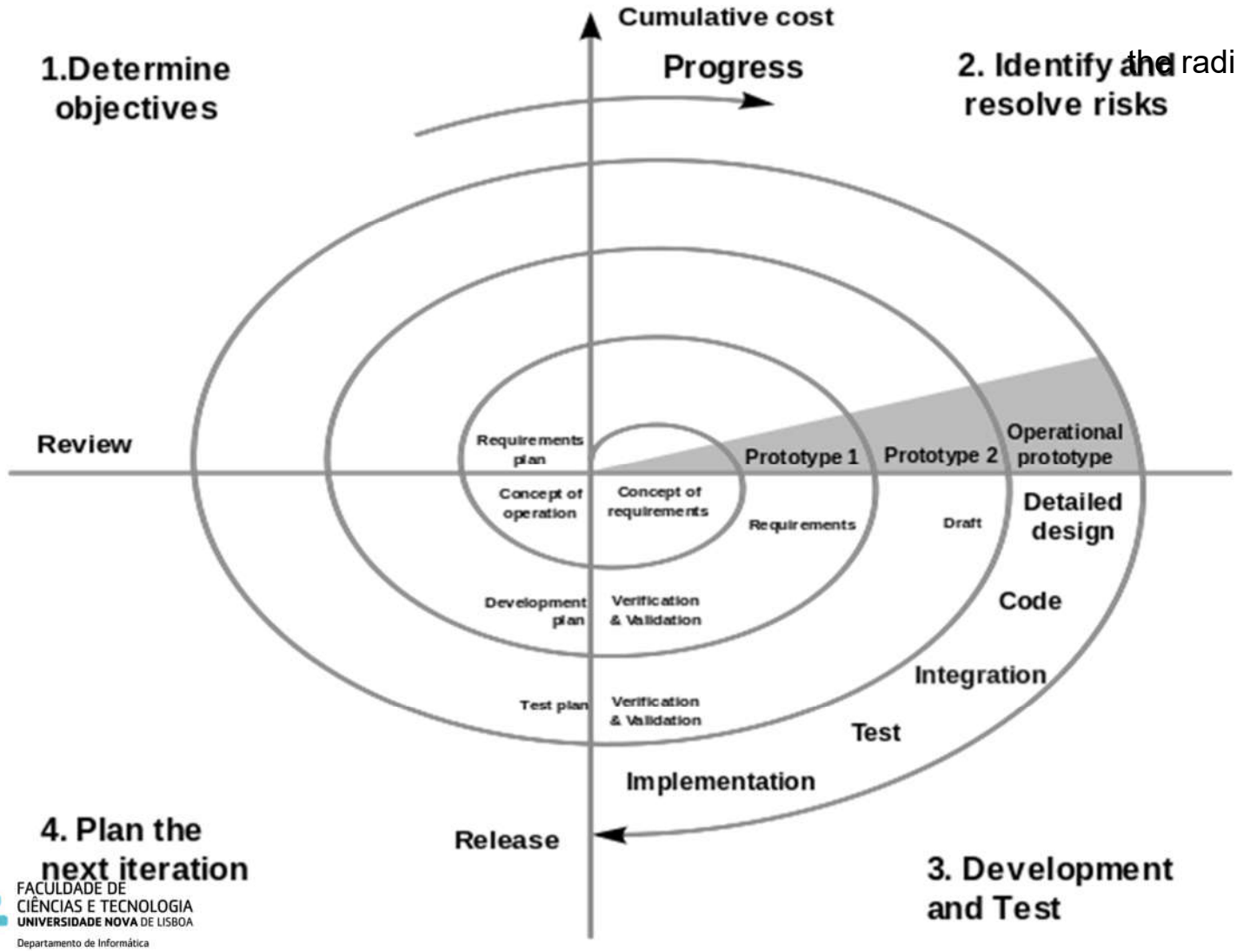
- The process is not visible – it is not cost effective to produce documents that reflect every version of the system
- Systems are often poorly structured – incorporating changes becomes increasingly costly and difficult
- Not adequate for large, complex long-lived systems with different teams developing different parts
- Difficult to establish a stable system architecture
- Usually should be used mixed together with waterfall: evolutionary approaches for uncertainties in specification (e.g. user interface) and waterfall for parts well understood.

Evolutionary Models

Process iteration

- The specification is developed in conjunction with the software
- There is no complete system specification until the final increment is specified.
- Requires new form of contract that large customers and Government agencies may find difficult to accommodate

The Spiral Model



the radius is the risk value

The spiral model

Development cycles through multiple (3-6) task regions (6 stage version)

- customer communication
- planning
- risk analysis
- engineering
- construction and release
- customer evaluation

Incremental releases

- early releases may be paper or prototypes
- later releases become more complicated

Models software until it is no longer used

The spiral model

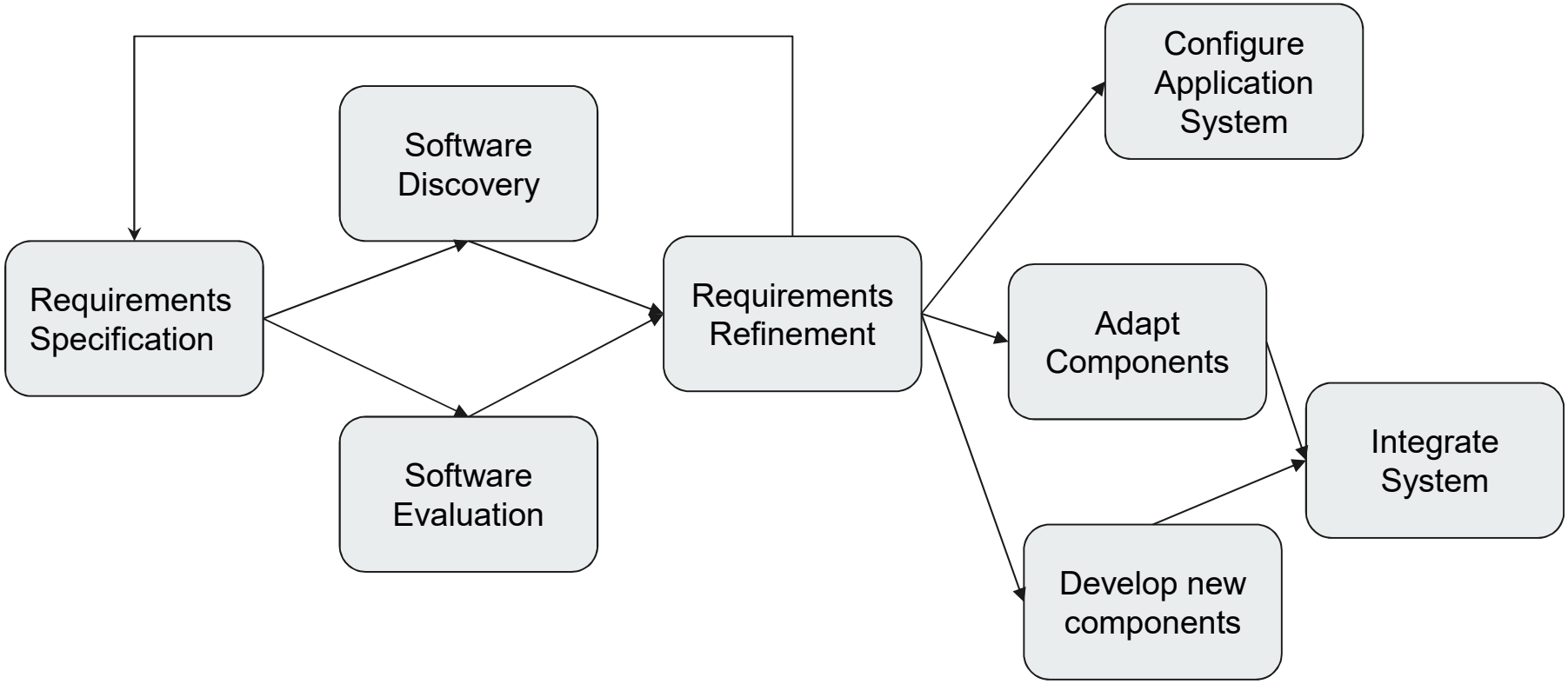
Good

- Not a silver bullet, but considered to be one of the best approaches
- Is a realistic approach to the problems of large scale software development
- Can use prototyping during any phase in the evolution of product

Bad

- Requires excellent management and risk assessment skills

Component Assembly



The majority of the projects have software reuse, components like:

- Stand alone applications
- packages
- web services

Component Assembly

Advantages

- Reduces amount of software to be developed
- reduces costs and risks
- Faster delivery

Disadvantages

- requirements compromises can lead to systems that don’t meet the user’s needs
- control over evolution is lost (not controlled by us)

Agile Software Development

Agile Methods – Motivation

Plan driven approaches were developed for software development by **large teams, working for different companies, often geographically dispersed working on software for long time.**

e.g. software for: Automotive, Avionics

But, too heavy for small and medium business systems more time spent on organizing and less on developing and testing

The Agile Manifesto values

The goal is to uncover better ways of developing software.
The Agile Manifesto values:

- **Individuals and Interactions** more than **processes and tools**
- **Working Software** more than **comprehensive documentation**
- **Customer Collaboration** more than **contract negotiation**
- **Responding to Change** more than **following a plan**

That is, while **there is value in the items on the right**,
the manifesto values the items on the left more.

Agile methods

Agile approaches focus on software and not on design and documentation.

e.g. small medium sized products and apps, custom system development within organization

Adequate when requirements change rapidly, meant to deliver working software quickly.

Agile principles

- Customer involvement
- Embrace change
- Incremental delivery
- Maintain simplicity
- People, not process

Agile requirements

User stories (collect stories)

Prescribe medication

- Kate is a doctor who wishes to prescribe medication for a patient attending a clinic. The patient record is already displayed on her computer so she clicks on the medication field and can select *"current medication"*, *"new medication"* or *"form"*.
- If she selects *"current medication"*...
- If she chooses *"new medication"*...
- If she chooses *"form"*...

User stories (break into tasks, refine and rank)

task 1: Change dose of prescribed drug

task 2: Formulary selection

task 3: Dose Checking

Dose checking is a safety precaution to check that the doctor has not prescribed a small or large dose. Using the formulary id for the drug name, look up the formulary and retrieve the recommended maximum and minimum dose

User stories – shortcomings

- Incompleteness
  - Do all the stories cover the essential requirements?
- Is a story a true picture of an activity?
  - A lot of implicit knowledge might be ignored

Agile development

Test first (implement later)

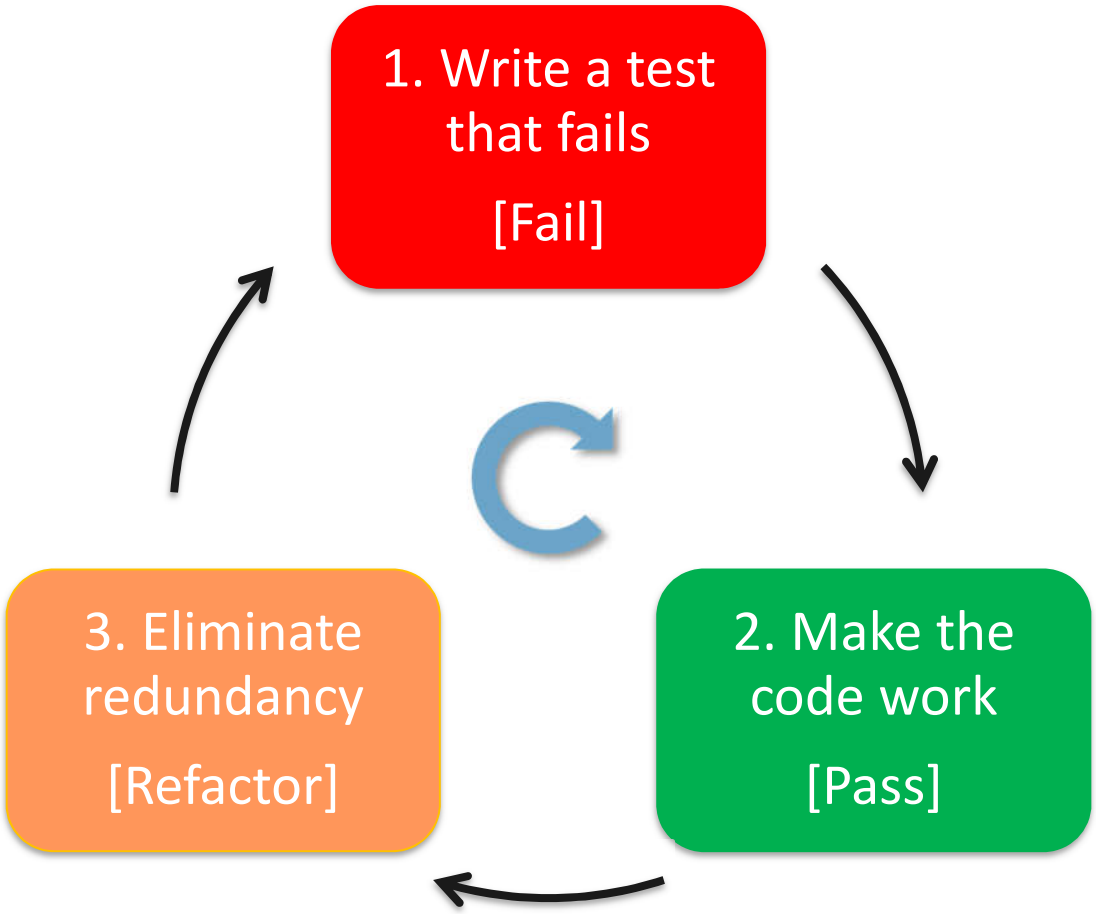
As it is not plan driven, an external test team can not base their work on the specification to build the tests

test first

- incremental test development from scenarios
- user involvement in test development and validation
- use automated testing frameworks

- Problems:
  - Judge completeness
  - Programmers shortcuts (they prefer code to tests)
  - Some tests are difficult to write incrementally

Test-driven development (TDD)



Pair programming

- shared code ownership and responsibility
- informal review process
- refactoring will promote sharing of benefits (and the programmer is not looked at as if being less efficient)

Extreme Programming (Kent Beck 1998)

Requirements as scenarios (user stories in story cards) implemented as tasks ranked by priority and time available

Programmers in pairs, writing tests before code

All tests must be successfully executed when integrating new code

Incremental planning with short time between releases adding functionality

Extreme Programming (Kent Beck 1998)

Customer involved in the development team, defines acceptance tests, responsible for requirements

Collective ownership

Continuous Refactoring

Criticism: hard to integrate with management practices

Refactoring

- XP sees planning for reuse as a waste of time
- Effort to build generalizable code to accommodate change that may not happen
- Implementation with the available knowledge at the time
- Code being developed should be constantly refactored, as new knowledge is acquired
- If team realizes that there is code to improve, it should be done immediately

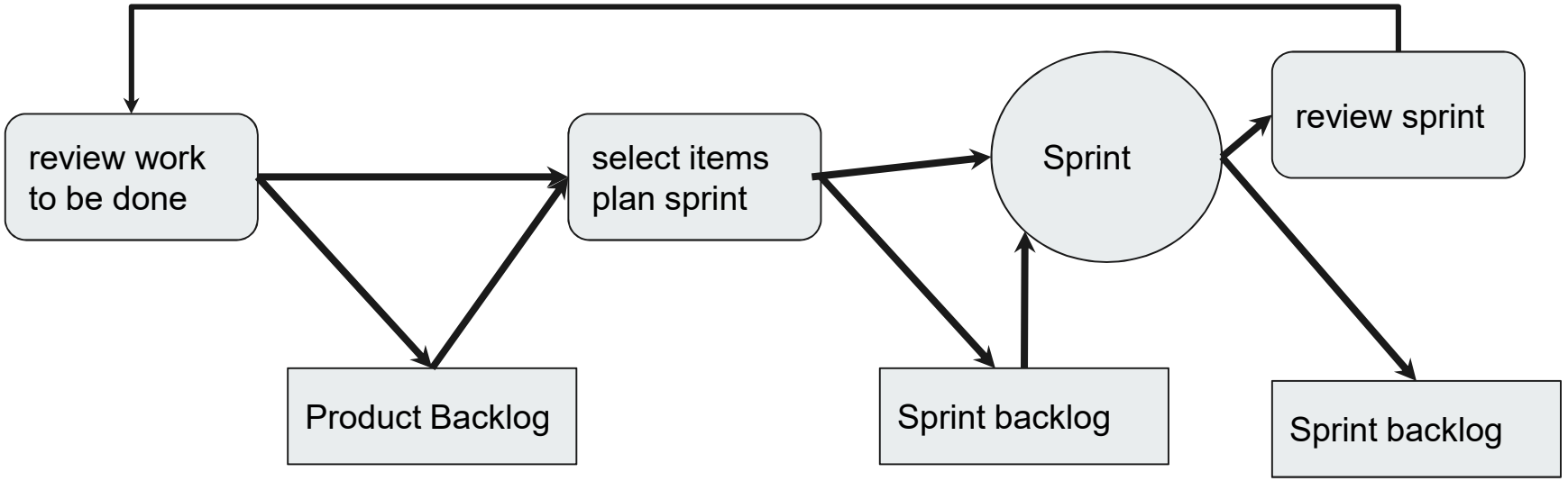
Agile Project Management

- Problem:
  - Original Agile processes were opaque to managers, because of:
  - self-organizing teams
  - did not produce documentation
  - planned developed in very short cycles
- Agile has to be managed for best use of time and resources
- Scrum agile method to organize agile methods

Scrum


WE'RE MOVING TO AN AGILE METHODOLOGY FOR SOFTWARE DEVELOPMENT.
I DON'T KNOW ALL OF THE DETAILS, BUT I THINK ONE OF YOU HAS TO BE DESIGNATED THE SCRUMBAG.
DOES THAT SOUND RIGHT?
IT'S BETTER THAN I EXPECTED.
Dilbert.com @ScottAdamsSays
© 2017 Scott Adams, Inc./Dist. by Andrews McMeel
2-6-17

Scrum

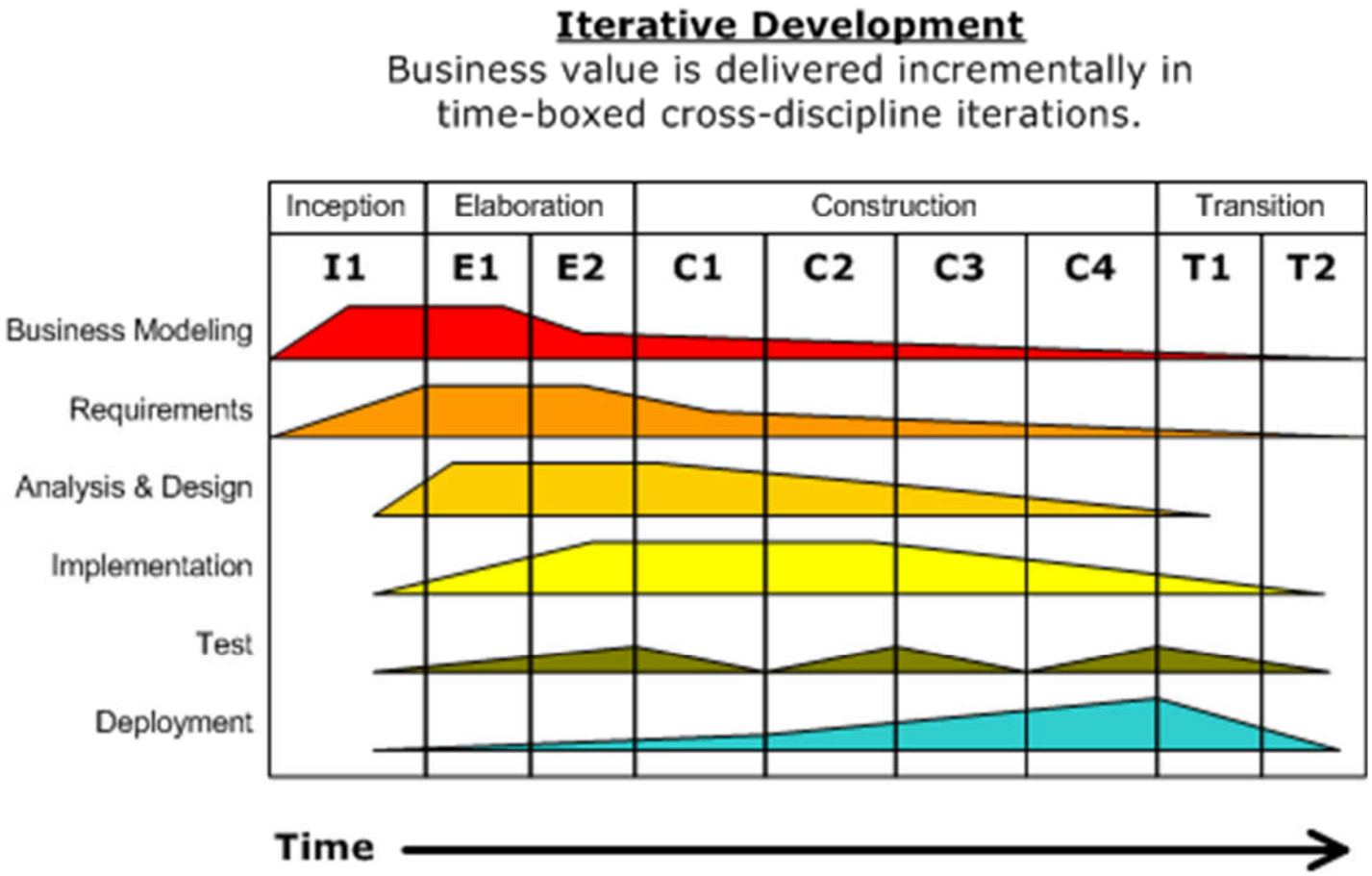

review work
to be done
select items
plan sprint
Sprint
review sprint
Product Backlog
Sprint backlog
Sprint backlog

Scrum

- Development team (no more than 7 people)
- Shippable product increment (incorporable in the final product)
- Product Backlog (list of “to do” activities)
- Product Owner (identify requirements)
- Scrum (daily face-to-face meeting of all team, where the team reviews and prioritizes work)
- ScrumMaster (responsible for ensuring Scrum process, avoid outside interference)
- Sprint (iteration, 2 to 4 weeks)
- Velocity (estimates how much product backlog effort a team can cover in a sprint)

Rational Unified Process (RUP)

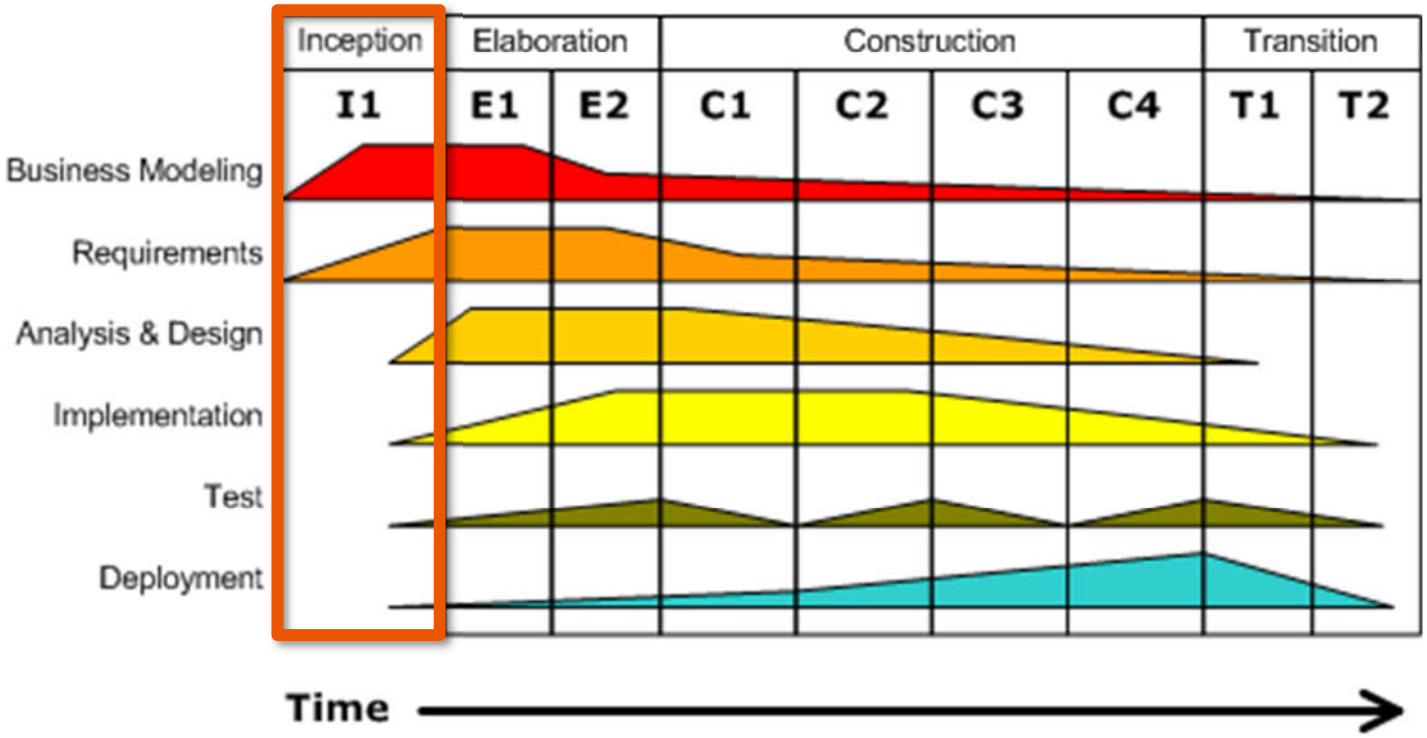
RUP phases


Iterative Development
Business value is delivered incrementally in time-boxed cross-discipline iterations.
Inception
Elaboration
Construction
Transition
I1
E1
E2
C1
C2
C3
C4
T1
T2
Business Modeling
Requirements
Analysis & Design
Implementation
Test
Deployment
Time

Define the scope the system

Inception - goal to get the project “off the ground”. Focus on Requirements and Analysis

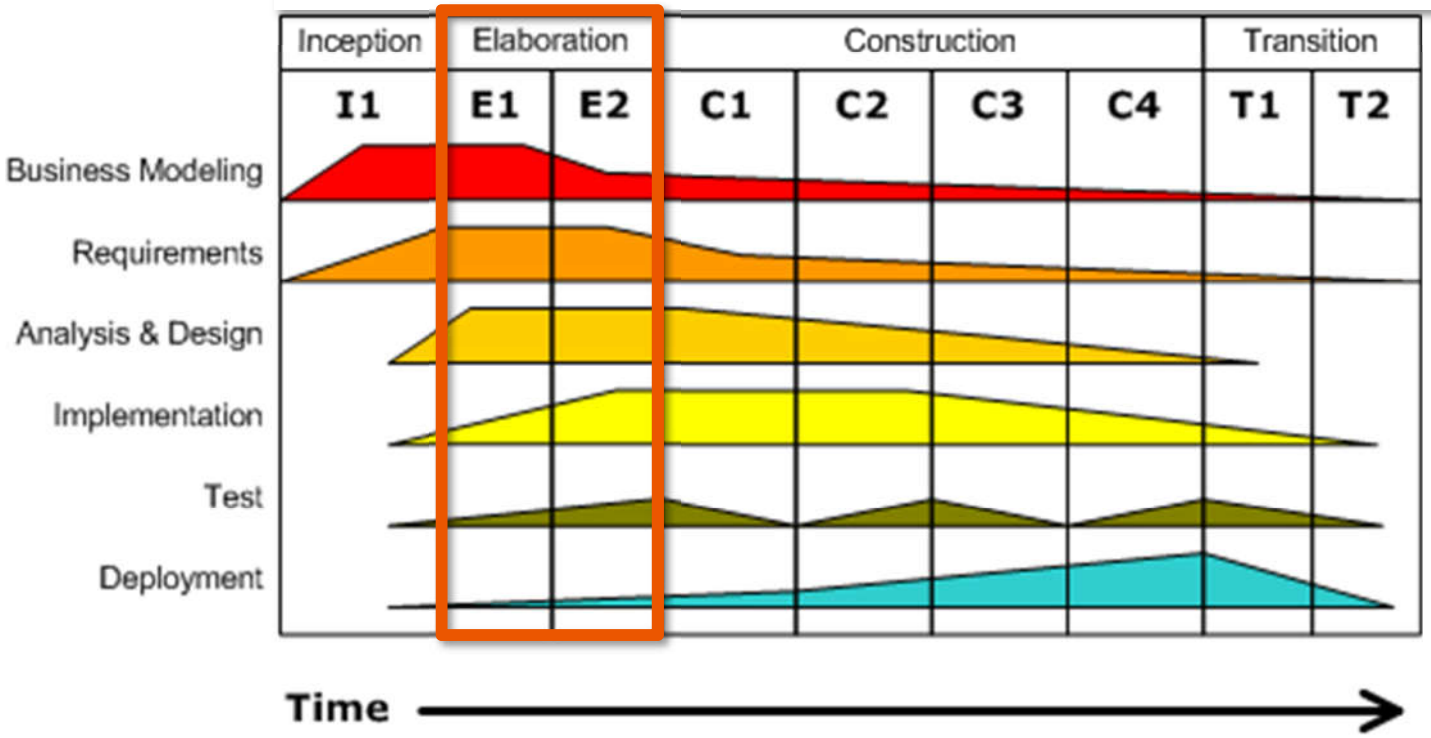
- Establish feasibility (may involve prototyping)
- Create business case
- capture essential requirements
- identify critical risks



Domain analysis and architecture defintion

Elaboration - build executable system built according to the specified architecture.

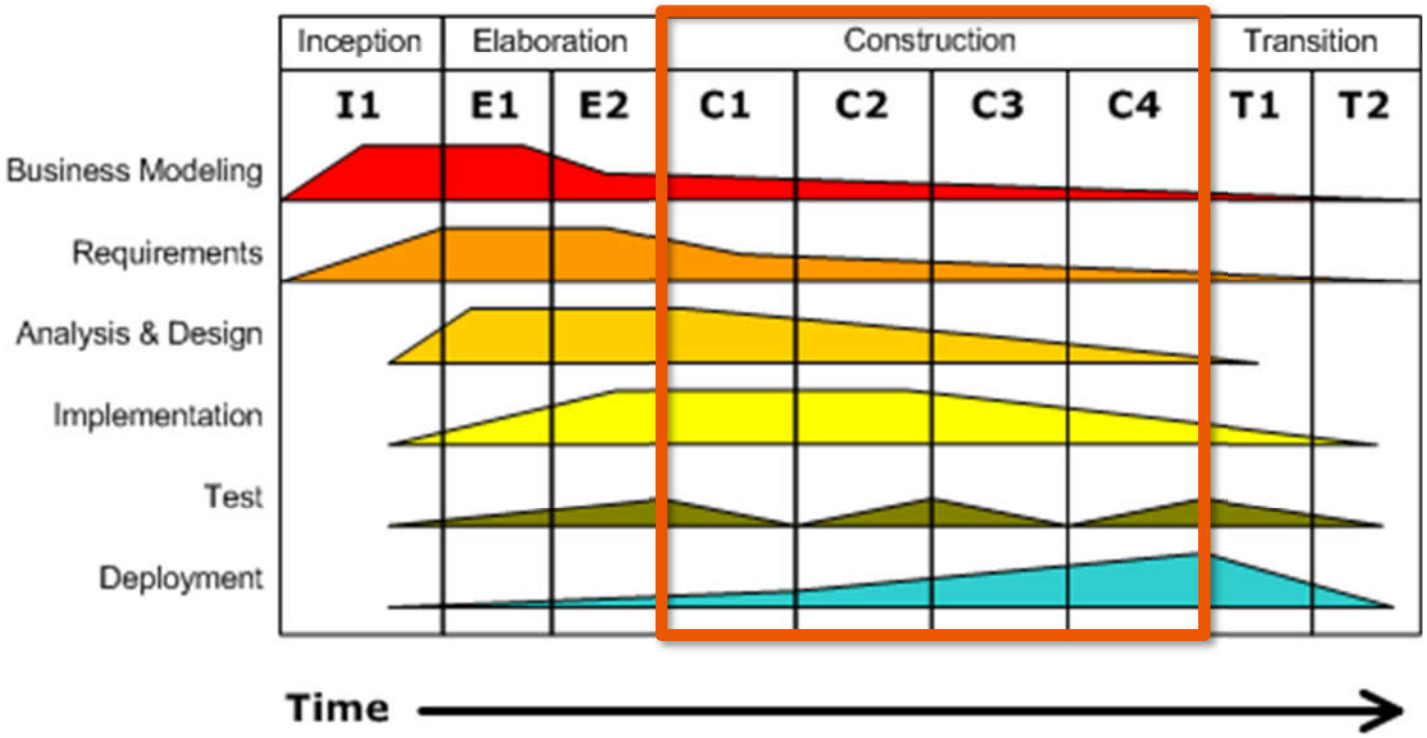
- Create executable
- refine risk assessment
- define quality attributes
- capture 80% functional requirements
- create detailed plan of construction
- formulate bid including resources, time, equipment, staff and cost



This is where the bulk of the development occurs

Construction - focus on the implementation workflow.

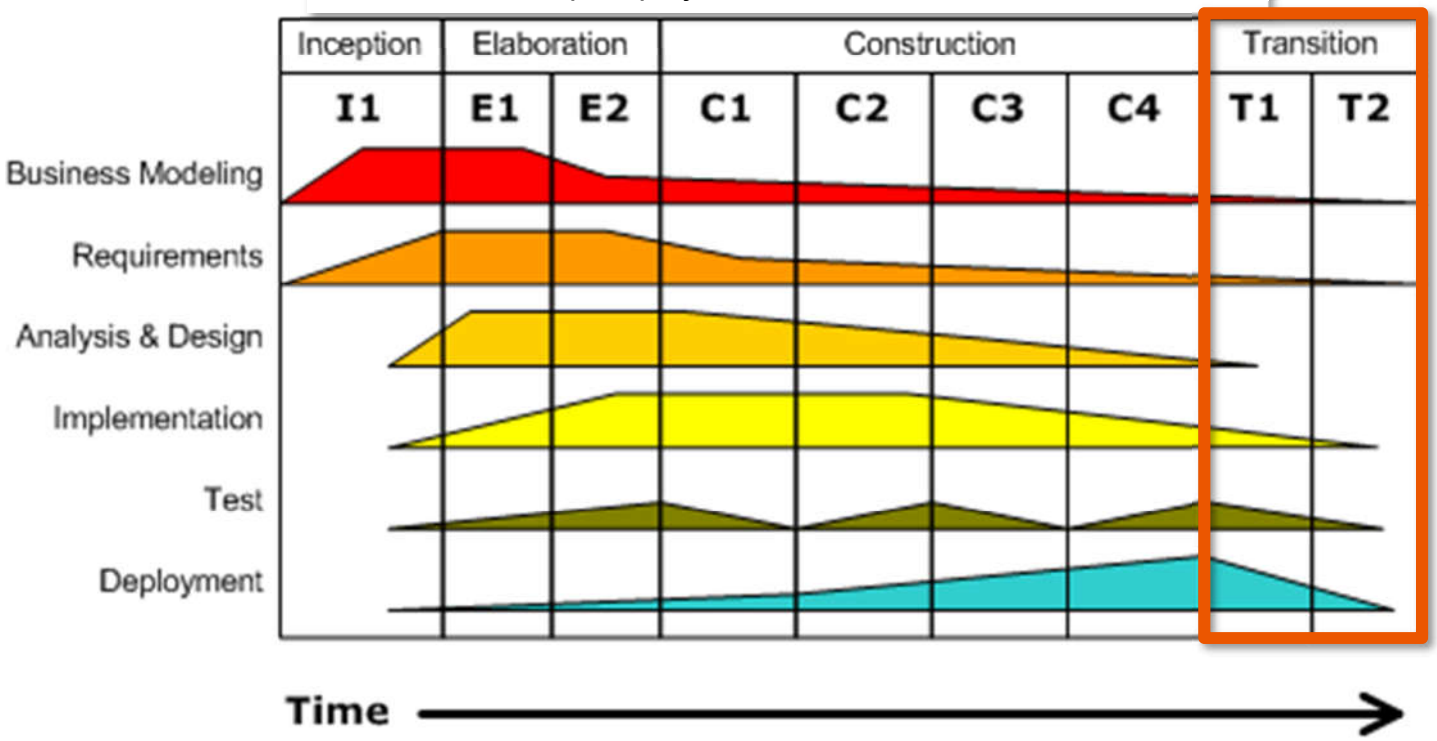
- complete requirements
- complete analysis
- complete design
- build the initial operational capability
- test the operational capability



The system goes to production (to customers)

Transition - focus on implementation and test workflows

- correct defects
- prepare user sites
- tailor software
- modify the software if unforeseen problems arise
- create manuals and other documentation
- conduct a post project review



Rational Unified Process – RUP

- Software Engineering Process that defines the who, what, when and how of Developing Software with UML as the visual language
- A framework for object-oriented Software Engineering using UML
- Use-case driven, architecture-centric, iterative and incremental software process

Main takeaways

Choosing an appropriate process model is crucial:
- How well do we understand the requirements?
- What is the expected lifetime of the project?
- What is the level of risk involved?
- What are the schedule constraints
- What is the expected level of interaction with the customer?
- What is the level of expertise of our team members with that process?

Bibliography

Ian Sommerville, Software Engineering, Tenth Edition, Chapters 2, 3