MSO
Development Process

Hans Philippi
(based on the course slides of Wouter Swierstra)

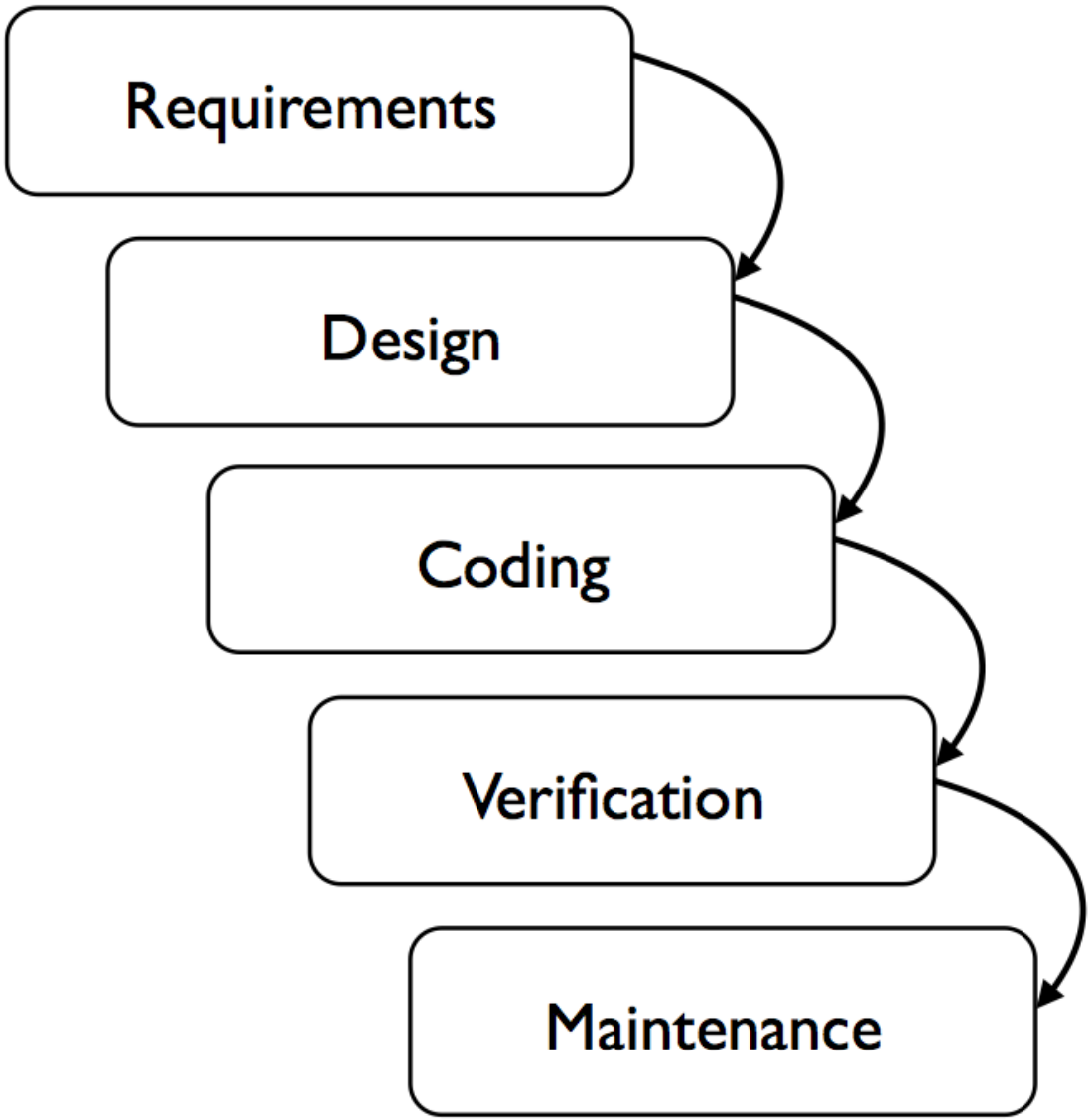
August 20, 2018

Development Process

How can I manage the process of constructing complex software?

- In areas such as physics, there are fixed methodologies for experiments and measurements
- This translates into concrete processes for in engineering disciplines, such as architecture
- But what about Computer Science?

Waterfall model (Royce, 1970)



Criticism on the waterfall model

- Traditionally, the waterfall model aims to complete every phase before moving on to the next
- Critics complain that the waterfall model is not robust against *change*:
  - A client can change his mind halfway through the project
  - The money can run out
  - A design can turn out to be too costly or complex to implement

"Many of the [system's] details only become known to us as we progress in the [system's] implementation. Some of the things that we learn invalidate our design and we must backtrack." – David Parnas

Alternative software Development Processes

- The spiral model (Boehm, 1988)
- Rapid Application Development
- The (Rational) Unified Process
- Agile development
- eXtreme Programming
- .. and many others

All these methods aim for a more incremental or evolutionary approach to software construction

About the software Development Processes

- Managing software development is a *hard problem*
- But choosing a methodology is a bit like choosing an operating system or text editor:
  - Each system is slightly different
  - There is no clear winner
  - There are zealous advocates for each system
- Our advice:
  - Learn to apply one methodology really well
  - Don't get stuck in a rut – stay hungry!
  - Always choose the right tool for the job

The Unified Process

- In this course, we will stick to one methodology – and how it meshes with the ideas of object orientation
- We will focus on the *Unified Process* (Jacobson, Booch and Rumbaugh, 1999)
  - It is one of the most popular philosophies used today, ...
  - ... but it is not as 'cool' as the more recent Agile/Scrum methodologies ...
  - ... although it can be adapted to work together with such other methodologies

The Unified Process

- The Unified Process emphasises *iterative software development*
- Develop software in time-boxed mini-projects called *iterations*
- Each iteration has clearly defined milestones, including a tested, integrated, executable system

Iterative development – philosophy

- The iterative lifecycle is based on the successive enlargement and refinement of a system though multiple iterations with feedback and adaptation
- The system grows incrementally over time, iteration by iteration
- The system may take many iterations to be production ready (or ready for deployment)

Iterative development – implementation

- The output of an iteration is not (just) an experimental prototype but a production subset of the final system
- Each iteration tackles new requirements and incrementally extends the system
- An iteration may occasionally revisit existing software and improve it

Embracing change

- Stakeholders usually have changing requirements.
  - End users
  - Project management
  - Upper management
  - Implementors
  - Testers
  - ...
- Each iteration involves choosing a small subset of the requirements and quickly design, implement and testing them
- This leads to rapid feedback, and an opportunity to modify or adapt understanding of the requirements or design

Unified process – phases

- UP defines four different phases, each split into multiple iterations:
  - Inception – Define the scope of the project
  - Elaboration – Plan the project, specify features, baseline architecture
  - Construction – Finish the construction
  - Transition – Hand over the project to end users
- Each phase overlaps with some elements of the waterfall model, ...
- but the balance between them shifts as time passes.

The Unified process: Best practices

- Develop software iteratively – involve users early!
- Manage requirements
- Visually model software (UML)
- Verify software quality – test, code reviews, release in every iteration
- Embrace change

The Unified Process

- Today we will see an *overview* of the UP...
- ... but applying the UP to your own projects is your responsibility
- This session is mostly focused on the first two stages of the UP: Inception and Elaboration

Inception

Inception

- Envision the product scope, vision, and business case
- Upon completion, the stakeholders have a basic agreement on the vision of the project and are able to decide whether to continue or not
- It typically only lasts ‘a few’ weeks

Inception: artifacts

- A vision document – what is the general vision of the project? What are the key features and constraints?
- A initial list of use-cases – how will end-users interact with the system?
- An initial project glossary – what is some of the domain specific lingo?
- An initial business case – how will we make money?
- An initial risk assessment – what might go wrong?
- A project plan, showing phases and iterations – how will we move forward?
- One or several prototype experiments

What Inception is and is not

- It's not about deciding how many weeks feature X will take to implement
- You need a ballpark estimate: 1 month or 1 year? 1M or 10K euros?

What Inception is and is not

- It's not about identifying every possible interaction that every imaginable user can have with your system
- You want to have an idea of which people will end up using the system
- You should have a few carefully thought out *use cases*

What Inception is and is not

- It's not about choosing the color of the icon, which GUI library to use, or which operating system to support
- You do want to think about whether it will run on tablets, mobile phones, desktops, or in the cloud

About risk

It can be very hard to identify where the *risks* lie in any project:

- Example: changing the limit on a creditcard
- Example: restaurant matching website

So much for Inception

Elaboration

Elaboration: goals

- Analyze the problem domain, establish an architectural foundation, develop a project plan, and eliminate highest risks
- You need to develop a 'mile high and inch deep' view of the system
- Now is the time to make architectural decisions
- Build an executable architecture prototype, thereby eliminating critical risk, for the central use cases developed in the Inception phase
- *Note*: you should start building a prototype early, even if the requirements have not been finalized yet

Elaboration: artifacts

- A use-case model describing how users will interact with the system
- Supplementary requirements capturing the non-functional requirements
- A Software Architecture Description
- *... a what?*

Software Architecture

- *Software Architecture* is a very popular term in the context of software development
- ... but when you try to read the corresponding lemma of Wikipedia (English), you might get a little bit confused
- Form Larman (p. 448):
  [...] the architecture includes - as the prior definition indicates - the organization and structure of the major elements of the system. Beyond this static definition, it includes the system behavior, especially in terms of large scale responsibilities of systems and subsystems, and their collaborations. In terms of a description, the architecture includes the motivations or rationale for why the system is designed the way it is.

Elaboration: artifacts

- A use-case model describing how users will interact with the system
- Supplementary requirements capturing the non-functional requirements
- A Software Architecture Description
- An executable architectural prototype
- A revised risk list and a revised business case
- A development plan for the overall project, including the coarse-grained project plan, showing iterations and evaluation criteria for each iteration

Elaboration outcomes

- A stable vision of the software system and its architecture
- Minimized the risks that could cause the project to fail if you choose to continue
- Better estimates for project costs and planning

So much for Elaboration

Construction

Construction: goals

- The main goal of the construction phase is to develop and test a baseline product
- Upon completion, there should be a clear description of the product status, together with user manuals. The tool should be ready to be deployed – even if not all features are fully implemented.

So much for Construction

Transition

Transition

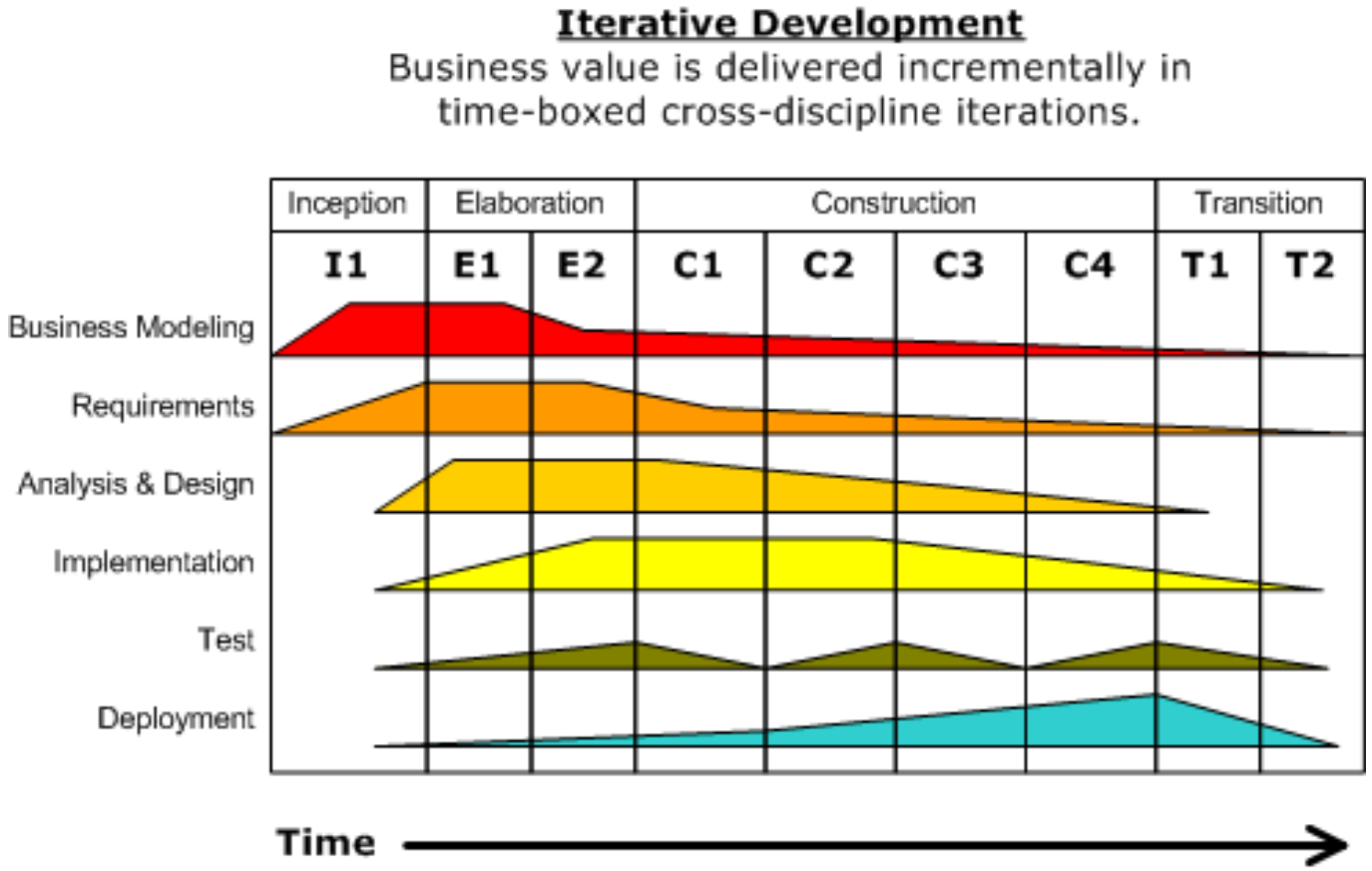
- The transition aims to deliver the first version of the software to its users
  - Beta testing
  - Training of new users and maintainers
  - Operating in parallel with existing legacy systems
  - Legacy issues: e.g. conversion of data bases
  - ...
- This process usually involves a lot of tuning, bug-fixing, processing enhancement requests, and implementing unfinished features

Transition: goals

- Enabling users to use your product
- Achieving consensus with all stakeholders that the baseline product is complete;
- Iteratively refining the baseline until the final product is implemented

So much for Transition

Unified process – phases in time



Activities

- UP identifies six distinct kinds of 'engineering activity'
  1. Business modeling
  2. Requirements engineering
  3. Analysis & Design
  4. Implementation
  5. Test
  6. Deployment
- And three kinds of 'supporting' activity:
  1. Project Management workflow
  2. Configuration and Change Management
  3. Environment: development kit, tools for building and process control
- For this moment, we will focus on the first three engineering activities

The first three engineering activities

Business Modeling aims to document existing business processes to establish a common understanding between various stakeholders about how the organization works

Requirements Engineering aims to describe *what* the system should do, allowing customers and developers to agree on that description

Analysis and Design aims to describe *how* the system should be implemented, in accordance with the requirements it should satisfy

Material covered

- Rational Unified Process Whitepaper – available on the MSO website.
- Craig Larman. Applying UML and Patterns. Pearson Education. 2002. Chapters 1–4.