Software Methods and Processes

- [ ] Methods
  - [x] Waterfall (paper Royce)
  - [ ] Scrum
  - [ ] DevOps
  - [x] Agile Manifesto
  - [x] RUP
  - [x] XP
- [x] Read about software methods and processes.
- [x] Provide a summary of these processes.
- [x] Explain what the difference is between them.
- [ ] Explain what problems they try to solve and how they approach it.

Method comparisons

Following this section are my summaries (liberaly quoting where others do better) of the waterfall paper, agile manifesto, RUP and XP.

Method	Year	Tries to solve	Approach
Waterfall	1970-1990	Deliver software on time and within budget	Rigid documentation and process, large design phase
RUP	1998+	Deliver high-quality software on time and within budget	Embodiment of iteration and testing, linking best-practices from disciplines, visual modelling on process
Agile	2001+	Distance between customer and developer (or software)	Trust the individual, deliver fast, collaborate with customer, prepare for change
XP	1996	Productivity burden	Short cycles, evolving planning, automate drudge, communicate structure through code, tests and oral communication

Similarities

Continuous delivery / being lean / not wasting resources (not throwing things out):

> > **Royce:**
> > **"What we have is an effective fallback position that tends to maximize the extent of early work that is salvageable and preserved."**

> > **Agile:**
> > **"Frequent delivery of working software provides stakeholders with concrete feedback"**
>
> > **RUP iteration:**
> > **"... which grows incrementally from iteration to iteration to become the final system"**
>
> > **XP:**
> > **"In XP you only do what you need to do to create value for the customer. You can't carry a lot of baggage and move fast."**

With regards to that last quote from XP, recall that Royce stated "also the kind of development effort for which most customers are happy to pay, since both steps involve genuinely creative work which directly contributes to the usefulness of the final product".

Specifically between RUP and waterfall

Between Royce's system requirements and RUP's inception phase:

> **RUP Inception phase:**
> **This involves identifying all use cases and describing a few significant ones**
>
> **Royce's System requirements:**
> **"Begin the design process with program designers, not analysts or programmers."**

Identical goals for RUP and Royce:

> **RUP's goal:**
> **"high-quality software that meets the needs of its end-users, within a predictable schedule and budget"**
>
> **Royce's definition of project success:**
> **"arriving at an operational state, on-time, and within costs"**

Special demands for a critical phase:

> **RUP Elaboration phase:**
> **"... you must have the "mile wide and inch deep" view of the system."**
>
> **Royce's Preliminary Program Design:**
> **"In this case a very special kind of broad competence is required on the part of the personnel involved"**

Differences

Role of documentation

Most clearly between XP and Waterfall, as should be expected.

> **Royce:**
> **"The first rule of managing software development is ruthless enforcement of documentation requirements."**
>
> **XP:**
> **"Reliance on oral communication, tests, and source code to communicate system structure and intent."**

Role of testing

Here we see a gradual shift from Royce's strict sequential approach (testing "phase"), via RUP's "larger allocation towards testing" (i.e. test the use-cases, plans, stakeholders, everything, as well as the construction phase), towards XP and Agile's aproach (embodies testing), up to Scrum (definition of done: passing regression tests).

Perspective

- Project management
  - Waterfall
  - RUP
- Programmer / Developer
  - Agile
  - XP

> **First, there was a distinct move towards collaborative development, with people being accorded privileges over processes that formerly constrained them.**

- A decade of agile methodologies
  Torgeir Dingsøyr

Waterfall

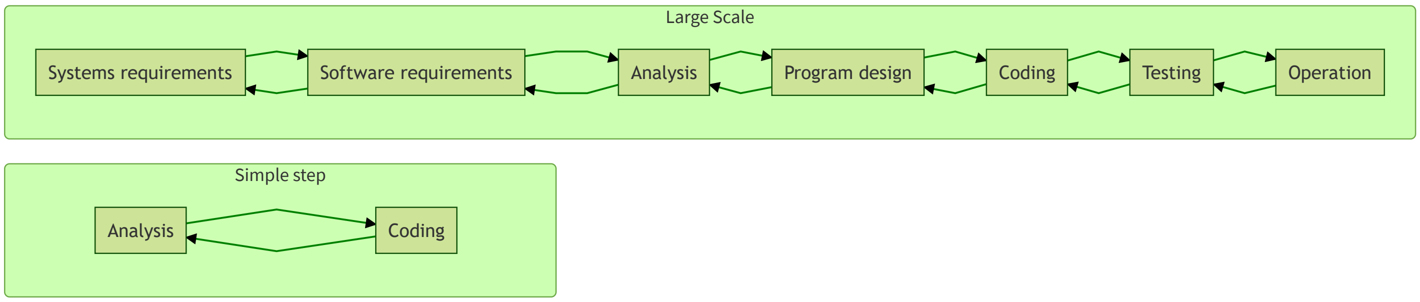
In his "Waterfall" paper from 1970, Royce documents based on personal experience his view on how large scale software development should be organized. Waterfall has become the de-facto "old-way" (and therefore *bad way*) of doing software development, as we have linked it to everything non-agile:

- Strong and rigid development process

- Distinct phases
- Slow cycles

This is to an extend valid, but the fundamental premise of the paper is that all software developent starts with **steps** of **analysis** followed by **coding**. Sounds very agile indeed! Noting that "many" additional steps are required for larger projects seemingly invisible to the customer, he sets out a general framework around the simple steps.

> **"They are preceded by two levels of requirements analysis, are separated by a program design step, and followed by a testing step."**



We don't get an anwer why these steps are necessary and sequential, only that they are **"distinctly different in the way they are executed"**, hinting it might make sense from a project management and resource allocation perspective. However he also admits **"the design iterations are never confined to the successive steps"**.

Perhaps key to Royce's philosophy is building on something known to be good.

> **What we have is an effective fallback position that tends to maximize the extent of early work that is salvageable and preserved.**

However it fails to tackle the implicit uncertainty of software itself, leaving testing as the ultimate test:

> **Yet if these phenomena fail to satisfy the various external constraints, then invariably a major redesign is required [...] The required design changes are likely to be so disruptive that the software requirements upon which the design is based and which provides the rationale for everything are violated.**

Excerpts and notes

Where does the documentation come from?

> **The first rule of managing software development is ruthless enforcement of documentation requirements.**
> **...**

> **The real monetary value of good documentation begins downstream in the development process during the testing phase and continues through operations and redesign.**

On a possible preliminary program design phase

> **In this case a very special kind of broad competence is required on the part of the personnel involved**

On the un-importance of actual coding

> **One might note that there has been a skipping-over of the analysis and code phases. One cannot, of course, produce software without these steps, but generally these phases are managed with relative ease and have little impact on requirements, design, and testing.**

Royce's definition of project success: arriving at an operational state, on-time, and within costs

Waterfall Summary

Design first, as much as possible. Testing is a separate stage. Document everything, ensure rigid process and adherence to standards.

Rational Unified Process (RUP)

The **Rational Unified Process** was designed by Rational Software (IBM), extending previous efforts by Phillippe Kruchten on the 1996 Rational Objectory Process (ROP) and itself a specific and detailed instance of a more generic process described by Ivar Jacobson, Grady Booch, and James Rumbaugh in the textbook, The Unified Software Development Process.

What is the Rational Unified Process?

- It is a **software engineering process**:
  It provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end-users, within a predictable schedule and budget.

- It is supported by **tools**:
  They are used to create and maintain the various artifacts—models in particular—of the software engineering process

- It is a guide for how to effectively use the **Unified Modeling Language** (UML)
  The Unified Process emphasizes the development and maintenance of models [...] The UML is an industry-standard language that allows us to clearly communicate requirements, architectures and designs.

The Rational Unified Process describes how to effectively deploy commercially proven approaches to software development for software development teams.

Best practices advertised

Iterative development

Account for changing requirements in planning, progressive integration, mitigate risks earlier (written for project managers?)

Requirements management

Systematic eliciting, organizing, communicating and managing requirements of a software application.

Architecture and use of Components

Early focus on creation and validation of software architecture which gruadually evolves into a final system. Multiple views on architecture (design, project management), capture style, choices and constraints.

A software component is a non-trivial piece of software (module, package, system), which has a clear boundary and can be integrated into a well-designed architecture. These components integrate into the final system.

Model visually

For instance, with UML.

Verify quality

By making testing a major part of any project.

Control changes

Make sure that changes made to a system are synchronized and verified constantly

Process

The process can be described in two dimensions, or along two axis:

- Time (phases and iterations)
- Who is doing what

Phases and Iterations - The Time Dimension

This is the dynamic organization of the process along time, each phase is concluded with a well-defined milestone. Each phase can be further broken down into iterations. An iteration is a complete development loop resulting in a release (internal or external) of an executable product, a subset of the final product under development, **which grows incrementally from iteration to iteration to become the final system**.

Similar to vision of Royce

Inception phase

During the inception phase, you establish the **business case** for the system and delimit the project scope. To accomplish this you must identify all external entities with which the system will interact (**actors**) and define the nature of this interaction at a high-level. This involves **identifying all use cases** and **describing a few significant ones**. The business case includes success criteria, risk assessment, and estimate of the resources needed, and a phase plan showing dates of major milestones.

Similar to the Waterfall system requirements

Elaboration phase

The purpose of the elaboration phase is to **analyze the problem domain**, establish a **sound architectural foundation**, develop the project plan, and **eliminate the highest risk elements** of the project. To accomplish these objectives, you must have the **"mile wide and inch deep"** view of the system. Architectural decisions have to be made with an understanding of the whole system: its scope, major functionality and nonfunctional requirements such as performance requirements.

Similar to the Waterfall preliminary design phase

With regards to the criticality of this phase:

> **For most projects, this also corresponds to the transition from a mobile, light and nimble, low-risk operation to a high-cost, high-risk operation with substantial inertia.**

Construction phase

During the construction phase, **all remaining components** and application features are developed and integrated into the product, and all features are thoroughly tested. **The construction phase is, in one sense, a manufacturing process** where emphasis is placed on managing resources and controlling operations to optimize costs, schedules, and quality. In this sense, the management mindset undergoes a transition from the development of intellectual property during inception and elaboration, to the development of deployable products during construction and transition.

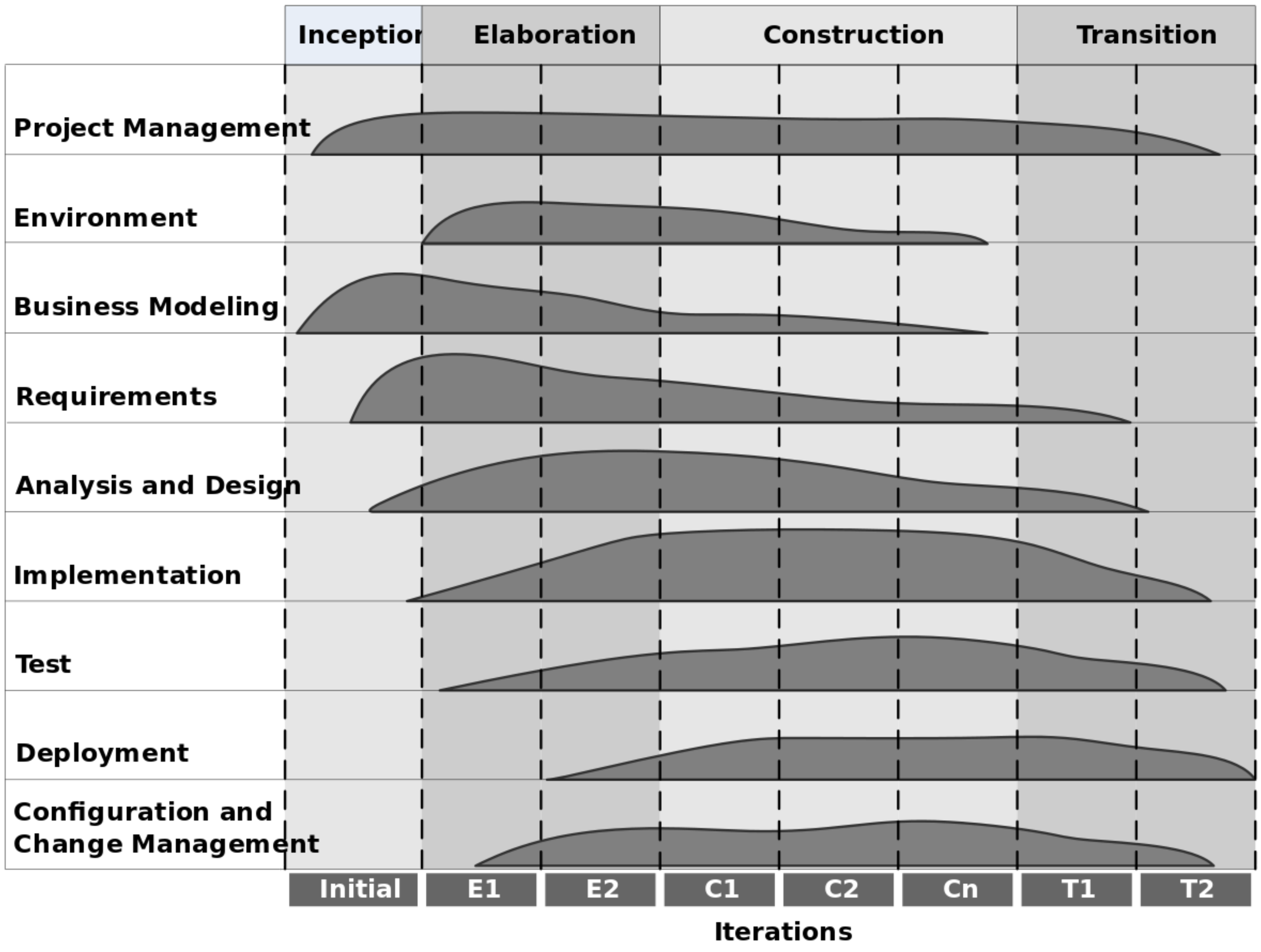
Transition phase

The purpose of the transition phase is to **transition the software product to the user community**. Once the product has been given to the end user, issues usually arise that require you to develop new releases, correct some problems, or finish the features that were postponed.

Static Structure - The who and what

The Rational Unified Process is represented using four primary modeling elements:

- Workers, the ‘who’
  - The worker is more the role defining how the individuals should carry out the work. The responsibilities we assign to a worker includes both to perform a certain set of activities as well as being owner of a set of artifacts.
- Activities, the ‘how’
  - An activity of a specific worker is a unit of work that an individual in that role may be asked to perform. The activity has a clear purpose, usually expressed in terms of creating or updating some artifacts, such as a model, a class, a plan. Every activity is assigned to a specific worker.
- Artifacts, the ‘what’
  - An artifact is a piece of information that is produced, modified, or used by a process. Artifacts are the tangible products of the project, the things the project produces or uses while working towards the final product.
- Workflows, the ‘when’
  - A workflow is a sequence of activities that produces a result of observable value. In UML terms, a workflow can be expressed as a sequence diagram, a collaboration diagram, or an activity diagram.



The core process workflows are divided into six core “engineering” workflows:

1. Business modeling workflow
2. Requirements workflow
3. Analysis & Design workflow
4. Implementation workflow
5. Test workflow
6. Deployment workflow

And three core “supporting” workflows:

1. Project Management workflow

2. Configuration and Change Management workflow

3. Environment workflow

   "Although the names of the six core engineering workflows may evoke the sequential phases in a traditional waterfall process, we should keep in mind that the phases of an iterative process are different and that these workflows are revisited again and again throughout the lifecycle."

"Compared to the traditional waterfall process, the iterative process has the following advantages:"

- Risks are mitigated earlier
- Change is more manageable
- Higher level of reuse
- The project team can learn along the way
- Better overall quality

RUP Summary

- The RUP model is built on three fundamental entities: roles, activities and artifacts.
- Disciplines are natural groupigns of process activities, roles and artifacts.
- Workflows relate activities, artifacts and roles in sequences that produce valuable results.
- Guidelines, templates and tooling complement the description of the process.

Agile Manifesto

The Agile Manifesto, written in 2001 by seventeen software developers, presents guidelines for (large-scale) software development based on personal experience. It highlights **four value preferences** which should be present in any software development:

1. **Individuals and Interactions** over processes and tools
2. **Working Software** over comprehensive documentation
3. **Customer Collaboration** over contract negotiation
4. **Responding to Change** over following a plan

On these four values, Scott Ambler remarks:

> **The interesting thing about these value statements is there are something that almost everyone will instantly agree to, yet will rarely adhere to in practice.**
>
> **Everyone will readily agree that the creation of software is the fundamental goal of software development, yet insist on spending months producing documentation describing what the software is and how it is going to be built instead of simply rolling up their sleeves and building it.**

Principles

The 12 principles of Agile, annotated/expanded again by Scott Ambler:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software

The goal of IT is to deliver software! Who do you think would develop a better system: five software developers and with their own tools working together in a single room or five low-skilled “hamburger flippers” with a well-defined process?

2. Welcome changing requirements, even in late development

Traditional software developers will often adopt change management processes which are designed to prevent/reduce scope creep, but when you think about it these are really change prevention processes, not change management processes. Agilists embrace change and instead follow an agile change management approach which treats requirements as a prioritized stack which **is allowed to vary over time**.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time scale

Frequent delivery of working software provides stakeholders with concrete feedback, making the current status of your project transparent while at the same time providing an opportunity for stakeholders to provide improved direction for the development team.

This is the same goal Royce sought

4. Business people and developers must work together daily throughout the project

Your project is in serious trouble if you don't have regular access to your project stakeholders. Agile developers adopt practices such as on-site customer and active stakeholder participation, and adopt inclusive tools and techniques which enable stakeholders to be actively involved with software development.

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done

Agile teams realize that you need build teams from people who are willing to work together collaboratively and learn from each other. They have the humility to respect one another and realize that people are a primary success factor in software development.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation

For a software development team to succeed its members must communicate and collaborate effectively. Face-to-face communication at a shared drawing environment (such as paper or a whiteboard) **is the most effective way** to do so.

7. Working software is the primary measure of progress

The primary measure of software development should be the delivery of working software which meets the changing needs of its stakeholders, not some form of "earned value" measure based on the delivery of documentation of the holding of meetings.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely

My experience is that you can only do high-quality, intellectual work for 5-6 hours a day before burning yourself out. Yes, the rest of the day can be filled up with email, meetings, water cooler discussions, and so on, but people's ability to do "real work" is limited. Yes, you might be able to do high-quality work for 12 hours a day, and do so for a few days straight, but after awhile you become exhausted and all you accomplish is 12 hours of mediocre work a day.

9. Continuous attention to technical excellence and good design enhances agility

It's much easier to understand, maintain, and evolve high-quality source code than it is to work with low-quality code. Therefore, agilists know that they need to start with good code, to keep it good via refactoring, and take a test-driven approach so that they know at all times that their software works. We also adopt and follow development guidelines, in particular coding guidelines and sometimes even modeling guidelines.

10. Simplicity – the art of maximizing the amount of work not done – is essential

Agile developers focus on high value activities, we strive to maximize our stakeholder's return on investment in IT, and we either cut out or automate the drudge work.

11. The best architectures, requirements, and designs emerge from self-organizing teams

This is one of the most radical principles of the agile movement, one which I would love to see researched thoroughly by the academic community. The Agile Model Driven Development (AMDD) and test-driven design (TDD) methods are the primary approaches within the agile community for ensure the emergence of effective architectures, requirements, and designs.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

Software process improvement (SPI) is a continual effort, and techniques such as retrospectives should be adopted to enable you to improve your approach to software development.

Extreme Programming (XP)

Quoting from Kent Beck's Extreme Programming Explained, XP is a style of software development focusing on excellent application of programming techniques, clear communication, and teamwork which allows us to accomplish things we previously could not even imagine. XP is a path of improvement to excellence for people coming together to develop software. It is distinguished from other methodologies by:

- Its **short development cycles**, resulting in **early, concrete, and continuing** feedback.
- Its incremental planning approach, which quickly comes up with an **overall plan that is expected to evolve** through the life of the project.
- Its ability to flexibly schedule the implementation of functionality, responding to **changing business needs**.
- Its reliance on **automated tests** written by programmers, customers, and testers to monitor the progress of development, to allow the system to evolve, and to catch defects early.
- Its **reliance on oral communication, tests, and source code to communicate system structure and intent**.
- Its **reliance on an evolutionary design process that lasts as long as the system lasts**.
- Its reliance on the close collaboration of actively engaged individuals with **ordinary talent**.
- Its reliance on practices that work with both the short-term instincts of the team members and the long-term interests of the project.

DevOps

Development == Operations, it has been since the 90's.

> **Netflix’s ChaosMonkey is an excellent, if extreme, example of a tool to ensure that a complex distributed application can survive outages; ChaosMonkey randomly kills instances and services within the application. The development and operations teams collaborate to ensure that the application is sufficiently robust to withstand constant random (and self-inflicted!) outages without degrading.**

> **As Allspaw points out, it’s important not to divorce developers from the consequences of their work since the fires are frequently set by their code**

> **Although we used to practice “root cause analysis” after failures, we’re recognizing that finding out the single cause is unhelpful. Almost every outage is the result of a “perfect storm” of normal, everyday mishaps.**

Devops 1
Devops (2014) 2

Scrum

Scrum is a framework for developing, delivering, and sustaining complex **products**. This Guide contains the definition of Scrum. This definition consists of Scrum’s roles, events, artifacts, and the rules that bind them together. Ken Schwaber and

Jeff Sutherland developed Scrum; the Scrum Guide is written and provided by them. Together, they stand behind the Scrum Guide.