CSE 70: Software Development Processes
Agile and Plan-Driven Development

Ingolf Krueger

Department of Computer Science & Engineering
University of California, San Diego
La Jolla, CA 92093-0114, USA

California Institute for Telecommunications
and Information Technologies
La Jolla, CA 92093-0405, USA
Learning Goals for Today
• Understand what a development process is good for, and what its main concepts are: WHO, WHAT, HOW, WHEN

• Be able to define a basic development process by ordering given activities and their actors over the timeline

• Know the characteristics of agile and plan-driven processes, and when to use which

• Know the Waterfall Process and be able to identify its challenges

• Know the basics of the Spiral Process and its advantages over the Waterfall

• Know the values and principles of eXtreme Programming (XP)
Customer:  
“I have a problem”
Manager: “Aha! I know exactly what you need!”
Analyst: “That ain’t gonna work – This is what we’ll do”
Programmer: “Hmmm, I think this will be cool.”
Business consultant: “Customer, you’ll get all the bells and whistles!”
Manager: “We need some documentation”
– Programmer: “Here you go!”
Manager: “Let’s deploy it”
Manager: “Show me the money”
Customer: “Arrgh – I need some help over here!”
Customer: “Let me explain one more time!”
How the customer explained it

How the Project Leader understood it

How the Analyst designed it

How the Programmer wrote it

How the Business Consultant described it

How the project was documented

What operations were installed

How the customer was billed

How it was supported

What the customer really needed
Is there a better way?

Can we do better?

Yes – Communication is key!
• Requirements deficiencies are the prime sources of project failures.

• Errors are most frequent during requirements and design activities and are the more expensive the later they are removed.

• Prototyping (significantly) reduces requirement and design errors

(adapted from [ER03])
Development Processes
How to get from Problem Statement to Solution?

Problem
Statement

Requirement

Solution

Product
Development Process!

Requirement → Product
The Four “P”s of Software Engineering

- **People**
  - WHO does it?

- **Project/Activity**
  - HOW is it done?

- **Process**
  - WHEN is it done?

- **Product**
  - WHAT is produced

- **Software Engineering**
  - Contains all four "P"s
Your Turn!
Team up with your neighbor for 5 minutes!

Consider the following activities:
- Code, Analyze, Release, Test, Plan, Design

and the following individuals/roles:
- Manager, UI-Designer, Programmer, Customer

1. Order the activities according to their dependencies.

2. Name who is involved in what activity.
People/Roles

- Customer/Users
- Managers
- Requirements Analysts
- Software Architects
- Information Architects
- Programmers/Developers
- UI-Designers
- Testers
- Deployers
- Maintainers
- Technical support personnel
- Customer support personnel

Often: one person plays multiple roles
Products/Artifacts

- Planning Documents
  - Business Plan
  - Milestones
  - Budget
  - Risk Management Plan
  - ...

- Requirements

- Software Architecture

- Detailed Design

- Documentation
Products/Artifacts

- Planning Documents
  - Business Plan
  - Milestones
  - Budget
  - Risk Management Plan
  - ...

- Requirements

- Software Architecture

- Detailed Design

- Documentation

- Source Code

- Object Code

- Tests

- ...

Project/Activities

- Analyze
- Plan
- Design
- Implement/Code
- Test
- Build

- Debug
- Review/Inspect
- Integrate
- Refactor
- Deploy/Release
- Maintain
Development Process

Defines **what activities** are performed by whom on **which product** at **what time**.

### Plan-Driven vs. Agile

**Plan-Driven**

- **Main Characteristics:**
  - Works for large teams (can scale to 1000s of developers)
  - Contract-centric
  - Focus is on plan/process/artifacts
  - Few, long iterations (~years)
  - Quality through planning for complexity, scale, diversity

**Agile**

- **Main Characteristics:**
  - Works for small teams (up to ~10 developers)
  - Customer-centric
  - Focus is on code/running SW
  - Many short iterations (~1-3 weeks)
  - Quality through simplicity, discipline and courage
Plan-driven Development
Waterfall Process

W. Royce (1970)
Waterfall Process

Note: 55% of all errors are made, but less than 10% are detected during RA

W. Royce (1970)
Waterfall with Iterations

W. Royce (1970)

Analysis -> Design "Product Definition"
Design -> Implementation "Design Specification"
Implementation -> Test & Integration "Code"
Test & Integration -> Maintenance "Validated Code"
Maintenance -> Implementation "Requirements Changes"
Implementation -> Analysis "Code"
Spiral Process

Determine goals, alternatives and context conditions

Analyze alternatives
Determine and resolve risks

Plan next phase

Design and verify next-level product

Spiral Process

Determine goals, alternatives and context conditions

Analyze alternatives
Determine and resolve risks

Plan next phase

Design and verify next-level product

REVIEW

Spiral Process

Determine goals, alternatives and context conditions

Analyze alternatives
Determine and resolve risks

Plan next phase

Design and verify next-level product

Risk analysis
Functioning prototype

Prototypes
1 2 3

REVIEW

Spiral Process

Determine goals, alternatives and context conditions

Analyze alternatives
Determine and resolve risks

Risk analysis
Simulations, Models, Benchmarks
Requirements
Product design
Detailed design

Concepts of Operation
SW Requirements
Product design
Detailed design

Design V&V
Acceptance, integration & component test

Maintenance
Design and verify next-level product

Plan next phase

REVIEW

Prototypes

1
2
3

Spiral Process

Plan-Driven Development Processes

Today is the three-year anniversary of our first meeting to discuss project requirements.

Copyright © 2002 United Feature Syndicate, Inc.
Plan-Driven Development Processes

TODAY IS THE THREE-YEAR ANNIVERSARY OF OUR FIRST MEETING TO DISCUSS PROJECT REQUIREMENTS.

AND WE'RE STILL DISCUSSING REQUIREMENTS. DOES ANYONE ELSE SEE A PROBLEM HERE?

Copyright © 2002 United Feature Syndicate, Inc.
Plan-Driven Development Processes

TODAY IS THE THREE-YEAR ANNIVERSARY OF OUR FIRST MEETING TO DISCUSS PROJECT REQUIREMENTS.

AND WE'RE STILL DISCUSSING REQUIREMENTS. DOES ANYONE ELSE SEE A PROBLEM HERE?

WHEN YOU'RE DONE, CAN WE TALK ABOUT REQUIREMENTS?

Copyright © 2002 United Feature Syndicate, Inc.
Iterative and Agile Development
Evolutionary Development

• Typical for smaller systems/projects
• Increasingly being used also for larger systems
Agile methods

- Rely on team knowledge instead of on documentation
- Iterative: several short cycles
- Incremental: development and delivery in several iterations
- Self organizing: teams decide on the best way to operate
- Emergence: processes, principles and work structure emerge during the project.
Manifesto

Manifesto for Agile Software Development

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.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

© 2001, the above authors this declaration may be freely copied in any form, but only in its entirety through this notice.
The 12 Principles (abbreviated)

1. Highest priority: satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements.

3. Deliver working software frequently

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

5. Build projects around motivated individuals.

6. The most efficient and effective method of conveying information is face-to-face conversation.
The 12 Principles (abbreviated)

7. Working software is the primary measure of progress.
8. Sustainable development!
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.
eXtreme Programming
XP

• 4 values:

  – Communication
    (most projects fail because of poor communication)

  – Simplicity
    (YAGNI = “You Ain’t Gonna Need It” instead of
    BDUF = “Big Design UpFront”)

  – Feedback
    (from customer and other developers)

  – Courage
    (make hard decisions and support other principles and
    practices)
XP Practices (1/2)

• Fine-scale feedback
  – Pair programming
  – Planning game
    (requirements captured on index cards, prioritized)
  – Test-driven development
  – Whole team

• Continuous process
  – Continuous integration
  – Refactoring (design improvement)
  – Small releases (no more than 3 weeks)
XP Practices (1/2)

- **Fine-scale feedback**
  - **Pair programming**
  - **Planning game**
    (requirements captured on index cards, prioritized)
  - **Test-driven development**
  - **Whole team**

- **Continuous process**
  - **Continuous integration**
  - **Refactoring** (design improvement)
  - **Small releases** (no more than 3 weeks)
• Shared understanding
  – Coding standards
  – **Collective ownership** of code
  – Simple design
  – **System Metaphor** (support single vision of success)

• **Programmer welfare**
  – Sustainable pace
• Shared understanding
  – Coding standards
  – **Collective ownership** of code
  – **Simple design**
  – **System Metaphor**
    (support single vision of success)

• **Programmer welfare**
  – Sustainable pace
WE'RE GOING TO TRY SOMETHING CALLED EXTREME PROGRAMMING.
WE'RE GOING TO TRY SOMETHING CALLED EXTREME PROGRAMMING.

FIRST, PICK A PARTNER. THE TWO OF YOU WILL WORK AT ONE COMPUTER FOR FORTY HOURS A WEEK.
WE'RE GOING TO TRY SOMETHING CALLED EXTREME PROGRAMMING.

FIRST, PICK A PARTNER. THE TWO OF YOU WILL WORK AT ONE COMPUTER FOR FORTY HOURS A WEEK.

THE NEW SYSTEM IS A MINUTE OLD AND I ALREADY HATE EVERYONE.
What have you learned today?
• Understand what a development process is good for, and what its main concepts are: WHO, WHAT, HOW, WHEN

• Be able to define a basic development process by ordering given activities and their actors over the timeline

• Know the characteristics of agile and plan-driven processes, and when to use which

• Know the Waterfall Process and be able to identify its challenges

• Know the basics of the Spiral Process and its advantages over the Waterfall

• Know the values and principles of eXtreme Programming (XP)
Development Process

Defines **what activities** are performed by whom on **which product** at **what time**.

**Plan-Driven** vs. **Agile**

**Plan-Driven**

Main Characteristics:
- Works for large teams (can scale to 1000s of developers)
- Contract-centric
- Focus is on plan/process/artifacts
- Few, long iterations (~years)
- Quality through planning for complexity, scale, diversity

**Agile**

Main Characteristics:
- Works for small teams (up to ~10 developers)
- Customer-centric
- Focus is on code/running SW
- Many short iterations (~1-3 weeks)
- Quality through simplicity, discipline and courage
Literature


Rational Unified Process

- Derived from several object-oriented analysis and design methods
- It is possible to tailor it toward a more agile or a more document-centric process (Tailor down approach)
- Risk-driven spiral process
- Fundamental tenets:
  - Reduce size or complexity of what needs to be developed
  - Improve the development process
  - Create more proficient teams
  - Use integrated tools and exploit automation
- Four phases (each consisting of multiple iterations)
  1. Inception
  2. Elaboration
  3. Construction
  4. Transition
# Rational Unified Process

The Rational Unified Process (RUP) is an object-oriented software development process framework. It is designed to be adaptable to the needs of different projects and organizations. The process is divided into four main phases: Inception, Elaboration, Construction, and Transition. Each phase is further divided into activities, which are represented in the following table:

<table>
<thead>
<tr>
<th>Activity</th>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configuration Management</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project Management</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The diagram is adapted from the Rational Unified Process book [RUP98].

---

© Ingolf Krueger, 2010
Rational Unified Process

adapted from [RUP98]
• Individual developer performance varies considerably.

• Development effort is a (non-linear) function of product size.

• Mature processes and personal discipline enhance planning, increase productivity, and reduce errors.

• Adding people-power to a late project makes it later.

• Project risks can be resolved or mitigated by addressing them early.