IT 632: Software Engineering

Software Process Models – RUP|XP|TDD
RUP – Rational Unified Process

- Life Cycle model proposed by Booch, Jacobson, and Rumbaugh ("The three Amigos") derived from the work on UML

- Rational Unified Process (RUP) uses Unified Modeling Language (UML) as core notation

- Described from 3 perspectives
  - A dynamic perspective that shows phases over time;
  - A static perspective that shows process activities;
  - A practice perspective that suggests good practice.

- Unified Process is distinguished by being
  - Use-case driven
  - Architecture-centric
  - Iterative and incremental
RUP – Rational Unified Process

• RUP proposes a phase model that identifies four discrete phases in the software process

  • Inception
    • Establish the business case for the system
    • Decide to cancel or continue the project

  • Elaboration
    • Develop an understanding of the problem domain and the system architecture.

  • Construction
    • System design, programming and testing.

  • Transition
    • Deploy the system in its operating environment.
Iterative Phase Model

- Each phase may be enacted in an iterative way with the results developed as increments.

- The whole set of phases may also be enacted incrementally. Whole set = cycle (later on..)

- An iteration represents a set of activities for which there is a milestone (“well-defined intermediate event”).

- The scope and results of the iteration are captured via discrete work products called artifacts.
Artifact sets

• Each artifact set has a different intention and uses different notations to capture the relevant artifacts.
• Management Set:
  • Notation: Ad hoc text, graphics, textual use cases
  • Goal: Capture plans, processes, objectives, acceptance criteria.
• Requirements set:
  • Notation: Structured text, models in UML (Use Case, Class, Sequence)
  • Goal: Capture the problem in the language of the problem domain
• Design set:
  • Notation: Structured text, models in UML
  • Goal: Capture the engineering blueprints
• Implementation set:
  • Notation: Programming language
  • Goal: Capture the building blocks of the solution domain in human-readable format.
• Deployment set:
  • Form: Machine language
  • Goal: Capture the solution in machine-readable format.
Iterative Development
Business value is delivered incrementally in time-boxed cross-discipline iterations.

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>E1</td>
<td>C1</td>
<td>C2</td>
</tr>
<tr>
<td>I1</td>
<td>E2</td>
<td>C1</td>
<td>C2</td>
</tr>
<tr>
<td>I1</td>
<td></td>
<td>C1</td>
<td>C2</td>
</tr>
<tr>
<td>I1</td>
<td></td>
<td>C1</td>
<td>C2</td>
</tr>
<tr>
<td>I1</td>
<td></td>
<td>C1</td>
<td>C2</td>
</tr>
<tr>
<td>I1</td>
<td></td>
<td>C1</td>
<td>C2</td>
</tr>
<tr>
<td>I1</td>
<td></td>
<td>C1</td>
<td>C2</td>
</tr>
<tr>
<td>I1</td>
<td></td>
<td>C1</td>
<td>C2</td>
</tr>
</tbody>
</table>

Business Modeling
Requirements
Analysis & Design
Implementation
Test
Deployment
Life of a Unified Process

• Unified Process repeats over a series of cycles each concluding with a product release (increment) to the users

• Cycles have no specific name but characterize the stage of maturity of the software system (like “birth” → “death”)

• Each cycle has four phases (each with a number of iterations)
  • Inception, Elaboration, Construction & Transition
  • Phases have goals (→ result in artifacts or models)

• Delivered products will be described by related models each with “trace” dependencies which chain backwards and forwards
  1. Use Case Model
  2. Analysis Model
  3. Design Model
  4. Deployment Model
  5. Implementation Model
  6. Test Model
Life of a Unified Process

- Unified Process repeats over a series of cycles each concluding with a product release (increment) to the users.

- Cycles have no specific name but characterize the stage of maturity of the software system (like “birth” → “death”).

- Each cycle has four phases (each with a number of iterations):
  - Inception, Elaboration, Construction & Transition
  - Phases have goals (→ result in artifacts or models)

- Delivered products will be described by related models each with “trace” dependencies which chain backwards and forwards:
  1. Use Case Model
  2. Analysis Model
  3. Design Model
  4. Deployment Model
  5. Implementation Model
  6. Test Model
Life of a Unified Process

A cycle with its phases and its iterations

Birth → Cycles concluded with a release → Death

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1</td>
<td>i2</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>in-1</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>in</td>
</tr>
</tbody>
</table>

i = iteration

A cycle with its phases and its iterations

Releases
RUP - Summary

- The RUP is not a suitable process for all types of development but it does represent a new generation of generic processes.

- Most important innovation:
  - Combination of many views
  - Deployment of software is part of the process (almost ignored in other process models)

- Based on standards
  - Object-oriented Modeling
  - Unified Modeling Language
XP is a lightweight methodology for small to medium sized teams developing software in the face of vague or rapidly changing requirements.

Kent Beck
Agile

WHAT IS XP?

Lightweight
Humanistic

Discipline
Software development
DEVELOPING IS LIKE DRIVING
MENTALITY OF SUFFICIENCY

How would you program if you had all the time in the world?

- Write tests
- Restructure often
- Talk with fellow programmers and with the customer often
Extreme Programming consists of:

- Pair Programming
- Code Review
- Unit Testing
- Integration Testing
- Planning Game
- Courage
- Respect

The process cycle includes:

1. Planning
2. Design
3. Coding
4. Testing
5. Software Increment
6. Release

Key practices:

- Simple designs
- User stories, acceptance test criteria, iteration plan
- Spike solutions prototypes
- Unit test
- Continuous integration
- Pair programming
- Acceptance testing
TDD – Test Driven Development

1. Add a Test
2. Run the Test
   - Pass
   - Fail
3. Make a little change
4. Run the Test
   - Fail
   - Pass
5. Refactor

Flowchart:
- Add a Test → Run the Test
- Run the Test → Make a little change
- Make a little change → Run the Test
- Run the Test → Refactor
- Refactor → Add a Test
Test-driven development is not about testing.

Test-driven development is about development (and design), specifically improving the quality and design of code.

**Cycle:**
- Write the test
- Run the test (there is no implementation code, test does not pass)
- Write just enough implementation code to make the test pass
- Run all tests (tests pass)
- Refactor
- Repeat
Ensures quality

Keeps code clear, simple and testable

Provides documentation for different team members

Repeatable tests

Enable rapid change
Questions???