Unified Process

Peter Dolog
dolog [at] cs [dot] aau [dot] dk
5.2.03
Intelligent Web and Information Systems
April 7, 2009
Outline

History
Principles
Artifacts
Practices
Roles
UP

Unified Process, 1990’s
Iterative, not agile
Risk-driven development in early iterations focusing on creation of the core architecture and driving down the high risks
2-6 weeks iterations
History of UP

Some of the roots in “spiral model” of Barry Boehm
Core initial development around 1995-98
Large Canadian Air Traffic Control project as test bed
Phillippe Kruchten chief architect of UP/RUP
Rational Corporation had commercial product in mind (RUP) but also reached out to public domain (UP)
Unified Process (UP)

Popular iterative process framework, especially its refinement:

- Rational Unified Process (RUP)

Key practices and guidelines:

- Short time-boxed iterations
- Develop high-risk elements in early iterations
- Deliver value to customer
- Accommodate change early in project
- Work as one team
Unified Process

An iteration includes all workflows.
Originally developed by Rational

An iterative process framework

Project lifecycle phases

Identifies workers, activities, artifacts

Promotes certain practices:

- Develop software iteratively
- Manage requirements
- Use component-based architectures
- Visually model software
- Verify software quality
- Control changes to software Analyst

Key Ideas and Practices
Classification of UP

Average projects: iteration length of 2-6 weeks
Very flexible in degree of ceremony, over 50 optional work products usable to increase ceremony if needed
Yet encourages light touch
Characteristics of UP

Iterative process framework, typically customized to be a process description for the organization.
All work products ("artifacts") are optional and their order arbitrary. Work products serve as common vocabulary for the team.
RUP is a process framework and licensed product (tool plus web pages).
Artifacts are information abstractions, e.g. Vision or Risk List, organized in disciplines, e.g. Requirements Discipline.
Disciplines within iterations

Example disciplines: Requirements, Design, Project Management, Implementation

Development Case of UP: UP tailored for each project, choose sets of practices and work products to create (“less is better”)

Disciplines addressed in each iteration but to varying degree
Life cycle in four phases

Inception
Business case, vision, identify high risks & 10% of key reqs in detail, estimate elaboration effort

Elaboration
Core & architecturally significant parts coded/tested, key risks identified/mitigated, 80% of major reqs evolved/defined

Construction
Builds remaining system in short iterations, efficient and predictable due to solid elaboration

Transition
Exposes release candidate for review/feedback, then deployment
Some prominent work products

**Vision**: summary of objectives, features, business case

**Software Architecture Document**: Short learning aid to understand the system

**Test Plan**: summary of goals and methods of testing

**Iteration Plan**: detailed plan for the next iteration

**Change Request**: uniform way to track all requests for work, e.g. defects
Example roles in UP

**Stakeholder**: customer, product manager, etc

**Software Architect**: establishes and maintains architectural vision

**Process Engineer**: leads definition and refinement of Development Case

**Graphic Artist**: assists in user interface design, etc
Some UP Guidelines

- Attack risks early and continuously before they will attack you
- Stay focused on developing executable software in early iterations
- Prefer component-oriented architectures and reuse of existing components
- Baseline an executable architecture early
Six Best “must” Practices

Time-boxed iterations

Avoid attempting large, up-front requirements

Strive for cohesive architecture and reuse existing components

On large projects: reqs & core architecture developed by small co-located team; then early team members divide into sub-project leaders

Continuously verify quality

Test early, often, and realistically by integrating all software each iteration
Six Best “must” Practices (2)

**Visual modeling**

Prior to programming, do at least some visual modeling to explore creative design ideas.

**Manage requirements**

Find, organize, and track requirements iteratively through skillful means. Use tools.

**Manage change**

Disciplined configuration management and version control, change request protocol, base-lined releases at the end of each iteration.
How to fail with UP

elaboration phase goal is to create a throwaway prototype

- prototypes are acceptable in UP, e.g. during inception, but elaboration goal is creation of subset of final system

iterations too long

- iterations should typically be 2-6 weeks long not months

team should do lots of modelling and UML diagrams, and use a CASE tool

- UP contains optional models with potential use of UML but UP also compatible with agile approach, e.g. whiteboard hand sketches etc
How to fail with UP (2)

Not conforming to official UP work product or phase names

- common vocabulary vital within organization and across global UP-conforming teams

Development Case too complex, too many work products

- “less is better,” UP recommends adding work products that really add value

Software Architecture Document “finished” before end of elaboration

- UP SAD is learning aid, so this would imply “up-front design”
Signs that your “UP” expert is not worth her money

describes UP phases similar to waterfall phases

suggests iteration lengths > 6 weeks

recommends inception phase several weeks long

does not stress importance of early programming

near the start, defines “believable plan” specifying number of iterations, their duration, etc

encourages more and more work product creation
UP in “The Real World”

Large: Canadian Air Traffic Control System

Ten years, Ada and C++, test bed for practices RUP, previous failed waterfall attempt 11 years & $2.6 billion USD

Medium: Ogre Nextgen Economic Modeling System

2 years, Java technologies, decision support system for oil/gas asset holders

Small: QUICKcheck point-of-sale, 1 year, six people, Java technologies, self-checkout system for grocery stores (main developer: Kyrus)
Is RUP agile?

RUP can be used in a very traditional waterfall style or in an agile manner.

“You can use RUP as a agile process, or as a heavyweight process - it all depends on how you tailor it in your environment. “ – Martin Fowler

Craig Larman is a strong proponent of using the RUP in an agile manner.
Rational Unified Process

Wide spread methodology championed by Rational Corporation
Combines water-fall and evolutionary development
Plan a little, design a little, code a little.
Aims to minimizes risk of failure
Breaks system into mini-projects, focusing on riskier elements first

Other (claimed) advantages
- Encourages all participants, including testers, integrators, and documenters to be involved earlier on
- Mini-waterfalls centered around UML, a particular OO-methodology
- CASE-TOOL support (of course, from Rational)

Does it work?
- Many positive case studies although benefits difficult to quantify
Rational Unified Process (RUP)

Philippe Kruchten
Ivar Jacobson
Grady Booch
James Rumbaugh

ISBN: 0-201-70710-1
RUP - overview

RUP - philosophy
- Iterations / Increments
- Use case driven
- Architecture centered
- Visual (UML)
- Configurable process
- Risk driven

RUP - key elements
- Phases
- Iterations
- Workflows
- Activities
- Roles
- Artifacts

Contents of a workflow
- Artifacts
- Roles
- Activities

Risk & iterations
- Architecture & system
- Domain knowledge
Rational Unified Process

(a form of controlled iteration)

Process Workflows

- Business Modeling
- Requirements
- Analysis & Design
- Implementation
- Test
- Deployment

Supporting Workflows

- Configuration Mgmt Management Environment

Phases

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
</table>

Iterations within phases

- Preliminary Iteration(s)
- Iter. #1
- Iter. #2
- Iter. #n
- Iter. #n+1
- Iter. #n+2
- Iter. #m
- Iter. #m+1
Larman’s Design Process
### Cockburn scale

**Criticality**

<table>
<thead>
<tr>
<th>Category</th>
<th>L6</th>
<th>L20</th>
<th>L40</th>
<th>L100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Life (L)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Essential Money (E)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Discretionary Money (D)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comfort (C)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**People**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>E6</td>
<td>E20</td>
<td>E40</td>
<td>E100</td>
<td></td>
</tr>
<tr>
<td>D6</td>
<td>D20</td>
<td>D40</td>
<td>D100</td>
<td></td>
</tr>
<tr>
<td>C6</td>
<td>C20</td>
<td>C40</td>
<td>C100</td>
<td></td>
</tr>
</tbody>
</table>
Degree of Ceremony and Cycles

- **Waterfall strict (no iteration)**
  - Formal Steps, many doc.
  - Many short iterations

- **SCRUM**
  - Few Steps, few, doc
  - ceremony

- **XP**
  - Few Steps, few, doc
  - ceremony
Key points

Guidance with more (optional) artifacts
Roles
Iterations with all disciplines used differently in different stage of the project
Architecture Centric
Use case driven