



Making RUP Agile

CHOOSE SIG Beer

February 12, 2004

17:00-18:00

Berne, Switzerland

Michael Hirsch
Zühlke Engineering AG
hm@zuhlke.com

Objectives

In this Talk, you will...

- n Get a very brief overview of RUP**
- n Hopefully get an idea how to use RUP for agile development**
- n See some important artifacts from a real world agile RUP project**

Background

About Zühlke Engineering AG

- n **Independent systems- and software development and consulting company**
- n **Founded 1968, 240 employees today**
- n **Offices in Zurich, Basle, Frankfurt, and London**
- n **Customers: European industry and financial services companies**

About myself

- n **Active in software development since 1981 in various roles, from developer to project manager**
- n **Grown up with Unix / C, switched to OO development in 1991**
- n **Experience with RUP since 1998**
- n **Currently project and process manager at Zühlke AG**

Contents

1. Introduction

2. A brief overview of RUP

3. Configuring RUP for agile development

4. Demo of a real world agile RUP project

- Summary

6. Q & A

What does "Agile Development" mean ?

Official definition from the Agile Manifesto (www.agilemanifesto.org):

"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 items on the left more."

What is RUP ?



RUP at a glance:

- n RUP is a **framework** to tailor development processes
- n RUP is a huge software engineering **knowledge base**
- n RUP mandates **iterative and incremental** development
- n RUP covers almost all activities of software development
- n RUP is based on OO and UML
- n Requirements are expressed as **Use Cases** and **Features**
- n **Software architecture** is of central importance
- n Project management is **risk oriented**
- n RUP is a **product** of Rational (now IBM)
- n RUP is very well documented

RUP is NOT....



- n **A CASE tool**
- n **A project management tool**
- n **A substitute for first class developers and project managers**
- n **A handicap because of too much administration and documentation activities**
- n **A cure for all possible problems of software development**
- n **A tool lock-in of IBM / Rational**

A Brief History of RUP

198x	Objectory process from Ivar Jacobson Developed and marketed by Objectory AB in Sweden
1992	Simplified version of Objectory published in Ivar Jacobson's book <i>Object Oriented Software Engineering</i>
Oct. 1995	Rational buys Objectory AB and with it Objectory V3.8
Oct. 1996	Rational Objectory Process (ROP) Version 4.0 Adds iterative development and software architecture as first class citizens
Sept. 1997	Rational Objectory Process (ROP) Version 4.1 UML replaces Objectory notation
Nov. 1998	Rational Unified Process (RUP) Version 5.0 Adds business modeling, extensions in project management
1999-2002	Many new versions (5.1, 2000, 2001, 2002)
June 2003	RUP 2003 (current version)

The Agile Methods Landscape circa 2004

- n **Extreme Programming (1999)** – Kent Beck et. al.
- n **Crystal (2001)** – Alistair Coburn
- n **Scrum (2001)** – Ken Schwaber
- n **Lean Software Development (2003)** – Mary Poppendieck

How does RUP compare to XP ?

	RUP	XP
Foundations	6 best practices	12 XP practices
What is it ?	Process framework	Process
Team size	2 or more, no upper limit	up to 15
Process model	Iterative and incremental	Iterative and incremental
Documentation	HTML based online documentation, with guidelines and examples	Books
Motto	All activities must be balanced in a successful project	Code is at the center of the software universe

Overall, there are more similarities than differences !

Contents

1. Introduction

2. A brief overview of RUP

3. Configuring RUP for agile development

4. Demo of a real world agile RUP project

- Summary

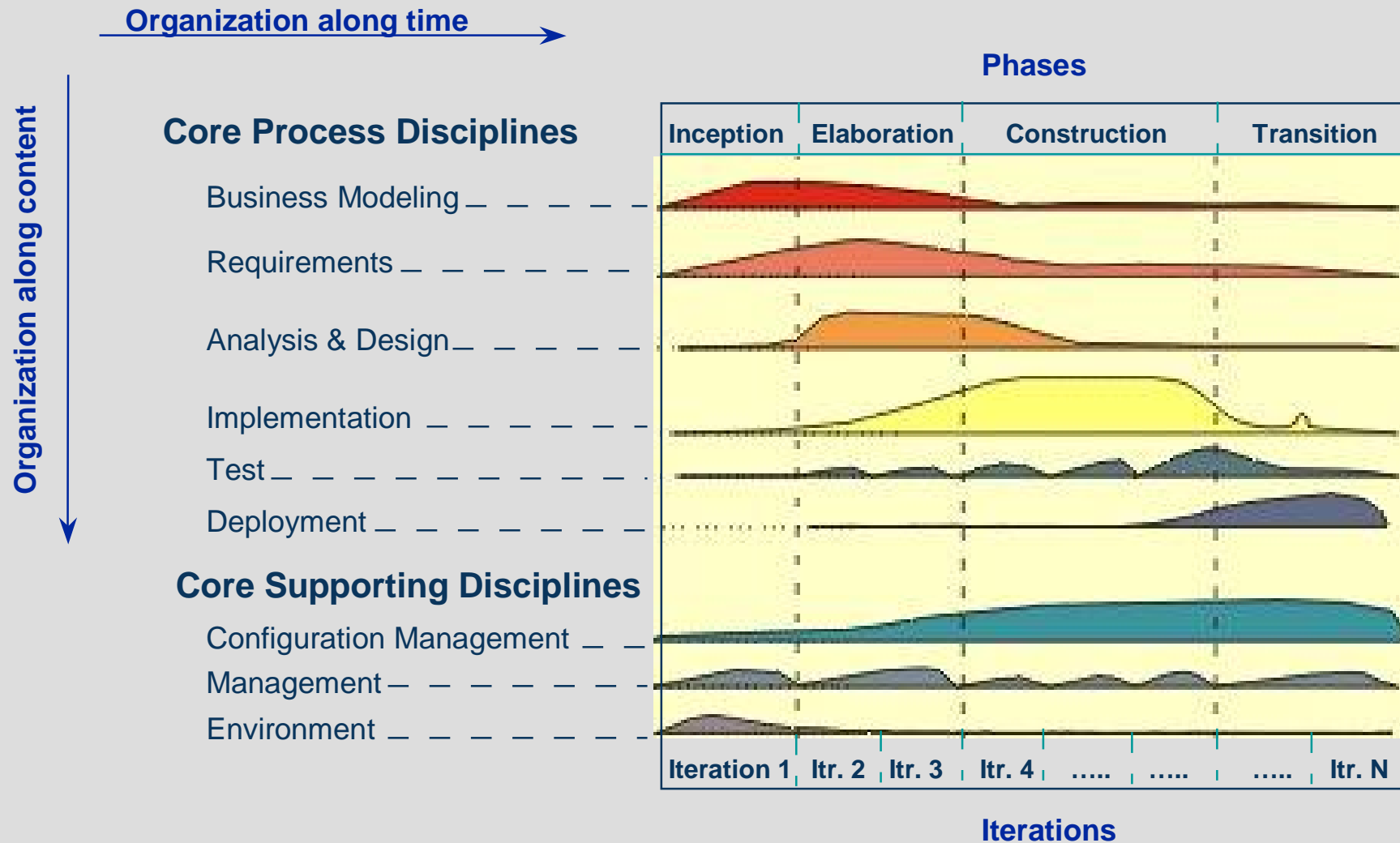
6. Q & A

RUP is based on six Best Practices

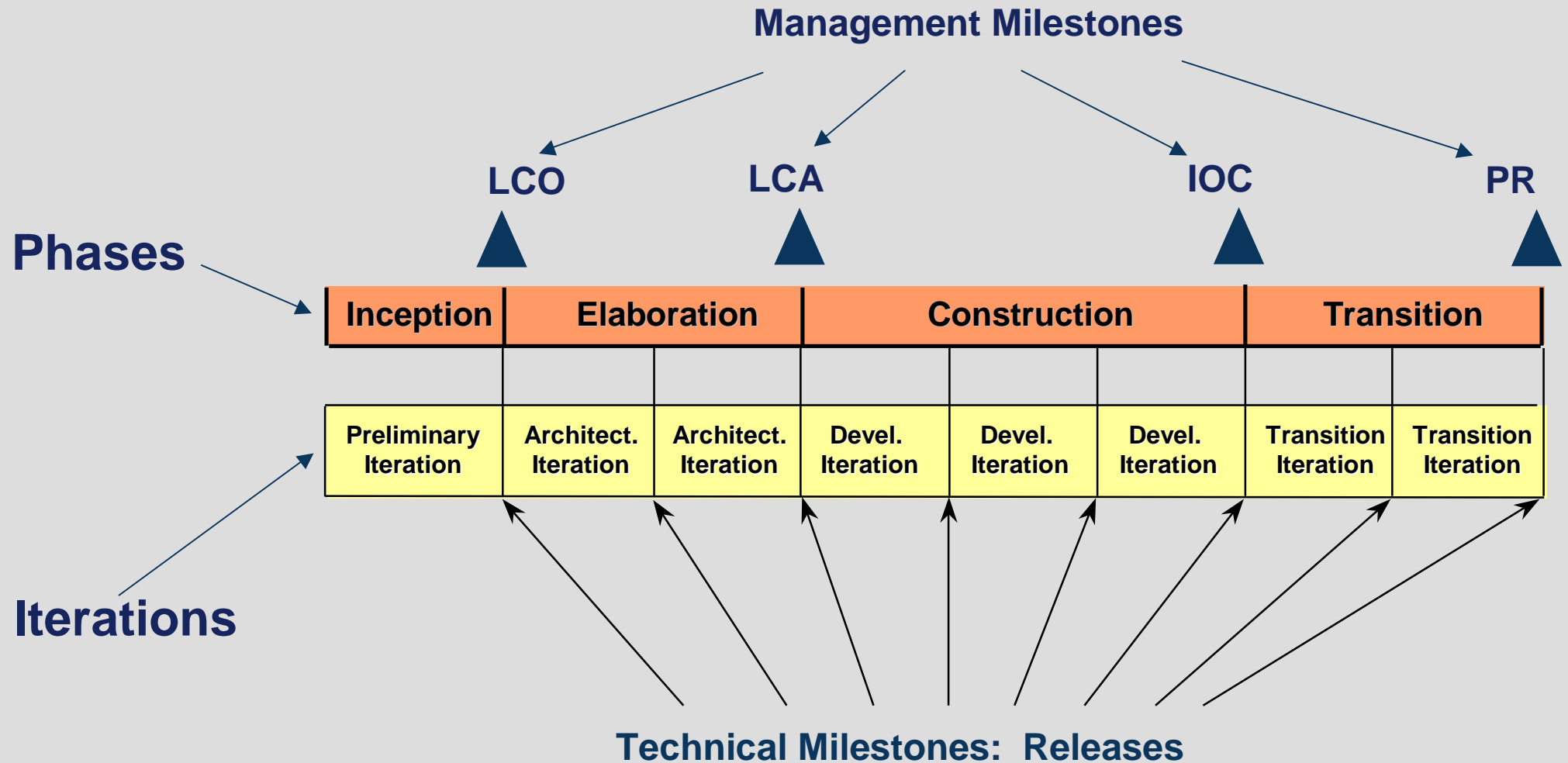
1. **Develop iteratively**
2. **Manage requirements**
3. **Use component architectures**
4. **Model visually** (with the UML)
5. **Continuously verify quality**
(In plain english: test a lot)
6. **Manage change**



RUP Overview



Organization along Time Axis



Management Milestones:

LCO = Lifecycle Objectives LCA = Lifecycle Architecture IOC = Initial Operational Capabilities PR = Product Release

Organization along Content Axis

Discipline = A set of coherent activities and artifacts

Discipline	Topics / Key Concepts
Business Modeling	Business Use Cases, Business Object Model
Requirements Engineering	Use Cases, Features, Requirements Attributes
Analysis and Design	Object Modeling, UML
Implementation	OOP, Unit Tests
Test	System- and Integration Testing
Deployment	Productizing, Deployment into Production Env.
Project Management	Iterative / Incremental Development, Risk Mgmt.
Configuration and Change Mgmt.	Release and Change Management
Environment	Process Configuration, Development Environm.

Most Important Process Elements

Role:

A role that may be played by an individual or a team in the development organization

Activity:

A unit of work a worker may be asked to perform

Role

**Use-Case
Specifier**

Activity

Describe a
Use Case

responsible for

Use Case

**Use-Case
Package**

Artifact

Artifact:

A piece of information that is produced, modified, or used by a process

Additional Process Elements

These are the "knowledge base" parts of RUP



- n **Guidelines**
Practical advice on how to carry out activities and "measures of goodness" for artifacts
- n **Concepts**
The rationale and theory behind practices recommended by RUP
- n **Templates for artifacts**
- n **Workflow diagrams**
For each discipline, define the order of activities
- n **Tool mentors**
Tool specific guidelines, mostly for Rational tools
- n **Plug-Ins**
Configurable process elements ("include / don't include")

Some RUP Statistics

RUP defines:

- n 40 roles
- n 80 major artifacts
- n 150 activities

Can this be agile ?

**Yes it can, if you know what to choose
(remember, RUP is a process *framework*)**

Contents

1. Introduction
2. A brief overview of RUP
- 3. Configuring RUP for agile development**
4. Demo of a real world agile RUP project
 - Summary
6. Q & A

Principles for Agile Configurations of RUP

- n **Start with artifacts: select a small set of critical artifacts (activities will follow from artifacts)**
- n **Obviously, select only artifacts which create value for the project**
- n **If in doubt, skip an artifact**
- n **Treat activities as part of the knowledge base, but don't use them for detailed iteration planning**
- n **Treat roles as a checklist: do we have all skills we need ?**
- n **Document the choice of artifacts in a "development case"**

Assumptions

The recommendations in the rest of this section are for agile projects with the following characteristics:

- n Team size of 2 to 10 developers**
- n Project duration of 3 to 18 months**
- n Mix of unknown / unclear and well understood requirements**
- n Frequent requirements changes**
- n Developed software is of non life-critical nature**
- n Time to market is important**
- n And so is staying within a tight budget**

Project Management

Mandatory Artifacts



Risk List

The top 10 risks, what to do to prevent them, and what to do if they still materialize



Software Development Plan

Breakdown of the project into phases and iterations



Iteration Plans

Lists Features / use cases / change request / bug fixes to be implemented within a single iteration



Iteration Assessments

Actual versus planned results of a single iteration, and lessons learned

As of RUP 2003, there are 11 more project management artifacts

Recommendations for Long Term Planning

- n **Maintain a coarse grained plan with the sequence of planned iterations (the "Software Development Plan")**
- n **For each planned iteration, define:**
 - **Start / end dates**
 - **Major objectives of the iteration**
 - **Dependencies, if any**
 - **Planned customer release(s)**
- n **Sample iteration objectives:**
 - **“Implement first version of fee calculation engine”**
 - **“Finalize interface to stock quote server”**
- n **Schedule work on high risk parts of the system as early as possible**

- n Use iterations of 4 +/- 2 weeks**
- n Small projects = shorter iterations,
Large projects = longer iterations**
- n Typical number of iterations:**
 - Inception: 1 to 2**
 - Elaboration: 2 to 4**
 - Construction: 2 to 5**
 - Transition: 1 to 3**
- n Update the Software Development Plan after completion of every iteration**

- n **Have a brief but very clear plan for each iteration**
- n **Prepare iteration plans a few days before the iteration starts**
- n **A good iteration plan specifies **measurable results rather than activities****
- n **Sample result types to plan for:**
 - **Features implemented and tested**
 - **Infrastructure components implemented and tested**
 - **Change requests implemented and tested**
 - **Bug fixes implemented and tested**
- n **Have one person responsible for each result**
- n **Customer must be involved in iteration planning**

Recommendations for Managing an Iteration

- n **Track progress with weekly estimates of *remaining* work to do**
- n **Estimates must be done by the responsible developer(s)**
- n **When you realize that you can't achieve all goals for an iteration: drop goals, *don't* extend the schedule for more than 2 to 3 days**
- n **Try to keep the plan for the running iteration stable (analogy: an aircraft can't be changed while in flight)**
- n **One “official” 1 - 2 hour status meeting per week is enough**
- n **Informal meetings are held ad hoc and as needed**

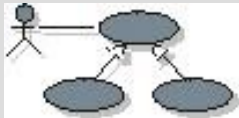
Requirements

Mandatory Artifacts



Vision Document

Requirements at the highest level of abstraction. Basically a list of features. Provides an overview of the “What” of the project.



Use Case Model

Brief description of actors and use cases (typical ways of using the system or product).

**As of RUP 2003, there are 7 more requirements artifacts.
Of these, the following are potentially useful for agile projects:**

- **Supplementary Specifications:** descriptions of non functional requirements
- **Stakeholder Requests:** the “wish list”
- **Use Case Storyboard:** the UI aspects of use cases
- **Glossary**

Recommendations for Requirements Management

- n **At a minimum, maintain a list of required features to define the scope of the system**
- n **Have one person on the team perform the role of an “requirements engineer”**
- n **How much upfront requirements definition is really needed ?**
 - **End of Inception:**
 - **List of features identified (first version)**
 - **List of important use cases identified (first version)**
 - **End of Elaboration:**
 - **Ideally: 80% of features and use cases briefly described**
 - **Workable: features and use cases for the next iteration briefly described**

Analysis and Design

Mandatory Artifacts



Software Architecture Document

Brief description of the high level design (architecture) of the system.
Provides an overview of the “How” of the project.



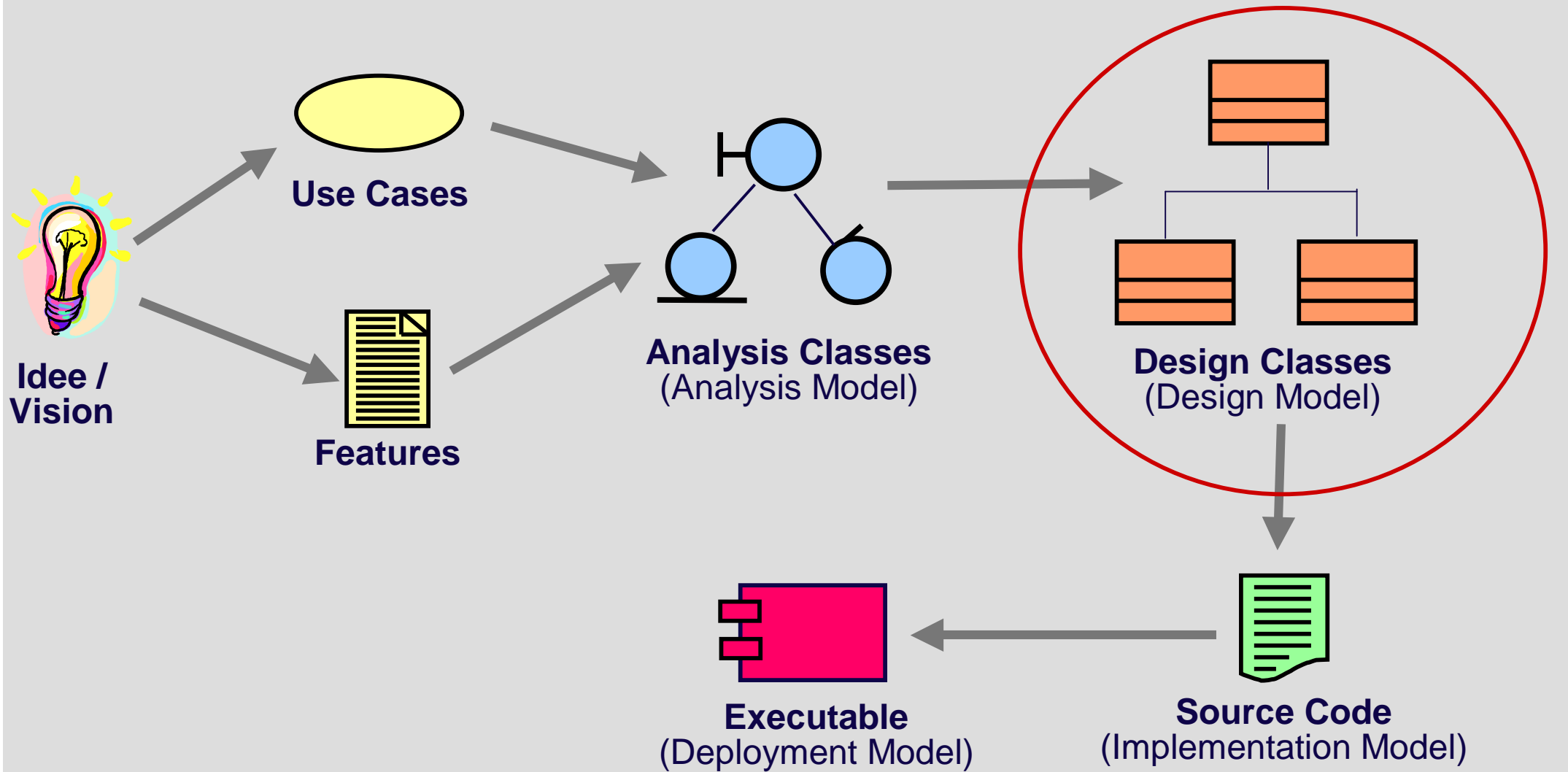
Design Model

An abstraction of the design of the system, documented with UML diagrams.

As of RUP 2003, there are about 8 more analysis and design artifacts

Analysis and Design

From the Idea to the Executable System



Recommendations for Analysis and Design

- n **Explicitly assign the role of a “software architect”**
- n **Good strategies for the design model:**
 - **The bare minimum approach: A few UML class and sequence diagrams in the Software Architecture Document to describe the important mechanisms of the system, created with any drawing tool (e.g. Visio, OmniGraffle, etc.)**
 - **The recommended approach: Use a CASE tool which keeps design and code in sync automatically (e.g. Rational XDE, Borland Together)**
- n **Using a CASE tool which doesn't keep design and code in sync and therefore requires frequent “reverse engineering” is not a good idea (at least not for agile projects)**

Test

Mandatory Artifacts



Test Plan

Brief description of the testing strategy adopted by the project



Test Case

One specific test, with input conditions and expected results.
For system tests, typically derived from use cases.



Defect List

A list of known defects

**As of RUP 2003, there are 12 more test artifacts.
Of these, the following are potentially useful for agile projects:**

- Test Evaluation Summary
- Test Data

Test

Recommendations for Test

- n **Have one person responsible for the overall testing effort (“test manager”)**
- n **A practical minimal testing strategy:**
 - **Unit tests for all critical (complex) classes**
 - **Manual system tests for testing of functionality at the user interface level**
 - **Even better: automated system tests**
- n **Secret tip: CS students are great (and affordable) resources for designing and executing tests**

Summary of Agile RUP Configuration: Artifacts

Project Management

- n Risk List
- n Software Development Plan
- n Iteration Plans
- n Iteration Assessments

Requirements

- n Vision Document
- n Use Case Model

Analysis and Design

- n Software Architecture Document
- n Design Model

Implementation

- n Implementation Model

Test

- n Test Plan
- n Test Cases
- n Defect List

Deployment

- § Release Notes
- § Deployment Unit

Configuration and Change Mgmt

- n Configuration Mgmt Plan
- n Change Request
- n Project Repository

Environment

- n Development Case
- n Programming Guidelines

The 19 artifacts of an agile RUP configuration

Contents

1. Introduction
2. A brief overview of RUP
3. Configuring RUP for agile development
- 4. Demo of a real world agile RUP project**
 - Summary
6. Q & A



Project Mission:

**Build a workflow
system for designing
steam turbines**

Customer:

ALSTOM Power, Switzerland

Some Project Statistics

Team size:	3 to 6
Number of use cases:	11
Number of features:	30
Number of iterations:	8
Project duration:	8 months
Project effort:	250 person days
Number of bugs found and fixed:	approx. 15
Number of change requests implemented:	approx. 20
Number of Java classes implemented:	188
Total LOC, excluding blank and comment lines:	17'700

Technology used: Java J2SE 1.3, Swing, JDBC, Eclipse, JUnit, Together/J, Postgres, Oracle, CVS

Project Artifacts

Project Management

- n **Risk List** (Word, 5 pages, 13 risks)
- n **Software Development Plan** (Word, 10 pages)
- n **Iteration Plans** (Word, 5 pages, plus Excel sheet)
- n **Iteration Assessments** (Word, 5 pgs)

Requirements

- n **Vision Document** (Word, 31 pages)
- n **Use Case Model** (Word, 12 pages)
- n **Glossary** (Word, 3 pages)

Analysis and Design

- n **Software Architecture Document** (Word, 14 pages)
- n **Design Model** (Together/J)

Implementation

- n **Implementation Model** (Java, 188 classes, 17'700 LOC)

Test

- n **Test Plan** (Word, 8 pages)
- n **Test Cases** (Excel, 31 test cases)
- n **Defect List** (Bugzilla)

Deployment

- § **Release Notes** (HTML, 3 to 5 pages)
- § **Deployment Unit** (ZIP file)

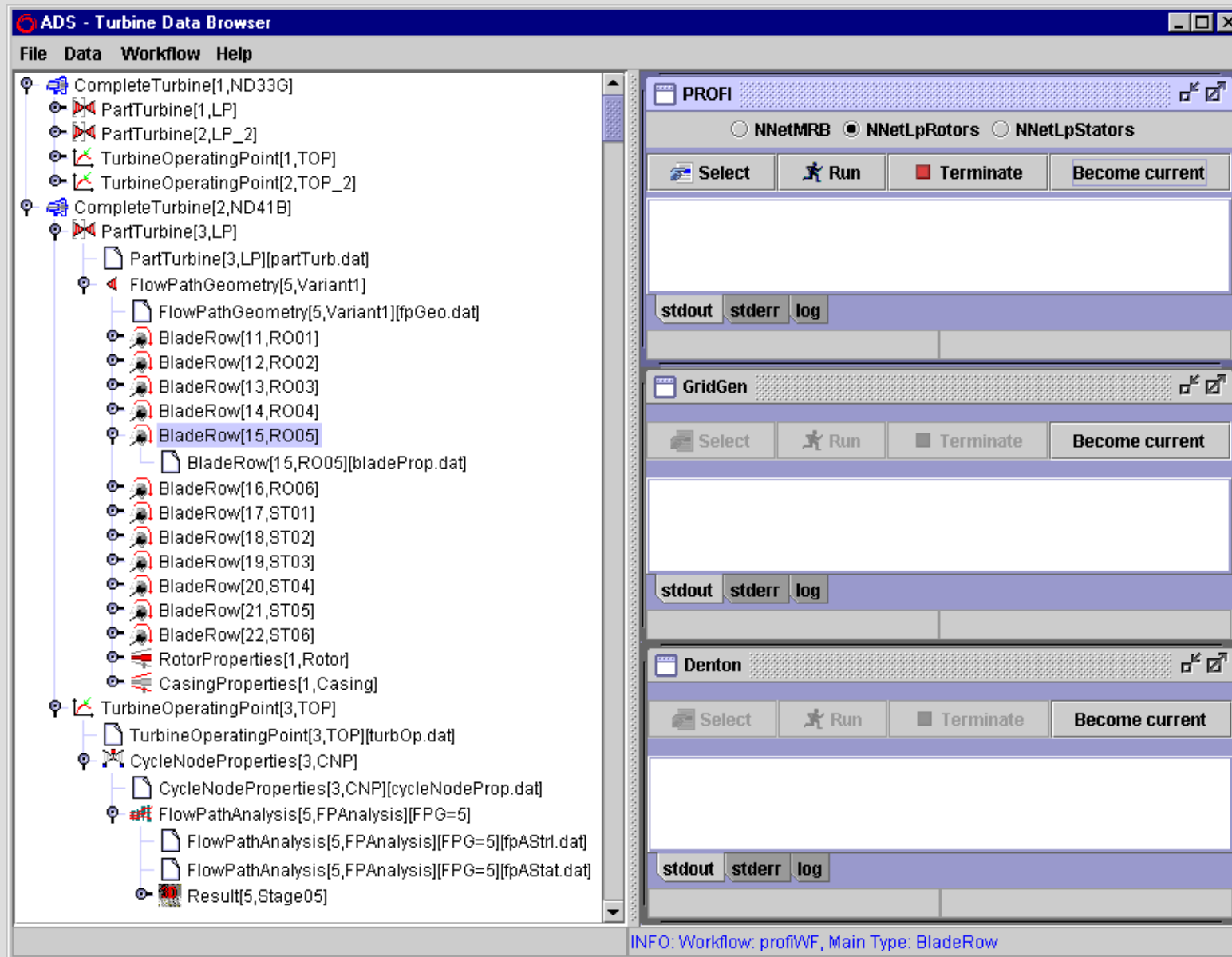
Configuration and Change Mgmt

- n **Configuration Mgmt Plan** (Word, 8 pages, reused from previous project)
- n **Change Request** (Bugzilla)
- n **Project Repository** (CVS)

Environment

- n **Development Case** (Word, 6 pages)
- n **Programming Guidelines** (Reused from previous project, 20 pages)

Now the Demo....



Contents

1. Introduction
2. A brief overview of RUP
3. Configuring RUP for agile development
4. Demo of a real world agile RUP project
- **Summary**
6. Q & A

What to expect from Agile RUP

Project	Technology	Total Effort [PD]	Total LOC	Total Classes	Total Use Cases	FPS delivered	Productivity [LOC/PY] [FP/PY]	
Airport MIS	Java J2EE, Swing	327	15'000	202	12	430	9'175	260
ATM Software	Win2k native, C++	2'940	128'000	2'180	16	2'415	8'705	165
Workflow Engine	Java J2SE, Swing	252	17'700	188	11	505	14'045	400
Product Data Mgmt System	Java J2EE, Web client	461	40'000	470	12	1'140	17'315	495
Protocol Converter	Java J2SE	215	26'400	507	9	725	24'560	675
Quality Control System (Prototype)	.Net, C#	85	14'300	119	9	410	33'650	965

PD = Person Days
LOC = Lines of Code
FP = Function Points

PY = Person Year (200 working days of 8 hrs)

Key Factors to Make RUP Agile

Individuals and interactions over **processes and tools**

- Use a lightweight process configuration with few artifacts

Working software over **comprehensive documentation**

- Make sure every iteration delivers an executable release

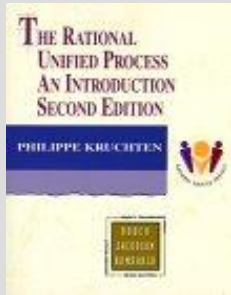
Customer collaboration over **contract negotiation**

- Involve the customer in iteration planning and make sure the customer delivers feedback for every release

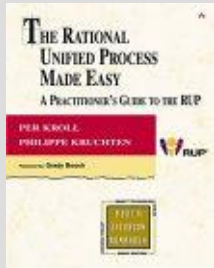
Responding to change over **following a plan**

- Plan iterations “just in time”, based on risk and current requirements

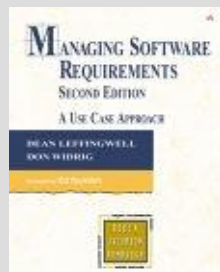
References



The Rational Unified Process
Philippe Kruchten
Addison-Wesley, 2000



The Rational Unified Process Made Easy
Peer Kroll, Philippe Kruchten
Addison-Wesley, 2003

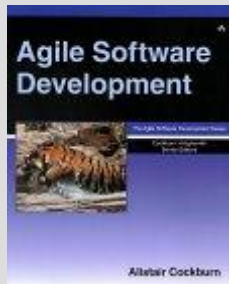


Managing Software Requirement
Dean Leffingwell, Don Widrig
Addison-Wesley, 2003

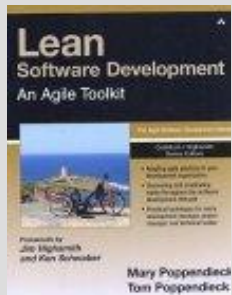
References



Extreme Programming Explained
Kent Beck
Addison-Wesley, 1999



Agile Software Development
Alastair Cockburn
Addison-Wesley, 2001



Lean Software Development
Mary Poppendieck, Tom Poppendieck
Addison-Wesley, 2003

References

On the Web:

- n **Agile Alliance**
www.agilealliance.org
- n **Rational Unified Process**
www.ibm.com, then search for "RUP"
- n **Extreme Programming**
www.extremeprogramming.org

Running Agile Software Development Projects with RUP

Thank you for your attention !

Questions ?

hm@zuehlke.com