Software Processes in Practice
The Rational Unified Process

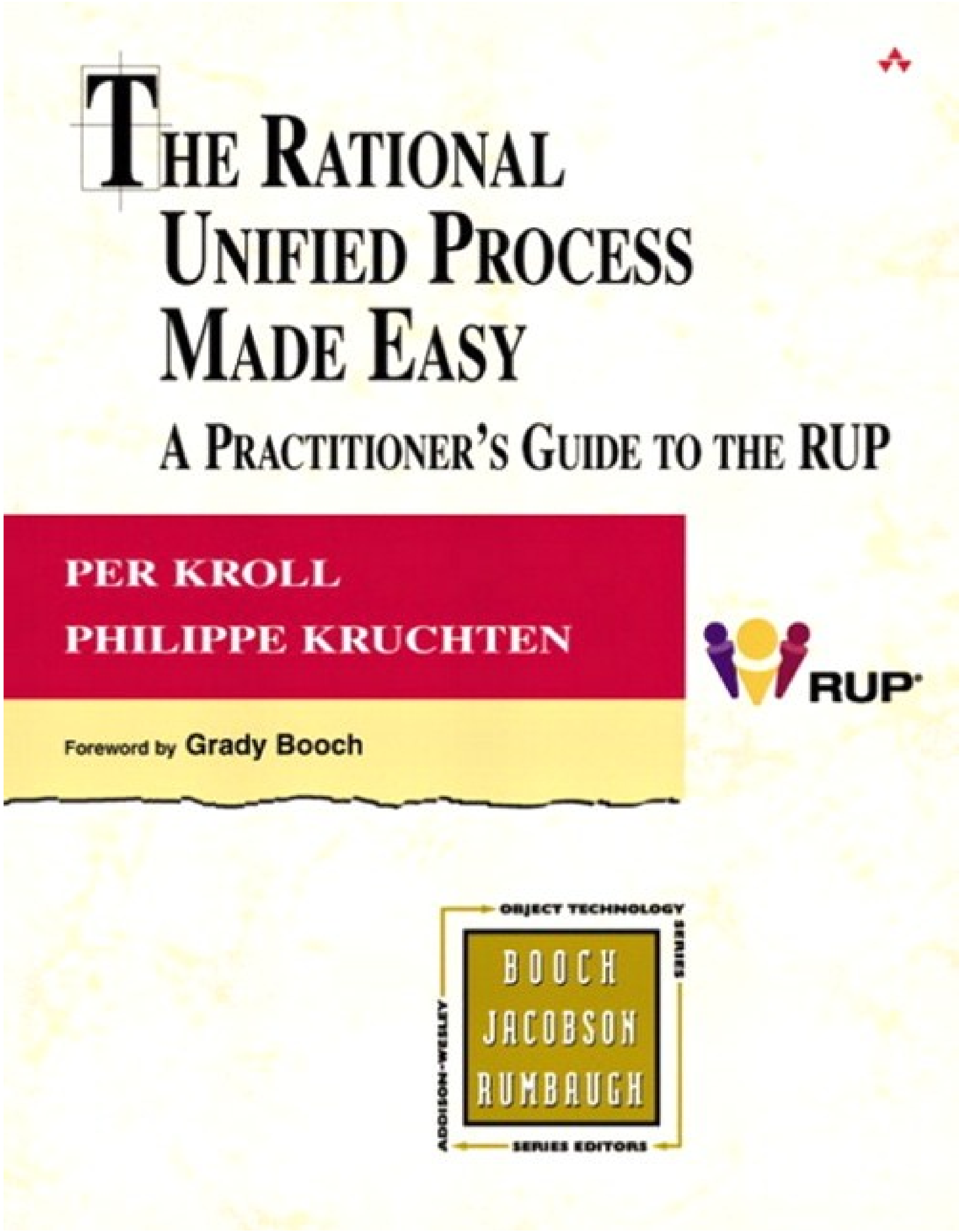
Werner Heijstek

Leiden Institute of Advanced Computer Science
Leiden University, the Netherlands

Software Engineering 2010
B.Sc. Computer Science

Thu, September 30th 2010, 11:15 – 13:00

Leiden University



THE RATIONAL UNIFIED PROCESS MADE EASY

A PRACTITIONER'S GUIDE TO THE RUP

PER KROLL

PHILIPPE KRUCHTEN

Foreword by Grady Booch

RUP

OBJECT TECHNOLOGY SERIES

BOOCH
JACOBSON
RUMBAUGH

ADDISON-WESLEY

SERIES EDITORS

These slides are based on the book

The Rational Unified Process Made Easy –
A Practitioner's Guide to RUP,

Kroll and Kruchten, Addison-Wesley, 2003

Agenda

Part I

Introducing the Rational Unified Process

Part II

The Lifecycle of a Rational Unified Process Project

Part III

Adopting the Rational Unified Process

What is RUP

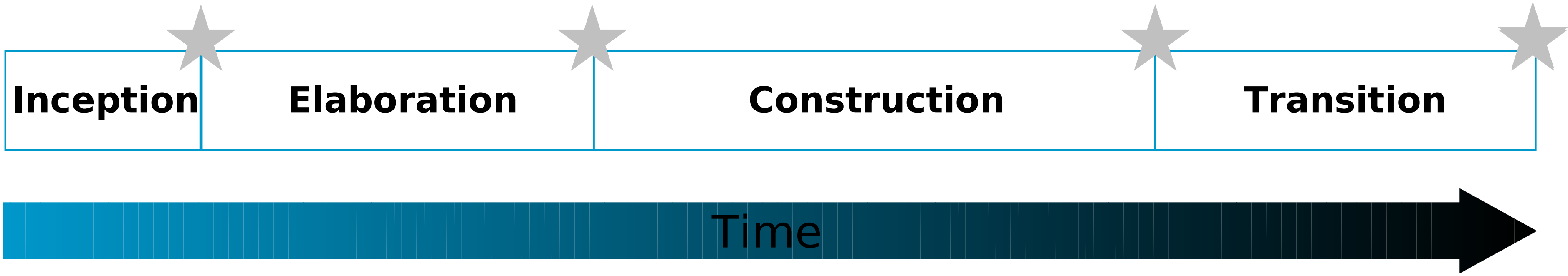
A software development approach that is iterative, architecture-centric and use-case driven

A well-defined and structured software engineering process

A process product providing a customizable process framework

Iterative Development Phases

Major Milestones



Inception: Understand what to build

- Vision, high-level requirements, business case
- Not detailed requirements

Elaboration: Understand how to build it

- Baseline architecture, most requirements detailed
- Not detailed design

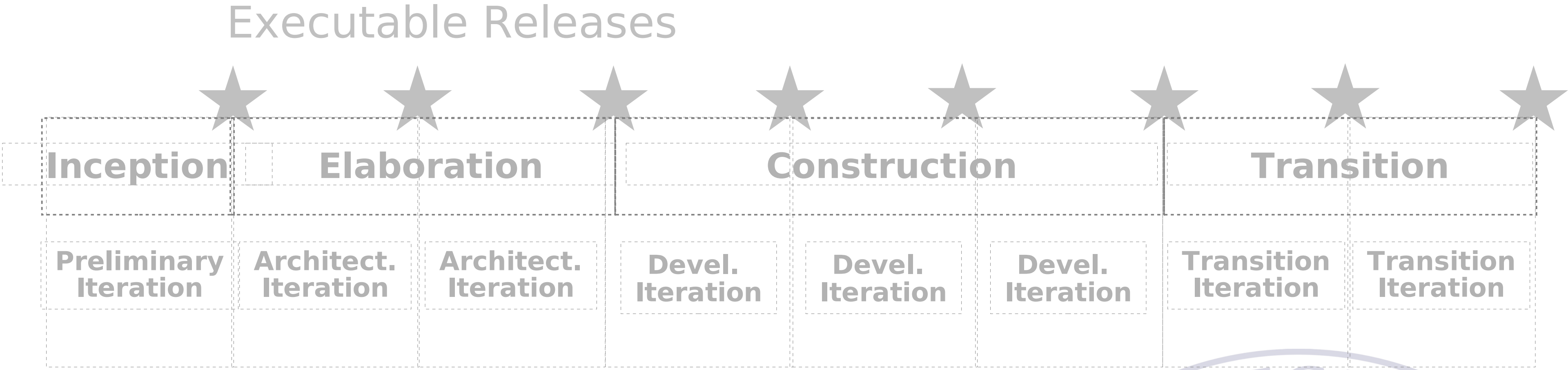
Construction: Build the product

- Working product, system test complete

Transition: Validate solution

- Stakeholder acceptance

Iterations and Phases



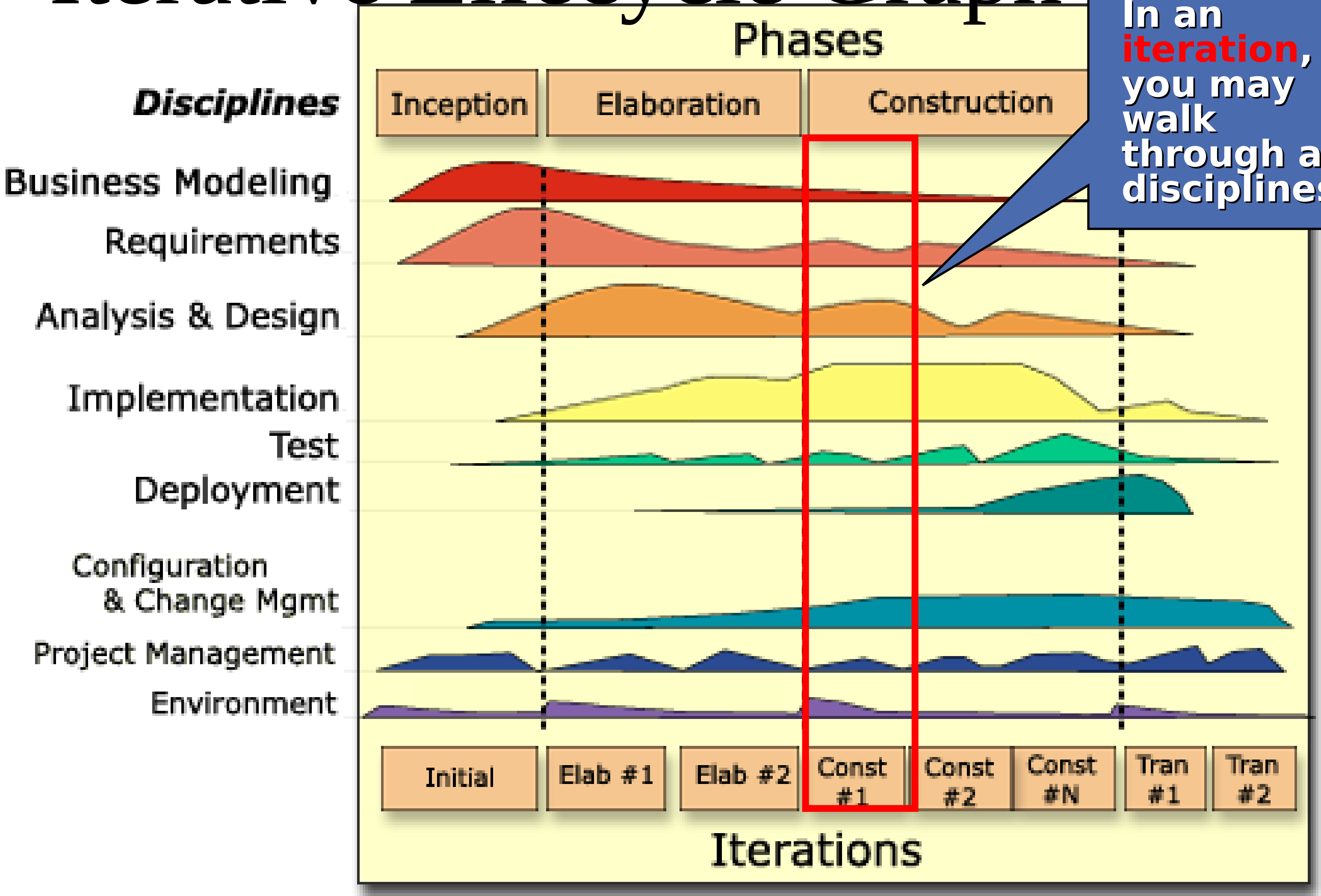
Executable Releases

Inception | Elaboration | Construction | Transition

Preliminary Iteration | Architect. Iteration | Architect. Iteration | Devel. Iteration | Devel. Iteration | Devel. Iteration | Transition Iteration | Transition Iteration

An iteration is a distinct sequence of activities with an established plan and evaluation criteria, resulting in an executable release.

Iterative Lifecycle Graph



In an iteration, you may walk through all disciplines

CONTENT STRUCTURE

TIME

Leiden University

Inception: Know What to Build

Prepare vision document and initial business case

- Include risk assessment and resource estimate

Develop high-level project requirements

- Initial use-case and domain models (10-20% complete)

Manage project scope

- Reduce risk by identifying all key requirements
- Acknowledge that requirements will change
  - Manage change, use iterative process

Inception	**Elaboration**	**Construction**	**Transition**

Elaboration: Know How to Build It

Detail requirements as necessary (~80% complete)

- Less essential requirements may not be fleshed out

Produce an executable and stable architecture

- Define, implement and test interfaces of major components
- Identify dependencies on external components and systems. Integrate shells/proxies of them.
- Some key components will be partially implemented
- Roughly 10% of code is implemented.

Drive architecture with key use cases

- 20% of use cases drive 80% of the architecture
- Design, implement and test key scenarios for use cases

Inception	**Elaboration**	**Construction**	**Transition**

Elaboration: Know How to Build It

Verify architectural qualities

- Reliability: Stress test
- Scalability and Performance: Load test

Continuously assess business case, risk profile and development plan

Inception	**Elaboratio**	**Construction**	**Transition**

Construction: Build The Product

Complete requirements and design model

Design, implement and test each component

- Prototype system and involve end users
- Incrementally evolve executable architecture to complete system

Build daily or weekly with automated build process

Test each build

- Automate regression testing
- Load and stress test to ensure architectural integrity

Deliver fully functional software (beta release)

- Includes training material, user and deployment documentation

Produce release descriptions

Inception	Elaboration	**Construction**	Transition

Transition: *Deploy to End Users*

Produce incremental ‘bug-fix’ releases

Update user manuals and deployment documentation

Update release descriptions

Execute cut-over

Conduct “post-mortem” project analysis

Inception	**Elaboration**	**Construction**	**Transition**

Key Best Practices and Principles

Best Practices

Process Made Practical

Develop Iteratively

Manage Requirements

Use Component Architectures

Model Visually (UML)

Continuously Verify Quality

Manage Change

RUP®

Develop only what is necessary

- Lean process, agility

Minimize paperwork

Be flexible

- Requirements, plan, usage of people, etc…

Learn from earlier mistakes

- Feedback loops
- Process improvement

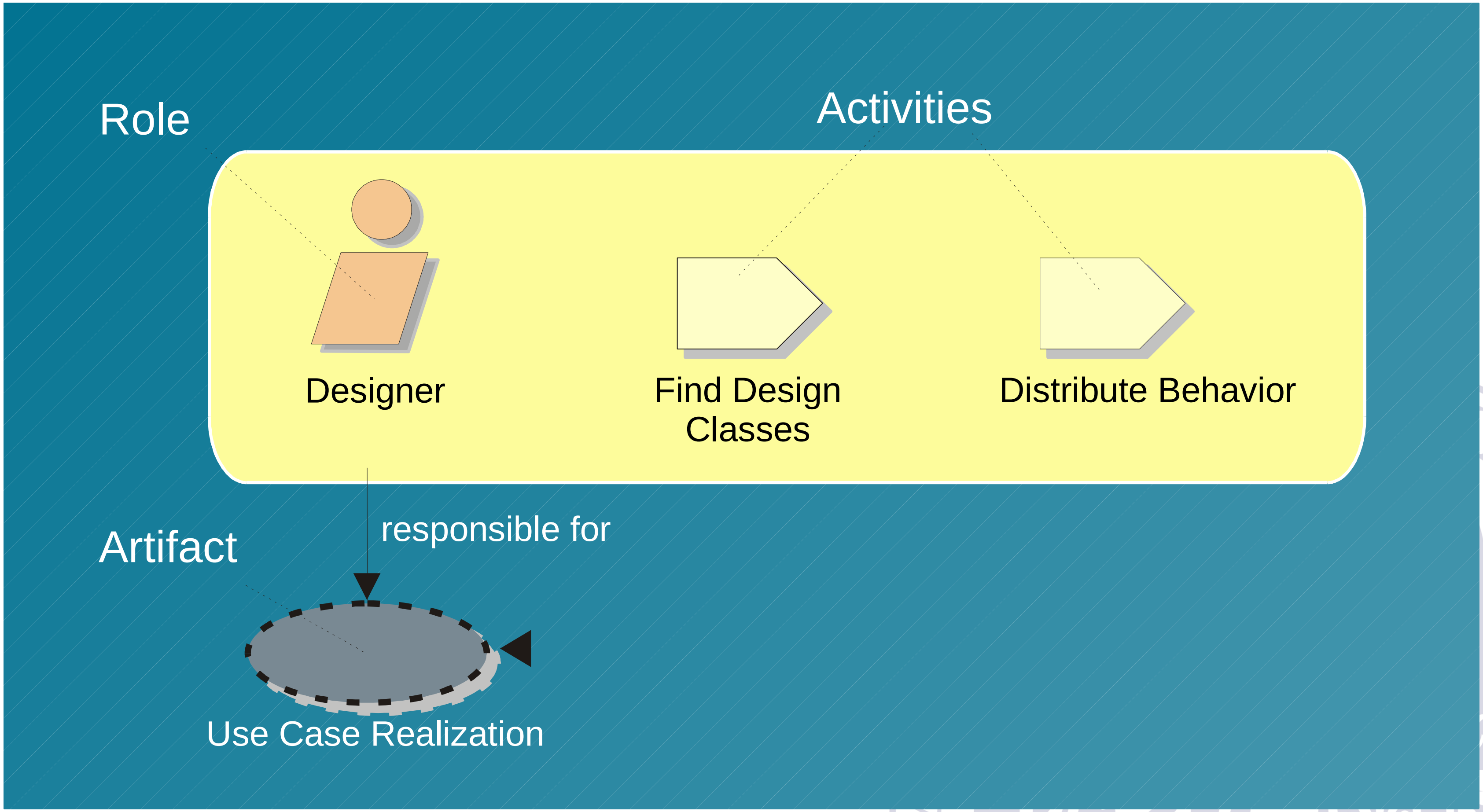
Revisit risks regularly

Establish objective, measurable criteria for progress

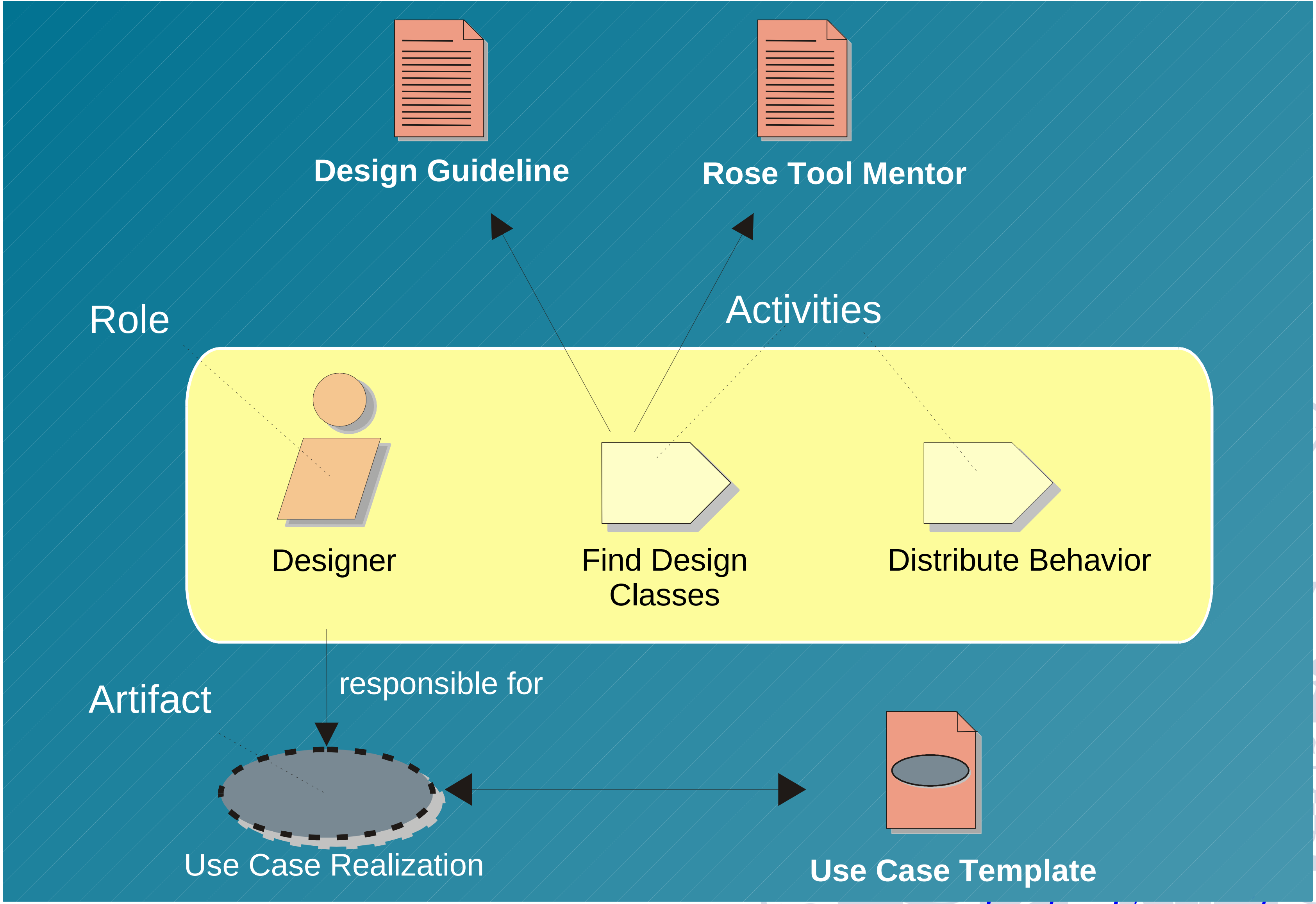
Automate

- Support process with software development tools

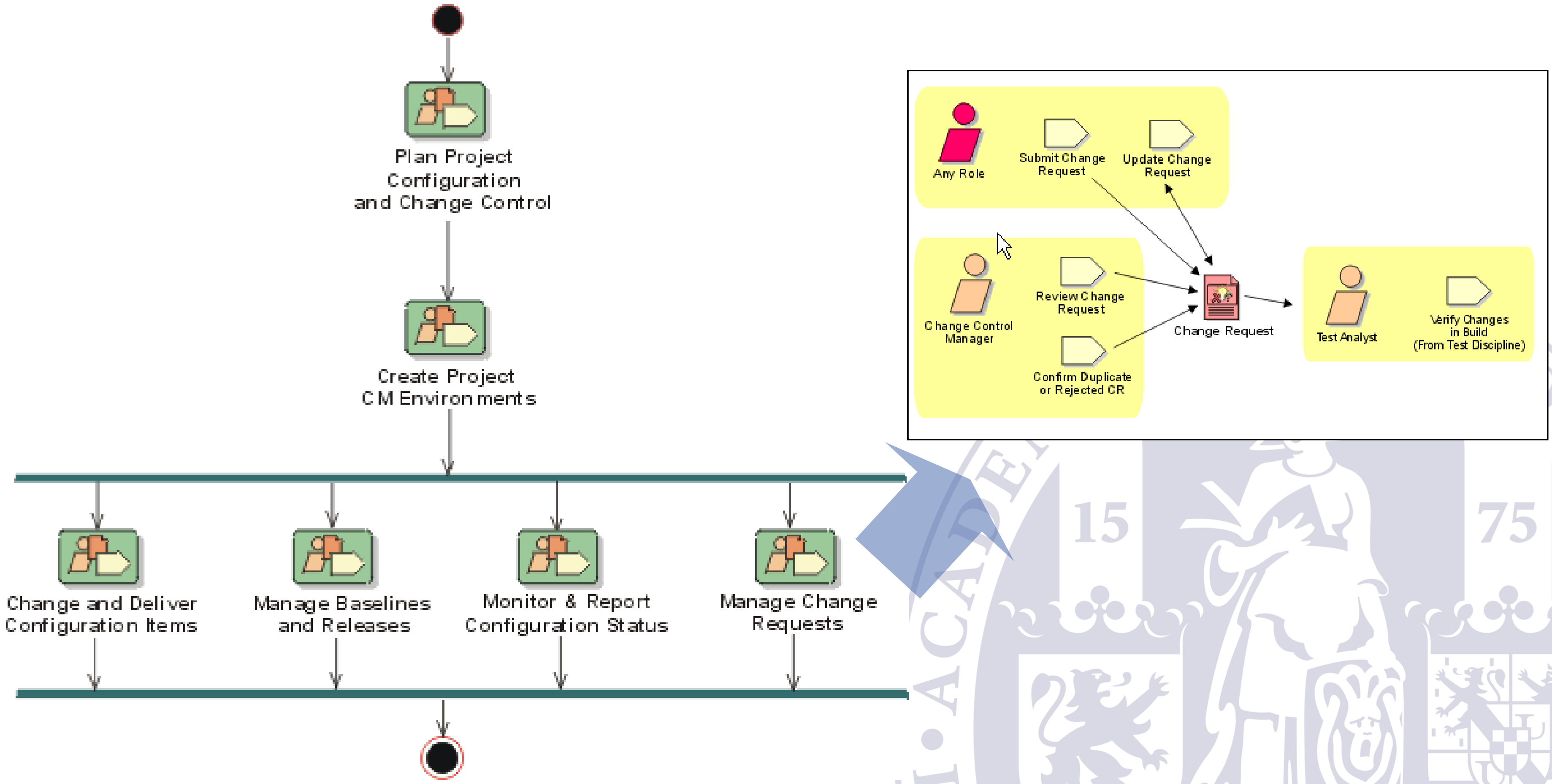
A Structured Process: Role, Artifact, Activity

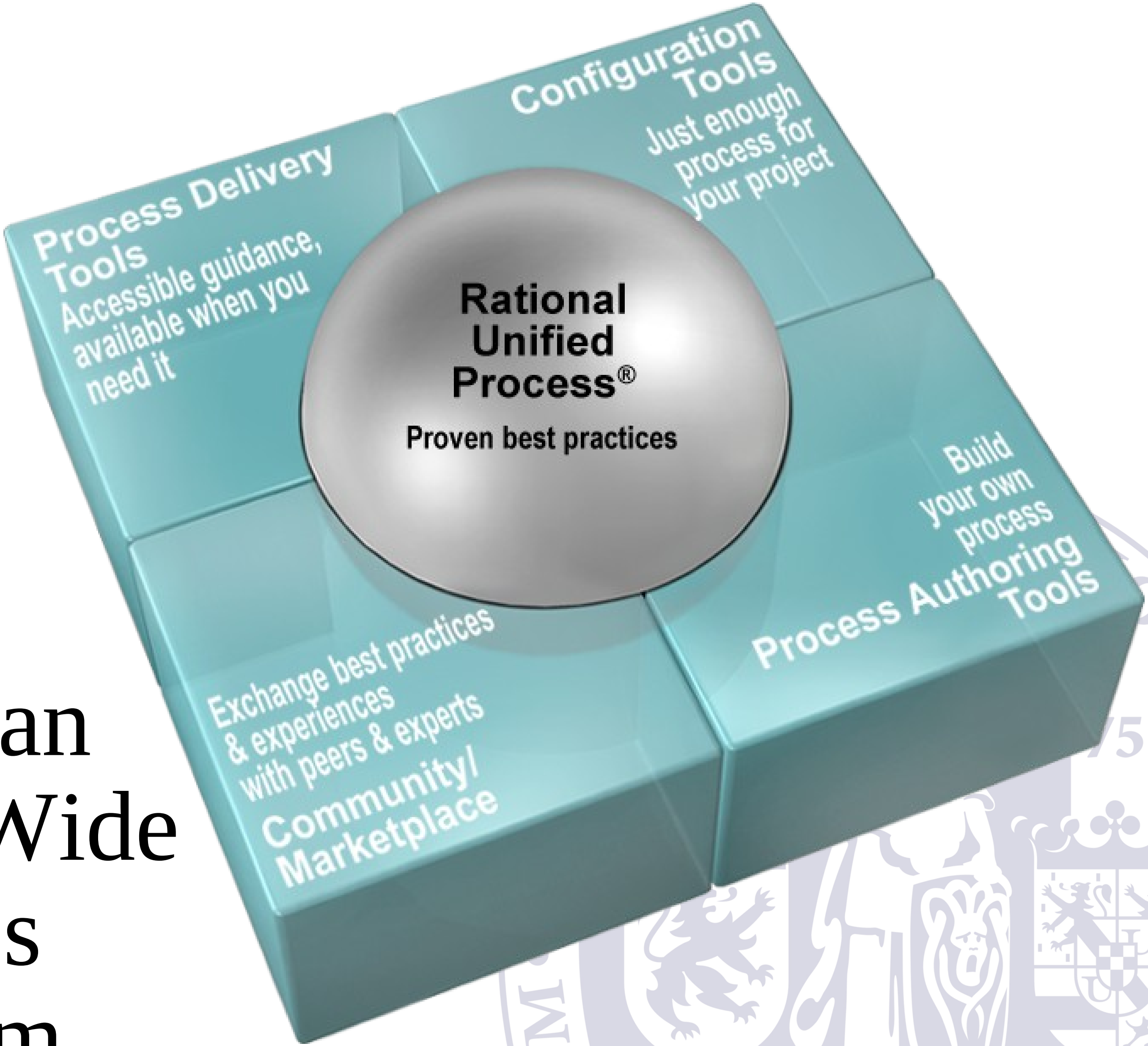


Guidelines, Templates, Tool Mentors, …



Expressed as Workflows and Workflow Details

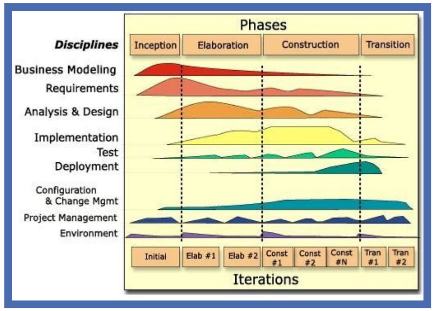

Plan Project Configuration and Change Control
Create Project CM Environments
Change and Deliver Configuration Items
Manage Baselines and Releases
Monitor & Report Configuration Status
Manage Change Requests
Any Role
Submit Change Request
Update Change Request
Change Control Manager
Review Change Request
Confirm Duplicate or Rejected CR
Change Request
Test Analyst
Verify Changes in Build (From Test Discipline)


Process Delivery Tools
Accessible guidance, available when you need it
Configuration Tools
Just enough process for your project
Rational Unified Process®
Proven best practices
Build your own process
Process Authoring Tools
Exchange best practices & experiences with peers & experts
Community/ Marketplace

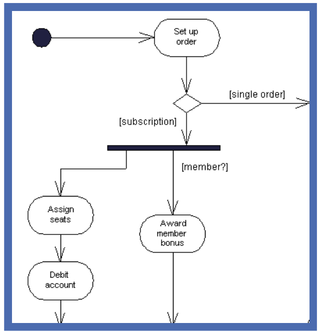
RUP is an Industry-Wide Process Platform

Delivering a More Configurable Process to a Broader Audience

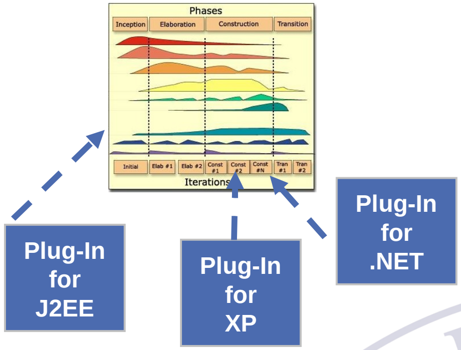
Core RUP



Customize



Configure

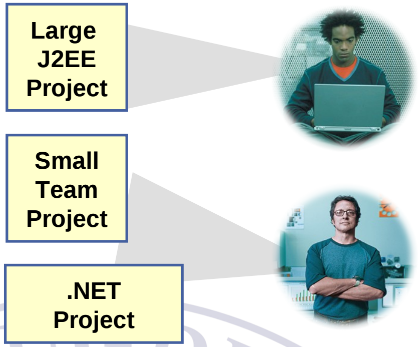


Plug-In for J2EE

Plug-In for XP

Plug-In for .NET

Personalize



Large J2EE Project

Small Team Project

.NET Project

Development organization

Process engineers, program/project offices

Project managers & team leads

Practitioners

Common methodology

- Shared understanding of terminology, deliverables, and responsibilities

Process authoring

- Leverage internal knowledge and process assets

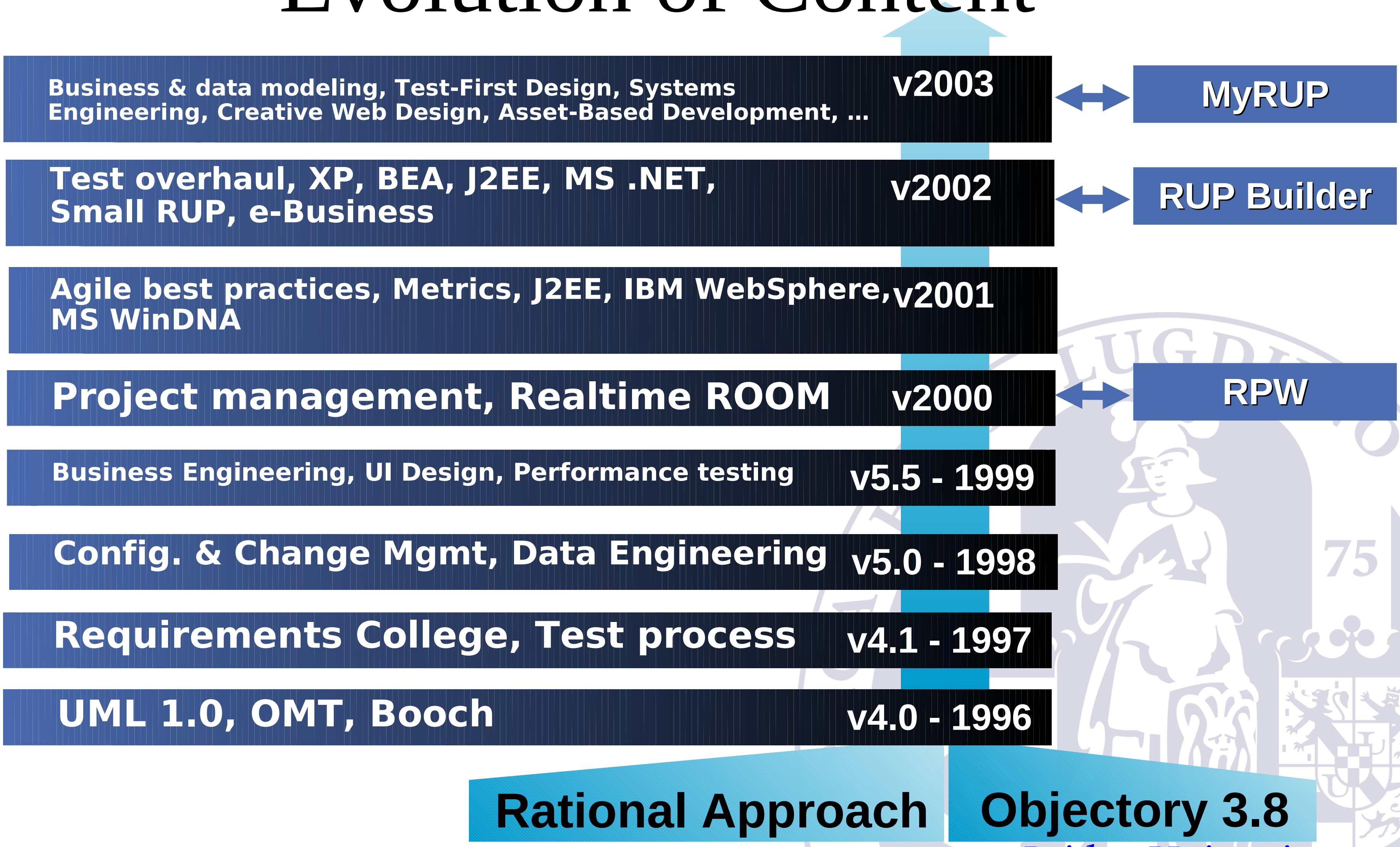
Process configuration

- Configure and deploy process for specific tools, technologies, domains

Process delivery

- Filter project content and customize tree browser

Evolution of Content



Business & data modeling, Test-First Design, Systems Engineering, Creative Web Design, Asset-Based Development, … v2003 ↔ MyRUP

Test overhaul, XP, BEA, J2EE, MS .NET, Small RUP, e-Business v2002 ↔ RUP Builder

Agile best practices, Metrics, J2EE, IBM WebSphere, MS WinDNA v2001

Project management, Realtime ROOM v2000 ↔ RPW

Business Engineering, UI Design, Performance testing v5.5 - 1999

Config. & Change Mgmt, Data Engineering v5.0 - 1998

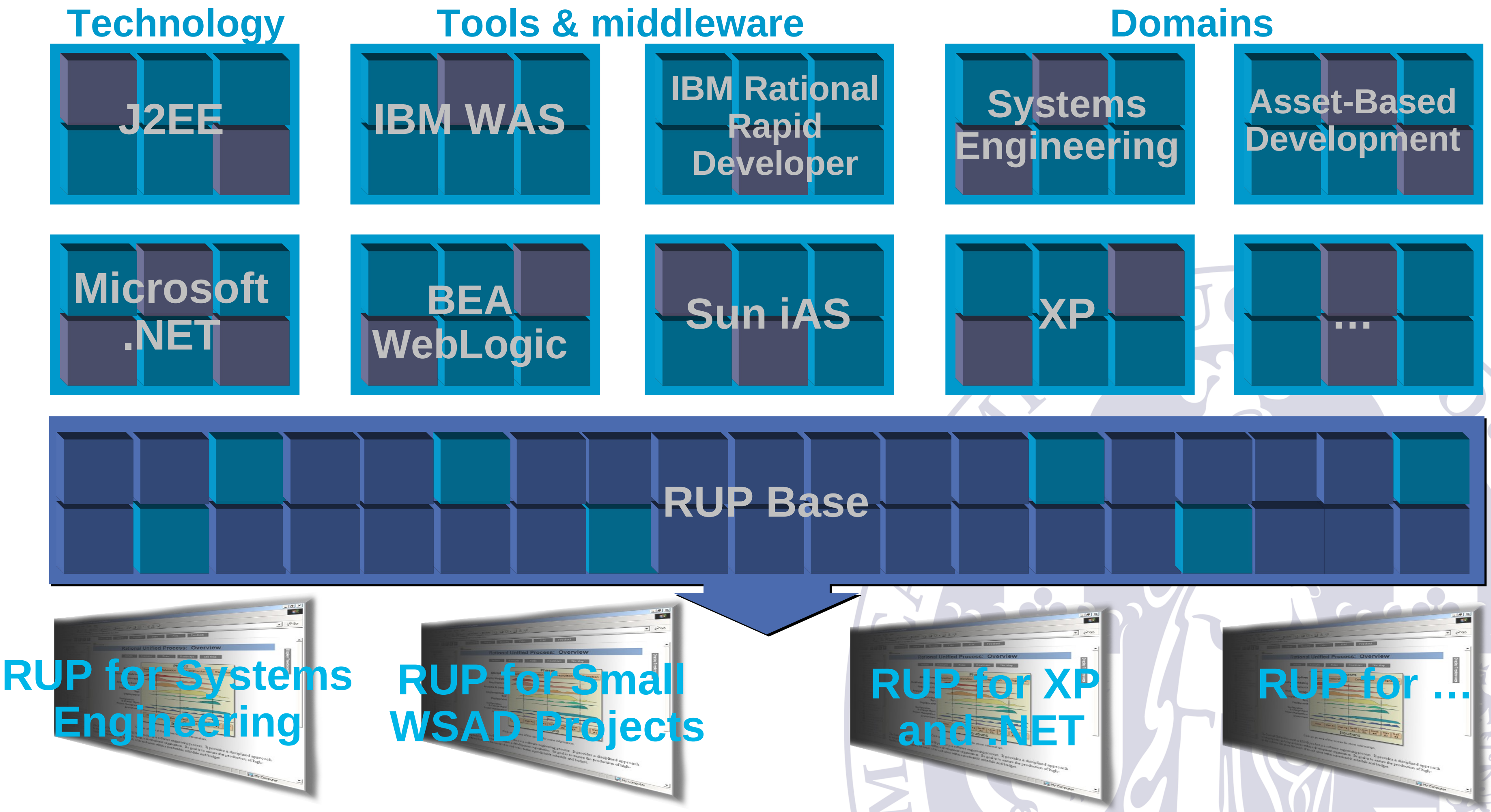
Requirements College, Test process v4.1 - 1997

UML 1.0, OMT, Booch v4.0 - 1996

Rational Approach

Objectory 3.8

RUP: Highly Configurable



Technology

Tools & middleware

Domains

J2EE

IBM WAS

IBM Rational Rapid Developer

Systems Engineering

Asset-Based Development

Microsoft .NET

BEA WebLogic

Sun iAS

XP

…

RUP Base

RUP for Systems Engineering

RUP for Small WSAD Projects

RUP for XP and .NET

RUP for …

Configuration Tools: *RUP Builder*

Project Manager: "I need to adapt RUP to my project needs"

Right-size your process through fine-granular process selection

- +100 selectable units

Small, medium, and large project configurations available as starting point

Produce role-based views

Easy access to latest content through RUP plug-in exchange

Assemble the right process

Practitioner: *MyRUP*

Practitioner:
“I want to easily find the info I need”

Personalized views

- Role-based and personalized views into your project’s process
- Add links to external and internal resources

Project Web and Extended help integrated with RUP browser

Closer integration with RDN

- Hotlinks to RDN, etc. from MyRUP
- Seamless search across RUP and RDN

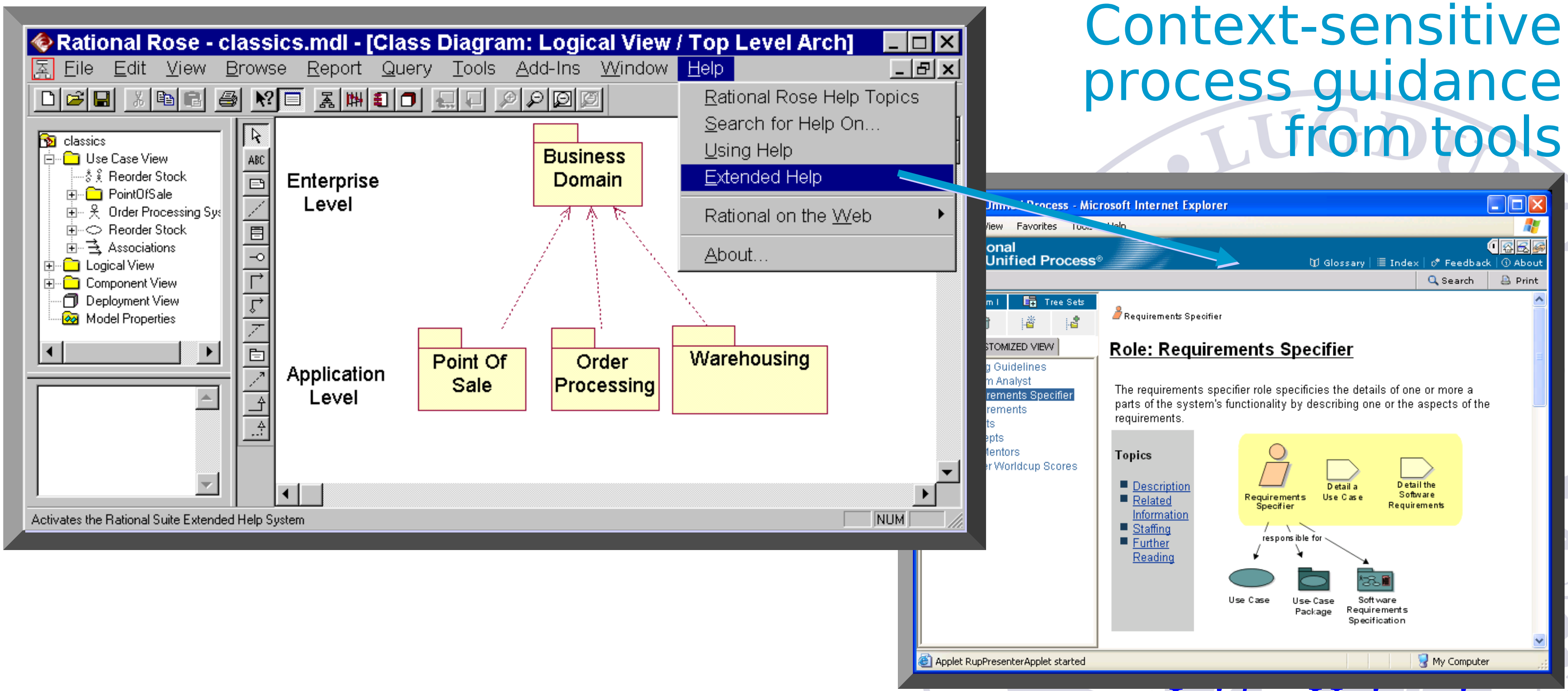
Assets available through MyRUP

Easy access through clean workspace

RUP: Integrated with Tools

- **Tool mentors**: Web-based assistance for tool use
- **Extended Help**: Process guidance from within any tool

Context-sensitive process guidance from tools



Activates the Rational Suite Extended Help System

Role: Requirements Specifier

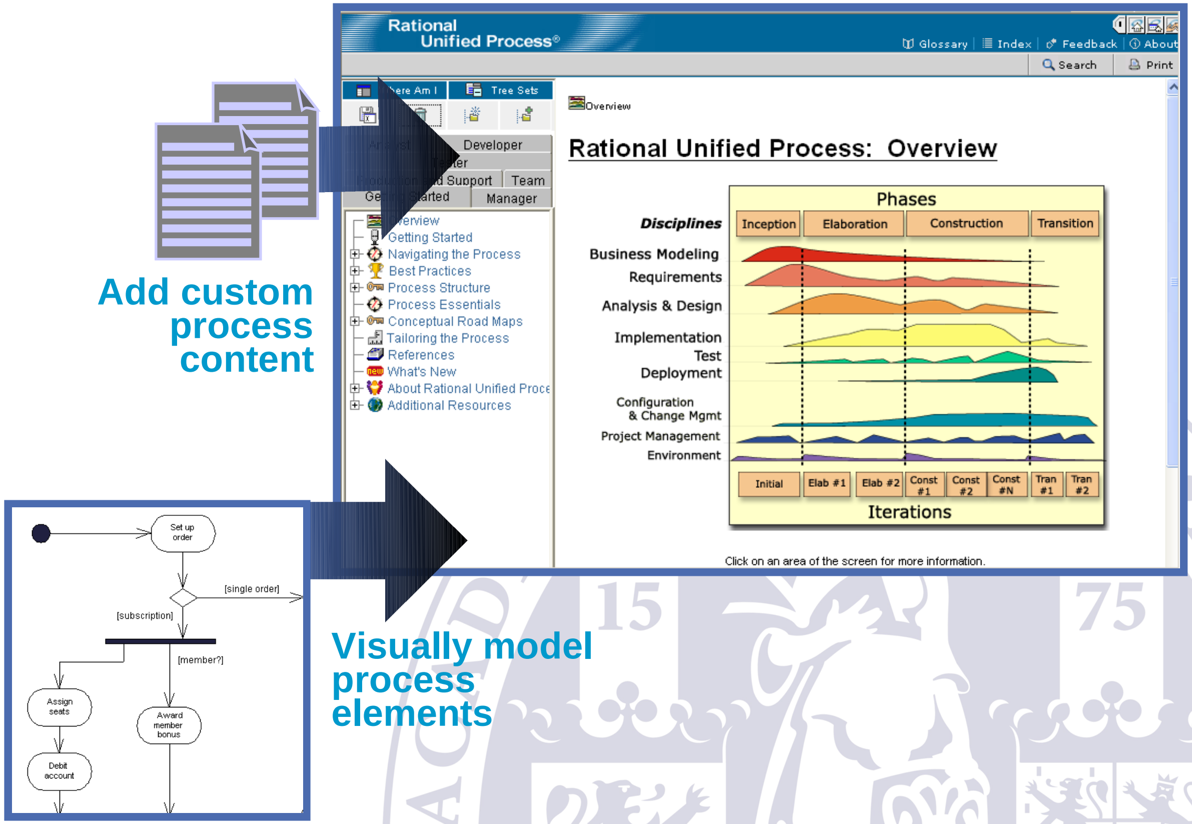
The requirements specifier role specificies the details of one or more a parts of the system's functionality by describing one or the aspects of the requirements.

Applet RupPresenterApplet started

Process Authoring: *Rational Process Workbench (RPW)*

RUP Organizer feature simplifies management of custom guidance, descriptions, examples and templates

RUP Modeler feature leverages IBM Rational XDE for visual process authoring



Add custom process content

Visually model process elements

Rational Unified Process®

Glossary | Index | Feedback | About

Search | Print

Where Am I | Tree Sets

Analyst | Developer | Tester | Production and Support | Team | Getting Started | Manager

- Overview
- Getting Started
- Navigating the Process
- Best Practices
- Process Structure
- Process Essentials
- Conceptual Road Maps
- Tailoring the Process
- References
- What's New
- About Rational Unified Proce
- Additional Resources

Overview

Rational Unified Process: Overview

Click on an area of the screen for more information.

RUP Versatility

Used in project of varying size and “ceremony” levels

- Majority of RUP projects have <15 people
- Also used in programs with thousands of people
- Facilitates Extreme Programming to formal process standards

Used in a broad set of industries such as:

- Financial institutes and insurance
- Automotive, system integrators, government, ..
- Telecommunication, defense industry, …

Provides explicit guidance for:

- Custom application development
- Systems engineering
- Legacy evolution

Extended by customers to guide in:

- Package implementation

The Spirit of The Rational Unified Process

1. Attack major risks early and continuously… or they attack you
2. Ensure that you deliver value to your customer
3. Have a maniacal focus on working software
4. Accommodate change early in the project
5. Baseline an executable architecture early on
6. Build your system with components
7. Work closely together as one team
8. Make quality a way of life, not an afterthought

Waterfall Development Lifecycle

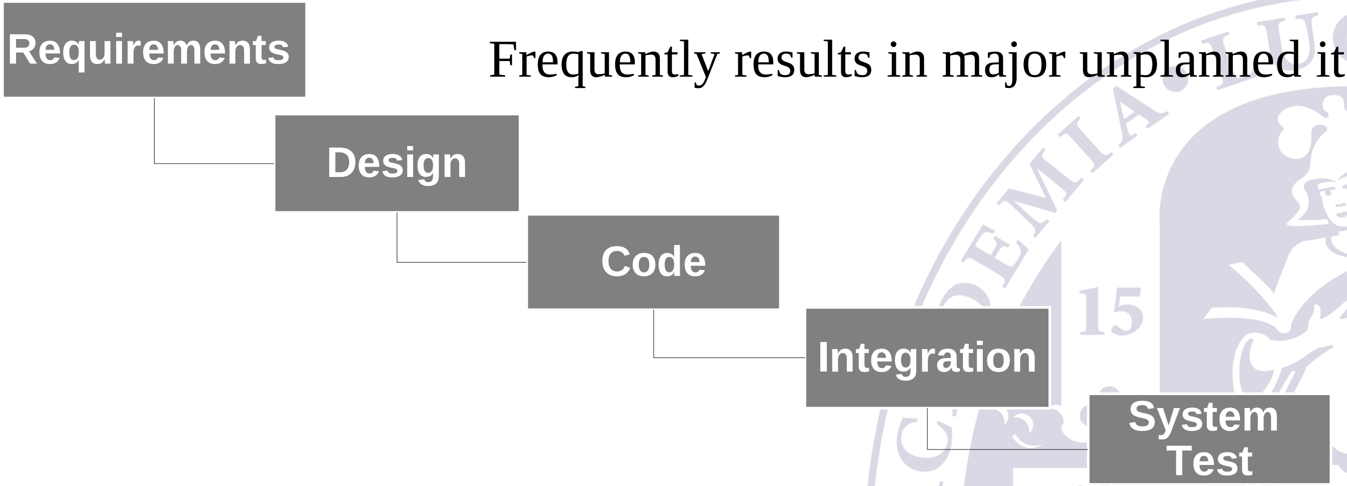
Late discovery of issues

Subjective and error-prone measure of progress

Late integration and testing

Precludes early deployment

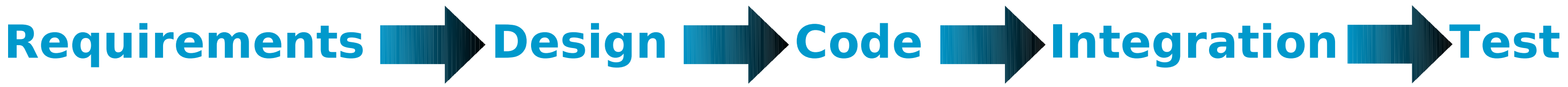
Frequently results in major unplanned iterations

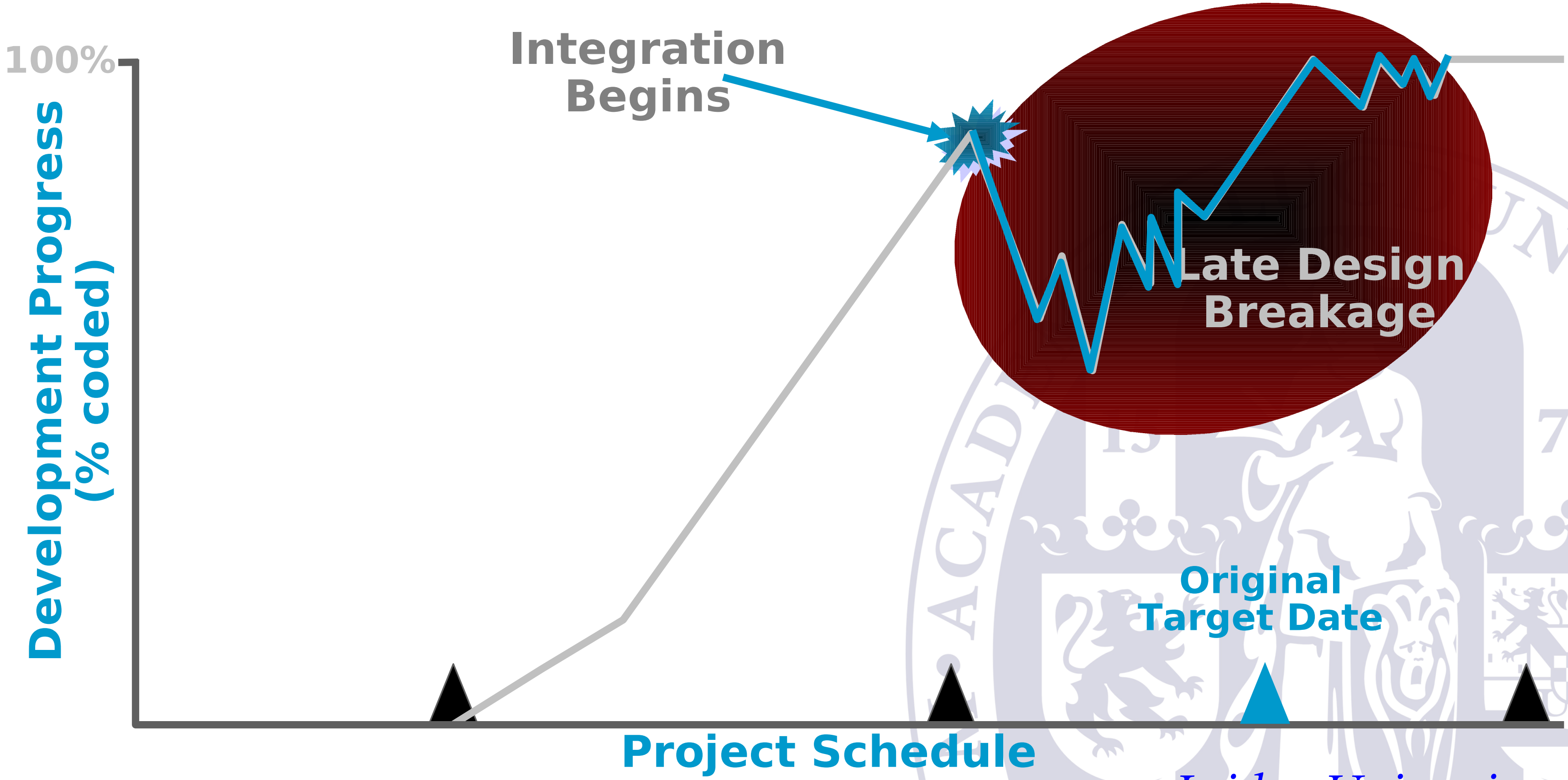


Winston Royce, 1971

What Happens in Practice

Sequential activities:





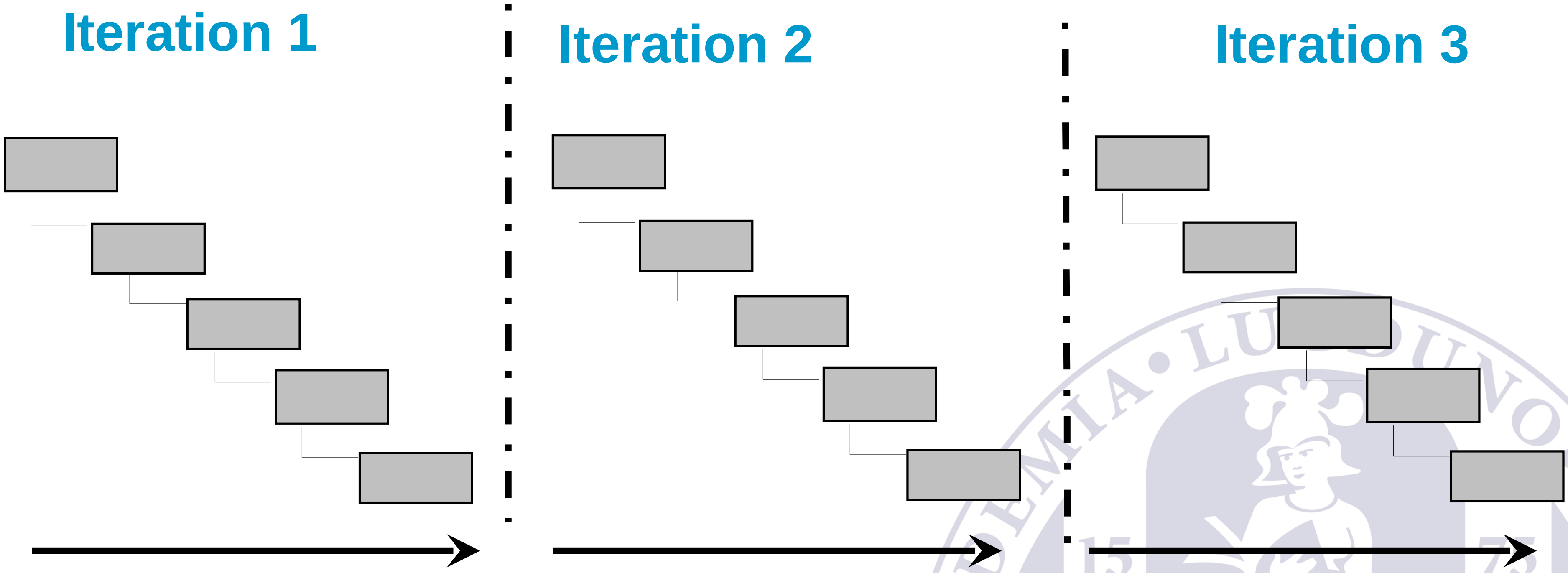
Waterfall: Hard to Scale up

A waterfall approach can not properly handle the growing complexity associated with

- Increased duration
- Increased application size
- Larger and/or distributed team
- Increased technical complexity
- Novelty of technology

The root cause of the problem with the waterfall lifecycle is that it does not allow to identify and mitigate risks early enough

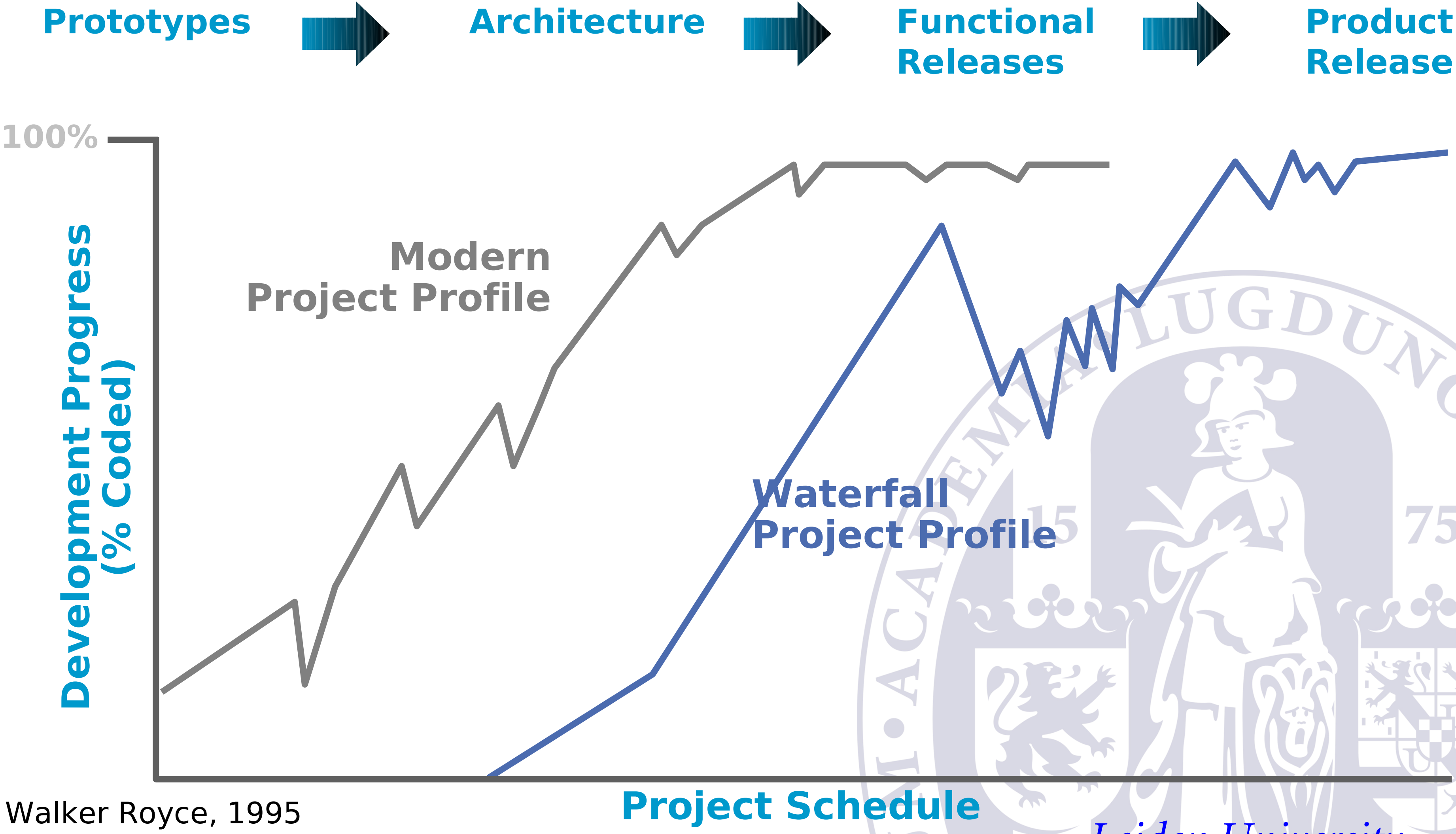
Iterative Development



- Earliest iterations address greatest risks
- Each iteration produces an executable release
- Each iteration includes integration and test

Better Progress Profile

Sequential phases, but iterative activities



Walker Royce, 1995

Risk Mitigation: Hitting Hard Problems Earlier

When the initial risks are mitigated, new ones emerge

Do not do just the easy stuff, to look good

Keep re-planning based on all new information

In iterative development, you cannot lie to yourself very long

2. Ensure That You Deliver Value to Your Customer

Focus on key requirements

- Capture, document
- Organize, prioritize

Requirements will change

- Evaluate impact of change and decide what changes to implement
- Propagate changes to all team members

Make requirements accessible

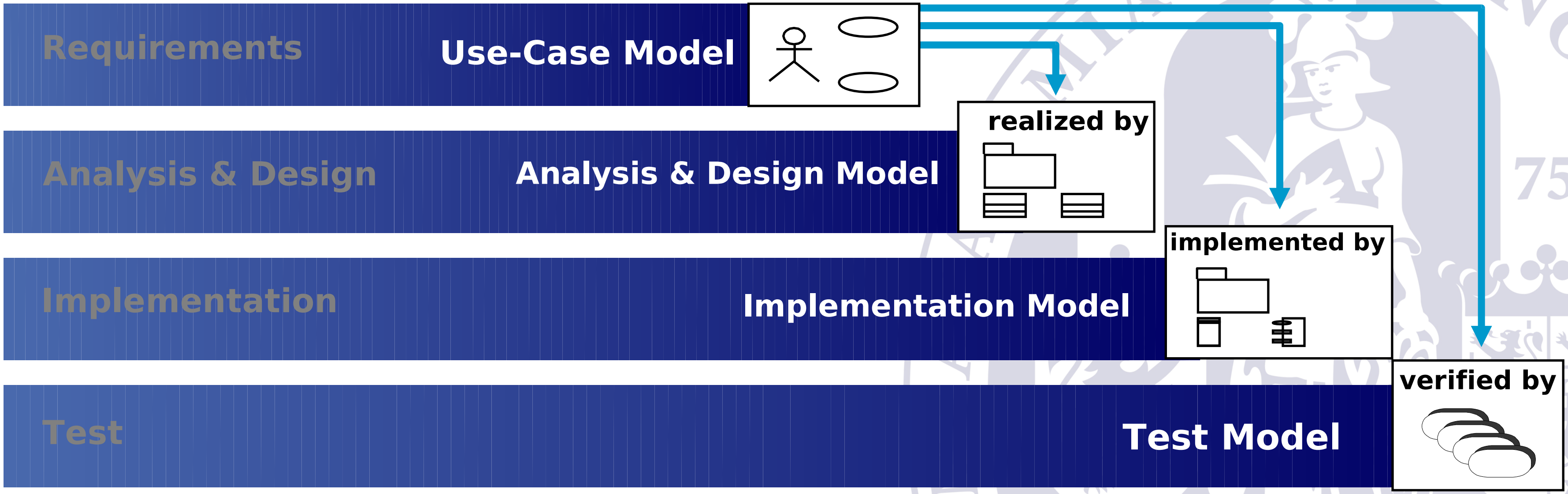
Requirements management leverages your ability to deliver products that meet user needs

Use-Case Driven Development

A use case describes complete and meaningful services that your system offers to users and other systems

Use cases drive the work through each iteration

- Planning of iterations
- Creation and validation of the architecture
- Definition of test cases and procedures
- Design of user interfaces and creation of user documentation



Requirements

Use-Case Model

realized by

Analysis & Design

Analysis & Design Model

implemented by

Implementation

Implementation Model

verified by

Test

Test Model

3. Have a Maniacal Focus on Working Software

Measure progress primarily by reviewing executable code, and test results

- Plans, requirements, designs and other by-products often provide a false perception of progress and status

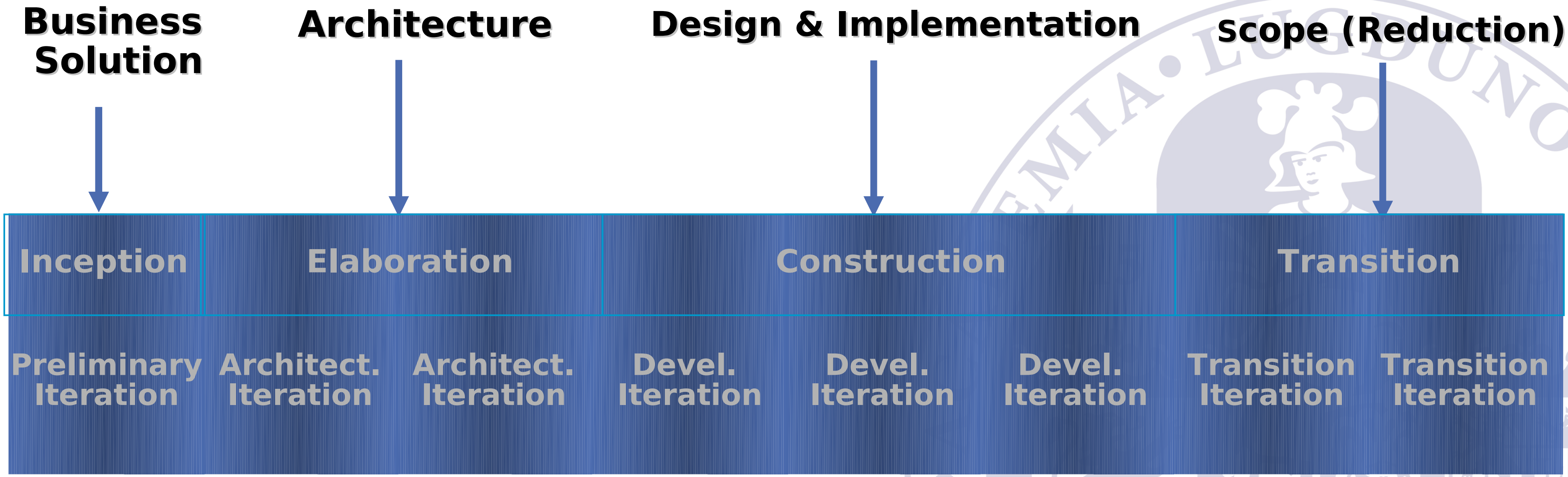
Focus on the final, delivered product, and only the artifacts that matter to get at this goal consistently

- Streamline the process
- Do not use all of the RUP!
  Only use what makes sense to your project

4. Accommodate Change Early in the Project

Today’s systems are too complex to get the requirements, architecture, design, implementation and scope right the first time

Provides freedom to change:



Business Solution

Architecture

Design & Implementation

Scope (Reduction)

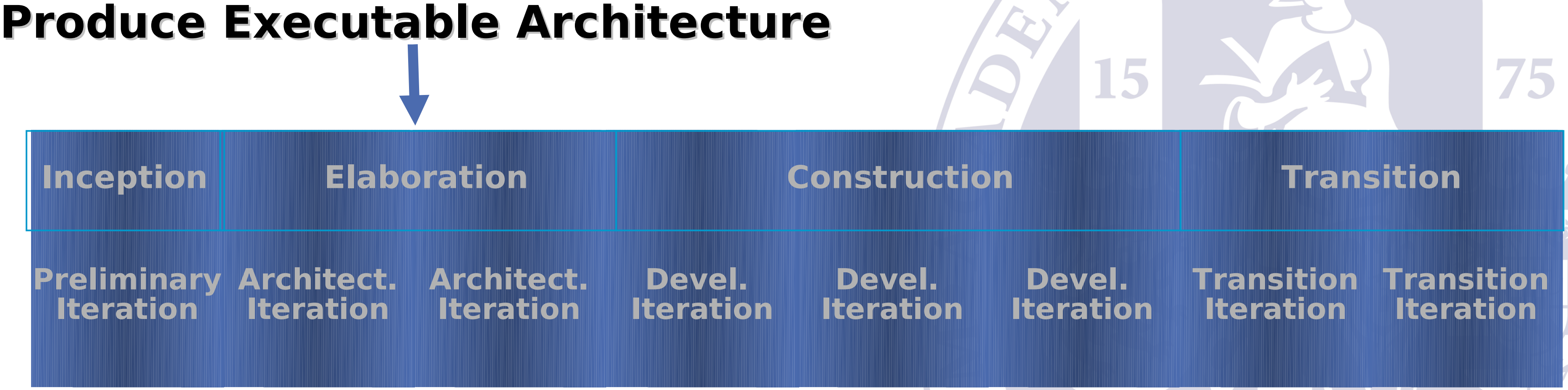
Inception	Elaboration		Construction			Transition	
Preliminary Iteration	Architect. Iteration	Architect. Iteration	Devel. Iteration	Devel. Iteration	Devel. Iteration	Transition Iteration	Transition Iteration

5. Baseline an Executable Architecture Early

Architecture provides a skeleton structure of your system

- Subsystems, key components, interfaces, architectural mechanisms (solutions for common problems, such as persistency, inter-process communication, …)

Implementing and testing the architecture mitigates most technical risks


Produce Executable Architecture

Inception	Elaboration		Construction			Transition	
Preliminary Iteration	Architect. Iteration	Architect. Iteration	Devel. Iteration	Devel. Iteration	Devel. Iteration	Transition Iteration	Transition Iteration

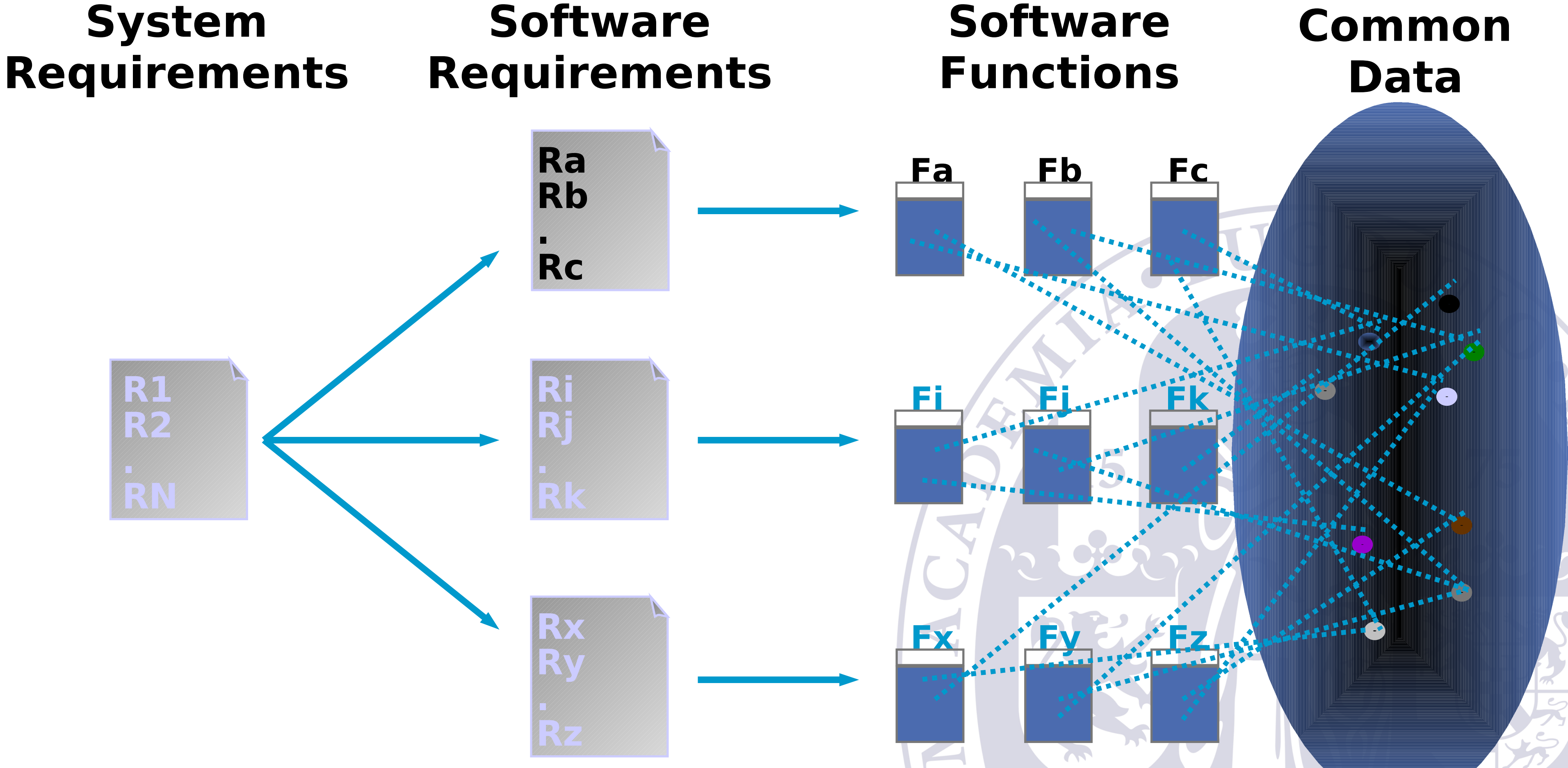
The Spirit of The Rational Unified Process

1. Attack major risks early and continuously… or they attack you
2. Ensure that you deliver value to your customer
3. Have a maniacal focus on working software
4. Accommodate change early in the project
5. Baseline an executable architecture early on
6. Build your system with components
7. Work closely together as one team
8. Make quality a way of life, not an afterth

Traditional Functional Decomposition

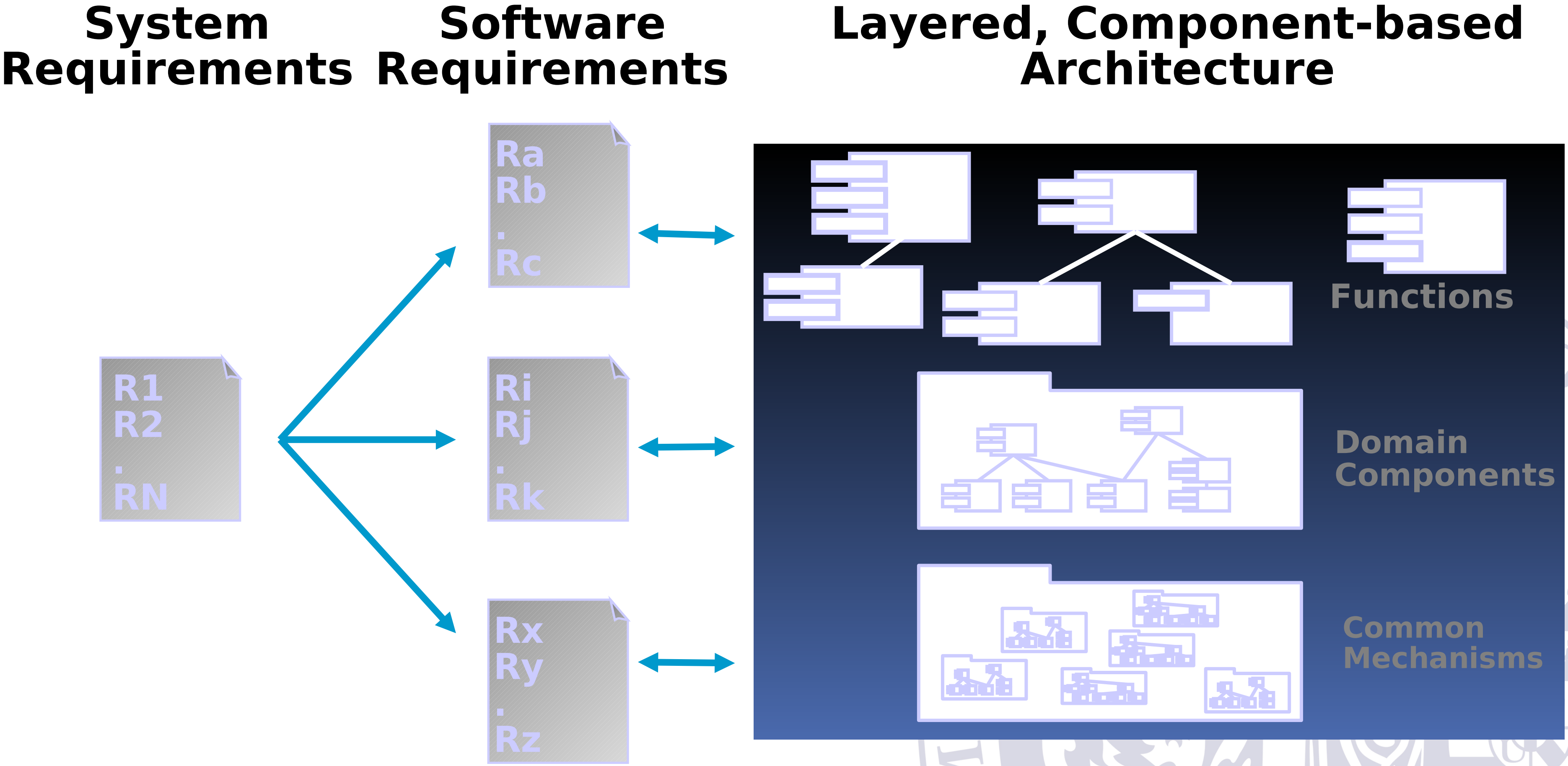
Requirements driven

Many dependencies creates inflexible systems



6. Build Your System with Components

Component architecture provides flexibility



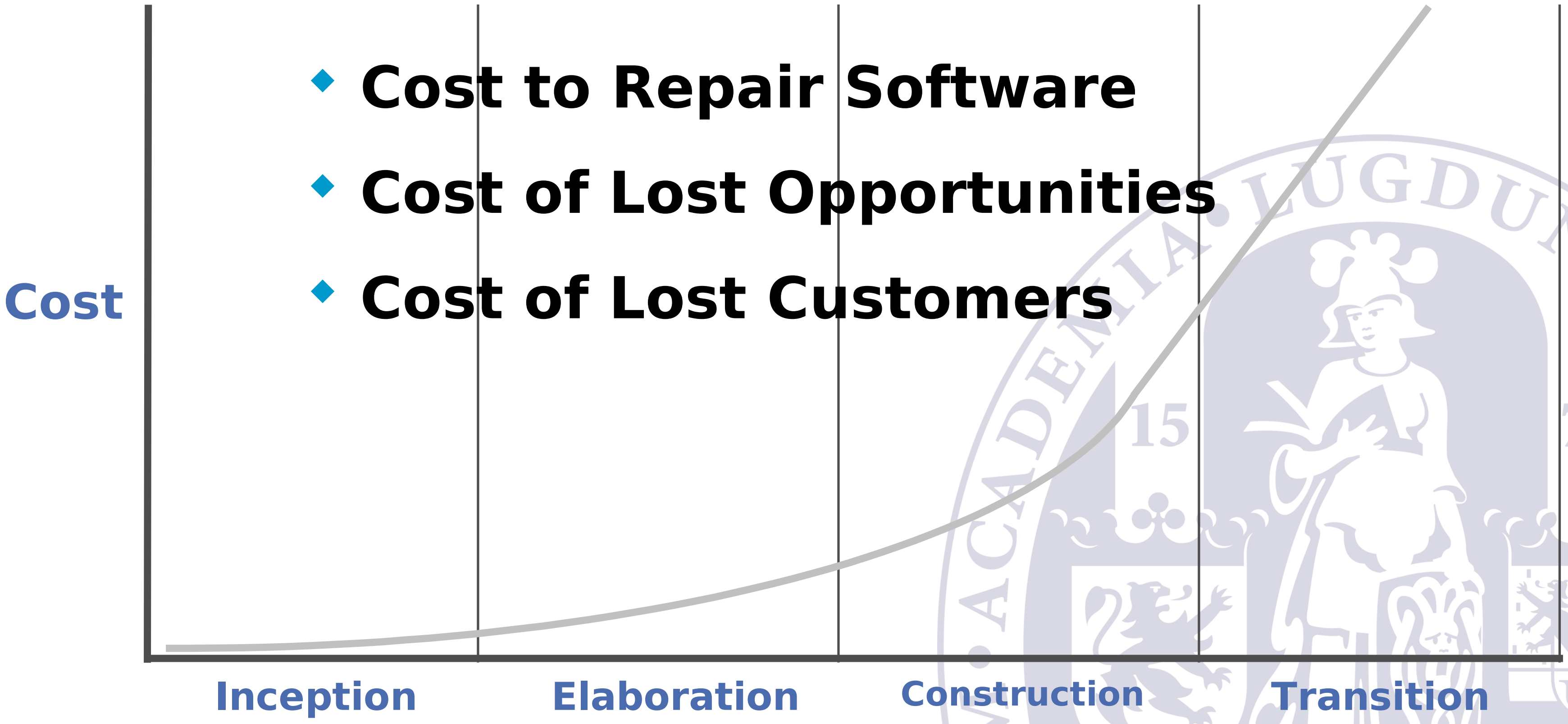
Leiden University

7. Work Closely Together As One Team

- Empowered and self-managed
  - Clear vision
- Accountable for team results
  - Clear expectations
  - All for one, one for all - avoid
    “My design was good, your code didn’t work”
- Optimized communication
  - Face-to-face rather than e-mail
  - Effective process (right-sized for your project)
  - Organize around architecture, not around functions
  - Get the right tool support
    - Easy access to current requirement
    - Private workspaces
    - Easy access to defects….
    - …

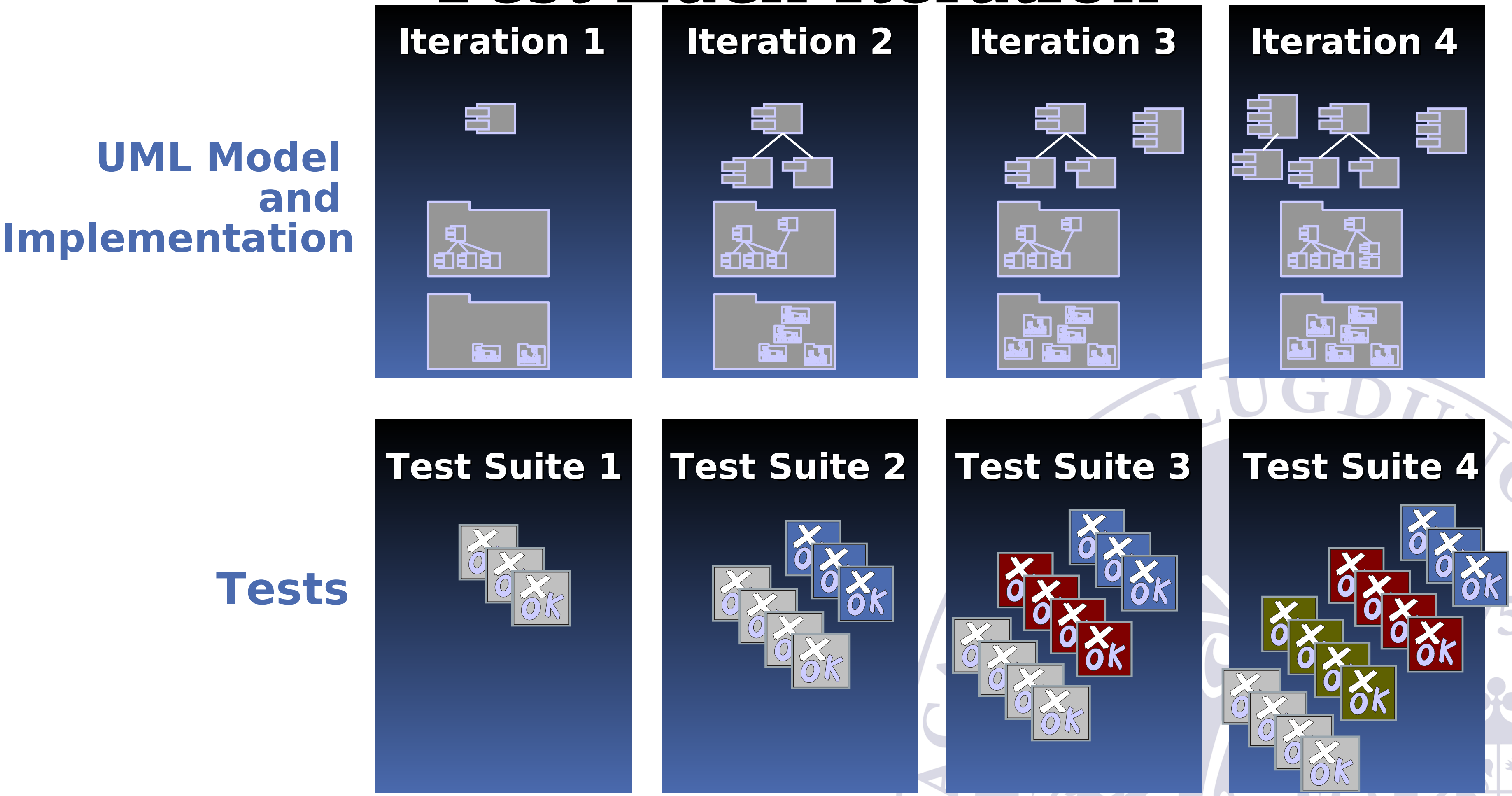
8. Make Quality a Way of Life, Not an Afterthought

Software problems are
100 to 1000 times more costly
to find and repair after deployment



- Cost to Repair Software
- Cost of Lost Opportunities
- Cost of Lost Customers

Test Each Iteration



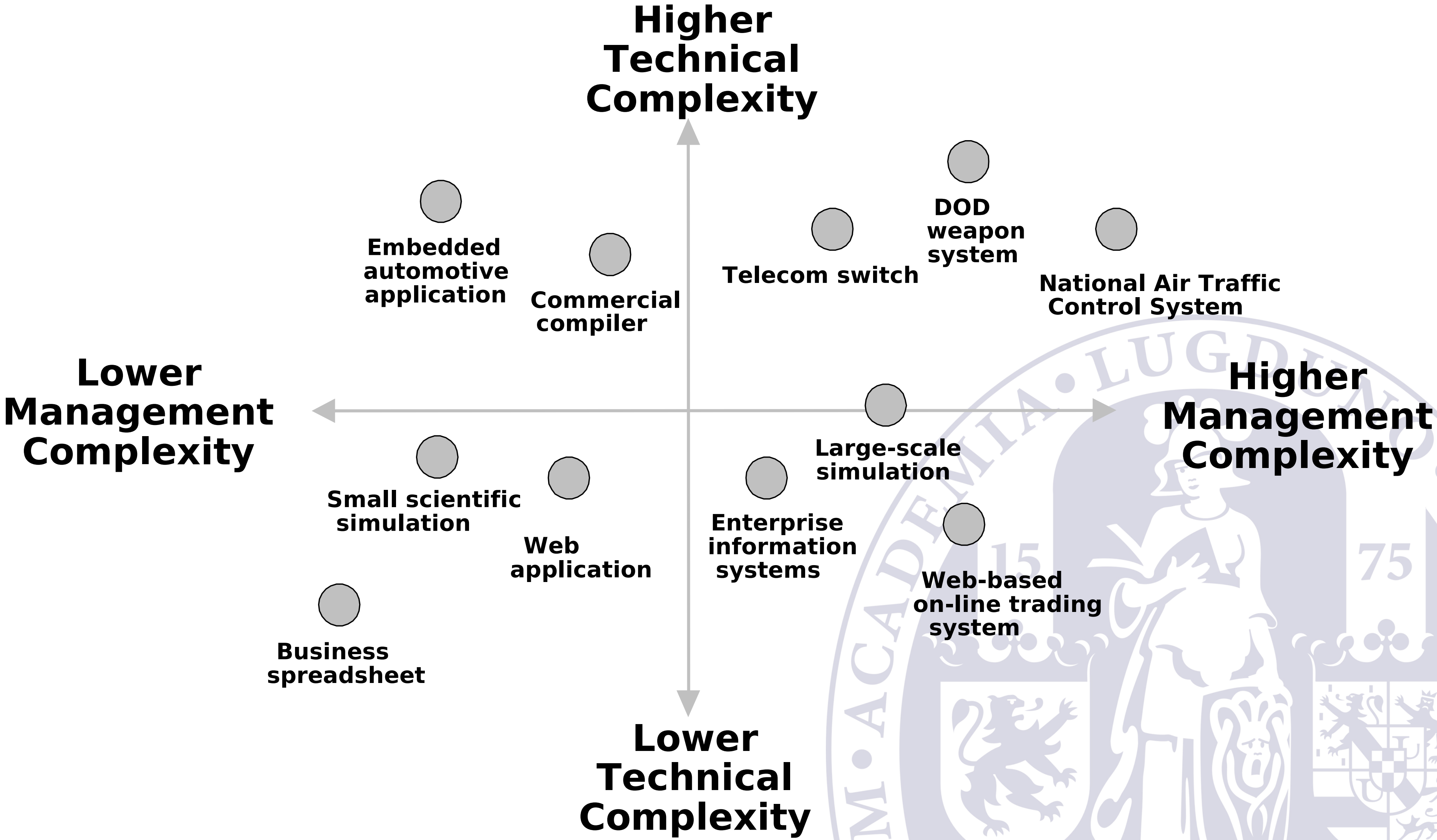
It is often cheaper to find a problem through early implement-ation and testing, than through detailed design review.

Leiden University

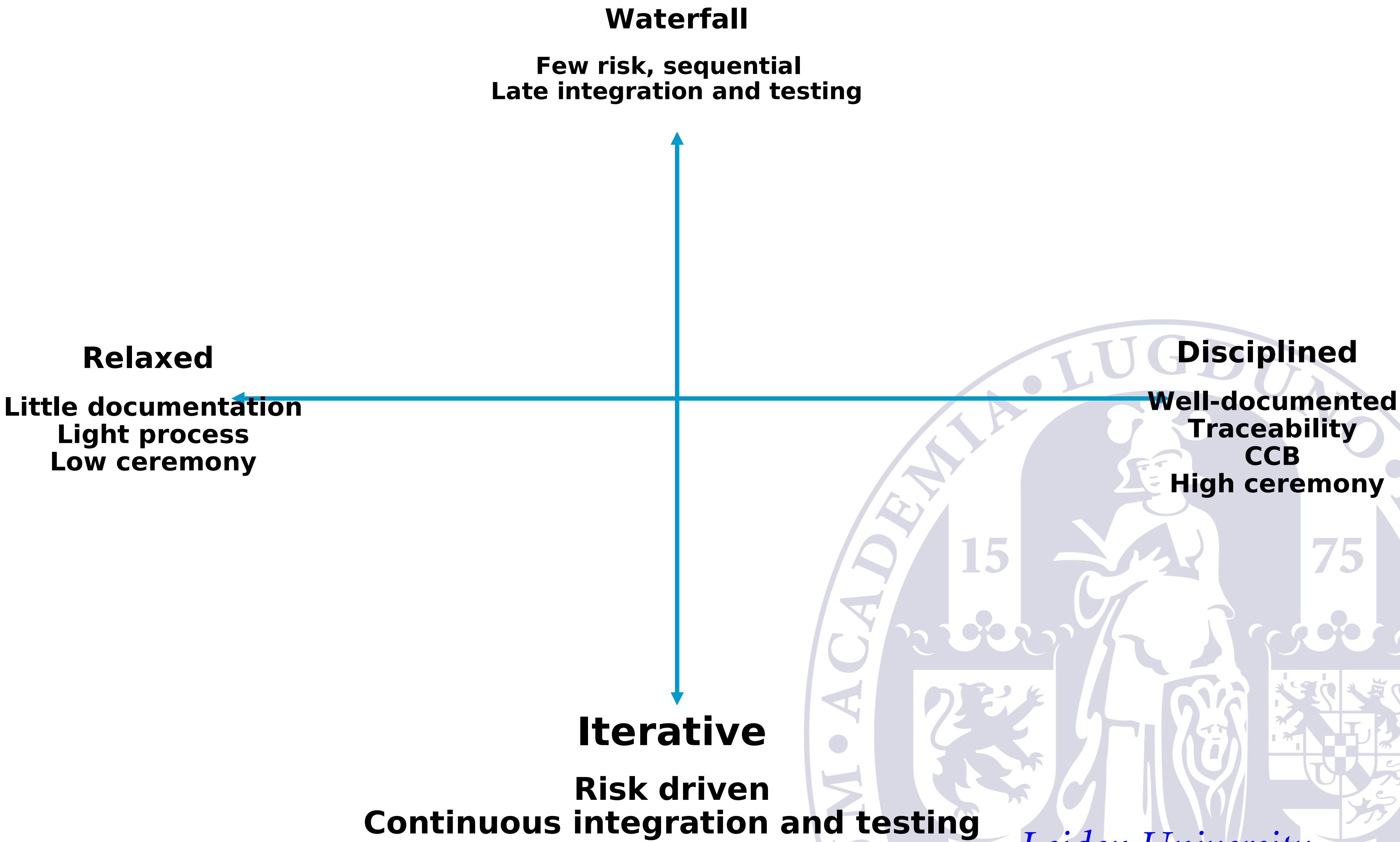
Summary: *Spirit of RUP*

- RUP embodies the principles of Spirit of RUP
- When adopting RUP, focus on the principles that will add the most value to your organization
- Continuously improve your ability to follow these principles
- Continuously revisit the Spirit of RUP

Can a single process fit all these?


Higher Technical Complexity
Embedded automotive application
Commercial compiler
Telecom switch
DOD weapon system
National Air Traffic Control System
Lower Management Complexity
Higher Management Complexity
Large-scale simulation
Small scientific simulation
Web application
Enterprise information systems
Web-based on-line trading system
Business spreadsheet
Lower Technical Complexity

Two dimensions, four (or more) process styles



Waterfall

Few risk, sequential
Late integration and testing

Relaxed

Little documentation
Light process
Low ceremony

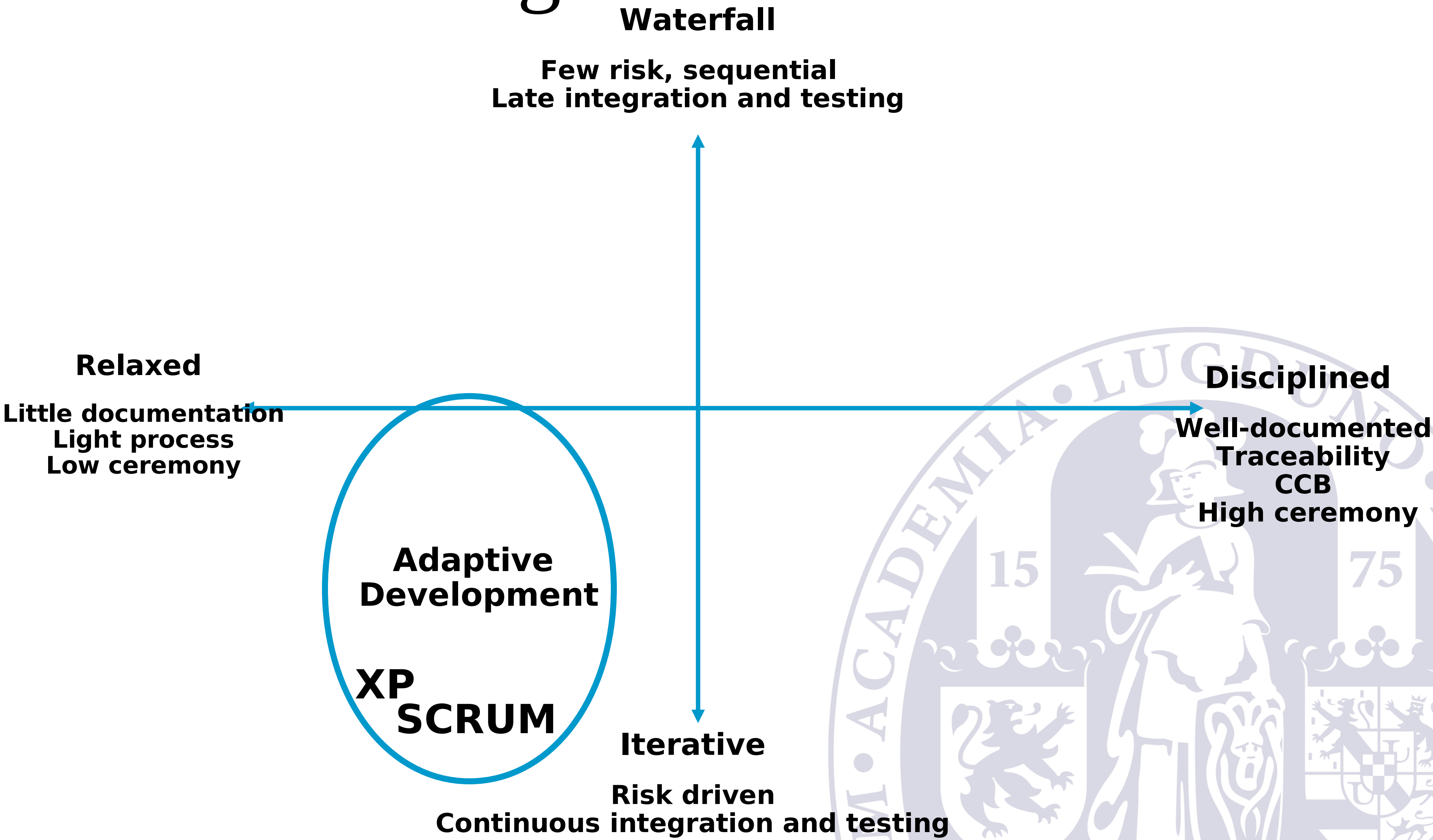
Disciplined

Well-documented
Traceability
CCB
High ceremony

Iterative

Risk driven
Continuous integration and testing

Agile Processes



Waterfall

Few risk, sequential
Late integration and testing

Relaxed

Little documentation
Light process
Low ceremony

Disciplined

Well-documented
Traceability
CCB
High ceremony

Adaptive Development

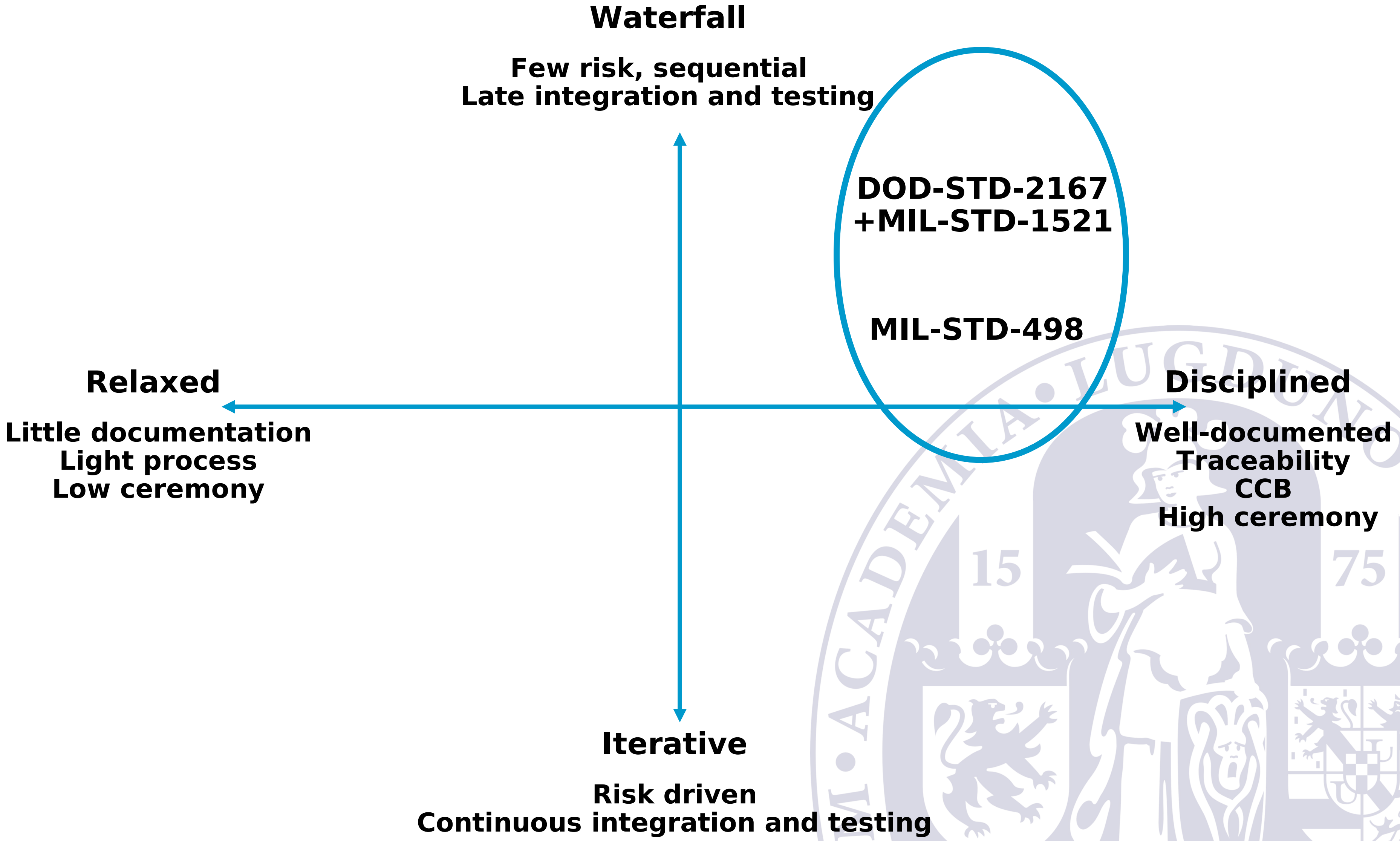
XP

SCRUM

Iterative

Risk driven
Continuous integration and testing

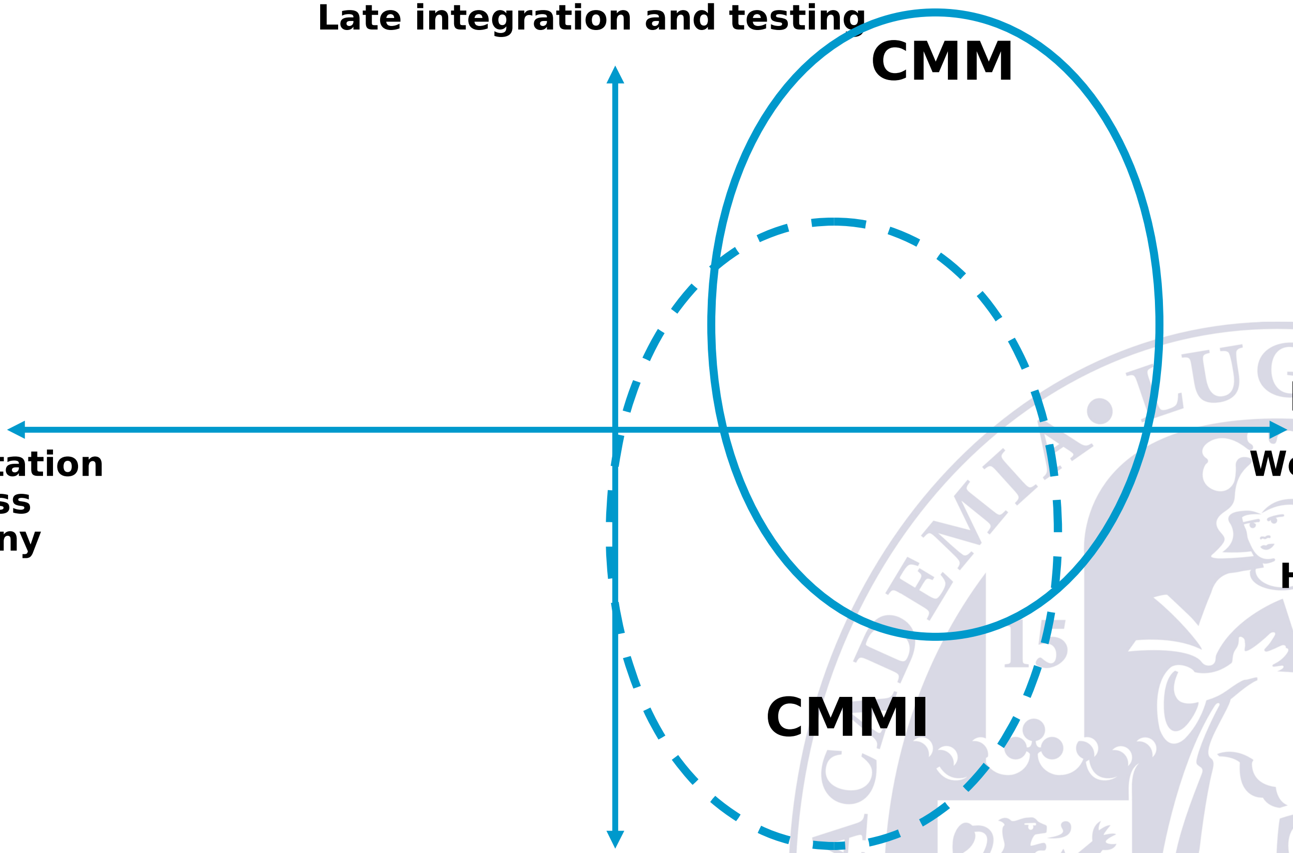
DoD Standards



SEI CMM and SEI CMMi

Waterfall

Few risk, sequential
Late integration and testing



Relaxed

Little documentation
Light process
Low ceremony

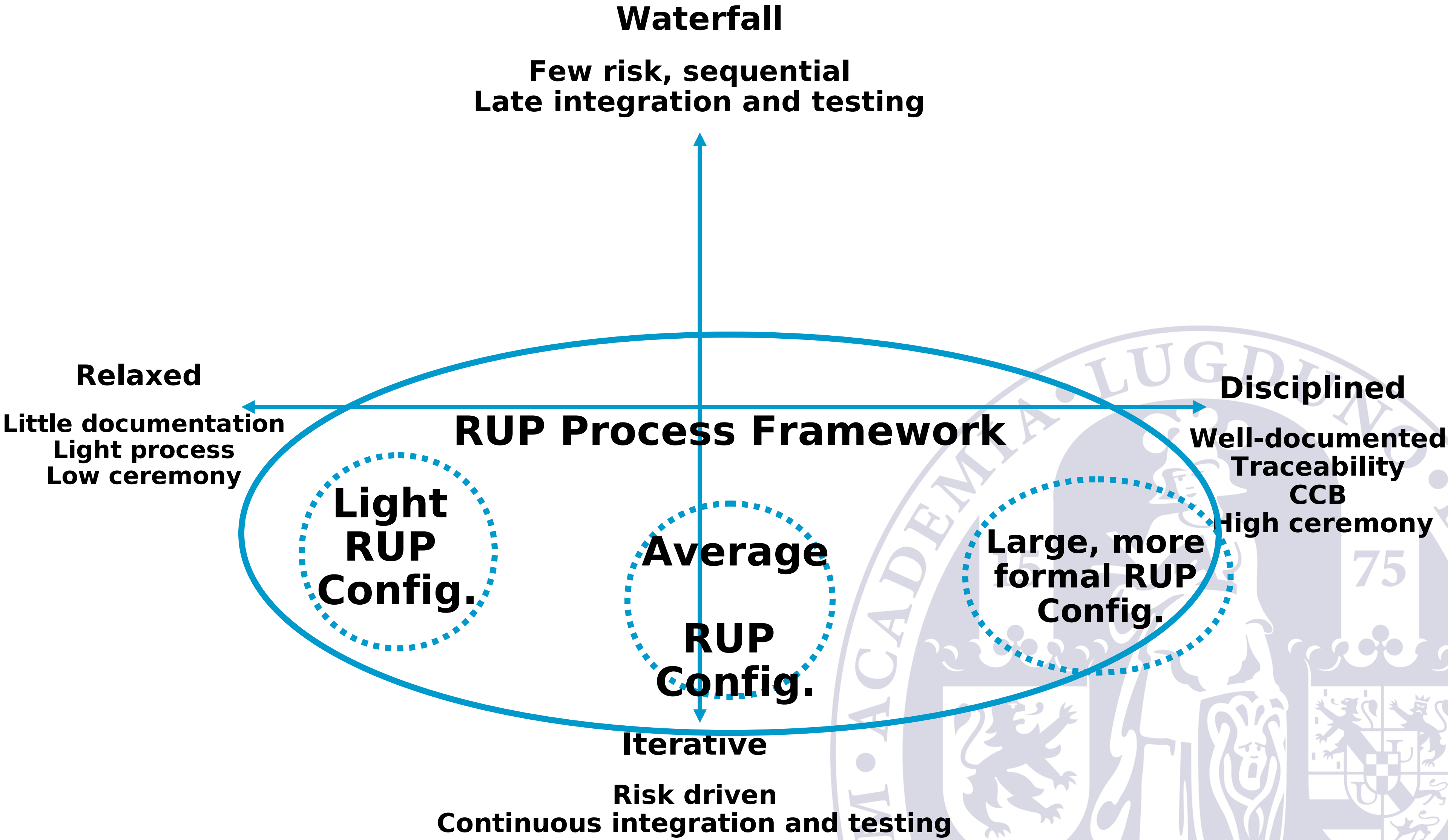
Disciplined

Well-documented
Traceability
CCB
High ceremony

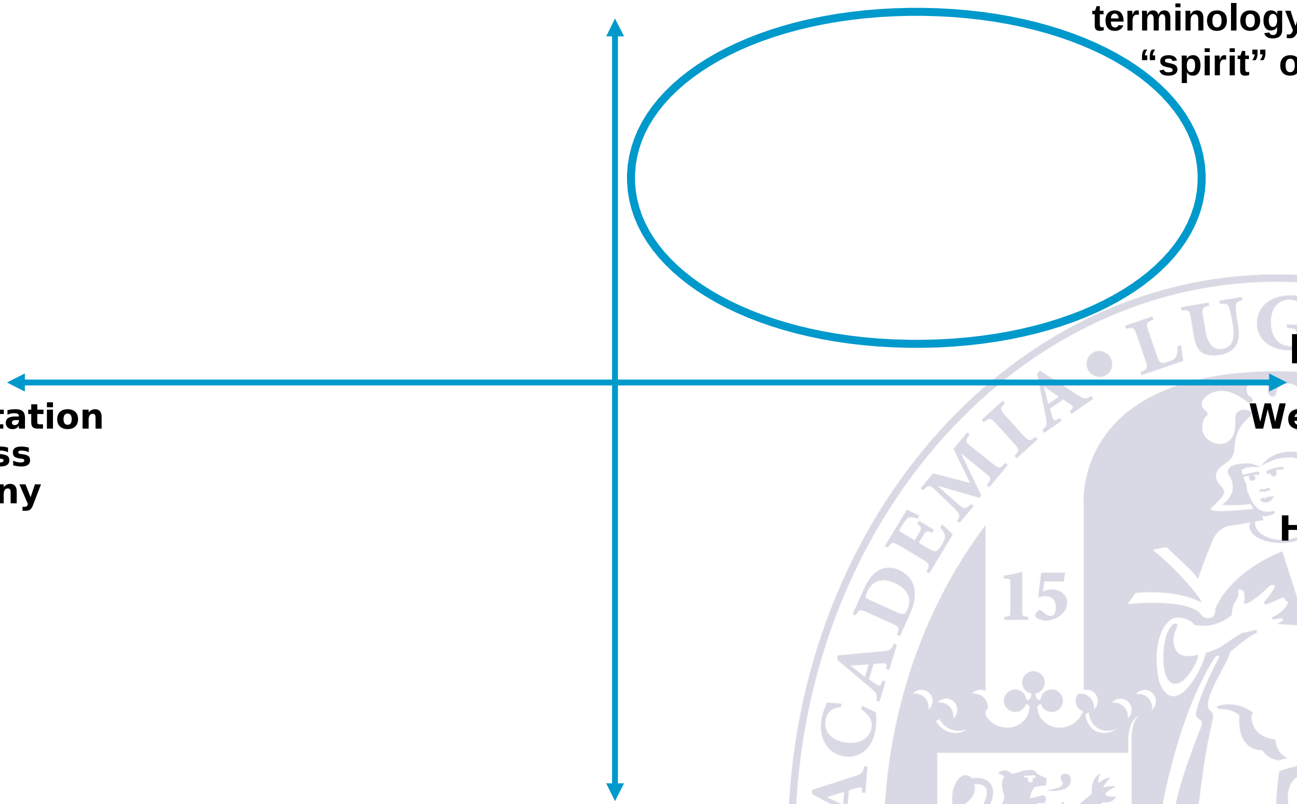
Iterative

Risk driven
Continuous integration and testing

Rational Unified Process Framework



Too Common RUP “Misusage”



Waterfall

Few risk, sequential
Late integration and testing

Adopting RUP is not about adopting a terminology, but the “spirit” of RUP

Relaxed

Little documentation
Light process
Low ceremony

Disciplined

Well-documented
Traceability
CCB
High ceremony

Iterative

Risk driven
Continuous integration and testing

Tailoring is key

RUP Configuration

RUP Development case

Drivers:

- Management complexity
- Technical complexity
- Risks, novelty
- Size, duration, size of team, distribution
- Business context

Objectives with Inception

1. Understand what to build
   - Vision, including who wants the system and it’s value
   - The scope of the system
   - Preliminary use-case model
1. Identify key requirements
   - Critical use cases and non-functional requirements
1. Determine at least one potential solution
   - Identify candidate architecture
1. Understand costs, schedule and risk
   - Business case, Software Development Plan, and Risk List
1. Understand what process to follow and tools to use
   - RUP configuration, development case, and customized tools

Objective 1: Understand What to Build

Agree on a high-level vision

Provide a “mile-wide, inch-deep” description

Detail key actors and use cases

Detail key non-functional requirements

Mile-Wide, Inch-Deep

1. Identify as many actors as you can
2. Associate each actors with use cases
3. Find additional actors for each use case
4. Briefly describe each actor (1-2 sentences) and use case (1-2 paragraphs)
5. Create a Glossary
6. Do a lifecycle analysis of key glossary items
7. Identify the most essential and critical use cases (<20% of use cases)
8. Detail key actors and use cases

Detail Key Actors and Use Cases

Done for 10-20% most critical use cases

Outline main flow of events

Identify alternative flow of events

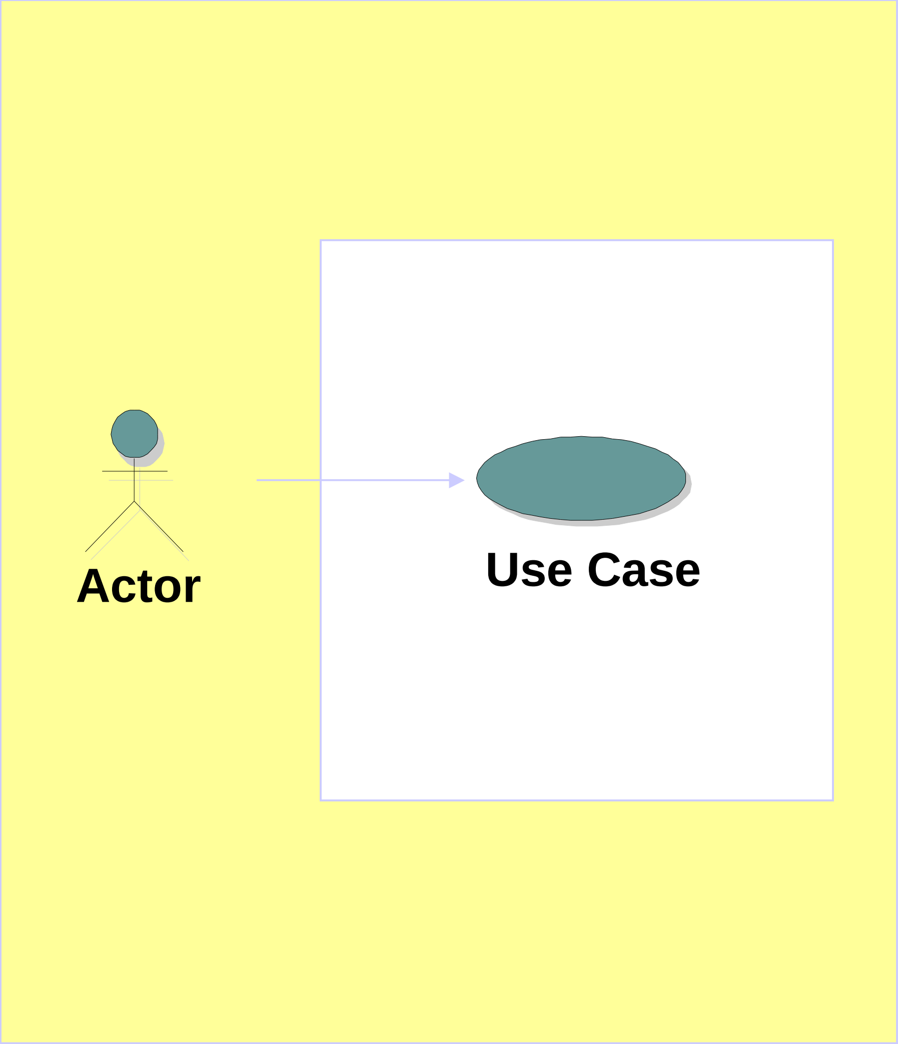
Complement textual descriptions with use-case prototypes

Time-box the writing, you never get “done”

Use less detail

- Small projects
- Analyst and developer

Major Concepts in the Use-Case Model



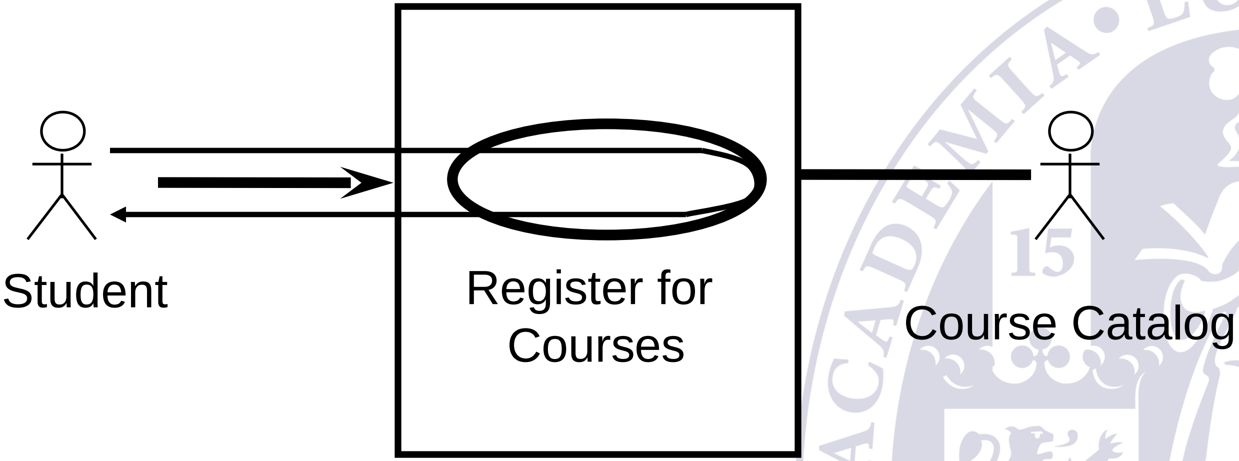
An **actor** represents a person or another system that interacts with the system.

A **use case** defines a sequence of actions a system performs that yields a result of observable value to an actor.

A Scenario - One Path Through a Use Case

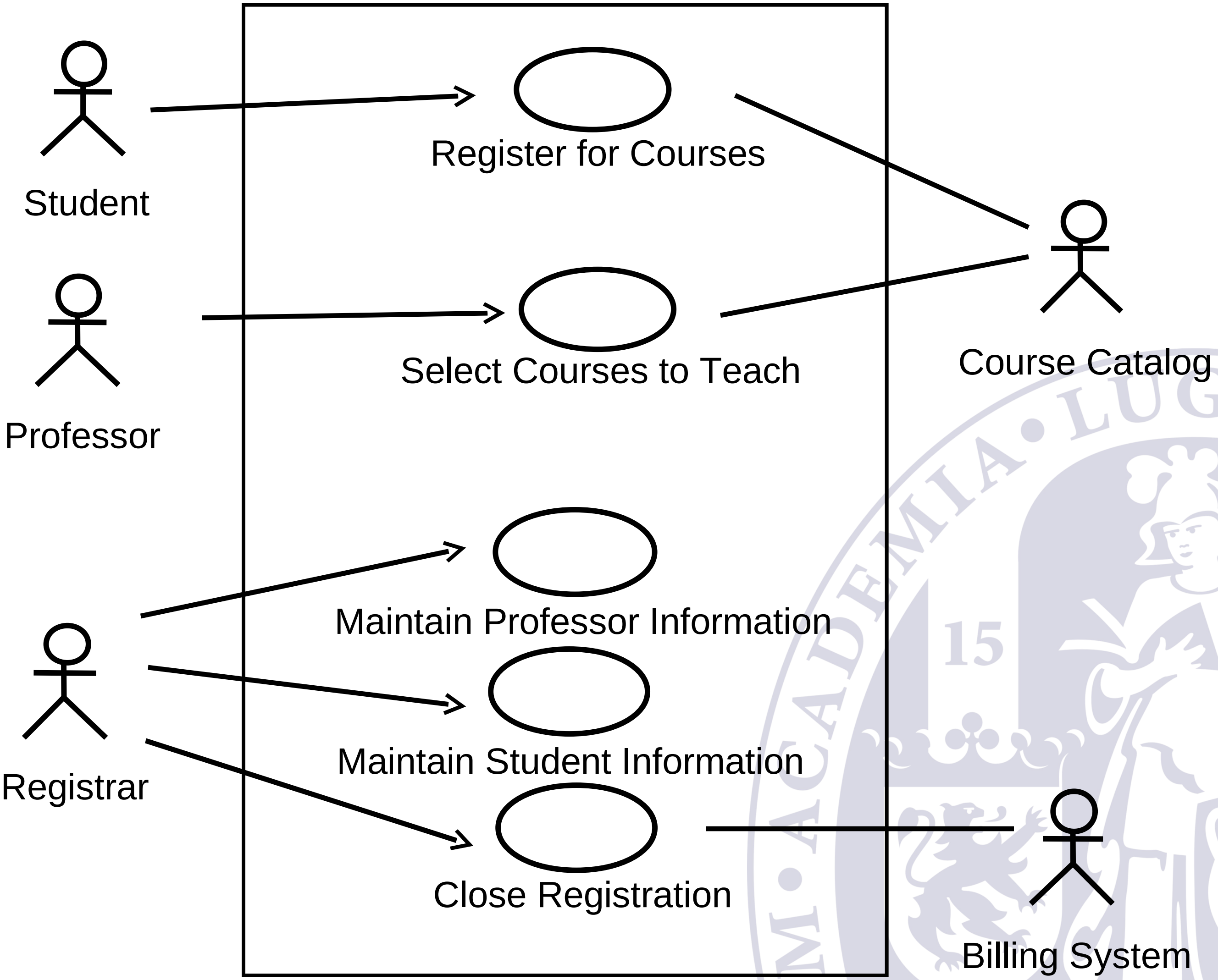
A use case can have many instances.

A **scenario** is a described use-case instance: a specific sequence of actions that illustrates behaviors of the system.



A Sample UML Diagram: Use Cases

A University Course Registration System



Objective 2: Identify Key System Functionality

Functionality is core the application

- Exercises key interfaces
- Deals with risks related to performance, redundancy, data security, …
- Example: “Check Out” for e-commerce application

Functionality *must* be delivered

- Captures the essence of the system
- Example: “Book conference room” for conference room booking system

Functionality covers an otherwise untouched area of the system

- May conceal unexpected technical difficulties

Objective 3: Determine at Least One Potential Solution

Should answer questions providing major impact on

- Can you build the application with a sensible amount of risk at a reasonable cost.
  - Have you built similar systems? With what architecture at what cost?
  - Will current architecture work, or will rework be required?
- Staffing profile – Will you be able to acquire personnel with the right competency to succeed?
- Required target environment – If it will have major impact on the cost profile of the overall project
- Required software components? Can they be purchased? At a reasonable cost?

Objective 4: Understand Cost, Schedule and Risk

Business case

- Ultimately answers the question: Should we fund the project?
- Cost
- Return of Investment

Software Development Plan

- Coarse project plan
- Resource needs

Objective 5: Decide on Process and Tools

- Decide on RUP configuration
  - Select plug-ins and process components
  - Produce process views
- Decide on development case
  - What artifacts should be produced with what formality
- Tool deployment plan
  - Required customizations
  - Reusable assets

Project Review: Lifecycle Objective Milestone

Do you have agreement on

- Scope definition
- Key requirements have been captured
- Cost and schedule estimates
- Priorities understood
- Development process and tools defined
- Initial risks identified and risk mitigation strategy exist

Objectives with Elaboration

1. Get a more detailed understanding of requirements
   - Move from 20% to 80% of requirements captured
1. Design, implement, validate and baseline the architecture
   - Make critical design decisions; buy vs. build, patterns, ..
   - Baseline a skeleton structure of your system
   - Perform initial load and performance test
1. Mitigate essential risks, and produce more accurate schedule and cost estimates
   - You now know what to build, and have reduced risk => More accurate schedule
1. Fine-tune and deploy development environment
   - Harvest experiences from Inception
   - Rollout development environment

Sample Elaboration Profile: Multiple Iterations

Iteration 1

Design, implement and test a small number of critical scenarios to outline architecture and required patterns

Identify, implement and test a small set of architectural patterns

Do a preliminary logical database design

Detail flow of events of ~half of target UCs, with most important UCs first

Do sufficient testing to ensure key risks have been mitigated

Iteration 2

Fix whatever did not work in previous iteration

Design, implement and test remaining architecturally significant scenarios (for the <20% of key UCs)

Address key risks by outlining and implementing concurrency, processes, threads, and physical distribution

Focus on performance and load testing, and interface testing

Identify, implement and test architectural patterns

Design, implement and test preliminary version of database

Detail the remaining 50% of target Ucs

Refine and test your architecture so you can baseline it

Objective 1: Get a More Detailed Understanding of Requirements

By end of elaboration

- Detail ~80% of use cases
- Produce prototypes of graphically oriented use cases (at least mock-up screens)
- Walk through use cases with stakeholders
- For use case with partial implementations—demo
- Detail non-functional requirements (all that have an impact on the architecture)
- Time box!!! You will never be done, and you can fix issues in later iterations

What is not done

- Use cases with no or very limited associated risk (If you have done one “print” use case, do you need to do more?)
- Use cases that are expected to be volatile, and have little impact on end solution or stakeholder satisfaction

Objective 2: Design, Implement, Validate and Baseline the Architecture

What is included in “architecture”?

The most important building blocks of the systems

- Build, buy, or reuse?

Interaction between these building blocks to provide key scenarios

- Required to verify the architecture

Run-time architecture

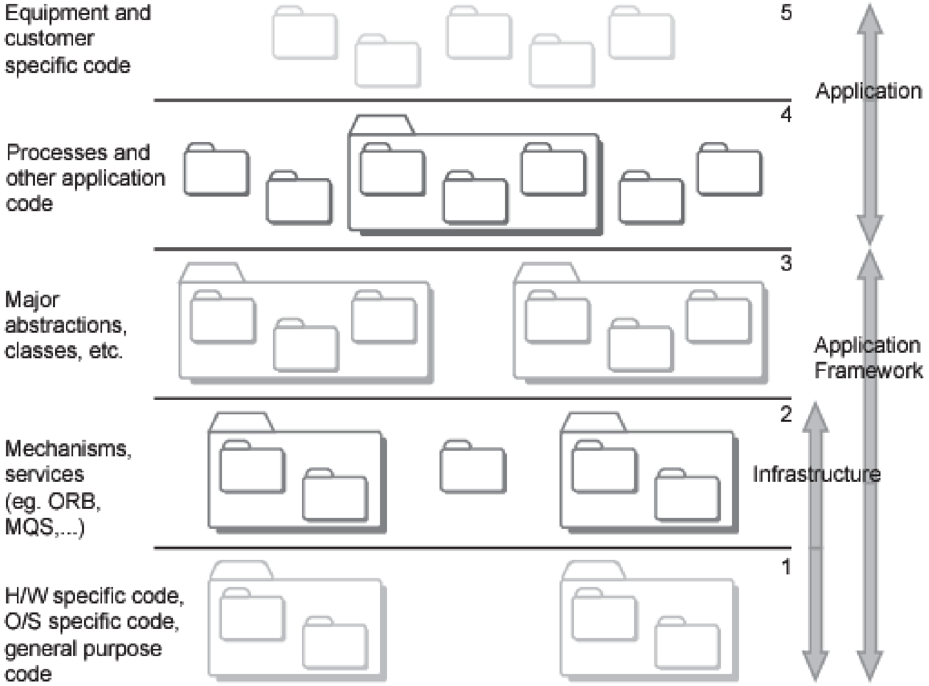
- Processes, threads, nodes, …

Architectural patterns

- Dealing with persistency, inter-process communication, recovery, authentication, garbage collection, …

Test framework allowing testing of key capabilities

- Performance, scalability, reliability, load, protocols, and other key non-functional requirements


Equipment and customer specific code
5
Application
4
Processes and other application code
3
Major abstractions, classes, etc.
Application Framework
2
Mechanisms, services (eg. ORB, MQS,...)
Infrastructure
1
H/W specific code, O/S specific code, general purpose code

How Do You Design, Implement, Validate and Baseline the Architecture

Use architecturally significant use cases / scenarios

- Provides a focus for implementation, aiming at driving out key technical risks

Identify required components to implement the use cases / scenarios

- Build, buy, or reuse?

Design and implement “just enough” to support key scenarios

- A lot of stubs
- Only 10-20% of overall code is implemented

Integrate and verify that key scenarios works as expected

- Typically only one or two scenarios for key use cases supported
- No or few “alternative flow of events” supported

Test the quality attributes of the architecture

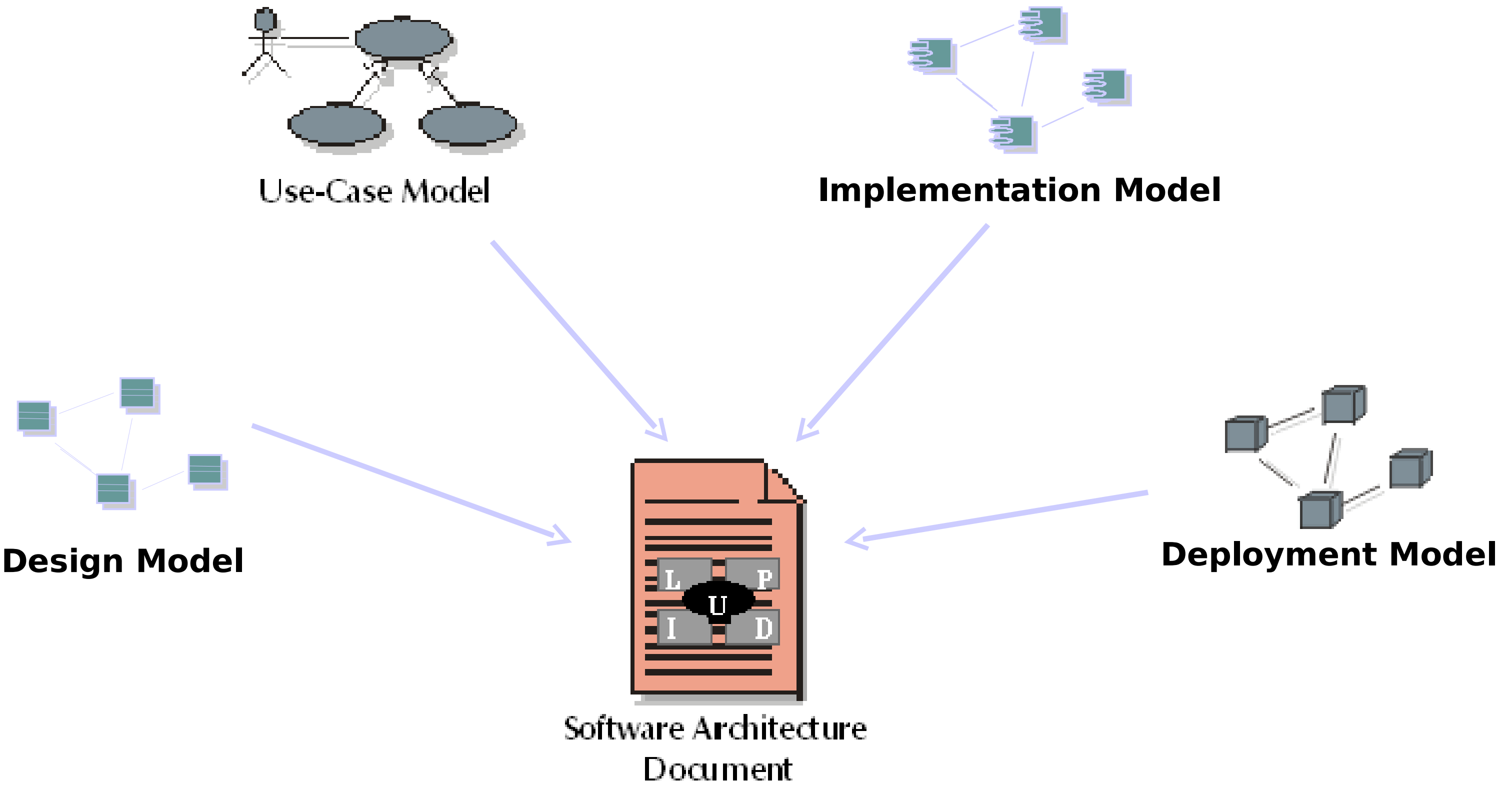
- Performance, scalability, reliability, load, …

Architecture Description

- Software Architecture Document
  - Represents comprehensive overview of the architecture of the software system
  - Includes
    - Architectural Views
    - Goals and constraints
      - Requirements that architecture must support
      - Technical constraints
      - Change cases
    - Size and performance characteristics
    - Quality, extensibility, and portability targets

Architecture Description (cont.)

Views are pulled from other artifacts



Use-Case Model

Implementation Model

Design Model

Deployment Model

Software Architecture Document

Objective 3: Mitigate Essential Risks, and Produce More Accurate Schedule and Cost Estimates

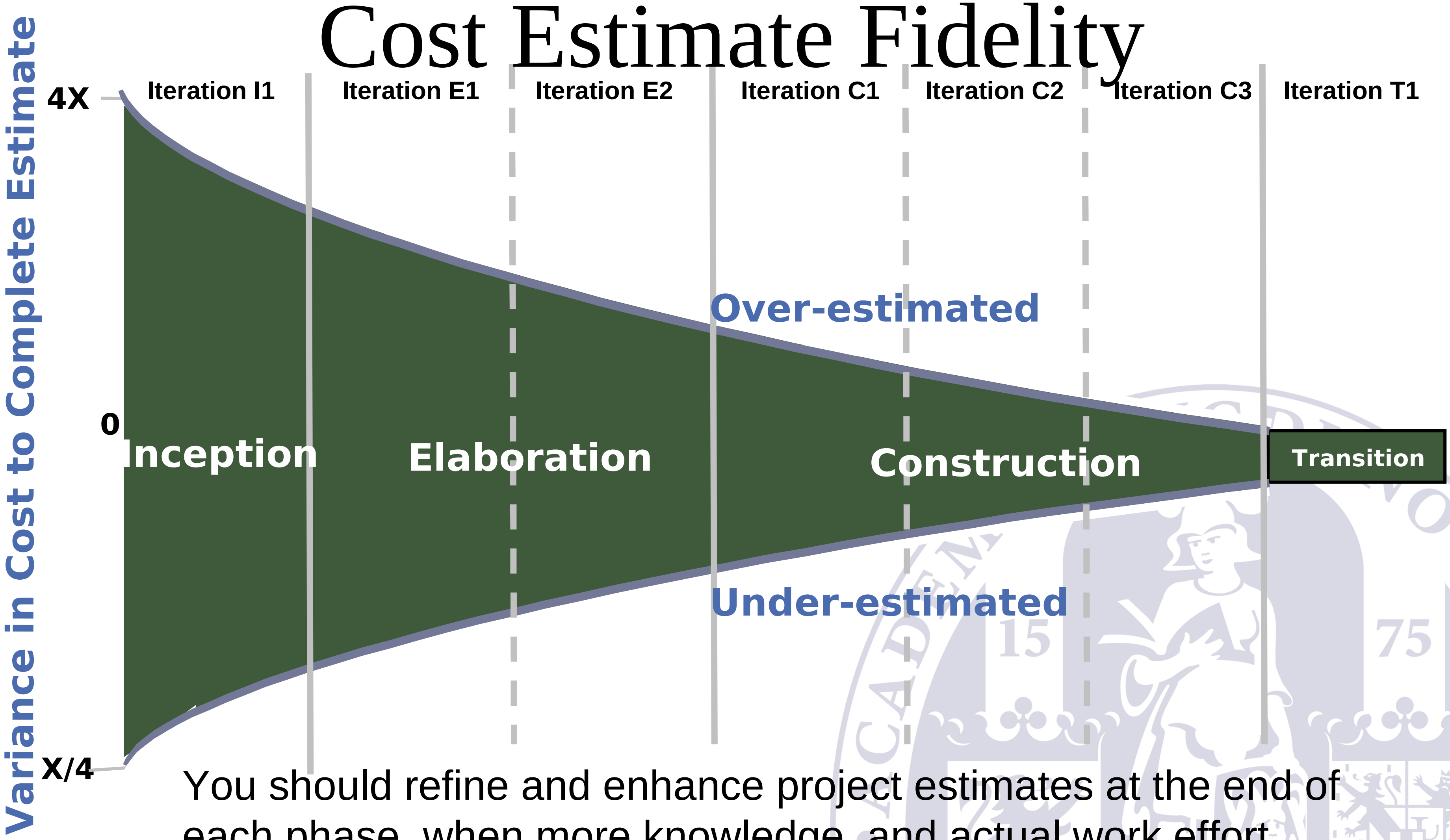
Most key risks addressed

- Technical risks by implementing and testing the architecture
- Business risks by implementing and testing key functionality
- Team- and tool-oriented risks by having the team going through the full software lifecycle implementing real code, using the tools at hands

Schedule and cost estimates can be radically improved since we

- Have mitigated key risks
- Understand a vast majority of the requirements
- Understand which building blocks needs to be implemented and tested
- Understand which building blocks can be acquired, and to what cost
- Understand how effective our team is

Cost Estimate Fidelity



You should refine and enhance project estimates at the end of each phase, when more knowledge, and actual work effort experience is available.

Leiden University

Objective 4: Fine-tune and Deploy Development Environment

Fine-tune your development case based on the experiences so far

Do customizations and improvements of the tool environment as required

Make it easy for all team members to find and use reusable assets, including the architectural patterns you developed in Elaboration

Do training of your team and deploy tools

Project Review: Lifecycle Architecture Milestone

Are the product Vision and requirements stable?

Is the architecture stable?

Are the key approaches to be used in testing and evaluation proven?

Have testing and evaluation of executable prototypes demonstrated that the major risk elements have been addressed and resolved?

Are the iteration plans for Construction of sufficient detail and fidelity to allow the work to proceed?

Are the iteration plans for the Construction phase supported by

credible estimates?

Do all stakeholders agree that the current Vision, as defined in the Vision Document, can be met if the current plan is executed to develop the complete system in the context of the current architecture?

Are actual resource expenditures versus planned expenditures acceptable?

What Did You Achieve In Elaboration

You moved from a high-level understanding of key requirements to a detailed understanding of roughly 80 percent of the requirements

You moved from a conceptual architecture to a baselined, executable architecture

You mitigated key risks and produced more accurate schedule/cost estimates for the remaining lifecycle phases

You decided whether to move ahead with the project, cancel it, or radically change it

You refined the development case and put the development environment in place

You laid the groundwork for scaling up the project with a minimum of financial, business, and technical risks

But What is Left to Do?

Most use cases have not been implemented at all

- The ones that have been implemented, have only been partially implemented

Subsystems are primarily shells, with only interfaces and the most critical code implemented

- Roughly only 10-15% of overall code has been implemented

Even though a majority of technical risks have been mitigated, new risks will keep popping up…

The average project spends roughly 50% of duration, and 65% of overall effort, in the Construction phase.

Objectives with Construction

1. Minimize development costs and achieve some degree of parallelism in the work of the development teams
   - Optimize resources and avoid unnecessary scrap and rework
1. Iteratively develop a complete product that is ready to transition to its user community
   - Develop the first operational version of the system (beta release)
   - Determine whether the software, the sites, and the users are all ready for the application to be deployed.

Sample Construction Profile: Multiple Iterations

	Elaboration (End)	Construction 1	Construction 2	Construction 3
Require-ments	▪ 15 UCs identified ▪ 8 detailed ▪ 4 some depth ▪ 3 briefly	▪ 12 UCs detailed ▪ 3 some depth	▪ 14 UCs detailed ▪ 1 removed (scope reduction)	▪ 14 UCs detailed
Comp-onents	▪ 18 main components ▪ 4 -> 50% done ▪ 10 –> interfaces ▪ Lower arch. layers almost done ▪ All code unit tested	▪ 19 main components ▪ 3 almost done ▪ 8 -> 50% ▪ 6 -> interfaces ▪ Lower arch. layers almost done ▪ All code unit tested	▪ 18 main components ▪ 10 almost done ▪ 8 -> 50% ▪ All code unit tested	▪ 18 main components ▪ All almost done, minor tuning left ▪ Beta release, all functionality implemented
Tests	▪ Initial load & performance test of arch. ▪ 4 UCs functionally tested	▪ Continued load & perf. testing ▪ Functional UC testing as completed	▪ Continued load & perf. testing ▪ Functional UC testing as completed	▪ Continued load & perf. testing ▪ Functional UC testing as completed

Objective 1: Minimize Development Costs and Achieve Some Degree of Parallelism

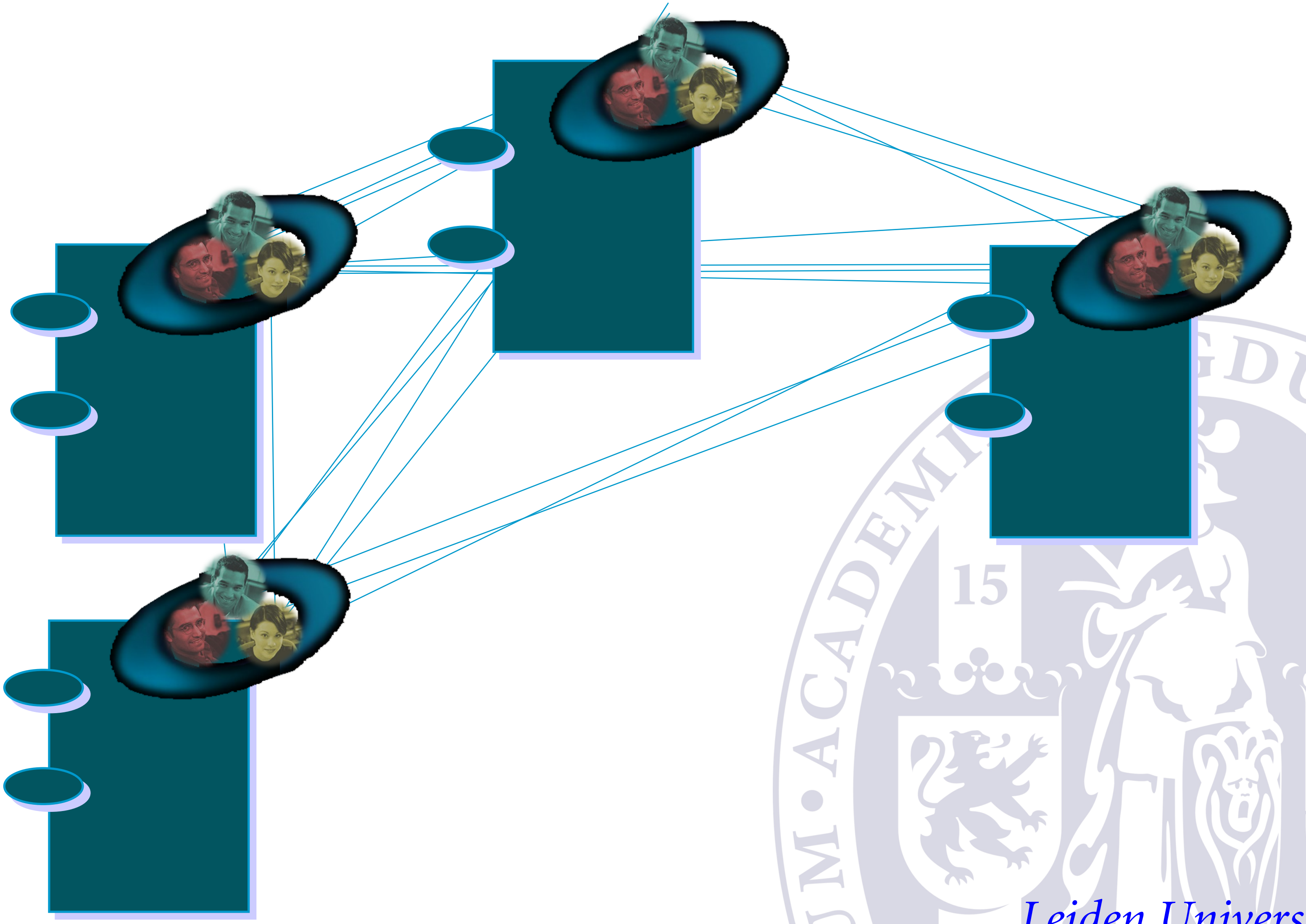
Organize your team around software architecture

Implement effective configuration management

Enforce the architecture

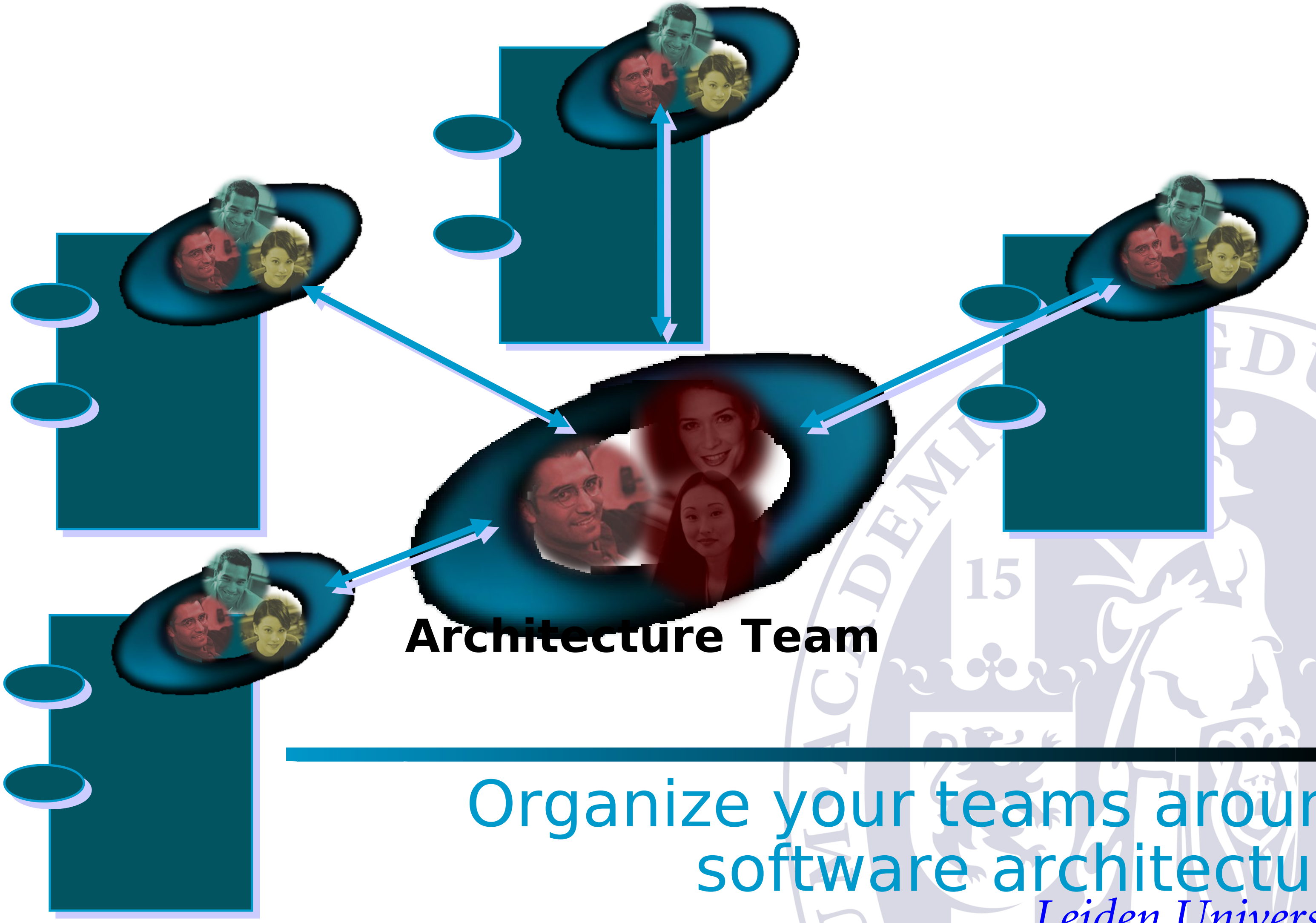
Ensure continual progress

Improve Project Efficiencies: *Organization*



Leiden University

Improve Project Efficiencies: *Organization*



Organize your teams around software architecture

Leiden University

Implement Configuration Management

Iterative development characteristics:

Frequent builds => Track multiple versions of files

Parallel work on multiple streams => merge success solutions into main stream

Difficult to understand when and where defects are introduction => Enable back tracking to functioning version and to understand when defect was introduced

For larger teams:

Isolate the work of one team member from the rest of team when desired

Control who is allowed to do what changes

High end configuration management systems can automate all of the above

Enforce the Architecture

Leverage patterns developed during Elaboration

- Train your team and produce guidelines on what is available and how to use it
- Do you need to produce a reuse / pattern library for easy access?
- Arrange with reviews to ensure architectural integrity

Manage changes to interfaces

- Configuration management support
- How will you communicate changes to concerned parties?

Enforcement of the architecture needs to be formalized for large or distributed teams

Ensure Continual Progress

Leverage iterations to create short term goals and a sense of urgency

- Avoid hockey stick

Create cross-functional teams with a joint mission

- Quick daily meeting (SCRUM)

Set clear achievable goals for developers

- Break down iteration objective of an iteration to a set of 1-week goals, if possible

Continually demonstrate and test code

- Measure progress through demonstration and testing, not through “status reports”

Force continuous integration

- Daily builds if possible

Objective 2: Iteratively Develop a Complete Product

Describe the remaining use cases and other requirements

Fill in the design

Design the database

Implement and unit-test code

Do integration and system testing

Early deployment and feedback loops

Prepare for beta deployment

Prepare for final deployment

Considerations for Beta Deployment

How many beta customers do you need?

What type of feedback do you need?

How long beta program do you need to get required feedback?

What do you need to do to ensure beta program is valuable to participants?

- What’s in it for them?

What do you need to do to ensure required feedback?

- Training?
- Face-to-face time with beta users?
- Balance between independent use (to ensure realistic usage) and handholding (to ensure correct usage, or usage at all)

Is data conversion required?

Do you need to run current systems in parallel?

A well-thought plan needs to be in place, including identified participants, at the day of the beta release

Considerations for Final Deployment

Training material

User documentation

Prepare deployment site(s)

- Hardware
- Software
- Training
- Facilities, …

Database conversations

Budget and resource coordination

- Acquisitions and hiring
- Transition costs, such as training or running parallel systems…

Even though final deployment happens at end of Transition, you often need to prepare in Construction

Conclusions: Construction

You develop software cost-effectively by taking advantage of the architectural baseline from Elaboration

You are able to scale up the project to include more team members

You build and assess several internal releases

You move from an executable system to the first operational version of your system

Objectives with Transition

1. Beta test to validate that user expectations are met.
   - bug fixing and making enhancements for performance and usability.
1. Train users and maintainers to achieve user self-reliability.
   - Are adopting organization(s) qualified to use the system
1. Prepare deployment site and convert operational databases.
   - Purchase new hardware, add space for new hardware, and data migration.
1. Prepare for launch-packaging, production, and marketing rollout; release to distribution and sales forces; field personnel training.
   - Especially relevant for commercial products.
1. Achieve stakeholder concurrence that deployment baselines are complete and consistent with the evaluation criteria of the vision.
2. Improve future project performance through lessons learned.
   - Document lessons learned and improve process and tool environment.

Objective 1: Beta Test to Validate That User Expectations Are Met

Capturing, Analyzing, and Implementing Change Requests

Transition Testing

- Continued test design and implementation to support ongoing development.
- Regression testing, which will require variable effort and resources, depending on the chosen approach; for example, retest everything or retest to an operational profile.
- Acceptance testing, which may not require the development of new tests.

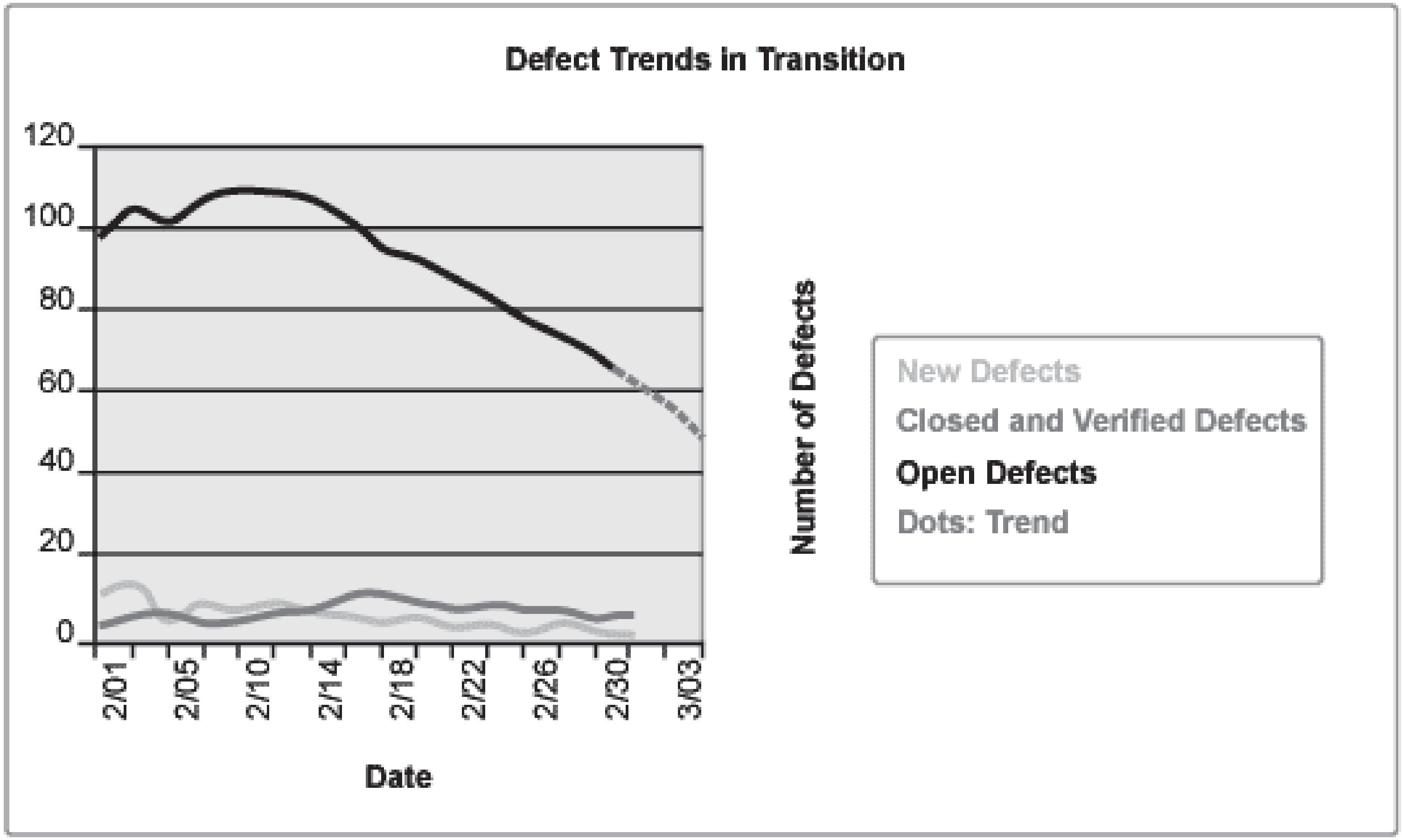
Patch Releases and Additional Beta Releases

Metrics for Understanding When Transition Will Be Complete

- Defect metrics
- Test metrics

Example: Defect Metrics

Trend analysis often provides reasonably accurate assessment of when you can complete the project



Objective 4: Prepare for Launch: Packaging, Production, and Marketing Rollout

Packaging, Bill of Materials, and Production

Marketing Rollout

- Core Message Platform (CMP).
  - A one- to two-page description of the product, its positioning, and key features and benefits.
  - Used as a baseline for all internal and external communication related to the product.
- Customer-consumable collateral.
  - Data sheets, whitepapers, technical papers, Web site, prerecorded demos, demo scripts, ...
- Sales support material.
  - Sales presentations, technical presentations, field training material, fact sheets, positioning papers, competitive write-ups, references, success stories, and so on.
- Launch material.
  - Press releases, press kits, analyst briefings, and internal newsletters.

Conclusions: Transition

You performed one or more beta tests of the new system with a small set of actual users and fine-tuned it as necessary.

You trained users and maintainers to make them self-reliant.

You prepared the site for deployment, converted operational databases, and took other measures required to operate the new system successfully.

You launched the system with attention to packaging and production; rollout to marketing, distribution, and sales forces; and field personnel training. This is specifically a focus for commercial products.

You achieved stakeholder concurrence that deployment baselines are complete and consistent with the evaluation criteria of the vision.

You analyzed and used lessons learned to improve future project performance.

Common Pitfalls in Inception

Too much formality / too many artifacts

- Only produce the artifacts that add value, minimize formality if possible
- When in doubt of value, don’t do it

Analysis Paralysis

- You can improve upon things later on – move on
- Focus on objectives with Inception
- Do NOT describe all requirements in detail

Too long initial iteration

- Cut scope rapidly
- You fail with first iteration, project likely to fail

Common Pitfalls in Elaboration

Functional, Specialized Organization

- Teams of generalists and multitasking experts
- No place for “I only do <X>” mentality

Save the tricky part for later

- Attack risks early, or they attack you
- Hard on you now, but makes life easier later

No implementation and validation of architecture

- You cannot get the architecture right or address major risks without implementing and testing the architecture

No willingness to change things

- Change enables improvement

Common Pitfalls in Construction

Basing work on unstable architecture

- Major rework and integration issues
- High price to pay for insufficient work in Elaboration

Reinventing solutions to common problems

- Were architectural mechanisms (patterns) developed in Elaboration and communicated to everybody?

Continuous integration not happening

- Daily or weekly build minimizes rework

Testing not initiated until end of construction

- You are unlikely to meet deadlines
- Beta may be of too low quality to offer value

Common Pitfalls in Transition

Not enough beta users

- Did you have beta customers lined up and prepared at end of Construction?

Not all functionality beta tested

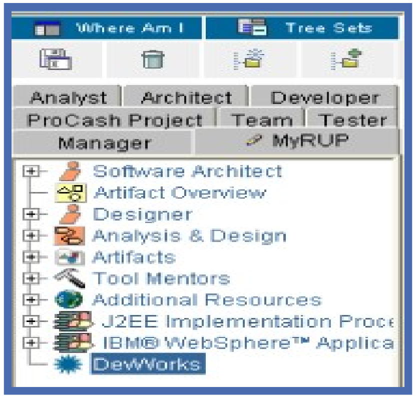
- Did you include the functionality in the beta release?
- Was it usable? (Ease of use, performance, documented, …)

Customer not happy with delivered functionality

- Was acceptance criteria approved by customer?
- Did you involve customer throughout the project?

Tools for Tailoring RUP - A Roadmap

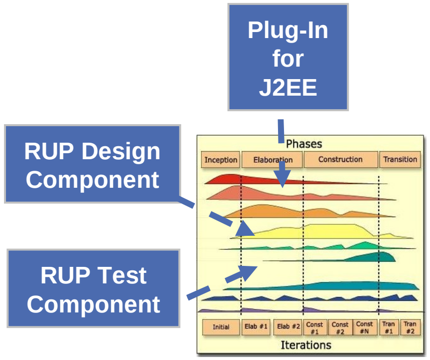
MyRUP Facility



Practitioner

Personalize your view of the RUP Web site on your desktop

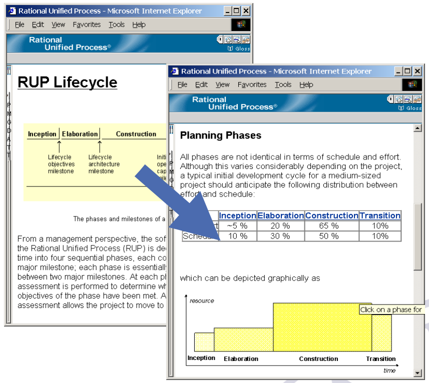
RUP Builder



Project Manager

Publish a new RUP version (plug-in) based on existing process components and plug-ins

RUP Organizer


Rational Unified Process - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Rational Unified Process®
Gloss
Inception
Elaboration
Construction
Lifecycle objectives milestone
Lifecycle architecture milestone
resource
time
Click on a phase for

RUP Lifecycle

The phases and milestones of a

From a management perspective, the sof the Rational Unified Process (RUP) is de time into four sequential phases, each co major milestone; each phase is essentially between two major milestones. At each ph assessment is performed to determine wh objectives of the phase have been met. A assessment allows the project to move to

Planning Phases

All phases are not identical in terms of schedule and effort. Although this varies considerably depending on the project, a typical initial development cycle for a medium-sized project should anticipate the following distribution between effort and schedule:

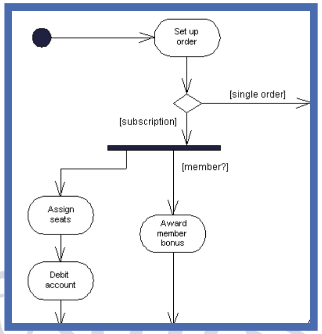
	Inception	Elaboration	Construction	Transition
rt	~5 %	20 %	65 %	10%
Schedu	10 %	30 %	50 %	10%

which can be depicted graphically as

Content Developer

Add/Modify/Remove content pages associated to process elements

RUP Modeler



Process Engineer

Add/Modify/Remove process elements

Rational Process Workbench

Leiden University

Practitioner: *MyRUP*

Practitioner:
"I want to easily find the info I need"

Personalized views

- Role-based and personalized views into your project's process
- Add links to external and internal resources

Project Web and Extended help integrated with RUP browser

Closer integration with RDN

- Hotlinks to RDN, etc. from MyRUP
- Seamless search across RUP and RDN

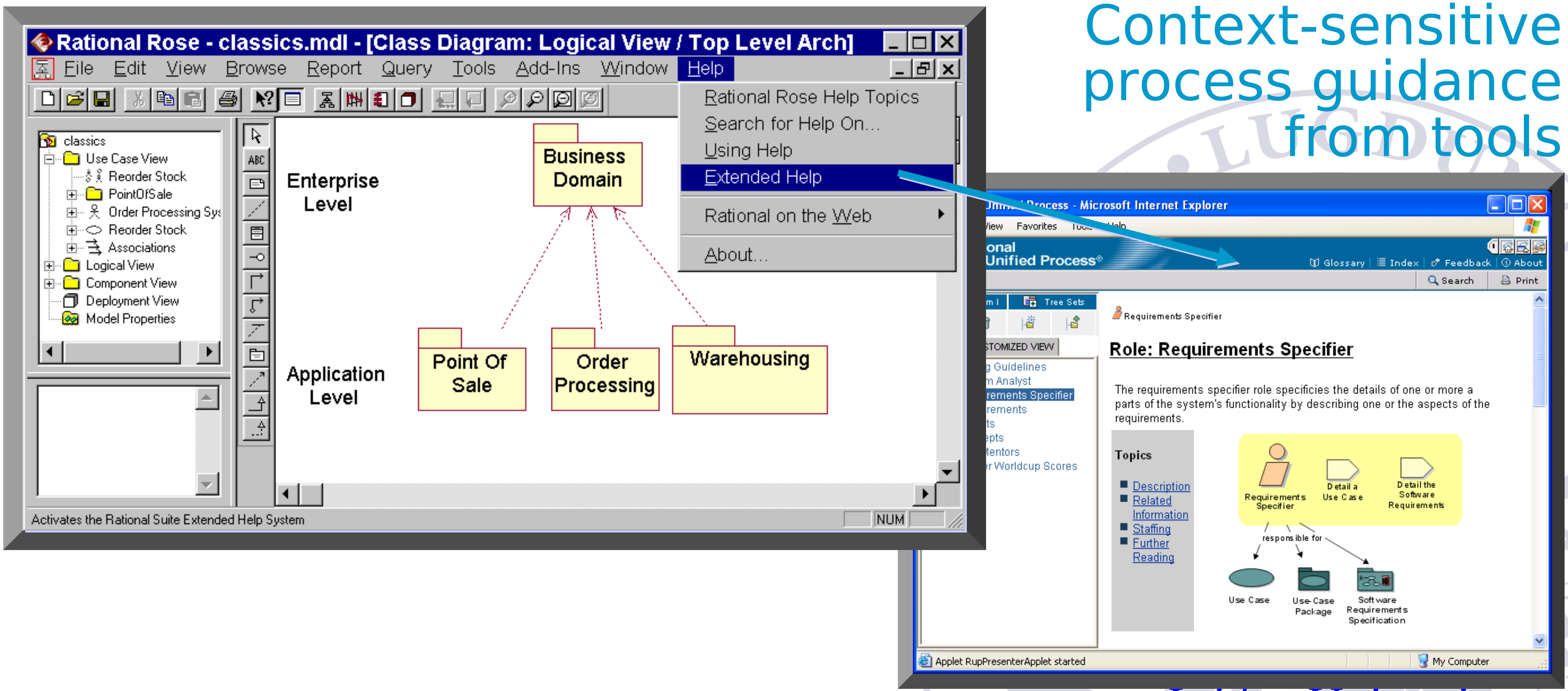
Assets available through MyRUP

Easy access through clean workspace

RUP: Integrated with Tools

- **Tool mentors**: Web-based assistance for tool use
- **Extended Help**: Process guidance from within any tool

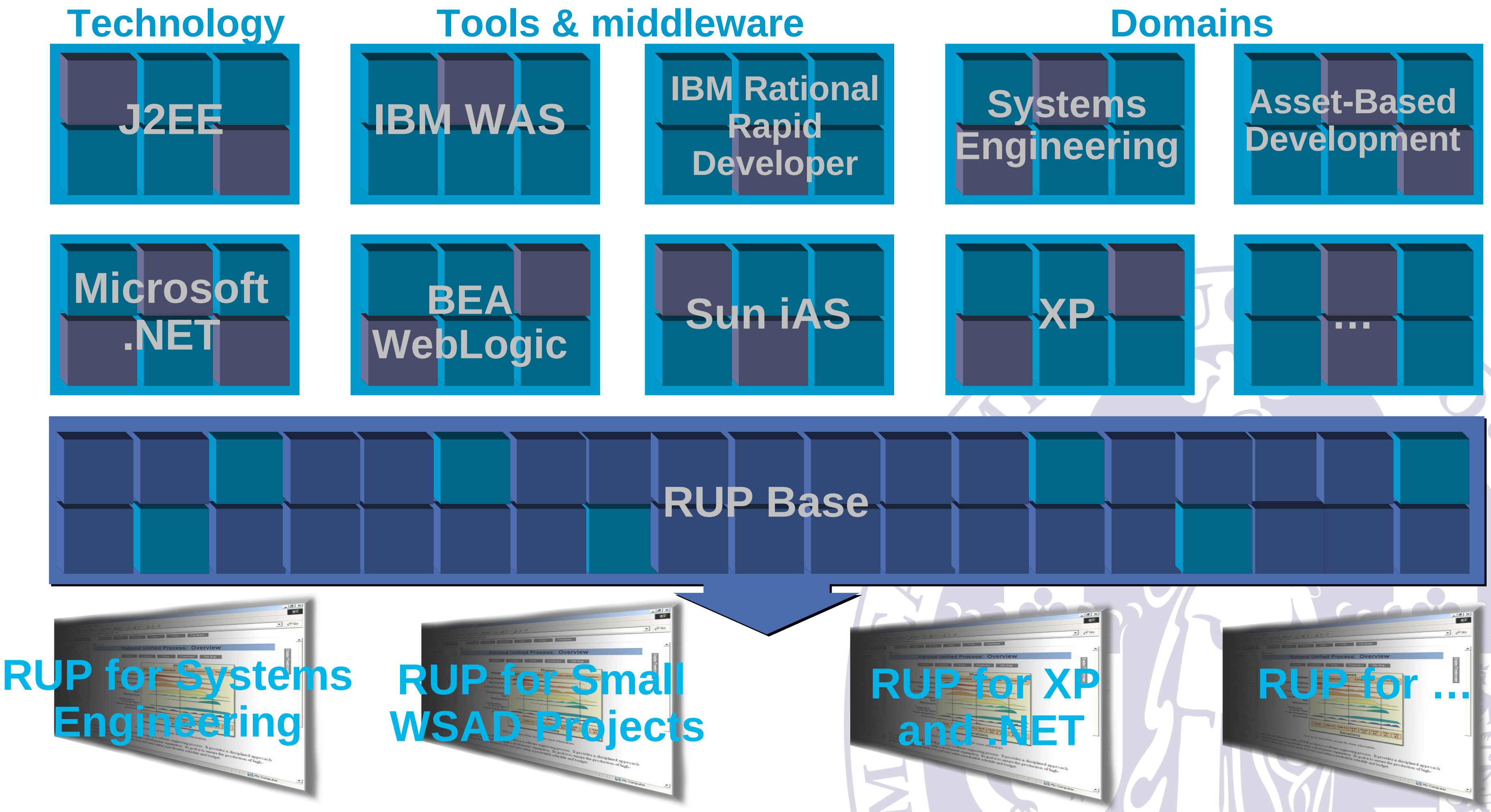
Context-sensitive process guidance from tools



Role: Requirements Specifier

The requirements specifier role specificies the details of one or more a parts of the system's functionality by describing one or the aspects of the requirements.

RUP: Highly Configurable



Technology

Tools & middleware

Domains

J2EE

IBM WAS

IBM Rational Rapid Developer

Systems Engineering

Asset-Based Development

Microsoft .NET

BEA WebLogic

Sun iAS

XP

…

RUP Base

RUP for Systems Engineering

RUP for Small WSAD Projects

RUP for XP and .NET

RUP for …

Configuration Tools: *RUP Builder*

Project Manager: "I need to adapt RUP to my project needs"

Right-size your process through fine-granular process selection

- +100 selectable units

Small, medium, and large project configurations available as starting point

Produce role-based views

Easy access to latest content through RUP plug-in exchange

Assemble the right process

Configuration Tools: *Extended RUP Plug-In Library*

Now more than 21 Plug-Ins to choose from

New Plug-Ins	Updated Plug-Ins
RUP Plug-In for System Engineering	RUP Plug-In for Extreme Programming (XP)
RUP Plug-In for Creative Web Design	Java™ 2 Enterprise Edition (J2EE) Plug-In
RUP Plug-In for Asset-Based Development	Microsoft .NET Plug-In
RUP Plug-In for IBM Rational Rapid Developer	Plug-In for IBM WebSphere Application Server
RUP Plug-In for Sun iAS	BEA WebLogic™ Server
	RUP Plug-In for User-Experience Modeling

Demo Steps

Anna
roject Manager

Anna defines a configuration for her project using RUP Builder

- Select what plug-ins to use
- Do fine-granular selection of process
- Produce role-based views

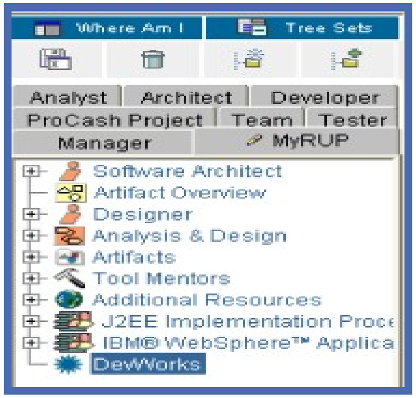
Per
Practitioner

Per finds guidance on how to work using MyRUP

- Personalize my tree browser
- Find relevant information
- Find elaborate tool guidance

Tools for Tailoring RUP - A Roadmap

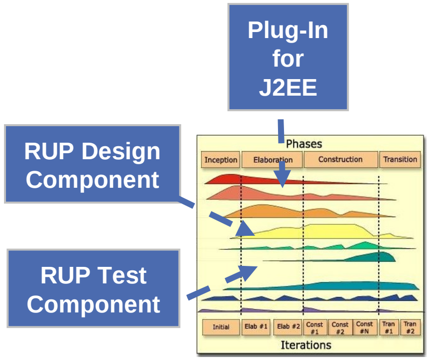
MyRUP Facility



Practitioner

Personalize your view of the RUP Web site on your desktop

RUP Builder


Plug-In for J2EE

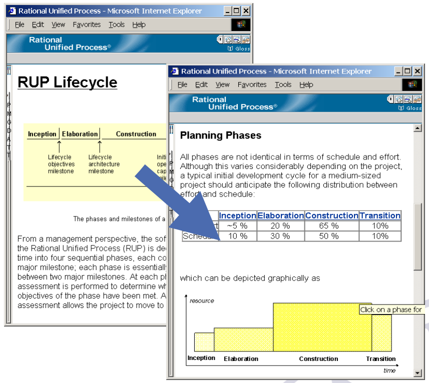
RUP Design Component

RUP Test Component

Project Manager

Publish a new RUP version (plug-in) based on existing process components and plug-ins

RUP Organizer


Rational Unified Process - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Rational Unified Process®
Gloss

RUP Lifecycle

Inception
Elaboration
Construction
Lifecycle objectives milestone
Lifecycle architecture milestone

The phases and milestones of a

From a management perspective, the sof the Rational Unified Process (RUP) is de time into four sequential phases, each co major milestone; each phase is essentially between two major milestones. At each pl assessment is performed to determine wh objectives of the phase have been met. A assessment allows the project to move to

Rational Unified Process - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Rational Unified Process®
Gloss

Planning Phases

All phases are not identical in terms of schedule and effort. Although this varies considerably depending on the project, a typical initial development cycle for a medium-sized project should anticipate the following distribution between effort and schedule:

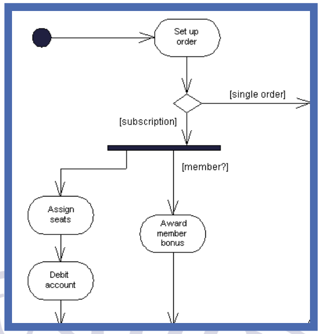
	Inception	Elaboration	Construction	Transition
t	~5 %	20 %	65 %	10%
Schedu	10 %	30 %	50 %	10%

which can be depicted graphically as

Content Developer

Add/Modify/Remove content pages associated to process elements

RUP Modeler



Process Engineer

Add/Modify/Remove process elements

Rational Process Workbench

Tools for Tailoring RUP - *RUP Organizer*

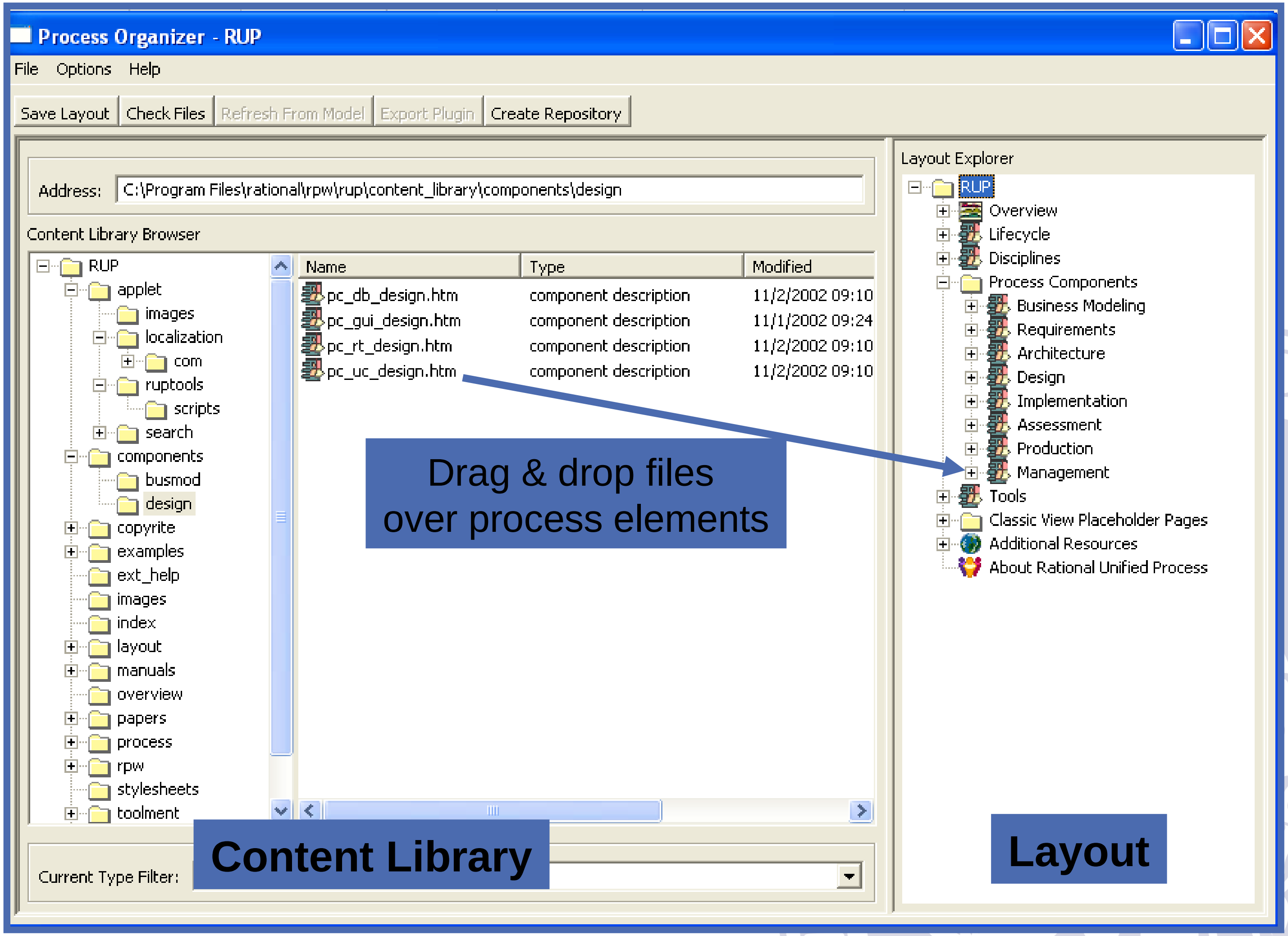
Process Authoring Tool used to Tailor the Process Content

- Used to add/modify/remove content pages associated to process elements
- Drag & drop new or changed files onto existing model elements
- Commonly used to package reusable company/project assets
  - Guidelines, Concepts, White Papers,Templates, Examples, etc.
- Many customizations can be done using only RUP Organizer (e.g. internationalization).

RUP Content Developer: "I need to modify or add content"

Simplify content changes

Tools for Tailoring RUP - *RUP Organizer*



Address: C:\Program Files\rational\rpw\rup\content_library\components\design

Name	Type	Modified
pc_db_design.htm	component description	11/2/2002 09:10
pc_gui_design.htm	component description	11/1/2002 09:24
pc_rt_design.htm	component description	11/2/2002 09:10
pc_uc_design.htm	component description	11/2/2002 09:10

Drag & drop files
over process elements

Content Library

Layout

Leiden University

Tools for Tailoring RUP - *RUP Modeler*

Process Authoring Tool used to Tailor the Process Structure

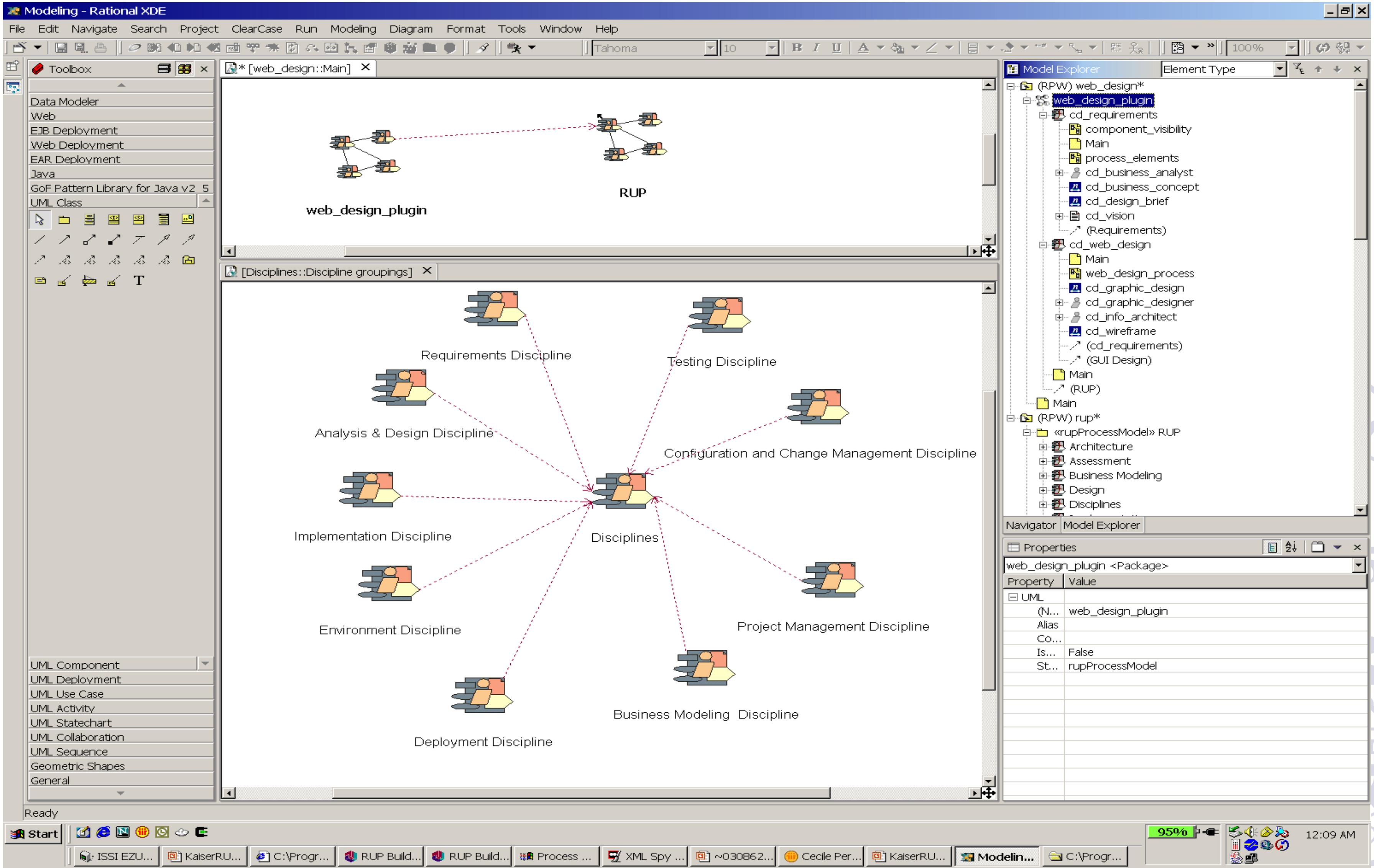
- Use to make extensive customizations to the process
- Allows adding, removing, or modifying core process elements
  - Discipline, workflow, activity, artifact, role, etc.
- Leverages Rational XDE Modeler for visual process authoring
- Open & extensible: Based on OMG's SPEM (Software Process Engineering Meta Model)

Process Engineer: "I need to modify or add process elements"

Accelerate advanced process modeling

Leiden University

Tools for Tailoring RUP - *RUP Modeler*



Property	Value
UML	
(N...	web_design_plugin
Alias	
Co...	
Is...	False
St...	rupProcessModel

Demo Steps – Rational Process Workbench

Jim,
ntent Developer

Creating a Thin Plug-in with RUP Organizer

- Create the plug-in.
- Create a guideline page.
- Drag and drop to attach to the RUP
- Export as a plug-in for RUP Builder

Tom,
rocess Engineer

Creating a Structural Plug-in with RUP Modeler

- Model a new artifact, role, and an activity
- Attach description pages in RUP organizer
- Walkthrough the J2EE plug-in for more advanced modeling examples (extension, composition).

Tailor RUP to Fit Your Needs

Use *MyRUP* to personalize RUP

- Hide unnecessary information
- Add hyperlinks

Use *RUP Builder* to publish a RUP configuration for your project

- Add relevant content through plug-ins
- Remove/add content you do not need through process components
- Produce role-based process views

Use *RUP Organizer* (RPW) to modify the process content

- Add your company-specific guidelines, templates, examples, etc.
- At the end of a project, spend a few hours on packaging your assets for reuse in other projects

Use *RUP Modeler (RPW)* to modify the process structure

- Modify activities, artifacts, roles, etc…
- Only recommended for people that have used RUP in several projects

Common Mistakes: *Adopting RUP*

Not coupling process improvement with business results

Adopting too much of what is in RUP

Customizing too much of RUP too early

Not Coupling Process Improvement With Business Results

The only reason to adopt RUP is to improve business results

- Identify weaknesses in current tools and processes – Customer pain

Macro rollout process to follow

- Treat the rollout as a project by itself
- Identify which business results your project / organization are trying to achieve
- Identify which best practices and tools can help you achieve those business results
- Identify how to measure whether you have achieved expected business results
- Communicate all of the above to all team members
- Roll out process and tools
- Evaluate against expected business results
- Calibrate rollout plan and reset expectations based on findings

Adopting Too Much Of What Is In RUP

Just because it is in RUP, you should not necessarily use it

Understand first the needs of your project

- Project size
- Technology (J2EE, .NET, Real-time, …)
- Tools (Rational RequisitePro, WebSphere, Rational XDE, …)
- Level of ceremony (formal / informal)
- …

Only use what adds value

- Configure / streamline your process using RUP Builder
- Too many artifacts slow you down, too few artifacts and you solve the same problems over and over
- When in doubt, start without it, and add when you see the need

Customizing Too Much of RUP Too Early

Adopt an incremental and iterative approach to customizing RUP

- Do a little customization, try it out, customize more, try it out, and then do maybe a lot if needed….

Almost all projects / organizations should configure RUP, using RUP Builder, on their first project

Most projects / organizations should not customize RUP using RUP Modeler on their first project(s)

- RUP Organizer is normally OK
- Maybe what is in RUP is not exactly what you would like to see, but is it good enough?
- Exception: Process-mature organizations with good RUP knowledge

When you know what problem you want to fix, customize as needed

Approaches for Adopting RUP

Alternative 1: Use it as a knowledgebase

- Non-intrusive with minimal effort and risk
- No different than training, books, and magazines

Alternative 2: Focused implementation aiming at fast results

- Focused and incremental adoption with training and mentoring aiming at changing behavior - takes time and effort
- Focus on critical areas first, it is not an all or nothing
- Use mentors / consultants to accelerate results

Adopting RUP is a continuum between alternative 1 and 2

Choosing the Right Pilot

Purpose

- Mitigate key risks with adopting RUP
- To succeed with the RUP adoption

Team size

- Normally 6-8 people

Duration

- 3-9 months
- You do not have to complete the pilot to achieve your objectives

Staffing

- People interested in learning
- People with the ability to mentor others

Importance and complexity

- Most cases => Important, but not critical
- When nothing to loose => Most critical project you have. Gives you the best people,and sufficient funds

What Results Can You Expect From RUP Adoption

On first project => Improved quality and more satisfied end users

- Early capabilities up and running early
- Early testing
- Manage and control change
- User involvement and acceptance

On later projects => Improved productivity, lead times and precision

- “Smooth development”
- Increased reuse
- Improved predictability

Core Development Problems and Tool Automation

Problems Encountered	Business Impact	Possible Solutions
Product builds have missing files or old versions of files. • Development is based on old files.	• Quality is reduced. • Time is spent on rediscovering old defects.	• Introduce a Configuration and Change Management (CCM) system. • Example: Rational ClearCase and ClearQuest.
• Developers in each other's way, preventing parallel development.	• Longer lead-time, resulting in longer time-to-market.	• Introduce a CCM system providing private workspaces and diff & merge capabilities. • Example: Rational ClearCase and ClearQuest.
• Iterative development introduces increased testing burden on regression testing.	• Test costs increase or defects are found late, making them more costly to fix.	• Automate testing. • Example: Rational Suite TestStudio.
• Unclear which requirements are current, and when they were changed.	• Investments in development are done toward obsolete requirements, increasing development costs and decreasing customer satisfaction.	• Manage requirements using a requirements management tool. • Example: Rational RequisitePro.
• Unclear which change requests should be implemented. • Changes that should be implemented are not implemented.	• Reduced quality and increased development costs.	• Introduce a change and defect tracking system. • Example: Rational ClearQuest.

Summary: *Adopting RUP*

Couple process improvements to business results

Adopt RUP incrementally

Don’t adopt too much of RUP

Pilot projects are crucial, choose them wisely

Have realistic expectations of what you can achieve

- Quality and end customer satisfaction comes early
- Productivity and precision comes first after a few projects

Customizing and configuring RUP

- “Everybody” should use RUP Builder
- Most organizations should use RUP Organizer
- Process mature organizations, or organizations with very specific needs should use RUP Modeler

RUP Projects are ITERATIVE

Work is undertaken within an iteration.

The iteration plan defines the artifacts to be delivered, roles and activities.

An iteration is clearly measurable.

An iteration is TIME BOXED

Iterations are RISK DRIVEN

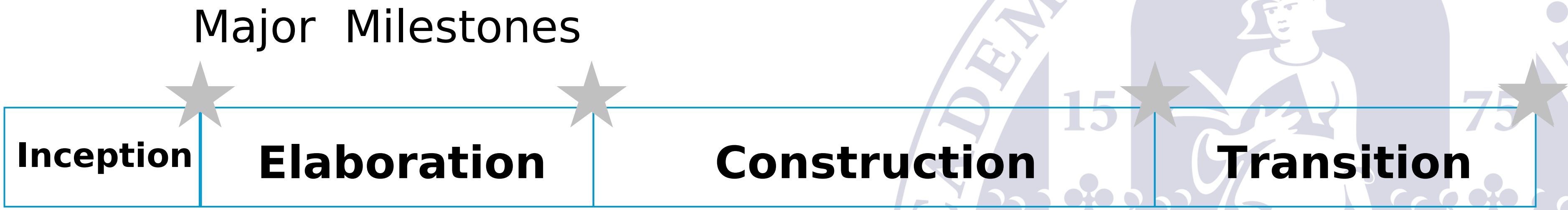
Project progress is made against MILESTONES

Each phase is defined by a milestone.

Progress is made by passing the milestones.

The emphasis of Phases - THEY ARE NOT TIMEBOXED.

Milestones measure success



Major Milestones

Inception

Elaboration

Construction

Transition

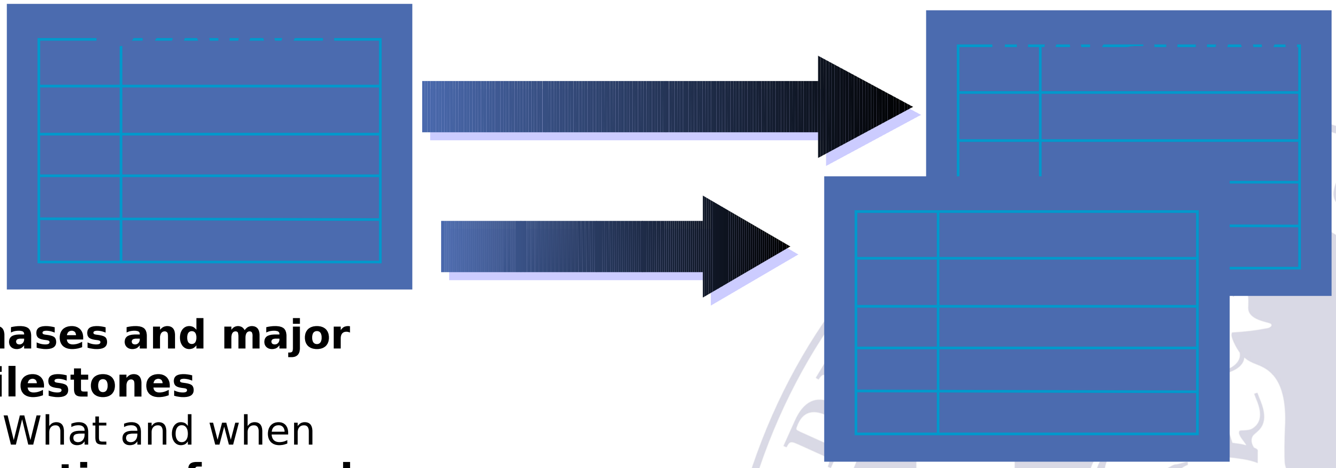
Plan With Evolving Levels of Detail

Coarse-grained Plan:
Project Plan

Fine-grained Plans:
Iteration Plans

One For Entire

Next



Current Iteration

Phases and major milestones
- What and when

Iterations for each phase
- Number of iterations
- Objectives and Duration

- Iterative does not mean less work and shorter schedule
- It is about greater predictability

The Project Plan Defines….

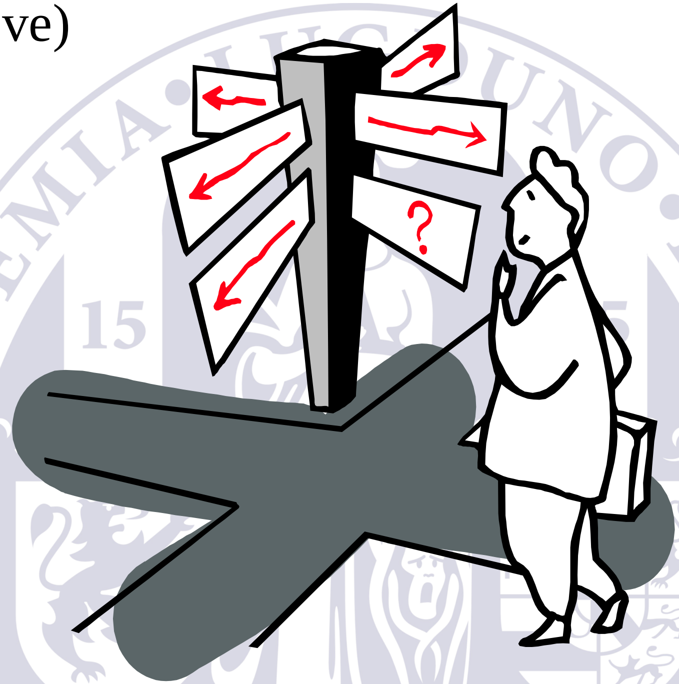
Phase plan with major milestones

Iterations and their objectives

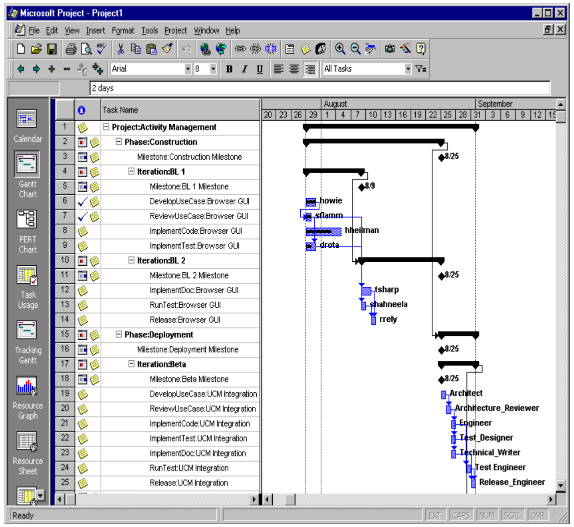
Releases, what and for whom

High-level schedule (with the information above)

Resources and staffing



The Iteration Plan Defines

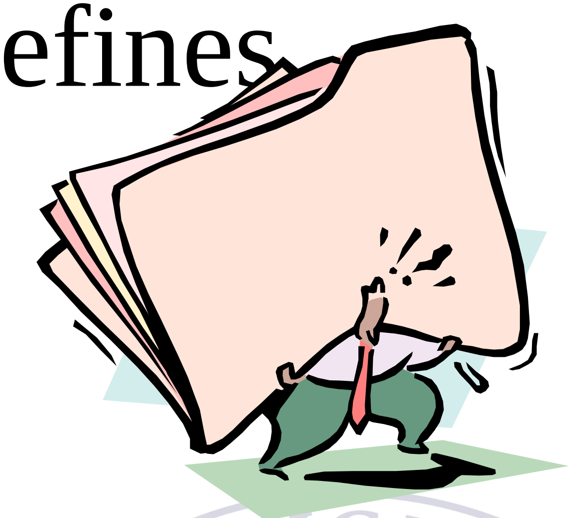


2 days

	ℹ	Task Name
1		⊟ **Project:Activity Management**
2		⊟ **Phase:Construction**
3		Milestone:Construction Milestone
4		⊟ **Iteration:BL 1**
5		Milestone:BL 1 Milestone
6	✓	DevelopUseCase:Browser GUI
7	✓	ReviewUseCase:Browser GUI
8		ImplementCode:Browser GUI
9		ImplementTest:Browser GUI
10		⊟ **Iteration:BL 2**
11		Milestone:BL 2 Milestone
12		ImplementDoc:Browser GUI
13		RunTest:Browser GUI
14		Release:Browser GUI
15		⊟ **Phase:Deployment**
16		Milestone:Deployment Milestone
17		⊟ **Iteration:Beta**
18		Milestone:Beta Milestone
19		DevelopUseCase:UCM Integration
20		ReviewUseCase:UCM Integration
21		ImplementCode:UCM Integration
22		ImplementTest:UCM Integration
23		ImplementDoc:UCM Integration
24		RunTest:UCM Integration
25		Release:UCM Integration

Ready

The deliverables for that iteration.



The to do list for the team members

Phase definition

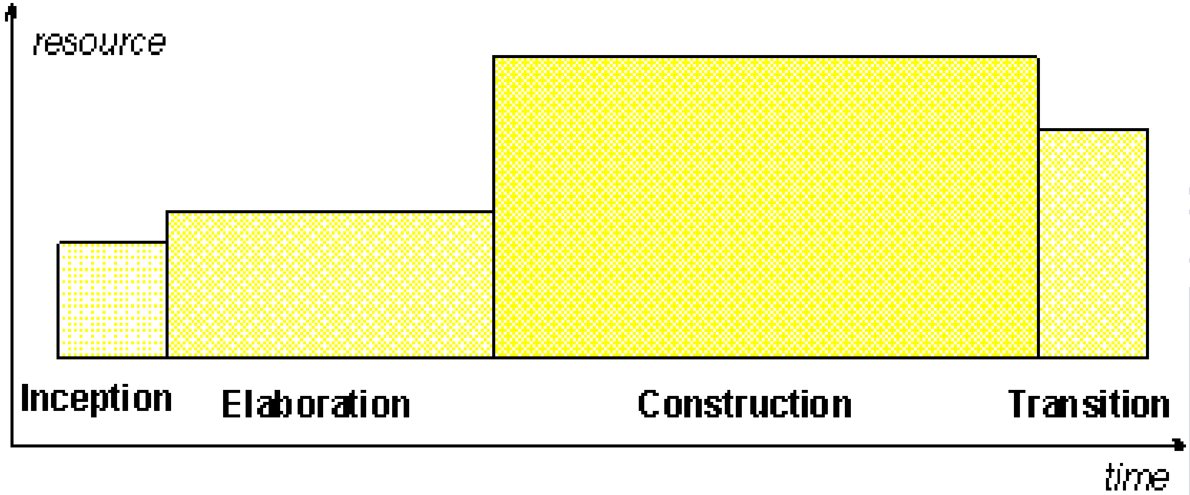
Phases map to types of risks for an iteration.

- Inception – Risks associated with Scope
- Elaboration – Risks associated with Architecture
- Construction – Risks associated with Production
- Transition – Risks associated with the Roll out

Use this knowledge to identify the right milestones

Phases

Use ‘average’ project for basis



Effort	***~5%***	***20%***	***65%***	***10%***
Schedule	***10%***	***30%***	***50%***	***10%***

Source: Walker Royce 1995

Phases

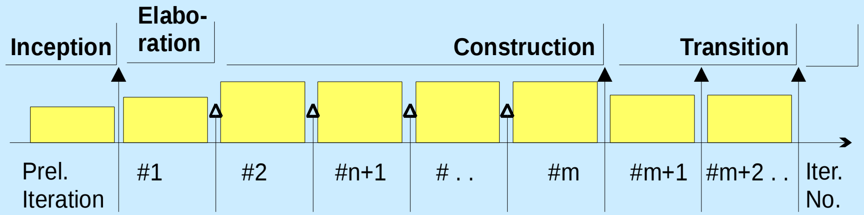
Review the project domain and determine how your risk profile relates to most projects.

Adjust relative duration of the phase by mapping risks to each phase. For example

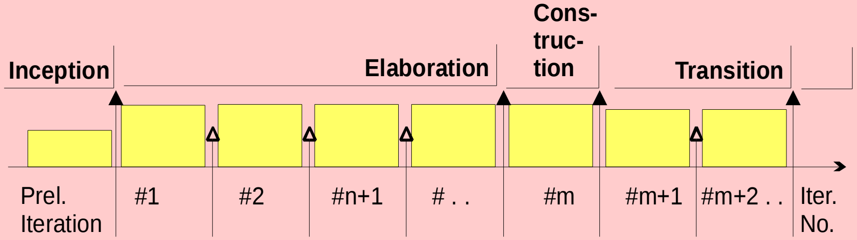
- Finding scope is a real problem – Increase relative length of inception by 5%
- Architecture is unknown – Increase relative length of elaboration by 5%
- Using pre-defined architecture – decrease relative length of elaboration….

Examples of projects…..

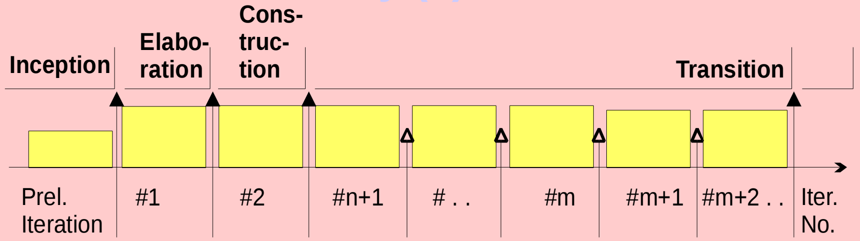
Incremental (1)



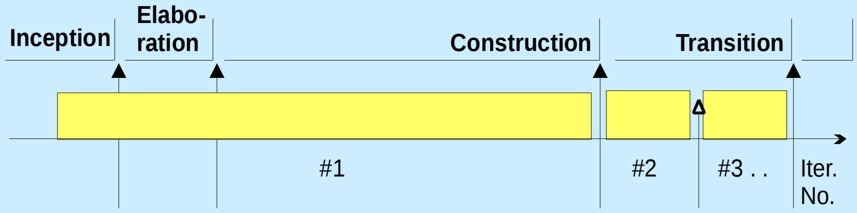
Evolutionary (2)



Incremental delivery (3)



"Grand design" (4)



Iteration Length

An iteration should typically be 2-8 weeks long

Drivers for longer iterations

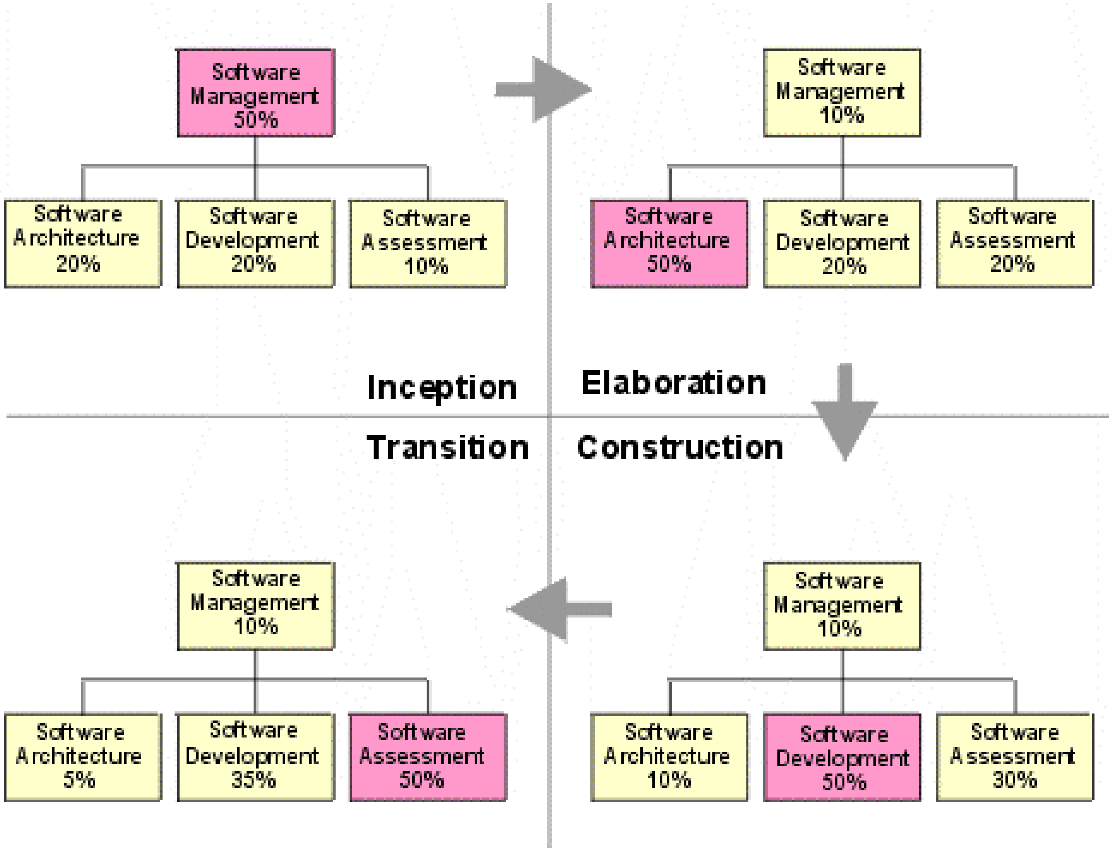
- Complex stakeholder relations, regulatory constraints
- Immature tools (Change management, IDEs, test environments, …)
- Many project members, inexperienced project members
- Complex technology, large code base, brittle architectures
- Long project duration

Sample projects

- 5 people, 1-2 week
- 20 people, 3-4 weeks
- 80 people, 8 weeks

Staffing levels

Staffing profile is dependent on phase.



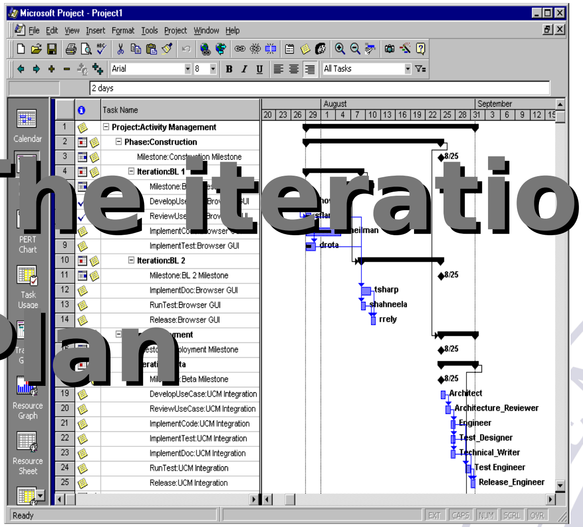
Software project team evolution over the life-cycle

The iteration plan describes….

the work to be done

the artifacts to be delivered

the iteration length



2 days

	Task Name
1	**Project:Activity Management**
2	**Phase:Construction**
3	Milestone:Construction Milestone
4	**Iteration:BL 1**
	Milestone:B
	DevelopUse Brow GUI
	ReviewUse Brow GUI
	ImplementC owser
9	ImplementTest:Browser GUI
10	**Iteration:BL 2**
11	Milestone:BL 2 Milestone
12	ImplementDoc:Browser GUI
13	RunTest:Browser GUI
14	Release:Browser GUI
	ment
	estc ployment Milestone
	erati ta
	Mil :Beta Milestone
19	DevelopUseCase:UCM Integration
20	ReviewUseCase:UCM Integration
21	ImplementCode:UCM Integration
22	ImplementTest:UCM Integration
23	ImplementDoc:UCM Integration
24	RunTest:UCM Integration
25	Release:UCM Integration

The iteration Plan

the risks to be mitigated

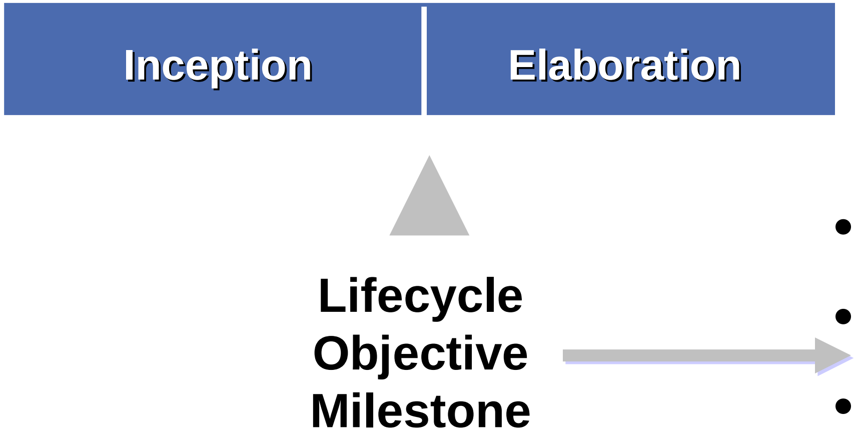
Who is doing what

how you measure the iteration

Leiden University

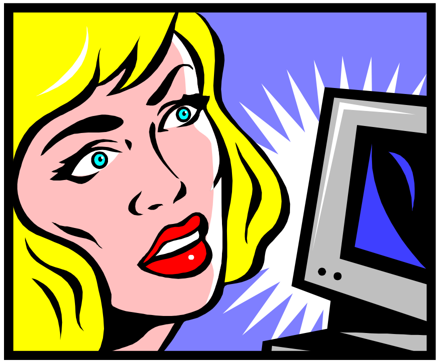
Consider Your Current Phase of Development

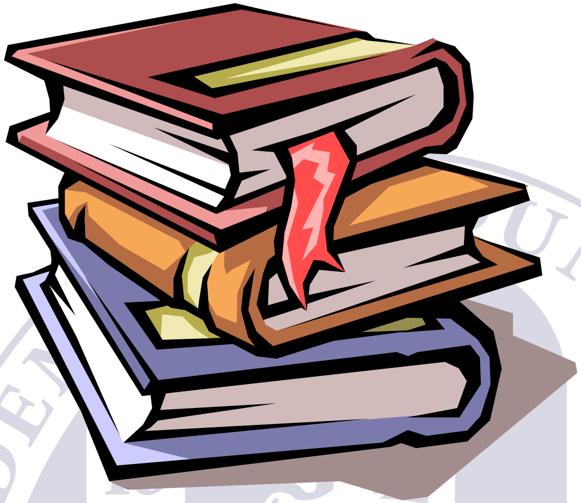
Determine the objective of this iteration by reviewing the milestones for the phase.



- **Project Scope**
- **Candidate Architectures**
- **Risks**
- **Supporting Environment**

Decide on the Deliverables





We need to deliver

*** Risk**

*** Business Case**

*** Architectural synthesis**

Iteration Assessment

At the end of each iteration

- Assess how you did compared to iteration plan and success criteria as specified in project plan
- Success criteria should primarily be defined by measurable deliverables
  - And those should have been defined before the iteration started!
- Focus on demonstrable progress (executable code, actual deliverables, …), not what activities have been completed

Update project plan based on iteration assessment

- Provides a very objective assessment whether you are on track or not

Summary: *Planning Iterative Development*

Iterative development requires more planning

- Plans evolve as understanding improves
- Well-defined and objective milestones provides improved picture of true project status
- Allows for killing bad projects early rather than late

Two plans in a RUP project

- Coarse-grain project plan
- Iteration plan for each iteration

Iteration plan drives development

- Describes the things that will be delivered
- Details the activities to be undertaken
- Has a fixed time box
- Defines assessment criteria the for the end of the iteration

Project plan defines milestones and assessment criteria

- “Big picture” plan

Iterative Development

In iterative development, you constantly move between requirements, architecture, design, implementation, and test

Requires different team dynamics and values

Puts special requirements on people, process and tools

Iterative development requires a mind shift.

Team Dynamics

OLD THINKING	NEW THINKING
Functional teams of either all analysts, all developers, or …	Cross-functional teams consisting of analysts, developers, testers, …
OK with low-bandwidth communication	Must have high-bandwidth communication
Communicate primarily through documents (requirements, design, ...)	Communicate through evolving models, tools and face-to-face
Narrow specialists	Generalists and specialists with bird-eye perspective
"That's not my job"	"We are all responsible for the application"

Analyst

OLD THINKING	**NEW THINKING**
Requirements need to be finalized early	Requirements should evolve so they address customer needs
I specify requirements, developers implement them	I am responsible for requirements, but seek input from users, developers, testers, technical writers, …
The more detailed requirements, the better	Requirements should be detailed enough to address stakeholder needs, but not more since it increases cost

Developer

OLD THINKING

Build high-quality code addressing the requirements

I take pride in developing my own unique solution

Testers test my code

Testers are responsible for the quality

NEW THINKING

Build high-quality code addressing user needs

I take pride in delivering a workable solution (reuse is good)

I test my own code

I am responsible for the quality of my code, but testers verify that I did a good job

Tester

OLD THINKING	**NEW THINKING**
Testing is an afterthought	Testing is an integral part of the development effort
A quality police	A knowledge resource that assists the team in reaching high-quality code
Test according to well-defined and detailed test specification	Test according to well-defined test criteria

Manager

OLD THINKING

Hide risks so people do not know how bad things are

Understand status by asking for status reports

More documentation is better

Requirements and test are the high value activities

Team members probably do a bad job – we need to check everything

Distrust

NEW THINKING

Expose risks so everybody knows what to focus at

Understand status by observing what works (e.g cross-functional team demos latest capabilities)

The right amount of documentation is good

The right balance between requirements, design, code and test provides the value

The different perspectives of an integrated team motivates us to excel

Trust

Customer

OLD THINKING	**NEW THINKING**
They build the product	We build the product
I specify what I want, later on they deliver	I specify what I want as well as I can, and then assist in getting it to what I really want

People Guidelines

Staff the project with people with complementary skills

Staff the project with different levels of experience and expertise

- Expert, Journeyman, Apprentice
- Complementary skills

Make sure everyone understands the vision (goal) of the project

Provide a learning environment

- Everyone on the team should learn something
- Can be technical or other
- Make learning a goal and provide the time to learn

People Guidelines

Build a high-trust environment

- Trust can be lost but don’t force it to be earned from the beginning
- If you have an issue with somebody, confront them. Do not just complain...

Allow disagreements

- Disagreement can be a sign of synergy
- Have a resolution strategy and make sure the team understands it
- Have open discussions

Provide time for interactions among all team members

- Be creative at meetings
- Have lunch together

People Guidelines

Recognize achievement sincerely and often

- A sincere “thank you” works wonders

Allow peer recognition

- When did you last time send an impromptu e-mail acknowledging the performance of a peer?
- But watch out for the “you scratch my back, I’ll scratch yours” syndrome…

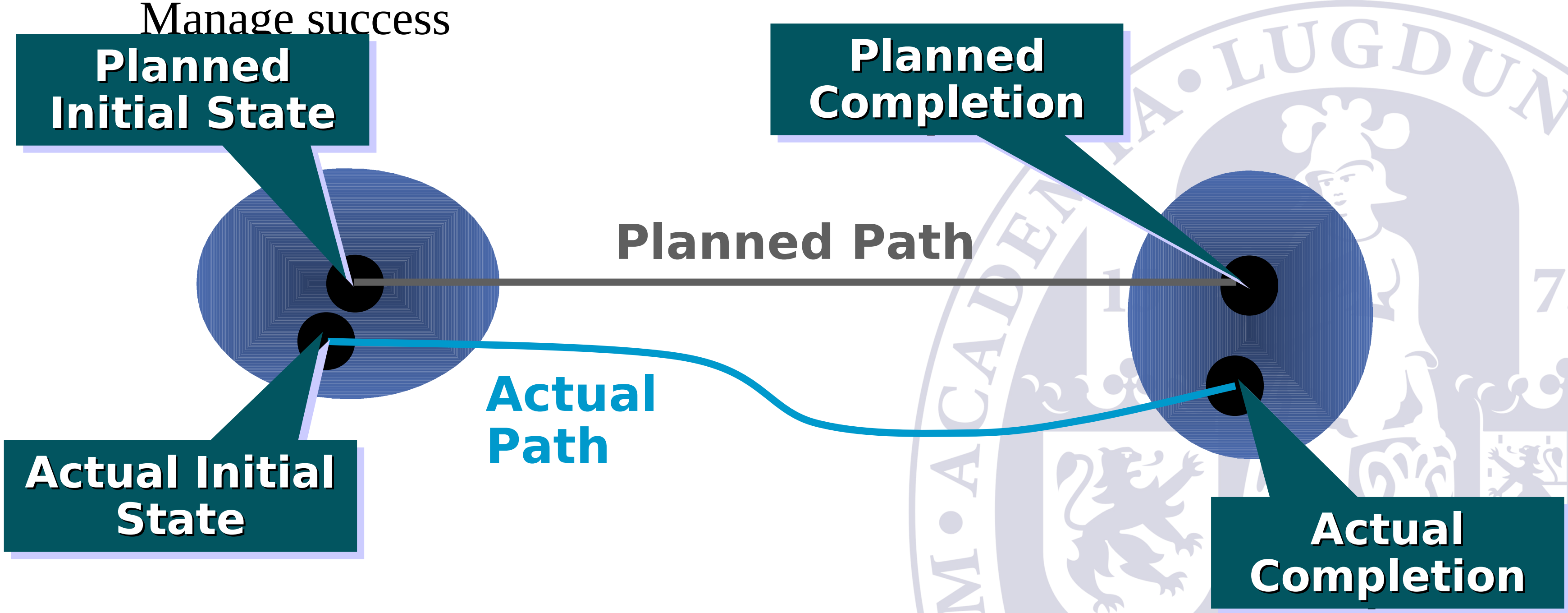
Leadership Principle: *Participate, Not Oversee*

Maintain the big picture

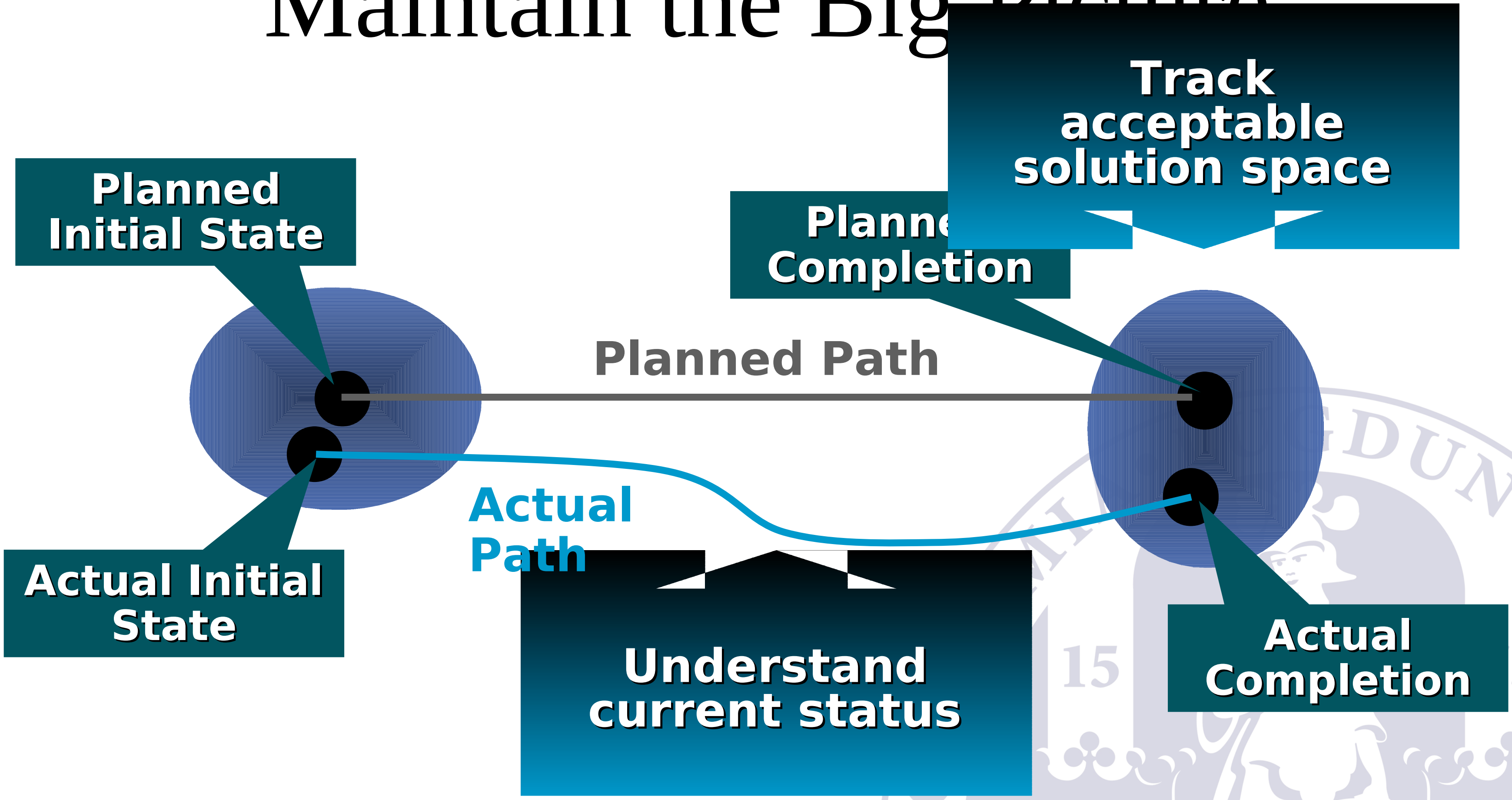
Steer, not just track

Share the risk

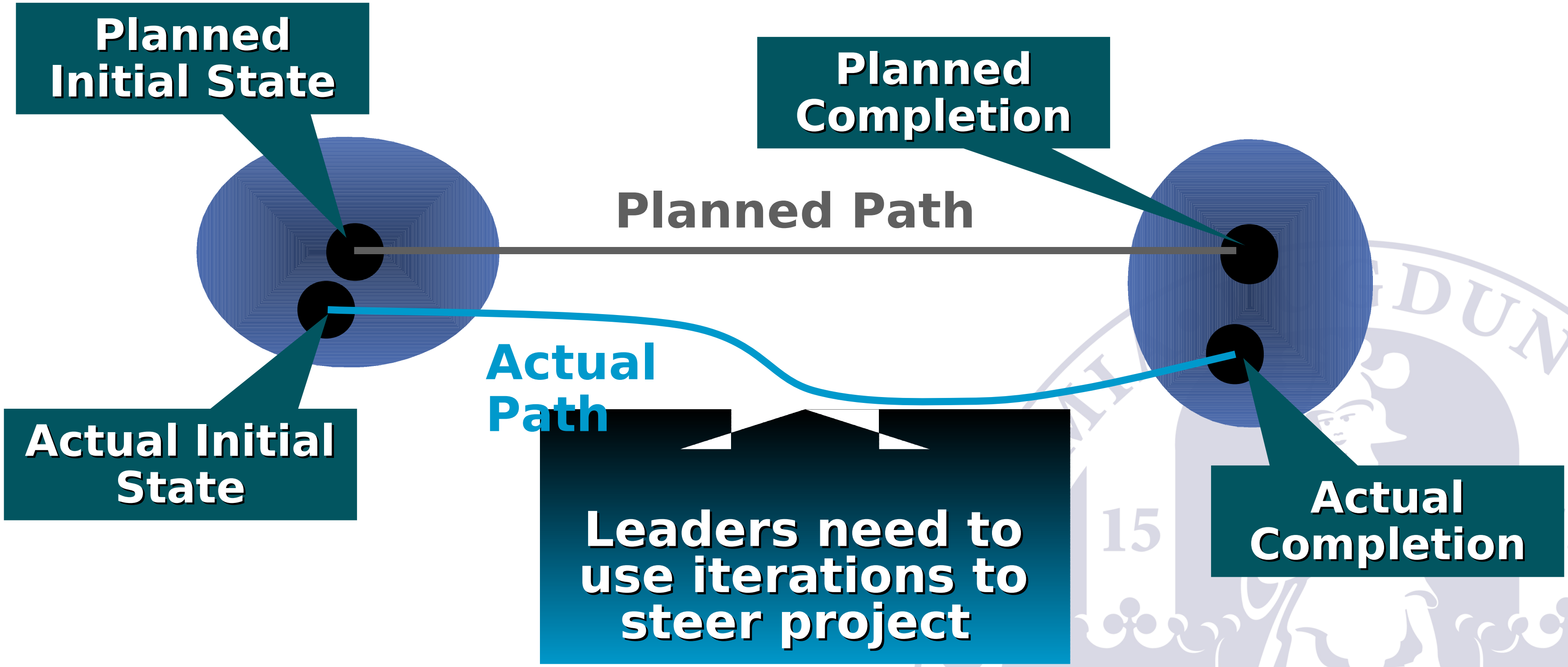
Manage success



Maintain the Big Picture



Steer, Not Just Track



Planned Initial State

Planned Completion

Planned Path

Actual Initial State

Actual Path

Actual Completion

Leaders need to use iterations to steer project

Monitoring Progress

Tracking project:

- **Stability** – changes in planned requirements, design, sizing
- **Progress** – actual demonstrable progress against plans
    - Content in the iterations
- **Quality** – test status, reports of iterations
- **Schedule stability** – iteration content
- **Budget** – cost, schedule variance based on iteration content

If not on track,

- Ask for reasons
- Adjust the resources
- Find ways to help

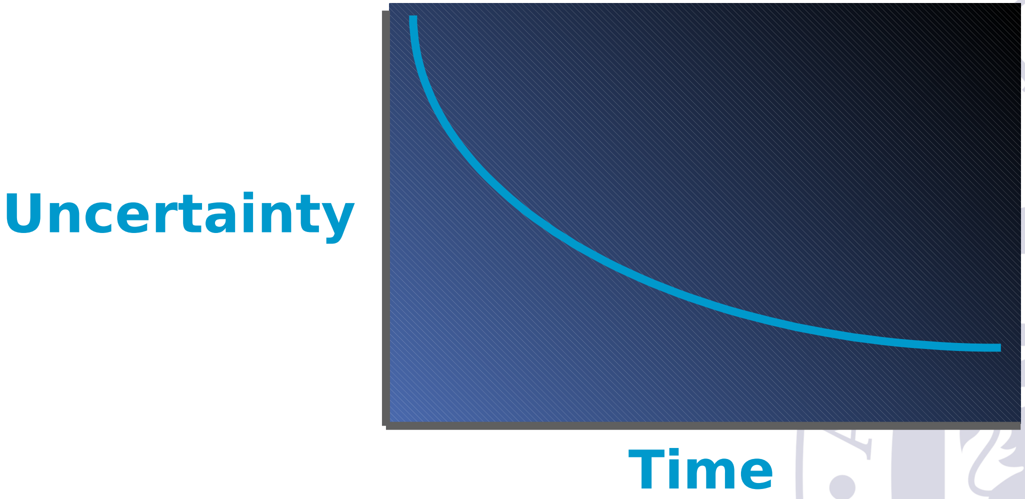
Share and Manage the Risk

Show appreciation of the uncertainty curve

Don’t ask for commitments that are impossible to make

Share in estimations and planning

Use iterations to manage uncertainty



Manage Success

In software development, success breeds success

- Team members motivated by being on successful projects
- Better motivator than fear or berating

Managers can

- Create opportunities for success using iteration plans
- Show appreciation of the success of early iterations
- Head off failure by uncovering problems early

Summary: *The Soft Side of Managing Iterative Development*

Iterative development requires a mind shift

- The way people collaborate needs to change
- Large projects organize around architecture, not functional roles

As a manager you need to

- Understand the nature of iterative software development
- Provide the big picture
- Steer and participate, not oversee
- Staff projects with different skill
- Praise people
- Create a winning team