CSE 210: Development Processes
Agile and Plan-Driven Development

Ingolf H. Krueger
ikrueger@ucsd.edu, http://sosa.ucsd.edu

with contributions from
Michael Meisinger, Massimiliano Menarini, Bernhard Rumpe

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

California Institute for Telecommunications
and Information Technologies
La Jolla, CA 92093-0405, USA
Quote of the Day

The software field is not a simple one and, if anything, it is getting more complex at a faster rate than we can put in order.

B.W. Boehm*

Overview

- Background and Motivation
- Software Project Risks
- Development Processes – Basics
- Agile Development
- Plan-Driven Development
- Balancing Agility and Discipline
- Process Models – Time for Tailoring
- Summary and Outlook
Software Engineering Topics

- Project Management
- Process Modeling
- Software Development Methods
  - Requirements Engineering
  - Software Architecture and Design
  - Software Maintenance
  - Re-Engineering
- Quality Management (incl. Testing)
- Notations and Languages (UML, Java, ...)
- Tool Support (incl. CASE, CVS, make)
# Software Engineering Topics

<table>
<thead>
<tr>
<th>Project Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Modeling</td>
</tr>
<tr>
<td>Software Development Methods</td>
</tr>
<tr>
<td>Requirements Engineering</td>
</tr>
<tr>
<td>Software Architecture and Design</td>
</tr>
<tr>
<td>Software Maintenance</td>
</tr>
<tr>
<td>Re-Engineering</td>
</tr>
<tr>
<td>Quality Management (incl. Testing)</td>
</tr>
<tr>
<td>Notations and Languages (UML, Java, ...)</td>
</tr>
<tr>
<td>Tool Support (incl. CASE, CVS, make)</td>
</tr>
</tbody>
</table>
June 4, 1996: “Ariane-5” Catastrophe

- Superfluous calibration program for inertia sensors runs in-flight; the software was “reused” from Ariane-4
- Measured values in Ariane-5 exceed those assumed for the Ariane-4 software.
- The corresponding (Ada-) exception is handled by stopping the control computer and switching over to a redundant system.
- The second system contains the same error and deals with the exception the same way...

- Costs of the Ariane-5-program until 1996: 
  ~ $ 8 Billion
- Value of destroyed satellite: 
  ~ $ 500 Million
Software Catastrophes Abound

• Financial Catastrophes:
  – 1993: Stopped CA project aiming at integration of systems for
driver licenses and vehicle registrations: $44 Mio
  – 1997: Stopped CA project State Automated Child Support System
“SACSS”: $300 Mio

• Deadlines:
  – 1994: Denver International Airport delayed 18 Months due to
software problems of the luggage transportation system
  – 2003: Toll Collect: start date for German road toll system delayed to
1/2005, claimed losses > 4 Billion Euro

• Technical Catastrophes:
  – 3/1999: Launch failure of Titan/Centaur-Rocket due to wrong
software version
  – 9/1999: Loss of “Mars Climate Orbiter” due to wrong unit
conversion

• USA: Losses during year 2000 due to software defects:
$100 Billion.
Why is High-Quality Software Difficult to Build?

- Software-Quality is difficult to measure
- Software is easily changeable at any stage during the development process
- Implementation platforms and technologies change rapidly
- Complexity is rapidly increasing
  - For every 25% increase in problem complexity there is a 100% increase in complexity of the software solution \[\text{Glass02}\]
- Adequate methods and tools do not exist or are only partially adopted
- Requirements change
Why is High-Quality Software Difficult to Build?

• 0.1%-defect rate means:
  – per year:
    • 20,000 errors in medication
    • 300 failing heart pacers
  – per week:
    • 500 errors in medical surgeries
  – per day:
    • 16,000 lost letters in the postal system
    • 18 airplane crashed
  – pro hour:
    • 22,000 checks posted incorrectly

• Therefore:
  – Massive QA efforts required also in the future.
How Complex are These Systems Anyhow?

- Telecommunications
- Networking
- Embedded Systems, Automotive
- Business Information Systems (Web Services)
- Mobile, dynamic services, ad-hoc networks
Growing Complexity (1)

- Siemens EWSD V8.1: 12,5 Million LOC, ~190,000 pages of documentation
Growing Complexity (2)

- SAP’s Enterprise-Resource-Planning Software R/3®

<table>
<thead>
<tr>
<th>year</th>
<th>LOC</th>
<th># function points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1994</td>
<td>7 Million</td>
<td>14 000</td>
</tr>
<tr>
<td>1997 (Rel. 3.1)</td>
<td>30 Million</td>
<td>200 000</td>
</tr>
<tr>
<td>1999 (Rel. 4.5)</td>
<td>50 Million</td>
<td>400 000</td>
</tr>
</tbody>
</table>

- Further info on R/3 Release 4:
  - 11 000 external database tables
  - 500 000 table fields (external and internal)
Growing Complexity (3)

- Total size of software used in select corporations (beginning of 2000):
  - Chase Manhattan Bank: 200 Mio. LOC
  - Citicorp Bank: 400 Mio. LOC
  - AT&T: 500 Mio. LOC
  - General Motors: 2 Billion LOC
Complexity Growth and Error Rate

- **#errors in 1000 LOC**
  - 1977: 20
  - 1994: 0.2

- **Program size (1000 LOC)**
  - 1977: 10
  - 1994: 800

- **Resulting #errors (absolute)**
  - 1977: 200
  - 1994: 160

True quality increase requires **overcompensating** the increase in program complexity!

(Averages, source: Balzert 96)
Increasing Quality Requirements

- Found defects in 1000 LOC:
  - 1977: 7 - 20
  - 1994: 0,05 - 0,2

- Quality increased by factor 100 over 17 years.

- But: must compensate complexity increase!
  - ~ factor 10 in 5 years

- Increasing concern for “legacy debts”:
  - Business systems exist for 20 years and more.
  - In some companies 60-70% of software costs spent on adaptation of legacy software!
Success of IT Projects

Top 10 Project Success Factors

1. Executive support (18%)
2. User involvement (16%)
3. Experienced project manager (14%)
4. Clear business objectives (12%)
5. Minimized scope (10%)
6. Standard software infrastructure (8%)
7. Firm basic requirements (6%)
8. Formal methodology (6%)
9. Reliable estimates (5%)
10. Other criteria (5%)


A software development process can have significant influence on these success criteria.
Overview

• Background and Motivation
• **Software Project Risks**
• Development Processes – Basics
• Agile Development
• Plan-Driven Development
• Balancing Agility and Discipline
• Process Models – Time for Tailoring
• Summary and Outlook
Core Software Project Risks

- **Schedule Flaw**
  - Underestimate the size of the project
  - Lack of estimation

- **Requirements Inflation**
  - Requirements do change

- **Turnover**

- **Specification Breakdown**
  - Problem areas avoided early in the process resurface during construction

- **Under-Performance**

see [DML03]
Risk Management

- There is a risk-list capturing general software project risks and risks specific to the project at hand.
- Risk-discovery is performed throughout the project duration.
- Uncertainty due to risk is captured.
- Goals and Estimates are captured separately.
- Each risk is associated with
  - Transition indicator (when does risk materialize?)
  - Contingent actions/Mitigation strategy
  - Likelihood
- Project value is assessed.
- Perform incremental development.

see [DML03]
Overview

- Background and Motivation
- Software Project Risks
- Development Processes – Basics
- Agile Development
- Plan-Driven Development
- Balancing Agility and Discipline
- Process Models – Time for Tailoring
- Summary and Outlook
### Facts of the Trade

- Requirements deficiencies are the prime sources of project failures.

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

<table>
<thead>
<tr>
<th>Coding</th>
<th>Unit-Test</th>
<th>Component-Test</th>
<th>System-Test</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Prototyping (significantly) reduces requirement and design errors, especially for user interfaces.
Facts of the Trade

• Individual developer performance varies considerably.

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

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

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

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

adapted from [ER03]
Phase- and Process Models

- **Phase Model:**
  Separation of the development process of a (software-) product into defined and distinct phases
  - Specification of an order in the execution of phases
  - Guidelines for the definition of intermediate results/artifacts

- **Process Model:**
  Detailed phase model + Definition of artifacts

- What process models do you know?
Activities in Software Development

• Distinguish between timed phases and activities occurring within a phase.

• Basic activities:
Activities in Software Development

• Distinguish between timed phases and activities occurring within a phase.

• Basic activities:
  – Analysis
  – Design
  – Implementation
  – Test (unit + integration, synonym: validation)
  – Deployment (installation, training)
  – Evolution (maintenance)
  – others (version management, reviews, ...)

© Ingolf H. Krueger
Classic Waterfall

Analysis -> Design

Product Definition

Design Spec

Implementation

Change Requests

Test, Integration

Validated Code

Maintenance

W. Royce (1970)
Classic Waterfall

Product Definition 10%

Design Spec 20%

Code 20%

Validated Code 50%

W. Royce (1970)
Quality Assurance in the V-Model

Boehm 1979 ("V-Model")
Evolutionary Development

- Typical for smaller systems/projects
- Increasingly being used also for larger systems
Iterative and Incremental Development

Analysis

Specification

Design

Implementation
Process Components (V-Model XT)

• Project Management
• Quality Assurance
• Configuration Management
• Defect and Change Management
• Client-side Management
• Contractor-side Management
• Controlling
• Metrics and Project Evaluation

• Requirements Analysis
• System Design & Integration
• Software Development
• Hardware Development
• Integrated Logistics Support
• Ready-to-use products (COTS)
• Usability
• Safety and Security
• Migration and Software Evolution

• Organization wide process Implementation and Improvement
Agility vs. Discipline

- **Agile methods:**
  - Light process, less documentation
  - Short iterations
  - Quality: customer satisfaction
  - Continuous technical and process improvement
  - Close relationship with the customer

- **Traditional development:**
  - Heavy process, lot of documentation
  - Waterfall approach
  - Quality: plan/process compliance
  - Aims to improve process
  - Upfront software & systems architecture, design (BDUF)

- **Successful projects need both**
  - Both approaches have home grounds
  - Different mixes of agility and discipline are needed
  - Process should have the right “weight” for the specific project

- **Key to success lies in risk analysis**

adapted from [BT03]
### Agile and Plan-Driven Homegrounds

adapted from [BT03]

<table>
<thead>
<tr>
<th>Project Characteristics</th>
<th>Agile Home Ground</th>
<th>Plan-Driven Home Ground</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary Goals</td>
<td>Rapid value,</td>
<td>Predictability,</td>
</tr>
<tr>
<td></td>
<td>responding to</td>
<td>stability; high</td>
</tr>
<tr>
<td></td>
<td>change</td>
<td>assurance</td>
</tr>
<tr>
<td>Size</td>
<td>Smaller teams and</td>
<td>Larger teams and</td>
</tr>
<tr>
<td></td>
<td>projects</td>
<td>projects</td>
</tr>
<tr>
<td>Environment</td>
<td>Turbulent, high</td>
<td>Stable, low change,</td>
</tr>
<tr>
<td></td>
<td>change, project</td>
<td>project and organization focused</td>
</tr>
<tr>
<td></td>
<td>focused</td>
<td></td>
</tr>
<tr>
<td>Project Characteristics</td>
<td>Agile Home Ground</td>
<td>Plan-Driven Home Ground</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td><strong>Management</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customer Relations</td>
<td>Dedicated on-site customers, focused on prioritized increments</td>
<td>As-needed customer interaction, focused on contract provisions</td>
</tr>
<tr>
<td>Planning and Control</td>
<td>Internalized plans, qualitative control</td>
<td>Documented plans, quantitative control</td>
</tr>
<tr>
<td>Communications</td>
<td>Tacit interpersonal knowledge</td>
<td>Explicit documented knowledge</td>
</tr>
</tbody>
</table>
## Agile and Plan-Driven Homegrounds

adapted from [BT03]

<table>
<thead>
<tr>
<th>Project Characteristics</th>
<th>Agile Home Ground</th>
<th>Plan-Driven Home Ground</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Technical</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements</td>
<td>Prioritized informal stories and test cases, undergoing unforeseeable change</td>
<td>Formalized project, capability, interface, quality, foreseeable evolution requirements</td>
</tr>
<tr>
<td>Development</td>
<td>Simple design, short increments refactoring assumed inexpensive</td>
<td>Extensive design, longer increments, refactoring assumed expensive</td>
</tr>
<tr>
<td>Test</td>
<td>Executable test cases define requirements, testing</td>
<td>Documented test plans and procedures</td>
</tr>
</tbody>
</table>
## Agile and Plan-Driven Homegrounds

adapted from [BT03]

<table>
<thead>
<tr>
<th>Project Characteristics</th>
<th>Agile Home Ground</th>
<th>Plan-Driven Home Ground</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personnel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customers</td>
<td>Dedicated, colocated Crack(^1) performers</td>
<td>Crack(^1) performers, not always colocated</td>
</tr>
<tr>
<td>Developers</td>
<td>At least 30% full-time highly skilled expert programmers</td>
<td>50% process and architecture experts early, 10% throughout; 30% intermediate programmers workable</td>
</tr>
<tr>
<td>Culture</td>
<td>Comfort and empowerment via many degrees of freedom (thriving on chaos)</td>
<td>Comfort and empowerment via framework of policies and procedures (thriving on order)</td>
</tr>
</tbody>
</table>

\(^1\)Collaborative, representative, authorized, committed and knowledgeable
Overview

- Background and Motivation
- Software Project Risks
- Development Processes – Basics
- **Agile Development**
- Plan-Driven Development
- Balancing Agility and Discipline
- Process Models – Time for Tailoring
- Summary and Outlook
Agile methods

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

Verify
Select Feature Set
Multiple Short Iterations 7-30 days
Implement
Manifesto

**Manifesto for Agile Software Development**

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

---

Kent Beck  
Mike Beedle  
Arie van Bennekum  
Alistair Cockburn  
Ward Cunningham  
Martin Fowler

James Grenning  
Jim Highsmith  
Andrew Hunt  
Ron Jeffries  
Jon Kern  
Brian Marick

Robert C. Martin  
Steve Mellor  
Ken Schwaber  
Jeff Sutherland  
Dave Thomas

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

1. We follow these principles: Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

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

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
The 12 Principles

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.
Examples of Agile Processes

- SCRUM
- eXtreme Programming (XP)
- [Tailored RUP/VModelXT]
Scrum

- Very light
- More a management tool than a software process
  - Product Backlog / Product owner
  - Team / Scrum master
  - Sprint (30 day iteration)
  - Sprint backlog
  - Daily scrum meetings
- Empirical process: continuous feedback to control process and obtain desired output
- Team completely free to organize itself
- Any development process and practice chosen by the team can be used during each sprint
  - Example xP@Scrum & XBreed
Scrum

Scrum: 15 minute daily meeting. Teams members respond to basics:
1) What did you do since last Scrum Meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?

Sprint Backlog: Feature(s) assigned to sprint

Backlog items expanded by team

Product Backlog: Prioritized product features desired by the customer

New functionality is demonstrated at end of sprint
XP

• 4 values:
  
  – Communication  
    (most projects fail because of poor communication)
  
  – Simplicity  
    (YAGNI = You Ain’t Gonna Need It)
  
  – Feedback  
    (from customer and other developers)
  
  – Courage  
    (make hard decisions and support other principles and practices)
XP

- **Practices**
  - Customer *continuously present* on site
  - Planning game (requirements captured on index cards)
  - Metaphor support single vision of success
  - Short cycle time (no more than 3 weeks)
  - Collective ownership of code
  - Simple design
  - Pair programming
  - Refactoring
  - Coding standards
  - Test first approach
  - Continuous integration
  - 40-hour work week
Overview

- Background and Motivation
- Software Project Risks
- Development Processes – Basics
- Agile Development
- Plan-Driven Development
- Balancing Agility and Discipline
- Process Models – Time for Tailoring
- Summary and Outlook
Plan Driven Methods

- **Goals:**
  - Predictability
  - Stability
  - High assurance

- Well defined process
- Plans and specifications are required for certification
- Fit well with complex and safety critical projects
- Scale up very well
- Work best with stable requirements

Diagram: Define Process → Perform Project → Control Process → Improve Process → Measure Process
Examples of Plan-Driven Processes

- CMM/PSP/TSP
- Cleanroom
- [Tailored RUP/VModelXT]
# Capability Maturity Model (CMM)

- CMU-developed maturity assessment method

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Control Over</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1: Initial</td>
<td>Cost, time and quality unpredictable</td>
<td></td>
</tr>
<tr>
<td>Step 2: Repeatable</td>
<td>Stable time, but varying cost and quality</td>
<td></td>
</tr>
<tr>
<td>Step 3: Defined</td>
<td>Stable cost and time, but varying quality</td>
<td></td>
</tr>
<tr>
<td>Step 4: Managed</td>
<td>Control over product quality</td>
<td></td>
</tr>
<tr>
<td>Step 5: Optimizing</td>
<td>Control over process quality</td>
<td></td>
</tr>
</tbody>
</table>
Capability Maturity Model (CMM)

- CMMI & SW-CMM
- Set of practices that improve organization process capability
- Aims at eliminating main causes of poor quality and poor productivity
- Maturity levels for SW-CMM (build upon each other):
  1. Initial: No particular process
  2. Repeatable: Requirements Management, SW Project Planning, SW project tracking, SW Subcontract Management, SW Quality Assurance, SW Configuration Management.
  4. Managed: Quantitative process management, Software quality management
Cleanroom

• Mathematically-based verification with certified reliability
• Computer programs are complex mathematical functions and desired operations can be precisely specified
• Focus is on defect-free code
• Include processes for:
  – Planning
  – Specification
  – Design
  – Verification
  – Coding
  – Testing
  – Certify software
• Software specified as a black-box, development generates a clear-box
Overview

- Background and Motivation
- Software Project Risks
- Development Processes – Basics
- Agile Development
- Plan-Driven Development
- Balancing Agility and Discipline
- Process Models – Time for Tailoring
- Summary and Outlook
Balancing Agility and Discipline adapted from [BT03]

• Three possible approaches:
  – Use plan to scale up Agile Methods
  – Use agility to streamline Plan-Driven Methods
  – Tailor a life cycle process as needed for the project

• Agile Methods:
  – Effort to add features does not increase with time
  – Tacit knowledge is enough to capture information
  – YAGNI.

• Plan-Driven Methods:
  – Requirements are fixed
  – Development process can be predicted
  – Technology is stable
  – The environment is not changing

• When do these assumptions break down?
Using Risks

- Identify risks in using practices and techniques of agile and plan-driven methods
  - If necessary use prototyping, data collection, and analysis to gather information about risks
- Compare agile and plan-driven risks
  - If agile riskier than plan-driven go for a risk-based plan-driven process
  - If plan-driven riskier then agile go for a risk-based agile process
  - If the project does not fit either agile or plan-driven methods tailor an architecture where agile parts are encapsulated
- Tailor the life cycle process for the project
- Execute and monitor
  - If needed modify the process

adapted from [BT03]
Development Process Risks

• Environmental:
  – E-Tech: Technology uncertainties
  – E-Coord: Many diverse stakeholders to coordinate
  – E-Cmplx: Complex systems of systems

• Agile risks:
  – A-Scale: Scalability and criticality
  – A-YAGNI: Use of simple design
  – A-Churn: Personnel turnover or churn
  – A-Skill: Not enough people skilled in agile methods

• Plan-Driven risks:
  – P-Change: Rapid change
  – P-Speed: Need for rapid results
  – P-Emerge: Emergent requirements
  – P-Skill: Not enough people skilled in plan-driven methods
Risk-Driven Development Process Definition [BT03]

Step 1. Risk analysis
Rate the project’s environmental, agility-oriented, and plan-driven risks

Uncertain about ratings?

Yes
Buy information via prototyping, data collection, and analysis

No

Step 2. Risk comparison
Plan-driven risks dominate
Agility risks dominate
Neither dominate

Step 3. Architecture analysis

Step 4. Tailor life cycle
Tailor life-cycle process around risk patterns and anchor point commitment milestones

Step 5. Execute and monitor
Deliver incremental capabilities according to strategy
Monitor progress and risk/opportunities, readjust balance and process as appropriate

Note: Feedback loops present, but omitted for simplicity

Go risk-based agile
Go risk-based plan-driven
Go risk-based agile for agile parts; go risk-based plan-driven elsewhere
Overview

- Background and Motivation
- Software Project Risks
- Development Processes – Basics
- Agile Development
- Plan-Driven Development
- Balancing Agility and Discipline
- Process Models – Time for Tailoring
- Summary and Outlook
What is a Process Model?

**Definition: Process Model**

A process model describes systematic, quantifiable engineering approaches to solve tasks of specific classes repeatably. A process model is based on a number of principles. It consists of a product model, activity model, role model and workflow model and considers methods, (de facto) standards and tools to implement the principles of the process model.
Rational Unified Process

• Derived from several object-oriented analysis and design methods
• It is possible to tailor it toward a more agile or a more document centric process (Tailor down approach)
• Risk-driven spiral process
• Fundamental tenets:
  – Reduce size or complexity of what needs to be developed
  – Improve the development process
  – Create more proficient teams
  – Use integrated tools and exploit automation
• Four phases (each consisting of multiple iterations)
  1. Inception
  2. Elaboration
  3. Construction
  4. Transition
<table>
<thead>
<tr>
<th>Activity</th>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configuration Management</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project Management</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Rational Unified Process (RUP) is a process model for software engineering. It is divided into four phases: Inception, Elaboration, Construction, and Transition. Each phase is further divided into activities such as Analysis, Design, Implementation, Test, Configuration Management, and Project Management. The diagram illustrates the timing of these activities across the phases.
V-Model XT (Germany, 2004)

- The **V-Model XT** describes a **Process Model** to **Control and Organize system development projects** considering the entire system life cycle.

- The V-Model XT provides
  - concrete **Workflows** and **Activities** *(WHEN)*
  - the description of respective **Work Results** *(WHAT)*
  - and responsible **Roles** *(WHO)*

- The V-Model XT aims at
  - minimizing project risks
    - increase project transparency
    - improve the ability of project planning
    - facilitate project execution
  - increasing and guaranteeing result quality
  - reducing total cost over the entire system life cycle
  - improving communication between all stakeholders

- **Goal:** Improved controllability of the “Magic Triangle”
V-Model XT Philosophy: Goal and result oriented

- Products are in the center of interest. They form THE actual project results.
- Project execution scenarios and decision points determine the order of product completion and therefore the basic structure of the project progress.
- Detailed project planning and control is based on the editing and completion of products.
- Every product has an associated responsible role; in the project this is a defined person playing that role.
- Product quality is verifiable by checking the defined product requirements and explicit descriptions of dependencies to other products.
V-Model XT – What do you get?

• Model size
  – 3 project types (client, contractor, process improvement project)
  – 7 process execution strategies
  – 18 decision points
  – 18 process components
  – 99 products and 99 activities (each with multiple subjects/steps)
  – 72 explicit product dependencies of different types
  – 30 role descriptions
  – mappings to 6 standards, conventions (AQAP 150, CMMI, ISO 9000, ISO 15288, CPM, V-Model 97)
  – Tutorial, glossary, method and tool references, ...

• V-Model documentation and tool support
  – PDF/Word documents (635 pages)
  – HTML version
  – 76 generated Word product templates
  – Process editing tool (Open Source), process tailoring tool
Overview

- Background and Motivation
- Software Project Risks
- Development Processes – Basics
- Agile Development
- Plan-Driven Development
- Balancing Agility and Discipline
- Process Models – Time for Tailoring
- Summary and Outlook
Summary

• High-quality software is difficult to develop.
• Software complexity key problem.
• Various development processes have been devised to address aspects of software complexity
• One size does not fit all.
• It is possible to combine agility and discipline.
• Risk based assessment helps in creating the right mix.
• Analysis of real projects shows that often successful projects have applied a balanced mix of agility and discipline.
Literature


[GOF95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns – Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995


