Agile Software Engineering Practice to Improve Project Success

Dietmar Winkler

Vienna University of Technology
Institute of Software Technology and Interactive Systems

dietmar.winkler@qse.if.tuwien.ac.at
http://qse.if.tuwien.ac.at
Motivation

- The construction of high-quality software products requires (a) professional approaches (software processes), an appropriate set of methods, and well-trained engineers.
- Rapid and late changing requirements are success-critical challenges in common industrial projects because they have a strong impact on product quality, design, and project schedule.

Question

→ How can we address unclear, rapid and late changing requirements in (industry) software projects?

Topics

→ Structured Software Engineering Processes (e.g., V-Modell XT).
→ Agile Software Development Processes (e.g., SCRUM).
→ Software Development Practices (MDD / TDD / Pair Programming)
Why Requirements are Important …

The hardest single part of building a system is deciding what do build. (B.W. Boehm, 1997)

- Requirements represent the needs of the customer (what does he need?) from user/customer point of view.
- Requirements contribute to the solution of a real-world-problem. [SWEBOK, 2004]

→ A requirement is an expression of desired behavior from user perspective.

- Requirements management is the science and art of gathering and managing user, business, technical, and functional requirements within a product development project.

→ Requirements management deals with a set of requirements to handle complex systems.

Note: Requirements must be auditable and testable!
Impact of Requirements

- Reasons for project interruption - survey including 365 industrial responses (8,380 applications) [Chaos Report, 1994]:
  1. Incomplete requirements (13.1%)
  2. Lack of User Involvement (12.4%)
  ...
  6. Changing Requirements and Specifications (8.7%)
  ...

- Selection of “Top-Ten” risk items for project failure [Boehm, 1991]
  ...
  3) Developing wrong software functions.
  4) Developing the wrong user interfaces.
  5) Gold plating.
  6) Continuing stream of requirement changes.
  ...

→ Software Processes help to address requirements elicitation.
Software Life-Cycle

- The Software Life Cycle is a **general purpose process** including all process steps from the first idea to the retirement of a software product.
- A Software Process is a **subset of the life cycle approach** and defines the **sequence of steps** within the project course.
- Support of Software / Systems Development.
- Provide consistent guidelines, method and tool support, embedded within the process.

In common industrial practice, several different software processes emerged:
- Focus on specific **application domains** and **project types**.
- Limited to specific types of products and their attributes.
- Need for selection criteria for software processes.

![Software Life Cycle Diagram](chart.png)
Structured Software Engineering
Processes Example: V-Modell (XT)

Pro:
- Focus on deliverables (products)
- Different levels of abstraction (user, architects, programmers).
- Defect detection and prevention in early stages of development.

Con:
- Clear definition of system requirements necessary.
- Well-known application domain required.
- Focus on documentation (Documentation overhead).
- Critical on defects in early stages of software development.

Application:
- Large projects with clear defined goals and requirements.
Incremental Software Development

Note:
- analysis team
- design team
- implementation team

Build 1
Build 2
Build 3
Build n
Incremental Software Development

- Stepwise product development (several releases, builds)
- Continuous integration phases
- Small steps of development (planning of software increments)
- Planning of iterations including milestone definition after each development cycle.

**PRO**
- Unclear requirements.
- Long development duration.
- Quick delivery of (parts) of the system to customers.

**CON**
- Problems, if releases will not fit together.

Application
- Large and complex software systems.
- Project with long development duration
Some Questions …

- Structured and systematic software processes define the sequence of steps within a project course.
  - Is it always possible / reasonable to follow a strict process?
  - Does a structured process address rapid/late changing requirements?

- Structured processes (e.g., the V-Model 97) require comprehensive documentation.
  - Is a comprehensive documentation necessary all the time?

- Structured processes allow a detailed project plan because of the pre-defined steps over the whole project.
  - What happens, if modified/additional requirements occur in later stages of development?

- Typically software projects are based on contracts (based on a detailed specification).
Agile SE Approaches\textsuperscript{1}

- 4 Key Value Aspects of Agile Software Development
  - Individuals and interaction over processes and tools
  - Working software over comprehensive documentation.
  - Customer Collaboration over contract negotiation
  - Responding to change over following a plan.

- Key Principles (Selection):
  - Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
  - Welcome changing requirements, even late in the development.
  - Deliver working software frequently.
  - Collaboration of business people and developers.
  - Simplicity of the solution.
  - The best architectures, requirements, and designs emerge from self-organizing teams.
  - etc.

\textsuperscript{1}http://www.agilemanifesto.org/
SCRUM

- Agile Software Process from Project Management (PM) view.
- SCRUM is not an acronym; it is based on the scrum formation in Rugby sports.

Characteristics:
- One team builds one unit.
- Clear distribution of work.
- Clear priorities of project deliverables (backlog items).
- One common goal (= delivery of the product)
- A “Sprint” is a central element.
- Temporal structure = daily Scrum Meeting + Review + Retrospective.

Basic Roles:
- Product Owner
- (Self-organizing) Team
- Scrum Master
SCRUM Phases

PRE-GAME

Develop

Adjust

SPRINT

Sprints

Wrap

Review

POST-GAME

Planning & System Architecture

Closure
**SCRUM Sprints**

- Scrum represents a set of procedures, roles and methods for project management.
- Agile software development
- Self-organizing teams.

Every Day, a 15-min meeting is held, and the SCRUM Master asks 3 Questions:

1) What have you accomplished since the last meeting?
2) Are there any obstacles in the way of meeting your goals?
3) What will you accomplish before the next meeting?
SCRUM – Definition of terms

- **Backlog**: All work to be performed in the near future, both well defined and requiring further definition.

- **Sprint**: A period of 30 days or less where a set of work will be performed to create a deliverable.

- **Sprint Backlog**: A set of defined work packages for a sprint duration of about 1 month (incremental deliverables). No or only a few changes are possible.

- **Scrum**: A daily meeting for progress discussion to clarify questions and to remove obscurities.

- **Scrum Meeting rules**: Protocol for effective Scrum daily meetings.

- **Scrum Team**: The cross-functional team working on the sprint's backlog.

- **Burndown Chart**: Graph that represents the project progress.
Agile Practices

- Software processes require **suitable methods** to support engineers in constructing high-quality software products, e.g.,

  - Model-Driven Development.
  - Test-Driven Development.
  - Pair Programming.
Model-Driven Development

- Software Engineering requires the construction of consistent views on the system.

- Models support to keep an overview on the system and its components.
  - Basis for effective and efficient team work.
  - Common notation (language) and consistent meaning (different stakeholders / domains typically use their own „language“)
  - Basis for automation (e.g., automated code generation, test case generation, testing, etc.)

- Examples (based on the UML 2¹ diagram family):
  - 6 Structural diagrams: e.g., component, package, and class diagrams
  - 3 Behavioral diagrams, e.g., activity diagrams, state charts, use cases.
  - 4 Interaction diagrams, e.g., sequence and timing diagrams.

UML = Unified Modelling Language
> Notation for Modeling a System
> describes statical and dynamical aspects
> Note: Perspective selection depends on the type of the system
## Model Driven Development

<table>
<thead>
<tr>
<th>Real World Problem</th>
<th>Problem Definition</th>
<th>Models (e.g., based on the UML Family)</th>
<th>Modeling</th>
<th>Transformation</th>
<th>Automation</th>
<th>Artifacts (e.g., Code)</th>
<th>Realization</th>
</tr>
</thead>
</table>

- **Description of real-world problems in a common language (e.g., UML notation).**
- **Basis for communication between consumers and developers.**
- **Translation**
  - From models to artifacts (e.g., code)
  - From models to other models
  - Etc.
- **Basis for automation (e.g., deriving software code and test cases based on models)**
Test-Driven Development (TDD)

- Goal: Every feature in an application that gets implemented has to be testable.
- Testing either automatically using unit tests, automated UI tests, etc. or manually executed by following a predefined test plan.
- Test comes before or parallel to the implementation.
- Traditional Testing Approaches based on test strategies and plans:

  ![Diagram](image)

  - Finding Bugs becomes difficult, particularly in a team-environment
  - Shorter Cycles are quite better.
Test-Driven Development (TDD) (2)
Test-Driven Development (TDD)

- Unit Tests: construction of executable test cases.
- Derive assertions for test case execution (expected results)
  - Normal case: should be handled without problems.
  - „correct defect“: should be handled by the system (predictable exception handling)

Basic TDD „Process“:
- Identify the component / class
- Write Test cases (e.g., JUnit)
  - Execute Test cases → Test cases should fail
- Implement the component/class
  - Execute Test cases → Test cases should be successful.
- Cleanup code
Pair Programming

- Pair Programming (PP) is a common practice in the area of agile software engineering.
- PP focuses on the construction of software code.
- PP involves two engineers (“Power of Two Brains”)

**Typical Roles and tasks**

- **Driver:**
  - responsible for code implementation.
- **Observer:**
  - supports the driver by observing his activities.
  - keeper of the focus and the pace of the current tasks.
  - performs implicit quality assurance activities (e.g., continuous reviews)
- The role assignment (driver and observer) **should change frequently**.
Pair Programming Pros & Cons

Reported benefits of PP vs. Solo Programming
- More disciplined (2 persons involved)
- Improved software code and higher code quality (implicit quality assurance)
- Improved productivity (change of roles)
- Collective code ownership (2 persons involved)
- Mentoring & learning (e.g., senior/junior as “pair”), …

But …
- Additional effort (2 persons involved)
- Possible authority problems.
- Team building might be difficult
- Copyright issues
- What are the deliverables of the observer? …

- Nevertheless, PP is a promising approach for the delivery of high-quality software products (e.g., reported from industry and academic studies)
Next Steps in Pair Programming\(^1\)?

- **“Pair X”: Enhancing Pair Programming.**
  Application of “Pairs” to other software engineering activities, e.g., Pair Requirements Analysis, Pair Design, Pair Testing …
  - Will the involvement of two persons increase productivity and quality in these areas?
  - Are pair activities facilitators for learning, training and mentoring of juniors?
  - Empirical studies will provide answers to this question.

- **IPP: Integrated Pair Programming**
  Extending Pair Programming with systematic Quality Assurance to enable repeatable, traceable and auditable software products required by several application domains e.g., security and safety-critical systems.
  - Currently, the deliverables of the observer are unclear.
  - Systematic QA (e.g., inspection) enable traceable, repeatable, and auditable software products.
  - The integration of constructive and analytical method might bring up benefits of different disciplines.

\(^1\)Ongoing / planned projects at TUW.
Summary

Software Processes:
- Requirements are success-critical in software engineering projects.
- Structured software processes typically require stable requirements because of a sequential order of process steps with limitations of process backtracking.
- Agile approaches focus on a tight customer involvement, small iterations and support frequent changing requirements.

Agile Practices:
- Models present real-world scenarios, support communication between consumers and developers (common language), and are the basis for automation (e.g., automated code generation based on models).
- Test-Driven Development (TDD) focuses on the generation of test cases before or (at least) in parallel to the development of software code.
- Pair Programming is a team activity - involving two persons - to increase productivity and software quality and supports learning.

- The application of agile software development processes and practices promises to support the construction of high-quality software products with respect to frequent changing requirements.
References

Books & Papers:

Web References:
- V-Modell XT; http://www.v-modell-xt.de.
Thank you for your attention

Contact:
Dipl.-Ing. Dietmar Winkler

Vienna University of Technology
Institute of Software Technology and Interactive Systems
Favoritenstr. 9-11/188, A-1040 Vienna, Austria

dietmar.winkler@qse.ifstuwien.ac.at
http://qse.ifstuwien.ac.at
This research work has been supported by a Marie Curie Transfer of Knowledge Fellowship of the European Community's 6th Framework Programme under the contract MTKD-CT-2005-029755: CzechVMXT.