Best-Practice Software Engineering: Software Processes to Support Project Success

Dietmar Winkler

Vienna University of Technology
Institute of Software Technology and Interactive Systems

Dietmar.Winkler@qse.ifs.tuwien.ac.at
http://qse.ifs.tuwien.ac.at/

Sixth Framework Programme

Information Society Technologies

Introduction

- How can we achieve better software projects / products?
- How can we improve collaboration within a (distributed) software development team?

→ The application of a defined software process enable the construction of high-quality software products within a software development team.

Table of Contents:

Motivation and Goals

- Software is a major part in our daily life (e.g. commercial systems, embedded systems, web applications, agents, etc.)
- Increasing complexity of projects (e.g., regarding size, functionality, technology) and growing (distributed) teams require professional processes.
- Systems engineering was traditionally focused on mechanical and electrical engineering with only little software engineering. Nowadays, software gets increasingly a larger part of techn. systems → need to bring in existing solutions from software engineering research and practice.
- Software processes help to construct valuable high-quality software products because of a disciplined, structured approach.
- Different projects require different process approaches → decision support for selecting appropriate processes.
- Methods and Tool support engineers in conducting successful projects and deliver valuable products.

Software Engineering Goals

- Major objective in software engineering is the delivery of high-quality software products.

- Examples:
  - Compliance of the software solution with customer requirements.
  - Minimum number of remaining defects within the software product.
  - Product delivery in time and budget.
  - ...

- To achieve these goals, we need
  - Suitable constructive approaches to enable the construction of products (e.g., software processes on organizational level, methods and tool on engineering level).
  - Suitable analytical methods to verify and validate the solution towards the specification (verification) and customer requirements (validation).

Project Classification (Application Domain)

- Different application domains include a various requirements.

Project Type	Requirements	Examples
Commercial Software	usability, availability, support	database transactions
Embedded / Real-time Systems	time-driven, safety & security, real-time requirements	Cell phones, ABS, lift control
Scientific Software	computational accuracy, correctness, reliability	Medical and aerospace applications
Computer Games	usability, functionality, efficiency	
Web Applications	usability, security, availability	Web Shops

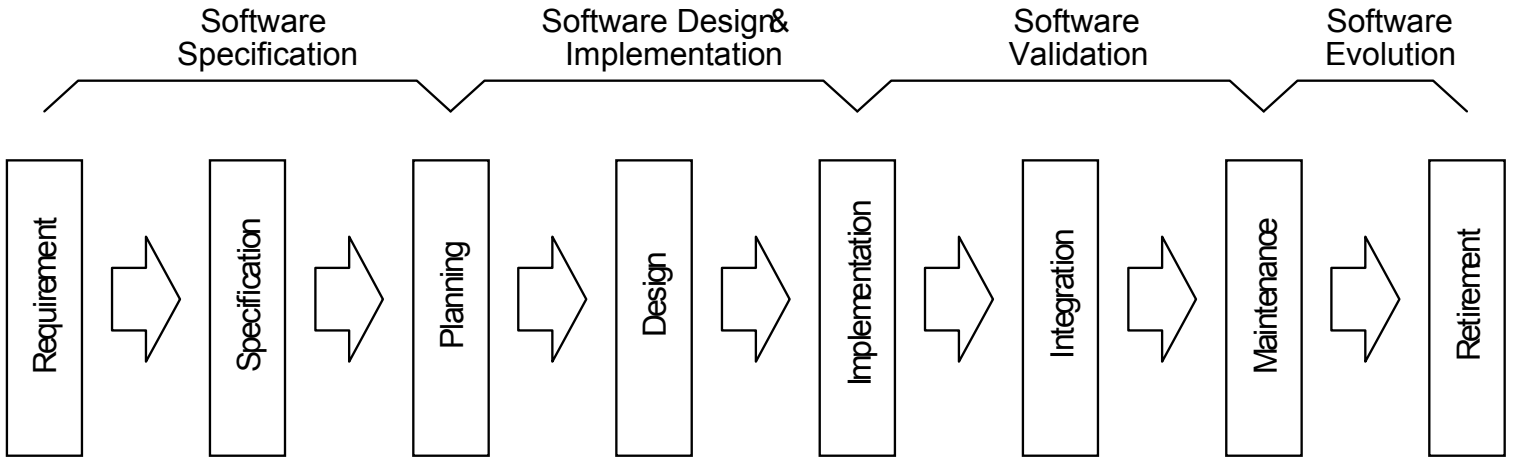
Project Classification (Project Size)

Size	Criteria	Examples
Small	Up to 6 persons 0-8 person months (PM) Number of technologies: <5	Calculation problems, algorithms
Medium	10-30 persons 9-24 PM Number of technologies: 5-12	Accounting applications, Stock management
Heavy (large)	50-100 persons 25-45 PM Number of technologies: 12-20	Compiler, database
Super Heavy	100+ persons >45 PM Number of technologies: >20	Aerospace, nuclear power plant, electronic brokerage

- Depending on the project, engineers have to apply a suitable software process.

Software Life Cycle

- A software process model is a sequence of steps involving activities, constraints, and resources that produce an intended output.
- The software life-cycle describes a basic approach for a software engineering process from the conceptual phase, via design, implementation, operation and maintenance, until the retirement of the software product.



Software Life-Cycle Process (2)

- Requirements represent the needs of the customer (what does he need?) regarding the software product (user/customer view).
- A specification describes the system in a technical way (engineering view).
- Planning: Definition of the project course according to time, duration, deliverables, and cost (project manager).
- Design: Detailed technical solution of the system requirements, including modularization, components, packaging, etc.
- Implementation considers the construction of the software product. (coding/testing).
- Integration: assembling and testing of software components.
- Operation and Maintenance: Defect correction, support, extensions of the software solution.
- Retirement: Replacement of software products, if they are obsolete.

Software Requirements

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 !

Why requirements are important …

- **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.

  …

We have to know, what the customer needs, to construct the right system!

Basic Requirements Classification (1)

- Functional requirements
  - Services (operations) of a system (which problem should be solved?).
  - Functional behavior (system responses on defined input parameters).
  - Data formats (Input and Output). etc.

- Non-functional requirements
  - Performance: e.g., information flow-rate.
  - Usability and human factors: e.g., required user training, simplicity of applications.
  - Security: e.g., access control, separation of application and data.
  - Reliability and availability: e.g., Backup strategies, system recovery mechanisms.
  - Maintainability: Simplicity to modify/add features.
  - Time-to-Delivery / Cost: predefined project schedule, budget limitations, etc.

Basic Requirements Classification (2)

- Design Constraints
  - Physical environment: e.g., development environment, co-located vs. distributed development.
  - Interfaces
    - Need for communication between different systems.
    - Data format definitions for communication.
  - Users
    - User target group (experienced users, less experienced users).

- Process Constraints
  - Resources: e.g., material, developers, skills of engineering staff.
  - Documentation: e.g., type of documents (electronically, printed, etc.), target audience.

Stakeholders

- Depending on the role, the software product must meet requirements according to their individual expectations.
  - Clients / Customers pay for the software product
    → Cheap and fast system delivery, etc.
  - Users will operate on the software system
    → Functional requirements, non-functional requirements (e.g., usability, simplicity, stability), etc.
  - Developer, e.g., software engineers, technology experts, will design and construct the software system
    → Latest technology, “gold plating”, etc.
  - etc.
- Major goal is to develop and deliver a software system that meets the requirements of important stakeholders, according to function, non-functional requirements design constraints, and process constraints (requirements elicitation).

Excursus: Verification and Validation

- Software defects have a heavy impact on project quality, project duration and project budget.
- Rework effort increases the later a defect is detected within the project course.
- Thus, a major goal is to identify and correct defects (deviation of a solution and its specification / expected behavior) as soon as possible.
- Verification vs. Validation:
  - Verification is a quality assurance approach to find out, if the product is in accordance with the specification, i.e., did we create the product in the right way.
  - Validation is a quality assurance approach to finde out, if the product is in accordance with the user requirements, i.e., did we produce the right product.

Excursus: Testing

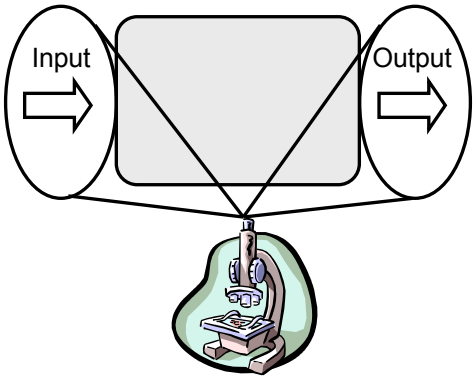
- Software testing is an analytical quality assurance approach for software product improvement.
- Testing is considered with program execution to find deviations and defects.
- In traditional software processes test cases are generated in early cycles of development (e.g. in the analysis / design phase) and they are executed during / after software implementation (module, integration, acceptance tests).
- Flexible and agile software processes include test case generation and execution at the same time (e.g., test driven development).

Excursus: Basic Test Principles

"Testing is a quality assurance activity in order to find defects".

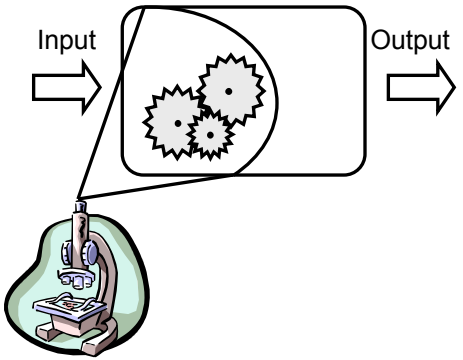
Black Box Tests

- Based on the specification document.
- Independent on the realization of the module.
- Data-driven (Input/Output).
- Requirements coverage.
- Equivalence classes of input data.
- No defect localization possible.



White Box Tests

- Based on software code.
- Knowledge of internal representation necessary.
- logic-driven tests.
- Control-flow coverage.
- Equivalence classes of internal branches and loops.
- Enables defect localization.



Software Life Cycle vs. Software Process

- 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.
- Software processes define the sequence of steps within the project course.
- Most of them focus on the technical part and start at the requirements definition phase and end with the deployment of the solution at the customer site.
- In industrial practice exist a wide range of different software process approaches with emphasis on project related criteria.
  - Standardized software processes, e.g., Rational Unified Process, V-Modell XT, Scrum, Incremental Development Processes, etc.
  - Customized / company-wide software processes, e.g., stdSEM by Siemens PSE.

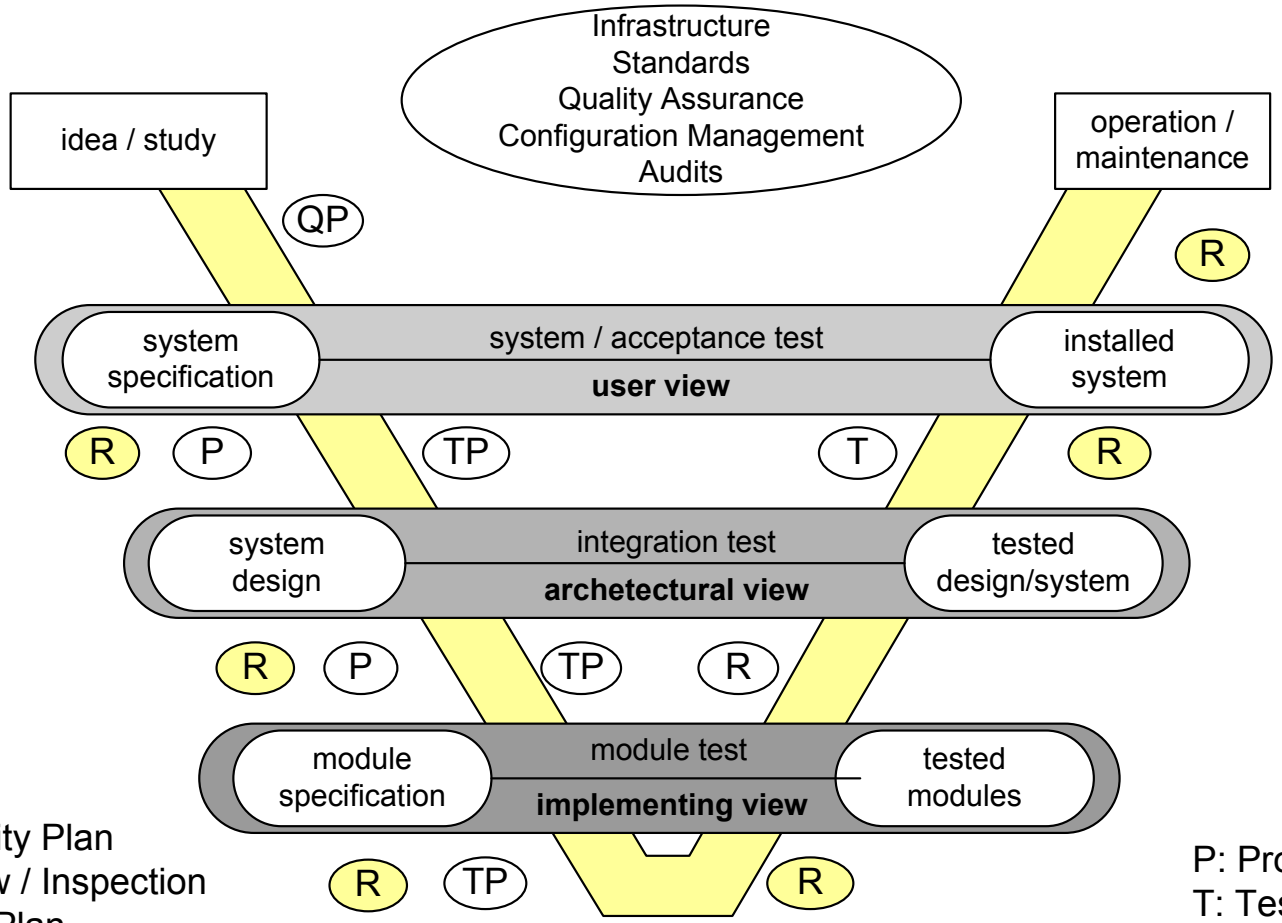
Process Model Selection

Selection of a applicable software process model / framework depends on:

- Project Types (e.g., commercial system, embedded system)
- Project Size (e.g., „small“ vs. „super heavy projects“)
- Project Duration vs. Project Effort.
- Applied Technology (New vs. approved Technology)
- System complexity
- Risk (e.g., New vs. well-known application area)
- Roles (Distributed vs. Co-located development teams)

Selection of the „best-practice“ software process approach is not simple!

The Technical V-Model Concept



QP: Quality Plan
R: Review / Inspection
TP: Test Plan

P: Prototype
T: Testing

Basic V Model Concept (2)

PRO

- Specification phase vs. realization and testing.
- Context of products and tests.
- Different levels of abstraction (user, architectural and implementation view).
- Defect Handling in early stages of software development because of reviews at different milestones.
- Basic Concept for VM 97 and VM XT.

CON

- Clear definition of system requirements necessary.
- Well-known application domain required.
- Documentation overhead.
- Still critical on defects in early stages of software development.

Application

- Well-known application domains.
- Large projects in the public sector.

V-Modell XT (VM XT)

- The V-Modell XT (VM XT) is based on the basic V-Model concept.
- VM XT is the mandatory SE process standard for public projects in Germany.
- Addresses the responsibilities of the system acquirer (customer) and producers.
- VM XT supports Value-Based Systems Engineering by considering requirements of suppliers (producers) and systems acquirers (customers).
- VM XT supports iterative systems engineering processes
- Process modules encapsulate products, roles, and activities.
- VM XT provides a flexible basic system process for development projects without restriction to a specific application domain.
- Process modules include guidelines for hard/software products, logistics, security, etc.
- VM XT allows a flexible arrangement of mandatory and optional process modules (tailoring and customization).
- Open source tool support for process tailoring and customizing is available.

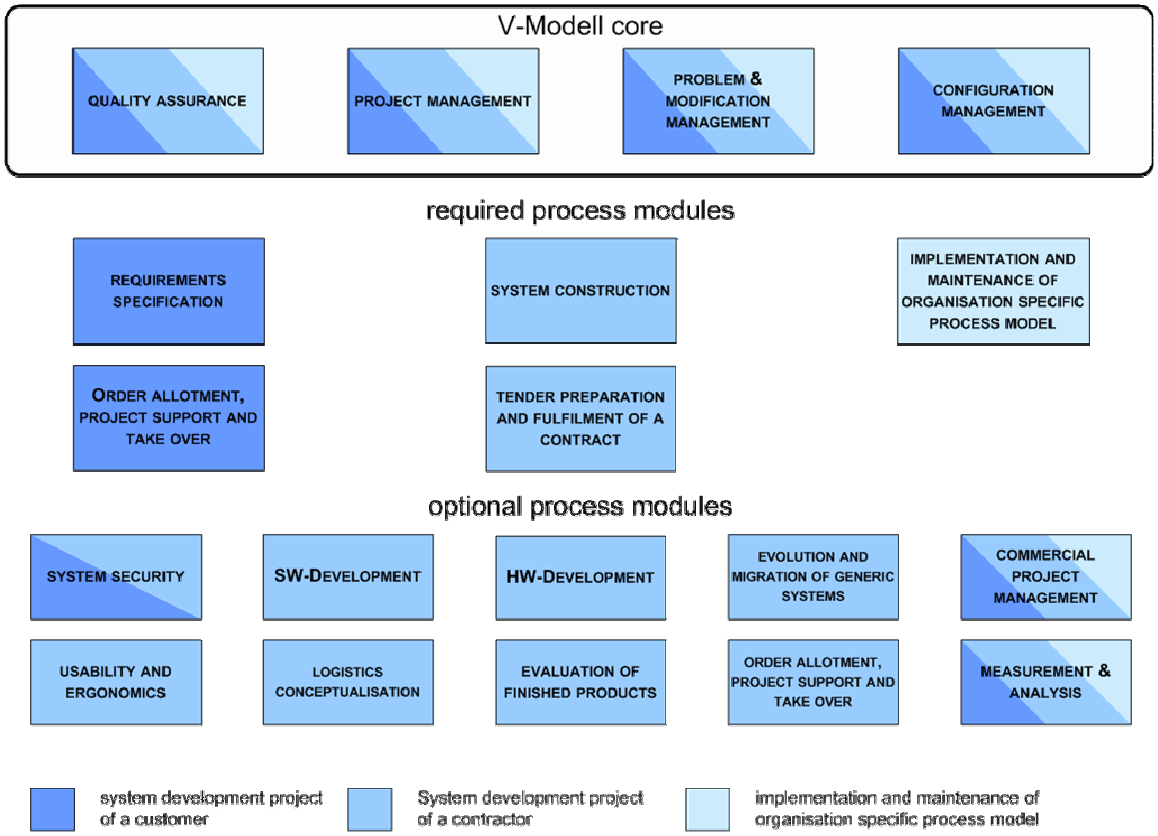
V-Modell XT – Basic Components

V-Modell XT is a systematic framework for development, planning and process improvement.

- Project Types (from customer / supplier point of view).
- Products, Activities, Roles encapsulated within process modules.
- Integrated method and tool support linked to
  - products (generation), activities (proceeding) and
  - roles (who is responsible for the product).
- Process Modules including
  - *core components* and *optional elements* to meet the individual requirements of the application domain.
- Decision gates represent the state of treatment.
- Project Operation Strategy as a defined sequence of decision gates for project course.
- Mapping strategies for application of common software processes (reuse of approved industrial practices)

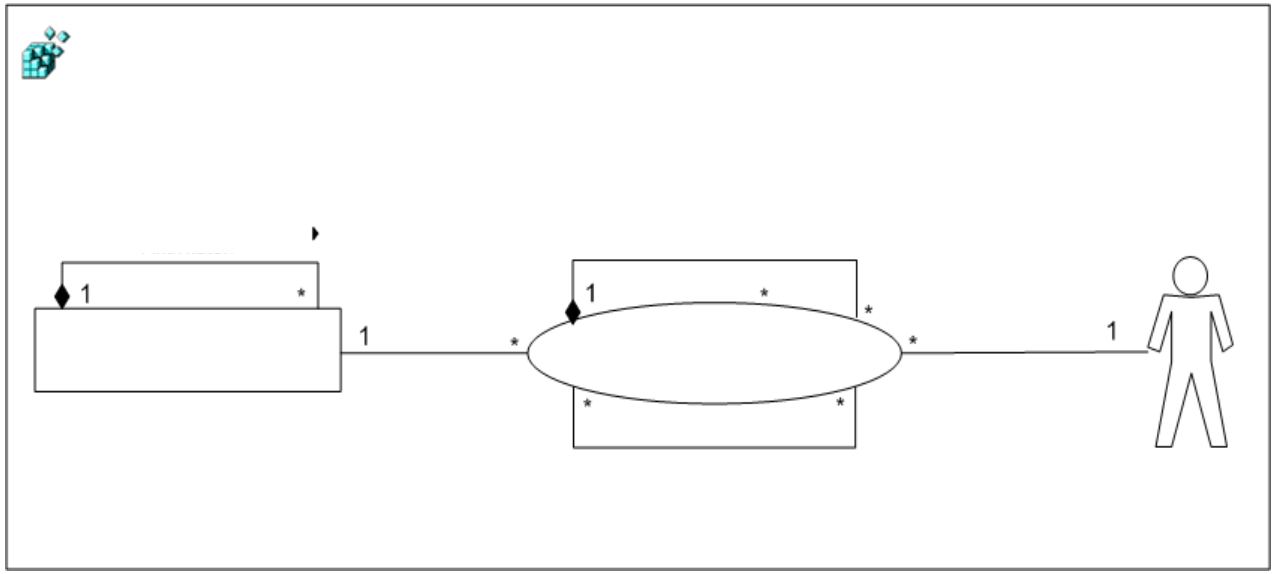
V-Modell XT – Process Module Overview

- Core Modules (mandatory for all project types)
- Required Modules (depending on the project type and application domain)
- Optional Modules (depending on the application domain)
- Selection support by V-Modell XT Project Assistant tool.



VM XT Process Module Concept

- Process Modules are the basic elements of the V-Modell XT.



- A Process module
  - Encapsulation of roles, products, and activities.
  - Independent component (maintainability) for application purposes.
  - Defined interfaces to be replaced in case of updates or extensions.

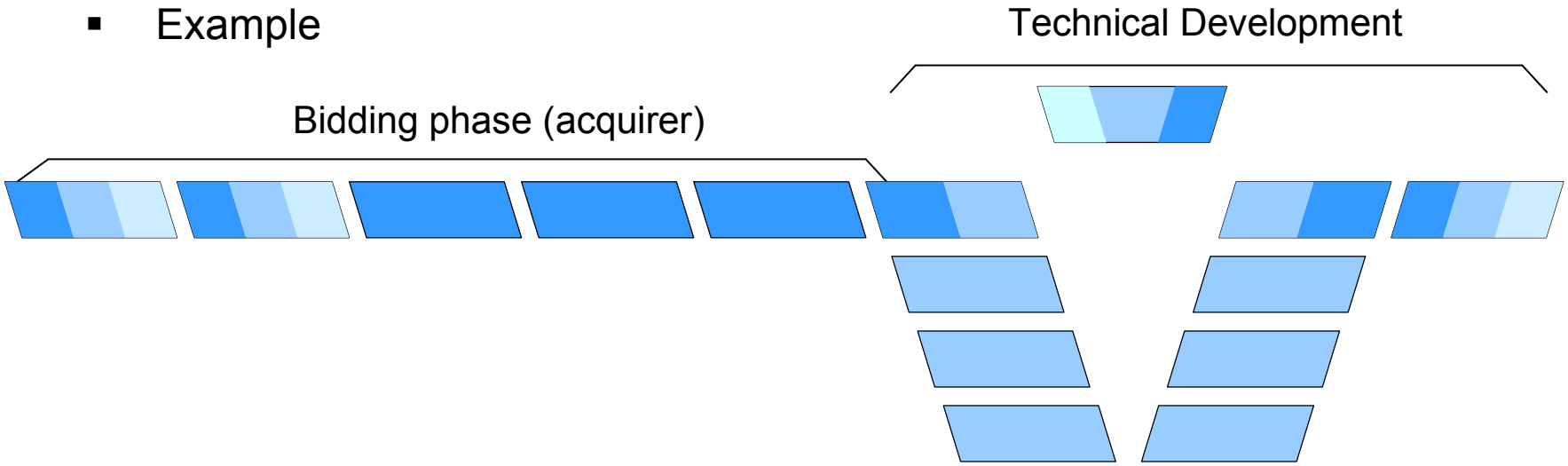
Process Module

Contains sub

Project Execution Strategy based von VM XT

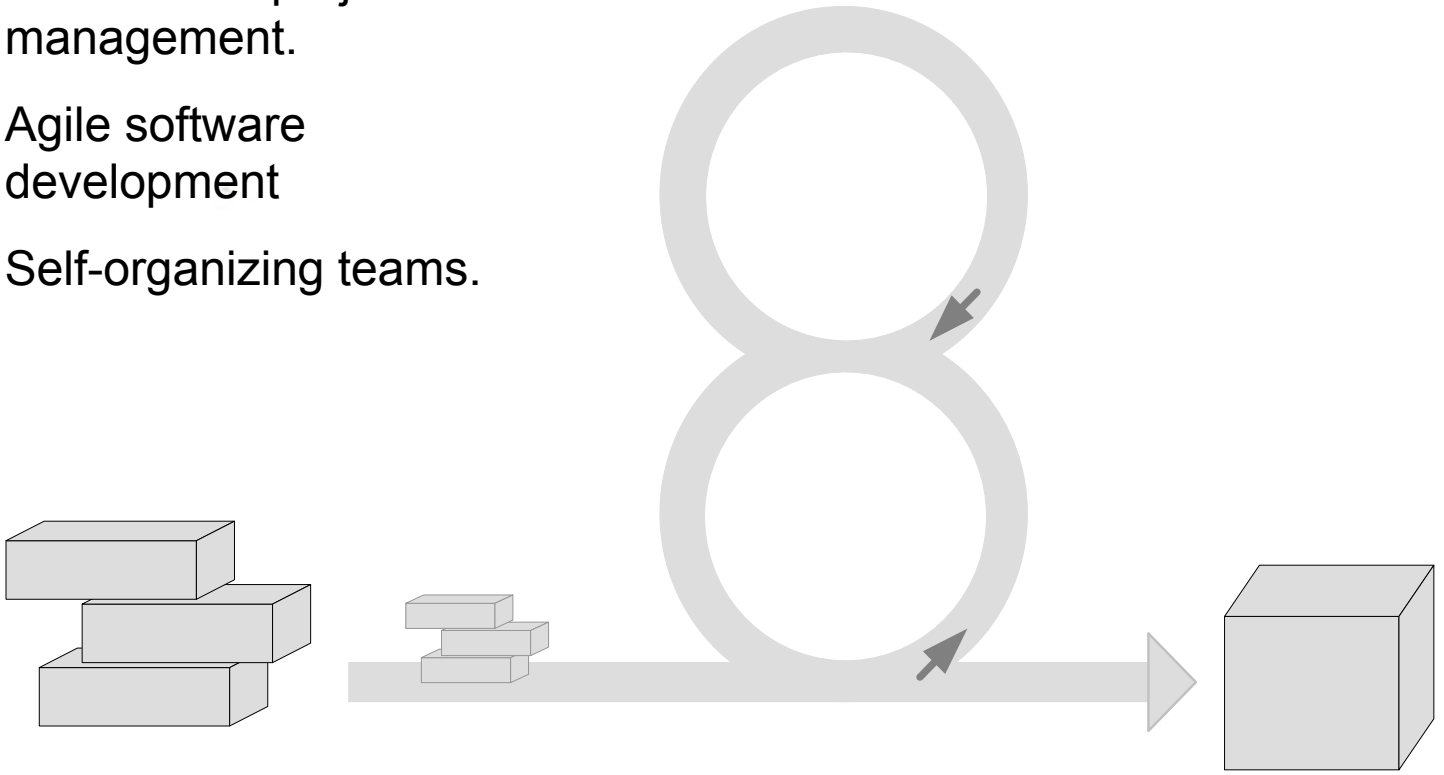
Selection of an Execution Strategy (including Decision Gates)

- Definition of the sequence of decision gates (comparable to milestones)
- Decision Gates require a defined set of products (linked to process modules)
- Example

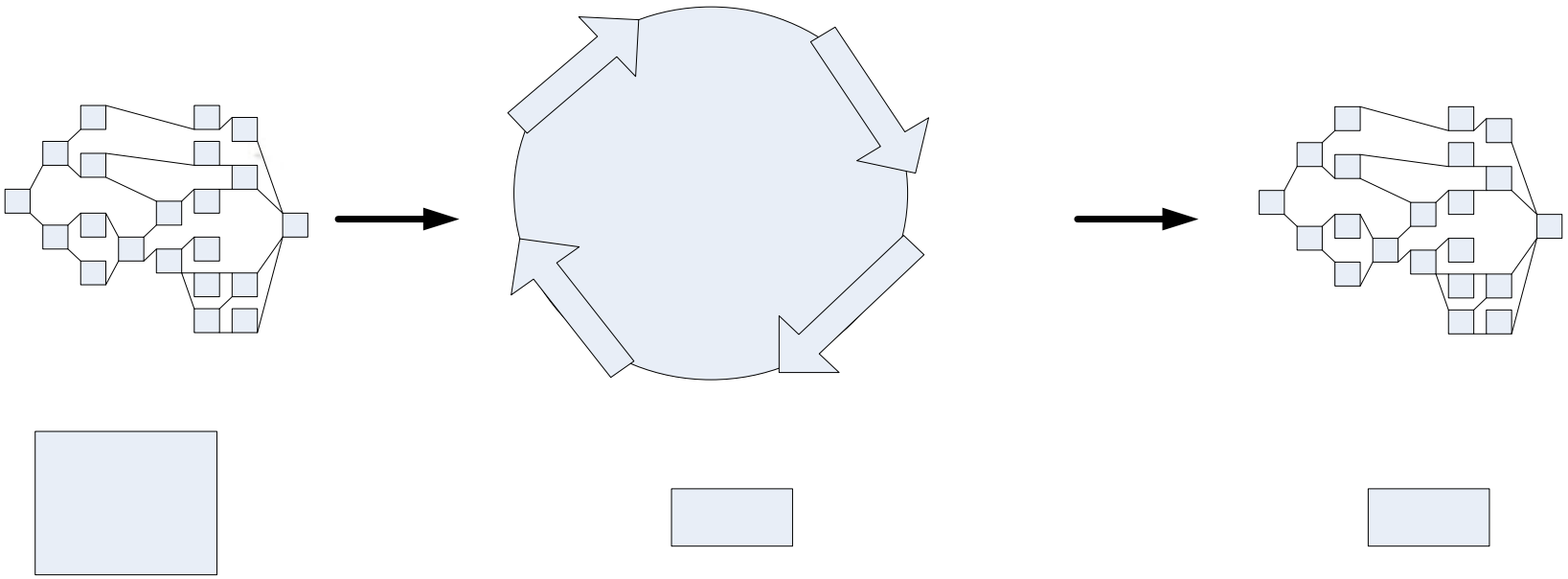


SCRUM

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



SCRUM – Phases (1)



Develop

SCRUM – Phases (2)

- Pregame
  - Definition of new release based in product backlog.
  - Design how backlog will be implemented.
  - Estimate time and costs (deliverables).
- Sprints:
  - Typically 1-4 weeks (depending on product complexity and risk)
  - Multiple iterative sprints to construct the system (4 steps: development; wrapping; reviewing; adjusting).
  - Interaction with variables of time, requirements, quality, costs and competition define the end of this phase.
- Postgame
  - Preparation for release, pre-release staged testing & release.
  - Closing all issues for the current release.

SCRUM – Characteristics

- The flexible process model approach enables immediate respond to changed requirements during the project course.
- The iterative approach enables earlier delivery of product parts (e.g. components).
- Project content is determined by the environment variables *time*, *competition*, *cost*, and *functionality*.
- Deliverables depend on market information, customer contact, and skill of developers.
- Small but multiple teams (if necessary).
- Scrum defines frequent reviews of functional executables.

- The last couple of years Scrum became more popular, as the software market requires very quick new software products, e.g. for mobile communication.

SCRUM – Vocabulary

- 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.

SCRUM – Roles

- Product Owner (comparable to the project manager):
  - defines goals, deliverables and is responsible for backlog items.
  - Release management.
- Team (3-6 members):
  - Estimation of backlog item effort, implementation,
  - self-organizing teams.
- Scrum Master:
  - organization and observation of detailed planning and development processes.
  - He is no member of the SCRUM team!
- External stakeholders:
  - Customers
  - Marketing
  - Sales

Conclusion

- The construction of high-quality software products require professional processes.
- The software life-cycle process includes a sequence of basic steps from the first idea to the retirement of a product.
- Requirements represent the view of the customer and must be auditable and testable.
- The basic V-Model concept enables is a systematic and structured project course including several views on a software product.
- The V-Modell XT is a mandatory software process model for public IT project in Germany.
- Scrum is a flexible and agile software process with the ability to respond to frequently changing requirements due to tight customer interaction.

Recommended Literature

- Barry Boehm: Software Risk Management: Principles and Practices, IEEE Software 8(1), pp32-41, 1991.
- Ian Sommerville: "Software Engineering", 8th edition, Addison-Wesley Longman, 2007.
- Kruchten Philippe: The Rational Unified Process, An Introduction, Third Edition, 2004, Addison-Wesley.
- SWEBOK: Guide to the Software Engineering Body of Knowledge, http://www.swebok.org, 2005.
- Software Engineering – Best practices: http://best-practice-software-engineering.blogspot.com/
- SCRUM: www.controlchaos.com, last access: August 2007.
- V-Modell XT: www.v-modell-xt.de, last access: August 2007.

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.ifs.tuwien.ac.at
http://qse.ifs.tuwien.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.