

1. Praktikum — Software Engineering

<i>Lecturer:</i>	Prof. Oscar Nierstrasz Schützenmattstr. 14/103
<i>Tel:</i>	631.4618
<i>Email:</i>	Oscar.Nierstrasz@iam.unibe.ch
<i>Assistants:</i>	Matthias Rieger David Jud, Adrian Kuhn, Michael Locher
<i>WWW:</i>	www.iam.unibe.ch/~scg/Teaching/PSE ➡ <i>Requirements</i> documents ➡ PSE <i>wiki</i> ➡ Entry point to <i>team pages</i> ➡ Pointers to <i>technical documentation</i>

Agenda

- ☐ Introduction to PSE (Prof. Nierstrasz)
- ☐ Client presentation (Dr. Guenter)
- ☐ Student presentations (20 seconds each)
- ☐ Form teams (break)
- ☐ Schedule meetings

Overview

- ❑ Goals of this workshop
- ❑ Project overview
- ❑ Schedule — milestones, deliverables
- ❑ Analysis and Design documents: guidelines
- ❑ Prototyping — reducing risk
- ❑ Testing — coverage and regression tests
- ❑ Tools — UML, cvs, ANT, SNIFF+,...
- ❑ Teamwork — roles and responsibilities

Goals of this Workshop...

Methodological skills	<i>Practising</i> Responsibility-Driven Design
	<i>Evaluating</i> Implementation Strategies
	<i>Planning</i> and <i>Reporting</i>
	<i>Prototyping</i>
Practical skills	Working with <i>open</i> requirements (setting scope...)
	Developing a <i>complete product</i> (documentation, installation...)
	<i>Teamwork</i> (division of labour, planning, collaboration...)
Technical skills	Enterprise Java Beans
	UML
	<i>Testing</i>

The Customer

The *Swiss Federal Institute of Intellectual Property* is responsible for all matters relating to Intellectual Property in Switzerland. It was founded in 1888. The Institute's job is to publicise intellectual property legislation — and particularly its advantages — in Switzerland.



It is only too clear that intellectual works and creative products can be easily copied and misused. It is the task of immaterial property law to prevent this by offering effective protection.

The Institute wants to *automate administration* for these immaterial property laws. The IT department of the institute wants to base the architecture of its in-house software on *open standards* and has chosen *EJB/Corba*.

Project Characteristics

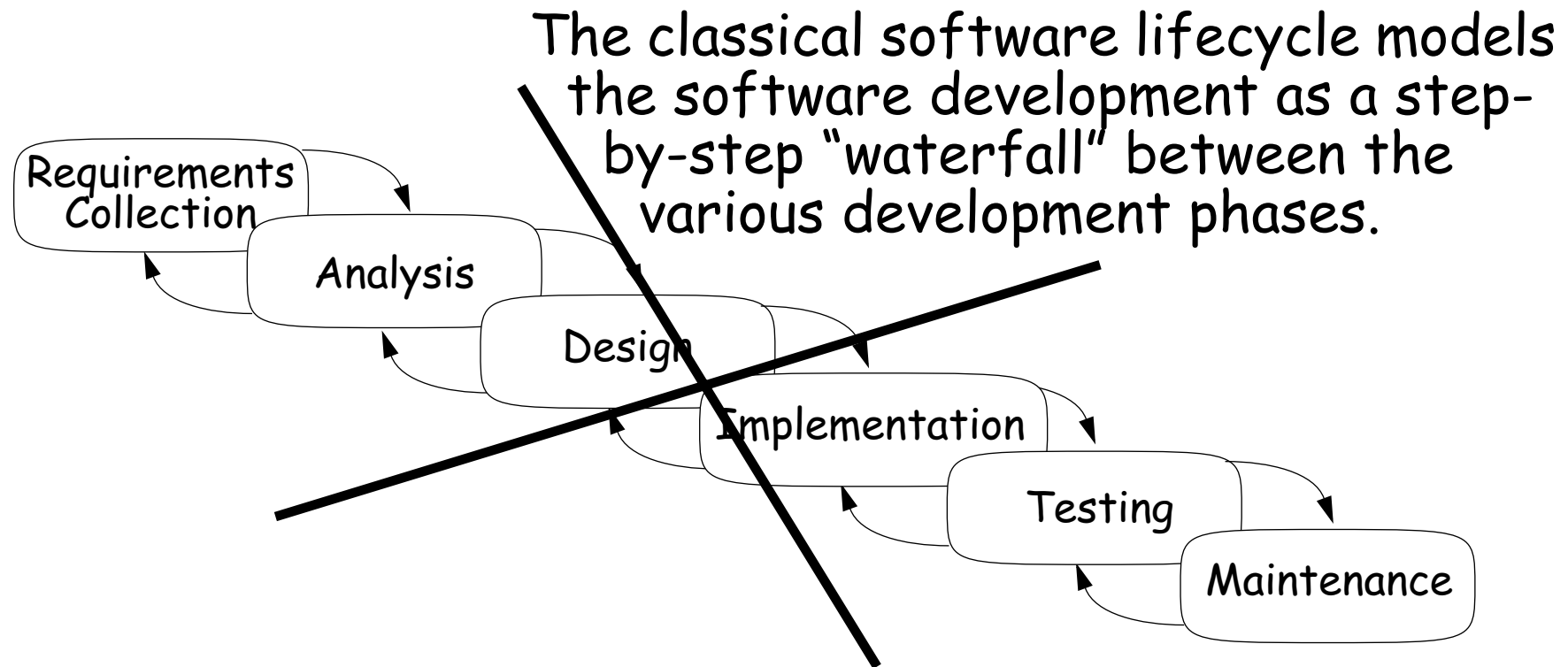
Several characteristics of “real” projects:

- ☐ *Open, changing* requirements
- ☐ Stringent *non-functional* requirements
- ☐ Must connect to an existing *relational DB*
- ☐ Solution will be built using *standard* packages
- ☐ Multiplatform
- ☐ Project *lifetime* extends beyond the end of the course!

Non-issues:

- ☐ Requirements collection, done by the client
- ☐ Communication: client is an IT department itself.
- ☐ No integration with existing applications or legacy software

The Classical Software Lifecycle

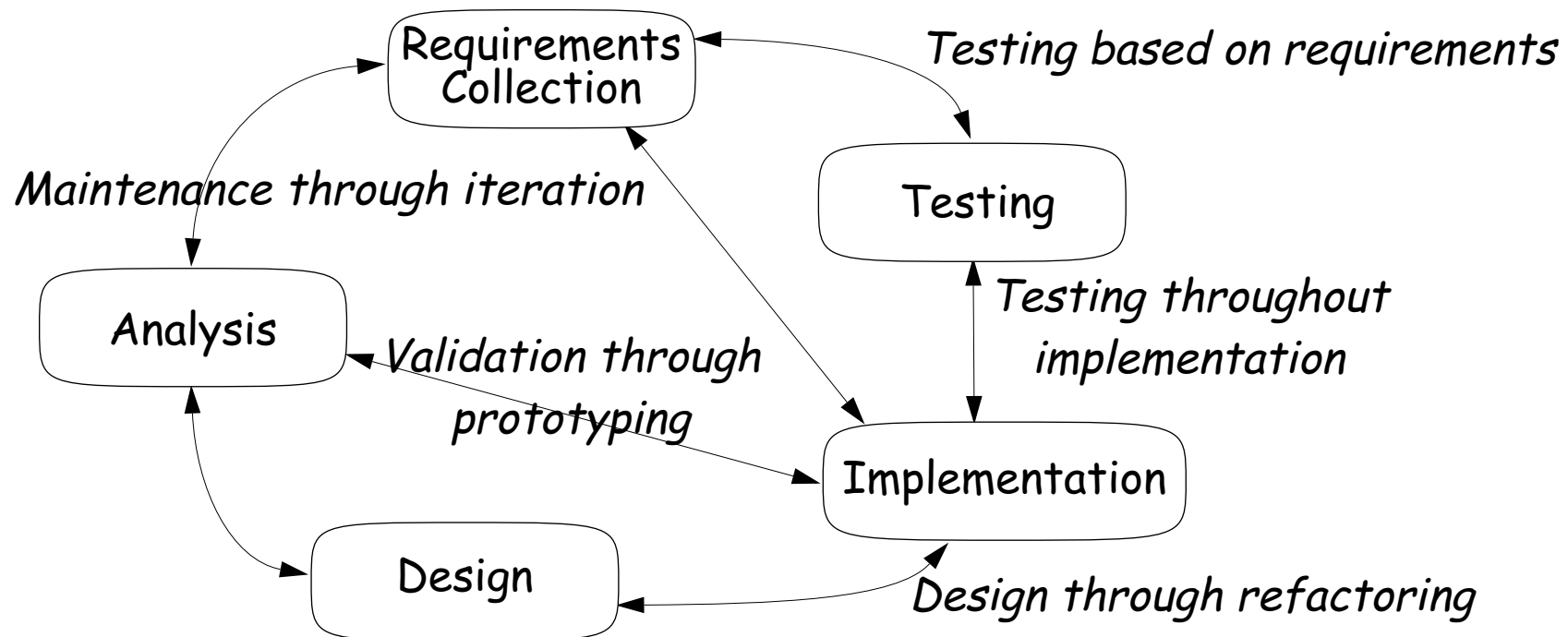


The waterfall model is unrealistic for many reasons, especially:

- ❑ requirements must be "frozen" too early in the life-cycle
- ❑ requirements are validated too late

Iterative Development

In practice, development is always iterative, and *all* software phases progress in parallel.



Preliminary Schedule

Meeting / Deliverable

Homework/Consultation

1	2001 03-28	Introduction; Presentation of client and project; <i>Team forming</i>	Study project descriptions & software documentation; prepare <i>risk analysis</i> & workplan; prepare interview questions
2	04-04	<i>Presentation</i> of Risk Analysis <i>Deliver workplan v1</i>	Implementation and testing...
3	04-11	<i>To be defined ...</i>	
4	04-18	End of first iteration: <i>DEMO</i>	Revise workplan for second iteration ...
5	04-25	Planning Game for second iteration	
6	05-02	Deliver workplan v2	Implementation and testing ...
7	05-09	<i>To be defined ...</i>	
8	05-16	End of second iteration: <i>DEMO, at client site</i>	Revise workplan for third iteration

*Meeting / Deliverable**Homework/Consultation*

9	05-23	Planning Game for third iteration	Product revision ...
10	05-30	<i>To be defined ...</i>	<i>Review test cases</i> (other team) (planned)
11	06-06	Present test case review (planned)	Product revision ...
12	06-13	End of third iteration: <i>Final DEMO, at client site</i>	Review final product (other team); finalize architecture documentation
13	06-20	<i>Deliver full architecture</i> documentation	Revise final product
14	06-27	<i>Deliver revised final product;</i> <i>Feedback and Testat</i>	

This schedule will be revised as the workshop progresses...

Evaluation

Every Team must:

- ❑ provide deliverables of *acceptable quality*
 - deliverables will be *reviewed* by the client or another team
 - unacceptable deliverables must be revised

Every Team Member must:

- ❑ *assume responsibility* for and present at least one deliverable
- ❑ *contribute "fairly"* to the team effort

Deliverables

Group web pages

- ❑ each group will have a group account for development
- ❑ all deliverables (documentation, demos and source code) must be accessible from the *group's web page*
- ❑ keep deliverables *up-to-date* as the project progresses
 - ☞ each version and revision of a deliverable must be accessible
- ❑ every deliverable will be *reviewed* by another team
 - ☞ write documentation and code to be read by others!

Log your activities

Keep minutes of all meetings

- ☐ Date & time; participants
- ☐ Decisions and actions *with deadlines*

Log all effort

- ☐ Use simple metrics
 - “I spent 80 minutes debugging 15 short methods”
 - “We spent 120 minutes filling out 12 CRC cards”

Estimate cost of each task

- ☐ Iteratively improve your estimates
 - “We’ll need about 8 CRC cards, so we’ll probably need 80 minutes”

Weekly status reports are requested by the client!

Workplans

Planning and Cost Estimation

- ❑ use *conventional tools!* (e.g., Gantt charts)
- ❑ prepare a *workplan* with delivery times and costs for each deliverable
- ❑ break down each task into *subtasks* whose *cost* you can estimate
- ❑ detailed cost estimates must be made by the team member who accepts *responsibility* for it
- ❑ *revise* and *refine* your plan as the project progresses
- ❑ keep precise logs of how much time you *actually spend* on every subtask
 - ☞ gradually try to *improve* your estimates!

NB: goal is to improve your estimates, not to evaluate productivity!

Requirements Collection and Analysis

Requirements Specification

- ☐ Clients provides requirements specification in the form of *Use Cases*.
- ☐ Requirements will be examined and probably refined during the planning games.
- ☐ Clients keeps *up-to-date* use-cases

or

- ☐ Teams keep up-to-date uses cases

Risk Assessment

- ❑ Identify the *risks* and trade-offs
 - ☞ what *open* questions must be answered before you can start implementing a solution? (what *prototyping* is needed?)
- ❑ Identify *priorities*
 - ☞ what are the minimal requirements for a *first product*?

Prototyping

Prototyping is an essential activity carried out during all phases of the software process.

Requirements validation

- ❑ Prototype a user interface as *early* as possible to validate your requirements specification.

Evaluating design decisions

- ❑ Prototype parts of your design to evaluate *feasibility* and *usability* of technical alternatives.
 - ☞ prototype to reduce risks!

Iterative development

- ❑ Integrate parts as early as possible to *always have a running prototype* of the target application that can be tested and demoed.

Architecture

- ❑ Choose a *simple* architecture that can cope with all known requirements
 - ☞ what are the principal parts of the system and how do they communicate?
- ❑ Architecture will be heavily influenced by the *framework* that is used
- ❑ Develop *prototypes* to test the architecture

Design

- ❑ Iteratively apply *responsibility-driven design*
- ❑ *Evaluate* technical alternatives and *document* design decisions
 - ☞ *keep it simple*; add complexity only when necessary
 - ☞ prototype when trade-offs are unclear
- ❑ *Refactor* the design as the implementation evolves

Testing

Coverage

- ❑ Design tests that will exercise *all required/implemented functionality*
 - ☞ every time you add a feature, write a test for that feature!

- ❑ Check that *all possible execution paths* are tested
 - ☞ Apply both black-box and white-box testing

Testing

Regression

- ❑ *Automate* testing so that all tests can be carried out after any system change
- ❑ Set up tests so they can run in either
 - ➡ "*verbose*" mode (i.e., logging every interesting event), or in
 - ➡ "*silent*" mode (i.e., only reporting when and where tests have failed)

Tools

Use (at least) the following tools (or equivalent ones)!

- UML** Use UML to *document* all your models (esp. requirements specification and design).
- SNiFF+** SNiFF+ integrates *software development tools* (compilers, version management etc.) and provides support for development of software in teams.
- cvs** Use *version control* for all text documents (i.e., both source code and documentation).
- ANT** Use ANT to *automate compilation*, installation, testing and cleanup.
- javadoc** Automate *generation of HTML documentation* from source code.

Many other tools are available — use them!

Teamwork

Break down and distribute work incrementally

- ☐ always *estimate cost* when you accept a task
- ☐ check and *revise workplans* and estimates as work progresses

Use the team to your advantage

- ☐ use *role-playing* with CRC cards to elaborate the design
- ☐ *distribute responsibilities* according to *skills*
- ☐ someone else should *test* your code
- ☐ all code and documentation should be *reviewed* by someone else

Program in pairs

- ☐ *code review* as you program to increase quality

Roles and Responsibilities

Roles may be *fixed* or *floating*, but must always be assigned to some team member.

Sample roles and responsibilities

- ☐ Project Administrator
- ☐ Chief programmer/architect
- ☐ Backup programmer
- ☐ Tester/test case developer
- ☐ Toolsmith
- ☐ Component librarian
- ☐ Documentation editor

The *team* is responsible for all the deliverables.

Individuals assume responsibilities for specific subtasks.

Supporting roles

Customer

- ☐ answer questions about requirements ➡ *use the wiki!*
- ☐ accept/reject requirements specs
- ☐ evaluate prototypes, final system

System support

- ☐ system administration
- ☐ maintain installation of required software
- ☐ (limited) help for technical problems ➡ *use the wiki!*

Consultants

- ☐ meet regularly (at least twice weekly) with their teams
- ☐ oversee quality of work; give advice
- ☐ tool support; crisis detection; trouble-shooting

Forming Teams

1. Identify your skills: *strong and weak points*
 - ☞ What skills would complement your own?
2. Round table: *20 seconds to present yourself*
 - ☞ What do you have to offer; who are you looking for?
3. Form teams of six: *look for suitable partners*
 - ☞ Seek complementary skills that cover responsibilities
4. Prepare your strategy and tactics:
 - ☞ What questions do you need to ask of the client?
 - ☞ What interactions do you anticipate with other teams?