Applied Software Engineering – WS 2017/18

16. 03. 2018, Thilo Kratzer (thilo.kratzer@fau.de)

Inhaltsverzeichnis

Introduction

Software Engineering is a systematic, disciplined, quantifiable approach to the design, development, and operation of software systems. Goals of a software development project (the „Magic Triangle“):

- best quality of the software system to be produced
- stick to the time schedule
- optimized deployment of available resources

1 Basiscs and Process Models

1.1 Processes and Activities

Activity oriented: Process is defined as a hierarchy of activities. Activities take input and create output data.
Document oriented: Process is defined as a hierarchy of (partial) results (so called artifacts) with interdependencies.

A project is an endeavor

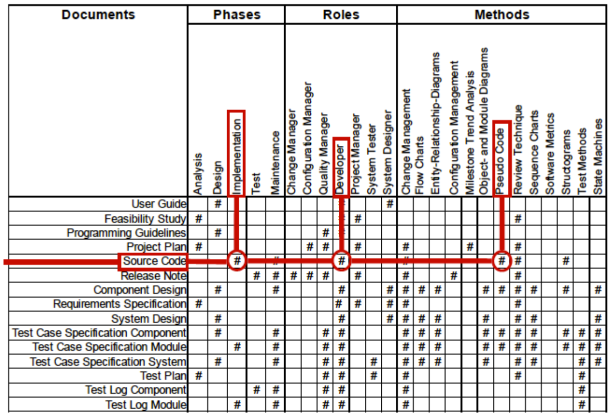
- limited in time by a defined start and end date
- that is characterized by its unique context and conditions (goals, scope, resources, ...)
- that usually follows or conducts activities which are predefined by a process

1.2 Workflows and Activities

A workflow consists of one or more sequential or parallel Activities. An activity is a well-defined task.

1.3 Process Parameters

Phases are the internal clock of a project
Roles defined by responsibilities and qualifications
Artifacts physical results produced during the project (associated to at least two roles: author and approver)
Activity describes single work elements using a sequence of steps
Methods describe the principles of work
Practices describe, how the work should be performed
Tools help in performing the work
Document plan shows what, when who and how:



Documents	Phases					Roles							Methods												
	Analysis	Design	Implementation	Test	Maintenance	Change Manager	Configuration Manager	Quality Manager	Developer	Project Manager	System Tester	System Designer	Change Management	Flow Charts	Entity-Relationship-Diagrams	Configuration Management	Milestone Trend Analysis	Object- and Module Diagrams	Pseudo Code	Review Technique	Sequence Charts	Software Metrics	Structograms	Test Methods	State Machines
User Guide		#							#			#													
Feasibility Study	#								#	#										#					
Programming Guidelines		#						#	#																
Project Plan	#						#	#		#			#				#			#					
Source Code			#		#				#				#						#	#			#		
Release Note				#	#	#	#	#		#			#			#				#					
Component Design		#			#				#			#	#	#	#			#	#	#	#		#		#
Requirements Specification	#								#	#		#	#							#					
System Design		#							#			#	#	#	#			#		#	#				#
Test Case Specification Component		#			#			#	#				#	#	#			#	#	#	#		#	#	#
Test Case Specification Module			#		#			#	#				#	#	#			#	#	#	#		#	#	#
Test Case Specification System		#			#			#	#		#		#	#	#			#		#	#			#	#
Test Plan	#							#	#		#		#							#				#	
Test Log Component				#	#			#	#				#											#	
Test Log Module			#		#			#	#				#											#	

1.4 Solution Principles and History

Divide and Conquer : divide up big somthing into smaller somethings
Simulation : understand reality using models

1.5 Waterfall and V-Model

Requirement Analysis
. ⇔ System and Modul Design
. ⇔ Coding and Modul Test
. ⇔ Integration and System Test
. ⇔ Installation and Maintenance

Problems: risks are discovered too late, focus on documentation not on product

1.6 Iterative Models and XP

Approach: stepwise development of the product
Development of a release → shipping to customer → gathering feedback → define requirements for next release

1.6.1 Spiral Model

Iterative (risk driven) incremental process, each loop represents a phase of the software process:
requirement analysis
. → risk analysis
. → prototypes, models, simulation
. → implementation
. → validation, review

Advantages:

- identify risks withing short time
- miss-understandings can be solved in early phase
- time to market is extremely short
- distribute work load across many cycles → optimized testing

1.6.2 Xtreme Programming (XP)

for small teams and short projects with changing requirements

- rapid feedback: as much as possible feedback among all participants
- assume simplicity: do not make things more complex than they are
- incremental change: small steps are faster and easier to complete
- embrace change: better to learn how to react on changes, than to try to predict the future
- qulity first: team should be proud about their work
- communication: use collective knowledge of the entire team

roles: user, coach, project lead, developer
advantages:

- focus on the important: the code
- less pressure due to planning game
- more success stories by short release
- better feedback from the user → more and earlier
- less risk at code changes through unit tests
- optimized knowledge transfer among the team

1.7 V-Model XT and RUP

V-Model XT: good example for huge projects with focus on documentation
Rational Unified Process (RUP): huge process for OOD, needs massive tailoring

2 Agile Approaches

2.1 Overview

agility: fast, ready for change, fun, active, flexible, not rigid
methods: XP, Scrum, KanBan, ...
values due to the *agile manifesto*:

- individuals and interactions over processes and tools
- working software over comprehensive documentation
- customer collaboration over contract negotiation
- responding to change over following a plan

2.2 Agile Principles

- early and continuous delivery of software
- welcome changing requirements
- deliver working software frequently
- business people and developers must work together daily
- build projects around motivated individuals
- most effective method of communication is face-to-face
- working software is the primary measure of progress
- agile processes promote sustainable development
- continuous attention to technical excellence
- simplicity is essential
- self-organizing teams
- reflect on how to become more effective

2.3 Scrum

2.3.1 Roles

- product owner
  - responsible for return of investment (ROI) of the product
  - prioritizes features by business value

  - responsible for the product backlog
  - accepts or rejects work
- scrum master
  - ensures team is functioning and productive
  - removes impediments
  - shields team from external interference
  - facilitates planning
  - no traditional project manager (coaching role)
- Team
  - cross functional (5-9 members)
  - self directed, organizes itself and tasks
  - fix during Sprint

2.3.2 Phases

- sprint planning: plan activity for one iteration/sprint (e.g. 8h for a 4-week sprint)
  - product owner presents product backlog
  - sprint goal is selected
  - product backlog is analyzed and evaluated
- sprint review/demo (ca. 1-2h): team presents what it accomplished during sprint (inspect and adapt)
- sprint retrospective (ca. 30 min): reflection and identification of improvements
- daily scrum (15 min): team, product owner, scrum master
  - What did I do since last daily scrum meeting?
  - What will I do until next daily scrum meeting?
  - Are there any impediments?
- sprint (2-4 weeks): development of a shippable function within fixed time, team and quality goals

2.3.3 Artefacts

- Product Backlog: Prioritized feature list for product to be developed
  - highly dynamic, never complete
  - Represents current state of knowledge
  - Contains features, bugfixes, technical work, optimizations, know-how
- Sprint Backlog: tasks of team for the next sprint (Updated in Daily Scrum)
- Burndown: presents effort completed in sprint
- Done: Defines, what needs to be done by the team in order to successfully complete a user story

3 Project Management

3.1 Project Start

- need to do something
- ususally a responsible person
- typical scenario (e.g. vague idea of product)
- define and understand goals with customer

- derive common vision!

Project Communication during project start phase:
Bilateral Meeting : with all known team members, assign roles, exchange information about project
Kick-Off-Meeting : information event (no discussions), get-together of team members
Project Start Workshop : moderated discussion of project vision, discuss project goals (more than one day)
Use Case Workshop : contact stakeholder, get requirements
Content of Project Definition:

- Project motivation
  → Why does project exist? Why is it desirable?
- Project goals (quantified!)
  → Objective targets for controlling: time, budget oder qualitiy goals like norms
- Products i.e. results of the project
  → One-pager: high-level abstact of requirements
- Development strategy
  → process model, make/use/buy overview, milestones
- Stakeholder of the project
  → external communication matrix, roles of external parties
- Mandatory participation of the customer
  → milestones an deliverables (with acceptance criteria)
- Project organization, project team and roles
  → responsibilities and authorization
- Context, boundary conditions, assumptions, chances and risks
  → external forces, risk list, open issues

3.2 Project Planning

3.2.1 Define Scope and Milestones

- use Project Definition from Project Start
- add aditional information, deadlines and financial budgets
- define deliverables on abstact layer
- decide which processes, methods, tools should be used
- definde main milestones
- define interval for monitoring and controlling

Attributes for milestones: verifiable, equally distributed, short term

3.2.2 Define Work Packages

Work Package is the smallest entity or task in a project (with own effort and responsibility)
→ should be possible to complete by one person in time frame of two reportings
All work packages build „work breakdown structure“ (WBS)

- hierarchical structure (mind map, hierarchical graph or org-chart) with packages as leaves
- can be organized along different process parameters (phases, products, roles, ...)

3.2.3 Estimate Work Packages

- Delphi Estimation Technique:
  - moderator explains project to team of experts
  - discussion with experts for better understanding
  - experts estimate work package separately
  - discuss differences in estimations
  - re-estimation until experts agree on common value or average is taken
- 3 Point Estimation Technique
  - moderator explains project to team of experts
  - discussion with experts for better understanding
  - experts estimate work package separately: best case, realistic case, worst case
  - value taken for further planning (weighted median): $\frac{opt + 4 \times real + worst}{6}$

3.2.4 Schedule Work Packages

- assign ressources
- determine duration
- decide dependencies
- allocate in calender
- derive earliest start and end date
- optimize schedule: if necessary split work packages in WBS

→ derive workpackage schedule (e.g. Gantt chart) from WBS and estimation!

- critical path: longest path of (parallel) activities depending on each other
- minimal project length
- total float: amount of available time
- mapping ressources to work packages

3.2.5 Evaluate Costs of Work Packages

- cost planning is done based on work packages
- based on assigned staff, tooling, material
- knowledge of which budget is needed at what time
- create budget plan

3.3 Risk Management

risk: unfortunate event which may happen in the future with certain probability

- risk identification: typically done in workshops
- risk analysis: evaluation of identified risks
- risk planning: countermeasures, inititate tasks, documentation
  → prevention, transfer, mitigation, accaptance
- risk tracking: regular project status meeting, defined responsibilities

risk list: fundamental document which contains all risks with a

- unique identifier

- description
- status
- responsible person
- next action
- due date

3.4 Configuration Management

- monitor changes in project outcome
- manage different versions and configurations of project outcome
- project efficiency: central repository, single point of truth
- automation: celiver project oucome fast, reproducable, QA checked
- control: enforce process, determine KPI, ...
- synchronization: overcome (remote) team boundaries

configuration : set of consistent work products, that are qualitiy checked and formalley released
configuration element : identifyable, machine readable part of the overall project outcome
item list : identifier, version, owner, reviewer, milestone of release, status of item
configuration plan : Defines organizational details and methodology for the activities

- introduction: purpose, scope, definitions, references
- management: respinsibilities
- acitvities
- schedule
- resources
- CM plan management

configuration management : generates baselines that are

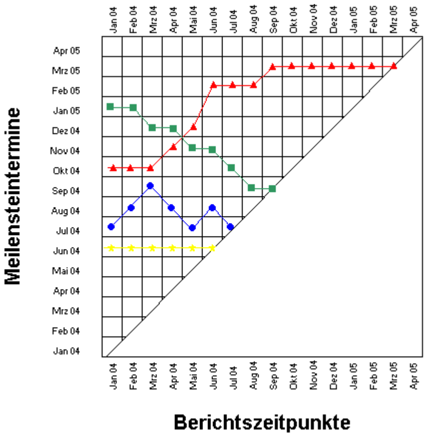
- snapshots of all available project work products
- persistent, reproducable
- semantically meaningful (e.g. can be characterized)
- officially reviewed and released

Conventions on versioning defined in the configuration management plan

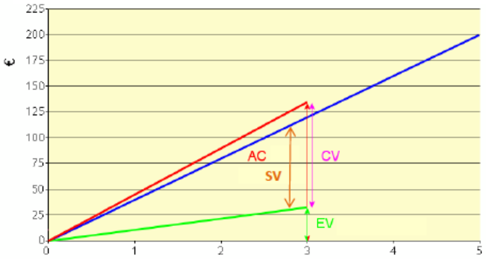
3.5 Project Controlling

Controlling is the effective adaptation of project performance in order to avert unwanted deviations form the plan.

- informal monitoring: good feeling, interviews to participants (team, stakeholders)
- qualitative: regular reporting of properties, red/yellow/green indicators
- quantitative: set of metrics to characterize state, analyze trends
  - key performance indicator (KPI): derived from performance data, meaningful to state
  - milestone trend anaylsis (MTA): monitor progress with focus on schedule



- earned value analysis (EVA): focus on planned costs (PC) and schedule
  accumulated costs (AC), earned value (EV), schedule variance (SV), cost variance (CV)



3.6 Compile Project Plan

- project definition
- milestone plan
- work breakdown structure
- resources
- cost plan
- list of risks
- configuration management plan
- quality assurance plan
- communication plan (meetings, reports, ...)

4 Product Quality

Quality: the degree to which a system, component, or process meets specified requirements (ISO/IEC 9126-1:2001)

- funcionality
- reliability
- usability
- efficiency
- maintainability
- portability

4.1 Basics

magic square: quality ↔ functionality/scope ↔ time ↔ costs/effort
typical questions: how to...

- ...make customers happy?
- ...find out true requirements?
- ...find out and design necessary processes and activities?
- ...find suitable resources and best suppliers?
- ...deal with changes?

4.2 Metrics

measuring: reproducible, quantitative statements can be trusted more than quality statements

software : product of thought consisting of programs, procedures and relevant descriptions that belong to a data processing system

metric : numerical value of a property

software metric : function mapping a software unit to a numerical value

process metrics (estimates, resources):

- review effort
- team size
- duration of phase/project
- number of methods
- number of tools

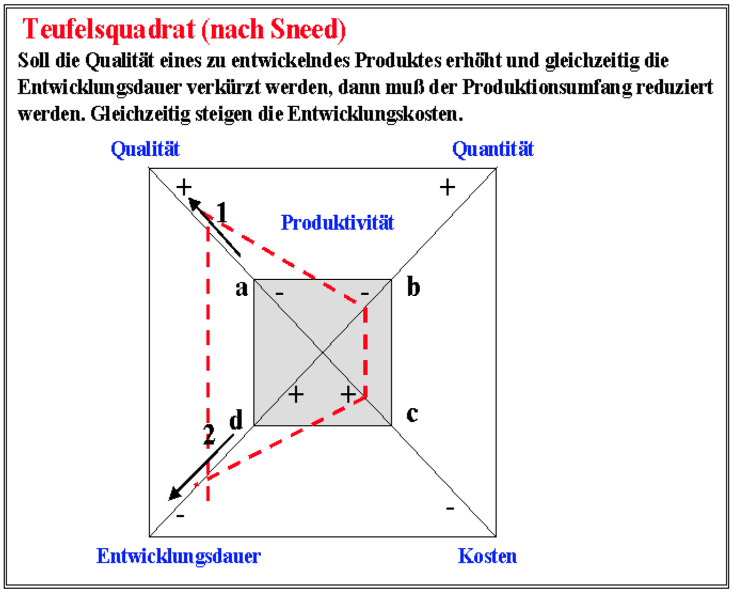
product metrics (design metrics, source-code metrics, object oriented metrics):

- function points
- runtime
- storage
- number of documentation pages
- testing effort

Goal-Question-Metric Model (GQM): How to find the right metrics

- define specific goals for improvement
- define metrics for each goal (quantitative measurement)
- define hypothesis for each metrics
- proof hypothesis by measuring

relationship between quality, quantity/scope, time/duration and costs (devil's square):



Teufelsquadrat (nach Sneed)

Soll die Qualität eines zu entwickelndes Produktes erhöht und gleichzeitig die Entwicklungsdauer verkürzt werden, dann muß der Produktionsumfang reduziert werden. Gleichzeitig steigen die Entwicklungskosten.

4.3 Proofs

- validation: „has the right system been developed?“
  → suitable for its intended purpose
- verification: „has the system been developed right?“
  → conforms to a given specification

Climbing Steps of Trust by means of Analytical Methods

- reviews, walkthroughs: all program standards are fulfilled
- static Analysis: interfaces, program run and data flow correct
- white Box Test: all program units tested (statement, desicion or path coverage)
- black Box Test: correct output at correct input
- black Box Test: defined behavior at arbitrary input
- black Box Test: correct output in stress test
- assertion Test: no assertions violated
- formal proof: program ⇔ design
- formal proof: program ⇔ requirements

4.4 Reviews

Procedure by which a certain document (artifact) is analyzed by one or more experts

- proof reading
- walkthrough
- inspection
- extended review

5 Verification and Validation

5.1 Definitions

validation : Did we develop the right (a valid) solution?
verification : Did we develop the solution in the right way?

error/mistake : discrepancy between a computed value and the specified correct value (caused human)
fault : incorrect step, process or data definition in a computer program (also known as Bug)
failure : inability of a system to perform its required functions (caused by fault)

static tests : performed without actually executing programs (e.g. walkthrough, review)
dynamic tests : performed by executing programs on a real or virtual processor (e.g. unit tests)

5.2 Test Process

5.2.1 Planning

Test strategy:

- setup of test environment (tooling)
- black box oder white box strategy
- test coverage measurement
- end of testing criteria
- regression strategy

Test plan:
goal, system under test, priority, methods, resources, schedule, responsibilities, test end criteria, pass/fail criteria

Priorities:

- the most important test suits / cases need to be executed in the available time
- Safety critical requirements have to be tested first
- Legal based requirements have to be tested next
- Mission critical requirements have to be tested next
- Customer based requirements have to be tested next

Test end criteria: finished when ...

- time for testing is over
- all test cases with a defined priority have been tested
- planed test coverage is reached
- only a certain number of failures per test time unit is found

5.2.2 Design

Test Cases are always based on requirements and organized in test suites

- reference to its origin
- set of input values
- set of pre-conditions
- set of expected results and post-conditions

Define the right expected test output data by test oracle:

- dealized source of information offering the correct target result for any test case
- behaves exactly like implementation and is absolutely reliable
- as difficult as to create a perfect implementation
- Results can be defined by means of specification, simulation, estimation, expertise and experience

5.2.3 Implementation

- A logical test case is a test case without values for input and output
- An instance of a logical test case is called specific test case
- During implementation the specified test cases will be converted into test scripts

5.2.4 Executing

The testing environment consists of

- SuT (System under Test) with its defined interfaces
- test driver, dummies (Stubs), and test conditions
- the operating system, if necessary
- additional applications, if necessary
- testing tools for capturing the results

5.2.5 Evaluation

Compare expected and recorded behavior and results

- In case of differences, review test case implementation for possible faults
- If test case has faults, correct test case
- If the test case is correct, write a failure report

5.2.6 Reporting

A Test Report

- lists all test records and failure reports
- summarizes the executed test cases
- allows to proof that the intended test strategy was applied

5.2.7 Checking and End of Test

- The evaluation of test end checks the test-end criteria as defined in the test plan
- The evaluation of test end is based on test reports
- The result of testing is evaluated with pass/fail

Each test execution is followed by a test status report including the following information:

- progress of testing
- recommendation for testers or the continuation of testing
- justification of this decision in accordance with the test plan

5.3 Test Phases in the V-Model

5.3.1 Module Test

- test focus: module features, structure, performance
- classification of failures: missing paths, calculation faults
- conditions: module specification and design, source code

5.3.2 Integration Test

- test focus: procedure/parameter interface
- classification of failures: wrong usage/understanding of interface
- conditions: interface description, at least two modules

5.3.3 System Test

- test focus: requirements, usability, efficiency, maintainability, security
- classification of failures: missing/wrong fulfillment of requirement
- conditions: requirement specification, fully integrated system

5.3.4 Accaptance Test

- test focus: usability, completeness
- classification of failures: missing/wrong fulfillment of requirement
- conditions: requirement specification, running system

5.4 Psychology of Testing

tbd.

6 Requirements Engineering

6.1 Definitions

Roles:

- sponsor: providing development goal in terms of requirements
- project manager: planing, controlling, steering
- domain expert: domain viewpoint
- system analyst: clarify requirements
- system architect: technical feasibility
- system tester: verifiability

Artifacts:

- requirements specification: informal description of the requirements
  - External Requirements Specification
  - System/Software Requirements Specification
  - should be complete, precise, consistent, efficient, verifiable
- system glossary: definition of the terminology used and the central domain concepts

- feasbility study: check of critical requirements, evalutation of alternatives

6.2 Classification and Principles

Requirements define the functional, quantitative and qualitative traits of the target system from a stakeholder's perspective

- functional requirements: features
- non-functional requirements: quanitative, qualitative, management requirements
  - performance
  - operational quantities (no. of users, . . .)
  - usability requirements
  - modifiability
  - safety
  - mandatory norms or standards
  - technical
  - management requirements

Requirements shall be classified as: must, should, could, won't

Requirements shall be *smart*: specific, measurable, assignable, realistic, tracable

6.3 Acivities in Requirements Engineering

6.4 Requirements Modeling

- context analysis
- actors
- use cases: description, diagrams, methodology
- scenarios

6.5 Requirements Management

tbd.

7 System Analysis

7.1 Introduction

primary goal of the system analysis phase is the creation and documentation of a system model that:

- meets the requirements
- describes the software system sufficiently

result of the system analysis phase is a domain oriented problem solution

- requirements specification
- system glossary
- user interface prototypes
- user manual

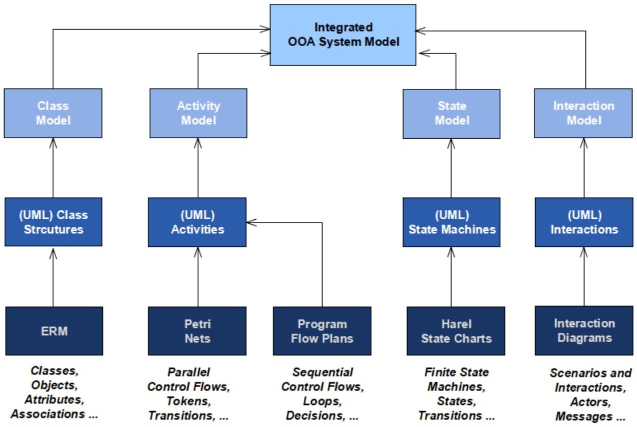
Roles:

- Project Manager

- Domain Expert
- System Analysist
- Customer
- Software Ergonomics Specialist
- Technical Editor

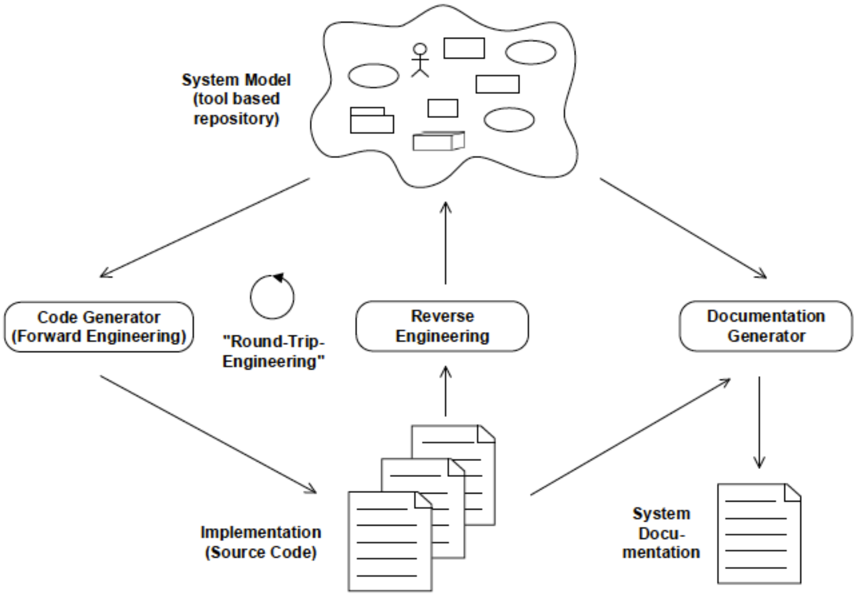
7.2 Models

main task in the early phases of a development project is the creation of a (system/software) model as an abstraction of the target system:


Integrated OOA System Model
Class Model
Activity Model
State Model
Interaction Model
(UML) Class Strcutures
(UML) Activities
(UML) State Machines
(UML) Interactions
ERM
Petri Nets
Program Flow Plans
Harel State Charts
Interaction Diagrams
Classes, Objects, Attributes, Associations ...
Parallel Control Flows, Tokens, Transitions, ...
Sequential Control Flows, Loops, Decisions, ...
Finite State Machines, States, Transitions ...
Scenarios and Interactions, Actors, Messages ...

advantages of model based development (MBD):

- automated code generation
- consistent system documentation
- changes need to made only on model stage


System Model (tool based repository)
Code Generator (Forward Engineering)
"Round-Trip-Engineering"
Reverse Engineering
Documentation Generator
Implementation (Source Code)
System Docu-mentation

system analysis model is the essential link between requirements and system design

→ describes essential properties of the target system form domain perspective

- subjects of modeling
- modeling domain

- modeling concepts
- modeling notation
- modeling views
- modeling methodology

During system analysis, the subjects of modeling are divided up into four groups

- structure and relationships of required **data**
- classification of the system in **functions** implementing a certain funcitonality
- **dynamics** of intended system (processes, communication and timing)
- functinoal and ergonomic setup of the **user interface**

A viewpoint emphasizes certain aspects while suppressing others

- function oriented view (as a hierarchy of functions)
- data oriented view (data structures, relationships and manipulation operators)
- object oriented view (using interacting objects)
- scenario based view (by means of application scenarios and message exchange)
- state based view (by means of defined states and transitions between them)
- control flow oriented view (using algorithmic control structures)
- rule based view (by means that define conditions and following actions)

A modeling methodology defines

- a set of basic modeling concepts
- a notation for the chosen concepts
- a set of viewpoint models
- a set of modeling rules
- a mapping from viewpoint to integrated model
- a procedure to create and maintain integrated model

7.3 Unified Modeling Language (UML)

tbd.

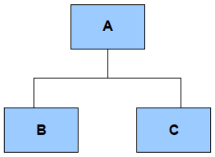
7.4 Basic Concepts

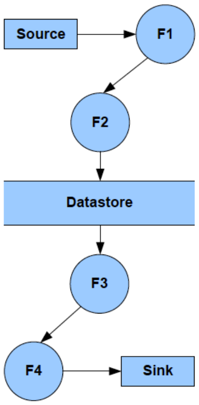
7.4.1 Function Oriented View

- function trees: systematic break-down of general functions into sub-functions
- data flow diagrams: flow of data between processing (functions), persistence (storage), and context (interfaces)
    - unique names
    - contains at least one interface
    - interfaces unique and only placed once
    - no direct data flow between two interfaces
    - no direct data flow between two data stores
    - no direct data flow between interfaces and data stores
    - data store has at least one incoming and one outgoing data flow
    - functions receive all data they require via data flows
    - Functions do not get data
    - equal data source/sink instances have to be mapped to the same interface

  - Interfaces represent the actual sources and sinks (not proxies)

Use function hierarchies at the beginning of system analysis to get a systematic knowledge of all required functions and their structure

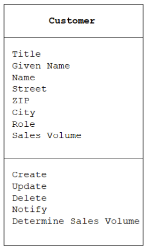


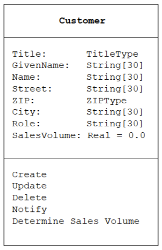


7.4.2 Object Oriented View

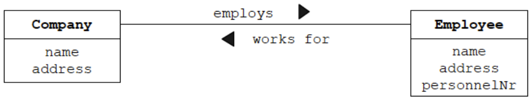
- class models

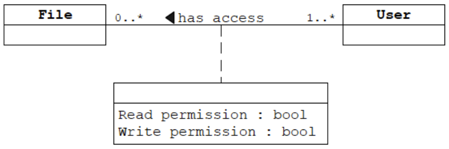
  An object has properties and behavior as defined by its class (attributes, operations)



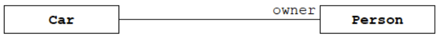


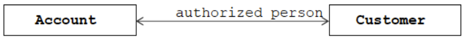
- associations (with properties)



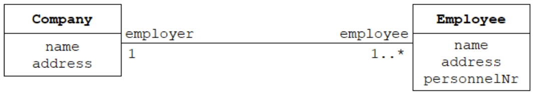


- roles and navigability

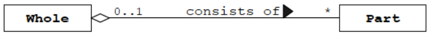


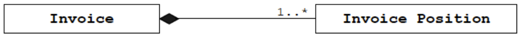


- cardinality (1, 1..*, 0..*, 0..1, c, n..m)

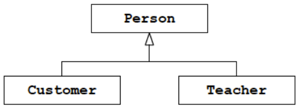


- aggregation and composition

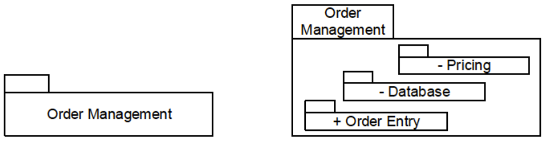




- generalization/specification



- packages



analysis class models quickly tend to become incomprehensible → structure by modularization and encapsulation

typical steps of an iterative OOAD process:

1. identification of the relevant classes and objects
2. rough definition of their properties and responsiblities
3. specification of the semantical relationships between the classes
4. modularization of the system model using packages
5. rendering the system model more precisely using constraints and assertions

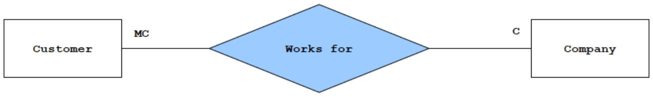
How to find Packages:

- top-down: partitioning of the system into packages before identifying classes and associations
- bottom-up: refactor the system model when a critical size resp. complexity threshold has been reached

7.4.3 Data Oriented View

- entity-relationship models

  ER models only specify data and their relationships, but no dynamic system behavior



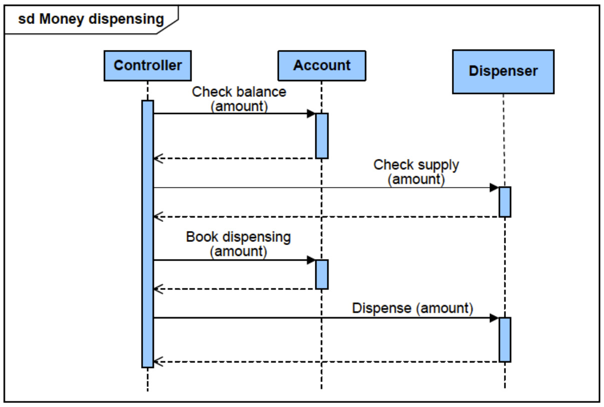
- data catalogues: contains information about data structures, characteristics and usage

  notation:

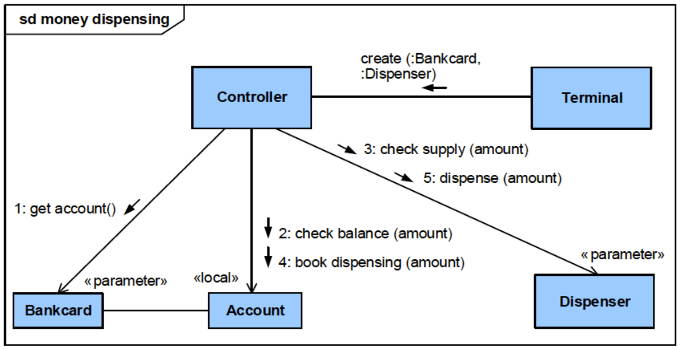
  - sequence: `X = X1 + X2 + X3`
  - selection (XOR): `A = [B|C]`
  - iteration (arbitrary): `A = {B}`
  - iteration (m to n): `A = m{B}n`
  - option: `A = B + (C)`

7.4.4 Scenario Based View

- scenarios (e.g. sequence diagram)

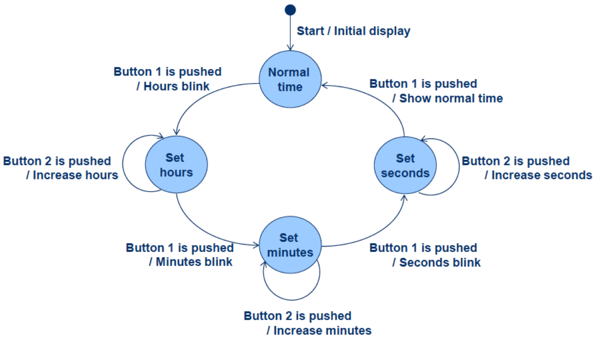


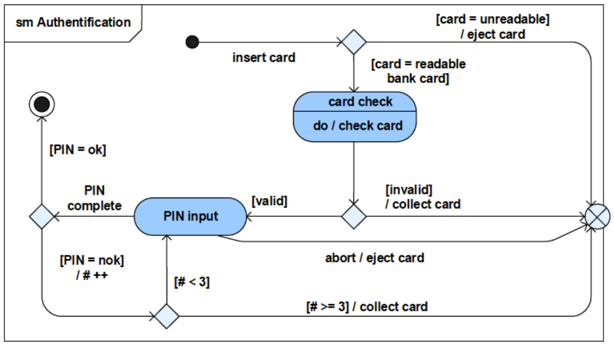
- interactions (e.g. communication diagram)



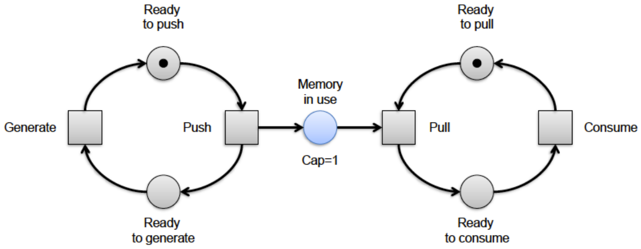
7.4.5 State Based View

- state charts


Start / Initial display
Normal time
Button 1 is pushed / Hours blink
Button 1 is pushed / Show normal time
Set hours
Set seconds
Button 2 is pushed / Increase hours
Button 2 is pushed / Increase seconds
Set minutes
Button 1 is pushed / Minutes blink
Button 1 is pushed / Seconds blink
Button 2 is pushed / Increase minutes



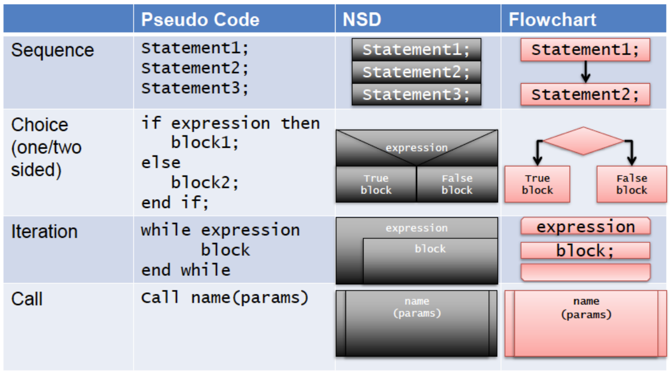
- petri nets



live: transitions are enabled alternately forever, safe: upper bound to the number of tokens

7.4.6 Control Flow Based View

- pseudo code
- control structures



	Pseudo Code	NSD	Flowchart

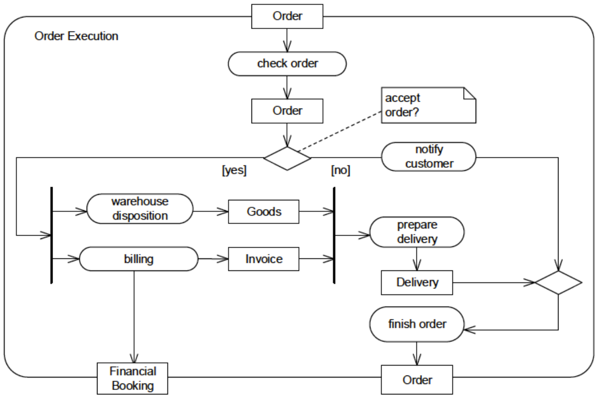
Sequence	`Statement1;`
	`Statement2;`
	`Statement3;`
		Statement1;
		Statement2;
		Statement3;
			Statement1;
			Statement2;

Choice (one/two sided)	`if expression then`
	`    block1;`
	`else`
	`    block2;`
	`end if;`
		expression
		True block
		False block
			True block
			False block

Iteration	`while expression`
	`      block`
	`end while`
		expression
		block
			expression
			block;

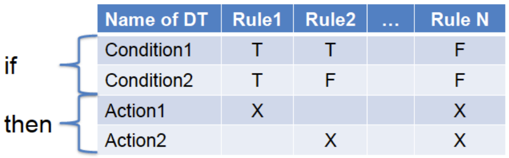
Call	`Call name(params)`	name (params)	name (params)

- activities



7.4.7 Rule Based View

- decision tables / decision trees



	Name of DT	Rule1	Rule2	...	Rule N
if	Condition1	T	T		F
	Condition2	T	F		F
then	Action1	X			X
	Action2		X		X

- rule engines

8 System Design

After defining what to do it is now the question how to do concepts, behavior?
Building architecture

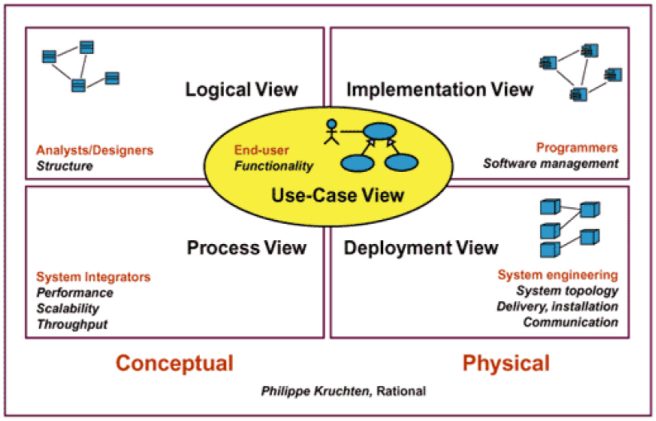
- planning: requirements and system analysis
- designing: subsystems and components, programming languages
- constructing: generate code, unit/integration testing
- maintaining: add features, repair

modularization : system, subsystem, component, module

cohesion : degree of functional association

coupling : number of interfaces, informal dependencies

Kruchten's 4+1 System :



Layer structure along logical aspects ↔ tier structure along physical aspects

presentation, middleware, data access ↔ frontend, backend

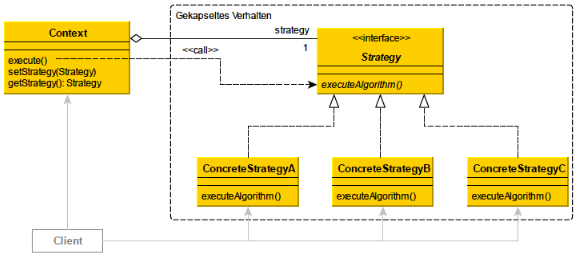
Design model abstraction level

1. code
2. design model
3. architecture model
4. software analysis model
5. use case model

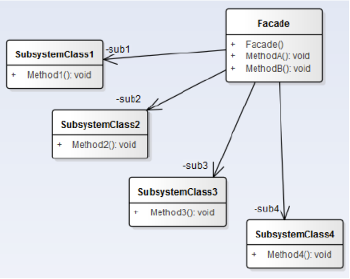
8.1 Design Patterns

classification as structural, behavioral, creational:

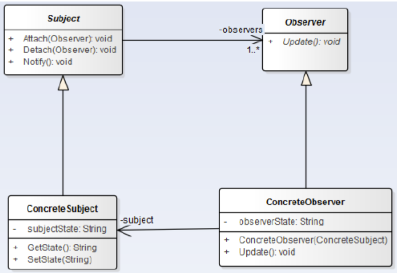
- strategy pattern (behavioral)



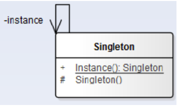
- facade (structural)



- observer (behavioral)



- singleton (creational)



9 Coding, Tools & Configuration Management

- write code

- documentation
- unit testing
- review
- check-in branches
- integrate

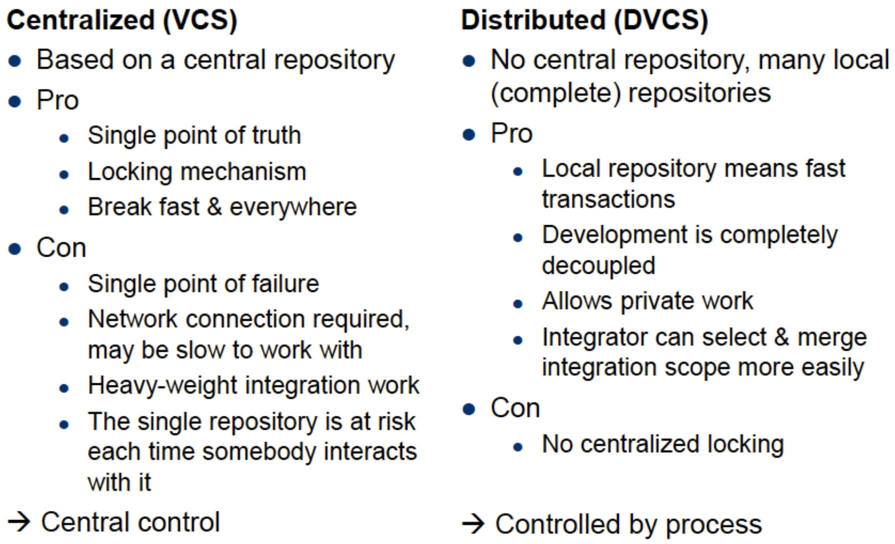
coding metrics:

- lines of code (LOC)
- McCabe complexity (control flow)
- Halstead effort and difficulty
- object oriented metrics

9.1 Object Oriented Metrics

- Halstead effort and difficulty:
  - number of operators (e.g. key words, logical operators): N_1
  - number of operands (e.g. variables, constants): N_2
  - number of unique operator: n_1
  - number of unique operands: n_2
  - information: volume $= N \times \log_2 n$
  - probability of errors: difficulty $= \frac{n_1}{2} \times \frac{N_2}{n_2}$
  - effort to create software: effort $=$ volume $\times$ difficulty
- depth of inheritance tree (DIT): The deeper a class in the hierarchy, the greater the number of methods it will probably inherit which makes it harder to predict its behavior
- response for class (RFC): Includes the methods of the class itself as well as the methods that are called
- lack of cohesion in methods (LCOM): If two methods access the same attributes of a class, they are cohesive → Sum up the number of pairs that do share instance variables and the number of pairs that don't

9.2 Version Control



Centralized (VCS)

- Based on a central repository
- Pro
  - Single point of truth
  - Locking mechanism
  - Break fast & everywhere
- Con
  - Single point of failure
  - Network connection required, may be slow to work with
  - Heavy-weight integration work
  - The single repository is at risk each time somebody interacts with it

→ Central control

Distributed (DVCS)

- No central repository, many local (complete) repositories
- Pro
  - Local repository means fast transactions
  - Development is completely decoupled
  - Allows private work
  - Integrator can select & merge integration scope more easily
- Con
  - No centralized locking

→ Controlled by process