Systeme hoher Qualität und Sicherheit
Universität Bremen, WS 2013/14

Lecture 03 (04.11.2013)
Quality of the Software Development Process

Christoph Lüth
Christian Liguda

Universität Bremen

Your Daily Menu

- Models of Software Development
  - The Software Development Process, and its rôle in safety-critical software development.
  - What kind of development models are there?
  - Which ones are useful for safety-critical software – and why?
  - What do the norms and standards say?
- Basic Notions of Formal Software Development:
  - How to specifiy: properties
  - Structuring of the development process

Where are we?

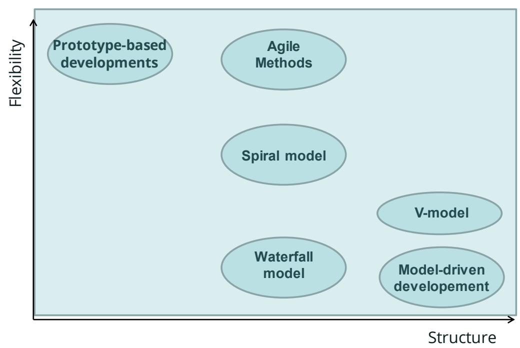
Software Development Models

Universität Bremen

Software Development Process

- A software development process is the **structure** imposed on the development of a software product.
- We classify processes according to *models* which specify
  - the artefacts of the development, such as
    - the software product itself, specifications, test documents, reports, reviews, proofs, plans etc
  - the different stages of the development,
  - and the artefacts associated to each stage.
- Different models have a different focus:
  - Correctness, development time, flexibility.
- What does quality mean in this context?
  - What is the *output*? Just the sofware product, or more? (specifications, test runs, documents, proofs...)

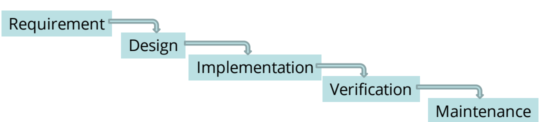
Software Development Models



from S. Paulus: Sichere Software

Waterfall Model (Royce 1970)

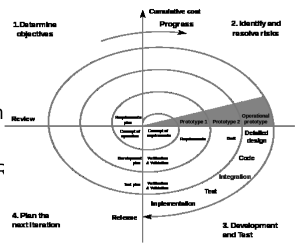
- Classical top-down sequential workflow with strictly separated phases.



- Unpractical as actual workflow (no feedback between phases), but even early papers did not *really* suggest this.

Spiral Model (Böhm, 1986)

- Incremental development guided by **risk factors**
- Four phases:
  - Determine objectives
  - Analyse risks
  - Development and test
  - Review, plan next iteration
- See e.g.
  - Rational Unified Process (F
- Drawbacks:
  - Risk identification is the key, and can be quite difficult

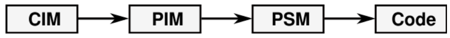


Agile Methods

- Prototype-driven development
  - E.g. Rapid Application Development
  - Development as a sequence of prototypes
  - Ever-changing safety and security requirements
- Agile programming
  - E.g. Scrum, extreme programming
  - Development guided by functional requirements
  - Less support for non-functional requirements
- Test-driven development
  - Tests as *executable specifications:* write tests first
  - Often used together with the other two

Model-Driven Development (MDD, MDE)

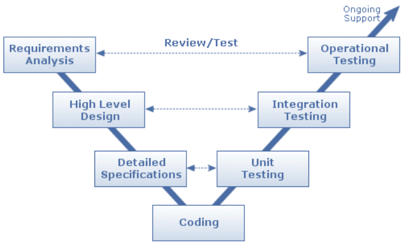
- Describe problems on abstract level using *a modelling language* (often a *domain-specific language*), and derive implementation by model transformation or run-time interpretation.
- Often used with UML (or its DSLs, eg. SysML)



- Variety of tools:
  - Rational tool chain, Enterprise Architect
  - EMF (Eclipse Modelling Framework)
- Strictly sequential development
- Drawbacks: high initial investment, limited flexibility

V-Model

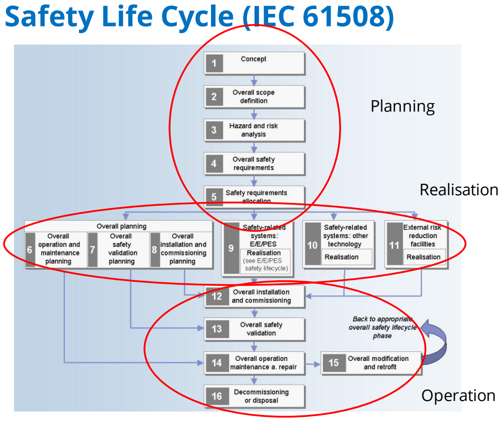
- Evolution of the waterfall model:
  - Each phase is supported by a corresponding testing phase (verification & validation)
  - Feedback between next and previous phase
- Standard model for public projects in Germany
  - … but also a general term for models of this „shape“



Development Models for Critical Systems

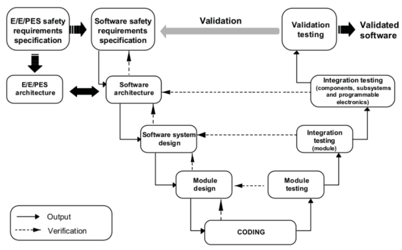
- Ensuring safety/security needs structure.
  - …but *too much* structure makes developments bureaucratic, which is *in itself* a safety risk.
  - Cautionary tale: Ariane-5
- Standards put emphasis on *process*.
  - Everything needs to be planned and documented.
- Best suited development models are variations of the V-model or spiral model.

The Safety Life Cycle (IEC 61508)



Development Model in IEC 61508

- IEC 61508 prescribes certain activities for each phase of the life cycle.
- Development is one part of the life cycle.
- IEC *recommends* V-model.



Development Model in DO-178B

- DO-178B defines different *processes* in the SW life cycle:
  - Planning process
  - Development process, structured in turn into
    - Requirements process
    - Design process
    - Coding process
    - Integration process
  - Integral process
- There is no conspicuous diagram, but these are the phases found in the V-model as well.
  - Implicit recommendation.

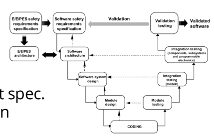
Artefacts in the Development Process

Planning:
- Document plan
- V&V plan
- QM plan
- Test plan
- Project manual

Specifications:
- Safety requirement spec.
- System specification
- Detail specification
- User document (safety reference manual)

Implementation:
- Code

Verification & validation:
- Code review protocols
- Tests and test scripts
- Proofs



Possible formats:
- Word documents
- Excel sheets
- Wiki text
- Database (Doors)
- UML diagrams
- Formal languages:
  - Z, HOL, etc.
  - Statecharts or similar diagrams
- Source code

Documents must be *identified* and *reconstructable*.
- Revision control and configuration management *obligatory*.

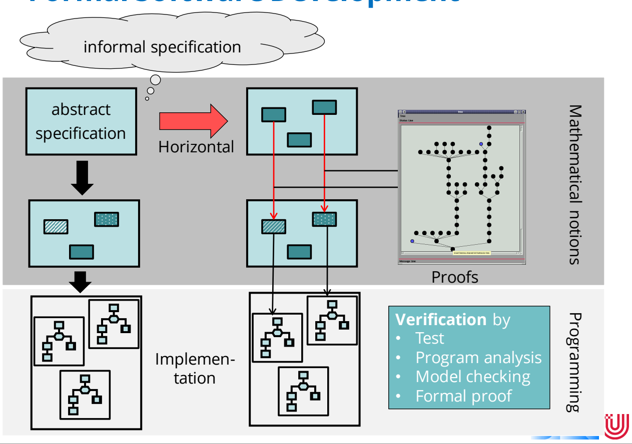
Basic Notions of Formal Software Development

Universität Bremen

Formal Software Development

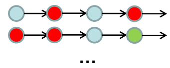
- In **formal** development, properties are stated in a rigorous way with a precise mathematical semantics.
- These formal specifications can be **proven**.
- Advantages:
  - Errors can be found **early** in the development process, saving time and effort and hence costs.
  - There is a higher degree of trust in the system.
  - Hence, standards recommend use of formal methods for high SILs/EALs.
- Drawback:
  - Requires **qualified** personnel (that would be *you*).
- There are tools which can help us by
  - **finding** (simple) proofs for us, or
  - **checking** our (more complicated proofs).

Formal Software Development

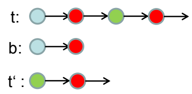


Properties

- A general notion of **properties**.
- Properties as set of infinite execution traces (i.e. infinite sequences of states)



- Trace t satisfies property P, written $P \vDash t$, iff $t \in P$
- b ≤ t iff ∃ t′ . t = b • t′
  - i.e. b is a *finite* prefix of t



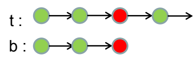
Safety and Liveness Properties

Alpen & Schneider (1985, 1987)

- **Safety** properties
  - *Nothing bad happens*
  - partial correctness, program safety, access control
- **Liveness** properties
  - *Something good happens*
  - Termination, guaranteed service, availability
- **Theorem**: $\forall$ P . P = $Safe_P \cap Live_P$
  - Each property can be represented as a combination of safety and liveness properties.

Safety Properties

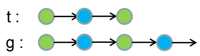
- Safety property S: „Nothing bad happens"
- A bad thing is *finitely* observable and *irremediable*
- S is a safety property iff
  - $\forall t. t \notin S \rightarrow (\exists b. \text{finite } b \wedge b \leq t \rightarrow \forall u. b \leq u \rightarrow u \notin S)$



  - a finite prefix b always causes the bad thing
- **Safety is typically proven by induction**
  - Safety properties may be enforced by run-time monitors.

Liveness Properties

- Liveness property L: „Good things will happen"
- A good thing is always possible and possibly infinite:
- L is a liveness property iff
  - $\forall t. \text{ finite } t \rightarrow \exists g. t \leq g \wedge g \in L$
  - i.e. all finite traces t can be extended to a trace g in L.



- **Liveness is typically proven by well-foundedness.**

Underspecification and Nondeterminism

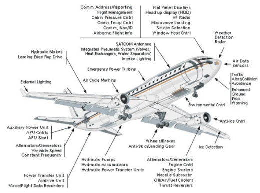
- A *system* S is characterised by a *set of traces*.
- A system S *satisfies* a property P, written
  $S \vDash P$ iff $S \subseteq P$
  (i.e. $\forall t \in S. t \in P$, all traces satisfy the property P).
- Why more than one trace? Difference between:
  - *Underspecification* or *loose specification* – we specify several *possible* implementations.
  - Non-determinism – different program runs might result in different traces.
- Example: a simple can vending machine.
  - Insert coin, chose brand, dispense drink.
  - Non-determinisim due to *internal* or *external* choice.

Structure in the Development

- Horizontal structuring
  - Modularization into components
  - Composition and Decomposition
  - Aggregation
- Vertical structuring
  - Abstraction and refinement from design specification to implementation
  - Declarative vs. imparative specification
  - Inheritence
- Layers / Views
  - Adresses multiple aspects of a system
  - Behavioral model, performance model, structural model, analysis model(e.g. UML, SysML)

Horizontal Structuring (informal)

- Composition of components
  - Dependent on the individual layer of abstraction
  - E.g. modules, procedures, functions,...
- Example:



Horizontal Structuring: Composition

- Given two systems S_1, S_2, their *sequential composition* is defined as

$$S_1; S_2 = \{s \cdot t \mid s \in S_1, t \in S_2\}$$

  - All traces from S_1, followed by all traces from S_2.
- Given two traces s, t, their *interleaving* is defined (recursively) as
  $<> \parallel t = t$
  $s \parallel <> = s$
  $a \cdot s \parallel b \cdot t = \{a \cdot u \mid u \in s \parallel b \cdot t\} \cup \{b \cdot u \mid u \in a \cdot s \parallel t\}$
- Given two systems S_1, S_2, their *parallel composition* is defined as

$$S_1 \parallel S_2 = \{s \parallel t \mid s \in S_1, t \in S_2\}$$

  - Traces from S_1 interleaved with traces from S_2.

Vertical Structure - Refinement

- Data refinement
  - Abstract datatype is „implemented" in terms of the more concrete datatype
  - Simple example: define stack with lists
- Process refinement
  - Process is refined by excluding certain runs
  - Refinement as a reduction of underspecification by eliminating possible behaviours
- Action refinement
  - Action is refined by a sequence of actions
  - E.g. a stub for a procedure is refined to an executable procedure

Refinement and Properties

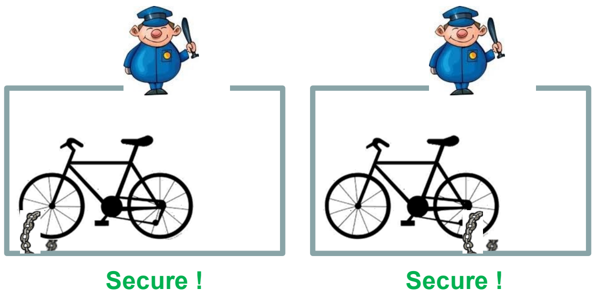
- Refinement typically preserves safety properties.
  - This means if we start with an abstract specification which we can show satisfies the desired properties, and refine it until we arrive at an implementation, we have a system for the properties hold *by construction*:

$$SP \rightsquigarrow SP_1 \rightsquigarrow SP_2 \rightsquigarrow \ldots \rightsquigarrow Imp$$

- However, **security** is typically **not** preserved by refinement nor by composition!

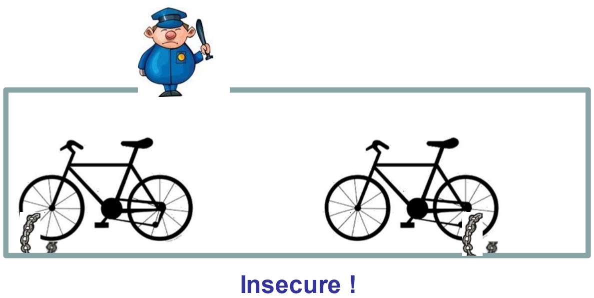
Security and Composition

Only complete bicycles are allowed to pass the gate.



Security and Composition

Only complete bicycles are allowed to pass the gate.



Conclusion & Summary

- Software development models: structure vs. flexibility
- Safety standards such as IEC 61508, DO-178B suggest development according to V-model.
  - Specification and implementation linked by verification and validation.
  - Variety of artefacts produced at each stage, which have to be subjected to external review.
- Properties include safety and liveness properties.
- Structuring of the development:
  - Horizontal – e.g. composition
  - Vertical – refinement (data, process and action ref.)