Verifikation von C-Programmen
Universität Bremen, WS 2014/15

Lecture 03 (06.11.2014)

Was ist eigentlich Verifikation?

Christoph Lüth

Synopsis

- **If** you want to write safety-criticial software,
  then you need to adhere to state-of-the-art practise
  as encoded by the relevant norms & standards.
- Today:
  - What is safety and security?
  - Why do we need it? Legal background.
  - How is it ensured? Norms and standards
    - IEC 61508 – Functional safety
    - IEC 15408 – Common criteria (security)

The Relevant Question

- If something goes wrong:
  - Whose fault is it?
  - Who pays for it?
- That is why most (if not all) of these standards put a lot of emphasis on process and traceability. Who decided to do what, why, and how?
- The **bad** news:
  - As a qualified professional, you may become personally liable if you deliberately and intentionally (*grob vorsätzlich*) disregard the state of the art.
- The **good** news:
  - Pay attention here and you will be sorted.

Safety: norms & standards

What is Safety?

- Absolute definition:
  - „Safety is freedom from accidents or losses."
    - Nancy Leveson, „Safeware: System safety and computers"
- But is there such a thing as absolute safety?
- Technical definition:
  - „Sicherheit: Freiheit von unvertretbaren Risiken"
    - IEC 61508-4:2001, §3.1.8
- Next week: a safety-critical development process

Some Terminology

- Fail-safe vs. Fail operational
- Safety-critical, safety-relevant (*sicherheitskritisch*)
  - General term -- failure may lead to risk
- Safety function (*Sicherheitsfunktion*)
  - Techncal term, that functionality which ensures safety
- Safety-related (*sicherheitsgerichtet, sicherheitsbezogen*)
  - Technical term, directly related to the safety function

Legal Grounds

- The machinery directive:
  The Directive 2006/42/EC of the European Parliament and of the Council of 17 May 2006 on machinery, and amending Directive 95/16/EC (recast)
- Scope:
  - Machineries (with a drive system and movable parts).
- Structure:
  - Sequence of whereas clauses (explanatory)
  - followed by 29 articles (main body)
  - and 12 subsequent annexes (detailed information about particular fields, e.g. health & safety)
- Some application areas have their own regulations:
  - Cars and motorcycles, railways, planes, nuclear plants ...

What does that mean?

- Relevant for **all** machinery (from tin-opener to AGV)
- **Annex IV** lists machinery where safety is a concern
- Standards encode current best practice.
  - Harmonised standard available?
- External certification or self-certification
  - Certification ensures and documents conformity to standard.
- **Result:** CE
- Note that the scope of the directive is market harmonisation, not safety – that is more or less a byproduct.

The Norms and Standards Landscape

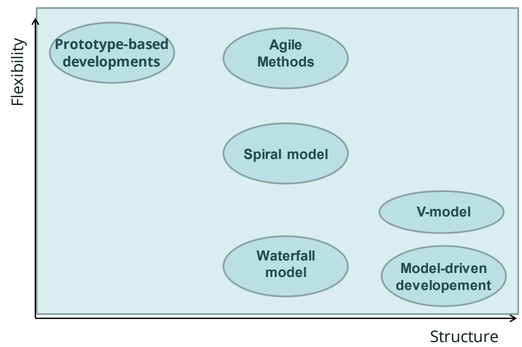
- First-tier standards (*A-Normen*):
  - General, widely applicable, no specific area of application
  - Example: IEC 61508
- Second-tier standards (*B-Normen*):
  - Restriction to a particular area of application
  - Example: ISO 26262 (IEC 61508 for automotive)
- Third-tier standards (*C-Normen*):
  - Specific pieces of equipment
  - Example: IEC 61496-3 ("*Berührungslos wirkende Schutzeinrichtungen*")
- Always use most specific norm.

Norms for the Working Programmer

- IEC 61508:
  - *"Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems (E/E/PE, or E/E/PES)"*
  - Widely applicable, general, considered hard to understand
- ISO 26262
  - Specialisation of 61508 to cars (automotive industry)
- DIN EN 50128
  - Specialisation of 61508 to software for railway industry
- RTCA DO 178-B:
  - "*Software Considerations in Airborne Systems and Equipment Certification*"
  - Airplanes, NASA/ESA
- ISO 15408:
  - *"Common Criteria for Information Technology Security Evaluation"*
  - Security, evolved from TCSEC (US), ITSEC (EU), CTCPEC (Canada)

Software Development Models

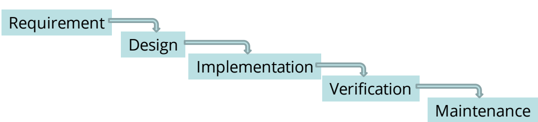
Software Development Models



from S. Paulus: Sichere Software

Waterfall Model (Royce 1970)

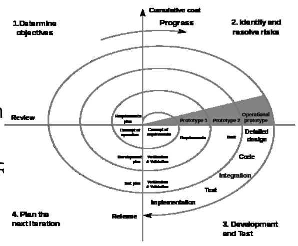
- Classical top-down sequential workflow with strictly separated phases.



- 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)

CIM → PIM → PSM → Code

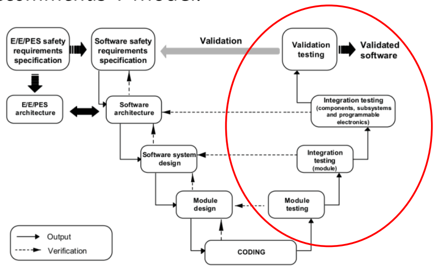
- Variety of tools:
  - Rational tool chain, Enterprise Architect
  - EMF (Eclipse Modelling Framework)
- Strictly sequential development
- Drawbacks: high initial investment, limited flexibility

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.

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.



Verification & Validation

V & V

- Verification
  - Making sure the system satisfies safety requirements
  - „Is the system built right (i.e. correctly)?“
- Validation
  - Making sure the requirements are correct and adequate.
  - „Do we build the right (i.e. adequate) system?“

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.

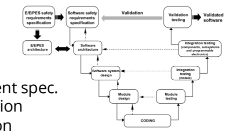
Artefacts in the Development Process

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*.

Introducing IEC 61508

Introducing IEC 61508

- Part 1: Functional safety management, competence, **establishing SIL targets**
- Part 2: Organising and managing the life cycle
- Part 3: **Software requirements**
- Part 4: Definitions and abbreviations
- Part 5: Examples of methods for the determination of safety-integrity levels
- Part 6: Guidelines for the application
- Part 7: Overview of techniques and measures

How does this work?

1. Risk analysis determines the safety integrity level (SIL)
2. A hazard analysis leads to safety requirement specification.
3. Safety requirements must be satisfied
   - Need to verify this is achieved.
   - SIL determines amount of testing/proving etc.
4. Life-cycle needs to be managed and organised
   - Planning: verification & validation plan
   - Note: personnel needs to be qualified.
5. All of this needs to be independently assessed.
   - SIL determines independence of assessment body.

Safety Integrity Levels

SIL	High Demand (more than once a year)	Low Demand (once a year or less)
4	10^{-9} < P/hr < 10^{-8}	10^{-5} < P/yr < 10^{-4}
3	10^{-8} < P/hr < 10^{-7}	10^{-4} < P/yr < 10^{-3}
2	10^{-7} < P/hr < 10^{-6}	10^{-3} < P/yr < 10^{-2}
1	10^{-6} < P/hr < 10^{-5}	10^{-2} < P/yr < 10^{-1}

- P: Probabilty of dangerous failure (per hour/year)
- Examples:
  - High demand: car brakes
  - Low demand: airbag control
- Which SIL to choose? → Risk analysis
- Note: SIL only meaningful for specific safety functions.

Establishing target SIL I

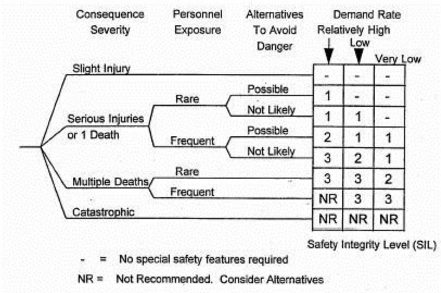
- IEC 61508 does not describe standard procedure to establish a SIL target, it allows for alternatives:
- Quantitative approach
  - Start with target risk level
  - Factor in fatality and frequency

Maximum tolerable risk of fatality	Individual risk (per annum)
Employee	10^{-4}
Public	10^{-5}
Broadly acceptable („Neglibile")	10^{-6}

- Example:
  - Safety system for a chemical plant
  - Max. tolerable risk exposure A=10^{-6}
  - B= 10^{-2} hazardous events lead to fatality
  - Unprotected process fails C= 1/5 years
  - Then Failure on Demand E = A/(B*C) = 5*10^{-3}, so SIL 2

Establishing target SIL II

- Risk graph approach



Example: safety braking system for an AGV

What does the SIL mean for the development process?

- In general:
  - „Competent" personnel
  - Independent assessment („four eyes")
- SIL 1:
  - Basic quality assurance (e.g ISO 9001)
- SIL 2:
  - Safety-directed quality assurance, more tests
- SIL 3:
  - Exhaustive testing, possibly formal methods
  - Assessment by separate department
- SIL 4:
  - State-of-the-art practices, formal methods
  - Assessment by separate organisation

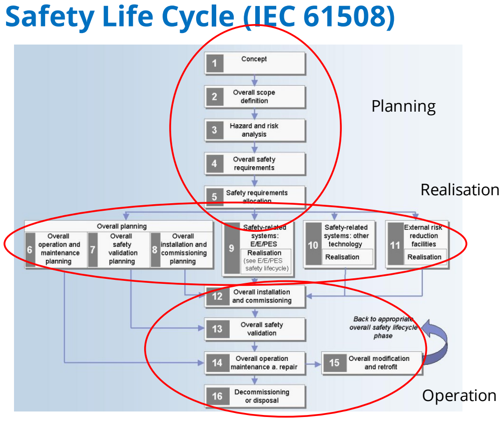
Increasing SIL by redudancy

- One can achieve a higher SIL by combining **independent** systems with lower SIL („*Mehrkanalsysteme*").
- Given two systems A, B with failure probabilities P_A, P_B, the chance for failure of both is (with P_{CC} probablity of common-cause failures):

$$P_{AB} = P_{CC} + P_A P_B$$

- Hence, combining two SIL 3 systems may give you a SIL 4 system.
- However, be aware of **systematic** errors (and note that IEC 61508 considers all software errors to be systematic).
- Note also that for fail-operational systems you need three (not two) systems.

The Safety Life Cycle (IEC 61508)



The Software Development Process

- 61508 „recommends" V-model development process
- Appx A, B give normative guidance on measures to apply:
  - Error detection needs to be taken into account (e.g runtime assertions, error detection codes, dynamic supervision of data/control flow)
  - Use of strongly typed programming languages (see table)
  - Discouraged use of certain features: recursion(!), dynamic memory, unrestricted pointers, unconditional jumps
  - Certified tools and compilers must be used.
    - Or `proven in use´

Proven in Use

- As an alternative to systematic development, statistics about usage may be employed. This is particularly relevant
  - for development tools (compilers, verification tools etc),
  - and for re-used software (in particular, modules).
  - Note that the previous use needs to be to the same specification as intended use (eg. compiler: same target platform).

SIL	Zero Failure		One Failure	
1	12 ops	12 yrs	24 ops	24 yrs
2	120 ops	120 yrs	240 ops	240 yrs
3	1200 ops	1200 yrs	2400 ops	2400 yrs
4	12000 ops	12000 yrs	24000 ops	24000 yrs

Table A.2, Software Architecture

Tabelle A.2 – Softwareentwurf und Softwareentwicklung: Entwurf der Software-Architektur (siehe 7.4.3)

Verfahren/Maßnahme *	siehe	SIL1	SIL2	SIL3	SIL4
1 Fehlererkennung und Diagnose	C.3.1	o	+	++	++
2 Fehlererkennende und -korrigierende Codes	C.3.2	+	+	+	++
3a Plausibilitätskontrollen (Failure assertion programming)	C.3.3	+	+	+	++
3b Externe Überwachungseinrichtungen	C.3.4	o	+	+	+
3c Diversitäre Programmierung	C.3.5	+	+	+	++
3d Regenerationsblöcke	C.3.6	+	+	+	+
3e Rückwärtsregeneration	C.3.7	+	+	+	+
3f Vorwärtsregeneration	C.3.8	+	+	+	+
3g Regeneration durch Wiederholung	C.3.9	+	+	+	++
3h Aufzeichnung ausgeführter Abschnitte	C.3.10	o	+	+	++
4 Abgestufte Funktionseinschränkungen	C.3.11	+	+	++	++
5 Künstliche Intelligenz – Fehlerkorrektur	C.3.12	o	--	--	--
6 Dynamische Rekonfiguration	C.3.13	o	--	--	--
7a Strukturierte Methoden mit z. B. JSD, MASCOT, SADT und Yourdon.	C.2.1	++	++	++	++
7b Semi-formale Methoden	Tabelle B.7	+	+	++	++
7c Formale Methoden z. B. CCS, CSP, HOL, LOTOS, OBJ, temporäre Logik, VDM und Z	C.2.4	o	+	+	++

Table A.4- Software Design & Development

Tabelle A.4 – Softwareentwurf und Softwareentwicklung: detaillierter Entwurf (siehe 7.4.5 and 7.4.6)
(Dies beinhaltet Software-Systementwurf, Entwurf der Softwaremodule und Codierung)

Verfahren/Maßnahme *	siehe	SIL1	SIL2	SIL3	SIL4
1a Strukturierte Methoden wie z. B. JSD, MASCOT, SADT und Yourdon	C.2.1	++	++	++	++
1b Semi-formale Methoden	Tabelle B.7	+	++	++	++
1c Formale Methoden wie z. B. CCS, CSP, HOL, LOTOS, OBJ, temporäre Logik, VDM und Z	C.2.4	o	+	+	++
2 Rechnergestützte Entwurfswerkzeuge	B.3.5	+	+	++	++
3 Defensive Programmierung	C.2.5	o	+	++	++
4 Modularisierung	Tabelle B.9	++	++	++	++
5 Entwurfs- und Codierungs-Richtlinien	Tabelle B.1	+	++	++	++
6 Strukturierte Programmierung	C.2.7	++	++	++	++

Table A.9 – Software Verification

Tabelle A.9 – Software-Verifikation (siehe 7.9)

Verfahren/Maßnahme *	siehe	SIL1	SIL2	SIL3	SIL4
1 Formaler Beweis	C.5.13	o	+	+	++
2 Statistische Tests	C.5.1	o	+	+	++
3 Statische Analyse	B.6.4 Tabelle B.8	+	++	++	++
4 Dynamische Analyse und Test	B.6.5 Tabelle B.2	+	++	++	++
5 Software-Komplexitätsmetriken	C.5.14	+	+	+	+

Table B.1 – Coding Guidelines

- Table C.1, programming languages, mentions:
  - ADA, Modula-2, Pascal, FORTRAN 77, C, PL/M, Assembler, …
- Example for a guideline:
  - MISRA-C: 2004, Guidelines for the use of the C language in critical systems.

Tabelle B.1 – Entwurfs- und Codierungs-Richtlinien (Verweisungen aus Tabelle A.4)

Verfahren/Maßnahme *	siehe	SIL1	SIL2	SIL3	SIL4
1 Verwendung von Codierungs-Richtlinien	C.2.6.2	++	++	++	++
2 Keine dynamischen Objekte	C.2.6.3	+	++	++	++
3a Keine dynamischen Variablen	C.2.6.3	o	+	++	++
3b Online-Test der Erzeugung von dynamischen Variablen	C.2.6.4	o	+	++	++
4 Eingeschränkte Verwendung von Interrupts	C.2.6.5	+	+	++	++
5 Eingeschränkte Verwendung von Pointern	C.2.6.6	o	+	++	++
6 Eingeschränkte Verwendung von Rekursionen	C.2.6.7	o	+	++	++
7 Keine unbedingten Sprünge in Programmen in höherer Programmiersprache	C.2.6.2	+	++	++	++

ANMERKUNG 1 Die Maßnahmen 2 und 3a brauchen nicht angewendet zu werden, wenn ein Compiler verwendet wird, der sicherstellt, dass genügend Speicherplatz für alle dynamischen Variablen und Objekte vor der Laufzeit zugeteilt wird, oder der Laufzeittests zur korrekten Online-Zuweisung von Speicherplatz einfügt.

* Es müssen dem Sicherheits-Integritätslevel angemessene Verfahren/Maßnahmen ausgewählt werden. Alternative oder gleichwertige Verfahren/Maßnahmen sind durch einen Buchstaben hinter der Nummer gekennzeichnet. Es muss nur eine(s) der alternativen oder gleichwertigen Verfahren/Maßnahmen erfüllt werden.

Table B.5 - Modelling

Tabelle B.5 – Modellierung (Verweisung aus der Tabelle A.7)

Verfahren/Maßnahme *	siehe	SIL1	SIL2	SIL3	SIL4
1 Datenflussdiagramme	C.2.2	+	+	+	+
2 Zustandsübergangsdiagramme	B.2.3.2	o	+	++	++
3 Formale Methoden	C.2.4	o	+	+	++
4 Modellierung der Leistungsfähigkeit	C.5.20	+	++	++	++
5 Petri-Netze	B.2.3.3	o	+	++	++
6 Prototypenerstellung/Animation	C.5.17	+	+	+	+
7 Strukturdiagramme	C.2.3	+	+	+	++

ANMERKUNG Sollte eine spezielles Verfahren in dieser Tabelle nicht vorkommen, darf nicht angenommen werden, dass dieses nicht in Betracht gezogen werden darf. Es sollte zu dieser Norm in Einklang stehen.

* Es müssen dem Sicherheits-Integritätslevel angemessene Verfahren/Maßnahmen ausgewählt werden.

Certification

- Certiciation is the process of showing **conformance** to a **standard**.
- Conformance to IEC 61508 can be shown in two ways:
  - Either that an organisation (company) has in principle the ability to produce a product conforming to the standard,
  - Or that a specific product (or system design) conforms to the standard.
- Certification can be done by the developing company (self-certification), but is typically done by an **accredited** body.
  - In Germany, e.g. the TÜVs or the Berufsgenossenschaften (BGs)
- Also sometimes (eg. DO-178B) called `qualification'.

Basic Notions of Formal Software Development

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.

Summary

- Norms and standards enforce the application of the state-of-the-art when developing software which is
  - *safety-critical* or *security-critical*.

- Safety standards such as IEC 61508, DO-178B suggest development according to V-model:
  - **Verification** and **validation** link specification and implementation.
  - Variety of artefacts produced at each stage, which have to be subjected to external review.