Topics

• Relation System and Software Engineering
• Why (automotive) software engineering?
• Process models
• V-model
• Standards
  • IEC50861
  • ISO26262
• Software Design
  • SysML
Systems engineering is the interdisciplinary field of engineering focusing on how complex engineering projects should be designed and managed over their life cycles.
(Embedded) software engineering is part of system engineering

Functionality can be implemented in
- dedicated hardware and little software
- general purpose hardware with more software

Consumer electronics moved from dedicated processors to general purpose processors
- from hardware to software
Why Software Engineering?

- The nature of software ...
  - untrained people can hack something together
    - quality problems are hard to notice
  - software is easy to modify
    - people make changes without fully understanding it
  - software does not ‘wear out’
    - it deteriorates by having its design changed:
      - erroneously, or
      - in ways that were not anticipated, thus making it complex
Why Automotive Software Engineering?

- Today cars have
  - between 30 and 100 Electronic Control Units (ECU) connected by more than
  - 5 different bus systems
  - 10M-100M LOC
  - more than 2000 functions are controlled by software
- Up to 40 % of the production costs of a car are due to electronics and software
- In 30 years the amount of software in cars went from 0 lines of code to more than 10,000,000 lines of code
Why Automotive Software Engineering?

• **Embedded Software as Innovation Driver**
  • Software is today the most crucial innovation driver for technical systems, in general
  • By software
    – innovative functions are realized,
    – new ways of implementing known functionality with reduced costs, less weight, less emission, or higher quality,
    – energy and fuel is saved
    – we combine functions and correlate them into multi-functional systems
What is Software Engineering?

• Large, high quality software systems
  • software engineering techniques are needed because large systems cannot be completely understood by one person
  • identification of missing quality aspects before building
    – end-product must be of high quality
• teamwork and co-ordination are required
  – key challenge: dividing up the work and ensuring that the parts of the system work properly together
What is Software Engineering?

- Cost, time and other constraints
  - finite resources
  - benefit must outweigh the cost

- Quality attributes:
  - usability, efficiency, reliability, maintainability, reusability
  - different qualities can conflict
    - increasing efficiency can reduce maintainability
    - increasing usability can reduce efficiency
Software Quality...

• **Usability**
  • Users can learn it and fast and get their job done easily

• **Efficiency**
  • It doesn’t waste resources such as CPU time and memory

• **Reliability**
  • It does what it is required to do without failing

• **Maintainability**
  • It can be easily changed

• **Reusability**
  • Its parts can be used in other projects, so reprogramming is not needed
What is Automotive Software Engineering?

• Today the engineering of software in cars is still in its infancy
• Lifecycle management of software in cars is in its early stage
  • Many suppliers and even some OEMs are not even at Capability Maturity Model level 2
• Reuse of solutions from one car to the next is insufficient and only done in a consequent way in some limited areas
What is Automotive Software Engineering?

• In many sub-domains the functionality from one car generation to the next is only changed and enhanced by 10% while more than 90% of the software is rewritten
  • reason is a low level, hardware specific implementation, which makes it difficult to change, adopt, and port existing code
  • amount of automation in software production for software in cars is quite low
What is Automotive Software Engineering?

• Process models:
  • Capability Maturity Model Integration® (CMMI)
  • Software Process Improvement and Capability Determination (SPICE)
  • V-Model

• Standards:
  • IEC 61508 (Functional Safety standard)
  • ISO26262 (Functional Safety standard)
  • MISRA-C
Software process models

• Waterfall model
• Agile software development
• Prototyping
• Incremental development
• Rapid application development
• V-model
Software development models

- How the customer explained it
- How the Project Leader understood it
- How the Analyst designed it
- How the Programmer wrote it
- How the Business Consultant described it

- How the project was documented
- What operations installed
- How the customer was billed
- How it was supported
- What the customer really needed
Software development model

Waterfall model

- requirements engineering
  - V&V
- design
  - V&V
- implementation
  - V&V
- testing
  - V&V
- maintenance
  - V&V
Software development model

• Waterfall model
  • Document oriented
  • Suited for (very) large projects (> 50 people)
  • Too many design activities during coding and testing
Software development model

- Comprehensive documentation

- ESA Life cycle:
  - User Requirements Document
  - Software Requirements Document
  - Architectural Design Document
  - Detailed Design Document
  - Software Transfer Document
  - Project History Document

- Software Project Management Plan
- Software Configuration Management Plan
- Software Validation and Verification Plan
- Software Quality Assurance Plan
- Meeting minutes
- Progress reports
- …
Software development model

- See http://agilemanifesto.org/
- Agile methods
  - Individuals and interactions are more important than processes and tools
  - Working software is more important than comprehensive documents
  - Customer collaboration is more important than contract negotiation
  - Responding to change is more important than following a plan
Software development model

• Processes and tools
  • Fixed hierarchy, roles and team structure
  • Many, many rules
  • Management of process, not people
  • Emphasis on process, not customer value
Software development model

• XP practices
  • Co-location
  • Self organizing teams
  • Pair programming
  • Collective code ownership
Software development model

- Working software
  - User stories
    - As a librarian, I want to be able to search for books by publication year.
- Unit testing
- Continuous build and integration
Software development model

• Customer collaboration
  • Scrum: Prioritized backlog of user stories
  • Must be accessible for questions
  • Frequent delivery of working software
  • Acceptance testing
Software development model
Software development model

• Agile methods
  • No extensive architectural or design phase
  • No energy spend on documentation
Software development model

- Extreme programming in practice:
  - Planning game
  - Small releases
  - Metaphor
  - Simple design
  - Testing
  - Refactoring
  - Pair programming
  - Collective ownership
  - Continuous integration
  - 40-hour week
  - On-site customer
  - Coding standards
Software development model

Prototyping

requirements engineering → design → implementation → testing

design → implementation → testing

design → implementation → testing

maintenance
Software development model

• Advantages of prototyping
  • Resulting system is easier to use
  • Resulting system has less features
  • User needs are better accommodated
  • Design is of higher quality
  • Problems are detected earlier
  • Resulting system is easier to maintain
  • Development costs less effort
Software development model

- Disadvantages of prototyping
  - Resulting system has more features
  - Design is of lower quality
  - Performance of resulting system is worse
  - Resulting system is harder to maintain
  - Team members should be more experienced
Software development model

• Prototyping
  • Is useful in situations with unclear or ambiguous requirements
  • Is useful in when emphasis is on user interface
  • Users and designers must be aware of pitfalls
  • Must planned and controlled
Software development model

- Incremental development
Software development model

- Incremental development
  - First focusing on essential features
  - Additional functionality is only included if needed
  - Resulting systems are leaner but provide sufficient support to the user
Software development model

• Rapid application development (RAD)
  • Similar to iterative development process models
    – User involvement
    – Prototyping
    – Reuse
    – Automated tools
    – Small development teams
• Time boxing
Software development model

• RAD has four phases:
  • Requirements planning
  • Application design
  • Construction
  • Cutover (testing, training, installation)

• MoSCoW:
  • Must haves
  • Should haves
  • Could haves
  • Won’t haves
Software development model

V-model

- Requirements Analysis
- High Level Design
- Detailed Specifications
- Coding
- Integration Testing
- Unit Testing
- Operational Testing
- Review/Test
- Ongoing Support
Original V-model
V-model

- The Lifecycle Development “V” Model or “V-model” for short, is a framework or structure for undertaking the design, execution, and commissioning of a design project.
- It is called the V-model because of its characteristic “V” shape. The left hand edge of the V is where the project is defined and specified in greater and greater detail. The bottom point of the V is the execution step of the project. The right-hand edge of the V is where the commissioning and qualification testing of the installed system is performed.
- The V-model provides a logical sequence that helps to organize the complex activities of defining a project scope, executing it, and qualifying it.
- The V-model creates a solid basis for testing the functionality of the system against the original design specs and proving that it does what it is supposed to do.
V-model

- V-Model is a systems development model designed to simplify the understanding of the complexity associated with developing systems.
- In systems engineering it is used to define a uniform procedure for product or project development.
V-model
V-model

• User Requirements Specification (URS)
  • Meant to be a very high level view of the overall project drivers, in terms of the over-arching project objectives and the underlying business case.
  • Should be phrased in strategic business terms.
  • Should talk about total installed cost targets, resultant net capacity, it should discuss unit costs, cycle time, locations, raw materials, labor content.
  • Should address all the critical issues that will ultimately influence whether or not the project is deemed a success or not.
V-model

- **User Requirements Specification (URS)**
  - Functional Requirements Spec (FRS) is a subsection of the URS. It is reserved for the required functionality of the system.
  - The FRS is also written from a very high level perspective, dealing more with overall capabilities rather than getting into specifics of how the system is to be operated.
  - The URS/FRS should not contain specific design stipulations unless the client has already invested time, money, and/or resources toward achieving a very specific objective.
V-model

• **Functional Design Specification (FDS)**
  • Prepared by a Vendor or Supplier in response to the URS.
  • Should address all the elements of the URS, providing sufficient detail to indicate how vendor is planning on meeting all the user requirements.
  • There are two flavors of FDS documents: One in which the project is being bid out competitively, and the other where the work has already been awarded or is being sole-sourced.
  • The FDS provides a deeper layer of detail, so, in the first case, the FDS provides the client with the basis for comparing and evaluating the technical merit and cost parameters of the proposed solution.
V-model

- **Functional Design Specification (FDS)**
  - In the second case, the FDS is intended to begin engaging the client in design details and decisions that will influence the final product.
  - All major functionality aspects should be documented and agreed upon at the completion of the FDS.
  - The completed, approved FDS will form the basis for design change control management. As functions and features are added or subtracted from the design scope, the URS / FRS and the FDS should be updated and re-issued to all involved members of the project team for impact analysis and possible re-costing of the job.
V-model

• Detailed Design Specification (DDS)
  • Detailed Design Specs encompass all design documents produced by the Vendor or Supplier of services.
  • DDS may be a single document or it may be the entire body of detailed design deliverables that are generated during the course of the design phase.
  • At the time when the Detailed Design Specs are being prepared, the scope of work should be very clear and the detailed design is generated to fulfill and implement the features that were described in the Functional Design Spec.
V-model

- Design Qualification (DQ)
  - The purpose of Design Qualification is to keep track of any changes that have occurred in the Design Specifications throughout the period of time that active design work has been going on.
  - Specific items added to or deleted from the scope of the project must be picked up and carried backward through the appropriate design specs and requirements documents, as well as the appropriate qualification protocols.
• Design Qualification (DQ)
  • A traceability matrix is used for keeping track of all the design elements. It is appropriate to have at least one or more enhanced design reviews in which the traceability matrix is reviewed, along with drawings and/or other design documents, so that all involved parties are fully apprised of the status of the design.
  • Other issues that should be addressed during the Enhanced Design Reviews include Constructability issues, Process Safety Management issues (PHA, HAZOP, FMEA, etc), Construction Safety Issues, Construction Coordination, Shutdown Management, and Commissioning Planning and Execution Strategy.
V-model

• Implementation Phase
  • This is the phase of the project during which the actual physical work is performed.
  • Construction of the equipment occurs during this phase, as does the programming of the software.
  • Note that the Implementation Phase is the “turning point” of the V-model. At this point, all the tasks turn from specification of what is supposed to happen to verification of what actually happens.
  • Very critical that any RFI and As-built information is recorded and worked back into the detailed design documents!
V-model

• Installation Qualification (IQ)
  • This is the first of the qualification protocols that is executed during the start-up and commissioning phase of a project.
  • The IQ ascertains and documents that the equipment that was ultimately installed is truly what was described by the design documents.
  • It is often described as a “hands in the pocket” activity, in that no equipment is actually operated during IQ.
  • Part numbers are verified off the original design drawings, parts lists, bills of materials, and the like.
Installation Qualification (IQ)

- P&ID’s are walked down in the field insuring that all piping runs match up with what is shown on the drawings.
- Note that the IQ protocol is directly across from the Detailed Design Spec, with an arrow between the two. This is meant to imply that the IQ Protocol is written directly from the Detailed Design Spec.
- Every element of the Detailed Design Spec should be listed and checked by the IQ.
• **Operation Qualification (OQ)**
  • The OQ is the second qualification protocol that is executed in the start-up and commissioning phase of a project.
  • During OQ, the equipment is operated to demonstrate that the performance parameters of the various pieces of equipment conform to the design parameters that were originally specified.
  • Things like temperature, flow, and pressure ranges are checked and verified.
• **Operation Qualification (OQ)**
  
  • Accuracy and precision of these parameters and conformance to requirements are verified and documented. This can involve water batches, pilot runs, and not-for-production runs.

  • Note that the OQ protocol is directly across from the Functional Design Spec, with an arrow between the two. This is meant to imply that the OQ Protocol is written directly from the Functional Design Spec. Every element of the Functional Design Spec should be listed and checked by the OQ.
V-model

- Performance Qualification (PQ)
  - The PQ is where the final functionality of the equipment is truly verified, in that product is actually made under production conditions, and the quality of the product is compared to the product specs.
  - Generally, three consecutive batches are made as part of a Qualification run. All three batches must be within product specs in order for the Qualification to pass.
  - Note that the PQ protocol is directly across from the User Requirements Doc, with an arrow between the two. This is meant to imply that the PQ Protocol is written directly from the User Requirements Doc. Every element of the User Requirements Doc should be listed and checked by the PQ.
V-model

- Underlying principles
  - Independence of methods and tools
  - Independence of organizations
  - Separation into “submodels”
  - Orientation to activities and products
  - Adaptability of the model
V-Model

- V-Model provides guidance for planning and realization of projects.
  - Minimization of Project Risks:
    - improves project transparency and project control by specifying standardized approaches and describing the corresponding results and responsible roles.
    - permits an early recognition of planning deviations and risks and improves process management, thus reducing the project risk.
  - Improvement and Guarantee of Quality:
    - ensures that the results to be provided are complete and have the desired quality.
    - defined interim results can be checked at an early stage.
    - uniform product contents will improved readability, understandability and verifiability.
V-model

• V-Model provides guidance for planning and realization of projects.
  • Reduction of Total Cost over the Entire Project and System Life Cycle:
    - effort for the development, production, operation and maintenance of a system can be calculated, estimated and controlled in a transparent manner
    - results obtained are uniform and easily retraced.
  • Improvement of Communication between all Stakeholders:
    - standardized and uniform description of all relevant elements and terms is the basis for the mutual understanding between all stakeholders
V-model

- The value of standardization
  - Reduction of cost in the entire lifecycle
  - Improvement of software quality
  - Better communication between customer and contractor

- Regulations on 3 levels
  - Procedure (what)
  - Allocation of methods (how)
  - Functional tool requirements (what)
V-model

- All levels of the regulations are structured according to activity areas
  - Systems development
  - Quality assurance
  - Configuration management
  - Project management

- Development standard developed for each area.
V-model

- Refines high-level system architectures, decomposes these into smaller components, finally leading to system realization.
- Requires many design decisions, covering multiple disciplines and multiple concerns, such as:
  - Performance
  - Reliability
  - Evolvability
  - Resource usage
  - Energy usage
  - Costs
V-model

- Testing & Integration = composition, i.e., combining and checking components
- T&I is expensive, time-consuming (30-70%), with sub-optimal quality results
V-model

• Procedure
  • What has to be done:
    – Activities to be carried out
    – Results that have to be produced
    – Content of the results
    – Roles
  • Lifecycle process model
Tailoring

- No important document is “forgotten”
- Tailoring relevant for tendering
  - Selection of the necessary activities and products
  - Deletion conditions
  - The resulting subset of the Lifecycle is put together in the Project Manual

Technical Tailoring

- Adapting to conditions during the course of the project
V-model
V-model

• **Software development**
  • **SD1-System Req. analysis**
    - Description of the requirements of the system and its environment,
    - Risk Analysis, and
    - User requirements
  • **SD2–System design**
    - Segmentation of the system into SW/HW
  • **SD3-SW/HW requirement analysis**
    - Detail Technical Requirements
  • **SD4-Preliminary SW design**
    - Structuring of the interfaces and interaction of SW components
V-model

- **Software development**
  - SD5-Detailed SW design
    - Description of functionality, Data administration and error handling of SWC
  - SD6-SW implementation
    - Coding in chosen programming language
  - SD7-SW integration
    - Integration of modules
  - SD8-System integration
    - Integration of SW and HW components
  - SD9-Transition to utilization
    - Activities for Deployment into Production
V-model
V-model

• Quality assurance
  • QA1-Initialization
    – Generated the QA and Assessment Plan
  • QA2-Assessment Preparation
    – Generation of unambiguous Assessment specification and procedures & Req. of Assessment Environment
  • QA3-Process Assessment of Activities
    – Specified procedures were adhered to during the realization of an activity
  • QA4-Product Assessment
    – Assessment with respect to the formal criteria & the contents of the product. Assessment Report generated.
  • QA5-Reporting
    – Assessment Reports are assessed in regular intervals and the results submitted to PM.
V-model

- Configuration management
  - CM1-CM Initialization
    - Generation of the CM plan and setting of the CM resources
  - CM2-Product and CM
    - Administration of products, configurations and rights
  - CM3-Change Management
    - Controlled artifacts are recorded and administered
  - CM4-CM Services
    - General services (e.g. Product Catalog)
V-model

• Project management
  • PM1-Project Initialization
  • PM2-Placement/Procurement
  • PM3-Contractor Management
  • PM4-Detailed Planning
  • PM5-Cost/Benefit Analysis
  • PM6-Phase Review
  • PM7-Risk Management
V-model

- Project management
  - PM8-Project Control
  - PM9-Information Service/Reporting
  - PM10-Training/Instruction
  - PM11-Supplying Resources
  - PM12-Allocation of Work Orders
  - PM13-Staff Training
  - PM14-Project Completion
    - Final Project Report
Example for the activity and product flow within the four submodels
V-model

• Tool requirements
  • What is to be used to do something:
    − Functional characteristics must the tools have
  • Introduces the Software Development Environment
    − User Interface
    − Work flow management
    − Security and integrity requirements
    − Software development
    − Quality assurance
    − Configuration Management
    − Project Management

• Use for:
  − Selection of tools
  − Evaluation
  − Further development of tools
V-model and CMM

![Diagram showing comparison between V-model and CMM]
What’s it good for?

- The original mission of the V-model was to impart a high degree of quality assurance in the development efforts for the design of automated systems and the creation of the associated software programming.
- The primary goal of the guideline was to establish the roles and responsibilities of the involved parties, and to create strong links between the project specifications and the acceptance testing done at the end of the project.
- The basic elements and techniques of the V-model apply to virtually all engineering projects, not just automation and software!
V-model

What are the Benefits?

- The V model creates a tight linkage between the design specifications and the subsequent test protocols and methods used during start up.
- Using the v-model virtually assures that the project, as installed, will fulfill the objectives that were intended for it.
- By using the v-model, the project design is documented thoroughly and fully capable of being validated.
- Lastly, the V model provides an excellent basis for design control and tracking design changes through the proceeding of the project.
Standards

• Most important requirement in Automotive:
  • A vehicle should not harm its passengers or (people in) its environment

• IEC 61508 (Functional Safety standard)
• ISO26262 (Functional Safety standard)
Standards

- IEC 61508 (Functional Safety standard)
- Categories of likelihood of occurrence

<table>
<thead>
<tr>
<th>Category</th>
<th>Definition</th>
<th>Range (failures per year)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequent</td>
<td>Many times in system lifetime</td>
<td>$&gt; 10^{-3}$</td>
</tr>
<tr>
<td>Probable</td>
<td>Several times in system lifetime</td>
<td>$10^{-3}$ to $10^{-4}$</td>
</tr>
<tr>
<td>Occasional</td>
<td>Once times in system lifetime</td>
<td>$10^{-4}$ to $10^{-5}$</td>
</tr>
<tr>
<td>Remote</td>
<td>Unlikely times in system lifetime</td>
<td>$10^{-5}$ to $10^{-6}$</td>
</tr>
<tr>
<td>Improbable</td>
<td>Very unlikely to occur</td>
<td>$10^{-6}$ to $10^{-7}$</td>
</tr>
<tr>
<td>Incredible</td>
<td>Cannot believe that it could occur</td>
<td>$&lt; 10^{-7}$</td>
</tr>
</tbody>
</table>
Standards

- IEC 61508 (Functional Safety standard)
- Consequence categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catastrophic</td>
<td>Multiple loss of life</td>
</tr>
<tr>
<td>Critical</td>
<td>Loss of a single life</td>
</tr>
<tr>
<td>Marginal</td>
<td>Major injuries to one or more persons</td>
</tr>
<tr>
<td>Negligible</td>
<td>Minor injuries at worst</td>
</tr>
</tbody>
</table>
**Standards**

- **IEC 61508 (Functional Safety standard)**
  - These are typically combined into a risk class matrix

<table>
<thead>
<tr>
<th>Likelihood</th>
<th>Catastrophic</th>
<th>Critical</th>
<th>Marginal</th>
<th>Negligible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequent</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>II</td>
</tr>
<tr>
<td>Probable</td>
<td>I</td>
<td>I</td>
<td>II</td>
<td>III</td>
</tr>
<tr>
<td>Occasional</td>
<td>I</td>
<td>II</td>
<td>III</td>
<td>III</td>
</tr>
<tr>
<td>Remote</td>
<td>II</td>
<td>III</td>
<td>III</td>
<td>IV</td>
</tr>
<tr>
<td>Improbable</td>
<td>III</td>
<td>III</td>
<td>IV</td>
<td>IV</td>
</tr>
<tr>
<td>Incredible</td>
<td>IV</td>
<td>IV</td>
<td>IV</td>
<td>IV</td>
</tr>
</tbody>
</table>
Standards

• IEC 61508 (Functional Safety standard)
  • Categories in risk class matrix
    − Class I: Unacceptable in any circumstance;
    − Class II: Undesirable: tolerable only if risk reduction is impracticable or if the costs are grossly disproportionate to the improvement gained;
    − Class III: Tolerable if the cost of risk reduction would exceed the improvement;
    − Class IV: Acceptable as it stands, though it may need to be monitored.
Standards

• Safety Integrity Level is determined primarily from the assessment of three factors:
  1. Improved reliability
  2. Failure to safety
  3. Management, Systematic Techniques, Verification and Validation

SIL refers to a single method of reducing injury (as determined through risk analysis), not an entire system, nor an individual component
Standards

• **Improved Reliability**
  • For systems that operate continuously (continuous mode) the allowable frequency of failure must be determined
  • For systems that operate more than once a year (high demand) the allowable frequency of failure must be determined
  • For systems that operate intermittently (less than once a year / low demand) the probability of failure is specified as the probability that the system will fail to respond on demand
Standards

• Failure to Safety
  • Calculation of safe failure fraction (SFF) determines how Fail-safe the system is

• Management, Systematic Techniques, Verification and Validation
  • Specific techniques ensure that mistakes and errors are avoided across the entire life-cycle.
  • Errors introduced anywhere from the initial concept, risk analysis, specification, design, installation, maintenance and through to disposal could undermine even the most reliable protection
• ISO 26262 is the adaptation of IEC 61508 to comply with needs specific to the application sector of E/E systems within road vehicles:
  • Provides an automotive safety lifecycle (management, development, production, operation, service, decommissioning) and supports tailoring the necessary activities during these lifecycle phases.
  • Provides an automotive-specific risk-based approach for determining risk classes (Automotive Safety Integrity Levels, ASILs).
# Standards

<table>
<thead>
<tr>
<th>IEC 61508</th>
<th>ISO 26262</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part 1: General requirements</td>
<td>Part 1: Vocabulary</td>
</tr>
<tr>
<td>Part 2: Requirements for electrical/electronic/</td>
<td>Part 2: Management of functional safety</td>
</tr>
<tr>
<td>programmable electronic safety-related systems</td>
<td></td>
</tr>
<tr>
<td>Part 3: Software requirements</td>
<td>Part 3: Concept phase</td>
</tr>
<tr>
<td>Part 4: Definitions and abbreviations</td>
<td>Part 4: Product Development: System Level</td>
</tr>
<tr>
<td>Part 5: Examples of methods for the determination of safety integrity levels</td>
<td>Part 5: Product Development: Hardware Level</td>
</tr>
<tr>
<td>Part 6: Guidelines on the application of parts 2 and 3</td>
<td>Part 6: Product Development: Software Level</td>
</tr>
<tr>
<td>Part 7: Overview of techniques and measures</td>
<td>Part 7: Production and Operation</td>
</tr>
<tr>
<td></td>
<td>Part 8: Supporting Processes</td>
</tr>
<tr>
<td></td>
<td>Part 9: ASIL-orientated and safety-oriented analysis</td>
</tr>
<tr>
<td></td>
<td>Part 10: Guideline</td>
</tr>
</tbody>
</table>
Standards

**ASIL**

- ISO 26262 replaces SILs with ASILs (Automotive Safety Integrity Levels)
- ASILs designed to specify the measures required to avoid unreasonable residual risk
- ASIL levels A-D, with D being the most demanding
- Risk of each hazardous event is evaluated on the basis of:
  - Frequency of the situation (or “exposure”)
  - Impact of possible damage (or “severity”)
  - Controllability
Standards

- **ISO 26262:**
  - Uses ASILs for specifying the item's necessary safety requirements for achieving an acceptable residual risk.
  - Provides requirements for validation and confirmation measures to ensure a sufficient and acceptable level of safety is being achieved.
  - Covers functional safety aspects of the entire development process (including such activities as requirements specification, design, implementation, integration, verification, validation, and configuration).
Standards

• Specification of the Technical Safety Requirements in ISO26262
  • Objective is to develop the technical safety requirements, which refine the functional safety concept considering the preliminary architectural design.
  • To verify through analysis that technical safety requirements comply to the functional safety requirements.
  • To bring item-level functional safety requirements into system-level technical safety requirements, down to the allocation to hardware and software elements.

Requirements Engineering of this nature requires additional initial effort but benefits will be obtained whenever changes are required.
Standards

• System Design
  • To develop the *system design* and the *technical safety concepts* that comply with the functional requirements and the technical safety requirements specification.
  • Verify the *system design* and *technical safety concept* comply with technical safety requirements specification.
  • Need to have bidirectional traceability between *system design* and *technical safety requirements* specification.
• **Item integration and testing**
  - To integrate the different parts that compose the system, included other technologies and/or external entities, and to test the obtained product to comply with each safety requirement and to verify that the design has been correctly implemented.
  - The integration and testing is carried out from software-hardware integration, and going through integration of systems up to vehicle integration, with specific tests performed at each integration phase.
  - Each functional and technical safety requirements shall be tested at least once in the complete integration phase.
### Item integration and testing

Each functional and technical safety requirements shall be tested at least once in the complete integration phase.

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL A</th>
<th>ASIL B</th>
<th>ASIL C</th>
<th>ASIL D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a - Analysis of requirements</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1b - Analysis of internal and external interfaces</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1c - Generation and analysis of equivalence classes for hardware/software integration</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1d - Analysis of boundary values</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1e - Knowledge or experience-based error guessing</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1f - Analysis of functional dependencies</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1g - Analysis of common limit conditions, sequences, and sources of common cause</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1h - Analysis of environmental conditions and operational use cases</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1i - Analysis of field experience</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL A</th>
<th>ASIL B</th>
<th>ASIL C</th>
<th>ASIL D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a - Requirements-based test</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1b - Fault injection test</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>1c - Back-to-back test</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>

**Notes:**
- **++** The method is highly recommended for this ASIL.
- **+** The method is recommended for this ASIL.
- **No** The method has no recommendation for or against its usage for this ASIL.
Standards

• Safety Validation
  • To provide evidence of due compliance with the functional safety goals and that the safety concepts are appropriate for the functional safety of the item.
  • To provide evidence that the safety goals are correct, complete and fully achieved at vehicle level.
  • The validation plan shall include:
    1. The configuration of the item
    2. The specification of test cases and acceptance criteria
    3. The required environmental conditions
Standards

• Functional safety assessment
  • To assess the functional safety that is achieved by the item.
Standards

• Release for production
  • To specify the criteria for the release for production at the completion of the item development.
  • The release for production confirms that the item complies with the requirements for functional safety at vehicle level.
  • The documentation shall include
    a) the name and signature of the person in charge of release
    b) the version/s of the released item
    c) the configuration of the released item
    d) references to associated documents
    e) the release date
Standards

• Product Development – Software Level
  • ISO 26262 (Part 6) refers more specifically to the development of software, particularly:
    − Initiation of product development at the software level
    − Derivation of software safety requirements from the system level (following from part 4) and their subsequent verification
    − Software architectural design
    − Software unit design and implementation
    − Software unit testing, and
    − Software integration and testing
Standards

- **Initiation of Product Development at Software Level**
  - To plan and initiate the functional safety activities for the sub phases of the software development
Standards

• Modeling and Coding Guidelines
  • To support the correctness of the design and implementation, the design and coding guidelines for the modeling, or programming languages, shall address following topics:

<table>
<thead>
<tr>
<th>Topics</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a</td>
<td>++</td>
</tr>
<tr>
<td>1b</td>
<td>++</td>
</tr>
<tr>
<td>1c</td>
<td>++</td>
</tr>
<tr>
<td>1d</td>
<td>++</td>
</tr>
<tr>
<td>1e</td>
<td>++</td>
</tr>
<tr>
<td>1f</td>
<td>++</td>
</tr>
<tr>
<td>1g</td>
<td>++</td>
</tr>
<tr>
<td>1h</td>
<td>++</td>
</tr>
</tbody>
</table>

---

\[ a \] An appropriate compromise of this method with other methods in ISO 26262-6 may be required.

\[ b \] The objectives of method 1b are
- Exclusion of ambiguously defined language constructs which might be interpreted differently by different modellers, programmers, code generators or compilers.
- Exclusion of language constructs which from experience easily lead to mistakes, for example assignments in conditions or identical naming of local and global variables.
- Exclusion of language constructs which might result in unhandled run-time errors.

\[ c \] The objective of method 1c is to impose principles of strong typing where these are not inherent in the language.
Standards

• Specification of Software Safety Requirements
  • To specify the software safety requirements which are derived from technical safety concept and system design to detail hardware-software requirements and verify that the software safety requirements are consistent with the technical safety concept and the system design specification.
  • The technical safety requirements are divided into hardware and software safety requirements. The specification of the software safety requirements considers constraints of the hardware and the impact of these constraints on the software.
Standards

- Verification of Software Safety Requirements
  - A verification activity shall be performed to demonstrate that the software safety requirements are traceable with the technical safety requirements, the system design and consistent with the relevant parts of the hardware safety requirements achieving complete traceability.

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a Informal verification by walkthrough</td>
<td>++</td>
</tr>
<tr>
<td>1b Informal verification by inspection</td>
<td>+</td>
</tr>
<tr>
<td>1c Semi-formal verification</td>
<td>+</td>
</tr>
<tr>
<td>1d Formal verification</td>
<td>0</td>
</tr>
</tbody>
</table>

*a Method 1c can be supported by executable models.
Standards

Software Design

- To develop a software architectural design that realizes the software safety requirements and verify the software architectural design achieving bi-directional traceability.
- The software architectural design represents all software components and their interactions with one another in a hierarchical structure.
Standards

• Software Design
  • The software architectural design shall exhibit
    − Modularity
    − Encapsulation
    − Minimum complexity

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a Hierarchical structure of software components</td>
<td>A: ++ B: ++ C: ++ D: ++</td>
</tr>
<tr>
<td>1b Restricted size of software components(^a)</td>
<td>A: ++ B: ++ C: ++ D: ++</td>
</tr>
<tr>
<td>1c Restricted size of interfaces(^b)</td>
<td>A: +   B: +   C: +   D: +</td>
</tr>
<tr>
<td>1d High cohesion within each software component(^b)</td>
<td>A: +   B: ++  C: ++  D: ++</td>
</tr>
<tr>
<td>1e Restricted coupling between software components(^a, b, c)</td>
<td>A: +   B: ++  C: ++  D: ++</td>
</tr>
<tr>
<td>1f Appropriate scheduling properties</td>
<td>A: ++ B: ++ C: ++ D: ++</td>
</tr>
<tr>
<td>1g Restricted use of interrupts(^a, d)</td>
<td>A: +   B: +   C: +   D: ++</td>
</tr>
</tbody>
</table>

\(^a\) In methods 1b, 1c, 1e and 1g “restricted” means to minimise in balance with other design considerations.

\(^b\) Methods 1d and 1e can, for example, be achieved by separation of concerns which refers to the ability to identify, encapsulate, and manipulate those parts of software that are relevant to a particular concept, goal, task, or purpose.

\(^c\) Method 1e addresses the limitation of the external coupling of software components.

\(^d\) Any interrupts used have to be priority-based.
Standards

Verification of Architectural Design

- The software architectural design shall be verified by using the following software architectural design verification methods:

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a</td>
<td>++</td>
</tr>
<tr>
<td>1b</td>
<td>+</td>
</tr>
<tr>
<td>1c</td>
<td>+</td>
</tr>
<tr>
<td>1d</td>
<td>o</td>
</tr>
<tr>
<td>1e</td>
<td>o</td>
</tr>
<tr>
<td>1f</td>
<td>+</td>
</tr>
<tr>
<td>1g</td>
<td>+</td>
</tr>
</tbody>
</table>

- Informal verification is used to assess whether the software requirements are completely and correctly refined and realised in the software architectural design. In the case of model-based development this method can be applied to the model.
- Method 1c requires the usage of executable models for the dynamic parts of the software architecture.
- Control and data flow analysis can be carried out informally, semi-formally or formally.
- Control and data flow analysis may be limited to safety-related components and their interfaces.
Standards

• **Software Unit design & implementation**
  • To specify the software units in accordance with the SW architectural design and the associated SW safety requirements, to implement the software units as specified and to verify the design of the SW units and implementation.
  • The specification of the software units shall describe the functional behavior and the internal design. The design and implementation of software unit shall achieve
    − Avoidance of unnecessary complexity
    − Testability
    − Maintainability
Standards

• Software Unit Design
  • The design principles for software unit design and implementation shall be applied to follow below properties
    – Correct execution order
    – Interface consistency
    – Correct data/control flow
    – Simplicity
    – Readability and comprehensibility
    – Robustness
    – Change suitability
    – Testability
## Standards

- **Software Unit Design**

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a One entry and one exit point in subprograms and functions&lt;sup&gt;a&lt;/sup&gt;</td>
<td>++</td>
</tr>
<tr>
<td>1b No dynamic objects or variables, or else online test during their creation&lt;sup&gt;a, b&lt;/sup&gt;</td>
<td>++</td>
</tr>
<tr>
<td>1c Initialisation of variables</td>
<td>++</td>
</tr>
<tr>
<td>1d No multiple use of variable names&lt;sup&gt;a&lt;/sup&gt;</td>
<td>++</td>
</tr>
<tr>
<td>1e Avoid global variables or else justify their usage&lt;sup&gt;a&lt;/sup&gt;</td>
<td>+</td>
</tr>
<tr>
<td>1f Limited use of pointers&lt;sup&gt;a&lt;/sup&gt;</td>
<td>+</td>
</tr>
<tr>
<td>1g No implicit type conversions&lt;sup&gt;a, c&lt;/sup&gt;</td>
<td>+</td>
</tr>
<tr>
<td>1h No hidden data flow or control flow&lt;sup&gt;b, d&lt;/sup&gt;</td>
<td>+</td>
</tr>
<tr>
<td>1i No unconditional jumps&lt;sup&gt;a, c, d&lt;/sup&gt;</td>
<td>++</td>
</tr>
<tr>
<td>1j No recursions</td>
<td>++</td>
</tr>
</tbody>
</table>

---

<sup>a</sup> Methods 1a, 1b, 1c, 1d, 1e, 1f, 1g and 1i may not be applicable for graphical modelling notations used in model-based development.

<sup>b</sup> If these compiler features are “tool qualified” in accordance with ISO 26262-8:— Clause 10. Method 1b need not be applied if a compiler is used which ensures that there will be enough program storage allocated for all dynamic variables and objects before run-time or which inserts run-time tests for correct online-allocation of program storage, i.e. stack bounds checking.

<sup>c</sup> Methods 1g and 1i are not applicable in assembler programming.

<sup>d</sup> Methods 1h and 1i reduce the potential for modelling data flow and control flow through jumps or global variables.
Standards

• Verification of software unit design and implementation
  • The software unit design and implementation shall be verified to demonstrate
    – Compliance with the hardware software interface
    – Completeness regarding the software safety requirements and the software architecture through traceability
    – Compliance of the source code with its specification
    – Compliance of the source code with the coding guidelines
    – Compatibility of the software unit implementations with target hardware.
Standards

• Software Unit Testing
  • Software units fulfill the software unit specifications and do not contain undesired functionality.
  • The following testing methods can be used for proving compliance with specification and Hardware Software interface, correct implementation, absence of unintended functionality, robustness, and sufficiency of the resources.

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a Requirement-based test</td>
<td>++</td>
</tr>
<tr>
<td>1b Interface test</td>
<td>++</td>
</tr>
<tr>
<td>1c Fault injection test(^a)</td>
<td>+</td>
</tr>
<tr>
<td>1d Resource usage test(^b)</td>
<td>+</td>
</tr>
<tr>
<td>1e Back-to-back test between model and code, if applicable(^c)</td>
<td>+</td>
</tr>
</tbody>
</table>

\(^a\) This includes injection of arbitrary faults in order to test safety mechanisms (e.g. by corrupting values of variables)

\(^b\) Some aspects of the resource usage test can only be evaluated properly when the software unit tests are executed on the target hardware or if the emulator for the target processor supports resource usage tests.

\(^c\) This method requires a model that can simulate the functionality of the software units. Here, the model and code are stimulated in the same way and results compared with each other.
• SW integration & Testing
  • To integrate the software components and demonstrate that the software architectural design is correctly realised by the embedded software.
  • Integrated software tested against architectural design and interfaces between the software units and the software component.
  • Software integration test shall demonstrate
    – Compliance with the software architectural design
    – Compliance with the specification of the hardware-software interface
    – Correct implementation of the functionality
    – Robustness
    – Sufficiency of the resources to support the functionality.
Standards

- **Verification of SW safety requirements**
  - To demonstrate that the embedded software fulfills the software safety requirements and embedded software satisfies its requirements in the target environment.
  - The results of the verification of the software safety requirements shall be evaluated in accordance with:
    - Compliance with the expected results
    - Coverage of the software safety requirements
    - Pass or fail criteria

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a Hardware-in-the-loop</td>
<td>+</td>
</tr>
<tr>
<td>1b Electronic control unit network environments&lt;sup&gt;a&lt;/sup&gt;</td>
<td>++</td>
</tr>
<tr>
<td>1c Vehicles</td>
<td>++</td>
</tr>
</tbody>
</table>

<sup>a</sup> Examples are "lab-cars", "rest of the bus" simulations or test benches partially or fully integrating the electrical systems of a vehicle.
Software Design

- V model for software development
Software Design

• Definition:
  • *Design* is a problem-solving process whose objective is to find and describe a way:
    − To implement the system’s *functional requirements*...
    − While respecting the constraints imposed by the *quality, platform and process requirements*...
      − including the budget
    − And while adhering to general principles of *good quality*
Software Design

- Design as a series of decisions
  - A designer is faced with a series of design issues
    - These are sub-problems of the overall design problem
    - Each issue normally has several alternative solutions:
      - design options: software vs hardware
  - The designer makes a design decision to resolve each issue
    - Process involves choosing best option from alternatives
Software Design

• To make each design decision, the software engineer uses knowledge of
  • the requirements
  • the design as created so far
  • the technology available
  • software design principles and ‘best practices’
  • what has worked well in the past
Software Design

• System:
  • A logical entity, having a set of definable responsibilities or objectives, and consisting of hardware, software or both.
    – A system can have a specification which is then implemented by a collection of components.
    – A system continues to exist, even if its components are changed or replaced.
    – The goal of requirements analysis is to determine the responsibilities of a system.

• Subsystem:
  • A system that is part of a larger system, and which has a definite interface
Software Design

• **Component:**
  - Any piece of software or hardware that has a clear role
    - A component can be isolated, allowing you to replace it with a different component that has equivalent functionality
    - Many components are designed to be reusable
    - Conversely, others perform special-purpose functions

• **Module:**
  - A component that is defined at the PL level
    - For example: classes and packages are modules in Java

• **Function:**
  - Unit at programming level with specific behaviour
    - For example: methods in Java, functions in C
Software Design

• Top-down design
  • First design the very high level structure of the system.
  • Then gradually work down to detailed decisions about low-level constructs.
  • Finally arrive at detailed decisions such as:
    – the format of particular data items;
    – the individual algorithms that will be used.
Software Design

• **Bottom-up design**
  • Make decisions about reusable low-level utilities.
  • Then decide how these will be put together to create high-level constructs.

• **A mix of top-down and bottom-up approaches are normally used:**
  • Top-down design is almost always needed to give the system a good structure.
  • Bottom-up design is normally useful so that reusable components can be created.
Software Design

- **Architecture design:**
  - The division into subsystems and components,
    - How these will be connected
    - How they will interact
    - Their interfaces.
- **Class design:**
  - The various features of classes
- **User interface design**
- **Algorithm design:**
  - The design of computational mechanisms
- **Protocol design:**
  - The design of communications protocol
Software Design

- System modeling
Software Design

• Model Based Systems Engineering benefits
  • Shared understanding of system requirements and design
    – Validation of requirements
    – Common basis for analysis and design
    – Facilitates identification of risks

• Assists in managing complex system development
  – Separation of concerns via multiple views of integrated model
  – Supports traceability through hierarchical system models
  – Facilitates impact analysis of requirements and design changes
  – Supports incremental development & evolutionary acquisition
Software Design

• **Model Based Systems Engineering Benefits**
  • Improved design quality
    – Reduced errors and ambiguity
    – More complete representation
  • Supports early and on-going verification & validation to reduce risk
  • Provides value through life cycle (e.g., training)
  • Enhances knowledge capture
Software Design

What is SysML?

- A graphical modeling language in response to the UML for Systems Engineering developed by the a.o. OMG
  - a UML Profile that represents a subset of UML 2 with extensions
- Supports the specification, analysis, design, verification, and validation of systems that include hardware, software, data, personnel, procedures, and facilities
- Supports model and data interchange via XML Metadata Interchange (XMI)

SysML is an enabler for Model Driven SE
What is SysML?

- Is a visual modeling language that provides
  - Semantics = meaning
  - Notation = representation of meaning

- Is not a methodology or a tool
  - SysML is methodology and tool independent
Software Design

- Reuse of UML 2.0 for SysML
Software Design

- **SysML Taxonomy of Diagrams**

![Diagram](image)

[1] Modified UML Class Diagram
Software Design

- Four pillars of SysML

1. Structure
2. Behavior
3. Requirements
4. Parametrics
V-model for system of systems