Curriculum for

Certified Professional for Software Architecture (CPSA)®
Advanced Level

Module
EMBEDDED

Dependable Embedded Systems

2021.1-EN-20211124

Table of Contents

List of Learning Goals

- LG 5-4: Evolution of Embedded Systems
- LG 5-5: Tools for Supporting Variability and Adaptability

Introduction: General information about the iSAQB Advanced Level

What is taught in an Advanced Level module?

- The iSAQB Advanced Level offers modular training in three areas of competence with flexibly designable training paths. It takes individual inclinations and priorities into account.
- The certification is done as an assignment. The assessment and oral exam is conducted by experts appointed by the iSAQB.

What can Advanced Level (CPSA-A) graduates do?

CPSA-A graduates can:

- Independently and methodically design medium to large IT systems
- In IT systems of medium to high criticality, assume technical and content-related responsibility
- Conceptualize, design, and document actions to achieve quality requirements and support development teams in the implementation of these actions
- Control and execute architecture-relevant communication in medium to large development teams

Requirements for CPSA-A certification

- Successful training and certification as a Certified Professional for Software Architecture, Foundation Level® (CPSA-F)
- At least three years of full-time professional experience in the IT sector; collaboration on the design and development of at least two different IT systems
  - Exceptions are allowed on application (e.g., collaboration on open source projects)
- Training and further education within the scope of iSAQB Advanced Level training courses with a minimum of 70 credit points from at least three different areas of competence
  - existing certifications (for example: Sun/Oracle Java architect, Microsoft CSA) can be credited upon application
- Successful completion of the CPSA-A certification exam

Essentials

What does the module “EMBEDDED” convey?

Embedded systems are computer systems that are integrated in a larger cyber-physical system, via which they interact with the physical environment through sensors and actuators. As the requirements concerning the functionality and the quality characteristics of these systems are increasing, the size and the complexity of embedded systems software grows considerably.

In domains such as automotive systems, automation, rail, or medical devices for treatment or diagnostics, particular quality characteristics need to be achieved:

- systems must be highly reliable and are often safety-critical
- systems must react to events in their environment in a timely, predictable way, leading to hard real-time requirements
- systems need to be adaptable to quickly react to changing market requirements

The module “EMBEDDED” conveys a methodical approach to architecture design for dependable embedded systems. It shows how software architecture interacts with the overall systems architecture, and how the requirements with regards to safety, dependability, real-time and adaptability are addressed in a systematic way. In addition to patterns and solution concepts for designing appropriate software architectures, the module also addresses the analysis of software architectures with regard to the required quality characteristics.

Curriculum Structure and Recommended Durations

Content	Recommended minimum duration (minutes)
1. Systems Development for Embedded Systems	120
2. Software Development for Embedded Systems	180
3. Dependability and Functional Safety	330
4. Real-Time and Concurrency	360
5. Adaptability	90
Total	1080 (18h)

Duration, Teaching Method and Further Details

The times stated below are recommendations. The duration of a training course on the EMBEDDED module should be at least 3 days, but may be longer. Providers may differ in terms of duration, teaching method, type and structure of the exercises and the detailed course structure. In particular, the curriculum provides no specifications on the nature of the examples and exercises.

Licensed training courses for the EMBEDDED module contribute the following credit points towards admission to the final Advanced Level certification exam:

Methodical Competence:	10 Points
Technical Competence:	20 Points

Communicative Competence:	0 Points

Prerequisites

Participants **should** have the following prerequisite knowledge:

- Foundations of software architecture, as they are taught in CPSA-F accredited trainings
- Practical experience with the implementation and design of embedded systems and the typical challenges in developing embedded systems

Knowledge in the following areas may be **helpful** for understanding some concepts:

- Practical experience with developing safety-relevant systems and/or real-time systems

Structure of the Curriculum

The individual sections of the curriculum are described according to the following structure:

- **Terms/principles**: Essential core terms of this topic.
- **Teaching/practice time**: Defines the minimum amount of teaching and practice time that must be spent on this topic or its practice in an accredited training course.
- **Learning goals**: Describes the content to be conveyed including its core terms and principles.

This section therefore also outlines the skills to be acquired in corresponding training courses.

Supplementary Information, Terms, Translations

To the extent necessary for understanding the curriculum, we have added definitions of technical terms to the iSAQB glossary and complemented them by references to (translated) literature.

1. Systems Development for Embedded Systems

Duration: 120 min.

1.1. Terms and Principles

Systems architecture, functional architecture, technical systems architecture, event chain, systems development, software development, hardware development

1.2. Learning Goals

LG 1-1: Systematic Approach to Systems Development

Participants understand the systematic approach to systems development:

- Participants understand the cross-cutting nature of aspects like time behavior, dependability and safety with regards to system-, software- and hardware-development, and the necessity to address these with an integrated systems development approach.
- Participants understand that systems development follows an iterative approach that includes both top-down activities like working with models, estimations and simulations, and bottom-up activities like tests and measurements on real systems to validate the estimates and simulations.
- Participants understand that modeling and model-based analysis enable the evaluation of different architectures in a cost-efficient way. Model-based approaches therefore enable a more thorough exploration of the design space.
- Participants understand the costs and benefits of developing and maintaining multiple architectural abstraction levels, such as functional and technical systems architectures: A functional architecture enables reasoning about a system in a way that is independent of a specific technical realization and allows for different technical variants. These benefits need to be weighed against the costs for developing and maintaining several architectural models.
- Participants know methods and frameworks for model-based systems development, such as Harmony [Douglass 2015], SYSMOD [Weilkiens 2008].
- Participants understand that development approaches for embedded systems are often subject to domain specific standards.

LG 1-2: Modeling and Analysis of Functional Architectures

Participants can apply a systematic approach to modeling and analysis of functional architectures:

- Model the system context with external systems, physical entities, users of the system and the logical events that occur between the system and its environment.
- Hierarchically decompose the system from a purely functional perspective, identifying function blocks and their interactions via events or information flows.
- Define event chains based on the functional architecture. An event chain traces the end-to-end behavior from an external triggering event to the desired system reaction through the involved function blocks, events and information flows.

Participants understand that a functional architecture enables reasoning about system quality characteristics at an abstract level, e.g., regarding time behavior, dependability, or flexibility.

Participants know examples of how functional architectures can be modeled (e.g., with SysML or FAS [Weilkiens+2015]).

LG 1-3: Modeling and Analysis of Technical System Architectures

Participants understand the systematic approach to modeling and analysis of technical systems architectures:

- Analyze the impact factors, like quality requirements, organizational constraints, technological constraints, as defined in the CPSA Foundation Level curriculum. For the technical system architecture of an embedded system, impact factors are often related to dependability and safety requirements, performance and real-time requirements, and flexibility requirements to support different product variants and product lines. Examples for further considerations are physical proximity to sensors and actuators, limited construction space, connectivity requirements, hardware cost, and development schedules.
- Based on the impact factors, identify architectural issues and develop solution strategies for topics such as error handling, resource efficiency, real-time architectural design and variability.
- Evaluate alternative technical system architectures, and select a technical system architecture for further refinement.
- Hierarchically decompose the system into technical building blocks, and establish a mapping to the elements of the functional architecture, if applicable.
- Map event chains to the technical architecture, if applicable: For example, allocate a function to determine a measurement to a physical sensor, or allocate an information flow to a bus.
- Evaluate the defined technical system architecture with regard to the impact factors.

Participants know examples of how technical system architectures can be modeled, for example with SysML.

Participants understand that the design of a technical systems architecture is an iterative process that requires feedback from software development and hardware development.

Participants know typical decisions that need to be made in the design of the technical systems architecture:

- How a function block is implemented, for example in software, with an ASIC, an FPGA, or a combination of these.
- Deciding the number of control units and their hardware characteristics (e.g., single core, homogeneous multi core, heterogeneous multi core, amount of RAM, ROM, NVRAM, peripherals like sensors and actuators, custom hardware).
- Allocation of function blocks to one or more control units, considering for example performance and isolation requirements.
- Decide on the communication paradigm (e.g., time triggered vs. event triggered).
- Decide on network topologies (e.g., bus vs. star) and network technologies, both wired (e.g., Ethernet, FlexRay, CAN), and wireless (e.g., Bluetooth, WiFi, cellular).

Participants understand the challenges of distributed systems regarding latency, throughput, lack of a global clock, and the impact on dependability.

1.3. References

[Weilkiens 2008], [Douglass 2015]

2. Software Development for Embedded Systems

Duration: 180 min.

2.1. Terms and Principles

Software architecture, UML, domain-specific languages, code generation

2.2. Learning Goals

LG 2-1: Modeling Software for Embedded Systems

Participants know different approaches for modeling software for embedded systems. They understand the basic concepts, strengths and weaknesses of these approaches:

- UML as a general-purpose modeling language
- UML profiles to adapt UML to a specific domain (e.g., MARTE as a UML profile for modeling real-time systems).
- Graphical and textual domain-specific languages (e.g., AADL)
- State machines for modeling reactive systems.
- Modeling data- and signal-flows (e.g., function-block diagrams)

Participants understand how models can be used for analyzing the software, for example regarding failures, schedulability, error propagation, and latency.

Participants are able to select a suitable modeling approach based on the requirements and boundary conditions of the system under development.

LG 2-2: Implementing Software for Embedded Systems

Participants understand how programming languages influence quality characteristics and cross-cutting concepts:

- Impact of the programming language on analyzability (e.g., with static and dynamic analysis tools).
- Impact of the programming language abstractions and concepts for concurrency, memory management and error handling on cross-cutting concepts and dependability.

Participants know different programming language paradigms (e.g., procedural, object-oriented, functional). Participants know examples for programming languages supporting these paradigms that are suitable for embedded systems, and understand their strengths and weaknesses.

LG 2-3: Memory Management

Participants understand how memory management strategies and memory-management-related capabilities of the programming language affect memory safety, flexibility, predictability, performance and programming complexity:

- Understand the trade-offs in choosing dynamic or static allocation of memory.

- Know managed approaches to dynamically allocated memory (e.g., garbage collection, automated reference counting, ownership / non-lexical scoping) and their trade-offs.
- Know the impact of system-architectural decisions on memory management (e.g., amount of memory, availability of memory management hardware like an MMU or MPU, CPU caches), know software-based approaches to memory safety.

LG 2-4: Error Handling

Participants know different language features and patterns for error handling and their strengths and weaknesses, such as return values, global error status variables, global error handlers, exceptions, monadic error handling (e.g., "Result" types, "Either" types).

LG 2-5: Model-Based Development of Embedded Systems

Participants know how models can be used to generate artifacts for the realization, and understand the trade-offs:

- Different kinds of models and model elements can be used: structural models (e.g., UML component models, AADL) and behavioral models (e.g., state charts, activity diagrams).
- Different kinds of artifacts can be generated, e.g. implementation code, test cases, configurations, or analysis models.
- Using a model-based approach can have the following benefits:
  - Reduced effort since these artifacts do not have to be created manually.
  - Increased quality since manual errors can be eliminated.
  - Can support adaptability in a resource-efficient way by resolving variability during code generation.
- Using a model-based approach has the following potential costs:
  - Creating a sufficiently detailed model requires significant effort, which needs to be weighed against the potential savings.
  - Generated code often has increased resource demands.
  - Potential vendor lock-in, specifically with commercial tools.
  - The generated code may be of insufficient quality or not suitable for the target platform.
  - Using a model-based approach in a safety-critical environment may lead to significant tool qualification efforts.

LG 2-6: Verification of Embedded Software

Participants can explain and give examples for different error categories (e.g. memory-management errors, synchronization errors, type defects, timing errors, domain errors).

Participants understand how analysis techniques can be used to detect errors, and know the limitations of these techniques:

- Participants can define static analysis, understand the difference between sound and unsound analysis, and know the application areas of static analysis tools. Participants know examples for static analysis tools.

- Participants can define dynamic analysis and know the application areas of dynamic analysis tools. Participants know examples for dynamic analysis tools.

LG 2-7: Design Paradigms

Participants know design principles and paradigms for embedded software, such as design by contract, the robustness principle, or defensive programming.

2.3. References

[Koopman 2021], [Weilkiens 2008], [Andrews+2008], [Samek 2008], [Selic+1994], [Selic+2014], [Feiler+2012], [Kordon 2013]

3. Dependability and Functional Safety

Duration: 330 min.

3.1. Terms and Principles

Dependability, safety, availability, reliability, fault tolerance, security, fault, error, failure, fault prevention, error detection, error recovery, error mitigation, redundancy, replication, fail safe vs fail operational, fail silent, fail stop, fail consistent, freedom from interference.

3.2. Learning Goals

LG 3-1: Fundamental Terms of Dependable and Safety-Related Systems

Understand the following terms and how these are related to each other: dependability, safety, availability, reliability, fault tolerance, security, fault, error, failure, redundancy, replication, freedom from interference, spatial interference, temporal interference, communication interference, fail operational, fail silent, fail stop, fail consistent.

LG 3-2: Developing Safety-Related Systems

Participants understand the fundamental approach for developing a safety-related system:

- Perform a hazard and risk analysis.
- Decide the kind of risk treatment (mitigate, eliminate, transfer, or accept).
- Define and allocate safety requirements.
- Implement safety requirements on system, hardware, and software level.
- Analyze the system's residual risk to verify that an acceptable level of risk has been achieved (e.g., with FMEA and FTA).

Participants understand that functional safety must be considered on the system level, and impacts systems architecture, software architecture, and hardware architecture.

Participants understand the influence of standards in the development of safety-related systems.

Participants understand the difference in approach to safety-related system development of standards such as the IEC 61508, ISO 26262, ISO 62304 and other domain-specific safety standards.

LG 3-3: Safety Classification of Requirements and Building Blocks

Participants understand the process of safety classification:

- Based on the risk analysis, risks are classified regarding their severity (e.g., a Safety Integrity Level in IEC 61508).
- The classification of a risk is propagated to safety goals and/or safety requirements that are defined to control risk, and further propagated to derived safety requirements and to the building blocks to which a safety requirement is allocated.
- Spatial and temporal coupling between building blocks can lead to interference between building blocks and to a propagation of the associated risk.

- By decomposing a building block, its associated risk can be distributed over and encapsulated in multiple building blocks. This can lead to a lower risk classification for the individual building blocks. To do so, freedom from interference must be ensured between these building blocks.
- The details of safety classification and decomposition are subject to specific norms and standards.

LG 3-4: Tight Relationship between System, Software and Hardware Development

Participants understand that systems development, software development and hardware development are closely intertwined when developing dependable and safety-related systems. For example, software plays a critical role in diagnosing hardware failures, and hardware plays a critical role in ensuring freedom from interference between software units.

LG 3-5: Developing Fault-Tolerant Systems

Participants understand fundamental concepts and activities for developing fault-tolerant systems and how they interact with each other:

- Fault prevention is concerned with reducing the number of faults that are present in a system (e.g., by applying design methodologies or performing reviews).
- It is not possible to completely prevent the introduction of faults, therefore a fault-tolerant system needs to be able to handle unanticipated errors.
- A fault-tolerant system needs to detect errors and then perform error recovery or error mitigation to prevent error propagation. The propagation of undetected or unhandled errors may lead to system failure.
- The externally observable behavior of component in the event of an error can be classified as fail stop, fail operational, fail silent, or fail consistent behavior.
- Redundancy can help to build fault-tolerant systems. Homogeneous and heterogeneous approaches exist. Depending on the types of errors to handle, a suitable redundancy approach or a combination of approaches has to be selected.

LG 3-6: Architectural Patterns and Techniques

Participants can apply architectural patterns and techniques for developing fault-tolerant and safety-related systems:

- Participants can distinguish between structural, functional, informational and temporal redundancy.
- Participants know replication as a means to achieve structural redundancy. Replication is applied to an architectural building block. Therefore, the sphere of replication has to be aligned with architectural building blocks. Strong cohesion and loose coupling principles have to be taken into consideration for that. The sphere of replication is often aligned with isolation zones with respect to memory regions (i.e., address spaces) and control flows (i.e., threads).
- Participants can select and apply architectural patterns and techniques to achieve fault tolerance and know advantages, disadvantages and limitations of these patterns.
- Participants know examples of architectural patterns and techniques, such as Units of Mitigation, Escalation, Fault Observer, Process-Level Redundancy, Ground State, Cold/Warm/Hot Standby, Triple-Modular Redundancy, Duo-Duplex Redundancy, Centralized vs. Distributed Voting, N-Version Programming, Recovery Blocks, Correcting Audits, Software Update, software- and hardware-based memory protection techniques.

- Participants can determine the increase in reliability gained by introducing redundancy. They are aware of the importance of replicate reliability.
- Participants understand that even after introducing redundancy, single points of failure may remain. They know techniques for hardening these, such as control-flow protection.
- Participants understand what common-mode failures are and know their sources.
- Participants know techniques for testing hardened systems, such as fault injection.

LG 3-7: Approaches to Fault Prevention

Participants understand that suitable processes and methods for design, implementation, verification, and testing are crucial to prevent faults in the system.

Participants know that in a safety-related system, norms and standards give recommendations for processes and methods.

LG 3-8: Patterns for Detecting Errors and Failures

Participants an select and apply patterns and techniques such as Checksum, Parameter Checking, Heartbeat, Leaky Bucket Counter, System Monitor, Watchdog, Plausibility Checks, Control-Flow Monitoring.

Participants can select and apply detection patterns that are used with redundancy such as Voting and Lockstep.

Participants understand the importance of timely error detection.

LG 3-9: Patterns for Error Recovery

Participants can select and apply patterns for error recovery such as Data Reset, Error Handler, Failover, Retries, Limiting Retries, Quarantine, Deactivation, Restart, Return to Reference Point, Rollback, Roll-Forward.

LG 3-10: Patterns for Error Mitigation

Participants can select and apply patterns for error mitigation such as Error-Correcting Codes and Marked Data.

3.3. References

[Hanmer 2007], [NASA 2004], [Smith+2016], [IEC 61508], [ISO 26262:2018]

4. Real-Time and Concurrency

Duration: 360 min.

4.1. Terms and Principles

Hard and firm vs. soft real-time requirements, cyclic executive and cyclic schedules, interrupts, real-time operating systems, jobs, tasks, threads, processes, static vs dynamic scheduling, time-triggered vs event-triggered real-time systems, priority inversion, priority inheritance, priority ceiling, schedulability analysis, worst-case execution time (WCET), worst observed execution time (WOET), execution time estimation, CPU load estimation, real-time architectural design simulation and verification, concurrent resource access and coordination techniques, black-box simulation vs. white-box simulation.

4.2. Learning Goals

LG 4-1: Basic Terms and Properties of Real-Time Systems

Participants understand that real-time requirements of embedded systems are the result of their interaction with the environment.

Participants can explain the difference between timeliness and speed.

Participants can explain the difference between hard/firm real-time requirements and soft real-time requirements.

Participants understand that all actions along the event chain from the occurrence of a relevant event to the system reaction must be examined when implementing real-time requirements (e.g., sensor data read-out, operating-system context switch, sensor-data processing, actuator control). For hard/firm real-time requirements, a deadline must be defined for each of these actions. Missing that deadline is not tolerable.

Participants understand the different types of events (periodic, sporadic, aperiodic).

Participants understand that real-time architectural design should be performed early on in the project.

Participants know that real-time architectural design may also encompass choosing and configuring an operating system, which poses a technological impact factor.

LG 4-2: Developing Real-Time Systems

Participants understand the fundamental approach to developing real-time systems:

- Identify and specify external real-time requirements.
- Choose a time-triggered or event-triggered approach to real-time architectural design.
- Identify logical tasks and their properties.
- Identify shared resources required by logical tasks and their properties.
- Map logical tasks to jobs.
- Decide on technical solutions (e.g., RTOS tasks), and allocate jobs to the technical solution elements.
- Implement the real-time architectural design.

- Continuously analyze and verify the real-time architectural design and the implementation to iteratively refine the real-time architectural design.

Participants know how real-time system design is integrated into the overall development life cycle.

Participants understand that extending functions or adding new functions at a later time can have a large impact on time behavior and schedulability. The degree to which such extensions must be supported is an important impact factor to real-time architectural design. They understand that real-time affects the entire life cycle of the system and influences many disciplines and activities (e.g., requirements elicitation, systems architecture, hardware architecture, software architecture, software implementation, test and verification, deployment).

Participants understand that real-time is a cross-cutting concern.

LG 4-3: Real-Time Requirements

Participants are able to specify and model real-time requirements. They know different approaches to specifying real-time requirements (e.g., specifying real-time requirements in UML, SysML, or AADL) and can select a suitable approach for a particular system.

LG 4-4: Time-Triggered vs. Event-Triggered Approaches

Participants can describe the time-triggered and event-triggered approaches to real-time architectural design. They understand the trade-offs of both approaches regarding determinism and flexibility.

Participants understand that the decision for an approach has a profound, cross-cutting impact on the architecture and the implementation and is therefore hard to change.

LG 4-5: Technical Solutions to Scheduling and Concurrency

Participants can describe technical solutions like cyclic executive, RTOS-based event-triggered approach, and application-level interrupt handling.

Participants know criteria for deciding between different technical solutions. They are able to select a technical solution for a specific system and reasonably justify their decision.

Participants can describe how different technical solutions can be combined (e.g., cyclic executive and interrupts).

Participants understand approaches to handling non-periodic events (interrupt-driven execution, background execution, slack-stealing). They know further approaches (periodic, deferrable or sporadic server).

Cyclic schedules and frame-based scheduling:

- Participants can describe the benefits and drawbacks of cyclic scheduling.
- Participants can analyze the real-time characteristics of a solution based on a cyclic executive.
- Participant can design cyclic schedules for simple real-time problems.

RTOS-based event-triggered approach:

- Participants can describe the differences between static and dynamic scheduling methods.

- Participants can describe the advantages and disadvantages of using preemption.
- Participants understand rate-monotonic, deadline-monotonic and earliest deadline first scheduling.
- Participants can select an appropriate scheduling approach according to the real-time requirements.
- Participants can identify jobs with similar properties and allocate them to the same elements of the technical solution (e.g., processes or threads).
- Participants can define the RTOS-tasks and their characteristics.
- Participants can describe important timing properties of RTOSes and hardware (e.g., interrupt latency, scheduling latency, dispatch latency)

Application-level interrupt handling:

- Participants understand the strengths and weaknesses of using interrupts at the application level.
- Participants can define a concept for using interrupts at the application level: prioritizing interrupts, rules for blocking interrupts, using the prologue/epilogue model
- Participants understand that the concepts for using interrupts at the application level depend on the processor architecture (e.g., number of priorities) and the RTOS (e.g., support for prologue/epilogue model).

LG 4-6: Concurrent Access to Shared Resources

Participants understand the potential problems caused by concurrent access to shared resources (e.g., blocking time). They understand that whether these problems can actually occur depends strongly on the chosen technical solution to concurrency (e.g., time-triggered vs. event-triggered approach).

Participants understand solution approaches for interactions between jobs and their trade-offs (e.g., message passing, shared data).

Participants know strategies for preventing typical concurrency problems (e.g., critical sections, non-preemptive critical sections protocol, semaphores, synchronization objects and mutexes, atomic actions, non-blocking synchronization). They can select appropriate strategies for a specific system.

Participants know how to optimize critical sections (e.g., by assigning program elements which share the same software resources to the same RTOS task).

Participants understand how deadlocks occur and how they can be avoided.

Participants understand priority inversion and know solution approaches (priority ceiling, priority inheritance).

LG 4-7: Impact of the Operating System on Real-Time Characteristics

Participants can explain the characteristics of a real-time operating system. They understand that real-time operating systems and general-purpose operating systems serve different purposes.

Participants know a minimum of two examples of real-time operating systems and their characteristics.

Participants can refine the real-time architectural design in the context of the selected operating system (e.g., refine tasks priorities and assign program elements to task entities).

LG 4-8: Real-Time Analysis

Participants know different approaches how the real-time architectural design can be evaluated, such as design reviews, performing schedulability analysis, or using tools for scheduling simulation and verification.

Schedulability analysis:

- Participants understand how to gather data on which a schedulability analysis can be based (e.g., past experience, measurements, simulation, mathematical methods, static analysis).
- Participants know simulation and analytical methods as approaches to gain confidence that the real-time requirements can be met.
- Participants understand that the schedulability of a set of tasks cannot be shown reliably by individual measurements (e.g., due to blocking time).
- Participants know approaches to schedulability analysis for rate-monotonic, deadline-monotonic, or earliest-deadline-first scheduling.
- Participants can apply rate-monotonic analysis to perform a dependable schedulability analysis.
- Participants understand the limitations of analytical approaches regarding complex real-time systems, due to dependencies among jobs.
- Participants understand simulation as an approach for complex real-time systems.

Worst-case execution time (WCET) analysis:

- Participants understand that determining the maximum execution time of a job is crucial for real-time analysis. Soundness and tightness are important quality characteristics of WCET analysis.
- Participants understand that the precision of WCET analysis is limited. The precision is influenced by hardware and software complexity (e.g., impact of caches, pipelines, shortest vs. longest program path, memory management, dynamic dispatch, approach to error handling).
- Participants can explain advantages, disadvantages and limitations of static analysis, dynamic analysis and hybrid approaches to WCET analysis.
- Participants can estimate the overall CPU load to support schedulability analyses such as rate-monotonic analysis.

Shared resource analysis:

- Participants understand the difference between WCET and WCRT.
- Participants understand how shared resources and the resource-access protocol affect the WCRT. They understand that therefore shared-resource analysis needs to be performed as part of real-time analysis.

LG 4-9: Tools for Real-Time Architectural Design and Analysis

Participants understand that tools for specification, design and analysis of real-time systems are needed for complex embedded systems with many external real-time requirements.

Participants know application areas of tools for real-time architectural design and analysis, such as modeling the real-time architectural design, static WCET analysis, analytical schedulability analysis, white-box vs. black-box simulation of real-time systems. Participants know examples of specific tools and their

application areas.

LG 4-10: Relationship to Distributed and Multi-Core Systems Architectures

Participants understand the challenges introduced when building distributed or multi-core real-time systems (e.g., proper methods of synchronization, global scheduling, global time base, latencies imposed by communication).

Participants know different approaches to scheduling (partitioned, clustered, global scheduling) and migration in multicore systems (e.g., offline vs. online migration).

4.3. References

[Kopetz 2011], [Liu 2000], [Baruah+2015], [Stallings 2014], [Douglass 2014]

5. Adaptability

Duration: 90 min.

5.1. Terms and Principles

Variants, feature models, platform development, product lines, product platforms, variation point, adaptability, variability, variability in space, variability in time, binding time.

5.2. Learning Goals

LG 5-1: Foundations of Adaptability and Variability

Participants understand the motivation and need for adaptability (supporting the evolution over time, variability in time) and variability (supporting multiple product variants, variability in space). They understand how the need for adaptability and variability can be determined and specified.

Participants know possible binding times when variability can be resolved (before runtime, at startup/load time, at runtime).

Participants know trade-offs and drawbacks of increased adaptability and variability as well as competing quality goals (e.g., testability, performance, safety, security, and backwards compatibility) and complementing quality goals (e.g., maintainability, reusability).

Participants know high level approaches to adaptability and variability such as the 150%-model, 80%-model, product platforms or product lines.

LG 5-2: Modeling Variability in the Architecture

Participants understand that variability can occur at the functional level (e.g., product variants differ in the provided functions) and at the technical level (e.g., alternative technical solutions exist for the same functionality).

Participants understand how the separation between functional architectures and technical architectures can support adaptability and variability by allowing alternative technical architectures for the same functional architecture.

Participants know how to model variation points in requirements, functional architectures, and technical architectures.

LG 5-3: Supporting Adaptability and Variability

Participants know how adaptability and variability are supported by:

- principles such as modularity
- architecture and design patterns such as plugin, adapter, facade
- frameworks and libraries
- architecture decisions such as deployment decisions (e.g., edge vs. cloud computing)

Participants can apply these solutions to implement variation points in the architecture.

LG 5-4: Evolution of Embedded Systems

Participants understand that adaptability in embedded software is driven both by requirement changes and by evolution of the systems architecture. This evolution needs to be addressed in the software architecture and in the development process.

Participants understand the trade-offs of using legacy code and how its usage can restrict adaptability and variability.

Participants know how to maintain adaptability and deal with new adaptability requirements (e.g., by refactoring).

Participants know approaches how to handle system and software updates, as well as trade-offs of updates.

LG 5-5: Tools for Supporting Variability and Adaptability

Participants know common tools to model and manage variability.

Participants understand how code generation can support adaptability and variability and know common tools for code generation.

5.3. References

[Anastasopoulos+ 2001], [Clements+2002], [Kang+1990]

References

This section contains references that are cited in the curriculum.

A

- [Anastasopoulos+ 2001] Michalis Anastasopoulos, Cristina Gacek: Implementing Product Line Variabilities, in: ACM SIGSOFT Software Engineering Notes, May 2001
- [Andrews+2008] Jason Andrews, Kamal Hyder, Bob Perrin, Colin Walls, Rick Gentile, Keith E. Curtis, David J. Katz, Jack Ganssle, Jean J. Labrosse, Robert Oshana: Embedded Software, Elsevier, 2008

B

- [Baruah+2015] Sanjoy K. Baruah, Marko Bertogna, Giorgio C. Buttazzo: Multiprocessor Scheduling for Real-Time Systems, Springer 2015

C

- [Clements+2002] Paul Clements, Linda M. Northrop: Software product lines - practices and patterns, Addison Wesley 2002

D

- [Douglass 2014] Bruce Powel Douglass: Real Time UML Workshop for Embedded Systems, Newnes 2014
- [Douglass 2015] Bruce Powel Douglass: Agile Systems Engineering, Morgan Kaufman 2015

F

- [Feiler+ 2012] Peter H. Feiler, David P. Gluch: Model-Based Engineering with AADL, Addison-Wesley 2012

H

- [Hanmer 2007] Robert Hanmer: Patterns for Fault Tolerant Software, Wiley 2007

K

- [Koopman 2021] Philip Koopman: Better Embedded System Software, Independently published 2021
- [Kopetz 2011] Hermann Kopetz: Real-Time Systems - Design Principles for Distributed Embedded Applications, Springer 2011
- [Kordon 2013] Fabrice Kordon (Editor): Embedded Systems - Analysis and Modeling with SysML, UML and AADL, Wiley 2013

L

- [Kang+1990] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, A. Spencer Peterson: Feature-Oriented Domain Analysis (FODA) Feasibility Study, Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, 1990

I

- [IEC 61508] IEC 61508:2010 Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, The International Electrotechnical Commission 2010
- [ISO 26262:2018] ISO 26262-1:2018 Road vehicles — Functional safety, International Organization for Standardization, 2018

L

- [Liu 2000] Jane W.-S. Liu: Real-time systems. Prentice Hall 2000

N

- [NASA 2004] NASA Software Safety Guidebook, NASA-GB-8719.13, 2004

S

- [Samek 2008] Miro Samek: Practical Statecharts in C/C++, CRC Press 2008
- [Selic+1994] Bran Selic, Garth Gullekson, Paul T. Ward: Real-Time Object-Oriented Modeling, Wiley 1994
- [Selic+2014] Bran Selic, Sébastien Gérard: Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE, Morgan Kaufman 2013
- [Smith+2016] David Smith, Kenneth Simpson: Safety Critical Systems Handbook, Butterworth Heinemann 2016
- [Stallings 2014] Stallings, W: Operating Systems: Internals and Design Principles, Prentice Hall

W

- [Weilkiens 2008] Tim Weilkiens: Systems Engineering with SysML/UML: Modeling, Analysis, Design, Morgan Kaufman 2008
- [Weilkiens+2015] Tim Weilkiens, Jesko G. Lamm, Stephan Roth, Markus Walker: Model-Based System Architecture, Wiley 2015