Curriculum für

CPSA Certified Professional for Software Architecture®

– Advanced Level –

Modul:
EMBEDDED

Sicherheitskritische Eingebettete Systeme

Version 1.3 (Februar 2015)

Inhaltsverzeichnis

0 Einleitung: Allgemeines zum iSAQB-Advanced-Level

0.1 Was vermittelt ein Advanced-Level-Modul?

- Der iSAQB-Advanced-Level bietet eine modulare Ausbildung in drei Kompetenzbereichen mit flexibel gestaltbaren Ausbildungswegen. Er berücksichtigt individuelle Neigungen und Schwerpunkte.
- Die Zertifizierung erfolgt als Hausarbeit. Die Bewertung und mündliche Prüfung wird durch vom iSAQB benannte Experten vorgenommen.

0.2 Was können Absolventen des Advanced Level (CPSA-A)?

CPSA-A-Absolventen können:

- Eigenständig und methodisch fundiert mittlere bis große IT-Systeme entwerfen.
- In IT-Systemen mittlerer bis hoher Kritikalität technische und inhaltliche Verantwortung übernehmen.
- Konzeption, Entwurf und Dokumentation von Maßnahmen zur Erreichung nicht-funktionaler Anforderungen. Begleitung des Entwicklungsteams bei der Umsetzung dieser Maßnahmen durchführen.
- Architekturrelevante Kommunikation in mittleren bis großen Entwicklungsteams steuern und durchführen.

0.3 Voraussetzungen zur CPSA-A-Zertifizierung

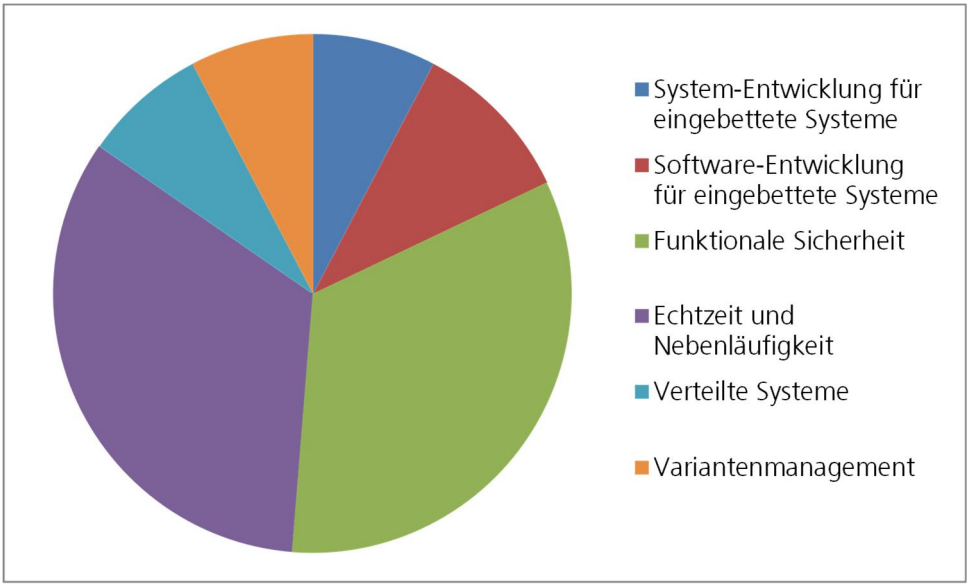
- Eine erfolgreiche Ausbildung und Zertifizierung zum CPSA-F (Certified Professional for Software Architecture, Foundation Level).
- Mindestens drei Jahre Vollzeit-Berufserfahrung in der IT-Branche, dabei Mitarbeit an Entwurf und Entwicklung von mindestens zwei unterschiedlichen IT-Systemen.
    - Ausnahmen auf Antrag zulässig (etwa: Mitarbeit in Open Source Projekten).
- Aus- und Weiterbildung im Rahmen von iSAQB-Advanced-Level-Schulungen im Umfang von mindestens 70 Credit-Points aus den drei unterschiedlichen Kompetenzbereichen (detailliert geregelt in Abschnitt 1.6).
    - Bestehende Zertifizierungen können ggfs. auf Antrag auf diese Credit Points angerechnet werden. Die Liste der aktuellen Zertifikate, für die Credit Points angerechnet werden, ist auf der iSAQB-Homepage zu finden.
    - Sonstige Aus- und Weiterbildungen können auf Antrag beim iSAQB ebenfalls anerkannt werden, sofern ein Bezug zu Software-Architektur vorliegt. Die Entscheidung hierüber trifft im Einzelfall die Arbeitsgruppe „Advanced Level“ des iSAQB.
- Erfolgreiche Bearbeitung der CPSA-A-Zertifizierungsprüfung.

1 Grundlegendes zum Modul Sicherheitskritische Eingebettete Systeme

1.1 Gliederung des Lehrplans für Sicherheitskritische Eingebettete Systeme und empfohlene zeitliche Aufteilung

- System-Entwicklung für eingebettete Systeme (mind. 1,5 h)
- Software-Entwicklung für eingebettete Systeme (mind. 2 h)
- Funktionale Sicherheit (mind. 6,5 h)
- Echtzeit und Nebenläufigkeit (mind. 6,5 h)
- Verteilte Systeme (mind. 1,5 h)
- Variantenmanagement (mind. 1,5h)

Die angegebenen Zeiten sind inklusive Übungen.



1.2 Dauer, Didaktik und weitere Details

Die genannten Zeiten sind Empfehlungen. Die Dauer einer Schulung zum Modul Sicherheitskritische Eingebettete Systeme sollte mindestens 3 Tage betragen, kann aber länger sein. Anbieter können sich durch Dauer, Didaktik, Art und Aufbau der Übungen sowie der detaillierten Kursgliederung voneinander unterscheiden. Insbesondere die Art der Beispiele und Übungen lässt der Lehrplan komplett offen.

1.3 Voraussetzungen für das Modul Sicherheitskritische Eingebettete Systeme

Teilnehmer **sollten** folgende Kenntnisse und/oder Erfahrung mitbringen:

- Grundlagen des Entwurfs, der Beschreibung und der Bewertung von Software-Architekturen wie sie im CPSA-F Foundation Level vermittelt werden.
- Erfahrungen in der Entwicklung von Software für eingebettete Systeme.

1.4 Gliederung des Lehrplans für Sicherheitskritische Eingebettete Systeme

Die einzelnen Abschnitte des Lehrplans sind gemäß folgender Gliederung beschrieben:

- **Begriffe/Konzepte**: Wesentliche Kernbegriffe dieses Themas.
- **Unterrichts-/Übungszeit**: Legt die Unterrichts- und Übungszeit fest, die für dieses Thema bzw. dessen Übung in einer akkreditierten Schulung mindestens aufgewendet werden muss.

- **Lernziele**: Beschreibt die zu vermittelnden Inhalte inklusive ihrer Kernbegriffe und -konzepte.
- Dieser Abschnitt skizziert damit auch die zu erwerbenden Kenntnisse in entsprechenden Schulungen. Die Lernziele werden differenziert in folgende Kategorien bzw. Unterkapitel:
- Was sollen die Teilnehmer **können**? Diese Inhalte sollen die Teilnehmer nach der Schulung selbständig anwenden können. Innerhalb der Schulung werden diese Inhalte durch Übungen abgedeckt und sind Bestandteil der Abschlussprüfung des iSAQB Advanced Levels.
- Was sollen die Teilnehmer **verstehen**? Diese Inhalte können geprüft werden.
- Was sollen die Teilnehmer **kennen**? Diese Inhalte (Begriffe, Konzepte, Methoden, Praktiken oder Ähnliches) können das Verständnis unterstützen oder das Thema motivieren. Diese Inhalte sind nicht Bestandteil der Prüfungen, werden in Schulungen thematisiert, aber nicht notwendigerweise ausführlich unterrichtet.

1.5 Ergänzende Informationen, Begriffe, Übersetzungen

Soweit für das Verständnis des Lehrplans erforderlich, haben wir Fachbegriffe ins iSAQB-Glossar aufgenommen, definiert und bei Bedarf durch die Übersetzungen der Originalliteratur ergänzt.

1.6 Credit Points für diese Schulung

Vom iSAQB lizensierte Schulungen gemäß diesem Lehrplan tragen zur Zulassung zur abschliessenden Advanced-Level-Zertifizierungsprüfung folgende Punkte (Credit Points) bei:

Methodische Kompetenz:	10 Punkte
Technische Kompetenz:	20 Punkte

2 Einführung in das iSAQB-Zertfizierungsprogramm

Dauer: 15 Min (optional)	Übungszeit: keine

Dieser Abschnitt ist nicht prüfungsrelevant. Falls Teilnehmer bereits CPSA-F zertifiziert sind, kann dieser Abschnitt entfallen.

2.1 Begriffe und Konzepte

iSAQB, Advanced-Level-Zertifizierung und Voraussetzung dazu.

2.2 Lernziele

Die Teilnehmer lernen den Kontext des iSAQB-Zertifizierungsprogrammes und der zugehörigen Prüfungen beziehungsweise Prüfungsmodalitäten kennen.

2.2.1 Was sollen die Teilnehmer kennen?

- iSAQB als Verein
- Advanced Level in Abgrenzung zu anderen Level
- Randbedingungen und Vorgehen beim iSAQB-Zerifizierungsprogramm

3 System-Entwicklung für eingebettete Systeme

Dauer: 90 Min	Übungszeit: 30 Min

3.1 Begriffe und Konzepte

Systemarchitektur, Funktionale Architektur, System-Entwicklung, Software-Entwicklung, Hardware-Entwicklung, Hardware-Software-Schnittstelle.

3.2 Lernziele

Die Teilnehmer können:

- Die grundlegende Struktur eines System-Entwicklungsprozesses erklären:
  - Einbettung von Software- und Hardware-Entwicklungsprozessen in einen System-Entwicklungsprozess
  - Wechselwirkungen zwischen Software-Entwicklung, System-Entwicklung und Hardware-Entwicklung.
- Inhalte einer System-Architektur beschreiben.
- Unterschiede und Zusammenhänge zwischen funktionaler Architektur und technischer Systemarchitektur erklären.
- Hardware-Software-Schnittstellen spezifizieren.

Die Teilnehmer verstehen:

- Domänenspezifische Normen, Standards und rechtliche Anforderungen haben signifikanten Einfluss auf den Systementwicklungsprozess.

Die Teilnehmer kennen:

- Beispiele für System-Entwicklungsprozesse
- Beispiele für Modellierungs-Notationen für System-Architekturen, z. B. SysML
- Beispiele für Werkzeuge für die System-Architektur
- Beispiele für relevante Normen und Standards, z. B. IEC 61508, ISO 15504

3.3 Referenzen

[Andreas+2008]

[Liggesmeyer+2005]

[Wolf 2007]

4 Software-Entwicklung für eingebettete Systeme

Dauer: 120 Min	Übungszeit: 45 Min

4.1 Begriffe und Konzepte

Software-Architektur, UML, Domänenspezifische Sprachen, Codegenerierung.

4.2 Lernziele

4.2.1 Modellierung von Software-Architekturen für eingebettete Systeme

Die Teilnehmer können:

- Zu verschiedenen Ansätzen für die Modellierung von Software-Architekturen für eingebettete Systeme deren grundlegende Konzepte, Stärken und Schwächen beschreiben:
  - UML
  - Graphische und textuelle domänenspezifische Sprachen, z. B. ROOM
- Anhand der Anforderungen und Randbedingungen eines Projekts einen geeigneten Modellierungs-Ansatz auswählen.

Die Teilnehmer verstehen:

- UML kann durch die Verwendung von Profilen für die Modellierung eingebetteter Systeme angepasst werden (z. B. MARTE).

Die Teilnehmer kennen:

- Beispiele für Modellierungswerkzeuge

4.2.2 Umsetzung von Software-Architekturen für eingebettete Systeme

Die Teilnehmer können:

- Zu verschiedenen Ansätzen für die Umsetzung von Software-Architekturen für eingebettete Systeme deren grundlegende Konzepte, Stärken und Schwächen beschreiben:
  - Codegenerierung aus Zustandsmodellen
  - Codegenerierung aus Datenflussmodellen (z. B. Simulink)
  - Codegenerierung von Rahmen und Schnittstellen
  - Codegenerierung von Infrastruktur-Code
  - Codegenerierung aus graphischen oder textuellen domänenspezifischen Sprachen, z. B. ROOM
  - Manuelle Implementierung.
- Erklären, wie verschiedene Ansätze zur Umsetzung kombiniert werden können.
- Anhand der Anforderungen und Randbedingungen eines Projekts einen geeigneten Ansatz zur Umsetzung auswählen.

Die Teilnehmer verstehen:

- Bei der Auswahl eines Ansatzes für die Modellierung oder die Umsetzung von Software-Architekturen für eingebettete Systeme müssen häufig normative Anforderungen berücksichtigt werden (z. B. Einsatz von Codegeneratoren in sicherheitsbezogenen Systemen, Zertifizierung von Werkzeugen).
- Bei eingebetteten Systemen spielen begrenzte System-Ressourcen häufig eine wichtige Rolle bei der Umsetzung (z. B. möglicherweise erhöhter Ressourcen-Bedarf generierten Codes gegenüber manuell implementiertem Code).

Die Teilnehmer kennen:

- Beispiele für Werkzeuge zur Codegenerierung für eingebettete Systeme

4.3 Referenzen

[Hruschka+2002]

[Liggesmeyer+2005]
[Rupp+2012]
[Samek 2008]
[Selic+1994]
[Wolf 2007]

5 Funktionale Sicherheit

Dauer: 390 Min	Übungszeit: 150 Min

5.1 Begriffe und Konzepte

Funktionale Sicherheit, Zuverlässigkeit, Safety vs. Security, IEC 61508, Gefahrenanalyse, Sicherheits-Integritätslevel (SIL), Sicherheitskonzept, Funktionale Architektur, Technische Architektur, HW- und SW-Fehlertoleranz, fail safe vs. fail operational.

5.2 Lernziele

5.2.1 Grundlegende Begriffe, Normen und Standards

Die Teilnehmer können:

- Den Begriff „Funktionale Sicherheit" definieren und von den Begriffen Zugriffssicherheit (Security) und Zuverlässigkeit abgrenzen.
- Ein sicherheitsbezogenes System gemäß IEC 61508 definieren.
- Umfang, Einflussbereich und Struktur der IEC 61508 erklären: Prozess-Vorgaben, Methoden-Einsatz, konkrete Architektur-Vorgaben.
- Den Geltungsbereich der IEC 61508 erklären.

Die Teilnehmer verstehen:

- Ob ein sicherheitsbezogenes System normenkonform entwickelt wird, hat rechtliche Konsequenzen im Schadensfall (z. B. auf Haftung und Strafverfolgung).
- Die IEC 61508 ist eine generische Norm. Für bestimmte Anwendungsbereiche gibt es spezifische Normen, die anstelle der IEC 61508 gelten. Diese können von der IEC 61508 abgeleitet sein oder unabhängig sein.

Die Teilnehmer kennen:

- Beispiele für domänenspezifische Normen für sicherheitsbezogene Systeme, z. B. ISO 26262, Normengruppe IEC 60601, IEC 62304, IEC 61511, DO 178B.

5.2.2 Vorgehen bei der Entwicklung sicherheitsbezogener Systeme

Die Teilnehmer verstehen:

- Die IEC 61508 definiert einen Sicherheitslebenszyklus, der unter anderem folgende Elemente umfasst:
  - Gefahrenanalyse und Risikoanalyse
  - Definition und Zuordnung von Sicherheitsanforderungen
  - Realisierung auf System-, Hardware- und Software-Ebene
- Funktionale Sicherheit muss auf Systemebene betrachtet werden
- Bei der Umsetzung von Sicherheitsmaßnahmen sind Hardware und Software eng verzahnt, z. B. bei der Diagnose von Hardware-Defekten

Die Teilnehmer kennen:

- Die Methoden FTA, FMEA, FMEDA und ihre Einsatzbereiche.

5.2.3 Sicherheits-Integritätslevel

Die Teilnehmer verstehen:

- Das grundlegende Vorgehen bei der Ermittlung eines Sicherheits-Integritätslevels
- Wie sich der Sicherheits-Integritätslevel im Produkt vererbt (z. B. von Sicherheitszielen auf Sicherheitsanforderungen, Systembestandteile, abgeleitete Sicherheitsanforderungen)
- Der Sicherheits-Integritätslevel von Teilsystemen kann unter bestimmten Voraussetzungen reduziert werden
- Die Reduzierung des Sicherheits-Integritätslevels kann zu kostengünstigeren Lösungen führen

- Die Reduzierung des Sicherheits-Integritätslevels setzt unabhängige Teilsysteme voraus
- Der Sicherheits-Integritätslevel hat unmittelbare Auswirkungen auf die Software-Architektur und auf die Arbeitsweise des Software-Architekten. Abhängig vom Sicherheits-Integritätslevel werden bestimmte Methoden und Architekturmaßnahmen empfohlen.

5.2.4 System-Architektur

Die Teilnehmer kennen:

- Die Sicherheitsarchitekturen 1oo1, 1oo2, 2oo2, 1oo2D und 2oo3 und deren Anwendungsbereiche.

5.2.5 Software-Architektur

Die Teilnehmer können:

- Anhand der Sicherheitsanforderungen angemessene Lösungen für die Software-Architektur auswählen, z. B.:
  - Überwachungsfunktionen (Watchdogs, Kontrollflußüberwachung, Überwachung von Kommunikation, Sicherstellung der Datenintegrität)
  - Software-basierte Speicherschutztechniken
  - Diversitäre Software als Maßnahme gegen systematische Fehler
  - Replizierte Ausführung als Maßnahme gegen transiente Fehler
  - Maßnahmen bei der Erkennung persistenter Fehler (z. B. Deaktivierung von Funktionen)
  - Auswahl von Programmiersprachen, Verwendung von Sprachenteilmengen und Coding Standards (z. B. MISRA C).

5.2.6 Werkzeuge

Die Teilnehmer verstehen:

- Werkzeuge, z. B. Code-Generatoren und Compiler, können direkte Auswirkungen auf die Sicherheit des Produkts haben.
- Werkzeuge werden häufig nicht gemäß der für das Produkt relevanten Sicherheitsnorm entwickelt.

Die Teilnehmer können:

- Verschiedene Ansätze für den Einsatz von Werkzeugen für die Entwicklung sicherheitsbezogener Produkte benennen:
  - Einsatz eines betriebsbewährten Werkzeugs
  - Bewertung des Entwicklungsprozesses des Werkzeug-Anbieters
  - Zertifizierung des Werkzeugs
  - Verifikation des Werkzeug-Ergebnisses (z. B. bei einem Code-Generator die Verifikation des generierten Codes gemäß der relevanten Sicherheitsnorm)
  - Einsatz eines Werkzeugs, das gemäß der Sicherheitsnorm entwickelt wurde
- Für ein konkretes Werkzeug und ein konkretes Produkt bewerten, welche Ansätze geeignet sind .

5.2.7 Einsatz von bestehenden Software-Komponenten

Die Teilnehmer verstehen:

- Bestehende Software-Komponenten (z. B. Fremdsoftware oder aus anderen Projekten übernommene Software) sind häufig nicht gemäß der für das Produkt relevanten Sicherheitsnorm entwickelt.

Die Teilnehmer können:

- Ansätze benennen, wie bestehende Software-Komponenten in ein sicherheitsbezogenes Produkt integriert werden können:
  - Einsatz von betriebswährten Software-Komponenten
  - Verifikation der Software-Komponenten gemäß der relevanten Sicherheitsnorm.

- Für eine konkrete Software-Komponenten und ein konkretes Produkt bewerten, welche Ansätze geeignet sind.

5.3 Referenzen

[Börcsök 2011]
[IEC 61508]
[ISO 26262]
[Liggesmeyer+2005]
[NASA 2004]
[Smith+2011]

6 Echtzeit und Nebenläufigkeit

Dauer: 390 Min	Übungszeit: 150 Min

6.1 Begriffe und Konzepte

Harte vs. weiche Echtzeitanforderungen, Cyclic Executive, Interrupts, Echtzeit-Betriebssysteme, Jobs, Tasks, Threads, Prozesse, statisches vs. dynamisches Scheduling, zeitgesteuerte vs. ereignisgesteuerte Echtzeitsysteme, Priority Inversion, Priority Inheritance, Priority Ceiling, Planbarkeitsanalyse, WCET, konkurrierender Ressourcen-Zugriff und Koordinierungsmaßnahmen.

6.2 Lernziele

6.2.1.1 Grundlegende Begriffe von Echtzeitsystemen

Die Teilnehmer verstehen:

- Die Echtzeitanforderungen für eingebettete Systeme ergeben sich aus der Interaktion mit der Umwelt
- Bei der Umsetzung von Echtzeitanforderungen müssen grundsätzlich alle Aktionen vom Eintritt eines relevanten Ereignisses bis zur Systemreaktion betrachtet werden: z. B. Auslesen von Sensordaten, Kontextwechsel des Betriebssystems, Auswertung der Sensordaten, Ansteuerung des Aktuators
- Bei der Umsetzung harter Echtzeitanforderungen muss für jede Aktion eine definierte obere Zeitschranke gegeben sein.

Die Teilnehmer können:

- Den Unterschied zwischen harten Echtzeitanforderungen und weichen Echtzeitanforderungen erklären
- Den Unterschied zwischen Rechtzeitigkeit und Geschwindigkeit erklären
- Zeitgesteuerte und ereignisgesteuerte Ansätze beschreiben
- Die Lösungsansätze *Cyclic Executive*, *Multitasking* und *Interrupts* beschreiben und ihre Stärken und Schwächen benennen
- Kriterien für die Auswahl der Lösungsansätze benennen
- Beschreiben wie verschiedene Lösungsansätze kombiniert werden können (z. B. Cyclic Executive und Interrupts)
- Für ein konkretes System einen Lösungsansatz auswählen und die Entscheidung nachvollziehbar begründen.

6.2.1.2 Cyclic Executive

Die Teilnehmer können:

- Vorteile und Nachteile von Cyclic Executive beschreiben
- Die Echtzeit-Eigenschaften einer Lösung auf Basis von Cyclic Executive analysieren

6.2.1.3 Multitasking

Die Teilnehmer können:

- Den Unterschied zwischen statischen und dynamischen Scheduling-Verfahren beschreiben
- Präemptive und nicht-präemptive Scheduling-Verfahren definieren
- Prioritätenbasierte Scheduling-Verfahren beschreiben
- Stärken und Schwächen der verschiedener Scheduling-Verfahren benennen (statisch vs. dynamisch, präemptiv vs. nicht-präemptiv)
- Anhand der Echtzeitanforderungen ein geeignetes Scheduling-Verfahren auswählen und die Tasks und ihre Eigenschaften definieren
- Mögliche Probleme durch konkurrierenden Zugriff auf gemeinsame Ressourcen benennen
- Strategien zur Verhinderung von typischen Nebenläufigkeitsproblemen beschreiben und für ein konkretes System auswählen (kritische Abschnitte, Semaphoren, Synchronisationsobjekte/Mutexe, nicht-blockierende Synchronisation)

- Erklären wie Verklemmungen entstehen und für ein konkretes System aufzeigen, wie Verklemmungen vermieden werden
- Priority Inversion erklären und Lösungsansätze aufzeigen (Priority Ceiling, Priority Inheritance).

6.2.1.4 Interrupts

Die Teilnehmer können:

- Stärken und Schwächen bei der Verwendung von Interrupts auf Applikationsebene benennen
- Ein Konzept für den Einsatz von Interrupts auf Applikationsebene definieren: Priorisierung von Interrupts, Regeln für die Sperrung von Interrupts, Einsatz des Prolog-/Epilog-Modells

Die Teilnehmer verstehen:

- Konzepte für den Einsatz von Interrupts auf Applikationsebene sind grundsätzlich von der Prozessor-Architektur abhängig (z. B. Anzahl der Prioritäten)

6.2.1.5 Interaktion zwischen Jobs

Die Teilnehmer können:

- Nachrichten-Austausch und Nutzung gemeinsamer Daten als grundlegende Ansätze zur Interaktion zwischen Jobs definieren und deren Konsequenzen benennen.
- Auswirkungen von Interaktionen zwischen Jobs auf das Echtzeit-Verhalten analysieren.
- Konzepte definieren, wie Interaktionen mit den Echtzeitanforderungen von Jobs vereinbart werden können.

6.2.1.6 Echtzeit-Betriebssysteme

Die Teilnehmer können:

- Die Merkmale eines Echtzeit-Betriebssystems erklären.

Die Teilnehmer verstehen:

- Echtzeit-Betriebssysteme und General Purpose Betriebssysteme verfolgen unterschiedliche Ziele.

Die Teilnehmer kennen:

- Mindestens zwei Beispiele für Echtzeit-Betriebssysteme und ihre Eigenschaften (z. B. QNX, eCos, OSEK-OS)

6.2.1.7 Analyse von Echtzeit-Eigenschaften

Die Teilnehmer verstehen:

- Die Bestimmung der maximalen Ausführungszeit eines Jobs ist grundlegend für die Analyse der Echtzeit-Eigenschaften (Worst Case Execution Time/WCET).
- Messungen der Laufzeit erlauben bei komplexen Prozessorarchitekturen keine verlässliche Abschätzung der maximalen Ausführungszeit (Auswirkungen von Caches, Pipelines, …).
- Eine obere Grenze für die maximale Ausführungszeit eines Jobs kann für bestimmte Prozessoren werkzeuggestützt bestimmt werden.

Die Teilnehmer kennen:

- Werkzeuge für die Bestimmung von maximalen Ausführungszeiten.
- Die Einschränkungen solcher Werkzeuge bzgl. Komplexität der Prozessoren und der Genauigkeit.

Die Teilnehmer verstehen:

- Die Einplanbarkeit von Tasks kann durch einzelne Messungen nicht verlässlich nachgewiesen werden.
- Es existieren analytische Verfahren für die Analyse der Einplanbarkeit (z. B. RMA).
- Analytische Verfahren haben Limitierungen bzgl. Abhängigkeiten zwischen Tasks und deren Auswirkungen (pessimistische Ergebnisse).

Die Teilnehmer kennen:

- Werkzeuge für die Analyse der Einplanbarkeit

6.3 Referenzen

[Liggesmeyer+2005]
[Liu 2000]
[Norton+1997]
[Simon 1999]
[Stallings 2003]
[Wolf 2007]

7 Verteilte Systeme

Dauer: 90 Min	Übungszeit: 30 Min

7.1 Begriffe und Konzepte

Zeitgesteuerte Kommunikation, ereignisgesteuerte Kommunikation, Latenz, Durchsatz.

7.2 Lernziele

Die Teilnehmer können:

- Zeitgesteuerte und ereignisgesteuerte Ansätze zur Kommunikation in verteilten eingebetteten Systemen erklären.
- Für ein konkretes System einen Ansatz zur Kommunikation auswählen und die Entscheidung nachvollziehbar begründen.
- Ausgehend von den Eigenschaften des Bussystems eine konkrete Verteilung der Funktionen ableiten und nachvollziehbar begründen.

Die Teilnehmer verstehen:

- Die Auswirkungen, die die Eigenschaften eines Bussystems auf die Verteilung haben
- Die Voraussetzungen, die für die Zusammenfassung von Funktionen auf einem Steuergerät gegeben sein müssen, insbesondere bei Funktionen unterschiedlicher Kritikalität.

Die Teilnehmer kennen:

- Gängige Bussysteme für eingebettete Systeme (z. B. Ethernet, CAN, Flexray, LIN, Feldbusse) und ihre Eigenschaften bzgl. Latenz, Durchsatz, Echtzeit
- Gründe für die Verteilung von Funktionen auf mehrere Steuergeräte: Weniger Kosten und Gewicht durch geringeren Verkabelungsaufwand, wenn Steuergeräte nahe bei Sensoren und Aktuatoren platziert werden können; erhöhte Sicherheit durch Redundanz; organisatorische Ursachen
- Gründe für die Zusammenfassung von Funktionen auf einem Steuergerät: Reduzierung der Komplexität; weniger Kosten und Gewicht durch geringere Steuergeräte-Anzahl; zunehmende Leistungsfähigkeit von Microcontrollern; erhöhter Durchsatz und reduzierte Latenz bei lokaler Kommunikation.

7.3 Referenzen

[Liggesmeyer+2005]
[Liu 2000]
[Stallings 2003]
[Wolf 2007]

8 Variantenmanagement

Dauer: 90 Min	Übungszeit: 30 Min

8.1 Begriffe und Konzepte

Varianten, Merkmal-Modelle, FODA, Produktlinien.

8.2 Lernziele

Die Teilnehmer können:

- Variabilität modellieren, z. B. mit FODA (Feature Oriented Domain Analysis)
- Verschiedene Mechanismen zur Umsetzung von Variabilität sowie deren Stärken und Schwächen beschreiben:
  - Aggregation/Delegation
  - Vererbung
  - Parametrisierung
  - Auswahl alternativer Bausteine
  - Bedingte Kompilierung
  - Aspektorientierte Programmierung
  - Generierung
- Für ein konkretes System Mechanismen für die Umsetzung von Variabilität auswählen und die Entscheidung nachvollziehbar begründen.

Die Teilnehmer verstehen:

- Mechanismen zur Umsetzung von Variabilität wirken zu unterschiedlichen Zeitpunkten, z. B. Übersetzungszeit vs. Laufzeit.

Die Teilnehmer kennen:

- Werkzeuge für das Variantenmanagement.

8.3 Referenzen

[Anastasopoulos 2000]

[Böckle+2004]

[Clements+2003]

[Linden+2007]

9 Quellen und Referenzen zu Sicherheitskritische Eingebettete Systeme

Dieser Abschnitt enthält Quellenangaben, die ganz oder teilweise im Curriculum referenziert werden.

[Anastasopoulos 2000]
M. Anastasopoulos, C. Gacek: Implementing Product Line Variabilities, IESE Report 089.00/E, Fraunhofer IESE

[Andreas+2008]
J. Andreas et al: Embedded Software, Elsevier, 2008

[Böckle+2004]
G. Böckle, P. Knauber, K. Pohl. K. Schmid: Software-Produktlinien – Methoden, Einführung und Praxis, dpunkt.verlag

[Börcsök 2011]
J. Börcsök: Funktionale Sicherheit, VDE Verlag

[Clements+2002]
P. Clements, L. Northrop: Software Product Lines – Practices and Patterns, Addison Wesley

[Hruschka+2002]
P. Hruschka, C. Rupp: Agile Softwareentwicklung für Embedded Real-Time Systems mit der UML, Hanser

[Kang+1990]
K. C. Kang et al: Feature Oriented Domain Analysis (FODA) Feasibility Study, Technical Report CMU/SEI-90-TR-21

[IEC 61508]
Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme, 2010

[ISO 26262]
Road vehicles - Functional safety, 2011

[Liggesmeyer+2005]
P. Liggesmeyer, D. Rombach (Hrsg.): Software Engineering eingebetteter Systeme, Elsevier

[Linden+2007]
F. J. van der Linden, K. Schmid, E. Rommes: Software Product Lines in Action, Springer

[Liu 2000]
J.W.S. Liu: Real Time Systems, Prentice Hall

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

[Norton+1997]
S. Norton, M. Dipasquale: Threadtime, Prentice Hall

[Rupp+2012]
C. Rupp et al: UML 2 glasklar, Hanser

[Samek 2008]
M. Samek: Practical Statecharts in C/C++, Butterworth Heinemann

[Selic+1994]
B. Selic et al: Real-Time Object-Oriented Modeling, Wiley

[Simon 1999]
D. Simon: An Embedded Software Primer, Addison Wesley

[Smith+2011]
D. Smith, K. Simpson: Safety Critical Systems Handbook, Butterworth Heinemann

[Stallings 2003]
W. Stallings: Betriebssysteme, Pearson

[Wolf 2007]
W. Wolf: High Performance Embedded Computing, Morgan Kaufmann