Design for Test und Design for Safety – Software-Architektur nach Maß

Unser Alltag ist heute wie selbstverständlich von miteinander vernetzten Geräten und Systemen geprägt. Ob man mit dem Smartphone unterwegs den schnellsten Weg zum Ziel findet, auf dem Sofa mit dem Tablet die Zeitung liest oder die smarte Heizung über eine App auf dem Smartphone steuert, diese Systeme machen unser Leben komfortabler. Der Gewinn an Komfort erfordert jedoch auch strengere Security- und Safety-Anforderungen, mit denen die Entwickler solcher Systeme Schritt halten müssen. Dies gilt besonders für das autonome Fahren – hier haben schlüssige Safety-Konzepte oberste Priorität.

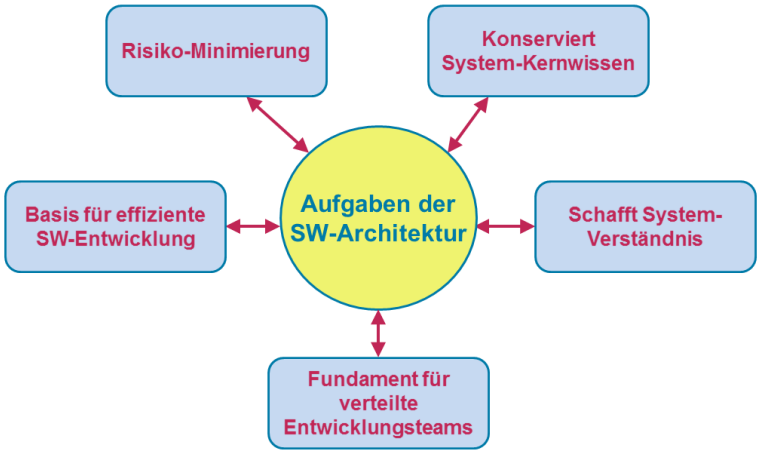

Bild 1: Die Software-Architektur – fit für das Projekt

Knowhow und Fertigkeiten, über die Software-Architekten verfügen sollten

Mit steigender Produktkomplexität und immer leistungsfähigerer Hardware erhöhen sich auch der Umfang und die Komplexität der Software von Embedded-Systemen. In vielen Produkten setzt die Software den wesentlichen Teil der Funktionalität um. Die Abteilungen, die Embedded-Software entwickeln, wachsen kontinuierlich. Dies ist insbesondere in der Automobilindustrie zu sehen und spiegelt sich auch auf dem aktuellen Arbeitsmarkt wider. So plant Mercedes-Benz beispielsweise, ab 2030 den überwiegenden Anteil am Umsatz durch softwarebasierte Systeme zu erzielen. Es wird nicht mehr in einer „One-Man-Show" entwickelt, sondern in großen Teams – verteilt auf unterschiedliche Standorte auf der ganzen Welt.

Der Stellenwert von Embedded-Software hat sich in den vergangenen Jahren in den meisten Unternehmen – sogar im Produktbereich der Mechatronik – der Embedded-Branche drastisch erhöht. Doch dies ist erst der Anfang.

Agile Software-Entwicklungsmethoden stehen verstärkt im Rampenlicht

In agilen Softwareprojekten wird die Software-Architektur evolutionär entwickelt, unter anderem basierend auf testgetriebenen Software-Entwicklungsmethoden. Dabei herrschen zwei unterschiedliche Entwicklungsansätze vor:

- **Funktionsarchitektur**: Das Softwaresystem wird in Funktionen bzw. Features und deren Abhängigkeiten dargestellt.
- **Komponentenarchitektur**: Entwickelt einen Grobentwurf sowie mehrere Feinentwürfe, die eine feingranulare Struktur der Software enthalten.

Software-Architektur spielt eine tragende Rolle für den Projekterfolg

Damit der Software-Architekt seiner verantwortungsvollen Tätigkeit gerecht werden kann, benötigt er fundiertes Knowhow zu folgenden maßgeblichen Aspekten:

Grundverständnis der Software-Architektur

Auf abstrakter Ebene stellt die Software-Architektur eine Brücke zwischen den Anforderungen und der Implementierung der Software dar. In der Software beschreibt die Architektur die grobe Struktur (nur in Ausnahmefällen auch Module und Klassen), z.B. bestehend aus Software-Komponenten, Software-Schichten, Software-Subsystemen, Interfaces und deren Abhängigkeiten. Für diese Architekturelemente lassen sich aber auch ein interaktives und ein individuelles Verhalten beschreiben. Ein wesentlicher Bestandteil der Software-Architektur ist auch die **Laufzeitarchitektur**.

Die Rolle des Software-Architekten

Jeder mit dem notwendigen Knowhow kann im Unternehmen die Rolle des Software-Architekten in einem Projekt einnehmen. Doch um das Thema wirklich professionell zu betreiben, sollte man aus meiner Sicht die personengebundene Rolle bevorzugen. Abhängig von der Projektgröße sind einer oder mehrere Software-Architekten an einem Projekt beteiligt.

Der Chef-Architekt leitet Software-Architektenteams an

Der Software-Architekt stimmt sich mit mehreren Rollen im Projekt ab und benötigt dafür fachliches und nicht-fachliches Wissen – je mehr Erfahrung er hat, desto besser. So sollte die Rolle des Software-Architekten idealerweise nicht mit einem Absolventen direkt von der Hochschule besetzt werden. Hier sind extrovertierte, innovationsfreudige, entscheidungsfreudige und erfahrene Mitarbeiter gefragt.

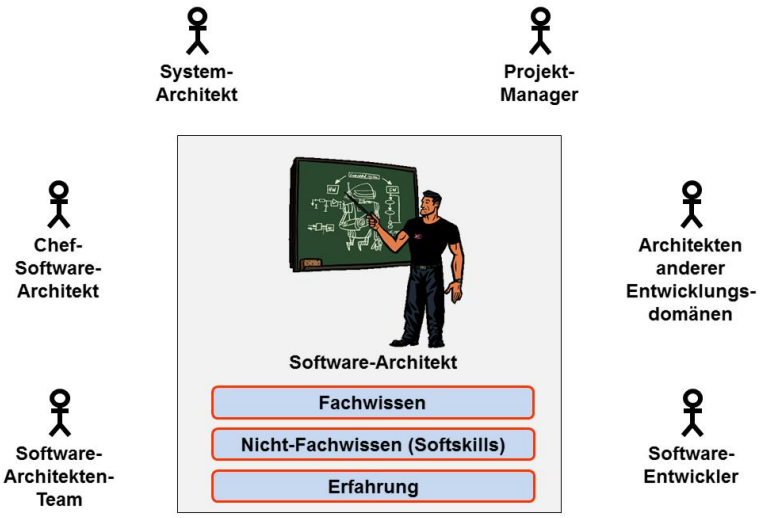

Bild 2: Wichtigster Kontext des Software-Architekten

Entwurfsvorgehen – so entsteht eine Software-Architektur

Das Entwurfsvorgehen beschreibt den Entstehungsprozess der Software (-Architektur). Jedes Unternehmen muss das für sich am besten geeignete Vorgehen finden und implementieren. An dieser Definition ist der Software-Architekt wesentlich beteiligt.

Basierend auf einer V-Modell-artigen Darstellung kann das Entwurfsvorgehen für die Entwicklung eines kompletten Embedded-Systems eingesetzt werden, also nicht nur für die Software-Entwicklung.

Anforderungen (WAS) und entsprechende Architekturen (WIE)

In den Analyseaktivitäten identifizieren und dokumentieren die Analysten (real betrachtet meist auch die Architekten) der einzelnen Ebenen die entsprechenden Anforderungen (das WAS). Darauf basierend entstehen die Architekturen (das WIE). Auf den Subsystem-Architekturen aufbauend entwickelt der Software-Architekt die Software-Architektur für ein Subsystem – in Abstimmung mit den anderen Entwicklungsdomänen auf gleicher Ebene (z.B. Hardware-Entwicklung).
Parallel zu den Anforderungen entwickelt das Testteam die Testfälle, um später die korrekte Umsetzung nachzuweisen. Dies erfolgt ebenfalls auf den unterschiedlichen Ebenen. „Design for Test“ und neu „Design for Safety“ sind dabei grundlegende Software-Architekturthemen.

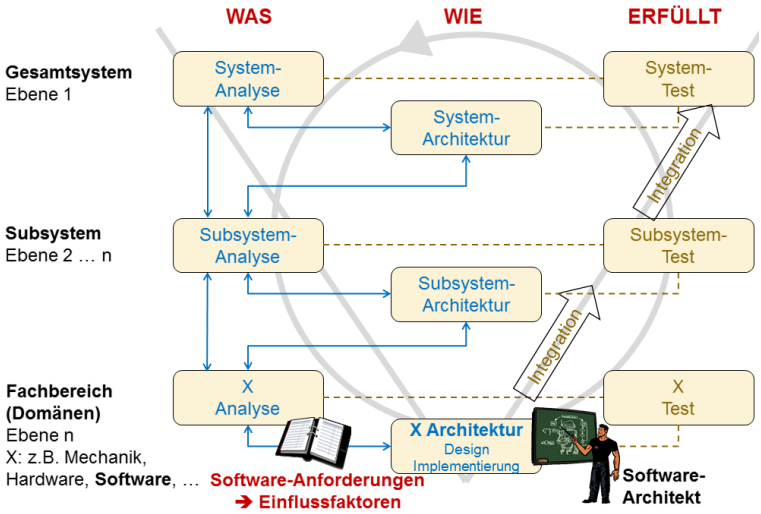

Bild 3: Entwurfsvorgehen für Embedded-Systeme

Entwurfsbasis und Einflussfaktoren

Aus der in Bild 3 dargestellten X-Analyse (hier Software-Analyse) entstehen die Software-Anforderungen (funktional und nicht-funktional).

Während einer **Einflussfaktorenanalyse** entscheidet der Software-Architekt über die

- Relevanz der Anforderung für die Software-Architektur
- Veränderbarkeit der Anforderung in der Zukunft
- Ableitung der Konsequenzen für die Software-Architektur

Teil der nicht-funktionalen Software-Anforderungen sind die **Software-Qualitätsmerkmale**, die die Software erbringen muss, wie z.B.

- Portierbarkeit
- Wartbarkeit
- **Zuverlässigkeit**
- **Sicherheit**
- Ressourcenverbrauch
- Performance
- Echtzeitfähigkeit

Software quality attributes	Implementation consequences for the software architecture
SAFETY (Functional safety)	▪ Continuous failure observation ▪ Failure-dependent execution of correct – failure handling – failure recovery ▪ Failure logging ▪ Status – Generation, test, evaluation, reaction ▪ Protection of – Boundary values – Data memory content and boundaries – Program code execution / runtime ▪ No use of dynamic memory ▪ No use of software components if they are – unknown – without source code – not certified

Tabelle 1: Sicht durch die Brille der relevanten Software-Qualitätsmerkmale – Safety

Software quality attributes	Implementation consequences for the software architecture
RELIABILITY	▪ **Redundancy** (multi channels) – Redundant data storage – Redundant program execution – Safe data types (uint32_t ➔safe_uint32_t) ▪ **Diversity** – Software (different solutions) – Development tools (different solutions, tools) – Human (different solutions, tools and persons) ▪ **Status evaluation** ▪ **Protection** ▪ **Fail-over handling** ▪ **Recovery** ▪ **Dynamic memory management** is not allowed

Tabelle 2: Aus Sicht der relevanten Software-Qualitätsmerkmale – Reliability

Bei den Qualitätsmerkmalen ist zu beachten, dass manche mitläufig, andere aber auch gegenläufig sind. Stellen wir uns mit diesem Wissen die folgende Frage: Welche Anforderungen beeinflussen die Architektur mehr – funktionale oder nicht-funktionale?

Richtige Antwort: die nicht-funktionalen Anforderungen!

Somit bilden die Software-Anforderungen und die daraus resultierenden Einflussfaktoren neben der Subsystem-Architektur die wichtigste Entwurfsbasis für die Software-Architektur.

Kommunikation und Dokumentation

Der Software-Architekt schafft mit einer verständlichen Dokumentation der Software-Architektur für alle Stakeholder die Basis für jedes Projekt und damit eine lückenlose Nachvollziehbarkeit für alle Projektbeteiligten. Damit sichert er den Fortbestand des Unternehmens.

Die Dokumentation bildet gleichzeitig die Kommunikationsbasis, die kontinuierlich mit den Stakeholdern abzugleichen ist. Wichtigster Stakeholder ist dabei der Software-Entwickler, der die Software-Architektur im Design verfeinert und final in der Zielprogrammiersprache implementiert. Neben dem Software-Entwickler haben weitere Rollen, wie beispielsweise das Software-Testteam, ein berechtigtes Interesse an der Software-Architektur. Nur wenn ich weiß, was gewollt ist, kann ich prüfen, ob dies auch wirklich so umgesetzt wurde

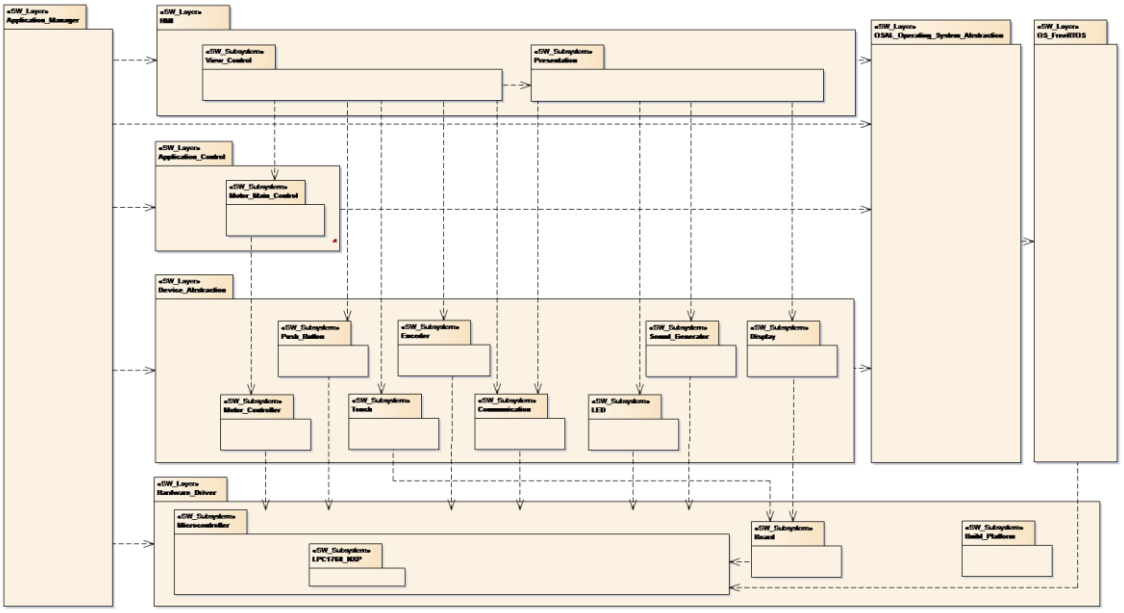

Bild 4: Beispiel einer Software-Schichtenarchitektur

UML (Unified Modeling Language) ist die Notation, mit der sich verschiedenste Sichten und Aspekte der Software-Architektur dokumentieren und im Design verfeinern lassen – bis hin zur automatischen Codegenerierung. Mit dem Paketdiagramm sind in Bild 4 die unterschiedlichen Software-Schichten modelliert.

Software-Entwurfsprinzipien verbessern die Softwarequalität

Unser gesamtes Leben ist bestimmt von Regeln, auch wenn manche meinen, sich nicht daran halten zu müssen. Jeder hat sicher eigene Erfahrungen mit dem Thema Corona und dem „Einhalten“ von Vorschriften und Regeln gemacht. Bestimmt haben Sie früher selbst als Kind Lego gespielt oder tun dies heute mit Ihren Kindern. Auch hier gibt es Regeln, wie die Steine zusammenpassen.

Der Software-Architekt entwickelt mit immer neuem Wissen den Styleguide der Software-Entwicklung. Darin werden die Regeln beschrieben, nach denen die Software-Architekturen entwickelt werden soll. Nicht für alle Architekturen können alle Regeln angewendet werden bzw. gelten diese. Die Anwendung ist abhängig von den jeweils benötigten Anforderungen. Die Anwendung der Regeln auf eine Software-Architektur verbessert in jedem Fall die Softwarequalität.

Ein Architektur-Entwurfsprinzip ist beispielsweise das der **hohen Kohäsion**. Ziel hierbei ist es, zur Vermeidung von Redundanzen logisch Zusammengehöriges in einem Architekturelement zu bearbeiten und gleichartige Aufgaben nicht auf mehrere Architekturelemente zu verteilen. Inzwischen existieren Veröffentlichungen zu speziellen Entwurfsprinzipien, die für Embedded-Software-Architekturen anwendbar sind. Mit **Software-Architekturpatterns** kann der Software-Architekt die Entwurfsprinzipien in der Praxis umsetzen.

Architekturentwicklung und Architekturpatterns müssen Safety-Anforderungen erfüllen

Basierend auf seinem fachlichen Wissen entwickelt der Software-Architekt die Software-Architektur. Dabei greift der Architekt auf seinen **Pattern-Katalog** zurück. Allgemein sind Patterns bekannte, bewährte, bewertete und anpassbare Lösungen zu immer wiederkehrenden Problemstellungen (Herausforderungen).

Für Safety-relevante Systeme müssen hier Aspekte wie funktionale Sicherheit und Zuverlässigkeit berücksichtigt und erfüllt werden.

Bei Systemen, die uns vollautomatisiert unterstützen sollen (denken wir an das automatisierte Fahren), sind die Aspekte **Safety** und **Reliability** maßgeblich für den Erfolg des Produktes:

Software quality attributes	Implementation consequences for the software architecture
SAFETY and RELIABILITY	▪ **Hardware protects hardware** – Clock protection – Overvoltage and undervoltage protection – Overcurrent (and undercurrent) protection – Temperature protection
	▪ **Software protects hardware** – Processor tests – Memory tests, checksum protection – Memory boundary protection (stack) – (Peripheral) register value protection / observation – Input value boundary protection / observation

Tabelle 3: Sicht durch die Safety-Brille

Software quality attributes	Implementation consequences for the software architecture
SECURITY (Manipulation from outside) ➔ Access security ➔ Data security	▪ **External accesses**: – As few as possible, but only protected – As many as necessary – Protected access is mandatory
	▪ **Security of critical application parts**: – Bordering by partitioning
	▪ Use of (assumed to be) **secure protocols** – https, ipsec, ssl, …
	▪ **Encryption** – Data and data memory content – Program memory content
	▪ **No use of software components** if they are – unknown – without source code – not certified

Tabelle 4: Und nicht zu vergessen – die Sicht durch die Security-Brille

Der Einsatz von Patterns kann eine Herausforderung bei der Software-Architekturentwicklung sein:

So kann eine Problemstellung z.B. runde Konturenverläufe erfordern, auch wenn nur rechteckige Bausteine zur Verfügung stehen. Eine Lösung wäre, die Steine – nach dem Lego-Prinzip – jeweils um eine oder mehrere Reihen abgestuft zu setzen. Da wir nicht die erste Generation sind, die Software entwickelt, existieren heute für fast alle Bereiche der Softwareentwicklung Patterns, so auch für die Software-Architekturentwicklung.

Ein Beispiel ist hier das Software Layer Pattern (strikt oder nicht strikt). In Bild 4 ist eine nicht strikte Software-Schichtenarchitektur dargestellt. Nicht strikt bedeutet dabei, dass darin schichtübergreifende Zugriffe enthalten sind. Das ist gerade bei Embedded-Software sinnvoll, um die geforderte Performance einzuhalten. In diesem Beispiel sind neben den klassischen horizontalen Schichten auch vertikale enthalten.

Qualitätssicherung und Qualitätsbewertung

Der Software-Architekt ist für die Software-Qualität verantwortlich sowie für die Qualitätssicherung mitverantwortlich. Bevor der Software-Architekt eine Architektur entwickelt, müssen die Qualitätsmerkmale definiert sein. Der Architekt kennt den Einfluss dieser Merkmale auf seine Software-Architektur, und das Software-Testteam weiß, wie sie nachzuweisen sind. Übrigens lässt sich Qualität nicht erst am Ende der Entwicklung in ein Produkt hineintesten!

Bei der Qualität ist zu unterscheiden zwischen

- **interner Qualität** (z.B. Software-Architektur) und
- **externer Qualität** (das sieht der Kunde).

Die **Prozessqualität** beeinflusst maßgeblich auch die Produktqualität. Um auch hier die Analogie zu Lego nochmals zu strapazieren – alle Legobausteine müssen so zusammengebaut sein, dass sie die Konstruktion tragen, sonst fällt diese spätestens bei zusätzlichen Erweiterungen in sich zusammen. Übertragen gilt das auch für die Software-Architektur. Sie muss sämtliche zuvor definierten Qualitätsmerkmale und Funktionalitäten erfüllen. In der Vergangenheit musste eine Software-Architektur oft 20 Jahre und länger tragfähig bleiben. Heute wird sie ständig erweitert und verbessert. Das liegt an immer neuen Anforderungen, Vorschriften und Gesetzen. Ein Entwicklungsprozess, der dies zulässt, ist deshalb enorm wichtig für die Fortentwicklung der Produkte.

Bild 5: Qualitätssicherung und Bewertung

Die einfachste qualitätssichernde Maßnahme

Reviews mit anderen Architekten und Stakeholdern stellen die einfachste qualitätssichernde Maßnahme bei Entwicklung der Software-Architektur dar. Hier wird bewertet, ob die Architektur den geforderten Qualitätsmerkmalen gerecht wird oder nicht. Als Basis für ein Review eignet sich die Software-Architekturdokumentation basierend auf einem UML-Modell. Im **szenarienbasierten Review** spielen die Teilnehmer mit der Architektur zuvor definierte Fälle durch. Muss eine Architektur beispielsweise portierbar in Bezug auf die Hardware sein, wird in einem Gedankenspiel die Hardware getauscht und so nachgewiesen, dass die Software-Architektur dem gerecht wird. Eine sehr umfangreiche Methode dazu hat das **SEI** (Software Engineering Institute der Carnegie Mellon University) entwickelt. Sie trägt die Bezeichnung ATAM (Architecture Tradeoff Analysis Method).

Andere qualitätssichernde Maßnahmen sind beispielsweise das Erstellen von Prototypen oder mathematischen Modellen, die Ausführung einer Simulation und das Ermitteln von Metriken.

Der Tool-Einsatz erleichtert die Entwicklung der Software-Architektur

Der Software-Architekt ist für die Toolwelt rund um die Softwareentwicklung verantwortlich oder zumindest mitverantwortlich.

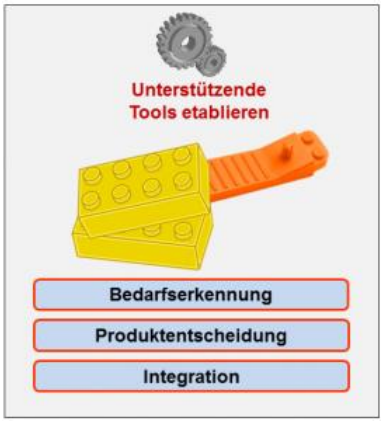

Bild 6: Einsatz von Tools

Er kennt den Toolmarkt, identifiziert den Bedarf, entwickelt Toolanforderungen, evaluiert Tools und wählt sie letztendlich fachlich aus. Gibt es im Unternehmen keine Toolgruppe, ist er auch für die Toolintegration verantwortlich. Die Tools sollen allen Mitwirkenden in der Softwareentwicklung die Arbeit erleichtern. Ein wenig Egoismus schadet nicht, davon profitiert (in erster Linie) der Software-Architekt.

Toolthemen, die die Arbeit des Software-Architekten erleichtern:

- Anforderungsmanagement
- Versions- und Konfigurationsmanagement
- Modellierung
- Generierung von Dokumentation und Programmcode
- Buildsysteme
- Statische Analyse
- Dynamische Analyse
- Implementierung der Software-Architektur

Der Software-Architekt übergibt Teile oder die gesamte Architektur zur weiteren Verfeinerung (Design und Implementierung) an einen oder mehrere Software-Entwickler. Im Coding Styleguide, den der Software-Architekt zusammen mit den Software-Entwicklern verfasst, ist unter anderem ersichtlich, wie sich die Software-Architektur in der Zielprogrammiersprache widerspiegelt. Typische Zielprogrammiersprachen für die Programmierung von Embedded-Systemen sind aktuell C und C++.

Mit C++ ist die Software-Architektur sehr gut über Namespaces im Programmcode abbildbar.

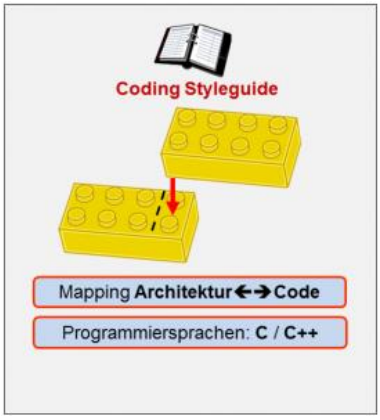

Bild 7: Coding Styleguide

Software-Architekt und Software-Entwickler müssen unbedingt dafür sorgen, dass die vorgegebene Software-Architektur während ihres gesamten Lebenszyklus erhalten bleibt und nicht „kaputt"-programmiert wird – ein auch als Software-Erosion bekannter Vorgang.

Erkennt der Software-Entwickler Änderungsbedarf an der Architektur, so laufen unbedingt alle Änderungsentscheidungen und die Architekturänderung selbst über den verantwortlichen Software-Architekten.

Je höher die Anforderungen an Safety, Security und Modularität der Produkte werden, desto wichtiger und erfolgsentscheidender wird die Funktion des Software-Architekten im gesamten Entwicklungsprozess.

Weiterführende Informationen

- Software Engineering Institute, Carnegie Mellon University: ATAM/Architecture Tradeoff Analysis Method
- V-Modell
- Object Management Group (OMG)
- MicroConsult Download: Prinzipien für Embedded-Software-Architekturen

MicroConsult-Training & Coaching:

- **Software-Architekturen für Embedded- und Echtzeitsysteme**
- **RTOS-Mechanismen und deren Anwendungen in Embedded- und Echtzeitsoftware**
- **Design Patterns (nicht nur) für Embedded-Systeme**
- **Funktionale Sicherheit (Safety) von Elektronik und deren SW**
- **Security: Sicherheit von Embedded-Systemen im Kontext der funktionalen Sicherheit**
- **Embedded-Software-Test: Best Practices für den Unit-/Modul-/Komponenten-Test**
- **Agiles Testen und Test-Driven Development (TDD) von Embedded-Systemen**
- **Übersicht über alle Trainings**

Autor

Thomas Batt studierte nach seiner Ausbildung zum Radio- und Fernsehtechniker Nachrichtentechnik. Seit 1994 arbeitet er kontinuierlich in verschiedenen Branchen und Rollen im Bereich Embedded-/ Realtime-Systementwicklung. 1999 wechselte Thomas Batt zur MicroConsult GmbH. Dort verantwortet er heute als zertifizierter Trainer und Coach die Themenbereiche Systems/ Software Engineering für Embedded-/Realtime-Systeme sowie Entwicklungsprozess-Beratung.