Embedded Software Engineering Kongress

ESE Kongress 2016

Vortragsskript:

Software-Architektur braucht Verantwortung und Können

Welche Themen sollte der Software-Architekt beherrschen?

Thomas Batt, MicroConsult GmbH, t.batt@microconsult.de

1. Vorwort

Mit der steigenden Produktkomplexität und immer leistungsfähigerer Hardware erhöhen sich ebenfalls 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 spiegelt sich auch am aktuellen Arbeitsmarkt wider. Software wird nicht mehr in einer „One-Man-Show“ entwickelt, sondern in Teams, verteilt auf verschiedene Standorte, u.U. rund um die Welt. Somit hat sich der Stellenwert von Embedded-Software in den vergangenen Jahren in den meisten Unternehmen (sogar im Produktbereich der Mechatronik) der Embedded-Branche drastisch erhöht.

Die Erkenntnis, dass die VHIT (vom Hirn ins Terminal) Methode nicht mehr länger haltbar ist, führt zu einem neuen Vorgehen in der Entwicklung und im Test. Dabei spielen die Software-Architektur eine tragende Rolle und somit auch der Software-Architekt. Was der Software-Architekt wissen und tun muss, um seiner verantwortungsvollen Rolle gerecht zu werden, erfahren Sie im folgenden Beitrag.

Die komplette und aktuelle Dokumentation zu diesem Thema haben wir unter [1] für Sie zum Download bereitgestellt.

2. Themenübersicht

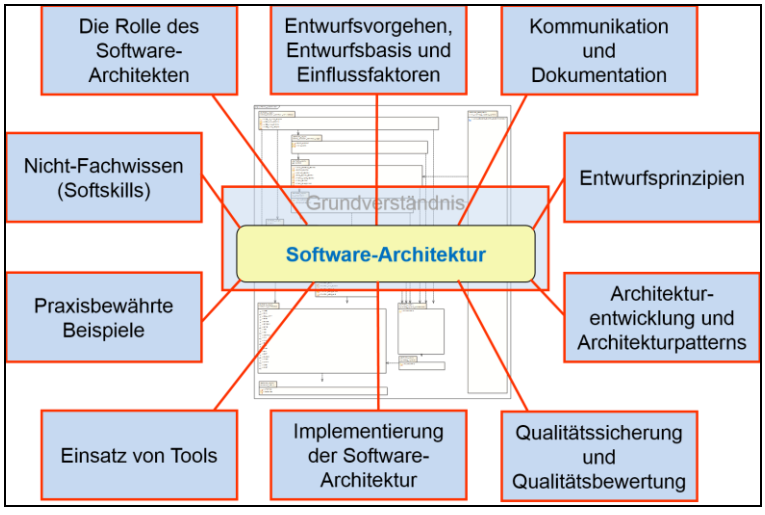

Bild 1: Themenübersicht für den Software-Architekten

Mit denen im Bild 1 dargestellten Themen beschäftigt sich dieser Beitrag.

3. Grundverständnis Software-Architektur

Bei einer Internetrecherche finden sich zahlreiche Definitionen zu Software-Architektur, die hier unerwähnt bleiben.

Auf abstrakter Ebenen repräsentiert die Software-Architektur die Brücke zwischen den Anforderungen und der Implementierung der Software. Im realen Brückenbau gibt es Brücken unterschiedlichster Art und Qualität – manchmal fehlen sie sogar! Dies ist 1:1 auf Software-Architektur übertragbar.

Bild 2: Brücken unterschiedlicher Art und Qualität

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 lässt sich aber auch ein interaktives und ein individuelles Verhalten beschreiben. Wesentlicher Bestandteil der Software-Architektur ist auch die Laufzeitarchitektur [2].

4. Die Rolle des Software-Architekten

Die Rolle des Software-Architekten ist im Unternehmen entweder eine personengebundene Rolle oder jeder, der möchte, darf diese Rolle in einem Projekt besetzen. Um das Thema Software-Architektur professionell zu treiben, ist aus meiner Sicht die personengebundene Rolle zu bevorzugen.

Abhängig von der Projektgröße sind einer oder mehrere Software-Architekten an einem Projekt beteiligt. Ein Chef-Architekt leitet Software-Architekten-Teams 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, umso besser. So sollte die Rolle des Software-Architekten idealerweise nicht mit einem Absolventen direkt von der Hochschule besetzt weder. Hier sind extrovertierte, innovationsfreudige, entscheidungsfreudige und erfahrene Mitarbeiter gefragt.

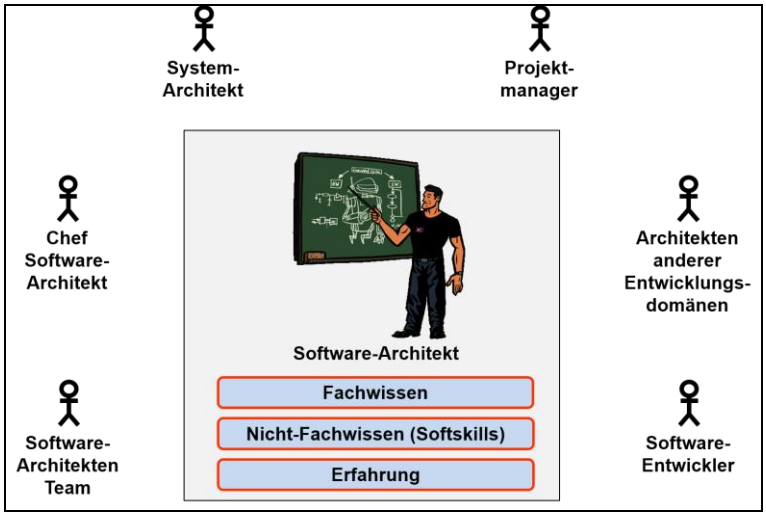

Bild 3: Wichtigster Kontext des Software-Architekten

5. Entwurfsvorgehen

Das Entwurfsvorgehen beschreibt den Entstehungsprozess der Software (-Architektur). Hier muss jedes Unternehmen das für sich am besten geeignete Vorgehen finden. Auch an dieser Definition ist der Software-Architekt wesentlich beteiligt.

Basierend auf einer V-Modell-artigen [6] Darstellung zeigt Bild 4 dazu ein mögliches Entwurfsvorgehen für die Entwicklung eines kompletten Embedded-Systems, also nicht nur für Software. In den Analyseaktivitäten identifizieren und dokumentieren die Analysten (real betrachtet meist auch die Architekten) der einzelnen Ebenen die entsprechenden Anforderungen (WAS). Darauf basierend entstehen die Architekturen (WIE). Auf den Subsystemarchitekturen basierend entwickelt der Software-Architekt die Software-Architektur für ein Subsystem - in Abstimmung mit den anderen Entwicklungsdomänen auf gleicher Ebene (z.B. Hardware-Entwicklung).

Parallel zur Entwicklung der Anforderungen entwickelt das Testteam die Testfälle, um später die korrekte Umsetzung nachzuweisen. Dies erfolgt ebenfalls auf den unterschiedlichen Ebenen. „Design for Test“ ist hierzu ein Software-Architekturthema.

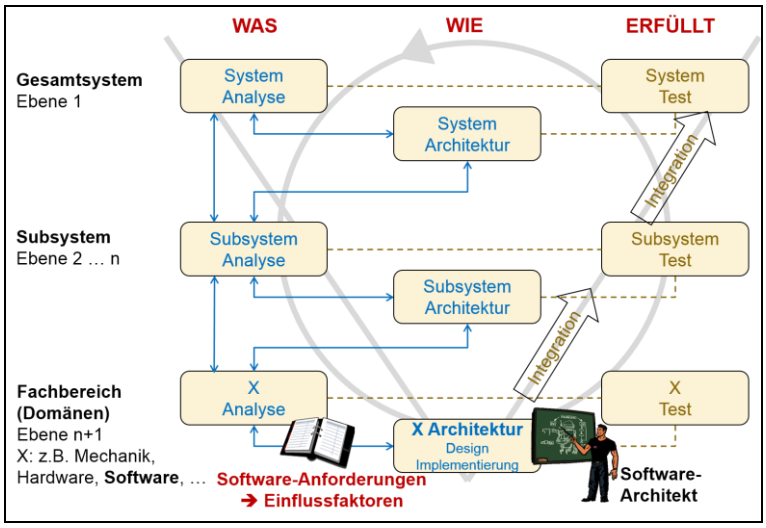

Bild 4: Entwurfsvorgehen für Embedded-Systeme

6. Entwurfsbasis und Einflussfaktoren

Aus der in Bild 4 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 (z.B. Portierbarkeit, Wartbarkeit, Zuverlässigkeit, Sicherheit, … aber auch Ressourcenverbrauch, Performance, Echtzeitfähigkeit, …), die die Software erbringen muss. 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 sind die Software-Anforderungen und die daraus resultierenden Einflussfaktoren neben der Subsystem-Architektur die wichtigste Entwurfsbasis für die Software-Architektur.

7. Kommunikation und Dokumentation

Für das Projekt und auch für den weiteren Fortbestand des Unternehmens ist es wichtig, dass der Software-Architekt die Software-Architektur für alle Stakeholder verständlich dokumentiert. Die Dokumentation repräsentiert gleichzeitig die Kommunikationsbasis, um sich kontinuierlich mit den Stakeholdern abzugleichen. 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.

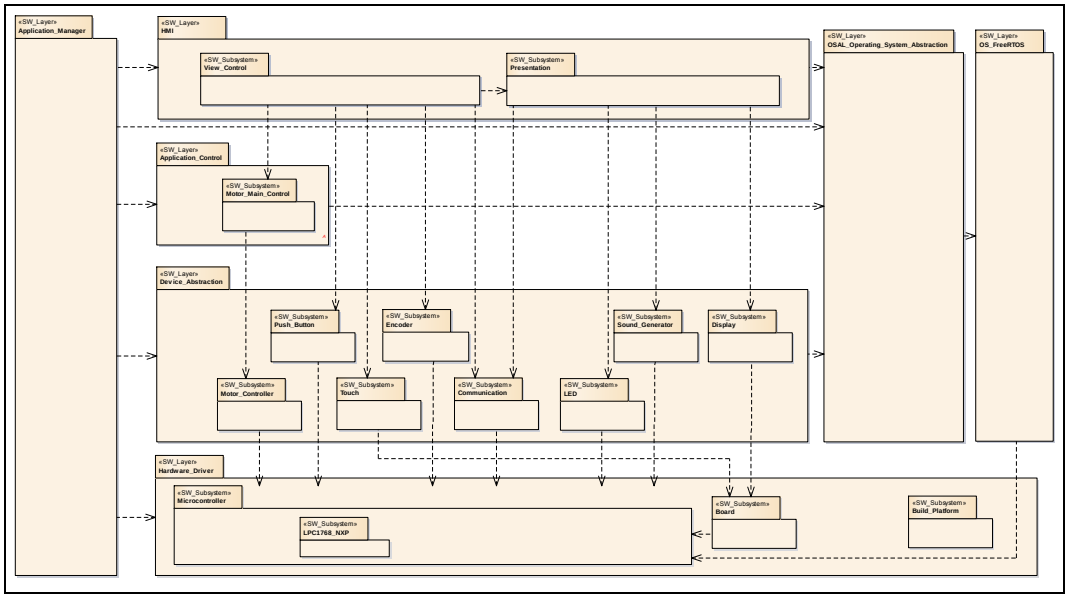

Bild 5: Exemplarisches Beispiel einer Software-Schichtenarchitektur

UML (Unified Modeling Language) [7] ist die Notation, um verschiedenste Sichten und Aspekte der Software-Architektur zu dokumentieren, im Design zu verfeinern - bis hin zur automatischen Codegenerierung. Mit dem Paketdiagramm sind in Bild 5 Software-Schichten modelliert.

8. Entwurfsprinzipien

Unser gesamtes Leben ist bestimmt von Regeln, auch wenn manche meinen, sich nicht daran halten zu müssen. Bestimmt haben Sie früher als Kind oder heute mit Ihren Kindern Lego gespielt. Auch hier gibt es Regeln, wie die Steine zusammenpassen.

Bild 6: Entwurfsprinzipien

Der Software-Architekt entwickelt mit immer neuem Wissen seinen Software-Architektur Style Guide. Darin beschreibt er seine Regeln, nach denen er Software-Architekturen entwickelt.

Nicht für alle Architekturen können alle Regeln gelten – die Anwendung ist abhängig von den Anforderungen. Die Anwendung der Regeln auf eine Software-Architektur verbessert die Softwarequalität.

Ein Architektur-Entwurfsprinzip ist beispielsweise das der hohen Kohäsion. Ziel hierbei ist es, logisch Zusammengehöriges in einem Architekturelement zu bearbeiten und gleichartige Aufgaben nicht auf mehrere Architekturelemente zu verteilen (vermeiden von Redundanzen) ➔ hohe Kohäsion.

Inzwischen existieren Veröffentlichungen zu Entwurfsprinzipien [4], die für Embedded-Software-Architekturen anwendbar sind. In der Praxis setzt der Software-Architekt die Entwurfsprinzipien mit Software-Architekturpatterns um.

9. Architekturentwicklung und Architekturpatterns

Mit 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).

Bild 7: Patterns

Um bei der Analogie mit Lego zu bleiben: Eine Problemstellungen wäre es, runde Konturenverläufe mit rechteckigen Bausteinen zu bauen. Eine Lösung dazu ist, die Steine je 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 5 ist eine nicht strikte Software-Schichtenarchitektur dargestellt. Nicht strikt bedeutet, dass 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.

10. Qualitätssicherung und Qualitätsbewertung

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

Bei der Qualität ist zwischen interner Qualität (z.B. Software-Architektur) und externer Qualität (das sieht der Kunde) zu unterscheiden. Die Prozessqualität beeinflusst maßgeblich auch die Produktqualität.

Um auch hier die Analogie zu Lego nochmals zu strapazieren – alle Legobausteine müssen so zusammengebaut sein, das sie die Konstruktion tragen, sonst fällt sie spätestens bei Erweiterungen in sich zusammen.

Übertragen gilt das auch für die Software-Architektur. Sie muss die zuvor definierten Qualitätsmerkmale und Funktionalitäten erfüllen. Häufig muss die Software-Architektur 20 Jahre und länger tragfähig bleiben.

Bild 8: Qualitätssicherung und Bewertung

Die einfachste qualitätssichernde Maßnahme für die Software-Architektur ist die Durchführung von Reviews mit anderen Architekten und Stakeholdern. Hier wird bewertet, ob die Architektur den geforderten Qualitätsmerkmalen gerecht wird oder nicht. Als Basis für das Review eignet sich die Software-Architektur-Dokumentation 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 gedanklich 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 – ATAM (Architecture Tradeoff Analysis Method) [8].

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

11. Einsatz von Tools

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

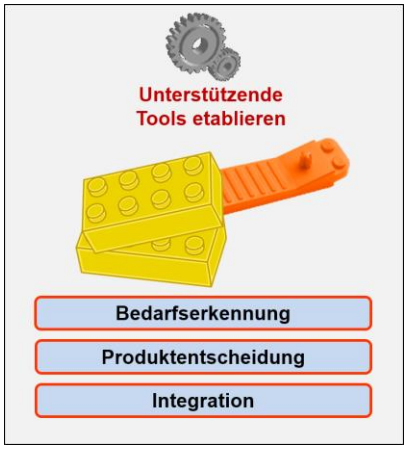

Bild 9: Einsatz von Tools

Er kennt den Toolmarkt, identifiziert den Bedarf, entwickelt Toolanforderungen, evaluiert Tools und wählt sie final fachlich aus. Gibt es im Unternehmen keine Toolgruppe, ist er auch für die Toolintegration verantwortlich.

Die Tools sollen allen in der Softwareentwicklung und darüber hinaus die Arbeit erleichtern. Ein wenig Egoismus schadet nicht, somit profitiert (in erster Linie) der Software-Architekt.

Toolthemen, denen sich der Software-Architekt annehmen könnte, sind:

- Anforderungsmanagement
- Versions- und Konfigurationsmanagement
- Modellierung
- Generierung von Dokumentation und Programmcode
- Buildsysteme
- Statische Analyse
- Dynamische Analyse

12. 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 Style Guide, 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.

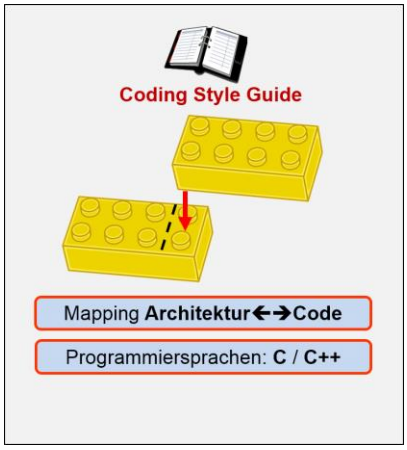

Bild 10: Coding Style Guide

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.

13. Praxisbewährte Architekturbeispiele

Der Begriff Embedded-System abstrahiert eine unendliche Produktvielfalt. Nahezu jedes Embedded-System enthält eine sehr spezifische Hardware und somit ebenfalls eine sehr spezifische Software und Software-Architektur. In der Software liegt oft die Kernkompetenz des Unternehmens. Das ist mit ein Grund dafür, dass es im Bereich Embedded nur wenige und wenn nur oberflächliche Veröffentlichungen zum Thema Software-Architektur gibt. Es existieren nur eine begrenzte Anzahl von Standards und nahezu keine standardisierten Softwarekomponenten. Branchenspezifische Standardisierungsversuche erfreuen sich leider nur einer mittleren Beliebtheit und kommen teilweise nicht einmal zum Einsatz.

Wieder einmal ist der Software-Architekt auf sein Wissen und seine Erfahrung angewiesen. In einem kontinuierlichen Lernprozess lernt der Software-Architekt mit jedem neuen Projekt dazu.

Aus diesen Gründen neigt der Embedded-Bereich dazu, das Rad immer wieder neu zu erfinden und so eine extrem hohe Engineering-Blindleistung zu erzeugen.

Software-Architekten im IT-Bereich haben es mit standardisierter Hardware und standardisierten Softwarekomponenten wesentlich einfacher.

14. Nicht-Fachwissen (Softskills)

Irgendwann ist es soweit: Der Software-Architekt verlässt sein Büro und interagiert mit anderen Rollen im Unternehmen. Sein Auftritt ist dabei selbstbewusst, aber nicht überheblich.

Bild 11: Nicht-Fachwissen (Softskills)

Hierfür greift er auf sein Nicht-Fachwissen (Softskills) zurück und beherrscht die folgenden Themen:

- Kommunikation
- Team-Building
- Konfliktlösung
- Coaching
- Führung

15. Beruflicher Werdegang mit Belohnung

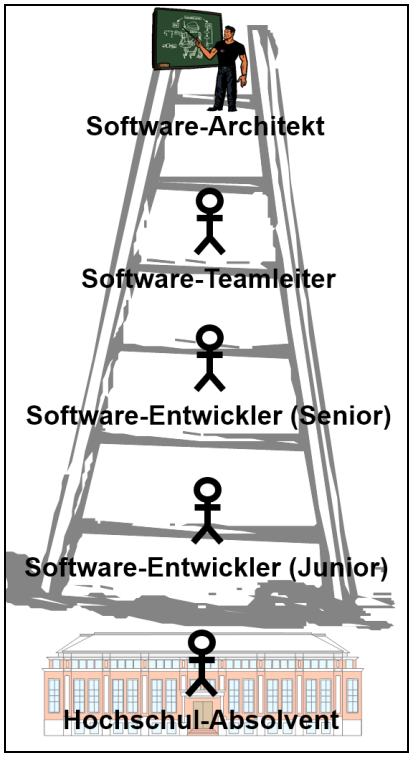

Bild 12: Beruflicher Werdegang eines Software-Architekten

Der idealisierte berufliche Werdegang eines Software-Architekten könnte wie folgt aussehen:

Nach seinem Hochschulstudium hat er verschiedene Stellen in der Embedded-Softwareentwicklung, in verschiedenen Unternehmen mit unterschiedlichen Produkten. Dabei erweitert er kontinuierlich sein fachliches Wissen.

Anschließend leitet er Software-Teams und erlernt dabei das Nicht-Fachwissen (Softskills).
Nun hat er die idealen Voraussetzungen für die Rolle des Software-Architekten. Der Lernprozess endet hier jedoch nicht, und die Person selbst muss auch wirklich Spaß an ihrer Arbeit haben.

Letzen Endes sollte sich das Ganze auch bezahlt machen – im wahrsten Sinne des Wortes. In vielen Unternehmen ist die Rolle des Software-Architekten sehr angesehen und auch sehr gut bezahlt. So wie die im Bild 12 dargestellte Karriereleiter nach oben geht, sollte sich auch das Gehalt entwickeln.

16. Resümee

Der Software-Architekt verantwortet die gesamte Software sowie deren Entstehungsprozess und trifft dabei wichtige Entscheidungen.

Für den Software-Entwickler ist er Ansprechpartner für alle fachlichen Fragen, die sich während Design und Implementierung ergeben.

Der Software-Architekt berät den Projektmanager bei der Teamzusammenstellung und hilft ihm, eine akkurate und aktuelle Projektplanung abzugeben.

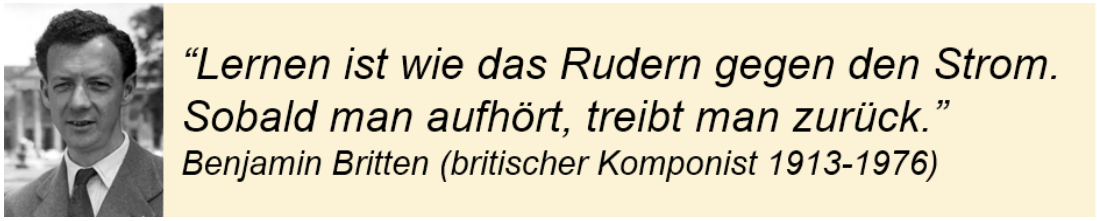

Bild 13: Ein für Software-Architekten relevantes Zitat

17. Referenzierte und weiterführende Links:

[1] MicroConsult-Download für komplette und aktuelle Dokumentation
http://download.microconsult.net/ese2016/architekt.zip

[2] MicroConsult-Download: Betriebssysteme: mit oder ohne?
http://download.microconsult.net/ese2015/laufzeitarchitektur.zip

[3] MicroConsult Download: OSAL mit C++
http://download.microconsult.net/ese2014/osal.zip

[4] MicroConsult Download: Prinzipien für Embedded-Software-Architekturen
http://download.microconsult.net/ese2014/archprinz.zip

[5] MicroConsult-Training & Coaching:

Moderne Software-Architekturen für Embedded- und Echtzeitsysteme

RTOS-Mechanismen und deren Anwendungen in Laufzeitarchitekturen von Embedded- und Echtzeitsoftware

[6] V-Modell
http://www.cio.bund.de/Web/DE/Architekturen-und-Standards/V-Modell-XT/vmodell_xt_node.html

[7] Object Management Group (OMG)
http://www.omg.org

[8] Software Engineering Institute | Carnegie Mellon University
ATAM (Architecture Tradeoff Analysis Method)
http://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm

18. Autor:

Dipl.-Ing. (FH) Thomas Batt ist gebürtiger Freiburger. Nach seiner Ausbildung als Radio- und Fernsehtechniker studierte er Nachrichtentechnik in Offenburg. Seit 1994 arbeitet er kontinuierlich in verschiedenen Branchen im Bereich Embedded-/Real-Time 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.