"View"-basierte Softwarearchitektur-Dokumentation

Sinnvoll dokumentieren mittels verschiedenen Views

Matthias Künzi, visuellklar

Eine Software Architektur sinnvoll zu dokumentieren bedeutet, dass die relevanten Informationen eines Software Systems dokumentarisch so festgehalten werden, dass diese zu gegebener Zeit wieder abgerufen werden können. Was sind jetzt aber die relevanten Informationen und wie halte ich diese so fest, dass der Aufwand und der Nutzen in einem sinnvollen Verhältnis stehen? Genau mit dieser Thematik befasst sich der folgenden Artikel.

Ausgehend vom Golden Circle, werde ich das Thema Software Architektur Dokumentation in 3 Schritten beleuchten. Ich werde mit dem "Warum" beginnen. Warum sollte man Architektur überhaupt dokumentieren. Danach werde ich auf das "Wie" eingehen. Hier ist die View basierte Dokumentation das Hauptthema. Zum Schluss werde ich auf das "Was" eingehen und aufzeigen was denn genau dokumentiert werden sollte. Ich hoffe damit eine umfassende Darlegung dieser Thematik zu erreichen.

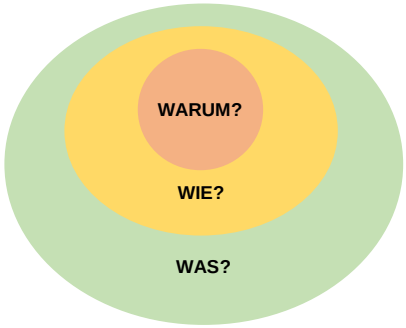


Abb. 1: "The golden circle"

Warum?

Software Architektur dokumentieren – warum? Was ist der Nutzen einer solchen Dokumentation?
Es gibt aus meiner Sicht und Erfahrung die folgenden 2 Haupt Use-Cases, bei welchen eine Architektur Dokumentation ein essentielles Hilfsmittel ist:

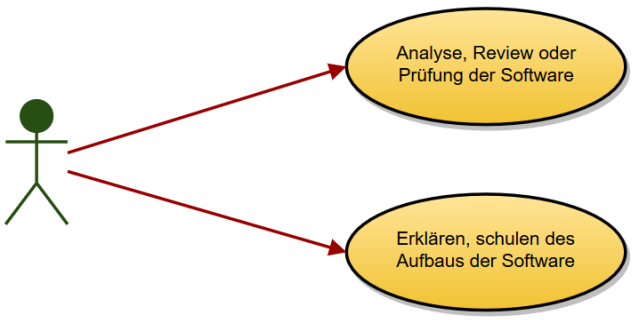


Abb. 2: Haupt Use-Cases für die Erstellung einer Software Architektur Dokumentation

1. Analyse, Review oder Prüfung der Software
   z.B. für folgende Aufgaben:
   a. Architekturreview, bevor das System gebaut wird (Earliest Artefact to Analyse)
   b. ATAM Analyse zur Sicherstellung der auferlegten Qualitätsanforderungen.
   c. Aufzeigen der Erfüllung von regulatorischen Auflagen (Medtech, Safety,..)
   d. Impactanalyse einer Software Änderung (zusätzliche oder geänderte Anforderungen). Wartung eines Softwaresystems.
   e. …

2. Erklären, schulen des Aufbaus der Software
   z.B. für folgende Szenarien:
   a. Erklären des geplanten Aufbaus des Systems innerhalb des Entwicklerteams. Sicherstellen, dass das Softwaresystem so wie geplant gebaut wird.
   b. Einarbeitung neuer Projektmitarbeiter
   c. Ableiten der Aufwände zur Entwicklung des Systems (Work Estimation).
   d. Ableiten geeigneter Schnittstellen für die Abarbeitung (Work Assignment)
   e. Aufzeigen der gemachten Design Entscheidungen. Verhindern, dass die gleichen Fragestellungen später wieder aufgeworfen werden.
   f. …

Es gibt also einen klaren Nutzen einer Architektur-Dokumentation.
Zusammenfassend ist die Dokumentation einer Architektur ein wichtiges Kommunikationsvehikel im LifeCycle eines Softwaresystems.

Was aber ist mit dem Verhältnis Aufwand zum Nutzen? Wieviel Aufwand lohnt sich in eine Dokumentation zu investieren? Das ist eine sehr gute und berechtigte Frage. Obwohl man sich die oben aufgeführten Use Cases gut vorstellen kann, sind diese noch lange nicht für jedes System relevant. Es lohnt sich also in jedem Fall den konkreten Nutzen dem Aufwand gegenüberzustellen.
Das heisst diese Formel anzuwenden:

$$A = \sum Kosten\ f\ddot{u}r\ Erstellung\ SAD - \sum Zusatzkosten\ f\ddot{u}r\ Use\ Cases\ ohne\ SAD$$

Wenn diese Formel eine negativen Wert ergibt, dann lohnt sich der Aufwand für die Erstellung einer Dokumentation. Allerdings muss man hier den gesamten LifeCycle eines Systems betrachten. Eine sinnvolle Architektur Dokumentation wird im gesamten LifeCycle immer wieder gebraucht werden.
Insbesondere ist eine geeignete Software Architekturdokumentation nach der Erstentwicklung sehr hilfreich. Und wenn man das Verhältnis zwischen

Initialentwicklung und Wartungs- und Weiterentwicklungsaufwand eines typischen Softwaresystems betrachtet, dann umso mehr:

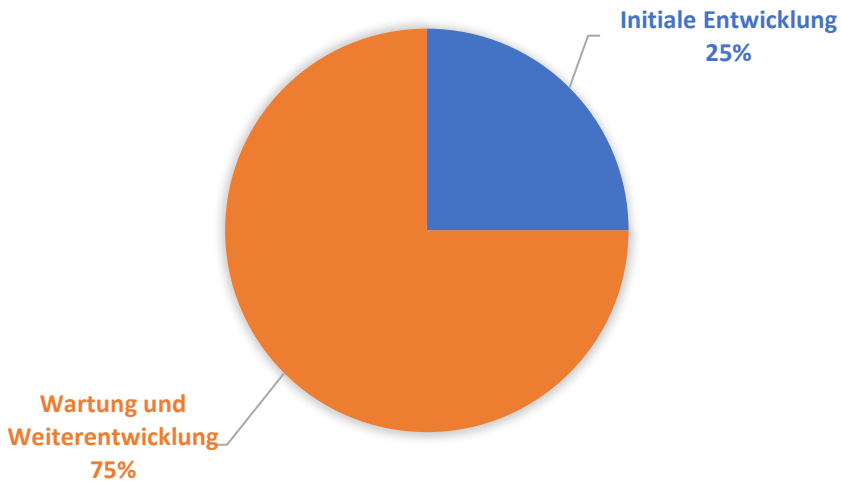


Abb. 3: Totale Kosten / Aufwände aufgeteilt in Initiale Entwicklung und Wartung und Weiterentwicklung

«Working Software over comprehensive documentation»
Dieser agile Grundsatz aus dem Manifest ist richtig und wichtig. Er sagt aber nicht aus, dass es keine Dokumentation geben soll. Die Meinung hier ist aber, dass es genau um die richtige Balance zwischen Aufwand und Nutzen geht.

Es heisst ja nicht, dass keine Dokumentation erstellt werden soll, sondern nur, dass die laufende Software über der Dokumentation stehen soll – oder eben auch, dass eine sinnvolle Dokumentation eine Basis für laufenden Software sein soll.

working Software
over
comprehensive documentation

Abb. 4: Working Software over comprehensive documentation

Wie?
Wie sollte nun eine sinnvolle Dokumentation erstellt werden? Für eine Software Architektur Dokumentation wie auch für andere Dokumentationen gibt es ein paar sinnvolle Grundsätze:

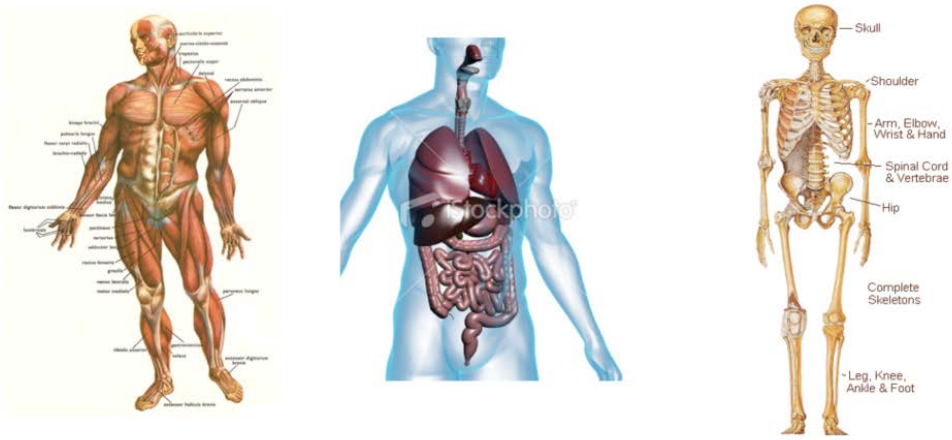
1. Schreibe für den Leser
2. Vermeide unnötige Wiederholungen
3. Vermeide Mehrdeutigkeiten und Unklarheiten
4. Verwende, wenn immer möglich eine Standard Struktur der Dokumentation
5. Dokumentiere auch Hintergründe und Begründungen
6. Stelle sicher, dass die Dokumentation aktuell ist – aber vermeide unnötige und häufige Aktualisierungen
7. Lass die Dokumentation überprüfen

Auf die Punkte 1., 3, sowie 4 möchte ich in der Folge genauer eintreten:

1. Schreibe für den Leser.
Das bedeutet auch, man sollte die Stakeholder seiner Dokumentation kennen. D.h. die Nutzer. Es lohnt sich also hier eine Stakeholder-Map zu machen und sich zu überlegen, welche Informationen jeder Stakeholder haben sollte.

Da verschiedene Stakeholder unterschiedliche Informationsbedürfnisse haben, macht es Sinn, verschiedene Sichten zu erstellen und dadurch eine umfassende Stakeholder-basierte Dokumentation zu erstellen.

Das folgende Beispiel aus der Anatomie soll die Begriffe View (Ansicht), Stakeholder (Interessengruppe) und Concerns (Interessen) veranschaulichen:



View	Muscle View	Organs View	Skeleton View
Stakeholder	Physical Therapist	Internist	Orthopedist
Concerns	Muscles and ligaments	Organs and their functions	Bones and their functions

Abb. 5: View, Stakeholder und Concerns

Im Folgenden ein Auszug aus einem konkreten Beispiel einer Architektur View:

Viewpoint	Concerns addressed	Elements, relations, properties, and constraints	Applicable evaluation/analysis techniques
Risk Classifications	– Risk classification according to EN62304 – Segregation between different safety classes	Elements: – Software elements with assigned risk classification Relations: – Connections between software elements that are relevant for segregation inspection	– Review

Abb. 6: Auszug aus einer Architektur View – View Definition

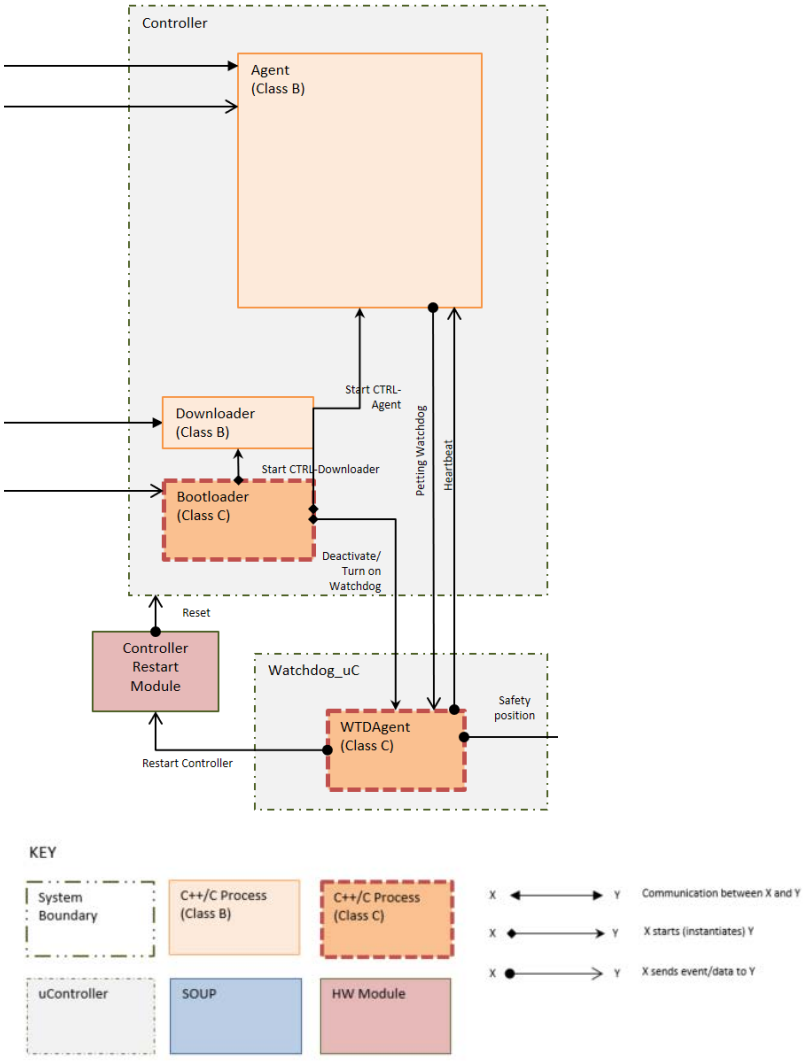


Abb. 7: Auszug aus einer Architektur View

3. Vermeide Mehrdeutigkeiten und Unklarheiten

Für eine gute Dokumentation ist Klarheit essentiell. Speziell für eine SW Architektur Dokumentation kann man viel Klarheit mit Bilder, d.h. Diagrammen erreichen. Allerdings ist hier Vorsicht geboten. Diagramme müssen klar verständlich sein. Hier helfen entweder Standarddiagramme wie UML zu verwenden – oder mit klaren Legenden die verwendeten Symbole und Diagrammelemente zu erläutern. Das folgende Beispiel eines Diagrammes aus einer Architektur View zeigt wie das aussehen könnte:

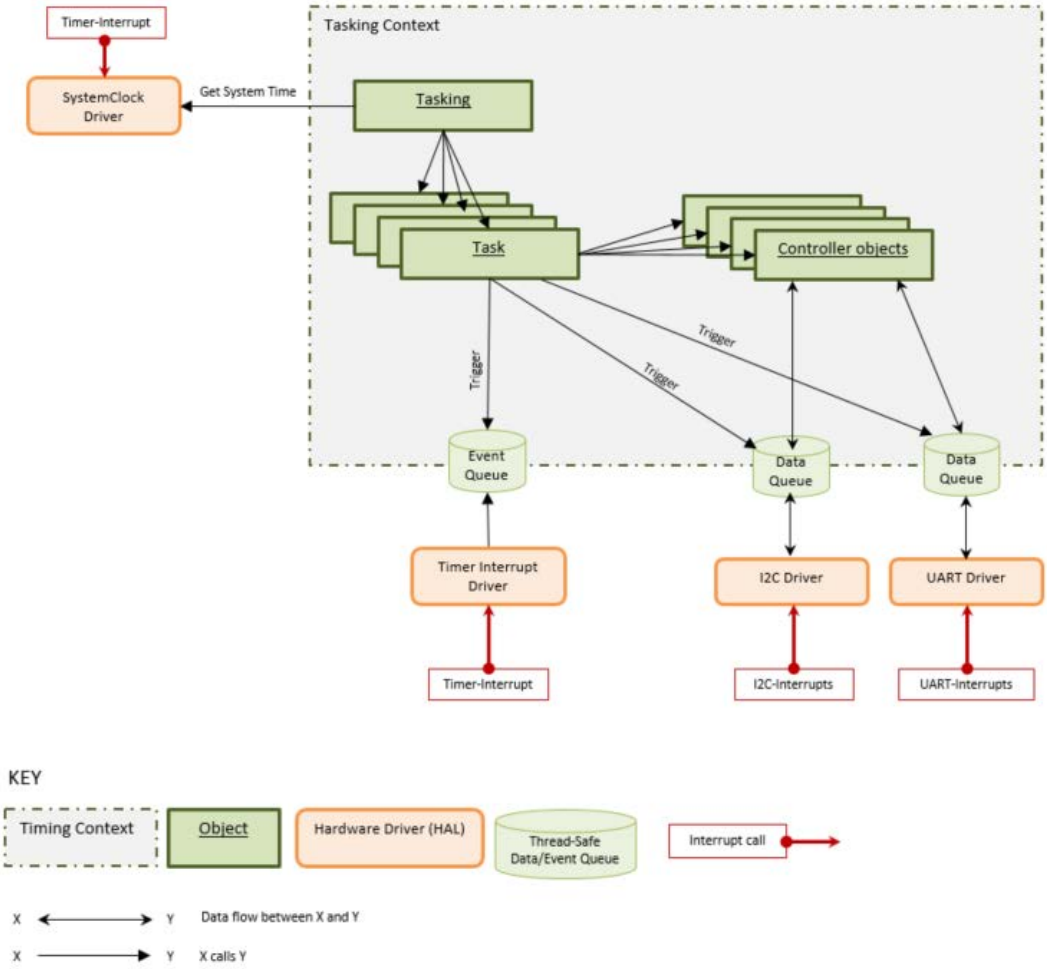
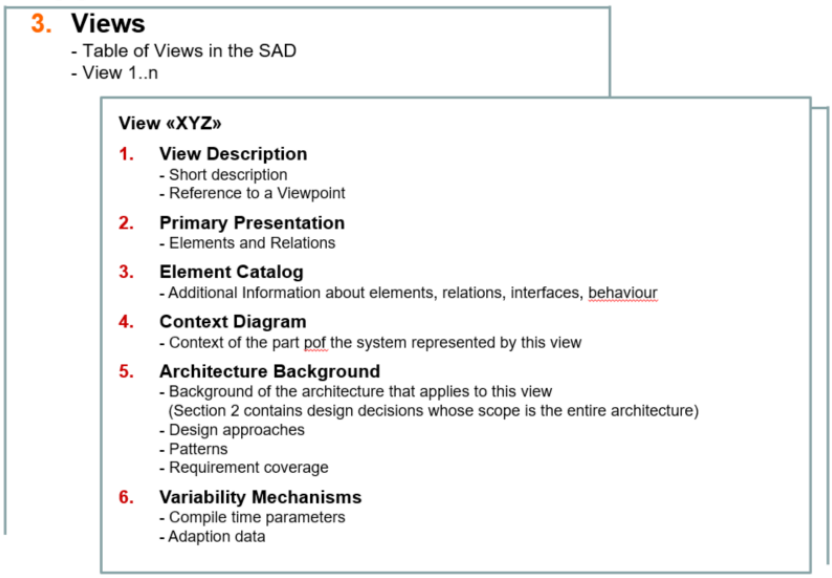


Abb. 8: Architektur Diagramm mit einer klaren Legende

4. Verwende, wenn möglich eine Standard Struktur der Dokumentation
Eine klare immer wieder verwendete Struktur einer Dokumentation hilft enorm. Die folgenden Strukturen für eine View sowie für eine komplette Software Architektur Dokumentation haben sich bewährt (Quelle SEI):

1. **Documentation Roadmap**
   - Purpose and Scope of the SAD
   - How the SAD Is Organized
   - Stakeholder Representation
   - Viewpoint Definitions
   - How a View is Documented
2. **Architecture Background**
   - Problem Background
   - System Overview
   - Architectural Approaches
3. **Views**
   - Table of Views in the SAD
   - View 1..n
4. **Relations among Views**
   - General Relations Among Views
   - View-to-View Relations
5. **Referenced Materials**
6. **Glossary**



Zu beachten ist hier, dass eine View nicht einfach ein Diagramm ist – sondern eine komplette View anhand obigem Template dokumentiert werden sollte.

Was?
Was genau sollte dokumentiert werden? Na klar, die "Software Architektur". Aber was ist das genau? Dazu lohnt es sich, die Definitionen von Architektur anzuschauen. Insbesondere der Unterschied zwischen Design und Architektur. Das gibt nämlich neben den benötigten Informationen der Stakeholder eine gute Entscheidungshilfe was dokumentiert werden soll und was nicht.

„The set of structures needed to reason about the system, which comprises software elements, relations among them, and properties of both. The definition emphasizes the plurality of structures present in every software system. These structures, carefully chosen and designed by the architect, are the key to achieving and reasoning about the system’s design goals. And those structures are the key to understanding the architecture.”
(Aus „Documenting Software Architectures” Views and Beyond (2nd Edition), Clements et al, Addison-Wesley, 2010)

„All architecture is design but not all design is architecture. Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change.” (Grady Booch)

Genau diese Definitionen sind aus meiner Erfahrung zu Hilfe zu nehmen um sicherzustellen, dass die notwendige (=essentielle) Dokumentation vorhanden ist und aber dass nicht alle Details dokumentiert werden. Zudem ist nach diesen Definitionen auch sichergestellt, dass diese Art der Dokumentation einen langen Bestand hat und nicht dauernd aktualisiert werden muss. Also genau das Ziel, was erreicht werden sollte.

Zusammenfassung:

1. Aufwand und Nutzen einer Architektur Dokumentation sollte in einem sinnvollen Verhältnis stehen.
2. Verwende Views um eine Stakeholder-basierte Architektur zu erstellen.
3. Beachte eine View ist nicht einfach ein Diagramm, sondern enthält genau die Information, welcher für die entsprechenden Stakeholder relevant sind.
4. Stelle sicher, dass die gesamte Dokumentation insbesondere Architekturdiagramme klar und unmissverständlich sind. Verwende eine Diagrammlegende, wenn nötig.
5. Dokumentiere die Architektur relevanten Teile und nicht Design- und Implementationsdetails.

Literatur- und Quellenverzeichnis:

- Documenting Software Architectures: Views and Beyond (2nd Edition) von Paul Clements, Felix Bachmann, Len Bass , David Garlan und anderen. Publisher: Addison-Wesley Professional; 2 edition (October 15, 2010)
- 4+1 architectural view model (Philipp Kruchten) https://en.wikipedia.org/wiki/4%2B1_architectural_view_model
- ISO/IEC 42010 https://en.wikipedia.org/wiki/ISO/IEC_42010

Autor

Matthias Künzi (Dipl. El. Ing. HTL, Dipl. Software Ing. FH) arbeitet seit 2015 bei Belimo AG. In seine jetzige Tätigkeit als „Head of Software“ bringt er über 20 Jahre Erfahrung in die Umsetzung von kritischen Softwaresystemen im Embedded Umfeld mit ein. Mit seiner eigenen Firma „visuellklar“ berät und unterstützt Matthias Künzi Firmen bei der Umsetzung komplexer Problemstellungen im Projekt- und Softwareumfeld. Dabei kommen vor allem visuelle und agile Methoden zum Einsatz.

Kontakt

Internet: www.visuellklar.ch
Email: matthias.kuenzi@visuellklar.ch