# Beschreibung von Architekturen mit der UML

Assignment zum Modul:

Softwareentwicklung (SWE23)

28.05.2020

Vladimir Zhelezarov

.....

*Studiengang*: Digital Engineering und angewandte Informatik - Bachelor of Engineering

AKAD University

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# 1 Einleitung

Die Entwicklung eines Software-Systems kann verschiedene Ansätze oder Modelle folgen, kann aber auch komplett vom Grund aus innovativ entstehen. Unabhängig davon steht immer am Anfang die Konzipierung des Grunddesigns oder anders gesagt die Architektur. Egal in welcher Form die Entwicklung stattfindet, ist eine der Hauptaufgaben des Architekten, seine Ideen nach außen verständlich zu kommunizieren. Mit der Größe des Projekt steigt noch mehr die Wichtigkeit der Dokumentation. Nicht nur erhöht sich die Anzahl der Beteiligten, sondern sie haben auch verschiedene Blickwinkel auf das Projekt und verfolgen zum Teil auch unterschiedliche Ziele. Um die Software-Architekten, Programmierer, Benutzer und andere Beteiligten eine gemeinsam verständliche Sprache reden zu lassen, wurden Standarte entwickelt, wie z.B. die hier Betrachteten für die Beschreibung einer Software-Architektur und für die UML-Sprache.

In dieser Arbeit wird die Beschreibung von Software-Architekturen nach diesen Standards in deren aktuellen Form diskutiert, sowie auch die Rolle der UML-Sprache dabei. Zusätzlich wird versucht eine Schätzung der aktuellen Situation in der Praxis zu gewinnen.

Grundlage für alle Betrachtungen sind zuerst die einheitliche Definition der benutzten Begriffe. Hierzu kommen wieder die Standards in Frage, weil das genau eins derer Ziele ist - Begriffe zu vereinheitlichen.

Mit diesen Begriffen diskutieren wir ein allgemeinen Rahmen für die Architekturbeschreibung, wie es vom aktuellen OMG-Standard empfohlen wird. Auch wenn die Vielfalt der Architekturen praktisch unbegrenzt ist, kann immer noch ein Grundgerüst für deren Beschreibung benutzt werden, um die Verständlichkeit zu verbessern.

Wir betrachten die Gründe warum UML ein hervorragender Kandidat für Architekturbeschreibungssprache ist und ihre Möglichkeiten mit dem vorgeschriebenen Grundgerüst zu arbeiten.

Die vorgestellten Konzepte können wir praktisch anwenden, indem wir die Architektur eines Open-Source Verschlüsselung Projekt mit UML beschreiben.

Wie die Konzepte in der Software-Community umgesetzt werden, können wir mithilfe einer Analyse der zehn Top-Projekte im populären Open-Source Hosting Dienst GitHub herausfinden. Man kann vermuten, dass, wenn so viele Beteiligte dabei sind, auch eine andere Betonung auf die Dokumentation der Architektur gesetzt wird. Ob und inwieweit das so ist wird sich in der Analyse zeigen.

# 2 Grundlagen

## 2.1 Software-Architektur

Nach dem aktuellen ISO 42010 Standard ist der Begriff Software-Architektur definiert wie folgt[1]:

> „Die grundlegenden Konzepte oder Eigenschaften eines Systems in ihrer Umgebung, ausgedrückt in ihren Komponenten, Zusammenhänge und Prinzipien ihrer Konstruktion und Weiterentwicklung." (Übersetzung durch den Verfasser)

Eine Folge dieser Definition ist, dass die Architektur kein Zusatz-Artefakt bei der Software ist, sondern immer gegeben ist und existiert, auch wenn sie nicht explizit definiert oder beschrieben ist. Als solche ist sie auch von Grundbedeutung für das Verstehen des Software-Systems[2].

## 2.2 Weitere Begriffe

Der ISO 42010 Standard definiert zusätzlich folgende Begriffe[3]:

- Sicht: Arbeitsprodukt, welches die Architektur eines Systems aus der Perspektive einer spezifischen Angelegenheit darstellt;
- Standpunkt: Arbeitsprodukt, das die Konventionen für die Konstruktion, die Interpretation und die Benutzung von der Sichten spezifischer Angelegenheiten festlegt;
- Beteiligten: eine Person, ein Team, eine Organisation oder Klassen davon, die Interesse an einem System haben;
- Angelegenheiten: Interesse an einem System, relevant für einer oder mehreren Beteiligten.

(Übersetzung durch den Verfasser)

## 2.3 Bedeutung der Dokumentation von Software-Architekturen

Es ist bekannt, dass der größere Anteil der Arbeit bei der Software daraus besteht, existierende Anwendungen zu warten oder zu verbessern[4]. Bei fehlender Dokumentation ist nicht nur die Arbeit dabei sehr schwierig - das s.g. Reverse-Engineering - sondern es entsteht

[1] ISO (2011), S.2
[2] Goll, J. (2011), S.77
[3] ISO (2011), S.2
[4] Vgl. Endres, A., Rombach, D. (2003), S. 161

auch schnell der Effekt der Architektur-Erosion - die ursprüngliche Architektur verliert immer mehr an Relevanz, je älter die Anwendung wird[1]. Die Dokumentation gehört zu den wichtigsten Aufgaben des Software-Entwicklers[2].

## 2.4 Unified Modeling Language

UML steht für Unified Modeling Language, eine Modelierungssprache, die von OMG (Object Management Group)[3] standardisiert wurde, mit dem Ziel[4]:

> „...um Systemarchitekten, Software-Ingenieure und Software-Entwicklern Werkzeuge für die Analyse, Design und Implementierung von Software-Systeme sowie für die Modellierung Geschäftsprozesse oder andere Prozesse anzubieten“ (Übersetzung durch den Verfasser)

UML besteht aus Diagrammen. Ein Diagramm stellt die Projektion eines Modells eines Systems aus einer bestimmten Perspektive dar[5]. Die aktuelle 2.5.1 Version bietet folgende Diagramme an[6]:

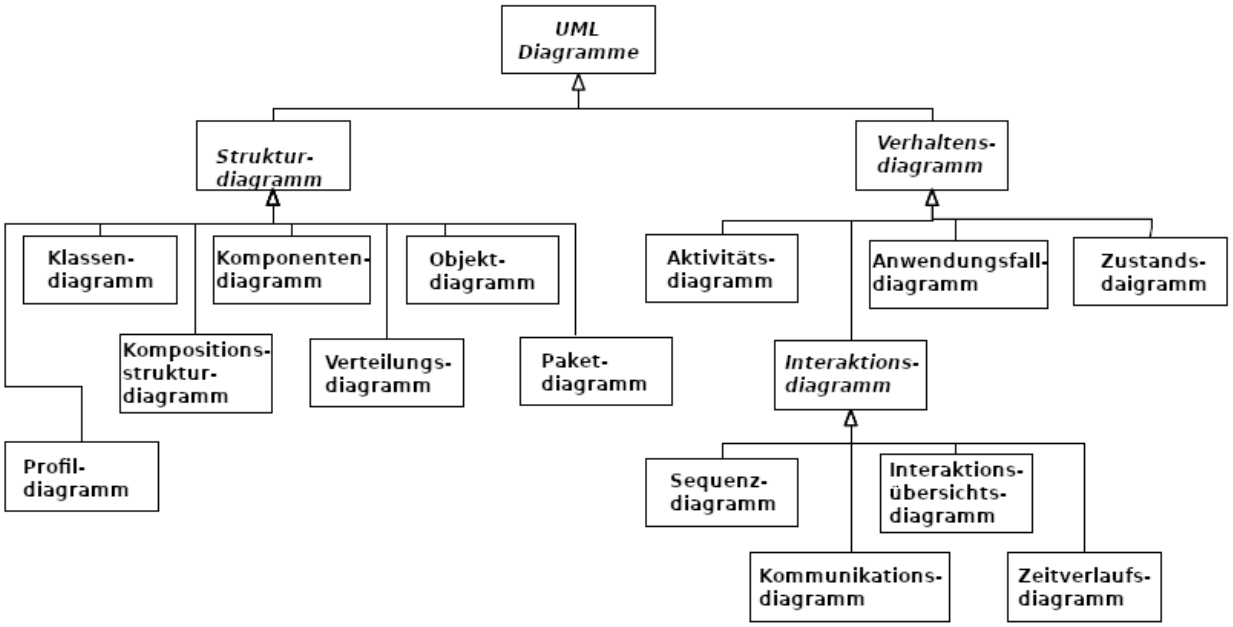


Abbildung 1: UML 2.5.1 Diagramme

Die UML ist ausreichend formal, um z. B. die automatisierte Synthese von Code-Rahmen

[1] Vgl. Jahnke, J. (2009), S.199-200
[2] Hoffmann, D. (2013), S.141
[3] https://www.omg.org/ (Zugriff am 28.05.2020)
[4] OMG (2017), S.1
[5] Goll, J. (2011), S.317
[6] OMG (2017), S.685 (Übersetzung durch den Verfasser)

zu unterstützen und gleichzeitig informell genug, um eine hohe Verständlichkeit zu gewährleisten[1].

Der ISO 42010 Standard schreibt keine einzige Modelierungssprache vor, die Auswahl ist dem Entwickler überlassen[2]. Obwohl UML keine reine Architekturbeschreibung ist, wird sie häufig für die Beschreibung von SA eingesetzt[3]. In der Praxis hat sich UML als de-facto Standard für Diagramme durchgesetzt[4].

### 2.5 Linux Unified Key Setup - Second Version

LUKS - oder Linux Unified Key Setup, ist eine Open-Source Spezifikation für Management von Verschlüsselung auf Datenträger. Die aktuelle Entwicklungs-Version hat eine ausführliche Text-Architekturbeschreibung[5], die sich gut in UML-Diagramme veranschaulichen lässt.

## 3 Architekturbeschreibung - Grundaufbau

Die Beschreibung der Architektur ist grundsätzlich dem Entwickler überlassen, was zu einer praktisch unbegrenzter Vielfalt von Ansätzen führt. Die vorliegende Arbeit hält sich allerdings an dem ISO 42010 Standard, dieser empfiehlt folgender Inhalt bei der Architekturbeschreibung[6]:

1. Überschritten inkl. Autoren, Daten, Änderungen etc.;
2. Identifizierung von den Beteiligten und deren Angelegenheiten - Die Beteiligten können z.B. Benutzer, Käufer, Eigentümer, Entwickler sein, deren Angelegenheiten können z.B. Funktionalität, Durchführbarkeit, Nutzung, Eigenschaften, Verhalten sein;
3. Auswahl der Standpunkte;
4. Eine Sicht für jeden Standpunkt;
5. Zusammenhänge, eventuelle Inkonsistenzen wenn vorhanden;
6. Begründung für die Architekturentscheidungen.

[1] Hoffmann, D. (2013), S.145
[2] ISO (2011), S.5
[3] Behrens, J. et al. (2009), S.61
[4] Rumpe, B. (2011), S.2
[5] Brož, M. (2018)
[6] Vgl. ISO (2011), S.11

Wenn die anderen Punkte mehr oder weniger eindeutig sind, gibt es bei den Punkten 3. und 4. die meisten Meinungsunterschiede. Verschiedene Autoren empfehlen verschiedene Ansätze. Eine gemeinsame Schnittmenge für die Standpunkteauswahl lässt sich aber wie folgt bestimmen[1]:

- *Statischer Standpunkt*: aus diesem Standpunkt werden Sichten erstellt, die den Aufbau und die Zusammenwirkung der beteiligte Komponenten darstellen;
- *Dynamischer Standpunkt*: das ist der Standpunkt, woraus das Verhalten des System zur Laufzeit erkannt und beschrieben wird;
- *Verteilungs-Standpunkt*: hieraus wird die physikalische und organisatorische Verteilung der Komponenten erkannt.

Bei der Auswahl von Standpunkte und Sichten ist auch die persönliche Schätzung des Entwicklers von bedeutender Rolle. Das Hauptziel ist nicht sich strikt nach den Vorgaben zu halten, sondern mehr die Verständlichkeit zu verbessern, damit die Arbeit der Programmierer bei der Implementierung des Designs erleichtert wird[2]. Es ist auch nicht nötig, alle möglichen Informationsbedürfnisse der Beteiligten zu befriedigen. Dabei gilt als Faustregel, dass wenn 80% dieser erfüllt sind, dies gut genug ist, damit alle ihre Arbeit machen können[3].

# 4 Architekturbeschreibung mit UML

Von den 15 Diagrammen, die UML 2.5.1 anbietet, sind manche häufiger benutzt als andere. In diesem Kapitel betrachten wir die wichtigsten und meist benutzten Diagramme unter Beachtung der entsprechende Standpunkte. Da UML eine Diagrammensprache ist, werden die dargestellten Konzepte gleich mit konkreten Diagrammen veranschaulicht. Die vorhandenen Diagrammen beschreiben die Architektur von LUKS2 (Linux Unified Key Setup) in der aktuell entwickelten Version 1.0.0.

Grundsätzlich lassen sich die Struktur-Diagramme gut für Sichte aus dem statischen Punkt benutzen, mit der Ausnahme vom Verteilungsdiagramm, das wie der Name vermutet, bestens für den Verteilungs-Standpunkt geeignet ist. Die Verhaltensdiagramme bedienen dementsprechend den dynamischen Standpunkt.

Oftmals sind mehrere Diagramme für einen bestimmten Zweck geeignet. Es ist dann eine Frage des persönlichen Geschmacks, welches genau benutzt werden soll[4].

[1] Vgl. Behrens, J. et al. (2009), S.36-37
[2] Vgl. Fowler, M. (2003), S.16
[3] Bass, L. et al. (2013), S.343
[4] Fowler, M. (2003), S.101

## 4.1 Statischer Standpunkt

Der statische Standpunkt ist der innerhalb der UML am breitesten vertretene Teil[1]. Dazu zählt unter anderem das wichtigste und am meisten benutzte UML Diagramm - das Klassendiagramm[2], das hier auch für die LUKS2 Architektur gezeigt wird.

Ohne bei allen beteiligten Elemente in den Einzelheiten zu gehen, bietet sich hier an, die Komposition, die Hierarchie und die Zusammenhänge der LUKS2 Architektur zu zeigen. Auch wenn es um eine Datei-Struktur geht, können die einzelne Elemente als Objekte betrachtet werden, was eine verständliche Darstellung mit Klassen erlaubt. Die Komposition, im Sinne von „besteht aus“ bezeichnen wir mit einer gefüllter Raute. Die „hat ein“ Beziehung mit einer leeren Raute benutzen wir für die Einzelheiten der beinhalteten Objekte. Vererbung, mit dem leeren Pfeil, zeigt dass ein Objekt verschiedene Möglichkeiten hat, implementiert zu werden. Bei den Objekten, wo das nötig und von der Spezifikation verlangt wird, sind auch die enthaltenen Felder nach dem Muster „Name:Typ“ dargestellt.

## 4.2 Dynamischer Standpunkt

Der dynamischer Standpunkt stellt das System während der Benutzung dar. Für die LUKS2 Architektur wurden hier Aktivitätsdiagramme ausgewählt. Es sind natürlich auch andere Möglichkeiten denkbar, wie z.B.:

- *Anwendungsfalldiagramm*: die verschiedene Szenarien, die bei der Arbeit mit der Software vorstellbar sind;
- *Interaktionsübersichtsdiagramm*: das Aussehen ist etwas anders, der Sinn ist allerdings dem Aktivitätsdiagramm ähnlich;
- *Zustandsdiagramm*: wie die Zustände vom Header und die Metadaten ineinander wechseln und unter welchen Bedingungen;
- *Sequenzdiagramm*: wie der zeitliche Verlauf für jede Aktivität abläuft.

Was die Beschreibung des Verhaltens angeht, sind die Diagramme mehr oder weniger ähnlich, da sie schließlich das selbe System beschreiben.

## 4.3 Verteilungs-Standpunkt

Ein typisches Beispiel, wo dieser Standpunkt besonders wichtig wäre, ist z.B. bei Web-Anwendungen mit Mikroservices. Es ist in diesem Fall für die Entwicklung und Implemen-

[1] Behrens, J. et al. (2009), S.38
[2] Fowler, M. (2003), S.25

tierung von Bedeutung, wie und wo die einzelnen Komponenten auf den Netzknoten verteilt werden.

Für unsere Beispiel-Anwendung von UML bei LUKS2 verzichten wir auf ein Verteilungsdiagramm, da dies kein verteiltes System ist und das Diagramm dafür trivial wäre.

## 4.4 LUKS2 UML-Diagramme

Mit den obigen Argumentationen lassen sich die Diagramme wie folgt darstellen:

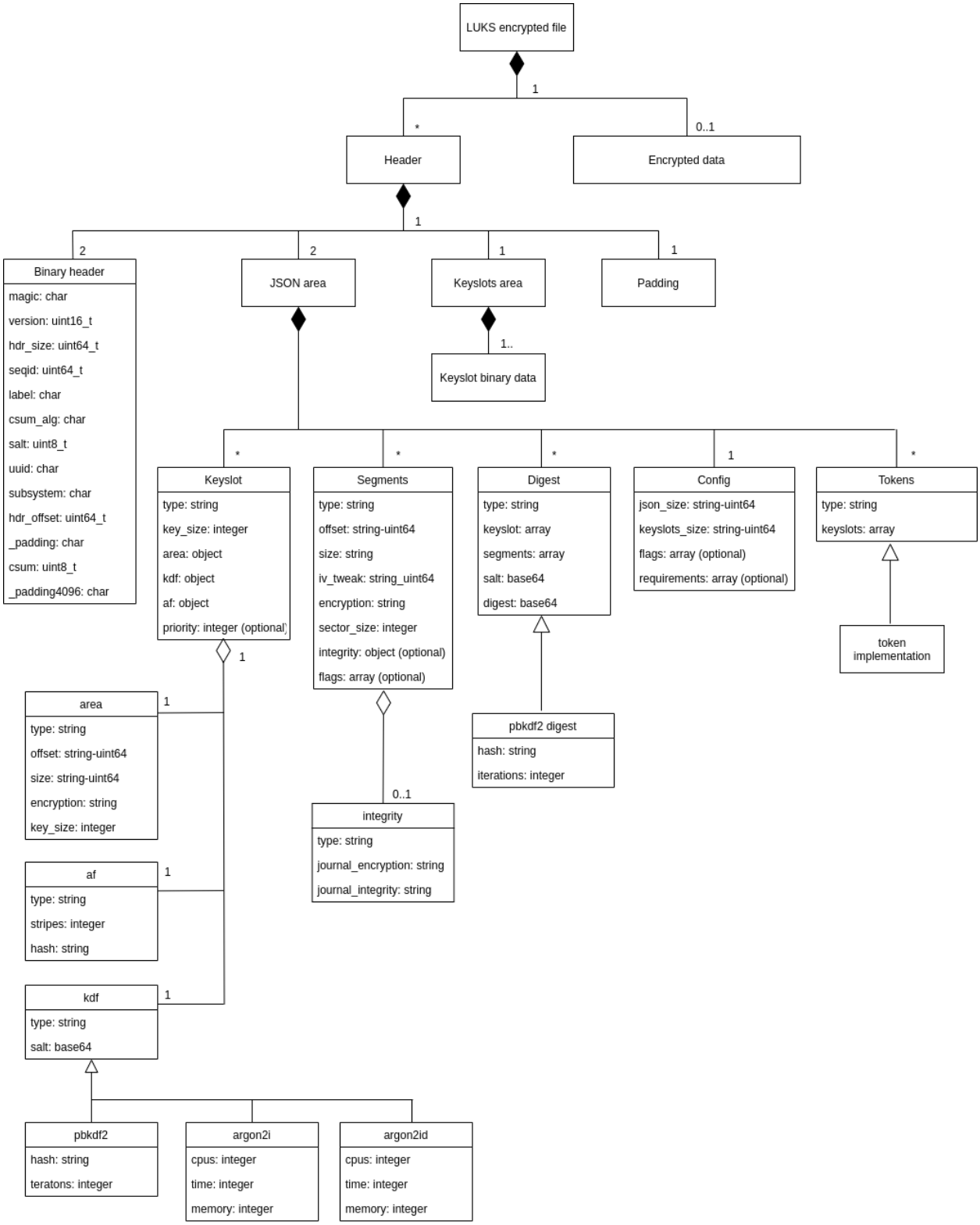


Abbildung 2: LUKS2 Klassendiagram (statischer Standpunkt)

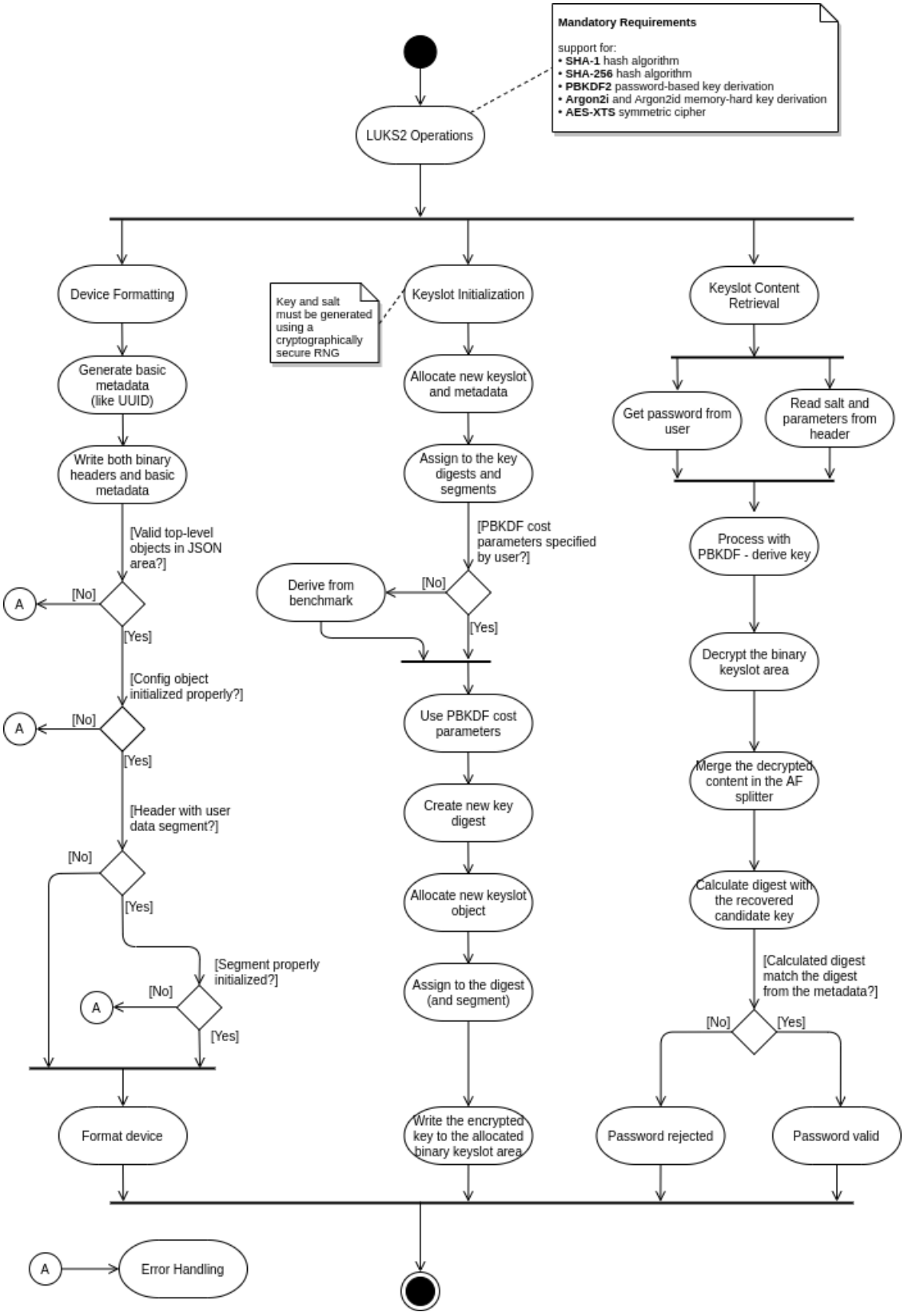


Abbildung 3: LUKS2 Aktivitätsdiagram 1 (dynamischer Standpunkt)

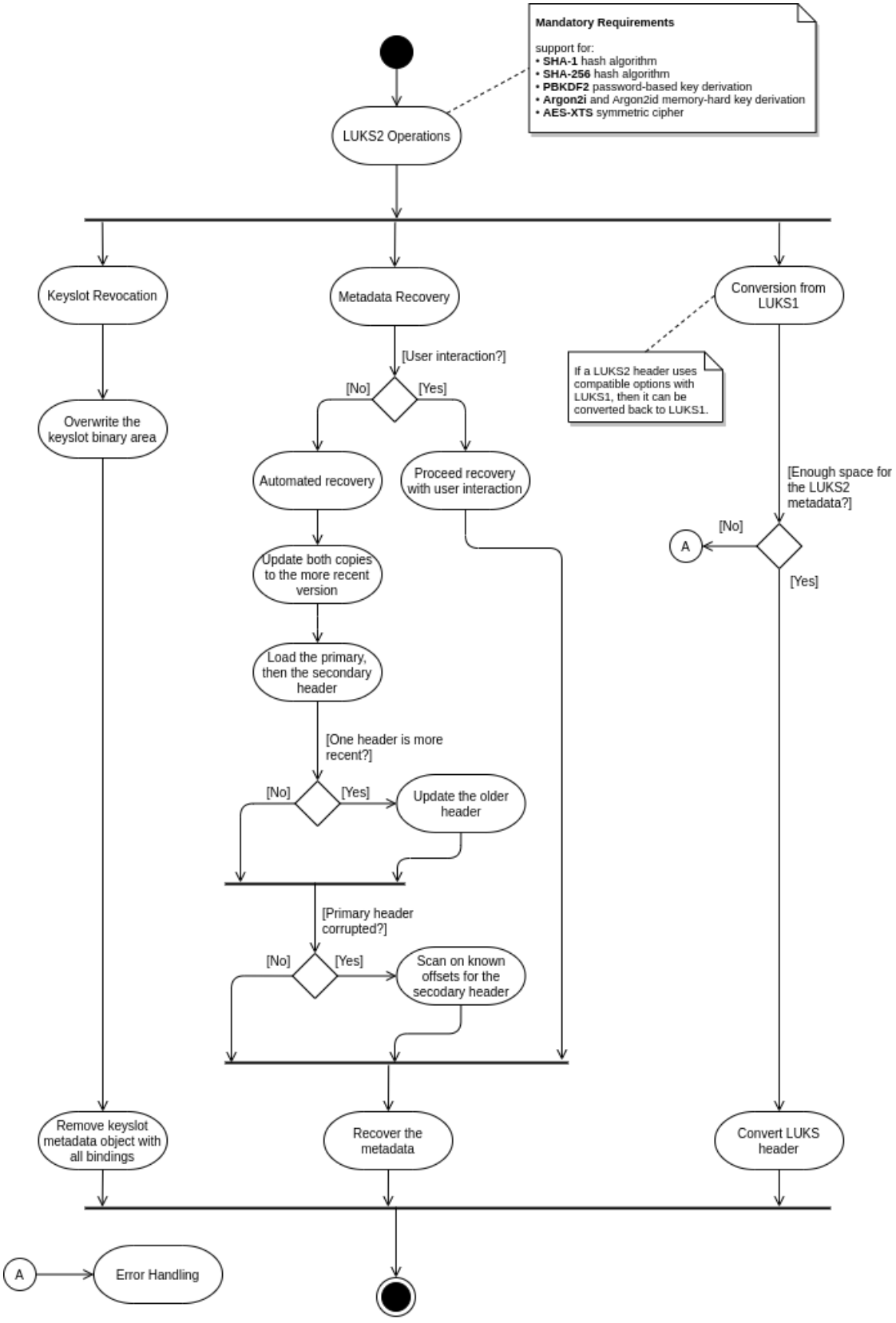


Abbildung 4: LUKS2 Aktivitätsdiagram 2 (dynamischer Standpunkt)

# 5 Architekturbeschreibung in der Praxis

Wie die Software-Architekturen in der Praxis beschrieben werden, untersuchen wir anhand den zehn meist bewerteten Open-Source Projekte im Online-Dienst *GitHub*[1]. Die benötigte Sortierung - nach Sterne (Bewertung), bietet das Open-Source Projekt *gitstar-ranking*[2] an. Wir untersuchen das Vorhandensein und die Art der Architekturbeschreibung, die Benutzung von Diagrammen, insbesondere UML und als Zusatzfeld in der Tabelle auch die wirtschaftlichen Subjekte, die hinter dem Projekt stehen.

## 5.1 Übersicht

| # | Projektname | Architekturbeschreibung | UML | Organisation |
|---|---|---|---|---|
| 1 | freeCodeCamp | sehr kurz | Nein | non-profit |
| 2 | 996.ICU | irrelevant (Katalog) | Nein | private Person |
| 3 | Vue | Projektstruktur | Nein | private Person |
| 4 | React | ausführlich | Nein | kommerziell |
| 5 | free-programming-books | irrelevant (Katalog) | Nein | non-profit |
| 6 | TensorFlow | ausführlich | Ja* | kommerziell |
| 7 | Bootstrap | Nein | Nein | private Personen |
| 8 | awesome | irrelevant (Katalog) | Nein | private Person |
| 9 | You-Dont-Know-JS | irrelevant (Bücher) | Nein | private Person |
| 10 | Coding Interview University | irrelevant (ToDo-Liste) | Nein | private Person |

*eigene, UML-ähnliche Diagramme

Tabelle 1: Übersicht

## 5.2 Einzelheiten und Analyse

1. *FreeCodeCamp*: Online Tutorials für das Programmieren [3].
   Die Architekturbeschreibung[4] besteht aus drei kurzen Sätze. Es sind keine Diagramme verfügbar. Im Sinne vom Standard ist dies keine richtige Architekturbeschreibung.

[1] https://github.com
[2] https://gitstar-ranking.com/ (Zugriff am 28.05.2020)
[3] https://github.com/freeCodeCamp/freeCodeCamp (Zugriff am 28.05.2020)
[4] https://contribute.freecodecamp.org/#/index (Zugriff am 28.05.2020)

2. *996.ICU*: Listen mit Arbeitgeber, sortiert nach Arbeitszeitpolitik[1].
   Architekturbeschreibung nicht benötigt, weil das Projekt ein Katalog ist.

3. *Vue*: JavaScript-Framework[2].
   Die Architekturbeschreibung [3] besteht aus einer Übersicht der Projektstruktur und ist nach dem Maß des Standards eher ungenügend, da viele Sichtpunkte fehlen.

4. *React*: JavaScript-Framework[4].
   Sehr ausführliche Beschreibung[5] [6] [7], die verschiedenen Standpunkte und Sichten beachtet. Es sind allerdings keine UML oder andere Diagramme zu finden.

5. *free-programming-books*: Bücherkatalog für Programmierer[8].
   Architekturbeschreibung nicht benötigt, weil das Projekt ein Katalog ist.

6. *TensorFlow*: machine-learning Framework[9].
   Die Architekturbeschreibung[10] [11] ist ausführlich und mit Diagrammen, die UML ähneln, ohne richtig aus dem UML-Standard zu stammen.

7. *Bootstrap*: HTML, CSS, and JavaScript framework[12].
   Architekturbeschreibung ist leider nirgendwo im Projekt zu finden.

8. *awesome*: Nutzergepflegte Listen mit Skripten, Codes, Büchern, Tutorials u.a.[13]
   Architekturbeschreibung nicht benötigt, weil das Projekt ein Katalog ist.

9. *You-Dont-Know-JS*: Eine Buchreihe über JavaScript.[14]
   Architekturbeschreibung nicht benötigt, weil das Projekt eine Buchreihe ist.

10. *Coding Interview University*: Selbststudium-Lernplan für Programmierer[15]
    Architekturbeschreibung nicht benötigt, weil das Projekt eine ToDo-Liste ist.

[1] https://github.com/996icu/996.ICU (Zugriff am 28.05.2020)
[2] https://github.com/vuejs/vue (Zugriff am 28.05.2020)
[3] https://github.com/vuejs/vue/blob/dev/.github/CONTRIBUTING.md#project-structure (Zugriff am 28.05.2020)
[4] https://github.com/facebook/react (Zugriff am 28.05.2020)
[5] https://reactjs.org/docs/codebase-overview.html (Zugriff am 28.05.2020)
[6] https://reactjs.org/docs/implementation-notes.html (Zugriff am 28.05.2020)
[7] https://reactjs.org/docs/design-principles.html (Zugriff am 28.05.2020)
[8] https://github.com/EbookFoundation/free-programming-books (Zugriff am 28.05.2020)
[9] https://github.com/tensorflow/tensorflow (Zugriff am 28.05.2020)
[10] https://github.com/tensorflow/docs/blob/master/site/en/r1/guide/low_level_intro.md (Zugriff am 28.05.2020)
[11] https://github.com/tensorflow/docs/blob/master/site/en/r1/guide/extend/architecture.md (Zugriff am 28.05.2020)
[12] https://github.com/twbs/bootstrap (Zugriff am 28.05.2020)
[13] https://github.com/sindresorhus/awesome (Zugriff am 28.05.2020)
[14] https://github.com/getify/You-Dont-Know-JS (Zugriff am 28.05.2020)
[15] https://github.com/jwasham/coding-interview-university (Zugriff am 28.05.2020)

## 5.3 Schlussfolgerungen

Von den zehn analysierten Projekten, ist praktisch nur die Hälfte davon wirklich Software-intensive Systeme, der Rest sind Listen oder Kataloge. Bei den fünf Software-Projekten ist UML so gut wie nirgendwo zu finden, nicht zuletzt weil oftmals eine sehr sparsame bis gar keine Architekturbeschreibung vorhanden ist. Interessant ist, dass diese Lage wesentlich schlimmer bei Projekten ist, die keine kommerzielle Unterstützung haben. Bei den massiven Projekten von *Facebook Inc.* (React) und *Google Inc.* (TensorFlow) ist eine sehr ausführliche Architekturbeschreibung, sowie auch sehr detaillierte und organisierte allgemeine Dokumentation zur Verfügung gestellt. Man kann vermuten, dass bei Projekte ohne stabile finanzielle Unterstützung die Architekturdokumentation sowie die Dokumentation im Allgemein nicht an erster Stelle steht. Allerdings stellt das ein Problem für neue Entwickler dar, die ebenfalls ins Projekt einsteigen möchten.

Unsere Betrachtungen entsprechen der Feststellungen von Brass et al.[1]:

> „Die traurige Wahrheit ist, dass die Architekturbeschreibung heutzutage, wenn überhaupt vorhanden, als nachträgliche Arbeit zu betrachteten ist. Etwas was man macht, weil man muss.“ (Übersetzung durch den Verfasser)

Eine potenzielle Ursache dabei könnte z.B. eine falsche Interpretation der Agile-Prinzipien sein. Im Agilen-Manifest[2] steht dass eine funktionierende Software mehr Wert als eine umfassende Dokumentation hat, allerdings steht auch, dass die Dokumentation wichtig ist.

Eine andere Ursache könnte z.B. die rasante Entwicklung der Software sein. In diesem Fall empfiehlt sich anstatt die Dokumentation komplett zu vernachlässigen, sie unter Einhaltung folgender Punkte zu pflegen[3]:

- Gemeinsamkeiten zwischen allen Versionen des Systems dokumentieren;
- Dokumentieren von alles, was Änderungen unterliegen darf.

[1] Bass, L. et al. (2013), S.327
[2] https://agilemanifesto.org/ (Zugriff am 28.05.2020))
[3] Vgl. Bass, L. et al. (2013), S.355-356

# 6 Zusammenfassung

Sogar die beste Software-Architektur ist nutzlos, wenn die Leute, die damit arbeiten, sie nicht verstehen, oder noch schlimmer - wenn sie sie falsch verstehen und dementsprechend falsch anwenden. Der enorme Aufwand bei der Entwicklung wäre in diesem Fall verschwendet[1].

Die Entwicklung von einer Software-Architektur ist eine schöpferische Tätigkeit und die Varianten, das Endprodukt zu beschreiben sind so vielfältig, wie die Entwickler selbst. Um einen gemeinsamen Ausgangspunkt zu schaffen, der Verständlichkeit halber, wurden dafür Standards, wie der hier betrachtete ISO/IEC/IEEE 42010, erstellt.

Der Standard macht keine Vorschriften, sondern nur Empfehlungen. Darunter ist unter anderem der Prozess der Architekturbeschreibung, welcher die Konzepte der Standpunkte und Sichten beinhaltet. Das sind die zwei Punkte, wo UML seine Stärken zeigen kann.

Auch wenn UML nicht vorgeschrieben wird, oder auch keine reine ADL-Sprache ist, hat sie sich als Standard für Diagramme etabliert, wenn solche benutzt werden.

Es sind grundsätzlich drei allgemeine Standpunkte bei der Beschreibung zu beachten - statisch, dynamisch und von der Verteilung.

Auf dem Basis der Architektur vom Open-Source Management Format LUKS2 haben wir UML Diagramme erstellt, wobei der dritte Standpunkt in diesem Fall sich als unnötig gezeigt hat.

Wie das alles in der Praxis angewendet wird, wurde mit einer Untersuchung der 10 best bewerteten Projekte in GitHub analysiert. Die Ergebnisse sind eher enttäuschend, weil die Dokumentation im Grunde bei viele vernachlässigt wird. Anders sah es bei kommerziell unterstützten Projekte aus, allerdings waren da auch wenige Diagramme zu finden.

Es bleibt noch die Frage offen, wie die Lage bei kommerziellen Projekte ist. Aufgrund mangelnder Informationen haben wir hier auf weitere Recherche in dieser Richtung verzichtet.

Es kann auch diskutiert werden, ob die Kriterien für Auswahl in dieser Arbeit realistisch sind. Vielleicht sind nicht die best-bewerteten Projekte auch die mit der höchsten Qualität.

Das ausgewählte Software-System LUKS2 ließ sich sehr gut nur mit den zwei Standpunkte - statisch und dynamisch - beschreiben. Es gibt auch Systeme, für deren Beschreibung alle drei Standpunkte wichtig sind. Weil die hier vorgestellten Standpunkte auch die wichtigsten und meist vertretenen sind, haben wir auf die weitere Forschung bei dem Verteilungspunkt verzichtet.

[1] Vgl. Bass, L. et al. (2013), S.327

# Literaturverzeichnis


International Organisation for Standardization (2011)
ISO/IEC/IEEE 42010: Systems and software engineering — Architecture description, o.O.

Object Management Group (2017)
OMG Unified Modeling Language (OMG UML), Version 2.5.1, o.O.

Brož, M. (2018)
LUKS2 On-Disk Format Specification Version 1.0.0, o.O.

Fowler, M. (2003)
UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition, Boston et al.

Bass, L. et al. (2013)
Software Architecture in Practice, Third Edition, New York et al.

Behrens, J., Giesecke, S., Jost, H., Matevska, J., Schreier, U. (2009)
Architekturbeschreibung, in: Reussner, R./ Hasselbring, W. (Hrsg.): Handbuch der Software Architektur, 2., überarbeitete und erweiterte Auflage, Heidelberg

Jahnke, J. (2009)
Reverse Engineering von Software Architekturen, in: Reussner, R./ Hasselbring, W. (Hrsg.): Handbuch der Software Architektur, 2., überarbeitete und erweiterte Auflage, Heidelberg

Hoffmann, D. (2013)
Software-Qualität, 2., aktualisierte und korrigierte Auflage, Karlsruhe

Goll, J. (2011)
Methoden und Architekturen der Softwaretechnik, Wiesbaden

Rumpe, B. (2011)
Modellierung mit UML: Sprache, Konzepte und Methodik, Berlin, Heidelberg

Hofmeister, C. (1999)
Describing Software Architecture with UML, in: Working IEEE/IFIP Conference on Software Architecture, SanAntonio

Endres, A.;Rombach, D. (2003)
A Handbook of Software and Systems Engineering Empirical Observations, Laws and Theories, Essex