Benjamin Wolf
Gernot Starke

Software-architekturen pragmatisch dokumentieren

Eine Einführung

INNOQ

Softwarearchitekturen pragmatisch dokumentieren

Eine kompakte Einführung in arc42

Benjamin Wolf
Gernot Starke

ISBN —
innoQ Deutschland GmbH
Krischerstraße 100 · 40789 Monheim am Rhein · Germany
Phone +49 2173 33660 · WWW.INNOQ.COM

Layout: Tammo van Lessen with XeLaTeX
Design: Sonja Scheungrab
Print: Pinsker Druck und Medien GmbH, Mainburg, Germany

Softwarearchitekturen pragmatisch dokumentieren – Eine kompakte Einführung in arc42
Published by innoQ Deutschland GmbH
2. Auflage · September 2019

Inhaltsverzeichnis

1 (Statt einer) Einleitung

Danke, dass Sie sich mit dem wichtigen Thema „Architekturdokumentation“ beschäftigen und dafür gerade unseren kleinen *Primer* gewählt haben. Wir vermuten, Sie entwickeln Software und einige der folgenden Aussagen treffen auf Sie zu:

- Sie möchten Ihre Architektur dokumentieren.
- Sie haben noch nie von arc42 gehört und suchen einen kompakten Einstieg.
- Sie möchten wissen, was die 42 bedeutet.
- Sie haben schlimme Dinge über Architekturdokumentation gehört, beispielsweise *ständig veraltet*, *viel zu formal* oder *macht keinen Spaß*. Sie suchen einen pragmatischen und effektiven Weg, selbst unter Zeitdruck angemessene (und leicht wartbare!) Dokumentation zu erstellen.

Bevor es los geht, noch ein kurzer Disclaimer:

Sie werden in diesem Buch des Öfteren die Begriffe „Softwaredokumentation“ und „Architekturdokumentation“, sowie „Softwarearchitektur“ und „Architektur“ antreffen. Diese Begriffe werden jeweils synonym verwendet.

Was ist arc42?

Im Kern besteht arc42 aus einem pragmatischen Template zur **Entwicklung, Dokumentation und Kommunikation von Softwarearchitekturen**. Es stammt aus der Praxis und basiert auf Erfahrungen internationaler Architekturprojekte und Rückmeldungen vieler Anwender.

Vergleichen Sie arc42 mit einem Schubladenschrank: Die Schubladen sind ordentlich beschriftet und enthalten zusammengehörige Informationen. arc42 enthält zwölf solche Fächer (also einige mehr als auf dem Bild nebenan). Die Bedeutung dieser arc42-Fächer ist leicht verständlich.

arc42 gibt Ihnen damit eine einfache, klare Struktur zur Beschreibung Ihrer (komplexen!) Systeme. Beginnend bei den Zielen und Anforderungen an Ihr System und die

Einbettung in die fachliche und technische Umgebung, können Sie nahezu alle Beteiligten oder Interessenten Ihres Systems mit passenden Informationen zur Architektur versorgen.

arc42 ist auf Verständlichkeit optimiert: Es leitet Sie auf ganz natürliche Weise an, jede Art von Architekturinformation und -entscheidung in einem verständlichen und nachvollziehbaren Kontext zu erklären. Anwender von arc42 loben die Verständlichkeit der Ergebnisse, die aus diesem etablierten Aufbau resultiert, und den überschaubaren Aufwand, eine solche Dokumentation zu erstellen.

Wie sieht's aus?

Die vorstehende Grafik zeigt die zwölf Teile von arc42. Was die einzelnen Teile *genau* bedeuten, was dazu gehört und in welcher Form, all das erklären wir am praktischen Beispiel in Kapitel 3.

Einsatzgebiete

arc42 eignet sich für beliebige Technologien und Werkzeuge.[1] Es passt großartig zu agilen Projekten, aber auch andere Organisationsformen profitieren davon.

[1] Sie können das Template auf der arc42-Website unter anderem in den Formaten als docx, pdf, AsciiDoc, Markdown, Textile, Confluence und HTML herunterladen.

Neben dem Template schlägt arc42 einige Kernaufgaben zur Entwicklung und Konstruktion effektiver Softwarearchitekturen vor, auf die wir an dieser Stelle allerdings nicht weiter eingehen.

arc42 funktioniert für Informationssysteme, Embedded- und Real-Time Systeme, Web- und Desktop-Anwendungen, für Microservices, Self-contained Systems, Client-Server und mobile Anwendungen; sogar große Data-Warehouse-Anwendungen konnten wir damit entwickeln und dokumentieren.

Wie geht das?

arc42 funktioniert unabhängig von Ihrem Entwicklungsvorgehen, d.h. egal ob Sie agil oder eher traditionell arbeiten. Sie können Ihren arc42 *Schrank* in jeder beliebigen Reihenfolge bearbeiten, ganz wie es Ihre konkrete Situation erfordert. Bei einer Neuentwicklung werden Sie wahrscheinlich mit den Teilen rund um Anforderungen und Ziele beginnen, bei der Arbeit an bestehenden Systemen möglicherweise sofort in die Tiefen der querschnittlichen Konzepte oder Bausteinsicht abtauchen.

Sie können die Arbeit an arc42-basierter Architekturdokumentation jederzeit unterbrechen oder wieder aufnehmen. Die feste Struktur des *Schrankes* stellt problemlose Weiterarbeit sicher. Voraussetzung dafür ist natürlich, dass die beteiligten Stakeholder etwas Verständnis für die arc42-„Schrankfächer“ besitzen. Manche nennen diese positive Eigenschaft von arc42 übrigens „prozessagnostisch“.

Herkunft

arc42 stammt von Dr. Gernot Starke (einem der Autoren dieses Buches) und Dr. Peter Hruschka[2]. Beide haben um 2002 angefangen, die Erfahrungen vieler Entwicklungs- und Architekturprojekte zusammenzutragen und eine flexibel verwendbare Struktur („Template“) für Softwarearchitekturen zu entwickeln.

Als Vorbild fungierte Volere[3], ein Template für Software-Requirements, das Sie flexibel in beliebigen Entwicklungsprojekten zur Kommunikation und Dokumentation von Anforderungen (sprich: *des Problems*) einsetzen können. arc42 stellt das Pendant für

[2] https://b-agile.de
[3] http://www.volere.co.uk/templates.htm

Softwarearchitekturen dar (sprich: *die Lösung*). Von Anfang an stand arc42 unter der Prämisse, vollständig frei verfügbar (im Sinne von Open-Source) zu sein.

42

Ach ja, 42: Die etwas ältere Generation von Entwicklerinnen und Entwicklern mag einen unserer damaligen Lieblingsautoren, Douglas Adams, noch kennen, einen verschrobenen Science-Fiction Autor. Douglas hat die Trilogie *Per Anhalter durch die Galaxis*[4] geschrieben, in der die Zahl 42 als die „Antwort auf die Frage nach allem" erscheint. arc42 soll für Ihre Systeme und Softwarearchitekturen die Antworten auf alle (oder zumindest auf viele) Fragen bereithalten.

Beispiele für den Einsatz von arc42

Falls Sie Beispiele für arc42 sehen möchten: einfach noch ein paar Seiten Geduld mitbringen. Ein kommentiertes Beispiel finden Sie in Kapitel 3.

Eine umfangreiche Sammlung von sechs Beispielen finden Sie als eBook arc42-by-Example – für Sie kostenfrei (siehe Literaturliste).

Unserer Einschätzung nach sind solche Beispiele der beste Weg, arc42 in realen Projekten einzuführen.

Warum Architektekturdokumentation?

Nehmen wir an, wir kaufen uns einen Schlafzimmerschrank eines bekannten schwedischen Möbelherstellers mit ein paar Einlegeböden. Der Schrank wird in Einzelteilen und in Begleitung einer Bauanleitung geliefert.
Die Anleitung beschreibt – neben den notwendigen Personen und Werkzeugen für den Aufbau – detailliert, welche Bauteile verwendet werden, wie diese zu einzelnen Modulen zusammengesetzt werden, welche dann zusammen den Schrank ergeben.

[4] Douglas Adams: Per Anhalter durch die Galaxis (https://de.wikipedia.org/wiki/Per_Anhalter_durch_die_Galaxis). Das Buch, von britischen Humor und skurrilen Szenen geprägt, ist ein zeitloser Klassiker. Den Film finde ich (Gernot) miserabel.

Wollen wir ihn um neue Funktionen - wie etwa eine Sockenschublade und eine Kleiderstange - erweitern, so ist auch das ausführlich beschrieben. Ebenso können wir die Anleitung zu Rate ziehen, falls ein Teil ersetzt werden muss, weil es defekt geliefert wurde oder im Laufe der Zeit kaputt gegangen ist.

Dieses Szenario kommt Ihnen sicherlich bekannt vor. Ohne die Bauanleitung ist es nahezu unmöglich, den Schrank exakt so aufzubauen oder zu erweitern, wie es von den Herstellern ursprünglich gedacht war. Genau so ist es auch in der Softwareentwicklung – zugegeben, vielleicht nicht ganz genau so, aber Sie werden gleich sehen, worauf wir hinauswollen.

Wenn wir ein neues Softwareprojekt beginnen, erstellen wir zuerst einen groben Bauplan, wie das fertige Produkt aussehen soll. Dieser Plan wird schrittweise verfeinert, bis alle damit einverstanden sind. Bei der Umsetzung werden wir ab und an feststellen, dass unsere Annahmen nicht ganz korrekt waren und Dinge anders realisiert werden müssen. Diese Änderungen tragen wir im Bauplan nach.

Sie ahnen es sicher schon: Der *Bauplan* ist nichts anderes als die Dokumentation unserer Software und deren Architektur. Er beschreibt, aus welchen kleinen Teilen die einzelnen Module bestehen, wie sich das Gesamtsystem zusammensetzt, welche Technologien eingesetzt werden und welche Designentscheidungen getroffen wurden.
Wenn neue Features implementiert werden sollen, so ist anhand der Dokumentation schnell ersichtlich, an welchen Stellen die Software erweitert werden muss.
Wollen wir gewisse Bestandteile austauschen, weil es inzwischen modernere Technologien oder Herangehensweisen gibt, so können wir auch hier auf unsere Softwaredokumentation zurückgreifen und herauslesen, an welchen Schrauben wir dafür drehen müssen – und wovon wir vielleicht lieber die Finger lassen.
Stoßen neue Entwicklerinnen und Entwickler zum Projekt, so können sie schnell den Aufbau der Software und das Zusammenspiel der einzelnen Komponenten nachvollziehen.

Eben genau so, wie wir das mit einer Aufbauanleitung eines Schranks tun können. Und deswegen ist Architekturdokumentation so wichtig.

Warum arc42?

Das arc42-Template bietet Kreativitäts- und Arbeitshilfsmittel, Diskussionsgrundlage für Teams, Einarbeitungsunterstützung für neue Mitarbeiter und noch viel mehr.

Bei der Arbeit mit dem arc42-Template fällt kein zusätzlicher Aufwand für Sie oder Ihre Teams an: Sie

- beschreiben Sachverhalte, die Stakeholder des Systems kennen müssen,
- erklären Zusammenhänge, die Voraussetzungen für das Verständnis des Systems oder einzelner Entwurfsentscheidungen sind,
- heben nur wichtige Entscheidungen auf, die Sie ohnehin treffen müssen.

arc42 hilft Ihnen, für jede dieser Entscheidungen und Sachverhalte eine passende Schublade zu finden, in der alle Beteiligten diese Informationen leicht wiederfinden können.

arc42 steht unter einer liberalen Open-Source-Lizenz zur Verfügung, daher ist die Nutzung von arc42 in jedem Fall kostenfrei.

Sie haben spezielle Anforderungen

Das sehen wir ein: Sie und Ihre Organisation haben sehr spezielle Anforderungen an Ihre Architektur und deren Kommunikation oder Dokumentation.

Dafür könnten Sie natürlich ein eigenes, unternehmensspezifisches Template entwerfen. Aber:

- Sie müssen Zeit in dieses Template investieren, die Ihnen dann bei dem Entwurf und der Implementierung Ihres Systems verloren geht. Ist es das wirklich wert?
- Sie müssen Ihr eigenes Template dann Ihrem Team erklären, möglicherweise sogar ein paar Beispiele für die Nutzung Ihres eigenen Templates bereitstellen. All das gibt es für arc42 bereits: fertig, bewährt und frei verfügbar.

Im Zweifel können Sie arc42 nehmen und um spezifische Themen erweitern oder auch Teile weglassen – derartiges *Maßschneidern* halten wir für normal und ist seitens der arc42-Autoren ausdrücklich erwünscht!

Warum dieses Buch?

Seit 2005 verwenden Unternehmen arc42, um Software- und Systemarchitekturen zu dokumentieren. Seither durften wir dabei helfen, arc42 einzuführen, bestehende Dokumentation im Sinne von arc42 aufzubereiten und neue Systeme mit Hilfe von arc42 zu entwickeln und zu dokumentieren. Mit diesem Buch wollen wir die Verbreitung von arc42 weiter steigern und Sie dazu ermutigen, sich intensiver mit arc42 auseinanderzusetzen.

Das arc42-Template selbst enthält kurze Hinweise und Ratschläge zu den einzelnen Teilen („Schubladen"). Die Struktur ist übersichtlich und verständlich, es gibt keine Hürden zur Anwendung von arc42.

So einfach und klar strukturiert arc42 jedoch auch ist: Im Projektalltag treten immer wieder Fragen auf, die wir in diesem Primer in Auszügen adressieren möchten.

Danksagung

Wir bedanken uns bei allen Kunden, Mandantinnen und Mandanten, bei denen wir viel über die (teils bittere) Realität von Softwarearchitektur und deren Kommunikation und Dokumentation lernen durften.

Danke an unsere INNOQ-Kolleginnen und -Kollegen, mit denen wir über technische Dokumentation im Allgemeinen und arc42 im Speziellen diskutieren durften. Danke für Eure Anregungen, Ideen, Reviews und die großartigen Events.
Danke an INNOQ für die langjährige Unterstützung von arc42 sowie die Unterstützung dieses Buchprojektes.
Danke an Sonja Scheungrab (INNOQ) für die grafische Gestaltung dieses Buches.
Danke an Lars Hupel (INNOQ) für das intensive Lektorat und den Highscore beim *Book-Bug-Hunting*.

Danke an die Mitwirkenden bei arc42, insbesondere Ralf D. Müller und Dr. Peter Hruschka.

Gernot: Danke an meine Traumfrau Uli! Mit Dir scheint für mich die Sonne!

Ben: Danke an meine Frau Kerstin. Ohne Dich wäre ich nie so weit gekommen!

2 Vor- (und Vorur-)teile

Technische Dokumentation stellt in der Softwareentwicklung oft[1] ein Problem dar, oftmals begleitet von Vorurteilen gegenüber deren Erstellung und Pflege. Als würde das Schmerzen bereiten oder wäre eine Schande.

Neben der allseits bekannten Aussage „Dafür fehlt uns die Zeit“ (übrigens ein Indikator für heftige Fehlplanung) treffen wir in Entwicklungsprojekten noch auf eine Reihe typischer irriger Annahmen - die wir Ihnen hier gerne widerlegen möchten.

„Dokumentation ist doch immer total veraltet, da schreiben wir erst gar keine!“

Diesen Satz haben wir vermutlich am häufigsten gehört. Er umschreibt – neben der eingangs beschriebenen Fehlplanung und dem Mangel an Interesse an einer (guten) Dokumentation – folgendes Problem:

Es ist viel zu kompliziert, eine neue Dokumentation zu erstellen oder die bestehende Dokumentation zu warten.

Hier wirkt sich die Struktur von arc42 positiv auf Erzeugung *und* Aktualisierung von (Architektur-) Dokumentation aus. Das Grundgerüst der Dokumentation können Sie mit einer vorhandenen Vorlage[2] schnell erzeugen und initial mit den wichtigsten Punkten befüllen. Anschließend können Sie Stück für Stück weitere Kapitel – wenn benötigt – ergänzen.

arc42 ist ein leichtgewichtiges Werkzeug, das Sie sehr einfach auf Ihre konkreten Bedürfnisse anpassen können. Im Verlauf der Entwicklung eines Systems können Sie einzelne Kapitel leicht aktualisieren. Dank des Aufbaus ist sofort klar, welche Stellen Sie oder Ihr Team anpassen müssen.

„Es ist super aufwändig, eine Dokumentation für eine Softwarearchitektur zu erstellen.“

[1] Der Open-Source Survey (https://opensourcesurvey.org/2017/) beispielsweise zählt schlechte Dokumentation zu den Top-Problemen vieler Open-Source Projekte.

[2] Unter https://arc42.org/download stehen Vorlagen in mehr als acht Formaten und vier Sprachen zum Download bereit.

Diese Aussage ist an sich korrekt, denn Dokumentation bedeutet Arbeit – und somit Aufwand. Bei dem Gedanken an Softwaredokumentation erscheint vor dem inneren Auge nahezu sofort ein Wälzer mit mehr als 300 Seiten, der diese Reaktion hervorruft. Häufig ist ein Dokumentationsartefakt dieser Größe massiv überdimensioniert und unnötig. Die Dokumentation einer Software(architektur) sollte dem Projekt angemessen gestaltet sein.

Der modulare Aufbau von arc42 ermöglicht es Ihnen, den Informationsgehalt genau auf Ihre Software abzustimmen. Wenn Sie ein Kapitel nicht benötigen, so lassen sie es einfach weg. Der Fokus bei der Dokumentation liegt auf den Informationen, welche Ihre Stakeholder benötigen. Sie dokumentieren nur die Tatsachen und Probleme, die notwendig sind, um das System oder einzelne Designentscheidungen zu verstehen. So erzeugen Sie keinen unnötigen Mehraufwand. Denn alles, was Sie dokumentieren, müssen Sie zukünftig auch warten.

„Bei Dokumentation kann nichts Gutes rauskommen, denn die will ja keiner schreiben."

Hier liegt das Problem an den Rahmenbedingungen, nicht an der Dokumentation selbst. Aus Sicht der Entwicklerinnen und Entwickler ist die Dokumentation von Software oft mit Schmerzen verbunden: Formalitäten, unpassende Werkzeuge und Formate, fehlende Zielvorgaben sowie unklare Anweisungen, was genau sie eigentlich dokumentieren sollen. Die Fachbereiche meckern, der Dokumentationsvorgang ist zäh. Das war's dann auch mit der Kreativität. Verständlich, dass niemand Lust hat, sich um die Dokumentation zu kümmern.

Sie finden bereits eine ganze Reihe verbreiteter Dateiformate vom arc42-Template auf der Website zum Download. Zudem haben Sie die Möglichkeit, das Template auf ein beliebiges Werkzeug oder Format zu übertragen. Damit erhöhen Sie die Akzeptanz von Dokumentation bei den Personen, die sie schreiben dürfen. Die Struktur von arc42 macht Ihnen und Ihrem Team Vorschläge, *was* Sie dokumentieren können. Sie bekommen darüber hinaus viele Vorschläge, *wie* Sie das tun könnten. Gleichzeitig sind die Vorgaben recht frei, so dass sich der kreative Geist Ihres Teams nahezu uneingeschränkt entfalten kann. Durch diese Kombination aus Anweisungen und Freiheit geht die Dokumentation schnell von der Hand.

„Eine riesige Architekturdokumentation ist doch niemals überschaubar.“

Architekturdokumentation von Software wird meist auf **mehrere** Dokumente verteilt. Dadurch wirkt sie umfangreich und schwer zugänglich. Dieser Eindruck verfestigt sich dann, wenn jemand versucht, spezifische Informationen zu finden oder zu aktualisieren. Solche *Dokumentationshaufen* entstehen, wenn die gesamte Dokumentation beliebig über Netzlaufwerke und Wiki-Einträge verteilt ist und sich niemand über die Struktur der Dokumentation Gedanken macht.

Eine Stärke von arc42 ist genau diese Vorgabe von Struktur und Inhalt. Sie können damit Qualitätsziele, betroffene Stakeholder, Rahmenbedingungen und Anwendungsbereich der Software ebenso beschreiben wie den Lösungsansatz, Baustein-, Laufzeit- oder Deployment-Sichten, Architekturentscheidungen und übergreifend gültige Konzepte. Mit diesem Aufbau erhöhen Sie die Übersichtlichkeit der gesamten Dokumentation und bieten einen schnellen Einstieg für Ihre Teammitglieder, die neu zum Projekt hinzustoßen oder die ihr Wissen über das Projekt auffrischen wollen. Alles befindet sich an einer zentralen Stelle.

„Ich weiß nicht, wo ich all die Dokumente finden soll.“

Diese Aussage provoziert fast das Kontra „Es gibt vermutlich auch keine“, und wahrscheinlich wäre es oft korrekt. Sie beschreibt im Grunde einen ähnlichen Aspekt, der auch bei der Übersichtlichkeit weiter oben zu finden ist. Wären alle Orte der Dokumentation bekannt, dann wirkte sie mächtig und überfordernd. Diese Orte sind aber selten in ihrer Gänze bekannt. Dies führt zu mehrfachen Beschreibungen der gleichen Dinge und unterschiedlichen, manuell erstellten Versionen, die wiederum Verwirrung stiften oder gar Fehler verursachen.

Halten Sie und Ihr Team sich, wenn Sie die Dokumentation schreiben, an die die Struktur von arc42, so entfallen diese Probleme. Da Sie alle Informationen an genau einer Stelle ablegen, löst sich Verwirrung in Klarheit auf, und statt verwirrender Kopien mit verschiedensten Dateinamen gibt es nun eindeutige Versionen der Dokumentation. Sie erstellen die Dokumentation effizient, vermeiden Mehraufwand und reduzieren aktiv die Wahrscheinlichkeit von Fehlern durch Dopplung.

„Ich habe gar nicht das Werkzeug, um die Dokumentation anzuschauen."

Die Dokumentation von Softwarearchitektur wird sehr oft mit dem Einsatz spezieller Modellierungswerkzeuge verbunden, welche kommerziellen Lizenzen unterliegen und somit Kosten verursachen und nur nach einer mehrtägigen Schulung bedient werden können.

Der Einsatz von arc42 ist für Sie absolut frei von Kosten oder kommerziellen Lizenzen und ist unabhängig von Werkzeugen. Pflegen Sie die Dokumentation beispielsweise in Confluence, so benötigen Sie allein einen Browser als Werkzeug. Entscheidet sich Ihr Team dazu, die Dokumentation in reiner Textform im Repository des Quellcodes zu verwalten, so genügt Ihnen ein einfacher Texteditor oder ebenfalls der Browser. Wählen Sie Word als ihr Werkzeug für Softwaredokumentation, ist auch diese Anwendung in der Regel bei allen Beteiligten vorhanden. Darüber hinaus können Sie die Dokumentation sowohl von Word als auch von Confluence in ein PDF-Dokument exportieren, welches Sie ebenso überall lesen können – sei es an einem Rechner, auf einem Smartphone oder auf einem Tablet.

„Für die Dokumentation muss man UML können, das kann ich nicht."

Folgen wir der verbreiteten Lehrmeinung an Hochschulen, nach der UML *die* Modellierungssprache für Softwarearchitekturen schlechthin ist, so ist diese Aussage durchaus nachvollziehbar. Und beschreibt genau das Problem: Es gibt keine oder zu wenige Teammitglieder, die UML beherrschen. Im Laufe der Jahre trafen wir auf viele heterogene Teams. Nur wenige Teammitglieder hatten den „klassischen" Weg eines Informatikstudiums beschritten. Es ist nicht hilfreich, wenn dann die eine Person, die UML kann, die Softwarearchitektur damit beschreibt. Denn sie ist dann die einzige, die das auch zu 100% versteht.

Aus diesem Grund ist arc42 Modellierungswerkzeug-agnostisch gestaltet. Es ist prinzipiell egal, welche Sprache der Modellierung Sie wählen, so lange alle wichtigen Stakeholder die Diagramme verstehen können. Im einfachsten Fall sind das Kästen und Pfeile. Die kann jede Person verstehen und auch zeichnen, sei es über PowerPoint, ein einfaches Zeichenprogramm oder ein mächtiges Modellierungswerkzeug, das in

jedem Fall Kästen und Pfeile beherrscht. Selbstverständlich können Sie auch andere Formen wählen, wie beispielsweise das C4-Modell[3] oder natürlich ein UML-Modell in Enterprise Architect[4] - für das es im Übrigen auch ein arc42-Template gibt.

„Das versteht doch niemand, was da steht. Das ist doch viel zu komplex.“

Mal abgesehen von der bereits angesprochenen UML-Problematik, kommt diese Aussage meist von Personen, die nicht direkt in die Entwicklung der Software involviert sind, etwa Projektleiter oder Verantwortliche aus den Fachbereichen. Die Dokumentation von Software ist für nahezu alle Beteiligte relevant und auch interessant. Meistens wird die Architektur – wenn überhaupt – von denjenigen und vor allem für diejenigen geschrieben, die den Code schreiben.

Wenn Sie sich den Aufbau des arc42-Templates ansehen, so können Sie an Hand der Kapitelstruktur eine Verfeinerung des Detailgrads erkennen. Je weiter Sie sich in der Dokumentation vorarbeiten, umso detaillierter und ausführlicher werden die Konzepte und Module erläutert. Während Sie mit den ersten beiden Kapitel die Anforderungen an das eigentliche System und dessen Qualität, die betroffenen Stakeholder, sowie die technischen und organisatorischen Rahmenbedingungen beschreiben, skizzieren Sie in Kapitel 3 und 4 die fachlichen und technischen Kontexte Ihrer Anwendung sowie die angepeilte Lösungsstrategie. Diese Kapitel sind für alle Stakeholder relevant und sollten daher auch für alle verständlich sein.

[3] https://c4model.com
[4] https://www.sparxsystems.de

„Diese Dokumentation ist mir zu oberflächlich, die hilft für die Umsetzung doch sowieso nicht."

Dieses Vorurteil – übrigens das genaue Gegenteil der Aussage weiter oben auf dieser Seite – ist durchaus berechtigt, wenn wir an die Architekturposter an Wänden oder Placebo-Dokumentationen in Schreibtischschubladen denken, die uns in diversen Projekten begegnet sind. Speziell die Poster sind oft ein sehr oberflächliches Schaubild einer Architektur, das außerdem extrem veraltet ist. Zudem ist es weit verbreitet, eine Softwarearchitektur zwar auf oberen (abstrakten) Ebenen zu dokumentieren, jedoch vor den interessanten Themen in höheren Detailgraden aufzuhören.

Wenn Sie die Gliederung von arc42 betrachten, finden Sie verschiedene Punkte, an welchen Platz für genau diese Details ist. Die Building-Blocks in Kapitel 5 erlauben es, stufenweise von einer sehr hohen Ebene in tiefere Ebenen hinabzusteigen. Je nach Anforderung können Sie sich hier bis auf Klassenebene begeben. Unterstützend dazu - und dort vielleicht besser aufgehoben - können Sie mit Kapitel 8 ihre übergreifenden Konzepte bis ins Detail beschreiben. Dort ist auch Platz für Ihr Business-Entity-Modell oder ihr Datenmodell. Wichtige Abläufe bringen Sie perfekt in Kapitel 6 unter, sei es als Sequenzdiagramm, Aktivitätsdiagramm oder einfach als textuelle Beschreibung. Diese Dokumentation besitzt dann auch für Ihr Team, welches das System umsetzt, einen riesigen Mehrwert.

„Wir wissen ja dann trotzdem nicht, warum das damals so beschlossen wurde."

Mit Glück finden sich in Architekturdokumenten Informationen über Entscheidungen, die im Laufe des Projektes getroffen wurden. Sehr selten allerdings, warum. Tatsächlich sahen nur sehr, sehr wenige Dokumentationsvorlagen, die wir bisher angetroffen hatten, einen Platz für Architekturentscheidungen vor. Genauer gesagt, einen Platz für die Erklärungen, *warum* eine Entscheidung getroffen wurde, welche Kriterien *dafür* und *dagegen* sprachen und welche *Alternativen* betrachtet wurden. Wir sind uns sicher, dass auch Sie sich einmal in einem Projekt die Frage stellten, weshalb nur diese augenscheinlich ungünstige Entscheidung getroffen worden war.

Es ist offensichtlich, dass diese Erklärungen sehr wichtig sind, denn nur damit lassen sich die Entscheidungen zu späteren Zeitpunkten nachvollziehen. Aus diesem Grund

steht Ihnen im Template von arc42 ein ganzes Kapitel zur Verfügung, das nur diesen Entscheidungen und Erklärungen gewidmet ist. In unserem Beispiel ist das Kapitel 9. Dabei ist die Form weniger wichtig, Sie können Ihre Entscheidungen in Listen, Tabellen oder in dedizierten Unterkapiteln dokumentieren.

Eine sehr charmante Form für die Dokumentation von Mikro-Architekturentscheidungen in Projekten ist die der Architecture Decision Records[5]. Damit dokumentieren Sie Ihre Architekturentscheidungen direkt bei der eigentlichen Software, im Code-Repository. Architekturentscheidungen und deren Umsetzung sind so ganz nah beieinander. In Ihrer arc42-Dokumentation verweisen Sie einfach auf den Unterordner im Repository samt Repository-URL. Dies gilt natürlich für alle Repositories, die zu Ihrer Software gehören. Sowohl Ihre Entwicklerinnen und Entwickler als auch andere Projektbeteiligte ohne Wissen über den Quellcode haben dadurch Zugriff auf die Architekturentscheidungen Ihrer Projekte.

[5] http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions

3 arc42 am Beispiel

Wir stellen Ihnen arc42 anhand eines Beispiels vor und verwenden dafür einen frei verfügbaren (Open-Source) Checker für HTML. Dieser HTML-Sanity Checker[1] (kurz: HtmlSC) überprüft HTML-Dateien auf semantische Fehler, insbesondere falsche URIs (Uniform Resource Identifier). Dazu gehören falsche Querverweise, fehlende Bilder, mehrfach vorhandene Sprungziele (Link-Targets), defekte externe Links oder fehlerhafte Alternativtexte für Bilder. Als Resultat liefert HtmlSC ansprechend aufbereitete Reports, ähnlich wie nachfolgendes Beispiel.

Der gesamte Quellcode inklusive der sehr ausführlichen Dokumentation liegt auf Github[2]. Sie können HtmlSC dazu nutzen, ihre eigene Dokumentation (sofern Sie beispielsweise mit AsciiDoc oder Markdown arbeiten) zu überprüfen.

In diesem Primer finden Sie einen Auszug der kompletten Architekturdokumentation von HtmlSC, jeweils gepaart mit einigen methodischen und praktischen Tipps für die jeweiligen Teile von arc42. Wir verwenden dabei die *linken* Buchseiten immer für das *konkrete* Beispiel, die *rechten* Buchseiten für die *methodischen Erläuterungen* dazu. Die Nummerierung der einzelnen Unterkapitel entspricht denen des arc42-Templates. Aus Übersichtsgründen haben wir davon abgesehen, jeweils die Nummer dieses Hauptkapitels vorne anzustellen.

[1] https://hsc.aim42.org
[2] https://github.com/aim42/htmlSanityCheck

1 Einführung und Ziele

1.1 Aufgabenstellung

HtmlSC unterstützt beim Schreiben digitaler Formate, indem es HTML auf semantische Fehler überprüft, beispielsweise defekte oder fehlerhafte Links (URIs), fehlende Bilder, mehrfach definierte Sprungziele oder fehlende Alternativtexte.

Anforderungen

Anforderung	Erläuterung
Semantische Prüfung von HTML	siehe Bild
Unterstützung von Gradle und Maven	HtmlSC läuft als Gradle- und Maven-Plugin.
Liefern von Verbesserungsvorschlägen	Wenn HtmlSC Fehler entdeckt, sollte es sinnvolle Verbesserungsvorschläge unterbreiten.
Einfach konfigurierbar	Checks sollen konfigurierbar sein, bspw. Name von Eingabedateien, Timeouts und Verhalten bei bestimmten Statuscodes für externe Links, etc.

Motivation

Bevor Sie zur eigentlichen Architektur des Systems kommen, sollten Sie die Anforderungen in groben Zügen klarstellen. Teil 1 von arc42 stellt die treibenden Kräfte vor: die Anforderungen, die Qualitätsziele (Teil 1.2) sowie betroffene Stakeholder (Teil 1.3).

Was und wie

Teil 1.1 umreißt die zu lösende Aufgabe grob und gibt Verweise auf die (hoffentlich vorhandene) Dokumentation der Anforderungen.

Tipps

- Stellen Sie in Kurzform die wesentlichen Aufgaben Ihres Systems dar.
- Verweisen Sie auf ihr Pflichtenheft.
- Sollten Sie gar keine anderweitig dokumentieren Anforderungen haben, steht in diesem Kapitel etwas mehr.

1.2 Qualitätsanforderungen

Prio	Qualitätsziel	Szenario
1	Sicherheit	(Im Sinne von *Unversehrtheit*) HtmlSC verändert *niemals* Inhalte von zu prüfenden Dateien.
1	Korrektheit	Jeder defekte interne Verweis (Cross-Referenz) wird gefunden.
1	Korrektheit	Jeder potentielle semantische Fehler (der jeweils konfigurierten Prüfungen) wird gefunden. Im Zweifelsfall[3] wird der Fehler berichtet und die Nutzer müssen entscheiden, ob es sich wirklich um einen Fehler handelt.
2	Flexibilität	Verschiedene Arten von Prüfungen sind konfigurierbar, ebenso Report-Formate.
3	Performance	Prüfung einer 100kB großen HTML Datei in unter 10 Sekunden (Exklusive der Startup-Zeit von Gradle).
4	Integrierbarkeit	HtmlSC läuft als Plugin mit Gradle[4] und Maven, ebenso kann es per Kommandozeile gestartet werden.

[3] Besonders bei der Prüfung externer (http und https) Links hängt das Ergebnis von einer Reihe von externen Faktoren ab, wie Verfügbarkeit des Netzwerks, Latenz oder Konfiguration des betroffenen Webservers. HtmlSC kann nicht in allen Fällen die endgültige Ursache von möglichen Problemen identifizieren.

[4] https://gradle.com

Motivation

Qualitätsziele und -anforderungen beeinflussen grundlegende Architekturentscheidungen oft *maßgeblich*, daher sollten Entwicklungsteams diese Kategorie von Anforderungen möglichst explizit und genau kennen.

Was und wie

Beschreiben Sie die wichtigsten vier bis fünf Qualitätsziele oder -anforderungen mit möglichst konkreten Szenarien in einer Tabelle, am besten geordnet nach Prioritäten.

Tipps

Obwohl Sie alle Qualitätsanforderungen erfüllen müssen, die das Pflichtenheft verlangt, sollten Sie diesen Teil knapp halten. Die Konzentration auf die wichtigsten Ziele ist wertvoll, aber bitte mit kurzen Erläuterungen, nicht nur als Schlagwörter.

Achten Sie auf die Unterscheidung von (oft kurzfristigen) Projektzielen und den (meist langfristigen) Qualitätsanforderungen.

1.3 Stakeholder

Hinweis

Für unser kleines Beispiel gibt es nur eine sehr kleine Anzahl von Stakeholdern. In *realen* Situationen haben Sie mit Sicherheit viele mehr.

Rolle & Beschreibung	Ziele	Erwartungshaltung
Autoren von Dokumentation	müssen Dokumente auf korrekte Verweise prüfen	HtmlSC als laufende Software, sind an dessen Architektur nicht interessiert
arc42 User	suchen ein praktisches Beispiel, wie sie arc42 *anwenden* können	HtmlSC Architektur mit arc42 dokumentiert, einfach zugänglich (sprich: dieser Primer)
Softwareentwickler*innen	möchten ein Beispiel einer pragmatischen Architekturdokumentation	HtmlSC Architektur dokumentiert, zusätzlich Zugriff auf Quellcode
Studierende der Informatik	möchten das Template-Method Pattern verstehen	Konzeptioneller Einsatz des Musters, siehe Kapitel 8.

Motivation

Das Entwicklungsteam sollte grundsätzlich einen Überblick über die beteiligten Stakeholder haben, d. h. über alle Personen, die:

- vom System oder dessen Architektur betroffen sind,
- besondere Anforderungen an das System oder dessen Architektur haben,
- die Architektur kennen sollten oder
- von der Architektur überzeugt werden müssen.

Ihr Team muss wissen, welches Informationsbedürfnis diese Stakeholder hinsichtlich der Architektur haben (also deren *Erwartungshaltung*).

Was und wie

Führen Sie Ihre Stakeholder als Tabelle inklusive der jeweiligen Ziele oder Aufgaben in Bezug auf das System.

Erklären Sie zusätzlich die *Erwartungshaltung* der einzelnen Stakeholder an die Kommunikation und/oder Dokumentation der Architektur: Klären Sie, was diese Personen oder Organisationen von der Architektur erwarten, welche Informationen oder Liefergegenstände sie benötigen. Das ist in der Regel *nicht* die Erwartung an das System (denn DIE haben Stakeholder hoffentlich bereits als Anforderungen formuliert).

Tipps

- Sie können Teil 1.3 (manchmal) einsparen, wenn Ihr Management eine konsistente Stakeholderliste pflegt.
- Klären Sie den (aus unserer Sicht wichtigen) Punkt der *Erwartungshaltung* am besten durch die gemeinsame Diskussion von Beispielen: Zeigen Sie die jeweiligen Teile von arc42 und beschaffen sich dazu Feedback!

2 Randbedingungen

ID	Beschreibung
RB-1	HtmlSC soll in Java oder Groovy implementiert werden und auf den verbreiteten Betriebssystemen (Windows(TM), Linux, and Mac-OS(TM)) laufen.
RB-2	HtmlSC soll mit dem Buildwerkzeug Gradle integriert werden (als Plugin), ebenso mit Maven.
RB-3	HtmlSC muss unter einer liberalen Open-Source-Lizenz stehen, etwa einer Creative-Commons ShareAlike oder der Apache-Lizenz. Sämtliche benötigten Bibliotheken oder sonstige zur Entwicklungs- und Laufzeit benötigten Bestandteile müssen kompatibel mit der Creative-Commons Sharealike Lizenz (oder vergleichbaren Lizenzen) sein.
RB-4	Code und Dokumentation sollen auf einer der namhaften Open-Source-Plattformen (etwa: Github oder Bitbucket) gehostet werden.
RB-5	Fehler und neue Anforderungen sollen mit einem Issue-Tracker (etwa Github-Issues) gepflegt werden und öffentlich einsehbar sein.

Motivation

Sie sollten Ihre Freiheitsgrade bezüglich Entwurfsentscheidungen kennen, aber auch wissen, welche Randbedingungen für die Entwicklung und/oder das System gelten.

Nur in seltenen Fällen können Sie Randbedingungen mit Ihren Stakeholdern verhandeln.

Was und wie

Als Randbedingungen gelten sämtliche Faktoren, die Ihre Freiheit bei Entwurfsentscheidungen beschränken. Sie bekommen diese Einschränkungen meist von Auftraggebern oder anderen Stakeholdern (siehe Kapitel 1.3) vorgegeben.

Sie können technische (z. B. RB-1, RB-2) oder organisatorische (z. B. RB-3, RB-4) Randbedingungen vorfinden oder auch geltende Konventionen (z. B. Programmierrichtlinien, Dokumentations-, Namens- oder Ordnungskonventionen).

Eine einfache Tabelle genügt für die meisten Randbedingungen.

Tipps

- Softwarearchitekten sollten die Konsequenzen von Randbedingungen erkennen – beispielsweise resultierende (Mehr-)Kosten oder Aufwände.
- Falls Randbedingungen unangemessene Konsequenzen mit sich bringen (beispielsweise nur mit übermäßig hohem Aufwand erfüllbar sind), sollten Sie mit den entsprechenden Stakeholdern darüber verhandeln.
- Randbedingungen gelten in Organisationen oftmals über Systemgrenzen hinweg. Falls Sie für Ihr System keine Randbedingungen kennen, beginnen Sie Ihre Suche bei anderen Systemen innerhalb der Organisation.

3 Kontextabgrenzung

3.1 Fachlicher Kontext

Nachbar	Beschreibung
user	Erstellt technische Dokumentation mit einer Werkzeugkette, die Html erzeugt. Muss sicherstellen, dass Verweise innerhalb dieser Dokumentation korrekt sind.
build system	Gradle und/oder Maven.
local HTML file(s)	HtmlSC prüft lokale HTML-Dateien und benötigt daher Zugriff auf diese Dateien.
local images	HtmlSC prüft Verweise auf lokal vorhandene Bilder.
external websites & resources	HtmlSC kann auch für die Prüfung externer Verweise konfiguriert werden. Dann muss HtmlSC per http-GET oder http-HEAD auf diese externen Ressourcen zugreifen. **Risiken**: Durch externe Faktoren (Verfügbarkeit des Netzes generell, Netzwerklatenz, Verfügbarkeit/Konfiguration des betreffenden Servers) können solche Prüfungen lange dauern und sogar ungültige Resultate liefern.

Motivation

An den Schnittstellen zwischen einem System und seiner Umgebung treten Fehler besonders gerne und häufig auf. Die Schnittstellen zu Nachbarsystemen gehören zu den kritischsten Aspekten jedes Systems. Stellen Sie daher sicher, dass Sie die Außenschnittstellen komplett verstanden haben.

Was und wie

Der fachliche Kontext zeigt alle fachlichen Nachbarn des betrachteten Systems mitsamt den fachlichen Daten, Ereignissen oder Nachrichten, die ausgetauscht werden. Falls nötig, können Sie zusätzlich Datenformate und Protokolle der Kommunikation mit Nachbarsystemen und der Umwelt annotieren, falls diese nicht erst bei den betroffenen Bausteinen präzisiert werden.

Tipps

- Benutzen Sie den fachlichen Kontext als Feedback-Instrument für möglichst viele der Stakeholder (siehe Kapitel 1.3).
- Stellen Sie durch dieses Feedback sicher, dass Sie *alle* fachlichen Nachbarsysteme identifiziert haben.
- Suchen Sie an den externen Schnittstellen nach Risiken: Wo droht Ihrem System hohe Volatilität, hohe (inhaltliche oder technische) Komplexität, wo drohen besondere Kosten oder Aufwände?
- An Schnittstellen können besondere Qualitätsanforderungen gelten, etwa Durchsatz-, Sicherheits- oder Verfügbarkeitsanforderungen. Diese sogenannten „Service Levels“ können sehr hohe Implementierungs- und Betriebsaufwände und -risiken nach sich ziehen – daher ist besondere Vorsicht geboten.
    - Markieren Sie solche Risiken im Kontextdiagramm (in unserem Beispiel kann die Prüfung externer Websites lange dauern und ist daher kritisch für unsere Performanceanforderung).
    - Diskutieren Sie solche Risiken mit Ihrem Management, Ihren Product-Ownern oder sonstigen Beteiligten.

3.2 Technischer- oder Verteilungskontext

Das folgende Diagramm zeigt die beteiligten Computer (Knoten, engl. Nodes) mit ihren technischen Kanälen, sowie die wesentlichen Artefakte bei Nutzung und Entwicklung von HtmlSC. Weitere Details finden Sie in Kapitel 7.

Knoten / Artefakt	Beschreibung
hsc-development	Dort findet die Entwicklung von HtmlSC statt.
hsc-plugin-binary	Kompilierte und gepackte Version von HtmlSC, inklusive benötigter Abhängigkeiten.
artifact repository	Ein globales, öffentliches Cloud-Repository für binäre Artefakte, wie MavenCentral (https://search.maven.org/), dem Gradle Plugin Portal (https://plugins.gradle.com) o.Ä. Binaries von HtmlSC werden auf diesen Server hochgeladen.
hsc user computer	Dort wird eine beliebige HTML-Dokumentation erzeugt.
build.gradle	Gradle Build-Script, welches (neben anderen Dingen) das HtmlSC-Plugin konfiguriert, die HTML-Überprüfung durchzuführen.

Motivation

Manchmal trägt es zum Verständnis des Systems bei, die beteiligte Infrastruktur wie Computer, Prozessoren und Devices im Überblick zu sehen, zusammen mit den technischen Übertragungswegen (Kanäle, Protokolle).

Was und wie

Der technische Kontext zeigt die Infrastruktur, also insbesondere die Prozessoren oder Computer, auf denen Ihr System und die relevanten Nachbarsysteme ablaufen, zusammen mit den physischen Übertragungskanälen zwischen den Infrastrukturkomponenten.

Zeigen Sie eine grafische Übersicht und erläutern Sie die darin enthaltenen Elemente tabellarisch.

Tipps

- Diese Informationen sowie weitere Details könnten Sie auch in der Verteilungssicht (siehe Kapitel 7) darstellen.
- Eventuell unterscheiden sich die Zielgruppen des fachlichen und des technischen Kontexts. Im Beispiel sehen Sie im technischen Kontext den Rechner des HtmlSC-Entwicklers – der im fachlichen Kontext nicht auftaucht.
- Manchmal gibt es unterschiedliche Varianten des technischen Kontextes (oder des Deployments) – etwa für unterschiedliche Nutzungs- oder Einsatzszenarien Ihres Systems.
- Der technische Kontext besitzt insbesondere für hardwarenahe Systeme (z.B. Embedded-Systeme) hohe Bedeutung. Bei Informationssystemen (etwa Web- oder Desktop-Anwendungen oder mobilen Systemen) können Sie auf den technischen Kontext normalerweise verzichten.

4 Lösungsstrategie

1. HtmlSC wird in Groovy und Java implementiert, mit Fokus auf minimalen externen Abhängigkeiten.
2. Damit HtmlSC leicht in automatischen Builds eingesetzt werden kann, verpacken wir die Implementierung auch als Gradle-Plugin. Details beschreibt der Gradle Userguide[5]. Das entsprechende Plugin für Maven befindet sich aktuell noch in Konzeption.
3. Die Flexibilität bezüglich der verschiedenen Prüfalgorithmen wird durch das *Template-Method-Pattern*[6] erreicht. Das erlaubt HtmlSC mehrere unterschiedliche Prüfalgorithmen. Mehr Details beinhaltet das entsprechende Konzept in Kapitel 8.
4. HtmlSC verwendet die (gut standardisierten) Mechanismen von Gradle zur Konfiguration. Dabei stehen alle relevanten Informationen in einer einzigen, zentralen Konfigurationsdatei. Für das geplante Maven-Plugin muss HtmlSC von diesem Konzept möglicherweise abweichen.

[5] https://docs.gradle.org/current/userguide/userguide.html
[6] https://sourcemaking.com/design_patterns/template_method/

Motivation

Bevor Ihre Leserinnen und Leser in die Details der Architektur (arc42-Kapitel 5 bis 9) einsteigen, bietet Ihnen die Lösungsstrategie die Gelegenheit, grundlegende strategische Entscheidungen für die Architektur zu kommunizieren. So haben Sie die Möglichkeit, Ihre Lösung auf abstrakter Ebene zu skizzieren, die für alle Stakeholder verständlich ist, ohne sich in Details zu verlieren.

Was und wie

Auf maximal ein bis zwei Seiten sollten Sie wesentliche Technologieentscheidungen festhalten oder Strategien erläutern, die zur Erreichung der Qualitätsziele hilfreich oder aufgrund kritischer Randbedingungen notwendig sind. Diese Strategieentscheidungen sollten solide durchdacht, praktisch verifiziert und ziemlich unumstößliche Festlegungen sein, da von ihnen viele Folgeentscheidungen abhängen.

Es sind die „Eckpfeiler“ Ihrer Architektur.

Tipps

- Halten Sie diesen Teil kurz. Ausführliche Erklärungen und Begründungen können Sie oftmals auf Kapitel 8 (Querschnittliche Konzepte) verschieben.
- Die Lösungsstrategie sollte im Verlaufe der Entwicklung entstehen und wachsen: Sie können zunächst Teile des Systems implementieren (und erproben), bevor Sie die Schlüsselideen abstrahiert hier festhalten.
- Motivieren Sie Ihre gewählten Strategien: Die Gründe dafür finden sich oft in der Aufgabenstellung, den Qualitätszielen und/oder den Randbedingungen, oder aber in den Fähigkeiten und Technologiekenntnissen des Teams. Verweisen Sie auf aus der Lösungsstrategie auf ausführliche Erklärungen oder technische Details (oft erklärt in den nachfolgenden arc42-Teilen oder aber im Quellcode).

5 Bausteinsicht

5.1 HtmlSanityChecker (Whitebox, Level-1)

Whitebox (HtmlSC)

Begründung: Wir haben den Quellcode von HtmlSC nach funktionalen Aspekten strukturiert und alle Ein-/Ausgaben vom funktionalen Kern separiert.

Enthaltene Bausteine

Baustein	Beschreibung
`HSC Core`	HTML-Prüfungen und Parsing
`HSC Gradle Plugin`	Macht HtmlSC als Gradle Plugin verfügbar, im Gradle User Guide beschrieben. Code: `org.aim42.htmlsanitycheck`
`NetUtil`	prüft Internet-Connectivity, konfiguriert http-Statuscodes als Fehler, Warnung, usw.
`FileUtil`	Hilfsklassen für Dateitypen und -selektion
`HSC Graphical UI`	(geplant)

Motivation

Ihr System besteht aus einer möglicherweise sehr großen Menge an Quellcode – die Sie im Überblick behalten möchten.

Was und wie

Die Bausteinsicht erklärt den Aufbau des gesamten Quellcodes, hierarchisch (in Levels) organisiert.

Verwenden Sie jeweils ein Diagramm zur Erläuterung der Struktur einer Whitebox (wie etwa Abb. „Whitebox (HtmlSC)“ nebenan).

Tipps

- Sorgen Sie in jedem Fall dafür, die höchste Abstraktion (in arc42: Level 1) Ihres Quellcodes explizit zu beschreiben. Level 1 hilft Ihnen, den Aufbau von Systemen „im Großen“ zu verstehen.
- Level 1 der Bausteinsicht sollte bezüglich der (externen) Schnittstellen strikt konsistent zum fachlichen Kontext sein: Sämtliche externen Schnittstellen von Level 1 sollten im fachlichen Kontext ebenfalls vorkommen.
- Auf der hohen Abstraktionsebene von Level 1 können Sie High-Level Informationen (wie etwa Programmiersprache) oder auch organisatorische Informationen (wie etwa Teamzuständigkeiten oder im Beispiel „Fertigstellung“) darstellen.
- Ob Sie UML oder eine andere grafische Notation verwenden ist völlig egal, solange die Bedeutung der Symbole für Ihre Stakeholder klar ist. Im Zweifel fügen Sie eine Legende zum Diagramm hinzu!

5.2 Bausteinsicht Level-2

5.2.1 HSC Core (Whitebox)

HSC-Core (Whitebox)

Begründung: Die interne Struktur von `HSC Core` folgt funktionaler Zerlegung.

Enthaltene Bausteine (“Blackboxes”)

Baustein	Beschreibung
`Checker`	Enthält die eigentlichen Checks.
`AllChecksRunner`	konfigurierbare *Facade* zum Aufruf mehrerer/aller Checker.
`Configuration`	Konfiguration von Input- und Output-Verzeichnissen und -Dateien, Timeouts, Statuscode-Behandlung sowie den auszuführenden Checks.
`Reporter`	Berichtet die Prüfergebnisse.
`Suggester`	Schlägt für einige Checker im Fehlerfall mögliche Alternativen vor, die das jeweils gefundene Probleme beheben könnten (*meinten Sie xyz?*). Werden in Prüfergebnissen inkludiert.

Motivation (Level-2 und weitere)

An manchen Stellen Ihres Systems möchten Sie neben dem Überblick aus Level 1 noch weitere Details der inneren Struktur Ihres Systems zeigen, beispielsweise von einigen Bausteinen deren Schnittstellen oder Zusammenhang zum restlichen System.

Je weiter Sie Ihre Bausteinsicht verfeinern, desto mehr Wartungs- und Pflegeaufwand schaffen Sie für Ihre Dokumentation. Wir sind der Meinung, dass häufig eine Level-1 Beschreibung ausreicht, sofern Sie wichtige technische Details dann in Konzepten (arc42 Kapitel 8) ausführlicher erklären.

Was und wie (Level-2 und weitere)

Abschnitte der Bausteinsicht sind immer gleichartig aufgebaut – egal, welche Detailebene Sie erklären:

1. Die Whitebox-Darstellung zeigt mit Hilfe eines Diagramms die innere Struktur und interne Schnittstellen für eine „höhere" Blackbox.
2. Blackbox-Beschreibungen (meist tabellarisch, bei Bedarf als Text) erklären Zweck/Verantwortung sowie Schnittstellen der Bausteine des Whitebox-Diagramms. Hier können Sie direkt auf Quellcode verweisen. Bausteine sind dabei beliebige Abstraktionen von Quellcode: Module, Pakete, Subsysteme, Komponenten, Klassen, Funktionen, Skripte oder Konfigurationen.

Sie können zur Darstellung von Whitebox-Diagrammen UML verwenden, aber jede andere Notation funktioniert ebenfalls – sofern Ihre Stakeholder diese verstehen.

Tipps

- Konzentrieren Sie sich bei der Verfeinerung von Bausteinen selektiv auf interessante, besondere, kritische, riskante oder komplexe Bausteine.
- Lassen Sie „normale" Dinge (sogenannte 08/15-Bausteine) bei der Dokumentation besser weg.
- Sie können mehrere Blackboxes einer Ebene auch gleichzeitig verfeinern – das kann hilfreich sein, wenn diese Bausteine intensiv zusammenarbeiten und komplexe gegenseitige Schnittstellen besitzen.

5.3 Building Blocks - Level 3

5.3.1 Reporter (Whitebox)

Reporter (Whitebox)

Begründung: Strukturiert entlang der Hierarchie von Prüfungen (Durchlauf/„Run“, Seite/„Page“ sowie Prüfalgorithmus/„Check“).

Enthaltene Bausteine:

Baustein	Beschreibung
`PerRunResults`	Aggregierte Resultate für eine oder mehrere HTML-Dokumente.
`SinglePageResults`	Aggregierte Resultate für eine einzelne HTML-Seite/-Datei.
`SingleCheckResults`	Resultate für einen einzelnen Check einer einzelnen Seite (z.B. missing-images-check oder broken-internal-link-check)
`Finding`	Ein einzelnes gefundenes Problem oder Warnung, (z.B. „image ‘arc42-logo.png’ missing“). Kann `Suggestions` enthalten.

Tipps (für Bausteinsicht allgemein)

- Bei der Erklärung von Whiteboxen in der Bausteinsicht bieten sich Verweise auf die Laufzeitsicht an, um das dynamische Verhalten von Bausteinen und deren Schnittstellen zu beschreiben (siehe Abschnitt 6).
- Ausgehend von der Kontextabgrenzung (Abschnitt 3) können Sie in der Bausteinsicht Ihren Quellcode beliebig detailliert erklären. Dabei können Sie Ihren Aufwand selbst steuern – indem Sie nur an wenigen Stellen detailliert arbeiten. In der untenstehenden Abbildung sehen Sie eine Hierarchie solcher Sichten für unser Beispiel.
- Die Whitebox-Diagramme bilden einen Baum mit der Kontextabgrenzung als Wurzel. Dieser Baum kann und sollte asymmetrisch sein, d. h. nur an wenigen Zweigen in die Tiefe gehen.

Hierarchie der Bausteinsichten

6 Laufzeitsicht

6.1 Führe alle Prüfungen aus

Das wichtigste Szenario für HtmlSC ist die Ausführung aller Prüfungen (Checks).

Erläuterung:

0. Das Build-Target `htmlSanityCheck` wird ausgeführt.
1. Gradle ruft `sanityCheckHtml`
2. HSC konfiguriert Input-Dateien und Output-Verzeichnis
3. HSC erzeugt eine `AllChecksRunner` Instanz und
4. sammelt alle konfigurierten Input-Dateien in `allFiles`
5. (geplant) sammelt alle verfügbaren `Checker` Unterklassen anhand ihrer Annotationen
6. führt die eigentlichen Checks aus

Motivation

Sie sollten verstehen, wie die Bausteine Ihres Systems zur Laufzeit ihre jeweiligen Aufgaben erfüllen und wie Instanzen von Bausteinen zur Laufzeit miteinander kommunizieren.

Was und wie

Die Laufzeitsicht zeigt konkrete Abläufe und Beziehungen, meist in unterschiedlichen Szenarien.

Für solche Szenarien bieten sich folgende typische Fälle an: * Normalfall wichtiger Anwendungsfälle * Interaktionen an kritischen (externen) Schnittstellen * Fehler- oder Ausnahmeszenarien

Verwenden Sie beispielsweise Sequenz- oder Aktivitätsdiagramme, nummerierte Listen oder Pseudo-Code, am besten mit kurzen Erläuterungen.

Tipps

- Beschränken Sie sich in der Dokumentation der Laufzeitsicht auf wenige Szenarien. Fokussieren Sie auf die Besonderheiten des Systems, auf kritische, interessante oder riskante Abläufe oder die für das Verständnis wesentlichen!
- Stellen Sie sicher, dass aus der Beschreibung von Laufzeitszenarien die beteiligten Bausteine klar hervorgehen.
- Vorsicht vor zu vielen Details. Sie dürfen auch in Abläufen abstrahieren und sich auf wesentliche, interessante und kritische Teile beschränken.

7 Verteilungssicht

HtmlSC deployment (for use with Gradle)

Knoten/Artefakt	Beschreibung
hsc plugin binary	Kompilierte Version von HtmlSC.
artifact repository	Öffentliches Repository für binäre Artefakte, ähnlich wie MavenCentral
hsc user computer	Hier wird Dokumentation als HTML generiert und durch HtmlSC geprüft.

HtmlSC User benötigen eine Java Runtime (> 1.6) sowie das Buildfile `build.gradle`. Nachfolgend ein Auszug daraus:

{title="HtmlSC in `build.gradle` konfigurieren",lang=groovy} ~~~~~~~~ apply plugin: 'org.aim42.htmlSanityCheck' htmlSanityCheck { sourceDir = new File("${buildDir}/html5") sourceDocuments = ["many-errors.html", "no-errors.html"] failOnErrors = false httpConnectionTimeout = 2000 ignoreLocalHost = false .... ~~~~~~~~

Motivation

Software benötigt technische Infrastruktur, Rechner, Speicher, Netzwerke etc. Insbesondere wenn ein Softwaresystem auf mehr als einem Knoten (Rechner, Prozessor etc.) abläuft, sollten Sie die Verteilung der Software auf diese Knoten erklären.

Was und wie

Die Verteilungssicht erfüllt zwei Zwecke: 1. Sie zeigt die Infrastrukturelemente (in UML „Knoten" genannt) und deren physische Verbindungskanäle. 2. Sie dokumentiert das Mapping der Bausteine aus der Bausteinsicht (oder präziser: der Artefakte, die aus den Bausteinen entstehen) auf die Infrastrukturelemente.

Häufig laufen Systeme in unterschiedlichen Umgebungen ab, beispielsweise Entwicklung, Test und Produktion/Wirkbetrieb. In solchen Fällen sollten Sie im Idealfall diese Umgebungen aufzeigen, zumindest aber deren relevante Unterschiede explizit festhalten.

Verwenden Sie ein Diagramm (UML-Deployment Diagramme eignen sich gut!) mit textueller Erläuterung.

Tipps

- Das Mapping von Software auf Hardware kann grafisch erfolgen wie in unserem Beispiel links (Zeichnen der Artefakte direkt in die Infrastrukturboxen) oder aber durch eine tabellarische Zuordnung.
- Bleiben Sie auch hier sparsam: Zeichnen und beschriften Sie nur so viel in den Diagrammen, wie Sie mit Ihren Stakeholdern diskutieren wollen.
- Für Infrastruktur zeichnen sich oftmals andere Stakeholder als wir Softwarearchitektinnen und -architekten verantwortlich.
- Arbeiten Sie schon *Cloud-native*? Dann können Sie, getreu dem Motto „Infrastructure-as-Code[7]" Ihre Verteilung sehr konkret und detailliert als Mischung

[7] „Infrastructure as code, or programmable infrastructure, means writing code (which can be done using a high level language or any descriptive language) to manage configurations and automate provisioning of infrastructure in addition to deployments. This is not simply writing scripts, but involves using tested and proven software development practices that are already being used in ap-

aus Code und Konzept beschreiben. Die Details dazu sprengen leider den Rahmen dieses kleinen Buches.

plication development.“ Zitiert aus https://www.thoughtworks.com/de/insights/blog/infrastructure-code-reason-smile

8 Querschnittliche Konzepte

8.1 Mehrere Prüfalgorithmen

HtmlSC verwendet das Template-Method Pattern[8], um mehrere Prüfalgorithmen auf ähnliche Weise zu implementieren. Die abstrakte Oberklasse `Checker` definiert dazu die Methode `performCheck` und delegiert die eigentlichen Prüfalgorithmen an Unterklassen von `Checker`.

{title="Template Method für einen Check",lang=groovy} ~~~~ public CheckingResults-Collector performCheck() { initResults() return check() // subclass performs actual checking } ~~~~

Template Method (Auszug)

Komponente	Beschreibung
Checker	Abstrakte Basisklasse, enthält die Template-Methode `check()`.
`ImageFileExistChecker`	Prüft, ob lokale Bild-Dateien existieren.
`DuplicateIdChecker`	Prüft, ob eine ID mehrfach definiert ist.
...	Es gibt noch einige weitere Checker-Unterklassen.

[8] https://sourcemaking.com/design_patterns/template_method/

Motivation

Manche Entscheidungen betreffen mehr als einen Baustein und sind querschnittlich für Ihre Architektur relevant. Dafür enthält arc42 den Abschnitt „Konzepte“.

Was und wie

Ein Konzept besteht meist aus einem Lösungsansatz, gepaart mit ganz konkreten Technologieentscheidungen.

Wählen Sie nur diejenigen aus, die Sie zentral entscheiden möchten oder müssen, und dokumentieren Sie Ihre Konzepte mit Diagrammen, mit Source-Code-Ausschnitten und mit erläuternden Texten.

Tipps

- Erarbeiten Sie Konzepte in enger Zusammenarbeit mit dem Entwicklungsteam und/oder Technologieexperten.
- Manchmal besteht ein Konzept nur aus einer kompakten Formulierung einer Technologieentscheidung (Beispiel: „Hibernate als Object/Relational-Mapper“), manchmal umfassen Konzepte mehrere Seiten detaillierter (wie beispielsweise ein Sicherheitskonzept mit typischen Bedrohungsszenarien und passenden Lösungen).
- Konzepte können und sollen helfen, die Implementierung zu *regeln*, d.h. konkrete Vorgaben für einzelne oder mehrere Bausteine zu machen.
  In unserem Beispiel regelt das Konzept „Template-Method“ (siehe nebenan), wie die Implementierung *aller* Subklassen von `Checker`aussehen muss.
- Konzepte enthalten oftmals die meisten (gerne technischen) Details und können helfen, die Bausteinsicht (Abschnitt 5) deutlich kompakter zu halten.

8.2 Konzept „Reporting“

Für das Verständnis der Implementierung von Reporting sollten Sie die Begriffe „RunResult“, „PageResult“ sowie „SingleCheckResult“ verstehen. Dazu dient folgender Beispiel-Report:

Beispiel-Report mit den Kernbegriffen

Die Implementierung des Reportings finden Sie in den Packages `collect` und `report`.

Tipps

- „Konzepte“ im Sinne von arc42 umfassen Prinzipien, Regeln, Patterns, Vorgehensweisen, Implementierungshinweise, Regularien, Taktiken, Ansätze oder Aspekte.
- arc42 schlägt mehr als 20 solcher wiederkehrenden Themen wie Persistenz, Transaktionsbehandlung, Ausnahmebehandlung, Internationalisierung, Sicherheit usw. vor (siehe nachstehende Abbildung), aus denen Sie nur eine vermutlich kleine Zahl der für Ihr System relevanten auswählen sollten.

Vorschläge für Konzept-Themen

9 Entwurfsentscheidungen

9.1 Überprüfen von externen Links verschoben

In der aktuellen Version von HtmlSC werden wir externe Links nicht überprüfen. Diese Überprüfungen werden auf eine spätere Version verschoben.

9.2 HTML Parsen mit jsoup

Um den HTML-Code zu überprüfen, parsen wir ihn in eine interne (DOM-ähnliche) Repräsentation. Dafür verwenden wir Jsoup[9], einen Open-Source Parser ohne externe Abhängigkeiten.

Ein Zitat von der Website von Jsoup:

> **jsoup is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jQuery-like methods.**

Ziele dieser Entscheidung:

Das programmatische Überprüfen von HTML über ein bestehendes API, das Zugriffs- und Suchmethoden für den DOM-Baum der zu prüfenden Datei(en) bietet.

Entscheidungskriterien:

- Wenig Abhängigkeiten, so dass die Binärdatei des HtmlSC so klein wie möglich bleibt.
- Sehr benutzerfreundlich: Einfache und elegante (Zugriff, Suche) Methoden, um schnell Bilder, Links und Linkziele innerhalb des DOM-Baums zu lokalisieren.
- Äußerst flexibel: Kann sowohl Dateien als auch Strings (und andere Quellen) parsen.

[9] https://jsoup.org

Alternativen:

- jsoup: ein einfacher HTML-Parser ohne Abhängigkeiten (!) und mit einem umfangreichen API, um auf alle HTML-Elemente in einer DOM-ähnlichen Syntax zuzugreifen. Der klare Gewinner!
- HTTPUnit: Ein Test-Framework für Webanwendungen und Webseiten. Der Fokus liegt auf dem Testen von Webanwendungen, daher bringt es eine hohe Anzahl von Abhängigkeiten mit sich.
- HtmlCleaner[10]

[10] http://htmlcleaner.sourceforge.net/

Motivation

Stakeholder des Systems sollten wichtige Entscheidungen verstehen und nachvollziehen können.

Was und wie

Wichtige, teure, große oder riskante Architektur- oder Entwurfsentscheidungen inklusive der jeweiligen Begründungen. Mit „Entscheidungen“ meinen wir hier die Auswahl einer von mehreren Alternativen unter vorgegebenen Kriterien.

Für die Form der Entscheidungen haben Sie verschiedene Möglichkeiten: * Liste oder Tabelle, nach Wichtigkeit und Tragweite der Entscheidungen geordnet * ausführlicher in Form einzelner Unterkapitel je Entscheidung * ADR (Architecture Decision Record[11]) für jede wichtige Entscheidung

Tipps

- Wägen Sie ab, inwiefern Sie Entscheidungen hier zentral beschreiben, oder wo eine lokale Beschreibung (z.B. in der Whitebox-Sicht von Bausteinen) sinnvoller ist.
- Vermeiden Sie Redundanz. Verweisen Sie evtl. auf Abschnitt 4, wo schon grundlegende strategische Entscheidungen beschrieben wurden.
- Vergessen Sie nicht, zu jeder Entscheidung Alternativen und deren Bewertung unter Auswahlkriterien zu dokumentieren. Nur so lassen sich die Entscheidungen später nachvollziehen.

[11] http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions

10 Qualitätsszenarien

10.1 Qualitätsbaum

Anmerkung: Für unser kleines Beispiel ginge so ein Qualitätsbaum viel zu weit. Jedoch haben wir in Systemen aus dem wirklichen Leben Qualitätsbäume mit mehr als 100 Qualitätsszenarien gesehen. Daher halten wir uns daran, ein paar Szenarien (wiederholend) aufzuführen.

10.2 Qualitätsszenarien

Qualitätsmerkmal	Szenario
Korrektheit	Jeder defekte interne Verweis (Cross-Referenz) wird gefunden.
Korrektheit	Jedes fehlende (lokale) Bild wird gefunden.
Korrektheit	Die Korrektheit aller Überprüfungen wird durch automatisierte Positiv- und Negativtests sichergestellt.
Vollständigkeit	Der Ergebnisbericht muss *alle* Ergebnisse (auch Findings genannt) beinhalten.
Flexibilität	HtmlSC soll so gestaltet sein, dass es um neue Prüfalgorithmen und neue Anwendungsszenarien (z.B. von anderen Build-Systemen) erweitert werden kann.
Sicherheit	HtmlSC belässt alle Quelldateien in ihrer ursprünglichen Form: Die Inhalte der überprüften Dateien werden *niemals* verändert.
Performance	Prüfung einer 100kB großen HTML-Datei in unter 10 Sekunden (Exklusive der Startup-Zeit von Gradle).

Qualitätsbaum

Motivation

Die mit Prioritäten versehene Baumstruktur gibt Überblick über die – oftmals zahlreichen – Qualitätsanforderungen.

Was und wie

Der Qualitätsbaum (à la ATAM) ist eine baumartige Verfeinerung des Begriffes „Qualität“, mit „Qualität“ oder „Nützlichkeit“ als Wurzel. An den Blättern des Baumes finden sich jeweils die Qualitätsszenarien. Als Darstellungsform eignet sich beispielsweise eine Mindmap mit den Qualitätsoberbegriffen als Hauptzweigen.

Tipps

- In jedem Fall sollten Sie hier Verweise auf die Qualitätsszenarien des folgenden Abschnittes aufnehmen.
- Nehmen Sie hier auch Qualitätsanforderungen geringerer Priorität auf, deren Nichteinhaltung oder -erreichung geringe Risiken birgt.

Qualitätsszenarien

Motivation

Qualitätsszenarien operationalisieren Qualitätsanforderungen und machen deren Erfüllung mess- oder entscheidbar.

Was und Wie

Konkretisierung der (in der Praxis oftmals vagen oder impliziten) Qualitätsanforderungen durch Qualitätsszenarien. Diese Szenarien beschreiben, was beim Eintreffen eines Stimulus auf ein System in bestimmten Situationen geschieht. Wesentlich sind zwei Arten von Szenarien: * Anwendungsfallszenarien beschreiben, wie das System zur Laufzeit auf einen bestimmten Auslöser reagieren soll. Hierunter fallen auch

Szenarien zur Beschreibung von Effizienz oder Performance. Beispiel: Das System beantwortet eine Benutzeranfrage innerhalb einer Sekunde. * Änderungsszenarien beschreiben eine Modifikation des Systems oder seiner unmittelbaren Umgebung. Beispiel: Eine zusätzliche Funktionalität wird implementiert oder die Anforderung an ein Qualitätsmerkmal ändert sich.

Tipps

Insbesondere wenn Sie die Qualität Ihrer Architektur mit Methoden wie ATAM überprüfen wollen, bedürfen die in Kapitel 1.2 genannten Qualitätsziele einer weiteren Präzisierung bis auf die Ebene von diskutierbaren und nachprüfbaren Szenarien.

11 Risken und technische Schulden

Hinweis
Für unser kleines Beispiel sehen wir keine wirklich Risiken bei Architektur und Implementierung - daher wirken die unten genannten Risiken sicherlich etwas künstlich.

11.1 Technische Risiken

Risiko	Beschreibung	Status
Personeller Engpass	Aktuell hat nur ein einzelner Entwickler Zugriffsrechte, um neue Versionen von HtmlSC auf öffentlichen Servern wie Bintray oder dem Gradle Plugin Portal veröffentlichen zu können.	Erledigt, seit einiger Zeit kann der Travis-CI Build Server selbständig Releases erzeugen.
Großer Aufwand notwendig, um neuere Versionen von Gradle einzusetzen.	Das Update von Gradle v-3.x auf v-4.x hatte notwendige Änderungen an der Konfiguration von HtmlSC zur Folge. Dieser Aufwand könnte bei zukünftigen Updates der Gradle API erneut anfallen.	

11.2 Fachliche Risiken

Risiko	Beschreibung
Das System könnte überflüssig werden.	Für den Fall, dass AsciiDoc- oder Markdown-Prozessoren das Überprüfen von HTML nativ umsetzen, könnte HtmlSC obsolet werden.

Risiko	Beschreibung

Motivation

„Risikomanagement ist Projektmanagement für Erwachsene“
– (Tim Lister, Atlantic Systems Guild.)

Unter diesem Motto sollten Sie Architekturrisiken und/oder technische Schulden gezielt ermitteln, bewerten und Ihren Management-Stakeholdern (z.B. Projektleitung, Product-Owner) transparent machen. Denn nur wenn Sie die technischen Risiken Ihres Systems kennen, können Sie gleich mögliche zukünftige Probleme adressieren.

Was und wie

Eine nach Prioritäten geordnete Liste der erkannten Architekturrisiken und/oder technischen Schulden. Schreiben Sie diese als Tabelle von Risiko oder technischen Schulden, eventuell mit vorgeschlagenen Maßnahmen zur Risikovermeidung, Risikominimierung oder dem Abbau der technischen Schulden.

Tipps

- Unterscheiden Sie zwischen fachlichen und technischen Risiken.
- Verwenden Sie eine dreispaltige Tabelle, so dass Sie für Risiken einen Status dokumentieren können, z.B. „Nicht mehr relevant“, „behoben“ oder „in Arbeit“.
- Schreiben Sie auch geringere Risikos auf. Je mehr Sie von Anfang an dokumentiert haben, umso einfacher können Sie mit den Stakeholdern über Schritte zur Risikominimierung oder -beseitigung diskutieren.

12 Glossar

Grundbegriffe für Prüfung von HTML

Begriff (EN)	Deutsch	Erklärung
Anchor	Anker	Html Element für Links. Enthält Sprungziele der Form `<a href="sprungziel">`
Finding	-	Beschreibung eines von HtmlSC gefundenen Problems innerhalb einer Html Page.
HTML Page	-	Ein Stück HTML, in der Regel eine einzelne Datei. Sollte der Standard HTML Syntax genügen. Minimale Anforderung: Der JSOUP HTML Parser kann diese Seite erfolgreich parsen. Synonym: *HTML-Document*.
Internal Link	-	Verweis auf einen anderen Teil derselben Seite, oder zu einer anderen Seite derselben Domäne. Auch genannt Cross Reference oder *Lokaler Verweis*.
Link	Verweis	Ein Verweis in einer HTML Page, der auf einen anderen Teil derselben Seite (Internal Link) oder ein anderes Dokument, Bild oder Ressource verweist External Link).

Begriff (EN)	Deutsch	Erklärung
Link Target	Sprungziel	Ziel eines Link, z.B. eine Überschrift oder ein anderer Teil einer HTML Page, oder jegliche interne oder externe Resource, die über einen URI identifiziert wird.
Run Result	-	Die Ergebnisse der Prüfungen von HtmlSC von einer oder mehreren HTML Pages.
Single Page Result	-	Die Menge aller Prüfergebnisse einer einzigen HTML Page.
URI	-	Universal Resource Identifier. Erklärt in RFC-2396[12], der ultimativen Wahrheit betreffend Syntax und Semantik von Hyperlinks und ähnlichem Web-Zeugs.

[12] https://www.ietf.org/rfc/rfc2396.txt

Motivation

Ein Entwurf ist nur so verständlich wie die Begriffe, die Sie darin verwenden. Insbesondere neu hinzukommende Personen müssen die wichtigsten Begriffe mit der für das System spezifischen Semantik verstehen.

Das Glossar ist eine Ausprägung unseres Ratschlags „*besser explizit als implizit*“ – weil es Begriffen explizit Bedeutung zuordnet.

Was und wie

Eine alphabetische Liste oder Tabelle mit den wichtigsten Begriffen, die in Entwurf oder Implementierung verwendet wurden. Alternativ auch ein fachliches Modell (*grafisches Lexikon*), das neben den Begriffen ebenso deren inhaltliche Zusammenhänge aufzeigt.

Das Glossar weist eine wesentliche Überdeckung mit der „ubiquitous language“ aus dem Domain-driven Design auf.

Tipps

- Nehmen Sie die hohe Bedeutung dieses Kapitels ernst! Falls Sie uns nicht glauben: Lassen Sie einige Personen Ihres Teams unabhängig voneinander die wichtigsten drei Fachbegriffe in ein bis zwei Sätzen definieren. Falls Sie gravierende Unterschiede finden, hatten wir wohl Recht.
- Dieses Kapitel kann entfallen, wenn Sie anderweitig über ein gepflegtes Projektglossar verfügen (aber auch *nur* dann!).
- In agilen Projekten können Product-Owner das Glossar verantworten und pflegen, in eher klassischen Projekten dürfen das Projektleitung oder fachliche Verantwortliche übernehmen.
- Halten Sie die Anzahl der hier erklärten Begriffe im Rahmen: Oft haben wir mit zehn bis 30 Begriffen genug Einführung geben können. Sie wollen keine Enzyklopädie schreiben.

4 Tipps für die Praxis

Den Aufbau von arc42 haben Sie verstanden, das Beispiel von HtmlSC illustriert das Ganze, aber im täglichen Leben tauchen trotzdem noch Fragen auf.

Bevor Sie also loslegen, lassen Sie uns Ihnen noch einige Tipps für die praktische Arbeit mit arc42 geben. Danach beantworten wir einige der am häufigsten gestellten Fragen, für andere finden Sie unten Verweise auf praktische Informationsquellen aus dem arc42 Ökosystem (also frei verfügbar).

Die wichtigsten Tipps

arc42 selbst ist auf der Website https://docs.arc42.org Kapitel für Kapitel ausführlich erläutert. Zu jeder einzelnen Sektion finden Sie dort Tipps und Ratschläge für unterschiedliche Arten von Systemen, versehen mit Schlagworten (engl. *tags* oder *keywords*).

Die nachstehende Abbildung zeigt eine Übersicht dieser Schlagworte, nach denen Sie auf der docs.arc42.org Website natürlich auch suchen und filtern können. Da sollte für (fast) jeden Geschmack etwas Passendes auftauchen.

Schlagworte auf https://docs.arc42.org

Drei dieser Begriffe besitzen besondere Bedeutung: * **essential**[1]: Kennzeichnet Ratschläge, deren Missachtung über kurz- oder lang zu üblen Problemen führen könnte. * **lean**[2]: Bezeichnet Ratschläge für den besonders sparsamen Einsatz von arc42, etwa für eher risikoarme Systeme. Diese Tipps helfen, Zeit und Aufwand für Dokumentation zu sparen. * **thorough**[3]: Bezeichnet Tipps, die Ihnen bei kritischen oder besonders großen Systemen helfen können. Oftmals das Gegenteil der Spar-Tipps aus der Lean-Gruppe.

Einige der ganz wichtigen Tipps

1. Arbeiten Sie *immer* mit eindeutigen Qualitätsanforderungen. Erklären Sie diese mit Hilfe von Szenarien. Falls Sie keine Qualitätsanforderungen bekommen, treffen Sie Ihre Annahmen eindeutig. (Kapitel 1)
2. Suchen Sie breit gefächert nach Stakeholdern und beschreiben Sie deren Erwartungen. Am besten verwalten Sie die Stakeholder und deren Erwartungen in einer Tabelle. (Kapitel 1)
3. Zeigen Sie den Anwendungskontext als Diagramm und kombinieren Sie dieses mit einer Tabelle, in welcher Sie einzelne Elemente beschreiben. (Kapitel 3)
4. Beschränken Sie den Kontext auf eine Übersicht, vermeiden Sie zu viele Details. Zeigen Sie aber unbedingt alle (wirklich alle!) externen Schnittstellen auf. (Kapitel 3)
5. Beschreiben Sie ihre Lösungsansätze in Form einer Tabelle. (Kapitel 4)
6. Bauen Sie die Sichten der Building Blocks hierarchisch auf. Beschreiben Sie dabei *immer* mindestens den ersten Level der Building-Block-Sicht. (Kapitel 5)
7. Beschreiben Sie die Verantwortlichkeit oder den Zweck von jeder (wichtigen) Blackbox. Verstecken Sie den inneren Aufbau der Blackboxes. (Kapitel 5)
8. Erklären Sie die Zuordnung von Source-Code zu den einzelnen Building-Blocks. (Kapitel 5)
9. Dokumentieren Sie nur ein paar Laufzeit-Szenarien. (Kapitel 6)
10. Erklären Sie die Konzepte. (Kapitel 8)
11. Dokumentieren Sie mindestens das Geschäfts- oder Domänen(Daten-)modell. (Kapitel 8)

[1] https://docs.arc42.org/keywords/#essential
[2] https://docs.arc42.org/keywords/#lean
[3] https://docs.arc42.org/keywords/#thorough

12. Dokumentieren Sie Ihre Entscheidungskriterien und geben Sie Gründe für (oder gegen) wichtige Entscheidungen an. (Kapitel 9)
13. Nehmen Sie das Glossar ernst. Es kann extrem hilfreich sein. (Kapitel 12)

Die häufigsten Fragen

Muss ich alle Teile ausfüllen?

Gegenfrage: Müssen Sie einen Schrank unbedingt komplett füllen? Nein, natürlich nicht. Sie räumen die Dinge hinein, die hinein gehören.

Genauso verhält es sich mit arc42. Bearbeiten Sie **ausschließlich** diejenigen Teile, die für Ihr System relevant, wichtig, kritisch, besonders oder interessant sind.

Welche Teile von arc42 sollte ich mindestens bearbeiten?

Die essentiellen Teile

Falls Sie wenig Zeit haben und sich auf ein Minimum an Dokumentation beschränken wollen oder müssen, dann finden Sie unsere Meinung über die „Essenz“ in der vorstehenden Abbildung.

Gibt es arc42 auch für *mein* Werkzeug?

Das arc42-Team pflegt das Template im AsciiDoc Format, und generiert daraus über einen Build-Prozess eine ganze Menge unterschiedlicher Zielformate (siehe unten). Einige wenige Tools unterstützt die Community *manuell*. Sämtliche Formate von arc42 finden Sie im Download-Bereich von arc42[4].

[4] https://arc42.org/download

Die generierten Formate sind docx, Markdown (in verschiedenen Dialekten), LaTeX, ReStructuredText, HTML, Textile, Confluence und natürlich AsciiDoc selbst.

Manuell gepflegte Formate sind Enterprise-Architect und IBM-Rhapsody.

Darf ich arc42 auch für kommerzielle Zwecke einsetzen? Kostenfrei?

Ja und ja.

Sie dürfen für alle Arten von Systemen, Organisationen oder Anwendungsgebiete arc42 verwenden, ohne dass Sie dafür Lizenzgebühren entrichten müssen. Eine ausführliche Erklärung der verwendeten *Creative Commons* Lizenz finden Sie in den offiziellen FAQ[5].

Wo finde ich praktische Beispiele?

Falls Ihnen Kapitel 3 gefallen hat, finden Sie diverse weitere (deutlich umfangreichere) Beispiele im Buch arc42-by-Example. Als LeserIn dieses Buches bekommen Sie diese Beispiele sogar kostenfrei (siehe Literaturverzeichnis).

Dazu gehören eine Schach-Engine, ein Biking-Portal, eine große Datenmigration, ein ziemlich komplexes CRM-System sowie die docToolchain.

Muss ich UML verwenden?

Kurze Antwort: Müssen Sie nicht. Können Sie aber.

Eine längere Antwort liefert die FAQ, siehe Frage B-3[6]

[5] https://faq.arc42.org/questions/A-2/
[6] https://faq.arc42.org/questions/B-3/

Welche Alternativen gibt's zu arc42?

- Wir mögen Simon Brown's C4-Modell. Es ist pragmatisch und adressiert viele der Belange, die Sie für Architekturen kommunizieren und/oder dokumentieren sollten. Sie finden einen guten Einstieg hier[7].
- Wenn Sie Ihr eigenes Template für Ihre Architekturen entwickeln wollen, so müssen Qualitätsanforderungen und -ziele unbedingt hinein, ebenso die Kontextabgrenzung und die externen Schnittstellen. Und natürlich die wesentlichen Bausteine sowie die querschnittlichen Konzepte. Falls das bei Ihnen alles dabei ist, sind Sie übrigens schon ganz dicht bei arc42.
- Die schlechteste Alternative lautet, gar nichts über Ihre Architektur zu dokumentieren.

[7] https://www.voxxed.com/2014/10/simple-sketches-for-diagramming-your-software-architecture/

Quellen und Referenzen

arc42 by Example

Gernot Starke, Ralf D. Müller, Michael Simons, Stefan Zörner.

Die Architektur sechs realer IT-Systeme mit arc42 erklärt - auf über 250 Seiten. Erschienen als eBook bei Leanpub, zweite Auflage 2019.

Für Sie der kostenfreie Download-Gutschein: http://leanpub.com/arc42byexample/c/INNOQ42.

arc42 Dokumentation

arc42 selbst ist ausführlich unter https://docs.arc42.org dokumentiert. Diese Site enthält eine detaillierte Erklärung des Templates und fast 140 Tipps und Hinweise zur Nutzung.

arc42 FAQ

Die arc42 FAQ (*frequently asked questions*) beantworten unter https://faq.arc42.org über 130 solcher häufig gestellten Fragen.

arc42 in Aktion

Gernot Starke, Peter Hruschka: arc42 in Aktion. Carl-Hanser Verlag, 2017. Das *missing manual* für arc42. Sie können sich diesen Primer als eine super-Kurzfassung von "arc42 in Aktion" vorstellen – wobei wir (Ben und Gernot) für den Primer eine Menge Fehler beseitigt und Formulierungen vereinfacht haben.

arc42 Open Source

Sämtliche Dokumente, Websites, Templates und Beschreibungen von arc42 finden Sie frei zugänglich im Bereich der arc42 Github Organisation[8]. Dort können Sie uns

[8] https://github.com/arc42

Verbesserungsvorschläge melden, kritisieren, Anregungen geben oder auch eigene Beiträge als *pull request* loswerden.

Bildnachweis

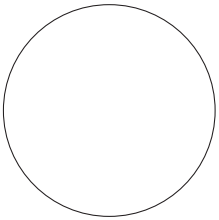
Das Titelbild wurde von Sonja Scheungrab (INNOQ)[9] für dieses Buch entworfen.

Der Schubladenschrank in Abschnitt 1.1 stammt von John Olson[10] aus seiner Sammlung bei OpenClipart[11].

[9] https://www.innoq.com/de/staff/sonja-scheungrab
[10] http://www.johnnyautomatic.com
[11] https://openclipart.org/detail/675/file-cabnet-drawers

Über die Autoren

Benjamin Wolf



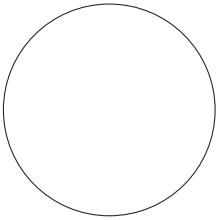
@ichaos1985

Benjamin Wolf (Senior Consultant bei INNOQ) ist Committer bei arc42 und hat das Template in diversen Beratungsmandaten eingesetzt.

Im Laufe der Jahre entwickelte er unterschiedliche Software in Java und .Net. Zu seinen Aufgaben gehören Konzeption von Architekturen, Entwickeln von Code und Refaktorieren von Legacy-Code. Softwarequalität sowie Kommunikation und Dokumentation von Softwarearchitekturen bilden seine aktuellen inhaltlichen Schwerpunkte.

2013 hat Benjamin die "CPSA-Foundation Level"-Zertifizierung des iSAQB e. V. erfolgreich absolviert. Seit 2018 unterstützt er dort die Arbeitsgruppen "Advanced Level" und "Hochschulen".

Gernot Starke



@gstarke

Dr. Gernot Starke (INNOQ Fellow) arbeitet seit über 25 Jahren in der Softwareentwicklung, heutzutage als Softwarearchitekt, Coach, Consultant und Trainer. Für Mandanten aus unterschiedlichen Branchen durfte er an Architektur, Entwurf und Implementierung großer und mittelgroßer IT-Systemen mitwirken, von denen zahlreiche auch nach langer Zeit noch produktiv laufen.

2005 hat Gernot gemeinsam mit Dr. Peter Hruschka das arc42-Template veröffentlicht und es seitdem in zahlreichen Entwicklungsprojekten eingesetzt und mehrere größere Systeme damit dokumentiert. Er pflegt und betreibt die arc42 Websites[1].

[1] https://arc42.org, https://docs.arc42.org, https://faq.arc42.org sowie die zugehörigen Repositories bei Github (https://github.com/arc42).

Gernot ist Gründungsmitglied des iSAQB e. V. und leitet dort die Arbeitsgruppen zum Foundation-Level. Er betreibt die (open-source) frei verfügbare aim42-Methode zur Verbesserung bestehender Systeme.

Er hat mehrere Bücher zum Thema Softwarearchitektur und Patterns geschrieben.