http://mbohlen.de/os0315/

Architektur flink dokumentieren: Geschrieben, gelesen, verstanden – Architekturdokumentation auf die schlanke Art

Agile Teams stellen sich die Frage: Wie stark sollen wir unsere Architektur dokumentieren, um agil zu bleiben? Dokumentieren wir zu wenig, so entstehen Missverständnisse und wiederholte Diskussionen. Dokumentieren wir zu viel, müssen wir ständig Dokumente pflegen, anstatt lauffähige Software zu schreiben. Es gibt darauf eine Antwort. Der Artikel zeigt auf, welches Maß an Dokumentation sinnvoll ist, wie ein schlanker Prozess dafür aussehen kann und welche Art von Toolkette für maximale Agilität sorgt.

Agil sein

Im agilen Manifest steht: „Wir schätzen funktionierende Software mehr als umfassende Dokumentation" (vgl. [Bec01]). Die Softwareentwickler, die sich im Februar 2001 in Utah trafen und das Manifest veröffentlichten, wussten, wovon sie redeten. Sequenzielle, schwergewichtige, plan- und dokumentengetriebene Entwicklungsprozesse waren zu der Zeit noch groß in Mode. Zum Teil wurde in diesen Prozessen mehr dokumentiert als Software geschrieben: Die Kunden bekamen lauffähige Software erst, wenn sie lange darauf gewartet und Änderungsanforderungen in anstrengenden Sitzungen eines *Change Control Boards* durchgefochten hatten.

Jim Highsmith schreibt in seinem Artikel zur Historie des Agilen Manifests: „Wir wollen eine Balance wiederherstellen. Wir schätzen Modellierung, doch nicht, um irgendein Diagramm in einem verstaubten Archiv der Firma abzulegen. Wir schätzen Dokumentation, aber nicht hunderte von Seiten in niemals aktualisierten und selten benutzten Schinken." (vgl. [Hig01]). Recht hatten die Väter des agilen Manifests. Jetzt, 14 Jahre später, sollten wir das Problem gelöst haben, oder?

Probleme

Mitnichten, denn es gibt immer noch typische Probleme bei der Architekturdokumentation. Warum ist es so schwer, angemessen zu dokumentieren und dadurch (oder trotzdem) agil zu bleiben?

Ohne Dokumentation zu langsam

Als Coach für Entwicklungsteams habe ich Projekte gesehen, die nicht vom Fleck kamen, weil sie zu wenig dokumentierten. In diesen Projekten herrschte eine Kultur der Unverbindlichkeit: Die Leute sprachen viel miteinander über ihre Architektur, trafen auch Entscheidungen dazu, doch sie schrieben sie nicht auf.

Nur wer beim Meeting dabei war, hatte genau mitbekommen, worum es ging. Allen anderen, insbesondere neuen Teammitgliedern, mussten sie immer wieder erklären, warum das System ist, wie es ist. Diskussionen führte man immer und immer wieder, in Abständen von ein paar Monaten, nur weil nicht festgehalten wurde, warum Architekturentscheidungen so gefällt worden waren.

Mit Dokumentation zu langsam

Auch habe ich Projekte erlebt, die fünf Jahre lang modellierten und dokumentierten und deshalb abgesagt wurden, kurz bevor sie endlich lauffähige Software hervorbringen konnten. Oder vermeintlich agile Teams mit Ordnern voller Dokumente, die vom Abstraktionsniveau her so dicht am Code gehalten waren, dass sie schneller veralteten, als die Teams reagieren konnten. Das beschäftigte und verlangsamte die beteiligten Teams so sehr, dass sie ihre Agilität verloren.

Prozessprobleme

Andere Teams hatten Schwierigkeiten mit dem Prozess: Dokumente erstellen, ablegen, wiederfinden, publizieren. Es war nicht klar, wer die Dokumentation zu schreiben hatte und wer sie konsistent zusammenführte. Beim Arbeiten mit mehreren Autoren überschrieben sich die Teammitglieder gegenseitig ihre Dokumente, weil sie nicht wussten, wo die aktuelle Version zu liegen hatte und woran man sie erkennt. Die Verwendung von Grafik-Werkzeugen war nicht standardisiert, also tauchten Fragen wie diese auf: Wo ist eigentlich die Originaldatei der Abbildung, die ich hier gerade im Wiki vor mir sehe? Welches Tool muss ich installieren, um sie bearbeiten zu können?

Kosten/Nutzen-Verhältnis

Das eindrucksvollste Problem bei der Architekturdokumentation zeigt sich meist erst am Ende: Die Dokumentation ist fertig, das Team ist stolz darauf – doch die Zielgruppen lesen die Dokumente nicht. Meist empfinden sie den Nutzen der Dokumente als nicht hoch genug und priorisieren in der Folgezeit die Dokumentation herunter. Immer wenn es terminlich eng wird, tendieren Teams dazu, die Dokumentation wegzulassen.

Vorteile

Dabei hat eine angemessene Architektur-Dokumentation deutliche Vorteile:

- Sie erleichtert die Einarbeitung neuer Teammitglieder. Die Alten sind in der Regel beschäftigt, die Neuen können die Dokumentation lesen und finden dort die meisten Fragen beantwortet.
- Die Dokumentation beantwortet die Fragen nach dem „Warum". Warum ist das System oder diese Komponente so? Das „Was" (z.B. Daten) und das „Wie" (z.B. Algorithmen und ausführbare Tests) stehen ja im Code – kein Problem. Doch die Entscheidungen, die dazu geführt haben, eben nicht und die Begründungen schon gar nicht. Kein anderes Artefakt außer der Architektur-Dokumentation leistet das.
- Die Dokumentation macht Architektur bewertbar. In regelmäßigen Abständen können Teams ein Review auf ihre Architektur machen und fragen: Ist sie angemessen? Leistet sie, was sie soll? Das geht nicht, wenn man keine einheitliche Vorstellung davon hat, was die Architektur ist.
- Die Dokumentation hilft, Fehler zu vermeiden. Beispiel: Eine Komponente wird um Dinge erweitert, die nicht ihrer Verantwortung entsprechen. Sie bläht

sich auf und wird unwartbar. Das können Sie verhindern, wenn jeder durch die Architektur-Dokumentation weiß, was die Verantwortung dieser Komponente ist und wie sie konzeptionell gedacht war. Aus dem Code gehen diese Prinzipien nicht hervor, immer nur die Details.

- Die Dokumentation hilft beim Denken. Wenn sich etwas denken, aber nicht verständlich aufschreiben lässt, ist es wahrscheinlich zu kompliziert und muss vereinfacht werden. Sie können mit Hilfe moderner Dokumentationsmethoden die Struktur der Dokumentation identisch zur Struktur des Entwurfs gestalten. So sparen Sie Zeit und Arbeit, indem Sie dokumentieren, *während* Sie entwerfen, nicht davor und nicht danach.
- Die Dokumentation unterstützt die Kommunikation im Projekt: Sie brauchen nicht immer alles erneut an die Tafel zu malen. Sie lernen aus dem Feedback, das andere Ihnen zu Ihrer Dokumentation geben, was die beste Art und Weise ist, eine bestimmte Stelle im Entwurf zu erklären, und wo die schwierigsten Stellen liegen. Gruppen kommen in der Kommunikation schneller weiter, wenn sie ein soziales Objekt wie die Dokumentation dazu nutzen können.

Kurz gesagt: Gute Architekturdoku unterstützt die Entwurfsarbeit, schafft Transparenz und bietet Leitplanken für die Umsetzung und Wartung der Software. Nur wenn man eine Dokumentation hat, kann man sich um die Architektur streiten und sich auf wirklich ein und dieselbe einigen.

Was in die Architekturdokumentation hineingehört

Wenn schlanke Architekturdokumentation solche Vorteile hat, lohnt es sich, diese Fragen konsequent zu beantworten:

- *Inhalt:* Was gehört in eine schlanke Dokumentation wirklich hinein?
- *Methode:* Wie viel Dokumentationsprozess brauchen wir wirklich?
- *Werkzeuge:* Welche Tools unterstützen einen schlanken Prozess und welche machen uns langsam?

Zuerst also zum Thema Inhalt: Wenn Sie die Architekturdokumentation schlank halten wollen, tun Sie sich einen Gefallen und nehmen Sie wirklich nur die Architektur (**siehe Kasten 1**) darin auf. Darunter fallen also zum Beispiel:

- Bausteine und deren Schnittstellen

> Architektur: „Die fundamentalen Konzepte oder Eigenschaften eines Systems in seiner Umgebung, verkörpert durch seine Elemente, Beziehungen und die Prinzipien seines Entwurfs und seiner Evolution“.

Kasten 1: Definition von Architektur nach ISO42010 (vgl. [ISO11]).

- die Zusammenarbeit dieser Bausteine zur Laufzeit
- der Einsatz der fertigen Bausteine auf der technischen Infrastruktur
- die technischen Konzepte für Persistenz, Logging, Authentisierung, Workflow, GUI usw.
- die wichtigen Entscheidungen, die zu obigen Themen getroffen wurden

Gute Vorlagen für diese Beschreibungen finden Sie z. B. bei Philippe Kruchten (vgl. [Kru95]), in arc42 (vgl. [Sta]) oder bei Simon Brown (vgl. [Bro12]).

Wann immer ich in Projekten veraltende Dokumentation gesehen habe, stellte ich den Teams die Frage: „Haben Sie außer der Architektur noch mehr hineingeschrieben?“ Meist war es tatsächlich so: Die Dokumentation enthielt z. B. den inneren Aufbau der Bausteine, bis hinunter auf Klassenebene, oder den genauen Aufbau der Domänenobjekte, bis auf das letzte Attribut und die letzte Methode hinab.

Das können Sie durchaus so schreiben. Doch dann haben Sie sehr detaillierte und damit sehr wartungsintensive Dokumente, die Sie genau so oft pflegen müssen wie Ihren Code. Das kann Ihre Agilität negativ beeinflussen. Überlegen Sie, ob Sie diese Dinge in Ihrer Architekturdokumentation wirklich benötigen oder ob für die Details nicht der Code und die Tests ausreichen. In vielen Fällen können die detaillierten Dokumente automatisch generiert werden, z. B. aus dem Code oder der Datenbank.

In der Architekturdokumentation wollen Sie eher die Dinge sehen, die Ihnen kein Generator der Welt zeigen kann: die Fragestellungen, Ideen, Gedanken und Strategien, die den Beteiligten beim Entwurf des Systems durch den Kopf gingen.

Wie viel Architekturdokumentation sinnvoll ist

Manager von Entwicklungsteams fragen oft: „Wie viel Architekturdokumentation ist eigentlich wirklich nötig?“ Als Erfahrungsgröße aus vielen Projekten kann ich sagen: Wenn Sie keine Dokumentation haben, beträgt der Anteil an Chaos im Team (also Rückfragen, Doppelarbeit, Endlosdiskussionen usw.) bis zu 50 Prozent der Teamkapazität. Investieren Sie bis zu 10 Prozent der Teamkapazität in Dokumentation (**siehe Abbildung 1**), so kann der Chaosanteil auf bis zu 10 Prozent absinken, darunter erfahrungsgemäß nicht. Das Team kann sich dann also zu 80 Prozent auf seine Arbeit konzentrieren. Mehr zu investieren, lohnt nicht, denn das geht auf Kosten des Teamfokus.

Die Startblöcke setzen

Um einen schlanken Dokumentationsprozess im Alltag zu verankern, brauchen Sie einige Dinge nur einmal am Anfang zu tun, andere Dinge hingegen regelmäßig bei jeder Veränderung an der Architektur. Beginnen wir zunächst mit den einmaligen Arbeiten, der Vorbereitung.

1. Einen Kümmerer finden

Ohne einen Anführer geht nichts – es gilt das Gesetz der Wohngemeinschaft (**siehe Kasten 2**). Das gilt auch für Dokumenta-

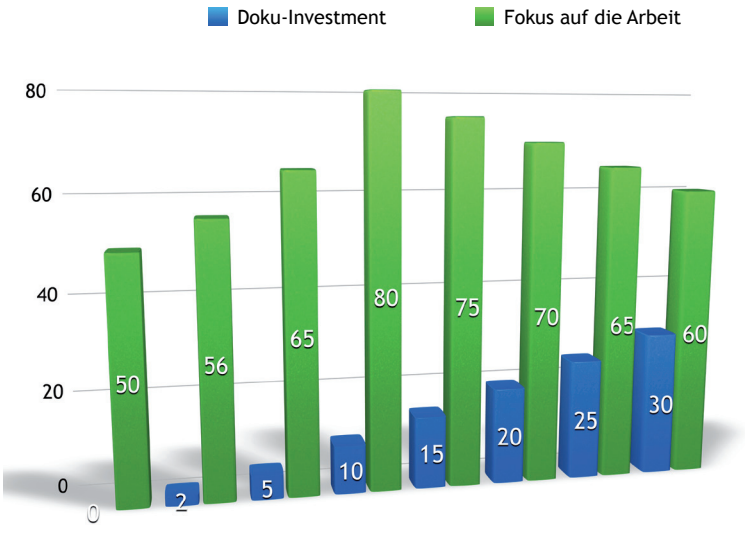

Abb. 1: Teamfokus abhängig vom Dokumentations-Investment.

Wenn sich keiner um den Abwasch kümmert, wird er auch nicht gemacht.

Kasten 2: Das Gesetz der Wohngemeinschaft.

tionsvorhaben. Von selbst geht so etwas nicht. Finden Sie also von Anfang an jemanden, der die Sache vorbereitet, plant, in die Wege leitet, den Leuten in den Hintern tritt und die Sache voranbringt. Machen Sie ihn verantwortlich dafür, dass die Dokumentation entsteht und dass sie gut wird. Wie viel er selbst dazu beiträgt und wie viel er von Kollegen machen lässt, muss er selbst entscheiden. Er muss *Überblick* schaffen, festlegen, *welche Dokumente* zum Release gehören, sich um die *Integration in die Toolkette* kümmern (Build, Versionskontrolle), er muss die *Teams* beim Anfertigen der Dokumentation *unterstützen*, beurteilen, ob die *Dokumentation fertig* ist, und – last but not least – schauen, ob sie *aktuell* und *konsistent* ist.
Geben Sie ihm Zeit und Anerkennung dafür. Das ist keine einfache Aufgabe.

2. Zielgruppen identifizieren
Finden Sie dann heraus, wer Ihre Leser sein werden. Das Schreiben wird wesentlich leichter fallen, wenn Sie wissen, welche Rollen, welche Absichten, welche Ziele und welchen Informationsbedarf Ihre Leser haben. Nehmen Sie die Stakeholder-Liste Ihres Vorhabens als Grundlage. In der Liste gibt es wahrscheinlich Kunden, Projektleiter oder Produktmanager, Entwickler, Architekten, Tester, Betreiber der Infrastruktur und viele mehr. Das ist individuell pro Vorhaben unterschiedlich. Ermitteln Sie das für Ihr eigenes Vorhaben, indem Sie die Leser besuchen und mit ihnen sprechen.

3. Inhalte pro Zielgruppe festlegen
Überlegen Sie genau, welchen Informationsbedarf welche Zielgruppe hat und warum. Wer braucht zum Beispiel den Systemkontext, die Qualitätsziele, technische Risiken, Architekturentscheidungen, Bausteinsicht, Verteilungssicht, Laufzeitsicht, übergreifende Konzepte wie Security oder Logging und Monitoring? Brauchen das alle Zielgruppen? Wer braucht Randbedingungen oder Qualitätsszenarios? Wen wird das interessieren und wen nicht? Wer braucht es nur im Überblick und wer braucht es im Detail?

4. Medien lesbar festlegen
Leser sind unterschiedlich. Manche brauchen ein Dokument aus Papier, das sie im Bus lesen können. Andere sind lieber online und lesen das Ganze als verlinkte HTML-Seiten mit Bildern im Intranet. Andere wollen es auf dem Kindle. Wieder andere schauen sich eine Präsentation an, die auf monatlichen Meetings persönlich gehalten wird und/oder als Video im Intranet veröffentlicht ist. Legen Sie Ihre Inhalte in ein Repository und erzeugen Sie daraus am besten eine Dokumentation pro Zielgruppe, je nachdem, was die Vertreter gern lesen, hören oder sehen.

5. Die Sache im Team angehen
Der Dokumentations-Kümmerer kann und soll nicht alles allein erstellen. Das gesamte Team muss mitmachen, auch die Vertreter der Zielgruppen. So etwas sollte explizit geplant werden, sonst konkurriert es mit den anderen Entwicklungstätigkeiten und die Dokumentation fällt dann „hinten runter". Jeder Beteiligte sollte das zur Architektur beitragen, was er weiß und hat. Die Entwickler kennen die Bausteine, Schnittstellen und wichtigen Abläufe. Die User-Experience-Spezialisten wissen, was gute Interaktion mit dem Benutzer bedeutet, und können einen Styleguide dafür beitragen. Das Business oder der Kunde kennen die wirtschaftlichen Randbedingungen und können diese beisteuern, damit der Entwurf sich daran orientieren kann.
Versuchen Sie nicht, das alles allein zu schaffen. Lassen Sie alle mitmachen und führen Sie die Ergebnisse zu einer konsistenten Dokumentation zusammen.

Schlank dokumentieren im Alltag

Arbeiten Sie iterativ: Zuerst eine Orientierungs-Iteration für die obigen Schritte 2 bis 4, um die Dokumentenarchitektur zu finden, zu testen und zu stabilisieren, dann mehrere Erstellungs-Iterationen, in denen die folgenden Schritte 6 und 7 immer wieder ablaufen.

6. Kontinuierlich publizieren
Sie kennen jetzt Ihre Leser, mit den benötigten Inhalten und den genutzten Medien. Also schreiben Sie, zeichnen Sie, erstellen Sie Präsentationen, drehen Sie Videos usw. Legen Sie alles gleich zentral und geordnet in einem Repository ab – nichts schlimmer, als wenn Aufwand in etwas fließt, das Sie anschließend nicht wiederfinden.
Kompilieren Sie dann aus den Inhalten des Repositorys automatisch die benötigten Dokumentationspakete für jede Zielgruppe. Wenn möglich, fassen Sie diese zu wenigen Paketen zusammen, z. B. können Architekten und Entwickler dasselbe Paket bekommen, nur werden sie unterschiedlich starken Fokus auf die Kapitel legen.
Fügen Sie dieses Kompilieren in das Build-Skript Ihres Vorhabens mit ein. Jedes Mal wenn es die Software baut, sollte es auch gleich die Dokumentation bauen und öffentlich ablegen. Das ermöglicht es Ihnen, Fehler in der Dokumentation schnell zu bemerken, ganz genau so wie beim Continuous Build Ihrer Software.
Wie alles funktioniert, wird gleich im Tools-Abschnitt erklärt.

7. Kommunizieren
Dokumentation muss – genauso wie Software – getestet werden. Veröffentlichen Sie Ihre Dokumentation möglichst regelmäßig und testen Sie sie mit den Lesern aus Ihren Zielgruppen! Fragen Sie, was geholfen hat und was überflüssig war. Was fehlt? Im Text? Im Glossar? Welche Diagramme sagen etwas anderes als der Text oder die Software selbst? Beziehen Sie neue Mitarbeiter explizit ein, denn diese sind unvorbelastet und entdecken Fehler oder Auslassungen in der Dokumentation viel einfacher und schneller als andere. Wenn Sie eine Präsentation über die Architektur halten, ermöglichen Sie eine Feedback-Runde mit Fragen und Antworten. Sie werden staunen.
Verbessern Sie dann Ihre Dokumentation anhand des Feedbacks, das Sie bekommen haben, und stellen Sie sie den Lesern erneut vor. Beobachten Sie die Veränderung, die dann eintritt.

Eine schlanke Toolkette

Stellen Sie sich nun einen Entwickler vor, der dokumentieren will (löblich und außergewöhnlich, oder?). Er hat gerade eine Änderung am Code vorgenommen, die sogar die Architektur verändert hat. Er weiß, er muss das dokumentieren und möchte dafür höchstens diese vier Schritte tun:

- Ein Diagramm ändern.
- Den zugehörigen Text ändern.
- Dokumentation bauen und sehen, ob sie richtig aussieht.
- Einchecken und die Sache vergessen.

Die Schritte, die der Entwickler *definitiv nicht* tun möchte, weil sie repetitiv und langweilig sind und seine Agilität herabsetzen würden, sind:

- Sicherstellen, dass er die aktuelle Version der Dokumentation ändert.
- Seine Änderung mit denen der Kollegen im Änderungsmodus der Textverarbeitung von Hand „mergen".
- Das Tool suchen, mit dem er diese spezielle Art von Diagramm ändern kann.
- Sich überlegen, ob er vielleicht einen Baustein umbenannt hat und statt des einen Diagramms auch noch andere

Diagramme ändern muss, die denselben Baustein zeigen.
- Nachdenken, ob das Diagramm größer geworden ist und das Bild noch auf die Druckseite passt.
- Sicherstellen, dass alle anderen Diagramme, die er nur implizit geändert hat, noch auf die Druckseite passen.
- Das Abbildungsverzeichnis aktualisieren.
- Die geänderte Version der Dokumentation an den Veröffentlichungsplatz kopieren.
- Beim nächsten Release diese Dokumentation als „zu diesem Release gehörig" markieren.

Also muss eine Toolkette für Architekturdokumentation so gestaltet sein, dass sie die Schritte in der ersten Liste möglichst einfach macht und die Dinge in der zweiten Liste automatisiert.
Die Arbeitsreihenfolge (**siehe auch Abbildung 2**) ist:

1. Aktuelle Version der Dokumentation aus einem Repository auschecken.
2. Texte und Bilder erstellen oder ändern.
3. Dokumentation bauen und überprüfen.
4. Ins Repository einchecken.

Automatisierte Tools bauen dann die Dokumentation zusammen und veröffentlichen sie an einem mit den Lesern verabredeten Platz. Die Leser geben Feedback mit Hilfe eines weiteren Tools. Die Autoren nehmen das Feedback und arbeiten es in die Dokumentation ein. Die Dokumentation wird dann automatisch dem nächsten Release zugeordnet und mit der Software mitversioniert.

Welche Tools brauchen wir also, um diese Arbeitsschritte zu ermöglichen?

Erstellungs-Tools
Zunächst benötigen wir Tools zur Erstellung von Texten und Bildern. Die Texte sollten möglichst einfach versionierbar sein. Ein Autor muss sie handhaben können wie Quellcode, nicht wie Dateien in einem Tool-spezifischen Format, bei dem Diff und Merge nur im Tool selbst möglich sind. Klartext-Editoren, die UTF-8-codierte Textdateien erzeugen, haben sich in agilen Projekten als die am einfachsten handhabbare Lösung herausgestellt. Autoren schreiben in einer leicht zu erlernenden Auszeichnungssprache, wie z. B. „Markdown" (vgl. [Gru04]).
Die meisten größeren Content-Management-Systeme, Wikis und Foren können per Plug-in auf Markdown nachgerüstet werden. Dadurch kann die Erfassung von Texten in Wikis beginnen, was für die Storming-Phase zu Anfang Ihres Vorhabens sehr nützlich ist. Später können Sie die Markdown-Texte nahtlos in Textdateien exportieren, die Sie unter Versionskontrolle stellen und von da an nicht mehr im Wiki, sondern direkt in den Textdateien pflegen.
Sehr nützlich ist „MultiMarkdown" (vgl. [Pen06]), eine Obermenge von Markdown. Es fügt einige syntaktische Features hinzu, z. B. Tabellen, Fußnoten und Zitate, sowie weitere Ausgabeformate wie LaTex, das zu sehr schönen PDF-Dokumenten führt (das Original-Markdown erzeugt nur HTML). Auch kann es „smarte" Typographie für verschiedene Sprachen (z. B. korrekte linke und rechte Anführungszeichen).
Die Bilder sollten leicht zu zeichnen und konsistent zu halten sein. Die Notation muss den Lesern bekannt und verständlich sein. Die Konsistenz erreicht man am besten mit Hilfe eines Modellierungswerkzeugs, das nicht etwa voneinander isolierte Diagramme erzeugt, sondern diese mit Hilfe eines dahinter liegenden Modells konsistent hält. Wenn Sie einen Baustein im Modell umbenennen, muss das Tool ihn konsequent in allen Diagrammen umbenennen, in denen der Baustein zu sehen ist. Ein Beispiel für solche Tools sind UML-Modellierungswerkzeuge.
Wichtig für die Anwendung im agilen Kontext ist Folgendes: Wenn ein Autor ein Diagramm ändert, darf er nicht darüber nachdenken müssen, welche anderen Diagramme er auch noch ändern und eventuell aus dem Modellierungswerkzeug neu exportieren muss. Ideal ist es, wenn der Export erst zum Zeitpunkt des *Continuous Build* geschieht (Das bedeutet: Ihr Modellierungswerkzeug muss auch im Batch laufen können, auf der Plattform, auf der das Build-Skript läuft). Doch auch eine vom Autor manuell ausgeführte Funktion „Exportiere alle Bilder in durchnummerierter Form an einem verabredeten Platz" löst dieses Problem zufriedenstellend. Der Autor checkt dann einfach alle Bilder wieder mit ins Repository ein, wo das Build-Skript sie finden kann.

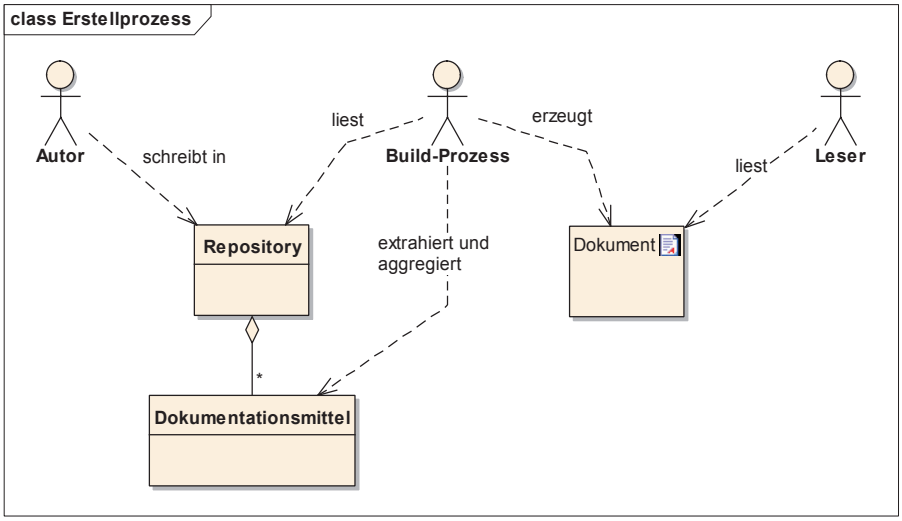

Abb. 2: Erstellungsprozess für Architekturdokumentation.

Das Repository
Meine Erfahrung aus vielen Projekten und deshalb ganz klare Empfehlung: Nicht etwa das Dateisystem, auch nicht die Textverarbeitung, auch nicht das UML-Modellierungswerkzeug, sondern das Versionskontroll-System sollte die Ablage (Repository) bilden. Hier können Sie leicht die aktuelle Version der Dokumentation finden, auschecken, verändern und versioniert einchecken. Aus dem Versionskontroll-System heraus können Sie auch leicht die aktuelle Version der Texte mit früheren oder abgespaltenen Versionen (Branches) vergleichen. Und es ist immer klar, was die aktuelle Version ist und zu welcher Version der lauffähigen Software sie gehört.
Am besten ist es, wenn das UML-Modellierungswerkzeug mit Ihrem Versionskontroll-System zusammenarbeitet. Dann ist es möglich, auch Differenzen auf verschiedene Modellstände zu bilden, die direkt mit den Versionen im Versionskontroll-System korrespondieren (z. B. „Wer hat wann diesen Teil des Modells geändert und was war die eigentliche Änderung?").

Der Dokumentations-Build
Die Aufgabe des Dokumentations-Builds ist es, die im Repository abgelegten Dokumentationsinhalte (Texte, Bilder usw.) zu

Tools für Dokumentation erfüllen vier Zwecke:

- Erstellung
- Ablage
- Zusammenbau
- Kommunikation/Feedback.

Kasten 3: Einsatzzwecke von Tools.

einem Ergebnis zusammenzufassen und für die Leser bereitzustellen, ohne dass ein Autor hinschauen oder dafür sorgen müsste. Dieses Ergebnis kann z. B. ein druckbares Dokument sein oder ein Satz von HTML-Seiten, die im Intranet Ihres Unternehmens publiziert werden.

Der Dokumentations-Build checkt den aktuellen Stand der Dokumentation aus dem Repository aus, lässt Übersetzungs- und Formatierungswerkzeuge wie MultiMarkdown und LaTex laufen und kopiert die Ausgabe an die mit den Lesern verabredeten Stellen. Der Dokumentations-Build ist in der Regel ein Skript, das im Batch abläuft, auf demselben Server und zum selben Zeitpunkt wie der Software-Build.

Falls Sie ein Wiki oder einen Intranet-Auftritt als Zielmedium haben, erstellen Sie am besten pro Zielgruppe Ihrer Leserschaft eine eigene Einstiegsseite mit dafür besonders interessanter Gliederung und Link-Sammlung. Automatisieren Sie das Deployment ins Wiki oder Intranet, damit Sie keinen Aufwand bei den Autoren erzeugen.

Kommunikations- und Feedback-Tools

Einen alten Spruch unter uns Architekten zeigt **Kasten 4**. Da ist etwas dran. Sobald Sie anfangen, Ihre Architektur zu veröffentlichen, werden Sie Meinungen und Feedback dazu bekommen. Das ist auch gut so, denn Architektur wird für Stakeholder gemacht (von denen die Entwickler selbst eine sehr wichtige Gruppe sind). Wenn die Stakeholder Feedback geben wollen, müssen Sie darauf vorbereitet sein, es geordnet entgegenzunehmen und in die Dokumentation einzuarbeiten.

Zuerst ein Beispiel, wie es *nicht* funktioniert:

- Sie schicken das fertige Dokument an die Stakeholder und bitten um Feedback per E-Mail. Sie bekommen dann 100 E-Mails mit jeweils fünf Sätzen, die sich auf unterschiedlichste Stellen in der Dokumentation beziehen. Dieses Feedback müssen Sie dann mühsam klassifizieren, Widersprüche beseitigen und das, was übrig bleibt, einarbeiten. Damit machen Sie sich eine Menge Arbeit und vergessen möglicherweise noch einiges dabei.

„Sie haben eine gute Architektur? Sind Sie sicher? Dann haben Sie sie nur noch niemandem gezeigt!"

Kasten 4: Architekten untereinander.

Jetzt drei Beispiele, wie es funktionieren kann:

- *Word:* Sie lassen den Dokumentations-Build ein Word-Dokument generieren, öffnen es, schalten den Änderungsmodus ein, speichern es und verschicken es per E-Mail an Ihre Leser. Der Leser öffnet eine Kopie, arbeitet sein Feedback direkt dort ein und schickt das Dokument an Sie zurück. Sie öffnen die Kopien, navigieren von Änderung zu Änderung und tragen die Ergebnisse zusammen, die Sie dann in die Textdateien Ihrer Dokumentation einpflegen. Dies kann für Stakeholder, die sich mit Klartextdateien nicht auskennen, sondern nur Textverarbeitung gewohnt sind, die beste Option sein.
- *Kommentare im Blog:* Sie lassen den Dokumentations-Build HTML-Seiten generieren und automatisch in einem Blog-System – zum Beispiel Wordpress – im Intranet veröffentlichen. Sie schicken eine E-Mail und bitten um Kommentare direkt unter jedem Kapitel, das in Wordpress als Seite veröffentlicht ist. Sie lesen die Liste der Kommentare, die Wordpress für Sie gesammelt hat, und pflegen sie als Änderungen in die Textdateien Ihrer Dokumentation ein. Diese Option ist für alle Stakeholder-Gruppen gleichermaßen gut geeignet.
- *Pull-Requests in git:* Sie verschicken eine E-Mail an die Entwickler und bitten um Feedback zu Ihrer Architekturdokumentation. Die Entwickler erzeugen in git einen Fork oder einen Branch, ändern Texte oder Bilder in der Dokumentation und schicken Ihnen einen Pull-Request. Sie sehen die Pull-Requests in git durch und akzeptieren oder lehnen ab. Wenn Sie sie akzeptieren, verändern Sie dadurch gleichzeitig Ihre Architekturdokumentation und sparen sich das manuelle Übertragen. Diese Option erspart Ihnen als Dokumentations-Kümmerer natürlich am meisten Arbeit, doch setzt es voraus, dass Ihre Stakeholder mit git umgehen können und wollen.

Womit Sie anfangen sollten

Bis hierher haben wir Probleme, Vorteile, Prozesse und Tools der agilen Architekturdokumentation kennengelernt. Nun stellt sich die Frage: Womit starten Sie am besten? Ehe Sie jetzt ganz begeistert die Tools holen oder kaufen und dann einfach loslegen, empfehle ich: Beginnen Sie wirklich mit den Schritten „Kümmerer finden", „die Zielgruppen kennenlernen und mit ihnen reden" sowie „Inhalte und Medien festlegen".

Dann schreiben Sie eine erste kleine Version Ihrer Architekturdokumentation und bitten um informelles Feedback in persönlichen Gesprächen. Nichts ist schlimmer, als wenn Sie 100 Seiten schreiben und sich im Nachhinein herausstellt, dass die meisten Leser z. B. kein UML verstehen oder mö-

Literatur & Links

[Bec01] K. Beck et al., Manifesto for Agile Software Development, 2001, siehe: `http://agilemanifesto.org`

[Bro12] S. Brown, Software Architecture for Developers, Leanpub Juni 2012, siehe: `https://leanpub.com/software-architecture-for-developers`

[Gru04] J. Gruber (Daring Fireball), Markdown, 2004, siehe: `http://daringfireball.net/projects/markdown/`

[Hig01] J. Highsmith, History: The Agile Manifesto, 2001, siehe: `http://agilemanifesto.org/history.html`

[ISO11] ISO/IEC/IEEE 42010 Systems and software engineering – Architecture description, siehe: `http://www.iso-architecture.org/42010/`

[Kru95] P. Kruchten, Architectural Blueprints –The ‚4+1' View Model of Software Architecture, in: IEEE Software 12 (6), November 1995, S. 42-50

[Pen06] F. Penney, MultiMarkdown, 2006, siehe: `http://fletcherpenney.net/multimarkdown/`

[Sta] G. Starke, P. Hruschka, Arc42, Ressourcen für Softwarearchitekten, siehe: `http://www.arc42.de`

gen. Dann haben Sie viel Arbeit vergeblich geleistet und dürfen von vorn beginnen.
Danach, wenn Sie sicher sind, dass Ihre Dokumentation in bestimmter Form willkommen sein wird, gehen Sie in eine „Iteration 0". Erstellen Sie eine erste ernst gemeinte Version Ihrer Dokumentation und versuchen Sie, dabei gleich Ihre Toolkette ins Leben zu rufen. Automatisieren Sie, wo es sinnvoll ist, und prüfen Sie, ob die Dokumentation wirklich auf Knopfdruck herauskommt. Beobachten Sie, ob die Teamarbeit klappt. Was passiert, wenn Sie das Ganze mit vielen Leuten machen, die je ein wenig beitragen und sich dabei gegenseitig nicht sehen? Klappt das fehlerfrei oder müssen Sie nachsteuern?
Stellen Sie die Dokumentation aus Iteration 0 erneut den Zielgruppen vor und holen Sie sich erneut Feedback, diesmal formalisiert durch das Feedback-Tool. Prüfen Sie, ob Feedback via Tool wirklich klappt – falls nicht, justieren Sie nach.
Planen Sie dann regelmäßige Dokumentations-Iterationen, immer mit Feedback am Ende. Das Ganze wird sich immer mehr einschleifen und es wird sowohl für die Autoren als auch für die Leser immer einfacher werden. Ab einem gewissen Zeitpunkt werden Sie sich sicher fühlen. Machen Sie Dokumentation zum Bestandteil des Entwicklungsalltags.
Dokumentieren Sie Ihre Erfahrungen im Unternehmens-Wiki. Stellen Sie Standards auf, die Ihre Autoren und Leser unterstützen. Schaffen Sie eine Dokumentenarchitektur, in der zu 100 Prozent klar ist, was wohin kommt: sowohl die Struktur des Repositorys als auch die innere Struktur jedes Kapitels in den Dokumenten.

Zusammenfassung und Ausblick

Der Artikel ging von einem wohlbekannten Problem aus: Wie können wir Architektur so sinnvoll und angemessen dokumentieren, dass wir agil bleiben und uns auf die Entwicklung lauffähiger Software konzentrieren können? In einem Überblick sahen Sie eine bewährte Vorgehensweise und bewährte Tools, mit denen Sie das Ziel Agilität und gute schriftliche Kommunikation erreichen können. Wenn Sie das Thema so konsequent angehen, haben Sie mehrere Vorteile:

- Sie werden den Aufwand für Ihre Dokumentation erheblich senken können.
- Das Dokumentieren wird sich leicht in den Entwicklungsalltag integrieren lassen.
- Die Leser werden die entstehende Dokumentation gern lesen und gut verstehen.

Viel Erfolg dabei! Zu den hier angesprochenen Themen habe ich weiteres Material zusammengestellt, das Sie unter der URL finden können, die auf der ersten Seite des Artikels genannt ist. Meine Homepage finden Sie unter `http://mbohlen.de`.

Der Autor

Matthias Bohlen
(mbohlen@mbohlen.de)
ist Experte für effektive Produktentwicklung. Teams engagieren ihn in vielen Projekten, damit er ihnen dabei hilft, sich zu organisieren und Architekturprozesse für sich selbst zu finden und aufzustellen. Methoden hierfür unterrichtet er in der Architektenausbildung nach iSAQB.