Der Software-Architekt und sein Unwissen

Wolfgang Golubski[1], Gerrit Beine[2]

[1]Westsächsische Hochschule Zwickau,
FG Informatik, 08056 Zwickau, Germany
golubski@fh-zwickau.de

[2]adesso AG, Rotherstraße 19, 10245 Berlin, Germany
gerrit.beine@adesso.de

Abstract: Software-Systeme sind stetigen Änderungen ausgesetzt. Unvorhersehbare Ereignisse beeinflussen die Entwicklungsarbeit erheblich. In diesem Paper wird Talebs Konzept der Antifragilität auf den Entstehungsprozess von Software-Architekturen übertragen. Antifragilität hilft dabei, mit Unwissen im Entwicklungsprozess proaktiv und konstruktiv umgehen zu können.

1 Einleitung

Software-Entwicklung ist immer noch eine große Herausforderung für alle Beteiligten. Immer mehr Software für die verschiedensten Aufgabenstellungen und Probleme mit ständig wechselnden Anforderungen und Möglichkeiten entsteht. Entwicklungen profitieren zum Einen von Wissen über Lösungsmöglichkeiten und zum Anderen von neuartigen Ideen. Bei letzteren sind häufig die Lösungswege unbekannt, unerforscht oder es existiert schlichtweg kein Wissen darüber. Um mit dem Wandel und dem Unbekannten besser umgehen zu können, kann das Wissen über das Nichtwissen helfen.

Taleb hat in in seinem Buch „Antifragilität: Anleitung für eine Welt, die wir nicht verstehen“ [Ta13] diese Thematik in allgemeiner Form ausführlich erörtert. In dem vorliegenden Paper wird seine These der Antifragilität auf die Entwicklung von Software-Architekturen übertragen, mit dem Ziel im Entstehungsprozess Denkanstöße zu geben und neue Sichtweisen zu initiieren. Als Nebeneffekt ordnen wir Fragilität, Resilienz und Antifragilität in der Software-Architektur-Entwicklung genau ein und räumen mit Missverständnissen, die in vielen englischsprachigen Foren zu finden sind, auf.

Das Paper ist wie folgt gegliedert. Im nachfolgenden beiden Abschnitte werden die Ausgangslage und die Definition der Begriffe Fragilität, Robustheit, Resilienz sowie Antifragilität vorgestellt. Dann folgt eine systematische Beschreibung von Wissen und dem Grad von Unwissen. Im Abschnitt wird der Bezug zur Software-Architektur hergestellt. Erste Schritte und ein Fazit beenden das Paper.

2 Ausgangslage

Software-Projekte werden oftmals mit unvorhergesehenen Anforderungen konfrontiert, die zu drastischen Änderungen der inneren Struktur der Software führen. Der allgemein angewendete Weg, solche Änderungen zu vermeiden, ist das Schaffen einer möglichst umfassenden, generischen und auf alle Eventualitäten vorbereiteten Software-Architektur. Allerdings erhöht sich durch diese Strategie unweigerlich die Komplexität der Software, was Änderungen bei tatsächlich nicht vorhergesehenen Ereignisse aufwändiger und komplizierter macht.

Als Beispiel aus der täglichen Projektpraxis mag das Thema der Performance-Optimierung dienen. Der übliche Weg besteht darin, die Software-Architektur ganzheitlich so zu optimieren, dass sie in Gänze den Performance-Anforderungen genügt. Bei der Arbeit mit agilen Teams hat es sich aber gezeigt, dass es sich positiv auf die Software-Architektur auswirkt, genau das nicht zu tun, sondern möglichst einfache Lösungen zu erstellen und Optimierungen nur punktuell an den Stellen durchzuführen, an denen Auftraggeber nach mehr Performance verlangen. Durch den Verzicht auf vorausschauende Optimierung wurde die Komplexität der Software reduziert, was sich bei zukünftigen Anforderungen, die als unvorhersehbare Ereignisse angesehen werden können, positiv auswirkt.

In den bisherigen Arbeiten oder Analysen zu Antifragilität in der Software-Entwicklung und Architektur, wie z. B. [Mi14,Mo14] werden nur Programmier-Methoden oder Werkzeuge als mögliche Heilsbringer angeboten. Allerdings können diese, wie in Abschnitt 4 begründet, hier nicht wirklich weiterhelfen. Antifragilität ist definitiv keine Eigenschaft einer Software-Architektur.

3 Was ist Antifragilität?

Das Wort Antifragilität ist eine Schöpfung von Nassim Nicholas Taleb. Er beschreibt Antifragilität als eine Eigenschaft von Systemen, die deren Umgang mit unvorhersehbaren Ereignissen bestimmt. Solche unvorhersehbaren Ereignisse, die Taleb als Schwarze Schwäne bezeichnet [Ta13], treten in verschiedensten Ausprägungen immer wieder auf und können Systeme nachhaltig stören. Viele Systeme sind unvorhersehbaren Ereignissen gegenüber fragil. Sie können mit diesen Ereignissen nicht umgehen und werden unter Umständen so massiv gestört, dass sie nicht weiterexistieren können. Dem Fragilen wird laut Taleb üblicherweise das Robuste gegenüber gestellt. Robuste Systeme zeichnen sich dadurch aus, dass sie durch einige unvorhersehbare Ereignisse nicht gestört werden. Allerdings kann ein System keine Robustheit gegenüber allen unvorhersehbaren Ereignissen aufweisen. Als weiteres Gegenteil von Fragilität beschreibt Taleb Resilienz. Ein resilientes System wird durch ein unvorhersehbares Ereignis gestört, kehrt aber nach dem Ereignis wieder in seinen ursprünglichen Zustand zurück.

Beide Eigenschaften sind aber nicht ausreichend, um das wirkliche Gegenteil von Fragilität zu beschreiben. Robuste als auch resiliente Systeme verbessern sich nicht durch unvorhersehbare Ereignisse, sondern bleiben im besten Fall unverändert. Diese Eigenschaft, anhand von Schwarzen Schwänen zu wachsen, bezeichnet Taleb als Antifragilität. Sie erlaubt es, mit Unbekanntem umzugehen, ohne dass ein völliges Verständnis dieses Unbekannten notwendig ist. Antifragilität ist eine Fähigkeit durch Schocks besser zu werden und können nur lebendig-organische Systeme aufweisen, Unbelebtes kann nicht antifragil sein. Antifragilität entsteht auf zwei Arten: Zufall oder Lernen. Durch Zufall gesteuerte Antifragilität lässt sich in der Natur wiederfinden, die Taleb auch als Beispiel verwendet. Antifragilität durch Lernen lässt sich unter Anderem anhand der Entwicklung menschlicher Kulturen oder der Art, wie Innovationen entstehen, erklären. Beiden Wegen gemein ist, dass sie von Irrtümern profitieren, also auch von falschen Reaktionen auf unvorhersehbare Ereignisse.

4 Die Five Orders of Ignorance und Antifragilität

Antifragilität profitiert vom Unbekannten und von unvorhersehbaren Ereignissen. Um mit unbekanntem systematisch umgehen zu können, muss aber zunächst definiert werden, was Unbekanntes eigentlich ist. Dazu können die Five Orders of Ignorance dienen, die von Phil Armour im Jahr 2000 zum ersten Mal beschrieben wurden [Ar00], um die Unsicherheit durch Unwissen, die mit der Entwicklung von Software einhergeht, strukturieren zu können. Es handelt sich dabei um folgende Kategorien:

- 0th Order of Ignorance (0OI) – Lack of Ignorance: Sicheres Wissen, Bekanntes.
- 1st Order of Ignorance (1OI) – Lack of Knowlegde: Unwissen auf einem bekannten Gebiet, Kenntnis der richtigen Fragen.
- 2nd Order of Ignorance (2OI) – Lack of Awareness: Unwissen und Unkenntnis über das Gebiet des Unwissens, Unkenntnis der richtigen Fragen.
- 3rd Oder of Ignorance (3OI) – Lack of Process: Unwissen, Unkenntnis über das Unwissen und darüber, wie herausgefunden werden kann, auf welchen Gebieten, das Unwissen exisitert. Unkenntnis der richtigen Fragen und des Weges diese Fragen zu herauszufinden.
- 4th Order of Ignorance (4OI) – Meta-Ignorance: Unkenntnis der Five Orders of Ignorance.

Armour argumentiert, dass die meisten Software-Projekte sich auf der 2OI befinden. Seiner Ansicht nach setzen die meisten Methoden zur Software-Entwicklung auf der 3OI an, wobei sie immer nur die Bereiche zeigen können, in denen Unwissen existiert. Sie liefern weder die Fragen, die von der der 2OI zur 1OI führen würden, noch die Antworten auf diese, die dann von der 1OI zu 0OI führen.

Unvorhersehbare Ereignisse sind als nicht prognostizierbar definiert. Sie sind somit auf der 3OI anzusiedeln: Es ist unbekannt, was passiert, wann es passiert und es gibt keine Möglichkeit beides bis zum Eintritt des Ereignisses herauszufinden. Praktisch alle zukünftigen Anforderungen an eine Software-Architektur können als unvorhersehbare Ereignisse betrachtet werden und sind somit als auf der 3OI anzusiedeln.

Antifragilität erlaubt es, mit diesem Unwissen umzugehen und trotzdem zielgerichtet Software-Architekturen zu entwerfen. Sofern bei der Transformation von Unwissen der 3OI zur 2OI und weiter Irrtümer zugelassen werden, besteht durch Antifragilität sogar die Möglichkeit, eine bessere Software-Architektur zu entwerfen. Da Antifragilität von Asymmetrie profitiert, also Versuchen, bei denen kleine, wahrscheinliche Verluste großen, unwahrscheinlichen Gewinnen gegenüberstehen, ist es unabdingbar, eine Menge solcher Versuche zum Entwurf einer Software-Architektur zu unternehmen, statt wo immer möglich, auf Bewährtes zu setzen.

Dieser Umgang mit unvorhersehbaren Ereignissen und der 3OI führt zwar nicht zu einer vorhersehbaren Software-Architektur, aber mit größerer Wahrscheinlichkeit zu einer Software-Architektur, die der zukünftigen Anforderung, also dem unvorhergesehenen Ereignis, angemessen ist.

Aus dieser Angemessenheit ergeben sich direkte ökonomische Vorteile. Diese resultieren aus der Untersuchung der Fragilität der entstandenen Software-Architektur, die ihrerseits messbar ist. Eine auf diesem Weg entworfene Software-Architektur wird notwendigerweise eine höhere Volatilität besitzen, als auf klassischem Weg entworfene Software-Architekturen. Das Prinzip der Antifragilität erlaubt es aber, die Folgen der Volatilität beherrschbar zu machen, was leichter ist, als die Folgen der Fragilität der Software-Architektur zu beherrschen.

5 Antifragilität und Software-Architekt

Die Arbeit des Software-Architekten wird (bisher) durch die folgende Fähigkeiten [SH09] gekennzeichnet: Entwerfen, Entscheiden, Vereinfachen, Implementieren, Dokumentieren, Kommunizieren, Schätzen, Balancieren, Beraten, Vermarkten können. Welche Fähigkeit fehlt aber noch? Der Umgang mit Schwarzen Schwänen – der 3 Order of Ignorance! Dies ist die Fähigkeit mit zukünftigen Anforderungen umgehen zu können, d. h. sich über den jederzeit möglichen Wandel im Klaren zu sein. Wenn die Anforderung sich ändert, kann die Architektur sich ändern. Ein Festhalten, wie es in der Vergangenheit gerne propagiert wurde, ist eher ein Hindernis, robuste Software-Architektur führt im besten Fall nur zur zweitbesten Lösung.

In der sogenannten Chaosfamilie hat Taleb mögliche Aspekte (Komplexitätstreiber, Kräfte) benannt, die im Zusammenhang mit Volatilität auf die Fragilität eines Systems wirken können. In Abbildung 1 ist dies auf die Entwicklung von Software-Architekturen übertragen worden, Dabei zeigen sich die Wirkkräfte der Antifragilität im Zusammenspiel von ständigem Wechsel, Zeit, Ablauf, Wissen und Kommunikation.

Der Software-Architekt ist ständigen Wechseln ausgesetzt, wie neu entdeckte oder geänderte Anforderungen oder Stakeholder, kundenspezifische Anpassungen der Lösung, neue Versionen eines Frameworks mit geänderten Features. Die zur Verfügung stehende Zeit ist natürlicherweise beschränkt, kann durch eine veränderte Marktsituation plötzlich reduziert werden. Die Geschwindigkeit des Time-to-Market nimmt zu. Auch sein Wissen über fachliche Inhalte, Technologien, Methoden, o.ä. ist nicht vollkommen. Der Ablauf oder die verwendete Entwicklungsmethode kann sehr agil sein. Das zwischenmenschliche Kommunikation zum Erfolg eines Projektes wesentlich beiträgt ist, ist mittlerweile unumstritten. Alle die genannten Wirkkräfte können positiv genutzt werden, wie Taleb es anregt, oder können eine Software sehr fragil machen.

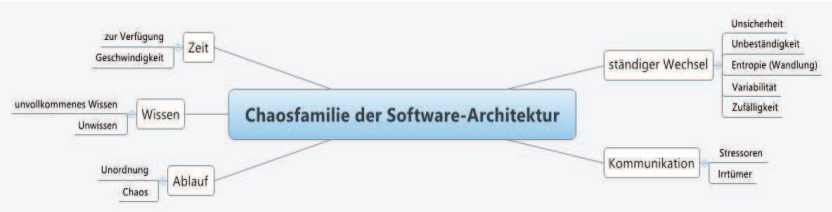

Abbildung 1: Chaosfamilie der Software-Architektur

Da Antifragilität eine Eigenschaft ist, lässt sie sich am ehesten einem Prinzip zuordnen. Der Begriff Tugend wird von Taleb in diesem Zusammenhang auch verwendet. Tugend bezeichnet eine hervorragende Eigenschaft (Qualität) oder vorbildliche Haltung. Somit könnte man Antifragilität auch als Tugend bezeichnen.

Übertragen auf die Software-Architektur kann Antifragilität als Qualitätseigenschaft des der Entstehung der Software-Architektur zugrundeliegenden Prinzips angesehen werden. Der Umgang mit dem Unbekannten ist die große Herausforderung.

6 Erste Schritte zur Unterstützung von Antifragilität

Konkrete Schritte auf dem Weg zur Antifragilität sollen nun benannt werden. Dabei werden aus drei Perspektiven nennenswerte und zu berücksichtigende Punkte aufgezählt.

Zur Normalität der Projektentwicklungsarbeit gehören die Akzeptanz von Wissen und Unwissen sowie der Umgang mit Änderungen und Veränderungen. Entscheidungen werden im Laufe des Projektes bewusst getroffen und bewusst NICHT getroffen. Letzteres kann wichtig sein, um weitere Optionen noch zu ermöglichen. Um die Grundlagen für Entscheidungen zu verbessern sind ständiges Lernen und Weiterbilden notwendig. Fehler können Kosten sparen, wenn es die richtigen sind, siehe auch Beispiel im Kapitel 2.

Software-Architekten arbeiten mit Teams zusammen. Diese Arbeit wird sich weiter verändern. Entscheidungen müssen gut vorbereitet und gut argumentiert werden.

Unterschiedliche Lösungsstrategien werden unterstützt. Die Rolle des Software-Architekten wird zu einem Manager des Veränderungsprozesses und von Entscheidungen und Alternativlösungen werden.

Aus technologischer Sicht sollten weniger komplexe Lösungen bevorzugt werden. Ein, bewusster und gemäßigter Einsatz von Generizität und Abstraktheit ist sehr zu empfehlen. Alle Eventualitäten oder zukünftig kommende Anforderungen vermeintlich abdeckende Lösungen sind zu vermeiden. Ein schwarzer Schwan ist nicht vorhersehbar. Mehr Praktikabilität im Sinne von „first make it run, then make it fast“ muss höchste Priorität haben.

Um den Umgang mit Antifragilität zu erlernen oder zu verbessern, können Ansätze aus dem Management 3.0 von Appelo [Ap11] helfen. Appelo stellt einen recht umfassenden Ansatz dar, von dem die agile Entwicklungsarbeit profitieren kann. Die Hauptaktivitäten „Plan, Do, Check, Act“ erläutert er ausführlich, wobei der Mensch, Wissen und der Wandel im Vordergrund stehen. Die Entwicklung von Kompetenz oder die Notwendigkeit der kontinuierlichen Veränderung (Change Management) genauso wie der (positive) Umgang mit Fehlern oder Fehlentwicklungen („try, again and again, until you have it right and learn from other people’s failures“) lassen sich aus der Management-Sicht auf die Arbeiten eines Software-Architekten übertragen.

7 Fazit

„Technology is inherently fragile“ - viel besser als mit diesem Zitat von Taleb kann die Herausforderung der Zukunft, der sich die Software-Architektur stellen muss, kaum beschreiben. Antifragilität kann eine Antwort auf diese Herausforderungen sein, jedoch nicht als Eigenschaft der Software-Architektur. Dazu müssen Software-Architekten lernen, systematisch mit dem Unwissen umzugehen, das der Entwicklung von Software inhärent ist und Methoden entwickeln, zielgerichtet Irrtümer mit kleinen Kosten und großen Chancen einzugehen.

Literaturverzeichnis

[Ap11] Appelo, J.: Management 3.0: Leading Agile Developers, Developing Agile Leaders (Addison-Wesley Signature Series (Cohn)). Addison-Wesley Professional (2011).

[Ar00] Armour, P.G.: The five orders of ignorance. Commun. ACM. 43, 2000, S. 17–20.

[Mi14] Miles, R.: Antifragile Software - Building Adaptable Software with Microservices. leanpub (2014).

[Mo14] Monperrus, M.: Principles of Antifragile Software, Draft (Working) Paper http://arxiv.org/pdf/1404.3056v1.pdf.

[SH09] Starke, G., Hruschka, P.: Software-Architektur kompakt angemessen und zielorientiert. Spektrum, Akad. Verl., Heidelberg, 2009.

[Ta13] Taleb, N.N.: Antifragilität: Anleitung für eine Welt, die wir nicht verstehen. btb-Verlag, 2013.