AGILE SOFTWAREARCHITEKTUR

VON DER IDEE ZUM SYSTEM – ABER NIE GANZ WIE GEPLANT

AUTOR **STEFAN TOTH**

Urlaub ist toll. Wochenlang habe ich mich darauf gefreut und Tage mit der Routenplanung und Hotelbuchungen verbracht. Nun stehe ich mit meiner Familie vor der Haustür, das Taxi kommt nicht und mein 5-jähriger Sohn jammert über die Hitze und zu warten findet er langweilig.

Als das Taxi endlich auftaucht, fehlt der bestellte Kindersitz aber irgendwie schaffen wir es zum Bahnhof. Viel zu spät kommen wir am Urlaubsort an und erfahren erst vor Ort, dass die Unterkunft wegen kurzfristiger Umbauten nicht wie geplant bewohnbar sei. Wir suchen uns ad hoc ein Hotel und sind uns nicht mehr ganz so sicher, wie toll Urlaub eigentlich noch ist …

Natürlich ist Urlaub toll. Aber man muss schon manchmal zugeben: So reibungslos und erholsam, wie gedacht, ist es dann doch nicht immer. Mit viel Reiseerfahrung entwickelt man eine gewisse Gelassenheit und weiß mit Überraschungen umzugehen. Es lässt sich eben nicht alles kontrollieren und Komödie ist schließlich nur Drama plus Zeit.

Auch Softwarearchitektur ist toll. Ich behaupte, dass Urlaube und Architekturen einige Parallelen aufweisen. Auch hier halten Pläne meist nur bis zum ersten Kontakt mit der Umsetzung. Was wir am Ende bauen, ist nie das, was wir anfangs auf ein Whiteboard gemalt haben. Das ist kein Versagen, sondern Normalität.

Moderne Softwareentwicklung ist schnell, verteilt und geprägt von ständigem Wandel. Neue Frameworks, Tools, Plattformen und Paradigmen drängen nach Aufmerksamkeit, Entscheidungen entfalten unerwartete Effekte und Technologien offenbaren ungeahnte Probleme. Hätten wir das nicht gleich richtig machen können? Hätte man das nicht vorher herausfinden können? Doch mit viel Architekturerfahrung entwickelt man eine gewisse Gelassenheit und weiß mit Überraschungen umzugehen. Es ist eben nicht alles kontrollierbar und Komödie ist schließlich nur Drama plus Zeit.

ARCHITEKTUR UND AGILITÄT ALS IDEALER PARTNER

Moderne Definitionen von Softwarearchitektur haben eins gemeinsam: Sie behaupten, die Disziplin kümmere sich um zentrale und später nur schwer änderbare Kernelemente eines Systems. Martin Fowler meint: „To me the term architecture conveys a notion of the core elements of the system, the pieces that are difficult to change. A foundation on which the rest must be built.“ In den 1980ern waren diese grundlegenden, schwer änderbaren Entscheidungen hauptsächlich struktureller Natur. Wie zerfällt unser System sinnvoll in Komponenten? Welche Schnittstellen sollte es geben? Wie legen wir Daten ab?

Programmiersprachen, Vernetzung und Virtualisierung von Systemen nahmen in den 1990ern zu. IBM-Mainframes und Unix-Workstations waren zuvor fast alternativlos. Neue Servertechnologien tauchten auf. Und allein 1995 erblickten Windows NT, der Apache HTTP Server und Programmiersprachen, wie Java, PHP, JavaScript, Ruby und Delphi das Licht der Welt. Sie brachten auch neue Muster und Implementierungsstrategien mit sich. Mit Etablierung des Internets wurden parallel die Nutzergruppen von Software größer und verteilter. Software musste nicht mehr nur wartbar, sondern auch sicher, zuverlässig, benutzbar oder skalierbar sein – unter komplexeren technischen Rahmenbedingungen.

Die heutige Menge an Programmierframeworks, -umgebungen und -werkzeugen ist enorm. Im KI-Bereich kommen täglich neue Frameworks hinzu. Alle relevanten Technologien zu überblicken ist schwierig – geschweige denn, tiefe Expertise in allen Bereichen zu besitzen.

Es entstehen verwobene Designaufgaben und Personen mit unterschiedlichen Expertisen müssen dynamisch zusammenarbeiten, um gute Software zu bauen. Spätestens seit dem agilen Manifest aus dem Jahr 2000 begleitet

Abbildung 1 – Das Manifest agiler Architekturarbeit [1]

Agilität die notwendige Vernetzung von Rollen und die Feedback-orientierte Arbeit an Software. Sie wird damit zum idealen Partner einer zeitgemäßen Architekturdisziplin.

Abbildung 1 zeigt das „Manifest" für moderne Architekturarbeit mit den vier zentralen Wertepaaren: kontinuierliche Verbesserung, zielorientiertes Handeln, Kollaboration und breit gestreute Verantwortung. Die Basis dafür bilden u. a. Arbeiten von David J. Snowden [2]. In seinen Beschreibungen zum Umgang mit komplexen Problemstellungen empfiehlt er eine Arbeitsweise, die auf Experimente optimiert ist: Ausprobieren – Beobachten – Reagieren. Diese breit gedachten theoretischen Grundlagen lassen sich gut auf Softwarearchitektur übertragen. Das Manifest zeigt einige Grundideen auf: die engmaschige Vernetzung von Konzeptions- und Umsetzungsarbeit.

DIE ZWEI MODI AGILER ARCHITEKTURARBEIT

Wer Softwarearchitektur in einem agilen Umfeld betreibt, bewegt sich idealerweise zwischen zwei Arbeitsmodi.

Modus 1: Struktur durch Guardrails

Teil von Architekturarbeit ist es, Leitplanken (Guardrails) zu etablieren, um die Softwareentwicklung über die Zeit hinweg in eine gewünschte Richtung lenken. Das können grundlegende Entscheidungen zu Programmiersprachen, Infrastruktur, Datenhaltung oder Sicherheitsmodellen sein:

- Technische Prinzipien: „Alle internen Aufrufe müssen verschlüsselt sein."
- Nicht-funktionale Anforderungen: „Latenzzeiten müssen immer unter 200 ms liegen."
- Referenzmuster: „Diese Docker-Base-Images sind immer zu verwenden."
- Automatisierte Prüfungen: „In der CI/CD-Pipeline werden Abhängigkeiten zu Fremdbibliotheken geprüft."

Guardrails verhindern Wildwuchs und sollen Teams Orientierung geben, sie in der Umsetzung entlasten und Raum für fokussierte Kreativität schaffen. Würde man NUR in diesem Modus verharren, käme Architektur sehr klassisch daher und würde bald auch als abgehoben und bremsend wahrgenommen werden. In agilen Kontexten wollen wir Guardrails deshalb stetig hinterfragen und weiterentwickeln. Dafür sollten sie schlank ausgeprägt sein und nicht zu viele Ressourcen binden.

Modus 2: Architektur durch Emergenz

Oft wissen Teams erst in der Umsetzung oder nach Auslieferung, ob ein Service wirklich skaliert, eine neue Idee praktikabel ist oder eine Änderung auf Dauer trägt. Statt eines ausgefeilten Plans brauchen wir Erfahrungsaufbau. In diesem zweiten Modus wird Architektur zur Hypothese, die getestet wird:

- durch kleine Experimente (Spikes),
- durch Auswerten von Telemetrie und Logs,
- durch Feature Flags oder Shadow Deployments,
- durch bewusste Kommunikation der Erkenntnisse (z. B. in Communities oder ADRs).

Diese Form der Architekturarbeit lebt von Beobachten, Ausprobieren und Lernen. Die besten Ideen entstehen dabei nicht im Meetingraum – sondern beim Debuggen, Reflektieren über Systemverhalten, beim Reiben an einem schwierigen Problem. Die Arbeit am echten Problem erzeugt über die Zeit neue Guardrails, weicht ggf. bestehende auf oder erzeugt eine „gelebte Architekturpraxis", die Teams weicher leitet als es Leitplanken täten.

Zur Ermöglichung dieser Bottom-Up Architektur bedarf es ausreichend Zielverständnis und Kontext bei den Umsetzenden, genügend Raum für Austausch zwischen Teams und rigoros gelebte Feedbackmechanismen auf technischer Ebene. Das ist jener Teil von Architekturarbeit, der seit dem Jahr 2000 stetig wichtiger wird. Die meisten Praktiken, die wir „agiler Architekturarbeit" zuordnen, versuchen diesen Modus effektiv und praktikabel zu machen.

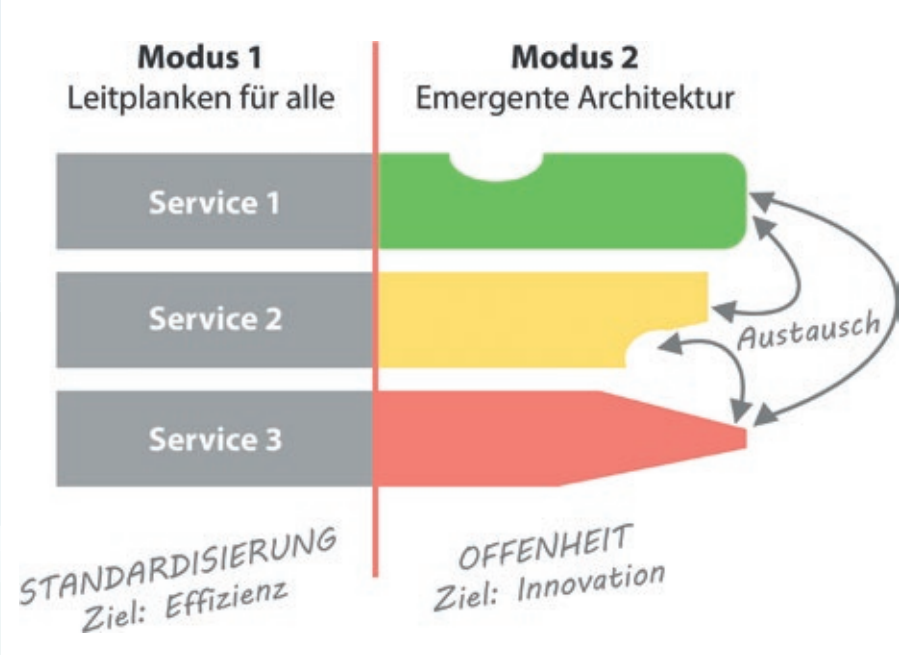

Abbildung 2 – die zwei Modi moderner Architekturarbeit

KONKRETE PRAKTIKEN ZUR VERZAHNUNG

Die erfolgreiche Etablierung emergenter Architekturpraxis als Ergänzung zu klassischen Leitplanken bedarf einiger Praktiken. Zum Abschluss seien einige genannt, ohne sie erschöpfend zu betrachten, wobei Interessierten besonders [1] ans Herz gelegt sei:

- **Kontinuierliche ArchitekturHypothesen:** Architektur wird als laufende, messbare Serie von Experimenten statt als einmaliges Design verstanden. Explorative Spikes mit Rückkopplung testen neue Ideen realistisch aus.
- **Qualitätsziele als Treiber:** Qualitätsziele sind zentral für Architekturarbeit und deshalb auch allen Mitwirkenden bekannt. Sie sind auch in Backlogs sichtbar.
- **FeedbackFirst:** Health & Statistik-Checks sind zentral, Architekturregeln werden kontinuierlich technisch geprüft.
- **Weiche Standards, harte Grenzen:** Nicht alles wird vorgeschrieben. Es gibt verbindliche Prinzipien und Best Practices neben explizitem Freiraum für Innovation.
- **Leichtgewichtige Reviews und Entscheidungsformate:** Niedrigschwellige Austauschformate wie Peer Reviews, Architektur-Office-Hours oder Entscheidungs-Workshops fördern Reflexion ohne Bürokratie.
- **OrgaArchKopplung:** Gute Architektur braucht auch auf organisatorischer Seite Voraussetzungen. So muss z.B. der Teamschnitt den fachlichen Grenzen gehorchen oder die Verantwortung der Teams zur erwarteten Innovationsbeteiligung passen. ■

QUELLEN:

[1] Toth, Stefan: „Vorgehensmuster für Softwarearchitektur. Kombinierbare

Praktiken in Zeiten von Agile und Lean“; Carl Hanser Verlag, 2025

[2] Snowden, David J.; Boone, Mary E.: „A Leader’s Framework for Decision Making“; in: Harvard Business Review, 2007

DER AUTOR

STEFAN TOTH
ist Gründer der embarc GmbH und Autor zahlreicher Artikel sowie der Bücher „Vorgehensmuster für Softwarearchitektur“ und „Software-Systeme reviewen“. Als Berater begleitet er Start-ups, Mittelständler und Großkonzerne bei der organisatorischen, methodischen und technischen Neuausrichtung.