Jens Coldewey
(jens_coldewey@acm.org)
praktiziert agile Verfahren seit 1998. Das Konzept der 3m²-Architektur hat er 1998 für ein System entwickelt, das noch heute in verschiedenen Installationen im Einsatz ist und weiterentwickelt wird. Heute begleitet er Organisationen auf ihrer Reise in die Agilität.

Architektur auf 3m²

Wie viel Architektur braucht ein agiles Team, bevor es starten kann? Die meisten agilen Methoden verraten erstaunlich wenig über diese doch recht wichtige Frage. Scrum sieht Architektur als Verantwortung des Entwicklungsteams – und hält sich als Management Framework aus Architekturfragen heraus. Durch die Aufnahme der „Agilen Entwicklungspraktiken" in den Scrum-Schulungskanon in den letzten Jahren hat sich das ein wenig geändert, aber diese sind weitgehend deckungsgleich mit den technischen Praktiken des Extreme Programming (XP), auf das ich unter anderem im nachfolgenden Artikel eingehen werde.

Als Methode zum evolutionären Change Management beschäftigt sich auch Kanban korrekterweise nicht mit Architekturfragen, auch wenn David Anderson persönlich Anhänger des Feature Driven Development ist.

Alistair Cockburns Crystal Clear schlägt einen Lead Designer vor, der die Beschreibung der Systemarchitektur verantwortet, „normalerweise ziemlich früh in der ersten Iteration". Crystal Clear betont aber auch, dass sich die Architektur „vermutlich weiterentwickelt" und bietet dafür zwei Strategien an: Das „Walking Skeleton" und „Incremental Rearchitecture" [Coc05].

Extreme Programming (XP) schlägt schließlich die „Metapher" vor, um die technische Vision des Systems aufrechtzuhalten – meines Erachtens leider eine der am wenigsten verstandenen und am meisten ignorierten XP-Praktiken.

XP stützt sich zudem stark auf das Refaktorisieren, um die Architektur weiterzuentwickeln, und auf Unit-Tests, um sie wie mit einem Gerüst abzusichern. In der zweiten Auflage seines Buchs führte Kent Beck schließlich die Rolle eines Architekten in einem XP-Team ein, nicht ohne zu betonen, dass „Architekten Programmieraufgaben übernehmen, wie jeder andere Entwickler" [Bec04].

Es verwundert daher nicht, dass zu den beliebtesten Themen in Open Spaces zu Agilität die Frage gehört: „Wie viel Architektur brauchen wir, bevor wir starten können?" Meine Standardantwort auf diese Frage ist: „Drei Quadratmeter". Die Antwort ist zugegebenermaßen nicht so ganz selbsterklärend, daher möchte ich sie hier erläutern.

Zunächst muss man sich vergegenwärtigen, dass jedes Softwaresystem eine Architektur hat. Die Architektur kann angemessen sein oder nicht, sie kann die nicht-funktionalen Anforderungen erfüllen oder eben nicht, sie kann hilfreich sein oder die Entwicklung an allen Ecken und Enden behindern, sie kann klar und naheliegend sein oder extrem verworren, aber sie ist immer vorhanden.

Man kann kein Softwaresystem bauen, ohne dass dabei eine Architektur entstehen würde. Aber natürlich gibt es tausende von Möglichkeiten, sehr schlechte Architekturen zu erstellen. Oder wie Brian Foote in seinem genialen Vortrag „Big Ball of Mud" formuliert: „Der ‚Große Haufen Mist' ist das vorherrschende Architekturmuster heutiger Softwaresysteme"[Foo].

Der zweite wichtige Aspekt ist, dass es nur einen einzigen Ort gibt, an dem die wahre Architektur des Systems niedergeschrieben ist: im Code. Sie können Gigabytes von Powerpoint-Präsentationen über „Unsere Architektur" haben oder keine einzige: In jedem Fall bestimmt ausschließlich die real existierende Architektur im Code das nicht-funktionale Verhalten des Systems, wie Wartbarkeit, Verständlichkeit, Performance, Skalierbarkeit und so weiter.

Von daher ähneln Architekten, die nicht über beide Ohren „im Code stecken" Kindergärtnerinnen, die sich weigern, mit Kindern zu arbeiten – und sind damit ähnlich hilfreich.

Das bedeutet aber auch, dass es vor dem Start der Entwicklung keine Architektur geben kann: Kein Code, keine Architektur. Alles, was Sie vorbereiten können, ist eine Architekturvision, ein Ziel, das alle Entwickler gemeinsam verfolgen. Die Vision kann sehr grob sein, wie die Metapher in XP oder extrem detailliert, wie die 200-seitige Architekturbeschrei-

bung, die ich 1993 für mein erstes gescheitertes Projekt geschrieben habe. Es ist aber nie mehr als eine Vision.

Von daher möchte ich die einleitende Frage umformulieren: Wie viel Architekturbeschreibung braucht das Team, um ein Softwaresystem mit adäquater Architektur zu bauen?

Heute werden viele Architekturentscheidungen bereits durch Frameworks und Application Server festgelegt, sodass man sich häufig nicht mehr mit den Details des Datenbankzugriffs, der Transaktionsmechanik oder der Verteilungsmechanismen beschäftigen muss – zumindest nicht vor Entwicklungsstart.

Aber jedes Projekt braucht eine Vision des Fachkerns, die vom ganzen Team verfolgt wird, eine Idee über die wesentlichen Fachklassen und deren Interaktion. Eine gute Architektur umfasst nur eine Handvoll Klassen, vielleicht sechs, höchstens zehn. Jeder im Team kennt diese Klassen in- und auswendig und jeder weiß, wie neue nicht ganz so bedeutende Klassen in dieses Puzzle gefügt werden müssen. Die Struktur dieser Kernklassen ist normalerweise recht stabil und bleibt meist auch beim Refaktorisieren erhalten.

Drei Quadratmeter ist die Fläche eines üblichen Whiteboards. Das sollte ausreichen, um ein UML-ähnliches Diagramm dieser Kernklassen aufzunehmen, wie es das Team sehr früh im Projekt entwickelt hat. Das Whiteboard sollte gut sichtbar für das gesamte Team im Teamraum hängen und ist der Platz, an dem Architektur- und Designdiskussionen stattfinden.

Solange ein Projekt keine spezifischen Dokumentationsanforderungen hat, kann dieses Whiteboard vollkommen ausreichen. Wenn weitergehende Dokumentationsanforderungen bestehen, empfehle ich normalerweise, die Architekturdokumentation relativ spät im Entwicklungsverlauf zu schreiben oder noch besser so weit wie möglich aus dem existierenden Code zu extrahieren.

Wenn man von dem Ergebnis überrascht ist, hat man ein schönes Thema für die nächste Retrospektive: Nicht nur, warum die Vision offensichtlich irgendwo verloren gehen konnte, sondern vor allem, warum das bei der Arbeit am Code niemandem aufgefallen ist und warum die Unit-Tests nie Alarm geschlagen haben. ■

Referenzen

[Coc05] Alistair Cockburn: Crystal Clear – A human powered Methodology for Small Teams, Addison-Wesley 2005

[Bec04] Kent Beck, Cynthia Andres: Extreme Programming Explained, 2nd Edition, Addison-Wesley 2004

[Foo] Brian Foote: Big Ball of Mud, in der deutschsprachigen Wikipedia: https://de.wikipedia.org/wiki/Big_Ball_of_Mud