Saubere Strukturen als Vermächtnis

Eine Frage der **Systematik**

Die Softwareentwicklung soll zuverlässig nicht nur funktionalen und effizienten Code liefern, sondern angesichts unvorhersehbarer Anforderungen auch wandlungsfähigen.

Dafür ist ein konstanter Fluss von Inkrementen nötig, die Feedback generieren. Dafür braucht sie einen starken Kunden/Product Owner, der engen, unmissverständlichen, aber wohlwollenden Zug ausübt. Dafür muss sie Qualitätsbewusstsein jenseits lokaler Optimierungen entwickeln.

Das sind die Eckpunkte der bisherigen Teile dieser Serie über nachhaltige Softwareproduktion. Doch das Bild ist noch nicht komplett. Denn was hat all das für Auswirkungen auf das unmittelbare Werkstück der Softwareentwicklung, den Code?

Moderne Vorgehensmodelle wie Scrum oder Kanban sind im Grunde leer. Sie sind nicht auf eine Branche fixiert. Sie interessieren sich nicht dafür, ob sie bei der Softwareentwicklung oder der Planung einer Veranstaltung zum Einsatz kommen. Das betont Jeff Sutherland auch nochmal in seinem aktuellen Buch „Scrum: The Art of Doing Twice the Work in Half the Time".

Und das zeigt sich dann auch bei ihrem Einsatz. Selbst dort, wo sie ordentlich implementiert werden, stößt die Softwareentwicklung bei ihrem Aufstieg an eine gläserne Decke. Dann geht es nicht weiter bei der Steigerung von Durchsatz oder Flexibilität. Und das liegt an ihrem Produkt, dem Code. Denn der kann auf Dauer nicht „irgendwie" aussehen, sondern braucht eine angemessene Form, um nachhaltig, wandlungsfähig zu sein.

Fehlende Kommunikation als Grundproblem

Code, der wandlungsresistent ist, wird monolithisch genannt. Er tritt dem Entwickler wie ein riesiger Block ohne Struktur entgegen. Die Entropie darin ist hoch. Was wo geschieht, ist schwer zu erkennen. Veränderungen an einer Stelle ziehen Veränderungen an vielen weiteren Stellen nach sich oder führen zu Regressionen. Der Aufwand für Anpassungen wächst exponentiell mit der Zeit.

Von dieser Art ist der Code, der sich in der Softwareproduktion „ergibt". Er kommt zustande als Summe einer Unzahl kleiner und kleinster lokaler und

kurzfristiger Optimierungen. Solcher Code ist das Ergebnis fehlender Kommunikation – obwohl doch das Vorgehensmodell agil ist, also Kommunikation wertschätzt.

Das klingt widersinnig, doch die Erklärung ist einfach: Es mangelt vielfach an einer angemessenen Kommunikation, d.h. einer, die eben auch über die Nachhaltigkeit von Code geführt wird.

Erstes Symptom der Abwesenheit solcher Kommunikation ist das Fehlen von Code Reviews. Unabhängig davon, wie die Vorstellungen davon sind, was nachhaltigen, auch sauber genannten Code ausmacht: wenn ihre Umsetzung nicht immer wieder überprüft wird, folgt der Code diesen Vorstellungen nicht. Es hat keinen Zweck, in der Hektik des Tagesgeschäftes an das Gewissen einzelner Entwickler zu appellieren, stets auch auf die Sauberkeit des Codes zu achten. Solange eine Überprüfung fehlt, solange keine Reflexion über die Vorstellungen und ihre Einhaltung in regelmäßigen Reviews stattfindet, ist sauberer Code sporadisch und zufällig – und die Nachhaltigkeit der Softwareproduktion stark begrenzt.

Saubere Strukturen

Und wenn zur Vorstellung von wandlungsfähigem Code gehört, dass er nicht monolithisch sei, sondern modular, wie entstehen dann saubere, modulare Strukturen? Auch hier ist wieder Kommunikation nötig.Korrektheit kann man nicht am Ende „hineintesten", Modularität kann man nicht am Ende „hineinreviewen". Beides muss vor der ersten Codezeile gewollt und geplant werden. Das geschieht am besten nicht durch Einzelne, sondern im Team.

Pair Programming kann dabei eine Hilfe sein. Doch auch Pair Programming braucht eine Grundlage, ein Meta-Modell von Software. Wie sollte Software grundsätzlich aussehen? Wie sollte eine konkrete Struktur grundsätzlich erarbeitet werden?

Die Agilität ist hier nicht hilfreich. Nicht nur Scrum und Kanban, auch das in Europa nicht sehr verbreitete eXtreme Programming (XP) machen dazu keine Aussage. Test-Driven Development (TDD) mahnt höchstens, dass man saubere Strukturen hinterlassen solle. Doch

„Nachhaltige Softwareproduktion braucht saubere Codestrukturen. Und saubere Codestrukturen brauchen eine saubere, systematische Kommunikation vor ihrer Implementation."

Ralf Westphal, freiberuflicher Berater, Projektbegleiter und Trainer für Themen rund um Softwarearchitektur und Teamorganisation

wie die aussehen, wie Entwickler sie erarbeiten... das bleibt im Dunkeln.

Collective code ownership

An dieser Stelle ginge es zu weit, darüber zu diskutieren, inwiefern Objektorientierte Programmierung (OOP) oder Funktionale Programmierung (FP) wandlungsfähige Strukturen zu denken, zu finden, zu implementieren erleichtern. Doch eine Entscheidung ist hier auch nicht nötig. Denn unabhängig von solchem Paradigma sollte eines klar sein:

- Ein Entwicklungsteam braucht eine einheitliche Vorstellung von den konzeptionellen Mitteln zur Strukturierung von Code. Leider ist das jenseits einiger Schlagwörter oft nicht der Fall. Von einer teilweise sehr uneinheitlichen technologischen Kompetenz ganz zu schweigen.
- Eine explizite Kommunikation zwischen Entwicklern aber auch mit dem Kunden/Anwender muss im Rahmen einer Anforderungsanalyse stattfinden. Entwickler müssen aber nicht nur wirklich verstehen, was gewünscht ist, sondern die Anforderungen sinnvoll schneiden. Das Ergebnis der Analyse müssen Inkremente sein, die einerseits für den Anwender/Kunden greifbar sind und andererseits Elemente der Codestruktur werden. Dem entsprechen die heute beliebten User Stories jedoch nur zur Hälfte. Es sind handfeste, gar testgestützte Inkremente – doch verschwinden sie bei der Codierung wie Zucker im Kaffee; es gibt kein Modul im Code, das einer User Story entspricht. Das ist negativ für das allseitige Verständnis, die Nachvollziehbarkeit und spätere Änderungen. Kommunikationsergebnisse werden vernichtet; sie spielen für die Modularisierung keine Rolle. Ein kontraproduktiver Zustand.
- Sobald geeignete Inkremente verabschiedet sind, muss eine explizite Kommunikation über den Lösungsansatz für die Anforderung beginnen, deren Ergebnis eine modulare, saubere Struktur ist. Nur wenn solcher Entwurf im Team stattfindet, kommt dessen gesamte Kreativität im Sinne einer Balance zwischen den unterschiedlichen Anforderungskategorien zum Einsatz. Außerdem werden so frühzeitig Abhängigkeiten entdeckt, die ansonsten den Umsetzungsfluss überraschend behindern würden. Und die von der Agilität angestrebte collective code ownership entsteht ganz natürlich nebenbei.

WEB-TIPP:
www.ccd-school.de

Fazit

Eine gemeinsame Vorstellung von systematischer Analyse und Entwurf im Hinblick auf saubere Codestrukturen plus gemeinsamer Analyse plus gemeinsamem Entwurf plus abschließendem gemeinsamem Review sind mithin die Bedingungen für die Möglichkeit modularen Codes.

Weniger Gemeinsamkeit bedeutet, dass nicht Teams an der Software arbeiten, sondern lediglich Gruppen gutwilliger Entwickler. Wird dieser Unterschied nicht erkannt, kommt es eher früher als später zu Stockungen in der Produktion.

Nachhaltige Softwareproduktion braucht saubere Codestrukturen. Und saubere Codestrukturen brauchen eine saubere, systematische Kommunikation vor ihrer Implementation.

Ralf Westphal