Agile Software Development

Nicole Wingenfeld, Lucas Bambauer, Rebecca Kleemann, Jaqueline Kremer

January 28, 2018

Explain why test-first development helps the programmer to develop a better understanding of the system requirements. What are the potential difficulties with test-first development?

Es gibt vier Vorteile für die ,,Test-First" Strategie.

Code Coverage: Zu jedem Codeteilstück gibt es einen zugehörigen Test.

Regression Testing: Ein Regressionstest testet bei jeder Neuerung am Code, ob alles Alte noch funktioniert. Der Regressionstest wächst also mit dem Code mit.

Simplified Debugging: Wenn ein Test fehlschlägt, weiß man direkt, wo der Fehler liegt. Es muss an dem neuesten Codestück liegen, da vorher alles funktionierte.

System Documentation: Die Tests können als Dokumentation des Codes genutzt werden.

Mögliche Probleme bei dieser Strategie:
Es ist schwierig anzuwenden, wenn man keine Erfahrung hat, da man nicht weiß, wie man etwas testen soll, was noch nicht existiert. Außerdem kommt es zu Fehlschlägen, sollten die Unittests nicht atomar sein. Ein weiterer Kritikpunkt ist, dass die Test-First Strategie kein Ersatz für andere Tests ist (z.B. für Gebrauchstauglichkeit sind Usabilitytest besser geeignet).

When would you recommend against the use of an agile method for developing a software system?

Für Entwicklerteams, die vorher eine herkömmliche Methoden benutzt haben, sind agile Ansätze schwierig umzusetzen. Damit agile Ansätze gut funktionieren können, sind große Veränderungen notwendig. Sollte sich das Entwicklerteam trotzdem für agile Entwicklung entscheiden und die damit einhergehende Umstellung, ist das noch keine Erfolgsgarantie. Mit vielen Stakeholdern ist es schwierig alle Änderungen umzusetzten. Bestimmte Projekte passen einfach nicht zur agilen Entwicklung. Ein Nachteil, welcher aber auch ein Vorteil sein kann, ist, dass man eng mit den Kunden und auch innerhalb des Teams zusammen arbeiten muss und somit viele Meetings entstehen. Es ist schwierg das Interesse des Kunden zu erhalten, der in den Prozess mit einbezogen ist.

Beispielsweise eignen sich sehr kleine oder vollkommen vertraute Projekte, für die schon perfekt einübte Arbeitsabläufe existieren, weniger gut für agiles Arbeiten.

Für große Entwicklerteams (bzw. große Projekte) mit oft wechselnder und/oder geografisch verteilter Besetzung ist es nahezu unmöglich, alle Entwickler in die Prozesse der agilen Software-Entwicklung einzubeziehen. Da dies essenziell für den agilen Ansatz ist, sind hier kleinere Teams, die im gleichen Haus arbeiten im Vorteil.

Compare and contrast the Scrum approach to project management with conventional plan-based approaches, as discussed in Chapter 23 of "Software Engineering" by Ian Sommerville. The comparisons should be based on the effectiveness of each approach for planning the allocation of people to projects, estimating the cost of projects, maintaining team cohesion, and managing changes in project team membership.

Generelle Unterschiede:

Bei plangetriebenen Entwicklungsmethoden wird das Projekt im Vorfeld detailliert durchgeplant (Wer arbeitet wann wieviele Stunden am Projekt? Welche Meilensteine gibt es? Welche Hardware und andere Ressourcen werden benötigt? Fahrtkosten? Mietkosten? ...). Daher wird bei der plangetriebenen Entwicklung bereits vor dem eigentlichen Projektstart alles festgelegt.

Bei agilen Entwicklungsmethoden werden potentielle Probleme von plangetriebenen Methoden berücksichtigt und beseitigt. Beispielsweise werden anfängliche Entscheidungen beim plangetriebenen Ansatz häufig wieder verworfen, was später zu einem erhöhten Arbeitsaufwand führt. Der agile Ansatz sieht vor, solche Entscheidungen erst im laufenden Projektbetrieb zu treffen. Somit können im Entwicklungsprozess wechselnde Anforderungen des Kunden berücksichtigt werden

Unterschiede bei der Aufteilung der Arbeit auf Mitarbeiter:

Plangetrieben: Die Aufteilung der Mitarbeiter ist hierbei schon beim Projektstart fix. Der Projektleiter legt fest, wer wann woran arbeitet (Gantt-Charts).

Agil: Die Aufteilung der Mitarbeiter ist im laufenden Projektbetrieb variabel bzw. anpassbar. Jeder Mitarbeiter entscheidet selbst, welche Story er implementiert. Vorher wurde vom gesamten Team gemeinsam ein Ranking für die Implementierungsdauer dieser Stories erstellt.

Unterschiede bei der Berechnung von Projektkosten:

Plangetrieben: Die Kostenberechnung ist vorausschauend und gut (vor dem eigentlichen Projektstart) planbar. Im Projektverlauf auftretende Anforderungsänderungen des Kunden können sich allerdings rückwirkend auf die berechneten Kosten auswirken.

Agil: Die anfängliche Kostenberechnung ist deutlich schwieriger als bei plangetriebenen Methoden. Üblicherweise werden erstmal ein paar Stories implementiert, anhand derer zusammen mit dem Kunden ermittelt werden kann, wie lange das Projekt dauern wird.

Unterschiede beim Team-Zusammenhalt:

Plangetrieben: Jeder Entwickler hat fest zugewiesene Tasks, die er in einem bestimmten Zeitraum zu erledigen hat. Somit ist bereits alles geregelt und es besteht kaum Kommunikationsbedarf innerhalb des Teams. Daher herrscht ein eher geringer Team-Zusammenhalt.

Agil: Alle Entwickler wissen jederzeit, woran alle anderen Entwickler arbeiten. Dadurch können Abhängigkeiten zwischen Stories effizient ermittelt und berücksichtigt werden. Es wird außerdem insgesamt viel kommuniziert. Dadurch: Hoher Team-Zusammenhalt.

Unterschiede bei personellen Veränderungen im Projektteam:

Plangetrieben: Personelle Ausfälle wirken sich hier deutlich auf den Projektverlauf aus. Bestehen beispielsweise Abhängigkeiten zwischen Tasks, kann sich durch einen personellen Ausfall im schlimmsten Fall die gesamte Projektdauer um die Dauer des Ausfalls verlängern.

Agil: Personelle Veränderungen im Projektteam können in einem bestimmten Rahmen gut verkraftet werden (wenn sich die halbe Belegschaft verflüchtigt, ist das auch hier schwierig). Da jeder Entwickler genau weiß, welche Story sich die betreffende Person herausgesucht hat, kann der Ausfall durch eine Neuverteilung recht gut kompensiert werden.

Suggest four reasons why the productivity rate of programmers working as a pair might be more than half that of two programmers working individually.

1. Produktivitätsverlust durch gegenseitiges Feedback, aber zeitleich Verbesserung der Software-Qualität. Dies liegt unter anderem am ständigen Codereview und dem sofortigen Feedback des ,,Navigators".
2. Verlieren in Details. Fehler werden früh erkannt (,,4 Augen sehen mehr als zwei"). Dadurch sieht bzw. diskutiert man oft aber auch über Fehler, die keine sind.
3. Gegenseitiges Lernen wird hierdurch gefördert. Entsprechend müssen aber auch Zusammenhänge gegenseitig erklärt und erläutert werden, was wiederum einen Zeitverlust bedeutet.
4. Abrutschen in Off-Topic Themen. Z.B. wie verbringt man die Mittagspause, wie war das Wochenende, ...