Flow Check

15 Schritte zum besseren Software-Lieferprozess

Matthias Berth
Christoph Lefkes
SoftwareLiefern.de

Der Flow Check ist ein einfacher Test, den wir entwickelt haben, um Software-Lieferprozesse einzuschätzen. Er besteht aus ein paar Ja/Nein Fragen, die man schnell beantworten kann. Der Check ist unabhängig von der Methodologie, die Sie einsetzen. Es ist also egal, wieviel Agil, Scrum, Kanban oder sonstwas Sie machen.

Die Fragen zielen auf Bereiche, in denen Verbesserungen eine große Hebelwirkung haben. Man sollte erstmal einige der Themen hier abarbeiten, bevor man sich Continuous Delivery oder DevOps vornimmt.

Der Score ist auf Projektorganisationen zugeschnitten. Wenn Sie Software-Wartung oder kontinuierliche Weiterentwicklung von Produkten und Services betreiben, lassen Sie einfach die projektbezogenen Fragen weg.

1 Liefern Sie regelmäßig, mindestens einmal pro Monat, Software?

Liefern heißt: die Software ist im Produktivbetrieb, mit echten Anwendern, echten Daten und echten Transaktionen. Wenn man weniger oft liefert, wächst die Versuchung, Aufgaben nicht komplett abzuschließen. Dann kann es leicht zu bösen Überraschungen kommen, wenn das Projekt dann „fertig“ ist. Außerdem kann es echtes Feedback nur für Features geben, die im Produktivbetrieb angekommen sind.

Man kann natürlich auch im ersten Monat an eine kleine Gruppe von Anwendern ausliefern und diese Gruppe dann nach und nach vergrößern.

2 Priorisieren Sie nach Verzögerungskosten?

Zur Bestimmung von Verzögerungskosten stellt man sich die Frage: *Wie ändert sich das finanzielle Ergebnis über die gesamte Lebensdauer, wenn wir die Aufgabe oder das Projekt später fertigstellen?*

Wenn Sie z.B. einen Arbeitsablauf automatisieren und damit 50.000 Euro pro Monat sparen können, betragen die Verzögerungskosten 50.000 Euro pro Monat.

Verzögerungskosten sollten auf Projektebene und auf Feature-Ebene bestimmt werden. Dazu gehört, dass das Team die Verzögerungskosten kennt: „Wenn wir Feature A eine Woche später ausliefern, kostet das 2.000 Euro.“ „Das gesamte Projekt hat Verzögerungskosten von 20.000 Euro pro Monat.“

Die Frage nach dem wirtschaftlichen Nutzen eines Projekts wird dabei gleich mitbeantwortet.

3 Begrenzen Sie die Anzahl parallel laufender Projekte?

Parallel laufende Projekte verursachen Unterbrechungen und damit Multitasking, eine der Hauptursachen für Verschwendung. Wenn Sie die Anzahl parallel laufender Projekte nicht begrenzen, ist es auch leicht, eine Fachabteilung zu beruhigen, indem man

„schon mal anfängt“. Einfach ein neues Projekt anfangen zu können vermeidet in solchen Situationen oft harte Prioritätseintscheidungen.

Parallel laufende Projekte werden nicht früher fertig, im Gegenteil. Die Verzögerungskosten sind höher, wenn man Projekte parallel abarbeitet, die man auch sequenziell abarbeiten könnte.

4 Ist die Projektlaufzeit 6 Monate oder weniger?

Innerhalb von 6 Monaten ändern sich die Prioritäten im Unternehmen so stark, dass es sinnvoll ist, neue Entscheidungen über Projekte zu treffen. Das heißt nicht, dass ein größeres Vorhaben nicht in drei Teile aufgeteilt werden kann. Allerdings muss dann jeder Teil einen selbstständigen Nutzen haben und man muss damit rechnen, dass der folgende Teil erstmal geparkt wird zugunsten anderer Projekte.

Mit einer Begrenzug der Laufzeit vermeidet man auch Zombieprojekte und das übermäßige Anhäufen von Projektzielen und Features.

5 Ist jeder Entwickler höchstens einem Projekt zugeordnet?

Wenn jemand mehr als einem Projekt zugeordnet ist, entstehen automatisch Unterbrechungen: während der Arbeit an Projekt A kommt jemand mit einer Frage zu Projekt B, die auch noch möglichst schnell beantwortet werden sollte. Jede Unterbrechung bedingt einen Kontextwechsel, der ca. 20 Minuten Zeitverlust bringt.

Außerdem wird das Zeitbudget pro Woche schnell kleiner bei mehreren Projekten. Nehmen wir mal 3 Projekte, in der Aufteilung 60% (das Hauptprojekt soll bitteschön den Großteil der Zeit des Entwicklers bekommen) 20%, 20%. Schon damit haben wir nur noch 8 Stunden pro Woche für Projekte 2 und 3 übrig. Bei dem normalen overhead (E-Mmail lesen, Meetings, Fragen beantworten,. . .) sind es 6 Stunden oder weniger an konzentrierter Arbeitszeit, die jeweils noch für Projekte 2 und 3 zur Verfügung stehen.

Damit wird der Entwickler schnell zum Engpass. Außerdem verliert er den Kontext: die Lücken zwischen den Arbeitsepisoden am jeweiligen Projekt werden größer.

Sie können sich diesen Punkt auch noch gutschreiben, wenn ein Entwickler wenigstens für eine ganze Woche nur einem einzigen Projekt zugeordnet ist.

6 Haben Entwickler direkten Zugang zur Fachabteilung?

Kleine und große Fragen zur „Fachlichkeit“ entstehen laufend. Oft sind es Fragen, die man beim Schreiben der Spezifikation nicht wirklich vorhersehen konnte. Schließlich ist es der ultimative Verständnis-Test, wenn man in der Lage ist, etwas nicht nur zu erklären, sondern auch noch zu programmieren (einem dummnen Computer zu erklären).

Viel Zeit geht verloren, wenn ein Entwickler zur Beantwortung solcher Fragen warten muss. Oft wird der Zugang zum Wissen der Fachseite rationiert (z.B. auf die monatlichen Projektsitzungen beschränkt) oder erschwert, indem z.B. alle Anfragen bitte über den Projektleiter and den Ansprechpartner der Fachseite weiterzugeben sind.

In so einer Situation hat der Entwickler zwei Möglichkeiten: erstens, auf die Antwort warten und sich solange eine andere Aufgabe vornehmen; oder zweitens, die Frage garnicht erst stellen, sondern die wahrscheinlichste Antwort annehmen und weitermachen. Beide Möglichkeiten sind mit Verlusten verbunden: im ersten Fall verzögert sich das Feedback, es entstehen Umschaltkosten und eine weitere Aufgabe kommt auf den Stapel. Im zweiten Fall rät der Entwickler manchmal falsch und verbaut dann diese falsche Annahme solange weiter, bis das Problem irgendwann im Test zutage tritt.

Falls Sie schon interdisziplinäre Teams haben, Glückwunsch!

7 Kann jede Aufgabe von mindestens zwei Entwicklern bearbeitet werden?

So etwas wie: „Das Problem in der Produktsuche muß von Steve behoben werrden, er kennt sich da aus“ klingt erstmal sinnvoll. Problematisch wird es dann, wenn Steve der einzige ist, der sich ohne lange Einarbeitungszeit dem Problem widmen kann. Schon zufällige Schwankungen sorgen dann dafür, dass einzelne Mitarbeiter immer mehr „exklusive Kompetenzen“ aufhäufen und dadurch leicht zum Engpass werden. Urlaub, Krankheit und Kündigung von Steve brauchen wir dabei noch garnicht zu berücksichtigen. Es reicht völlig, dass die drei dringendsten Aufgaben alle aus Steves exklusivem Kompetenzbereich kommen.

Darüber hinaus hindern Sie Paula und Tim daran, sich weiterzuentwickeln, indem sie Steve anspruchsvollere Aufgaben abnehmen.

8 Haben Software-Entwickler planbare unterbrechungsfreie Zeiten?

Während viele Manager kein Problem damit haben, ständig von einer Aufgabe zur nächsten zu springen, brauchen Entwickler lange unterbrechungsfreie Zeiten, um sich in ihre Arbeit vertiefen zu können. Schon die Aussicht auf eine Unterbrechung (der Chef will irgendwann heute nachmittag nochmal anrufen) reicht oft, um die nötige Konzentration zu verhindern. Daher sollten Sie dafür sorgen, dass es diese unterbrechungsfreien Zeiten gibt. Eine Regel wie „Keine Meetings zwischen 8 und 14 Uhr“ kann helfen.

Übrigens sind Unterbrechungen auch ein Grund dafür, dass viele Entwickler gern spät kommen und spät gehen – nach 16 Uhr klingelt das Telefon nicht mehr so oft. Falls Sie die Arbeitszeit noch nicht flexibilisiert haben, oder falls Arbeiten von zu Hause aus noch nicht erlaubt sein sollte, probieren Sie es aus.

9 Gibt es automatisierte Unit Tests?

Unit Tests testen, ob der Code aus der Sucht des Entwicklers das Richtige tut. Zum Beispiel, ob ein Betrag von 1345 Euro und 99 Cents angezeigt wird als „1.345,99 €“. Diese Tests werden von Entwicklern geschrieben, und laufen so schnell ab, dass tausende Tests in wenigen Sekunden oder Minuten einen Großteil des Systems abdecken können. Unit tests geben schnelles Feedback, falls ihre Änderungen am Code zu (erwarteten) Fehlern führen. Wenn z.B. ein Entwickler den Code aus Versehen so geändert hätte, dass Zahlen nach US-amerikanischer Art (mit Tausender-Komma und Dezimal-Punkt) dargestellt werden, würde der Unit Test oben das finden, weil statt des erwarteten „1.345,99 €“ nun „1,345.99 €“ ausgegeben wird.

10 Gibt es automatisierte User Acceptance Tests?

User Acceptance Tests überprüfen, ob das System aus Sicht des Anwenders das Richtige tut. Die Tests sollten mindestens alle geschäftskritischen Prozesse abdecken. Ein Beispiel:

```
Szenario: Rabattaktionen lassen sich nicht kombinieren,
  es wird aber automatisch der günstigste Rabatt übernommen

Angenommen,
  es gibt eine Rabattaktion für 10% mit dem
  Gutscheincode "springsale"
Und ich bin ein Neukunde, der 10€ Rabatt für die
  Erstbestellung bekommt
Wenn ich für 50 Euro einkaufe
Dann bekomme ich nur den Neukundenrabatt
Und ich bezahle 40 Euro
```

Die Szenarien sind komplexer als bei Unit Tests, sie können z.B. durch Automatisierung des Web-Browsers getestet werden. Idealerweise sind sie so geschrieben, dass jemand ohne Programmierkenntnisse sie ohne weiteres lesen und überprüfen kann, wie im obigen Beispiel. Lediglich beim Schreiben neuer User Acceptance Tests braucht es dann etwas Unterstützung durch Entwickler, um Textbausteine wie *es gibt eine Rabattaktion für 10% mit dem*

Gutscheincode „springsale“ in die richtigen Datenbankoperationen umzusetzen. Falls Sie die Prosa für das obige Szenario etwas hölzern finden, liegt es sicher daran, dass es in dieser Form schon maschinenlesbar ist.

11 Praktizieren Sie Continuous Integration?

Continuous Integration bedeutet, dass die Änderungen mehrerer Entwickler in kurzen Zeitabständen zusammengeführt werden. Das Gesamtsystem wird dann erstellt („build“) und automatisiert getestet. So werden Unstimmigkeiten schnell bemerkt und sofort behoben. Oft werden z.B. alle Unit Tests und alle automatisierten User Acceptance Tests als Teil einer Build-Pipeline ausgeführt. Falls einer der Tests fehlschlägt, werden die Entwickler benachrichtigt und beheben das Problem sofort.

So wird gesichert, dass das System immer in einem „guten“ Zusatnd ist. Wenn Fehler auftreten, sind die Ursachen leicht auf eine überschaubare Anzahl an Änderungen einzugrenzen: Eben hat noch alles funktioniert, jetzt schlägt der Test zur Formatierung von Euro-Beträgen fehl. Die Ursache muß also irgendwo in den letzten Änderungen liegen, zum Glück sind das nur 50 Zeilen Code.

12 Können Sie eine neue Testumgebung automatisiert bereitstellen?

Wenn neue Testumgebungen erst langwierig manuell eingerichtet werden müssen, sind sie oft ein knappes Gut. Ein Tester kann dann nicht mal eben eine (mehr oder weniger originalgetreue) Kopie der Produktivumgebung parallel zu einer Umgebung mit der Version von letzter Woche laufen lassen, um einem schwierigen Problem auf den Grund zu gehen. Manuelle Einrichtung heißt auch, die Konfiguration der Produktivumgebung wird sich in vielen kleinen Details von den Testumgebungen unterscheiden. Schon kleine Unterschiede können zu Stillständen und Trouble shooting von unbestimmter Dauer führen, wenn die Software live geht.

13 Begrenzen Sie Work in Process (WIP)?

Zuviele angefangene Aufgaben (work in process - WIP) verlängern die Durchlaufzeit. Jede Aufgabe, die zwar „fertig entwickelt“, aber noch nicht getestet oder abgenommen ist, kann zu Nacharbeit und weiteren Unterbrechungen führen, wenn das Feedback endlich hereinkommt. Qualitätsprobleme bleiben unentdeckt, falsche Annahmen werden weiter fortgeschrieben.

Das Sichtbarmachen von WIP ist der erste Schritt, um es zu begrenzen und schließlich systematisch zu verringern.

14 Messen Sie Durchlaufzeiten?

Wissen Sie, wie lange eine Aufgabe im Entwicklungsprozess verbringt, von der Entscheidung „wir machen das jetzt“ bis zur Freischaltung im Produktivbetrieb? Wisssen Sie, wieviel davon auf die eigentliche Entwicklung, das Testen, das Deployment entfällt? Auch hier gilt: Durchlaufzeiten zu messen ist der erste Schritt, um sie zu verringern.

15 Gibt es eine kontinuierliche Verbesserung des Prozesses?

Ohne die bewußte kontinuierliche Verbesserung des Software-Lieferprozesses gibt es nur die spontane kontinuierliche Verschlechterung. Für diesen Punkt ist es erstmal egal, ob Sie die Verbesserungen im Rahmen einer Sprint-Retrospektive anstoßen, oder ob das Team jeden Tag mit 30 Min Verbesserungen startet usw. Trotzdem, es zählen nur Verbesserungen, die auch wirklich umgesetzt werden.

Ergebnis

1 Liefern Sie regelmäßig, mindestens einmal pro Monat, Software?

2 Priorisieren Sie nach Verzögerungskosten?

3 Begrenzen Sie die Anzahl parallel laufender Projekte?

4 Ist die Projektlaufzeit 6 Monate oder weniger?

5 Ist jeder Entwickler höchstens einem Projekt zugeordnet?

6 Haben Entwickler direkten Zugang zur Fachabteilung?

7 Kann jede Aufgabe von mindestens zwei Entwicklern bearbeitet werden?

8 Haben Software-Entwickler planbare unterbrechungsfreie Zeiten?

9 Gibt es automatisierte Unit Tests?

10 Gibt es automatisierte User Acceptance Tests?

11 Praktizieren Sie Continuous Integration?

12 Können Sie eine neue Testumgebung automatisiert bereitstellen?

13 Begrenzen Sie Work in Process (WIP)?

14 Messen Sie Durchlaufzeiten?

15 Gibt es eine kontinuierliche Verbesserung des Prozesses?

Wenn Sie weniger als 12 von 15 Fragen mit Ja beantwortet haben, verschenken Sie definitiv Geld.

Einige Veränderungen sind einfach, man muss sie nur machen. Andere erfordern Aufwand und Struktur. Falls Sie dabei Unterstützung brauchen:

Dr. Matthias Berth, Christoph Lefkes
kontakt@SoftwareLiefern.de
SoftwareLiefern.de