Der dornige Weg zum agilen Team, Teil 3: Warum muss der Build immer brechen?

Der Güte-Indikator

VON JIRI LUNDAK

Kontinuierliche Integration der Arbeitsergebnisse aller Entwickler (der so genannte Build), bei der alle Programmteile fehlerfrei kompilieren und alle Unit-Tests laufen müssen, hat sich in der Zwischenzeit zum Standardwerkzeug vieler Entwicklungsteams, besonders bei solchen, die unter der agilen Fahne segeln, durchgesetzt. Und gerade der tägliche – oder gar kontinuierliche – Build ist ein Indikator für die Güte eines agilen Teams. Weshalb kann man das so sagen?

Warum sind häufige und regelmäßige Builds des Softwaresystems so wichtig für ein agiles Team? Weil sie ein Indikator dafür sind, ob die zu entwickelnde Software funktioniert, ob die Team-Mitglieder in der Lage sind, die programmierten Teile in ein harmonisches Ganzes zu integrieren und – dank der Möglichkeit der automatischen Testausführung –, und ob die Qualität des Endproduktes stimmt.

Je häufiger der Build läuft, desto intensiver und früher das Feedback. Dabei kommt es nicht in erster Linie darauf an, ob der Build erfolgreich durchgelaufen ist oder nicht, sondern darauf, dass er überhaupt regelmäßig, in möglichst kurzen Zeitabständen, ausgeführt wird. Denn dieses Feedback – ob nun negativ oder positiv – sollten wir zur Verbesserung des Entwicklungsprozesses nutzen.

Insofern ist eine Häufung von fehlgeschlagenen Builds ein Zeichen dafür, dass – besonders in einem agilen Team – etwas nicht stimmt. Nicht dass wir uns falsch verstehen: Nicht jedes Fehlschlagen eines Builds ist ein alarmierendes Ereignis. Es gleicht vielmehr einem Blick in den Spiegel. Ich stelle fest, dass die Krawatte nicht richtig sitzt und habe zwei Möglichkeiten: nachzubessern oder es sein zu lassen. Wenn ich es sein lasse, in einer Viertelstunde wieder in den Spiegel schaue und mich erneut ärgere, dass sie schief sitzt, dann habe ich nichts aus dem letzten Blick in den Spiegel gelernt.

Wenn der Build zusammenbricht

Die Gründe für fehlschlagende Builds sind vielfältig, wie man in Abbildung 1 sieht. In einem großen Projekt mit mehreren, zum Teil verteilten Teams brach nahezu jeder der täglichen Builds. Viele der genannten Gründe kamen zusammen, sodass es sehr schwer war, die wahren Ursachen klar auszumachen. Man experimentierte mit den verschiedensten Vorgehensweisen. So verlegte man den nächtlichen Build auf die Mittagszeit, damit man sich gemeinsam um den Build kümmern konnte, falls er fehlschlug. Man ernannte einen Build-Verantwortlichen, der – jede Woche rotierend – den anderen Entwicklern auf den Schlips treten musste, damit der Build so schnell als möglich geflickt wurde. Bis ein gebrochener Build wieder lief, wurde ein Eincheckstopp eingeführt.

Mit welchem Erfolg? Nun, die Builds schlugen trotzdem mit schöner Regelmäßigkeit fehl. Warum war dies wohl so? Wie immer in einem komplexen System gab es mehrere Faktoren, die sich gegenseitig verstärkend beeinflusst haben [1]. Erst aufgrund retrospektiver Kohärenz konnte man erkennen, was die wahren Ursachen für die Probleme mit dem Build waren.

Eines der Probleme, das die Teams plagte, bestand darin, dass einzelne Team-Mitglieder die Unit-Tests nicht lokal ausführten, bevor sie ihre Änderungen eincheckten. Wir wissen jedoch, dass in einer agilen Umgebung das Ausführen der Unit-Test essenziell ist. Sie sind das Sicherheitsnetz jedes Team-Mitglieds, das es ihm ermöglicht, mit möglichst ruhigem Gewissen seine Änderungen in die gemeinsame Code-Basis zu integrieren.

Schnell stellte sich heraus, dass die Ursachen für dieses kontraproduktive Ver-

Der dornige Weg zum agilen Team – die Serie

- Warum agile Teams so schwer zu schaffen und am Leben zu erhalten sind
- Was ist kein agiles Team?
- Warum muss der Build immer brechen?
- Selbstorganisation – Chaos oder Wunderwaffe?
- Was soll heißen: „Ich bin fertig“?
- Metriken: Ballast oder Notwendigkeit?
- Abnahmen: Verhinderer des Projekterfolges?
- Brauchen agile Teams agile Kunden?
- Gedeihen agile Teams nur in agilen Unternehmen?
- Agiles Vorgehen als rotes Tuch
- Nur die Besten sind für das agile Team gut genug
- Cross-funktionale Teams – eine Illusion?
- Wo bleibt der Manager im agilen Team?
- Agile Teams skalieren nicht!
- Wie erhalte ich ein agiles Team am Leben?

halten noch tiefer lagen. Die Unit-Tests lokal auszuführen dauerte recht lange (rund 25 Minuten), was dazu führte, dass die Entwickler die Tendenz hatten, seltener einzuchecken, das heißt, ihre Ergebnisse weniger oft zu integrieren [2]. Dies wiederum hatte zur Folge, dass gegen Ende einer Monatsiteration die Entwickler unter Druck standen, ihren Code doch noch zu integrieren. Je mehr auf einmal eingecheckt wurde, desto mehr hatten die einzelnen Entwickler lokal wieder zu mergen, was wiederum die Eincheckzeit erhöhte, was weniger Tests zur Folge hatte, was wieder zu gebrochenen Tests führte, was wiederum bewirkte, dass das Eincheckfenster immer kleiner wurde. Eine tödliche, abwärts führende Spirale.

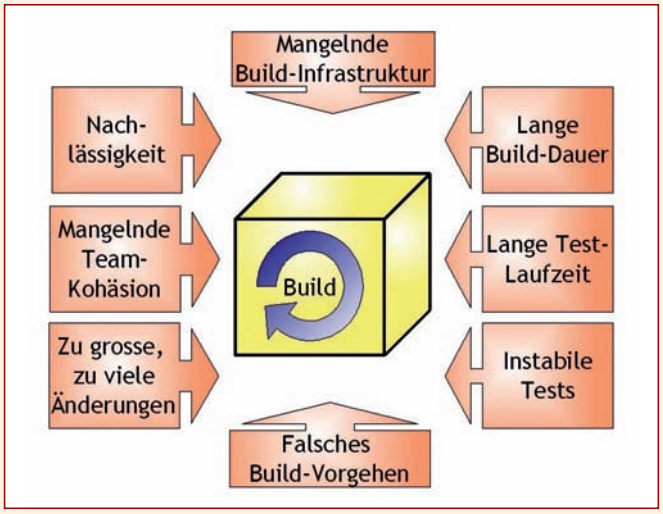
Noch schlimmer wirkt es sich aus, wenn wir die Tester mit in dieses Bild einbeziehen. Wenn man es während mehrerer Tage nicht schafft, einen stabilen Build zu schaffen, kann nichts auf die Testsysteme ausgeliefert werden, was wiederum bewirkt, dass die automatisierten Akzeptanztests nicht greifen und die Tester das System nicht rechtzeitig manuell prüfen können. Der Teufelskreis wird somit fortgesetzt.

Der Pulsschlag des agilen Teams

Wir sehen somit, dass die Realität von einem agilen Vorgehen weit entfernt sein kann. Je kürzer die Build-Zyklen und je weniger die Builds brechen, desto agiler das Team. Bezeichnend ist auch, was passiert, wenn ein Build bricht. Wie reagieren die Team-Mitglieder? Ist es ihnen egal? Schaut jeder nur zu seinem eigenen Gärtchen? Dass überhaupt ein Build-Verantwortlicher ernannt werden muss, deutet auf mangelnde Initiative des Einzelnen und eventuell fehlendes „Wir"-Gefühl hin. Leider gibt es in vielen Organisationen immer noch die Einstellung, dass jeder nur für seinen Bereich, sein Programmmodul, sein Tortenstück des Entwicklungskuchens verantwortlich ist. Das der Gesamt-Build bricht, muss dann wohl das Problem eines anderen sein!

Gerade darin, wie ein Team oder gar mehrere Teams mit so praktischen Dingen wie dem Build umgehen, zeigt sich, inwiefern die einzelnen Individuen sich wirklich einem Team und dem gemeinsamen Ziel verpflichtet fühlen [3]. Wenn der gemeinsame Erfolg vor dem Erfolg des einzelnen Individuums steht, dann wird allen daran gelegen sein, dass die Builds so wenig wie möglich brechen und dass gleichzeitig die automatischen Tests so flächendeckend wie nötig sind, damit man als Team sein Iterationsziel erreichen kann und diese Zuversicht mit jedem Build untermauert wird.

Abb. 1: Negative Kräfte



... und er dreht sich doch!

In Anlehnung an Galileis Worte wollen wir festhalten, dass Builds nicht brechen müssen, sondern durchaus verbessert werden können. Dazu müssen die notwendigen Rahmenbedingungen geschaffen werden. Frühe und häufige Integrationen sind zwingend, damit in einem agilen Entwicklungsprozess dem Kunden qualitativ hochstehende Software ausgeliefert werden kann. Wie es auch heißt: „Never accept big-bang integration!" [4]

Damit wir häufig integrieren können, müssen unsere Tests auf Geschwindigkeit getrimmt sein. Je schneller sie durchlaufen, desto häufiger werden Entwickler sie auch ausführen und desto weniger werden sie eine Bürde für die Zeit des Eincheckens. Es kann durchaus auch Sinn machen, nur eine Untermenge aller verfügbarer Tests vor dem Einchecken zur Pflicht zu machen, während lange laufende Tests ausschließlich auf dem Build-Server ausgeführt werden. Hier gilt es, erfinderisch zu sein und zum Beispiel die meisten Tests möglichst von einer Datenbank unabhängig zu machen.

Laufende Builds sind auch ein Musterbeispiel für einen Bereich, in dem die Gemeinschaftsverantwortung zum Tragen kommt. Man kann die Verantwortung nicht einer Person zuschieben, die für einen reibungslosen Build verantwortlich ist, sondern *jedes* Team-Mitglied muss einen Teil dazu beitragen. In diesem Zusammenhang ist auch leicht zu verstehen, warum es so wichtig ist, dass die von Extreme Programming bekannte Praxis vom kollektiven Code-Besitz (Collective Code Ownership) so wichtig ist. Denn wer wird den Fehler im Build korrigieren, wenn der Verursacher im Urlaub oder in einer Sitzung ist?

Jiri Lundak ist mit mehr als 20 Jahren Erfahrung ein Urgestein der Softwareentwicklung. Er arbeitet gegenwärtig als Entwicklungsleiter für Löwenfels Partner in der Schweiz. Er entwickelt als Architekt immer noch aktiv Software, hat aber erkannt, dass die meisten Probleme in Entwicklungsprojekten nicht technischer, sondern vielmehr menschlich-sozialer Natur sind. Als praktizierender Certified ScrumMaster lebt er die Rolle eines Vorkämpfers für agiles Verhalten innerhalb der Firmenorganisation. Kontakt: jiri.lundak@loewenfels.ch.

Links & Literatur

[1] Ralph D. Stacey: Managing the Unknowable: Strategic Boundaries Between Order and Chaos, Jossey-Bass, 1992

[2] Jutta Eckstein, Hubert Baumeister: Extreme Programming and Agile Processes in Software Engineering (Proceedings): Scaling Coninuous Integration, Springer, 2004

[3] Jean Tabaka: Collaboration Explained: Facilitation Skills for Software Project Leaders, Addison-Wesley, 2006

[4] Venkat Subramanian, Andy Hunt: Practices of an Agile Developer, Pragmatic Bookshelf, 2006