Software-Tests in PHP-Anwendungen

Von: Gjero Krsteski, Web-Programmierer, Berater, Trainer
Datum: 15.06.10
Ort: Köln
Homepage: krsteski.de
E-Mail: gjero@krsteski.de

Inhaltsverzeichnis

Software-Tests in PHP-Anwendungen (Teil I)

Zum Testen von Software gehören sowohl das Aufspüren von Fehlern während der Entwicklung, als auch die Überprüfung des Gesamtproduktes. Das heißt, man sucht zunächst in einzelnen Codefragmenten nach Fehlern, und überprüft dann das Gesamtpaket auf seine Vollständigkeit und Korrektheit hin. Unzureichende oder unvollständige Dokumentationen führen häufig zu einer unzulänglichen Erfassung von fehlerhaften Anforderungen. Dies fällt besonders bei sich wiederholenden und sich schnell verändernden PHP-Entwicklungen ins Gewicht.

Dies macht eine Überprüfung des Gesamtproduktes schwer und ist ein Argument gegen die klassische Software-Überprüfung in PHP-Projekten. Einige Eigenschaften von PHP in der LAMP-Umgebung (LAMP= kombinierter Einsatz von **L**inux, **A**pache, **M**ySQL und **P**HP) beeinflussen die Herangehensweise an Tests von Applikationen mit hohen Qualitätsansprüchen.

Der Grund dafür ist, dass PHP als nicht typisierte Sprache die Möglichkeit bietet, in hohem Tempo neue Funktionalitäten zu bestehender Software hinzuzufügen und zu ändern. Anwendungsteile, die mit PHP implementiert wurden, bedürfen keiner Neukompilierung. Die Genauigkeit des Gesamtkontextes kann noch während der Laufzeit des Prozesses geprüft werden. Des Weiteren muss auch sichergestellt werden, dass die Rückgabewerte von Methoden der Quellcode-Dokumentation bzw. dem erwarteten Typ entsprechen. In Projekten mit größeren Teams wird es häufig zur Wiederverwendung von Komponenten – oft auch in einem vom Software-Autor nicht erwarteten Kontext – kommen. Somit ist immer noch das Wichtigste nicht genannt: Sind neue Funktionen korrekt umgesetzt, und funktioniert die alte Funktionalität noch?

Techniken zur Organisation von Software-Tests

Es ist notwendig, die verfügbaren Techniken zum Testen von Software so zu organisieren, dass die positiven Eigenschaften der LAMP-Umgebung sicher und kontrollierbar bleiben. Ein schnell wachsender Quellcode, der stetig verändert wird, kann nicht zeit- und kosten-effektiv getestet werden. Deswegen sollte zunächst sichergestellt werden, dass die Kernfunktionen fehlerfrei arbeiten. Die fehlende Typisierung der Programmiersprache macht die Überprüfung von Rückgabewerten insbesondere im Grenzbereich notwendig. Da Kunden nach einem möglichst kurzen Entwicklungsprozess funktionierende Software erwarten, sollten Tests die Entwicklung unterstützen und für ein rasches Vorankommen sorgen. Um dies zu gewährleisten, sollte ein wiederkehrender Automatismus in das Projekt eingebaut werden, damit die Tests jeden Tag vom Server automatisch ohne jegliche Beziehung zu den neuen Erweiterungen am Quellcode ausgeführt werden können. Zeigt dabei ein Test plötzlich eine Unstimmigkeit auf, hat sich in die neuen Quellcode-Erweiterungen ein Fehler eingeschlichen.

Es fällt oft leichter, Software zu testen, die man nicht selbst entwickelt hat – und ist häufig auch effektiver. Fehler sind dabei schnell gefunden, ein Verursacher eventuell ebenso. Ein bisschen anders liegen die Dinge, wenn eigene Projekte auf Fehler hin untersucht werden sollen. Dann fehlt die Objektivität und der Entwickler neigt dazu, mit sich selbst nicht zu hart ins Gericht zu gehen. Manchmal sieht er auch einfach den Wald vor lauter Bäumen nicht mehr. In diesem Fall ist es notwendig, nach erfolgter Implementierung die eigene Rolle zu wechseln und einen neuen Blickwinkel einzunehmen.

Wenn man eine Software testet, wird man auf der Suche nach folgenden Dingen sein:

- Nicht definierte Anforderungen: Eigenschaften und Funktionen der Software, die im Zuge der Entwicklung der Anwendung notwendig wurden, um deren vollständige Funktionsfähigkeit zu erhalten, obwohl diese Eigenschaften nicht explizit angefordert

wurden.
- Unvollständige oder fehlende Anforderungen: Eigenschaften der Software, die angefordert waren, aber nicht so implementiert wurden wie erforderlich.
- Allgemeine Fehler: Dinge, die nicht so funktionieren, wie sie sollten.

Während der Entwicklung ist man als Programmierer natürlich darum bemüht, die Anforderungen zu erfüllen und gleichzeitig Projektmanager und Kunden darauf hinzuweisen, welche Erweiterungen das Projekt außerdem verbessern könnten. Doch gibt es einige Hindernisse, die dem Programmierer dabei im Wege stehen:

- Manchmal reicht das Budget einfach nicht aus
- Entwickler legen die Anforderungen der Kunden falsch aus
- Knappe Abgabetermine. Die Zeit reicht gerade aus, um die Anforderungen zu erfüllen
- Bei einem Wechsel von Projektbetreuern und Entwicklern entsteht eine Lücke in der Wissensübertragung
- Immer wiederkehrende Tätigkeiten führen beim Entwickler zur Abstumpfung, Details werden übersehen

Software-Tests in PHP-Anwendungen (Teil II)

Um gezielt und systematisch nach Fehlern suchen zu können, ist es sinnvoll, dass die Testarbeit schnell, kosten-effizient und auch interessant ausfällt. Eine gute Organisation der hierfür notwendigen Werkzeuge kann dabei sehr hilfreich sein. So ist es beispielsweise wichtig, vor Beginn der Arbeit alle vorhandenen Dokumentationen wie Pflichten- oder Lastenhefte oder Protokolle aus Besprechungen zur Verfügung zu haben, um Informationen schnell nachschlagen zu können. In einem System zur Fehlererfassung können zudem alle gefundenen Fehler mit ihrem jeweiligen Status erfasst werden. Dabei sollten die Fehler den Software-Versionen zugeordnet werden, um den Verlauf der Fehlerfindung zu dokumentieren. Alle am Projekt beteiligten Personen sollten Zugang zu diesem System haben und Einträge darin verfassen können. Eine kleine Anleitung zur Fehlerbeschreibung kann hilfreich sein.

Testen ist eine geistig anstrengende Tätigkeit

Damit die Tests zu einem zufriedenstellenden Ergebnis führen und effektiv sind, sollte dem Entwickler oder Tester Zeit ausschließlich für das Testen zur Verfügung gestellt und diese wenn möglich bereits zu Beginn bei der Aufwandsschätzung berücksichtigt werden. Testen ist eine Arbeit, die Konzentration erfordert und nicht unter Zeitdruck geschehen sollte, um Fehler zu vermeiden. Denkbar sind je nach Projekt sowohl eine große als auch mehrere kleine Testphasen. Die entwickelte Software sollte möglichst erst auf einem Zwischensystem installiert und getestet werden. Die Testumgebung, bzw. das Zwischensystem, sollte dabei der späteren Betriebsumgebung technisch möglichst ähnlich sein. Wird im Zwischensystem ein Fehler entdeckt, wird dieser zunächst im Entwicklungssystem behoben und die Änderung dann wiederum ins Zwischensystem eingespeist. Auf diese Weise wird die Entwicklung nicht durch die Testvorgänge beeinträchtigt.

Fehler im Fehlererfassungssystem sammeln

Im Laufe der Testphase sammeln sich viele Fehler im Fehlererfassungssystem, von denen jedoch wahrscheinlich nicht alle so schwerwiegend sind, dass sie den Betrieb der Software stören. Sie müssen deshalb auch nicht sofort behoben werden. Somit ergibt sich eine Fehlerhierarchie. Den Systemeinträgen sollte man darum einen der folgenden Status zuweisen:

- **Behoben:** Eine Fehlerbehebung wurde erstellt und geprüft.
- **Verschoben:** Ein Fehler ist erkannt worden, er wird jedoch erst in einer späteren Projektphase behoben.
- **Feature-Wunsch:** Eine noch nicht vorhandene Funktionalität wurde als benötigt im Erfassungssystem eingestellt. Mit dem Projektverantwortlichen sollte geklärt werden, ob und wann die neue Anforderung implementiert wird und ob dafür mehr Budget oder eine längere Entwicklungszeit nötig sein werden. Man sollte beachten, dass Zeit und Budget verloren gehen, wenn die neuen Anforderungen einfach zum aktuellen Projektschritt hinzugefügt werden.
- **Abgelehnt:** Einträge, die weder Fehler noch hinzuzufügende Anforderungen sind, bekommen diesen Status und werden nicht weiter bearbeitet.
- **Neu:** Ein Fehler ist neu hinzugefügt worden.
- **Re-Test:** Eine Fehlerbehebung wurde bereits erstellt, muss aber noch auf ihren Erfolg hin überprüft werden.

Sind die Testarbeiten abgeschlossen, können die Ergebnisse dank des Fehlererfassungssystems zusammengefasst und im Idealfall auch ausgewertet werden.

Software-Tests in PHP-Anwendungen (Teil III)

Für das Testen und Analysieren der funktionalen Eigenschaften einer Softwarekomponente gibt es bereits verschiedene Testarten:

Exploratives Testen

Beim explorativen Testen versucht der Entwickler, zusammen mit dem Lernprozess, der notwendig ist, um die fachlichen Anforderungen an die Anwendung zu verstehen, erste Prüfungen der Funktionalität vorzunehmen. Dabei gibt es kein Drehbuch, nach dem vorzugehen wäre, sondern eines ergibt hier das andere, und der Tester erstellt aus den Informationen des ersten Tests einen zweiten und so weiter und so fort. Der Vorteil dieser Methode ist, dass keine lange Vorbereitungsphase nötig ist. Einarbeitung und Tests geschehen somit in einem Vorgang und es entstehen Testfälle, die sich eher an den Anforderungen eines mit der Software noch nicht vertrauten Benutzers orientieren. So werden unter anderem auch Benutzerfreundlichkeits-Fehler aufgedeckt. Ein Nachteil dieser Methode ist, dass jemand, der neu an einem Projekt arbeitet, wahrscheinlich noch nicht über den fachlichen Hintergrund verfügt, die Testreihen nicht komplett sind und Zeit für die Beantwortung von Fragen des Testers zur Verfügung stehen muss.

Test & Tune

Leider wird diese Testart allzu oft ausgeführt. Meist wird sie gegen Ende des Entwicklungsprozesses gestartet. Es wird also erst das fertige Produkt getestet und die daraus gewonnenen neuen Erkenntnisse über die Fehler in der Software werden sofort in Änderungen im Quelltext umgesetzt. Dieses Vorgehen birgt allerdings das Risiko, dass Änderungen im Quelltext wiederum zu negativen Seiteneffekten an anderen Stellen führen.

Regressionstests

Beim Regressionstest werden alle Testfälle, die während der Entwicklung stattgefunden haben, noch einmal in einer Testreihe wiederholt, um Nebenwirkungen aufzuspüren und zu vermeiden, die

durch Modifikationen an der Software während des Entwicklungsprozesses entstanden sein können. Zusätzlich zu den im Verlauf der Entwicklung erstellten Tests ist es möglich, das Fehlererfassungssystem als Quelle für neue Tests heranzuziehen. Aus jedem Fehlerreport, der Klassen und Methoden betrifft, die über Tests verfügen, wird ein neuer Testfall erstellt. So ist es leicht möglich, zu jedem späteren Zeitpunkt zu prüfen, ob ein Fehler wiederholt und an der gleichen Stelle begangen wurde.

Smoke-Tests

Der Begriff "Smoke Test" hat in der Elektrotechnik seinen Ursprung, wo nach Instandsetzungsarbeiten das Gerät wieder unter Strom gesetzt wird, um zu sehen, ob danach Rauch aufsteigt. Auf PHP-Unit-Tests übertragen sind Smoke-Tests die einfachsten vorstellbaren Tests für einzelne Funktionen, unabhängig vom Gesamtprozess. Schlägt einer dieser Teiltests fehl, ist dies ein Gegenstück zum Rauch, der aus dem elektronischen Gerät aufsteigt, wenn nach der Reparatur ein Kurzschluss vorliegt. Smoke-Tests sind schnell erstellt und geben Auskunft über Fehler in den Teilfunktionen, haben jedoch wenig Aussagekraft darüber, ob die Gesamtanforderung richtig funktioniert. Smoke-Tests können in zwei Varianten auftreten:

1. Hier geht es darum, ob vom System vom Tester provozierte Ausnahmefehler wie erwartet erkannt und angezeigt werden. Dazu werden alle Methoden und Klasseninstanzen ohne Parameter aufgerufen.

2. Dies ist der umgekehrte Fall: Der Tester gibt alle Methoden und Klasseninstanzen mit den korrekten Parameterwerten ein und prüft, ob das System diese erkennt und erwartungsgemäß reagiert.

Die Aussagekraft solcher Tests über das Gesamtprodukt mag gering sein, aber immerhin gewinnen wir durch sie Informationen über das Vorhandensein der Methode, die Ausführbarkeit ohne fatale Laufzeitfehler, die Erstellung von Warnungen und darüber, ob die grundsätzliche Funktionalität erfüllt wird. Da Smoke-Tests keinen großen Zeitaufwand bedeuten, lohnt sich ihr Einsatz unter diesen Gesichtspunkten also dennoch.

Acceptance-Tests

Für den Entwickler ist es in seinem Arbeitsalltag schwierig, sich vorzustellen, wie der Endbenutzer das Produkt verwenden wird und in welcher Form es für ihn am benutzerfreundlichsten sein wird. Deswegen und weil die Berichterstattung durch den Kunden eine ideale Quelle für Informationen über eventuell auftretende weitere Fehler ist, ist es sinnvoll, sich nach einer bestimmten Zeit mit dem Kunden zusammenzusetzen. Ein schriftlicher Report wird selten zu einem genauen und verständlichen Fehlerbericht führen, mit dem der Entwickler arbeiten kann. Am besten setzt er sich deshalb mit dem Kunden gemeinsam vor den Rechner und beobachtet diesen bei der Benutzung der Software, ohne ihm dabei Hilfestellung zu geben. Der Benutzer sollte dem Entwickler jedoch Unstimmigkeiten und sich für ihn ergebende Fragen mitteilen. Der Entwickler sollte sich dabei Uhrzeit, Programmteil und Anmerkung des Benutzers notieren, um später gezielt an der Schwachstelle arbeiten zu können.
Anmerkungen des Benutzers können beispielweise Anwenderfehler sein, die auf fehlende Benutzerfreundlichkeit zurückzuführen sind, fehlende Anforderungen, Programmfehler oder Vorschläge zur Verbesserung.

Die so gewonnen Informationen können nun in einen Fehlerreport eingespeist werden. Für den Entwickler ergeben sich daraus eventuell neue Anforderungen, die es zu bearbeiten gilt. Beim

Kunden führen diese Tests zu höherer Zufriedenheit, da auf seine Wünsche eingegangen und er individuell betreut wird. Auch der Entwickler profitiert von dieser Methode, da er auf diese Weise wichtige Informationen über Benutzerfreundlichkeit und Anwenderfehler erhält. Dieses Wissen kann bei zukünftigen Projekten berücksichtigt werden.

Ausblick

Das Erstellen von Tests kann genauso spannend und interessant sein wie die Entwicklung von Software selbst. Mit der Durchführung von Tests und der Anwendung der jeweils geeigneten Methode können Fehler im Endprodukt deutlich reduziert werden. Dadurch erhält man ein sicheres Endprodukt, kann zu Beginn des Projektes bessere Aufwands-, Zeit- und Kostenschätzungen erstellen, benutzerfreundlicher arbeiten und eine bessere Kundenberatung gewährleisten. Insbesondere bei Sprachen wie PHP, die keine starke Typisierung aufweisen, lässt sich so ein qualitativ hochwertigeres und somit zuverlässiger funktionierendes Endergebnis erzielen.

Zitat:

Prüfe die Brücke, die dich tragen soll. (altes Sprichwort)

Quellen:

- *Quality Newsletter - Juni 2004 Schwerpunktthema: Software-Testen http://www.software-quality-lab.at/swql/uploads/media/SWQL-Newsletter-200406_01.pdf*
- *Rätzmann, Manfred: Software-Testing und Internationalisierung. 2., aktual. und erw. Auflage. Galileo Press, Bonn 2004*
- *Buch: Enterprise PHP 5, Hartmann, Schotte, Serviceorientierte und webbasierte Anwendungen für den Unternehmenseinsatz.*
- *Advanced Unittesting http://www.codeproject.com/KB/cs/autp1.aspx*