Bug-Tracking und Bug-Fixing

Von Michael Kluge, Stand: November 2012

Inhaltsverzeichnis:

Abstrakt

Die Dokumentation von fehlerhaftem Verhalten einer Anwendung mit Hilfe von Bugtrackern ist in großen Softwareprojekten gängige Praxis. Doch auch in kleinen und privaten Projekten ist die Nutzung solcher Tools sinnvoll. Dieses Dokument soll einen kurzen Überblick über die Motivation, einige gängige Tools zum Dokumentieren von fehlerhaftem Code und Tipps zu ihrer Beseitigung geben.

Der Webentwickler *Savage* führt in seinem Artikel Gründe für die Nutzung von Bugtrackern in privaten Projekten auf. Demnach führe das häufige und alleinige Arbeiten häufig fälschlicherweise zu der Auffassung, man kenne alle Probleme und Schadstellen des Quellcodes; eine Dokumentation sei überflüssig, da man sich später daran erinnere. Zudem sei es eine gute Angewohnheit, sich die generelle Nutzung eines Bugtrackers anzugewöhnen. So müsse man sich nicht umstellen, wenn man an größeren Projekten arbeite.

In größeren Projekten kommen zudem weitere Vorteile zum Tragen. So kann zum einen die Kommunikation innerhalb des Teams wie auch außerhalb zwischen Entwicklern und Kunden verbessert werden. Eine einheitliche Dokumentation, komplexe Suchanfragen, hohe Verfügbarkeit, Priorisierung und Kategorisierung von Fehlern mit Hilfe von Bugtracking Systemen tragen zur erhöhten Produktivität bei.

Software

In Folge wird eine Übersicht gängiger Tools gegeben. Neben der Möglichkeit zur Dokumentation von Fehlern bieten einige des Weiteren Projektmanagement-Funktionen wie ein eigenes Wiki, Feature-Requests oder die Integration einer Versionsverwaltung an. Alle untersuchten Tools weisen eine Stable-Version auf, werden derzeit weiterentwickelt und lassen sich per Demo-Installation über den Webbrowser ausprobieren.

Bugzilla

Einer der am weitesten verbreiteten Bug-Tracker, wird von einer Vielzahl an namhaften Unternehmen wie Apache, GNOME, Facebook und NASA verwendet. Ein sehr mächtiges Tool mit hoher Sicherheit und der Möglichkeit, Bugs per Email einzutragen und zu ändern. Ermöglicht umfangreiche Suchanfragen; bietet jedoch nur wenige Projektmanagement Funktionen. Benötigt Root Access auf dem Webserver.

MantisBT

Freier Bug-Tracker mit Web-Oberfläche und vielen Projektmanagement Funktionen wie RSS-Feeds, Chat und Wiki. Ermöglicht umfangreiche Suchanfragen. Mit MantisTouch existiert ein mobiler Client für iPhone, Android und Windows Phone, dieser ist allerdings kostenpflichtig.

Trac

Timeline, Wiki, Commit-Messages, Roadmap und SVN-Repository Anzeige sind nur einige Funktionen, die Trac neben der Möglichkeit zum Tracken von Bugs auf einer intuitiven Bedienungsoberfläche unterbringt. Wird unter anderem von großen Projekten wie *Wordpress*, *Pidgin* und *jQuery*verwendet.

Redmine

Redmine wurde stark von Trac beeinflusst und bietet ein vielfältiges Berechtigungssystem zur Benutzerkontrolle sowie einen eigenen kostenlosen mobilen Client für iPhone und Android an.

Fossil

Fossil ist in C geschrieben und benutzt eine SQLite Datenbank. Dank einer einzelnen ausführbaren Datei gestaltet sich die Installation sehr einfach. Neben der Funktion zum Bug Tracking bietet es Projekt-Wikis an. Es gibt weder einen Desktop-Client, noch eine deutschsprachige Version.

WebIssues

Open-Source Tool zum Dokumentieren von Bugs mit der Möglichkeit des Exportierens nach HTML, PDF und CSV und der Versendung von Benachrichtigungen per Email. Kaum zusätzliche Projektmanagement Funktionen. Unterstützt mittlerweile den Zugriff per Webbrowser, dessen Oberfläche sich vielfältig anpassen lässt.

Jira

Sehr umfangreicher aber auch kostenpflichtiger Bug-Tracker mit vielen Projektmanagement Funktionen. Wahlweise Eigeninstallation oder Möglichkeit zur Nutzung der Server des Herstellers, sodass ein Installationsaufwand entfällt. Empfehlenswert für wichtige, kommerzielle Projekte.

	Bugzilla	**MantisBT**	**Trac**	**Redmine**	**Fossil**	**WebIssues**	**Jira**
Hersteller	Mozilla Foundation	Mantis Team	Edgewall Software	Jean-Philippe Lang	D. Richard Hipp	WebIssues Team	Atlassian
Lizenz	MPL	GPL	BSD	GNU v2	BSD	GNU v3	proprietär

Kosten	keine	keine	keine	keine	keine	keine	Gering bis moderat (gem. Anzahl Benutzer)
Source Code	frei	frei	frei	frei	frei	frei	Nicht frei
Datenbanken	MySQL, PostgreSQL und Oracle	MySQL, MS SQL oder PostgreSQL	SQLite, PostgreSQL, oder MySQL	MySQL, PostgreSQL, Oracle, SQL Server, SQLite	SQ Lite	MySQL, PostgreSQL oder SQL	kA
Entwicklungsstand	Stable, wird weiterentwickelt	Stable, wird weiterentwickelt	Stable, wird weiterentwickelt	Stable, wird weiterentwickelt	Stable, wird weiterentwickelt	Stable, wird weiterentwickelt	Stable, wird weiterentwickelt
Web Client	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Desktop Client	Ja (Windows, Linux, Mac)	Ja (Windows, Linux, Mac), Drittanbieter	Ja (Windows, Linux, Mac), Drittanbieter	Ja (Windows, Linux, evtl. Mac)	Nein	Ja (Windows, Linux, Mac)	Ja (Windows, Linux, Mac)

Mobile Client	Ja (iPhone, Android), Drittanbieter	Ja (iPhone, Android und Windows Phone), kostenpflichtig	Nein	Ja (iPhone, Android)	Nein	Nein	Ja (iPhone, Android)
Demo Umgebung	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Dokumentation / Support	Exzellent (große Community, aber auch Paid Support)	Gut	Gut	Gut	Gut	Gut	Exzellent
Benötigt Webserver	Ja	Ja	Ja	Ja	Nein	Ja	Ja
Anspruch Installation	moderat	moderat	moderat	moderat	gering	moderat	nicht vorhanden / gering
Deutschsprachig	Ja	Ja	Ja	Ja	Nein	Ja	Ja
Projekt Wiki	Ja (Addon)	Ja	Ja	Ja	Ja	Nein	Ja (*Confluence* zusätzliche Kosten)

Projektmanagement-Funktionen	wenige	umfangreich	umfangreich	umfangreich	wenige	wenige	Sehr umfangreich (teilweise m. zusätzliche Kosten)

Bug Fixing

Nachdem das Fehlverhalten der Software dokumentiert wurde, gilt es, den fehlerhaften Quellcode zu identifizieren und auszubessern. Einen groben Anhaltspunkt hierfür bietet meist bereits die Dokumentation. So sollte im Vorwege bereits akribisch aufgeschrieben werden, unter welchen Umständen das Problem aufgetreten ist. Im Folgenden werden einige Tipps und Referenze zum Ausbessern von Programmfehlern gegeben.

“Making Programs Fail”

Im Buch von *Zeller* heißt es, dass bevor mit dem eigentlichen Debuggen begonnen werden könne, das Programm zunächst in eine Zustand gebracht werden müsse, in dem es getestet werden könne – genauer gesagt müsse es mit der Absicht ausgeführt werden, den Fehler zu erzwingen (Reproduktion).

Relevanten Code identifizieren

Nun gilt es, den für den Fehler verantwortlichen und den nicht relevanten Code zu identifizieren, und das Problem dadurch einzugrenzen. *Matloff* bezeichnet diesen Prozess in seiner Abhandlung über Debugging als Vorgang der schrittweisen Validierung von Annahmen – also der Überprüfung von Dingen, die ein Programmierer für richtig annimmt, die sich gegebenenfalls aber als nicht richtig herausstellen. Ein übliches Mittel hierbei ist der Ansatz des “Teile und herrsche” oder der binären Suche: Falls man beispielsweise bemerkt, dass der Wert einer Variablen am Ende des Quellcodes den falschen Wert aufweist, überprüft man dessen Wert in der Mitte des Quellcodes erneut. Ist er bereits an dieser Stelle fehlerhaft, so springt man wiederum zur Hälfte dieses Teils und grenzt so den Ursprungsort des Fehlers schrittweise ein.

Siehe hierzu auch den Artikel von *Pruehs*.

Gut, besser, Debugger

Lässt man sich auf den Vergleich von *Kolawa* ein, der die Nutzung von Print- Debugausgaben als ein Relikt der Bronzezeit der Softwareentwicklung bezeichnet, so gehören die heutzutage verfügbaren Debugger, mit der Möglichkeit, einen Fehler automatisc und schrittweise auf seinen Ursprungsort zurückzuverfolgen, der Neuzeit der Softwareentwicklung an.

Aus Fehlern lernen; ein Fazit

Gut, wenn das Debuggen Software fehlerfrei macht. Noch besser ist es, wenn man aus Fehlern lernen kann und dadurch ein Auge für fehlerträchtigen Quellcode bekommt. So gesehen kann die konsequente Benutzung von Bugtrackern gleichwohl zum Lernprozess beitragen und die Anzahl zukünftig gemachter Fehler verringern.

Referenzen

[Savage] Why Tracking Bugs In Personal Projects Matters; December 9, 2009; Zugriff November 27, 2012 http://www.brandonsavage.net/why-tracking-bugs-in-personalprojects-matters/

[Natarajan] Top 10 Open Source Bug Tracking System; August 31, 2010; Zugriff November 28, 2012 http://www.thegeekstuff.com/2010/08/bug-tracking-system/

[Zeller] Why Programs Fail: A Guide to Systematic Debugging; July 3, 2009; Zugriff November 27, 2012 http://www.whyprogramsfail.com/

[Kolawa] The Evolution of Software Debugging; Entstehungsdatum unbekannt; Zugriff November 28, 2012 http://www.parasoft.com/jsp/products/article.jsp?articleId=490

[Matloff] Guide to Faster, Less Frustrating Debugging; April 4, 2002; Zugriff November 28, 2012 http://heather.cs.ucdavis.edu/~matloff/UnixAndC/CLanguage/Debug.html

[Pruehs] Guide to Faster, Less Frustrating Debugging; February 21, 2012; Zugriff November 28, 2012 http://www.npruehs.de/?p=12