Berücksichtigung von Softwareperformanz im Entwicklungsprozess

David Georg Reichelt

Betriebliche Informationssysteme, Institut für Informatik
Universität Leipzig
Augustusplatz 10
04109 Leipzig
davidgeorg_reichelt@dagere.de

Abstract: Software-Performance-Engineering (SPE) bezeichnet verschiedene Ansätze zur Performanzverbesserung von Software. Diese wurden bisher nicht vergleichend untersucht. Der vorliegende Beitrag beschreibt ein Vorgehen, um zu überprüfen, inwiefern die SPE-Ansätze die Entstehung performanter Software sicherstellen.
Trotz des Vorhandenseins der SPE-Ansätze lassen sich bspw. beim Applikationsserver Apache Tomcat temporär im Entwicklungsversionen auftretende Performanzprobleme nachweisen. Dies deutet darauf hin, dass SPE-Techniken nicht bzw. nicht erfolgreich eingesetzt werden. Die semiautomatische Generierung von Performanztests könnte dieses Problem lösen, da mit dieser Performanzprobleme mit geringem Zusatzaufwand erkennbar wären. Dieser Beitrag schildert deshalb ein Vorgehen zur semiautomatischen Performanztestgenerierung.

1 Motivation

Softwareperformanz misst, wie effizient ein Softwaresystem in Bezug auf Zeit- und Ressourcenallokation ist [CMI11]. Indikatoren der Performanz sind u.a. Antwortzeit, Arbeitsspeicherverbrauch und CPU-Auslastung. Software Performance Engineering (SPE) bezeichnet Techniken, deren Ziel die Verbesserung der Performanz ist [WFP07].[1]

Wenngleich die Techniken des SPE in Praxisprojekten evaluiert wurden, erfolgte bisher keine Gesamt-Evaluierung an einer repräsentativen Menge von Projekten. Ein Vorgehen zur vergleichenden Evaluierung der SPE-Techniken wird hier vorgestellt. Eine erste Evaluierung anhand von Tomcat [RB14] zeigt, dass Performanzprobleme bei der Entwicklung von Tomcat 7 auftraten. Diese traten bei Aufruf von Seiten durch einen Nutzer auf. Da eine gute Einzelnutzer-Performanz Bedingung für eine gute Gesamt-Performanz ist [Pod08], soll der Einzelnutzer-Performanztest im Mittelpunkt stehen. Der Aufwand für derzeitig existierende SPE-Techniken ist sehr hoch [WFP07]; um diesen Aufwand zu reduzieren,

[1] Die Ursprungsdefinition von SPE umfasst lediglich das Refactoring der Architektur basierend auf Performanzvorhersagen, die auf Basis der Architektur abgeleitet wurden. Hier wird von Woodsides umfassenderen Definition ausgegangen.

sollen die Performanztests automatisch generiert werden. Daher soll hier ein Ansatz vorgestellt werden, um Einzelnutzer-Performanztests automatisiert zu generieren.

Dieser Beitrag gibt eingangs einen Überblick über SPE-Techniken. Anschließend werden Forschungsfragen erörtert. Darauf aufbauend wird auf das Forschungsdesign eingegangen.

2 Techniken des Software Performance Engineering

Nach [WFP07] unterteilen sich die Methoden des SPE grundlegend in zwei Kategorien: modellbasierte und messbasierte Ansätze. Bei modellbasierten Ansätzen werden Performanzinformationen an Architekturmodelle bzw. Ausführungsszenarien annotiert. Anschließend wird die Gesamtperformanz der Anwendung, bspw. durch Übertragung der Performanzinformationen in Warteschlangenmodelle und Analyse dieser, vorhergesagt. Sollte die dabei vorhergesagte Gesamtperformanz den Performanzanforderungen nicht genügen, werden Refactorings an der Architektur durchgeführt, bis eine angemessene Gesamtperformanz erreicht ist [WFP07]. Daneben existieren messbasierte Ansätze. Diese behandeln zwei Bereiche: Die eigentliche Messung und das Testen der Messwerte. Zum Messen gehört Monitoring, d.h. die Überwachung der Performanz im Einsatz befindlicher Softwaresysteme. Es ist möglich, aus den gemessenen Monitoringdaten die Architektur zu rekonstruieren und die Daten in Relation zu dieser bereitzustellen [vHWH12].

In der Praxis etabliert ist das Testen der Softwareperformanz durch Lasttests, das durch verschiedene Methoden und Werkzeuge unterstützt wird [Mol09]. Problematisch ist hier der große Aufwand für das Definieren, Ausführen und Auswerten der Tests. Aktuelle Forschung versucht den Lasttestprozess zu verbessern, bspw. indem aus Logs automatisiert potentielle Performanzprobleme erkannt werden [Jia10]. Weiterhin existieren verschiedene Arbeiten zur automatisierten Generierung von Lasttests (bspw. [SGH+14]).

In [Pod08] wird das kontinuierliche Testen der Performanz für einen Nutzer ohne zusätzliche Last gefordert. Grund hierfür ist, dass die Implementierungsqualität einzelner Abschnitte die Gesamtperformanz negativ beeinflussen kann. Ein Teil der Performanzprobleme können also durch Einzelnutzer-Performanztests erkannt werden. Probleme, die nur durch Mehrbenutzerbetrieb entstehen, können damit hingegen nicht gefunden werden. Forschung zur automatisierten Einzelnutzer-Performanztest-Generierung existiert bislang nicht.

3 Forschungsfragen

Das vorliegende Promotionsvorhaben wird empirisch die Behandlung von Softwareperformanz in Entwicklungsprojekten betrachten. Darauf aufbauend wird eine Möglichkeit untersucht, die Einhaltung von Performanzgrenzwerten besser sicherzustellen. Dieses Kapitel stellt Forschungsfragen zu diesen beiden Forschungszielen vor.

Eine zentrale Fragestellung ist es, inwiefern die beschriebenen Performanzverbesserungstechniken Performanzprobleme verhindern können. Dabei ist nach Zeitpunkt des Auf-

tretens der Performanzprobleme zu differenzieren: Beispielsweise werden Implementierungsfehler kurz vor einem Release durch anfängliches Architekturrefactoring nicht verhindert werden können. Untersucht werden muss also, zu welchem Zeitpunkt in Relation zum Release welche Arten von Performanzproblemen auftreten. Diese Erkenntnis liefert den zentralen Zugang zur Fragestellung, welche Performanzprobleme durch welche Techniken erkannt bzw. gelöst werden können. Es ergibt sich damit folgende Forschungsfrage:

1. Welche Performanzverbesserungstechniken können das Auftreten welcher Performanzprobleme an welcher Stelle im Softwareentwicklungsprozess verhindern?

Als Vorarbeit wurde gezeigt, dass sich auch bei großen Projekten Softwareperformanz ggf. stark verschlechtert [RB14]. Basierend auf der Beantwortung der ersten Forschungsfrage soll deshalb eine Methode entwickelt werden, die die Überprüfung der Erfüllung von Performanzanforderungen während des gesamten Softwareentwicklungsprozesses ermöglicht. Auch einzelne Programmeinheiten sollen analog zum Vorgehen bei funktionaler Korrektheit regelmäßig überprüft werden. Die dabei entstehenden Performanztests sind Performanz-Unit-Tests. Aufgrund der bereits in Kapitel 2 genannten Argumente für Einzelnutzer-Performanztests soll der Fokus auf diesen liegen.

Da die Implementierung umfassender Einzelnutzer-Performanz-Unit-Tests einen hohen Zusatzaufwand darstellen, werden Sie in der Praxis selten umgesetzt werden. Deshalb wird sich dieses Promotionsvorhaben mit der semiautomatischen Generierung der Tests beschäftigen. Zur Umsetzung der Testgenerierung existieren zwei Möglichkeiten: Die Generierung von Tests aus Anforderungen und die Generierung von Tests aus Monitoringdaten. Es ergibt sich somit folgende Forschungsfrage:

2. Welche Methoden eignen sich, um (semi-)automatisch Performanztests zu generieren?

4 Vorgehen

Anknüpfend an die beiden Forschungsfragen wird in diesem Kapitel auf die empirische Untersuchung des Softwareentwicklungsprozesses und anschließend auf die Möglichkeiten zur Performanztestgenerierung eingegangen.

4.1 Empirischen Untersuchung der Performanz im Softwareentwicklungsprozess

Um zu überprüfen, wann Performanzprobleme im Softwareentwicklungsprozess auftreten, wird die Performanzentwicklung großer Softwareentwicklungsprojekte untersucht werden. Hierbei können frei verfügbare Projekte als Grundlage genutzt werden. Zur Auswahl der Projekte sind Kriterien für die Mindestgröße und die Mindestanzahl an Revisionen zu definieren. Die empirische Untersuchung gliedert sich in drei Schritte: Ermittlung der Performanzverläufe, Ermittlung von Performanzproblemen und Untersuchung des Auftretens

der Performanzprobleme in Relation zum Einsatz der SPE Techniken.

Im ersten Schritt wird eine Anwendung implementiert, die aus vorhandenen Unit-Tests KoPeMe-Performanztests[2] generiert. Diese wird auf jede Revision eines Projektes angewandt, um für diese passende Performanztests zu generieren. Die Tests werden anschließend in der jeweiligen Revision ausgeführt. Im zweiten Schritt werden die sich daraus ergebenden Performanztestverläufe ausgewertet. Hierfür wird ein Auswertungswerkzeug implementiert werden, das signifikante Performanzveränderungen erkennt. Kriterien für die Signifikanz, wie bspw. ein betragsmäßig besonders großer Anstieg eines Performanzmesswertes über die Revisionen, sind noch zu definieren. Aufbauend auf den Messungen sollen Performanzprobleme erkannt werden. Hierzu sind Kriterien notwendig, um Veränderungen von Problemen zu unterscheiden. Dies ist notwendig, da bspw. neue Funktionalitäten starke Performanzveränderungen auslösen können, die keine Performanzprobleme sind. Im letzten Schritt wird durch die Auswertung der erkannten Performanzprobleme ermittelt, welche SPE-Technik zu welchem Zeitpunkt die Entstehung von Performanzproblemen unterbinden kann. Hierfür ist zu prüfen, welche SPE-Techniken angewandt wurden. Denkbar ist es, Teile der Überprüfung zu automatisieren, bspw. indem nach JMeter-Lastskripten[3] oder Palladio-Komponentenmodellen[4] im Software-Repositorium des Projekts gesucht wird. Indem geprüft wird, welche zeitliche Verteilung von Performanzproblemen mit welchem Einsatz von SPE-Techniken korreliert, kann die Wirkung der jeweiligen Technik beurteilt werden.

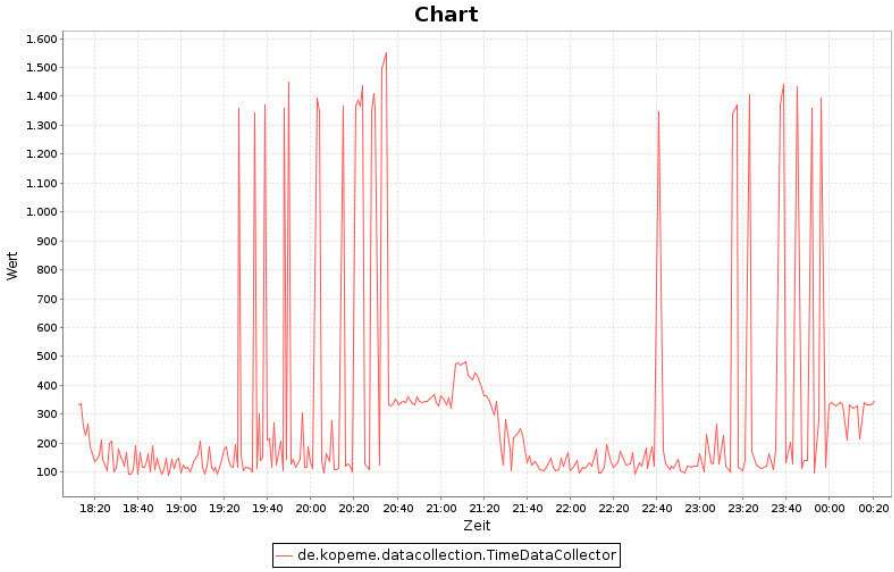

Abbildung 1: Downloadzeit einer Servletseite in verschiedenen Tomcat-Revisionen, aus: [RB14]

[2]Hierbei handelt es sich um Unit-Tests, die Performanzwerte ermitteln [RB14].
[3]JMeter ist ein Werkzeug für Lasttests, Offizielle Webseite: https://jmeter.apache.org/
[4]Palladio ist ein Werkzeug für modellbasiertes SPE, Offizielle Webseite: http://www.palladio-simulator.com

Eine Untersuchung der Performanz der Revisionen von Tomcat 6 hat im Rahmen von [RB14] stattgefunden. Abbildung 1 zeigt, wie sich die Performanz des Anwendungsfalls, eine Servlet-Seite von einem lokalen Server zu laden, verhält. Die Abszisse zeigt die Zeit der Ausführung, die jeweils einzelnen Revisionen zugeordnet werden kann, und die Ordinate zeigt die Downloadzeit der Servlet-Seite vom lokalen Server, der aus der Revision kompiliert wurde. Das starke Ansteigen der Antwortzeit legt nahe, dass Performanzprobleme mit aktuellen Techniken nicht effizient genug bearbeitet werden. Bei Tomcat wird lediglich ein Profiler zu Finden von Performanzproblemen eingesetzt.

4.2 Unterstützung des Entwicklungsprozesses

Es gibt zwei Varianten, Performanztests (semi-)automatisiert zu generieren: Unter Nutzung der Anforderungen oder der Monitoringdaten. Da Performanztests die Sicherstellung der Einhaltung von Performanzanforderungen ermöglichen sollen, können sie direkt aus den Anforderungen abgeleitet werden. Hierbei können analog zum Software Performance Engineering Performanzannotationen an UML-Modelle als Grundlage dienen. Um die dort angegebenen Grenzwerte zu überprüfen, ist es notwendig, Initialisierungsaufrufe und Parameter der zu prüfenden Anforderung manuell hinzuzufügen.

Es ist davon auszugehen, dass bei den Ausführungen von Tests die gemessenen Performanzwerte annähernd den Anforderungen entsprechend. Es ist deshalb möglich, diese als Test-Grenzwerte zu verwenden. Gegebenenfalls sind die gemessenen Performanzwerte manuell zu verändern, damit diese den Performanzzielen entsprechen.

Wird der Quelltext vollständig instrumentiert, wie es beispielsweise das Framework Kieker [vHWH12] ermöglicht, ist es denkbar, den Entwickler den Methodenaufruf aus einem Aufrufbaum auswählen zu lassen, für die ein Test erstellt werden soll. In diesem Fall sind die vorher notwendigen Initialisierungsaufrufe und die Parameter bekannt. Es ist für den Entwickler lediglich notwendig, ggf. Grenzwerte zu ändern. Eine Kombination mit der Grenzwertextraktion aus Performanzmodellen könnte es auch erlauben, Tests bis auf die Auswahl des zu testenden Aufrufs vollautomatisch durchzuführen. Ein vollautomatisches Testen aller Methoden ist nicht sinnvoll, da Performanztests sehr zeitintensiv sind.

Anschließend ist die Praxistauglichkeit des Ansatzes zu untersuchen. Dies ist möglich, indem äquivalent zum in Kapitel 4.1 dargestellten Vorgehen geprüft wird, ob bei semiautomatischer Performanztestgenerierung bei vorhandenen Projekten Performanzprobleme gefunden hätten.

Zusätztlich wird eine Evaluation an in Entwicklung befindlichen Projekten angestrebt. Hierbei wird ermittelt, wie viele Performanzprobleme diese hatten. Es wird angenommen, dass so viele Projekte analysiert werden, das Messwerte nicht durch Ausreißer verzerrt werden. Unter dieser Annahme ist der Quotient Performanzprobleme/Quelltextzeilen ein Indikator für den Einsatz guter Performanzverbesserungstechniken. Durch den Vergleich des Quotienten Performanzprobleme/Quelltextzeilen mit anderen Projekten unter Berücksichtigung der dort verwendeten Performanzverbesserungstechniken ist es möglich, die Wirkung der Performanztestgenerierung auf die Softwareperformanz zu beurteilen.

5 Zusammenfassung

Es wurde ein Promotionsvorhaben geschildert, dessen Ziel die Untersuchung der Wirkung aktueller SPE-Techniken und eines neuen Verfahrens zur semiautomatischen Performanztestgenerierung auf die Performanz der entstehenden Softwareprodukte ist.

Die Untersuchung der Wirkung aktueller Performanzverbesserungsverfahren soll durch eine Überprüfung der Performanz von frei verfügbaren Projekten erfolgen. Dabei sollen Performanzwerte einzelner Unit-Tests in Relation zum Einsatz von Performanzverbesserungsverfahren gesetzt werde. Als neues Performanzverbesserungsverfahren soll das semiautomatische Generieren von Einzelnutzer-Performanz-Unit-Tests untersucht werden. Die Generierung soll auf Basis von spezifizierten Anforderungen in UML-Diagrammen und auf Basis von Monitoringdaten eines Anwendungslaufs, bei dem Performanzanforderungen erfüllt wurden, erfolgen. Anschließend soll an frei verfügbaren Projekten geprüft werden, ob dieses Vorgehen Tests generieren kann, die Performanzprobleme erkannt hätten.

Literatur

[CMI11] Vittorio Cortellessa, Antinisca Di Marco und Paola Inverardi. *Model-Based Software Performance Analysis*. Springer, 2011.

[Jia10] Zhen Ming Jiang. Automated analysis of load testing results. In *ISSTA*, Seiten 143–146, 2010.

[Mol09] Ian Molyneaux. *The Art of Application Performance Testing: Help for Programmers and Quality Assurance*. O'Reilly Media, Inc., 1st. Auflage, 2009.

[Pod08] A. Podelko. Agile Performance Testing. In *Int. CMG Conference*, Seiten 267–278, 2008.

[RB14] David Georg Reichelt und Lars Braubach. Sicherstellung von Performanzeigenschaften durch kontinuierliche Performanztests mit dem KoPeMe Framework. In Wilhelm Hasselbring und Nils Christian Ehmke, Hrsg., *Software Engineering 2014: Fachtagung des GI-Fachbereichs Softwaretechnik*, Lecture Notes in Informatics, Seiten 119–124. Köllen Druck+Verlag GmbH, 2014.

[SGH+14] Eike Schulz, Wolfgang Goerigk, Wilhelm Hasselbring, Andre van Hoorn und Holger Knoche. Model-Driven Load and Performance Test Engineering in DynaMod. In *Proceedings of the Workshop on Model-based and Model-driven Software Modernization (MMSM '14)*, Seiten 10–11, März 2014.

[vHWH12] André van Hoorn, Jan Waller und Wilhelm Hasselbring. Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis. In *Proceedings of the 3rd joint ACM/SPEC International Conference on Performance Engineering (ICPE 2012)*, Seiten 247–248. ACM, April 2012.

[WFP07] Murray Woodside, Greg Franks und Dorina C. Petriu. The Future of Software Performance Engineering. In *2007 Future of Software Engineering*, FOSE '07, Seiten 171–187, Washington, DC, USA, 2007. IEEE Computer Society.