Empirische Bewertung von Performanz-Vorhersageverfahren für Software-Architekturen

Viktoria Firus, Heiko Koziolek, Steffen Becker,
Ralf Reussner und Wilhelm Hasselbring

Abteilung Software Engineering, Universität Oldenburg
OFFIS, Escherweg 2, D-26121 Oldenburg
{firus,koziolek,becker,reussner,hasselbring}@informatik.uni-oldenburg.de

Abstract: Die Architektur eines Software-Systems beeinflusst maßgeblich seine Qualitätseigenschaften wie Performanz oder Zuverlässigkeit. Daher sind Architekturänderungen oft die einzige Möglichkeit, Mängel bei diesen Qualitätseigenschaften zu beheben. Je später diese Änderungen an der Architektur während des Software-Entwicklungsprozesses vorgenommen werden, desto teurer und riskanter sind sie. Aus diesem Grund ist eine frühzeitige Analyse verschiedener Architektur-Entwurfsalternativen bezüglich ihrer Auswirkungen auf Qualitätseigenschaften vorteilhaft. Dieser Artikel beschreibt die Evaluation dreier verschiedener Performanz-Vorhersageverfahren für Software-Architekturen hinsichtlich ihrer Eignung, korrekte Empfehlungen für frühzeitige Entwurfsentscheidungen zu geben. Zusätzlich sollen diese Vorhersageverfahren prüfen, ob extern vorgegebene Performanz-Anforderungen realisierbar sind. Die Performanz-Vorhersageverfahren „SPE“, „Capacity Planning“ und „umlPSI“ wurden empirisch durch 31 Teilnehmer untersucht, die eine Menge vorgegebener Alternativen beim Entwurf der Architektur eines Webservers zu bewerten hatten. Die Ergebnisse zeigen, dass Entwurfsalternativen mit allen Verfahren richtig bewertet wurden, sofern deutliche Auswirkungen auf die Performanz vorhanden waren. Ohne den Einsatz der Performanz-Vorhersageverfahren wurden häufiger weniger performante Entwurfsalternativen vorgeschlagen. Darüber hinaus konnte das Verfahren Capacity Planning die absoluten Werte bei den meisten Entwurfsalternativen relativ genau vorhersagen.

1 Einleitung und Motivation

Eine der Motivationen zur Beschreibung von Software-Architekturen ist die explizite Behandlung von Qualitätseigenschaften [SG96]. Dies ist begründet in dem Einfluss, den die Architektur eines Software-Systems auf seine Qualitätseigenschaften hat. Heute gängige Software-Entwicklungsverfahren sind getrieben durch die Umsetzung funktionaler Anforderungen in Entwürfe und Implementierungen, wohingegen die Einhaltung nichtfunktionaler Eigenschaften erst in späteren Phasen des Entwicklungsprozesses beachtet wird. Werden in diesen späten Phasen gravierende Qualitätsprobleme (oft im Zusammenhang mit Skalierbarkeit) festgestellt, reichen meist vereinzelte lokale Code-Optimierungen nicht aus. Stattdessen muss dann die Architektur grundlegend überarbeitet werden, um ur-

sprüngliche Qualitätsvorgaben realisieren zu können. Angesichts der Kosten, Risiken und des Zeitbedarfs der späten Architekturänderungen, wird die Problematik eines solchen Vorgehens schnell offenbar.

Aufgrund dieser Herausforderungen beim Architekturentwurf ist das Gebiet der Architekturanalyse ein Feld aktiver Forschung (ein aktueller Überblick findet sich in [BDIS04]). Unter der Qualitätseigenschaft „Performanz“ verstehen wir alle zeitlichen Maße der Effizienz, insbesondere die Maße „Antwortzeit“, „Reaktionszeit“ und „Durchsatz“. Aufgrund der Wichtigkeit der Performanz für eine Vielzahl von Systemen betrachten wir im folgenden Architektur-Analyse-Verfahren für Performanz.

Wir beschäftigen uns mit der Frage, ob Performanz-Vorhersageverfahren die im Vergleich zur gemessenen Performanz der Implementierung richtige Entwurfsalternativen empfehlen. Bisher wurde diese Fragestellung noch nicht empirisch untersucht. Ein erster Ansatz ist die Feldstudie [BMDI04], die ein auf stochastischen Prozessalgebren basierendes Verfahren mit einem Simulationsverfahren vergleicht. Allerdings wurden die Vorhersagen nicht mit gemessenen Werten verglichen. Auch ist der Einfluss der durchführenden Person auf die Qualität der Vorhersage unklar, da nicht mehrere Personen ein Verfahren anwendeten. Gorton et. al. [GL03] vergleichen Vorhersagen und Messungen von verschiedenen Software-Architekturen auf Basis von Enterprise JavaBeans. Dabei werden aber keine verschiedenen Vorhersageverfahren verglichen.

Die Methodik des experimentellen Software Engineering wird erläutert in [Pre01, JM01, WRH$^+$00].

Für unsere Studie wurden die Vorhersageverfahren „Software Performance Engineering (SPE)“ [Smi02], „Capacity Planning (CP)“ [MAD04] und „umlPSI“ [Mar04] ausgewählt, zum einen da sie als prototypische Vertreter ihrer Klasse von Vorhersageverfahren angesehen werden können, also analytisch-schätzungenbasiert bei SPE, analytisch-messungenbasiert im Falle von CP und simulationsbasiert bei umlPSI. Zum anderen wurden diese Verfahren ausgewählt wegen ihrer Werkzeugunterstützung und Integration in einen Software-Entwicklungsprozess. Von 31 Versuchsteilnehmern werden vorgegebene Entwurfsalternativen für die Architektur eines Beispielsystems (eines Webservers) mit den drei verschiedenen Verfahren bewertet und diese Bewertung mit der gemessenen Performanz der Implementierungen der Entwurfsalternativen verglichen. Eine Kontrollgruppe löste die Aufgabenstellung intuitiv (d.h. ohne Anwendung eines Verfahrens). In dieser Studie wurden die Metriken zur Auswertung der Daten gemäß der *Goal, Question, Metric-Methode* [BCR94] (GQM) nach Basili und Rombach definiert. Das Ziel der hier vorgestellten Fallstudie bestand in der empirischen Bewertung der Anwendbarkeit von Performanz-Vorhersageverfahren für Software-Architekturen aus der Sicht des Entwicklers. Dabei wird unter der Anwendbarkeit zum einen eine nachvollziehbare Durchführbarkeit des Verfahrens und zum anderen die Ermittlung hilfreicher Ergebnisse verstanden.

Die Beantwortung der nachfolgend formulierten Fragestellungen trägt zur Erreichung des aufgestellten Ziels bei. Weitere Fragestellungen wurden in [Koz04] formuliert und beantwortet, deren Betrachtung liegt jedoch außerhalb des Fokus dieses Artikels.

1. Wie gut lässt sich mit den Verfahren die Realisierbarkeit quantitativer Performanz-Anforderungen feststellen?

2. In wie weit unterstützen die Verfahren die Auswahl der richtigen Entwurfsalternative?

Die zur Bestimmung der Antworten auf die Fragen notwendigen Metriken werden in Abschnitt 4 zusammen mit den jeweiligen Daten vorgestellt.

Der Beitrag dieses Artikels besteht primär in dem Vergleich und der Bewertung der o.a. Performanz-Vorhersageverfahren für Software-Architekturen hinsichtlich der korrekten Unterstützung von Entwurfsentscheidungen und Bewertung von Performanz-Anforderungen. Sekundär besteht der Beitrag in der Beschreibung einer Methode zur Evaluation von Performanz-Vorhersageverfahren, die auch auf andere Vorhersageverfahren für Performanz anwendbar ist.

Dieser Artikel ist wie folgt aufgebaut. In Abschnitt 2 werden die Performanz-Vorhersageverfahren vorgestellt. Abschnitt 3 beschreibt die Durchführung der empirischen Untersuchung, das Beispielsystem und die zu bewertenden Entwurfsalternativen. In Abschnitt 4 werden die Ergebnisse dargestellt. Abschnitt 5 diskutiert die Grenzen der Gültigkeit der Ergebnisse. In Abschnitt 6 wird der Artikel zusammengefasst und Folgefragestellungen werden diskutiert.

2 Vorstellung der untersuchten Verfahren

Die Performanz-Vorhersageverfahren unterscheiden sich im Wesentlichen in den Eingabedaten (die Architektursichten, bzw. die für die Analyse benötigten Performanz-Attribute), den Performanz-Modellen und den Analysemethoden.

SPE-Verfahren Beim SPE-Prozess [Smi02] werden für jedes Performanz-kritische Szenario in einer Architektur Software- und Systemausführungsmodelle erstellt. Das *Softwareausführungsmodell* in Form eines Ausführungsgraphen enthält Informationen über den Software-Ressourcenbedarf jeder einzelnen Aktion eines Szenarios, deren Ausführungswahrscheinlichkeiten und die Ausführungshäufigkeiten von Schleifen. Damit können 'best-/worst-case' Antwortzeiten für ein Szenario berechnet werden. Das *Systemausführungsmodell* in Form eines Warteschlangenmodells enthält Informationen über Art und Anzahl von Hardware-Ressourcen des Systems und deren Verbindungen untereinander. Die Eingabeparameter für dieses Warteschlangenmodell werden aus dem Softwareausführungsmodell bezogen, zusätzlich muss die Ankunftsrate der Anfragen bzw. die Anzahl der Benutzer im System angegeben werden. Durch Analyse der Warteschlangenmodelle können die Auslastungen der Ressourcen und mittlere Antwortzeiten des Systems berechnet werden.

Die Modelle werden aus einer erweiterten Form von UML-Sequenzdiagrammen und Verteilungsdiagrammen erstellt. Dabei wird der Entwickler durch das Werkzeug SPE-ED unterstützt, das auch die automatische Auswertung der Modelle ermöglicht sowie eine graphische Visualisierung der Ergebnisse bietet.

umlPSI-Verfahren Das umlPSI-Verfahren [BM03, BMDI04] beruht auf ausführbaren Simulationen. Das Verfahren lässt sich aufgrund der Nutzung von UML-Diagrammen und der Verfügbarkeit eines Werkzeugs zur automatischen Erstellung der Simulationen leicht in den Software-Entwicklungsprozess integrieren.

Das Verfahren geht aus von Anwendungsfall-, Verteilungs- und Aktivitätsdiagrammen der UML, mit denen die zu untersuchende Architektur modelliert wird. Alle Diagramme werden mit Hilfe des *UML Profile for Schedulability, Performance, and Time* [OMG03] mit Performanz-Angaben versehen. Verschiedene Simulationsparameter wie die maximale Dauer oder die gewünschte Genauigkeit der Ergebnisse können eingestellt werden. Die Ausführung der Simulation mit dem Werkzeug liefert für alle Anwendungsfälle die mittleren Antwortzeiten und für alle Ressourcen Durchsätze und mittlere prozentuale Auslastungen. Die Ergebnisse werden in das UML-Modellierungswerkzeug übertragen und sind dort einsehbar.

Capacity-Planning-Verfahren Das Capacity-Planning-Verfahren [MAD04] wird zur Modellierung eines bestehenden Systems verwendet. Das Ziel ist die Vorhersage künftiger Performanz-Eigenschaften bei einem sich ändernden Nutzungsprofil.

Ausgangspunkt für die Performanz-Modellierung ist eine meist textuelle Beschreibung der Systemarchitektur zusammen mit einem repräsentativen Nutzungsszenario, das die Typen und Anzahl der Eingaben an das System charakterisiert. Durch ein Monitoring des System werden möglichst viele Performanz-Daten des Systems erfasst. Diese Daten dienen als Eingabeparameter für ein Warteschlangenmodell, das auf Basis der Systembeschreibung erstellt wird. Mit diesem Warteschlangenmodell und Vorhersagen für das künftig zu erwartende Nutzungsprofil des Systems kann die zukünftige System-Performanz analysiert werden. Es kann festgestellt werden, bei welcher Systemlast bestimmte Ressourcen überlastet werden. Für verschiedene Entwurfsalternativen und Systemarchitekturen können die erwarteten Antwortzeiten und Ressourcenauslastungen vorhergesagt werden.

3 Methodik und Durchführung

Bei der Wahl einer Forschungsmethode wurde zunächst die Durchführung eines kontrollierten Experiments [Pre01] favorisiert. Die Kontrolle in einem solchen Experiment ergibt sich daraus, dass bis auf die eingesetzten Verfahren alle anderen Faktoren, die das Ergebnis beeinflussen könnten, konstant gehalten werden. Aufgrund der geringen Teilnehmerzahl konnte in dieser Untersuchung jedoch nicht die Qualifikation und Leistungsfähigkeit der Versuchsteilnehmer durch statistische Methoden kontrolliert werden. Es war nicht möglich, einen Hypothesentest durchzuführen. Das folgende Experiment hat den Charakter einer vergleichenden Fallstudie. Diese Fallstudie wurde parallel von 8 Teilnehmern pro Gruppe durchgeführt, um den Einfluß der individuellen Fähigkeiten auf die Ergebnisse zu kontrollieren.

Beteiligt an der Studie waren insgesamt 31 Informatik-Studenten mit abgeschlossenem Vordiplom, die alle erfolgreich ein Software-Praktikum absolviert hatten, jedoch noch unerfahren mit Performanz-Analysen waren.

Die Durchführung der Fallstudie erfolgte in drei Schritten. 24 Teilnehmer wurden zunächst in den drei genannten Verfahren geschult, während 7 weitere eine ungeschulte Kontrollgruppe bildeten. Von allen Teilnehmern wurden im zweiten Schritt jeweils fünf Entwurfsvorschläge für einen experimentellen Webserver untersucht. Dabei lag ihnen kein Pro-

grammcode vor, mit dem Messungen hätten angestellt werden können. Jeder Teilnehmer gab entweder auf Basis der Verfahren oder (im Fall der Kontrollgruppe) intuitiv eine Empfehlung ab, von welcher Variante des Webservers die beste Performanz zu erwarten sei. Im dritten Schritt wurde der Webserver in den fünf vorgeschlagenen Versionen implementiert und die Performanz gemessen. Die Vorhersagen der Studenten konnten nun mit den tatsächlichen Werten verglichen werden.

Schulung und Vortest Die drei Performanz-Vorhersageverfahren wurden den Teilnehmern jeweils während zweistündiger Trainingssitzungen vorgestellt. Zur aktiven Auseinandersetzung mit den Verfahren und Einarbeitung in die entsprechenden Werkzeuge wurde zusätzlich am Ende jeder Sitzung ein Übungsblatt ausgegeben, das die Studenten innerhalb einer Woche bearbeiteten.

Die Schulung konnte so auch als Vortest für die spätere Untersuchung des Webservers genutzt werden, um zu prüfen, dass die Aufgabenstellung verständlich und lösbar ist. Durch die Auswertung der Übungslösungen konnten Rückschlüsse auf die Leistungsfähigkeit der Studenten gezogen werden. Die Teilnehmer wurden daraufhin für das Experiment in drei Gruppen vergleichbarer Leistung eingeteilt, so dass jeweils acht Personen einzeln mit genau einem Verfahren die Architektur des Webservers untersuchten.

Experiment Das Experiment fand in einem Rechnerraum unter kontrollierten Bedingungen statt, so dass sich die Teilnehmer nicht untereinander beeinflussen konnten. Jedem Studenten standen zwei Stunden Zeit zur Verfügung.

Untersuchungsobjekt war ein experimenteller Webserver, der in unserer Gruppe zu Testzwecken in der Programmiersprache C# entwickelt wurde. Der Server bietet die üblichen Funktionen zur Beantwortung von Anfragen nach dem HTTP-Protokoll. Er ist multithreaded und kann über die Anbindung einer Datenbank auch dynamisch HTML-Dateien erzeugen. Die genaue Beschreibung des Webservers findet sich in [Koz04].

Folgende Entwurfsalternativen zur Optimierung der Performanz wurden für das Experiment zur Evaluation vorgesschlagen:
1a) Statischer Cache: für die Auslieferung statischer Inhalte wird ein Cache eingebaut. Auf diese Weise können teure Festplatten-Zugriffe eingespart und die Dateien aus dem Hauptspeicher bezogen werden. Die zu erwartende Wahrscheinlichkeit eines Cache-Hits wurde auf 70% festgelegt.
1b) Dynamischer Cache: für die Auslieferung dynamischer Inhalte, die sich nicht bei jeder Anfrage ändern (z.B. aktuelle Lagerbestände, tägliche Wetterdaten), wird ein Cache eingebaut. Dabei können die Zugriffe auf einen externen Datenbankserver eingespart werden, jedoch liegt die zu erwartende Wahrscheinlichkeit eines Cache-Hits hier nur bei 20%.
2) Single-threaded Server: Der Server wird in einer single-threaded Variante implementiert. Das bedeutet, dass nicht für jede Anfrage ein neuer Programmthread abgespalten wird, sondern die Anfragen werden sequentiell hintereinander bearbeitet werden. Bei einer niedrigen Auslastung des Servers könnte es so möglich sein, Zeit für den Start und Kontextwechsel von Threads einzusparen.
3) Komprimierung: Inhalte werden vor dem Versand auf dem Server komprimiert. Dies muss aufgrund der möglicherweise dynamischen Inhalte bei jeder Anfrage erfolgen und

verbraucht eine gewisse Rechenzeit für den Kompressionsalgorithmus. Jedoch könnte bei langsamen Internetverbindungen durch die reduzierte Datenmenge möglicherweise viel Zeit für das Versenden eingespart werden.
4) Clustering: Durch die Verteilung des Webservers auf zwei unabhängige Server kann die maximale Anzahl der beantwortbaren Anfragen erhöht werden. Jedoch ist zu prüfen, ob bei einer niedrigen Beanspruchung des Servers ein Performanz-Gewinn nicht durch den erhöhten Verwaltungsaufwand kompensiert wird.
Für diese Alternativen sollten die Teilnehmer folgendes Nutzungsszenario des Webservers untersuchen: Die Ankunftsrate von Anfragen betrug eine Anfrage pro Sekunde. Es wurden zu 40% statische und zu 60% dynamische Inhalte abgefragt und die mittlere Größe der abgefragten Dateien betrug 5 KByte. Clients griffen über eine einfache ISDN-Verbindung (64 KBit/s) auf den Server zu. Der Server wiederum war über eine LAN-Verbindung (10 MBit/s) mit einem Datenbankserver verbunden, aus dem Inhalte für die dynamischen Anfragen bezogen werden konnten. Dabei wurde die mittlere Zeit für eine Datenbankabfrage mit 0,5 Sekunden vorgegeben.

Den Teilnehmern wurden ergänzend zu diesen Angaben UML-Diagramme und Warteschlangenmodelle der Architektur vorgelegt. Die acht Studenten, die das Capacity Planning anwandten, erhielten zusätzlich eine künstlich erzeugte Logdatei, in der die Verbrauchszeiten von CPU, Festplatte und LAN für 1000 Anfragen an den Server vermerkt waren. Diese Werte wurden mittels Messungen am Prototypen des Webservers vor dem Experiment bestimmt. Die Vorgabe war notwendig, da die Teilnehmer bei diesem Verfahren diese Daten für ihre Berechnungen benötigten. Mittels der Verfahren berechnete jeder Teilnehmer für alle Varianten jeweils die Durchlaufzeiten für statische bzw. dynamische Anfragen.

Alle im Experiment untersuchten Entwurfsalternativen wurden implementiert und vermessen. Details über den technischen Aufbau des Meßverfahrens und der Ablaufumgebung sind in [Koz04] dokumentiert.

4 Ergebnisse

Als Ergebnis der Performanz-Analysen wurden bei jedem Verfahren die Durchlaufzeiten für die einzelnen Entwurfsalternativen erfasst. Unter der Durchlaufzeit wird in diesem Fall die Zeit verstanden, die das System zur Bearbeitung einer Benutzeranfrage vom Betreten des Systems bis zum Verlassen benötigt.

Nachfolgend definieren wir Metriken zur Beantwortung der gestellten Fragen und interpretieren die Ergebnisse hinsichtlich des Ziels der Fallstudie. Die formal definierten Metriken können allgemein in vergleichbaren Studien angewandt werden. Darüber hinaus vergleichen wir die Empfehlungen für Entwurfsentscheidungen, die mit Hilfe der Vorhersageverfahren erzielt wurden, mit denen, die intuitiv von den Mitgliedern der Kontrollgruppe ausgesprochen wurden.

Frage 1 *Wie gut lässt sich mit den Verfahren die Realisierbarkeit quantitativer Performanz-Anforderungen feststellen?* Zur Beantwortung dieser Frage haben wir Metrik 1 mit

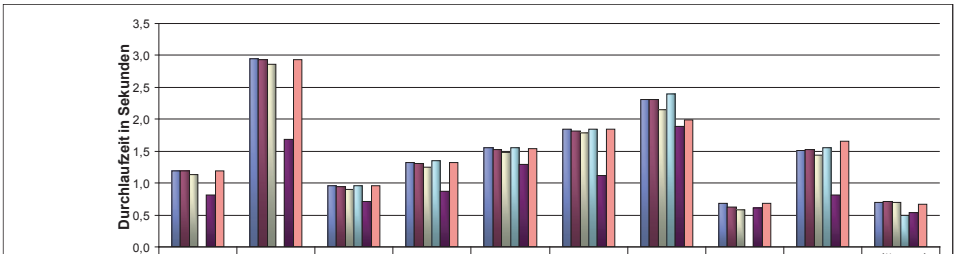
Hilfe der mittleren quadratischen Abweichung der vorhergesagten Werte vom Messwert wie folgt definiert:

$$\frac{1}{|\mathcal{A}|}\sum_{j=1}^{|\mathcal{A}|}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_{i,j}-X_{mess,j})^2}$$

Hier bezeichnet $\mathcal{A}$ die Menge aller Entwurfsalternativen, n die Anzahl der Teilnehmer der entsprechenden Gruppe, $X_{i,j}$ den von Teilnehmer i für die Alternative j vorhergesagten Wert und $X_{mess,j}$ den an der Implementierung der Alternative j gemessenen Wert. Dabei ist eine Abweichung um so besser zu bewerten, je kleiner sie ist.

Für jedes Verfahren haben wir die vorhergesagten Werte, sortiert nach Entwurfsalternativen und Teilnehmern, tabellarisch und in Form von Balkendiagrammen dargestellt. Zum Vergleich mit den Messungen an den entsprechenden Implementierungen enthält die vorletzte Spalte der jeweiligen Tabelle die gemessenen Werte. In der letzten Spalte steht die mittlere quadratische Abweichung vom Messwert. Das zur letzten Spalte automatisch generierte Diagramm ist nicht im Sinne der absoluten Durchlaufzeiten zu verstehen.

In der Abbildung 1 sind die mit dem *SPE-Verfahren* vorhergesagten Durchlaufzeiten dargestellt. Dabei gelang es mehreren Teilnehmern nicht, für Entwurfsalternative 2 eine



	1	2	3	4	5	6	7	8	Messung	mitt. quad. Abweichung
0: keine Optimierung	1,1928	2,9415	0,9562	1,3168	1,5503	1,8476	2,3094	0,6845	1,5100	0,7009
1a: statischer Cache	1,1883	2,9338	0,9499	1,3108	1,5196	1,8204	2,3030	0,6200	1,5300	0,7055
1b: dynamischer Cache	1,1317	2,8670	0,8981	1,2545	1,4868	1,7826	2,1478	0,5800	1,4400	0,6910
2: single-threaded			0,9546	1,3482	1,5477	1,8428	2,3944		1,5600	0,4874
3: Komprimierung	0,8188	1,6856	0,7096	0,8766	1,2979	1,1221	1,8822	0,6089	0,8100	0,5370
4: Clustering	1,1952	2,9367	0,9588	1,3152	1,5323	1,8421	1,9892	0,6878	1,6500	0,6656

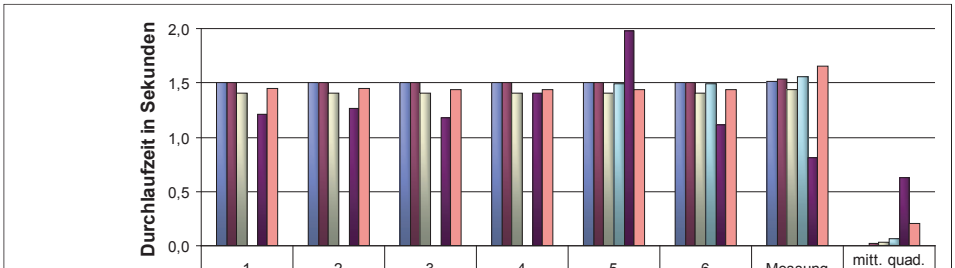
Teilnehmer

Figure 1: Vorhergesagte Durchlaufzeiten beim SPE-Verfahren

Durchlaufzeit zu berechnen, da das SPE-Verfahren die Modellierung von Threads nicht unterstützt. Das Verfahren benötigt Eingabewerte, die in der Entwurfsphase der Softwaresystem-Entwicklung noch nicht vorliegen. Diese Werte müssen zu diesem Zeitpunkt aus den vergleichbaren Systemen oder Prototypen bezogen oder basierend auf Erfahrungen geschätzt werden. In diesem Experiment wurden diese Werte von den Teilnehmern geschätzt. Das erklärt die großen Abweichungen der vorhergesagten Werte von der Messung. Die für dieses Verfahren berechnete Metrik 1 beträgt 0,6173.

Beim *Capacity-Planning-Verfahren* wurden Ergebnisse für statische und dynamische Seitenaufrufe getrennt berechnet bzw. gemessen. Bei den Vorhersagen, wie bei entsprechenden Messungen, dauerten Anfragen für Seiten mit dynamischen Inhalten im Schnitt 0,5

Sekunden länger im Vergleich zu statischen Seitenaufrufen. Ansonsten gab es keine Unterschiede im Verhaltensmuster der Anfragen. Deswegen präsentieren wir in diesem Artikel nur die Durchlaufzeiten für Seitenanfragen mit dynamischen Inhalten (Abbildung 2). In der Tabelle sind die Ergebnisse von nur 6 Telnehmern zusammengetragen, da zwei der Teilnehmer in der zur Verfügung stehenden Zeit mit der Lösung der Aufgaben nicht fertig geworden sind. Genauso wie beim SPE-Verfahren, gelang den meisten Teilnehmer keine



	1	2	3	4	5	6	Messung	mitt. quad. Abweichung
0: keine Optimierung	1,50536	1,50536	1,50533	1,50524	1,50510	1,50522	1,5100	0,0047
1a: statischer Cache	1,50536	1,50536	1,50533	1,50524	1,50510	1,50522	1,5300	0,0247
1b: dynamischer Cache	1,40399	1,40399	1,40397	1,40387	1,40373	1,40386	1,4400	0,0361
2: single-threaded					1,49110	1,49190	1,5600	0,0685
3: Komprimierung	1,21125	1,26893	1,18273	1,40061	1,98167	1,10916	0,8100	0,6220
4: Clustering	1,45336	1,45336	1,44094	1,44089	1,44082	1,44089	1,6500	0,2050

Teilnehmer

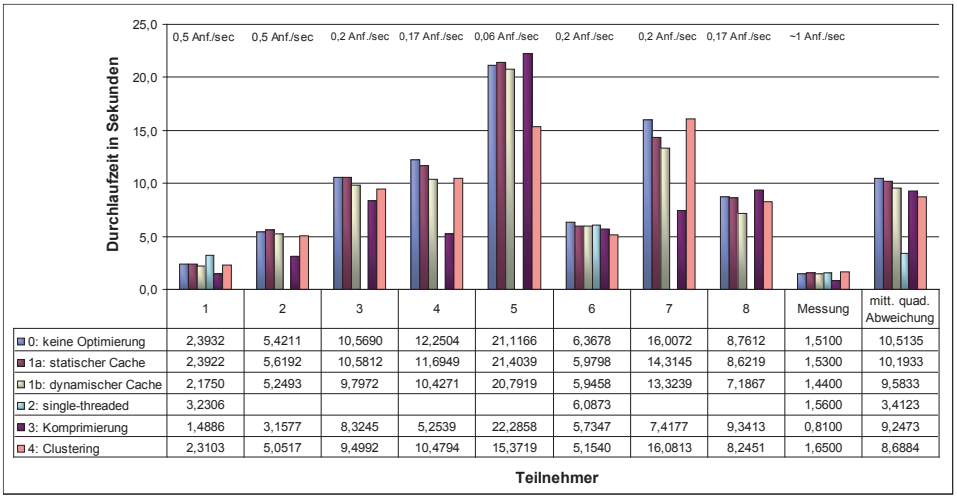
Figure 2: Vorhergesagte Durchlaufzeiten beim CP-Verfahren (dynamische Seitenabfragen)

Modellierung der Alternative 2. Bei diesem Verfahren unterscheiden sich die Vorhersagen der Teilnehmer, bis auf Alternative 3, sehr geringfügig von der Messung an den Implementierungen. Das liegt daran, dass das Verfahren größtenteils mit den am fertigen System gemessenen Daten arbeitet. Bei der Alternative 3 musste eine Kompressionsrate für die Dateien abgeschätzt werden, deshalb ergeben sich hier sichtbare Abweichungen in den Werten. Die für das CP-Verfahren berechnete Metrik 1 beträgt 0,1913.

Die mit *umlPSI* vorhergesagten Durchlaufzeiten für Seitenabfragen mit dynamischen Inhalten sind in der Abbildung 3 dargestellt. Beim umlPSI-Verfahren werden auf Schätzungen basierte Eingabedaten nicht nach Software- und Hardware-Ressourcen getrennt, wie es z.B. beim SPE-Verfahren der Fall ist. Außerdem stützt sich das umlPSI-Verfahren auf weniger Eingabedaten, was die Vorhersage ungenauer macht. Daher kommen auch die großen Abweichungen der vorhergesagten Werte von den Messungen an den Implementierungen. Der mit Metrik 1 berechnete Wert ist hier 8,2249.

Als Fazit lässt sich feststellen, dass das CP-Verfahren wegen seiner vergleichsweise hohen Genauigkeit bei der Vorhersage der absoluten Zeiten besonders zur Bewertung der Realisierbarkeit von Performanz-Anforderungen geeignet ist. Da allerdings das CP-Verfahren Meßwerte eines bestehenden Systems voraussetzt, kann es bei Neuentwicklungen erst in späteren Phasen der Software-Entwicklung eingesetzt werden.

Frage 2 *In wie weit unterstützen die Verfahren die Auswahl der richtigen Entwurfsalternative?* Zur Beantwortung dieser Frage wird zunächst für die Entwurfsalternativen anhand



	1	2	3	4	5	6	7	8	Messung	mitt. quad. Abweichung
0: keine Optimierung	2,3932	5,4211	10,5690	12,2504	21,1166	6,3678	16,0072	8,7612	1,5100	10,5135
1a: statischer Cache	2,3922	5,6192	10,5812	11,6949	21,4039	5,9798	14,3145	8,6219	1,5300	10,1933
1b: dynamischer Cache	2,1750	5,2493	9,7972	10,4271	20,7919	5,9458	13,3239	7,1867	1,4400	9,5833
2: single-threaded	3,2306					6,0873			1,5600	3,4123
3: Komprimierung	1,4886	3,1577	8,3245	5,2539	22,2858	5,7347	7,4177	9,3413	0,8100	9,2473
4: Clustering	2,3103	5,0517	9,4992	10,4794	15,3719	5,1540	16,0813	8,2451	1,6500	8,6884

Teilnehmer

Figure 3: Vorhergesagte Durchlaufzeiten beim umlPSI-Verfahren (dynamische Seitenabfragen)

der gemessenen Durchlaufzeiten eine Rangliste aufgestellt. Wir definieren eine Abbildung $Pos_M : \mathcal{A} \longrightarrow \{1, ..., |\mathcal{A}|\}$ mit $Pos(AlternativeA) \leq Pos(AlternativeB)$, falls die Durchlaufzeit der Alternative A kleiner oder gleich der Durchlaufzeit der Alternative B ist. Ebenso werden für jeden Teilnehmer die Entwurfsalternativen in eine Rangliste gemäß seiner vorhergesagten Durchlaufzeiten geordnet. Die Abbildung Pos_V ordnet analog zu Pos_M jeder Alternative ihren Platz in der Rangliste bez. der vorhergesagten Durchlaufzeiten. Im nächsten Schritt wollen wir berechnen, um wie viele Positionen die von Teilnehmern vorhergesagte Rangordnung von der gemessenen abweicht. Damit Platzvertauschungen der Rangordnung bei geringfügigen Abweichungen in den Durchlaufzeiten nicht in die Rechnung aufgenommen werden, werden die Alternativen mit einem dichtebasierten Clustering-Verfahren [ES00] in Klassen eingeteilt. Auf diese Weise werden Alternativen mit dicht nebeneinander liegenden Durchlaufzeiten als gleichwertig angesehen und gehören zur selben Klasse. Die monoton steigende Abbildung
$Klasse : \{1, ..., |\mathcal{A}|\} \longrightarrow \{1, ..., |Klassen|\}$ ordnet jeder Position der gemessenen Rangliste die Klasse der zugehörigen Alternative zu.

Um die Metrik 2 zum Vergleich der vorhergesagten mit den gemessenen Reihenfolge der Alternativen zu konstruieren, definieren wir
$Abweichung_i(Alt) := |Klasse(Pos_{V_i}(Alt)) - Klasse(Pos_M(Alt))|$.
Diese Formel gibt für jeden Teilnehmer i die Häufigkeit der Klassengrenzenüberschreitungen bei der Vorhersage der Rangordnung der Alternative Alt an. Platzvertauschungen innerhalb einer Klasse werden nicht erfaßt. Die Metrik 2 wird dann wie folgt definiert:

$$\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{|\mathcal{A}|} Abweichung_i(Alt_j),$$

wobei n die Anzahl der Teilnehmer der entsprechenden Gruppe ist. Je kleiner die auf diese Art berechnete Zahl ist, desto weniger Fehlplatzierungen wurden mit einem Verfahren

erzielt.

Die Abbildung 4 zeigt den Vergleich der gemessenen und der vorhergesagten Ranglisten unseres Experiments. Nach dem Clustering-Verfahren wurde die Alternative 3 der Klasse 1 zugeordnet und die anderen Alternativen der Klasse 2. Es zeigt sich, dass bei

			SPE								CP						umlPSI							
	Klasseneinteilung	Messung	Teilnehmer 1	Teilnehmer 2	Teilnehmer 3	Teilnehmer 4	Teilnehmer 5	Teilnehmer 6	Teilnehmer 7	Teilnehmer 8	Teilnehmer 1	Teilnehmer 2	Teilnehmer 3	Teilnehmer 4	Teilnehmer 5	Teilnehmer 6	Teilnehmer 1	Teilnehmer 2	Teilnehmer 3	Teilnehmer 4	Teilnehmer 5	Teilnehmer 6	Teilnehmer 7	Teilnehmer 8
Ranking Entwurfsalternativen:																								
1. (schnellste Durchlaufzeit)	Klasse 1	3	3	3	3	3	3	3	3	1b	3	3	3	3	1b	3	3	3	3	3	4	4	3	4
2.	Klasse 2	1b	1b	1b	1b	1b	1b	1b	4	3	1b	1b	1b	1b	4	1b	1b	4	4	1b	1b	3	1b	1b
3.	Klasse 2	1a	1a	1a	1a	1a	1a	1a	1b	1a	4	4	4	4	2	4	4	1b	1b	4	1a	1b	1a	1a
4.	Klasse 2	2	4	4	2	4	4	4	1a	4	1a	1a	1a	1a	1a	1a	1a	1a	1a	1a	3	1a	4	3
5. (langsamste Durchlaufzeit)	Klasse 2	4	2	2	4	2	2	2	2	2	2	2	2	2	3	2	2	2	2	2	2	2	2	2
Metrik 2:			0,25								0,33						0,75							

Figure 4: Bewertung der Entwurfsalternativen nach Durchlaufzeit

den SPE- und CP-Verfahren die vorhergesagten Reihenfolgen der Alternativen geringe Abweichungen von der Messung aufweisen. Die Abweichungen in den vorhergesagten Durchlaufzeiten bei den hinteren Platzierungen waren recht gering und die Reihenfolge hat sich oft geändert. Solche Platzvertauschungen werden aber nicht angerechnet, da die Alternativen 1b bis 4 zur selben Klasse gehören. Durch das umlPSI-Verfahren wurden Alternativen aus einer „langsameren“ Klasse oft als die Performantesten vorhergesagt. Hieraus resultiert auch der deutlich höhere Wert der Metrik 2. Als Fazit läßt sich feststellen, daß die SPE- und CP-Verfahren für das eingesetzte Beispielsystem am besten die Wahl der Entwurfsalternativen unterstützten.

Methodisches versus intuitives Vorgehen Die Teilnehmer aus der Kontrollgruppe der Fallstudie hatten keine Rangordnung der Entwurfsalternativen anzugeben, sondern mussten die ihrer Meinung nach beste Alternative angeben und begründen. Vier der sieben Mitglieder der Kontrollgruppe bevorzugten die Alternative 1b (dynamischer Cache), lediglich drei Teilnehmer entschieden sich für Alternative 3 (Komprimierung). Obwohl eine Stichprobe von nur sieben Personen kaum aussagekräftig ist, ist es trotzdem auffällig, dass der größte Teil der Kontrollgruppe sich für eine Alternative entschieden hat, die in nur 9% der vorhergesagten Fälle die besten Durchlaufzeiten hatte.

Damit ist ebenfalls gezeigt, dass die Aufgabenstellung des Experiments bei einem intuitiven Vorgehen nicht automatisch zur Entscheidung für die beste Alternative führt.

5 Einschränkungen

Bei Planung und Durchführung des Experimentes waren Einschränkungen der Gültigkeit der Ergebnisse notwendig, die im folgenden diskutiert werden. Als Versuchsteilnehmer konnten nur Studenten hinzugezogen werden. Praktiker aus der Software-Industrie mit Erfahrung in Peformanz-Analysen nahmen an dieser Studie nicht teil. Die studentischen

Teilnehmer hatten aber aufgrund ihres abgeschlossenen Vordiploms und eines absolvierten Software-Praktikums eine solide Grundqualifikation. Die Anzahl der Teilnehmer war begrenzt, nur 8 Personen untersuchten jeweils einzeln ein Verfahren. Damit konnten zwar Abweichungen in den Ergebnissen der Teilnehmer durch Mittelung zumindest geringfügig relativiert werden und der Einfluß der persönlichen Leistungsfähigkeit abgeschätzt werden. Für einen Hypothesentest lagen jedoch nicht genügend Daten vor.

Die Untersuchung beschränkte sich auf drei unterschiedliche Performanz-Vorhersageverfahren. Andere Verfahren, die als Notation z.B. stochastische Petri-Netze oder stochastische Prozessalgebren nutzen, wurden ebenso wenig untersucht wie Verfahren, die auf erweiterten oder geschichteten Warteschlangenmodellen beruhen. Die Eingabedaten konnten für die verschiedene Verfahren nicht bei allen Gruppen exakt gleich gehalten werden. Für das CP-Verfahren mussten Messungen an einem Prototypen vorgegeben werden, damit eine Vorhersage der Gesamtperformanz überhaupt möglich war.

Der Entwurf des untersuchten experimentellen Webservers ist ein einfaches Beispiel für eine Software-Architektur. Ob sich die Ergebnisse auf größere Projekte in der Praxis verallgemeinern lassen, ist unklar.

Die Messwerte beziehen sich auf eine Implementierung des Webservers. Prinzipiell kann das Zeitverhalten anderer Implementierungen abweichen.

6 Zusammenfassung

Es wurden drei Vorhersageverfahren in Fallstudien untersucht und die Vorhersagen verglichen mit den Messwerten einer realen Implementierung. Wegen der Evaluation eines Vorhersageverfahrens mit jeweils acht Fallstudien lassen sich durch den Vergleich der deskriptiven Daten einige Schlüsse und Hypothesen für weitere Untersuchungen ableiten. In der Studie eignete sich das SPE-Verfahren für die Unterstützung der Auswahl der besten Entwurfsalternative. Zur Validierung von Performanz-Zielen in frühen Entwicklungsphasen ist dieses Verfahren weniger geeignet, da Schätzungen stark die Vorhersagen beeinflussen können. Das umlPSI-Verfahren benötigt weniger Eingabedaten als SPE. Dadurch wirken sich Fehler bei geschätzten Werten stärker auf die Ergebnisse aus. Folglich ist es zur Unterstützung von Entwurfsalternativen nur bedingt und zur Validierung von Performanz-Zielen nicht geeignet. Das CP-Verfahren hatte die geringsten Abweichungen von den gemessenen Werten, da es auf gemessene Performanz-Werte einer bestehenden Systemvariante aufbaut. Folglich kann es beim Entwurf neuer Systeme nur bedingt eingesetzt werden.

Allgemein hat sich bestätigt, dass in den frühen Entwicklungsphasen die Verfahren zur Unterstützung der Entscheidung für verschiedene Entwurfsalternativen herangezogen werden können. In späteren Phasen der Entwicklung, wenn genauere Daten zur Realisierung des Systems vorliegen, können auch Performanz-Ziele validiert werden.

Durch die Studie wurden auch Erfahrungen gesammelt, die zur Formulierung neuer Hypothesen führen. Zum Beispiel war es einem hohen Anteil der Teilnehmer nicht möglich, Nebenläufigkeit in SPE oder CP zu modellieren. Das heißt, die Tatsache, dass ein System multi-threaded statt single-threaded arbeitet, war nicht adäquat ausdrückbar. Interes-

sant ist nun die empirische Untersuchung der Frage, unter welchen Umständen sich dies auf die Vorhersagequalität auswirkt. Daraus können sich konkrete Anforderungen zur Verbesserung von Vorhersageverfahren ergeben. Weiterhin ist die Sensitivität der Vorhersageverfahren auf die Genauigkeit des in der Vorhersage verwendeten Benutzungsprofiles zu untersuchen.

References

[BCR94] V. R. Basili, G. Caldiera, and H. D. Rombach. The Goal Question Metric Approach. *Encyclopedia of Software Engineering - 2 Volume Set*, pages 528–532, 1994.

[BDIS04] S. Balsamo, A. DiMarco, P. Inverardi, and M. Simeoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, May 2004.

[BM03] S. Balsamo and M. Marzolla. A Simulation-Based Approach to Software Performance Modeling. Technical report, Universitá Ca' Foscari di Venezia, 2003.

[BMDI04] S. Balsamo, M. Marzolla, A. DiMarco, and P. Inverardi. Experimenting different software architectures performance techniques: A case study. In *Proceedings of the Fourth International Workshop on Software and Performance*, pages 115–119. ACM Press, 2004.

[ES00] Martin Ester and Jörg Sander. *Knowledge Discovery in Databases - Techniken und Anwendungen*. Springer-Verlag, Berlin, 2000.

[GL03] I. Gorton and A. Liu. Performance Evaluation of Alternative Component Architectures for Enterprise JavaBean Applications. *IEEE Internet Computing*, 7(3):18–23, 2003.

[JM01] N. Juristo and A.M. Moreno. *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers, 2001.

[Koz04] H. Koziolek. Empirische Bewertung von Performance-Analyseverfahren für Software-Architekturen. Diplomarbeit, Universität Oldenburg, Fakultät II, Department für Informatik, Okt. 2004.

[MAD04] D. A. Menasce, V. A. F. Almeida, and L. W. Dowdy. *Performance by Design*. Prentice Hall, 2004.

[Mar04] M. Marzolla. *Simulation-Based Performance Modeling of UML Software Architectures*. PhD thesis, Universit'a Ca Foscari di Venezia, 2004.

[OMG03] Object Management Group OMG. Unified Modelling Language (UML), Version 1.5. http://www.omg.org/cgi-bin/doc?formal/03-03-01, 2003.

[Pre01] L. Prechelt. *Kontrollierte Experimente in der Softwaretechnik*. Springer Verlag, 2001.

[SG96] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. 1996.

[Smi02] C. U. Smith. *Performance Solutions: A Practical Guide To Creating Responsive, Scalable Software*. Addison-Wesley, 2002.

[WRH+00] C. Wohling, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering – An Introduction*. Kluwer Academic Publishers, 2000.