Methoden zur kontinuierlichen Performancemessung bei der agilen Entwicklung von Webanwendungen

Diplomarbeit

Patrick-Thomas Chmielewski
Matrikelnummer 3479870

Vorgelegt an der
Freien Universität Berlin
bei
Prof. Dr. Lutz Prechelt
am 24. September 2009
# Inhaltsverzeichnis

**Abbildungsverzeichnis** IV

**Tabellenverzeichnis** V

**Listings** VI

<table>
<thead>
<tr>
<th>1</th>
<th>Vorwort</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Einleitung</td>
<td>2</td>
</tr>
<tr>
<td>2.1</td>
<td>Motivation</td>
<td>2</td>
</tr>
<tr>
<td>2.2</td>
<td>Ziel</td>
<td>3</td>
</tr>
<tr>
<td>2.3</td>
<td>Gliederung</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Grundlagen</td>
<td>5</td>
</tr>
<tr>
<td>3.1</td>
<td>Was ist Performance?</td>
<td>5</td>
</tr>
<tr>
<td>3.2</td>
<td>Welche Faktoren beeinflussen die Performance?</td>
<td>7</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Clientseite</td>
<td>8</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Serverseite</td>
<td>11</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Netzwerk</td>
<td>14</td>
</tr>
<tr>
<td>3.3</td>
<td>Warum ist Performance wichtig?</td>
<td>16</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Psychologische Gründe</td>
<td>16</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Ökonomische Gründe</td>
<td>19</td>
</tr>
<tr>
<td>3.4</td>
<td>Wie wird Performance gemessen?</td>
<td>22</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Unterschiedliche Arten der Evaluation</td>
<td>22</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Benchmark vs. Load-Testing</td>
<td>23</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Workload</td>
<td>24</td>
</tr>
<tr>
<td>3.4.4</td>
<td>Metriken</td>
<td>24</td>
</tr>
<tr>
<td>3.4.5</td>
<td>Testumgebung</td>
<td>25</td>
</tr>
<tr>
<td>3.5</td>
<td>Vorhandene Werkzeuge zur Performancemessung</td>
<td>26</td>
</tr>
<tr>
<td>3.6</td>
<td>Agile Softwareentwicklung</td>
<td>28</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Kontinuierliche Integration</td>
<td>29</td>
</tr>
<tr>
<td>3.7</td>
<td>Rückblick</td>
<td>30</td>
</tr>
</tbody>
</table>

<p>| 4 | Das Continuous Performance Testing Framework | 31 |
| 4.1 | Vorüberlegungen | 33 |
| 4.2 | Kurzfassung der Funktionsweise | 35 |
| 4.3 | Server | 37 |
| 4.4 | Softwareagenten | 40 |
| 4.4.1 | Multiagentensystem | 42 |
| 4.4.2 | Verteilte Arbeiteragenten | 43 |</p>
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4.3 Monitoragensten</td>
<td>45</td>
</tr>
<tr>
<td>4.5 Technische Umsetzung</td>
<td>45</td>
</tr>
<tr>
<td>4.5.1 Model</td>
<td>48</td>
</tr>
<tr>
<td>4.5.2 View</td>
<td>52</td>
</tr>
<tr>
<td>4.5.3 Controller</td>
<td>53</td>
</tr>
<tr>
<td>4.5.4 Softwareagenten</td>
<td>56</td>
</tr>
<tr>
<td>4.5.5 CruiseControl.rb Zusatzmodul</td>
<td>64</td>
</tr>
<tr>
<td>4.6 Qualitätssicherung</td>
<td>66</td>
</tr>
<tr>
<td>4.7 Rückblick</td>
<td>71</td>
</tr>
<tr>
<td>5 Evaluierung</td>
<td>72</td>
</tr>
<tr>
<td>5.1 Durchführung</td>
<td>72</td>
</tr>
<tr>
<td>5.1.1 Kontinuierlicher Performancetest</td>
<td>72</td>
</tr>
<tr>
<td>5.1.2 Referenzmessung</td>
<td>74</td>
</tr>
<tr>
<td>5.2 Aufbau</td>
<td>75</td>
</tr>
<tr>
<td>5.3 Ergebnisse</td>
<td>76</td>
</tr>
<tr>
<td>5.3.1 Gruppe eins: Drei Stressclients</td>
<td>76</td>
</tr>
<tr>
<td>5.3.2 Gruppe zwei: Zwei Stressclients</td>
<td>77</td>
</tr>
<tr>
<td>5.4 Interpretation</td>
<td>78</td>
</tr>
<tr>
<td>6 Fazit</td>
<td>80</td>
</tr>
<tr>
<td>7 Ausblick</td>
<td>82</td>
</tr>
<tr>
<td>8 Zusammenfassung</td>
<td>84</td>
</tr>
<tr>
<td>Literatur</td>
<td>VII</td>
</tr>
<tr>
<td>A Anhang Entwurf</td>
<td>XII</td>
</tr>
<tr>
<td>B Anhang Evaluierung</td>
<td>XVI</td>
</tr>
<tr>
<td>B.1 Hard- und Softwarekonfiguration der Testumgebung</td>
<td>XVI</td>
</tr>
<tr>
<td>B.2 Szenarien</td>
<td>XVII</td>
</tr>
<tr>
<td>B.3 Drei virtuelle Benutzer</td>
<td>XVIII</td>
</tr>
<tr>
<td>B.3.1 Diagramme</td>
<td>XVIII</td>
</tr>
<tr>
<td>B.3.2 Messdaten</td>
<td>XXI</td>
</tr>
<tr>
<td>B.4 Zwei virtuelle Benutzer</td>
<td>XXV</td>
</tr>
<tr>
<td>B.4.1 Diagramme</td>
<td>XXV</td>
</tr>
<tr>
<td>B.4.2 Messdaten</td>
<td>XXVIII</td>
</tr>
<tr>
<td>B.5 Gegenüberstellung Gesamtperformance Gruppe eins und zwei</td>
<td>XXXII</td>
</tr>
<tr>
<td>C Anhang Datenträger</td>
<td>XXXIV</td>
</tr>
</tbody>
</table>
Abbildungsverzeichnis

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Abbildungstitel</th>
<th>Seitenzahl</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Generalisierte Testumgebung für Performancetests.</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>Kosten von Änderungen (cost of change) nach Boehm (1976).</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>Kosten von Änderungen (cost of change) nach Beck (2000).</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>Übersicht der Komponenten des ContinuousPerformanceTesting-Frameworks.</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>Auflistung aller Projekte und deren Gesamtperformance.</td>
<td>38</td>
</tr>
<tr>
<td>6</td>
<td>Ausgewähltes Projekt und dessen zugehörige Builds.</td>
<td>38</td>
</tr>
<tr>
<td>7</td>
<td>Detaillierte Ansicht eines Builds.</td>
<td>39</td>
</tr>
<tr>
<td>8</td>
<td>Detaillierte Ansicht eines Builds mit Szenariodefinition.</td>
<td>39</td>
</tr>
<tr>
<td>9</td>
<td>Performanceübersicht aller Szenarien eines Projekts.</td>
<td>41</td>
</tr>
<tr>
<td>10</td>
<td>Aktueller Zustand des Anführer-Agenten und seiner Diener.</td>
<td>41</td>
</tr>
<tr>
<td>11</td>
<td>Interaktion der Agenten untereinander und mit dem Server.</td>
<td>44</td>
</tr>
<tr>
<td>12</td>
<td>Model-View-Controller Architektur.</td>
<td>46</td>
</tr>
<tr>
<td>13</td>
<td>Datenmodell.</td>
<td>49</td>
</tr>
<tr>
<td>14</td>
<td>Versuchsauflauf zur Durchführung der Performancemessung.</td>
<td>75</td>
</tr>
<tr>
<td>15</td>
<td>Gesamtperformance (3 VU).</td>
<td>76</td>
</tr>
<tr>
<td>16</td>
<td>Gesamtperformance (2 VU).</td>
<td>77</td>
</tr>
<tr>
<td>17</td>
<td>Diagramm der Gesamtperformance: 3 Clients vs. 2 Client gemäß Tabelle 19.</td>
<td>79</td>
</tr>
<tr>
<td>18</td>
<td>Erster Papierprototyp des GUI.</td>
<td>XII</td>
</tr>
<tr>
<td>19</td>
<td>Zweiter Papierprototyp - Projektübersicht.</td>
<td>XIII</td>
</tr>
<tr>
<td>20</td>
<td>Zweiter Papierprototyp - Übersicht der Builds eines Projekts.</td>
<td>XIV</td>
</tr>
<tr>
<td>21</td>
<td>Zweiter Papierprototyp - Übersicht aller Szenarien eines Projekts.</td>
<td>XV</td>
</tr>
<tr>
<td>22</td>
<td>Zweiter Papierprototyp - Detaillansicht des Diagramms.</td>
<td>XV</td>
</tr>
<tr>
<td>23</td>
<td>Szenario add t-shirt to cart and continue shopping (3 VU).</td>
<td>XIX</td>
</tr>
<tr>
<td>24</td>
<td>Szenario add t-shirt to cart and increment quantity (3 VU).</td>
<td>XIX</td>
</tr>
<tr>
<td>25</td>
<td>Szenario browse through categories (3 VU).</td>
<td>XIX</td>
</tr>
<tr>
<td>26</td>
<td>Szenario show all brands (3 VU).</td>
<td>XX</td>
</tr>
<tr>
<td>27</td>
<td>Szenario show all categories (3 VU).</td>
<td>XX</td>
</tr>
<tr>
<td>28</td>
<td>Gesamtperformance (3 VU).</td>
<td>XX</td>
</tr>
<tr>
<td>29</td>
<td>Referenzmessung basierend auf Revision 10 (Tabelle 3, 3 VU).</td>
<td>XXI</td>
</tr>
<tr>
<td>30</td>
<td>Szenario add t-shirt to cart and continue shopping (2 VU).</td>
<td>XXV</td>
</tr>
<tr>
<td>31</td>
<td>Szenario add t-shirt to cart and increment quantity (2 VU).</td>
<td>XXVI</td>
</tr>
<tr>
<td>32</td>
<td>Szenario browse through categories (2 VU).</td>
<td>XXVI</td>
</tr>
<tr>
<td>33</td>
<td>Szenario show all brands (2 VU).</td>
<td>XXVI</td>
</tr>
<tr>
<td>34</td>
<td>Szenario show all categories (2 VU).</td>
<td>XXVII</td>
</tr>
<tr>
<td>35</td>
<td>Gesamtperformance (2 VU).</td>
<td>XXVII</td>
</tr>
<tr>
<td>36</td>
<td>Referenzmessung basierend auf Revision 10 (Tabelle 11, 2 VU).</td>
<td>XXVII</td>
</tr>
<tr>
<td>37</td>
<td>Diagramm der Gesamtperformance auf 3 vs. 2 Clients.</td>
<td>XXXII</td>
</tr>
<tr>
<td>38</td>
<td>Diagramm der Gesamtperformance 3 vs. 2 Clients (Medianbasis).</td>
<td>XXXIII</td>
</tr>
</tbody>
</table>
## Tabellenverzeichnis

<table>
<thead>
<tr>
<th>Nummer</th>
<th>Beschreibung</th>
<th>Seite</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prozentueller Anteil der Ladezeit verursacht durch das <em>Back-End</em></td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>Zeit für Routing zu einem entfernten Host (hin und zurück)</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>Auf Performance hin untersuchte Revisionen (3 VU)</td>
<td>XXI</td>
</tr>
<tr>
<td>4</td>
<td>Ermittelte Gesamtperformance (3 VU)</td>
<td>XXII</td>
</tr>
<tr>
<td>5</td>
<td>Messwerte (3 VU): scenario show all categories</td>
<td>XXII</td>
</tr>
<tr>
<td>6</td>
<td>Messwerte (3 VU): scenario browse through categories</td>
<td>XXIII</td>
</tr>
<tr>
<td>7</td>
<td>Messwerte (3 VU): scenario show all brands</td>
<td>XXIII</td>
</tr>
<tr>
<td>8</td>
<td>Messwerte (3 VU): scenario add t-shirt to cart and increment quantity</td>
<td>XXIV</td>
</tr>
<tr>
<td>9</td>
<td>Messwerte (3 VU): scenario add t-shirt to cart and continue shopping</td>
<td>XXIV</td>
</tr>
<tr>
<td>10</td>
<td>Referenzmessung, Revision 10 (3 VU)</td>
<td>XXV</td>
</tr>
<tr>
<td>11</td>
<td>Auf Performance hin untersuchte Revisionen (2 VU)</td>
<td>XXVIII</td>
</tr>
<tr>
<td>12</td>
<td>Ermittelte Gesamtperformance (2 VU)</td>
<td>XXVIII</td>
</tr>
<tr>
<td>13</td>
<td>Messwerte (2 VU): scenario show all categories</td>
<td>XXIX</td>
</tr>
<tr>
<td>14</td>
<td>Messwerte (2 VU): scenario browse through categories</td>
<td>XXIX</td>
</tr>
<tr>
<td>15</td>
<td>Messwerte (2 VU): scenario show all brands</td>
<td>XXX</td>
</tr>
<tr>
<td>16</td>
<td>Messwerte (2 VU): scenario add t-shirt to cart and increment quantity</td>
<td>XXX</td>
</tr>
<tr>
<td>17</td>
<td>Messwerte (2 VU): scenario add t-shirt to cart and continue shopping</td>
<td>XXXI</td>
</tr>
<tr>
<td>18</td>
<td>Referenzmessung, Revision 10 (Tabelle 11, 2 VU)</td>
<td>XXXI</td>
</tr>
<tr>
<td>19</td>
<td>Gegenüberstellung 3 vs. 2 Clients</td>
<td>XXXII</td>
</tr>
<tr>
<td>20</td>
<td>Gegenüberstellung 3 vs. 2 Clients (Medianbasis)</td>
<td>XXXIII</td>
</tr>
</tbody>
</table>
## Listings

<table>
<thead>
<tr>
<th>Listing</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Beispielhaftes Cucumber-Szenario</td>
<td>45</td>
</tr>
<tr>
<td>2</td>
<td>Migration der Klasse Build</td>
<td>47</td>
</tr>
<tr>
<td>3</td>
<td>Ausschnitt der Definition der beiden Models Build und Project</td>
<td>47</td>
</tr>
<tr>
<td>4</td>
<td>Active Resource-Stellvertreter</td>
<td>48</td>
</tr>
<tr>
<td>5</td>
<td>Berechnung der durchschnittlichen Ladezeit eines Szenarios</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>Berechnung der normalisierten Gesamtperformance eines Build</td>
<td>50</td>
</tr>
<tr>
<td>7</td>
<td>Die Klasse MeasuredData</td>
<td>51</td>
</tr>
<tr>
<td>8</td>
<td>Teil der create Methode des BuildController</td>
<td>52</td>
</tr>
<tr>
<td>9</td>
<td>Helfer zur Generierung von Box-Plots</td>
<td>53</td>
</tr>
<tr>
<td>10</td>
<td>Before-Filter im Controller</td>
<td>55</td>
</tr>
<tr>
<td>11</td>
<td>Routing-Definition</td>
<td>56</td>
</tr>
<tr>
<td>12</td>
<td>Anführersuche beim Distributed Worker</td>
<td>57</td>
</tr>
<tr>
<td>13</td>
<td>Aufruf von Cucumber durch einen Distributed Worker</td>
<td>59</td>
</tr>
<tr>
<td>14</td>
<td>Definition der Cucumber-Schritte zur Messung der Ladezeit</td>
<td>60</td>
</tr>
<tr>
<td>15</td>
<td>Synchronisierter Zugriff auf den Ergebnisspeicher des Anführer-Agenten</td>
<td>61</td>
</tr>
<tr>
<td>16</td>
<td>Benachrichtigung der Monitor Worker durch den Anführer</td>
<td>61</td>
</tr>
<tr>
<td>17</td>
<td>Initialisierung mehrerer Threads durch den Monitor Worker</td>
<td>62</td>
</tr>
<tr>
<td>18</td>
<td>Starten aller Monitor-Threads durch den Monitor Worker</td>
<td>62</td>
</tr>
<tr>
<td>19</td>
<td>Beispielhafte Hashtabelle mit Messdaten</td>
<td>63</td>
</tr>
<tr>
<td>20</td>
<td>Konfiguration zur Überwachung der Agenten durch GOD</td>
<td>63</td>
</tr>
<tr>
<td>21</td>
<td>Auszug der Capistrano-Konfiguration. Entfernte Steuerung von GOD</td>
<td>64</td>
</tr>
<tr>
<td>22</td>
<td>Kontinuierter Performance Testing-Zusatzmodul für CruiseControl.rb</td>
<td>66</td>
</tr>
<tr>
<td>23</td>
<td>Undenierte Erwartungen an das Statistics-Modul (statistics_spec.rb)</td>
<td>68</td>
</tr>
<tr>
<td>24</td>
<td>Ausgabe des ersten RSpec-Durchgangs</td>
<td>68</td>
</tr>
<tr>
<td>25</td>
<td>Definitionen von Erwartungen an das Statistics-Modul</td>
<td>69</td>
</tr>
<tr>
<td>26</td>
<td>Auszug der Ausgabe eines fehlschlagenen Tests</td>
<td>70</td>
</tr>
<tr>
<td>27</td>
<td>Erfolgreicher Testdurchlauf nach erfolgter Implementierung</td>
<td>70</td>
</tr>
<tr>
<td>28</td>
<td>Bestimmung des Maximalwerts durch Sortierung einer Liste</td>
<td>70</td>
</tr>
<tr>
<td>29</td>
<td>Berechnung des Maximalwerts durch Vergleichen von je zwei Werten</td>
<td>71</td>
</tr>
<tr>
<td>30</td>
<td>Spree Szenario 1: Add T-Shirt into the cart and continue shopping</td>
<td>XVI</td>
</tr>
<tr>
<td>31</td>
<td>Spree Szenario 2: Add T-Shirt into the cart and increment quantity</td>
<td>XVI</td>
</tr>
<tr>
<td>32</td>
<td>Spree Szenario 3: Browse through categories</td>
<td>XVII</td>
</tr>
<tr>
<td>33</td>
<td>Spree Szenario 4: Show all brands</td>
<td>XVIII</td>
</tr>
<tr>
<td>34</td>
<td>Spree Szenario 5: Show all categories</td>
<td>XVIII</td>
</tr>
</tbody>
</table>
1 Vorwort


An dieser Stelle bedanke ich mich bei Herrn Thomas Witt für dessen Unterstützung und Einsatz während der Themenwahl sowie die Möglichkeit, diese Abschlussarbeit in der Firma anfertigen zu können.

Mein Dank gilt ferner Prof. Dr. Lutz Prechelt am Institut für Informatik der Freien Universität Berlin für die wissenschaftliche Betreuung und die hilfreichen Anregungen.

2 Einleitung

Die Entwicklung von Webanwendungen ist ein schwieriger Prozess. Häufig ändern sich noch vor der Fertigstellung die Anforderungen und stellen damit die Entwickler vor neue Herausforderungen.

Die agilen Methoden haben sich bei der Bewältigung dieser Aufgaben als sehr hilfreich erweisen. Kurze Iterationen erlauben eine flexible und schnelle Umsetzung neuer Anforderungen. Testgetriebene Entwicklung hilft bei der Sicherstellung einer hohen Codequalität. Sie gibt durch die kontinuierliche Integration fortlaufend Auskunft über die funktionale Korrektheit des getesteten Codes.

Die hohe Komplexität moderner Webanwendungen macht die Betrachtung der Performance notwendig. Eine lange Ladezeit, bedingt durch eine schlechte Performance, hat nicht nur Auswirkungen auf die Zufriedenheit des Endanwenders; schon kleinste Verzögerungen im Bereich von 100 Millisekunden haben einen direkten Einfluss auf den Umsatz (vgl. Untersuchungen von Google [Far06] und Amazon [KL07]).


2.1 Motivation


Durch die Eingabe des URL1 im Browser wird die Anwendung gestartet. Die dabei generierte Anfrage wird vom Webserver empfangen und an das eigentliche Programm übergeben. Bis eine Antwort-Seite erzeugt und an den Browser über den Webserver gesendet

---

1URL steht für Uniform Resource Locator und ist die Adresse eine Ressource mitsamt des notwendigen Netzwerkprotokolls. Das URL des Internetauftritts des Instituts für Informatik der Freien Universität Berlin lautet beispielsweise http://mi.fu-berlin.de/

In die Antwortseite können weitere Elemente eingebettet sein, was zu weiteren Anfragen führt. Ferner können mit Ajax-Requests dynamische Inhalte angefordert werden und die bereits dargestellte Seite nachträglich verändert. Die klassische Seiten-Logik, bei der mit jeder Serveranfrage eine neue Seite vom Server erzeugt und übertragen wird, ist damit vollständig durchbrochen.

Existierende Werkzeuge analysieren nur einzelne ganz spezielle Aspekte einer ganzen Kette von Schritten. Meistens wird dabei nur die Serverseite betrachtet. Es wird weder die für die Darstellung und die Interaktion notwendige Clientseite untersucht noch wird die Gesamtperformance der Anwendung gemessen.

Die Analysewerkzeuge verfügen über keine einheitliche Ein- und Ausgabe, was ihre Kombinierbarkeit erschwert und eine Erweiterung der Test-Szenarien verkompliziert. Erschwerend kommt hinzu, dass viele Analysewerkzeuge erst im Produktivbetrieb eingesetzt werden können und damit die Performance-Untersuchung nach der Entwicklung der Anwendung stattfindet.

Ein weiteres Defizit der betrachteten Tools besteht darin, dass keine Benutzereingaben simuliert und die damit ausgelösten Verarbeitungszyklen untersucht werden. Dabei können so erzeugte Daten Auswirkungen auf die Performance der Anwendung haben und eine Optimierung notwendig machen.


2.2 Ziel

Im Rahmen der Diplomarbeit soll ein voll funktionaler Prototyp entwickelt werden, der es erlaubt, kontinuierliche Performancetests durchzuführen. Damit sollen die Auswirkungen von geändertem Programmcode auf die Gesamtleistung direkt sichtbar werden und eine frühzeitige Refaktorisierung und Leistungsoptimierung erlauben.
2 Einleitung

Der Entwickler soll die Möglichkeit haben, Szenarien zu definieren, die dann dazu verwendet werden, Benutzersitzungen (eng. user session) zu simulieren und die Leistung der Webanwendung und einzelner Komponenten zu ermitteln.

Die Messergebnisse sollen in einer Historie vorgehalten werden, um spätere Auswertungen zu ermöglichen und Tendenzen hinsichtlich der Leistungsentwicklung zu erkennen.

2.3 Gliederung

Kapitel 3 (Grundlagen) erklärt notwendige Grundlagen und Begriffe. Es werden nicht nur unterschiedliche Arten der Leistungsanalyse, sondern auch Methoden zur Messung der Performance vorgestellt. Im weiteren Verlauf werden psychologische und ökonomische Gründe für die Notwendigkeit der Performanceoptimierung von Webanwendungen dargelegt.


Kapitel 5 (Evaluierung) beschäftigt sich mit der Bewertung des Prototypen. Hierzu werden kontinuierliche Performance-Tests und Referenzmessungen zur Bestimmung der Messstabilität auf Basis eines quelloffenen Projekts simuliert und die Ergebnisse vorgestellt und diskutiert.

Kapitel 6 (Fazit) präsentiert die Ergebnisse der Diplomarbeit.

Kapitel 7 (Ausblick) gibt einen Überblick über zukünftige Arbeiten und weiterführende Fragestellungen, die sich im Zuge dieser Arbeit ergeben haben.

Kapitel 8 (Zusammenfassung) stellt einen Auszug der Arbeit und der Resultate dar.

Anhang A (Entwurf) umfasst Kopien der während der Entwicklung des Prototypen erstellten Paper Prototypes.

Anhang B (Evaluierung) beinhaltet alle Szenarien, die zur kontinuierlichen Performanceanalyse des quellofenen Projekts Spree verwendet wurden. Zusätzlich sind alle Messergebnisse in tabellarischer und grafischer Form angeführt.

Anhang C (Datenträger) enthält auf dem beigefügten Datenträger den Quelltext des Prototypen Continuous Performance Testing, alle Skripte und Messdaten.
3 Grundlagen


Im letzten Teil des Kapitels werden die Begriffe \textit{Agile Softwareentwicklung} und \textit{kontinuierliche Integration} erläutert.

3.1 Was ist Performance?

Bevor man sich mit der Thematik der Performancemessung befasst, gilt es erst einmal zu klären, wie \textit{Performance} überhaupt definiert ist. Das englische Substantiv hat gemäß \textit{New Oxford English Dictionary} zwei Bedeutungen: „an act of staging or presenting a play, concert, or other form of entertainment“ und „the capabilities of a machine or product, esp. when observed under particular conditions“.


Die \textit{erste Klasse} beschreibt Performance als die Fähigkeiten eines Computer. Dabei handelt es sich um eine Menge von Aktivitäten, die durchgeführt werden können, aber auch die Korrektheit der durchgeführten Operationen und deren Ergebnisse. Die Ergonomie, also die Benutzerfreundlichkeit (eng. usability) des Computersystems wird ebenso zu dieser Klasse gezählt.

Die **dritte Klasse** der Performance geht der Frage nach, wie schnell die Aufgaben durch das System ausgeführt werden. Zwei Aspekte werden dabei unterschieden: Zum einen geht es um die Zeit, die benötigt wird ein Ergebnis zu liefern; zum anderen geht es um die Anzahl von Aufgaben und Operationen, die innerhalb einer bestimmten Zeit ausgeführt werden können.


In der Arbeit „Object driven performance testing of Web applications“ [SS00] definieren Subraya et al. die Performanceziele in Abhängigkeit von der Zielgruppe. Darin schreiben sie, dass der Benutzer einer Webanwendung eine andere Vorstellung von Performance hat als der zuständige Webmaster: ersterer interessiert sich vor allem für schnelle Antwortzeiten und eine generelle Abwicklung eines gewünschten Vorgangs durch die Webanwendung; letzterer richtet seinen Fokus auf einen hohen Durchsatz und eine hohe Verfügbarkeit.


- **Erweiterbarkeit** bzw. Offenheit erlaubt das Hinzufügen von Erweiterungen durch Dritte.
- **Sicherheit** meint die Fähigkeit des Systems, sich gegen unerlaubte Zugriffe zu schützen. Eine Sicherheitslücke ist ein Defekt und hat damit Einfluss auf die Performance.
- **Flexibilität** ist die Fähigkeit des Systems, in anderen Umgebungen betrieben zu werden.
- **Zuverlässigkeit** meint die Hochverfügbarkeit des Systems und dessen Robustheit gegenüber starker Belastung und sicherer Abschaltung im Fehlerfall.
- **Funktionalität** ist der Umfang der Fähigkeiten eines Systems.
- **Nutzbarkeit** kann sowohl als Effizienz als auch Benutzerfreundlichkeit verstanden werden.
- **Konnektivität** ist die Fähigkeit des Systems, mit weiteren Systemen zu kommunizieren, um externe Dienste in Anspruch zu nehmen oder selber Dienste anzubieten.
- **Datenschutz** ist die Fähigkeit des Systems, den Umfang der Informationen über sich selbst zu kontrollieren, die nach Außen hin zugänglich gemacht werden.
Die übermäßige Gewichtung eines der acht aufgezählten Ziele und die Vernachlässigung der übrigen kann zu einem im Ganzen nicht performanten System führen und damit den Erfolg eines Systems verhindern, so die Autoren. Ferner kann die Anpassung an *einem* der Ziele negative Auswirkungen auf die *anderen* Performance-Ziele haben.

Nachdem der Begriff Performance geklärt wurde, beschäftigt sich das folgende Kapitel mit der Frage, welche Umstände die Leistung einer Webanwendung beeinflussen.

### 3.2 Welche Faktoren beeinflussen die Performance?

Die Suche nach möglichen Ursachen für die Beeinflussung der Performance bedarf einer Betrachtung des Ablaufs einer Webanwendung, dem *Request Cycle*. Der *Request Cycle* ist in weiten Teilen immer gleich. Es ist nebensächlich, ob die Anwendung gerade erst gestartet wurde oder bereits längere Zeit ausgeführt wird.


Hieraus lassen sich zwei beziehungsweise drei Hauptkomponenten extrahieren, die an der Ausführung einer Webanwendung beteiligt sind und daher einer genaueren Betrachtung bedürfen. Auf der einen Seite befindet sich der Client, mit dem die Webanwendung gesteuert wird. Meistens handelt es sich hierbei um den Webbrowser des Benutzers. Auf der anderen Seite befindet sich der Webserver³, welcher die Anfrage aarbeitet und daher seinen Teil zur Gesamtleistung beiträgt. Die dritte Komponente bildet das Netzwerk (Inter- oder Intranet). Es verbindet den Client mit dem Server und überträgt die Daten zwischen den beiden Endpunkten.

Performance-Engpässe können ihren Ursprung in allen drei Komponenten haben, weswegen diese drei Teile nun genauer untersucht werden.

² Ajax steht für „Asynchronous JavaScript and XML“ und ist ein asynchron erfolgender Datenaustausch zwischen Browser und Server. Auch wenn es der Name suggeriert, müssen die Daten nicht zwangsläufig im XML-Format vorliegen.
³ In der Regel handelt es sich dabei um viele Webserver, die durch einen Serverlastverteiler (eng. loadbalancer) mit der Abarbeitung der Daten betraut werden und wiederum die Dienste weiterer Server in Anspruch nehmen.
3 Grundlagen

3.2.1 Clientseite

Der Benutzer interagiert mit der Webanwendung über eine im Browser dargestellte Webseite. Aus diesem Grund definiert sich die Performance aus der Sicht eben dieses Benutzers als die Ladezeit der Seite. Die Ladezeit ist die Spanne, die verstreicht, während die vom Benutzer ausgelöste Anfrage durch den Webserver abgearbeitet und das Resultat an den Client übertragen wird. Die Anzahl der Requests, welche von einem Server pro Sekunde verarbeitet werden können, spielen aus Sicht des Benutzers keine, zumindest aber nur eine theoretische Rolle (wenngleich die Verarbeitungsrate serverseitig zur Gesamtperformance beiträgt), da der Benutzer nur seine eigene Anfrage wahrmimmt.

S. Souders beschäftigte sich während seiner Arbeit bei Yahoo! mit den Ursachen verzögterer Ladezeiten von Webseiten, der Front-End-seitigen Performance. In seinem Artikel zu diesem Thema misst er dem Front-End die größte Bedeutung bei. Im Rahmen seiner Untersuchungen ist S. Souders zu dem Schluss gekommen, dass der überwiegende Teil der gesamten Ladezeit einer Webseite im Webbrowser verbraucht wird (nahezu 90 Prozent), während HTML-Anweisungen im weitesten Sinne interpretiert und ausgeführt werden [Sou08].


<table>
<thead>
<tr>
<th>Webseite</th>
<th>Leerer Cache</th>
<th>voller Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.aol.com">www.aol.com</a></td>
<td>3%</td>
<td>3%</td>
</tr>
<tr>
<td><a href="http://www.ebay.com">www.ebay.com</a></td>
<td>5%</td>
<td>19%</td>
</tr>
<tr>
<td><a href="http://www.facebook.com">www.facebook.com</a></td>
<td>5%</td>
<td>19%</td>
</tr>
<tr>
<td><a href="http://www.google.com/search?q=flowers">www.google.com/search?q=flowers</a></td>
<td>53%</td>
<td>100%</td>
</tr>
<tr>
<td>search.live.com/results.aspx?q=flowers</td>
<td>33%</td>
<td>100%</td>
</tr>
<tr>
<td><a href="http://www.msn.com">www.msn.com</a></td>
<td>2%</td>
<td>6%</td>
</tr>
<tr>
<td><a href="http://www.myspace.com">www.myspace.com</a></td>
<td>2%</td>
<td>2%</td>
</tr>
<tr>
<td>en.wikipedia.org/wiki/Flowers</td>
<td>6%</td>
<td>9%</td>
</tr>
<tr>
<td><a href="http://www.yahoo.com">www.yahoo.com</a></td>
<td>3%</td>
<td>4%</td>
</tr>
<tr>
<td><a href="http://www.youtube.com">www.youtube.com</a></td>
<td>2%</td>
<td>3%</td>
</tr>
</tbody>
</table>

Tabelle 1: Prozentualer Anteil der Ladezeit verursacht durch das Back-End. Tabelle entnommen aus [Sou08].
Das Resultat der Forschung bilden S. Souders’ 14 vielbeachtete Regeln zur Performance-
optimierung von Webseiten [Sou07]. Ihre Einhaltung verkürzt die Ladezeiten nachhaltig
und steigert die Performance. Aus diesen Regeln lassen sich die Gründe für suboptimale
Front-End-seitige Performance extrahieren.

**Zu viele HTTP Requests.** Jedes in einer Seite eingebettete Element muss erst bei
einem Webserver angefordert und anschließend an den Webbrowser ausgeliefert werden.
Dabei fallen Wartezeiten an. Bei mehr als vier Elementen dominiert dieser Mehraufwand
die Ladezeit zu über 50 Prozent (vgl. [Kin08]). Die Ursache hierfür liegt darin, dass
gemäß der HTTP/1.1 Spezifikation der Browser nur zwei gleichzeitige Verbindungen
tau und dem selben Webserver haben kann. Müssen viele Elemente vom gleichen
Webserver geladen werden, führt dies zu entsprechenden Verzögerungen.

Gleichzeitig wirkt sich der Overhead von TCP/IP⁴ negativ auf diese Ladezeit aus, da bei
zahlreichen Elementen viele Verbindungen zum Server geöffnet werden müssen. TCP/IP
führt viele *Round Trips*⁵ durch, ehe eine Verbindung⁶ zu stande kommt. Zusätzliche
Verzögerungen sind die Folge.

In [NBK02] schreiben Nahum et al., dass die Verbindungsverwaltung von TCP nicht
dafür konzipiert wurde, viele Daten zwischen Client und Server zu übertragen, da dabei
mehr Pakete ausgetauscht werden als „semantisch notwendig“.

**Kein Verfallsdatum bei Elementen.** Kann der Browser eingebettete Elemente aus
seinem Cache laden, reduziert sich die Ladezeit. Um jedoch entscheiden zu können, ob
die zwischengespeicherte Kopie eines Elements verwendet werden kann, muss der Web-
browser Informationen über die Gültigkeit der lokalen Kopie besitzen. Wurde beim erst-
maligen Anfordern des Elements kein Verfallsdatum mitgesendet, muss der Webbrowser
bezüglich der Validität Auskunft beim Webserv er einholen. Diese kann wieder zu nicht
unerheblichen Wartezeiten führen (vgl. erster Punkt).

**Nicht reduzierte Datenmengen.** Die zu übertragene Datenmenge trägt im hohen
Maße zur Ladezeit einer Seite bei. Trotz immer schnelleren Internetzugängen kommt es
to Verzögerungen bei der Übertragung größerer Datenmengen (vgl. hierzu das *Fat File
Paradox* weiter unten). Dabei lassen sich viele Bestandteile einer Webseite um bis zu 70
Prozent reduzieren, wenn man sie komprimiert überträgt. Betroffen sind nicht nur die
eigentlichen HTML-Seiten, sondern auch textbasierte Skripte, Stylesheets, XML- und
JSON⁷-Daten.

---

⁴TCP/IP ist ein im Internet häuﬁg verwendetes Protokoll.
⁵Mit Round Trip ist gemeint, dass viele Datenpakete hin und her übertragen werden.
⁶TCP/IP führt zum Verbindungsaufbau einen sog. Dreieck-Handshake durch: Der Client sendet an
den Server ein SYN-Paket. Der Server bestätigt den Verbindungswunsch mit einem SYN-ACK, den
er zurück an den Client sendet. Der Empfang wird wiederum vom Client mit einem ACK-Paket
bestätigt. Erst jetzt ist eine Verbindung zwischen Client und Server aufgebaut.
⁷JSON steht für *JavaScript Object Notation* - ein Datenformat, das von Menschen und Maschinen
einfach gelesen werden kann.
Ungünstige Einbindung von Stylesheets und JavaScript. Werden die Stylesheets an einer ungünstigen Stelle eingebunden, so ist dem Browser beim Darstellen der Seite nicht klar, wie einzelne Elemente abgebildet werden sollen. Diese Ungewissheit hält an, bis die Stylesheets geladen und ausgewertet wurden. Solange sie noch nicht geladen sind, verhalten sich die Browser unterschiedlich: Beim Internet Explorer bleibt die Seite weiß, bis die Styles bekannt sind; Firefox zeichnet die Seite und verändert nach Bekanntwerden der Stylesheets das Aussehen der Elemente. Beide Verhaltensweisen können den Benutzer irritieren. Denn im ersten Fall hat der Benutzer den Eindruck, dass die Seite sehr lange zum Laden braucht. Im zweiten Fall kommt es zu einem Flackern und ggf. dem nachträglichen Ausblenden von bereits dargestellten Elementen, was durch den Benutzer als Defekt interpretiert werden könnte.


Zu viele DNS-Anfragen. Sind die Elemente einer Webseite auf unterschiedlichen Servern (mit anderen Hostnamen) verteilt, muss der Webbrowser all diese unbekannten Adressen erst einmal mittels Domain Name System (DNS) nachschlagen. Dieses Nachschlagen kostet wiederum Zeit und wirkt sich daher negativ auf die Gesamtperformance aus.


---

8Gemäß einer Untersuchung von Websiteoptimization.com machen Skripte 22 Prozent und Stylesheets 15 Prozent einer Webseite aus. Vgl. hierzu [LLC08]
9Um einen Hostnamen bei einem DNS-Server nachzuschlagen, muss jedesmal eine Verbindung zum entsprechenden Server aufgebaut werden, wenn die zugehörige IP-Adresse dem lokalen DNS-Server noch nicht bekannt ist. Wegen der entfernungsabhängigen Verzögerung (vgl. Tabelle 2) im Internet kann eine solche Anfrage bis zu 100ms dauern [Sou08]
3 Grundlagen

<table>
<thead>
<tr>
<th>Host</th>
<th>Entfernung</th>
<th>Dauer</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.fu-berlin.de">www.fu-berlin.de</a></td>
<td>&lt; 30 km</td>
<td>ca. 28 ms</td>
</tr>
<tr>
<td><a href="http://www.rwth-aachen.de">www.rwth-aachen.de</a></td>
<td>540 km</td>
<td>ca. 34 ms</td>
</tr>
<tr>
<td><a href="http://www.msu.ru">www.msu.ru</a></td>
<td>&gt; 1.500 km</td>
<td>ca. 82 ms</td>
</tr>
<tr>
<td><a href="http://www.ucla.edu">www.ucla.edu</a></td>
<td>&gt; 9.000 km</td>
<td>ca. 185 ms</td>
</tr>
<tr>
<td><a href="http://www.usyd.edu.au">www.usyd.edu.au</a></td>
<td>&gt; 16.000 km</td>
<td>ca. 344 ms</td>
</tr>
</tbody>
</table>

**Tabelle 2:** Zeit für Routing zu einem entfernten Host (hin und zurück).

Die Messung erfolgte von einem Client in Berlin, der über das Deutsche Forschungsnetz (DFN) an das Internet angebunden war.

Damit ähneln das Verhalten der Webanwendung mehr einem lokal ausgeführten Programm, was sogar soweit geht, dass Webanwendungen auch offline, also ohne eine Verbindung zum Webserver betrieben werden können (vgl. [Str]).

In [Res08] untersuchte J. Resig die Performance unterschiedlicher JavaScript-Engines gängiger Webbrowser. Hierzu verwendete der Autor die drei JavaScript-Benchmark-Umgebungen SunSpider\(^ {10} \), V8 Benchmark\(^ {11} \) und Dromaeo\(^ {12} \). Das Resultat seiner Untersuchung verdeutlicht signifikante Unterschiede zwischen den verschiedenen JavaScript-Engines: Bei Dromaeo, dem Benchmark, das aus Sicht von J. Resing wegen der vielen DOM- und JavaScript-Bibliothek-Tests am ehesten das Verhalten eines Browsers unter normalen Umständen simuliert, war die langsamste Engine\(^ {13} \) etwa 60 Prozent langsamer als die schnellste\(^ {14} \). Aber auch die beiden anderen Benchmarks zeigen teils gravierende Unterschiede in der Performance.


### 3.2.2 Serverseite


---

\(^{10}\)SunSpider: [http://www2.webkit.org/perf/sunspider-0.9/sunspider.html](http://www2.webkit.org/perf/sunspider-0.9/sunspider.html)

\(^{11}\)V8 Benchmark: [http://code.google.com/apis/v8/run.html](http://code.google.com/apis/v8/run.html)

\(^{12}\)Dromaeo: [http://v2.dromaeo.com/](http://v2.dromaeo.com/)

\(^{13}\)Die *Futhark*-Engine wird in Opera 9.5 und neuer verwendet.

\(^{14}\)Die *JavaScriptCore*-Engine fand ihre Verwendung im Safari-Browser bis zur Version 3.1
Eine genauere Betrachtung der serverseitigen Faktoren und ihrer Wirkung auf die Performance ist daher angebracht. Es lassen sich drei Bereiche erkennen, die getrennt inspiert werden: Die Softwareseite, die Hardwareseite und die Architektur der Infrastruktur.

Software. Auf der Softwareseite spielen die Implementierung der Anwendung, die verwendeten Technologien sowie das Betriebssystem eine wichtige Rolle und haben direkten Einfluss auf die Verarbeitungsgeschwindigkeit [GW09]. Ineffiziente Algorithmen können unnötig viele Ressourcen beanspruchen (beispielsweise CPU-Zyklen und Speicher), die dann für andere Aufgaben nicht mehr zur Verfügung stehen und so den Durchsatz begrenzen. Die Folge ist eine erhöhte Antwortzeit.

Der unbedachte Einsatz von Techniken, die eine objektrelationale Abbildung erlauben, stellen einen weiteren möglichen Verursacher schlechter Leistung dar. Das Problem liegt in der fehlenden Transparenz bei der Konstruktion der eigentlichen Datenbank-Abfragen und der Ungewissheit über die Menge der gelesenen Daten [Kil08]. Beides kann Kosten in nicht unerheblichen Ausmaß erzeugen und sich so negativ auf die Performance der Webanwendung auswirken.

In ihrer Arbeit zu Performance-Problemen bei Webservern ([NBK02]) schreiben Nahum et al., dass „[...] Webserver die meiste Zeit im Kernel verbringen“. Damit meinen sie die ineffiziente Art des Betriebssystems, mit der Abarbeitung typischer HTTP-Transaktionen zu verfahren. Ferner schreiben sie, dass speziell optimierte Betriebssysteme die Performance der Webserver signifikant steigern.

Weitere Merkmale des Betriebssystem limitieren die Leistung. Die Anzahl gleichzeitiger Threads und die Anzahl offener Sockets ist beschränkt. Im Endergebnis kann es passieren, dass der Server keine weiteren Anfragen verarbeiten kann und der Client dies als Fehlermeldung registriert.

Hardware. Dass die Hardware des Servers großen Einfluss auf die Performance hat, ist weitgehend klar. Doch welche Komponenten und deren Eigenschaften sind zu untersuchen? Die Serverhardware lässt sich auf den Prozessor, den Arbeitsspeicher, die Festplatte und das Netzwerkgerät abstrahlen. Auch wenn diese Teile heutzutage vergleichsweise schnell, groß und preiswert sind, können sie durch eine zu hohe Anfragelast sprichwörtlich in die Knie gezwungen werden. In [Sch06] verdeutlicht T. Schlossnagle, wie es dazu kommt:

\[\text{Der Durchsatz (eng. throughput) ist die Anzahl der Aufgaben, die ein System pro Zeiteinheit verarbeiten kann.}\]
\[\text{Antwortzeit (eng. response time) gibt an, wie schnell ein System auf Eingaben reagiert.}\]
\[\text{ORM (eng. object-relational mapping) erlaubt einer objektorientierten Programmersprache das Ablegen der Objekte in einer relationalen Datenbank. Zwei bekannte Beispiele solcher OR-Mapper sind Hibernate für Java oder ActiveRecord für Ruby.}\]

15Der Durchsatz (eng. throughput) ist die Anzahl der Aufgaben, die ein System pro Zeiteinheit verarbeiten kann.
16Antwortzeit (eng. response time) gibt an, wie schnell ein System auf Eingaben reagiert.
17ORM (eng. object-relational mapping) erlaubt einer objektorientierten Programmersprache das Ablegen der Objekte in einer relationalen Datenbank. Zwei bekannte Beispiele solcher OR-Mapper sind Hibernate für Java oder ActiveRecord für Ruby.
Ein Prozessorkern\textsuperscript{18} arbeitet eine Aufgabe (eng. task) nach der anderen ab. Die Zeitablaufsteuerung (eng. process scheduling) des Betriebssystems regelt dabei die Dauer einer Arbeitseinheit und die Umschaltung auf eine neue Aufgabe (sog. Context Switch). Der Context Switch findet selbst dann statt, wenn die gegenwärtige Aufgabe noch nicht vollständig ausgeführt wurde. Die Umschaltung kostet ihrerseits Zeit (einige Mikrosekunden). Da der Prozessorkern nur eine bestimmte Anzahl an Aufgaben pro Zeiteinheit abarbeiten kann, steigt mit der Last auch die Antwortzeit.


Die Festplatte wirkt sich, gemäß Schlossnagle in \cite{Sch06}, zusätzlich negativ aus, wenn viele datenintensive Operationen durchgeführt werden müssen, zum Beispiel bei vielen Datenbankzugriffen. Sie kann die Lese- und Schreiboperationen nicht schnell genug arbeiten\textsuperscript{20} und die Operationswarteschlange der Festplatte füllt sich nach und nach. Prozesse müssen nun länger bei Eingabe- und Ausgabeoperationen (I/O) warten und die Performance sinkt.

I/O-Operationen werden auch beim Zugriff auf die Netzwerkhardware durchgeführt. Müssen zu viele Daten übertragen werden und wächst gleichzeitig die Dichte der Anfragen durch zu viele Clients, so kann die maximale Bandbreite erschöpft werden und damit nicht nur die Prozesse ausbremsen, sondern auch die Antwortzeit erhöhen.

\textbf{Architektur der Infrastruktur}. Ob eine Webanwendung ausfallsicher und skalierbar ist, um auch beim Versagen einzelner Komponenten oder einem plötzlichen Anstieg der Arbeitsbelastung performant\textsuperscript{21} zu arbeiten, entscheidet die Architektur der Infrastruktur. Eine Webanwendung wird in der Regel auf mehreren Servern ausgeführt. Dies dient zum einen der Arbeitsteilung (Programmcode verteilt ausführen). Zum anderen werden Dienste, wie (verteilte) Datenbank- und Proxyserver oder Lastverteiler auf dedizierten Hosts installiert (vgl. \cite{Sch06}).

\textsuperscript{18}Moderne Prozessoren haben aktuell bis zu acht Kerne. Vgl. IBM „Cell“, Sun „UltraSPARC“, Intel „Dunnington“ oder AMD „Instanbul“.

\textsuperscript{19}Vergleiche: Speicherhierarchie in \cite{OV06}.

\textsuperscript{20}S. Souders \cite{SS00} spricht hier auch von Resource Parametern und meint die verantwortlichen inneren Werte, wie die Disk Seek Time, die Transferraten u.ä.

\textsuperscript{21}Vergleiche hierzu die drei Performanceklassen in Kapitel 3.1

Die weiter oben erwähnten software- und hardwareseitigen Ursachen für mangelnde Performance bei steigender Anfragedichte kommen zum Vorschein, wenn bei der Planung der Infrastruktur keine Serverlastverteilung eingeplant, bzw. nicht genug parallele Server bereit gestellt wurden (vgl. [Sch06]). Die Server laufen während einer Anfragespitze an ihren Kapazitätsgrenzen.

Zusammenfassend kann festgestellt werden, dass die Dichte der Anfragen einen kritischen Faktor bildet. Erst mit steigenden Last durch zu viele Clients, wenn also das System an seinen kapazitiven Grenzen gefahren wird, kommt es zu einer Sättigung (engl. saturation) und zu nachlassender Performance des Systems.

Um diesen Problemen rechtzeitig entgegen wirken zu können, ist eine Kapazitätsplanung (*Capacity Planning*) unerlässlich. D. A. Menasce und V. Almeida erklären diese Tätigkeit treffend mit „(To figure out) when future load levels will saturate the system and determining the most cost-effective way of delaying system saturation as much as possible“ [MA01].

### 3.2.3 Netzwerk

Welche Rolle spielt das Netzwerk bei der Performance einer Webanwendung, wo doch die Pakete mit nahezu Lichtgeschwindigkeit übertragen werden?

Das Problem ist unter dem Namen *Fat File Paradox* [Lei08] bekannt und hat die zu Grunde liegenden Netzwerkprotokolle zur Ursache. Eine große Datei benötigt zur Überquerung einer weiten Strecke sehr viel Zeit (bezogen auf die eigentlich sehr hohe Ausbreitungszögerung), weil in einem Netzwerk der Durchsatz eng an die Latenz geknüpft ist.


---

22 Die Ausbreitungszunverzögerung hängt vom Trägermedium ab und liegt bei Glasfaser oder Twisted-Pair-Kabeln im Bereich von $2 \cdot 10^8 \frac{m}{s}$ bis $3 \cdot 10^8 \frac{m}{s}$ [KR08].
werden, bis dieser den Empfang bestätigt. Die eigentlichen Anwendungsdaten werden segmentiert\(^\text{23}\) und als IP-Datengramme übertragen (vgl. [Hol02]).

Wegen der obligatorischen Empfangsbestätigung durch PAR sinkt der Datendurchsatz mit steigender Netzwerklatenz, da keine neuen Datenpakete an den Empfänger übertragen werden, solange der fehlerfreie Empfang nicht bestätigt wurde. Werden ausserdem Paketverluste erkannt, so reduziert das Protokoll zusätzlich die Anzahl der gesendeten Daten.

Da sich die Paketverluste mit zunehmender Streckenlänge häufen (vgl. [Lei08]), hat die Entfernung des Servers zum Client direkten Einfluss auf den Datendurchsatz und damit die Dauer der Übertragung.


In der Vergangenheit wurde hauptsächlich in den Ausbau der First und der Last Mile investiert, was zu 20- bzw. 50-fachen Steigerung der Kapazitäten in der letzten Dekade geführt hat. Die Middle Mile wurde gleichzeitig als „Niemandsland“ angesehen; hier wird weit weniger verdient und der Ausbau somit stark vernachlässigt. Folglich ist die Middle Mile häufig überlastet und für den Paketverlust\(^\text{24}\) verantwortlich, so T. Leighton.

War vor fünf bis zehn Jahren die Kapazität des Middle Mile noch ausreichend, weil langsame Einwahlverbindungen\(^\text{25}\) (eng. dial-up) den eigentlichen Flaschenhals bildeten, so hat sich das mit dem Auftreten von Technologien wie DSL oder UMTS stark gewandelt. Im vergangenen Jahr verfügten laut [Lei08] 59 Prozent der Internet-Nutzer des Dienstanbieters Akamai Technologies über eine Breitbandverbindung\(^\text{26}\). Die OECD schätzt, dass global betrachtet im Jahr 2008 21,3 Prozent über einen Breitbandzugang zum Internet verfügten ([OEC08]).

---

\(^{23}\) Ein Segment ist höchstens 64KB groß.

\(^{24}\) Paketverluste treten immer dann auf, wenn, vereinfacht gesagt, mehr Daten an einem Knotenpunkt eintreffen als verarbeitet werden können. Dadurch werden die Pufferspeicher dieser Knoten schnell vollständig gefüllt und neu eintreffende Daten mangels Zwischenspeicher verworfen.

\(^{25}\) Üblich waren Verbindungsraten von 9,6 bis 56 Kilobit pro Sekunde bei analoger und 64 bis 128 Kilobit pro Sekunde bei digitaler (ISDN) Einwahlverbindung.

\(^{26}\) Die United States Federal Communications Commission (FCC) klassifizierte den Begriff „Breitbandanbindung“. Dabei steht „Basic Broadband“ für eine Bandbreite zwischen 768 Kbps und 1,5 Mbps; „Broadband“ steht für eine Bandbreite zwischen 1,5 Mbps und 100 Mbps (und mehr).
Beim Blick auf die Wachstumszahlen bei Breitbandzugängen der OECD wird der Trend zu immer größerer Verbreitung der schnellen Anbindungen und der damit verbundenen Steigerung der Verkehrslast deutlich. Der Flaschenhals verschiebt sich also hin zum *Middle Mile* und wird von T. Leighton in seinem Artikel passend als „Distance Bottleneck“ bezeichnet.

### 3.3 Warum ist Performance wichtig?

Performanceoptimierung ist ein aufwendiger Prozess und es bleibt die Frage, ob und warum sich hier eine Investition überhaupt lohnt. Spielt es wirklich eine Rolle, wenn man nach einer langwierigen Optimierungsarbeit zu Besserungen im Bereich von 500 Millisekunden kommt?

Vorweg sei gesagt, dass Untersuchungen aus der näheren Vergangenheit eine Investition in diesem Bereich mehr als rechtfertigen, wie der Leser im weiteren Verlauf erkennen wird. N. Bhatti et al. schreiben ganz treffend „Benutzer glauben, dass wenn die Performance schlecht ist, die Sicherheit der Seite gefährdet sein könnte“ und „[...] mangelnde Performance zum Verlust der Kunden führt“ [BBK00].

Im Folgenden werden einige Untersuchungen vorgestellt, die genau dieser Frage nachgegangen sind und erstaunliche Ergebnisse zu Tage brachten. Beleuchtet werden dabei sowohl psychologische als auch ökonomische Aspekte.

#### 3.3.1 Psychologische Gründe

Bereits vor über vierzig Jahren befasste sich R. Miller mit der Frage nach nutzbaren Antwortzeiten während einer Interaktion von Menschen und Computern [Mil68]. Er verglich diese Wechselwirkung mit der Kommunikation zwischen zwei Menschen und untersuchte die dabei auftretende Erwartungshaltung.

R. Miller stellte heraus, dass der Mensch innerhalb von zwei Sekunden von einem Gegenüber eine Antwort oder Reaktion erwartet, wobei diese nicht unbedingt dem Kontext der tatsächlichen Anfrage entsprechen muss. Verstreicht eine Zeit von vier Sekunden und mehr, so führt dies zu einem Abriss der Kommunikation, vergleichbar mit einer plötzlich gekappten Telefonleitung.

empfunden werden. Der Mensch empfindet hingegen Störungen vor dem Erreichen eines Closure als frustrierend, da damit eine Kontinuität im Gedankenfluss verhindert wird.

Ein anschauliches Beispiel ist das Lesen eines Buches. Das übergeordnete Ziel besteht darin, das vollständige Buch zu lesen (wenn man davon absieht, dass ja oft der Weg das eigentliche Ziel sein kann). Die Subroutinen bestehen aus dem Lesen der einzelnen Kapitel. Hat der Leser ein solches Kapitel gelesen (Closure), kann er das Buch weglassen und am nächsten Tag weiter lesen. Wird der Leser jedoch innerhalb eines Kapitels unterbrochen, wird das als störend empfunden, da er aus dem Gedankenfluss gerissen wird und sich nach der Störung wieder hineinfinden muss.


\[27\text{Die Kapazität des Kurzzeitgedächtnisses wird durch die }\text{Miller'sehe Zahl} \text{ ausgedrückt. Diese nach George A. Miller benannte Zahl besagt, dass der Mensch } 7 \pm 2 \text{ Informationseinheiten (sog. } \text{Chunks}) \text{ disponibel halten kann} \text{ [Mil56].} \]

\[28\text{Die Gegenwartsdauer ist die Zeit, in der mehrere Stimuli als gegenwärtig oder augenblicklich realisiert werden.} \]

\[29\text{Ramsay et al. definiert die Antwortzeit als die Zeitpanne zwischen dem Auslösen einer Aktivität bis zum Beginn der Darstellung der Ergebnisse durch das Computersystem.} \]

\[30\text{In beiden Fällen wurde die Abtastbarkeit und die Relevanz der Seite ohne eine Ladeverzögerung höher bewertet.} \]

Zwar gaben die Teilnehmer der Studie von N. Bhatti et al. in Interviews an, dass sie durchaus auch in der Netzwerkanbindung Ursachen für größere Verzögerungen vermuten. Während der Experimente hingegen wurden für unzureichende Ladezeiten einzig die Seitenbetreiber in der Verantwortung gesehen – trotz der zuvor gemachten Aussagen im Interview [BBK00]. Viel gravierender ist diese Missdeutung bezogen auf die Meinung der Kunden über ein Produkt und das Unternehmen, denn der negative Affekt wurde direkt auf beide projiziert, wie N. Bhatti et al. in Befragungen erfuhren. Auf die ökonomischen Auswirkungen wird im Kapitel 3.3.2 ausführlicher eingegangen.


Beider „Zone of Satisfaction“ führt eine Ladezeit von zehn Sekunden und weniger und zwei bis drei nützlichen Elementen zu einer positiven Erfahrung beim Benutzer. In der „Zone of Tolerance“ wird der Benutzer auf die Ladezeit überhaupt erst aufmerksam, wenn sie die zehn Sekunden überschreitet. Dieses wird von vielen Benutzern noch geduldet. In der „Zone of Frustration“ befindet sich der Benutzer, wenn die vierfache Zeitspanne der ersten Zone verstrichen ist. Die Verzögerung war zu lang und er ist frustriert.

In ihrer Arbeit gehen N. Bhatti et al. davon aus, dass Benutzer eine bestimmte Vorstellung davon haben, wie lange eine Aufgabe dauern wird, wobei die Benutzer hier zwischen „High-tolerance Tasks“ und „Low-tolerance-tasks“ Unterscheiden.

---

31 Zone Research nennt in diesem Zusammenhang eine psychologische Verteilung von 25:50:25, bei der 25% besonders ungeduldig sind, 50% eine normale Geduld zeigen und 25% besonders tolerant sind [Zon01].


Zudem haben N. Bhatti et al. beobachtet, dass es Unterschiede hinsichtlich der Duldung von Ladezeiten gibt. Unterschieden wird dabei anhand der Darstellung der Seite verwendeten Verfahrens. Bei nicht-inkrementell\textsuperscript{34} dargestellten Seiten war lediglich eine Ladezeit von höchstens fünf Sekunden noch zufriedenstellend. Bereits ab 11 Sekunden wurde der Ladevorgang als unzureichend bewertet. Bei inkrementell\textsuperscript{35} geladenen Seiten wurden Ladezeiten von bis zu 39 Sekunden noch als positiv bewertet und erst ab 56 Sekunden als unzureichend deklariert (vgl. [BBK00]).


N. Bhatti et al. merken allerdings an, dass wegen der bereits in die Verrichtung investierten Zeit die Toleranz durch die Aussicht auf das baldige Abschließen des Vorgangs erneut steigt. Dennoch bleibt die Frustration nicht folgenlos: In den Kommentaren, die während der Experimente aufgezeichnet wurden, gaben die Benutzer an, künftig die Seite meiden zu wollen.

Welche wirtschaftlichen Konsequenzen die Unzufriedenheit des Benutzers in Folge unzulungreicher Ladezeiten haben kann, wird im folgenden Kapitel dargestellt.

### 3.3.2 Ökonomische Gründe

Im vorhergehenden Kapitel konnte oft vom Frust beim Benutzer eines Computersystems gelesen werden, der sich aufbaut oder steigert, wenn die Antwortzeiten bestimmte Werte überschritten haben. Dieser Frust führte bei den Teilnehmern der Untersuchungen zu negativen Assoziationen mit dem Inhalt einer Seite, den angebotenen Produkten oder dem Betreiber der Seite, bis hin zu einer Verweigerung in Form des Abbruchs einer Tätigkeit oder der zukünftigen Meidung einer Seite.


\textsuperscript{34}Die Seite erscheint erst, wenn sie vollständig geladen wurde.

\textsuperscript{35}Seite baut sich nach und nach auf und der Benutzer kann schon frühzeitig erste Elemente sehen.

\textsuperscript{36}Beispielsweise das Abschicken einer Bestellung, wo zuerst das Login, die Adresse, dann die Zahlungsart sowie die Bestätigung eingegeben werden muss.


Bei der Beharrlichkeit spielt der Studien zur Folge die Verfügbarkeit eines Produktes eine wichtige Rolle. Gab es viele vergleichbar teure Alternativen zu einem Produkt, so war die Beharrlichkeit eher gering. Waren die Produkte hingegen nur schwer zu finden, fiel die Beharrlichkeit insgesamt höher aus und der Kunde war Störungen gegenüber toleranter.

Die Toleranz steigt auch mit der Wichtigkeit der Transaktion. Bei Bankgeschäften liegt diese entsprechend hoch. Dies wird zum einen dadurch bedingt, dass der Kunde hier vom Anfang an eine andere Erwartung hat (vgl. High-tolerance Task in Kapitel 3.3.1). Auf der anderen Seite ist der Umzug eines Kontos zu einer anderen Bank mit mehr Aufwand für den Benutzer verbunden. Zona Research nennt das in der Untersuchung „pain level created by transferring one’s account“.

Aus diesen Variablen konstruierte Zona Research eine Formel zur Berechnung der Abbruchquote (Abandonment), wobei diese aus der Antwortzeit (Response Time) geteilt durch die Geduld (Patience) des Benutzers gebildet wird.

\[ Abandonment = TransmissionErrors + \frac{(ResponseTime)}{(Patience)} \]

Obwohl die Übertragungsfehler (Transmission Errors) nicht im Einflussbereich des Betreibers liegen (vgl. Kapitel 3.2.3), wirken sie sich direkt auf die Abbruchquote der Kunden aus, da sie, wie bereits von N. Bhatti et al. in der Untersuchung weiter oben bestätigt, subjektiv in den Verantwortungsbereich des Seitenbetreibers gerückt werden. Laut einer von Zona Research zitierten Untersuchung (dem Keynote Retail Industry Benchmark) wurden durchschnittlich 7,9 Prozent Übertragungsfehler ermittelt, was zu Einnahmeverlusten von 3,9 Milliarden geführt haben könnte.

Die Geduld (Patience) eines Benutzers hängt laut Zona Research wiederum von vier Faktoren ab, wie man in der nachfolgenden Formel sehen kann.

\[ \text{Zum Vergleich: Laut einer Studie der Gesellschaft für Konsumforschung aus dem Jahr 2009 wurden in der Bundesrepublik Deutschland im Jahr 2008 per Internet Waren und Dienstleistungen in Höhe von 13,6 Milliarden Euro in Anspruch genommen. Ausgehend von 29,5 Millionen Online-Käufern in der Bundesrepublik ergibt sich ein Schnitt von 461 Euro pro Kunde [GfK09].} \]
3 Grundlagen

Patience = (8seconds) · (Tenacity) · (Importance) · (naturalpatience)

Die Beharrlichkeit (Tenacity) meint, dass Benutzer mit langsameren Anbindungen an das Internet (beispielsweise mit mobilen Zugängen) generell bereit sind, etwas länger zu warten, da sie sich der Einschränkungen durch ihre Internetanbindung bewusst sind. Die Wichtigkeit (Importance) spiegelt die Signifikanz einer Transaktion wider. Handelt es sich beispielsweise um absolut notwendige Aktienverkäufe oder um Produkte, die der Kunde unbedingt haben will, ist die Wichtigkeit entsprechend höher angesetzt.

Die natürliche Geduld (Natural Patience) korreliert mit einer Acht-Sekunden-Abbruch-Regel, wobei erneut die psychologische Verteilung angewandt wird: Das ungeduldige Viertel wartet acht Sekunden, die normal und besonders geduldigen 50 bzw. 25 Prozent der Anwender warten auch 12 bzw. 16 Sekunden, ehe sie die Transaktion abbrechen.

Unter Verwendung dieser Formeln bezifferte Zona Research die etwaig entgangenen Einnahmen wie folgt: Fast vier Milliarden Dollar entgingen demnach allein durch Übertragungsfehler und ganze 21 Milliarden Dollar durch langsamen Seitenaufbau. Der kumulierten Frustration wurden immerhin 315 Millionen Dollar zugeschrieben (vgl. [Zon01]).


Zusammenfassend kann an dieser Stelle ein Zitat von D. Farber, dem Chefredakteur von zdnet, verwendet werden. Darin heißt es „For Google, or any other Web site, speed means people viewing more pages in same visit and spending more time overall online... and more money [Far06]“.

Die Ladezeit bietet sich aus den beschriebenen Gründen an, als Indikator für die Performance einer Webanwendung herangezogen zu werden.

38Zona Research geht nicht weiter auf den Ursprung dieser Regel ein.
3.4 Wie wird Performance gemessen?


3.4.1 Unterschiedliche Arten der Evaluation


In der **Simulation** wird ein vereinfachtes Modell des Systems und seiner Benutzer entwickelt. Die Simulation kann sowohl in Echtzeit als auch in Zeitlupe und Zeitraffer vollzogen werden. Letztendlich hat dies jedoch keinen Einfluss auf die Ergebnisse, postulierte Dirlewanger. Während der Ausführung werden die Daten aufgezeichnet und, wie schon zuvor, am Ende ausgewertet.


Die Resultate der drei Evaluationen weisen gravierende Unterschiede auf. Einzig bei der Messung wird ein echtes System untersucht und dessen Performance ermittelt. Bei der Simulation und bei der Modellierung wird jeweils nur ein vereinfachtes Modell geprüft. Die Ergebnisse der beiden letzten Performancemessungen sind daher nur Abschätzungen, die allerdings auch Vorhersagen des zukünftigen Verhaltens erlauben (vgl. [Dir06]).


Im weiteren Verlauf der vorliegenden Diplomarbeit ist eine Evaluierung der Performance von Systemen im Sinne der oben beschriebenen Messung eines echten Systems geplant.

\[39\] Vergleiche hierzu [Rob00]
3.4.2 Benchmark vs. Load-Testing

Bei der Performancemessung muss zwischen Benchmarking und dem Load-Testing differenziert werden, da beide jeweils unterschiedliche Aufgaben haben.

**Benchmarking** dient der Messung der Performance durch wohl definierte und standardisierte Workload-Modelle (siehe weiter unten im Text). Die messbaren Performance-Ziele sind auf unterschiedlichen Systemen wiederholbar und erlauben so den Vergleich verschiedener Systeme untereinander hinsichtlich ihrer Leistungsfähigkeit (vgl. [RSSP]).


**Load-Tests** erlauben die Einsicht in die tägliche Arbeit einer Webanwendung, wobei das Verhalten echter Benutzer modelliert und durch spezielle Skripte simuliert wird. Die Anzahl der gleichzeitigen Benutzer orientiert sich dabei an Werten des Realbetriebs, die während der Hauptzeit (eng. peak) auftreten können. Die meisten Aktionen werden dabei innerhalb weniger Sekunden durchgeführt und beinhalten eine Bedenkzeit (*user think time*), um die Simulation realistischer zu gestalten. Ein Test dauert in der Regel nicht länger als 10 bis 20 Minuten.

Eine spezielle Variante der Load-Tests ist der Stress-Test. Es gibt keine künstliche Bedenkzeit und die Skripte werden mit der maximalen Geschwindigkeit abgespielt. Mit dem so erstellten Worst Case Szenario sollen Performanceprobleme aufgedeckt werden, die im normalen Betrieb nicht aufgefallen wären. Die Belastung kann immer weiter gesteigert werden, um damit zu überprüfen, ob die Webanwendung zusammenbricht oder der Beanspruchung standhält (vgl. [SS00]).

Werden Load- oder Stress-Tests über einen langen Zeitraum gefahren (Stunden bis Wochen), so besteht ihre Aufgabe darin, unbekannte Defekte zu diagnostizieren (bspw. Speicherlöcher). Diese Ausdauertests dienen der Evaluation der Zuverlässigkeit einer Webanwendung auf längere Sicht (vgl. [SS00]).
3.4.3 Workload


Spätestens beim Begriff Rechenaufgaben assoziiert man jene Aufgaben, die ein Schüler in der Schule lösen muss. Hier wird auch ersichtlich, wie man einen Workload zur Messung der Performance oder Leistung verwenden kann: Muss ein Schüler eine bestimmte Anzahl von Rechenaufgaben in einer vorgegebenen Zeit lösen, kann der Lehrer die Leistungsfähigkeit eines Schülers messen und sie in Relation mit anderen Schülern setzen, die in der gleichen Zeit die gleichen Aufgaben lösen mussten.

Auf ähnliche Weise lässt sich durch die Verwendung des Workload die Performance von Computersystemen messen und vergleichen. Systeme werden mit einer definierten Last beansprucht und die Zeit, die sie zum vollständigen Abarbeiten benötigen, oder aber die Anzahl der bearbeiteten TeilAufgaben pro Zeiteinheit liefern ein Bewertungskriterium.


Es sei dem Leser überlassen, diese Überlegungen auf ein schulisches Workload zur Leistungsbeurteilung der Schüler (auch bekannt als „Klausur“) anzuwenden.

3.4.4 Metriken

Um beim Schulbeispiel zu bleiben, kann der Leistungsparameter als die Anzahl erreichter Leistungspunkte beim Abitur angesehen werden und ihre Abbildung in Nennwerte als Überführung in die entsprechenden verbalen Noten „ungenügend“ bis „sehr gut“.

3.4.5 Testumgebung

Gemäß Ceri et al. [CBF03] sollte der Performancetest auf der tatsächlichen Hardware unter Verwendung der endgültigen Software- und Hardwareinfrastruktur durchgeführt werden. Nach Möglichkeit sollte die echte Anwendung evaluiert werden. Da diese am Anfang der Entwicklung noch nicht (vollständig) vorliegt, empfehlen Ceri et al. die Verwendung eines Prototypen, der nur einen Teil der funktionalen Funktionen enthält.


In [Men02] beschreibt Menascé den RTE etwas genauer. Dieser besteht aus einem Load Generator, der das Verhalten der Webbrowser nachahmt und kontinuierlich Anfragen an den Webserver stellt. Der Workload, den der RTE bzw. Load Generator erzeugt, wird mittels zuvor erstellter Skripte generiert, welche den Ablauf einer Benutzersitzung (eng. user session) abbilden. Der Load Generator kann dabei einige Hundert bis Tausend

40Geläufige Namen sind, neben RTE und Load Generator, auch Stressclient (dt. Belastungsklient) bzw. Masterclient (vgl. [RSSP])
Benutzer emulieren, in dem er mehrere virtuelle Benutzer (eng. virtual user) steuert. Er fungiert dann als Master client.


3.5 Vorhandene Werkzeuge zur Performancemessung

Dass ein Bewusstsein für die Problematik der Performance von Webanwendungen durchaus vorhanden ist, erkennt man spätestens an der Fülle von Werkzeugen, die zur Analyse verwendet werden können. In diesem Abschnitt wird eine kleine Auswahl der Werkzeuge vorgestellt und nach Anwendungsbereich gruppiert.

In die Kategorie Webserver fallen Analysezwerkzeuge, die sich im Speziellen mit der Leistungsfähigkeit des Servers befassen. Zu den bekanntesten Vertretern gehören sicherlich httpperf, Apache Flood und OpenWebLoad.

Mit httpperf 42 lassen sich HTTP-Anfragen in einer bestimmten Häufigkeit generieren, um damit den Durchsatz des Webservers zu messen. Durch eine Steigerung der Anfrage-Rate kann man zusätzlich die Kapazitätsgrenze des Webservers ermitteln.

Apache Flood 43 ist eine profilgesteuerte Anwendung, die mit XML-Dateien konfiguriert werden kann und einen hohen Webtraffic erzeugt. Zum Testen dynamischer Inhalte lassen sich Variablen setzen und auslesen.

Mit der Messung der Antwortzeiten und Transaktionen pro Sekunde befasst sich das Programm OpenWebLoad 44. Dabei wird durch simultane Anfragen an eine vorher definierte Serveradresse ein Belastungs- test durchgeführt.

Zur Untersuchung der Ladezeit von Webseiten existieren die beiden Onlinedienste OctaGate SiteTimer und pingdom Full Page Test sowie clientseitig Yslow und Jiffy-Web.

41 Wegen des Umfangs der anfallenden Messdaten schlagen Anderson et al. in [AL90] vor, diese Daten zu filtern und nur die für die Optimierung relevanten zu übernehmen.
42Httpperf: http://www.hpl.hp.com/research/linux/httperf/
43Apache Flood: http://httpd.apache.org/test/flood/
44OpenWebLoad: http://openwebload.sourceforge.net/
OctaGate SiteTimer und pingdom Full Page Test (letzter nach Angaben der Entwickler stark vom ersteren inspiriert) bieten webbasierte Lösungen für die Analyse des Ladevorgangs einer Webseite. Hierzu wird die komplette Seite inklusive aller darin eingebetteten Elemente (Bilder, Stylesheets, Flash, JavaScript etc.) abgerufen und Auskunft über die Anzahl der Requests, die Größe der Elemente sowie die Antwort- und Ladezeiten nach Elementtyp gegeben.


Auf der Anwendungsseite existieren für das Framework Ruby on Rails diverse Programme, die durch die Auswertung der Log-Dateien der Anwendung im Produktivbetrieb langsame Actions des Controller identifizieren. Einige dieser Werkzeuge sind LogJuicer, Production-Log-Analyzer und Request-Log-Analyzer.


Die Speicherung und Auswertung der Ergebnisse über einen längeren Zeitraum (idealerweise über die gesamte Entwicklungszeit) ist bei den Programmen nicht vorgesehen, was das Erkennen von Tendenzen und entsprechendes Eingreifen in den Entwicklungsprozess verhindert.

---

45 OctaGate SiteTimer: http://www.octagate.com/service/SiteTimer/
46 pingdom Full Page Test: http://tools.pingdom.com/
47 Yslow: http://developer.yahoo.com/yslow/
48 Jiffy-Web: http://code.google.com/p/jiffy-web/
49 Gemeint sind hiermit die Programmmethoden des Controllers einer MVC-basierten Ruby on Rails-Anwendung. MVC steht für Model-View-Controller und ist einArchitekturmuster der Softwaredtechnik, bei dem das Datenmodell, die Präsentation und die Programmsteuerung voneinander getrennt werden (siehe Abbildung 12).
50 LogJuicer: http://logjuicer.org/
51 Production-Log-Analyzer: http://rails-analyzer.rubyforge.org/
3 Grundlagen

3.6 Agile Softwareentwicklung

Die Agile Softwareentwicklung wurde konzipiert, um auf die geänderten Vorgaben des Marktes zu reagieren. Software soll innovativ und von hoher Qualität sein. Sie soll die Anforderungen des Marktes erfüllen und dabei zügig fertig gestellt werden (vgl. [HC01]).

Die klassischen Methoden zur Softwareentwicklung haben sich als zu unflexibel erwiesen. Man versuchte im klassischen Vorgehen durch umfangreiche und ausgeklügelte Pläne die Anforderungen vollständig zu definieren, um damit die durch spätere Veränderungen anfallenden Kosten zu reduzieren. Gemäß Barry Boehm steigen die Kosten für Änderungen \( \text{cost of change} \) im Laufe der Entwicklung immer höher (vgl. Kapitel 4).

Statt also zu versuchen, die Änderungen möglichst zu vermeiden und damit Kosten zu sparen, geht die Agile Softwareentwicklung den entgegengesetzten Weg: Sie versucht den Entwicklungsprozess so zu verändern, dass die Kosten für Veränderungen reduziert werden (vgl. [HC01]).

Um das zu erreichen wird der Entwurf nicht mehr in einer langen dem Projekt vorangestellten Planungsphase erstellt. Vielmehr wird der Entwurf als eine stetige Tätigkeit angesehen und im Rahmen kurzer Iterationen angefertigt und erweitert.

Agile Teams bestehen aus kleinen Gruppen (bis zu zehn Entwickler [Sub05]) nahe bei einander, um so den Informationsaustausch zu vereinfachen. Innerhalb einer Iteration werden Leistungsmerkmale aus einer To-Do-Liste ausgewählt und implementiert. Notwendige Änderungen werden frühzeitig erkannt und durch Refaktorisierungen umgesetzt.

Während dieser kurzen Iterationen, die zwei bis sechs Wochen dauern können, wird lauffähiger Code implementiert und in häufigen Abständen dem Kunden präsentiert, um so frühzeitig eine Rückkopplung zu erhalten und auf Änderungswünsche schnell reagieren zu können.

Dem Kunden wird die Möglichkeit gegeben, spezielle Leistungsmerkmale der Software zu priorisieren. Er wird gleichzeitig als Mitglied der Entwicklungsgruppe angesehen und die daraus resultierende Nähe hilft Unklarheiten zu beseitigen, die im Laufe der Implementierung auftreten. Dazu ist die häufige Anwesenheit, zumindest aber die Erreichbarkeit des Kunden notwendig. Cockburn und Highsmith schreiben, dass man nicht von einem agilen Team sprechen kann, wenn die Rückmeldung vom Kunden und dem Management bloß alle sechs Monate geschieht („A team isn’t agile, if the feedback loop with customers and management is six months“ [HC01]).

Nur so lässt sich sicherstellen, dass der Kunde am Ende ein Produkt erhält, dass er wirklich braucht und nicht eines, das er in dieser Form vor einem oder zwei Jahren gebraucht hätte.

Die Prinzipien der Agilen Softwareentwicklung gehen über die hier vorgestellten Konzepte hinaus und können der zahlreichen Literatur und dem Agilen Manifest entnommen werden, das der Agilen Softwareentwicklung zu Grunde liegt. Die vom Autor getroffene Auswahl beschränkt sich auf die für diese Arbeit wesentlichen Aspekte.

3.6.1 Kontinuierliche Integration


Wird am Ende der Arbeit der geänderte Code in die Revisionsverwaltung übertragen, muss sichergestellt werden, dass alle Komponenten des Gesamtsystems weiterhin gemäß der Spezifikation miteinander funktionieren.

Dazu wird die Anwendung auf Grundlage des aktuellen Programmcodes übersetzt (eng. to compile), getestet und Programmpakete für die jeweiligen Laufzeitumgebungen erstellt und installiert. Oft finden bei so erstellten Instanzen weitere, von der Laufzeitumgebung abhängige Tests statt.

Kommt es während dieser Tests zu einem Fehler, so muss der Entwickler den Defekt, der die fehlerfreie Integration verhindert, beheben. V. Subramaniam und A. Hunt schreiben, dass häufiges und frühes Integrieren den Arbeitsaufwand reduziert, der durch die Suche nach und Korrektur von Ursachen missglückter Integration hervorgerufen wird. Wird die Integration hingegen hinausgeschoben, so die beiden Autoren weiter, resultiert das in signifikanten und weitreichenden Codeänderungen, Verzögerungen und generellem Durcheinander. Sie empfehlen daher, die Integration früh und häufig zu vollziehen [Sub05].

53 Manifesto for Agile Software Development: http://agilemanifesto.org/
54 Stets lauffähig meint, dass der in den Hauptzweig der Versionsverwaltung übertragene Code lauffähig ist.
Es existieren viele kommerzielle und freie Frameworks, die eine Revisionsverwaltung auf neue Revisionen hin untersuchen und zum gegebenen Zeitpunkt automatische Integrationstest durchführen. Der im weiteren Verlauf dieser Arbeit erstellte Prototyp bedient sich des in der Programmiersprache Ruby geschriebenen CruiseControl.rb.\(^{55}\)

### 3.7 Rückblick

Die Performance eines Computersystems kann sowohl für seine Fähigkeiten, Stabilität und Korrektheit als auch die Ausführungsgeschwindigkeit stehen. Sie kann auch als ein Netz konträrer Attribute verstanden werden, die erst in ein Gleichgewicht gebracht werden müssen.

Wird die Leistung als die für den Benutzer relevante Ladezeit der Seiten einer Webanwendung verstanden, können Ursachen mangelnder Performance an drei Stellen hervorgerufen werden: Auf der mit der Darstellung betrauten Clientseite, in der die Anwendung ausführenden Serverseite und dem Netzwerk, das die beiden Endpunkte verbindet.


Der wirtschaftliche Schaden, der durch die Unzufriedenheit der Anwender entstehen könnte, lässt sich durch Performancetests im Vorfeld auf ein Minimum reduzieren. Dazu werden Benutzersitzungen simuliert und Leistungsanalysen erstellt, welche die Notwendigkeit von Optimierungen aufzeigen.

Hohe Entwicklungskosten in Folge zu spät erfolgter Veränderungen lassen sich durch Methoden der Agile Softwareentwicklung vermeiden. Sie sehen stetige Optimierungen und Modifikationen vor und helfen die dabei entstehenden Kosten konstant zu halten. Ein zur kontinuierlichen Performancemessung notwendiger Prototyp wird im folgenden Kapitel vorgestellt.

---

4 Das ContinuousPerformanceTesting Framework

In den vorhergehenden Kapiteln wurden neben Ursachen mangelnder Performance bei Webanwendungen deren Auswirkungen beleuchtet. Hierbei wurden sowohl finanzielle Folgen als auch die psychologischen Ursachen und Konsequenzen der aus schlechter Performance resultierenden Unzufriedenheit auf Seiten des Endnutzers aufgezeigt und wie diese letztendlich miteinander gekoppelt sind.

Es steht außer Frage, dass die Optimierung der Performance einen wichtigen Teil in der Qualitätssicherung bildet. Werden die in den vorangegangenen Kapiteln genannten Grenzwerte von der getesteten Software nicht eingehalten (vgl. Kapitel 3.3.1) führt dies zwar zu keinem Versagen im Sinne der funktionalen Anforderung. Die nicht-funktionalen Anforderungen werden jedoch nicht erfüllt, so dass der ursächliche Defekt repariert werden muss.

Das Beseitigen von Defekten sollte möglichst frühzeitig erfolgen, da die Kosten für Änderungen mit fortschreitender Zeit exponentiell steigen (vgl. Abbildung 2). Dies setzt ein entsprechendes Werkzeug voraus, welches schon zu Beginn Defekte aufzuzeigen vermag. Damit die Kosten der Beseitigung nicht-funktionaler Defekte nachhaltig gemildert werden, schlägt Highsmith [Hig02] ein gutes Risikomanagement und entsprechende Entwicklungspraktiken vor. In seinem Buch zitiert Highsmith die Verfechter der agilen Methoden, indem er sagt „The right strategy [...] is to work on reducing the cost of change throughout the project by developing simple solutions [...], refactoring [...] and constant testing [...].“


31
Quelle der Abbildung: [And03]

Quelle der Abbildung: [And03]
4.1 Vorüberlegungen

„Der Anfang ist die Hälfte des Ganzen“ - (Aristoteles). Der Anfang ist in diesem Fall die Erörterung der Anforderungen für das zu erstellende Framework, seine Funktionen, das Datenmodell und die verwendeten Technologien. Diese werden hier vorgestellt ehe im Kapitel 4.5 ausführlicher auf die Impelementierung eingegangen wird.


Vorhandene Werkzeuge zur kontinuierlichen Integration (beispielsweise das freie Cruise-Control.rb) sind in der Lage, mehrere Softwareprojekte zu verwalten und die dazugehörigen Versionsverwaltungssysteme zu überwachen. Da die Tests nach jedem Eenchecken des Codes durchgeführt werden, dient die jeweilige Revisionsnummer zur Identifizierung der durchgeführten Tests. Es bietet sich daher an, auch beim kontinuierlichen Performancetest die Revisionsnummer als eindeutige Kennung zu verwenden und mehrere Projekte zu unterstützen.


Im Anschluss an den erfolgreich durchgeführten Integrationstest muss auf einem dedizierten Server die Webanwendung automatisiert gestartet und mit Testdaten initialisiert werden. Diese Arbeit kann durch das Deployment-Werkzeug Capistrano durchgeführt werden, das die getestete Revision auf dem SUT (System unter Test) installiert und startet.

56 Mit Übersetzung ist das Kompilieren der Quellen gemeint, das als Resultat zu einem lauffähigen Programm oder einer Bibliothek führt.
57 Die Revisionsnummer ist eine eindeutige Kennung und wird bei einigen Versionsverwaltungen in Form einer fortlaufenden Nummer gebildet (wie z. B. bei Subversion) oder durch einen längeren Hashwert über den eingecheckten Code, die Kommentare etc. erzeugt (z. B. bei Git und Mercurial).
Bei Performancetests wird das SUT durch viele simulierte Benutzer gestresst, wobei mit Skripten echtes Benutzerverhalten (Benutzersitzungen) abstrahiert wird (vgl. Kapitel 3.4.5). Dies setzt eine Möglichkeit zur Definition von Schritten voraus, welche dann die virtuellen Benutzer ausgeführt werden. Einzelne Schritte werden dabei zu einem umfangreichen Szenario gebündelt und können komplexere Businessprozesse darstellen. Das Cucumber-Werkzeug [cuc] bietet die Möglichkeit zur Beschreibung solcher Szenarien für Akzeptanztests und kann entsprechend als Vorbild dienen oder erweitert werden.

Während eines Tests werden, wie bereits im Kapitel 3 gefordert, viele Benutzer simuliert, so dass diese Anforderung entsprechende Beachtung finden muss. Dabei muss die Gesamtmenge dieser Stressclients durch eine Kontrollinstanz (Masterclient) überwacht und gesteuert werden. Ferner müssen die virtuellen Benutzer in der Lage sein, unterschiedliche Szenarien auszuführen und die entsprechenden Skripte vorhalten. Da sich im Laufe der Produktentwicklung diese Szenarien verändern können, sollten sie zentral abgelegt werden, von wo die Softwareagenen sie dann abrufen können.


Die Performancetests werden für ein bestimmtes Projekt durchgeführt, wobei mit jeder neuen Revision eine Analyse stattfindet. Da die einzelnen Software-Komponenten der getesteten Anwendung durch die entsprechend definierten Szenarien abgedeckt werden, müssen jeweils Messungen pro Szenario durchgeführt und ausgewertet werden. Für das Datenmodell ergibt sich hieraus eine Datenabhängigkeit zum Projekt, zur Revision und zum Szenario.

Wegen der Fülle an angefallenen Messdaten wurde von Anderson et al. in [AL90] die Aufbereitung und Filterung der Daten empfohlen (vgl. Kapitel 3.4.5). Yu et al. verwenden in ihrer Arbeit als Methoden zur Analyse der empirischen Daten das arithmetische

\[\text{Unter der Voraussetzung, dass das SUT und die Stressclients im gleichen Netzwerk bzw. Rechenzentrum ausgeführt werden.}\]
Das ContinuousPerformanceTesting Framework

Mittel sowie die Minimal- und Maximalwerte. Weil das arithmetische Mittel (Durchschnitt) anfällig für Ausreißer\textsuperscript{60} ist (vgl. [Kre05]) und keiner der Werte etwas über die eigentliche Verteilung aller Messwerte aussagt, sollte zur Glättung der Median\textsuperscript{61}, für die Verteilung das erste und dritte Quartil\textsuperscript{62} gespeichert werden.

Abschließend sollte die Ausgabe der Daten dem Anwender einen schnellen Überblick ermöglichen, es aber auch erlauben, Einzelereignisse zu betrachten, um auf diesem Wege Performanceprobleme identifizieren und lokaliseren zu können. Ähnlich der Darstellung im CruiseControl.rb sollten sowohl alle Projekte und deren Performance aufgelistet werden als auch die Performancemessungen einzelner Revisionen. Da bei jedem Test einer neuen Revision viele unterschiedliche Szenarien durchgetestet werden, sollte die Darstellung der Leistung einzelner Szenarien erfolgen, um damit genauer eine problematische Komponente eingrenzen zu können.

Gemäß H. Botthof wird durch die visuelle Darstellung die Komplexität unübersichtlicher Daten reduziert und das Erkennen von Zusammenhängen erleichtert [BHR08]. Aus diesem Grund offeriert sich hier der Gebrauch von Diagrammen. Um im Voraus Tendenzen erkennen zu können, bietet sich die Darstellung der gemessenen Performance über die Zeit als Liniendiagramm an. Bei der Betrachtung von Einzelmessungen einer Revision kann der Einsatz von Box-Plots auf einfache Weise Auskunft über die genaue Verteilung der Messwerte gewährleisten. Im Anhang A befinden sich die Papierprototypen des GUI, die während der Planungsphase entstanden sind.

Das folgende Kapitel stellt das basierend auf den Vorüberlegungen erstellte Framework zur kontinuierlichen Performancemessung sowie dessen Funktionsweise und seine Bestandteile vor.

### 4.2 Kurzfassung der Funktionsweise

Das Framework zur kontinuierlichen Performancemessung besteht aus mehreren Komponenten (Abbildung 4). Das Kernstück und der für den Anwender sichtbare Teil ist der Server. Er enthält eine Historie bereits durchgeführter Messungen der erfassten Projekte. Diese Messergebnisse werden in einer Weboberfläche gruppiert nach Projekt, Build...

\textsuperscript{60}Ein Ausreißer ist nach der Definition von U. Krengel ein Wert, der sich relativ weit weg von den anderen Werten befindet und bei dem eine Störung als Ursache für sein Zustandekommen vermutet wird. Ein solcher Wert kann das Ergebnis verfälschen und wird daher entfernt [Kre05].

\textsuperscript{61}Der Median ist der Wert, der bei einer sortierten Menge von Werten genau in der Mitte aller Werte liegt. Er wird auch als Zentralwert bezeichnet.

\textsuperscript{62}Das erste bzw. dritte Quartil ist der Wert unterhalb dessen sich 25 Prozent bzw. 75 Prozent der Werte kleiner gleich dem Quartil befinden. Das zweite Quartil wird auch als Median bezeichnet. So gesehen ist das erste Quartil der Median der Werte unterhalb des eigentlichen echten Medians einer Verteilung. Quartile sind ausreißerrobuste Streuungsmaße (vgl. [Hen08]).

und Szenario dargestellt und erlauben dem Entwickler einen schnellen Überblick und die genaue Analyse bestimmter Szenarien und Builds.

Ein Build ist hierbei auf einer bestimmten Revision aus dem Versionsverwaltungssystem basierende lauffähige Ausprägung der zu entwickelnden Software. Ein Szenario enthält eine Abfolge von Schritten, welche Benutzereingaben simulieren und damit einen an die Realität angelehnten Test der Webanwendung durchführen (vgl. Kapitel 3.4.3).


Neben diesen Komponenten sorgt ein Zusatzmodul beim Hilfsprogramm zur kontinuierlichen Integration dafür, dass nach einem erfolgreich durchgeführten Integrationstest...
4 Das ContinuousPerformanceTesting Framework

der Server über eine neu eingetroffene Revision informiert wird und einen Performancetest initiieren kann. Darüber hinaus werden die Szenarien-Definitionen auf den neuesten Stand gebracht und im Server gespeichert.

4.3 Server

Im Mittelpunkt des Frameworks steht die auf Ruby on Rails basierende Serveranwendung. Sie dient der Speicherung gesammelter Daten und bietet über das Web-GUI eine visuelle Auswertung der durchgeführten Performancetests.

Für den Zugriff auf das Datenmodell und die Daten stehen RESTful-Webservices\(^{63}\) zur Verfügung. Hierüber werden von den Agenten Jobs abgefragt (WorkerTasks), die Konfigurationen zur Durchführung der Tests abgerufen (Scenarios, DefinitionFiles) und die gemessenen Datensätze gespeichert (MeasuredData).


Die Visualisierung der Performancemessungen erfolgt gruppiert nach Projekten, Builds und Szenarien. Auf der Projekt-Seite werden alle Projekte gelistet und gleichzeitig deren Gesamtperformanceverlauf\(^{65}\) in Form eines Liniendiagramms dargestellt (Abbildung 5). Damit wird dem Entwickler die Möglichkeit gegeben, sich einen schnellen Überblick über die Performance aller Projekte zu verschaffen.


---

\(^{63}\)REST steht für Representational State Transfer. Bei RESTful Webservices wird jede Ressource eines Servers über eine eigene eindeutige URI angesprochen. Hierzu werden die HTTP-Methoden \texttt{GET, POST, PUT und DELETE} verwendet, um Ressourcen abzurufen und zu verändern.

\(^{64}\)CruiseControl.rb ist ein in Ruby geschriebenes quelloenes Werkzeug zur Durchführung kontinuierlicher Integrationstests. Nähere unter \url{http://cruisecontrolrb.thoughtworks.com/}

\(^{65}\)Die Performance ist antiproportional zur gemessenen Ladezeit: Je höher die gemessene Ladezeit ist, desto niedriger/schlechter ist die Performance.
4 Das ContinuousPerformanceTesting Framework

Abbildung 5: Auflistung aller Projekte und deren Gesamtperformance.

Abbildung 6: Ausgewähltes Projekt und dessen zugehörige Builds.
Abbildung 7: Detaillierte Ansicht eines *Builds*.

Abbildung 8: Detaillierte Ansicht eines *Builds* mit Szenariodefinition.
jeweils ausgewählten fünf Builds präsentiert wird. Auf diese Weise kann gezielt ein nicht performantes Build identifiziert und seine Details eingeblendet werden.


Wurde ein Szenario identifiziert, dessen Messwerte auf ein Problem deuten, so kann der Anwender durch den Wechsel in die Szenarioansicht die historische Entwicklung für dieses Szenario anzeigen lassen, um damit die Ergebnisse vergangener Zeiten mit denen der aktuellen Messung vergleichen zu können. Die Darstellung erfolgt nach Messgruppe und stellt die Maximal-, Minimal-, Durchschnitts- und Medianwerte dar. Zusätzlich wird die Definition des Szenarios eingeblendet, um so diejenige Funktionalität der Software identifizieren zu können, welche durch den ausgewählten Test abgedeckt wird (Abbildung 8).


Während einer laufenden Performancenvaluation kann der aktuelle Zustand der Agenten über den Reiter Worker abgefragt werden (Abbildung 10). Dort wird neben dem aktuell untersuchten Szenario auch eine Auflistung der damit befassten Dienende geboten.

Im folgenden Kapitel werden die Implementierung des Prototypen und die darin eingeflossenen Überlegungen dargestellt.

### 4.4 Softwareagenten

Nachfolgend werden die verteilten Agenten vorgestellt. Sie bilden einem Schwarm virtueller Benutzer, die Benutzereingaben simulieren und Messdaten zusammentragen. Währenddessen überwachen Monitoragenten den Zustand des Anwendungsservers.

\textsuperscript{66}Diagramm zur grafischen Darstellung der Verteilung statistischer Daten (siehe [Hen08]).
Abbildung 9: Performanceübersicht aller Szenarien eines Projekts.

Abbildung 10: Aktueller Zustand des Anführer-Agenten und seiner Diener.
4 Das Continuous Performance Testing Framework

4.4.1 Multiagentensystem

Um die eigentliche Arbeit der Agenten zu verstehen, muss erst einmal definiert werden, was ein *Agent* ist, was ein *Multiagentensystem* ausmacht und wo die Vor- und Nachteile eines solchen Systems liegen.

Das Wort *Agent* stammt vom lateinischen *agere*, das für „tun“ oder „treiben“ steht. Es meint demnach einen „Handelnden“ oder, wenn man die Übersetzung aus der englischen Sprache nimmt, den „Bearbeiter“. Sicherlich geläufiger ist der Begriff im Zusammenhang mit Spionagetätigkeiten, wie man sie vom weltbekannten MI6-007-Agenten *James Bond 007* kennt. Auch wenn es sich bei der vorliegenden Diplomarbeit nicht um eine Spionagegeschichte handelt, so ist der Vergleich gar nicht so weit hergeholt, denn die Aufgabe der popkulturellen Agenten liegt in der *Informationsgewinnung* und ggf. Sabotage. Geheimdienstagenten arbeiten *verteilt* auf der ganzen Welt und *kooperieren* ab und an mit anderen Agenten, um so dem eigenen Ziel näher zu kommen.


Ein *Softwareagent* besteht aus einem Netzwerk- und Kommunikationsinterface und verfügt über ein symbolisches Modell anderer Agenten [SB96]. In seiner Abhandlung über verteilte intelligente Agenten nennt K. Sycara einige Merkmale [SPW+96], über die ein Agent verfügen sollte. K. Sycara nach sollte ein Agent fähig sein, Arbeiten auszuführen („taskable“), wobei die Anweisungen hierzu von einem Menschen oder von einem anderen Softwareagenten kommen können. Ein Agent sollte verteilt, selbst organisierend („network-centric“) und in der Lage sein, längere Zeit ohne Aufsicht („persistent“) zusammen mit anderen Menschen oder Softwareagenten zu arbeiten („collaborative“).

Wenden mehrere Softwareagenten lose gekoppelt, um in einer Gruppe „gemeinsam Probleme durch Interaktion und Kollaboration“ [BDXT06] zu lösen, so spricht man von einem *Multiagentensystem*. Systeme dieser Art haben sich in der Qualitätssicherung von Software bereits mehrfach bewährt (vgl. [LP99], [YZX06], [BDXT06], [GK02]) und bieten eine Reihe von Vorteilen, aber auch Nachteilen.

Leung und Poon heben gerade bei der Qualitätssicherung die Reduzierung der Kosten und benötigten Arbeitskräfte, die Unterstützung bei der Erhöhung der Qualität der Software und die standardisierte Datenerhebung in Entwicklungsgeschehen hervor [LP99]. Hinzu kommt, dass Multiagentensysteme in der Regel gut skalieren, da weitere Agenten

---

67MI6, oder Secret Intelligence Service ist der britische Auslandsgeheimdienst.
leicht hinzu und wieder entfernt werden können - ein Aspekt, der beim Entwurf des Frameworks eine wichtige Rolle spielte, wie später noch gezeigt wird.

Nachteilig wirkt sich aus, dass durch die autonomen Agenten die Komplexität steigt, da diese gleichzeitig auf Ressourcen zugreifen können und daher eine Koordinierung unabdingbar wird. Wie diese Probleme beim vorliegenden Prototypen gelöst wurden, wird in den Kapiteln 4.4.2, 4.4.3 und speziell im Kapitel 4.5 erklärt.

4.4.2 Verteilte Arbeiteragenten

Die im Prototypen verwendeten Agenten unterteilen sich in zwei Gruppen, wobei die erste Gruppe, die *verteilten Arbeiter* oder *Distributed Worker*, hybrider Ausprägung sind. Gemeint ist damit, dass der *Distributed Worker* beim Starten noch nicht weiß, ob er der *Master* (bzw. Anführer) einer Gruppe wird oder als *Slave* (dt. Diener) die ihm auferlegte Arbeit verrichten wird.

Die Entscheidung hierüber wird während einer *Nomination-Phase* (dt. Ernennungsphase) getroffen, bei der alle *Distributed Worker* die Führung erst einmal für sich beanspruchen. Hat ein *Distributed Worker* sich gegen die anderen behauptet (Details hierzu im Kapitel 4.5), so wechselt er in den *Diktator-Modus*, um damit zu verhindern, dass ein anderer Arbeiter die Führungsrolle für sich beansprucht, ehe der Anführer mit seiner Arbeit fertig ist. Beendet der Anführer seine Aufgabe erfolgreich oder stirbt vorher, so versuchen die verbliebenen Softwareagenten einen neuen Anführer zu bestimmen, der dann neue Arbeiten koordiniert. „Der Diktator ist tot, lang lebe der Diktator.“


Findet der Anführer neue Aufträge vor, trägt er sich als Bearbeiter ein und präpariert die Aufgabe gemäß der darin enthaltenen Definitionen in Form eines *Szenarios*: das *Szenario* wird mehrfach in einer lokalen Warteschlange abgelegt. Haben sich *Monitoragenten* und *Slave-Agenten* zur Abarbeitung eingefunden, bekommen erstere die Order, mit der serverseitigen Überwachung zu beginnen und letztere die durchzuführende Arbeit anhand des *Szenarios* mitgeteilt (siehe Abbildung 11). Der Anführer lässt nur so viele *Slave-Agenten* zu, wie im *Szenario* angegeben (siehe Listing 1).


---

68In diesem Fall wird ein neuer *Distributed Worker* gestartet, da jeder Agent von einem Prozessüberwachungsdämon mit dem passenden Namen *GOD* ([GOD]) beaufsichtigt wird.
Nachdem die Slave-Agenten vom Master mitgeteilt bekommen haben, welches Szenario sie zu bearbeiten haben, holen sie sich hierfür die genauen Definitionen vom zentralen Server ab. Zu diesen Definitionen zählen die umgangsprachlich verfassten Schritte (siehe Listing 1) und deren Übersetzungen in Maschinencode. Sie werden in lokalen temporären Dateien und Ordern abgelegt und unter Zuhilfenahme von Cucumber\textsuperscript{69} ausgeführt. Die bei dem Lasttest gemessene Ladezeit wird am Ende an den Anführer übertragen.

Sind alle Slave-Agenten mit ihrer Arbeit fertig, stoppt der Anführer die Monitoragenten, wartet deren Messdaten ab und berechnet auf Grundlage aller Messwerte die statistischen Daten\textsuperscript{70} zum aktuellen Szenario, wobei eine Gruppierung nach Messart (Speicherverzehr und Prozessorbeanspruchung auf dem Anwendungs server sowie die clientseitige Ladezeit) stattfindet. Anschließend werden die Daten zum Server übertragen.

Der Vorteil dieser verteilten Arbeitsweise liegt darin, dass immer ein Anführer da ist, der die Arbeiten koordiniert. Fällt ein Anführer aus, übernimmt ein neuer die Koordination. Ausserdem wirkt sich das ganze günstig auf die Skalierung aus, da man jederzeit durch das Hinzufügen oder Entfernen von Computern zusätzliche Agenten hinzunehmen.


\textsuperscript{70}Hierzu zählen Minimal-, Maximal-, Durchschnittswert, Median sowie erstes und drittes Quartil.
Listing 1: Beispielhaftes Cucumber-Szenario

<table>
<thead>
<tr>
<th>Scenario: Total Performance Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>Given 10 stress clients</td>
</tr>
<tr>
<td>And I would like to measure the loadtime</td>
</tr>
<tr>
<td>When I go to projects</td>
</tr>
<tr>
<td>And I'd like to remember the preceding loadtime</td>
</tr>
<tr>
<td>And I follow &quot;Show details...&quot;</td>
</tr>
<tr>
<td>And I'd like to remember the preceding loadtime</td>
</tr>
<tr>
<td>And I follow &quot;Scenario no. 3b18a073&quot;</td>
</tr>
<tr>
<td>And I'd like to remember the preceding loadtime</td>
</tr>
<tr>
<td>And it should load within 5 seconds</td>
</tr>
<tr>
<td>And it should return elapsed loadtime</td>
</tr>
</tbody>
</table>

oder entnehmen kann, ohne dass größere Anpassungen und Konfigurationen notwendig wären.

4.4.3 Monitoragenten

Die devoten Monitoragenten führen ihren „Spitzeldienst“ auf den Anwendungsservern durch. Einmal bei einem Anführer angemeldet, warten sie auf dessen Start- und Stopp-Befehle, um so gezielt den Speicherverbrauch und die Beanspruchung des Prozessors durch die Serverdienste nur während des Lasttests zu messen.

Für den Fall, dass auf dem Anwendungsserver mehrere Serverprozesse für die Abarbeitung der Anfragen zuständig sind (sog. Cluster), werden pro Prozess einzelne Threads (dt. leichtgewichtige Prozesse) des Monitors mit der Überwachung betraut. Den Aufbau entnehmen sie dabei den vorgefundenen Konfigurationsdateien\(^{71}\) des Servers.

Die Ergebnisse der Messungen übermitteln die Agenten an ihren aktuellen Anführer, wenn sie von ihm aufgefordert werden, die Überwachung der Prozesse zu beenden.

4.5 Technische Umsetzung

Der während der Diplomarbeit erstellte Prototyp zur kontinuierlichen Performancemessung basiert auf der Skriptsprache Ruby und dem Framework Ruby on Rails. Ruby ist eine freie\(^{72}\), Mitte der Neunziger Jahre in Japan entwickelte rein objektorientierte\(^{73}\) Programmiersprache. Ihr Erfinder, Yuukihiro Matsumoto, beschreibt die Sprache dabei mit

---

\(^{71}\)Bei der Implementierung des Frameworks wurde durch den Autor der Diplomarbeit eine Einschränkung auf Mongrel als Ruby on Rails-Server vorgenommen. Näheres dazu im Kapitel 4.5.

\(^{72}\)Frei im Sinne von quelloffen und kostenlos.

\(^{73}\)Rein objektorientiert meint, dass wirklich alles ein Objekt ist.
1. Browser sendet Request
2. Interaktion des Controller mit Model
3. Aufruf des View durch Controller
4. View stellt Seite im Browser dar

Abbildung 12: Model-View-Controller Architektur.
Quelle der Abbildung: [THB+06]


\footnote{Ruby: http://www.ruby-lang.org/de/about/}
in einen bestimmten Datenbank-Dialog übernimmt *Ruby On Rails*. Listing 2 zeigt die Migration der Klasse *Builds*, die weiter unten konkretisiert wird.

**Listing 2: Migration der Klasse Build.**

```ruby
class CreateBuilds < ActiveRecord::Migration
  def self.up
    create_table :builds do |t|
      t.string :revision, :unique => true
      t.datetime :date
      t.string :developer
      t.integer :project_id
      t.timestamps
    end
  end

  def self.down
    drop_table :builds
  end
end
```

Durch das Angeben des Spaltennamens *project_id* wird *Active Record* mitgeteilt, dass ein *Build* einen Fremdschlüssel auf *Project* hat. Zusätzlich gibt man in der *Model*-Definition diese Abhängigkeit durch die Schlüsselwörter *belongs_to* und *has_many* an (Listing 3). Damit stehen zwei Instanzen der Klassen *Build* und *Project* jeweils die Methode *project* bzw. *builds* zur Verfügung. Im ersten Fall wird das zugehörige Projekt-Objekt, im zweiten Fall eine Liste aller einem Projekt angehörenden *Builds* geliefert.

**Listing 3: Ausschnitt der Definition der beiden Models Build und Project.**

```ruby
class Build < ActiveRecord::Base
  belongs_to :project
  ...
end

class Project < ActiveRecord::Base
  has_many :builds
  ...
end
```

Eine ähnliche Abbildung erlaubt *Active Resource*. Durch diese Erweiterung werden *RESTful-Models* clientseitig als *Ruby*-Objekte repräsentiert, was einen komfortablen Zugriff auf solche Ressourcen ermöglicht. Zur Verwendung dieser Überführung ist lediglich die Definition eines Stellvertreters (eng. proxy) notwendig, der einzig das URI

---

75Uniform Resource Identifier
4 Das Continuous Performance Testing Framework


Listing 4: Active Resource-Stellvertreter.

```ruby
require 'active_resource'
class Build < ActiveResource::Base
  self.site = 'http://localhost:3000/
end
```


4.5.1 Model

Entsprechend den Vorüberlegungen wurden die Models in mehrere Klassen unterteilt (siehe Abbildung 13). Die übergeordnete Klasse bildet das Project, das die Builds und Scenarios einem bestimmten Projekt zuordnet. Darüberhinaus sind Definitionen und Konfigurationen für das zur Erzeugung der Last verwendete Cucumber vorhanden. Sie werden durch die Klasse DefinitionFiles repräsentiert.


Weil die ermittelten Ladezeiten unterschiedlicher Szenarien stark variieren können, wurde Build zusätzlich mit einer normalisierten Gesamtperformance normalized_total_performance erweitert. Die Normalisierung eines Messwertes erfolgt durch die Division des Messwertes mit der durchschnittlichen Ladezeit eines ausgewählten Scenarios (Listing 5). Die so normalisierten Ladezeiten eines ausgewählten Build werden

76 Das kostenlose Buch „Ruby on Rails 2“ von Hussein Morsy und Tanja Otto erschienen im Galileo Computing Verlag kann unter dem folgenden URL abgerufen werden: http://openbook.galileocomputing.de/ruby_on_rails
77 Die Revisionsnummer kann, je nach verwendeter Versionsverwaltung, eine Zahl oder eine Zeichenkette sein.
78 Die kumulierten Ladezeiten einiger Szenarien sind naturgemäß größer als die nicht kumulierten Ladezeiten.
Abbildung 13: Datenmodell.
Das Con tin u u ms Per for man cPer form c u s Test c Fram e work

aufsummiert und bilden den normalisierten Performance-Indikator für dieses Build (Listing 6). Damit wird erreicht, dass alle Szenarien zum gleichen Teil zur Gesamtperformance beitragen.

Listing 5: Berechnung der durchschnittlichen Ladezeit eines Szenarios.
```ruby
class Scenario < ActiveRecord::Base
  ...
  def all_time_avg
    all_time_avg = 0
    count = 0
    self.measured_datas.find_all_by_generic_group("Loadtime").each do |md|
      count += 1
      all_time_avg += md.average
    end
    all_time_avg = all_time_avg / count
    all_time_avg
  end
end
```

Listing 6: Berechnung der normalisierten Gesamtperformance eines Build.
```ruby
class Build < ActiveRecord::Base
  ...
  def normalized_total_performance
    performance_index = 0.0
    self.measured_datas.find_all_by_generic_group("Loadtime").each do |md|
      performance_index += md.average/md.scenario.all_time_avg
    end
    performance_index
  end
end
```

Das Scenario enthält einen innerhalb des Projekts eindeutigen Namen (name) und die zur Ausführung notwendige in Klartext verfasste Darlegung der durchzuführenden Schritte (step_definition), wie im beispielhaften Listing 1 dargestellt. Die darin definierte Anzahl der Clients und ein optionaler Grenzwert werden in den Feldern client_number und limit beim Export der Szenarien abgelegt.

Die Cucumber-Definitionen werden in Instanzen der Klasse DefinitionFile abgelegt und bestehen aus den Attributen name und content, wobei das erste Attribut den Namen der ursprünglichen Datei enthält und zum späteren Erzeugen temporärer Dateien durch die Agenten verwendet wird. content beinhaltet den Quelltext dieser Datei.

Ein MeasuredData-Objekt enthält je einen Verweis auf die Elternobjekte Build und Scenario. Damit ist eine genaue Zuordnung der Messwerte zu einem Szenario.
und einem Build möglich. Zur Gruppierung der Messwerte ist in MeasuredData ein Gruppenname enthalten (generic_group). Die gemessenen Werte sind in den Attributen minimum, maximum, average, median sowie quartile_one und quartile_three gespeichert. Das Datum der Messung, die Revisionsnummer und der Name des gemessenen Szenarios werden von den übergeordneten Klassen Build und Scenario geerbt (Listing 7).

Listing 7: Die Klasse MeasuredData.

```ruby
class MeasuredData < ActiveRecord::Base
  belongs_to :build
  belongs_to :scenario
  validates_presence_of :minimum
  validates_presence_of :maximum
  validates_presence_of :average
  validates_presence_of :median
  validates_presence_of :generic_group

  def date
    self.build.date
  end

  def revision
    self.build.revision
  end

  def scenario_name
    self.scenario.name
  end
end
```

Die oben erwähnten Definitionen, Konfigurationen und Schritte werden unter Verwendung eines Rake-Skriptes\(^{79}\) vor jedem Integrationstest durch CruiseControl.rb synchronisiert. Dabei wird die RESTful-Schnittstelle des Prototypen verwendet (siehe Kapitel 4.5.3).

Die Arbeitswarteschlange für die Agenten wird durch das Model WorkerTasks realisiert. Objekte dieser Klasse werden automatisch angelegt, wenn ein neues Build kreiert wird. Hierzu wird pro existierendes Szenario in einem Projekt ein WorkerTask-Objekt erzeugt (Zeile 8 bis 10 im Listing 8) und stellt damit einen Arbeitsauftrag dar. Enthalten ist darin die Projekt-Identifizierung (project_id), das auszuführende Szenario (scenario_id) und die Revisionsnummer (revision) des neu angelegten Builds. Wird ein solcher Arbeitsauftrag von einem Master-Agenten ausgewählt, so trägt der

---

\(^{79}\)Rake ist ein in Ruby geschriebenes make-Werkzeug, das zur Ausführung von bestimmten Kommandos verwendet wird.
Agent seinen Namen in das Feld `worker` ein und verhindert damit, dass der Auftrag von einem anderen Master ausgeführt wird.

### Listing 8: Teil der `create` Methode des `BuildController`.
```ruby
def create
  @build = Build.new(params[:build])
  @build.project = @project
  @build.date = Time.now

  respond_to do |format|
    if @build.save
      scenarios = @project.scenarios.all
      scenarios.each{|s| WorkerTask.create :scenario => s,
                                             :revision => @build.revision,
                                             :project_id => s.project_id}
  ...
```

#### 4.5.2 View


Das Darstellen von Box-Plots wird in `flot` nicht unterstützt, so dass ein Helfer implementiert werden musste (Listing 9), der aus gegebenen Kollektionen von Objekten `JavaScript`-Code generiert (Zeile 16 bis 24) und als Resultat ein Box-Plot berechnet und ausgibt. Dazu werden pro Box-Plot Linien unterschiedlicher Stärke übereinander gelegt (Zeilen 5 bis 14), wobei die Start- und Endpunkte über die in den Objekten enthaltenen Quartile, Minimal-, Maximal- und Durchschnittswerte bestimmt werden.

---

81JQuery: [http://jquery.com/](http://jquery.com/)
82Der aktuelle Entwurf des HTML5-Standards bzw. der `Canvas`-Elemente kann unter [http://dev.w3.org/html5/spec/Overview.html#the-canvas-element](http://dev.w3.org/html5/spec/Overview.html#the-canvas-element) eingesehen werden.
4 Das Continuous Performance Testing Framework

Listing 9: Helfer zur Generierung von Box-Plots.

def box_plot(div_id, collection)
    data = []
ticks = []
    pass = 0
    collection.each do |md|
        data << {:data => [[md.minimum,pass],[md.maximum,pass]],
                  :lines => {:show => true}, :points => {:show => true}, :color => '#3465a4'}
        data << {:data => [[md.quartile_one,pass],[md.quartile_three,pass]],
                  :lines => {:show => true, :lineWidth => 15}, :color => '#ef5ad'}
        data << {:data => [[md.median,pass],[md.median,pass]],
                  :points => {:show => true, :radius => 5}, :color => '#cc0000'}
        ticks[pass] = [pass, md.generic_group.to_s]
        pass += 1
    end

    script = <<EOF
        <!--[if IE]>
        <script language="javascript" type="text/javascript" src="/javascripts/
            excanvas.pack.js" />
        <![endif]-->
        <script language="javascript" type="text/javascript" src="/javascripts/
            jquery.flot.pack.js" />
        <script type="text/javascript">
            $.plot($('##{div_id}'), #{data.to_json}, {yaxis: {min: -0.5, max: #{pass}
                #3465a4'}})
        </script>
    EOF
    end

Obwohl zum jetzigen Zeitpunkt nur einige Messdaten erfasst werden (Prozessorlast, Speicherverbrauch und Ladezeit), wurde von Anfang an die Möglichkeit der späteren Erweiterung vorgesehen. Zu diesem Zweck werden alle vorhandenen Messdaten (MeasuredData) dynamisch anhand der Messgruppe (generic_group) zusammengestellt und anschließend ausgegeben.

4.5.3 Controller

Die Controller erlauben die Interaktion mit dem System, indem sie die Anfragen der Benutzer (menschlicher und nicht-menschlicher) bearbeiten. Im hier vorgestellten Prototypen steht vor jedem Model ein Controller. Je nachdem, ob ein menschlicher Benutzer oder ein Stück Software auf die Anwendung zugreift, wird entweder eine HTML- oder
eine XML-Repräsentation\textsuperscript{84} der Daten gewählt. Über die Controller wird ein \textit{CRUD}\textsuperscript{85}-Zugriff zur Verfügung gestellt, was letztendlich den einzelnen Bestandteilen des Frameworks die Manipulation der Daten ermöglicht.


Da die beiden \textit{Models Build} und \textit{Scenario} sog. verschachtelte Ressourcen (eng. nested resource) sind, sie also nur in Verbindung mit einer ihnen übergeordneten Ressource angesprochen werden können, mussten die Controller dahingehend erweitert werden, dass vor dem Aufruf der Controller-Methoden ein Filter (sog. \textit{before filter}) das übergeordnete Objekt lädt (Listing 10).


Im Laufe der Zeit kann es viele darzustellende Objekte geben. Viele Daten müssen gelesen und visualisiert werden, was die Übersichtlichkeit einschränkt und die Ladezeit unnötig verlängert. Damit das nicht passiert, werden nicht alle Projekte, \textit{Builds} und \textit{Scenarios} angezeigt, sondern nur eine kleine Auswahl. Um das zu ermöglichen, wird die \textit{Ruby On Rails}-Erweiterung \textit{will\_paginate}\textsuperscript{86} verwendet. Sie bietet mit ihrer Methode \texttt{paginate} einen Ersatz für das gebräuchliche \textit{Active Record}-\texttt{find} und liefert nur Objekte eines selektierten Bereichs. Ein Helfer für die darstellenden \textit{Views} generiert gleichzeitig die notwendige Navigation.

\textsuperscript{84}Selbstverständlich kann auch ein menschlicher Benutzer die XML-Darstellung anfordern.

\textsuperscript{85}\textit{CRUD} steht für Create, Read, Update, Delete also Erstellen, Lesen, Aktualisieren und Löschen.

\textsuperscript{86}\textit{WillPaginate}: \texttt{http://wiki.github.com/mislav/will\_paginate}
Listing 10: Before-Filter im Controller.

class BuildsController < ApplicationController
  require 'lib/helpers/session_helper'
  before_filter :load_project_filter
  after_filter :store_id_in_session_filter
  # GET /builds GET /builds.xml
  def index
    respond_to do |format|
      ...
    end
  end
private
  def load_project_filter
    if params[:project_name]
      @project = Project.find_by_name(params[:project_name])
    else
      if params[:project_id]
        @project = Project.find(params[:project_id])
      else
        if session[:project_id]
          @project = Project.find(session[:project_id])
        end
      end
    end
    unless @project
      flash[:error] = "Select a project first"
      redirect_to :controller => "projects"
    end
  end
end

Die Controller\footnote{Der Prefix im Namen steht für den \textit{Model}-Namen, in dessen Zuständigkeitsbereich der jeweilige Controller fällt} ProjectsController, ScenarioController, MeasuredDataController, WorkerTasksController und DefinitionFilesController ähneln sich in der beschriebenen Implementierung und Funktionalität. Daher wird nicht auf jeden dieser Controller detailliert eingegangen.

Der letzte Controller (WorkersController) beschränkt sich auf das Anzeigen des aktuellen Status des Anführer-Agenten und unterscheidet sich damit grundlegend von den zuvor vorgestellten Controllern. Der Anführer wird über dessen \textit{Token}\footnote{Details hierzu werden im Kapitel 4.5.4 behandelt} auf dem zentralen \textit{Memcached}-Server\footnote{Memcached-Server sind verteilte Zwischenspeicher, die zur Pufferung häufiger lesender Datenbankfragen verwendet werden. Einige der bekanntesten Nutzer dieser Technologie sind YouTube, Facebook und Twitter. Obwohl als Puffer gedacht, lässt sich ein \textit{Memcached}-Server auch bestens} ermittelt und über \textit{Distributed Ruby} dessen Status-Methode...
Listing 11: Routing-Definition.

```ruby
ActionController::Routing::Routes.draw do |map|
  ...  
  map.resources :projects, :has_many => :builds
  map.resources :projects, :has_many => :scenarios
  ...
  map.connect 'builds/:project_name/:id', :controller => 'builds',  
  :action => "show", :conditions => { :method => :get }
  ...
```

aufgerufen, die als Rückgabewert den Zustand des Agenten liefert. Damit kann überprüft werden, ob gerade Aufträge bearbeitet werden, wie viele Slave-Agenten mit der Aufgabe betraut wurden und welches Szenario abgefahren wird.

4.5.4 Softwareagenten

Verteilte ausgeführte Prozesse erfordern eine Koordinierung, damit sich die weitestgehend unabhängigen Programme nicht gegenseitig in der Verrichtung der Arbeit stören. Diesen Zweck erfüllt das zur Implementierung der Agenten verwendete Zusatzmodul *Politics*\(^\text{90}\). Es setzt einen zentralen Memcached-Server ein, um einen Anführer zu bestimmen, nutzt mDNS\(^\text{91}\) zum Auffinden der Agenten und bedient sich DRb (Distributed Ruby), um die Kommunikation zwischen den Agenten zu ermöglichen.


Wegen der prototypischen Implementierung des *Static Queue Worker* wurden umfangreiche Anpassungen und Erweiterungen notwendig, die im folgenden behandelt werden.

Verdrängungskampf. Bei der Wahl des Anführers versuchen alle *Static Queue Worker* eines speziellen *Token* auf dem Memcached-Server zu hinterlegen. Nachdem sie den *Token* gesetzt haben, überprüfen die *Static Queue Worker* durch das Lesen des Tokens
dazu verwendeten Daten zwischen verteilten Prozessen auszutauschen.

\(^{90}\)Politics: \url{http://github.com/mperham/politics/tree/master}

\(^{91}\)mDNS ist Teil der als Zeroconf (\url{http://www.zeroconf.org/}) bekannten Technologie zur konfigurationsfreien Vernetzung von Geräten. mDNS (Multicast DNS, \url{http://www.multicastdns.org/}) dient dabei zum Auflösen von Hostnamen, ohne dass hierzu ein DNS-Server gebraucht wird.
und des darin enthaltenen URI, ob es sich um den selbst gesetzten Token handelt, sie also Anführer sind, oder ob ein anderer Softwareagent die Führungsposition erlangt hat. Das Token enthält eine zeitliche Beschränkung und verfällt nach der vorgegebenen Zeit. Dabei beginnt eine neue Wettbewerbsphase und die Agenten versuchen erneut Anführer zu werden.

Das Token enthält eine zeitliche Beschränkung und verfällt nach der vorgegebenen Zeit. Dabei beginnt eine neue Wettbewerbsphase und die Agenten versuchen erneut Anführer zu werden.


**Suche nach gegenwärtigem Anführer.** Bei der ursprünglichen Anführersuche versuchen die Static Queue Worker alle Teilnehmer per mDNS zu finden und durch das Erfragen jedes Namens, den aktuellen Anführer zu erhalten. Dieses Vorgehen erwies sich als sehr langsam und störanfällig, gerade dann, wenn die Anführersrolle getauscht wird.

Die Distributed Worker bedienen sich daher der Eigenschaften des zentral gespeicherten Anführer-Tokens: Es enthält als Parameter das URI des Anführers und ist immer aktuell. Will ein Arbeiter in Kontakt mit seinem Anführer treten, muss er dazu einzig das Token abfragen und aus dem darin enthaltenen URI ein neues DRbObject erzeugen (Listing 12).

### Listing 12: Anführersuche beim Distributed Worker.

```ruby
1  def leader
2    uri = leader_uri
3    DRbObject.new(nil, uri)
4  end

5  def leader_uri
6    @leader_uri ||= @memcache_client.get(token)
7  end
```

92 Wegen der vielen verwendeten Technologien können unentdeckte Fehler zu Abstürzen des Computer-programms führen und damit den Softwareagenten unerwartet beenden.
Die Arbeitsaufträge bestehen beim *Static Queue Worker* aus einfachen Zahlenwerten, die durch die Arbeiter heruntergezählt werden und einer Warteschlange, die immer wieder neu initialisiert wird. Da die *Distributed Worker echte Arbeit verrichten sollten, waren hier zwei Arten von Erweiterungen notwendig: Der Anführer muss sich die offenen Aufträge erst noch beschaffen; der Diener muss die speziellen Aufgaben erledigen, die ihm vom Anführer aufgetragen wurden.


**Ausführung von Aufträgen.** Hat ein Diener einen Auftrag erhalten, holt er sich anhand des darin enthaltenen Szenario-Bezeichners per RESTful-Websevice alle zur Ausführung notwendigen Konfigurationen. Die umgangssprachlich verfassten Schritte (genannt *features*) werden temporär im lokalen Dateisystem abgelegt. Die projektabhängigen Konfigurations- und Definitionsdateien (aus den *Cucumber*-Unterordnern `support` und `step_definitions`) werden in die Ordner `support` und `step_definitions` unterhalb des temporären Verzeichnisses mit dem Namen des aktuellen Auftrags\(^\text{94}\) abgelegt. Diese Unterscheidung ist wichtig, da *Cucumber* die Dateien in unterschiedlicher Reihenfolge liest\(^\text{95}\). Entspricht der Name des Folgeauftrags dem zuvor durchgeführten Auftrag, erfolgt kein neues Abrufen der Dateien und die bereits lokal gespeicherten Dateien werden wiederverwendet. Im Anschluss startet der Diener das so konfigurierte *Cucumber* (siehe Listing 13).

In Zeile 2 wird anhand des temporären Verzeichnisses des Systems, des Agenten- und des aktuellen Jobbezeichners der Pfad des Verzeichnisses ermittelt, in das zuvor die Kon-

\(^{93}\)Standardmässig werden 25 Läufe ausgeführt. Die Konfiguration erfolgt über die Datei `lib/worker/config.rb` und die darin definierte Variable `TEST_RUNS`.

\(^{94}\)Der Jobbezeichner wird vom Anführer erzeugt und besteht aus einer eindeutigen 32 Zeichen langen Bytefolge.

\(^{95}\)An erster Stelle werden die Dateien aus dem Verzeichnis `support` und an zweiter Stelle die Dateien aus dem Verzeichnis `step_definitions` durch *Cucumber* geladen.

Cucumber stellt eigentlich ein Werkzeug dar, mit dem das Verhalten von Software definiert werden und automatisiert Akzeptanztests durchgeführt werden können. Unterschiedliche Erweiterungen stehen zur Verfügung, die eine Webanwendungen in den gängigen Webbrowsern testen können, indem das Werkzeug die Webbrowser steuert und Benutzereingaben simuliert.

Mangels grafischer Ausgabe lassen sich echte Browser nicht in den Distributed Workers verwenden. Abhilfe schafft hier die Erweiterung Celerity. Sie ist ein sog. Wrapper (dt. Adapter) für HtmlUnit, einen in der Programmiersprache Java geschriebenen Browser

---

96Zombie-Prozesse sind bereits beendete Prozesse, die aber so lange in der Prozessstabelle verbleiben, bis der Elternprozess abfragt, auf welche Weise dessen Kindprozess beendet wurde. Möglichen sind hier die Zustände: erfolgreich beendet, mit Fehlern beendet, abgebrochen oder abgestürzt. Erst jetzt verschwinden die Prozesse aus der Prozessstabelle.

97Unterstützt werden derzeit der Internet Explorer, Firefox, Safari und Chrome (siehe Projektseite http://wiki.github.com/aslakhellesoy/cucumber)

98Celerity: http://celerity.rubyforge.org/

99HtmlUnit: http://htmlunit.sourceforge.net/
mit JavaScript-Unterstützung, der ohne eine grafische Benutzeroberfläche daher kommt. Um diesen verwenden zu können, muss Celerity über den in Java programmierten Ruby-Interpreter JRuby ausgeführt werden. Programme, die in Ruby ausgeführt werden, können mittels Celerity\(^{100}\) das Celerity benutzen, da hierfür die JRuby-Umgebung gestartet wird.

HtmlUnit bietet, neben der Möglichkeit, JavaScript zu interpretieren und damit auf modernen Webseiten beispielsweise Ajax auszuführen, optional die Ausgabe der Ladezeit der aktuell aufgerufenen Seite. Es liegt nahe, diesen Browser für die Messung der Performance einzubeziehen. Es wurden daher gesonderte Cucumber-Schritte (eng. step) definiert, die eine Messung während der Ausführung der Szenarien erlauben (Listing 14). Zur Veranschaulichung der Verwendung sei hier auf Listing 1 verwiesen, das die Definition eines Szenarios zur Messung der Ladezeit darstellt.

### Listing 14: Definition der Cucumber-Schritte zur Messung der Ladezeit.

```ruby
1 Given /^I would like to measure the loadtime$/ do
2   @loadtime = 0
3 end
4
5 When /^I’d like to remember the preceding loadtime$/ do
6   @loadtime = @loadtime + get_load_time
7 end
8 ...
10 Then /^it should return elapsed loadtime$/ do
11   puts "CPT_LOADTIME #{@loadtime}"
12 end
14 def get_load_time
15   $browser.page.web_response.get_load_time
16 end
```

Übermittlung der Resultate. Die gemessene Ladezeit wird per DRb-Methodenauftruf an den Anführer übermittelt. Gleichzeitig wird der Diener aus der Liste der Bearbeiter entfernt. Weil mehrere Softwareagenten gleichzeitig die Methoden des Anführers aufrufen können, erfolgt der Zugriff synchronisiert (Listing 15). In Zeile 1 wird eine Monitor-Variable definiert, die für den wechselseitigen Ausschluß (eng. mutual exclusion) sorgt. Erst wenn ein Thread (dt. leichtgewichtiger Prozess) den ausschließenden Zugriff auf diese Variable erhalten hat (Zeile 3) darf er auf die Ergebnisvariable results schreibend zugreifen. Diese Variable stellt eine Hashtabelle dar, bei der die Messergebnisse entsprechend ihrer Gruppenzugehörigkeit (generic_group dient dabei als key) in

---
\(^{100}\)Celerity: http://github.com/langalex/culerity/tree/master

60

**Listing 15:** Synchronisierter Zugriff auf den Ergebnisspeicher des Anführer-Agenten.

```ruby
@results_mutex = Monitor.new
...
@results_mutex.synchronize do
  results.each{|key,value|
    @results[key] ||= Array.new
    @results[key].concat value
  }
end
```

Auf die gleiche Art versorgt der *Monitor Worker* seinen Anführer mit Messdaten, wobei die Erfassung die Daten vom Anführer gestartet und gestoppt wird. Zu diesem Zweck meldet sich alle *Monitor Worker* beim Anführer an und werden von diesem benachrichtigt, wenn die für den Stresstest verantwortlichen Diener mit ihren Arbeiten beginnen bzw. diese beenden (Listing 16). Auch hier wird der Zugriff nur unter wechselseitigem Ausschluss gewährt (Zeile 2), weil sich mehrere *Monitor Worker* gleichzeitig an- bzw. abmelden können und das anderenfalls zu unvorhersehbarem Verhalten führt.

**Listing 16:** Benachrichtigung der *Monitor Worker* durch den Anführer.

```ruby
def notify_dmons
  @dmon_mutex.synchronize do
    case @my_worker.length
    when 0
      @dmons.each { |dm,status| dm.stop_monitoring }
    when 1
      @dmons.each {|dm,status|
        dm.start_monitoring(@job_id)
        @dmons[dm] = :working
      }
    end
  end
end
```

Arbeiten Diener an Aufträgen, beginnt die Messung auf dem Anwendungsserver (Zeile 6 bis 11). Der *Monitor Worker* ermittelt auf dem Server die Prozessnummern (engl. process identifier, PID) der Serverprozesse, indem er die Konfigurationsdatei der Anwendung analysiert. Werden, wie bei einem Cluster üblich, mehrere Server-Prozesse ausgeführt, verzweigt sich der *Monitor Worker* (Listing 17) und startet (Listing 18) für jeden zu
überwachenden Prozess mehrere\textsuperscript{101} Threads.

Listing 17: Initialisierung mehrerer Threads durch den \textit{Monitor Worker}.

```ruby
def initialize_monitor_threads
  pids = find_pids
  pids.each { |pid|
    @threads << memory_monitoring(pid)
    @threads << cpu_monitoring(pid)
  }
end
```

Listing 18: Starten aller Monitor-Threads durch den \textit{Monitor Worker}.

```ruby
def start_monitoring(job_id)
  @job_id = job_id
  @monitoring_condition = :run
  @threads.each { |thread| thread.wakeup }
end
```

Kalkulation der Häufigkeitsverteilung. Sind alle Arbeiten erledigt und haben sich alle Diener abgemeldet, wird die Messung auf dem Anwendungsserver gestoppt (Zeile 4 und 5, Listing 16). Der \textit{Monitor Worker} übermittelt daraufhin alle Messwerte an den Anführer und versetzt sich im Anschluß in den Wartezustand.

Der Anführer kann nun mit der Kalkulation der statistischen Verteilung der Werte beginnen. Dazu werden zu jeder Messgruppe die Minimal-, Maximal- und Durchschnittswerte sowie das erste, zweite (Median) und dritte Quartil berechnet. Die Daten werden per \textit{RESTful}-Webservice auf dem Server gespeichert. Pro Messgruppe entsteht ein \textit{MeasuredData}-Objekt, das dem jeweiligen \textit{Build} und Szenario zugeordnet ist.


\textsuperscript{101}Da zum Zeitpunkt dieser Arbeit lediglich die Prozessorlast und Speicherbeanspruchung der Prozesse gemessen wird, werden nur zwei Threads pro Prozess ausgeführt.
Das Cont in u tu s P er f or ma n ce T e st i ng F ra m ew or k

 Listing 19: Beispielhafte Hashtabelle mit Messdaten.

```ruby
{:DB=>[29, 53, 70, 5, 82, 45, 93, 23, 47, 60],
 :SomeGroup=>[375, 632, 902, 789, 273, 483, 785, 843, 575, 123],
 :IO=>[3, 7, 6.45, 7, 2.01, 5, 2, 0, 5, 6]}
```

Ü ber w ac h ung. Zur Ü berw ac h ung der Prozesse wird das GOD-F rame work verwendet. GOD wird auf allen Hostcomputern installiert, die später die Agenten beherbergen werden. Es ist in der Lage, die Agenten zu starten und zu stoppen sowie über deren Be finden zu wachen. Sollte ein Agent unerwartet terminieren oder nicht mehr antworten, wird von GOD ein neuer Agent hochgefahren. Listing 20 enthält die verkürzte Konfiguration.

Listing 20: Konfiguration zur Überwachung der Agenten durch GOD.

```ruby
3.times do |i|
  w.start = "usr/bin/ruby #{File.expand_path(File.dirname(__FILE__) + "/../../bin/worker.rb")}"
  w.grace = 10.seconds
  w.behavior(:clean_pid_file)
  w.start_if do |start|
    start.condition(:process_running) do |c|
      c.interval = 60.seconds
      c.running = false
    end
  end
  sleep 0.5
end
```


102GOD: [http://god.rubyforge.org/](http://god.rubyforge.org/)
Wenn GOD die Agenten überwacht, wer überwacht dann GOD?“ Diese Aufgabe wird vom Werkzeug Capistrano übernommen, das dafür zuständig ist, die Webanwendung mittels Fernzugang über die Secure Shell (SSH) auf einem dedizierten Server aufzustellen (engl. to deploy). Capistrano startet in diesem Zusammenhang auch GOD, der die Monitoragenten kontrolliert. Zusätzlich sorgt es dafür, dass auf allen Hostcomputern, von denen aus der Workload erzeugt wird, die mit der Steuerung der Stressclients (den Distributed Workern) designierten GOD-Prozesse gestartet werden (Listing 21).

**Listing 21:** Auszug der Capistrano-Konfiguration. Entfernte Steuerung von GOD.

```ruby
role :cpt, 'host1', 'host2'
namespace :cpt do
  namespace :god do
    desc "starts god monitoring on all configured hosts"
    task :start, :roles => :cpt do
      run "god -c #{current_path}/lib/worker/dw.god"
    end
    desc "stops all god monitoring instances on all hosts"
    task :stop, :roles => :cpt do
      run "god terminate"
    end
  end
  namespace :worker do
    desc "starts all distributed workers on all hosts"
    task :start, :roles => :cpt do
      run "god start CPT"
    end
    desc "stops all distributed workers on all hosts"
    task :stop, :roles => :cpt do
      run "god stop CPT"
    end
  end
end
```

Ruft man den Befehl `cap cpt:god:start` auf, werden nacheinander auf den in Zeile 1 eingetragenen Computern die GOD-Prozesse gestartet. Mittels `cap cpt:god:stop` erfolgt die Terminierung aller GOD-Prozesse. `cap cpt:worker:start` startet auf den Hostcomputern die verteilten Agenten (Distributed Worker).

### 4.5.5 CruiseControl.rb Zusatzmodul

Der Entwickler hat die Möglichkeit Aufgaben zu definieren, die vor dem Integrationstest durch CruiseControl.rb ausgeführt werden (beispielsweise das Konfigurieren der Testdatenbank oder das Einspielen von Testdaten). Zu diesem Zweck wird ein Rake\textsuperscript{[103]}-Task mit dem Namen cruise implementiert. CruiseControl.rb versucht standardmäßig vor jedem Integrationstest eine so benannte Rake-Aufgabe auszuführen.

Diese Eigenschaft wird beim Prototypen dazu verwendet, die vor einem Performancetest durchzuführenden Exporte vorzunehmen. Führt CruiseControl.rb diese Aufgabe aus, werden die für Cucumber benötigten Dateien eingelesen und via RESTful-Webservice auf dem Server des Prototypen gespeichert (siehe Datei cpt.rake). Damit wird gewährleistet, dass geänderte oder neu definierte Cucumber-Konfigurationen den Distributed Workern bereit gestellt werden.

CruiseControl.rb bietet des Weiteren eine Zusatzmodul-Schnittstelle. Es werden zu bestimmten Zeitpunkten während eines Tests Ereignisse generiert, die registrierte Zusatzmodule veranlassen ihre Aufgaben durchzuführen. Einige dieser Ereignisse sind polling_source_control, build_requested, build_started, build_finished, wobei den Namen leicht entnommen werden kann, wann sie ausgelöst werden.

Soll ein Zusatzmodul beim Ereignis build_finished ausgeführt werden, genügt es eine gleich lautende Methode zu definieren. Auf diese Weise wurde auch das Zusatzmodul für den kontinuierlichen Performancetest durch den Prototypen umgesetzt (Listing 22).


Ein neues Build wurde angelegt und der ContinuousPerformanceTesting-Server beginnt nun, wie in Kapitel 4.3 beschrieben, mit der Initialisierung der Warteschlangen und dem anschließendem Performancetest.

---

\textsuperscript{103}Rake ist das Ruby-Analogon zu make, maven oder ant.

```ruby
require "rest_client"

class ContinuousPerformanceTesting
  def initialize(project)
    @project = project
  end

  def build_finished(build)
    initialize_performance_test_for_build(build)
  end

private
  def initialize_performance_test_for_build(build)
    begin
      create_new_build_on_cpt_server(build)
      set_status "continuous performance test"
    rescue
    end
  end

  def create_new_build_on_cpt_server(build)
    rest = RestClient::Resource.new "http://127.0.0.1:3000/builds/#{@project.name}"
    rest.post "<build><build_number>#{build.label}</build_number></build>", :content_type => "text/xml"
  end
end

Project.plugin :continuous_performance_testing
```

Nachdem in diesem Kapitel die technische Umsetzung und einige Details der Implementierung behandelt wurden, widmet sich das nächste Kapitel der Qualitätssicherung.

4.6 Qualitätssicherung

Während der Entwicklung des voll funktionalen Prototypen ContinuousPerformanceTesting wurde zur Qualitätssicherung das Behavior Driven Development (dt. verhaltensgetriebene Entwicklung, kurz BDD) praktiziert, das eine Weiterentwicklung der testgetriebenen Entwicklung darstellt. Hierbei wurde das für Ruby on Rails entwickelte BDD-Testframework RSpec verwendet. Nachfolgend wird das Behavior Driven Development erklärt, ein kurzer Einblick in RSpec gegeben und die angewandten Tests anhand einiger
Beispiele erläutert.


Dan North, der den Begriff Behaviour Driven Development prägte, stellt auf seiner Internetseite [Nor06] die wesentlichen Schwierigkeiten bei der testgetriebenen Entwicklung vor, die ihm im Verlauf seiner Tätigkeit als Trainer aufgefallen waren. Seiner Meinung nach liegt das Hauptproblem darin, dass die testgetriebene Entwicklung falsch verstanden wurde und immer noch wird. Entwickler, denen Dan North im Rahmen seiner Schulungen begegnete, konfrontierten ihn stets mit ähnlichen Fragen: Ihnen war nicht klar, wie Tests benannt werden und was sie denn in welchem Umfang testen sollten.


Man erkennt also, dass die Unterschiede zwischen Test Driven Development und Behaviour Driven Development nicht wirklich groß sind. Dave Astels sagt daher auch, dass „wenn man TDD richtig praktiziert, man bereits BDD macht“ [Ast06].


Mock-Objekte sind Vertreter echter Objekte oder Subsysteme, die nicht die Funktionalität echter Objekte liefern aber das gewünschte Verhalten simulieren können ([Sub05]).
Anhand der zur Berechnung der statistischen Verteilung entwickelten Bibliothek Statistics wird die Verwendung anschaulich vorgestellt.


Nach der Installation werden zwei neue Dateien angelegt. Gemäß der Konvention enthält die Datei statistics_spec.rb die Tests\textsuperscript{106} und die Datei statistics.rb die eigentliche Implementierung. Dabei werden, wie von Dave Astels gefordert, zuerst einige Erwartungen an das Verhalten definiert (Listing 23).

\begin{lstlisting}[language=rb]
require 'statistics'

describe Statistics, "given an array of numeric values" do
  before(:each) do
    end

  it "should return the maximum value"
  it "should return the minimum value"
  it "should return the average value"
  it "should return the median"
  it "should return the first and the third quartile"
end
\end{lstlisting}


\begin{lstlisting}[language=rb]
Statistics given an array of numeric values
- should return the maximum value (PENDING: Not Yet Implemented)
- should return the minimum value (PENDING: Not Yet Implemented)
- should return the average value (PENDING: Not Yet Implemented)
- should return the median (PENDING: Not Yet Implemented)
- should return the first and the third quartile (PENDING: Not Yet Implemented)
\end{lstlisting}

Listing 25 zeigt die Definition der Erwartungen. Der before-Block (Zeile 4-9) wird vor jedem Test aufgerufen und setzt die Instanzvariablen (Zeile 5-9) neu, deren Sichtbarkeit

\textsuperscript{105}Es wird davon ausgegangen, dass bereits Ruby und RubyGems installiert sind.

\textsuperscript{106}Besser gesagt, sie definieren das gewünschte Verhalten der Bibliothek.
sich über die gesamte Instanz erstreckt (daher der Name). In Zeile 12 wird nun die Erwartung an die Methode *maximum* des Moduls *Statistics* geäußert. Darin wird erwartet, dass bei einem Feld mit den Werten aus @b (Zeile 5) als Eingabe, die Zahl 56 zurückgeliefert wird. Entsprechende Erwartungen werden in den Zeilen 16, 20 und 24-25 definiert. In Zeile 29 und 30 wird hingegen erwartet, dass ein Wertepaar bestehend aus 14 und 38 bzw. 1 und 3, also dem ersten und dritten Quartil der beiden Felder @b und @e, ausgegeben wird.

**Listing 25:** Definitionen von Erwartungen an das Statistics-Modul.

```ruby
require 'lib/statistics'

describe Statistics, "given an array of numeric values" do
  before(:each) do
    @b = [56, 28, 40, 12, 14, 0, 37, 21, 37, 38]
    @c = [0, 12, 14, 21, 28, 37, 37, 38, 40, 56]
    @d = [1,1,1,2,3,3,3]
    @e = [3,2,1,3,3,1,1]
  end

  it "should return the maximum value" do
    Statistics.maximum(@b).should == 56
  end

  it "should return the minimum value" do
    Statistics.minimum(@b).should == 0
  end

  it "should return the average value" do
    Statistics.avg(@b).should == 28
  end

  it "should return the median" do
    Statistics.median(@b).should == 32
    Statistics.median(@c).should == 32
  end

  it "should return the first and the third quartile" do
    Statistics.interquartile_range(@b).should == [14,38]
    Statistics.interquartile_range(@e).should == [1,3]
  end
end
```

Die Ausführung der Tests schlägt natürlich fehl (verkürzt dargestellt in Listing 26), weil die entsprechenden Methoden noch gar nicht definiert wurden (Zeile 5) und somit auch kein Resultat liefern können.

Implementiert man die fehlenden Methoden, läuft der Test durch (Listing 27), was gleich-
Listing 26: Auszug der Ausgabe eines fehlgeschlagenen Tests

```
1 Statistics given an array of numeric values
2 - should return the maximum value (FAILED - 1)
3 1)
4 NoMethodError in 'Statistics given an array of numeric values should
5 return the maximum value'
6 undefined method 'maximum' for Statistics:Module
7 ./spec/statistics_spec.rb:12:
```


Listing 27: Erfolgreicher Testdurchlauf nach erfolgter Implementierung.

```
1 Statistics given an array of numeric values
2 - should return the maximum value
3 - should return the minimum value
4 - should return the average value
5 - should return the median
6 - should return the first and the third quartile
7
8 Finished in 0.002573 seconds
9
10 5 examples, 0 failures
```

Listing 28: Bestimmung des Maximalwertes durch Sortierung einer Liste.

```
1 def self.maximum(numbers)
2   numbers.sort!
3   numbers.last
4 end
```

Die oben aufgeführten Beispiele bieten nur einen groben Überblick der Möglichkeiten von RSpec. Sie erlauben dennoch einen schnellen Einblick in das BDD-Framework und verdeutlichen am Beispiel der Methode maximum, was mit gleichbleibendem Verhalten bei unterschiedlicher Implementierung gemeint ist. In der Quelltext-Distribution auf der beigefügten Compact Disc im Ordner ContinuousPerformanceTesting/spec finden

70

### 4.7 Rückblick

In den vergangenen Kapiteln wurden einleitend die Gründe einer kontinuierlichen Optimierung der Performance erörtert. Dazu wurden unterschiedliche Modelle vorgestellt, welche die Kostenentwicklung bei Änderungen über die Projektlaufzeit abschätzen.

Es wurden die Überlegungen wiedergegeben, die der Implementierung des Prototypen zur kontinuierlichen Messung der Performance in Webanwendungen (*ContinuousPerformanceTesting*) vorangingen wurden und schließlich zur geschickten technischen Realisierung des Prototypen führten. Es folgte eine ausführliche Erklärung der Funktionsweise der unterschiedlichen Bestandteile des Prototypen sowie deren Implementierung und der dabei verwendeten Technologien.

Im letzten Abschnitt wurde auf die während der Implementierung verwendeten Methoden zur Qualitätssicherung eingegangen. Dabei wurden die beiden Entwicklungsverfahren *Test Driven Development* und *Behaviour Driven Development* erläutert und ihre Unterschiede und Gemeinsamkeiten betont. Anhand eines Beispiels wurde die Verwendung des Werkzeugs *RSpec* beleuchtet, das zum Festlegen des Verhaltens einzelner Bestandteile des Prototypen verwendet wurde.

---

\(^{107}\) *RSpec*: [http://rspec.info](http://rspec.info)
5 Evaluierung


Bei Spree handelt es sich um einen in Ruby on Rails programmierten Webshop. Die Anwendung beinhaltet eine Demonstrationsumgebung mit Beispieldaten und erlaubt damit eine Analyse auf der Basis gleichbleibender Datensätze.

Beginnend mit der Revision 87dbf984 aus der Versionsverwaltung des Projekts Spree werden bis einschließlich Revision dfcb5446 stichprobenweise zehn Revisionen untersucht und dabei die kontinuierliche Performance ermittelt. Zwischen den beiden begrenzenden Revisionen liegen etwa vier Monate Entwicklung mit 250 Checkins.

Zusätzlich werden Referenzmessungen basierend auf Revision dfcb5446 durchgeführt, um damit einerseits die Messstabilität zu untersuchen, andererseits die mittlere absolute Abweichung der Messwerte unter gleichbleibenden Bedingungen zu bestimmen.

Zur Durchführung der Messung wurden fünf Benutzer-Szenarien definiert. Sie können im Anhang B.2 eingesehen werden.

5.1 Durchführung


5.1.1 Kontinuierlicher PerformanceTest

Als zentrale Steuer- und Speicherinstanz wird der ContinuousPerformanceTesting- sowie der Memcached-Server gestartet. Anschließend werden auf den dafür dedizierten\textsuperscript{110}...

\textsuperscript{108}Spree: http://spreecommerce.com/

\textsuperscript{109}Die maximale Anzahl der Stressclients begründet sich aus der Menge der während des Experiments zur Verfügung stehenden Computern. Wegen der durch die Stressclients verursachten Auslastung des Hostcomputers musste die Anzahl entsprechend beschränkt werden. Dies wird im weiteren Verlauf des Kapitels erörtert.

\textsuperscript{110}Die Distributed Worker sind Teil der ContinuousPerformanceTesting-Distribution und liegen im Unterverzeichnis lib. Um die Distributed Worker auf anderen Computern zu installieren, kopiert man...
Computern die Distributed Worker hochgefahren\footnote{Zum Starten der Distributed Worker wird der Befehl `god -c lib/worker/dw.god` ausgeführt, nachdem zuvor in der Datei `dw.god` die Anzahl der zu startenden Prozesse entsprechend angepasst wurde.}. Hierbei ist zu beachten, dass die maximale Anzahl der Worker pro Computer der Anzahl der Prozessor-Kerne entsprechen sollte, da die Worker in der Ausprägung als Stressclients während der Verrichtung der Arbeit den Prozessor voll auslasten und es anderenfalls zu einer Verfälschung der Messergebnisse in Folge der Sättigung des Prozessors kommen kann.


In der zuvor erwähnten Datei `prepare_cpt_test.rake` sind die zu untersuchenden Revisionen bereits konfiguriert. Dabei handelt es sich um die in Tabelle 11 (Anhang B.4.2) aufgezählten Versionen. Da ein Testdurchgang etwa eine bis drei Stunden dauern kann, werden, wie eingangs erwähnt, nur zehn Revisionen untersucht.

Wegen der Verwendung von Google Analytics\footnote{Google Analytics ist ein kostenloser Dienst der Firma Google, der eine Analyse der Seitenzugriffe erlaubt.} im Projekt Spree muss der Zugriff auf die entsprechenden Server durch die Anwendung unterbunden werden. Ansonsten würde das Messergebnis verfälscht\footnote{Bei mehreren, während der Vorbereitung durchgeführten Messungen war eine Latenz von 200 bis über 1000 Millisekunden messbar, wenn versucht wurde, auf den Server von Google zuzugreifen.}.


### 5.1.2 Referenzmessung

Zur Ermittlung der Messstabilität dient die Referenzmessung. Ihr Ablauf ist in beiden Gruppen (zwei und drei Stressclients) gleich. Es wird eine einzige Revision der Anwendung verwendet und dabei in zehn Durchgängen ein Performancetest durchgeführt. Dazu wird die letzte Revision der weiter oben beschriebenen Tests verwendet.


Für den Test der zweiten Gruppe, wenn also lediglich zwei Stressclients verwendet werden, müssen bis auf drei `Distributed Worker` alle anderen Agenten abgeschaltet werden.\(^{120}\)

\(^{118}\) `scaling_governor` befindet sich im Verzeichnis `/sys/devices/system/cpu/cpu0/cpufreq/modern`.

\(^{119}\) Pro Durchgang fünf Szenarien mit je 25 Messungen.

\(^{120}\) Alternativ kann die Anzahl der Agenten über die Szenarien beschränkt werden. Hierzu muss in jedem
5 Evaluierung

Abbildung 14: Versuchsaufbau zur Durchführung der Performancemessung
SUT: System Under Test (Applikationsserver), RTE: Remote Terminal Emulation
(Stressclient), CPT: ContinuousPerformanceTesting-Server

5.2 Aufbau


Der Betrieb der beiden letztgenannten Server hat keinerlei Auswirkungen auf die Leistung des Computers und damit auf die darin ausgeführten Stressclients. Dies liegt zum einen daran, dass die Kommunikation mit dem ContinuousPerformanceTesting-Server lediglich vor und nach den Messungen stattfindet:

Der Anführer-Agent überprüft die serverseitige Auftragswarteschlange nur dann, wenn Messungen erfolgen. Er überträgt die Messresultate an den Server erst dann, wenn alle Diener ihre Messungen abgeschlossen haben. Die Diener rufen die zur Ausführung notwendigen Konfigurationen nur vor dem Beginn einer Messreihe vom Server ab und speichern sie lokal.

Zum anderen werden während der Kommunikation mit dem Memcached-Server nur vernachlässigbar wenige\textsuperscript{121} Bytes übertragen. Der Memcached-Server hält seine Daten im schnellen Arbeitsspeicher vor.

\textsuperscript{121} Es wird ein Token der Form „cpt_stressor_token“ sowie das URI der Agenten in Form von „druby://hostname:port“ übertragen.
5 Evaluierung

Der *Distributed Worker* mit der Rolle des Anführers befindet sich auf dem *Remote Terminal Emulation 1*-System (Anhang B.1).

Die ausführliche Hard- und Softwarekonfiguration der drei Computer ist im Anhang B.1 enthalten.

5.3 Ergebnisse

Nachfolgend werden die Ergebnisse der Untersuchung wiedergegeben, wobei eine Unterteilung in die zwei zu Beginn erwähnten Gruppen *drei Stressclients* und *zwei Stressclients* erfolgt. Die Resultate werden im anknüpfenden Kapitel 5.4 interpretiert.

5.3.1 Gruppe eins: Drei Stressclients


![Abbildung 15: Gesamtperformance (3 VU).](image)

In Abbildung 15 ist die gemessene Gesamtperformance zweier Durchgänge als Linendiagramm gegenüber gestellt. Auf der Ordinate ist die Gesamtladezeit in Millisekunden aufgetragen. Die Abszisse enthält die zu Grunde liegenden Revisionen gemäß Tabelle 3.

Abschnitt steigt von 4151 bzw. 4217 Millisekunden auf 5023 bzw. 4965 Millisekunden um etwa 20 bzw. 18 Prozent.

Bei den der Gesamtperformance zu Grunde liegenden Szenarien sticht besonders „add t-shirt to cart and continue shopping“ hervor (Abbildung 23 im Anhang B.3.1). Dort steigt die durchschnittliche Ladezeit von 1389 bzw. 1383 Millisekunden in der ersten Revision um etwa 30 Prozent auf 1823 bzw. 1840 Millisekunden in der letzten Revision. Der Tiefpunkt liegt mit 1287 bzw. 1270 Millisekunden bei Revision fünf.


Die verbliebenen Szenarien weisen nur einen leichten Anstieg der Ladezeit auf (vgl. Anhang B.3.1).

5.3.2 Gruppe zwei: Zwei Stressclients

Bei der Referenzmessung der zweiten Gruppe (Abbildung 36 im Anhang B.4.1) schwankte die Gesamtperformance zwischen 4204 und 4295 Millisekunden (vgl. Tabelle 18). Der Median lag bei 4267 Millisekunden, das erste und dritte Quartil bei 4234 und 4275 Millisekunden. Die mittlere absolute Abweichung bezüglich des Medians betrug 22,4 ms (0,52 Prozent).

Das Diagramm in Abbildung 16 zeigt die Graphen der Gesamtperformance aus der Messreihe der zweiten Gruppe, bei der zwei Stressclients verwendet wurden. Auf der Ordinate ist die Gesamtladezeit im Millisekunden aufgetragen. Die Abszisse nummeriert die untersuchten Revisionen gemäß Tabelle 11 (Anhang B.4.2).

Zwei der applizierten Szenarien weisen einen starken Wachstum der Ladezeit auf. Das Szenario „add t-shirt to cart and continue shopping“ (Abbildung 30, Anhang B.4.1) hat eine anfängliche durchschnittliche Ladezeit von 1249 bzw. 1246 Millisekunden. Im weiteren Verlauf steigt die durchschnittliche Ladezeit um über 20 Prozent auf 1541 bzw. 1605 Millisekunden.

Das zweite auffällige Szenario „browse through categories“ (Abbildung 32, Anhang B.4.1) steigt von durchschnittlichen 1439 bzw. 1437 Millisekunden in der ersten Revision auf 1720 bzw. 1698 Millisekunden in der letzten Revision. Der Zuwachs liegt damit ebenfalls bei etwa 20 Prozent.

Bei den restlichen Szenarien gibt es einen geringen Anstieg der durchschnittlichen Laufzeit (vgl. Anhang B.4.1).

5.4 Interpretation

Bei der Referenzmessung wird wegen der unveränderten Revision und somit gleichbleibendem Programmcode erwartet, dass die ermittelten Performancewerte stagnieren. Tatsächlich wird diese Erwartung bei der Messung mit zwei Stressclients erfüllt. Die Werte haben eine mittlere absolute Abweichung bezüglich des Medians (kurz MD) von 0,52 Prozent (22,4 Millisekunden). Die Messung ist also stabil.

Signifikant ist hingegen die Schwankung des Referenzwertes in der ersten Gruppe mit drei Stressclients. Sie fällt mit 1,88 Prozent (96,2 Millisekunden) fast viermal so hoch aus wie in der zweiten Gruppe. Die Messung kann aber wegen der insgesamt niedrigen Fluktuation noch als stabil angesehen werden.

Auffällig ist die Verschiebung der Leistungskurve auf der Y-Achse im Vergleich zur Referenzmessung der zweiten Gruppe. Der Median der ersten Gruppe ist knapp 800 Millisekunden höher als der zweiten Messgruppe.

Die Verschiebung der Messwerte um fast eine Sekunde wird auch bei der Gegenüberstellung der Gesamtperformance aus Gruppe eins mit Gruppe zwei sichtbar (Abbildung 17). Weiterhin fällt auf, dass die beiden Linien der zweiten Gruppe glatter sind als die der ersten Gruppe.
Eine Erklärung für diesen Sachverhalt könnte in der unterschiedlichen Beanspruchung des Serverprozesses auf dem SUT liegen. Während beim Experiment mit zwei Stressclients der Serverprozess den Mikroprozessor des Servers zu maximal 60 Prozent auslastete, wuchs diese Auslastung bei drei gleichzeitigen Zugriffen zeitweise auf bis zu 100 Prozent. Damit war die Sättigung des Prozessors erreicht und die Anfragen konnten nun nicht mehr sofort bearbeitet werden (vgl. Kapitel 3.2.2). Die Clients mussten daher auf freier werdende Ressourcen warten, die Gesamtladezeit stieg in Folge dessen an.

Abbildung 17: Diagramm der Gesamtperformance: 3 Clients vs. 2 Client gemäß Tabelle 19.

Sowohl die Graphen der ersten als auch der zweiten Gruppe weisen eine hohe Kongruenz auf (im Rahmen der ermittelten Abweichung in der Referenzmessung). Zusätzlich sind die beiden Gruppen nahezu parallel, was die Gesamtladezeit betrifft. Daraus lässt sich schliessen, dass die Gesamtperformance des untersuchten Projekts Spree im Laufe der Entwicklung immer weiter sinkt, da die Gesamtladezeit kontinuierlich ansteigt\textsuperscript{122}. Deutlich wird diese Abwärtsentwicklung bei den letzten vier untersuchten Revisionen. In den stabileren Messungen der zweiten Gruppe steigt die Ladezeit im letzten Viertel um 12 Prozent. Die kontinuierliche Performancemessung deckt somit die Tendenz hin zu stetigen Verschlechterung der Performance im untersuchten Projekt Spree auf, die von den Entwicklern nicht bemerkt wurde. Die ausführliche Bewertung erfolgt im folgendem Kapitel 6.

\textsuperscript{122}Die Performance verhältn proportional zur gemessenen Gesamtladezeit: Je höher die Ladezeit, desto niedriger/schlechter die Performance.
6 Fazit


Weil die Kosten für zu spät durchgeführte Optimierungen und Änderungen sehr hoch ausfallen können, ist es sinnvoll, getreu den Prinzipien der Agilen Softwareentwicklung, auch jene sich ungünstig auf die Leistung eines Systems auswirkenden Auslöser kontinuierlich zu erkennen und zu verbessern (vgl. Kapitel 3.6 und 4).


Wichtig beim Aufbau der Umgebung zur kontinuierlichen Performancemessung ist es, unvorhersagbare Störungen durch die mittlere Meile unbedingt auszuschliessen (vgl. Kapitel 3.2.3). Bei Messungen während der Vorbereitung wurde durch den in der Anwendung eingebetteten Google Analytics-Code eine fluktuierende Latenz von 200 bis 1000 Millisekunden beobachtet, die durch den Aufruf der Drittservices von Google während
der Darstellung einer Applikationsseite verursacht wurde. Störungen dieser Art müssen während der Leistungsanalyse unterbunden werden.

Da die Verarbeitungsgeschwindigkeit aktueller Prozessoren nicht konstant ist, weil das Betriebssystem die Taktfrequenz je nach Beanspruchung erhöhen und absenken kann, sollte hier eine Einschränkung vorgenommen und der Takt des Prozessors auf einen fixen Wert eingestellt werden. Anderenfalls werden die Messergebnisse verfälscht und können zum Teil willkürliche Ergebnisse anzeigen.

Wird beispielsweise bei einer Revision a ein Algorithmus implementiert, der eine Ladezeit x zur Folge hat, kann eine ungünstige Refaktorisierung des Algorithmus als Verbesserung missinterpretiert werden. Das passiert dann, wenn der Algorithmus wegen einer Veränderung in einer Revision b die CPU stärker beansprucht und damit das Betriebssystem die Erhöhung der Taktrate veranlasst: Die Berechnungen werden nunmehr dank des höheren Takts schneller abgeschlossen und die Ladezeit sinkt. Ist die jetzt gemessene Ladezeit y < x, wird fälschlicherweise angenommen, dass eine Performanceoptimierung stattgefunden hat, obwohl gegenteiliges passiert ist.

Bei der durchgeführten Evaluation liess sich durch Szenarien mit nur zwei Stressclients die Performance zuverlässig messen (der MD$^{123}$ lag bei 0,52 Prozent).

Es ist zu beachten, dass eine ungünstig gewählte Anzahl der Stressclients das Messergebnis beeinträchtigen kann, weil hierdurch eine Sättigung der serverseitigen Verarbeitungskapazität auftritt und die damit bedingte zusätzliche Wartezeit die Messungen manipuliert. Um eine zuverlässige Messung zu erhalten, empfiehlt es sich, in mehreren Läufen eine Referenzmessung durchzuführen und dabei pro Lauf die Zahl der Stressclients zu erhöhen. Je stärker die Werte dabei fluktuiieren, um so näher ist man an der Kapazitätsgrenze des Systems angelangt.


Angesichts der Möglichkeit in den Szenariendefinitionen die maximale Anzahl an Stressclients zu konfigurieren und der Option, unter Zuhilfenahme zusätzlicher Computer die Anzahl zur Verfügung stehender Stressclients nahezu beliebig zu erhöhen, liegt es nahe, mehrere ähnliche Szenarien vorzulegen, die sich lediglich in der Anzahl der Stressclients unterscheiden.

$^{123}$MD steht für die mittlere absolute Abweichung bezüglich des Medians.
7 Ausblick

Die vorliegende Arbeit geht der Frage nach, ob sich mittels kontinuierlicher Performance-Cetests negative Entwicklungen bei der Leistung einer Webanwendung erkennen lassen und damit eine frühzeitige Optimierung ermöglicht wird.

Es wurde ein Prototyp entwickelt, der fortlaufende Leistungstests durchführen kann und mit dessen Hilfe sich negative Tendenzen in der Leistung einer Webanwendung bestimmen lassen. Zur Evaluierung wurde ein quellofenes Projekt untersucht. Dabei wurden mehrere Coderevisionen auf ihre Leistung hin untersucht und damit der kontinuierliche Test simuliert.

Auf technischer Seite sind weitere Erweiterungen denkbar, die das Framework vervollständigen und damit die Suche nach Ursachen mangelnder Performance erleichtern würden. Die Betrachtung der Ladezeit kann hier nur ein Anfang sein.

Wünschenswert wären zusätzliche Monitore, die nicht nur die Ladezeit, Prozessorbeanspruchung und Speicherauslastung untersuchen würden, sondern auch die Aktivitäten der Datenbanken, der Ein- und Ausgabegeräte im Allgemeinen und gegebenenfalls eine direktere Codeanalyse auf Basis aufgerufener Methoden ermöglichen.


Um das Framework universeller verwenden zu können, wäre eine Eingliederung in weitere Umgebungen für kontinuierliche Integrationstests denkbar. Dank der vorhandenen RESTful-Schnittstelle ist das ohne weiteres möglich.

Auf wissenschaftlicher Seite bietet sich Begleitung eines Softwareprojekts über einen längeren Zeitraum an. Dabei kann die Eignung sowie der tatsächliche Nutzen des in dieser Arbeit entwickelten Prototypen untersucht werden. Zwar hat sich durch die Analyse des Webshops Spree unter Verwendung älterer Revisionen gezeigt, dass die Performanceentwicklung über die Zeit ermittelt werden kann und sich beim genannten Projekt sogar
ungünstig ausgeprägt hat. Ob die Entwickler bei Kenntnis dieser Ergebnisse schon in der Vergangenheit entsprechend reagiert und Refaktorisierung betrieben hätten, bleibt offen.

Im gleichen Zusammenhang sollte die Verwendung des Frameworks durch Entwickler untersucht werden, um so Schwächen und Probleme erkennen zu können und durch verbesserte Usability und zusätzliche Erweiterungen den Nutzen zu erhöhen. Wichtig wäre dabei zu erfahren, ob ein solches Werkzeug überhaupt angenommen wird und wie (und ob) sich das Bewusstsein der Entwickler im Hinblick auf die Leistung einer Anwendung herausbildet und verändert.
8 Zusammenfassung

Ziel der Arbeit war die Entwicklung eines Werkzeugs zur Durchführung kontinuierlicher Leistungstests, das während der Entwicklung einer Webanwendung verwendet werden kann. Mit dem Werkzeug sollen frühzeitig Engpässe erkannt werden und zu einer gezielten Optimierung durch Refaktorisierung anregen. Die Performanceoptimierung soll so Teil der in der Agilen Softwareentwicklung praktizierten häufigen Codeverbesserung durch Umgestaltung sein. Der fortlaufende Performancetest soll Bestandteil der kontinuierlichen Integrationstests werden.


Die Resultate der Performancenanalyse bestätigten die Messstabilität des Prototypen und deckten eine negative Entwicklung der Leistung im untersuchten Softwareprojekt auf.

Die Evaluierung bestätigte, dass der Prototyp zur kontinuierlichen Performanceanalyse seine Aufgabe erfüllt und frühzeitige Erkennung von Tendenzen hin zu einer schlechteren Leistung ermöglicht. Sie verdeutlichte gleichzeitig notwendige Voraussetzungen an die Testumgebung, damit eine zuverlässige und störungsfreie Analyse möglich wird.

An diese Arbeit anknüpfend bietet sich die Begleitung eines Projekts unter Einsatz des Prototypen an. In deren Verlauf sollte überprüft werden, wie sich das Vorhandensein des Werkzeugs zur kontinuierlichen Performancemessung auf die Leistung eines Systems auswirkt und wie dieses Werkzeug von Entwicklern angenommen wird.
Literatur


VII


Highsmith, J.: Agile software development ecosystems. 2002


WII I


Literatur

Abruf: 15. April 2009. – Performance-Vergleich der verbreiteten JavaScript-Engines


[Sch06] SCHLOSSNAGLE, T.: Scalable internet architectures. Sams Indianapolis, IN, USA, 2006


[Sou08] SOUDERS, Steve: High Performance Web Sites. In: Queue 6 (2008), Nr. 6, S. 30-37. – ISSN 1542-7730


X


Entwurf des GUI mit der Übersicht aller Revisionen (oben links), aller durchgetesteten Szenarien (oben rechts), die Detailansicht eines Szenarios (unten rechts) und die Ansicht vergangener Messungen eines ausgewählten Szenarios (unten links).
Abbildung 19: Zweiter Papierprototyp - Projektübersicht.

Der Papierprototyp zur Auflistung der Projekte und ihrer Performance. Hinter der Seiten-Nummerierung (engl. pagination) verbergen sich weitere Projekte. Die Beschränkung der Auswahl soll die Übersicht steigern.
Abbildung 20: Zweiter Papierprototyp - Übersicht der Builds eines Projekts.


Rechts ist die Gesamtperformance aller Szenarien eines Projekts dargestellt. Der Klick auf den Namen eines Szenarios führt zur dessen Detaildarstellung (links). Dort wird die zeitliche Entwicklung je Messgruppe dargestellt.

Abbildung 22: Zweiter Papierprototyp - Detailansicht des Diagramms.

Die Diagramme stellen die Maximal-, Minimal-, Durchschnitts- und Medianwerte auf einer Zeitskala dar.
B Anhang Evaluierung

B.1 Hard- und Softwarekonfiguration der Testumgebung

Applikationsserver (SUT)
- AMD Athlon\(^\text{tm}\) Dual Core Processor 5050e 2.6 GHz
- 8 GB RAM
- Ubuntu 9.04 x86\_64
- ruby 1.8.7 (2008-08-11 patchlevel 72) [x86\_64-linux]
- mysql Ver 14.12 Distrib 5.0.75, for debian-linux-gnu (x86\_64)
- Monitor-Agent
- 100 Mbit/s Ethernet

Remote Terminal Emulation 1
- Intel(R) Pentium(R) M processor 1.60GHz
- 1,25 GB RAM
- Ubuntu 9.04
- ruby 1.8.7 (2008-08-11 patchlevel 72) [i486-linux]
- jruby 1.2.0 (ruby 1.8.6 patchlevel 287) (2009-06-24 rev 6586) [i386-java]
- java version "1.6.0_14"
- 2 Distributed Worker (1 Anführer, 1 Diener)
- 100 Mbit/s Ethernet

Remote Terminal Emulation 2
- Intel Core 2 Duo 2,1 GHz
- 4 GB RAM
- Mac OS X 10.5.7 (9J61)
- ruby 1.8.6 (2008-08-11 patchlevel 287) [universal-darwin9.0]
- jruby 1.2.0 (ruby 1.8.6 patchlevel 287) (2009-06-24 rev 6586) [i386-java]
- java version "1.5.0_19"
- mysql Ver 14.12 Distrib 5.0.51a, for apple-darwin8.5.1 (i686)
- ContinuousPerformanceTesting-Server
- Memcached-Server
B Anhang Evaluierung

- 2 Distributed Worker (2 Diener)
- 100 Mbit/s Ethernet

B.2 Szenarien

**Listing 30: Spree Szenario 1: Add T-Shirt into the cart and continue shopping**

1. Scenario: Add T-Shirt into the cart and continue shopping
2. Given 3 stress clients
3. And I would like to measure the loadtime
4. When I go to spree
5. And I’d like to remember the preceding loadtime
6. And I follow "Clothing"
7. And I’d like to remember the preceding loadtime
8. And I follow "Ruby on Rails Jr. Spaghetti"
9. And I’d like to remember the preceding loadtime
10. And I press "Add To Cart"
11. And I’d like to remember the preceding loadtime
12. And I follow "Continue shopping"
13. And I’d like to remember the preceding loadtime
14. And it should load within 5 seconds
15. And it should return elapsed loadtime

**Listing 31: Spree Szenario 2: Add T-Shirt into the cart and increment quantity**

1. Scenario: Add T-Shirt into the cart and increment quantity
2. Given 3 stress clients
3. When I go to spree
4. And I follow "Clothing"
5. And I follow "Ruby on Rails Jr. Spaghetti"
6. And I press "Add To Cart"
7. And I would like to measure the loadtime
8. And I type in "100000" for "order_line_item_attributes\d+\_quantity"
9. And I press "Update"
10. And I’d like to remember the preceding loadtime
11. And it should load within 5 seconds
12. And it should return elapsed loadtime

**Listing 32: Spree Szenario 3: Browse through categories**

1. Scenario: Browse through categories
2. Given 3 stress clients
3. And I would like to measure the loadtime
4. When I go to spree
5. And I’d like to remember the preceding loadtime
6. And I follow "Clothing"
7. And I’d like to remember the preceding loadtime
8. And I follow "Bags"
9. And I’d like to remember the preceding loadtime
And I follow "Mugs"
And I’d like to remember the preceding loadtime
And I follow "Ruby"
And I’d like to remember the preceding loadtime
And I follow "Ruby on Rails"
And I’d like to remember the preceding loadtime
And I follow "Apache"
And I’d like to remember the preceding loadtime
And it should load within 5 seconds
And it should return elapsed loadtime

Listing 33: Spree Szenario 4: Show all brands

Scenario: Show all brands
Given 3 stress clients
When I go to spree
And I follow "Ruby"
And I would like to measure the loadtime
And I follow "Brands"
And I’d like to remember the preceding loadtime
And it should load within 5 seconds
And it should return elapsed loadtime

Listing 34: Spree Szenario 5: Show all categories

Scenario: Show all categories
Given 3 stress clients
When I go to spree
And I follow "Clothing"
And I would like to measure the loadtime
And I follow "Categories"
And I’d like to remember the preceding loadtime
And it should load within 5 seconds
And it should return elapsed loadtime

B.3 Drei virtuelle Benutzer

B.3.1 Diagramme

Abbildung 23: Szenario add t-shirt to cart and continue shopping (3 VU).

Abbildung 24: Szenario add t-shirt to cart and increment quantity (3 VU).

Abbildung 25: Szenario browse through categories (3 VU).
Abbildung 26: Szenario show all brands (3 VU).

Abbildung 27: Szenario show all categories (3 VU).

Abbildung 28: Gesamtperformance (3 VU).
Das Diagramm stellt die Gesamtperformance dar, wobei zwei Messdurchgänge ausgeführt wurden. Die Ordinate enthält die Gesamtladezeit in Millisekunden. Die Abszisse stellt die Revisionen dar.
Abbildung 29: Referenzmessung basierend auf Revision 10 (Tabelle 3, 3 VU).
Die Ordinate enthält die Ladezeit in Millisekunden, auf der Abszisse sind die Durchläufe aufgetragen.

B.3.2 Messdaten

<table>
<thead>
<tr>
<th>Nummer</th>
<th>Revisionsbezeichner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>87dbf98493359c076db573a30e298686878615d4</td>
</tr>
<tr>
<td>2</td>
<td>dd114e1af0e3c1c9bcebb0b1e8f3bd2e6368d80e8</td>
</tr>
<tr>
<td>3</td>
<td>2236ad5932a6e4460c9d202c0297320c4f2d63e</td>
</tr>
<tr>
<td>4</td>
<td>12c1a46f0da150de6bd07276317e4a0b80ad5450</td>
</tr>
<tr>
<td>5</td>
<td>e4f36e0a43c13a1b8ddc4kka9654e01d7cbebc0</td>
</tr>
<tr>
<td>6</td>
<td>302749476e3c58518deb0aa7fxd5b315a7e2b033</td>
</tr>
<tr>
<td>7</td>
<td>667a13a7b6aeea605d0ca016baa3a9d061c937c</td>
</tr>
<tr>
<td>8</td>
<td>d4f4c1f5e74a1d2c0f1d70969e146914b39600</td>
</tr>
<tr>
<td>9</td>
<td>8da74d8f45b0f4d62967f1f4b16f4d2c556f480</td>
</tr>
<tr>
<td>10</td>
<td>dfb54465e4cefe42799386db1979a57d2d817</td>
</tr>
</tbody>
</table>

### Tabelle 4: Ermittelte Gesamtperformance (3 VU).

Gesamtperformance pro Revision in Millisekunden bei Leistungsanalyse mit deaktiviertem Google Analytics. MD steht für die mittlere absolute Abweichung bezüglich des Medians.

<table>
<thead>
<tr>
<th>Revision</th>
<th>1. Lauf</th>
<th>2. Lauf</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4139</td>
<td>4158</td>
</tr>
<tr>
<td>2</td>
<td>4355</td>
<td>4344</td>
</tr>
<tr>
<td>3</td>
<td>4062</td>
<td>4128</td>
</tr>
<tr>
<td>4</td>
<td>4289</td>
<td>4266</td>
</tr>
<tr>
<td>5</td>
<td>4151</td>
<td>4217</td>
</tr>
<tr>
<td>6</td>
<td>4485</td>
<td>4635</td>
</tr>
<tr>
<td>7</td>
<td>4427</td>
<td>4540</td>
</tr>
<tr>
<td>8</td>
<td>4637</td>
<td>4715</td>
</tr>
<tr>
<td>9</td>
<td>4813</td>
<td>4741</td>
</tr>
<tr>
<td>10</td>
<td>5023</td>
<td>4965</td>
</tr>
</tbody>
</table>

### Tabelle 5: Messwerte (3 VU): scenario show all categories

#1 und #2 stehen für den ersten bzw. zweiten Durchgang. Avg und med für den Durchschnittswert bzw. den Median. 3 VU meint, dass bei der Messung drei virtuelle User (Stressclients) verwendet wurden.

<table>
<thead>
<tr>
<th>Revision</th>
<th>#1 avg</th>
<th>#2 avg</th>
<th>#1 med</th>
<th>#2 med</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>423</td>
<td>401</td>
<td>450</td>
<td>367</td>
</tr>
<tr>
<td>2</td>
<td>413</td>
<td>468</td>
<td>363</td>
<td>404</td>
</tr>
<tr>
<td>3</td>
<td>454</td>
<td>401</td>
<td>456</td>
<td>353</td>
</tr>
<tr>
<td>4</td>
<td>478</td>
<td>497</td>
<td>474</td>
<td>472</td>
</tr>
<tr>
<td>5</td>
<td>481</td>
<td>536</td>
<td>421</td>
<td>474</td>
</tr>
<tr>
<td>6</td>
<td>446</td>
<td>460</td>
<td>467</td>
<td>489</td>
</tr>
<tr>
<td>7</td>
<td>476</td>
<td>443</td>
<td>482</td>
<td>468</td>
</tr>
<tr>
<td>8</td>
<td>481</td>
<td>488</td>
<td>498</td>
<td>467</td>
</tr>
<tr>
<td>9</td>
<td>467</td>
<td>415</td>
<td>414</td>
<td>360</td>
</tr>
<tr>
<td>10</td>
<td>523</td>
<td>474</td>
<td>508</td>
<td>476</td>
</tr>
</tbody>
</table>
## Anhang Evaluierung

### Tabelle 6: Messwerte (3 VU): scenario browse through categories

#1 und #2 stehen für den ersten bzw. zweiten Durchgang. Avg und med für den Durchschnittswert bzw. den Median. 3 VU meint, dass bei der Messung drei virtuelle User (Stressclients) verwendet wurden.

<table>
<thead>
<tr>
<th>Revision</th>
<th>#1 avg</th>
<th>#2 avg</th>
<th>#1 med</th>
<th>#2 med</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1765</td>
<td>1763</td>
<td>1724</td>
<td>1752</td>
</tr>
<tr>
<td>2</td>
<td>1810</td>
<td>1797</td>
<td>1826</td>
<td>1756</td>
</tr>
<tr>
<td>3</td>
<td>1723</td>
<td>1752</td>
<td>1684</td>
<td>1765</td>
</tr>
<tr>
<td>4</td>
<td>1774</td>
<td>1708</td>
<td>1822</td>
<td>1694</td>
</tr>
<tr>
<td>5</td>
<td>1798</td>
<td>1795</td>
<td>1827</td>
<td>1784</td>
</tr>
<tr>
<td>6</td>
<td>1849</td>
<td>1926</td>
<td>1892</td>
<td>1907</td>
</tr>
<tr>
<td>7</td>
<td>1837</td>
<td>1895</td>
<td>1818</td>
<td>1900</td>
</tr>
<tr>
<td>8</td>
<td>1883</td>
<td>1890</td>
<td>1908</td>
<td>1930</td>
</tr>
<tr>
<td>9</td>
<td>1974</td>
<td>1931</td>
<td>1957</td>
<td>1892</td>
</tr>
<tr>
<td>10</td>
<td>1991</td>
<td>1958</td>
<td>1988</td>
<td>1897</td>
</tr>
</tbody>
</table>

### Tabelle 7: Messwerte (3 VU): scenario show all brands

#1 und #2 stehen für den ersten bzw. zweiten Durchgang. Avg und med für den Durchschnittswert bzw. den Median. 3 VU meint, dass bei der Messung drei virtuelle User (Stressclients) verwendet wurden.

<table>
<thead>
<tr>
<th>Revision</th>
<th>#1 avg</th>
<th>#2 avg</th>
<th>#1 med</th>
<th>#2 med</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>325</td>
<td>380</td>
<td>299</td>
<td>311</td>
</tr>
<tr>
<td>2</td>
<td>358</td>
<td>402</td>
<td>307</td>
<td>313</td>
</tr>
<tr>
<td>3</td>
<td>345</td>
<td>325</td>
<td>310</td>
<td>300</td>
</tr>
<tr>
<td>4</td>
<td>367</td>
<td>390</td>
<td>338</td>
<td>314</td>
</tr>
<tr>
<td>5</td>
<td>385</td>
<td>384</td>
<td>327</td>
<td>317</td>
</tr>
<tr>
<td>6</td>
<td>367</td>
<td>374</td>
<td>317</td>
<td>317</td>
</tr>
<tr>
<td>7</td>
<td>323</td>
<td>340</td>
<td>308</td>
<td>315</td>
</tr>
<tr>
<td>8</td>
<td>379</td>
<td>344</td>
<td>308</td>
<td>307</td>
</tr>
<tr>
<td>9</td>
<td>409</td>
<td>378</td>
<td>325</td>
<td>334</td>
</tr>
<tr>
<td>10</td>
<td>406</td>
<td>379</td>
<td>324</td>
<td>324</td>
</tr>
</tbody>
</table>
### Tabelle 8: Messwerte (3 VU): scenario add t-shirt to cart and increment quantity

<table>
<thead>
<tr>
<th>Revision</th>
<th>#1 avg</th>
<th>#2 avg</th>
<th>#1 med</th>
<th>#2 med</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>237</td>
<td>231</td>
<td>231</td>
<td>227</td>
</tr>
<tr>
<td>2</td>
<td>250</td>
<td>163</td>
<td>228</td>
<td>112</td>
</tr>
<tr>
<td>3</td>
<td>194</td>
<td>190</td>
<td>112</td>
<td>222</td>
</tr>
<tr>
<td>4</td>
<td>219</td>
<td>220</td>
<td>236</td>
<td>231</td>
</tr>
<tr>
<td>5</td>
<td>200</td>
<td>232</td>
<td>229</td>
<td>229</td>
</tr>
<tr>
<td>6</td>
<td>311</td>
<td>303</td>
<td>305</td>
<td>250</td>
</tr>
<tr>
<td>7</td>
<td>288</td>
<td>262</td>
<td>237</td>
<td>245</td>
</tr>
<tr>
<td>8</td>
<td>297</td>
<td>341</td>
<td>246</td>
<td>316</td>
</tr>
<tr>
<td>9</td>
<td>292</td>
<td>304</td>
<td>253</td>
<td>250</td>
</tr>
<tr>
<td>10</td>
<td>280</td>
<td>314</td>
<td>246</td>
<td>254</td>
</tr>
</tbody>
</table>

Die Werte zeigen die Durchschnittswerte (Avg) und Mediane (med) für zwei Durchläufe (1 und 2). 3 VU steht für die Verwendung von drei virtuellen Benutzern (Stressclients) während der Messung.

### Tabelle 9: Messwerte (3 VU): scenario add t-shirt to cart and continue shopping

<table>
<thead>
<tr>
<th>Revision</th>
<th>#1 avg</th>
<th>#2 avg</th>
<th>#1 med</th>
<th>#2 med</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1389</td>
<td>1383</td>
<td>1339</td>
<td>1373</td>
</tr>
<tr>
<td>2</td>
<td>1524</td>
<td>1514</td>
<td>1496</td>
<td>1511</td>
</tr>
<tr>
<td>3</td>
<td>1346</td>
<td>1460</td>
<td>1350</td>
<td>1448</td>
</tr>
<tr>
<td>4</td>
<td>1451</td>
<td>1451</td>
<td>1398</td>
<td>1407</td>
</tr>
<tr>
<td>5</td>
<td>1287</td>
<td>1270</td>
<td>1292</td>
<td>1255</td>
</tr>
<tr>
<td>6</td>
<td>1512</td>
<td>1572</td>
<td>1460</td>
<td>1548</td>
</tr>
<tr>
<td>7</td>
<td>1503</td>
<td>1600</td>
<td>1433</td>
<td>1619</td>
</tr>
<tr>
<td>8</td>
<td>1597</td>
<td>1652</td>
<td>1603</td>
<td>1577</td>
</tr>
<tr>
<td>9</td>
<td>1671</td>
<td>1713</td>
<td>1637</td>
<td>1675</td>
</tr>
<tr>
<td>10</td>
<td>1823</td>
<td>1840</td>
<td>1811</td>
<td>1814</td>
</tr>
</tbody>
</table>

Die Werte zeigen die Durchschnittswerte (Avg) und Mediane (med) für zwei Durchläufe (1 und 2). 3 VU steht für die Verwendung von drei virtuellen Benutzern (Stressclients) während der Messung.
### Anhang Evaluierung

<table>
<thead>
<tr>
<th>Durchgang</th>
<th>Messung</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5039</td>
</tr>
<tr>
<td>2</td>
<td>5261</td>
</tr>
<tr>
<td>3</td>
<td>5077</td>
</tr>
<tr>
<td>4</td>
<td>5097</td>
</tr>
<tr>
<td>5</td>
<td>5250</td>
</tr>
<tr>
<td>6</td>
<td>4897</td>
</tr>
<tr>
<td>7</td>
<td>5086</td>
</tr>
<tr>
<td>8</td>
<td>5139</td>
</tr>
<tr>
<td>9</td>
<td>5303</td>
</tr>
<tr>
<td>10</td>
<td>5205</td>
</tr>
<tr>
<td>MD</td>
<td>96,2</td>
</tr>
</tbody>
</table>

**Tabelle 10:** Referenzmessung, Revision 10 (3 VU).
Referenzmessung basierend auf Revision 10 mit drei Stressclients; 10 Läufe; MD ist die mittlere absolute Abweichung bezüglich des Medians.

### B.4 Zwei virtuelle Benutzer

#### B.4.1 Diagramme


![Diagramm](image)

**Abbildung 30:** Szenario add t-shirt to cart and continue shopping (2 VU).
Abbildung 31: Szenario add t-shirt to cart and increment quantity (2 VU).

Abbildung 32: Szenario browse through categories (2 VU).

Abbildung 33: Szenario show all brands (2 VU).
Abbildung 34: Szenario show all categories (2 VU).

Abbildung 35: Gesamtperformance (2 VU).
Das Diagramm stellt die Gesamtperformance dar, wobei zwei Messdurchgänge ausgeführt wurden. Die Ordinate enthält die Gesamtlaedezeit in Millisekunden. Die Abszisse stellt die Revisionen dar.

Abbildung 36: Referenzmessung basierend auf Revision 10 (Tabelle 11, 2 VU).
Die Ordinate enthält die Ladezeit in Millisekunden, auf der Abszisse sind die Durchläufe aufgetragen.
B.4.2 Messdaten

<table>
<thead>
<tr>
<th>Nummer</th>
<th>Revisionsbezeichner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>87dbf984933599e076db573a30e298686878615d4</td>
</tr>
<tr>
<td>2</td>
<td>dd114e1af0e3c1c9bcbb6b1e3f3bd2e6368d80e8</td>
</tr>
<tr>
<td>3</td>
<td>2236ad5932a65e4460c9d202c0297320c4f2d63e</td>
</tr>
<tr>
<td>4</td>
<td>12c1a46eda150dc6bd07276317c4a0b80acd450</td>
</tr>
<tr>
<td>5</td>
<td>e44f36e0a43c13a1b8ddc4dd9e9654e01d7cbeb0</td>
</tr>
<tr>
<td>6</td>
<td>302749476e3c58518deb0a7fdc5b315a7e2b033</td>
</tr>
<tr>
<td>7</td>
<td>667a13a7b6aeea605dbca016baa3a9d0e61c937c</td>
</tr>
<tr>
<td>8</td>
<td>d4f4c15e74a1d2c9b0f1d70969e146914b39600</td>
</tr>
<tr>
<td>9</td>
<td>8da74d8fd5b7f6d629671f14b164f42c556f480</td>
</tr>
<tr>
<td>10</td>
<td>dfcb54465e4cefd4e279938edab1979a57d2817</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Revision</th>
<th>1. Lauf</th>
<th>2. Lauf</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3594</td>
<td>3615</td>
</tr>
<tr>
<td>2</td>
<td>3632</td>
<td>3623</td>
</tr>
<tr>
<td>3</td>
<td>3675</td>
<td>3637</td>
</tr>
<tr>
<td>4</td>
<td>3839</td>
<td>3746</td>
</tr>
<tr>
<td>5</td>
<td>3709</td>
<td>3762</td>
</tr>
<tr>
<td>6</td>
<td>3852</td>
<td>3747</td>
</tr>
<tr>
<td>7</td>
<td>3846</td>
<td>3907</td>
</tr>
<tr>
<td>8</td>
<td>3896</td>
<td>3837</td>
</tr>
<tr>
<td>9</td>
<td>4057</td>
<td>4023</td>
</tr>
<tr>
<td>10</td>
<td>4261</td>
<td>4302</td>
</tr>
</tbody>
</table>

**Tabelle 12:** Ermittelte Gesamtperformance (2 VU). Gesamtperformance pro Revision in Millisekunden.
### Tabelle 13: Messwerte (2 VU): scenario show all categories

#1 und #2 stehen für den ersten bzw. zweiten Durchgang. Avg und med für den Durchschnittswert bzw. den Median. 2 VU meint, dass bei der Messung zwei virtuelle User (Stressclients) verwendet wurden.

<table>
<thead>
<tr>
<th>Revision</th>
<th>#1 avg</th>
<th>#2 avg</th>
<th>#1 med</th>
<th>#2 med</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>379</td>
<td>387</td>
<td>346</td>
<td>338</td>
</tr>
<tr>
<td>2</td>
<td>388</td>
<td>375</td>
<td>336</td>
<td>335</td>
</tr>
<tr>
<td>3</td>
<td>393</td>
<td>378</td>
<td>349</td>
<td>348</td>
</tr>
<tr>
<td>4</td>
<td>384</td>
<td>395</td>
<td>346</td>
<td>351</td>
</tr>
<tr>
<td>5</td>
<td>356</td>
<td>393</td>
<td>340</td>
<td>345</td>
</tr>
<tr>
<td>6</td>
<td>402</td>
<td>397</td>
<td>363</td>
<td>355</td>
</tr>
<tr>
<td>7</td>
<td>397</td>
<td>409</td>
<td>350</td>
<td>399</td>
</tr>
<tr>
<td>8</td>
<td>378</td>
<td>354</td>
<td>345</td>
<td>341</td>
</tr>
<tr>
<td>9</td>
<td>410</td>
<td>410</td>
<td>353</td>
<td>373</td>
</tr>
<tr>
<td>10</td>
<td>402</td>
<td>401</td>
<td>354</td>
<td>361</td>
</tr>
</tbody>
</table>

### Tabelle 14: Messwerte (2 VU): scenario browse through categories

#1 und #2 stehen für den ersten bzw. zweiten Durchgang. Avg und med für den Durchschnittswert bzw. den Median. 2 VU meint, dass bei der Messung zwei virtuelle User (Stressclients) verwendet wurden.

<table>
<thead>
<tr>
<th>Revision</th>
<th>#1 avg</th>
<th>#2 avg</th>
<th>#1 med</th>
<th>#2 med</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1439</td>
<td>1437</td>
<td>1407</td>
<td>1442</td>
</tr>
<tr>
<td>2</td>
<td>1457</td>
<td>1469</td>
<td>1458</td>
<td>1441</td>
</tr>
<tr>
<td>3</td>
<td>1491</td>
<td>1430</td>
<td>1481</td>
<td>1427</td>
</tr>
<tr>
<td>4</td>
<td>1570</td>
<td>1529</td>
<td>1554</td>
<td>1522</td>
</tr>
<tr>
<td>5</td>
<td>1547</td>
<td>1534</td>
<td>1518</td>
<td>1518</td>
</tr>
<tr>
<td>6</td>
<td>1537</td>
<td>1506</td>
<td>1529</td>
<td>1487</td>
</tr>
<tr>
<td>7</td>
<td>1516</td>
<td>1567</td>
<td>1509</td>
<td>1567</td>
</tr>
<tr>
<td>8</td>
<td>1550</td>
<td>1577</td>
<td>1539</td>
<td>1551</td>
</tr>
<tr>
<td>9</td>
<td>1676</td>
<td>1626</td>
<td>1632</td>
<td>1605</td>
</tr>
<tr>
<td>10</td>
<td>1720</td>
<td>1698</td>
<td>1694</td>
<td>1657</td>
</tr>
</tbody>
</table>
### Tabelle 15: Messwerte (2 VU): scenario show all brands

<table>
<thead>
<tr>
<th>Revision</th>
<th>#1 avg</th>
<th>#2 avg</th>
<th>#1 med</th>
<th>#2 med</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>309</td>
<td>334</td>
<td>297</td>
<td>299</td>
</tr>
<tr>
<td>2</td>
<td>311</td>
<td>304</td>
<td>299</td>
<td>298</td>
</tr>
<tr>
<td>3</td>
<td>305</td>
<td>315</td>
<td>300</td>
<td>298</td>
</tr>
<tr>
<td>4</td>
<td>321</td>
<td>337</td>
<td>303</td>
<td>304</td>
</tr>
<tr>
<td>5</td>
<td>310</td>
<td>318</td>
<td>303</td>
<td>304</td>
</tr>
<tr>
<td>6</td>
<td>325</td>
<td>346</td>
<td>313</td>
<td>319</td>
</tr>
<tr>
<td>7</td>
<td>313</td>
<td>333</td>
<td>306</td>
<td>307</td>
</tr>
<tr>
<td>8</td>
<td>322</td>
<td>325</td>
<td>305</td>
<td>305</td>
</tr>
<tr>
<td>9</td>
<td>335</td>
<td>361</td>
<td>311</td>
<td>315</td>
</tr>
<tr>
<td>10</td>
<td>342</td>
<td>344</td>
<td>314</td>
<td>320</td>
</tr>
</tbody>
</table>

#1 und #2 stehen für den ersten bzw. zweiten Durchgang. Avg und med für den Durchschnittswert bzw. den Median. 2 VU meint, dass bei der Messung zwei virtuelle User (Stressclients) verwendet wurden.

### Tabelle 16: Messwerte (2 VU): scenario add t-shirt to cart and increment quantity

<table>
<thead>
<tr>
<th>Revision</th>
<th>#1 avg</th>
<th>#2 avg</th>
<th>#1 med</th>
<th>#2 med</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>218</td>
<td>211</td>
<td>225</td>
<td>224</td>
</tr>
<tr>
<td>2</td>
<td>216</td>
<td>202</td>
<td>226</td>
<td>224</td>
</tr>
<tr>
<td>3</td>
<td>215</td>
<td>200</td>
<td>226</td>
<td>224</td>
</tr>
<tr>
<td>4</td>
<td>231</td>
<td>203</td>
<td>230</td>
<td>229</td>
</tr>
<tr>
<td>5</td>
<td>196</td>
<td>201</td>
<td>228</td>
<td>230</td>
</tr>
<tr>
<td>6</td>
<td>220</td>
<td>210</td>
<td>231</td>
<td>231</td>
</tr>
<tr>
<td>7</td>
<td>247</td>
<td>248</td>
<td>234</td>
<td>234</td>
</tr>
<tr>
<td>8</td>
<td>213</td>
<td>204</td>
<td>235</td>
<td>232</td>
</tr>
<tr>
<td>9</td>
<td>202</td>
<td>208</td>
<td>240</td>
<td>241</td>
</tr>
<tr>
<td>10</td>
<td>256</td>
<td>254</td>
<td>243</td>
<td>242</td>
</tr>
</tbody>
</table>

#1 und #2 stehen für den ersten bzw. zweiten Durchgang. Avg und med für den Durchschnittswert bzw. den Median. 2 VU meint, dass bei der Messung zwei virtuelle User (Stressclients) verwendet wurden.
Table 17: Messwerte (2 VU): scenario add t-shirt to cart and continue shopping
#1 und #2 stehen für den ersten bzw. zweiten Durchgang. Avg und med für den Durchschnittswert bzw. den Median. 2 VU meint, dass bei der Messung zwei virtuelle User (Stressclients) verwendet wurden.

<table>
<thead>
<tr>
<th>Durchgang</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4274</td>
</tr>
<tr>
<td>2</td>
<td>4286</td>
</tr>
<tr>
<td>3</td>
<td>4270</td>
</tr>
<tr>
<td>4</td>
<td>4234</td>
</tr>
<tr>
<td>5</td>
<td>4275</td>
</tr>
<tr>
<td>6</td>
<td>4295</td>
</tr>
<tr>
<td>7</td>
<td>4257</td>
</tr>
<tr>
<td>8</td>
<td>4264</td>
</tr>
<tr>
<td>9</td>
<td>4217</td>
</tr>
<tr>
<td>10</td>
<td>4204</td>
</tr>
<tr>
<td>MD</td>
<td>22,4</td>
</tr>
</tbody>
</table>

Table 18: Referenzmessung, Revision 10 (Tabelle 11, 2 VU).
Referenzmessung basierend auf Revision 10 (Tabelle 11) mit einem Stressclient; 10 Läufe; MD ist die mittlere absolute Abweichung bezüglich des Medians.
B.5 Gegenüberstellung Gesamtperformance Gruppe eins und zwei

<table>
<thead>
<tr>
<th>Revision</th>
<th>3 Stressclients</th>
<th>2 Stressclients</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Druchgang #1</td>
<td>Druchgang #2</td>
</tr>
<tr>
<td>1</td>
<td>4139</td>
<td>4158</td>
</tr>
<tr>
<td>2</td>
<td>4355</td>
<td>4344</td>
</tr>
<tr>
<td>3</td>
<td>4062</td>
<td>4128</td>
</tr>
<tr>
<td>4</td>
<td>4289</td>
<td>4266</td>
</tr>
<tr>
<td>5</td>
<td>4151</td>
<td>4217</td>
</tr>
<tr>
<td>6</td>
<td>4485</td>
<td>4635</td>
</tr>
<tr>
<td>7</td>
<td>4427</td>
<td>4540</td>
</tr>
<tr>
<td>8</td>
<td>4637</td>
<td>4715</td>
</tr>
<tr>
<td>9</td>
<td>4813</td>
<td>4741</td>
</tr>
<tr>
<td>10</td>
<td>5023</td>
<td>4965</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Revision</th>
<th>3 Stressclients</th>
<th>2 Stressclients</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Druchgang #1</td>
<td>Druchgang #2</td>
</tr>
<tr>
<td>1</td>
<td>3594</td>
<td>3615</td>
</tr>
<tr>
<td>2</td>
<td>3632</td>
<td>3623</td>
</tr>
<tr>
<td>3</td>
<td>3675</td>
<td>3637</td>
</tr>
<tr>
<td>4</td>
<td>3839</td>
<td>3746</td>
</tr>
<tr>
<td>5</td>
<td>3709</td>
<td>3762</td>
</tr>
<tr>
<td>6</td>
<td>3852</td>
<td>3747</td>
</tr>
<tr>
<td>7</td>
<td>3846</td>
<td>3907</td>
</tr>
<tr>
<td>8</td>
<td>3896</td>
<td>3837</td>
</tr>
<tr>
<td>9</td>
<td>4057</td>
<td>4023</td>
</tr>
<tr>
<td>10</td>
<td>4261</td>
<td>4302</td>
</tr>
</tbody>
</table>

**Tabelle 19:** Gegenüberstellung 3 vs. 2 Clients.

**Abbildung 37:** Diagramm der Gesamtperformance auf 3 vs. 2 Clients.
Gemäß Tabelle 19.
### Tabelle 20: Gegenüberstellung 3 vs. 2 Clients (Medianbasis).


<table>
<thead>
<tr>
<th>Revision</th>
<th>3 Stressclients</th>
<th>1 Stressclient</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Druchgang #1</td>
<td>Druchgang #2</td>
</tr>
<tr>
<td>1</td>
<td>4043</td>
<td>4030</td>
</tr>
<tr>
<td>2</td>
<td>4220</td>
<td>4096</td>
</tr>
<tr>
<td>3</td>
<td>3912</td>
<td>4088</td>
</tr>
<tr>
<td>4</td>
<td>4268</td>
<td>4118</td>
</tr>
<tr>
<td>5</td>
<td>4096</td>
<td>4059</td>
</tr>
<tr>
<td>6</td>
<td>4441</td>
<td>4511</td>
</tr>
<tr>
<td>7</td>
<td>4278</td>
<td>4547</td>
</tr>
<tr>
<td>8</td>
<td>4563</td>
<td>4597</td>
</tr>
<tr>
<td>9</td>
<td>4586</td>
<td>4511</td>
</tr>
<tr>
<td>10</td>
<td>4877</td>
<td>4765</td>
</tr>
</tbody>
</table>

### Abbildung 38: Diagramm der Gesamtpersonale 3 vs. 2 Clients (Medianbasis).

Gemäß Tabelle 20.
C Anhang Datenträger


Die ContinuousPerformanceTesting-Erweiterung zur Untersuchung eines Softwareprojekts befindet sich im Verzeichnis Erweiterung/spree.

Im Unterordner spree ist die Kopie des Repositoriums der Versionsverwaltung inkludiert. Das Projekt wurde zu Evaluierungszwecken eingesetzt. Enthalten sind auch die bei der Evaluierung erstellten Skripte zur automatisierten Testconfiguration. Sie befinden sich im Unterordner lib/tasks des Projekts.

Der Ordner messergebnisse beherbergt den Datenbankauszug mit allen während der Evaluierung erstellten Messungen. Dabei handelt es sich um die Datenbank des Prototypen ContinuousPerformanceTesting.

Eine elektronische Version der Ausarbeitung im Portable Document Format (kurz PDF) befindet sich im Verzeichnis ausarbeitung.
Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit mit dem Titel „Methoden zur kontinuierlichen Performancemessung bei der agilen Entwicklung von Webanwendungen“ ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Berlin, 24. September 2009

Ort, Datum

Unterschrift

XXXV