Diplomarbeit

Konzeption und Implementation eines Werkzeugs zur Unterstützung partizipativer Benutzbarkeitstests von Websites

Torsten Haß
im Januar 2004

Betreuer:
Prof. Dr. Horst Oberquelle
Prof. Dr. Winfried Lamersdorf

Universität Hamburg
Fachbereich Informatik
Arbeitsbereich Angewandte und Sozialorientierte Informatik (ASI)
Arbeitsbereich Verteilte Systeme und Informationssysteme (VSIS)

Danksagung

Mein Dank gilt besonders Hartmut Obendorf und Harald Weinreich, die mir sowohl für die Programmierung als auch für die schriftliche Arbeit wertvolle Tipps gaben. Des Weiteren möchte ich mich bei Frank Buhr, Antje Großmann, Frank Hohenschuh, Matthias Hultsch und Moritz Petersen für die Hilfe bei der Evaluation des UserTest-Tools bedanken. Meiner Frau Aleksandra danke ich für ihre Geduld und Unterstützung.

Inhaltsverzeichnis

1 Einleitung

Benutzertests sind inzwischen ein etabliertes Mittel, um Gebrauchstauglichkeitsprobleme zu entdecken. Dabei werden mit dieser Testmethode gerade die Probleme gefunden, die den Benutzer bei der praktischen Anwendung behindern. Allerdings sind diese Tests bis heute mit dem Vorurteil behaftet, aufwändig und teuer zu sein, da die Äußerungen des Teilnehmers analysiert, seine Aktionen, seine Eingaben und die besuchten Webseiten protokolliert und später ausgewertet werden müssen. Zudem wird von den Testleitern eine hohe Expertise erwartet.

Im klassischen Fall eines Benutzertests wird der Test mit einer Videokamera aufgezeichnet. Auf diese Weise muss sich der Testleiter nicht um das Protokollieren der Aktionen und Aussagen des Teilnehmers kümmern, da alle relevanten Daten von der Videokamera festgehalten werden. Er sorgt dafür, dass der Teilnehmer fortwährend seine Gedanken verbalisiert, hilft ihm weiter, falls er auftauchende Probleme nicht selbst lösen kann und sorgt im Fall früher explorativer Prototypen für den reibungslosen Testablauf. Der aufwändige Teil dieser Methode folgt nach den Testläufen, wenn der Testleiter die Videoaufzeichnungen viele Male durchgehen muss, um in Ruhe das Verhalten des Teilnehmers zu studieren, seine Aktionen auf den einzelnen Webseiten sowie seine Wege durch das Web nachzuvollziehen. Alle für den Test relevanten Aktionen des Teilnehmers müssen protokolliert werden, um sie mit den Aktionen anderer Teilnehmer vergleichen zu können. Dieses Analysieren kostet im Allgemeinen ein Vielfaches der Zeit des eigentlichen Testlaufs mit dem Teilnehmer.

Es gibt weniger aufwändige Variationen des Benutzertests, wie das *Discount Usability Engineering* von Jakob Nielsen, in dem beispielsweise auf die Videoaufzeichnung und die nachträgliche Analyse verzichtet wird (vgl. Nielsen 1993, Seite 17). Derartige Methoden sind wegen der ungenaueren Analyse allerdings weniger ergiebig als die klassische Variante.

Viele der Aufgaben des Testleiters sind Routineaufgaben, die ebenso von einem Computer erledigt werden können. Zum Beispiel lassen sich die Aktionen des Teilnehmers, wie angeklickte Links oder Interaktionen mit dem Browser, durch den Computer abfangen und speichern. Auch lässt sich etwa das Messen der für eine Aufgabe benötigte Zeit vom Computer übernehmen. Die gesammelten Daten können dem Testleiter dann in beliebiger Form bereitgestellt werden.

Das Ziel dieser Arbeit ist, zu ergründen, welche Routineaufgaben eines Tests mit Benutzerbeteiligung vom Computer übernommen werden können. Befreit von diesen Aufgaben kann sich der Testleiter ganz auf das Durchführen des Tests und das Deuten der Interface-Probleme konzentrieren. Auf diese Weise sollte sich der große zeitliche Aufwand, der viele Entwickler als potentielle Benutzer dieser Testmethode abschreckt, verkleinern lassen.

Kapitel zwei definiert den Begriff der Gebrauchstauglichkeit und geht genauer auf den Benutzertest und einige andere geläufige Testmethoden für Gebrauchstauglichkeit ein. Außerdem werden andere Arbeiten angegeben, die auf unterschiedliche Weise die Aktionen des Teilnehmers abgefangen haben.

In dem dritten Kapitel werden die einzelnen Aufgaben des Testleiters daraufhin untersucht, inwieweit es sinnvoll ist, diese von einem Computer ausführen zu lassen. So ist es beispielsweise sinnvoll, die Aktionen des Teilnehmers durch den Computer protokollieren zu lassen. Die Deutung der Probleme des Teilnehmers sollte aber weiterhin dem Testleiter vorbehalten sein.

Im Rahmen dieser Arbeit wurde ein Prototyp entwickelt, der die Routineaufgaben des Testleiters übernehmen und ihn so bei der Durchführung von Website-Tests mit Benutzern unterstützen soll. Die Implementation wird im vierten Kapitel behandelt. Das entwickelte Werkzeug mit dem Namen „UserTestTool“ stellt ein Aufgabenfenster bereit, in dem Aufgabenbeschreibungen und Fragebögen für den Teilnehmer angezeigt werden können. Über Eingabefelder, Wertungsskalen und Textauswahllisten kann der Teilnehmer beispielsweise Aufgabenlösungen eingeben oder persönliche Meinungen abgeben. Die Antworten und Aktionen des Teilnehmers werden für eine spätere Auswertung gespeichert. Das UserTestTool erlaubt des Weiteren die Steuerung des Internet Explorers und das Abfangen eines Großteils der Aktionen des Teilnehmers beim Surfen im Web. Beispielsweise werden die Aktionen „angeklickter Link“, „Aktualisieren“, „Eingeben einer URL“ sowie das „Vor- und Zurückspringen innerhalb der besuchten Webseiten“ gespeichert. Außerdem werden die URIs aller besuchten Webseiten mit dem Zeitpunkt ihres Aufrufs festgehalten.

Alle gespeicherten Aktionen und Eingaben des Teilnehmers werden in Protokolldateien abgelegt, die sich zur späteren Auswertung beispielsweise in viele Tabellenkalkulationsprogramme importieren lassen. Auf diese Weise bleibt dem Testleiter das nachträgliche Suchen der Benutzeraktionen im Video erspart und er kann sich voll auf den eigentlichen Test konzentrieren.

Der Inhalt der einzelnen Aufgabenfenster des Tests lässt sich durch eine XML-Datei festlegen, in der die benötigten Ein- und Ausgabekomponenten definiert und konfiguriert werden. Es wurden bewusst nur einige elementare Komponenten zur Verfügung gestellt, um das Erstellen derartiger XML-Dateien möglichst einfach zu halten. Diese wenigen Komponenten lassen sich aber in Aussehen und Funktion an den jeweiligen Zweck anpassen. Zu diesen Komponenten gehört eine Textausgabe, ein Texteingabefeld, eine Likert-Skala zum Abgeben von Wertungen, eine Drop-DownBox zur Auswahl von vorgegebenen Texten und eine Schaltflächenkomponente, mit der sich beispielsweise andere Komponenten hervorheben oder Webseiten im Browser aufrufen lassen. Das UserTestTool wurde so entwickelt, dass es sich leicht um zusätzliche Komponenten erweitern lässt.

Das fünfte Kapitel beschreibt mehrere Evaluationsschritte, in denen das UserTestTool auf Gebrauchstauglichkeit und Fehler untersucht wurde. Dadurch ergaben sich, zusätzlich zu den erwarteten Problemen der Gebrauchstauglichkeit, auch viele Anregungen, die nach einer Betrachtung ihrer Vor- und Nachteile größtenteils in das UserTestTool übernommen wurden.

Kapitel sechs fasst diese Arbeit zusammen und Kapitel sieben gibt einen Ausblick auf weiterführende Arbeiten. Anschließend wird die Liste der in dieser Arbeit verwendeten Literaturquellen angegeben.

Im Anhang A dieser Arbeit befindet sich eine Benutzerdokumentation für das UserTestTool, gefolgt von einer Komponentenreferenz für die XML-Testbeschreibungsdatei (Anhang B) und einer Beschreibung der möglichen Einträge der Protokolldatei (Anhang C). Der Anhang D führt den Inhalt der beiliegenden Daten-CD auf, die unter anderem eine lauffähige Version des UserTestTools enthält.

2 Grundlagen

In diesem Kapitel wird der Begriff der Gebrauchstauglichkeit eingeführt und es werden die geläufigsten Testmethoden zum Testen auf Gebrauchstauglichkeit vorgestellt. Abschnitt 2.1 gibt einen Überblick über die Gebrauchstauglichkeit und ihre Entstehung. Abschnitt 2.2 gibt Kriterien zur Klassifikation von Testmethoden an. Die vier geläufigsten Testmethoden werden im Abschnitt 2.3 vorgestellt und im Abschnitt 2.4 miteinander verglichen.

2.1 Entstehung der Gebrauchstauglichkeit

Es gibt viele Methoden, um Software oder Webseiten auf Gebrauchstauglichkeit zu testen. Abschnitt 2.1.1 gibt einen historischen Überblick über die Anfänge gebrauchstauglicher Software. Abschnitt 2.1.2 geht näher auf die menschlichen Merkmale ein. Abschnitt 2.1.3 definiert den Begriff Gebrauchstauglichkeit. Auf die Bedeutung von Gebrauchstauglichkeit für Software und Websites geht Abschnitt 2.1.4 ein. Der zum Testen nötige Kontext, wie typische Aufgaben des zu testenden Systems und das Wissen potentieller Benutzer, wird im Abschnitt 2.1.5 behandelt.

2.1.1 Historischer Überblick

Für die Entwickler der ersten Rechenanlagen und Programme, bis ca. 1960, spielten Gebrauchstauglichkeit und Software-Ergonomie noch so gut wie keine Rolle. Der Benutzer hatte das Programm entweder selbst entwickelt oder er war ein Experte für derartige Anwendungen und konnte sich mit Hilfe seines Wissens und seiner Erfahrung in das Programm einarbeiten. Im Vordergrund der damaligen Programme stand die Funktionalität der Systeme, da die Rechenzeit sehr viel teurer war, als die Arbeitszeit der Anwender.

Zwischen 1960 und 1980 wurden die damaligen Computer, im Allgemeinen Großrechner, nur von Experten bedient, die sich an die Systeme anpassen konnten und wollten. Es gab vereinzelte Versuche, die Mensch-Maschine-Kommunikation zu verbessern. Beispielsweise stellte Joseph C. R. Licklider in seinem Paper „Man-Computer

symbiosis“ (vgl. Licklider 1960) die unterschiedlichen Arten der Datenverarbeitung des Menschen und der des Computers einander gegenüber und suchte nach Wegen, eine engere Zusammenarbeit zwischen Mensch und Computer zu ermöglichen. 1963 entwickelte Ivan E. Sutherland am MIT eine Software, die es erlaubte, Grafiken in Echtzeit mit Hilfe eines Lightpens zu manipulieren. Mit seiner Arbeit „Sketchpad: A Man-Machine Graphical Communications System“ (vgl. Sutherland 1963) legte er damit nicht nur den Grundstein für Computer Aided Design (CAD), sondern erlaubte Ingenieuren ohne Computerkenntnisse, geometrische Figuren am Bildschirm zu zeichnen und zu verändern, um komplexe Probleme zu lösen. Diese Ansätze von Licklider und Sutherland gehörten zu den ersten Bestrebungen, Menschen die Arbeit mit dem Computer zu ermöglichen, die über keine fundierten Programmierkenntnisse verfügten. Nur wenige sahen die Notwendigkeit, die damals teure Rechenzeit zu benutzen, um dem Benutzer eine grafische Oberfläche anzubieten. Es dauerte aber noch Jahre, bis die Programme so benutzbar waren, dass Benutzer ohne spezielle Computerkenntnisse diese bedienen konnten.

Die ersten Bestrebungen, kommerzielle Software etwas gebrauchstauglicher zu machen, begannen ab 1980, denn nun sollten auch Arbeitnehmer ohne entsprechende Datenverarbeitungsqualifikation über Terminals auf die zentrale Datenverarbeitung der Firmen zugreifen. Unter dem Schlagwort „Human Factors“ sollte die Interaktion von Mensch und Maschine verbessert werden, indem Informationen über das Verhalten, die Möglichkeiten und Beschränkungen des Menschen gesammelt wurden. Diese Merkmale, wie die spezifische Wahrnehmung, das beschränkte und unzuverlässige Gedächtnis, die unterschiedlichen Lerntypen, motorische Fähigkeiten und vieles mehr, flossen in die Entwicklung von Hard- und Softwareprodukten ein. Zum Erforschen dieser Merkmale wurden in den frühen 80er Jahren Testzentren von großen Softwarefirmen, z. B. IBM, errichtet (vgl. Shneiderman 1998, Seite 128).

Inzwischen ist ein Großteil der Berufswelt auf Computerunterstützung angewiesen. Firmen, die Computersysteme einsetzen, achten beim Kauf von Software auf leichte Bedienbarkeit, um sich die kostspieligen Schulungen für die Angestellten zu sparen. Auch im privaten Bereich ist der Computer immer öfter anzutreffen. Die Anwendungen im Haushalt reichen vom Schreiben von Briefen bis zum Bestellen von Produkten im Inter-

net. Ein zunehmender Anteil der Benutzer sind Laien, die weder Zeit noch Lust haben, sich in komplizierte Programme einzuarbeiten oder deren Dokumentationen zu lesen. Programme und Websites sollen verständlich und einfach zu bedienen sein, sonst werden sie von den Benutzern nicht akzeptiert.

2.1.2 Menschliche Merkmale

Um Softwaresysteme einfacher bedienbar zu machen, mussten sie an die menschlichen Merkmale angepasst werden. Zu diesen Merkmalen gehören

- eine spezifische Wahrnehmung
- die motorischen Fähigkeiten
- die Eigenschaften des Kurz- und Langzeitgedächtnisses
- gelegentliche fehlerhafte Handlungen
- unterschiedliche Lerntypen (visuell, auditiv, haptisch)

Die Berücksichtigung dieser Merkmale war auch ein großer Bestandteil der Human-Factors-Forschung, die in den frühen 80er Jahren in den Testzentren als Grundlage diente, um die Interaktion zwischen Mensch und Computer zu verbessern.

Weitere Merkmale des Menschen sind das individuelle Wissen und das mentale Modell, das sich der Benutzer von der Aufgabe geschaffen hat. Das mentale Modell hilft dem Benutzer beim Verstehen komplexer Aufgaben und dient ihm als Grundlage zur Planung und Steuerung von Handlungen (vgl. Dutke 1994, Kapitel 1). Ein fehlerhaftes mentales Modell einer Aufgabe kann zu fehlerhaften Voraussagen und Handlungen führen.

2.1.3 Gebrauchstauglichkeit

Usability von Softwaresystemen ist in aller Munde. Der Begriff „Usability“ setzt sich aus den englischen Worten „to use“ (benutzen) und „ability“ (die Fähigkeit) zusammen und wird mit Gebrauchstauglichkeit übersetzt. Mit anderen Worten: Hat ein Software-

produkt eine hohe Usability, dann lässt es sich ohne Probleme benutzen (vgl. Institut für Software-Ergonomie und Usability 2002). Auch die deutsche Ausgabe der ISO-Norm 9241 Teil 11, auf die in diesem Abschnitt noch näher eingegangen wird, übersetzt den in der englischen Version benutzten Begriff Usability mit Gebrauchstauglichkeit (vgl. ISO 9241-11, 1998, Seite 1).

Die „Ergonomischen Anforderungen für Bürotätigkeiten mit Bildschirmgeräten“ wurden 1995 in der ISO-Norm 9241 zum internationalen Standard erhoben. Die Teile 3-9 dieser Norm behandeln die Ergonomie von Computerarbeitsplätzen, insbesondere die Anforderungen an die Hardware, auf die in dieser Arbeit nicht näher eingegangen wird. Teil 11 definiert den Begriff der Gebrauchstauglichkeit und gibt Kriterien für die Beschaffung, den Entwurf, die Entwicklung und die Evaluation von gebrauchstauglichen Produkten. Teil 10 präzisiert den Aspekt der Gebrauchstauglichkeit durch Angabe von Grundsätzen zur Dialoggestaltung. Die Teile 12-17 gehen auf die Gestaltung spezieller Interaktionsformen, wie Menüs, Bildschirmformulare und andere, ein.

Teil 11 dieser ISO-Norm definiert den Begriff der Gebrauchstauglichkeit folgendermaßen: „Das Ausmaß, in dem ein Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen.“ (ISO 9241-11, 1998, Seite 4).

Die ISO-Norm 9241-10 gibt sieben Eigenschaften an, die ein gebrauchstaugliches Dialogsystem definieren: Aufgabenangemessenheit, Selbstbeschreibungsfähigkeit, Steuerbarkeit, Erwartungskonformität, Fehlertoleranz, Individualisierbarkeit und Lernförderlichkeit. Diese Eigenschaften werden in den folgenden Grundsätzen definiert:

- Aufgabenangemessenheit: „Ein Dialog ist aufgabenangemessen, wenn er den Benutzer unterstützt, seine Arbeitsaufgabe effektiv und effizient zu erledigen“ (ISO 9241-10, 1996, Seite 4). Des Weiteren werden in diesem Abschnitt der Norm beispielsweise die folgenden Empfehlungen gegeben: Dem Benutzer sollen nur relevante Informationen gezeigt werden. Er soll nicht mit Aufgaben belastet werden, die vom System erledigt werden können. Wenn möglich, sollen Eingabefelder mit sinnvollen Standardwerten vorbelegt sein.

- Selbstbeschreibungsfähigkeit: „Ein Dialog ist selbstbeschreibungsfähig, wenn jeder einzelne Dialogschritt durch Rückmeldung des Dialogsystems unmittelbar verständlich ist oder dem Benutzer auf Anfrage erklärt wird.“ (ISO 9241-10, 1996, Seite 5). Zusätzlich werden folgende Empfehlungen ausgesprochen: Das System sollte dem Benutzer durch Rückmeldungen zeigen, was durch seine Aktion ausgelöst wurde. Vor Entscheidungen mit schwerwiegenden Folgen sollte der Benutzer vorher gewarnt werden. Rückmeldungen, Erklärungen und Hilfen sollten in einheitlicher Terminologie in dem Vokabular des Anwenders dargestellt werden. Rückmeldungen sollten genau auf die Situation zugeschnitten sein. Nach Möglichkeit sollten Standardwerte vorgegeben werden. Erwartete Eingaben sollten mit Erklärungen versehen werden. Meldungen sollten konstruktiv, ohne wertende Urteile sein.

- Steuerbarkeit: „Ein Dialog ist steuerbar, wenn der Benutzer in der Lage ist, den Dialogablauf zu starten, sowie seine Richtung und Geschwindigkeit zu beeinflussen, bis das Ziel erreicht ist.“ (ISO 9241-10, 1996, Seite 6). Folgende zusätzliche Empfehlungen werden gegeben: Dem Benutzer sollte keine Geschwindigkeit vorgegeben werden. Unterbrochene Dialoge sollten wieder aufnehmbar sein. Im Fall von mehreren Eingabeformularen sollte das Hin- und Herspringen zwischen den Formularen möglich sein.

- Erwartungskonformität: „Ein Dialog ist erwartungskonform, wenn er konsistent ist und den Merkmalen des Benutzers entspricht, z. B. seinen Kenntnissen aus dem Arbeitsgebiet, seiner Ausbildung und seiner Erfahrung, sowie den allgemein anerkannten Konventionen.“ (ISO 9241-10, 1996, Seite 6). Folgende weitere Empfehlungen werden ausgesprochen: Das Dialogverhalten und die Informationsdarstellung sollten einheitlich gehalten sein. Es sollte der Wortschatz des Benutzers für die Tätigkeit verwandt werden. Ähnliche Aufgaben sollten durch ähnliche Dialoge dargestellt werden. Die Eingabemarke sollte dort stehen, wo die Eingabe erwartet wird. Auf längere Wartezeiten sollte hingewiesen werden.

- Fehlertoleranz: „Ein Dialog ist fehlertolerant, wenn das beabsichtigte Arbeitsergebnis trotz erkennbar fehlerhafter Eingaben entweder mit keinem oder mit minimalem Korrekturaufwand seitens des Benutzers erreicht werden kann. (ISO 9241-10, 1996, Seite 7). Zu den weiteren Empfehlungen gehören: Fehler sollten dem Benutzer zu Korrekturzwecken erläutert werden. Der Dialog sollte beim Entdecken und Vermeiden von Fehler unterstützen. Es sollte verhindert werden, dass Fehleingaben zu Abstürzen führen.

- Individualisierbarkeit: „Ein Dialog ist individualisierbar, wenn das Dialogsystem Anpassungen an die Erfordernisse der Arbeitsaufgabe, sowie an die individuellen Fähigkeiten und Vorlieben des Benutzers zulässt.“ (ISO 9241-10, 1996, Seite 8). Des Weiteren werden folgende Empfehlungen ausgesprochen: Das Dialogsystem sollte sich an Sprache und kulturelle Eigenheiten, individuelles Wissen, Erfahrungen, Wahrnehmungsvermögen, sensomotorische- und geistige Fähigkeiten des Benutzers anpassen. Darstellung und Format sollten änderbar sein. Der Umfang von Erläuterungen sollte änderbar sein. Das System sollte unterschiedliche Dialogtechniken anwenden, z. B. Tastaturkürzel für Fortgeschrittene.

- Lernförderlichkeit: „Ein Dialog ist lernförderlich, wenn er den Benutzer beim Erlernen des Dialogsystems unterstützt und anleitet.“ (ISO 9241-10, 1996, Seite 9).

Die ISO-Norm 9241 wurde 1995 verfasst und später erweitert. Einige Jahre früher gab Ben Shneiderman (vgl. Shneiderman 1992, Seite 72f), einer der Pioniere auf dem Gebiet der Gebrauchstauglichkeit, acht goldene Regeln für eine gebrauchstaugliche Benutzungsschnittstelle an.

- Strebe nach Konsistenz

- Erlaube Abkürzungen

- Gib informatives Feedback

- Entwirf abgeschlossene Dialoge
- Erlaube einfache Fehlerbehandlung
- Ermögliche Rückschritte, um ausgeführte Aktionen rückgängig zu machen
- Gib dem Benutzer das Gefühl der Kontrolle
- Reduziere die Belastung des Kurzzeitgedächtnisses.

Diese acht Regeln von Ben Shneiderman decken sich ziemlich genau mit den sieben Grundsätzen der ISO-Norm 9241-10. Da die Texte der ISO-Norm sehr viel ausführlicher als Ben Shneidermans acht goldene Regeln sind, erwähnt dieser meist nur einen Aspekt aus den Grundsätzen der Dialoggestaltung. So weisen sowohl seine erste Regel: „Strebe nach Konsistenz“ als auch seine letzte Regel: „Reduziere die Belastung des Kurzzeitgedächtnisses“ auf einige Aspekte der Erwartungskonformität hin. Auch seine Regeln: „Erlaube Abkürzungen“ und „Erlaube einfache Fehlerbehandlung“ sprechen nur Teilaspekte der Grundsätze Individualisierbarkeit und Fehlertoleranz an. Im Gegensatz dazu decken sich seine Regeln: „Gib informatives Feedback“ und „Gib dem Benutzer das Gefühl der Kontrolle“ recht genau mit den Grundsätzen Selbstbeschreibungsfähigkeit und Steuerbarkeit. Die Regel: „Ermögliche Rückschritte, um ausgeführte Aktionen rückgängig machen zu können“ lässt sich sowohl auf Steuerbarkeit als auch auf Fehlertoleranz abbilden. Lediglich die Regel: „Entwirf abgeschlossene Dialoge“ lässt sich nur schwer in die Grundsätze einordnen. Sie ist entfernt mit der Erwartungskonformität verwandt. Die Grundsätze Aufgabenangemessenheit und Lernförderlichkeit werden von Shneiderman nicht angesprochen.

Ein Jahr später definierte Jakob Nielsen Gebrauchstauglichkeit durch die folgenden 5 Punkte (vgl. Nielsen 1993, Seite 26):

- Leicht erlernbar: Der Benutzer kann nach kurzer Zeit sinnvoll mit dem Softwareprodukt arbeiten.

- Leicht erinnerbar: Benutzer, die das System selten bedienen, finden sich auch nach längeren Pausen wieder schnell zurecht, ohne die Benutzung neu erlernen zu müssen.

- Effizienz: Nach der Einarbeitung lässt sich das System effizient nutzen.

- Wenige Fehler: Das System ist so gebaut, dass der Benutzer wenig Fehler macht. Wenn er Fehler macht, lassen sich diese rückgängig machen. Der Benutzer kann keine Fehler mit katastrophalen Folgen machen.

- Angenehm zu benutzen: Der Benutzer ist zufrieden, wenn er das System benutzt.

Bei dem Vergleich der fünf Punkte von Jakob Nielsen mit den Grundsätzen der ISO-Norm lassen sich größere Abweichungen feststellen. Das liegt unter anderem daran, dass er, im Gegensatz zur ISO-Norm mit ihren geforderten Grundsätzen, messbare Größen angibt, die sich in einem Test für Gebrauchstauglichkeit bestimmen lassen. Trotzdem lassen sich einige seiner Punkte direkt auf die Grundsätze abbilden: „Leicht erlernbar...“ passt auf den Grundsatz der Lernförderlichkeit, „Wenige Fehler...“ entspricht dem Grundsatz Fehlertoleranz und der Punkt „Effizienz“ deckt einen Großteil des Grundsatzes Aufgabenangemessenheit ab. Der Punkt „Angenehm zu benutzen“ und die darin enthaltene Zufriedenheit während der Systembenutzung lässt sich in der Definition der Gebrauchstauglichkeit wiederfinden (vgl. ISO 9241-11, 1998, Seite 4). Lediglich der Punkt „Leicht erinnerbar...“ besitzt kein passendes Gegenstück in der ISO-Norm. Die Grundsätze Steuerbarkeit, Erwartungskonformität und Individualisierbarkeit lassen sich in Nielsens fünf Punkten nicht direkt wiederfinden, indirekt sind sie aber in seinem Punkt „Effizienz“ enthalten.

2.1.4 Bedeutung von Gebrauchstauglichkeitstests

Zum Testen von Software und Websites auf Gebrauchstauglichkeit gibt es eine Reihe von Testmethoden. Wegen des z. T. erheblichen Aufwands werden derartige Tests aber zu selten angewandt (vgl. Nielsen 1993, Seite 16), obwohl sie dringend nötig sind.

Bei Software-Produkten, die auf klassische Weise im Geschäft gekauft werden, sind Gebrauchstauglichkeitstests nötig, um beim Anbieter die Kosten für Benutzerbetreuung niedrig zu halten. Eine gebrauchsuntaugliche Software wird die Kosten für eine Kunden-Hotline oder einen Helpdesk in die Höhe treiben. Außerdem wird Software von den Benutzern nicht gekauft, wenn diese beispielsweise durch Zeitschriften bewertet und für schlecht befunden wurde.

Bei Softwareprodukten, die über das Internet vertrieben werden, ist neben der Gebrauchstauglichkeit des Produkts auch die Gebrauchstauglichkeit der Website, die das Produkt anbietet, von Bedeutung. Das Internet ist inzwischen zu einem massiven ökonomischen Faktor geworden. Viele Anbieter vertreiben ihre Produkte bereits über eigene Websites. Sind derartige Websites aber nur wenig gebrauchstauglich, dann wird der Benutzer das gesuchte Produkt nicht finden und die Seite verlassen.

Websites werden noch für viele andere Zwecke eingesetzt. In den meisten Fällen bieten sie dem Benutzer Informationen an. Auch solche Websites müssen gebrauchstauglich aufgebaut sein, damit der Benutzer die Informationen finden und lesen kann. David Siegel (Siegel 1998, Seite 8) schreibt dazu: „Wen interessiert schon die Mächtigkeit einer Datenbank, wenn der Anwender nicht mit der Benutzerschnittstelle umgehen kann? Wer liest schon einen interessanten Inhalt, wenn er nicht attraktiv gestaltet oder nur schwer lesbar ist?“

2.1.5 Testen im Kontext

Zum Testen eines Produkts ist es unerlässlich, den Kontext der Produktbenutzung zu erfassen. Dazu gehört die Art, in der der Benutzer eine Aufgabe mit dem Produkt löst, und die Erfahrung und das Vorwissen der späteren Benutzer.

Das Erkennen der Aufgaben, die typischerweise mit dem System erledigt werden, ist nicht nur für den Entwicklungsprozess wichtig. Beim Testen von Systemen sollten es solche typischen Aufgaben sein, die der Experte für Gebrauchstauglichkeit oder der Testteilnehmer während des Tests bearbeiten soll. Auf diese Weise wird ein großer und häufig benutzter Teil des Systems getestet. Falls es mehrere Wege gibt, eine Aufgabe auszuführen, sollten diese alle getestet werden, um nicht bestimmte, möglicherweise oft benutzte Teile des Systems zu übergehen.

Anders als bei den menschlichen Merkmalen, die bei jedem Menschen annähernd gleich stark auftreten, verhält es sich mit den Eigenschaften der einzelnen Benutzer, die individuell verschieden sind. Dazu gehören (möglicherweise unvollständige) mentale Modelle von dem Dialogsystem und der zu erfüllenden Aufgabe, sowie persönliche Erfahrungen, angeeignetes Wissen und das Vokabular, das mit der Aufgabe verknüpft wird. Derartige Informationen über die späteren Benutzer müssen sorgfältig ermittelt werden, damit die Gebrauchstauglichkeitstests richtig eingesetzt werden können. Nur dann zeigen derartige Tests die gewünschte Wirkung.

So ist es beispielsweise unsinnig, ein Programm zur Verwaltung von Kochrezepten, das für den Gelegenheitsbenutzer im Privatbereich gedacht ist, in einem Benutzertest mit einem Experten für Datenbankanwendungen zu testen. Dieser Benutzer kennt das mentale Modell, das dem Programm zugrunde liegt, und er wird sich auch nicht an Phrasen wie „Neuen Datensatz anlegen“ stören. Der Gelegenheitsbenutzer hat im Allgemeinen keine Erfahrung mit Datenbanken und möchte sicherlich lieber ein Kochrezept hinzufügen als einen Datensatz anzulegen.

Ist das Interface für unterschiedliche Benutzergruppen gedacht, sollte beim Test zumindest die Gruppe mit dem geringsten Hintergrundwissen berücksichtigt werden. Auf diese Weise werden mehr Benutzer mit dem System umgehen können, als wenn ein Benutzer mit erweitertem Wissen als Grundlage zum Testen dient. Sofern es möglich ist, sollten aber auch andere Benutzergruppen getestet werden, um z. B. Abkürzungen für fortgeschrittene Benutzer nicht außer Acht zu lassen.

Im Fall einer zu testenden Website ist die Frage nach der Zielgruppe oft nicht genau zu beantworten, da viele Webanwendungen von einer heterogenen Benutzergruppe genutzt

werden. Petra Vogt schrieb dazu in dem Buch „Usability praktisch umsetzen“ (Heinsen und Vogt 2003, Seite 236).:

> „Websites werden im Verhältnis zu klassischer Software oft in einem viel breiteren und verschiedenartigeren Kontext genutzt, da sie meist eine breitere Zielgruppe bedienen und nicht nur für die Lösung einer klar umrissenen Spezialaufgabe konzipiert wurden. Dadurch unterliegen sowohl Nutzungsort, Ausstattung der Benutzer, Vorgehensweisen und Ziele der Nutzer einer größeren Streubreite“

Um die Gebrauchstauglichkeit einer derartigen Website zu bestimmen, müssen die unterschiedlichen Aufgaben und Nutzergruppen bestimmt und diese in den Test mit einbezogen werden.

Ein weiteres Problem bei dem Testen von Websites ist die Vielzahl der heute vorhandenen Hard- und Software. Werner Schweibenz und Frank Thissen (Schweibenz und Thissen 2003) schrieben dazu:

> „Das bedeutet, dass der Ersteller von Webseiten – im Gegensatz zum Printbereich – keine Möglichkeit hat, das Ergebnis, das der Nutzer dieser Web-Seiten sieht, eindeutig festzulegen, denn Nutzer verfügen über unterschiedliche Betriebssysteme und Monitore und haben diese auch noch unterschiedlich eingestellt [...]“

Daher sollten Websites auch mit unterschiedlichen Bildschirmauflösungen, Betriebssystemen und Browserversionen getestet werden.

2.2 Klassifikation von Testmethoden

Es gibt viele Möglichkeiten, nach denen die unterschiedlichen Testmethoden eingeteilt werden können. Die drei geläufigsten Methoden werden in den folgenden Abschnitten angegeben.

Testmethoden lassen sich allerdings nicht immer eindeutig klassifizieren, da sie – je nach Einsatz – unterschiedliche Daten produzieren oder im Entwicklungsprozess anders

angewendet werden können. Ein Beispiel dafür ist die Klassifikation nach dem Einsatz im Entwicklungsprozess.

Seit einigen Jahren geht der Trend in der Softwareentwicklung zum Prototyping. Im Prototyping werden schon früh in der Entwurfsphase lauffähige Prototypen erstellt, die dann in dem zyklischen Entwicklungsprozess immer wieder getestet und verbessert werden. **Formative Testmethoden** finden Probleme in der Benutzungsschnittstelle und bieten Lösungsvorschläge, die in den nächsten Prototyp einfließen. Je früher ein Fehler im Entwicklungsprozess gefunden wird, desto einfacher ist er zu beheben. So lassen sich gravierende Fehler auf diese Weise sehr früh feststellen und beheben, ohne die komplette Benutzungsschnittstelle am Ende der Entwicklung völlig umstrukturieren zu müssen. **Summative Testmethoden** bewerten das Interface am Ende des Entwicklungsprozesses. Durch sie lässt sich bestimmen, ob das Interface die gesetzten Ziele erfüllt.

Viele Testmethoden, beispielsweise Tests mit Benutzern, lassen sich sowohl formativ als auch summativ einsetzen. Ausschlaggebend sind dabei die Fragestellungen, die durch den Test beantwortet werden sollen. In einem **formativen Test** sollen Änderungsvorschläge gesammelt werden, die in den nächsten Prototypen einfließen sollen. Dazu werden Rückschlüsse auf die Probleme gezogen, die den Teilnehmer bei der Benutzung des Systems gestört haben, und entsprechende Lösungsansätze und Verbesserungsvorschläge formuliert. In einem **summativen Test** steht die abschließende Bewertung des Systems im Vordergrund. Hier ist beispielsweise die Ausführungsgeschwindigkeit bestimmter Funktionen oder die Anzahl der gefundenen Fehler in der Benutzungsschnittstelle relevant.

Eine weitere Möglichkeit, Testmethoden einzuordnen, ist die Art der erhobenen Daten. **Quantitative Ergebnisse** sind messbare Größen. Sie werden beispielsweise durch Zeitmessungen bestimmt oder mit Hilfe von Fragebögen ermittelt. Sie können statistisch ausgewertet werden und lassen Aussagen über Effektgrößen und Signifikanz zu. **Qualitative Ergebnisse** sind nicht messbare Werte, wie beispielsweise gefundene Fehler in einem evaluierten Interface. Derartige Ergebnisse können nachträglich aber auch zusammengefasst und quantitativ betrachtet werden, z. B. ist die Häufigkeit jedes einzelnen Fehlers ein quantitativer Wert.

Im Gegensatz zu den vorherigen Klassifikationen lassen sich die meisten Testmethoden sauber nach analytischer oder empirischer Testmethode klassifizieren: **Analytische Testmethoden** beruhen auf einem theoretischen Modell, beispielsweise dem kognitiven Model beim Cognitive Walkthrough, das den „echten Benutzer“ ersetzen soll. Anhand dieses Modells wird das Interface Stück für Stück analysiert. **Empirische Testmethoden** untersuchen die Auswirkungen des Interfaces. Zu den empirischen Methoden gehören unter anderem Tests mit Benutzern und Benutzerbefragungen.

2.3 Testmethoden

In diesem Abschnitt werden die geläufigsten Testmethoden für Gebrauchstauglichkeit vorgestellt. Abschnitt 2.3.1 behandelt Tests mit Benutzern und die Thinking-aloud-Methode. Im Abschnitt 2.3.2 wird die Befragung von Benutzern beschrieben. Der Cognitive Walkthrough wird im Abschnitt 2.3.3 besprochen, gefolgt von der heuristischen Evaluation im Abschnitt 2.3.4. Ein Vergleich dieser Methoden wird in dem Abschnitt 2.4 behandelt.

2.3.1 Tests mit Benutzern und die Thinking-aloud-Methode

Die geläufigste Methode bei Tests mit Benutzern ist die Thinking-aloud-Methode (Thinking-aloud vgl. Lewis 1982). Dabei werden die Auswirkungen der Benutzungsschnittstelle mit potentiellen späteren Benutzern getestet. Die Teilnehmer erledigen typische Aufgaben mit dem System und werden von dem Testleiter angehalten, ständig ihre Gedanken, Vorhaben und Probleme laut zu äußern. Jedes Mal, wenn der Teilnehmer auf Probleme trifft, oder nicht weiß, welcher Schritt ihn näher an die Lösung der Aufgabe führt, handelt es sich möglicherweise um einen Fehler in der Benutzungsschnittstelle.

In einigen Situationen ist es nicht angebracht, den Teilnehmer seine Gedanken verbalisieren zu lassen. Ein Beispiel dafür sind Situationen, in denen der Teilnehmer nicht abgelenkt werden darf, weil seine Fehler fatale Folgen hätten. In diesen Fällen kann der Teilnehmer später befragt oder mit einer Videoaufnahme seiner Aktionen konfrontiert werden. Auch bei der Messung von Ausführungszeiten würde das Verbalisieren der Gedanken stören, da der Teilnehmer dadurch länger brauchen würde. Neben der reinen

Thinking-aloud-Methode gibt es noch einige Variationen: Beispielsweise versuchen beim *Co-Discovery* zwei Teilnehmer gemeinsam die gestellten Aufgaben zu erledigen (vgl. Hom 1998). Dabei erzählen sie sich gegenseitig, was sie tun und weshalb sie es tun.

Die Benutzertest-Methode gehört zu den empirischen Testmethoden. Sie kann je nach Bedarf formativ, zum Finden von Gebrauchstauglichkeitsproblemen und passenden Lösungen, oder summativ, zum abschließenden Bewerten einer Benutzungsschnittstelle, eingesetzt werden. Wegen des hohen Zeit- und Arbeitsaufwands, der besonders durch die benötigte Nachbereitung entsteht, werden Tests mit Benutzern meist zur Erhebung qualitativer Daten eingesetzt, da quantitative Aussagen bei wenigen Benutzern keine signifikanten Aussagen zulassen.

Für den Test werden typische Aufgaben ausgewählt, die später mit dem System erledigt werden sollen. Diese Aufgaben sollen möglichst viele und möglichst alle wichtigen Dialoge des Interfaces abdecken, um das System ausgiebig zu testen. Falls durch eine fehlerhafte oder unvollständige Aufgabenanalyse nur unwichtige Aufgaben getestet werden, werden im schlimmsten Fall einige wichtige Funktionen der Benutzerschnittstelle gar nicht untersucht und die enthaltenen Fehler nicht erkannt. Des Weiteren ist zu beachten, das bestimmte Aufgaben von einigen Benutzergruppen möglicherweise auf anderem Wege bearbeitet werden. Dies ist beispielsweise bei Anfängern und fortgeschrittenen Benutzern der Fall: Während der Anfänger die Aufgabe Schritt für Schritt erledigt, macht der fortgeschrittene Benutzer gelegentlich von Abkürzungen Gebrauch. Eine genaue Aufgabenanalyse, sowie das Bestimmen der Zielgruppen ist also unerlässlich (siehe Abschnitt 2.1.5).

Über die Anzahl der nötigen Teilnehmer für einen Benutzertest finden sich in der Literatur unterschiedliche Angaben. Jakob Nielsen empfiehlt, einen Test mit maximal 5 Teilnehmern durchzuführen (vgl. Nielsen, 1993, Seite 169). Diese Methode nennt er „Discount Usability“. Er gibt an, dass mit 5 Teilnehmern bereits ein Großteil der Benutzungsprobleme gefunden werden, weitere Teilnehmer aber kaum neue Erkenntnisse bringen. Er empfiehlt aber auch, nachdem die gefundenen Fehler aus dem Interface entfernt wurden, mindestens einen weiteren Test mit 5 Teilnehmern durchzuführen. Da-

durch lässt sich überprüfen, ob die Fehler wirklich beseitigt wurden und ob durch die Änderungen neue Fehler hinzugekommen sind. Nielsen schrieb (Nielsen 2000):

> „A second test will discover whether the fixes worked or whether they didn't. Also, in introducing a new design, there is always the risk of introducing a new usability problem, even if the old one did get fixed."

Die Zahl der Teilnehmer sollte erhöht werden, wenn die Gruppe der potentiellen Benutzer heterogen ist, da unterschiedliche Benutzer auf unterschiedliche Weise mit der Benutzungsschnittstelle arbeiten können und somit andere Fehler finden. Die Teilnehmer für den Test sollten aus den unterschiedlichen Benutzergruppen stammen.

Jared Spool und Will Schroeder sind der Meinung, dass 5 Teilnehmer keinesfalls ausreichen (vgl. Spool und Schoeder 2001). Sie stellten fest, dass bei zwei Tests mit je 18 Teilnehmern, in denen CDs, Videos und DVDs gekauft werden sollten, auch die letzten Teilnehmer noch relevante Fehler fanden.

Die Frage nach der nötigen Anzahl der Teilnehmer war auch Gegenstand eines Panels der CHI-Konferenz 2003 mit Carol Barnum, Nigel Bevan, Gilbert Cockton, Jakob Nielsen, Jared Spool und Dennis Wixon (vgl. Barnum et al. 2003). Die meisten Teilnehmer des Panels waren sich einig, dass es keine pauschale Antwort auf die Frage nach der Anzahl der Test-Teilnehmer gäbe. Die Anzahl sei abhängig von vielen Faktoren. Gilbert Cockton schrieb dazu (Barnum et al. 2003, Seite 698f):

> „What we do understand are the variables that influence problem yield, for example, test user diversity, test protocol design, task performance diversity, application complexity, design quality (problems are easier to find for poor designs), problem reporting procedures and the usability goals set for a product (no goals, no problems!)"

Dennis Wixon und andere bemerkten auch (Barnum et al. 2003, Seite 699): „For whatever value of ‚N'[1] they choose, ‚N' users will always be better than zero."

Bei der Durchführung eines Tests mit Benutzern ist auf folgendes zu achten: Vor dem Test sollte der Testleiter den Teilnehmer darauf hinweisen, dass nicht der Teilnehmer selbst, sondern die Benutzungsschnittstelle getestet wird (vgl. Shneiderman 1998, Seite 129; Nielsen 1993, Seite 182). Das Gefühl, selbst getestet zu werden, kann Stress beim Teilnehmer erzeugen, der zu zusätzlichen Bedienungsfehlern und verändertem Verhalten führen kann. Da jedes Problem des Teilnehmers auf einen Fehler in der Benutzungsschnittstelle hinweisen kann, erschweren durch Stress erzeugte Unsicherheiten das Deuten der Fehler im Interface.

Im nächsten Schritt erklärt der Testleiter dem Teilnehmer die Aufgabe, wobei zu beachten ist, dass er den Teilnehmer nicht indirekt beeinflusst. Beeinflussungen durch den Testleiter können bei statistischen Auswertungen der Benutzeraktionen zu verfälschten Ergebnissen führen. Während des Tests sollte der Testleiter dafür sorgen, dass der Teilnehmer ständig erzählt, was er gerade macht und warum er es auf diese Weise macht. Auch hier sollte der Teilnehmer nicht beeinflusst werden. Fragen des Teilnehmers zur Benutzung sollten während des Tests möglichst nicht beantwortet werden, es sei denn, zur Benutzung der Schnittstelle wird ein bestimmtes Fachwissen vorausgesetzt, über das der Teilnehmer nicht verfügt.

Während eines derartigen Tests lassen sich Situationen erkennen, in denen der Teilnehmer durch das Interface verwirrt ist, er nicht mehr weiter weiß oder sogar den falschen Menüpunkt anwählt. Diese und viele andere Situationen weisen auf Gebrauchstauglichkeitsprobleme hin. Die Ursache dafür können ungenau beschriftete Bedienelemente, schlechtes Systemfeedback und ähnliches sein. Der Testleiter muss die Situationen deuten, die Probleme erkennen und notieren.

Es ist schwierig, dem Teilnehmer zuzuhören, gleichzeitig dessen Aktionen mit dem System zu verfolgen, seine Probleme zu erkennen und daraus Fehler des Interfaces ab-

[1] N steht für die Anzahl der benötigten Benutzer

zuleiten. Deshalb bietet es sich an, derartige Tests auf Video oder Tonband aufzuzeichnen. Dazu wird die Kamera auf den Bildschirm des Teilnehmers gerichtet und seine Äußerungen werden per Mikrofon aufgenommen. Je nach Variante der Testmethode kann zusätzlich auch der Teilnehmer gefilmt werden oder nur die Tonspur aufgezeichnet werden. Zur Auswertung lassen sich wichtige Teile der Aufzeichnung später beliebig oft wiederholen. Da der Testleiter auf viele Dinge gleichzeitig achten muss, beträgt die Auswertungsdauer eines Tests im Allgemeinen ein Vielfaches der eigentlichen Testdauer. Derartige Benutzertests werden entweder in speziell für diesen Zweck eingerichteten Laboratorien oder mit einer mobilen Ausrüstung am gewohnten Arbeitsplatz des Benutzers durchgeführt.

Benutzertests sind ideal, um Fehler im Interface von Software und Websites zu finden. Gerade die praxisrelevanten Fehler lassen sich durch Benutzertests entdecken. Beispiele für derartige Benutzbarkeitsprobleme sind fehlender Systemfeedback, nachdem der Benutzer nicht weiß ob seine Eingabe akzeptiert wurde und mehrdeutige oder schlecht gewählte Beschriftungen von Bedienelementen, die den Benutzer in die Irre führen. Probleme, die den Benutzer nur unbewusst stören, wie inkonsistente Schriftarten oder Farbwahlen, werden mit dieser Methode nicht erkannt.

Eine Besonderheit dieser Testmethode ist die automatische Einbeziehung der menschlichen Merkmale, wie die Aufmerksamkeitsspanne, die Grenzen des Kurzzeitgedächtnisses und einige andere (siehe Abschnitt 2.1.2). Da das Interface mit einem Menschen getestet wird, wird beispielsweise die Überforderung des Kurzzeitgedächtnisses als Problem des Teilnehmers auftreten. Der Testleiter wird erkennen, dass sich der Teilnehmer nicht alle Informationen merken konnte, die er für den nächsten Schritt bräuchte und wird nach einer Lösung suchen. In anderen Methoden, wie den Untersuchungen durch Experten, müssen die Fähigkeiten und Grenzen des Menschen explizit berücksichtigt werden, z. B. durch entsprechende Regeln in der heuristischen Evaluation oder spezielle Fragen bei der Simulation der kognitiven Prozesse in dem Cognitive Walkthrough.

2.3.2 Befragung von Benutzern

Befragungen von Benutzern zu einem bestimmten Interface bieten sich an, um den subjektiven Eindruck von Benutzern zu ermitteln. Befragungen können mündlich oder per Fragebogen durchgeführt werden. Dieses Verfahren wird auch gern mit dem Test mit Benutzern kombiniert, um die Probleme der Benutzungsschnittstelle und mögliche Lösungsvorschläge aus Sicht des Benutzers zu erfahren.

Die Befragung von Benutzern gehört ebenfalls zu den empirischen Testmethoden. In Verbindung mit Benutzertests werden die Ergebnisse der Befragung im Allgemeinen qualitativ gewertet. Das liegt an der meist geringeren Anzahl der befragten Personen wegen des höheren Aufwandes des Benutzertests. Dagegen lassen sich Befragungen mit Papier- oder Online-Fragebögen bei geringen Kosten mit vielen Teilnehmern durchführen. Das Graphic, Visualization & Usability Center erreichte beispielsweise mit seiner dritten WWW-Benutzerumfrage über 13.000 Teilnehmer (vgl. Graphic, Visualization & Usability Center 1995).

Wird dieses Verfahren benutzt, um die Probleme des Interfaces und eventuelle Lösungsansätze zu erhalten, muss beachtet werden, ob die Teilnehmer die aufgetretenen Probleme als Interface-Fehler werten. Oftmals meinen die Teilnehmer, es läge an ihnen, dass sie mit dem Interface nicht zurechtkommen. Werner Schweibenz und Frank Thissen schrieben dazu (Schweibenz und Thissen 2003):

> „Unsere Untersuchungen zur Web Usability haben gezeigt, dass der größte Teil der Versuchspersonen von Nutzertests die Ursachen von Problemen der Bedienbarkeit in der mangelnden Übung im Umgang mit dem Produkt, in fehlenden Grundkenntnissen oder der eigenen Unaufmerksamkeit sehen und extrem selten die Ursache in der Gestaltung des Produkts sehen. Aus diesem Grund ist die Befragung von Nutzern also auch nicht so erhellend wie erhofft.“

Dies zeigt, wie wichtig es ist, dem Teilnehmer die Angst zu nehmen. Er sollte darauf hingewiesen werden, dass es viele Benutzer mit einem ähnlichen Wissensstand gibt, und dass der Sinn dieser Befragung darin liegt, die vorgefundenen Schwierigkeiten zu beseitigen.

Jakob Nielsen rät allerdings davon ab, Benutzer nach Lösungsvorschlägen zu fragen. Er schrieb in seiner Alertbox: „To design an easy-to-use interface, pay attention to what users do, not what they say. Self-reported claims are unreliable, as are user speculations about future behavior." (Nielsen 2001).

Neben dem Aspekt der Benutzerbeteiligung können Befragungen auch genutzt werden, um Hintergrundinformationen zum Teilnehmer, seine Computererfahrung und weitere Fakten zu erfragen. Auch der subjektive Eindruck von dem Interface und die Gefühle nach der Benutzung lassen sich durch Befragungen ermitteln (vgl. Shneiderman 1998, Seite 133).

2.3.3 Cognitive Walkthrough

Echte Benutzer mit bestimmtem Vorwissen sind oft schwer zu beschaffen oder die Verfahren sind zu kostspielig. Der Cognitive Walkthrough (vgl. Rieman, Franzke & Redmiles 1995) ist eine Evaluationsmethode, die ohne „echte" Benutzer auskommt. Stattdessen wird das Interface von Systementwicklern untersucht, die die kognitiven Prozesse des Benutzers nachahmen. Unter Berücksichtigung des Wissens und der Ziele des Benutzers wird Schritt für Schritt überprüft, ob der Benutzer in der Lage wäre, durch Verwendung des Interfaces seinem Ziel näher zu kommen.

Der Cognitive Walkthrough gehört zu den analytischen Testmethoden. Diese Methode lässt sich, je nach Bedarf, formativ und summativ einsetzten.

Mit dem Cognitive Walkthrough lässt sich eine Benutzerschnittstelle besonders daraufhin testen, ob sie sich für „Lernen durch Ausprobieren" (explorative learning) eignet, der Benutzer also neue Funktionen des Interfaces erschließen kann, ohne vorher geschult zu werden oder die Dokumentation lesen zu müssen (vgl. Wharton, Rieman, Lewis und Polson 1994). Die meisten Benutzer, sowohl Anfänger als auch Fortgeschrittene, greifen nur ungern zur Dokumentation oder besuchen Schulungen. Statt dessen versuchen sie, die gewünschte Funktion durch Ausprobieren zu finden, sobald diese für ihre aktuelle Arbeit nützlich ist (vgl. Carroll und Rosson 1987).

Der Cognitive Walkthrough besteht aus zwei Phasen: der Vorbereitungsphase und der Analysephase. In der Vorbereitungsphase wird die Benutzergruppe des Systems bestimmt und die typischen Aufgaben und Ziele des Benutzers ermittelt (siehe Abschnitt 2.1.5). Die ermittelten Aufgaben werden in elementare Aktionen zerlegt, z. B. das Anwählen eines Bedienelements oder die Eingabe eines Textes in ein Textfeld.

In der zweiten Phase, der Analysephase, wird die Analyse des Interfaces durchgeführt. Eine Gruppe von Entwicklern und Experten anderer Bereiche arbeiten Aktion für Aktion durch. Dabei versucht die Gruppe, sich in die Lage der zukünftigen Benutzer zu versetzen.

Der Analysephase des Cognitive Walkthrough liegt die Theorie des erforschenden Lernens zugrunde (vgl. Wharton, Bradford, Jeffries und Franzke 1992). Danach wird die Mensch-Computer-Interaktion in vier Schritte unterteilt, die jeder Benutzer bei jeder Aktion abarbeitet (vgl. Rieman, Franzke & Redmiles 1995). Im ersten Schritt setzt sich der Benutzer ein Ziel, das er mit dem System erreichen will, z. B. Ausdrucken des aktuellen Dokuments. Im zweiten Schritt sucht er im Interface nach verfügbaren Aktionen. Im dritten Schritt wählt er die Aktion aus, die ihn dem Ziel näher bringt. Im vierten Schritt führt er die gewählte Aktion aus und analysiert das Feedback des Systems, um Hinweise auf die Annährung an sein Ziel zu erhalten.

Beim Nachahmen des menschlichen kognitiven Prozesses müssen die Evaluatoren auch die Merkmale des Menschen, wie die spezielle Wahrnehmung, das unzuverlässige Gedächtnis und andere (siehe Abschnitt 2.1.2) berücksichtigen.

Die Evaluatoren diskutieren die gefundenen Probleme der Benutzungsschnittstelle und suchen nach Lösungen und Verbesserungsvorschlägen, um diese bei Bedarf in den nächsten Prototyp einfließen zu lassen.

Der hier beschriebene Cognitive Walkthrough ist eine vereinfachte Version. Die Methode wird effektiver, wenn mindestens einer der Evaluatoren Hintergrundwissen über die kognitiven Prozesse hat und somit die Faktoren kennt, die den Benutzer bei der Entscheidungsfindung beeinflussen (vgl. Wharton und Lewis 1994).

Für das Testen von Webseiten wurde der Cognitive Walkthrough von Blackmon, Polson, Kitajima und Lewis modifiziert (vgl. Blackmon et al. 2002), um ihn an das Navigieren in Websites anzupassen. Es gibt drei große Unterschiede, die den „Cognitive Walkthrough for the Web“ vom normalen Cognitive Walkthrough unterscheiden: Erstens werden für den Cognitive Walkthrough for the Web umfassendere Benutzerziele definiert, die mehr Informationen über das konzeptionelle Modell der Aufgabe und die Motivation des Benutzers enthalten. Die Beschreibung der Benutzerziele sollten jeweils etwa 100-200 Worte lang sein. Zweitens sollten Aktionen, wie das Anklicken eines Links oder anderer Elemente, als zwei separate Schritte angesehen werden. Im ersten Teilschritt wird die Webseite in Regionen unterteilt und die richtige Region bestimmt. Im zweiten Teilschritt wird ein passendes Element der Region ausgewählt und damit agiert. Drittens arbeiten die Evaluatoren nicht Aufgabe für Aufgabe ab und springen dabei durch die einzelnen Seiten, sondern untersuchen eine Seite zur Zeit, auf der die vorher festgelegten Benutzerziele nacheinander abgearbeitet werden.

2.3.4 Heuristische Evaluation

Eine weitere Möglichkeit, ohne „echte“ Benutzer auszukommen, ist die heuristische Evaluation (vgl. Nielsen 1992). Hierbei untersuchen Experten jedes Element der Benutzungsschnittstelle anhand einer Liste von Grundsätzen.

Die heuristische Evaluation ist eine der bekanntesten analytischen Testmethoden. Sie kann formativ eingesetzt werden, da sich aus Verstößen gegen die Grundsätze gleich Verbesserungsvorschläge ableiten lassen.

Die heuristische Evaluation wird von erfahrenen Experten durchgeführt, die jedes Element einer Benutzungsoberfläche anhand einer Liste von Heuristiken überprüfen. Diese Heuristiken sind generelle Regeln, die die allgemeinen Eigenschaften einer gebrauchstauglichen Oberfläche aufführen. Beispiele für derartige Heuristiken sind: Entwerfe einfache und natürliche Dialoge, verwende die Sprache des Benutzers, minimiere die kognitive Last des Benutzers, achte auf Konsistenz, gib dem Benutzer Feedback und andere. Da dieses Verfahren ohne „echte“ Benutzer auskommt, müssen die Heuristiken derart geschaffen sein, dass die menschlichen Merkmale (siehe Abschnitt 2.1.2) berücksichtigt werden. Nur dann ist das Testen mit Heuristiken auch sinnvoll. In vielen

Büchern werden Heuristiken angegeben, die dieses Kriterium erfüllen (vgl. Nielsen 1993, Kapitel 5; ISO 9241-10 1996). Der Testleiter kann weitere Heuristiken hinzufügen, wenn die Benutzungsschnittstelle es verlangt. Auch für das Gestalten von Webseiten gibt es eine Reihe von Leitlinien (Borges, Morales und Rodriguez 1996; Lynch und Horton 2002; Weinreich 1997).

Studien haben gezeigt, dass ein Evaluator allein im Schnitt nur ca. 35% der Benutzbarkeitsprobleme findet (vgl. Nielsen 1992). Der Erfolg eines Evaluators hängt natürlich von der Erfahrung im Testen von Benutzungsoberflächen und dem Wissen über das zu testende Interface ab. Da unterschiedliche Evaluatoren unterschiedliche Fehler finden, bietet es sich an, mehrere Evaluatoren einzusetzen. (vgl. Nielsen 1994, Seite 34ff).

Vor der Durchführung einer heuristischen Evaluation sollte ein Szenario erstellt werden, anhand dessen die Evaluatoren die einzelnen Dialoge des Systems durchlaufen. Für das Szenario werden möglichst typische Aufgaben gewählt, die später mit dem System erledigt werden sollen. Auf diese Weise werden die wichtigsten Teile des Interfaces getestet (vgl. Abschnitt 2.1.5). In einigen Varianten der heuristischen Evaluation wird bewusst auf vorgeschriebene Szenarien verzichtet (vgl. Karat 1994), damit sich jeder Evaluator bei der Arbeit mit dem Interface seinen eigenen Weg sucht. Auf diese Weise sollen auch die weniger wichtigen Dialoge von einigen Evaluatoren untersucht werden, die bei einem gemeinsamen Szenario sonst herausgefallen wären.

Erst wenn alle Evaluatoren ihre Evaluation abgeschlossen haben, werden alle gefundenen Probleme in einer Sitzung durchgesprochen. An dieser Sitzung sollten neben den Evaluatoren auch die Entwickler teilnehmen, um hier mögliche Verbesserungen zu diskutieren.

Durch die heuristische Evaluation lassen sich sehr viele Fehler des Interfaces aufdecken. Voraussetzung dafür sind aber sehr gute Heuristiken, die an die Fragestellung des Evaluators angepasst sind. Es lassen sich auch Benutzbarkeitsprobleme entdecken, die durch Tests mit Benutzern gar nicht oder nur mit sehr großem Aufwand gefunden werden können (vgl. Nielsen 1994, Seite 46). Zum Beispiel lassen sich Inkonsistenzen, wie unterschiedliche Schriftarten oder Farben für ähnliche Zwecke, durch Tests mit Benutzern kaum ermitteln.

2.4 Vergleich der Testmethoden

Jede Testmethode hat ihre Stärken und Schwächen, die in den folgenden Abschnitten einander gegenübergestellt werden. Besonders beim Finden von Problemen aus bestimmten Bereichen unterscheiden sich die einzelnen Methoden. Ivory und Hearst schrieben dazu: „Each technique has ist own requirements, and generally different techniques uncover different usability problems“ (Ivory und Hearst 2001, Seite 471). Clare-Marie Karat hat in einer Fallstudie festgestellt, dass sich die identifizierten Probleme von je zwei der behandelten Evaluationsmethoden nur um 10-15% überlappten (vgl. Karat 1994, Seite 207).

2.4.1 Tests mit Benutzern und Benutzerbefragungen

Mit Hilfe von Benutzertests lassen sich die Probleme finden, die den späteren Benutzer bei der Arbeit mit dem System behindern. Besonders die schweren Probleme lassen sich durch diese Methode sicher erkennen. Die anderen Methoden finden hauptsächlich weniger schwere Probleme, die den Benutzer nicht – oder nur unbewusst – stören.

Die Studie von Pamela Savage ergab, dass sich aus den durch Benutzerbeteiligung erhaltenen Problemen leichter Lösungsvorschläge ableiten lassen (vgl. Savage 1996). Sie schreibt dazu: „Notice that expert feedback tended to signal areas needing further testing, while end-user feedback tended to result in design changes to the user interface.“ (Savage 1996, Seite 308).

Benutzertests sind aber auch sehr viel flexibler als die anderen behandelten Testmethoden. So lassen sich mit Benutzertests neben den gefundenen Interface-Problemen auch Aussagen wie die folgende ermitteln: „Die Benutzer werden die vorgegebene Aufgabe in weniger als 10 Minuten erledigen können“ (vgl. Karat 1994, Seite 205).

2.4.2 Cognitive Walkthrough

Eine besondere Stärke des Cognitive Walkthrough liegt im Testen von Benutzungsschnittstellen auf leichte Erlernbarkeit. Damit lässt sich beispielsweise feststellen, ob ein Benutzer das System ohne vorheriges Training benutzen kann. Ein weiterer Vorteil

dieser Methode liegt darin, dass der Cognitive Walkthrough auch von Systementwicklern durchgeführt werden kann. Es sind also keine teuren Experten für Gebrauchstauglichkeit oder Benutzer nötig.

Beim Aufspüren von Benutzbarkeitsproblemen schneidet diese Methode im Vergleich mit Benutzertests und heuristischer Evaluation allerdings nicht gut ab: Karat, Campbell und Fiegel haben in ihrer Studie den Cognitive Walkthrough mit dem Benutzertest verglichen, indem sie zwei Systeme jeweils mit den beiden Methoden getestet haben. Dabei haben sie festgestellt, dass durch den Benutzertest viermal mehr Probleme gefunden wurden, als durch den Cognitive Walkthrough (vgl. Karat et al. 1992).

Jeffries, Miller, Wharton und Uyeda haben in ihrer Studie festgestellt, dass durch die heuristische Evaluation dreimal mehr Probleme gefunden wurden als durch den Cognitive Walkthrough (vgl. Jeffries et al. 1991).

2.4.3 Heuristische Evaluation

Die heuristische Evaluation ist eine der erfolgreichsten analytischen Testmethoden. Sie findet sehr viele Probleme, von denen etwa ein Drittel zu den schweren Problemen zählen (Jeffries et al. 1991). Die Methode ist natürlich nur so gut, wie die Gruppe der Experten für Gebrauchstauglichkeit, die das System untersuchen. Derartige Experten sind allerdings rar und teuer.

In der Studie von Jeffries und Kollegen wurden mit Hilfe der heuristischen Evaluation fast dreimal mehr Probleme gefunden als durch den Benutzertest (vgl. Jeffries et al. 1991). Allerdings wird dieses Ergebnis z. B. durch Clare-Marie Karat angezweifelt, da die Experten 2 Wochen für die Evaluation hatten, jedem der sechs Teilnehmer des Benutzertests aber nur einige Stunden blieben (vgl. Karat 1994).

Im Vergleich zum Benutzertest zeichnet sich die heuristische Evaluation durch einen geringeren Zeitaufwand aus. Des Weiteren findet sie Probleme, die durch den Benutzertest nicht gefunden werden, beispielsweise inkonsistent gewählte Schriftarten.

2.4.4 Fazit

Da jede der besprochenen Methoden Stärken beim Lokalisieren von Problemen aus unterschiedlichen Bereichen hat, bietet es sich an, mehrere Methoden miteinander zu kombinieren. Auf diese Weise lässt sich die größtmögliche Fehlerzahl und Vielfalt an Problemen finden.

Viele Testmethoden anzuwenden kostet allerdings auch viel Zeit und Geld. Diese sind aber, besonders bei kommerziellen Produkten, nur beschränkt vorhanden. Daher muss die Anzahl der verwendeten Testmethoden und der Umfang der einzelnen Tests eingeschränkt werden. Aber auch wenige Tests sind besser als keine (vgl. Barnum et al. 2003).

Insgesamt hat sich allerdings gezeigt, dass empirische Methoden signifikante Vorteile aufweisen, da sie genau die Probleme entdecken, die den Benutzer bei der praktischen Anwendung der Benutzungsschnittstelle behindern. Nielsen schrieb dazu (Nielsen 1993, Seite 165):

> „User testing with real users is the most fundamental usability method and is in some sense irreplaceable, since it provides direct information about how people use computers and what their exact problems are with the concrete interface being tested.“.

Wegen der langen Nachbearbeitungszeit jedes Testlaufs ist der Benutzertest eine aufwändige Methode. Diese Nachbereitungs- und Analysezeit zu verringern ist ein Ziel dieser Arbeit.

3 Entwurf eines Werkzeugs

Tests mit Benutzern sind in der Lage, jene Fehler in der Benutzungsschnittstelle zu finden, die den Benutzer bei der praktischen Anwendung des Systems besonders stören. Benutzertests sind allerdings auch sehr aufwendig, da die benötigte Zeit für die Auswertung und Analyse ein Vielfaches der aufgewendeten Zeit für die eigentlichen Testläufe in Anspruch nehmen kann. Viele Teilaufgaben eines Benutzertests sind Routinearbeiten, die auch von einem Computer übernommen werden können. Der Testleiter ließe sich auf diese Weise entlasten.

Nach einer Begriffsbestimmung im Abschnitt 3.1 werden im Abschnitt 3.2 thematisch verwandte Arbeiten aufgeführt. Dabei wird besonders darauf geachtet, mit welchen Techniken die Aufgaben des Testleiters in diesen Arbeiten unterstützt wurden. Abschnitt 3.3 untersucht die einzelnen Aufgaben eines Testleiters darauf hin, ob sich diese auch vom Computer ausführen lassen und ob dies sinnvoll ist. Im Abschnitt 3.4 werden Besonderheiten besprochen, die erst durch die Automatisierung des Benutzertests entstanden sind. Abschnitt 3.5 fasst die Ergebnisse zusammen.

3.1 Begriffsbestimmung

Es gibt drei Schlüsselrollen bei Tests mit Benutzern: Die Rolle des *Testentwicklers*, des *Testleiters* und des *Teilnehmers*. Der Testentwickler stellt die Aufgaben zusammen und erstellt die Testablaufdatei, die das „UserTestTool“ steuert, das hier entworfen wird. Der Testleiter führt den Test durch. Testleiter und Testentwickler müssen nicht die selbe Person sein. Der Testteilnehmer arbeitet die vom Testentwickler erstellten Aufgaben ab und zeigt durch auftretende Probleme die Schwachstellen der Benutzungsschnittstelle auf.

Aktionen – oder Benutzeraktionen – sind alle Maus- oder Tastatureingaben, die der Teilnehmer während des Tests vornimmt. Hauptsächlich handelt es sich dabei um angeklickte Schaltflächen oder Links und ausgefüllte Texteingabefelder. Diese Aktionen und beispielsweise die besuchten URIs sind nötig, um später die einzelnen Schritte des Teilnehmers bei der Benutzung der Website nachvollziehen zu können.

3.2 Thematisch verwandte Arbeiten

Das UserTestTool, das den Testleiter entlasten soll, soll mehrere Aufgaben übernehmen. Eine Aufgabe ist das Abfangen der Benutzeraktionen, also die Art, in der der Benutzer die Webseiten angewählt hat und die benutzten Funktionen des Browsers, wie beispielsweise die History. Des Weiteren sollten die besuchten URIs gespeichert werden, um später den Weg des Benutzers durch die Website nachvollziehen zu können.

Eine weitere Möglichkeit des UserTestTools soll das Stellen der Aufgaben sein. Auf diese Weise werden dem Teilnehmer die Aufgaben konsistent und ohne Beeinflussung durch den Testleiter präsentiert.

In den folgenden Abschnitten werden Arbeiten vorgestellt, in denen ähnliche Funktionen entwickelt oder Benutzeraktionen auf bestimmte Weise abgefangen wurden.

3.2.1 Benutzeraktionen aus Protokolldateien

Die Idee, die Eingaben und Mausklicks der Benutzer abzufangen, während diese im Internet surfen, ist nicht neu. In vielen Studien zur Benutzung des World Wide Webs wurden die Interaktionen der Teilnehmer mit dem WWW auf unterschiedlichste Weise abgefangen.

Ein Beispiel dafür ist die Studie „A Model of Web Site Browsing Behavior - Estimated on Clickstream Data“ von Randolph E. Bucklin und Catarina Sismeiro (Bucklin und Sismeiro 2003, Seite 250f). Sie benutzen die Protokolldateien eines Webservers, um Informationen über Art und Zeitpunkt jedes einzelnen Seitenaufrufs zu erhalten. Dadurch lassen sich die IP-Adresse des Benutzers, Datum und Uhrzeit der Anfrage, der Name der angeforderten Datei und das Ergebnis des Aufrufes, z. B. erfolgreich oder fehlerhaft, ermitteln. Der Aufwand ist gering, da die Aufrufe jedes Benutzers automatisch protokolliert werden. Außerdem müssen keine Benutzer hinzugezogen werden, deren Auswahl aufwendig und kostspielig sein kann. Da es sich um anonyme Daten handelt, sind auch keine Einwilligungen der Benutzer nötig. Nachteil dieser Methode ist, dass die Daten unvollständig sind. Etgen und Cantor schreiben dazu (Etgen und Cantor 1999):

„First, servers cannot collect data on some potentially crucial user interactions that occur on the client-side only (e. g., within-page anchor links, form element interaction, java applets, etc.). A second related point is that the validity of the data is highly suspect because of caching by proxy servers and browsers and dynamic IP addressing."

Auf ähnliche Weise haben Cockburn und McKenzie Informationen zu den Benutzeraktionen gesammelt (vgl. Cockburn und McKenzie 2000). Allerdings benutzten sie statt der Protokolldateien des Webservers die History- und die Bookmark-Dateien des Browsers der Benutzer. Die History-Datei enthält die besuchten URLs mit Seitentitel und Zeitpunkt des ersten und des letzten Besuches. Die Bookmark-Datei enthält die URLs der gesammelten Lesezeichen (Favoriten) mit dem Datum des Anlegens und des letzten Besuchs. Wie bei der Studie von Bucklin und Sismeiro lassen sich allerdings auch hier keine Rückschlüsse auf konkrete Benutzeraktionen ziehen, die zum Aufruf der nächsten Webseite benutzt wurde.

3.2.2 Benutzeraktionen aus modifizierten Browsern

Eine andere Möglichkeit zum Sammeln der Benutzeraktionen stellt die Studie „How people revisit web pages: empirical findings and implications for the design of history systems" von Linda Tauscher und Saul Greenberg vor (vgl. Tauscher und Greenberg 1997, Seite 106). Sie modifizierten damals den Browser Mosaic, so dass alle relevanten Maus- und Tastaturaktionen sowie die besuchten URLs gespeichert wurden. Damit konnten ausnahmslos alle Aktionen des Benutzers abgefangen und auf beliebige Weise gespeichert werden. Der Nachteil dieser Methode ist, dass jeder Benutzer, der an der Studie teilnehmen möchte, genau diesen einen Browser benutzen muss. Ein aktueller Browser, der sich auf diese Weise verändern ließe, ist Mozilla (vgl. Mozilla Organization 2003). Dieser Browser ähnelt in Aussehen und Bedienung sehr dem Netscape Communicator und der Quelltext ist frei verfügbar. Allerdings werden diese Browser nur von ca. 8% der Webnutzer angewendet (vgl. WebReference.com, 17. November 2003). Der von 75% der Surfer genutzte Browser, Microsofts Internet Explorer, lässt sich nicht auf diese Weise modifizieren, da sein Quelltext nicht frei verfügbar ist.

Der Internet Explorer lässt sich zwar nicht verändern, bietet aber die Möglichkeit der Erweiterung. Mit Hilfe sogenannter „Browser Helper Objects“ (vgl. Roberts 1999, Kapitel 12) lassen sich Aktionen wie das Aufrufen und Laden von Webseiten und Änderungen von Titel- oder Statusleiste abfangen. Die Benutzung der internen Browserfunktionen wie „Vor“, „Zurück“ oder die Favoritenverwaltung lässt sich allerdings nicht abfangen.

3.2.3 Benutzeraktionen per JavaScript

Eine Variante, die vom Browser unabhängig ist, haben Etgen und Cantor vorgestellt (vgl. Etgen und Cantor 1999). Sie fingen die Benutzeraktionen über die „Document Object Model Events“ (vgl. Hegaret und Pixley 2003) ab. Damit erhielten sie sämtliche Aktionen des Benutzers, die er in dem Anzeigebereich des Browsers ausführte. Vorher musste allerdings jede Seite um einen Aufruf einer JavaScript-Datei erweitert werden. Dies lässt sich z. B. durch CGI-Skripte erreichen, sofern man Zugriff auf den Webserver hat und dieser serverseitige Programme unterstützt. Alternativ müssen alle benötigten Seiten lokal gespeichert und von Hand um den Aufruf der JavaScript-Datei ergänzt werden.

Einen völlig anderen Ansatz stellen „Web Intermediaries“ (vgl. Maglio und Barrett 2000), auch „Web Browser Intelligence“ (vgl. Barrett, Maglio und Kellem 1997) genannt, dar. Web Intermediaries (kurz WBI, englisch gesprochen Web-ee) sind Programme, die zwischen dem Webserver und dem Browser des Benutzers angeordnet sind. Sie können die aufgerufenen Seiten überwachen, angeforderte Seiten modifizieren und neue Dokumente erstellen. Auf diese Weise lassen sich beispielsweise bekannte Links mit zusätzlichen Informationen, wie Seitentitel, -größe und -sprache, versehen. Diese Informationen, die bei einem früheren Besuch in einer Datenbank abgelegt wurden, können dem Benutzer angezeigt werden bevor er den Link anklickt (vgl. Weinreich und Lamersdorf 2000). Ein weiteres Einsatzgebiet für WBIs ist das Einfügen von bestimmten Texten in bestehende Seiten. Maglio und Barrett schreiben dazu (vgl. Maglio und Barrett 2000):

„In fact, many contemporary Web services are intermediaries, performing such functions as transforming news feeds and airline reservation databases into categorized and personalized Web pages“.

Mit zusätzlichem Aufwand, wie dem Einbau eines Applets und JavaScript in jede aufgerufene Seite, lassen sich in Verbindung mit WBIs auch viele Aktionen des Benutzers abfangen (vgl. Haß 2002, Weinreich 2003). Dazu zählen beispielsweise angeklickte Links, besuchte URLs und benutzte Schaltflächen des Browsers. Die Benutzeraktionen werden dann beispielsweise per JavaScript erkannt und über das Applet an eine auswertende Applikation geschickt.

3.2.4 Steuern des Browsers

Das Steuern des Browsers ist nötig, um bei dem Test verschiedener Webseiten automatisiert die richtige Seite anzeigen zu lassen. Dies ist aber nur dann sinnvoll, wenn der Test entweder von dem Testleiter gesteuert wird, z. B. über einen anderen Rechner, oder wenn eine Applikation dem Benutzer die Aufgaben zeigt und an entsprechender Stelle die richtige Webseite aufruft.

Eine Möglichkeit, den Browser steuerbar zu machen, stellt die Modifikation eines Browsers dar (vgl. Tauscher und Greenberg 1997, siehe auch Abschnitt 3.2.2). Neben den Änderungen zum Abfangen von Benutzeraktionen ließe sich ein Browser mit frei verfügbarem Quelltext auch derart modifizieren, dass er für Steuerbefehle von außen empfänglich wäre.

Mit Hilfe der „Browser Helper Objects“ lässt sich der Internet Explorer ebenfalls zum Laden neuer Webseiten bewegen. Das funktioniert aber nur bei dem Internet Explorer Version 5 bis 6 und auch nur unter Windows.

Die Kombination von WBI und Java-Applet (siehe Abschnitt 3.2.3) bietet die Möglichkeit, per Socket-Verbindung über das Applet Webseiten im Browser aufzurufen und beliebige JavaScript-Befehle auszuführen. Auf diese Weise lassen sich viele Funktionen des Browsers steuern.

3.2.5 Stellen von Aufgaben

Eine weitere Funktion des Werkzeugs soll das Stellen der Aufgaben für den Teilnehmer sein. Der Aufgabentext sollte dem Teilnehmer auf konsistente Weise präsentiert werden und es sollte eine Eingabemöglichkeit für die erarbeitete Lösungen geben (siehe Abschnitt 3.3.2). Die Befragung von Teilnehmern wird gern mit dem Benutzertest kombiniert, daher sollte auch die Möglichkeit bestehen, Fragen oder Fragebögen in die automatisierten Tests mit einzubeziehen. Die Antworten sollten, wie bei den Aufgaben, für eine mögliche spätere Verwendung gespeichert werden.

In ihrer Arbeit „Using the Web as a Survey Tool: Results from the Second WWW User Survey“ entwickelten James E. Pitkow und Margaret M. Recker (vgl. Pitkow und Recker 1995) einen umfangreichen Fragebogen, den die Teilnehmer mit ihrem Browser aufrufen und beantworten konnten. Auf diese Weise erhielten sie über 18.000 Antworten, die vom System für die spätere Auswertung gespeichert wurden.

Nach Meinung des Autors gibt es aber kein Werkzeug, das dem Teilnehmer beliebig konfigurierbare Aufgaben oder Fragebögen präsentiert, die Antworten annimmt, einen Webbrowser steuert, die Aktionen des Teilnehmers während des Surfens abfängt und die gesammelten Daten für die weitere Verwendung speichert.

3.2.6 Fazit

Das zu erstellende Werkzeug soll möglichst universell einsetzbar sein und deswegen möglichst viele Aktionen des Benutzers erkennen und speichern. Besonders wichtig ist die Erkennung angeklickter Links oder die Benutzung der „Vor“- oder „Zurück“-Tasten des Browsers. Die Techniken, die sich der Protokoll-Dateien von Webserver oder Browser bedienen, bieten diese Informationen nicht und sind daher nicht geeignet. Auch die Erweiterung des Internet Explorers mit dem „Browser Helper Object“ liefert für viele Zwecke zu wenig detaillierte Benutzeraktionen.

Die Modifikation eines Browsers erlaubt sämtliche Benutzeraktionen abzufangen. Um einen Browser zu modifizieren, muss aber der Quelltext des Browsers verfügbar sein. Von den modifizierbaren Browsern ist Mozilla, zusammen mit dem sehr ähnlichen Net-

scape Navigator, der meistgenutzte. Die Verbreitung von nur 8% bedeutet aber, dass für viele Benutzer solch ein modifizierter Browser ungewohnt zu bedienen ist. Sie müssten sich beim Surfen anders verhalten als gewohnt, weil die Funktionen des Browsers anders als bei ihrem favorisierten Browser aufgerufen werden. Das kann bei bestimmten Fragestellungen zur Verfälschung der Testergebnisse führen.

Für das zu entwickelnde Werkzeug ist der Ansatz eines WBI, in Verbindung mit einem Java-Applet und JavaScript (vgl. Haß 2002, Weinreich 2003), vielversprechend. Auf diese Weise lassen sich, im Gegensatz zu dem Ansatz von Etgen und Cantor, auch viele Funktionen des Browsers als Benutzeraktion erkennen. Beispielsweise werden die Aktionen „Vor“, „Zurück“ und „Aktualisieren“ erkannt. Durch den Einsatz des WBI können beliebige Webseiten aufgerufen werden, ohne sie vorher modifizieren oder lokal speichern zu müssen.

Weitere Vorteile dieser Technik sind: Es lassen sich Webseiten im Browser aufrufen und beliebige JavaScript-Befehle ausführen. Für spezielle Tests kann auch das Aussehen beliebiger Webseiten automatisiert verändert werden, um beispielsweise die Wirkung von optisch veränderten Links zu untersuchen (vgl. Weinreich und Obendorf 2003).

Die übrigen Funktionen, wie das Stellen der Aufgaben, das Abfangen der Aktionen und Daten am Aufgabenfenster sowie das Einlesen der Testbeschreibungsdatei und das Ausgeben der Testprotokolle lässt sich in jeder höheren und fensterorientierten Programmiersprache erledigen. Allerdings sollten zumindest die Teile, die direkt auf den WBI zugreifen, in der gleichen Programmiersprache geschrieben werden, um die Kommunikation zu vereinfachen.

Das Framework Scone von Harald Weinreich (vgl. Weinreich 2003) bietet bereits die Ansteuerung eines WBI sowie eine Erkennung der Benutzeraktionen über ein Applet und JavaScript. Um die Funktionalität des in Java programmierten Frameworks vollständig ausnutzen zu können, bietet es sich an, das Projekt „UserTestTool“ ebenfalls in Java zu realisieren.

3.3 Unterstützung des Testleiters

In diesem Abschnitt werden die geläufigsten Teilaufgaben eines Benutzertests aufgeführt. Dabei wird untersucht, in wieweit die jeweilige Aufgabe von einem Computer übernommen werden kann und welche Vor- und Nachteile sich durch die Computerunterstützung ergeben.

3.3.1 Begrüßen und für eine entspannte Atmosphäre sorgen

Ein wichtiger Teil jedes Benutzertests ist die Begrüßung und das Sorgen für eine entspannte Atmosphäre. Solange der Teilnehmer glaubt, er würde getestet, steht er unter unnötigem Stress, macht möglicherweise unnötige Fehler und verhält sich unnatürlich. Da jeder Fehler des Teilnehmers auf einen Fehler in der Benutzungsschnittstelle hinweisen kann, stören diese durch Stress verursachten Fehler beim Deuten der Fehler im Interface. Nielsen empfiehlt des Weiteren, dem Teilnehmer zu versichern, dass keine Daten bezüglich der Leistungen individueller Teilnehmer weitergegeben werden, schon gar nicht an den Vorgesetzten (vgl. Nielsen 1993, Seite 182). Außerdem sollte auf eine entspannte Atmosphäre geachtet werden und Erfrischungen gereicht werden, besonders, wenn der Test mehr als eine Stunde dauert.

Die Begrüßung des Teilnehmers sollte weiterhin vom Testleiter vorgenommen werden, denn ein persönliches Gespräch kann sehr viel besser für eine entspannte Atmosphäre sorgen, als ein auf dem Computerbildschirm angezeigter Text. Auch die Tatsache, dass der Teilnehmer nicht getestet wird und keine individuellen Daten veröffentlicht werden, sollte dem Teilnehmer persönlich versichert werden. Diese Aufgaben durch den Computer übernehmen zu lassen, ist nach Meinung des Autors für diese Aufgabe eher kontraproduktiv.

3.3.2 Stellen der Aufgabe

Anders verhält es sich, wenn es um das Beschreiben der Aufgabe geht. Hier besteht die Gefahr, dass der Testleiter den Teilnehmer unbewusst beeinflusst, während er die Aufgabe erklärt. Dieser Effekt kann besonders dann auftreten, wenn die Studie von mehreren Testleitern durchgeführt wird, die die Aufgaben unterschiedlich vermitteln. Diese

Beeinflussung kann zu verfälschten Ergebnissen führen. Besonders beim Vergleich der Ergebnisse mehrerer Teilnehmer können falsche Schlüsse gezogen werden, wenn die einzelnen Teilnehmer unterschiedlich stark beeinflusst wurden.

Abhilfe schafft hier, die Aufgabe durch den Computer stellen zu lassen. Auf diese Weise wird ein Störfaktor der Untersuchung beseitigt, da jeder Teilnehmer die gleiche Aufgabenstellung erhält. Daraus resultiert eine bessere Reproduzierbarkeit der Testläufe und die Möglichkeit, mehrere Tests parallel mit unterschiedlichen Testleitern durchzuführen.

Beim Angeben des Aufgabentextes sollte der Testleiter die Möglichkeit haben, Textteile hervorzuheben und die Schriftgröße zu ändern. Des Weiteren sollten Schaltflächen definierbar sein, mit denen der Testverlauf von dem Teilnehmer gesteuert werden kann. Zum Beispiel sollte der Teilnehmer mit Hilfe einer Schaltfläche die fertiggestellte Aufgabe abschließen und die nächste starten können.

Häufig werden in Benutzertests Aufgaben gestellt, die von dem Teilnehmer eine Antwort erwarten. Ein Beispiel dafür sind Aufgaben, in denen Informationen innerhalb einer Website gesucht werden sollen. Damit der Testleiter die Antworten nicht selbst aufschreiben muss, sollte der Computer neben dem Anzeigen der Aufgabe auch die Möglichkeit bieten, die Antwort des Teilnehmers anzunehmen und zu speichern. Hier bietet sich ein Textfeld an, in dem die formulierte Antwort eingetragen werden kann.

Die Abfolge der Elemente, wie Text-, Textfeld- oder Schaltflächenkomponenten, sollte vom Testentwickler frei wählbar sein. Für Komponenten oder Komponentengruppen, die häufiger vorkommen, sollte die Möglichkeit bestehen, diese als Schablone zu definieren, um sie mit geringem Aufwand in die einzelnen Aufgaben einzubinden. Es sollte auch die Möglichkeit bestehen, einzelne Komponenten aus diesen Gruppen beim Einbinden leicht zu verändern. Beispielsweise sollte eine vordefinierte Likert-Skala bei jedem Einbinden mit einer neuen Frage versehen werden können.

3.3.3 Befragungen

Benutzertests werden gern durch Befragungen ergänzt, in denen die Meinungen der Teilnehmer erfasst werden. Dies wird im Allgemeinen mit Fragebögen erreicht. Dabei hat sich die Likert-Skala bewährt, in der der Teilnehmer seine Zustimmung zu einer gestellten Frage mit Hilfe von meist 5 oder 7 Einteilungen ausdrücken kann. Der Vorteil dieser Art der Befragung ist die statistische Auswertbarkeit.

Fragebögen lassen sich auch durch den Computer anzeigen. Sie können vom Teilnehmer bearbeitet werden. Die eingegebenen Daten können für eine spätere Verwendung gespeichert werden.

3.3.4 Steuerung des Webbrowsers

Wenn der Teilnehmer im Rahmen einer Aufgabe Informationen auf einer Website suchen sollte, musste der Testleiter bisher die entsprechende Startseite selbst aufrufen. Auch dies lässt sich gut durch den Computer erledigen.

Der Teilnehmer könnte beispielsweise den Aufgabentext lesen und eine Schaltfläche anklicken, sobald er die Aufgabe verstanden hat. Ausgelöst durch den Klick auf die Schaltfläche sollte dann eine bestimmte Webseite im Browserfenster erscheinen.

Genau wie das Laden einer bestimmten Seite im Browser sollte es auch möglich sein, eine leere Seite im Browser anzeigen lassen. Dies könnte geschehen, sobald der Teilnehmer die Information gefunden hat und daraufhin eine entsprechende Schaltfläche anklickt.

Mit Hilfe der beiden Schaltflächen zum Anzeigen der Startseite und einer leeren Seite könnte auch eine Stoppuhr betätigt werden, die die Zeit der Suche im Internet festhält (siehe Abschnitt 3.3.7).

3.3.5 Protokollieren der Eingaben und Aktionen

Die meiste Arbeit eines Tests mit Benutzern wird durch die Auswertung verursacht. Einen Teil der Arbeit der Auswertung macht das Protokollieren der Eingaben und

Aktionen des Teilnehmers aus. Die gefundene Antwort auf die gestellte Aufgabe oder die abgegebene Wertung zu einer Frage eines Fragebogens muss beim klassischen Benutzertest erst vom Testleiter abgetippt werden.

Alle Eingaben und jeder Klick auf eine Schaltfläche sollte – separat für jede Aufgabe – gespeichert werden. Auf diese Weise muss der Testleiter die Antworten der Teilnehmer nicht selbst eingeben. Die angeklickten Schaltflächen helfen beim Nachvollziehen des Testablaufs.

Die Aktionen und Eingaben einer Aufgabe sollten erst dann gespeichert werden, wenn die Aufgabe erfolgreich abgeschlossen wurde. Dadurch lässt sich vermeiden, dass bei einer Unterbrechung des Tests unsinnige Werte in eine spätere Auswertung einfließen.

3.3.6 Protokollieren der Aktionen beim Surfen

Die Benutzeraktionen des Teilnehmers beim Surfen zu protokollieren, kann einen großen Aufwand bedeuten. Dieser Aufwand ist von der Palette der benötigten Aktionen abhängig. Wenn es genügt, die besuchten Webseiten zu protokollieren, kann das von dem Testleiter möglicherweise noch während des Testlaufs gemacht werden. Falls aber auch von Bedeutung ist, wie der Benutzer die Seiten wechselt und wie lange er auf jeder Seite verweilt, dann müssen diese Informationen nachträglich durch stundenlanges Herumspulen im Video gesammelt werden. Diese Daten lassen sich aber auch mit Hilfe des Computers abfangen (siehe Abschnitte 3.2.1-3.2.3).

Die Aktionen, die der Teilnehmer beim Surfen mit dem Browser ausführt, sollten gespeichert werden, um dem Testleiter die aufwendige Suche in der Videoaufnahme zu ersparen. Besonders wichtig sind die Aktionen, die zum Seitenwechsel geführt haben: z. B. geklickter Link, abgeschicktes Formular, „Vor“- und „Zurück“-Taste des Browsers und selbst eingegebener URI. Des Weiteren sollten die besuchten URIs gespeichert werden, um später den Weg des Teilnehmers durch die Website nachvollziehen zu können.

Jede Aktion sollte mit einem Zeitstempel versehen werden, um später die Verweilzeiten auf den einzelnen Seiten errechnen zu können. Für den Fall, dass der Benutzer mehrere Browserfenster benutzt, sollten die einzelnen Fenster eindeutige Namen bekommen und

diese zusammen mit jeder Aktion gespeichert werden. Dadurch lassen sich später die einzelnen Aktionen ihren Fenstern zuordnen.

3.3.7 Stoppen der Zeit für bestimmte Aufgaben

Die benötigte Zeit für eine Aufgabe kann interessant sein, wenn die Performanz eines Teils einer Website beurteilt werden soll. Zum Messen von Zeitspannen hat der Testleiter beim traditionellen Benutzertest im Allgemeinen keine Zeit, da er mit dem Deuten von Problemen oder dem Protokollieren beschäftigt ist. Diese Arbeit müsste später mit Hilfe der Videoaufzeichnung nachgeholt werden.

Um die gemessene Zeit nicht durch das Lesen der Aufgabe seitens des Benutzers zu verfälschen, sollte die Zeitmessung erst dann gestartet werden, wenn der Teilnehmer die Aufgabe verstanden hat. Dies lässt sich durch eine Schaltfläche erreichen, die gleichzeitig mit der Zeitmessung auch das Anzeigen der Startseite im Webbrowser (siehe Abschnitt 3.3.4) auslöst. Sobald der Teilnehmer meint, er hätte die Information gefunden, kann er durch das Anklicken einer anderen Schaltfläche die Zeitmessung stoppen und die gefundene Information in ein Textfeld eintragen. Erscheint das Textfeld erst nach dem Betätigen der Schaltfläche, wird die Zeit für das Eingeben der Information nicht mitgemessen. Durch das Anklicken der Schaltfläche könnte auch die angezeigte Website aus dem Browserfenster entfernt werden, um dem Teilnehmer nicht weiterhin die Möglichkeit zu geben, in der Webseite zu lesen oder zu navigieren und dabei Informationen zu finden.

Gestoppte Zeiten sind allerdings nur dann sinnvoll, wenn es sich nicht um einen Thinking-Aloud-Test handelt, da das Verbalisieren der Gedanken die benötigte Zeit für eine Aufgabe heraufsetzt. Auch ist zu beachten, dass die Arbeitsgeschwindigkeit einzelner Individuen stark variieren kann.

3.3.8 Deuten der Probleme des Benutzers

Das Deuten der Probleme des Benutzers beim Interagieren mit einer Website ist der schwierigste Teil beim Testen mit Benutzern. Der Testleiter muss aus den verbalisierten

Gedanken des Teilnehmers Probleme heraushören und versuchen, dafür verantwortliche Fehler in der Benutzungsschnittstelle zu finden.

Ivory und Hearst diskutieren Arbeiten, in denen versucht wird, Web-Interfaces automatisiert auf Fehler hin zu analysieren (vgl. Ivory und Hearst 2001, Seite 489). Derartige Studien sind aber nicht Teil dieser Arbeit und werden daher nicht weiter untersucht.

Das Deuten der Fehler der Benutzerschnittstelle in einem Benutzertest lässt sich nach Meinung des Autors nicht durch Computer übernehmen. Der Computer kann dem Testleiter nur möglichst viele Routinearbeiten abnehmen, damit sich dieser voll auf seine Aufgabe konzentrieren kann.

3.4 Besonderheiten der Automatisierung

In diesem Abschnitt werden besondere Aspekte beschrieben, die bei Tests mit Benutzern auf herkömmliche Weise nicht bestanden. Abschnitt 3.4.1 beschreibt neue Möglichkeiten, die durch die Verwendung eines Web-Intermediarys entstehen. Abschnitt 3.4.2 behandelt den Abbruch und die Wiederaufnahme von Tests, die im Gegensatz zur klassischen Benutzertestvariante explizit bereitgestellt werden müssen. Abschnitt 3.4.3 gibt vor, auf welche Weise die einzelnen Testläufe definiert werden sollten. Abschnitt 3.4.4 behandelt die Ausgabe der Daten, die während eines Testlaufes gesammelt werden.

3.4.1 Verändern von Webseiten durch Intermediaries

Durch die Verwendung eines Web-Intermediarys lässt sich jede im Browser angeforderte Seite auf der Ebene der HTML-Tags modifizieren, d. h. jede HTML-Datei lässt sich beliebig um andere Tags erweitern, es lassen sich bestimmte Tags entfernen und sogar neue HTML-Seiten oder andere Dateien generieren. Sollen also bestimmte Webseiten um Text oder Funktionalität erweitert werden, ist dies mit Hilfe von Web-Intermediaries möglich.

Eine technischere Beschreibung folgt im vierten Kapitel, in dem der benutzte WBI vorgestellt wird und die Einzelheiten von dessen Verwendung beschrieben werden.

3.4.2 Abbruch und Wiederaufnahme von Tests

Bei einem Benutzertest ohne Computerunterstützung ist das Abbrechen und Wiederaufnehmen von Tests kein Problem. Wenn der Teilnehmer kurzfristig einen anderen Termin wahrnehmen muss, wird der Test einfach fortgesetzt, sobald der Benutzer wieder Zeit hat.

Anders verhält es sich bei computerunterstützten Tests. Hier muss die Möglichkeit des Testabbruchs und der Wiederaufnahme explizit bereitgestellt werden. Im Falle eines Abbruchs sollten alle bis dahin vollständig bearbeiteten Aufgaben gespeichert werden. Sobald der Teilnehmer wieder Zeit hat, sollte der abgebrochene Testlauf mit der ersten Aufgabe fortgesetzt werden, die nicht vollständig beendet wurde. Eine möglicherweise angefangene Aufgabe sollte vollständig wiederholt werden, um Zeitmessungen für bestimmte Aufgabenteile fehlerfrei durchführen zu können (siehe 3.3.7).

3.4.3 Definition des Testablaufes

Die Definition eines Testablaufes sollte in einem leicht verständlichen Format geschehen, so dass der Testentwickler nicht über Kenntnisse einer bestimmten Programmiersprache verfügen muss. Hier bietet sich XML wegen der weiten Verbreitung, der Möglichkeit zur Strukturierung und der relativ guten Lesbarkeit an.

Die einzelnen Aufgaben, die zusammen einen Test bilden, sollten in einer gemeinsamen Datei definiert werden. Für jede Aufgabe sollte ein XML-Tag angegeben werden. In jedem dieser Tags sollten nacheinander die gewünschten Komponenten als XML-Tags in der Reihenfolge aufgeführt werden, in der sie während des Tests im Aufgabenfenster erscheinen sollen. Jeder Komponenten-Tag sollte sich mit zusätzlichen Angaben konfigurieren lassen, um beispielsweise für eine Textfeld-Komponente eine Überschrift und den Inhalt des Textfeldes festzulegen. Die Likert-Skala sollte sich auf beliebig viele Unterteilungen einstellen lassen. Es sollte möglich sein, Programmierkenntnisse vorausgesetzt, die Testumgebung bei Bedarf um weitere Komponenten zu erweitern.

Jede Komponente, die sich anklicken oder mit Text füllen lässt, sollte einen Namen erhalten. Anhand dieses Namens lässt sich dann in der Protokolldatei erkennen, was der Benutzer in welches Feld eingetragen hat, oder welche Schaltfläche er angeklickt hat.

3.4.4 Bereitstellen aller gesammelten Daten

Alle gesammelten Daten, wie die eingegebenen Antworten, angeklickte Schaltflächen und Links sowie alle anderen Aktionen, sollten in einer Protokolldatei gespeichert werden. Um eine bessere Übersicht zu gewährleisten, sollten die Daten getrennt nach Aufgabe abgelegt werden. Innerhalb der Aufgabe bietet sich eine chronologische Abfolge der Aktionen an. Die Eingabefelder sollten am Ende jeder Aufgabe ausgelesen und an die Protokolldatei angehängt werden, da sie keinen eindeutigen Aktionszeitpunkt besitzen und somit nicht chronologisch eingeordnet werden können.

Diese Protokolldateien sollten im XML-Format abgespeichert werden, damit die Möglichkeit besteht, die Daten durch andere Programme weiterverarbeiten zu lassen. Außerdem lässt sich XML, mit etwas Übung, auch von Menschen lesen.

Jede Aktion und jedes ausgelesene Textfeld wird durch einen XML-Knoten beschrieben, der die relevanten Daten, wie Aktion, Zeitpunkt oder eingegebene Antwort, in den Attributen speichert.

Um die Daten auch in Programmen wie Microsoft Word, Excel oder ähnliche importieren zu können, sollten diese zusätzlich zum XML-Format auch im TDF-Format (Tab Delimited File) mit Tabstop-Trennung gespeichert werden. Auf diese Weise lässt sich die Datei in viele Datenbank- oder Tabellenverwaltungsprogramme einlesen und weiterverarbeiten.

3.5 Zusammenfassung

Dieses Werkzeug soll den Testleiter bei der Durchführung von Tests mit Benutzern bestmöglich entlasten, damit sich dieser auf die Deutung der Probleme konzentrieren kann, die ein Benutzer mit der getesteten Website hat.

Die Entlastung des Testleiters wird erreicht, indem viele Routineaufgaben vom Computer übernommen werden (siehe Abschnitt 3.3). Dazu gehören: das Stellen der Aufgaben, das Steuern des Browsers, das Protokollieren der Benutzeraktionen am Aufgabenfenster und am Browser, das Protokollieren der besuchten URIs und das Erweitern aller Aktionen um einen Zeitstempel. Die gesammelten Daten werden in einer Datei, unterteilt nach Aufgaben, abgelegt und lassen sich in viele Programme zur Nachbearbeitung importieren.

Der Ablauf des Tests wird durch eine XML-Datei gesteuert, die für jede Aufgabe die anzuzeigenden Komponenten aufführt (siehe Abschnitt 3.4.3). Diese Komponenten lassen sich für den jeweiligen Zweck anpassen, in dem beispielsweise ein frei wählbarer Text für die Überschrift angegeben werden kann. Programmierkenntnisse sind für die Definition einer Testablaufdatei nicht nötig.

Das Steuern des Browsers und das Abfangen der Benutzeraktionen wird durch eine Kombination aus Web-Intermediary, Java-Applet und JavaScript erreicht (siehe Abschnitt 3.2.6). Die Applikation, die den Test steuert wird in Java geschrieben, um mit dem WBI und Scone (Weinreich 2003) kommunizieren zu können.

4 Implementation

In diesem Kapitel wird die Implementation des UserTestTools vorgestellt. Einleitend beschreibt Abschnitt 4.1 grob die Funktionsweise des Werkzeugs. Abschnitt 4.2 gibt einen Überblick über das FrameWork Scone, in das das UserTestTool integriert wurde. Abschnitt 4.3 beschreibt ausführlich die Konzeption sowie die einzelnen Klassen und deren Interaktion. Abschnitt 4.4 beschäftigt sich mit Besonderheiten der Implementation.

4.1 Funktionsweise des UserTestTools

Das UserTestTool lässt sich starten, indem der Internet Explorer geöffnet wird und die Webseite „http://usertest.scone.de“ aufgerufen wird. Dieser Aufruf wird vom UserTestTool erkannt und daraufhin das Testauswahlfenster angezeigt (siehe Abbildung 4.1).

Das Auswahlfenster erlaubt die Auswahl der Testbeschreibungsdatei, die den Ablauf des Tests steuern soll. Als weitere Eingaben werden eine Nummer für diesen Test und der Name des Testteilnehmers erwartet. Als Testnummer wird die nächste unbenutzte Zahl für den ausgewählten Test vorgeschlagen.

Alternativ erlaubt das Auswahlfenster, einen abgebrochenen Testlauf wieder aufzunehmen. Dabei kann angegeben werden, mit welcher Aufgabe begonnen werden soll, sofern dabei keine unbearbeiteten Aufgaben übersprungen werden.

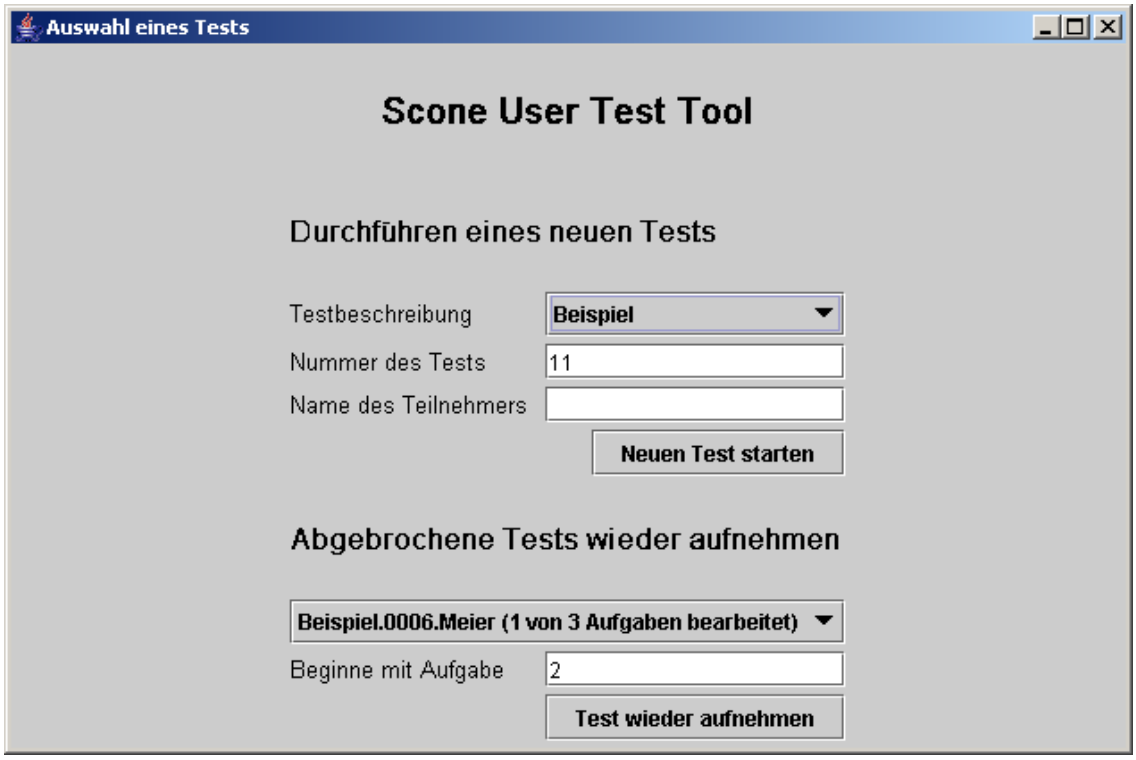


Abbildung 4.1: Testauswahlfenster des UserTestTools

Je nach ausgewählter Testablaufdatei wird nun die erste Aufgabe angezeigt. Das Aussehen dieser Aufgabe wird durch die Testablaufdatei bestimmt, die im XML-Format angegeben wird. In ihr werden die anzuzeigenden Komponenten, wie Textanzeige-, Schaltflächen-, Texteingabefeld-, Ausklappbox- (ComboBox) und Likert-Skala-Komponenten in der Reihenfolge aufgeführt, in der sie in dem Aufgabenfenster erscheinen sollen. Die Schaltflächen können beim Anklicken bestimmte Aktionen auslösen. Diese werden ebenfalls in der Testablaufdatei angegeben. Eingabekomponenten, wie die Texteingabefeld-, Ausklappbox- und Likert-Skala-Komponenten erlauben dem Testteilnehmer einen Text einzugeben, einen vordefinierten Text auszuwählen oder eine Wertung abzugeben. Diese Eingaben und Aktionen des Teilnehmers werden gespeichert.

Angeklickte Schaltflächen können andere Komponenten farblich hervorheben, aktivieren oder deaktivieren. Deaktivierte Komponenten werden gräulich dargestellt und lassen sich nicht mehr bedienen. Außerdem können Webseiten im Browser aufgerufen, die Größe des Browsers verändert und das Browserfenster in den Vorder- oder Hintergrund

geschickt werden. Es lässt sich auch eine Stoppuhr starten oder die nächste Aufgabe aufrufen.

Ein Testlauf kann aus einer beliebigen Anzahl von Aufgaben bestehen, die von dem Testteilnehmer abgearbeitet werden. Jede Aufgabe wird durch das Anklicken einer dafür vorgesehenen Schaltfläche beendet. Daraufhin wird die nächste Aufgabe gestartet, sofern diese vorhanden ist.

Die gesammelten Daten eines Testlaufes werden in zwei Protokolldateien abgelegt, einmal im XML-Format und einmal im TDF-Format (Tab Delimited File). Die Datei im XML-Format dient dem UserTestTool für interne Zwecke, ließe sich aber auch in speziellen Auswertungsprogrammen als Datenquelle nutzen. Das TDF-Format ist eine Variation des CSV-Formats (Comma Separated Values) und besonders bei Datenbank- und Tabellenkalkulationsprogrammen ein verbreiteter Standard. Daher lässt sich das TDF-Protokoll beispielsweise in Applikationen wie Microsoft Excel importieren. Weitere Informationen zur Weiterverwendbarkeit der Protokolle werden in den Abschnitten 4.4.3 und 7.2 angegeben.

Die Funktionsweise des UserTestTools wird genauer in der Benutzerdokumentation beschrieben, die im Anhang A abgedruckt ist. Anhang B und C geben eine Komponentenreferenz und eine Liste der möglichen Einträge der Protokolldateien an.

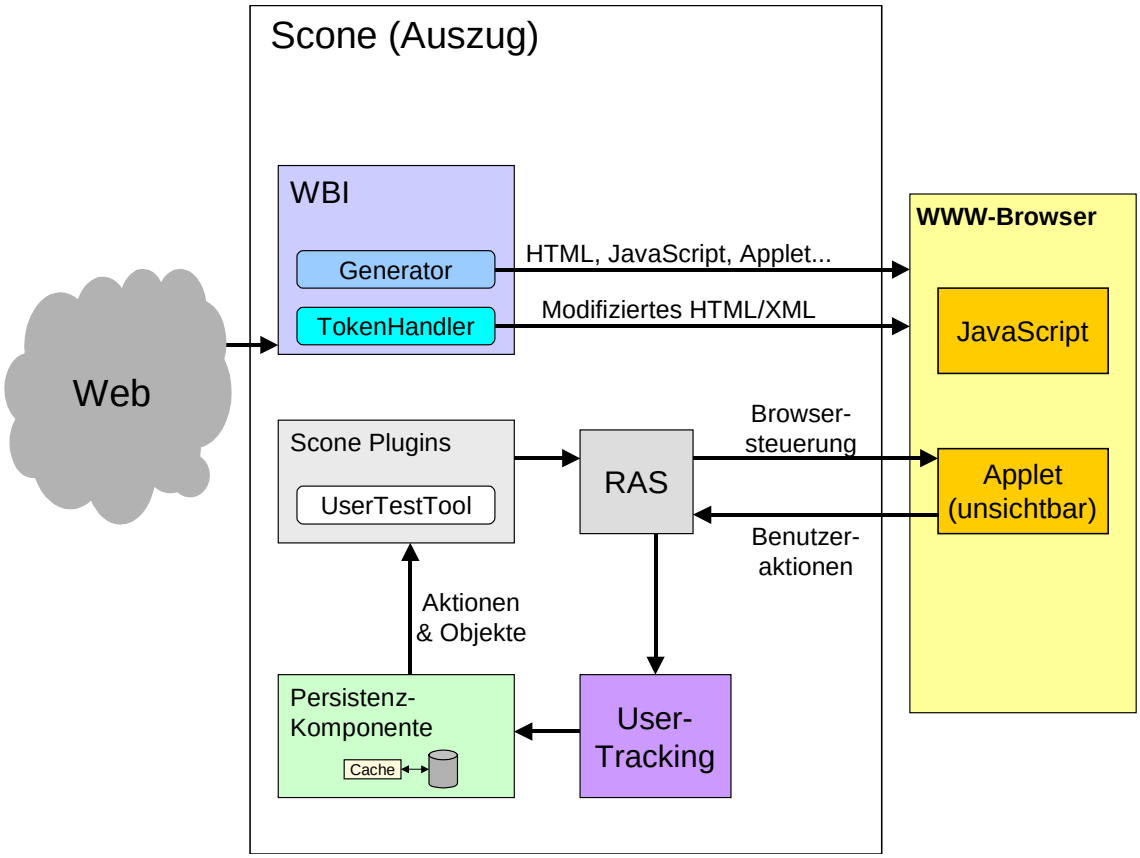
4.2 Überblick über Scone

Scone (Weinreich, Buchmann, Lamersdorf 2003, Weinreich 2003) ist ein Java-Framework, das die Entwicklung und Evaluation von Web-Erweiterungen zur Navigation und Orientierung im Web unterstützt. Das Framework bietet eine Reihe von Komponenten, die beispielsweise die Änderung der Anzeige von Dokumenten im Browser, sowie die Reaktion auf die Aktionen des Benutzers erlauben. Des Weiteren ist es möglich, im Browser Webseiten aufzurufen und JavaScript-Befehle auszuführen.

Abbildung 4.2 zeigt die Komponenten von Scone, die für das UserTestTool relevant sind. In den folgenden Abschnitten werden die einzelnen Komponenten näher beschrieben.

4.2.1 WBI

Der Web-Intermediary von IBM (vgl. IBM 2003) ist eine wichtige Komponente des Frameworks. Web-Intermediaries, auch WBIs genannt, werden als Proxy-Server in die HTTP-Verbindung zwischen Browser und Webserver integriert und können so die *Requests* und *Responses* überwachen und manipulieren. Auf diese Weise ist es beispielsweise möglich, Seitenaufrufe zu einem anderen Webserver umzubiegen oder die angeforderte HTML-Seite zu modifizieren, bevor sie an den Browser weitergeleitet wird.



Abbilung 4.2: Übersicht über Scone

WBIs verfügen über drei Funktioneinheiten, die mit dem „request-response stream“ zwischen Browser und Webserver interagieren: *Monitor*, *Editor* und *Generator* (vgl. Barret, Maglio und Kellem 1997). Monitore erhalten eine Kopie des gesamten

Kommunikationsstroms, und können beliebige Informationen speichern. Beispielsweise können sie Informationen über die bereits besuchten Seiten sammeln, um diese später den anderen Funktionseinheiten zur Verfügung zu stellen.

Ein Editor erlaubt, die Daten des Kommunikationsstroms zu verändern. Auf diese Weise lassen sich aufgerufene Seiten um weitere Informationen, z. B. zusätzliche Links oder andere Funktionalität, erweitern. Die Einbindung von Informationen kann dabei abhängig von bestimmten HTML-Tags geschehen. Es ist auch möglich, das Aussehen bestimmter Elemente auf jeder Webseite zu verändern. Harald Weinreich und Hartmut Obendorf (vgl. Weinreich und Obendorf 2003) haben auf diese Weise das Aussehen von Links modifiziert.

Generatoren bieten die Möglichkeit eingehende Anfragen in entsprechende Antworten umzuwandeln, d. h. der Generator kann den Aufruf einer Datei entgegennehmen und eine entsprechende Datei, etwa eine HTML-Datei, erzeugen und zurückschicken. Es lassen sich aber auch alle anderen geläufigen Formate des Webs zurückliefern. In die Erstellung einer solchen Datei können auch andere Informationen mit einfließen, z. B. solche, die durch Monitore gesammelt wurden.

In dem Fall von Scone ergänzt der WBI jede aufgerufene Seite um ein unsichtbares Applet und JavaScript-Code. Der JavaScript-Code fängt viele der Benutzeraktionen ab, dazu gehören angeklickte Links, abgeschickte Formulare und andere. Diese Aktionen werden als Textbotschaften an das Applet und von dort aus über eine TCP/IP-Verbindung an Scone weitergegeben. Scone speichert diese Informationen und stellt sie den Plugins zur Verfügung (vgl. Haß 2003). Über das Applet ist es auch möglich, den Browser von Scone aus zu steuern. Es lassen sich Webseiten aufrufen und beliebige JavaScript-Befehle ausführen.

4.2.2 Remote Access Server (RAS)

Der Remote Access Server erlaubt den Plugins von Scone TCP/IP-Verbindungen aufzubauen oder entgegenzunehmen. Auf diese Weise werden die Daten über die aufgetretenen Benutzeraktionen vom Applet zu Scone geschickt und der Browser von Scone aus gesteuert.

Applets ist es nur erlaubt, TCP/IP-Verbindungen zu dem Server aufzubauen, von dem das Applet geladen wurde. Diese Beschränkung ist Teil des Sandkasten-Modells, das von Sun in das Java Development Kit eingebaut wurde, um Applets unbekannter Herkunft den Zugriff auf den Rechner des Benutzers zu verweigern (vgl. Sun Microsystems 1999). Da das Applet von WBI zur Verfügung gestellt wird, kann die Verbindung des Applets von der RAS-Komponente entgegengenommen werden, ohne das Sandkasten-Modell umgehen zu müssen.

4.2.3 UserTracking

Die wichtigste Komponente für das UserTestTool ist das UserTracking von Scone. Es dekodiert die Nachrichten des Applets und speichert die erkannten Benutzeraktionen. Die folgenden Benutzeraktionen werden erkannt: Angeklickte Links, abgeschickte Formulare, eingegebene URLs, sowie die Browserfunktionen Vor, Zurück und Aktualisieren. Das UserTracking erkennt vom Benutzer angeklickte Links und unterscheidet sogar zwischen Links zu einer anderen Webseite, Links zur gleichen Seite und Links zu bestimmten Seitenfragmenten (Anchors). Es erkennt das Abschicken von Formularen und speichert seit der Modifikation für das UserTestTool auch den Inhalt der einzelnen Formularfelder ab. Auch vom Benutzer eingegebene URIs werden vom UserTracking festgestellt. Zusätzlich werden Informationen wie die Seitenladezeit und Verweilzeit, Name und URL jeder Webseite gespeichert. Die gesammelten Daten werden in einer Datenbank abgelegt. Die Plugins von Scone, zu denen auch das UserTestTool gehört, werden von den wichtigsten Aktionen über ein Observer-Pattern unterrichtet. Weitere Informationen über die Funktionsweise des UserTrackings sind in der Studienarbeit des Autors (vgl. Haß 2003) zu finden.

4.2.4 Persistenzkomponente

In der Persistenzkomponente werden alle Informationen über die Benutzeraktionen, die besuchten Webseiten, die verfügbaren Links und viele weitere Informationen abgelegt. Diese Plugins können objektorientiert auf derartige „NetObjects“ zugreifen. Damit der relativ langsame Zugriff auf den Sekundärspeicher nicht die Performanz von Scone

beeinträchtigt, bietet Scone ein Caching-Konzept, sowie eine Mapping-Strategie von Java-Objekten auf relationale Datenbanken an.

4.2.5 Plugins

Das Framework Scone wurde entwickelt, um bei der Entwicklung von Navigationshilfen und ähnlichen Programmen zu helfen. Diese Erweiterungen lassen sich als Plugins in Scone einbinden. Scone erlaubt den Plugins die Benutzung aller bereitgestellten Komponenten. Neben den hier erwähnten Komponenten bietet Scone noch einen Robot, der beispielsweise ganze Websites indizieren kann. Auch das UserTestTool ist ein solches Plugin.

Alle für Scone entwickelten Erweiterungen lassen sich natürlich auch mit dem UserTestTool testen.

4.3 Konzeption des UserTestTools

Abbildung 4.3 gibt eine grafische Übersicht über das UserTestTool, seine Klassen und seine Anbindung an Scone. Die Pfeile zwischen den Klassen zeigen den gerichteten Datenfluss an. Unter dem Namen „Klassen für das Aufgabenfenster“ sind all die Klassen zusammen gefasst, die für die Darstellung des Aufgabenfensters und dessen Inhalt verantwortlich sind. Die Klassen für das Aufgabenfenster werden in Abbildung 4.4 dargestellt. Auch hier zeigen die Pfeile die Richtung des Datenflusses an.

In den folgenden Abschnitten werden die Aufgaben und Funktionen jeder Klasse, sowie deren Interaktion mit anderen Klassen beschrieben.

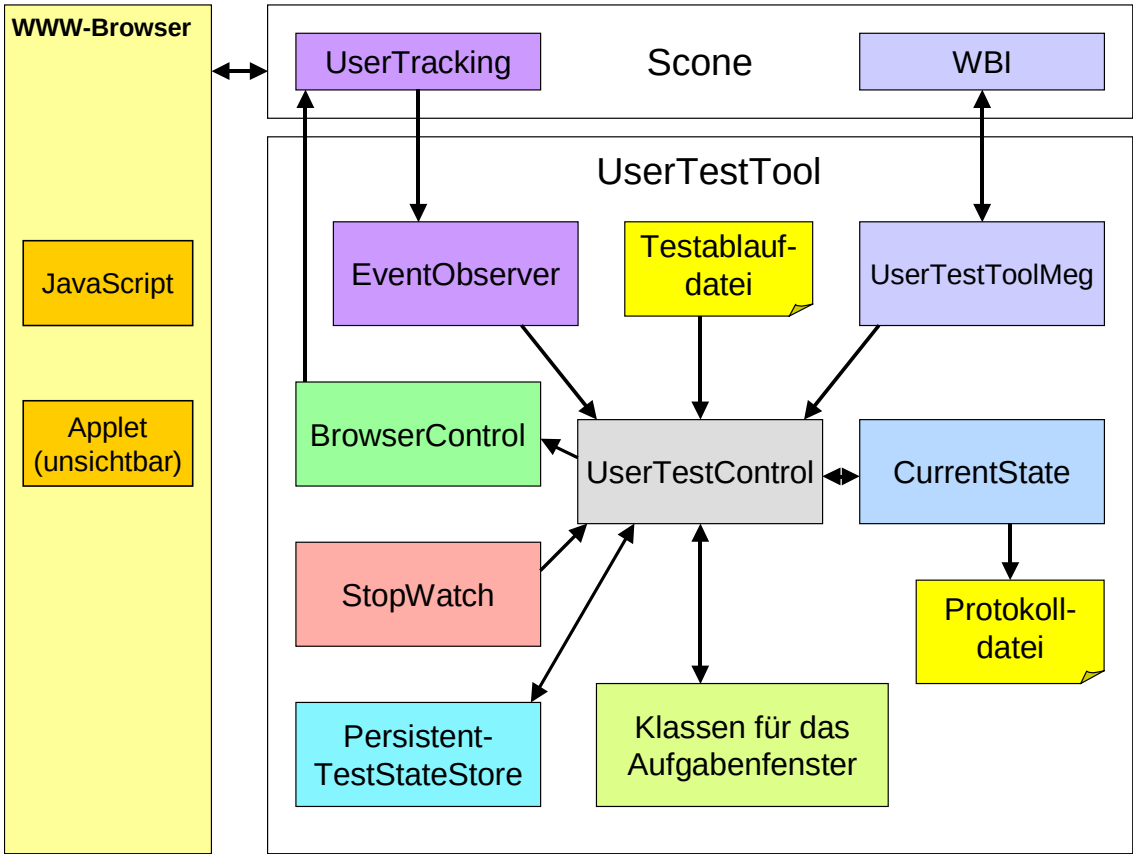


Abbildung 4.3: Klassen- und Datenflussdiagramm des UserTestTools

4.3.1 UserTestControl

Das Herzstück des UserTestTools stellt die Klasse UserTestControl dar. Sie koordiniert die einzelnen Aktionen des Benutzertests.

Sobald im Internet Explorer der URI „http://usertest.scone.de“ aufgerufen wird, erkennt der UserTestToolMeg dieses und ruft eine Methode von UserTestControl auf. Diese erzeugt ein Java-Fenster, das dem Testleiter die Möglichkeit bietet, einen Testverlauf auszuwählen oder einen abgebrochenen Test fortzusetzen. UserTestControl öffnet dann die entsprechende Testbeschreibungsdatei und zeigt die erste Aufgabe im Aufgabenfenster an. Im Fall eines wieder aufgenommenen Tests wird die vom Testleiter angewählte Aufgabe angezeigt.

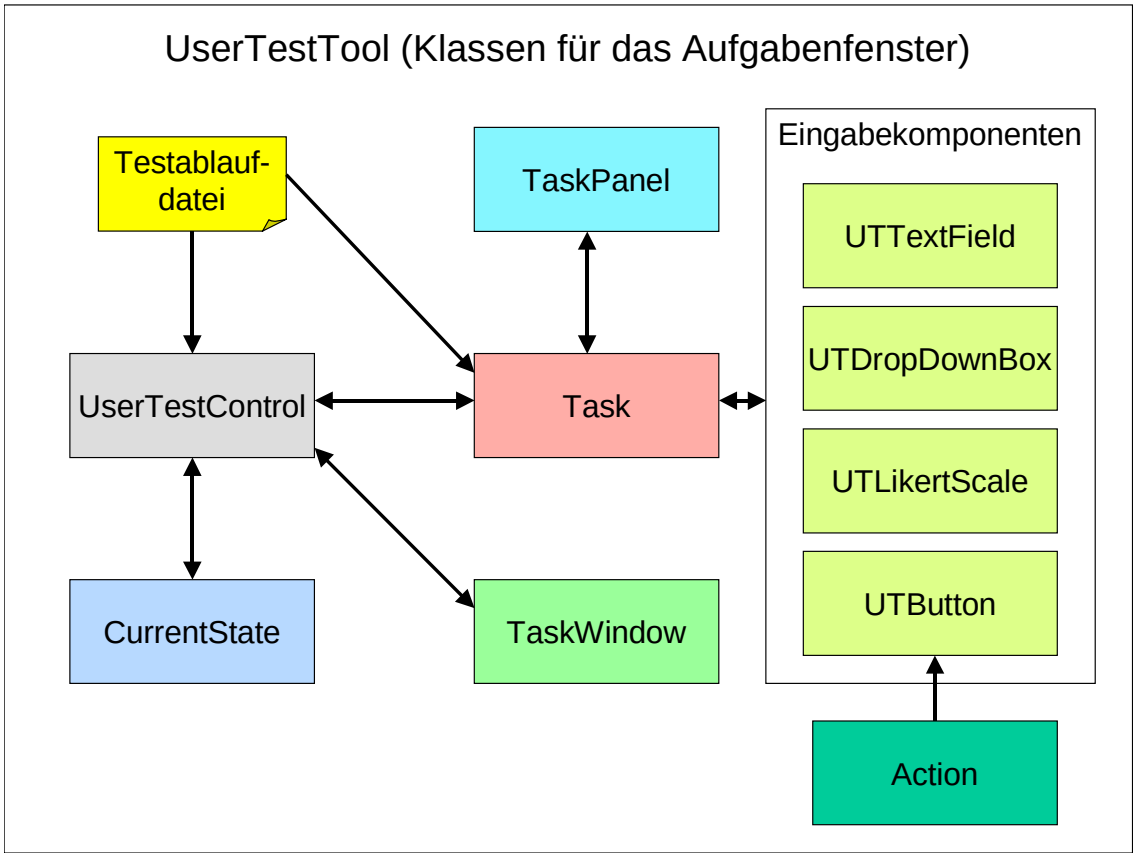


Abbildung 4.4: Klassen- und Datenflussdiagramm für das Aufgabenfenster

Alle vom Testteilnehmer angeklickten Schaltflächen des Aufgabenfensters rufen Methoden der UserTestControl-Klasse auf. Die Liste der auszuführenden Aktionen wird aus der Schaltflächenkomponente ausgelesen und von UserTestControl ausgeführt. Lediglich die Aktionen, die den Browser steuern, werden nicht von UserTestControl ausgeführt, sondern an BrowserControl weitergegeben. Die Benutzeraktionen, die der Teilnehmer mit dem Web-Browser ausführt, werden über den EventObserver gemeldet. Alle Aktionen, sowohl vom Aufgabenfenster als auch vom Browser, werden an die Klasse CurrentState übergeben, die diese speichert. Am Ende jeder Aufgabe werden die Eingabekomponenten ausgelesen, in die der Teilnehmer Texte eingeben oder auswählen und in denen er Wertungen abgeben konnte. Diese Daten werden ebenfalls an die Klasse CurrentState übergeben, die die gesammelten Daten der Aufgabe dann in die Protokolldateien schreibt.

Nach jeder vollendeten Aufgabe wird der aktuelle Fortschritt an die Klasse PersistentTestStateStore weitergegeben, in der der Zustand jedes Testlaufs und die verfügbaren Testbeschreibungsdateien abgelegt sind.

4.3.2 PersistentTestStateStore

Die Klasse PersistentTestStateStore speichert den Zustand jedes bisher durchgeführten Tests und jeder vorhandenen Testbeschreibungsdatei, um diese Daten beispielsweise für das Testauswahlfenster sofort bereit zu haben. Diese Daten werden in Form einer XML-Datei persistiert.

Bei jedem Start von Scone werden die Daten mit den vorhandenen Testbeschreibungsdateien und Testablaufdateien abgeglichen, um neue Testbeschreibungsdateien aufzunehmen und gelöschte Dateien aus der Liste zu entfernen.

4.3.3 UserTestToolMeg

Die Klasse UserTestToolMeg stellt einen von WBIs Generatoren (siehe Abschnitt 4.2.1) dar. Dieser Generator wartet auf den Aufruf des URI „http://usertest.scone.de“, fängt diesen ab und erzeugt eine HTML-Seite, die er an den Browser zurückschickt. Diese Seite enthält JavaScript-Code, der den Browser in den Hintergrund schickt, damit dieser nicht das Testauswahlfenster verdeckt.

Der WBI-Generator erzeugt noch eine andere HTML-Seite, die im Fall eines beendeten oder abgebrochenen Tests geladen wird. Diese Seite bietet dem Testleiter einen Link zum Starten eines neuen Tests, der auf den URI „usertest.scone.de“ verweist. Außerdem wird dem Testleiter die Möglichkeit zum Schließen des Browserfensters angeboten.

4.3.4 EventObserver

Der EventObserver fordert bei Scone die Benachrichtigung über sämtliche Benutzerereignisse an. Diese werden von Scone über ein Observer-Pattern bereitgestellt. Jedes Mal, wenn der Teilnehmer einen Link anklickt, oder auf andere Weise eine Webseite aufruft, wird dies von Scone erkannt und der EventObserver darüber informiert. Der EventObserver leitet die Benutzeraktion über UserTestControl weiter an CurrentState,

in dem die einzelnen Aktionen gespeichert werden und später in die Protokolldatei geschrieben werden.

4.3.5 BrowserControl

Die Klasse BrowserControl dient zur Steuerung des Web-Browsers. Damit lassen sich neue Webseiten im Browser aufrufen. Zusätzlich erlaubt diese Klasse das Ändern der Größe des Browserfensters und der Browser lässt sich in den Vorder- bzw. Hintergrund schicken.

Die Steuerung des Browsers geschieht über Scone. Scone schickt eine entsprechende Textnachricht an das Applet, dass die neue Seite aufruft oder entsprechende JavaScript-Befehle ausführt.

4.3.6 TaskWindow

Die Klasse TaskWindow erzeugt das Fenster, in dem die Aufgabe für den Teilnehmer angezeigt wird. Der Inhalt der Aufgabe, ein TaskPanel-Objekt, wird durch die Klasse Task erstellt und dann an TaskWindow übergeben.

4.3.7 Task

UserTestControl ruft die Klasse Task auf und übergibt ihr die Nummer der aktuellen Aufgabe. Task arbeitet die XML-Tags dieser Aufgabe aus der Testablaufdatei nacheinander ab und fügt dem TaskPanel-Objekt die entsprechenden Komponenten hinzu.

Die eingefügten Schaltflächenkomponenten rufen beim Anklicken eine Methode der Klasse UserTestControl auf und übergeben ihr dabei den eigenen Komponentennamen. UserTestControl kümmert sich dann um das Auslesen und Ausführen der Aktionen, die die Schaltfläche auslösen soll.

Alle anderen Komponenten, Textfeld, Likert-Skala und Ausklappliste (DropDownBox), dienen dem Teilnehmer als Eingabemöglichkeiten. Damit diese später ausgelesen werden können, wird jede dieser Komponenten nicht nur in das TaskPanel, sondern auch in eine Liste in UserTestControl eingefügt. Am Ende jeder Aufgabe werden die

Benutzereingaben aller Komponenten dieser Liste ausgelesen und an CurrentState weitergegeben.

Im Fall von eingebundenen Templates werden zunächst die im Template angegebenen Komponenten einzeln erzeugt und dann gegebenenfalls nachträglich verändert. Weitere Informationen zum veränderten und nicht veränderten Einbinden von Templates sind in der Benutzerdokumentation im Anhang A aufgeführt.

4.3.8 TaskPanel

Das TaskPanel-Objekt wird von Task erzeugt. Task fügt diesem Objekt alle Komponenten hinzu, die in der aktuellen Aufgabe der Testbeschreibungsdatei aufgeführt sind. Wenn alle Komponenten angeordnet sind, wird das TaskPanel an UserTestControl übergeben und von dort aus an das TaskWindow geschickt. TaskWindow stellt die Aufgabe dann im Aufgabenfenster dar.

4.3.9 UTButton

UTButton ist die Schaltflächenkomponente, die dem TaskPanel hinzugefügt wird und somit als Teil der Aufgabe angezeigt wird.

In der Testbeschreibungsdatei werden für jede Schaltflächenkomponente Aktionen definiert, die beim Anklicken der Schaltfläche ausgeführt werden sollen. Diese Aktionen werden in Form von Action-Objekten in einer Liste von UTButton gespeichert. Sobald die Schaltfläche angeklickt wird, informiert sie durch einen Methodenaufruf die Klasse UserTestControl, die sich daraufhin die Liste mit den Action-Objekten besorgt und diese nacheinander ausführt.

Neben dem Setzen der auszuführenden Aktionen lässt sich UTButton farblich hervorheben, aktivieren und deaktivieren. Natürlich kann auch die Schaltflächenbeschriftung bestimmt werden.

4.3.10 Action

Action-Objekte dienen zum Speichern einer der Aktionen, die beim Anklicken einer Schaltfläche ausgeführt werden sollen. Die Action-Objekte werden in einer Liste der jeweiligen Schaltfläche gespeichert.

Jedes Action-Objekt besteht aus einem Befehlstext, der die auszuführende Aktion angibt, und aus einem zusätzlichen Parameter, der für den Befehl wichtige Informationen enthält. Mögliche Befehle sind beispielsweise das Aktivieren, Deaktivieren und Hervorheben von anderen Komponenten. In diesen Fällen gibt der Parameter den Namen der Komponente an, für die der Befehl gedacht ist. Der Befehl zum Öffnen einer neuen Webseite im Browser erhält als Parameter den gewünschten URI.

4.3.11 UTTextField

UTTextField stellt ein einzeiliges Textfeld dar, welches dem Teilnehmer erlaubt, gefundene Informationen, eigene Meinungen oder Ähnliches einzugeben. UTTextField wird dem TaskPanel hinzugefügt, sofern es in der Testbeschreibungsdatei angegeben ist.

Jedes in das TaskPanel eingefügte UTTextField wird auch in eine Liste der UserTest-Control eingefügt, um am Ende einer Aufgabe den vom Teilnehmer eingegebenen Text auszulesen und zu speichern.

UTTextField erlaubt das farbliche Hervorheben, die Aktivierung und Deaktivierung, sowie das Setzen einer Überschrift und des Inhalts des Textfeldes.

4.3.12 UTLikertScale

UTLikertScale stellt eine Likert-Skala bereit, die ebenso wie UTTextField in das Task-Panel eingebunden werden kann. Eine Likert-Skala besteht meist aus 5 oder 7 Einteilungen. Der Befragte wählt eine der Einteilungen aus und gibt auf diese Weise eine Wertung zu einer gestellten Frage ab.

Die Anzahl der Unterteilungen, die UTLikertScale darstellen soll, kann in der Testablaufdatei angegeben werden. Wie die anderen Komponenten lässt sich auch

UTLikertScale aktivieren, deaktivieren und farblich hervorheben. Außerdem lassen sich Texte links und rechts von den Einteilungen anzeigen und UTLikertScale kann mit einer Überschrift versehen werden.

So wie UTTextField wird auch UTLikertScale am Ende der Aufgabe ausgelesen. Das Ergebnis ist ein numerischer Wert, der die Nummer der Einteilung wiedergibt. UserTestControl übergibt diese Information an CurrentState, damit der Wert in der Protokolldatei gespeichert wird.

4.3.13 UTDropDownBox

UTDropDownBox stellt dem Teilnehmer eine Ausklappliste, auch ComboBox genannt, zur Verfügung. Damit hat er die Möglichkeit, einen von beliebig vielen vordefinierten Texten auszuwählen.

In der Testbeschreibungsdatei können die Texte, aus denen der Teilnehmer später auswählen soll, angegeben werden. Es lässt sich auch festlegen, welchen dieser Texte der Teilnehmer im eingeklappten Zustand der Komponente sieht, sobald die Aufgabe angezeigt wird. Auch kann für jeden Text ein numerischer Wert angegeben werden, der später etwa für statistische Auswertungen benutzt werden kann.

4.3.14 Alle Komponenten

Eine Gruppe von Befehlen wird von allen Komponenten verstanden, nämlich das Aktivieren, Deaktivieren und Hervorheben durch Farbe. Außerdem kann jeder Komponente der räumliche Abstand vorgegeben werden, den sie zur folgenden Komponente einhalten soll. Diese Befehle wurden in Interfaces gruppiert, die beim Erstellen neuer Komponenten und deren Einbindung in das System helfen.

Alle Eingabekomponenten verfügen über den gleichen Auslesemechanismus. Dieser ist so ausgelegt, dass auch solche Komponenten, die mehr als einen Text bereitstellen, vollständig ausgelesen werden: UserTestControl liest die Komponente so oft aus, wie diese meldet, dass noch Informationen zum Protokollieren vorhanden sind. UTTextField meldet beispielsweise nach dem Auslesen des eingegebenen Textes, dass keine

weiteren Informationen vorhanden sind. Im Gegensatz dazu übergibt UTDropDownBox erst den ausgewählten Text, dann den dazu definierten numerischen Wert. Erst dann meldet diese Komponente, dass keine weiteren Daten vorliegen. Auch diese Funktionen werden durch ein abstraktes Interface vorgegeben, dass von jeder Komponente implementiert werden muss. Auf diese Weise müssen neue Eingabekomponenten der Klasse UserTestControl nicht bekannt gemacht werden. UserTestControl kann die Daten über das gemeinsame Interface auslesen.

Alle Eingabekomponenten können in der Testbeschreibungsdatei so definiert werden, dass sie vom Testteilnehmer ausgefüllt werden müssen. Andernfalls würde beim Verlassen der Aufgabe eine Meldung erscheinen, die den Teilnehmer auf die fehlenden Eingaben hinweist und den Eingabefokus auf diese Komponente setzt. Auf diese Weise ist sichergestellt, dass besonders wichtige Fragen nicht von dem Teilnehmer vergessen werden.

4.3.15 StopWatch

StopWatch-Objekte dienen zur Zeitmessung von komplexen Benutzeraktionen. Damit können beliebig viele Zeitspannen aufgabenübergreifend gemessen werden. Ein aufwendiges errechnen von Zeitspannen mit Hilfe der Zeitstempel aus der Protokolldatei entfällt somit.

Das StopWatch-Objekt wird von UserTestControl immer dann erzeugt, wenn durch die Testbeschreibungsdatei eine neue Stoppuhr gestartet wird. Alle erzeugten Stoppuhren werden in einer Liste von UserTestControl gespeichert.

4.4 Besonderheiten der Implementation

In diesem Abschnitt werden Besonderheiten der Implementation beschrieben. Dazu gehören Änderungen, die am UserTracking von Scone vorgenommen werden mussten, um auch die in Formulare eingetragenen Daten protokollieren zu können (Abschnitt 4.4.1). Abschnitt 4.4.2 stellt die ergriffenen Maßnahmen vor, durch die das Konfigurieren des UserTestTools einfacher gestaltet werden soll. Abschnitt 4.4.3 beschreibt die Möglichkeiten zur Weiterverwendung der Protokolldateien.

4.4.1 Änderungen von Scones Usertracking

Die Eingaben des Teilnehmers in ein Formular auf einer Webseite gehören genauso zu den wichtigen Benutzeraktionen, wie angeklickte Links oder die Browserfunktionen „Vor“ und „Zurück“. Es gibt zwei Möglichkeiten, Formulardaten an den Server zu übertragen: die *Post*- und die *Get*-Methode. Formulardaten, die über die Get-Methode übermittelt werden, lassen sich einfach abfragen, da diese auf besondere Art an den URI angehängt werden und in der Adresszeile des Browsers sichtbar sind. Bei der Post-Methode müssen die Daten aus dem Browseraufruf an den Webserver extrahiert werden.

Zum Zeitpunkt der Entwicklung des UserTestTools wurden die Post-Daten von Scone intern zwar ausgelesen, nicht aber den Plugins zur Verfügung gestellt. Um auch die Plugins auf die Formulardaten zugreifen zu lassen, mussten die Daten durch den WBI in die Seite eingefügt werden, die durch den Webserver zurückgeschickt wird. Das Applet musste modifiziert werden, um die Daten per TCP/IP-Verbindung an Scone zu schicken und die Datenbank musste modifiziert werden, um die Formulardaten aufzunehmen.

4.4.2 Einfache Konfigurierbarkeit

Um das Erstellen von Testablaufdateien einfach zu halten, wurden bewusst wenige Komponenten erzeugt, mit denen die Aufgaben gestaltet werden können. Die fünf Komponenten sind Schaltfläche, Textausgabefeld, Texteingabefeld, Likert-Skala und DropDownBox. Diese Komponenten lassen sich aber flexibel an den jeweiligen Zweck anpassen: Beispielsweise erlaubt das Textausgabefeld die Textstrukturierung durch viele HTML-Tags und die Likert-Skala kann mit einer Überschrift, seitlichen Texten und einer beliebigen Anzahl Unterteilungen dargestellt werden.

Die Definition einer Testbeschreibungsdatei erfolgt im XML-Format. Die einzelnen Komponenten werden durch entsprechende XML-Tags eingebunden und ihre Darstellung wird durch Attribute bestimmt. Die Aktionen, die durch das Anklicken von Schaltflächen ausgelöst werden sollen, werden als Unterelemente des entsprechenden XML-Tags angegeben. Insgesamt gibt es fünf XML-Tags für die Komponenten, 13 Tags für die Aktionen der Schaltflächen, zwei Tags für das Definieren und Einbinden

von Templates und drei Tags für die Einleitung der Testdatei-, Template- und Aufgabenbereiche.

Die Wahl des XML-Formates hat folgende Gründe: XML ist ein einfacher und flexibler Standard (vgl. W3C 2003a). XML-Dateien sind selbstbeschreibend und lassen sich hervorragend strukturieren. Außerdem sind diese Dateien auch von Menschen lesbar.

Das XML-Format hat allerdings auch einen Nachteil: Die Eingabe von XML-Dateien ist fehleranfällig, da falsch geschriebene Tags oder die Missachtung der Groß- und Kleinschreibung dazu führt, dass diese Elemente nicht erkannt werden. Abhilfe schaffen hier spezielle XML-Editoren, die syntaktische und strukturelle Fehler erkennen und anzeigen. Die Erkennung von falsch geschriebenen Tags lässt sich beispielsweise mit Hilfe von XML Schema Definitions erreichen, die die erlaubten Tags, Attribute und Namespaces vorgibt (vgl. W3C 2003b).

4.4.3 Weiterverwendbarkeit der Protokolle

Um die Weiterverwendung der Protokolldateien zu ermöglichen, werden diese in zwei unterschiedlichen Formaten abgelegt: Erstens als XML-Datei und zweitens als TDF-Datei (Tab Delimited File).

Das XML-Format wurde gewählt, um speziellen Auswertungsprogrammen einen strukturierten Zugriff auf die protokollierten Testdaten ohne Informationsverlust zu gewähren. Viele Programmiersprachen bieten die Möglichkeit, auf einfache Weise auf XML-Dateien zuzugreifen. Somit lassen sich mit geringem Programmieraufwand Programme erstellen, die die gesammelten Daten filtern oder statistisch auswerten. Weitere Möglichkeiten für nachfolgende Arbeiten werden im „Ausblick“ (Kapitel sieben) aufgeführt.

TDF (Tab Delimited File), ist ein verbreitetes Format zum Austausch von tabellenartigen Dateien. Dabei werden die einzelnen Daten durch Tabulatoren (Tabstops) getrennt. Das TDF-Format ist eine Variation des CSV-Formats (Comma Separated Values) und wird von vielen Datenbank- und Tabellenkalkulationsprogrammen, wie beispielsweise Microsoft Excel, verstanden und erlaubt so die Weiterverarbeitung der

gesammelten Benutzeraktionen. Im Gegensatz zu der Speicherung im XML-Format musste beim TDF-Protokoll die Struktur entfernt werden, um das Einlesen in Tabellenkalkulationsprogramme zu ermöglichen.

5 Evaluative Entwicklung

Das UserTestTool soll die Durchführung von Benutzertests unterstützen. Natürlich muss auch die Gebrauchstauglichkeit eines solchen Werkzeugs untersucht werden. In diesem Kapitel werden die einzelnen Evaluationsschritte des UserTestTools vorgestellt. Das Ziel dieser Evaluationen war, die Probleme des UserTestTools und entsprechende Lösungen zu finden, um sie in den nächsten Prototypen einbauen zu können. Die gefundenen Probleme werden bewertet und gegebenenfalls ein Lösungsvorschlag und seine Umsetzung angegeben. Obwohl einige Probleme durch mehrere Evaluationen aufgedeckt wurden, wird jedes Problem nur einmal erwähnt.

Abschnitt 5.1 beschreibt die wiederholten Evaluationen des UserTestTools durch zwei Doktoranden des Fachbereichs Informatik. Abschnitt 5.2 behandelt den ersten Einsatz des UserTestTools unter realen Bedingungen beim Test von CommSy. Abschnitt 5.3 führt die Probleme und Anregungen auf, die bei der Evaluation des UserTestTools mit Experten für Gebrauchstauglichkeit entdeckt wurden. Abschnitt 5.4 beschreibt ein kurzes Fazit.

5.1 Partizipativer Entwicklungsprozess

Das UserTestTool wurde in einem zyklischen Entwicklungsprozess in regelmäßigen Abständen mit Hilfe von zwei Doktoranden des Fachbereichs, Hartmut Obendorf und Harald Weinreich, evaluiert. Dabei wurden in heuristischen Evaluationen (vgl. Nielsen 1992) neue und veränderte Teile des UserTestTools untersucht und mögliche Änderungen und Erweiterungen diskutiert. Aufgrund der Erfahrung bei der Entwicklung und Durchführung von Tests mit Benutzern mit einem frühen Prototyp konnten die beiden Doktoranden wertvolle Tipps und Anregungen zum UserTestTool beisteuern. In diesem Abschnitt werden Beispiele zu den vielen gefundenen Optimierungsmöglichkeiten gegeben und entdeckte Fehler beschrieben.

Sehr früh im Design merkten die Doktoranden an, dass eine Likert-Skala für computerisierte Benutzerbefragungen unverzichtbar wäre. Die Likert-Skala bietet dem Benutzer die Möglichkeit, seine Zustimmung zu einer Aussage auszudrücken. Dazu gibt diese

Skala fünf oder sieben Einteilungen vor, von denen der Benutzer eine auswählen kann. Das UserTestTool wurde um die LikertScale-Komponente erweitert. Die Anzahl der Einteilungen ist frei wählbar und es können eine Überschrift und kurze Texte links und rechts der Einteilungen dargestellt werden.

Bei eigenen Studien mit einer ähnlichen Umgebung haben die beiden Doktoranden bemerkt, dass einige Benutzer versehentlich die falsche Schaltfläche betätigten und damit zur nächsten Aufgabe wechselten, statt die aktuelle Aufgabe zu starten. Dieses Problem trat nicht mehr auf, als sie die Schaltflächen für den nächsten logischen Schritt farblich hinterlegt hatten, um sie hervorzuheben. Diese Möglichkeit wurde in das UserTestTool übernommen. Sämtliche Eingabekomponenten lassen sich hervorgehoben erzeugen und auch nachträglich, beispielsweise durch das Anklicken einer Schaltfläche, hervorheben.

Ein weiterer Kritikpunkt für das UserTestTool war die fehlende Möglichkeit, abgebrochene Tests zu einem späteren Zeitpunkt wieder aufnehmen zu können. Diese Möglichkeit ist eine Selbstverständlichkeit bei Benutzertests, wenn sie ohne Computer durchgeführt werden. Da das UserTestTool aber auch das Stellen der einzelnen Aufgaben übernimmt, muss der Abbruch eines Tests, sowie die spätere Aufnahme, explizit vorgesehen werden. Da das UserTestTool die Möglichkeit bietet, Zeitmessungen über mehrere Aufgaben hinweg durchzuführen, wäre es nicht ausreichend, nur die abgebrochene Aufgabe zu wiederholen. Die Messung der Zeitspanne über zwei Aufgaben würde nicht funktionieren, wenn der Test unterbrochen und zu einem späteren Zeitpunkt mit der zweiten Aufgabe fortgesetzt würde. Daher sollte der Tester in der Lage sein, beim wieder Aufnehmen des Tests die Einstiegsaufgabe auswählen zu können. Auf diese Weise könnte mit der ersten Aufgabe angefangen werden, damit die Zeitmessung über die beiden Aufgaben möglich ist. Es wäre allerdings nicht sinnvoll, unbearbeitete Aufgaben zu überspringen. Deshalb stellt das UserTestTool beim Starten eines abgebrochenen Tests nur die Aufgabe, bei der der Test vorher abgebrochen wurde, sowie alle bereits bearbeiteten Aufgaben zur Auswahl.

Auch das Format des Zeitstempels in der TDF-Protokolldatei stellte einen Kritikpunkt dar. Das bisher verwendete Format gibt jeden Zeitpunkt als Anzahl der bisher

verstrichenen Millisekunden seit dem 1.1.1970 an. Dies resultiert in einer unhandlichen 13stelligen Zahl, die beim Importieren in Programme wie Microsoft Excel in der wissenschaftlichen Schreibweise, z. B. 1,0709E+12, angegeben wird. Dabei werden die relevanten Stellen, nämlich der Sekundenbereich, nicht mehr dargestellt. Dieses Problem wurde folgendermaßen gelöst: Die Zeitstempel werden nun als Anzahl der Millisekunden seit dem Start des Tests angegeben. Die Zahlen sind nun kürzer und leichter lesbar, und werden auch in den meisten Tabellenkalkulationsprogrammen korrekt angezeigt. Eine andere Möglichkeit wäre, den Zähler der Millisekunden beim Starten jeder Aufgabe auf Null zu setzten. Dies hätte aber den Nachteil, dass nachträgliche Zeitmessungen über mehrere Aufgaben schwieriger durchzuführen wären. Das Zeitformat in der XML-Version der Protokolldatei wurde bei der Anzahl der verstrichenen Millisekunden seit dem 1.1.1970 belassen. Die XML-Protokolldatei ist zum Importieren in zukünftige Programme gedacht, die die Protokolldaten weiterverarbeiten sollen. Da das verwendete Zeitformat ein Standard in vielen Programmiersprachen ist, ist es für Programmierer einfacher, mit diesem Format umzugehen. Ein neues Zeitformat müsste vom Programmierer erst zurückgewandelt werden, bevor er es in die gewünschte Form bringen könnte.

Eine weitere Anregung war, dem Testentwickler die Möglichkeit zu geben, die Größe des Browserfensters bestimmen zu können. Da die Benutzer in der Praxis viele unterschiedliche Bildschirmauflösungen verwenden – ein Gelegenheitsbenutzer arbeitet im Allgemeinen mit kleineren Auflösungen als ein Grafikdesigner – sollte das Aussehen von Websites auf unterschiedlichen Auflösungen getestet werden. In der Testbeschreibungsdatei des UserTestTools lassen sich die Breite und die Höhe des Browserfensters frei bestimmen, um Websites bei beliebigen Fenstergrößen testen zu können.

Das UserTestTool war schon früh im Entwicklungsprozess in der Lage, das Abschicken von Formularen zu erkennen, da Scone diese Information im Rahmen seines User-Trackings bereitstellt. Die von dem Benutzer eingegebenen Daten wurden von Scone zwar ausgelesen, sie standen den Plugins allerdings nicht zur Verfügung. Deshalb wurde das UserTracking von Scone im Rahmen dieser Arbeit derart modifiziert, dass die Plugins, und somit auch das UserTestTool, auf die Formulardaten zugreifen können.

Damit war es möglich, die Daten von abgeschickten Formularen, die ebenfalls wichtige Benutzeraktionen darstellen, auch in der Protokolldatei abzulegen.

Neben den Anregungen zu speziellen Funktionen gab es noch Vorschläge zum optischen Layout von Komponenten und Dialogen. Daraus ergab sich unter anderem die Möglichkeit, den Abstand zwischen zwei Komponenten auf einen Standardwert von 10 Pixel zu legen, dem Testentwickler aber die Möglichkeit zu geben, den Abstand zwischen je zwei Komponenten frei bestimmen zu können. Dies lässt sich in der Testbeschreibungsdatei durch ein bestimmtes Attribut erreichen, das von jeder sichtbaren Komponente verstanden wird. Dadurch lassen sich Komponentengruppen optisch zusammenfassen oder räumlich trennen.

5.2 Testen der CommSy-Hilfe

Das Community System, kurz CommSy, ist ein webbasiertes System zur Unterstützung von Lerngemeinschaften (vgl. Bleek, Kielas, Malon, Otto und Wolff 2000). Es stellt einen Gemeinschaftsraum zur Verfügung, in dem Präsentationen, Lehr- und Lernmaterialien veröffentlicht werden können. Außerdem lassen sich Projekträume für geschlossene Gruppen einrichten, in denen die einzelnen Teilnehmer dem Rest der Gruppe ihre Arbeitsergebnisse zur Verfügung stellen können.

In einem kleinen Benutzertest mit drei Teilnehmern wurde das UserTestTool eingesetzt, um CommSy und dessen Hilfefunktionen mit neuen Benutzern zu testen.

5.2.1 Ziele des Tests

Der Test sollte ergeben, ob der einzelne Benutzer genug Hilfe vom System erhält, um einfache Aufgaben ohne fremde Unterstützung zu erledigen. Die Teilnehmer hatten bereits von CommSy gehört, es aber bisher noch nicht benutzt. Die Aufgabe der Teilnehmer war, einen Projektraum für ihre Lerngruppe anzulegen, um darin Buchbesprechungen mit anderen Projektteilnehmern zu teilen. Das UserTestTool zeigte den Teilnehmern den Aufgabentext an und öffnete nach einem Klick auf die entsprechende Schaltfläche die Startseite des Community Systems.

In diesem Test wurde das UserTestTool zum ersten Mal unter realen Bedingungen eingesetzt. Daher wurden auch die Probleme der Teilnehmer mit dem UserTestTool und die Möglichkeiten der Weiterverwendung der Protokolldatei betrachtet.

5.2.2 Erkannte Probleme des UserTestTools

In diesem Abschnitt werden die Probleme behandelt, die bei der Benutzung des UserTestTools durch die drei Teilnehmer auftraten. Die Probleme, die die Teilnehmer mit dem Communitiy System hatten, sind für die Evaluation des UserTestTools nicht relevant und werden deshalb an dieser Stelle nicht beachtet.

Das Aufgabenfenster lag in diesem frühen Prototyp des UserTestTools neben dem Webbrowser. Eine derartige Anordnung ist bei Rechnern mit zwei Monitoren oder großen Bildschirmauflösungen kein Problem. Da der Test aber auf einem Laptop mit einer Bildschirmauflösung von 800x600 Pixel durchgeführt wurde, war jedes der Fenster nur 400 Pixel breit. Alle Testbenutzer mussten den Fensterinhalt des Browsers horizontal hin- und herschieben oder das Fenster des Browsers vergrößern, um die Webseite vollständig betrachten zu können. Ein von vornherein vergrößertes Browserfenster wäre keine gute Lösung gewesen, da dieses das Aufgabenfenster verdeckt hätte. Daher wurde das UserTestTool derart erweitert, dass das Browserfenster nach dem Start des UserTestTools in den Hintergrund geschickt wurde. Erst wenn durch das Anklicken einer Schaltfläche eine Webseite in den Browser geladen werden sollte, wurde das Browserfenster in den Vordergrund geholt. Durch diese Funktionserweiterung lassen sich derartige Tests nun auch auf Rechnern mit kleinen Bildschirmen durchführen.

Für den Evaluator ergab sich das Problem, dass sich die Testprotokolle nicht zur weiteren Auswertung in die Tabellenkalkulation Excel importieren ließen. Das lag daran, dass der Prototyp des UserTestTool zu diesem Zeitpunkt lediglich über eine Ausgabe ins XML-Format verfügte. Um beispielsweise die Zeitpunkte der Aktionen der Teilnehmer mit Microsoft Excel auswerten zu können, mussten die relevanten Aktionen von Hand aus der XML-Datei in die Excel-Tabelle übertragen werden. Der Evaluator äußerte den Wunsch nach einer Schnittstelle zu Microsoft Excel, da sich die Auswertung in kurzer Zeit mit Excel erledigen ließe. Deshalb gibt das UserTestTool die Protokolldaten inzwischen auch im TDF-Format (Tab Delimited File) aus. Dabei werden die

einzelnen Daten eines Datensatzes durch Tabulatoren getrennt. Viele tabellenorientierte Applikationen, wie Microsoft Excel, können das TDF-Format importieren.

5.3 Evaluation durch Experten

Das UserTestTool sollte durch Experten für Gebrauchstauglichkeit evaluiert werden. Um es umfassend zu testen, sollten drei unterschiedliche Tests aus verschiedenen Blickwinkeln durchgeführt werden: Erstens sollte das UserTestTool aus der Sicht des Teilnehmers evaluiert werden, der in der Praxis die Aufgaben des Tests abarbeiten und dazu mit dem UserTestTool interagieren muss. Zweitens sollte das Werkzeug aus der Sicht des Testentwicklers untersucht werden, der das Aussehen der einzelnen Aufgaben und den Ablauf des Tests in der XML-Testablaufdatei definiert. Drittens sollte das UserTestTool aus der Sicht des Evaluators betrachtet werden, der die resultierenden Protokolldateien deuten muss.

Es war problematisch, geeignete Teilnehmer für die Evaluationen zu finden, da sie für die Tests spezielles Vorwissen benötigten: Sie sollten Erfahrung in der Entwicklung von Software und Websites haben sowie mit dem Testen auf Gebrauchstauglichkeit vertraut sein. Aus dem Bereich der Universität haben sich drei Diplomanden mit entsprechender Expertise bereiterklärt, das UserTestTool zu evaluieren. Alle haben bereits Software und Websites entwickelt und zwei von ihnen haben bereits selbst Gebrauchstauglichkeitstests durchgeführt. Um weitere Evaluatoren zu finden, wurde das UserTestTool, zusammen mit einer Benutzerdokumentation (siehe Anhang A) an die Teilnehmer einer deutschen Newsgroup für die Gebrauchstauglichkeit von Websites geschickt. Die Teilnehmer wurden gebeten, sich an der Evaluation zu beteiligen. Allerdings erklärte sich keiner der Teilnehmer dazu bereit. Auch auf das direkte Anschreiben von zwei Teilnehmern erfolgte keine Reaktion.

Die drei Evaluationen wurden separat mit jedem der drei Diplomanden durchgeführt. Bei einer derartig geringen Teilnehmeranzahl lassen sich die gefundenen Ergebnisse zwar nicht statistisch auswerten, trotzdem sind auch die Probleme, die nur von einem Teilnehmer beobachtet werden, wichtige Hinweise auf Probleme, die auch im praktischen Einsatz auftauchen werden.

Die Evaluation aus Sicht des Teilnehmers eines Website-Tests wurde in Form eines Walkthroughs (vgl. Rieman, Franzke, Redmiles 1995) durchgeführt. Die Diplomanden versetzten sich in die Lage eines potentiellen Teilnehmers und arbeiteten die Aufgaben des Beispieltests ab, der dem UserTestTool beiliegt. Die gefundenen Probleme und Anregungen werden im Abschnitt 5.3.1 beschrieben. Um die Probleme aus der Sicht des Testentwicklers zu entdecken, wurden die Teilnehmer gebeten, den Beispieltest in einem Thinking-aloud-Test zu modifizieren und das Ergebnis zu überprüfen. Die auf diese Weise entdeckten Probleme gibt Abschnitt 5.3.2 wieder. Durch einen Thinking-aloud-Test sollten die Probleme aus der Sicht des Evaluators beim Deuten der Protokolldatei gefunden werden. Diese werden im Abschnitt 5.3.3 beschrieben.

5.3.1 Probleme aus der Sicht des Teilnehmers

Die meisten Probleme und Anregungen, die von den Diplomanden geäußert wurden, beziehen sich auf das UserTestTool aus der Sicht des Teilnehmers. Die Spanne der Anmerkungen reicht von umfangreicheren Hinweisen an den Benutzer bis hin zu zusätzlicher Funktionalität.

Zwei Diplomanden bemerkten, dass die Testperson von der Möglichkeit des Test-abbruchs informiert werden sollte. Der Teilnehmer würden nicht von selbst auf die Idee kommen, einfach das Aufgabenfenster zu schließen, da dies im schlimmsten Fall mit dem Verlust der bisher eingegebenen Daten verbunden sein könnte. Da das UserTest-Tool aber entworfen wurde, um einen Tester bei der Testdurchführung zu unterstützen – nicht um ihn zu ersetzen – könnte dieser den Teilnehmer bei Bedarf von der Abbruch-möglichkeit informieren.

Zwei Diplomanden bemängelten die Darstellung der Schaltflächen in dem Aufgaben-fenster. Sie seien unscheinbar und ließen sich leicht mit der DropDownBox ver-wechseln. Das Problem liegt bei Java, das für das „Look-and-Feel“ der sichtbaren Komponenten verantwortlich ist. Dieses ließe sich vom UserTestTool einstellen, bei-spielsweise auf das Windows-Look-And-Feel. Diese Einstellung würde dann aber für alle von Scone erzeugten Fenster gelten, auch für die Fenster von anderen Plugins und auch dann, wenn Scone unter anderen Betriebssystemen läuft. Daher wurde das Look-and-Feel neutral gehalten.

Ein anderer Kritikpunkt betraf die gelbe Farbe von hervorgehobenen Schaltflächen. Der Gelbton könnte als Warnung aufgefasst werden. Grün wäre als Farbe zum Hervorheben besser geeignet. Diese Anregung wurde aufgegriffen und der Farbton entsprechend geändert. Außerdem lässt sich der Farbton für alle Komponenten nun zentral in einer Klasse definieren.

Eine weitere Idee war, dem Testteilnehmer anzuzeigen, ob das UserTestTool gerade die vom Teilnehmer benötigte Zeit misst. Eine derartige Anzeige könnte den Benutzer allerdings unter Druck setzen, was zu unnötigen Fehler führen könnte. Daher wurde diese Möglichkeit nicht in die Implementation des UserTestTools aufgenommen.

Ein Student regte an, dem Teilnehmer das Hin- und Herspringen zwischen den einzelnen Aufgaben zu erlauben. Diese Möglichkeit ist nach Meinung des Verfassers aber nur dann sinnvoll, wenn es sich um Aufgaben handelt, die ähnlich wie Fragebögen aufgebaut sind. Bestimmte Fragen könnten erst einmal ausgelassen und später beantwortet werden. Bei Aufgaben, in denen der Teilnehmer beispielsweise möglichst schnell Informationen innerhalb einer Website suchen soll, ist es nicht sinnvoll, die angefangene Suche zu unterbrechen. Außerdem gäbe es Probleme beim Messen der benötigten Zeit, wenn der Benutzer zwischendurch nach anderen Informationen sucht.

5.3.2 Probleme bei der Modifikation eines Tests

Das Problem, das von allen Diplomanden gemeldet wurde, ist die Fehleranfälligkeit bei der Modifikation der XML-Testbeschreibungsdatei. Die Ursache liegt darin, dass falschgeschriebene Knoten- oder Attributnamen nicht vom UserTestTool erkannt und damit auch nicht bearbeitet werden. Das gilt auch für fehlerhafte Groß- und Kleinschreibung. Des Weiteren muss jedes XML-Dokument wohlgeformt sein, d. h. jeder XML-Knoten muss abgeschlossen und korrekt geschachtelt sein, sonst erzeugt der Parser, der das Dokument einliest, eine Fehlermeldung. Dies sind alles mögliche Fehlerquellen die die Eingabe oder Modifikation der Testbeschreibungsdateien erschweren. Lösungen für diese Probleme wären ein *GUI-Builder*, zum einfachen Zusammenstellen der einzelnen Aufgaben, oder eine XML Schema Definition (vgl. W3C 2003b), mit der die Korrektheit der XML-Datei überprüft werden kann. Ein GUI-Builder hat den Nachteil, dass er die Erweiterbarkeit des UserTestTools einschränkt. Wird dem UserTestTool

beispielsweise eine neue Eingabekomponente hinzugefügt, müssten die Änderungen auch am GUI-Builder vorgenommen werden. Da es sich bei einem *GUI-Builder* aber um ein komplexes Programm handelt, das in diesem Fall auch die Definition der Funktionalität der Schaltflächen erlauben müsste, sind derartige Änderungen aufwändig.

Der zweite Lösungsansatz ist eine XML Schema Definition. Sie beschreibt die erlaubten XML-Knoten, ihre Kindknoten, ihre Attribute und deren Werte. In Verbindung mit einem professionellen XML-Editor lassen sich damit Schreibfehler in den Knoten- oder Attributnamen erkennen und auf solche hinweisen. Viele XML-Editoren, wie der kommerzielle „Oxygen XML-Editor“ (vgl. SyncRo Soft 2003), unterstützen den Benutzer zusätzlich durch Vorgabe der nächsten möglichen Knoten- und Attributnamen, sowie die Überprüfung der XML-Syntax. Das UserTestTool-Projekt wurde um eine XML Schema Definition ergänzt, die bei Verwendung mit einem entsprechenden XML-Editor die Erstellung von Testbeschreibungsdateien unterstützt. Die XML Schema Definition lässt sich vergleichsweise einfach und schnell an zukünftige Komponenten anpassen.

Zwei Diplomanden wünschten sich alternative Ablaufpfade, abhängig von den Eingaben des Benutzers, um bestimmte Aufgaben oder Fragenblöcke zu einem bestimmten Thema bei Bedarf überspringen zu können. Dies ist hauptsächlich für Befragungen sinnvoll. Auf diese Weise ließen sich Fragen zu den Erfahrungen des Teilnehmers mit speziellen Programmen überspringen, wenn der Teilnehmer angibt, diese Programme bisher nicht benutzt zu haben. Eine derartige Erweiterung wäre für bestimmte Aufgaben sehr sinnvoll, die im Vorfeld betrachteten Anwendungsfälle des UserTestTools benötigen diese allerdings nicht. Da diese Erweiterung außerdem eine relativ umfangreiche Maßnahme wäre, wird sie in dem Ausblick im siebten Kapitel erwähnt.

Eine andere Möglichkeit alternativer Ablaufpfade wäre, bei jedem Test die Reihenfolge der definierten Aufgaben zu verändern. Auf diese Weise ließen sich beispielsweise zwei ähnliche Versionen einer Websites mit den gleichen Teilnehmern testen, ohne das die Gewöhnung der Teilnehmer an die grobe Funktionsweise der Websites die Ergebnisse verfälschen würde. Durch das regelmäßige Variieren der Reihenfolge der Websites ließen sich derartige Effekte minimieren. Um einen derartigen Test mit dem UserTestTool

durchzuführen, müssten zwei Testbeschreibungsdateien erstellt werden, die sich lediglich durch die Reihenfolge der Aufgaben unterscheiden. Falls mehrere Variationen benötigt werden, müssen entsprechend viele Testbeschreibungsdateien erstellt werden. Eine entsprechende Erweiterung des UserTestTools wäre sehr umfangreich und würde nur für spezielle Zwecke benötigt. Deshalb wird dieser Ansatz im Ausblick (Kapitel 7) skizziert.

5.3.3 Probleme bei der Analyse der Protokolle

Bei der Deutung der Protokolldatei wurde unter anderem folgender Kritikpunkt geäußert: Die Texteingaben, Wertungen und Auswahlen des Teilnehmers wurden bisher in dem Attribut „action“ des Komponentenknotens abgelegt. Dieser Attributname wurde aber als irreführend empfunden. Daher wurde das Attribut zu „value“ umbenannt, da z. B. im Fall des Textfeldes der Eintrag **`<textFieldText value=...>`** geläufiger ist und einheitlich auch für die übrigen Eingabekomponenten passt.

Bei der Wiederaufnahme eines vorher abgebrochenen Tests kann die Nummer der Aufgabe angegeben werden, mit der der Test wieder aufgenommen werden soll. Das Angeben der Aufgabennummer ist allerdings problematisch, da sich die Nummern der eigentlichen Aufgaben verschieben, sobald Einleitungen oder Einführungsaufgaben vor den eigentlichen Aufgaben definiert werden. Das UserTestTool zählt nämlich lediglich die definierten „task“-Bereiche und unterscheidet nicht zwischen Einleitungen und Aufgaben. Wird also die erste Aufgabe nach zwei Einleitungsfenstern definiert, dann muss diese mit Aufgabe 3 aufgerufen werden, was zwei Diplomanden negativ aufgefallen war. Eine Lösung dieses Problems ist, die gewünschte Startaufgabe anhand des Aufgabennamens auszuwählen, der bei der Definition jeder Aufgabe in der Testbeschreibungsdatei angegeben werden muss. Das UserTestTool bietet dem Tester nun eine DropDownBox an, in der die Namen der bereits bearbeiteten Aufgaben und die abgebrochene Aufgabe aufgeführt sind und ausgewählt werden können.

5.4 Fazit

Besonders die frühen Evaluationen haben einige Fehler aufgedeckt, die in den nächsten Prototypen beseitigt werden konnten. Außerdem haben sie viele Anregungen erbracht, die auch größtenteils in die Implementation eingeflossen sind.

Insbesondere der Test durch die Diplomanden, die sich zwar mit der Materie auskannten, für die aber das UserTestTool neu war, erbrachte wertvolle Hinweise auf ergänzende Funktionen. So lässt sich nun beispielsweise die Startaufgabe bei der Wiederaufnahme eines abgebrochenen Tests anhand des Aufgabennamens auswählen, statt mit der nicht eindeutigen Aufgabennummer.

Auch von der Erfahrung der beiden Doktoranden des Fachbereichs hat das UserTestTool sehr profitiert: Beispielsweise ist die Benutzung des UserTestTools für den Testteilnehmer einfacher geworden, seit sich die Schaltflächen für den nächsten logischen Schritt hervorheben lassen.

Der frühe Einsatz des UserTestTools beim CommSy-Test hat ein paar grundlegende Probleme des derzeitigen Prototyps aufgezeigt, z. B. die Wichtigkeit einer Schnittstelle zur Weiterverarbeitung der Protokolldaten.

6 Zusammenfassung

Tests mit Benutzern eignen sich besonders, um Probleme, die den Benutzer bei der Arbeit mit einem System behindern, zu entdecken. Allerdings ist die Durchführung dieser Tests auch sehr aufwendig, da die Äußerungen des Teilnehmers analysiert, seine Aktionen, seine Eingaben und die besuchten Webseiten protokolliert und später ausgewertet werden müssen.

Bei einem klassischen Benutzertest werden die Aktionen und Aussagen des Teilnehmers protokolliert oder auf Video oder Tonband festgehalten, damit die Probleme des Teilnehmers nachträglich gedeutet und alle benötigten Aktionen protokolliert werden können. Besonders das nachträgliche Auswerten benötigt ein Vielfaches der Zeit, die für die eigentlichen Testläufe mit den Testteilnehmern aufgewendet werden muss. Dieser hohe Aufwand lässt viele Entwickler vor dem Einsatz dieser Testmethode zurückschrecken.

Das Ziel dieser Arbeit war, den hohen Arbeitsaufwand des klassischen Website-Tests mit Benutzern zu verringern, indem Routineaufgaben von einem Computer erledigt werden. Zu diesen Routineaufgaben gehört das Protokollieren sämtlicher Benutzeraktionen, die der Teilnehmer mit dem Browser ausführt. Darüber hinaus lassen sich auch die Aufgaben durch den Computer stellen, sowie die Lösungen der Aufgaben eingeben und speichern. Eine derartige kontrollierte Testumgebung verringert beispielsweise das Risiko verfälschter Daten durch unbewusste Beeinflussung der Teilnehmer. Dadurch lassen sich die Ergebnisse der einzelnen Testläufe besser miteinander vergleichen. Auch Benutzerbefragungen, die gern mit Benutzertests kombiniert werden, lassen sich per Computer durchführen. Die resultierenden Antworten können ebenfalls gespeichert werden. Der Browser des Testteilnehmers kann gesteuert werden, um bei Bedarf bestimmte Webseiten anzuzeigen. Es gibt noch viele andere Aufgaben, z. B. automatisierte Zeitmessungen, die beim Testen von Websites vom Computer übernommen werden können.

Für viele Studien zum Verhalten des Benutzers im Web wurden die Benutzeraktionen mit Hilfe von Computerunterstützung gesammelt. Es gibt auch einige kommerzielle

Systeme, die das Testen von Benutzern automatisieren sollen. All diese Ansätze lieferten aber entweder nur sehr ungenaue Benutzeraktionen oder waren auf wenig benutzte Browser beschränkt. Dem Verfasser dieser Arbeit ist kein System bekannt, dass neben dem Erkennen und Speichern vieler Benutzeraktionen die Aufgaben stellt, den Browser steuert und Fragebögen anbietet.

Das in dieser Arbeit entwickelte UserTestTool kann dem Teilnehmer die Aufgaben stellen und speichert seine Aktionen und Eingaben. Ebenso lassen sich Benutzerbefragungen als Teil des Tests durchführen. Das UserTestTool kann den Internet Explorer steuern und erlaubt einen Großteil der Aktionen des Teilnehmers mit dem Webbrowser abzufangen und zu speichern (siehe Abbildung 6.1).

Abbildung 6.1: Das UserTestTool stellt die Aufgabe und steuert den Internet Explorer

Durch die Entlastung des Testleiters kann sich dieser nun auf den Test konzentrieren. Er hat die Möglichkeit den Teilnehmer zu beobachten und Notizen zu auffälligem Verhalten zu machen. Neben dem Testleiter wird auch der Teilnehmer entlastet: Durch die integrierte Browsersteuerung lassen sich die benötigten Webseiten automatisch im

Browser aufrufen. Dadurch lassen sich Fehler des Teilnehmers beim Aufrufen entsprechender Seiten verhindern.

Durch das Framework Scone, in welches das UserTestTool integriert wurde, lassen sich angeforderte Webseiten verändern oder ergänzen, ohne sie vorher modifizieren und lokal speichern zu müssen. Außerdem lässt sich die Navigation auf bestimmte Webseiten einschränken, um etwa den Zugriff auf Suchmaschinen zu unterbinden. Plugins von Scone, die beispielsweise bei der Orientierung im Web oder bei der Navigation unterstützen sollen, lassen sich mit dem UserTestTool ebenso einfach testen, wie komplette Websites.

Der Ablauf eines Tests wird durch eine XML-Datei definiert. Es wurden bewusst nur elementare Ein- und Ausgabekomponenten vorgesehen, um die Definition eines Tests möglichst einfach zu gestalten. Die einzelnen Komponenten lassen sich aber durch einheitliche Attribute in Aussehen und Anordnung verändern. Bei Bedarf lässt sich das UserTestTool um zusätzliche Komponenten erweitern. Um die Fehleranfälligkeit der Eingabe von XML-Code zu minimieren, wurde eine XML Schema Definition erstellt, die es XML-Editoren erlaubt, die Korrektheit der verwendeten Knoten und Attribute zu überprüfen. Einige XML-Editoren, wie z. B. Oxygen XML Editor (vgl. SyncRo Soft 2003), geben dem Benutzer sogar die nächsten möglichen Knoten und Attribute vor (siehe Abbildung 6.2).

Alle Aktionen und Eingaben des Teilnehmers stehen dem Testleiter in zwei Ausgabeformaten zur Verfügung, die sich einerseits gut von zukünftigen Auswertungswerkzeugen, andererseits einfach durch viele Tabellenkalkulationsprogramme weiterverarbeiten lassen (siehe Abbildung 6.3).

In einer Evaluation wurde das UserTestTool auf seine eigene Gebrauchstauglichkeit hin getestet. Durch die Mithilfe von Diplomanden und Doktoranden wurden einige Fehler gefunden und viele Anregungen gesammelt, die nach ihrer Prüfung größtenteils in die Implementation eingeflossen sind.

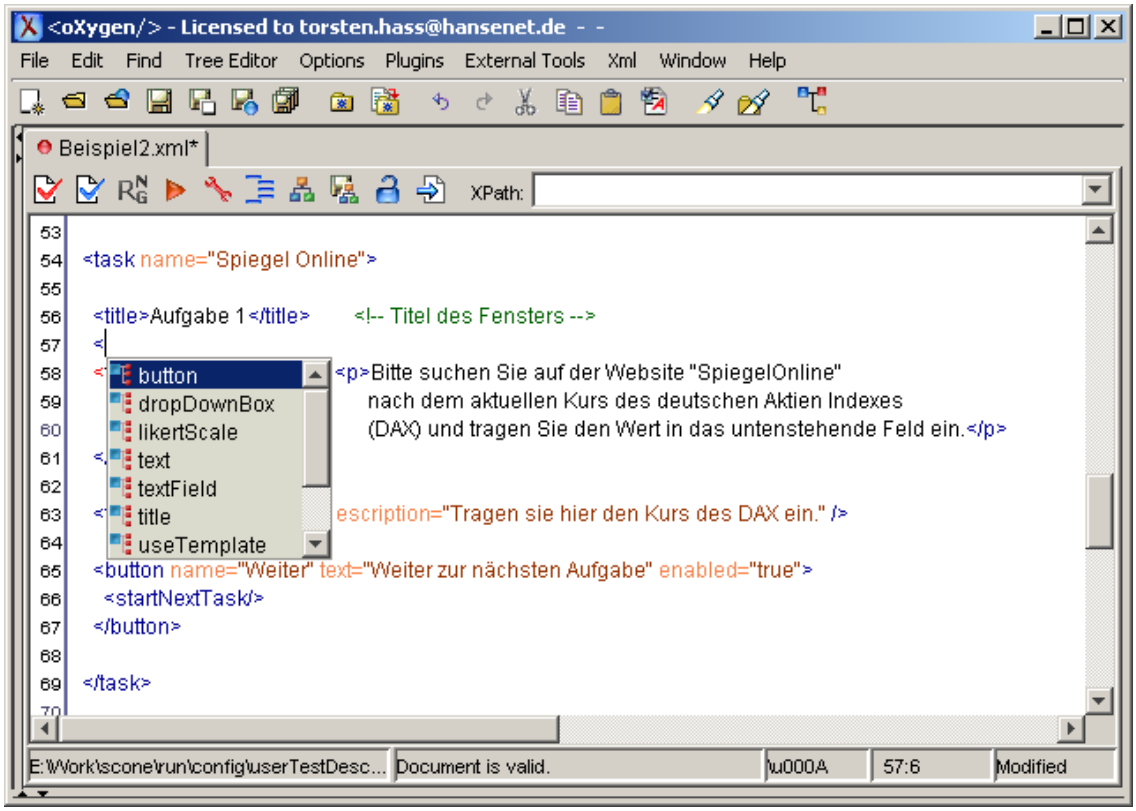


Abbildung 6.2: Oxygen XML Editor schlägt mögliche XML-Tags vor

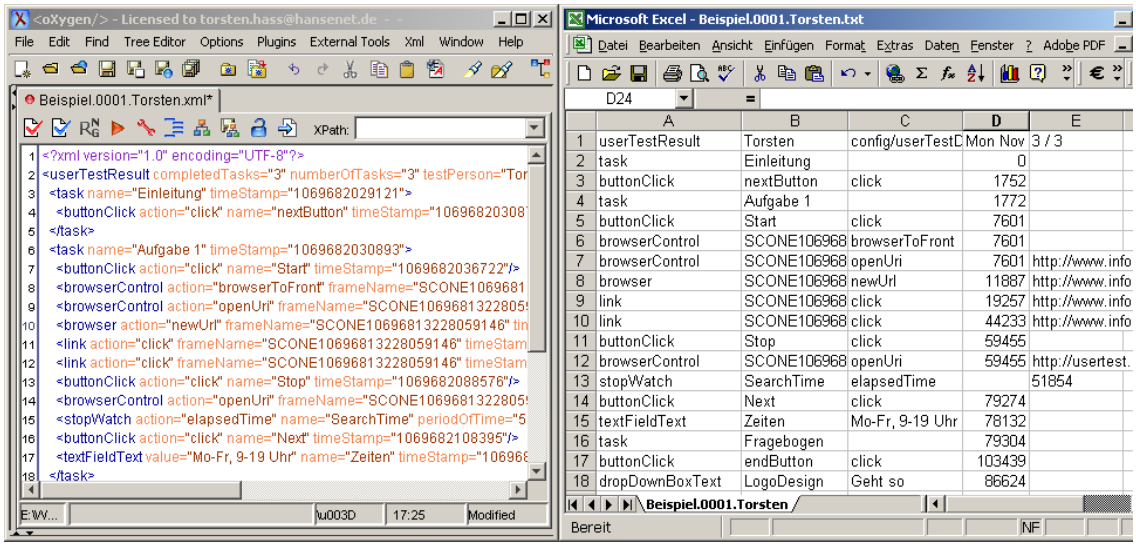


Abbildung 6.3: Die Protokolldateien im XML- und TDF-Format

Das UserTestTool hat aber auch Grenzen: Es wurde entwickelt, um den Testleiter bei der Durchführung von Website-Tests mit Benutzern zu unterstützen, indem ihm viele Routineaufgaben abgenommen werden. Obwohl das UserTestTool in der Lage ist, dem Benutzer die Aufgaben zu stellen und seine Antworten entgegenzunehmen, kann es den Testleiter nicht ersetzen. Der Testleiter sollte weiterhin beim Test anwesend sein, um dem Teilnehmer beispielsweise bei Problemen zu helfen, die dieser allein nicht überwinden könnte.

Die Deutung der Probleme muss ebenfalls weiterhin vom Testleiter vorgenommen werden. Mögliche Gefahren sind dabei die Überinterpretation der gesammelten Daten oder das Übersehen relevanter Daten bei zu großen Datenmengen. Daher ist es unerlässlich, vor dem Test festzulegen, welche Aspekte des Interfaces in der Auswertung untersucht werden sollen.

Für die meisten Zwecke sollten die erfassten Benutzeraktionen ausreichen. Bestimmte Fälle fordern aber genauere Daten über Interaktion des Teilnehmers mit der Website. In diesem Fall sollte zusätzlich eine Audio- oder Videoaufzeichnung durchgeführt werden, um zusätzliche Daten für die Auswertung zur Verfügung zu haben.

Für reine Benutzerbefragungen sind webbasierte Fragebögen häufig weniger aufwändig und damit dem UserTestTool vorzuziehen.

Das siebte Kapitel skizziert mögliche Erweiterungen, die die Grenzen des UserTest-Tools erweitern könnten.

7 Ausblick

Das UserTestTool ist in den vorherigen Kapiteln ausführlich beschrieben worden. Es unterstützt den Tester bei der Durchführung von Website-Tests mit Benutzern, indem es die Benutzeraktionen protokolliert, den Web-Browser steuert und dem Teilnehmer die Aufgaben stellt. Eigene Tests lassen sich durch eine XML-Testablaufdatei definieren. Trotzdem sind Erweiterungen denkbar, die dem Testleiter die Arbeit weiter erleichtern könnten. Abschnitt 7.1 gibt ein Beispiel an, das die Erstellung der Testaufgaben derart vereinfachen würde, dass auch technisch weniger versierte Testentwickler Tests erstellen könnten. Abschnitt 7.2 beschreibt Ansätze, mit denen sich die Auswertung der Protokolldateien vereinfachen ließe. Abschnitt 7.3 skizziert Möglichkeiten, den Umfang der gesammelten Daten zu erweitern. Abschließend beschreibt Abschnitt 7.4 Möglichkeiten, die Teststeuerung derart zu erweitern, dass die Reihenfolge der Aufgaben kontrolliert verändert oder nichtbenötigte Aufgaben ausgelassen werden können.

7.1 Unterstützung des Testentwurfs

Die Eingabe der XML-Testablaufdatei ist einfacher geworden, seit Schreibfehler und Fehler in der XML-Struktur von XML-Editoren mit XML Schema Definition erkannt werden. Trotzdem muss der Testentwickler wissen, welche Komponenten zur Verfügung stehen, mit welchen XML-Knoten sie eingebunden werden und mit welchen Attributen sie sich konfigurieren lassen. Diese Informationen lassen sich in der Komponentenreferenz der Benutzerdokumentation des UserTestTools finden – aber wer liest schon gerne Benutzerdokumentationen? Um die Auswirkungen der Änderungen sehen zu können, muss der modifizierte Test gestartet und bis zu der geänderten Aufgabe durchgeklickt werden. Dies kann besonders bei umfangreichen Tests recht aufwendig werden.

Eine Lösung wäre ein „What You See Is What You Get“-Editor, in dem der Testentwickler die gewünschten Komponenten per Mausklick auswählt. Der Entwickler könnte jederzeit das Aussehen der Aufgabe überprüfen und gegebenenfalls Änderungen in dem Dialog der entsprechenden Komponente vornehmen. Die Testbeschreibungs-

datei würde vom Editor erstellt werden, ohne dass der Entwickler die XML-Datei auch nur ansehen müsste.

Ein solcher Aufgabeneditor wäre allerdings eine komplexe Anwendung, da er nicht nur einen *GUI-Builder* darstellen müsste, sondern auch die Definition der Ablaufsteuerung, wie die einzelnen Aktionen der Schaltflächen, zulassen müsste.

Eines der Ziele dieser Arbeit war, das UserTestTool so zu entwickeln, das es sich leicht erweitern lässt. Diese Erweiterbarkeit würde durch einen Aufgabeneditor eingeschränkt werden, da jede Änderung an den Komponenten des UserTestTools auch an den Komponenten des Aufgabeneditors vorgenommen werden müsste. Wegen der hohen Komplexität des Aufgabeneditors wären derartige Änderungen entsprechend aufwändig.

Um diesen Nachteil zu umgehen, müsste der Editor auf die selben Klassen zugreifen, die das UserTestTool zum Erzeugen und Anzeigen der Aufgaben benutzt. Auf Grund des modularisierten Aufbaus des UserTestTools wäre dies nach einigen Änderungen möglich. Dazu müssten allerdings die einzelnen Komponenten sowie die Browsersteuerung und andere Funktionen selbstbeschreibend sein, damit sie von dem UserTestTool und dem Aufgabeneditor bei Bedarf eingebunden werden könnten. Diese Selbstbeschreibungsfähigkeit würde die Komplexität der einzelnen Komponenten erhöhen und möglicherweise die Erweiterbarkeit des UserTestTools erschweren.

7.2 Unterstützung der Auswertung

Bisher werden die gesammelten Aktionen des Teilnehmers, seine Eingaben und die gemessenen Zeitintervalle sowohl im XML-Format, als auch im TDF-Format (Tag Delimited File) gespeichert. Das Protokoll im TDF-Format wurde strukturell vereinfacht, damit es sich problemlos in Tabellenkalkulationsprogramme wie Microsoft Excel importieren und dort weiter verarbeiten lässt. Die Möglichkeiten solcher Programme sind aber begrenzt, da sie nicht explizit für die Auswertung entsprechender Dateien ausgelegt sind. Daher bietet es sich an, eine Anwendung zu entwickeln, die den Testleiter bei der Auswertung der gesammelten Daten unterstützt. Das Protokoll im XML-Format enthält alle erfassten Daten und könnte einer solchen Anwendung als Datenquelle dienen.

Eine Möglichkeit wäre das Filtern der aktuell relevanten Daten. Soll in einer Studie beispielsweise nur festgehalten werden, wohin der Benutzer springt, wenn er die Zurück-Taste des Browsers betätigt, könnten alle anderen Aktionen zeitweise ausgeblendet werden. Außerdem könnten bestimmte Ereignisse zur besseren Übersicht in bestimmten Farben dargestellt werden. Dadurch ließen sich die einzelnen Einträge besser unterscheiden.

Die Zeitstempel jeder Aktion könnten in einem beliebigen Format angezeigt werden. Neben der momentan benutzten Anzeige in Millisekunden wären auch Sekunden oder Minuten und Sekunden denkbar. Die Zeitstempel könnten die Zeitspanne seit dem Beginn des Tests, seit dem Beginn der Aufgabe oder seit einem beliebigen Ereignis darstellen. Für bestimmte Zwecke ist möglicherweise auch die absolute Zeit mit oder ohne Datum interessant, um etwa die Tageszeit oder den Wochentag des Tests zu berücksichtigen.

Es wäre auch sinnvoll, gemessene Zeiten oder andere quantitative Daten mit beliebigen anderen Testläufen zu vergleichen. Auch testweite Mittelwerte oder andere statistische Auswertungen über abgegebene Wertungen sollten möglich sein. Die einzelnen Werte der Testläufe sollten sich grafisch darstellen lassen.

Weitere mögliche Erweiterungen sind nachträglich berechenbare Zeitspannen zwischen beliebigen Ereignissen und das Auszählen von Testläufen, in denen die Teilnehmer auf der Suche nach bestimmten Informationen von einer vorgegebenen Route durch die Website abgekommen sind.

7.3 Erweiterung der Protokollierung

Bisher erfasst das UserTestTool einen Großteil der Aktionen und Eingaben des Teilnehmers beim Interagieren mit dem Aufgabenfenster und dem Webbrowser. Diese Daten reichen möglicherweise für spezielle Untersuchungen nicht aus. Dieser Nachteil ließe sich durch Erweiterungen der Protokollierungsmöglichkeiten beheben.

Das UserTestTool könnte beispielsweise derart erweitert werden, dass die Aussagen des Teilnehmers aufgezeichnet würden. Die Aussagen könnten für jede Aufgabe separat

aufgezeichnet werden, um die Zuordnung der einzelnen Aussagen zu den Aufgaben zu erleichtern. Synchron zu der Audio-Aufnahme ließen sich die Webseiten anzeigen, die der Teilnehmer während des Testlaufes im Browser betrachtete. Zeitlich synchron könnten die Links aufblinken, die der Benutzer während des Tests angeklickt hat. Eine derartige Visualisierung der gesammelten Daten käme einer Videoaufzeichnung des Bildschirms des Teilnehmers bereits recht nahe.

Eine weitere Möglichkeit wären regelmäßige softwaremäßige Bildschirmfotografien von dem Bildschirm des Teilnehmers. Diese ließen sich synchron zu den aufgezeichneten Aussagen des Teilnehmers abspielen, um ein videoähnliches Protokoll zu erzeugen.

Um den Rechner, an dem der Teilnehmer sitzt, nicht zusätzlich zu belasten, ließen sich Audio- oder Videoaufzeichnungen auch von eigenständigen Videokameras übernehmen. Diese könnten beispielsweise über zusätzliche Hardware vom UserTestTool ein- und ausgeschaltet werden. Bei der Verwendung einer Videokamera sind auch viele Zwischenlösungen möglich. So ließen sich beispielsweise, eine große Festplatte vorausgesetzt, die digitalen Videosignale einer digitalen Kamera direkt auf der Festplatte speichern. Das UserTestTool könnte die Aufnahme steuern und für jede Aufgabe eine eigene Videodatei erstellen. Der Testleiter könnte dann die Videoaufzeichnung nachträglich am Computer betrachten, ohne zusätzliche Hardware wie Videorekorder oder Fernseher zu benötigen.

7.4 Erweiterung der Teststeuerung

Bisher werden Tests statisch in der Testablaufdatei definiert und bei der Ausführung in der vorgegebenen Reihenfolge abgearbeitet. Als Erweiterung wäre auch ein nicht statischer Ablauf denkbar. Für spezielle Zwecke wäre es beispielsweise sinnvoll, Aufgaben oder Fragebögen zu überspringen, wenn dem Teilnehmer das erforderliche Fachwissen fehlt. Dies würde etwa erlauben, Fragebögen über die Entwicklung von Websites auszulassen, wenn der Benutzer angibt, keine Erfahrung in der Website-Entwicklung zu haben.

Eine andere Möglichkeit wäre, die Reihenfolge der definierten Aufgaben bei jedem Test zu verändern. Auf diese Weise würden sich beispielsweise zwei ähnliche Websites von

den gleichen Teilnehmern testen lassen. Um den Effekt auszuschließen, dass die Teilnehmer immer die zweite Website besser bedienen können, weil sie die Bedienung bei den Aufgaben für die erste Website geübt haben, müsste die Hälfte der Teilnehmer die Websites in umgedrehter Reihenfolge testen. Um einen derartigen Test mit dem aktuellen UserTestTool durchzuführen, müssten zwei eigenständige Testbeschreibungsdateien erstellt werden, die sich lediglich durch die Reihenfolge der Aufgaben unterscheiden. Wenn mehrere Variationen getestet werden sollen, müssen entsprechend viele Testbeschreibungsdateien definiert werden, was bei vielen Variationen recht aufwändig ist. Eine kontrollierte Mutation der Aufgaben könnte aber auch durch eine entsprechende Erweiterung des UserTestTools erreicht werden. Das Muster der Mutationen ließe sich beispielsweise in einer Meta-Aufgabendatei ablegen, die die möglichen Variationen definiert.

8 Literaturverzeichnis

Barnum, C., Bevan, N., Cockton, G., Nielsen, J., Spool, J., Wixon, D. (2003). The “Magic Number 5”: Is It Enough for Web Testing? Proceedings of CHI '03 extended abstracts on Human factors in computer systems, Ft. Lauderdale Florida, New York NY: ACM Press, 2003, S. 698-699

Barrett, R., Maglio, P., Kellem, D. (1997). How to Personalize the Web. Proceedings of the SIGCHI conference on Human factors in computing systems, Atlanta, Georgia, New York NY: ACM Press, 1997, S. 75-82

Blackmon, M. H., Polson, P. G., Kitajima, M., Lewis, C. (2002). Cognitive Walk-through for the Web. Proceedings of the SIGCHI conference on Human factors in computing systems, Minneapolis MN, New York NY: ACM Press, 2002, S. 463-470

Bleek, W., Kielas, W., Malon, K., Otto, T., Wolff, B. (2000) Vorgehen zur Einführung von Community Systemen in Lerngemeinschaften. In Engelien, M., Neumann, D. (Eds.) GeNeMe 2000: Gemeinschaften in Neuen Medien. Köln: Josef Eul Verlag, 2000

Borges, J., Morales, I., Rodriguez, N. (1996). Guidelines for designing usable World Wide Web pages. Proceedings of Con Conference on Human Factors and Computing Systems, Vancouver Canada, New York NY: ACM Press, S. 277-278

Bucklin, R. E., Sismeiro, C. (2003). A Model of Web Site Browsing Behavior - Estimated on Clickstream Data. Proceedings of Journal of Marketing Research, Volume 40, 2003, S. 249-267,
http://www.anderson.ucla.edu/faculty/randy.bucklin/papers/bucklinandsismeiro2003.pdf

Carroll, J. M., Rosson, M. B. (1987). Paradox of the Active User. In: Carroll, J. M. (Ed) Interfacing Thought: Cognitive Aspects of Human-Computer Interaction, Cambridge MA: MIT Press, 1987, S. 80-111

Cockburn, A., McKenzie, B. (2000). What Do Web Users Do? An Empirical Analysis of Web Use. Proceedings of International Journal of Human Computer Studies, Washington DC: Academic Press, 2000, S. 903-922,
http://www.cosc.canterbury.ac.nz/~andy/papers/ijhcsAnalysis.pdf

Dutke, S. (1994). Mentale Modelle: Konstrukte des Wissens und Verstehens – Kognitionspsychologische Grundlagen für die Software-Ergonomie. Göttingen: Verlag für Angewandte Psychologie

Etgen, M., Cantor, J. (1999). What does getting WET (Web Event-logging Tool) Mean for Web Usability? Proceedings of 5th Conference on Human Factors & the Web, 1999,
http://zing.ncsl.nist.gov/hfweb/proceedings/etgen-cantor/index.html

Graphic, Visualization & Usability Center (1995). GVU's 3rd WWW User Survey.
http://www.gvu.gatech.edu/user_surveys/survey-04-1995/

Haß, T. (2002). Studienarbeit: Java in heterogenen Umgebungen: Konzepte der Interaktion zwischen Javaprogrammen und Web-Browsern zur Erfassung von Benutzeraktionen. Studienarbeit an der Universität Hamburg, Fachbereich Informatik, Arbeitsbereich VSIS.
http://vsis-www.informatik.uni-hamburg.de/members/info.phtml/187/sa-torsten-hass.pdf

Hegaret, P., Pixley, T. (2003). Document Object Model Events – W3C Working Group Note,
http://www.w3.org/TR/2003/NOTE-DOM-Level-3-Events-20031107/events.html

Heinsen, S., Vogt, P. (2003). Usability praktisch umsetzen. München et al.: Carl Hanser Verlag

Hom, J. (1998). The Usability Methods Toolbox
http://jthom.best.vwh.net/usability/usable.htm

IBM (2003). WBI development Kit – Introduction.
http://www.almaden.ibm.com/cs/wbi/doc/index.html

Institut für Software-Ergonomie und Usability (2002). Usability Learning Center - Was ist Usability?
http://www.usability.ch/Deutsch/usab.htm

ISO 9241-10 (1996). Grundsätze der Dialoggestaltung, Berlin: Beuth-Verlag

ISO 9241-11 (1998). Anforderungen an die Gebrauchstauglichkeit – Leitsätze, Berlin: Beuth-Verlag

Ivory, M., Hearst, M. (2001). The State of the Art in Automating Usability Evaluation of User Interfaces. ACM Computing Surveys, Volume 33, No. 4, 2001, New York NY: ACM Press, S. 470-516

Jeffries, R., Miller, J., Wharton, C., Uyeda, K. (1991). User Interface Evaluation in the Real World: A Comparison of four Techniques. Proceedings of the SIGCHI conference on Human factors in computing systems, New Orleans LA, New York NY: ACM Press, S. 119-124

Lynch, P., Horton, S. (2002). Web Style Guide. 2nd Edition,
http://www.webstyleguide.com/index.html

Karat, C. (1994). A Comparison of User Interface Evaluation Methods. In: Nielsen, J. and Mack, R. L. (Eds.) Usability Inspection Methods, New York NY: John Wiley & Sons, 1994, S. 203-233

Karat, C., Campbell, R., Fiegel, T. (1992). Comparison of Empirical Testing and Walkthrough Methods in User Interface Evaluation. Proceedings of the SIGCHI conference on Human factors in computing systems, Monterey CA, New York NY: ACM Press, 1992, S. 397-404

Lewis, C. (1982). Using the „Thinking-aloud“ Method in Cognitive Interface Design, IBM Thomas J. Watson Research Center, Yorktown Heights NY

Licklider, J. (1960). Man-Computer Symbiosis. In proceedings IRE Transactions on human factors in electronics, Vol. HFE-1, 1960, S. 4-11

Maglio, P., Barrett, R. (2000). Intermediaries Personalize Information Streams – A Software-based middleman transforms data between client and server delivery and discovery. Communications of the ACM, Volume 43, Issue 8, New York NY: ACM Press 2000, S. 96-101

Mozilla Organization (2003). Mozilla 1.5 home page.
http://www.mozilla.org/products/mozilla1.x/

Nielsen, J. (1992). Finding usability problems through heuristic evaluation. Proceedings of the SIGCHI conference on Human factors in computing systems, Monterey CA, New York, NY: ACM Press, 1992, S. 373-380

Nielsen, J. (1993). Usability Engineering. Bosten et al.: Academic Press

Nielsen, J. (1994). Heuristic Evaluation. In: Nielsen, J. and Mack, R. L. (Eds.) Usability Inspection Methods, New York NY: John Wiley & Sons, 1994, S. 25-62

Nielsen, J. (2000). Jakob Nielsen's Alertbox, March 19, 2000: Why You Only Need to Test With 5 Users.
http://www.useit.com/alertbox/20000319.html

Nielsen, J. (2001). Jakob Nielsen's Alertbox, August 5, 2001: First Rule of Usability? Don't Listen to Users.
http://www.useit.com/alertbox/20010805.html

Pitkow, J., Recker, M. (1995). Using the Web as a Survey Tool: Results from the Second WWW User Survey 1995. Proceedings of the Third International World-Wide Web Conference, Computer Networks and ISDN Systems, Volume 27, Nr. 6, Darmstadt, 1995, S. 809-822,
http://www.cc.gatech.edu/gvu/user_surveys/papers/survey_2_paper.pdf

Rieman, J., Franzke, M., Redmiles, D. (1995). Usability Evaluation with the Cognitive Walkthrough. Conference companion on Human factors in computing systems, Denver CO, New York NY: ACM Press, 1995, S. 387-388

Roberts, S. (1999). Programming Microsoft Internet Explorer 5. Washington: Microsoft Press

Savage, P. (1996). User Interface Evaluation in an Interative Design Process: A Comparison of Three Techniques. Proceedings of CHI ’96 Short Papers, Vancouver Canada, New York NY: ACM Press, 1996, S. 307-308

Schweibenz, W., Thissen, F. (2003). Qualität im Web. Benutzerfreundliche Webseiten durch Usability Evaluation. Berlin et al.: Springer Verlag

Shneiderman, B. (1992). Designing the User Interface: Strategies for Effective Human-Computer Interaction. Second edition. Massachusetts et al.: Addison-Wesley

Shneiderman, B. (1998). Designing the User Interface: Strategies for Effective Human-Computer Interaction. Third edition. Massachusetts et al.: Addison Wesley Long-man

Siegel, D. (1998). Killer-Web-Sites der 3. Generation. Zweite Ausgabe. München: Markt und Technik Buch- und Softwareverlag

Spool, J., Schroeder, W. (2001). Testing web sites: Five users is nowhere near enough. CHI '01 extended abstracts on Human factors in computer systems, Seattle WA, New York NY: ACM Press, Seiten 285-286

Sun Microsystems (1999). Java Security Architecture, http://java.sun.com/j2se/1.4.1/docs/guide/security/spec/security-spec.doc1.html

Sutherland, I. (1963). Sketchpad: A Man-Machine Graphical Communications System. Proceedings of AFIPS Spring Joint Computer Conference, Jg. 23, 1963, S. 329-346

SyncRo Soft (2003). <oXygen/> XML editor, http://www.oxygenxml.com/index.html

Tauscher, L., Greenberg, S. (1997). How people revisit web pages: empirical findings and implications for the design of history systems. Proceedings of International

Journal of Human-Computer Studies archive. Volume 47, Washington DC: Academic Press, 1997, Seiten 97-137

W3C (2003a). Extensible Markup Language (XML).
http://www.w3.org/XML/

W3C (2003b). XML Schema.
http://www.w3c.org/XML/Schema

WebReference.com (2003). WebReference.com Browser Statistics,
http://www.webreference.com/stats/browser.html

Weinreich, H. (1997). Software-Ergonomie und das World Wide Web: 10 wichtige Leitlinien für die Gestaltung von ergonomischen WWW-Informationssystemen.
http://vsis-www.informatik.uni-hamburg.de/ergonomie/index.html

Weinreich, H. (2003). Scone – A Java Framework to Build Web Navigation Tools.
http://www.scone.de/

Weinreich, H., Buchmann, V., Lamersdorf, W. (2003). Scone: Ein Framework zur evaluativen Realisierung von Erweiterungen des Webs. In: Tagungsband Kommunikation in Verteilten Systemen - KiVS 2003. Heidelberg: Springer-Verlag, 2003, S. 31-42,
http://vsis-www.informatik.uni-hamburg.de/papers/kivs2003-scone.pdf

Weinreich, H., Lamersdorf, W. (2000). Concepts for Improved Visualization of Web Link Attributes. Proceedings of the 9th International World Wide Web Conference, Amsterdam, 2000,
http://vsys-www.informatik.uni-hamburg.de/projects/hyperscout/www9/

Weinreich, H., Obendorf, H. (2003). Comparing Link Marker Visualization Techniques – Changes in Reading Behavior. Proceedings of the twelfth international conference on World Wide Web, Budapest Hungary, New York NY: ACM Press, 2003, S. 736-745,
http://vsis-www.informatik.uni-hamburg.de/papers/www2003.pdf

Wharton, C., Bradford, J., Jeffries, R., Franzke, M. (1992). Applying Cognitive Walkthrougs to More Complex User Interfaces: Experiences, Issues, and Recommendations. Proceedings of the SIGCHI conference on Human factors in computing systems, Monterey CA, New York NY: ACM Press, 1992, S. 381-388

Wharton, C., Lewis, C. (1994). The Role of Psychological Theory in Usability Inspection Methods. In: Nielsen, J. and Mack, R. L. (Eds.) Usability Inspection Methods, New York NY: John Wiley & Sons, 1994, S. 341-350

Wharton, C., Rieman, J., Lewis, C., Polson, P. (1994). The Cognitive Walkthrough Method: A Practitioner's Guide. In: Nielsen, J. and Mack, R. L. (Eds.) Usability Inspection Methods, New York NY: John Wiley & Sons, 1994, S. 105-140

Anhang A: Benutzerdokumentation

Scone UserTestTool – Kurzeinführung

Torsten Haß

Diplomarbeit in den
Arbeitsbereichen ASI und VSIS
Fachbereich Informatik
Universität Hamburg

Das UserTestTool wurde entwickelt, um partizipative Gebrauchstauglichkeitstests von Websites zu erleichtern. Dieses Werkzeug unterstützt den Testleiter, indem es ihm viele nötige Routineaufgaben abnimmt und dadurch den Zeitbedarf dieser Testmethode verringert. Das UserTestTool zeigt dem Teilnehmer die zu erfüllenden Aufgaben und speichert die Eingaben des Teilnehmers. Des Weiteren kann das UserTestTool den Internet Explorer steuern und fast alle Aktionen des Teilnehmers, wie angeklickte Links, besuchte URLs und vieles mehr speichern.

Das UserTestTool ist im Rahmen einer Diplomarbeit an der Universität Hamburg, Fachbereich Informatik, Arbeitsbereiche ASI und VSIS entwickelt worden. Fragen, Anregungen und Kritik senden Sie bitte an Torsten.Hass@hansenet.de.

A.1 Einleitung

Tests, in denen Websites mit „echten“ Benutzern getestet werden, sind sehr zeit- und arbeitsaufwendig: Die Äußerungen des Teilnehmers müssen analysiert, ihre Aktionen, ihre Eingaben und die besuchten Internetseiten protokolliert und später ausgewertet werden. Zusätzlich müssen die Aufgaben erklärt werden, ohne den Teilnehmer zu beeinflussen.

Das UserTestTool wurde entwickelt, um den Testleiter zu unterstützen. Es nimmt dem Testleiter viele Routineaufgaben ab und verringert dadurch den Zeitbedarf dieser Testmethode. Das UserTestTool zeigt dem Teilnehmer die zu erfüllenden Aufgaben und speichert seine Eingaben. Des Weiteren kann das UserTestTool den Internet Explorer steuern und fast alle Aktionen des Teilnehmers, wie angeklickte Links, besuchte URLs und vieles mehr speichern.

A.2 Funktionsweise

Ein kompletter Test besteht aus mehreren Aufgaben, z. B. das Suchen von Informationen auf Websites, das Beantworten von Fragen. Jede Aufgabe besteht aus Texten, Schaltflächen oder Eingabemöglichkeiten, die dem Teilnehmer in einem Fenster präsentiert werden. Der Inhalt dieser Aufgabenfenster wird in einer XML-Datei definiert, in der die anzuzeigenden Texte, Schaltflächen und Eingabemöglichkeiten aufgeführt werden, und deren Verhalten festgelegt wird. Die Schaltflächen können bei Betätigung bestimmte Aktionen ausführen, z. B. andere Komponenten aktivieren, deaktivieren, hervorheben, Stoppuhren starten oder auslesen, Internetseiten im Browser öffnen oder die nächste Aufgabe starten. Für den Teilnehmer steht ein Textfeld für die Texteingabe und eine Auswahlliste zum Auswählen aus vordefinierten Texten zur Verfügung. Alternativ kann eine Likert-Skala eingesetzt werden, die die Auswahl des Teilnehmers als numerischen Wert speichert.

Die von dem Testteilnehmer eingegebenen Daten, die angeklickten Schaltflächen und viele Aktionen am Webbrowser werden gespeichert. Die gesammelten Daten werden im XML- und im TDF-Format abgelegt und können in andere Programme, z. B. Microsoft Excel, importiert werden. Eine genaue Beschreibung aller Einträge der Protokolldateien finden Sie im Anhang C.

A.2.1 Definition eines Tests

Der Ablauf eines solchen Tests mit all seinen Aufgaben wird in einer XML-Datei festgelegt. Jede Aufgabe wird in einem eigenen Bereich der XML-Datei definiert. Dabei wird die Aufgabe aus festgelegten Komponenten wie Text-, Schaltflächen- und Texteingabefeld-Komponenten zusammengesetzt. Dem Programmpaket liegt eine Beispieldatei bei, die den Dateiaufbau demonstriert und sich leicht um Komponenten oder Aufgaben erweitern lässt. Häufig wiederkehrende Komponenten oder Komponentengruppen lassen sich im Layout-Bereich der Datei vordefinieren. Diese können dann eingebunden und für die aktuelle Situation modifiziert werden.

A.3 Download

Das UserTestTool läuft innerhalb des Frameworks Scone ab. Scone ist ein Java-Framework, das die Programmierung und Evaluation von Navigationswerkzeugen unterstützt. Eine vorkonfigurierte Version von Scone, in der das UserTestTool enthalten ist, liegt unter

http://www.scone.de/userTestTool.html

zum Download bereit.

Das UserTestTool läuft unter Windows NT, 2000 und XP. Als Browser wird Microsofts Internet Explorer in der Version 5.0, 5.5 oder 6.0 benötigt.

Des Weiteren sind folgende Komponenten nötig:

- Java 2 Platform, Standard Edition (J2SE) Version 1.4.0 oder höher. Download unter: http://java.sun.com/j2se/downloads.html

A.4 Installation

- Installieren Sie das Java Development Kit (JDK), sofern es nicht bereits installiert ist.

- Installieren Sie Scone, indem Sie es in ein Verzeichnis Ihrer Wahl entpacken. Bitte benutzen Sie nur die vorkonfigurierte Version von der oben angegebenen Internetseite. Darin ist das UserTestTool bereits enthalten und vorkonfiguriert.

A.5 Konfiguration

Im Microsoft Internet Explorer müssen einige Einstellungen vorgenommen werden, damit er über Scone auf das Internet zugreift. Scone muss als Proxy-Server eingerichtet werden, „*.scone.de“ muss als vertrauenswürdige Site bekannt gemacht werden, das Cache muss ausgeschaltet und die Kommunikation mit Applets erlaubt werden. Außerdem muss Windows dem Internet Explorer erlauben, das Browserfenster zum obersten Fenster zu machen.

Im Verzeichnis „scone/setup“ befindet sich eine Datei mit dem Namen „Config_IE_5+6_For_Scone.reg“. Starten Sie diese, um alle erwähnten Einstellungen vorzunehmen. Um die Einstellungen rückgängig zu machen, starten sie die Datei mit dem Namen „Config_IE_5+6_No_Scone.reg“, die sich im gleichen Verzeichnis befindet. Die Einstellungen werden erst nach einem Neustart übernommen.

Die Proxy-Einstellungen dieser Konfigurationsdateien beziehen sich nur auf Verbindungen über die Netzwerkkarte. Im Fall von Internetverbindungen über Modem oder ISDN muss der Proxy-Server für die entsprechende DFÜ-Verbindung von Hand eingestellt werden. Das wird im nächsten Absatz beschrieben.

Wenn Sie über die Netzwerkkarte ins Internet gehen und, wie oben beschrieben, die Datei „Config_IE_5+6_For_Scone.reg“ ausgeführt haben, können Sie nun zum nächsten Kapitel springen.

Alternativ gibt es die Möglichkeit, die Einstellungen von Hand vorzunehmen. Die Proxy-Server-Einstellungen befinden sich im Internet Explorer unter dem Menü *Extras / Internetoptionen...* und dort auf der Registerkarte *Verbindungen*. Zum Einstellen des

Proxy-Servers für Internetverbindungen über die Netzwerkkarte klicken Sie bitte im Bereich *LAN-Einstellungen* auf die Schaltfläche *Einstellungen* (siehe Abbildung A.1). Zum Einstellen des DFÜ-Proxy-Servers klicken Sie bitte auf die Schaltfläche *Einstellungen* im Bereich *DFÜ- und VPN-Einstellungen.*

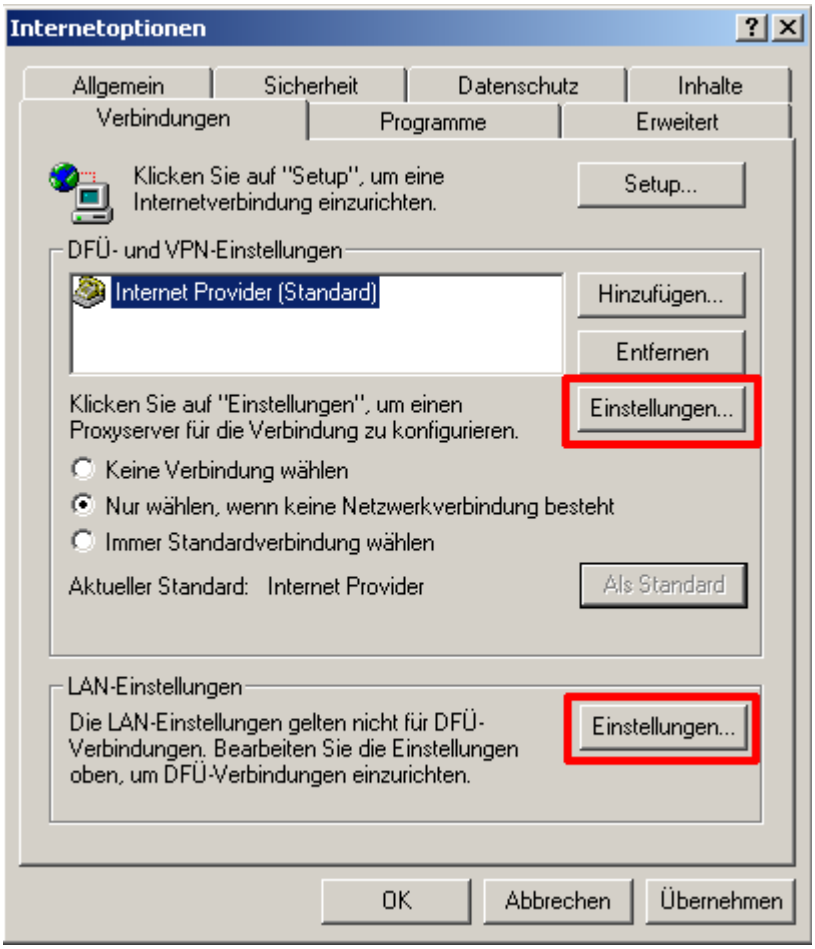


Abbildung A.1: Internetoptionen des Internet Explorers

Im erscheinenden Fenster aktivieren Sie das Kästchen vor *Proxyserver für...* und geben als Adresse: *localhost*, und als Port *8088* an (siehe Abbildung A.2 für Internetverbindungen über die Netzwerkkarte und Abbildung A.3 für Internetverbindungen über Modem oder ISDN).

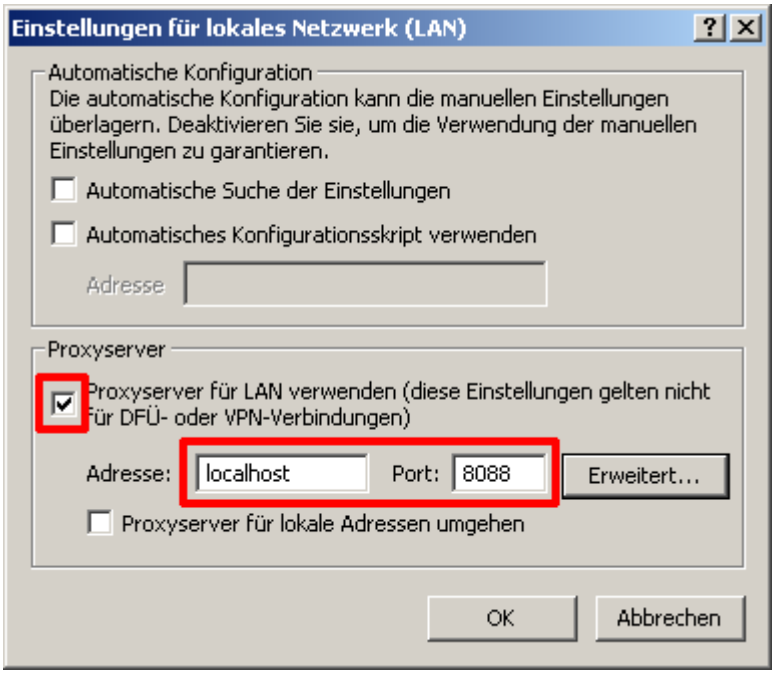


Abbildung A.2: Einstellungen für Verbindungen über das lokale Netzwerk

Zum Deaktivieren des Cache öffnen Sie bitte das Fenster *Internetoptionen* über das Menü *Extras / Internetoptionen...* und klicken Sie auf die Registerkarte *Allgemein*. Im Bereich *Temporäre Internetdateien* klicken Sie auf die Schaltfläche *Einstellungen*. In dem Fenster *Einstellungen* aktivieren Sie den Punkt *Bei jedem Zugriff auf die Seite*.

Die übrigen Einstellungen der Reg-Datei sind für das UserTestTool nicht erforderlich.

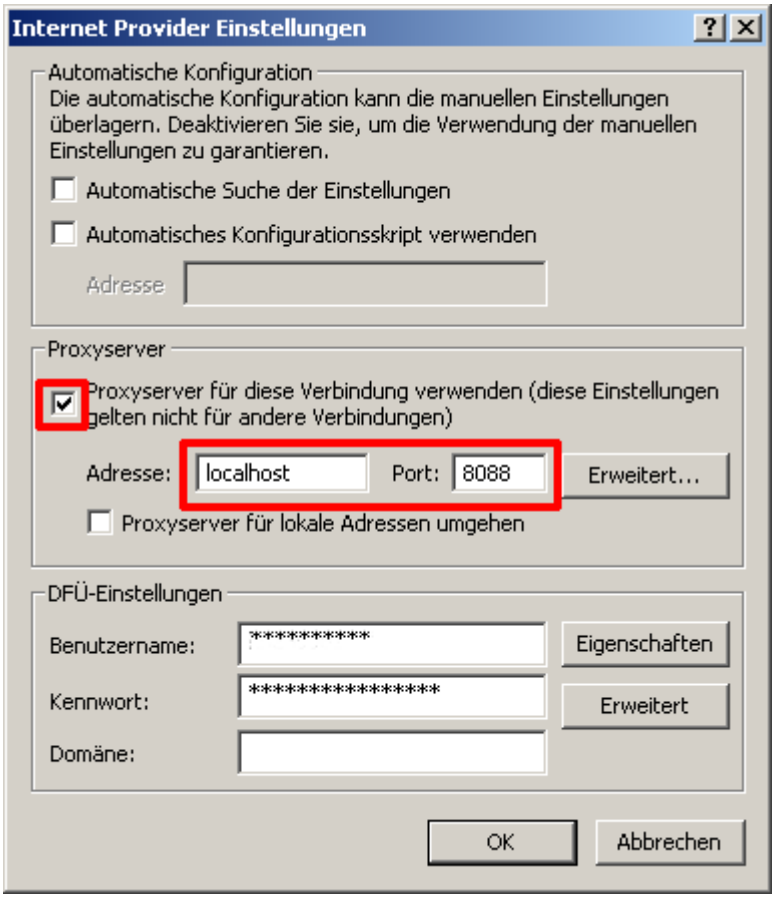


Abbildung A.3: Einstellungen für Internetprovider über Modem oder ISDN

A.6 Scone starten und beenden

Um Scone zu starten, rufen Sie bitte die Datei „runScone.bat“ in Scones run-Verzeichnis auf. Sobald Scone in seinem Textfenster die Meldung **`Console>`** ausgibt, können Sie das UserTestTool starten. Dazu öffnen Sie den Internet Explorer und rufen Sie die Seite „usertest.scone.de“ auf. Scone wird daraufhin auf der linken Bildschirmhälfte ein Auswahlfenster erzeugen. Wählen Sie unter Testbeschreibung den Beispieltest namens **`Beispiel`** aus, geben Sie einen Teilnehmernamen ein und klicken Sie auf „Neuen Test starten“. Dieses Beispiel sollte sich selbst erklären.

Sollten während des Ablaufes Fehler auftreten, z. B. bei fehlerhaften Testbeschreibungsdateien, dann werden diese in dem Textfenster ausgegeben, in dem Scone

gestartet wurde. Während des Starts werden alle Testbeschreibungsdateien überprüft und gefundene Fehler der XML-Struktur ausgegeben.

Um Scone zu beenden wechseln Sie zu dem Textfenster, in dem Scone gestartet wurde. Geben Sie dort den Befehl **`quit`** ein, um das UserTestTool und Scone zu beenden.

A.7 Definition der Testbeschreibungsdatei

In diesem Abschnitt wird der grobe Aufbau der XML-Testbeschreibungsdatei und der XML-Knoten beschrieben, aus denen die XML-Testbeschreibungsdatei zusammengesetzt werden kann. Die Testbeschreibungsdateien liegen in dem Verzeichnis „**`scone/run/config/userTestDescriptions`**“. Dort finden Sie auch die Beispiel.xml-Datei, die Sie gern als Vorlage benutzen können. Eine detaillierte Beschreibung der möglichen XML-Knoten finden Sie in der Komponentenreferenz im Anhang A.

Wichtig für eine funktionierende Testbeschreibungsdatei ist, dass sie auf jeden Fall den „**`?xml`**“-Knoten (Abschnitt A.1), mindestens einen „**`task`**“-Knoten (Abschnitt B.6) und als dessen Kindknoten die gewünschten Komponentenknoten enthält.

Jeder „**`task`**“-Bereich enthält genau eine Aufgabe. Im Bereich „**`layout`**“ gibt es mehrere „**`template`**“-Bereiche, in denen Komponenten vordefiniert werden. Diese Templates werden in die einzelnen Aufgaben eingebunden.

Des Weiteren ist zu beachten, dass XML die Groß-Kleinschreibung beachtet und keine ungeschlossenen Knoten erlaubt. Das heißt: jeder Knoten (z. B. **`<text ...>`**) muss durch einen abschließendes Tag (z. B. **`</text>`**) geschlossen werden. Alternativ kann ein Knoten im öffnenden Tag gleich geschlossen werden (z. B. **`<text ... />`**). Es ist auch nicht erlaubt, Attribute ohne Gleichheitszeichen und folgenden Wert in Anführungszeichen zu notieren. Attribute müssen immer in der Form **`attributName= “wert“`** innerhalb eines Knotens angegeben werden. Weitere Informationen zu XML finden Sie im Praktikum Internet des Fachbereichs Informatik unter:

http://print-www.informatik.uni-hamburg.de/Dokumentation/09_XML.pdf

oder auf den Internetseiten von Sun unter

http://java.sun.com/webservices/docs/1.0/tutorial/doc/IntroXML.html

Jede Komponente hat einen Namen, der ihr über das Attribut „**`name`**“ mitgegeben wird. Diese Namen dienen dazu, Komponenten während der Laufzeit zu beeinflussen. Diese Namen sind nur lokal für die aktuelle Aufgabe gültig. Komponenten anderer Aufgaben lassen sich nicht beeinflussen. Lediglich die Namen der Stoppuhrkomponenten: „**`stopWatchStart`**“, „**`stopWatchElapsed`**“ und „**`stopWatchLapTime`**“ sind aufgabenübergreifend gültig, um Zeitmessungen über mehrere Aufgaben und „Zeitspanne pro Aufgabe“ zu ermöglichen.

Jede Komponente hält 10 Pixel Abstand zur folgenden. Dieser Wert kann verändert werden, indem die obere Komponente das Attribut **`bottomPadding`** benutzt. Auf diese Weise lassen sich Komponenten räumlich von einander trennen oder zusammenziehen.

A.8 Protokolldatei

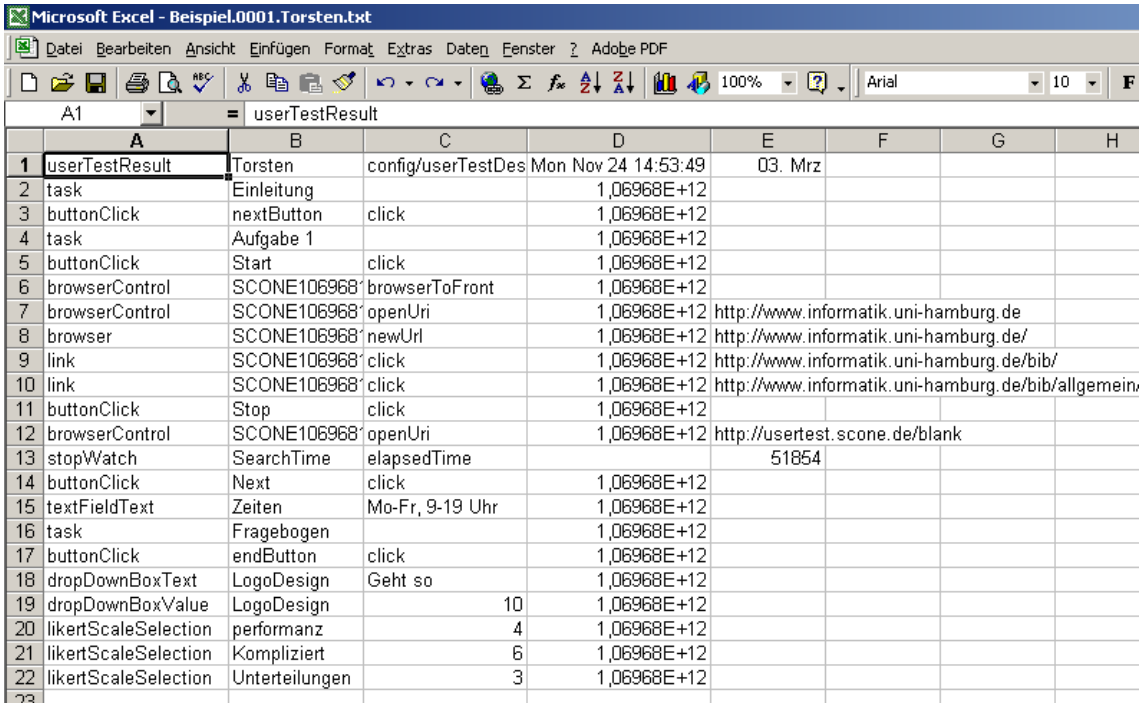
Alle Eingaben, Maus- und Tastaturaktionen des Teilnehmers werden in Protokolldateien gespeichert. Für jeden Test werden zwei Protokolldateien erzeugt: eine im XML-Format und eine im TDF-Format (Tab Delimited File).

Diese Dateien werden im Verzeichnis „**`scone/run/log/userTestResults`**“ abgelegt. Der Dateiname der jeweiligen Protokolldatei setzt sich aus dem Namen der Testbeschreibungsdatei, einer vierstelligen laufenden Nummer und dem Namen des Teilnehmers zusammen. Eine typische Protokolldatei hätte beispielsweise den Namen **`BeispielTest.0002.Meier.xml`**. Genau wie die Testbeschreibungsdatei ist auch die XML-Protokolldatei in Aufgaben („task“-Knoten) unterteilt. Innerhalb einer Aufgabe werden zunächst alle Aktionen des Teilnehmers abgelegt, danach sind die Daten der Textfelder und andere Eingabekomponenten gespeichert.

Jeder Eintrag innerhalb einer Aufgabe hat einen Knotennamen, der die Herkunft der Daten angibt. Die Attribute der Knoten geben die übrigen Daten an. So hat jeder Knoten innerhalb einer Aufgabe ein „**`action`**“-Attribut, in dem die Eingabe oder Aktion des

Teilnehmers gespeichert ist. Das „**`name`**“-Attribut gibt den Namen der Komponente an, von dem die Daten stammen. Jeder Komponente wurde in der Testbeschreibungsdatei ein Name zugewiesen. Das „**`timeStamp`**“-Attribut speichert den Zeitpunkt der Aktion, z. B. bei Schaltflächen, oder den Zeitpunkt des Auslesens, z. B. bei Textfeldern.

Eine genaue Liste der möglichen Einträge finden Sie im Anhang C: Einträge der Protokolldatei.



	A	B	C	D	E	F	G	H
1	userTestResult	Torsten	config/userTestDes	Mon Nov 24 14:53:49	03. Mrz			
2	task	Einleitung		1,06968E+12				
3	buttonClick	nextButton	click	1,06968E+12				
4	task	Aufgabe 1		1,06968E+12				
5	buttonClick	Start	click	1,06968E+12				
6	browserControl	SCONE106968	browserToFront	1,06968E+12				
7	browserControl	SCONE106968	openUri	1,06968E+12	http://www.informatik.uni-hamburg.de			
8	browser	SCONE106968	newUrl	1,06968E+12	http://www.informatik.uni-hamburg.de/			
9	link	SCONE106968	click	1,06968E+12	http://www.informatik.uni-hamburg.de/bib/			
10	link	SCONE106968	click	1,06968E+12	http://www.informatik.uni-hamburg.de/bib/allgemein			
11	buttonClick	Stop	click	1,06968E+12				
12	browserControl	SCONE106968	openUri	1,06968E+12	http://usertest.scone.de/blank			
13	stopWatch	SearchTime	elapsedTime		51854			
14	buttonClick	Next	click	1,06968E+12				
15	textFieldText	Zeiten	Mo-Fr, 9-19 Uhr	1,06968E+12				
16	task	Fragebogen		1,06968E+12				
17	buttonClick	endButton	click	1,06968E+12				
18	dropDownBoxText	LogoDesign	Geht so	1,06968E+12				
19	dropDownBoxValue	LogoDesign	10	1,06968E+12				
20	likertScaleSelection	performanz	4	1,06968E+12				
21	likertScaleSelection	Kompliziert	6	1,06968E+12				
22	likertScaleSelection	Unterteilungen	3	1,06968E+12				

Abbildung A.4: Microsoft Excel mit der importierten Protokolldatei

Die Protokolldatei im TDF-Format enthält die gleichen Einträge wie die XML-Protokolldatei. Jede Zeile beginnt mit dem Knotennamen. Es folgt der Komponenten- oder Framename, danach die Aktionsdaten, die Zeitangabe und der URI.

Die einzelnen Werte sind durch Tabstops voneinander getrennt. Auf diese Weise lässt sich die Datei in Programme wie Microsoft Excel importieren (siehe Abbildung A.4).

A.9 Fragen, Anregungen und Kritik

Das UserTestTool ist im Rahmen einer Diplomarbeit am Fachbereich Informatik der Universität Hamburg entstanden. Fragen, Anregungen und Kritik senden Sie bitte an Torsten Haß, E-Mail: Torsten.Hass@hansenet.de. Anregungen und konstruktive Kritik werden sowohl in das Werkzeug, als auch in die Diplomarbeit eingebaut.

Anhang B: Komponentenreferenz

In diesem Abschnitt werden die XML-Knoten beschrieben, aus denen die XML-Testbeschreibungsdatei zusammengesetzt werden kann. Die Testbeschreibungsdateien liegen in dem Verzeichnis „**scone/run/config/userTestDescriptions**“.

B.1 „?xml“

Der „?xml“-Konten informiert den XML-Parser über die verwendete XML-Version und die Zeichenkodierung. Dieser sollte in der ersten Zeile der Testbeschreibungsdatei stehen und die folgende Form haben:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
```

Diese Zeichenkodierung sollte der UTF-8-Kodierung vorgezogen werden, da viele Editoren diese Kodierung verwenden und damit die deutschen Umlaute in der Testbeschreibungsdatei korrekt erkannt werden.

B.2 „button“

Der „button“-Knoten erzeugt eine Schaltfläche in dem Aufgabenfenster. Ein Beispiel für eine Schaltflächendefinition ist die folgende:

```
<button name="Start" text="Aufgabe starten" enabled="false"
highlighted="true" bottomPadding="30">
  <browserToFront/>
  <browserResize width="800" height="600"/>
  <openUri>www.informatik.uni-hamburg.de</openUri>
  <enable>Stop</enable>
  <disable>Start</disable>
  <highlight>Stop</highlight>
  <stopWatchStart>SearchTime</stopWatchStart>
</button>
```

Folgende Attribute sind möglich:

- „**`name`**“ gibt dem Button einen Namen. Mit Hilfe dieses Namens kann die Schaltfläche durch andere Schaltflächen aktiviert werden und die Aktion der Schaltfläche in der Protokolldatei identifiziert werden.

- „**`text`**“ enthält die Beschriftung der Schaltfläche.

- „**`enabled`**“ legt fest, ob die Schaltfläche aktiviert und benutzbar („**`true`**“) oder ausgegraut und unbenutzbar („**`false`**“) ist. Wird das Attribut „**`enabled`**“ nicht angegeben, ist die Schaltfläche automatisch aktiviert.

- „**`highlighted`**“ gibt an, ob die Schaltfläche gelblich hinterlegt und damit hervorgehoben dargestellt werden soll.

- „**`bottomPadding`**“ legt die Anzahl der Pixel fest, die diese Komponente räumlich von der nächsten trennen.

Die Kindknoten des „**`button`**“-Knotens stehen für Aktionen, die beim Anklicken der Schaltfläche ausgeführt werden. Es werden folgende Kindknoten erkannt:

- „**`browserToFront`**“-Knoten bringen das Browserfenster in den Vordergrund. Dieser Knoten sollte zusammen mit dem „openUri“-Knoten verwendet werden, um das Browserfenster zum obersten Fenster zu machen, wenn eine neue Internetseite gezeigt werden soll.

- „**`browserResize`**“-Knoten erlauben das Ändern der Größe des Browserfensters. Über die Attribute width und height können die gewünschte Breite und Höhe des Browserfensters angegeben werden.

- „**`cancelTest`**“-Knoten bricht den Test ab. Die bisherigen Eingaben der Testperson in dieser Aufgabe werden nicht gespeichert. Die Wirkung dieses Knotens entspricht dem Schließen des Aufgabenfensters. Abgebrochene Tests können

später durch das Testauswahlfenster an beliebiger Stelle wieder aufgenommen werden.

- „**`enable`**“-Knoten aktivieren die Komponente, deren Name in den öffnenden und den schließenden Tag eingeschlossenen ist.

- „**`disable`**“-Knoten deaktivieren die Komponente, deren Name in dem Tag angegeben ist.

- „**`highlight`**“-Knoten heben die angesprochene Komponente hervor, indem diese gelblich unterlegt wird.

- „**`unhighlight`**“-Knoten stellen die normale graue Farbe der angesprochenen Komponente wieder her.

- „**`openUri`**“-Knoten öffnen in dem Browser, mit dem das UserTestTool aufgerufen wurde, eine bestimmte Internetseite. Der zu öffnende URI wird in dem öffnenden und schließenden Tag eingeschlossen. Um eine leere Seite zu öffnen, kann „_blank“ statt eines URIs angegeben werden.

- „**`requestFocus`**“-Knoten lassen die Eingabemarke zu der Komponente springen, deren Name von dem öffnenden und schließenden Tag eingeschlossen wird.

- „**`startNextTask`**“-Knoten speichert die Eingaben dieser Aufgabe und startet die nächste Aufgabe, wenn vorhanden. Dieser Knoten sollte der letzte in der Liste der Aktionen sein, da nach ihm keine weiteren Aktionen ausgeführt werden. Ansonsten würden z. B. danach notierte Stoppuhr-Komponenten nicht gestartet. Auch die letzte Aufgabe sollte durch Druck auf eine Schaltfläche mit diesem Knoten beendet werden, da sonst die Daten der letzten Aufgabe nicht gespeichert würden.

- „**stopWatchStart**“-Knoten starten eine Stoppuhr mit dem im Tag eingeschlossenen Namen. Derartige Stoppuhren sind Aufgabenübergreifend. Somit können auch Zeitspannen über mehrere Aufgaben gemessen werden.

- „**stopWatchElapsed**“-Knoten geben die seit dem Start der Stoppuhr verstrichene Zeit in der Protokolldatei aus. Der Name der zu verwendenden Stoppuhr wird in den Tag eingeschlossen.

- „**stopWatchLapTime**“-Knoten geben die seit dem letzten Aufruf dieses Knotens verstrichene Zeit dieser Stoppuhr in der Protokolldatei aus. Wurde dieser Knoten bisher nicht für diese Stoppuhr aufgerufen, wird die Zeit ausgegeben, die seit dem Start dieser Stoppuhr verstrichen ist. Der Name der zu verwendenden Stoppuhr wird in den Tag eingeschlossen.

B.3 „dropDownBox“

Der „dropDownBox“-Knoten erzeugt eine ausklappbare Auswahlbox, aus der sich die Testperson einen Text auswählen kann. Jeder Text kann mit einem ganzzahligen Wert versehen werden, der zusammen mit dem ausgewählten Text später in der Protokolldatei erscheint. Die Definition einer solchen Auswahlbox könnte derart aussehen:

```
<dropDownBox name="bewertung" description="Wie finden Sie
das Design der Website?" required="true">
  <item selectable="false" showThisFirst="true">Bitte
wählen Sie</item>
  <item value="5">Schlecht</item>
  <item value="10">Geht so</item>
  <item value="15">Ganz gut</item>
  <item value="20">Echt klasse</item>
</dropDownBox>
```

Folgende Attribute sind möglich:

- „**name**“ weist der Auswahlbox einen Namen zu, über den die Auswahlbox von anderen Komponenten angesprochen werden kann. Dieser Name wird auch in der Protokolldatei angegeben, wenn der ausgewählte Test gespeichert wird.

- „**description**“ legt den einzeiligen Text fest, der über der Auswahlbox angezeigt wird.

- „**required**“ gibt an, ob die Testperson einen selektierbaren (siehe selectable) Text auswählen muss (**required=“true“**) oder die Aufgabe auch ohne Auswahl eines Textes verlassen darf (**required=“false“**).

- „**enabled**“ dient zum Aktivieren oder Deaktivieren der Auswahlbox. Siehe Abschnitt B.1.

- „**highlighted**“ dient zum Hervorheben der Auswahlbox. Siehe Abschnitt B.1.

- „**bottomPadding**“ legt die Anzahl der Pixel fest, die diese Komponente räumlich von der nächsten trennen.

Jeder Texteintrag wird in einen Kindknoten mit dem Namen „**item**“ eingeschlossen. Jeder dieser Knoten kann mit den folgenden Attributen versehen werden:

- „**selectable**“ gibt an, ob dieser Texteintrag eine gültige Auswahl darstellt. Wird **selectable=“false“** angegeben, kann der Testbenutzer nicht zur nächsten Seite springen, bis er einen Text in der Auswahlbox gewählt hat, der ohne „**selectable**“-Eintrag oder mit **selectable=“true“** definiert wurde. Wird in einem Eintrag **selectable** nicht angegeben, wird **selectable=“true“** angenommen.

- „**showThisFirst**“ gibt an, ob dieser Texteintrag als erstes in der Auswahlbox gezeigt wird. Falls es mehr als einen Eintrag gibt, der das Attribut **showThisFirst=“true“** hat, wird das zuerst definierte gezeigt.

- „**value**“ erlaubt, jedem Eintrag einen ganzzahligen Wert zuzuweisen. Dieser Wert wird, genau wie der ausgewählte Text, in der Protokolldatei gespeichert. Dies kann für statistische Erhebungen nützlich sein.

B.4 „layout“

Der „layout“-Knoten bezeichnet den Bereich, in dem die Templates untergebracht sind. Darin lassen sich „template“-Knoten definieren, die eine oder mehrere Komponenten enthalten können. Diese Komponenten(-gruppen) lassen sich dann in die Aufgaben einbinden.

Ein Beispiel für einen „layout“-Bereich könnte etwa so aussehen:

```
<layout>
  <template name="kopf">
    <text name="einText">Oft verwendeter Text</text>
  </template>
</layout>
```

Der „**template**“-Knoten kennt nur das Attribut „name“. Hier wird ein Name erwartet, der unter den Templates dieser Aufgabe einmalig ist. Als Kindknoten des „**template**“-Knotens können beliebig viele Komponenten definiert werden.

Weitere Informationen zum Erstellen von Templates und deren Einbindung in die Aufgaben finden Sie im Abschnitt B.7 und B.12.

B.5 „likertScale“

Die Likert-Skala ist eine bewährte Technik bei der Erstellung von Fragebögen. Sie erlaubt dem Befragten, seine Wertung mit Hilfe von 5 oder 7 Einteilungen abzugeben. Ein Beispiel für das Einbinden einer Likert-Skala sieht folgendermaßen aus:

```
<likertScale name="lesbar" description="Die Seite ist
lesbar" numberOfRatings="7" minLabel="Ich stimme nicht zu"
maxLabel="Ich stimme zu" />
```

Folgende Attribute sind möglich:

- „**name**“ weist der Likert-Skala einen Namen zu, über den sie von anderen Komponenten angesprochen werden kann und der zur Identifikation der Daten in der Protokolldatei dient.
- „**description**“ legt den einzeiligen Text fest, der über der Likert-Skala angezeigt wird.
- „**numberOfRatings**“ gibt die Anzahl der Unterteilungen an, mit der die Likert-Skala gezeigt wird.
- „**minLabel**“ setzt die Beschriftung links von der Likert-Skala. Dies ist meist die minimale Wertung.
- „**maxLabel**“ setzt die Beschriftung rechts von der Likert-Skala. Dies ist meist die maximale Wertung.
- „**required**“ gibt an, ob die Testperson eine Wertung an der Likert-Scala abgeben muss (**required=“true“**) oder ob die Aufgabe auch ohne Abgabe einer Wertung verlassen werden darf (**required=“false“**).
- „**enabled**“ dient zum Aktivieren oder Deaktivieren der Likert-Skala. Siehe Abschnitt B.1.
- „**highlighted**“ dient zum Hervorheben der Likert-Skala. Siehe Abschnitt B.1.
- „**bottomPadding**“ legt die Anzahl der Pixel fest, die diese Komponente räumlich von der nächsten trennen.

Der „likertScale“-Knoten benötigt keine Kindknoten.

B.6 „task“

Dieser Knoten umfasst eine komplette Aufgabe, die der Testperson als ein Fenster dargestellt wird. Jeder „**task**“-Knoten ist Kindknoten des „**userTest**“-Knotens. Die Kindknoten des „**task**“-Knotens, die einzelnen Komponenten, formen das Aussehen des Aufgabenfensters.

Der „**task**“-Knoten erwartet lediglich das Attribut „**name**“. Dieses Attribut weist der Aufgabe einen Namen zu, über den die Aufgabe in der Protokolldatei erkannt werden kann.

B.7 „template“

Templates enthalten Komponenten oder Komponentengruppen, die in den Aufgaben eingebunden werden können. Sie müssen im „**layout**“-Bereich der XML-Datei definiert werden. Diese Knoten kennen lediglich das „**name**“-Attribut, über das sie in die Aufgaben eingebunden werden. Beispiel für ein Template:

```
<template name="kopf">
  <text name="einText">Oft verwendeter Text</text>
</template>
```

Kindknoten des „**template**“-Knotens können eine oder mehrere Komponenten sein, die nach belieben voreingestellt werden können. Beim Einbinden mit Hilfe von „**useTemplate**“ können bereits vorgenommene Einstellungen bei Bedarf wieder überschrieben werden. Weitere Informationen zum Einbinden von Templates finden Sie im Abschnitt B.12.

B.8 „text“

Dieser Knoten erlaubt das Einfügen von Text in das Aufgabenfenster. Seine einzigen Attribute sind „**name**“ und „**bottomPadding**“. Der auszugebende Text wird zwischen dem öffnenden und dem schließenden „**text**“-Tag notiert. Zur Formatierung können hier viele HTML-Tags benutzt werden, wie **<p/>** und **<h2></h2>**. Auch hier

gilt: da es sich um eine XML-Datei handelt, müssen alle Tags, auch die HTML-Tags, geschlossen werden. Ein ** ** ohne schließendes ** ** führt zu einer Fehlermeldung beim Start von Scone.

Beispiel eines korrekten „**text**“-Knotens:

```
<text name="einText" bottomPadding="30">
  <h2>Aufgabe 2</h2>
</text>
```

B.9 „textField“

Die Textfeldkomponente erlaubt der Testperson, einen Text einzugeben, der später in der Protokolldatei abgelegt wird. Beispiel eines „**textField**“-Knotens:

```
<textField name="Zeiten" description="Ihr Text" text=""/>
```

Folgende Attribute sind möglich:

- „**name**“ weist dem Textfeld einen Namen zu, über den das Textfeld von anderen Komponenten angesprochen werden kann und der zur Identifikation der Daten in der Protokolldatei dient.
- „**description**“ legt den einzeiligen Text fest, der über dem Textfeld angezeigt wird.
- „**required**“ gibt an, ob die Testperson den Text des Textfeldes verändern muss (**required=“true“**), oder ob die Aufgabe auch ohne Änderung des Textes verlassen werden darf (**required=“false“**).
- „**text**“ erlaubt, einen Text im Textfeld anzuzeigen.
- „**enabled**“ dient zum Aktivieren oder Deaktivieren des Textfeldes. Siehe Abschnitt B.1.
- „**highlighted**“ dient zum Hervorheben des Textfeldes. Siehe Abschnitt B.1.

- „**bottomPadding**“ legt die Anzahl der Pixel fest, die diese Komponente räumlich von der nächsten trennen.

B.10 „title“

Der Text, der zwischen dem öffnenden und dem schließenden „**title**“-Tag notiert wird, wird in die Titelleiste des Aufgabenfensters übernommen.

B.11 „userTest“

Der „**userTest**”-Knoten ist das Wurzelelement der XML-Testbeschreibungsdatei. Jeder „**task**“- und der „**layout**“-Knoten müssen Kindknoten des „**userTest**“-Knotens sein.

B.12 „useTemplate“

Dieser Knoten dient zum Einbinden von Templates in eine Aufgabe. Es gibt zwei Möglichkeiten, Templates einzubinden: Unveränderndes Einbinden und veränderndes Einbinden.

Beim unverändernden Einbinden wird ein Template so eingebunden, wie es im „**template**“-Knoten definiert wurde. Dazu genügt eine Zeile in der Definition der Aufgabe:

```
<useTemplate name="kopf"/>
```

Die Komponenten, die in dem Template namens „**kopf**“ definiert wurden, werden ohne Änderung übernommen.

Beim verändernden Einbinden werden ebenfalls alle Komponenten aus dem Template übernommen. Es besteht aber die Möglichkeit, beliebig viele der Komponenten in der Aufgabe umzukonfigurieren. Dabei müssen nur die Komponenten erwähnt werden, die auch verändert werden sollen. Alle anderen Komponenten werden, wie im Template angegeben, erstellt. Die Reihenfolge der Komponenten lässt sich durch nachträgliches Konfigurieren nicht verändern. Beispiel für veränderndes Einfügen:

```
<useTemplate name="auswahlfeld">
   <dropDownBox name="DesignFrage"
   nameInTemplate="bewertung" description="Wie finden Sie
   das Design der Website?" />
</useTemplate>
```

Wenn vorher in einem Template namens „**auswahlfeld**“ eine DropDownBox mit mehreren Einträgen und dem Namen „**bewertung**“ definiert wurde, kann diese nun, wie im obigen Beispiel, in die Aufgabe eingebunden werden. Dabei wird sie um einen eindeutigen Namen und eine Beschriftung ergänzt. Das Attribut „**nameInTemplate**“ ist nötig, um genau eine Komponente anzusprechen. Auch wenn es im Template nur eine DropDownBox gibt, muss sie über „**nameInTemplate**“ direkt angesprochen werden.

Der „**useTemplate**“-Knoten kennt nur das „**name**“-Attribut. Es gibt an, nach welchem Template gesucht werden soll.

Alle Komponenten, die verändert werden sollen, müssen als Kindknoten des „**useTemplate**“-Knotens aufgeführt werden. Diese können dann konfiguriert werden. Jede dieser Komponenten muss um das „**nameInTemplate**“-Attribut erweitert werden, das den Namen der Komponente aus dem Template enthalten muss.

Das Attribut „**name**“ der zu ändernden Komponenten enthält den neuen Namen der Komponente. Der neue Name ist nötig, damit im Falle eines mehrfach importierten Templates, nicht alle importierten Textfelder den gleichen Namen tragen, wenn sie in der Protokolldatei abgelegt werden.

Anhang C:
Einträge der Protokolldatei

In diesem Abschnitt werden alle Einträge beschrieben, die in der Protokolldatei eines Testlaufs stehen können.

C.1 „userTestResult“

Dieser Knoten ist der Hautpknoten der XML-Protokolldatei. Darunter hängen die Daten aus den einzelnen Aufgaben.

Attribute:

- „**completedTasks**“ gibt die Anzahl der abgearbeiteten Aufgaben an.
- „**numberOfTasks**“ gibt die Anzahl der im Test enthaltenen Aufgaben an.
- „**testPerson**“ enthält den Namen der Testperson.
- „**timeStamp**“ speichert den Zeitpunkt des Testbeginns.
- „**xmlFile**“ gibt an, welche Testbeschreibung für diesen Test verwendet wurde.

C.2 „task“

Dieser Knoten bezeichnet den Beginn einer neuen Aufgabe. Alle Kindknoten sind Aktionen und Daten, die während dieser Aufgabe erfasst wurden.

Attribute:

- „**name**“ gibt den Namen der Aufgabe an.
- „**timeStamp**“ speichert den Zeitpunkt des Aufgabenstarts.

C.3 „buttonClick“

Die Testperson hat eine Schaltfläche im Aufgabenfenster angeklickt.

Attribute:

- „**name**“ gibt den Namen der Schaltfläche an, auf die geklickt wurde.
- „**action**“ enthält die Aktion der Testperson. Dieser Knoten kennt nur die Aktion „click“.
- „**timeStamp**“ speichert den Zeitpunkt des Anklickens.

C.4 „browserControl“

Durch das UserTestTool wurde im Browser eine Internetseite aufgerufen. Diese Aktion wurde ausgelöst, als die Testperson eine Schaltfläche angeklickt hat.

Attribute:

- „**frameName**“ gibt den Namen des Browserfensters an.
- „**action** enthält die Aktion der Browsersteuerung. Dieser Knoten kennt nur die Aktionen „openUri“, die eine neue Seite im Browser lädt, „browserToFront“, die das Browserfenster in den Vordergrund bringt und „browserResize“, die die Größe des Browserfensters ändert.
- „**timeStamp**“ speichert den Zeitpunkt der Browsersteuerung.
- „**uri**“ speichert den aufgerufenen URI.

C.5 „browser“

Der Browser hat eine Aktion gemeldet, z. B. das Laden einer neuen URL durch die Schaltflächen des Browsers.

Attribute:

- „**frameName**“ gibt den Namen des Browserfensters an.
- „**action**“ enthält die Aktion des Browsers. „formSubmit“ bezeichnet ein abgeschicktes Formular, „back“, „next“ und „reload“ die entsprechenden Schaltflächen des Browsers. „newUrl“ steht für eine von der Testperson eingegebene Internetadresse oder für die Fernsteuerung durch das UserTestTool oder durch JavaScript.

- „**timeStamp**“ speichert den Zeitpunkt der Browsersteuerung.
- „**uri**“ gibt die neue URL des Browsers an.

C.6 „formData“

Nach dem Abschicken eines Formulars aus einer Internetseite werden die Formulardaten an den Server übermittelt. Diese Daten werden abgefangen. Für jedes Formularfeld wird einer dieser Knoten erzeugt.

- „**frameName**“ gibt den Namen des Browserfensters an.
- „**action**“ enthält den Namen des Formularfeldes, gefolgt von einem Gleichheitszeichen und dem in das Feld eingetragenen Text.
- „**timeStamp**“ speichert den Zeitpunkt der Aktion.
- „**uri**“ gibt die neue URL des Browsers an.

C.7 „link“

Der Browser hat das Laden einer neuen URL durch das Anklicken eines Links gemeldet.

Attribute:

- „**frameName**“ gibt den Namen des Browserfensters an.
- „**action**“ enthält die Aktion, die durch das Anklicken des Links ausgelöst wurde. „click“ bezeichnet einen Link zu einer anderen Internetseite, „fragmentOnSamePage“ steht für einen Link, der auf der aktuellen Seite auf ein bestimmtes Fragment verweist. „samePage“ zeigt einen Link an, der auf die aktuelle Seite verweist.
- „**timeStamp**“ speichert den Zeitpunkt der Aktion.
- „**uri**“ gibt die neue URL des Browsers an.

C.8 „stopWatch“

Eine Zeitspanne wurde gemessen.

Attribute:

- „**name**“ gibt den Namen der Stoppuhr-Komponente an.
- „**action**“ enthält die Aktion der Stoppuhr. Im Fall von „elapsedTime“ wird die vergangene Zeit seit dem ersten Aufruf eines Timers dieses Namens gespeichert. Bei „lapTime“ wird die Zeitspanne seit dem letzten Aufruf des Timers dieses Namens gespeichert.
- „**periodOfTime**“ speichert die oben beschriebene Zeitspanne.

C.9 „dropDownBoxText“

Beim Beenden der Aufgabe wurde der Text der DropDownBox ausgelesen, den die Testperson ausgewählt hat.

Attribute:

- „**name**“ gibt den Namen der DropDownBox an.
- „**action**“ enthält den Text, den die DropDownBox zuletzt gezeigt hat.
- „**timeStamp**“ speichert den Zeitpunkt des Auslesens.

C.10 „dropDownBoxValue“

Beim Beenden der Aufgabe wurde der Wert, der dem angezeigten Text der DropDownBox zugewiesen wurde, ausgelesen.

Attribute:

- „**name**“ gibt den Namen der DropDownBox an.
- „**action**“ enthält den Wert, der dem zuletzt angezeigten Text zugewiesen ist.
- „**timeStamp**“ speichert den Zeitpunkt des Auslesens.

C.11 „likertScaleSelection“

Beim Beenden der Aufgabe wird die Bewertung, die die Testperson in der Likert-Skala angeklickt hat, ausgelesen.

Attribute:

- „**name**“ gibt den Namen der Likert-Skala an.
- „**action**“ enthält die entsprechende Zahl, die die Testperson angeklickt hat. Falls die Testperson keine Wertung über die Likert-Skala abgegeben hat, ist die Zeichenkette leer.
- „**timeStamp**“ speichert den Zeitpunkt des Auslesens.

C.12 „textFieldText“

Beim Beenden der Aufgabe wird der Text des Textfeldes ausgelesen, den die Testperson eingegeben hat.

Attribute:

- „**name**“ gibt den Namen des Textfeldes an.
- „**action**“ enthält den Text, den die Testperson eingegeben hat. Falls die Testperson keinen Text eingegeben hat, enthält dieses Attribut den Text, den das Textfeld angezeigt hat.
- „**timeStamp**“ speichert den Zeitpunkt des Auslesens.