

Software-Qualitätssicherung, Teil I: Grundlagen des Softwaretests

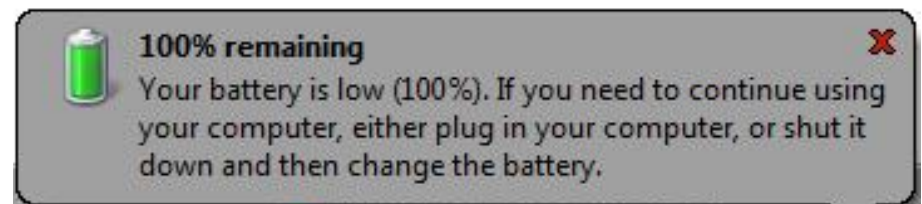
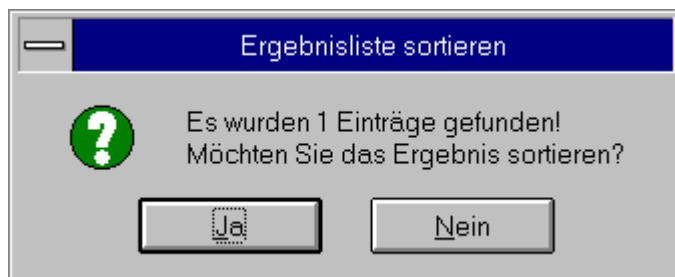
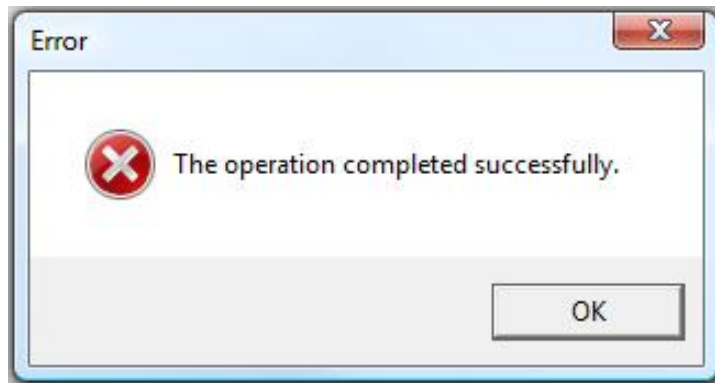
Friedrich-Schiller-Universität Jena, Wintersemester 2017/2018
Ronny Vogel, Xceptance GmbH

Agenda

- Einführung
- Klassifikation von QS-Maßnahmen
- Testfallerstellung
- Ökonomie des Testens
- Systematische Testfallermittlung
- Testausführung
- Fehlerverfolgung
- Kurzüberblick Testmanagement

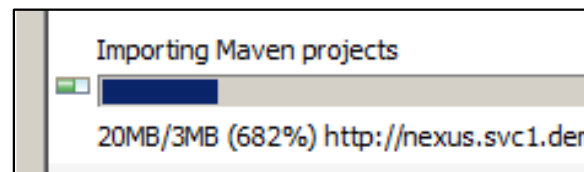
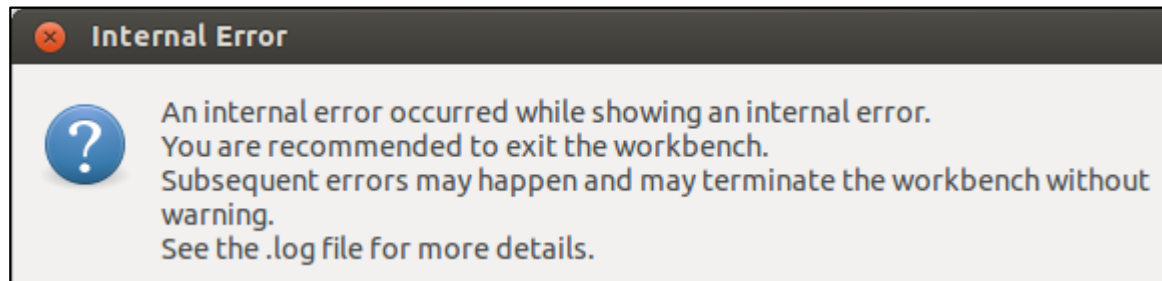
Einführung >

Notwendigkeit von Softwaretests (1)



Einführung >

Notwendigkeit von Softwaretests (2)



The "Select View" control allows the user to change the view they would like to pick up the items at. When the user clicks this control a lightbox is displayed similar to product pages.

Einführung >

Notwendigkeit von Softwaretests (3)

- **1962:** Verlust der Venus-Sonde Mariner 1 (80 Mio. \$) durch fehlenden Bindestrich
- **1971:** Rund um die Erde explodieren 72 von 141 Wetterballons bei einem Experiment aufgrund eines falsch interpretierten Satelliten-Kommandos.
- **1982:** Die Vertauschung der Steuerung zwischen Höhenruder und Seitenruder beim Bau eines F-117-Tarnkappenbombers führte zum Absturz des Flugzeugs.
- **1985:** Die Software eines Roboters bei GM verarbeitete keine Information von schwarzen Autos, die die Montagehalle daher ohne Windschutzscheibe verließen.
- **1985-1987:** Mehrere Verletzte und drei Tote durch Bestrahlungsgerät Therac-25
- **1989:** Innerhalb einer halben Stunde wurden 2000 Mio. Pfund an Kunden in England und USA doppelt überwiesen. Zinsverlust trotz Rückzahlung: bis 0.5 Mio. Pfund/Tag.
- **1991:** Signalsystem verursacht Zusammenstoß zweier Züge in Japan mit 42 Toten
- **1992:** Ein Programm der British Telecom zur Annahme von Notrufen spielte nur "Bitte warten" vor. Einige Anrufer verstarben während dieser Wartezeit.
- **1993:** Ein Airbus A320 rollte in Warschau über die Landebahn hinaus, da das Aquaplaning auf der Landebahn falsch berechnet wurde.
- **1995:** Massive Verzögerungen im bundesweiten Zugverkehr durch einen Stack Overflow in der Software eines Hamburger Stellwerks

Einführung >

Notwendigkeit von Softwaretests (4)

- **1996:** Sprengung des Prototyps der Ariane-5 kurz nach dem Start, weil der von der Ariane-4 wiederverwendete Code aufgrund der jetzt höheren Beschleunigung in einen Fehler bei einer Typumwandlung lief
- **1997:** die „USS Yorktown“ trieb bei einem Manöver drei Stunden hilflos im Mittelmeer, da ein Mitarbeiter einen fehlerhaften Sensorwert in der Datenbank mit 0 überschrieb; es kam zu einer Division durch Null, die nicht korrekt abgefangen wurde; der temporäre Speicher lief in den Speicherbereich des Antriebssystems über; das Netzwerk aus mehreren Windows-NT-Servern fiel aus
- **1999:** Verlust der NASA-Sonde „Mars Climate Orbiter“ beim Marsanflug, weil der Schub in Pfund x Sekunde statt in Newton x Sekunde berechnet wurde (125 Mio. \$)
- **2003:** Verzögerungen bei Toll Collect mit Vertragsstrafen und Einnahmeausfällen in Milliardenhöhe, u.a. aufgrund der fehlenden Kompatibilität von Software-Modulen
- **2005:** Im russischen Plessezk führte ein Programmfehler zum Fehlstart einer Trägerrakete und zum Verlust des Satelliten CryoSat.
- **2005:** Durch mangelnde Umstellungsmöglichkeiten der Software für das ALG II überwies die BfA 300 Mio. € zu viel an die Krankenkassen, die 20% davon für den zusätzlichen Verwaltungsaufwand behielten.

Einführung >

Notwendigkeit von Softwaretests (5)

- **2006:** Ein missglücktes Annäherungsmanöver zweier US-Satelliten führte zu einer veränderten Umlaufbahn mit 1 Mio. \$ Schaden. Ursache waren Softwarefehler in einem der Satelliten und im GPS-System.
- **2007:** Zehn Personen der südafrikanischen Armee kamen aufgrund eines Programmfehlers in einem vollautomatisierten 35-mm-Flakgeschütz ums Leben.
- **2008:** Gepäcksystem des Heathrow Airport's Terminal 4 zwei Tage offline aufgrund von Softwareproblemen; Verzögerungen und Beschwerden tausender Passagiere
- **2009:** Ein Airbus A380 musste aufgrund von Problemen mit dem Navigationssystem umkehren und zum Ausgangsflughafen in New York zurückfliegen.
- **2010:** Die Bezahlung und das Holen von Bargeld mit deutschen Girocard- und Kreditkarten waren bei etwa 30 Millionen Karten nicht möglich. Ursache war ein Chip, dessen Software Fehler bei der Verarbeitung der Jahreszahl 2010 aufweist.
- **2010:** Die Amerikanerin Louise Chavez gewann an einem Münzautomaten in Colorado knapp 43 Millionen Dollar. Laut Casinobetreiber handelte es sich um einen Software-Fehler. Eine Untersuchung wurde eingeleitet, das Casino sprach ihr \$20 zu.
- **2010:** Viele kleine Rückbuchungen wurden bei der britischen TUI Travel über Jahre nicht korrekt verbucht. Die TUI AG musste nun 120 Millionen Euro abschreiben. Der Finanzchef trat zurück, das Eigenkapital wurde um diese Summe gesenkt.

Einführung >

Notwendigkeit von Softwaretests (6)

- **2011:** Honda rief weltweit etwa 2,5 Millionen Autos in die Werkstätten zurück, da ein Software-Problem zu Schäden am Automatik-Getriebe führen konnte.
- **2011:** Fehlerhafte Software eines 2,7 Mrd. USD teuren Aufklärungssystems behinderte die US-Armee in Afghanistan und im Irak. Es versagte selbst bei einfachen Aufgaben und reagierte verzögert bei mehreren parallelen Nutzern.
- **2011:** Hackerangriff auf das Playstation-Netzwerk von Sony; Daten von mehr als 70 Millionen Nutzern gestohlen
- **2011:** Hacker haben Daten von hunderttausenden US-Kunden der US-Großbank Citigroup ausgespäht
- **2012:** 45min lang überflutete ein neues Programm des Finanzdienstleisters "Knight Capital Group" die Wall Street mit fehlerhaften Handelsaufträgen; die Firma blieb auf vielen zu teuer gekauften Aktien sitzen; 440 Millionen Dollar Verluste
- **2012:** Ein Fehler in der WLAN-Software bestimmter Telekom-Router ermöglichte jedermann das unbemerkte Eindringen in das Netzwerk.
- **2012:** Das Messsystem verweigerte bei Olympia 2012 die Annahme der Weite von Hammerwerferin Betty Heidler, da es aufgrund des gleichen Messwerts der Werferin vor ihr einen Fehler der Kampfrichter zugrunde legte. Die Medaille war gefährdet.

Einführung >

Notwendigkeit von Softwaretests (7)

- **2012:** Ein Fehler im Internet Explorer ermöglichte Angreifern, Computer unter ihre Kontrolle zu bringen, das BSI warnte.
- **2013:** Ein Kommando flutete das Steuerungssystem der Europäischen Elektrizitätsnetze mit Daten; das führte beinahe zum Zusammenbruch der Elektrizitätsnetze in Österreich; möglicher Blackout in halb Europa
- **2013:** Eine Sicherheitslücke im Heizungssystem Vaillant EcoPower 1.0 ermöglichte Angreifern das Ein- und Ausschalten und die Beschädigung über das Internet
- **2013:** Probleme an der Motorsteuerung des Opel Insignia können zu Motorschäden führen. Opel ruft 61.000 Autos zurück.
- **2013:** Chrysler ruft eine halbe Million Jeeps zurück, da durch ein Softwareproblem die Gangschaltung unbeabsichtigt verstellt werden kann.
- **2013:** Durch Softwareprobleme war drei Stunden lang kein Handel an der New Yorker Tech-Börse NASDAQ möglich. Amerikas gesamter Tech-Markt (2712 Aktienwerte) war nicht handelbar, auch nicht an anderen Börsen.
- **2013:** Xerox-Multifunktionsgeräte ersetzen beim Scannen als PDF (ohne OCR-Texterkennung) klein gedruckte Ziffern willkürlich durch andere, was z.B. in Bauplänen zu falschen Zahlenangaben führte.

Einführung >

Notwendigkeit von Softwaretests (8)

- **2013:** „Heartbleed“-Sicherheitslücke in der Open-Source-Bibliothek „OpenSSL“; private Daten über verschlüsselte Verbindungen können ausgelesen werden
- **2013:** monatelange Wartezeiten und nicht auffindbare Teile im neuen, von IBM und SAP implementierten Ersatzteil-Logistiksystem „ATLAS“ bei BMW
- **2014:** Ein Softwarefehler legt die Luftverkehrskontrolle um Los Angeles lahm; der Flugplan eines Militärflugzeugs in 20 km Höhe wurde fälschlich mit zivilen Flügen koordiniert, obwohl sich die Routen aufgrund verschiedener Höhen nicht kreuzen
- **2014:** Toyota ruft weltweit 1,9 Millionen Prius III zurück; durch einen Softwarefehler kann es zur Überlastung der Elektronik kommen, das Fahrzeug schaltet in ein Notprogramm und wird langsamer; im Extremfall kann es sogar stehen bleiben
- **2014:** „ShellShock“-Sicherheitslücke in der Unix-Shell „bash“; beliebiger Schadcode über das Netz ausführbar; bedroht sind vor allem Webserver
- **2014:** ein Fehler in einem Tool von RepricerExpress setzt in UK den Preis hunderter Produkte im Amazon Marketplace auf 0,01 Pfund; immense Verluste einiger Anbieter
- **2015:** ein Fehler bei der Preismstellung führte für 2h zu nur 6 ct/l bei Super Plus an 30 Tankstellen um Nürnberg; die zentralen Rechner zeigten den korrekten Preis

Einführung >

Notwendigkeit von Softwaretests (9)

- **2015:** eine bestimmte Tastenkombination beim Roulette in Merkur-Spielautomaten führte immer zu hohen Gewinnen; 10 Mio. € Verlust in einer Nacht; Tage vorher wurde der Softwarefehler vom Hersteller entdeckt und die Spielhallen-Betreiber gewarnt, aber diese hatten keine Zeit mehr, zu reagieren
- **2015:** administrativer Zugriff auf Hosting-Pakete und Domains mit einem beliebigen Passwort im Reseller-Kundenportal von 1und1.de, gmx.de, web.de durch SW-Fehler
- **2015:** potentieller Strom- und Steuerungsausfall in Boeing 787 durch Überlauf eines Zählers in der Generatorsteuerung nach 248 Tagen mit Abschaltung des Generators; vorgeschriebener Neustart pro Maschine alle 120 Tage als temporärer Workaround
- **2015:** Sicherheitslücke bei BMW; mehr als 2 Millionen Fahrzeuge hätten unberechtigt geöffnet werden können
- **2015:** Android-Sicherheitslücke „Stagefright“ ermöglicht das Übernehmen und Fernsteuern hunderter Millionen Smartphones durch manipulierte MMS oder Videos
- **2015:** Absturz eines Airbus A400M in Spanien bei einem Testflug; kurz nach dem Start erhielten drei der vier Triebwerke widersprüchliche Befehle und drosselten die Leistung, vermutlich durch fehlerhaft installierte Software zur Triebwerkssteuerung; vier Tote und zwei Schwerverletzte

Einführung >

Notwendigkeit von Softwaretests (10)

- **2015:** ein Jeep Cherokee wurde bei einer Demonstration remote gehackt und ferngesteuert, u.a. Bremsen und Motor; Chrysler ruft 1,4 Millionen Fahrzeuge zurück
- **2015:** im US-Bundesstaat Washington wurden von 2002 und 2015 bis zu 3200 Gefängnisinsassen zu früh entlassen; die genaue Zahl ist nicht bekannt; einige Gefangene wurden kurz danach wieder eingesperrt
- **2016:** die Software der Gepäck-Förderanlage am Flughafen Düsseldorf erkannte den 29. Februar nicht als Tag; etwa 1200 Koffer blieben dadurch liegen; Mitarbeiter mussten das Gepäck von Hand sortieren
- **2016:** das 286 Millionen USD teure japanische Weltraumteleskop Hitomi zerbrach etwa einen Monat nach Inbetriebnahme bei einem Ausrichtungsmanöver im Orbit durch einen Softwarefehler in der Lageregelung
- **2016:** beim Online-Banking von Comdirect sahen Kunden Kontodaten und Postfächer fremder Nutzer nach einem Software-Update
- **2016:** Absturz der Marssonde "Schiaparelli"; sie warf schon beim Landeanflug die Bremsfallschirme ab und schaltete die Bremstriebwerke zu kurz ein, wahrscheinlich durch Überlastung einer Messkomponente, die zeitweilig nicht reagierte, so dass falsche Höhenangaben berechnet wurden (unterhalb der Marsoberfläche)

Einführung >

Notwendigkeit von Softwaretests (11)

- **2017:** Fehlerhaftes Firmware-Update legt über WLAN steuerbare Türschlösser von „RemoteLock“ lahm; keine Öffnung mehr per Tastenkombination; kein Update mehr möglich; die Hardware muss getauscht werden
- **2017:** das cloudbasierte „Smart Garden System“ von Gardena löscht in der Haupturlaubszeit Daten einiger Benutzer, auch Backups und lokale Daten in den gesteuerten Geräten; Neueinrichtung erfordert physischen Zugriff auf die Geräte
- **2017:** ein Krimineller stahl in den USA jahrelang aus mehr als 100 Hotels Wertgegenstände; er nutzte eine Sicherheitslücke in den Schlössern von Onity mit einem selbstgebauten Arduino-basierten Gerät über den zugänglichen Diagnoseport
- **2017:** Firmware-Update für unsichere Herzschrittmacher und Heim-Basisstationen von Abbot nötig; fast 500.000 Patienten betroffen; Sicherheitslücken ermöglichten Konfigurationsänderungen und die Ausführung von Befehlen über Funk
- **2017:** beim Drucken von PDF-Dateien mit dem Browser Microsoft Edge erscheint unter bestimmten Umständen statt "1234" auf dem Ausdruck die Ziffernfolge "1144"

Einführung >

Notwendigkeit von Softwaretests (12)

Beobachtung

- potentielle Auswirkungen von Softwarefehlern nehmen offenbar eher zu statt ab

Einerseits: Potentiell höhere Softwarequalität

- Anstrengungen und Fortschritte in der ingenieurmäßigen Erstellung von Software in den Prozessen, Werkzeugen, usw., ermöglichen höhere Qualität

Andererseits: Potentiell höhere Fehlerauswirkungen

- Software wird essentieller Bestandteil von immer mehr Lebensbereichen
- Software wird immer umfangreicher (Lines of Code), Projekte werden größer, die Anzahl der Beteiligten und deren lokale Verteilung steigt
- immer kürzere Time-To-Market neuer Software
- immer höhere Frequenz von Änderungen in Betrieb befindlicher Software
- immer größerer Einflussbereich einzelner auf Software basierender Systeme, z.B. Online-Systeme mit Millionen oder sogar über 1 Milliarde Nutzer

Einführung >

Notwendigkeit von Softwaretests (13)

Fehler sind trotz aller Anstrengungen nicht vermeidbar!

- komplexe Systeme sind nicht vollständig durch Menschen erfassbar
 - Vielzahl von Eigenschaften und Kombinationen nicht überschaubar
 - Ursache-Wirkungs-Beziehungen nicht mehr überschaubar
- komplexe Technologien, komplexe fachliche Fragen, viele Beteiligte, ...

Gründe sind unter anderem

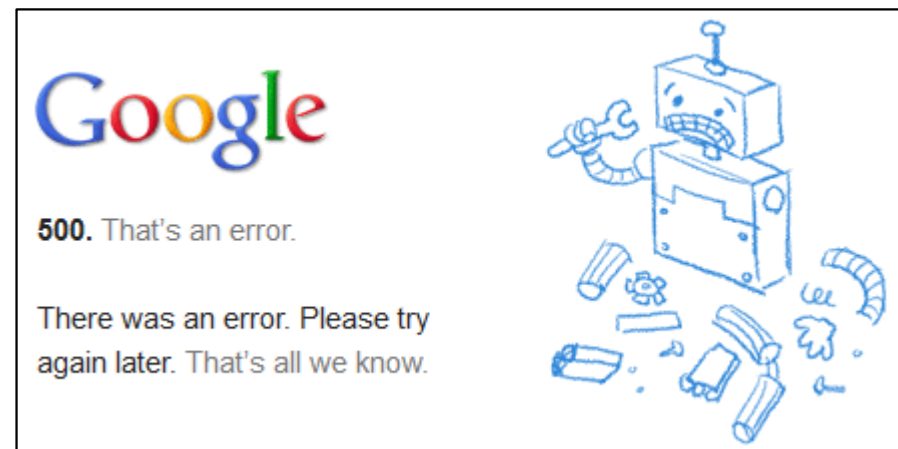
- Kommunikationsprobleme
 - extern, zwischen Menschen
 - intern, bei der Transformation von Sprache in Verständnis
- begrenztes Gedächtnis
- fachliches Problemverständnis

→ Tests sind unbedingt notwendig!

Fehlerbegriff (1)

Fehlerwirkung („failure“)

- auch: „äußerer Fehler“, „Fehlfunktion“
- Wirkung eines Fehlerzustands, die bei der Ausführung des Testobjekts nach außen in Erscheinung tritt
- Abweichung des Ist-Verhaltens vom Soll-Verhalten
- Beispiele:
 - System reagiert nicht mehr auf Eingaben
 - unerwarteter Programmabbruch
 - unter hoher Last sieht ein Nutzer Daten fremder Nutzer
 - fehlende Fehlermeldung
 - falsche Ausgabe



Fehlerbegriff (2)

Fehlerzustand („fault“)

- auch: „innerer Fehler“, „Defekt“
- Ursache für einen äußeren Fehler
- Zustand einer Softwarekomponente, der unter spezifischen Bedingungen eine geforderte Funktion beeinträchtigen kann
- Beispiele:
 - falsche Anweisung im Code
 - fehlende Fehlerbehandlung
 - Race Condition
 - inkorrekte Datendefinition



© Giant Microbes

Fehlerbegriff (3)

Fehlhandlung einer Person („error“)

- menschliche Handlung, die zu einem falschen Ergebnis führt, beispielsweise zu falschem Code
- Beispiele:
 - inkorrektes Verständnis einer Anforderung
 - fehlerhafte Programmierung durch Copy&Paste von Codeteilen
 - Einfügen einer falschen Vergleichsoperation in den Code
 - Entwickler vergisst Fehlerbehandlung
 - Einchecken von Änderungen in einen falschen Code-Zweig



Wechselwirkung von Fehlern

Fehlermaskierung

- Kompensierung eines Fehlerzustands durch einen oder mehrere andere Fehlerzustände
- nach Behebung eines Fehlerzustands wird der andere wirksam

Fehlerzustände blockieren andere fehlerhafte Funktionen

- in der Praxis können Fehlerzustände oft nicht erreicht werden, weil der Weg zum weiteren fehlerhaften Code durch andere Fehlerzustände blockiert ist
- in der Folge arbeitet man sich iterativ vor: Fehlerwirkung finden, Fehlerbehebung, weiterer Code erreichbar, weitere Fehlerwirkung tritt auf, ...

Entfernung zwischen Fehlerzustand und Fehlerwirkung

- eine Fehlerwirkung kann weit entfernt vom Fehlerzustand auftreten
- Beispiele: inkorrekte Pointerarithmetik, Buffer-Overflow, Verfälschung gespeicherter Daten, die später wieder benötigt werden

Sollte man den eigenen Code testen?

JA! Jeder Entwickler sollte im Rahmen des Entwicklertests seinen Code testen, aber:

- man begegnet eigenem Code zu optimistisch, der kritische Abstand fehlt
- unbewusst möchte man die korrekte Funktion zeigen, keine Fehler finden
- Menschen passen ihr Verhalten vorhandenen oder vermuteten Fehlern an
- Blindheit gegenüber eigenen Fehlern:
 - Das eigene Testverhalten folgt der bekannten, eigenen Implementierung.
 - Wer etwas falsch verstanden hat, erkennt das auch nicht beim Test.
 - Wenn jemand seine eigenen Fehler leicht findet, wieso sollte er sie dann überhaupt erst gemacht haben?
- Entwickler finden Testen oft zu aufwendig, es macht ihnen oft weniger Spaß
- Interessenkonflikt zwischen Entwicklungs- und Testaufgaben

→ Der Entwicklertest hat prinzipbedingte Grenzen.

→ Test durch unabhängige Tester ist ebenfalls notwendig!

Einführung >

Stufen der QS-Organisation

Stufen der QS-Organisation von Null bis zur zentralen QS

- 1) Entwickler testen nur ihren eigenen Code, sonst keine QS
- 2) Entwickler testen nur gegenseitig
- 3) dedizierte Tester im Entwicklungsteam
- 4) separates Testteam in der Entwicklungsabteilung
- 5) unabhängige, zentrale QS

Stufen 1) und 2) sind meist mangelhaft.

→ Traditionelle Sicht: Optimal ist ein unabhängiges, zentrales QS-Team (5).

→ Agile Sicht: Optimal sind Tester als Teil des Entwicklungsteams (3).

Vorteile einer zentralen QS

QS-Team als zentraler Pool von

- QS-Wissen (Methodik, Vorgehensweisen,...)
- Test Engineers
- Testressourcen (Hardware, Software-Werkzeuge, Testdaten,...)

Übernahme zentraler Aufgaben

- Pflege von Testdaten, Verwaltung von Testumgebungen
- Weiterentwicklung der QS-Maßnahmen

Unabhängigkeit

- nicht der Entwicklung unterstellt: kein Interessenkonflikt zu Lasten der QS bei Berichterstattung, Fehlerklassifikation, Ressourcenknappheit,...
- unabhängiges Mindset: keine Übernahme von Annahmen und Denkfehlern der Entwicklung, Nutzersicht statt Entwicklersicht
- QS als Mediator zwischen Entwicklung und Produktmanagement

Psychologie beim Test durch Andere

Konfliktpotential

- Menschen machen Fehler, geben sie jedoch ungern zu
- Ziel des Tests ist aber die Aufdeckung und Mitteilung der Fehler



Häufige falsche Denkweisen bei unerfahrenen Entwicklern

- "der Test verursacht Verzögerungen" (da er Zeit benötigt und viele Fehler findet)
- "Entwicklung ist konstruktiv, Test ist destruktiv"
- "Tester freuen sich über meine Fehler, wollen mir Unfähigkeit nachweisen"
- "Entwickeln ist wesentlich anspruchsvoller, als Testen"

Häufige falsche Denkweisen bei unerfahrenen Testern

- Tester freuen sich tatsächlich über Fehler, um Entwickler vorführen zu können
- Tester denken abwertend über Entwickler: „Was haben sie denn hier wieder für einen Unsinn gemacht!“
- „Was hat sich der Auftraggeber nur gedacht?“



Einführung >

Fehlerkultur

- Fehler sind nicht notwendigerweise ein Zeichen von Schwäche, sondern systemimmanent – "Errare humanum est"
- Fehler werden akzeptiert und genutzt, um die Qualität zu verbessern
- jeder gefundene Fehler ist besser, als ein nicht gefundener
- Testdesign und Test sind hochwertige, kreative Tätigkeiten
- Entwickler und Tester akzeptieren sich gegenseitig und kooperieren
- Test entlastet die Entwicklung
- Entwickler freuen sich über Feedback und Diskussionspartner
- Tester sind nicht schadenfroh und überheblich
- Entwickler akzeptieren auch Fehler hinsichtlich impliziter Anforderungen

→ Konstruktive Test- und Fehlerkultur schaffen und leben!
→ Im Zweifelsfall hilft Kommunikation!

Einführung >

Was ist ein Test? (1)

IEEE 610 “Standard Glossary of Software Engineering Terminology“, 1990:

“Der Prozess des Betriebs eines Systems oder einer Komponente unter spezifizierten Bedingungen, unter Beobachtung oder Aufzeichnung der Ergebnisse und mit einer Evaluierung einiger Aspekte des Systems oder der Komponente.”

Glenford Myers, “The Art of Software Testing“, 1979:

“Testen ist der Prozess der Ausführung eines Programms oder Systems mit der Absicht, Fehler zu finden.”

Einführung >

Was ist ein Test? (2)

IEEE 829, “Standard for Software and System Test Documentation“:

“Eine Menge von einem oder mehreren Testfällen.“

Martin Pol, “Management und Optimierung des Testprozesses“, 2002:

„Prozess des Planens, der Vorbereitung und der Messung, mit dem Ziel, die Eigenschaften eines IT-Systems festzustellen und den Unterschied zwischen dem tatsächlichen und dem erforderlichen Zustand aufzuzeigen. Testen reduziert das Maß an Unsicherheit in Bezug auf die Qualität eines IT-Systems.“

→ Wir verwenden den Begriff meist in diesem weiteren Sinne, zum Beispiel in „Teil I: Grundlagen des Softwaretests“.

Einführung >

Was ist ein Test? (3)

Eigenschaften eines Tests

- Planung, Entwurf und Ausführung von einem oder mehreren Testfällen
- berücksichtigt
 - explizite Anforderungen
 - implizite Anforderungen
- beweist oder widerlegt Produkteigenschaften bzw. Qualitätsmerkmale

Ziele des Testens

- Feststellung der Unterschiede zwischen Soll-Vorgaben und Ist-Zustand
- gezieltes und systematisches Aufdecken von Fehlerwirkungen

- ➔ Ziel ist es, Fehler zu finden, nicht die Korrektheit zu beweisen!
- ➔ Nur ein Test, der möglichst viele vorhandene Fehler findet, ist erfolgreich!

Einführung >

Explizite und implizite Anforderungen

Explizite Anforderungen

- durch Dokumente erklärte Anforderungen
 - funktionale Anforderungen
 - nichtfunktionale Anforderungen
- Beispiele: Anforderungsdokumente, Fachkonzepte, Business Blueprint, SLA

Implizite Anforderungen

- offensichtliche und selbstverständliche Anforderungen, die sich aufgrund von Schlussfolgerungen ergeben, so dass sie nicht in einem Dokument festgehalten sind
- berechnete Erwartungen

→ Explizite und implizite Anforderungen sind zu testen!

Einführung >

Testorakel

- Informationsquelle zur Ermittlung der Soll-Ergebnisse eines Testfalls
- sollte verbindlichen Charakter besitzen
- Beispiele:
 - Anforderungsdefinition
 - zwischen Anforderer und Implementierer abgestimmtes Fachfeinkonzept
 - Change Requests (aktueller Stand oft nur aus sequentieller Anwendung von ursprünglichen Dokumenten + Change Requests ableitbar)
 - User Stories mit Akzeptanzkriterien in einem agilen Prozess
 - Anwenderhandbuch
 - Spezialwissen einer Person
 - Referenzsystem, dessen Funktionen durch das zu testende System exakt nachgebildet werden sollen
- sollte im Testkonzept festgelegt sein
- Testorakel darf nicht der Code oder das zu testende System selbst sein!

Einführung >

Eigenschaften eines guten Tests

Folgende Eigenschaften kennzeichnen einen guten Test

- hohe Testabdeckung bei minimalem Aufwand (hohe Effizienz)
- findet möglichst viele der vorhandenen Fehler
- überprüft alle expliziten Anforderungen
- überprüft auch implizite Anforderungen
- überprüft alle vereinbarten Qualitätsmerkmale
- gut dokumentiert, beispielsweise im Testmanagementwerkzeug
- eindeutig und wiederholbar

Einführung >

Was ist Softwarequalität?

Helmut Balzert: Lehrbuch der Softwaretechnik, 1998:

„Unter Softwarequalität versteht man die Gesamtheit der *Merkmale und Merkmalswerte* eines Softwareprodukts, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen (Ist/Soll).“

Für die Praxis ist eine Konkretisierung erforderlich:

- ➔ ISO/IES 25010 legt konkrete Qualitätsmerkmale für Software fest (löst ISO 9126 ab)
- ➔ ISO/IES 25010 unterteilt in Qualitätsmodelle für
 - Produktqualität
 - Qualität in der Benutzung
- ➔ Wir beziehen uns auf die Produktqualität.

Einführung >

Qualitätsmerkmale (ISO/IEC 25010) (1)

Funktionale Eignung

Wie entsprechen die Funktionen den expliziten und impliziten Anforderungen?

- Funktionale Vollständigkeit: Grad, in dem der Funktionsumfang alle spezifizierten Aufgaben und Benutzerziele abdeckt
- Funktionale Korrektheit: Grad, in dem die richtigen Ergebnisse mit der erforderlichen Präzision geliefert werden
- Funktionale Angemessenheit: Grad, in dem die Funktionen die Erfüllung bestimmter Aufgaben und Ziele erleichtern



Einführung >

Qualitätsmerkmale (ISO/IEC 25010) (2)

Leistungseffizienz

Welche Leistung erbringt das System im Verhältnis zur Menge der verbrauchten Ressourcen?

- Zeitverhalten: Zeit- und Antwortverhalten, Durchsatz
- Ressourcenverbrauch: Umgang mit Betriebsmitteln, Ressourcenbedarf
- Kapazität: Höchstgrenzen von Parametern, z.B. Anzahl speicherbarer Elemente, gleichzeitige Benutzer, Kommunikationsbandbreite



Einführung >

Qualitätsmerkmale (ISO/IEC 25010) (3)

Kompatibilität

Wie gut kann eine Software Informationen mit anderer Software austauschen *oder* die erforderlichen Funktionen erfüllen, während sie dieselbe Hardware- oder Softwareumgebung gemeinsam nutzen?

- Interoperabilität: Grad, in dem Systeme, Produkte oder Komponenten Informationen austauschen und diese Informationen nutzen können
- Koexistenz: Erfüllung der erforderlichen Funktionen ohne schädliche Auswirkungen auf ein anderes Produkt



Qualitätsmerkmale (ISO/IEC 25010) (4)

Benutzbarkeit

Wie gut kann ein System von bestimmten Nutzern verwendet werden, um bestimmte Ziele mit Effektivität, Effizienz und Zufriedenheit zu erreichen?

- Angemessenheitserkennbarkeit: Erkennbarkeit, ob die Software für den beabsichtigten Zweck/die Bedürfnisse geeignet ist
- Erlernbarkeit: Unterstützung des Benutzers, die effektive, effiziente, risikofreie und zufriedene Nutzung zu erlernen
- Bedienbarkeit: einfach bedienbar und steuerbar, dabei fehlertolerant und erwartungskonform
- Fehlertoleranz gegenüber Anwenderfehlern: Grad, in dem das System den Benutzer vor Fehlern schützt und bei der Korrektur unterstützt



Einführung >

Qualitätsmerkmale (ISO/IEC 25010) (5)

Benutzbarkeit (fortgesetzt)

- Ästhetik der Benutzeroberfläche: Grad, in dem die Benutzeroberfläche eine angenehme und zufriedenstellende Interaktion ermöglicht
- Barrierefreiheit: leichter Zugang für Menschen mit den unterschiedlichsten Eigenschaften und Fähigkeiten; auch bei alterbedingten Behinderungen

(siehe auch DIN EN ISO 9241, Teil 110, „Grundsätze der Dialoggestaltung“)



Qualitätsmerkmale (ISO/IEC 25010) (6)

Zuverlässigkeit

Grad, in dem ein System für einen bestimmten Zeitraum bestimmte Funktionen unter bestimmten Bedingungen ausführt

- Reife: Grad, in dem ein System die Anforderungen im Normalbetrieb zuverlässig erfüllt
- Verfügbarkeit: Grad, in dem ein System betriebsbereit und zugänglich ist, wenn es gebraucht wird
- Fehlertoleranz: Grad des bestimmungsgemäßen Betriebs trotz vorhandener Hardware- oder Softwarefehler oder falscher Eingabewerte an Schnittstellen
- Wiederherstellbarkeit: Grad, bis zu dem im Falle einer Unterbrechung oder eines Ausfalls betroffene Daten und der gewünschte Systemzustand wiederhergestellt werden können



Qualitätsmerkmale (ISO/IEC 25010) (7)

Sicherheit

Grad, in dem Informationen geschützt werden, so dass Personen oder andere Systeme den Grad des Datenzugriffs entsprechend ihres Typs und ihres Berechtigungsniveaus haben

- Vertraulichkeit: Daten sind nur den Zugriffsberechtigten zugänglich
- Integrität: kein unbefugter Zugriff auf und keine Veränderung von Programmen oder Daten
- Nachweisbarkeit: Nachweis, dass Aktionen oder Ereignisse stattgefunden haben, so dass sie später nicht mehr zurückgewiesen werden können
- Zurechenbarkeit: eindeutige Zurückführbarkeit von Aktionen einer Entität auf die Entität
- Authentizität: Nachweis der Identität eines Subjekts oder einer Ressource



Einführung >

Qualitätsmerkmale (ISO/IEC 25010) (8)

Wartbarkeit

Effektivität und Effizienz, mit der ein System durch die vorgesehenen Instandhalter modifiziert und aktualisiert werden kann

- Modularität: modularer Aufbau, so dass eine Änderung an einer Komponente nur minimale Auswirkungen auf andere Komponenten hat
- Wiederverwendbarkeit: Grad der Verwendbarkeit in mehr als einem System
- Analysierbarkeit: Beurteilbarkeit der Auswirkungen einer beabsichtigten Änderung auf andere Teile; einfache Bestimmbarkeit von Mangelursachen und änderungsbedürftigen Teilen
- Modifizierbarkeit: effektive und effiziente Modifizierbarkeit, ohne dass Fehler auftreten oder die bestehende Qualität beeinträchtigt wird
- Prüfbarkeit: Effektivität und Effizienz, mit der Prüfkriterien festgelegt und deren Erfüllung durch Tests bestimmt werden können



Einführung >

Qualitätsmerkmale (ISO/IEC 25010) (9)

Übertragbarkeit

Effektivität und Effizienz, mit der ein System, Produkt oder eine Komponente von einer Hardware-, Software- oder anderen Betriebs- oder Nutzungsumgebung auf eine andere übertragen werden kann

- Anpassbarkeit: Grad der effektiven und effizienten Anpassbarkeit an unterschiedliche oder sich entwickelnde Hardware-, Software- oder andere Betriebs- oder Nutzungsumgebungen
- Installierbarkeit: Effektivität und Effizienz, mit der ein Produkt oder System erfolgreich installiert und deinstalliert werden kann
- Austauschbarkeit: Grad, in dem ein Produkt ein anderes für denselben Zweck spezifiziertes Softwareprodukt in derselben Umgebung ersetzen kann



Einführung >

Qualitätsmerkmale (ISO/IEC 25010) (10)

Dokumentation (kein Merkmal nach ISO/IEC 25010)

- Vollständigkeit
- Übersichtlichkeit
- Verständlichkeit
- Strukturiertheit
- Korrektheit
- Editierbarkeit
- Nachvollziehbarkeit
- Integrität/Authentizität (z. B. Änderungshistorie)
- Objektivität



Agenda

- Einführung
- Klassifikation von QS-Maßnahmen
- Testfallerstellung
- Ökonomie des Testens
- Systematische Testfallermittlung
- Testausführung
- Fehlerverfolgung
- Kurzüberblick Testmanagement

Klassifikation nach Testarten

Testarten kennzeichnen die Vorgehensweise, meist passend zu einem oder mehreren Qualitätsmerkmalen, z.B.:

- manueller Funktionstest (Funktionale Eignung, Kompatibilität, ...)
- automatisierter Funktionstest (Funktionale Eignung, Kompatibilität, ...)
 - automatisierter Modultest/Unit-Test
 - automatisierter Integrationstest
 - automatisierter End-Zu-End-Test, z.B. über die Benutzeroberfläche
- Usability-Analyse (Benutzbarkeit)
 - Expertenevaluation
 - Test mit Probanden
- Security-Test, anwendungsseitig oder Penetrationstest (Sicherheit)
- Last- und Performance-Test (Leistungseffizienz)
- manueller Code Review (Wartbarkeit, Übertragbarkeit, Sicherheit, ...)
- werkzeuggestützte Code-Analyse, auch „Code Quality Management, CQM“

Klassifikation von QS-Maßnahmen >

Konstruktiv versus Analytisch

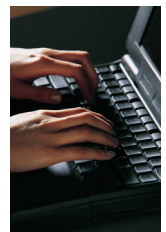
Konstruktive QS-Maßnahmen

- Entstehen von Fehlern schon *bei der Erstellung* der Software verhindern
- „Einbau von Qualität in das Produkt“
- gute Anforderungs- und Entwicklungsprozesse, Projektmanagement
- moderne Architekturen, Programmiersprachen, -methoden, -werkzeuge, Pair Programming, TDD, automatisierte Tests, Continuous Integration
- Defensives Programmieren gegen Eingaben von außen (Nutzereingaben, Daten an Schnittstellen)
- Nutzung von Standards, Richtlinien (Code-Conventions, ...) und Checklisten
- Schulungen, Dokumentation



Analytische QS-Maßnahmen

- Erkennung von Fehlern *nach Erstellung* der Software
- dynamische Tests
- Reviews (Code, Architektur, Dokumente)



Statisch versus Dynamisch (1)

Statische QS-Maßnahmen

- Analyse des Testobjektes ohne dessen Ausführung
- manuelle Reviews
- allgemeine automatisierte Code-Analyse
 - Prüfung der Code- und Architekturqualität mit zahlreichen Metriken
 - projektspezifische Konfiguration notwendig
 - erzeugt oft große Mengen an Warnungen, die bewertet werden müssen
 - Werkzeuge: Checkstyle, FindBugs, PMD, Sonarlint, jQAssistant, ...
- spezifische automatisierte Code-Analyse, zum Beispiel auf
 - Probleme mit Nebenläufigkeit (ThreadSafe, ...)
 - Security-Probleme (AttackFlow, Coverity, Xanitizer, ...)
 - Memory-Leaks (Coverity, ...)

Klassifikation von QS-Maßnahmen >

Statisch versus Dynamisch (2)

Dynamische QS-Maßnahmen

- Ausführen des Testobjektes und Prüfung des Verhaltens
- benötigt oft Testumgebung mit weiteren Komponenten oder Simulatoren
- wichtigste Art der QS
- die meisten Testarten ordnen sich hier ein, zum Beispiel:
 - manueller und automatisierter Funktionstest
 - Lasttest
 - Usability-Test

Klassifikation von QS-Maßnahmen >

Black-Box versus White-Box

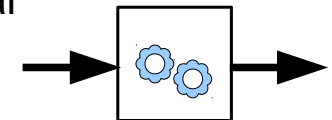
Black-Box-Verfahren

- Test ohne Kenntnis der inneren Funktionsweise
- primär auf Grundlage der Anforderungen
- Steuerung des Ablaufs nur durch entsprechende Eingaben möglich
- beobachtet nur das von außen sichtbare Verhalten
- aufgrund der hohen Anzahl an Kombinationen systematische Auswahl nötig



White-Box-Verfahren

- Test mit Kenntnis der inneren Funktionsweise, auf Grundlage des Codes
- während der Ausführung der Testfälle wird der innere Ablauf analysiert, z.B. zur Ermittlung der Testabdeckung; dazu Code instrumentierbar
- nur im Code umgesetzte Anforderungen werden geprüft
- Wissen über die Programmstruktur zur Testfallerstellung nutzbar
- vor allem für untere Teststufen geeignet, nicht für komplexe Systemtests

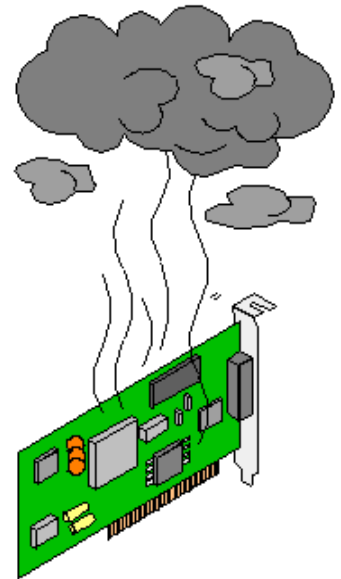


Klassifikation von QS-Maßnahmen >

Klassifikation nach Testzweck (1)

Smoke Test

- beim Klempnern Suche von Lecks mit echtem Rauch; in der Elektronik erster Anschluss an eine Spannungsquelle
- grundlegender Probelauf, zum Beispiel um festzulegen, ob die Software für weitere Tests angenommen wird
- auch: „Build Verification Test“
- breiter Test mit geringer Testabdeckung
- typischer Testaufwand bei manueller Ausführung: 1h..3h



Featuretest

- Test neuer/geänderter Features auf die vereinbarten Qualitätsmerkmale
- Umfang sehr unterschiedlich
- Testaufwand je nach Projekt etwa 15%-50% des Entwicklungsaufwands, durchschnittlich etwa 30% des Entwicklungsaufwands zusätzlich für Test

Klassifikation nach Testzweck (2)

Fehlernachtest (Re-Test)

- Wiederholung fehlgeschlagener Testfälle nach der Fehlerkorrektur
- dient der Überprüfung auf korrekte Fehlerbehebung
- sehr wichtig, da
 - Fehler oft nicht hinreichend behoben sind
 - Entwickler den Test oft nicht im nötigen Umfang durchführen können

Regressionstest

- erneuter Test bereits getesteter Komponenten nach Modifikationen, um nachzuweisen, dass keine Fehler eingebaut oder freigelegt wurden
- oft am Ende einer längeren Test- und Fehlerbehebungsphase
- Testumfang variiert von einer breit gestreuten Auswahl vorhandener Testfälle bis zur kompletten Neuausführung aller Testfälle

Klassifikation nach Teststufen (1)

Definition „Teststufe“

- Gruppe gemeinsam verwalteter und ausgeführter Testaktivitäten
- Die Einteilung in diese Gruppen ist organisatorisch und technisch begründet, zum Beispiel durch:
 - Integrationsgrad der einzelnen Komponenten
 - Einordnung im angewendeten Vorgehensmodell
 - Testziele
 - damit auch Verantwortlichkeiten, Testumgebung, Methoden und Werkzeuge
- Einteilung und Benennung der Teststufen oft nicht einheitlich
- Durchführung meist zeitlich nacheinander

Klassifikation von QS-Maßnahmen >

Klassifikation nach Teststufen (2)

Typische Teststufen

- Entwicklertest
- Komponententest
- Integrationstest
- Systemtest
- Abnahmetest



Klassifikation von QS-Maßnahmen >

Teststufe Entwicklertest

Erster Test durch Entwickler

- während der Entwicklung einer Komponente oder eines Systems
- oft in der eigenen, lokalen Entwicklungsumgebung durchgeführt
- meist informell
- heute oft durch automatisierte Unit-Tests

Grenzen

- Denkfehler bei der Entwicklung macht man auch beim Test
- nur implementierte Anforderungen werden auch getestet
- inkorrekte Anforderungen nicht erkannt
- inkorrekte Annahmen nicht erkannt
- lokal begrenzt, z.B. Schnittstellen oft nicht berücksichtigt



Klassifikation von QS-Maßnahmen >

Teststufe Komponententest

- erster unabhängiger Test einer isolierten Komponente
- Komponente = kleinste einzeln testbare Softwareeinheit
- betrachtet Testobjekt in Isolation, um Einflüsse anderer Komponenten auszuschließen und Fehlerzustände besser eingrenzen zu können
- die Granularität der Komponente kann sehr unterschiedlich sein
- ein Schnittstellentest einer isolierten Komponente, beispielsweise durch einen automatisierten Test, ist ein Komponententest
- erfolgt im Idealfall bereits durch dedizierte Tester, nicht durch Entwickler
- oft mit dem Entwicklertest gleichgesetzt



Klassifikation von QS-Maßnahmen >

Teststufe Integrationstest

- Test des Zusammenspiels von Komponenten und ihrer Schnittstellen
 - *Komponentenintegrationstest (Integrationstest "im Kleinen")*:
Test von Schnittstellen und Zusammenwirken mehrerer Komponenten
 - *Systemintegrationstest (Integrationstest "im Großen")*:
Test der Integration von ganzen Systemen; Test der Schnittstellen zu einer externen Organisation
- ein Schnittstellentest unter Nutzung der beteiligten Komponenten oder Systeme ist ein Integrationstest
- die Fehlerzustände hier gefundener Fehlerwirkungen sind oft schwer zu finden und schwer zu beheben



Klassifikation von QS-Maßnahmen >

Teststufe Systemtest

- Test des komplett integrierten Systems / des fertigen Gesamtprodukts
- Überprüfung des Gesamtsystems auf Erfüllung der Anforderungen
- möglichst realistische Hardware- und Softwareumgebung; möglichst nah am Produktivbetrieb
- Test durchgängiger Anwendungsfälle über alle Teilsysteme



Klassifikation von QS-Maßnahmen >

Teststufe Abnahmetest

- formaler Test aus Kunden- und Benutzersicht hinsichtlich der realen Benutzeranforderungen und Geschäftsprozesse
- auch: „Akzeptanztest“
- Prüfung der Aspekte:
 - vertragliche Akzeptanz
 - Benutzerakzeptanz
- letzter Test vor der formellen Übergabe der Software an den Kunden
- Entscheidungsgrundlage für den Auftraggeber oder eine bevollmächtigte Instanz, ob ein System offiziell abzunehmen ist oder nicht



Agenda

- Einführung
- Klassifikation von QS-Maßnahmen
- Testfallerstellung
- Ökonomie des Testens
- Systematische Testfallermittlung
- Testausführung
- Fehlerverfolgung
- Kurzüberblick Testmanagement

Testfallerstellung > **Testobjekt**

Begriff „Testobjekt“

- bestimmt, was zu testen ist = „*Testgegenstand*“, „*Testbereich*“, „*Testgebiet*“
- bezeichnet je nach Granularität
 - eine fachliche Funktion oder einen zu testenden Aspekt
 - eine Komponente, ein Teilsystem oder das zu testende System
- meist durch hierarchische Baumstruktur weiter gegliedert
- in Testtools durch Ordner oder Testsuiten abgebildet
- Definition der Testobjekte auf Basis der Anforderungen ist Voraussetzung für die Testfallerstellung

Beispiele für Testobjekte

- E-Mail-Funktionalität
- Anmeldedialog
- Logging
- anwendungsseitige Security



Vorgehen bei der Testfallerstellung (1)

Voraussetzung

- Definition der Testobjekt-Hierarchie
- Priorisierung der Testobjekte, wenn nötig

Vorgehen

- Auswahl eines Testobjekts, eventuell nach Priorität
- Sammeln von Informationen über das Testobjekt
 - Soll-Vorgaben (Testorakel)
 - sonstige Informationen (unverbindlich, zum Verständnis)
- Berücksichtigung der zu testenden Qualitätsmerkmale und der Teststufe
- Auswahl der passenden QS-Maßnahmen - Wie ist zu testen?
 - Testarten
 - statisch oder dynamisch
 - manuell oder automatisiert...

Testfallerstellung >

Vorgehen bei der Testfallerstellung (2)

Vorgehen (fortgesetzt)

- Festlegung der Testabdeckung (gewünschte Testtiefe)
- Sammeln von Testideen (Überschriften der Testfälle)
- Erstellung der einzelnen Testfälle mit ihren
 - notwendigen Vorbedingungen
 - benötigten Testdaten
 - Testschritten
- Review der Testfälle (wünschenswert)
- erste Ausführung und dabei Anpassung der Testfälle

→ Testfälle müssen meist bei der ersten Ausführung angepasst werden!

→ Endgültige Testfallerstellung erfolgt am besten am lauffähigen Testobjekt!

Testfallerstellung > **Testfall (1)**

Ein Testfall umfasst folgende Informationen

- Name oder Kurzbezeichnung, aus der die Intention des Testfalls hervorgeht, beispielsweise „Suchphrase mit Leerzeichen“
- Vorbedingungen
- auszuführende Schritte (eindeutige Prüfanweisung), dabei pro Schritt:
 - kurzer Name oder Nummer
 - auszuführende Aktionen und Eingaben
 - erwartetes Ergebnis

Optional am Testfall

- Attachments
- Keywords
- Verweise auf Anforderungen
- Verweise auf gefundene Fehler im Fehlerverwaltungstool

Testfallerstellung > **Testfall (2)**

Ein Testfall sollte folgende Eigenschaften besitzen

- zusammenhängend
- in sich abgeschlossen
- atomar
- eindeutig
- prüfbares Ergebnis

Testfallerstellung >

Testfall – Beispiel im Mercury TestDirector

Mercury TestDirector 8.0 SP2 - Microsoft Internet Explorer

Datei Bearbeiten Ansicht Favoriten Extras ?

Zurück Suchen Favoriten Medien

Adresse http://www.de/tdbin/start_a.htm

TestDirector 8.0

Project : e-Telco Wartung [rvo2] REQUIREMENTS TEST PLAN TEST LAB DEFECTS TOOLS HELP LOGOUT

Planning View Analysis

Test Plan Tree

- Portal
 - Service-Bereich
 - SSI
 - PK
 - PK, GK
 - Abmeldung
 - Anmeldung
 - T01 - Anmeldung L3, nicht
 - T02F - Anmeldung L3, nicht
 - T03 - Anmeldung L3, L4-
 - T04 - Anmeldung L3, ohne
 - T05F - Anmeldung L3, falsch
 - T06 - Anmeldung L3, gleich
 - T07 - Anmeldung L3, für
 - T08 - Anmeldung L4
 - T09 - Anmeldung L4, ohne
 - T10F - Anmeldung L4, falsch
 - T11 - Anmeldung L4, gleich
 - T12 - Anmeldung L4, für
 - T13 - Anmeldung L4, Sek
 - T14 - Anmeldung L3 und

| Step Name | Description | Expected Result |
|--------------------|--|---|
| Vorbedingungen | Nutzer mit Festnetzanschluss, Nutzer nicht für L4 registriert (im Simulator Flag "noLevel4User" gesetzt), nicht angemeldet | Es ist darauf zu achten, dass der Austausch sensibler Daten nur über HTTPS und nicht über HTTP erfolgt. |
| L3-Dienst | Navigation zu "Bestellstatus" | Anmeldedialoge für registrierte Nutzer (L4) mit Eingabefeldern
- Benutzername
- Passwort
und nicht registrierte Nutzer (L3) mit Eingabefeldern
- Vorwahl
- Rufnummer
- Kundennummer |
| Anmeldung L3 | Eingabe der gültigen Werte des Nutzers für
- Vorwahl
- Rufnummer
- Kundennummer
und klicken von "Ok". | Anmeldung erfolgreich. Im Navigationsbereich werden jetzt zusätzlich angezeigt:
- Vorname
- Name
- Kundennummer
- Link "Logout" |
| weiterer L3-Dienst | Navigation zu ... "Mein Anschluss" | Die Seite wird angezeigt, ohne dass ein |

Total Steps: 4 Server Time: 08:46 AM 09.08.05

Fertig Internet

Testfallerstellung >

Testfall – Beispiel in TestLink

The screenshot displays the TestLink 1.7.4 web interface. The top navigation bar includes links for 'Startseite', 'Definition', 'Ausführen', 'Ergebnisse', 'Benutzerverwaltung', 'Meine Einstellungen ändern', and 'Testfall ID:'. The left sidebar shows a tree view of test suites, with 'Multimessenger (898)' expanded to show sub-items like 'Multimessenger-Client (632)', 'SN Messenger (12)', 'Fleximail (173)', 'Import von Adressbüchern anderer Dienste (44)', 'eBay Alerts (24)', 'Vorbedingungen - Zielgruppe (0)', and 'Kernfunktionen (20)'. The main content area is titled 'ID 14900 :: Testfall Deaktivieren des eBay-Dienstes' and contains the following sections:

Navigationen Filter & Einstellungen

Testsuite:
Menü nach jeder Änderung neu laden ☒
Menü aktualisieren

Buttons: Bearbeiten, Löschen, Verschieben / Kopieren, Version deaktivieren, Neue Version erstellen, Export

Version 1

Zusammenfassung

Der Nutzer muss in der Lage sein, den Dienst temporär zu deaktivieren. Hierzu wurde ein neuer Dialog auf der Seite "Einstellungen" hinzugefügt.

Schritte

- Gehe auf die Seite "Einstellungen" und klicke links auf den Menüpunkt "Info-Dienste"
- Es öffnet sich der Einstellungsdialog "Info-Dienste"
- Selektiere einen vorhandenen Ebay-Account und klicke auf den Button "Deaktivieren"

Erwartete Ergebnisse

- Auf der Einstellungsseite erscheint ein rotes Symbol für dieses Ebay-Konto.
- Der Button "Aktivieren" ist verfügbar.
- Der entsprechende Ebay-Account wird im Roster nicht mehr angezeigt.

Stichworte: Port.998

Erstellt 20.10.2008 13:00:21 von FKurz
Letzte Änderung 28.11.2008 16:24:58 von FKurz




Testfallerstellung >

Testsuite – Beispiel in TestRail (1)

[Return to Dashboard & Projects](#) [Feedback](#) [My Settings](#) [Help](#) [Logout](#)

XLT

[Overview](#) [Todo](#) [Milestones](#) [Test Runs & Results](#) [Test Suites & Cases](#) [Reports](#)

S536 Script Developer    Run Report

Test Suites & Cases » Script Developer

Record and Edit

| ID | Title |
|-------|------------------|
| C9147 | Record Toolbar |
| C9149 | Line Numbers |
| C9154 | Actions |
| C9155 | Editing Scripts |
| C9156 | Editing Commands |

Element Identification Strategies

| ID | Title |
|-------|------------------------------------|
| C9138 | Identification via XPath (Default) |
| C9139 | Identification via ID |
| C9140 | Identification via Name |
| C9141 | Identification via Link Text |
| C9142 | Identification via DOM Expression |
| C9143 | Identification via Option Label |
| C9144 | Identification via Value |

Contains 12 sections and 47 cases.

Sections & Cases


Test Runs


Groups

- Record and Edit
 - Element Identification Strate
 - Attribute Filtering
 - Assertions
- Manage Test Data
- Modules
- Manage Objects
- Manage Project
- Export
- Replay
- Migration
- Command Tests



Testfallerstellung >

Testfall – Beispiel in TestRail (2)

[Return to Dashboard & Projects](#)  [Feedback](#) | [My Settings](#) | [Help](#) | [Logout](#)

XLT 

[Overview](#) | [Todo](#) | [Milestones](#) | [Test Runs & Results](#) | **[Test Suites & Cases](#)** | [Reports](#)

C9263 Delete Suite in Script Developer  

[Test Suites & Cases](#) » [Script Developer](#) » [Manage Objects](#) » [Delete Suite in Script Developer](#)

| Type | Priority | Estimate | Milestone |
|------------------|---------------|----------|-----------|
| Functionality | unprioritized | None | None |
| Smoketest | | | |
| Yes | | | |

Preconditions

Different test suites were opened in script developer.

Test Steps

- 1 The drop-down box above the project view contains all loaded test suites.
- 2 It's possible to mark a test suite with the help of the arrow keys and press DEL to delete the suite from the script developer.
- 3 Confirm that the test suite is not deleted from the file system.

In section [Manage Objects](#).

Details

Test Results

History

Attachments

None.

Granularität von Testfällen

Testfälle grundsätzlich klein und atomar gestalten!

- Das erleichtert:
 - statistische Auswertung
 - Re-Test
 - Verwaltung
 - Wartung
 - Wiederverwendung
 - Automatisierung
- Bei mehreren zu testenden Wegen zum Ziel getrennte Testfälle erstellen!
- Schritte mit Alternativen generell vermeiden! Nicht: "A oder B ausführen"
- Idee:
 - jeweils ein Testfall für einen durchgängigen Geschäftsfall, z.B. Bestellung
 - ➔ schneller Überblick, gut nutzbar für Smoketest und Regressionstest
 - viele Testfälle für detaillierte Einzelaspekte

Vorbedingungen

Was sind Vorbedingungen?

- benötigter, spezieller Zustand vor der Ausführung der Testschritte
- wenn die Herstellung eines benötigten Zustands nicht selbst Testobjekt ist, wird sie als Vorbedingung notiert
- keine Bedingungen, die sowieso immer durch das zu testende System erfüllt oder selbstverständlich sind

Beispiele:

- „ein Nutzeraccount mit der gleichen E-Mail-Adresse existiert bereits“
- „das System befindet sich im Zustand der Happy Hour“
- „das Billing-System XYZ ist deaktiviert“
- „es befinden sich mindestens 1 Million Produktdatensätze in der Datenbank“

Nicht:

- „der Server läuft und nimmt Requests entgegen“
- „die Anwendung wurde erfolgreich gestartet“

Testschritte

Beschreibung der Testschritte - Faustregel

- so genau, dass ein Teammitglied die Testfälle ohne Rückfrage ausführen kann
- so allgemein wie möglich, so spezifisch wie nötig

Erkennbar sein müssen

- *Intention des Testfalls* (wenn nicht komplett aus dem Testfallnamen erkennbar)
- *auszuführende Aktionen*
- *Eigenschaften der benötigten Testdaten*

Beispiele für die Angabe von Testdaten

- „Nutzer mit zwei Postfächern, z.B. Nutzer 0815“ (Nicht nur „Nutzer 0815“!)
- „Produkt mit monatlichen Kosten, z.B. Produkt-ID 4711“

Erwartete Ergebnisse

Erwartete Ergebnisse ausreichend detailliert beschreiben:

- nicht nur: "Warenkorb aktualisiert"
- sondern: "Preis, Menge, Name und Produktbild im Warenkorb aktualisiert"



Für jeden Testschritt alle relevanten Ergebnisse angeben, auch wenn sie nicht direkt sichtbar sind, zum Beispiel:

- direkt sichtbare Änderungen im Webbrowser
- Nutzer erhält Bestätigungs-E-Mail
- Buchung im System angelegt
- Log-Eintrag

Konzentration auf das Testobjekt

Der Fokus eines Testfalls liegt auf dem jeweiligen Testobjekt

- mitbenutzte Funktionen, die nicht selbst Testobjekt sind, müssen nicht jedes Mal mit getestet werden → Notation als Vorbedingung
- Beispiel Login-Funktionalität:
 - ist sie Testobjekt: Testfallerstellung für Normal-, Sonder- und Fehlerfälle
 - ist sie nicht Testobjekt: „Vorbedingung: Nutzer ist angemeldet“
- man kann auch vorbereitende Aktionen in der Vorbedingung zusammenfassen, die ein Stück Weg bis zum eigentlichen Testobjekt beschreiben, z.B.:
 - Vorbedingung: Navigation zu "Shop" => "Technik" => "Audio/Video"
 - Schritt 1: <Test des Testobjekts>
 - ...

→ Auszuführende Schritte im Testfall testen das Testobjekt.
→ Sonstige mitbenutzte Funktionen werden als Vorbedingungen notiert.

Testfallerstellung >

Normalfall, Sonderfälle, Fehlerfälle (1)

Normalfall

- einfache, offensichtliche „Standard“-Nutzung des Testobjekts
- auch „Happy Path“ (manchmal auch „Positivfall“ genannt)
- oft in expliziten Anforderungen beschrieben

Sonderfälle

- Eingaben, Daten und Abläufe, die eine spezielle Behandlung erfordern
- die gewünschte Funktion kann trotzdem erfolgreich ausgeführt werden
- oft nicht hinreichend in expliziten Anforderungen beschrieben

Fehlerfälle

- Behandlung von Eingabe-, Schnittstellen- und anderen Fehlern
- oft nicht hinreichend in expliziten Anforderungen beschrieben
- (manchmal auch „Negativfälle“ genannt)

Testfallerstellung >

Normalfall, Sonderfälle, Fehlerfälle (2)

Häufige Situation für ein Testobjekt:

- wenige Normalfälle
- viele Sonderfälle und Fehlerfälle

Konsequenz für die Entwicklung:

- viel Code für die Behandlung von Sonder- und Fehlerfällen

Konsequenz für den Test:

- hoher Testaufwand für Sonder- und Fehlerfälle

Testfallerstellung >

Normalfall, Sonderfälle, Fehlerfälle (3)

Beispiel „Division von Gleitkomma-Zahlen“

- Testobjekt: Dialog zur Division von Gleitkomma-Zahlen
 - Parameter sind Dividend und Divisor als Zeichenketten
 - Rückgabewert ist der Quotient ($= \text{Dividend} / \text{Divisor}$)

- Aufgabe:
 - Welche Normalfälle gibt es?
 - Welche Sonderfälle?
 - Welche Fehlerfälle?

Gleitkomma-Division

Dividend:

Divisor:

Quotient:

Testfallerstellung >

Normalfall, Sonderfälle, Fehlerfälle (4)

Beispiel „Division von Gleitkomma-Zahlen“ (fortgesetzt)

■ Normalfälle:

- gültige Operanden mit gültigem erwarteten Ergebnis
- bei Betrachtung von Vorzeichen als Normalfall, auch die Kombinationen
 - positiver Dividend, positiver Divisor
 - negativer Dividend, positiver Divisor
 - positiver Dividend, negativer Divisor
 - negativer Dividend, negativer Divisor
- Rücksetzen

Gleitkomma-Division

Dividend:

Divisor:

Quotient:

Testfallerstellung >

Normalfall, Sonderfälle, Fehlerfälle (5)

Beispiel „Division von Gleitkomma-Zahlen“ (fortgesetzt)

■ Sonderfälle:

- Division durch 1 (Quotient gleich Dividend?)
- eine sehr große und eine sehr kleine Zahl als Operanden kombiniert
- unterschiedliche Exponentialdarstellungen der Zahlen (4,6E-3...)
- regionale Zahlenformate (Dezimalpunkt, Tausender-Gruppen)
- Quotient ist periodische Zahl (z.B. 5/9) - Anzahl der Stellen? Rundung?
- Grenzen für kleinste und größte akzeptierte Parameter

■ Fehlerfälle:

- Division durch Null
- Overflow (Zahlenbereichs-Überlauf)
- Underflow (Zahlenbereichs-Unterlauf)
- fehlende oder ungültige Operanden (Syntaxfehler)
- eventuell Operanden außerhalb eines erlaubten Wertebereichs

The screenshot shows a web application titled "Gleitkomma-Division" with a green header. It contains three input fields: "Dividend:", "Divisor:", and "Quotient:". Below these fields are two buttons: "Berechnen" (Calculate) and "Rücksetzen" (Reset).

Testfallerstellung >

Normalfall, Sonderfälle, Fehlerfälle (6)

Beispiel „Dateiexport“

- Testobjekt: Funktion zum Export von Geschäftsdaten in eine Datei
 - Nutzereingabe: Pfad unterhalb des festgelegten Export-Verzeichnisses, dann Betätigen des Buttons „Speichern“
 - die Datei wird unterhalb des Export-Verzeichnisses angelegt
 - die Korrektheit der exportierten Daten ist nicht Testgegenstand
 - es werden verschiedene Betriebssysteme und Dateisysteme unterstützt

- Aufgabe:
 - Welche Normalfälle gibt es?
 - Sonderfälle?
 - Fehlerfälle?



Dateiexport

\Daten\Export\

Testfallerstellung >

Normalfall, Sonderfälle, Fehlerfälle (7)

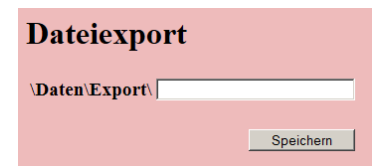
Beispiel „Dateiexport“ (fortgesetzt)

■ Normalfall

- einfacher gültiger Pfad, gültige Daten, keine Fehler, Button „Speichern“
- wie oben, aber Abschluss der Eingabe mit ENTER

■ Sonderfälle

- Datei existiert bereits
- Unterscheidung von Groß-/Kleinschreibung in Pfad und Dateinamen
- erlaubte Sonderzeichen im Pfad (Umlaute, Klammern, Space, ...)
- Testfälle mit Schrägstrich und Backslash im Pfad
- im Pfad angegebene Unterverzeichnisse existieren noch nicht
- keine Daten zum Export verfügbar (leere Zieldatei, Dateigröße 0)
- paralleler Zugriff mehrerer Nutzer auf die gleiche Datei
- spezielle Zeichenkodierungen im Pfad (UTF-8, ...)
- verschiedene Betriebssysteme und Dateisysteme
- keine und verschiedene Dateinamenserweiterungen
- ein Punkt am Anfang, mehrere Punkte im Dateinamen
- Wildcards im eingegebenen Pfad



Dateiexport

\Daten\Export\

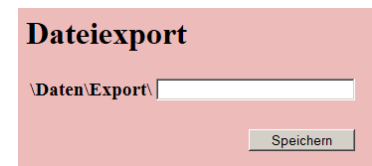
Testfallerstellung >

Normalfall, Sonderfälle, Fehlerfälle (8)

Beispiel „Dateiexport“ (fortgesetzt)

■ Fehlerfälle

- reservierte Dateinamen
- Dateiname entspricht existierendem Verzeichnisnamen
- Dateiname zu lang
- Verzeichnisname im Pfad zu lang
- Pfad insgesamt zu lang
- Pfad leer (keine Eingabe)
- Pfad enthält nicht erlaubte Zeichen (für jedes Zeichen separat testen)
- Speicherplatz auf Ziellaufwerk reicht nicht aus
- Pfad enthält ".." (unerlaubte Navigation im Dateisystem nach oben)
- Laufwerk oder Pfad nicht schreibbar (read-only)
- Nutzer hat keine Berechtigung zum Schreiben
- Datei größer, als vom Dateisystem unterstützt



Dateiexport

\Daten\Export\

Speichern

Testfallerstellung >

Normalfall, Sonderfälle, Fehlerfälle (9)

Schlussfolgerungen zur Testabdeckung

- Ein umfassender Test muss auch Sonder- und Fehlerfälle berücksichtigen!
- Sonder- und Fehlerfälle sind oft auch Ansatzpunkt für die Skalierung der Testtiefe.

Schlussfolgerungen zu menschlichen Eigenschaften

- Systemverständnis, Qualifikation und Erfahrung eines guten Test Engineers sind nicht durch allgemeine Anleitungen und Verfahren zum Testen ersetzbar!
- Die Testabdeckung hängt auch von der individuellen Genauigkeit, Ausdauer und Kreativität des Test Engineers ab.

→ Testabdeckung durch Testfall-Review verbessern!

- Zusätzliche Beteiligte finden oft weitere, bisher fehlende Testfälle (und Fehler).
- Der Umfang dieser Ergänzungen sinkt mit jedem weiteren Beteiligten. Ab fünf ist kaum mehr eine Verbesserung des Testumfangs gegeben.
- Bereits ein Review durch eine zweite Person verbessert die Abdeckung oft stark.

Review von Testfällen

Checkliste:

- Ist der Testfall verständlich?
- Wieviel technisches Hintergrundwissen ist nötig?
- Ermöglicht der Testfall, Fehler zu finden?
- Sind die Ergebnisse validierbar?
- Sind die Testfälle unterschiedlich, gibt es Redundanzen?
- Fehlen wichtige Testfälle?
- Werden auch Sonderfälle und Fehlerfälle geprüft?
- Ist der Testfall in dieser Form ausführbar (manuell oder mittels eines Tools)?
- Ist der Testfall reproduzierbar?



Der Review kann z.B. von einem Tester, Entwickler oder Business-Experten durchgeführt werden.

Erweiterung der Testabdeckung

1. Breite Testabdeckung für Normalfälle

- gültige Eingaben, reguläre Berechnung ("Happy Path"), ...

2. Häufige Sonderfälle und Fehlerfälle

- Passwort falsch, Pflichtfeld leer, Eingabe ungültig, Sonderzeichen, zu lesende Datei existiert nicht, zu erzeugende Datei existiert bereits, 0, 1, ...

3. Tiefer Testabdeckung

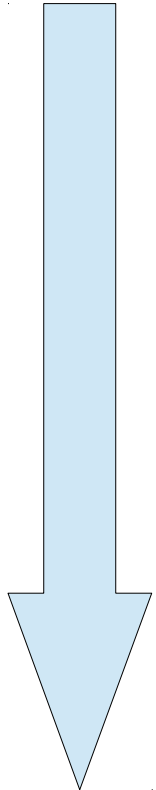
- weitere Sonder- und Fehlerfälle, diverse Optionen oder Modi, parallele Zugriffe, verschiedene Zeichenkodierungen, ...

4. Seltene Sonderfälle und seltene Grenzwerte

- max, sehr lange Zeichenketten, ...

5. Seltene, aber kritische Fehlersituationen

- Ausfall eines Teil- oder Third-Party-Systems, kein Plattenplatz mehr, ...



Priorisierung (1)

Allgemeines

- Priorisierung der Testobjekte, wenn die Zeit nicht für die Erstellung und Ausführung aller Testfälle ausreicht
- Mögliche Kriterien:
 - nach vorgegebenen Anforderungen
 - Qualitätsmerkmale (Sicherheit vor Effizienz, ...)
 - zu erwartende Fehlerschwere in diesem Testobjekt
 - Wahrnehmbarkeit der Fehlerwirkung
 - Wahrscheinlichkeit von Fehlern
 - ➔ Risikogebiete
 - ➔ Risk-Based Testing (RBT)
- gewählte Priorisierung der Testobjekte meist projektspezifisch

Priorisierung (2)

Risikogebiete

- neue Bereiche (neue Features, neue Technologien)
- geänderte Bereiche, späte Änderungen
- schlecht entworfene Bereiche, schlecht wartbare Bereiche
- Bereiche mit eigenmächtigen Veränderungen der Entwickler
- Testobjekte mit unklaren, widersprüchlichen oder sich oft ändernden Anforderungen
- Bereiche mit hoher Komplexität oder vielen Schnittstellen
- Bereiche mit schlechter Testbarkeit
- bisher ungenügend getestete Bereiche
- extern entwickelte Komponenten



Agenda

- Einführung
- Klassifikation von QS-Maßnahmen
- Testfallerstellung
- Ökonomie des Testens
- Systematische Testfallermittlung
- Testausführung
- Fehlerverfolgung
- Kurzüberblick Testmanagement

Ökonomie des Testens >

Ökonomie des Testens

Testen von Software ist ökonomisch sinnvoll, solange gilt:

QS-Aufwand < Fehlerkosten während der Nutzung

- ➔ Wie hoch sind die Fehlerkosten?
- ➔ Wie hoch ist der QS-Aufwand?



Fehlerkosten (1)

Operative Fehlerkosten, z.B.:

- Neudiskussion und Festlegung von Anforderungen
- Nacharbeit durch die Entwicklung, Bereitstellung von Fixes
- Re-Test der behobenen Fehler und Regressionstest der Anwendung
- Aufwand für erneute Installation/Deployment, Rückrufaktionen
- Kosten durch Ausfallzeiten
- Aufwand für Recovery-Maßnahmen bei Beschädigung von Daten
- wiederholte Zulassungsverfahren
- Vertragsstrafen, Gewährleistung, Kulanz

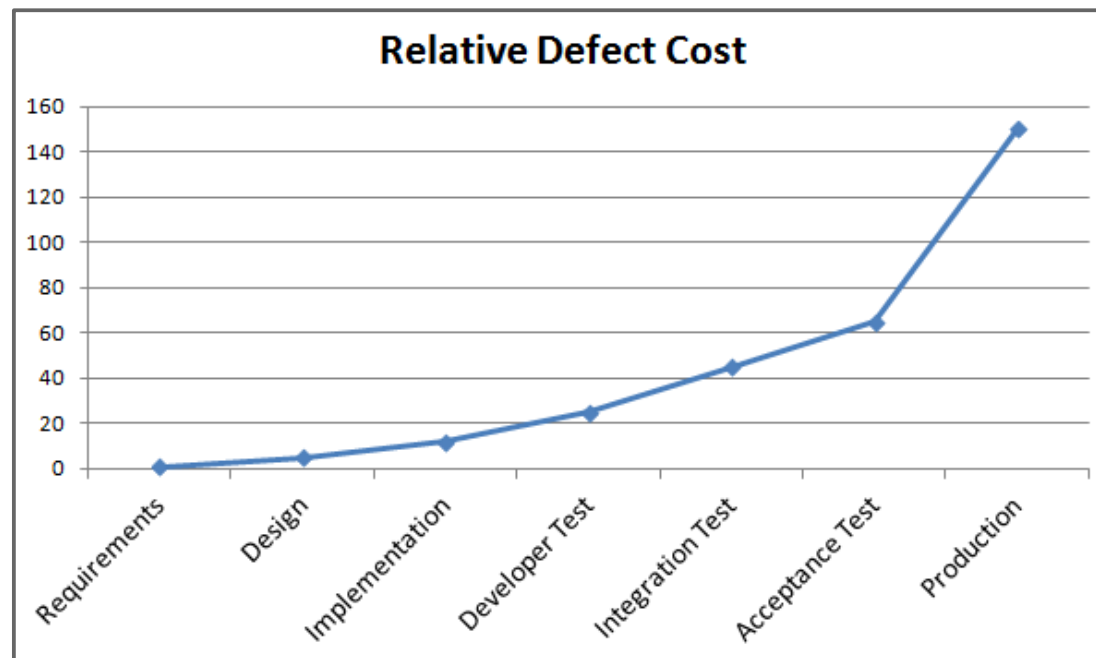
Strategische Fehlerkosten, z.B.:

- Imageverlust
- Verlust von Kunden
- Verlust von Marktanteilen

Ökonomie des Testens >

Fehlerkosten (2)

Fehlerkosten steigen stark an, je später ein Fehler gefunden wird:



Beispiel: 1 Million \$ für eine Änderung im Code eines bereits zugelassenen Flugzeugs

Ökonomie des Testens >

QS-Aufwand (1)

Der QS-Aufwand steigt bei einem vollständigen Test aller Kombinationen exponentiell mit der Anzahl der Parameter:

- Anzahl von Hardware- und Software-Konfigurationen
- Anzahl möglicher Eingaben
- Datenkonstellationen, Datenbankinhalte
- Umgebungsbedingungen (Drittsysteme, Sensoren, ...)
- Reihenfolge und Timing von Aktionen



→ Phänomen „Testfallexplosion“

→ Astronomisch hohe Anzahl von Testfällen

Ökonomie des Testens >

QS-Aufwand (2)

Aus der Testfallexplosion folgt:

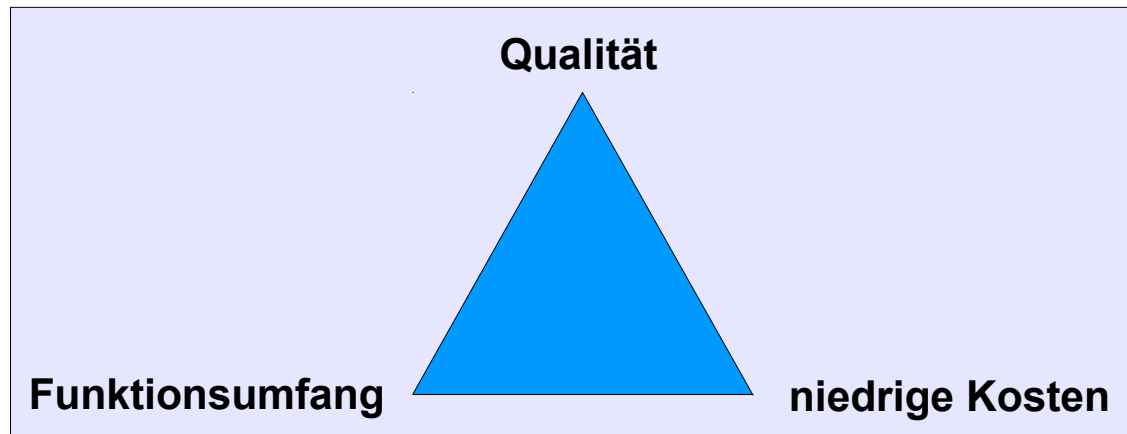
- Der vollständige Test einer Software ist nicht möglich!
- Tests sind immer nur Stichproben.
- Die Abwesenheit von Fehlern ist nicht beweisbar.
- Der QS-Umfang ist ein Kompromiss zwischen QS-Aufwand und Restrisiko.

→ Festlegung von Qualitätszielen nötig

→ Sorgfältige Kommunikation zum Testumfang

Ökonomie des Testens > **Qualitätsziele**

Qualität konkurriert mit anderen Projektzielen:



- es gibt keine allgemein gültigen Qualitätsziele; sie sind projektspezifisch
- Beispiele: Medizintechnik oder Zahlungssysteme versus Web-Anwendung zur Darstellung des Wetterberichts
- Qualitätsziele sollten durch den Kunden oder Anforderer festgelegt werden
- Qualitätsziele werden zum Teil durch Regelungen und Vorschriften definiert

Ökonomie des Testens >

Kommunikation zum Testumfang

Da ein „vollständiger Test“ nicht möglich ist, sollte auch niemals dieser oder ein ähnlicher Begriff verwendet werden.

VERMEIDEN:

- vollständiger/kompletter Test
- volle/100% Testabdeckung

GUT:

- umfassender/ausführlicher/detaillierter Test
- Test im abgestimmten/vereinbarten/abgesprochenen Umfang
- Test im erforderlichen/notwendigen Umfang
- Test im empfohlenen/sinnvollen Umfang
- in die Tiefe gehender/gründlicher/genauer Test

Schlussfolgerungen

- Sinnvollen Testaufwand nach Risiko und Gefährdung im Fehlerfall bestimmen
→ *Festlegung von Qualitätszielen!*
- Je früher ein Test im Entwicklungsprozess stattfindet, um so effizienter ist er.
→ *Frühestmögliche QS-Maßnahmen, entwicklungsbegleitende Tests!*
- Risikominimierung durch möglichst hohe Testabdeckung mit einer minimalen Anzahl von Testfällen
→ *Anwendung systematischer Methoden zur Testfallermittlung*
- Reduktion des formellen Aufwands
→ *Evtl. Skalierung der Testfallbeschreibung bis hinunter zu bloßen Testideen*
- Diese Fakten werden oft nicht richtig verstanden.
→ *Sorgfältige Kommunikation und Auswahl der Begriffe.*

Agenda

- Einführung
- Klassifikation von QS-Maßnahmen
- Testfallerstellung
- Ökonomie des Testens
- Systematische Testfallermittlung
- Testausführung
- Fehlerverfolgung
- Kurzüberblick Testmanagement

Systematische Testfallermittlung >

Systematische Testfallermittlung

- Ziel: hohe Testabdeckung mit minimalem Aufwand
- Vorgestellte Methoden:
 - Äquivalenzklassenanalyse
 - Grenzwertanalyse
 - Entscheidungstabellen
 - Testmatrix
 - Zustandsbasierte Testfallermittlung
 - CRUD-Matrix
 - Anforderungsbasierte Testfallermittlung
 - Intuitive Testfallermittlung
 - Exploratives Testen
 - Session-Based Test Management
- Mehrere Methoden werden oft kombiniert.

Äquivalenzklassenanalyse (1)

Prinzip

- isolierte Betrachtung einzelner Eingabe- oder Ausgabeelemente, z.B. eines Eingabefelds
- Zerlegung der Menge aller möglichen Eingabe- oder Ausgabe-Daten in Untermengen („Äquivalenzklassen“)...
- ...so dass der Test mit einem beliebigen Wert aus einer Klasse äquivalent zum Test mit jedem anderen Wert aus der gleichen Äquivalenzklasse ist.

- ➔ alle Werte einer Klasse führen zu äquivalentem Verhalten
- ➔ für alle Werte einer Klasse werden die gleichen Codezweige durchlaufen
- ➔ der Test eines Repräsentanten je Klasse wird als ausreichend angesehen

Äquivalenzklassenanalyse (2)

Test mit Repräsentanten

- Auswahl des Repräsentanten je Klasse nach Plausibilität und Häufigkeit
- zunächst die Repräsentanten aller Klassen einzeln testen
- dann Kombination von Repräsentanten verschiedener Äquivalenzklassen, unter denen Abhängigkeiten bestehen

Beispiel 1: Vom Warenpreis abhängige Versandkosten, mit drei Äquivalenzklassen

- Warenpreis < 50 Euro: Versandkosten = 8 Euro
- Warenpreis ≥ 50 und < 300 Euro: Versandkosten = 4 Euro
- Warenpreis ≥ 300 Euro: Versandkosten = 0 Euro

Äquivalenzklassenanalyse (3)

Beispiel 2: Äquivalenzklassen für die Eingabe eines Datums

- Datum in der Vergangenheit
- Datum heute
- Datum in der Zukunft
- 29.02.2016 (Sonderfall mit eigener Äquivalenzklasse)
- leere Eingabe
- logisch falsches Datum, z. B. "31.02.2016"
- syntaktisch falsches Datum, z. B. "31.ab;2xy6"
- vielleicht eigene Äquivalenzklassen für die Tage der Sommerzeitumstellung mit 23 h und 25 h benötigt
- archäologische Verwaltungssoftware benötigt vielleicht auch v. Chr./n. Chr.

→ Die Klasseneinteilung hängt oft auch vom Anwendungsfall ab!

→ Äquivalenzklassen können hierarchisch weiter unterteilt werden: z.B. syntaktisch falsche Datums-Eingaben in leer, Leerzeichen, Buchstaben, ...

Äquivalenzklassenanalyse (4)

Elementklassengetriebener Test

- die Eingaben werden einer Äquivalenzklassenanalyse unterzogen und der Test mit den identifizierten Repräsentanten jeder Klasse durchgeführt

Wirkklassengetriebener Test

- Äquivalenzklassenanalyse der möglichen Ergebnisse (Wirkungen) und Konzeption der Testfälle so, dass alle als Repräsentanten identifizierten Wirkungen wenigstens einmal eintreten

Kombination

- oft deckt bereits der elementklassengetriebene Test auch alle Wirkungen ab, ansonsten können beide Arten kombiniert werden

Äquivalenzklassenanalyse (5)

Beispiel, elementklassengetrieben

- Eingaben sind Produktpreise einzelner Produkte, drei Äquivalenzklassen:
 - positiv: normale Waren
 - 0: kostenloser gedruckter Katalog
 - negativ: Gutschriften

Beispiel, wirkklassengetrieben

- Wirkung ist der Gesamtpreis des Warenkorbs, drei Äquivalenzklassen:
 - positiv: normale Waren, normale Versandkosten
 - 0: Warenpreise + Versandkosten = Gutschriften
 - negativ: Gutschriften > Warenpreise + Versandkosten

Systematische Testfallermittlung >

Grenzwertanalyse (1)

Prinzip

- Ränder der Äquivalenzklassen (ÄK) werden geprüft
- Nutzung der Grenzwerte an den Wertebereichs-Enden der Äquivalenzklassen

| | | | | | | | | | | | | | | | | | | | | |
|-----|----|---|---|---|---|-----|----|----|----|----|----|----|-----|----|----|----|-----|-----|-----|-----|
| ... | -1 | 0 | 1 | 2 | 3 | ... | 31 | 32 | 33 | 34 | 35 | 36 | ... | 97 | 98 | 99 | 100 | 101 | 102 | ... |
|-----|----|---|---|---|---|-----|----|----|----|----|----|----|-----|----|----|----|-----|-----|-----|-----|

Beispiel

- Funktion "Passwort ändern", Passwort darf 8 bis 16 Zeichen lang sein
- ÄK für Länge der Passwörter: [0..7], [8..16], [17..X], besser [0] separat testen
- zu prüfen: neues Passwort mit 0, 7, 8, 16 und 17 Zeichen

Grenzwertanalyse (2)

Auch technisch bedingte Grenzen sind zu prüfen

- oft nicht durch Anforderungen, sondern durch die Realisierung vorgegeben
- Beispiele:
 - größtmögliche numerische Eingabe in einem Integer-Eingabefeld ("999...9")
 - längste mögliche Zeichenkette in einem Eingabefeld
 - maximal darstellbare Zahl, z.B. bei 32-Bit-Integer
- Technische Grenzen müssen manchmal explorativ ermittelt werden

Systematische Testfallermittlung >

Kombinationen von Eingabewerten (1)

Problem

- die zu testende Funktion hat mehrere Eingabeparameter; für jeden Eingabeparameter gibt es Sonderfälle und Fehlerfälle
- Müssen auch alle Kombinationen daraus getestet werden?

Überlegung

- Test aller Kombinationen aufgrund der hohen Anzahl meist nicht möglich
- Code behandelt Sonderfälle und Fehlerfälle oft sequentiell und isoliert

```
public boolean isValidAddress(PostalAddress address)
{
    if (address == null) { return false; }
    // ...
    if (address.street == null) { return false; }
    if (address.street.trim().equals("")) { return false; }
    // ...
    if (address.zipcode == null) { return false; }
    address.zipcode = address.zipcode.trim();
    if (address.zipcode.length() != 5 &&
        address.zipcode.length() != 8) { return false; }
    // ..
    return true;
}
```

Systematische Testfallermittlung >

Kombinationen von Eingabewerten (2)

Schlussfolgerung

- Normalfall für alle Eingabeparameter testen
- Sonderfälle und Fehlerfälle isoliert für jeden Eingabeparameter testen; alle anderen Parameter entsprechen dabei einem Normalfall
- Kombinationen nur testen, wenn das Wissen oder die Erwartung besteht, dass erst eine bestimmte Kombination zu einer anderen Reaktion führt.

Beispiel „Division von Gleitkommazahlen“

- Normalfälle mit Kombinationen der Vorzeichen von Dividend und Divisor
- alle Sonderfälle für den Dividenden, dabei Normalfall für den Divisor
- alle Sonderfälle für den Divisor, dabei Normalfall für den Dividenden
- alle Fehlerfälle für den Dividenden, dabei Normalfall für den Divisor
- alle Fehlerfälle für den Divisor, dabei Normalfall für den Dividenden

Systematische Testfallermittlung >

Kombinationen von Eingabewerten (3)

Weitere Beispiele für zu testende Kombinationen

- Adressprüfung mit Eingabefeldern für Straße, Hausnummer, PLZ, Ort:
 - ➔ Nur bestimmte Kombinationen ergeben eine existierende Adresse
 - ➔ Kombination von Normalfällen
- Prüfung auf ausgefüllte Pflichtfelder:
 - ➔ Kombination leerer Pflichtfelder führt evtl. zu Liste von Fehlermeldungen
 - ➔ Kombination von Fehlerfällen

Entscheidungstabellen (1) – Prinzip

Prinzip

- dienen dem Test von Kombinationen, eine Zeile für jede Kombination
- Spalten für Bedingungen (Zustände oder Eingabewerte)
- Spalten für erwartete Ergebnisse

Vorgehensweise

- je Bedingung eine Spalte anlegen
- alle Wertekombinationen der Bedingungen eintragen
- ungültige/nicht mögliche Kombinationen grau markieren
- Spalten für erwartete Ergebnisse ausfüllen (für gültige Kombinationen)
- jede Zeile mit einer gültigen Kombination entspricht einem Testfall

→ gut geeignet für komplexere Logik oder regelbasiertes Verhalten

Systematische Testfallermittlung >

Entscheidungstabellen (2) – Beispiel

| | Bedingung 1 | Bedingung 2 | Bedingung 3 | Ergebnis 1 | Ergebnis 2 |
|---------|-------------|-------------|-------------|------------|--------------|
| Fall 1 | grün | 1 | ja | genehmigt | E-Mail |
| Fall 2 | grün | 1 | nein | genehmigt | E-Mail |
| Fall 3 | grün | 2 | ja | genehmigt | keine E-Mail |
| Fall 4 | grün | 2 | nein | abgelehnt | E-Mail |
| Fall 5 | grün | 3 | ja | | |
| Fall 6 | grün | 3 | nein | Fehlertext | keine E-Mail |
| Fall 7 | rot | 1 | ja | genehmigt | E-Mail |
| Fall 8 | rot | 1 | nein | | |
| Fall 9 | rot | 2 | ja | | |
| Fall 10 | rot | 2 | nein | abgelehnt | keine E-Mail |
| Fall 11 | rot | 3 | ja | genehmigt | E-Mail |
| Fall 12 | rot | 3 | nein | abgelehnt | E-Mail |
| ... | ... | ... | ... | ... | ... |

Systematische Testfallermittlung >

Entscheidungstabellen (3) – Beispiel

| | Voreinstellungen | Buchungseingaben | | Aktion des VG | Ergebnis | | Aktion des SP | Ergebnis |
|----|--------------------------------------|---------------------------|----------------------------|----------------------|--|----------------------------|----------------------|----------|
| | OrgE des Lerner | Buchung der Weiterbildung | Workflow | VG | SP | Wer ist Genehmiger in AMS? | SP | Training |
| | Zustimmung BR erforderlich (IT 9100) | beruflich | mit WF / ohne VG / ohne WF | genehmigt / lehnt ab | Einbindung SP (sukzessiv, parallel, keine) | BR / ÖWA | genehmigt / lehnt ab | gebucht |
| 1 | x | x | mit WF | genehmigt | sukzessiv | ÖWA | genehmigt | x |
| 2 | x | x | mit WF | genehmigt | sukzessiv | ÖWA | lehnt ab | |
| 3 | x | x | mit WF | lehnt ab | parallel | - | genehmigt | |
| 4 | x | x | mit WF | lehnt ab | parallel | - | lehnt ab | |
| 5 | x | x | ohne VG | genehmigt | sukzessiv | ÖWA | genehmigt | x |
| 6 | x | x | ohne VG | genehmigt | sukzessiv | ÖWA | lehnt ab | |
| 7 | x | x | ohne VG | lehnt ab | | | genehmigt | |
| 8 | x | x | ohne VG | lehnt ab | | | lehnt ab | |
| 9 | x | x | ohne WF | genehmigt | | | genehmigt | |
| 10 | x | x | ohne WF | genehmigt | | | lehnt ab | |
| 11 | x | x | ohne WF | lehnt ab | | | genehmigt | |
| 12 | x | x | ohne WF | lehnt ab | | | lehnt ab | |
| 13 | x | | mit WF | genehmigt | sukzessiv | je nach ÖWA-Flag | genehmigt | x |
| 14 | x | | mit WF | genehmigt | sukzessiv | je nach ÖWA-Flag | lehnt ab | |
| 15 | x | | mit WF | lehnt ab | parallel | - | genehmigt | |
| 16 | x | | mit WF | lehnt ab | parallel | - | lehnt ab | |
| 17 | x | | ohne VG | genehmigt | sukzessiv | je nach ÖWA-Flag | genehmigt | x |
| 18 | x | | ohne VG | genehmigt | sukzessiv | je nach ÖWA-Flag | lehnt ab | |
| 19 | x | | ohne VG | lehnt ab | | | genehmigt | |
| 20 | x | | ohne VG | lehnt ab | | | lehnt ab | |
| 21 | x | | ohne WF | genehmigt | - | - | genehmigt | x |
| 22 | x | | ohne WF | genehmigt | | | lehnt ab | |
| 23 | x | | ohne WF | lehnt ab | | | genehmigt | |
| 24 | x | | ohne WF | lehnt ab | | | lehnt ab | |
| 25 | | x | mit WF | genehmigt | sukzessiv | ÖWA | genehmigt | x |
| 26 | | x | mit WF | genehmigt | sukzessiv | ÖWA | lehnt ab | |
| 27 | | x | mit WF | lehnt ab | parallel | - | genehmigt | |
| 28 | | x | mit WF | lehnt ab | parallel | - | lehnt ab | |
| 29 | | x | ohne VG | genehmigt | sukzessiv | ÖWA | genehmigt | x |
| 30 | | x | ohne VG | genehmigt | sukzessiv | ÖWA | lehnt ab | |
| 31 | | x | ohne VG | lehnt ab | | | genehmigt | |

Entscheidungstabellen (4) – Pairwise

Pairwise-Methode

- ergänzendes Verfahren zu Entscheidungstabellen
- reduziert die Anzahl der Testfälle stark, indem nicht alle möglichen Kombinationen getestet werden
- Mindestforderung: Jeder Wert einer Bedingung (Zustand oder Eingabe) wird paarweise mit jedem Wert der anderen Bedingungen zusammen getestet:
 - rot + 1, rot + 2, rot + 3
 - rot + ja, rot + nein
 - 1 + ja, 1 + nein
 - ...
- jede dieser Zweierkombinationen muss wenigstens einmal in einer zu testenden Gesamtkombination vorkommen, z.B.
 - rot + 1 + nein (enthält: rot + 1, rot + nein, 1 + nein)
- unter Umständen werden Fehler nicht entdeckt, die nur bei bestimmten Kombinationen von mehr als zwei Bedingungen auftreten

Systematische Testfallermittlung >

Testmatrix

- Ziel: Effiziente Abdeckung aller Einzelwerte von Bedingungen, nicht primär der Test bestimmter Kombinationen
- Aufbau analog zu Entscheidungstabellen, aber Inhalt weniger systematisch und nicht notwendigerweise vollständig
- Tabelle mit zu testenden Kombinationen von Bedingungen erstellen
- Fälle so wählen, dass jeder Repräsentant pro Bedingung wenigstens einmal vorkommt, zusätzlich eventuell wichtige Kombinationen eintragen
- jede Zeile entspricht einem Testfall

| | Browser | Bestelltyp | Zahlungsart | Versandart | Erwartetes Ergebnis |
|-------------------|------------|---------------|-------------|-------------|---------------------|
| Testfall 1 | Firefox 57 | registriert | PayPal | DHL Express | ... |
| Testfall 2 | Firefox 57 | unregistriert | Kreditkarte | UPS | ... |
| Testfall 3 | Chrome 62 | registriert | Nachnahme | DHL | ... |
| Testfall 4 | Safari | unregistriert | Nachnahme | UPS | ... |
| Testfall 5 | MS IE 11.0 | registriert | Bankeinzug | TNT Express | ... |

Zustandsbasierte Testfallermittlung (1)

Prinzip

- gut für Abläufe, die sich durch Zustandsautomaten beschreiben lassen
- alle Zustände und Zustandsübergänge des Testobjekts ermitteln
- für jeden möglichen Zustandsübergang einen Testfall erstellen
- damit wird auch jeder Zustand wenigstens einmal durchlaufen

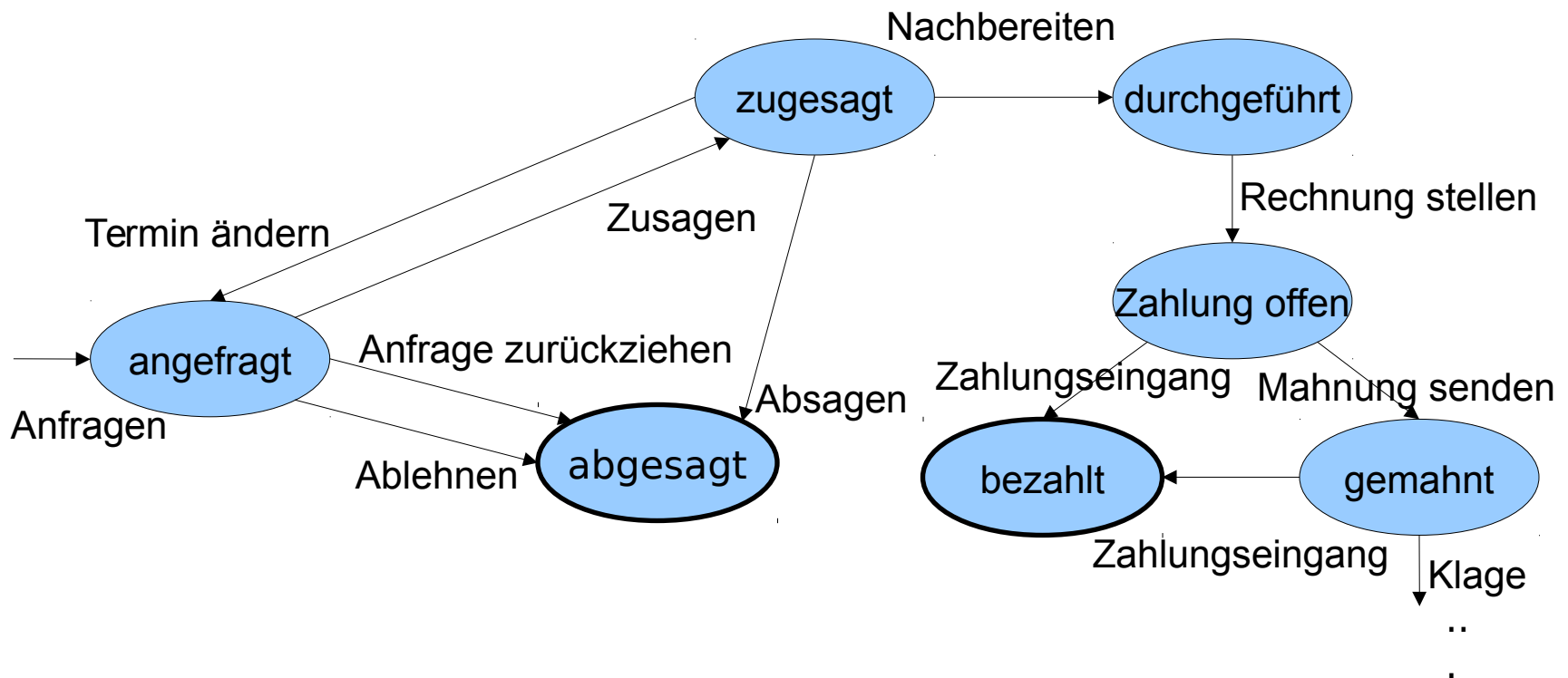
Hinweise

- wenn ein Wechsel zwischen zwei Zuständen durch verschiedene Ereignisse ausgelöst werden kann, auch verschiedene Testfälle dafür erstellen
- wenn nötig, auch Fehlerfälle für nicht erlaubte Zustandsübergänge erstellen

Systematische Testfallermittlung >

Zustandsbasierte Testfallermittlung (2)

Beispiel: Objekt "Werkstatttermin"



CRUD-Matrix

Prinzip

- Lebenszyklus-Operationen von Datenobjekten in Tabelle aufnehmen und testen (Create, Read, Update, Dele^te):
 - Zeilen: relevante Funktionen/Features der Software
 - Spalten: Typen der Datenobjekte
 - Felder: mögliche CRUD-Operationen
- Korrektheit aller angegebenen CRUD-Operationen testen
- bei Black-Box-Test: Testfälle für CRUD-Operationen der nach außen sichtbaren Geschäftsobjekte
- bei White-Box-Tests auch auf interne Datenobjekte beziehbar

| | Datenobjekt-Typ 1 | Datenobjekt-Typ 2 | Datenobjekt-Typ 3 |
|------------|-------------------|-------------------|-------------------|
| Funktion 1 | R | C, U, D | - |
| Funktion 2 | C | - | R |
| Funktion 3 | C, R, D | - | D |

Systematische Testfallermittlung >

Anforderungsbasierte Testfallermittlung

Prinzip

- für jede Anforderung einen oder mehrere Testfälle erstellen

Probleme und Grenzen

- oft nur explizite Anforderungen berücksichtigt
- selbst explizite Anforderungen sind oft lückenhaft
- meist nicht alle Sonderfälle und Fehlerfälle enthalten

Besser

- Anforderungen dienen zunächst zur Definition der Testobjekt-Hierarchie
- Testfälle für Testobjekte unter Berücksichtigung der Anforderungen erstellen

→ Sinnvoll als Basis, je Anforderung mindestens ein Testfall.

→ Ergänzen durch andere Verfahren und Testfälle für implizite Anforderungen.

Systematische Testfallermittlung >

Intuitive Testfallermittlung

- auch: „Error Guessing“
- Testfallermittlung allein auf Basis von Erfahrung und Intuition des Testers
 - Erfahrung über in der Vergangenheit aufgetretene Fehler
 - Intuition wo Fehler in Zukunft wahrscheinlich auftreten können
- erfordert erfahrene Tester
- Erfolg von Tester subjektiv abhängig

→ Sollte nicht als primäres Testverfahren eingesetzt werden.

→ Sinnvoll zur Ergänzung der systematischen Testverfahren.

Systematische Testfallermittlung >

Exploratives Testen

Idee

- Fehler treten oft lokal gehäuft auf (Risikogebiete!)
- Wo ein Fehler gefunden wurde, testet man intensiver.

Vorgehensweise

- Testdurchführung und Testfallerstellung erfolgen gleichzeitig:
 - Test eines Testobjekts beginnen, dabei gleichzeitig Testfälle erstellen
 - wenn Fehler auftreten, weitere Testfälle für das fehlerhafte Testobjekt erzeugen
- ➔ die Testabdeckung ist am höchsten, wo die meisten Fehler gefunden wurden
- Ergänzung zu anderen Methoden
- erfahrene Tester liefern auch hier wesentlich bessere Ergebnisse
- kann ohne oder mit Testzielvorgaben/Test-Charta durchgeführt werden
- Session-Based Testing ist eine Form des explorativen Testens

Session-Based Test Management (1)

Allgemeines

- verbindet exploratives Testen mit erweiterter Steuerbarkeit und Auswertbarkeit
- Testablauf in sogenannte „Sessions“ strukturiert
- Jonathan & James Bach, 2000

Elemente je Session

- *Mission:*
 - Zweck der Session
 - Was ist zu testen, nach welchen Problemen wird gesucht?
- *Charta:*
 - Agenda für die Session
- *Session selbst:*
 - ununterbrochene Testzeit, fokussiert auf die Charta
 - time-boxed, ideal sind 1-2 h



Session-Based Test Management (2)

Elemente (fortgesetzt)

■ *Session Report:*

- Charta
- Testobjekt
- ausgeführte Testschritte
- aufgetretene Defekte
- aufgetretene Fragen, Bedenken
- Beginn- und Endezeitpunkt
- Prozentsatz der aufgewendeten Session-Zeit für
 - Testdurchführung
 - Fehlerverfolgung
 - Setup/Sonstiges
- manchmal in Form eines formellen, elektronisch auswertbaren Dokuments



Session-Based Test Management (3)

Elemente (fortgesetzt)

■ *Debriefing:*

- Abschlussbesprechung zwischen Testmanager und Tester anhand des Session Reports, folgt "PROOF"
 - **Past:** Ablauf
 - **Results:** Ergebnisse
 - **Obstacles:** Hindernisse
 - **Outlook:** Ausblick
 - **Feelings:** Gefühl, Einschätzung des Testers

Vorteil gegenüber einfachem Explorativem Testen

- Strukturierung der Arbeitsweise und Kontrolle → gut zur Anleitung gelegentlich mittestender Personen (Fachabteilung, Endnutzer, ...)

Nachteil

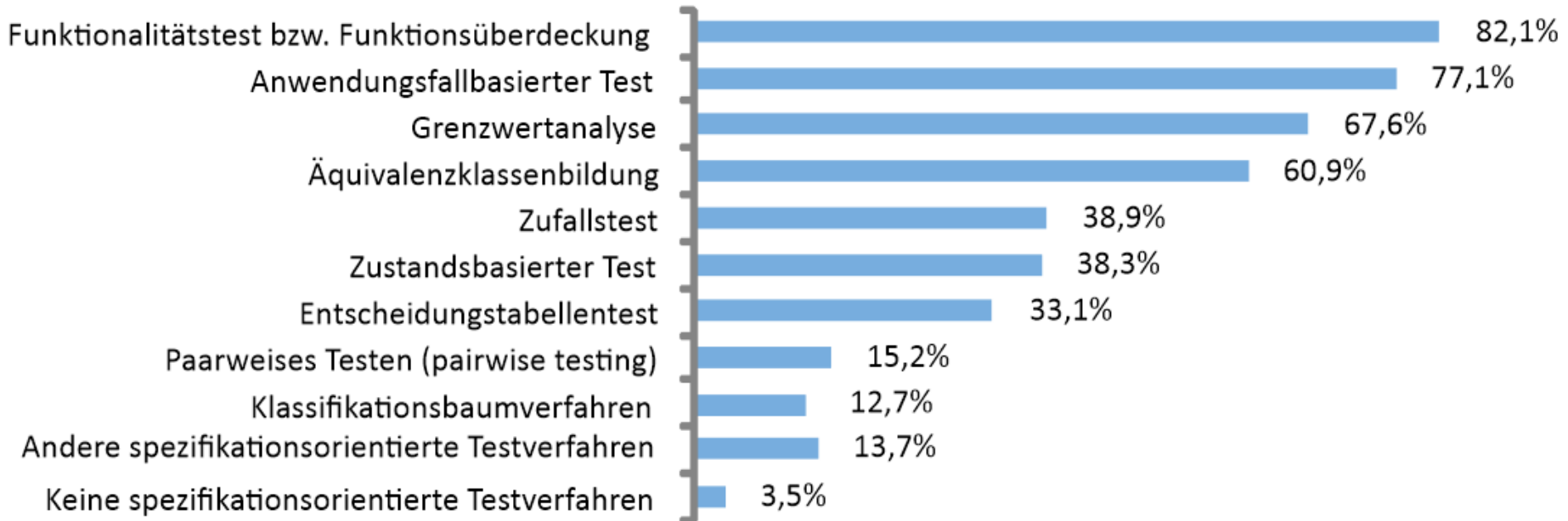
- formeller Aufwand



Systematische Testfallermittlung >

Umfrageergebnisse (1)

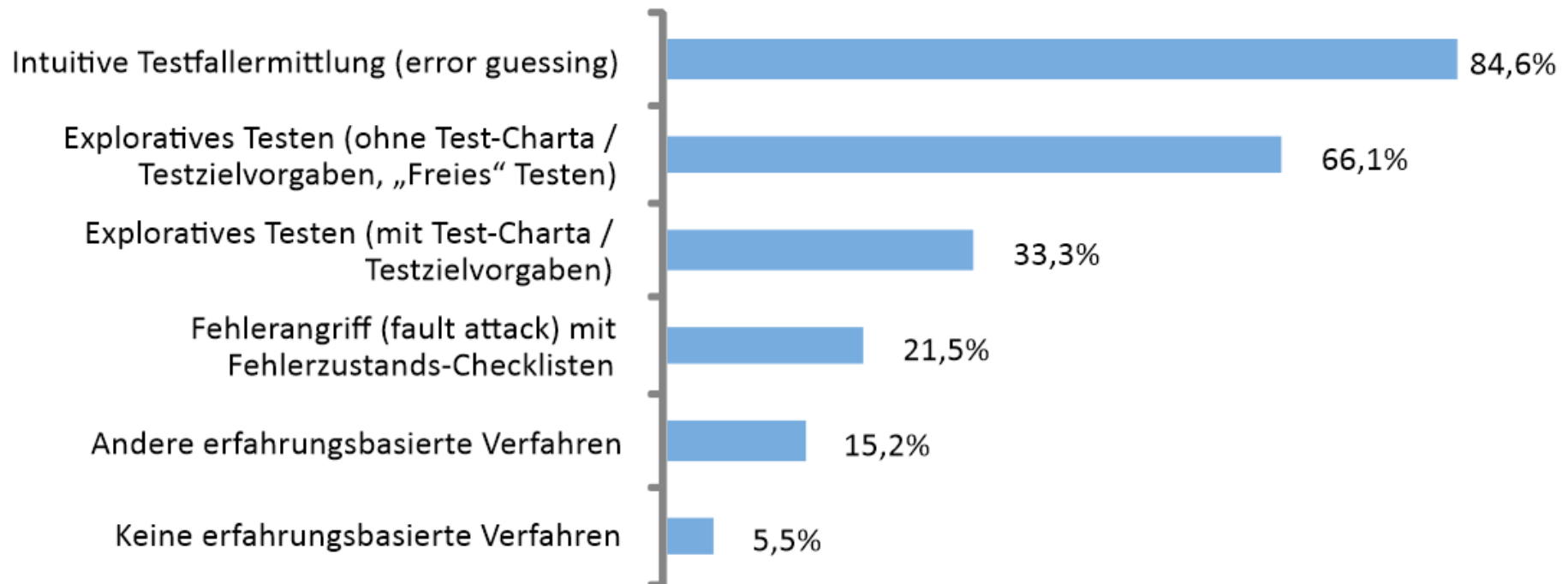
Welche spezifikationsorientierten Testentwurfsverfahren (Black-Box) werden in Ihrem Unternehmen eingesetzt?



© Softwaretestumfrage 2011: HS Bremen, HS Bremerhaven, FH Köln, ANECON, GTB, STB

Systematische Testfallermittlung > Umfrageergebnisse (2)

Welche erfahrungsbasierten Verfahren werden in Ihrem Unternehmen eingesetzt



© Softwaretestumfrage 2011: HS Bremen, HS Bremerhaven, FH Köln, ANECON, GTB, STB

Agenda

- Einführung
- Klassifikation von QS-Maßnahmen
- Testfallerstellung
- Ökonomie des Testens
- Systematische Testfallermittlung
- Testausführung
- Fehlerverfolgung
- Kurzüberblick Testmanagement

Testausführung >

Testausführung (1)

Voraussetzungen prüfen oder herstellen

- Testfälle erstellt
- Testumgebung verfügbar
 - passende Hardware-Umgebung
 - korrekte, zu testende Softwareversion installiert
 - Drittsysteme oder Simulatoren angebunden
 - Netzwerkzugang
 - korrekte Testdaten eingespielt/verfügbar
 - bei Zugangsbeschränkungen Benutzer eingerichtet
- ...

Testausführung (2)

Vorgehen mit Testsets

- Testset = Teilmenge von auszuführenden Testfällen
- Zuordnung der auszuführenden Testfälle zu einem Testset
- separate Testsets für verschiedene Zwecke, beispielsweise für:
 - Smoketest (kleine, breite Auswahl von Testfällen)
 - Featuretest für eine Release (Testfälle für neue Features)
 - Regressionstest (größere, breite Auswahl von Testfällen)
 - Tests in verschiedenen Systemumgebungen (Testumgebung, Produktion)
 - Tests in verschiedenen Teststufen
- Testsets sind oft selbst hierarchisch gegliedert
- Testobjekt-Hierarchie und Testset-Hierarchie können sich stark unterscheiden
- der gleiche Testfall kann mehreren Testsets zugeordnet werden
- Ausführung der Testfälle auf den nächsten Folien beispielhaft im TestDirector

Mercury TestDirector 8.0 SP2 - Microsoft Internet Explorer

Datei Bearbeiten Ansicht Favoriten Extras ?

Zurück Suchen Favoriten Medien

Adresse http://.../tdbin/start_a.htm

TestDirector 8.0

Project : e-Telco Wartung [rvo2] REQUIREMENTS TEST PLAN TEST LAB DEFECTS TOOLS HELP LOGOUT

Execution Test Sets Tests Search Hosts Analysis

Test Sets Tree

- Portal
 - Service-Bereich
 - SSI
 - ContentClient
 - GK
 - Abmeldung
 - Anmeldung** (Testset)
 - CC-Kunde
 - Dienstaufrufe Teil Mod
 - Exceptions und Locks
 - Gewinnspiel-Monitoring
 - L3-Anmeldung Brute-F
 - Plausibilitätschecks An
 - Plausibilitätschecks Zu
 - Registrierung L4
 - Warenkorb Kontaktfor
 - XSS Eingaben mit spit
 - XSS korrekte Filterung
 - Zugangsdaten änder
 - Zugangsdaten verges

Execution Grid Execution Flow Test Set Properties

Select Tests Run Run Test Set

Sorted By: Plan: Test Name Run (Ctrl+F9)

| Plan: Test Name | Plan: Type | Status | Planned Host | Responsible | Exec Date |
|--|------------|----------|--------------|-------------|------------|
| [1]T06 - Anmeldung L3, gleicher Nutzer parallel | MANUAL | ✓ Passed | | | 11.07.2005 |
| [1]T07 - Anmeldung L3, Formularziel HTTPS | MANUAL | ✓ Passed | | | 11.07.2005 |
| [1]T08 - Anmeldung L4 | MANUAL | ✓ Passed | | | 11.07.2005 |
| [1]T09 - Anmeldung L4, ohne Cookies | MANUAL | ✓ Passed | | | 08.08.2005 |
| [1]T10F - Anmeldung L4, falsche Kombination | MANUAL | ✓ Passed | | | 11.07.2005 |
| [1]T11 - Anmeldung L4, gleicher Nutzer parallel | MANUAL | ✓ Passed | | | 05.07.2005 |
| [1]T12 - Anmeldung L4, Formularziel HTTPS | MANUAL | ✓ Passed | | | 05.07.2005 |
| [1]T13 - Anmeldung L4, Secure Cookie | MANUAL | ✓ Passed | | | 05.07.2005 |
| [1]T14 - Anmeldung L3 und L4, gleicher Nutzer | MANUAL | ✓ Passed | | | 11.07.2005 |
| [1]T15 - Anmeldung L3 und L4, verschiedene Nutzer | MANUAL | ✓ Passed | | | 11.07.2005 |
| [1]T16 - An- und Abmeldung L4, Anmeldung L3 | MANUAL | ✓ Passed | | | 11.07.2005 |
| [1]T17 - Anmeldung L3 und L4, gleicher Nutzer parallel | MANUAL | ✓ Passed | | | 11.07.2005 |
| [1]T25 - Anmeldung, Plausibilitätstests | MANUAL | ▶ No Run | | | |

Last Run Result

Test 18 of 18 Server Time: 09:06 AM 09.08.05

Fertig Internet

Manual Runner Test Set: <Anmeldung> Test: <[1]T25 - Anmeldung, Plausibilitätstests>

Exec Steps End of Run Cancel

Run Execute Steps

Run Name: Run_9-8_9-6-33 [Operating System Info](#)

Tester: rvo2

Status: Not Completed

Exec Date: 08.09.2005

Exec Time: 09:06:33

Browser: IE 6.0

Version: 4.0.0.0.141

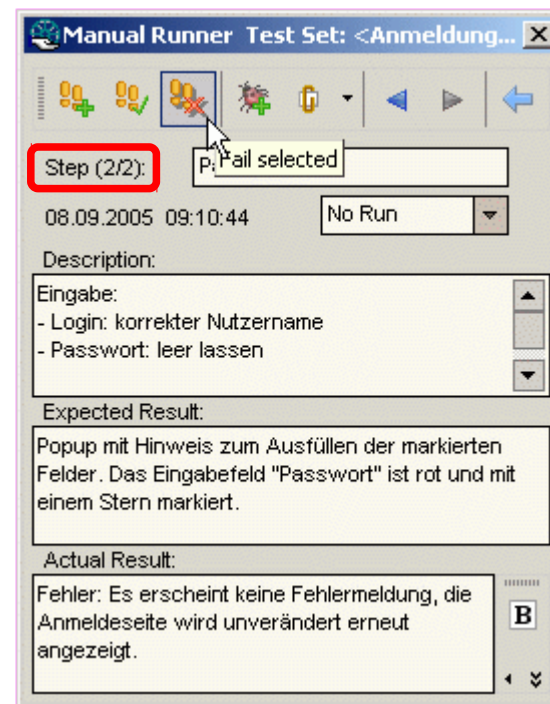
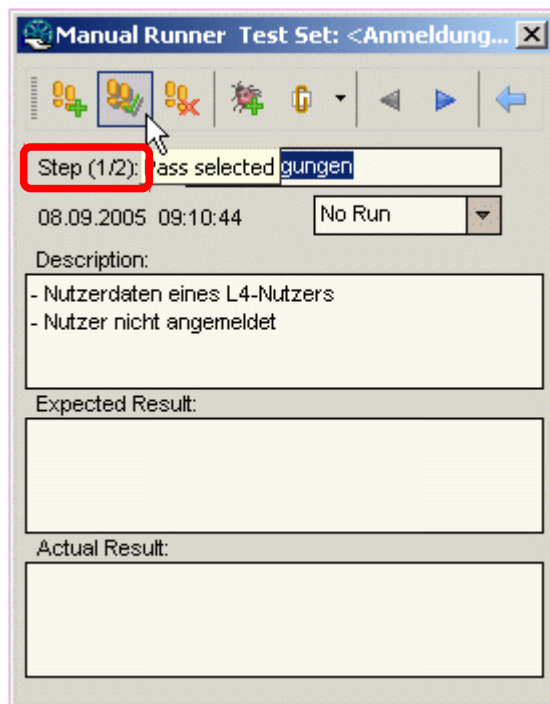
Testsystem: Simulator

Test Details

Name: T25 - [More...](#)

Testausführung >

Testausführung (5)



Mercury TestDirector 8.0 SP2 - Microsoft Internet Explorer

Datei Bearbeiten Ansicht Favoriten Extras ?

Zurück Zurück Suchen Favoriten Medien

Adresse http://.../tdbin/start_a.htm

TestDirector 8.0

Project : e-Telco Wartung [rvo2] REQUIREMENTS TEST PLAN TEST LAB DEFECTS TOOLS HELP LOGOUT

Execution Test Sets Tests Search Hosts Analysis

Test Sets Tree

- Portal
 - Service-Bereich
 - SSI
 - ContentClient
 - GK
 - Abmeldung
 - Anmeldung
 - CC-Kunde
 - Dienstaufrufe Teil Moc
 - Exceptions und Locks
 - Gewinnspiel-Monitorin
 - L3-Anmeldung Brute-f
 - Plausibilitätschecks Ani
 - Plausibilitätschecks Zu
 - Registrierung L4
 - Warenkorb Kontaktfor
 - XSS Eingaben mit spit
 - XSS korrekte Filterung
 - Zugangsdaten ändern
 - Zugangsdaten verges

Execution Grid Execution Flow Test Set Properties

Select Tests Run Run Test Set

Sorted By: Plan: Test Name[Asc]

| Plan: Test Name | Plan: Type | Status | Planned Host | Responsible | Exec Date |
|--|------------|----------|--------------|-------------|------------|
| [1]T06 - Anmeldung L3, gleicher Nutzer parallel | MANUAL | ✓ Passed | | | 11.07.2005 |
| [1]T07 - Anmeldung L3, Formularziel HTTPS | MANUAL | ✓ Passed | | | 11.07.2005 |
| [1]T08 - Anmeldung L4 | MANUAL | ✓ Passed | | | 11.07.2005 |
| [1]T09 - Anmeldung L4, ohne Cookies | MANUAL | ✓ Passed | | | 08.08.2005 |
| [1]T10F - Anmeldung L4, falsche Kombination | MANUAL | ✓ Passed | | | 11.07.2005 |
| [1]T11 - Anmeldung L4, gleicher Nutzer parallel | MANUAL | ✓ Passed | | | 11.07.2005 |
| [1]T12 - Anmeldung L4, Formularziel HTTPS | MANUAL | ✓ Passed | | | 05.07.2005 |
| [1]T13 - Anmeldung L4, Secure Cookie | MANUAL | ✓ Passed | | | 05.07.2005 |
| [1]T14 - Anmeldung L3 und L4, gleicher Nutzer | MANUAL | ✓ Passed | | | 11.07.2005 |
| [1]T15 - Anmeldung L3 und L4, verschiedene Nutzer | MANUAL | ✓ Passed | | | 11.07.2005 |
| [1]T16 - An- und Abmeldung L4, Anmeldung L3 | MANUAL | ✓ Passed | | | 11.07.2005 |
| [1]T17 - Anmeldung L3 und L4, gleicher Nutzer parallel | MANUAL | ✓ Passed | | | 11.07.2005 |
| [1]T25 - Anmeldung, Plausibilitätstests | MANUAL | ✗ Failed | | | 08.09.2005 |

Last Run Result

Test 18 of 18 Server Time: 09:17 AM 09.08.05

Fertig Internet

Protokollierung der Testausführung

Dokumentation der Testfallausführung

- verwendete Testumgebung, getestete Software-Version, ...
- Nachweis über die Ausführung des Testfalls (wann, durch wen)
- Ergebnis des Testfalls (passed/failed)
- konkret verwendete Testdaten (z.B. Produkt-ID)
- erfolgt im Testmanagementwerkzeug oder in Dokumenten

Dokumentation gefundener Fehlerwirkungen

- siehe Abschnitt „Fehlerverfolgung“

Auch beim Re-Test:

- Dokumentation der erneuten Testfallausführung
- Dokumentation im Fehlereintrag

Re-Test

Re-Test nach einer Fehlerbehebung

- wiederholte Ausführung von Testfällen, die zu einem Fehler geführt hatten + Dokumentation im zugehörigen Fehlereintrag
- Ziel ist der Nachweis:
 - der ordnungsgemäßen Fehlerbeseitigung
 - dass unmittelbar betroffene Funktionalität nicht beeinträchtigt wurde
- erfolgt im Idealfall durch denjenigen, der den Fehler gefunden hatte

Re-Test sind sehr wichtig, da häufig

- Fehler nicht hinreichend behoben oder neue Fehlerzustände erzeugt wurden
- vorher maskierte Fehlerwirkungen auftreten
- weitere Fehlerwirkungen jetzt erst erreichbar sind

→ Re-Test möglichst zeitnah, da oft mehrere Iterationen nötig!

Hinweise zur Testausführung

Blick zur Seite angewöhnen

- Testinhalt = Testfälle + erfahrener Blick zur Seite
- Ausnahme: hochsensible Systeme, bei denen mit vielfachem Aufwand möglichst alles genauestens definiert wird
- Tests prüfen nicht nur, dass die geforderten Wirkungen vorhanden sind, sondern auch, dass unerwünschte Wirkungen nicht vorhanden sind

Variationen

- kleine Variationen bei jeder Testausführung unter Beibehaltung der Testintention
- gewählte Variante notieren, um potentielle Defekte reproduzieren zu können
- Testintention beachten, ein Grenzwerttest muss ein Grenzwerttest bleiben

Beispiele

- Modifikation von Eingabedaten innerhalb einer Äquivalenzklasse, z. B. jedes Mal ein anderes Datum oder ein anderes Produkt verwenden
- verschiedene Wege zum Auslösen einer Aktion (Button, Menü, ...), sofern dies nicht als Teil des Testfalls vorgegeben ist

Agenda

- Einführung
- Klassifikation von QS-Maßnahmen
- Testfallerstellung
- Ökonomie des Testens
- Systematische Testfallermittlung
- Testausführung
- Fehlerverfolgung
- Kurzüberblick Testmanagement

Fehlerverfolgung >

Fehlerverfolgung

Ziele

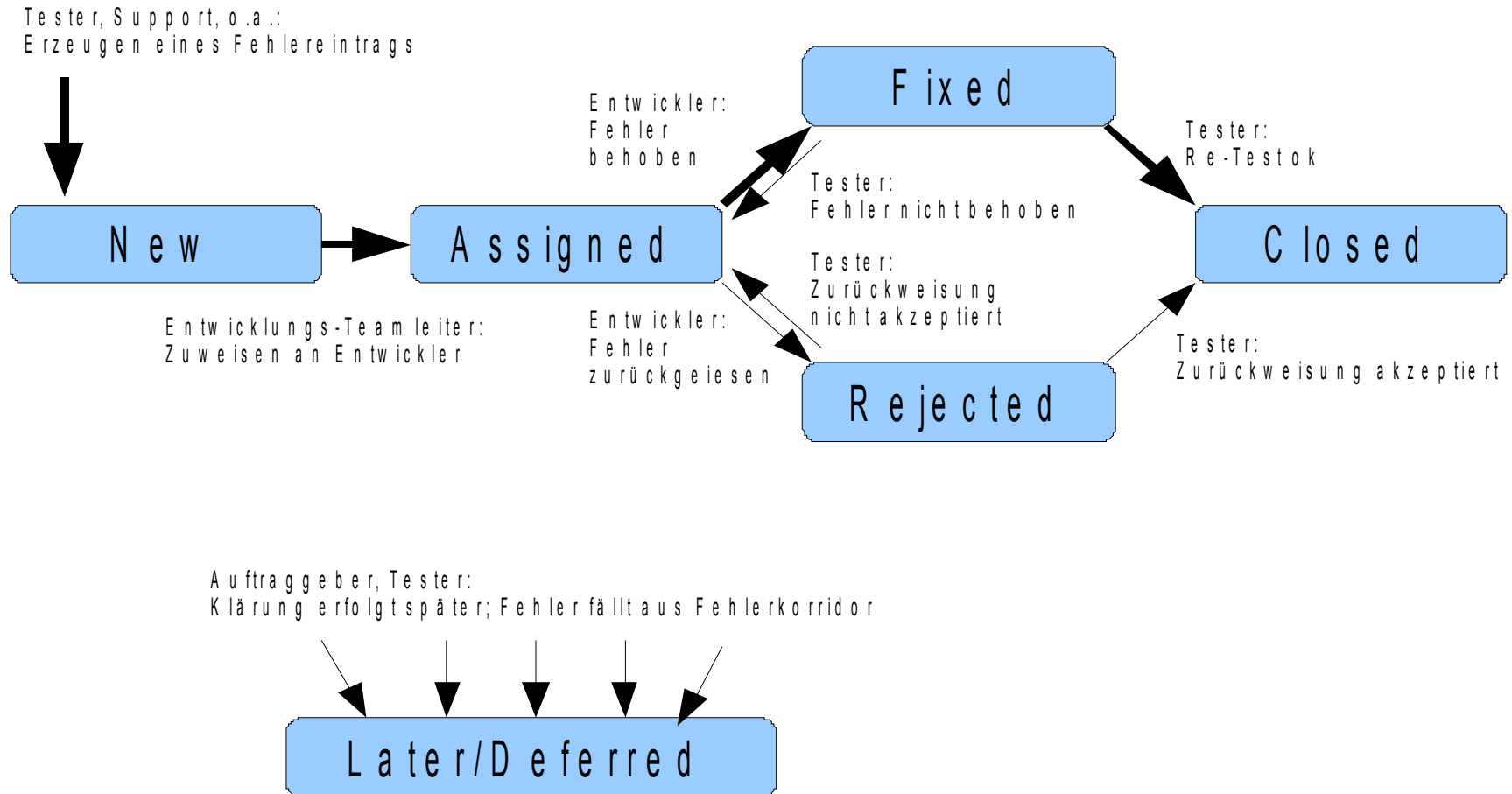
- strukturierte Kommunikation der gefundenen Fehlerwirkungen
- Nachverfolgung der Fehlerbeseitigung

Wie?

- mit Hilfe von Werkzeugen, die den Fehlerprozess abbilden
- für jeden Fehler wird ein Fehlereintrag angelegt
- jeder Fehlereintrag durchläuft einen Fehlerprozess
- Fehlerprozesse in der Praxis sehr unterschiedlich
- hier beispielhaft ein sehr einfacher Prozess behandelt

→ Fehlereinträge sind ein wichtiges Kommunikationsmittel eines Testers

Fehlerverfolgung > Einfacher Fehlerprozess



Fehlerverfolgung >

Zustände eines Fehlereintrags (1)

New (Startzustand, Aktion durch Entwicklung nötig)

- Fehlereintrag neu erzeugt und
- Fehlereintrag noch nicht von der Entwicklung angenommen

Assigned (Aktion durch Entwicklung nötig)

- Fehlereintrag einem Entwickler zur Bearbeitung zugeteilt oder
- Fehlereintrag zurück an die Entwicklung zur Nachbesserung gegeben

Fixed (Aktion durch QS nötig)

- Fehler behoben
- Re-Test durch QS bei Verfügbarkeit des Fixes auf Testumgebung notwendig

Closed (Endzustand)

- Re-Test erfolgreich oder
- Fehler von der Entwicklung begründet zurückgewiesen und vom Tester akzeptiert

Zustände eines Fehlereintrags (2)

Rejected (Aktion durch QS nötig)

- Fehler von Entwicklung zurückgewiesen
 - kein Fehler
 - älterer Fehlereintrag existiert bereits
 - Beschreibung nicht ausreichend
 - ...
- Tester kann dies akzeptieren (→ Closed) oder Fehler zurück an die Entwicklung geben (→ Assigned)

Deferred (Klärung später)

- Fehlerbehandlung wird aufgeschoben, ist aber später noch durchzuführen
- Fehlereintrag zählt nicht in den Fehlerkorridor des Tests (bei verbindlich vereinbartem Fehlerkorridor als Abnahmekriterium)
- Beispiel: Fehlerbehebung erfolgt mit späterem Projekt oder erstem Hotfix

Zustände eines Fehlereintrags (3)

Hinweise

- vollständig bearbeitete Fehler befinden sich im Zustand „Closed“; für Fehler in einem anderen Zustand sind noch Aktionen nötig
- Fehler, die sich am Ende des QS-Zeitraums nicht in „Closed“ oder "Deferred" befinden, werden im Testergebnis (Fehlerkorridor) berücksichtigt
- fett dargestellte Pfeile: „idealer Ablauf“ für einen tatsächlichen Fehler
- neben den dargestellten Zustandsübergängen sind weitere Übergänge möglich, allerdings sollten diese eine Ausnahme darstellen
- reale Fehlerprozesse sind sehr unterschiedlich gestaltet und haben oft viele weitere Zustände

Fehlerverfolgung >

Art der Fehlerbehandlung kennzeichnen

Wunsch

- Kennzeichnung, wie ein Fehler behandelt bzw. warum er geschlossen wurde

Schlecht

- neben „Closed“ weitere Endzustände einführen („No Defect“, „Duplicate“, ...)
 - ➔ unübersichtlich
 - ➔ viele Zustände beim Erstellen aktueller Fehlerlisten auszufiltern

Gut

- zusätzliches Feld, das die Art der Fehlerbehandlung beschreibt
- Feldname häufig „Solution“, „Resolution“ oder „Substate“
- Werte: „Fixed“, „Will Not Be Fixed“, „Duplicate“, „No Defect“, „Not Reproducible“, ...
- Nur ein Endzustand („Closed“), daher
 - ➔ Fehlerprozess einfacher und übersichtlicher
 - ➔ Fehlereinträge leichter filterbar

Felder eines Fehlereintrags (1)

Typische Felder eines Fehlereintrags

- **ID:**
eindeutiger Identifikator, meist numerisch, automatisch vergeben
- **Summary / Title / Short Description:**
aussagekräftige Kurzbeschreibung der Fehlerwirkung
- **Detected By / Opened by / Submitter / Creator:**
Person, die den Fehlereintrag erstellt hat
- **Version / Detected in Version:**
Softwareversion, in der der Fehler gefunden wurde
- **Severity:**
Schwere der Fehlerwirkung, initial gesetzt vom Ersteller
- **Detected on Date / Creation Date:**
Datum, an dem der Fehler gefunden wurde
- **State:**
globaler Bearbeitungszustand des Fehlers; bei Erstellung „New“

Felder eines Fehlereintrags (2)

- **Solution / Resolution:**
Behandlung des Eintrags, Grund für das Schließen
- **Description / Long Description:**
Beschreibung mit Schritten zur Reproduktion, genutzten Daten, Ergebnis
- **Assigned To:**
Person, die den Fehler weiter bearbeitet
- **Priority:**
Priorität der Behebung des Fehlers; meist von Entwicklung festgelegt; kann deutlich von der Severity abweichen
- **Reproducible:**
Reproduzierbarkeit; Prozentwert, „yes“/„no“ oder Stufen „always“, „often“...
- **Attachments:**
angefügte Screenshots, Testdaten, Logdateien, ...
- **Product / Project / Component / Feature / Subject / Topic:**
Angabe, in welcher Komponente der Fehlerzustand vermutet wird

Fehlerverfolgung >

Felder eines Fehlereintrags (3)

Defect Details

Defect: 2 Die Checkbox zur Bestätigung der AGB wird immer automatisch unchecked, sobald ein Eingabefeld

Details

* Detected By: rov
* Severity: 3 - mittel
Assigned To:
Modified:
Project:
Subject:

* Detected on Date: 15.02.2007
* Status: New
Detected in Version:
Priority:
Reproducible: Y

Planned
Planned Closing Version:
Estimated Fix Time: (Days)


Actual
Closed in Version:
Actual Fix Time: (Days)
Closing Date:


Execution Report

OK Cancel

Fehlerverfolgung >

Felder eines Fehlereintrags (4)



Anonymous | [Login](#) | [Signup for a new account](#)
2014-09-02 06:31 EDT
 Project: 

[My View](#) | [View Issues](#) | [Change Log](#) | [Roadmap](#) | [Wiki](#) | [Repositories](#)

View Issue Details [[Jump to Notes](#)] [[Wiki](#)] [[Related Changesets](#)] [<<] [>>] [[Issue History](#)] [[Print](#)]

| ID | Project | Category | View Status | Date Submitted | Last Update |
|---------------------------|---|-------------------------|-------------|------------------------|------------------|
| 0016554 | mantisbt | bugtracker | public | 2013-10-31 05:46 | 2014-02-04 10:39 |
| Reporter | RGM | | | | |
| Assigned To | dregad | | | | |
| Priority | normal | Severity | major | Reproducibility | always |
| Status | resolved | Resolution | fixed | | |
| Platform | | OS | | OS Version | |
| Product Version | 1.2.15 | | | | |
| Target Version | 1.3.x | Fixed in Version | 1.3.x | | |
| Summary | 0016554: Project privacy change from public to to private kicks manager out | | | | |
| Description | When a project manager changes the privacy from public to private, he can't see the project anymore.
The user has global manager rights and also in the project. | | | | |
| Steps To Reproduce | Create public Project or subproject
Change privacy to private
--> No user except the global admin has access to the project | | | | |
| Tags | No tags attached. | | | | |
| Attached Files | | | | | |

Erstellen eines Fehlereintrags (1)

1. Vor Erstellen eines Fehlereintrags vorhandene suchen!

- existiert ein noch nicht geschlossener Fehlereintrag zum gleichen Problem, keinen neuen erstellen, sondern eventuell Kommentar zufügen
- Geschlossene Fehlereinträge nicht wieder öffnen!
- Tipp: Täglichen Überblick über alle aktuellen Fehlereinträge verschaffen!

2. Vor Erstellen eines Fehlereintrags prüfen, ob das Fehlverhalten durch inkorrekte Testdaten verursacht wird.

- im Zweifelsfall mit weiteren Datensätzen prüfen
- wichtig bei „datenlastigen“ Geschäftsanwendungen



© Giant Microbes

Erstellen eines Fehlereintrags (2)

3. Sammeln der Informationen für den Fehlereintrag

- Testschritte, die zum Fehler führen („Schritte zum Reproduzieren“)
- erwartetes Ergebnis („Soll:“)
- tatsächliches Ergebnis, wenn erforderlich („Ist:“)
- Fehlermeldungen, Screenshots, Logeinträge...
- benutzte Testdaten
- Testumgebung (Versionen, Betriebssysteme, Webbrowser, Datenbank...)

4. Fehlereintrag erstellen

- Fehler dabei am besten noch einmal reproduzieren



© Giant Microbes

Fehlerverfolgung >

Erstellen eines Fehlereintrags (3)

5. Wurde der Defekt zufällig entdeckt

- passenden Testfall erstellen oder
- einen Testfall passend aktualisieren

6. Vor Testfortsetzung wieder definierten Zustand herstellen

- Beispiel: Bei einem Fehler im Session Handling vor Testfortsetzung Cookies löschen, Browser schließen, Browser neu starten und neue Session starten.



© Giant Microbes

Fehlerbeschreibung

Klare Kurzbeschreibung, die die Fehlerwirkung benennt!

■ GUT:

- „Inkorrekte Fehlermeldung bei falsch eingegebenem Passwort“
- „Löschen von Adressen nicht möglich“
- „Memory Leak in xyz.exe bei Suchanfragen“

■ SCHLECHT:

- „Aktualisierung des Bearbeiten-Menüs“
- „Fehler bei Registrierung“

Klare Beschreibung!

- Je weiter man den Fehler vor der Erstellung des Eintrags für sich analysiert, desto kürzer und treffender wird die Beschreibung sein.
- anhand der Beschreibung sollte der Fehler reproduzierbar sein
- Irrelevantes weglassen, komplizierte Beschreibungen verwirren

Fehlerschwere – Severity (1)

Beispiel für Severities

■ Sehr schwer

- Nutzung einer Grundfunktion nicht sinnvoll möglich
- häufiger Absturz bei Ausführung einer Grundfunktion
- Software lässt sich nicht installieren
- Datenverlust
- Berechnungsfehler
- Sicherheitsproblem

■ Schwer

- nicht-essentielle Funktion nicht verfügbar oder stark gestört
- Berechnungsfehler in weniger relevanten Funktionen
- Daten im falschen Format gespeichert

Fehlerschwere – Severity (2)

Beispiel für Severities (fortgesetzt):

- Mittel
 - Funktion eingeschränkt, aber grundsätzlich nutzbar
 - keine Plausibilitätsprüfung von Eingabewerten
 - schwerer Fehler, für den es einen Workaround gibt
- Leicht
 - kleinere, unwesentliche Funktionseinbussen
 - kleine, unauffällige Layoutfehler
 - Schreibfehler an unwesentlichen Stellen
- zusätzlich zu den Fehlerklassen eventuell auch Klasse „Hinweis“
 - wenn bereits klar ist, dass angemerktetes Verhalten keine Anforderung war
 - Probleme in Bereichen, die kein Testobjekt sind, z.B. gefühlte Performance schlecht oder ungünstiger Bedienungsablauf

Fehlerverfolgung >

Hinweise zum Eintragen von Fehlern

- Fehler sofort eintragen!
 - ➔ Details gehen nicht verloren
 - ➔ mehr Zeit für Bearbeitung und Re-Test
 - ➔ Fehler einfacher gemeinsam reproduzierbar
- Nie davon ausgehen, dass ein offensichtlicher Fehler bereits eingetragen ist!
- Auch schwer reproduzierbare Fehler eintragen, sie könnten Zeitbomben sein!
- Für jeden Fehler einen eigenen Fehlereintrag erzeugen, keine „Sammelfehler“
 - ➔ Nur so kann der Status für jedes Problem separat verfolgt werden.
- Auch kleinere Fehler sind es wert, eingetragen zu werden!
- Genutzten Testdatensatz nicht für weitere Tests modifizieren, da sonst evtl. die Fehlerwirkung nicht mehr reproduziert werden kann!
- Fehlereinträge sollten für alle Interessenten zugänglich sein (Entwickler, Test, Support, Produktmanager, ...); für bestimmte Gruppen auch nur lesend

Fehlerverfolgung > **Fehlerkultur**

Gute Fehlerkultur wahren!

- sachliche Darstellung
- Übertreibungen vermeiden
- Ausrufungszeichen vermeiden
- keine Schuldzuweisungen
- keine persönliche Betroffenheit ausdrücken



- **Tester werden oft an der Qualität ihrer Fehlereinträge gemessen.**
- **Der beste Tester ist der, dessen Fehlereinträge auch behoben werden!**

Agenda

- Einführung
- Klassifikation von QS-Maßnahmen
- Testfallerstellung
- Ökonomie des Testens
- Systematische Testfallermittlung
- Testausführung
- Fehlerverfolgung
- Kurzüberblick Testmanagement

Testphasen

Testplanung (Testmanager)



Testvorbereitung (Testmanager, Tester)



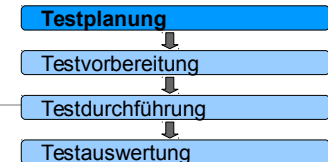
Testdurchführung (Testmanager, Tester)



Testauswertung (Testmanager)

Testplanung >

Testplanung – Inhalte (1)

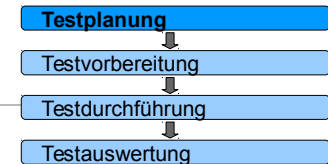


Hauptaufgaben des Testmanagements

- Abstimmung mit dem Auftraggeber des Tests
 - grundsätzliche Projektinhalte
 - zu testende Qualitätsmerkmale
 - zu testende Leistungsmerkmale (welche Systeme, Features, Konfigurationen, ...)
 - nicht zu testende Leistungsmerkmale, sofern Abgrenzung notwendig
 - Qualitätsziele
 - Zeiträume, Zyklen
 - Budget
- Abschätzung von Testaufwänden, Planung von Zeiten
- Festlegung der Teststrategie
- Festlegung von Testumfang, Teststufen und Testarten
- Festlegung von Testende- oder Abnahmekriterien

Testplanung >

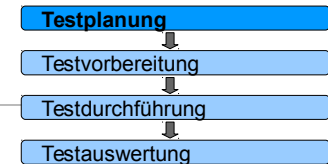
Testplanung – Inhalte (2)



Hauptaufgaben des Testmanagements (fortgesetzt)

- Festlegung der Testvoraussetzungen
 - zu verwendende Soll-Vorgaben (explizite Anforderungen)
 - sonstige Dokumentation
 - Testumgebung (client- und serverseitige Hardware, Software, Zugänge...)
 - besondere Testdaten
- Festlegung der Aufgaben, Verantwortlichkeiten, Rollen
- Kommunikationswege (wer was an wen), Eskalationswege und Meetings
- Festlegung der Testfallverwaltung
 - Testobjekthierarchie und Testfalldesign
 - Testausführungen
- Festlegung der Fehlerverwaltung
- Erstellung eines Testkonzepts
- ...

Testkonzept



Testkonzept ist zentrale Planungsgrundlage des Testprojektes

- enthält alle für den Test relevanten Festlegungen
 - organisatorisch
 - fachlich
 - technisch
- Festlegungen sind mit dem Auftraggeber abzustimmen
- mit eventuellem Angebot und Abschätzungen abgleichen → abweichende Empfehlungen mit Ersteller Angebot und gegebenenfalls mit Auftraggeber klären

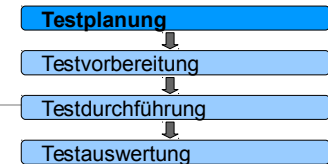


Rahmentestkonzept

- für wiederkehrende Projekte und kontinuierliche Softwareentwicklung:
Rahmentestkonzept mit dauerhaft gültigen Festlegungen (→ „Testhandbuch“)
- je Software-Release zusätzliche spezifische Festlegungen möglich, z.B.
zu Testobjekten, Qualitätsmerkmalen und damit verbundenen Testarten

Testplanung >

Testende-/Abnahmekriterien (1)



Typische Metriken

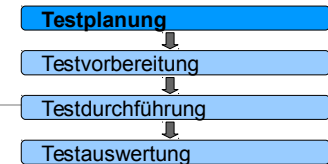
- Fehlerstatus: nach Fehlerschwere und gegebenenfalls Priorität
- Testfallstatus: offene, erfolgreiche und fehlgeschlagene Testfälle

Beispielhafte Abnahmekriterien

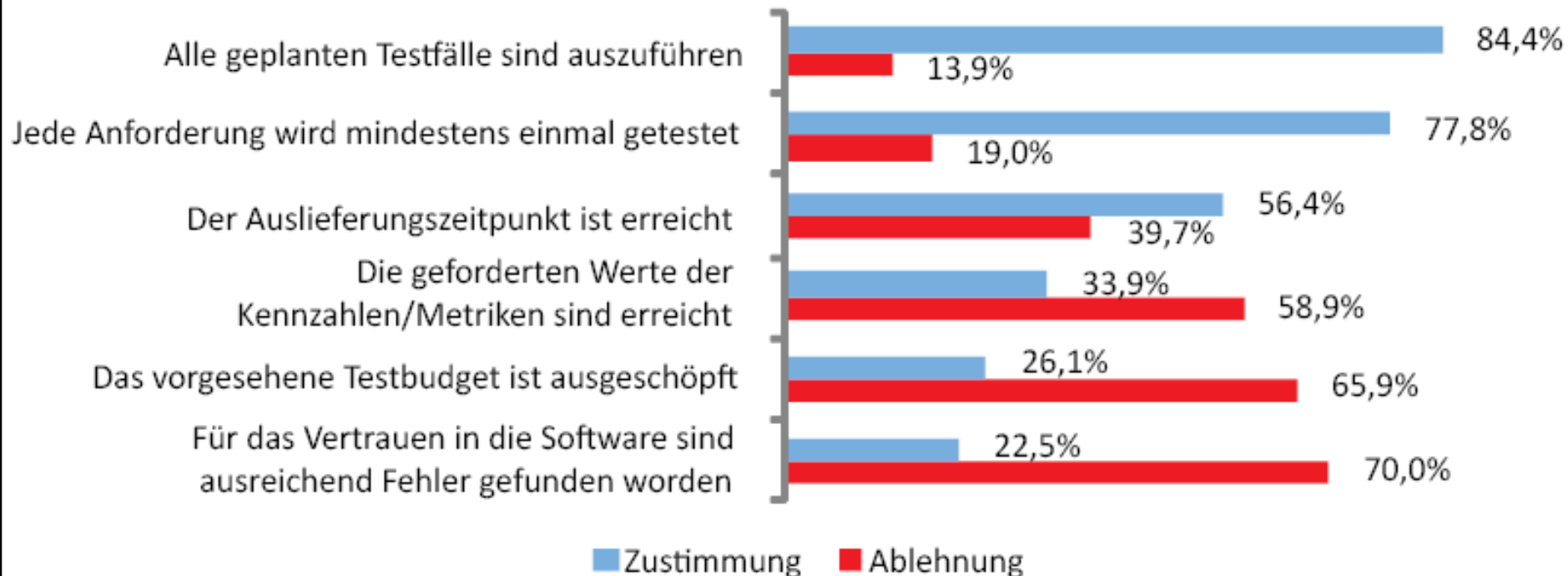
- alle geplanten Testfälle wurden mindestens einmal ausgeführt
- alle aufgetretenen Fehler sind im verwendeten Werkzeug dokumentiert
- Fehlerkorridor eingehalten, z.B. bei vierstufiger Fehlerschwere: 0/0/10/25
 - max. 0 Fehler mit Fehlerschwere „sehr schwer“ oder „schwer“
 - max. 10 Fehler mit Fehlerschwere „mittel“
 - max. 25 Fehler mit Fehlerschwere „niedrig“
- Fehlerkorridor bezieht sich auf „offene“ Fehler = noch nicht erfolgreich re-testet
- Go-/No-Go-Empfehlung meist auf Basis der offenen Fehler, nicht der fehlgeschlagenen Testfälle

Testplanung >

Testende-/Abnahmekriterien (2)



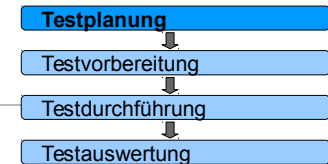
Zur Beendigung der Tests werden folgende Kriterien verwendet ...



© Softwaretestumfrage 2011: HS Bremen, HS Bremerhaven, FH Köln, ANECON, GTB, STB

Testplanung >

Planung von Zeit + Aufwand (1)



Planungsprinzip Bottom Up

- Aufwände für Teilaufgaben im Detail definieren
- Teilaufwände zu geschätztem Gesamtaufwand konsolidieren
- Verfügbarkeit der Tester ermitteln
- Aufwände abschätzen + verfügbare Tester einplanen → möglicher Termin

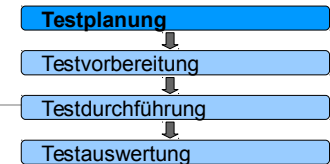
Planungsprinzip Top Down

- Zieltermine und Budget werden vorgegeben → Planung ist einzupassen
- $\text{budgetierte Aufwände} / \text{verfügbare Zeitspanne} = \text{benötigte Tester}$
- Beispiel: 100 PT Aufwand / 20 Arbeitstage Zeit = 5 Tester
- Feste und variable Faktoren: Aufwand, Teamgröße, Zeitspanne, Wartezeiten, ...

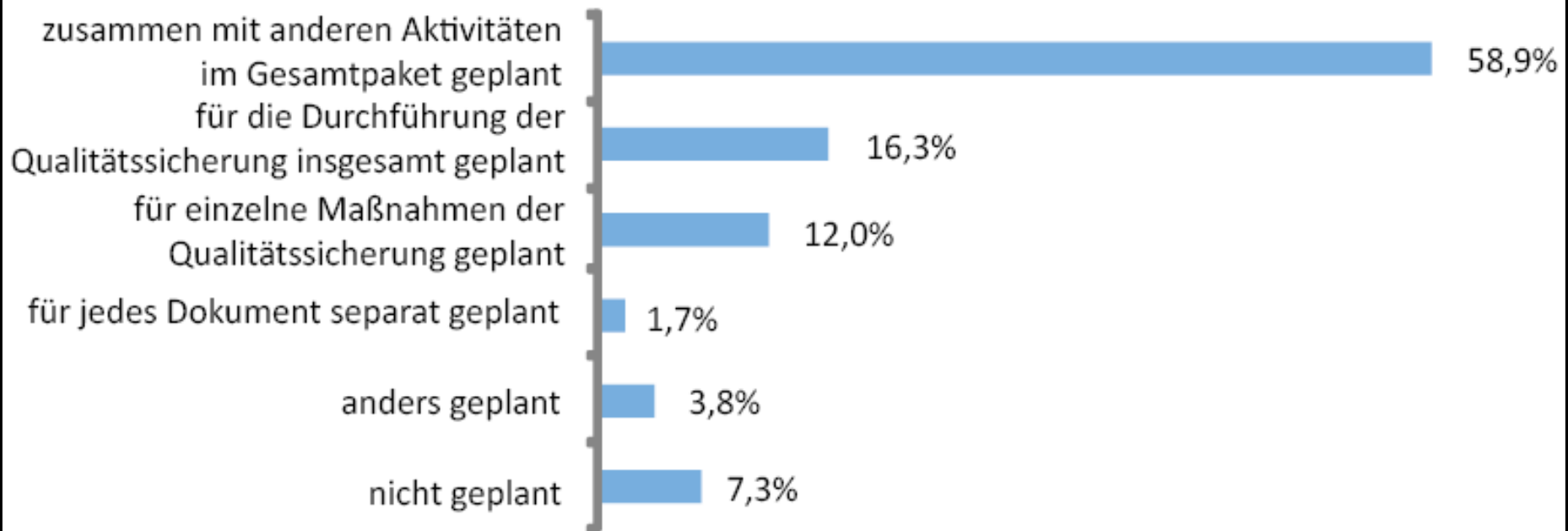


Testplanung >

Planung von Zeit + Aufwand (2)



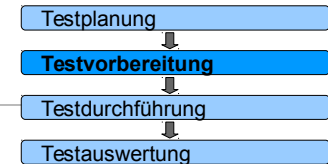
Der Aufwand (Budget und Zeit) der Qualitätssicherung wird in Ihren Projekten ...



© Softwaretestumfrage 2011: HS Bremen, HS Bremerhaven, FH Köln, ANECON, GTB, STB

Testvorbereitung >

Testvorbereitung – Inhalte



Hauptaufgaben des Testmanagements

- Detaillierte Testplanung
- weitere Abgrenzung und Gliederung der Testobjekte (Testobjekthierarchie)
- Priorisierung
- Risikoanalyse und -verfolgung
- Konfiguration von Tools zur Verwaltung der Testfälle+Testausführung, Fehler
- Konfiguration von Tools zur Erzeugung der Testberichte und Statistiken
- Bereitstellung der Testumgebung

Hauptaufgaben der Tester

- Erstellung der Testfälle (Testspezifikation)
- Definition von Testdaten
- Bereitstellung der Testumgebung

Testvorbereitung >

Feinplanung Testdurchführung

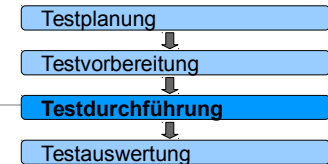


Feinplanung der Testdurchführung pro Testobjekt

- meist iterativ
- Smoketests
- Testobjekte/Testarten nach Priorität einplanen
- Aufwand und gegebenenfalls zeitliche Dauer und Tester einplanen:
 - Tester gleich Testdesigner (am effizientesten)
 - Tester ungleich Testdesigner
- Software-Lieferungen (Häufigkeit, Termine, Dauer Deployments)
- Re-Tests:
 - während Testdurchführung parallel zur Testfallabarbeitung einplanen
 - Zeitraum, gegebenenfalls Ende der Testdurchführung jedes Testobjektes oder pauschal (nicht mehr differenziert nach Testobjekten)
- Regressionstests

Testdurchführung >

Testdurchführung – Inhalte



Hauptaufgaben des Testmanagements

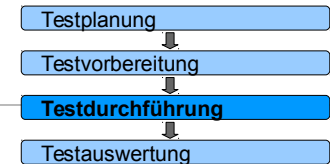
- Aufgabenverteilung
- Steuerung und Überwachung des Testfortschritts und Budgets
- Risikomanagement, Priorisierung
- Überprüfung, ob die Eingangskriterien für die jeweilige Teststufe erfüllt sind
- Überprüfung der Testergebnisse
- Erstellung von zyklischen Testberichten, Statistiken und Metriken
- Kommunikation im Projekt

Hauptaufgaben der Tester

- Durchführung der Tests nach Testspezifikation
- Dokumentation der Testergebnisse
- Erstellung von Fehlermeldungen
- gilt auch für Smoketests, Re-Tests und Regressionstests

Testdurchführung >

Monitoring Testdurchführung



Statusmonitoring

- Ampelstatus, Probleme/Hindernisse
- Fehlerstatistik
- Testfallstatistik
- Beispiel:
 - 100 Testfälle geplant
 - 40% schlagen beim ersten Durchlauf fehl (40)
 - 10% davon (4) schlagen beim zweiten Durchlauf fehl
 - => 144 Testfall-Ausführungen in Summe
- bei Bedarf Fehler- und Testfallverlauf (Trend) und Durchsatz erfassen

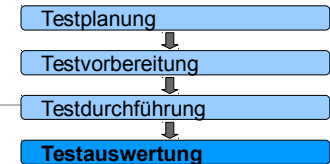


Risikoanalyse und -verfolgung

- Risiken benennen, gewichten
- Maßnahmen vorschlagen

Testauswertung >

Testauswertung (1)

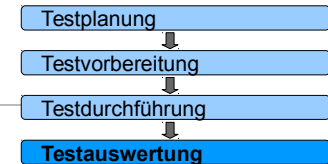


Hauptaufgaben des Testmanagements

- Bewertung der Fehlerentwicklung und Fehlerendstandes
- Empfehlung zur Abnahme/Go Live des getesteten Softwaresystems
- gegebenenfalls Erstellung eines Testabschlussberichts
- Archivierung aller Testergebnisse
- Projektabschlussmaßnahmen
 - Nachkalkulation
 - Abnahmeprotokoll
 - ...
- Lessons Learned mit Testteam und Entwicklung

Testauswertung >

Testauswertung (2)



Testat (formell)

- Was wurde getestet?
- Wo wurde getestet?
- Wieviele Testfälle mit welchem Ergebnis?
- Fehlerkorridor
- Bewertung Testziel



Bericht (informativ)

- Was wurde getestet?
- Wo wurde getestet?
- Endstatus Testfälle und Bewertung Bewertung der Fehlerentwicklung und Fehlerendstandes
- Resümee über Test - was war gut, was war schlecht
- Bsp.: Zusammenarbeit Entwicklung, Softwarereife bei Übergabe in den Test
- Wieviel Aufwand mit welchem Ergebnis?

Vielen Dank für Ihre Aufmerksamkeit!

Haben Sie Fragen?

Kontakt

Xceptance Software Technologies GmbH
Leutragraben 2-4
07743 Jena

Tel.: +49 (0) 3641 55944-0
Fax: +49 (0) 3641 376122

E-Mail: info@xceptance.de
<http://www.xceptance.de>

Referenzen

Bücher

- Andreas Spillner, Tilo Linz: „Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester - Foundation Level nach ISTQB-Standard“, dpunkt.verlag GmbH, 5. Auflage, 2012
- Andreas Spillner, Thomas Roßner, Mario Winter, Tilo Linz: „Praxiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester - Advanced Level nach ISTQB-Standard“, dpunkt.verlag GmbH, 3. Auflage, 2012
- Glenford J. Myers, 1979, revised and updated by Tom Badgett and Todd M. Thomas with Corey Sandler, 2004: „The Art of Software Testing, 2nd Edition“

Studie/Umfrage

- Peter Haberl, Prof. Dr. Andreas Spillner, Prof. Dr. Karin Vosseberg, Prof. Dr. Mario Winter: „Softwaretest in der Praxis, Umfrage 2011“, dpunkt.verlag GmbH, 1. Auflage, 2012, <http://www.softwaretest-umfrage.de> (PDF verfügbar)

Copyrights

Alle für die Vorlesung zur Verfügung gestellten Unterlagen unterliegen dem Copyright und sind ausschließlich für den persönlichen Gebrauch im Rahmen der Vorlesung „Qualitätssicherung von Software“ freigegeben. Die Weitergabe an Dritte und die Nutzung für andere Zwecke sind nicht erlaubt.