Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN

HOCHSCHULE LUZERN - INFORMATIK

Software Testing (SWT)

Zusammenfassung von Oliver Amstutz, 5. März 2021

5. März 2021

INHALTSVERZEICHNIS

1 SW01

1.1 Getestet wird immer

Getestet wird immer, es bleibt nur offen:

- Wann wird getestet?
- Wer testet?
- Wo werden die Test Resultate veröffentlicht?

Es lohnt sich das Testen vor der Entwicklung einzuplanen (Kosten, Zeitpunkt, Image Schaden). Beispiele sind:

- Unzureichender Last-Test für Ski-Gebiete Online Buchung
- Sirenentest musste wegen Systemfehler wiederholt werden
- Oberflächenbehandlungsmaschine welche bereits mit ungenügenden Test geplant wurde (Funktion Düsen nicht interdisziplinär getestet, Positionsensor nicht auf voller Leistung getestet wegen Lärm)

1.2 Die 10er Regel

Die Abbildung 1.1 besagt, dass je weiter ein Fehler sich unentdeckt in die späteren Phasen des Werdeganges eines Produktes oder Prozesses bewegt, umso höher werden die Kosten zur Behebung dieses Fehlers [1].

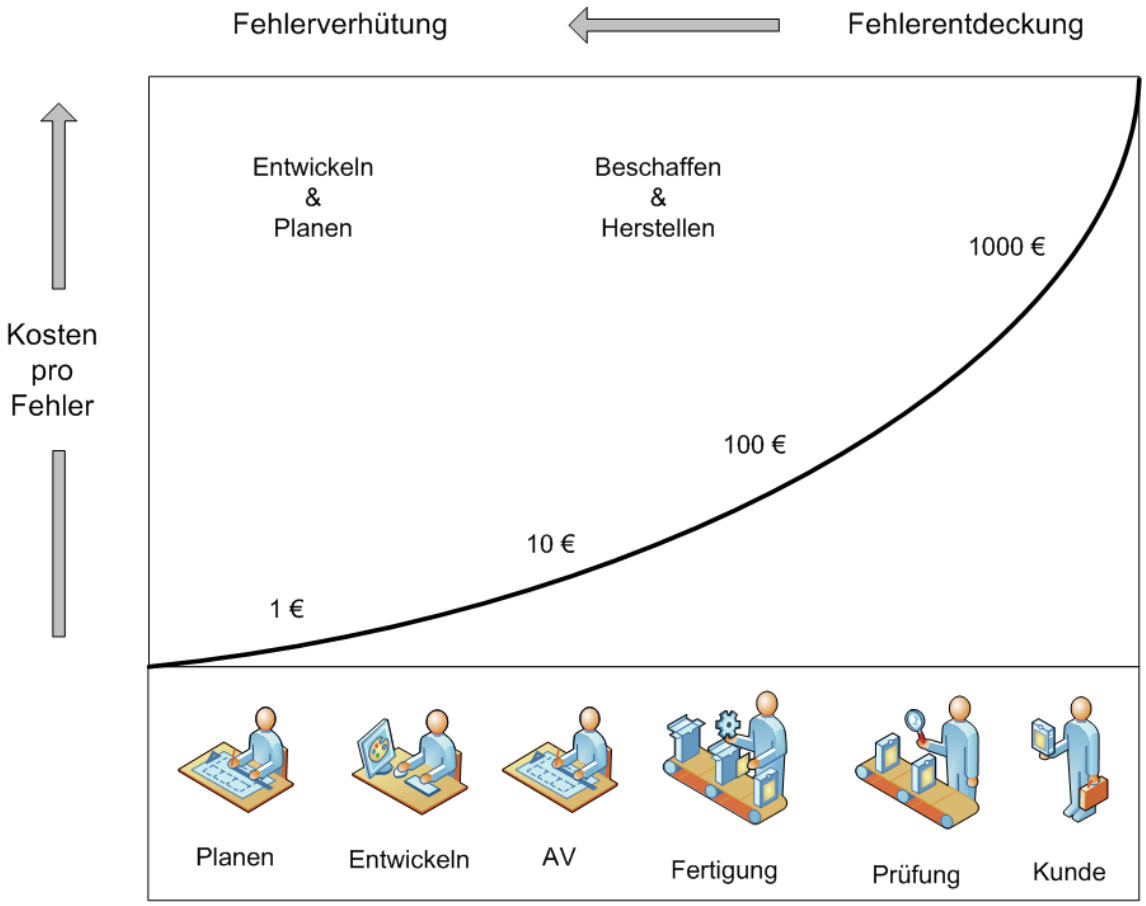

Abbildung 1.1: Fehler-kosten der 10er Regel (rule of ten).

Beispiele:

- Beispiele aus Unterkapitel 1.1
- Heroes of Newearth Game-Entwicklung: nicht-Initialisierung einer Variable hat im 64Bit System zum Fehler geführt, welches erst beim Release durch die User bemerkt wurde. Rollback war notwendig.

Je früher ein Fehler erkannt wird, desto billiger ist die Behebung!

1.3 Testen bringt Nutzen

Mehr Qualität:

- wird messbar und quantifizierbar und dadurch sichtbar und verkaufbar
- was messbar ist, wird steuerbar
- kann somit gezielt gesteigert werden

Mehr Wirtschaftlichkeit:

- Frühes Erkennen von Fehler ist billiger
- After Sales Support wird weniger (Einsparen von Kosten)
- Testen dokumentiert Wissen und schützt so Investitionen
- Ressourcen werden sinnvoll eingesetzt

Risikomanagement:

- Risiken werden frühzeitig identifiziert
- Risiken werden aktiv gemanagt
- Das Eintreffen der Risiken kann vermieden werden
- Es wird dort getestet, wo es sinnvoll ist (in der Regel wird das getestet was man: kennt, schon immer getestet hat, gut bekannt ist)

Besserer Wettbewerbsvorteil:

- keine negativ Presse
- hohe Reputation dank Zuverlässigkeit
- Erweiterbarkeit und Wartbarkeit ergeben Nachhaltigkeit
- swissness, Qualitätsbewusstsein

1.4 Testqualität vs. Produktqualität

Abbildung 1.2 zeigt auf der X-Achse die Testqualität und auf der Y-Achse die Produktqualität [2].

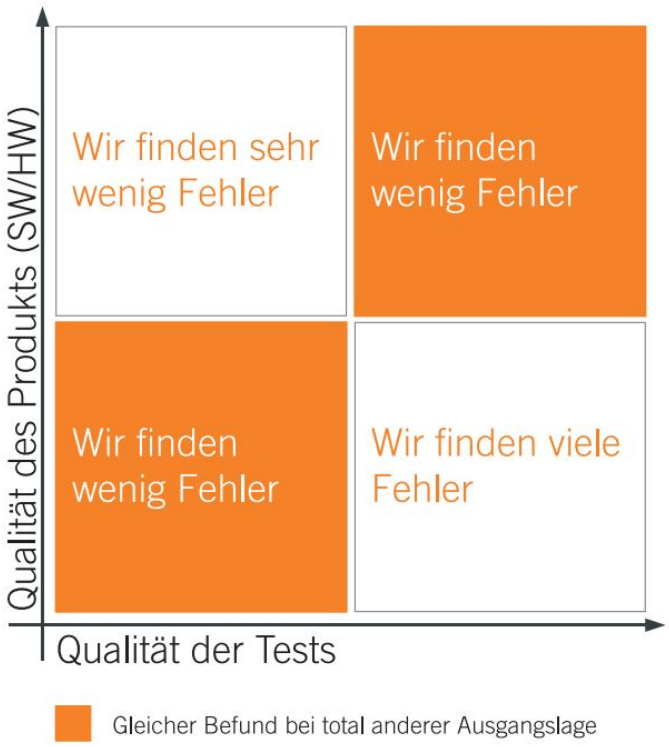

Abbildung 1.2: Testqualität gegenüber Produktqualität.

Aus der Abbildung lassen sich folgende Rückschlüsse ziehen:

- Testqualität ist entscheidend, nur "halbes Testing"bringt nichts!
- Der Grund fürs Testen ist es herauszufinden ob man eine gute oder schlechte Produktqualität hat.
- Ist die Testqualität schlecht, können keine Rückschlüsse auf die Produktqualität gewonnen werden -> Provokativ: Warum testet man dann überhaupt?
- Man findet sehr wenig Fehler -> gute Entwickler, schlechte Tester
- Man findet sehr viele Fehler -> schlechte Entwickler, gute Tester
- Aufpassen da man den gleichen Befund haben kann bei total unterschiedlicher Ausgangslage!
- Je besser das Testen, desto schlechter erscheint die Produktqualität
- Als Konsequenz kann man also sagen - nur die besten testen!

Das Bewerten der Fehlerzahl kann über das Messen der gefundenen Fehler im Verlaufe des Projektes bewertet werden. Abbildung 1.3 zeigt das Aufkumulieren der Fehler über die Zeit. Die X-Achse ist die Zeit, die Y-Achse sind alle gefundenen Fehler.

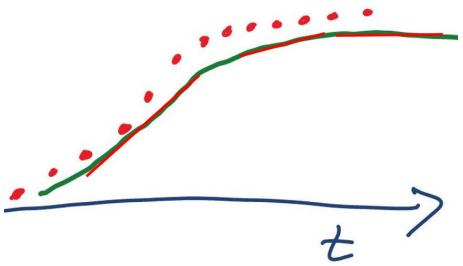

Abbildung 1.3: Fehlerkumulation eines SW-Produktes über die Zeit.

Wenn der Gradient der gefundenen Fehler (rote Linien) flacher wird, ist dies ein Zeichen dafür, das mit den angewandten Methoden keine weiteren Fehler gefunden werden können. Das kann auch so interpretiert werden, das in Abbildung 1.2 man die Qualität des Produktes gesteigert hat. Das sich die Qualität der Tests verschlechter über Zeit wird nicht angenommen. Weitere Info's zu diesem Thema in Kapitel Noch zu definieren - später!.

1.5 Anforderungen der Stakeholder

Stellen Sie sich ein Glas / Ball vor. Ich kann ein Glas/Ball nicht testen ohne die Anforderungen der Stakeholder zu kennen. Dies beinhaltet:

- Anwendungszweck
- Umgebung
- Bedürfnisse
- Kontext

Ein Softwaretest prüft und bewertet Software auf Erfüllung der für ihren Einsatz definierten Anforderungen und misst ihre Qualität. Fürs Testen braucht's Erwartungen, Anforderungen und Akzeptanzkriterien!
Noch am passenden Ort einfüllen: Code qualität bei der Entwicklung bereits prüfen mit:

- 4 Augen Prinzip
- Codereview
- Pair-Programming

Die erste Teststufe liegt bereits beim Software Entwickler!

2 SW02

2.1 Testen vs Experimentieren

Testen unterscheidet sich von Experimentieren dadurch, dass es beim Testen eine Erwartung gibt die belegt werden soll, während das Ergebnis beim Experimentieren offen ist oder nur vermutet werden kann:

- Anforderungen (= erwartetes Verhalten) müssen bekannt sein
- Testen bedingt Anforderungen und prüft auf Unterschiede
- ohne Anforderungen ist Testen schwierig bis unmöglich
- Der Kontext vom Testen ist wichtig (Formale Nachweise, zb für Flugzeugindustrie, benötigen einen viel höheren formellen Aufwand für das selbe in einem anderen Kontext)

2.2 Agiles Testen

Vorteile von agilem Vorgehen:

- kurze Zyklen bringen schnell Feedback
- CI / CD unterstützen schnelles Feedback
- Jede Teillieferung bringt Nutzen
- Wichtiges erfolgt zuerst

Gefahren bei agilem Vorgehen:

- Ganzheitlicher (Test-) Ansatz fehlt (= keine Teststrategie)
- Keine Negativ-Tests
- Keine Grenzwerttests
- Keine NFA – Tests (Nicht Funktionale Anforderungen)
- Automatisierung erfolgt zu spät weil Produkt zulange instabil ist

Aus Test Sicht wichtige Punkte welche Aufmerksamkeit benötigen beim agilen Vorgehen:

- PO hat (zu) grosse Verantwortung (und oft wenig bis kein Testwissen)
- Ein Team kann «Alles», jede und jeder im Team können «Alles»
- Es gibt Akzeptanzkriterien (AK)
- Es gibt eine DOD (Definition of Done) und teilweise eine DOR (Definition of Ready: Wenn die Product Backlog Items "ready"für den Sprint sind [3])
- Es darf / muss Teststories[1] geben, diese beinhalten folgende Tätigkeiten:
  - Codereview
  - Testanalyse (Was muss getestet werden)
  - Testdesign (Wie wird getestet, wie sieht der Testablauf aus)
  - Tests bauen (auch automatisierte)
  - Testdaten (Passende, realistische Testdaten definieren und bereitstellen)
  - Tests durchführen

2.3 Rollen/Tätigkeiten im Testing

Abbildung 2.1 zeigt einen Vorschlag von möglichen Rollen mit den jeweiligen Verantwortlichkeiten und notwendigen Skill Set.

[1] Teststory ist ein Bestandteil einer User Story. Weiter Bestandteile können sein: Technical Story, Infrastructure Story, ...

Testmanager Führung	• Ansprechpartner für PL und Management in kritischen Phasen • Plant Ressourcen, sucht verschiedene Lösungen und bewertet sie • Kann Leute mit unterschiedlichen Hintergründen führen • Entwickelt Teststrategien • Ist verlässlich und integer
Testarchitekt, Testengineer Ingenieur	• Plant und entwickelt eine den Projekten angepasste Testinfrastruktur • Entwickelt und verbessert die Testmethoden • Entwickelt und verbessert Testwerkzeuge • Entwickelt Teststrategien und bringt sie in die Projekte ein
Testanalyst Ingenieur	• Leitet aus Anforderungen Testszenarien ab • Entwickelt komplexe Testabläufe • Bestimmt die notwendigen Testdaten
Testdatenverantwortlicher Informatiker	• Pflegt, verwaltet und bewirtschaftet die Testdaten • Kommuniziert mit Testanalysten, den Fachvertretern sowie der IT • Betreibt Tools und Hilfsmittel zur Bewirtschaftung der Testdaten • Erweitert die Testdaten auf Basis der fachlichen Möglichkeiten
Tester Fachperson	• Führt zuverlässig und exakt Tests aus • Dokumentiert präzis und wertfrei/neutral die Ergebnisse sowie die Abweichungen • Reproduziert auf Wunsch die Testfälle • Unterstützt die Testanalysten und IT-Spezialisten bei der Fehlersuche

Abbildung 2.1: Vorschlag von 5 möglichen Rollen/Tätigkeiten im Testing.

2.4 Test-Prozess

Prozesse definieren Abläufe, Rollen und Verantwortlichkeiten klären. Orientiert man sich beim Testen an einem Prozess, wird sichergestellt, dass Tests:

- geplant
- vorhersehbar
- gesteuert
- nachvollziehbar
- systematisch

sind. Es soll dabei beachtet werden das der Prozess nur ein Hilfsmittel ist. Beispiel: Zertifizierte Schwimmwesten aus Beton sind zwar nach Prozess erzeugt, jedoch überhaupt nicht zweckmässig. Abbildung 2.2 zeigt eine Übersicht, wo der Testprozess einzuordnen ist.

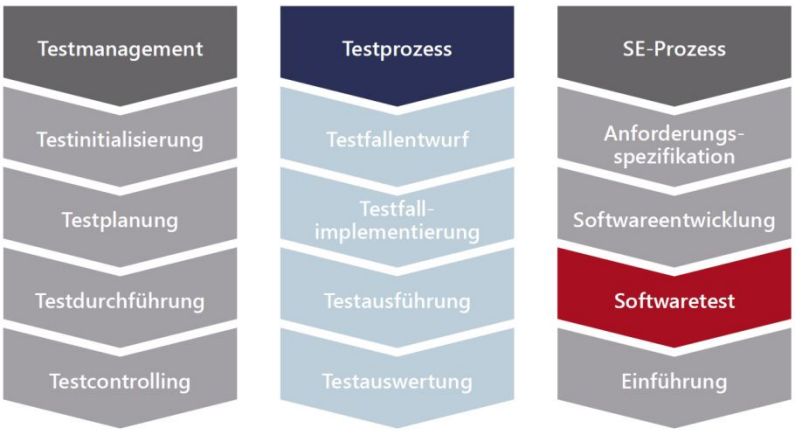

Abbildung 2.2: Übersicht, wo der Testprozess in der Software Entwicklung (SE) Einzuordnen ist.

Nachfolgend in Abbildung 2.7 sind die vier Phasen des Testen's gezeigt.

2.5 Testprozess, Phase 1: Analysieren

Abbildung 2.8 zeigt die Details in Phase 1.

Abbildung 2.3: Phase 1: Analysieren.

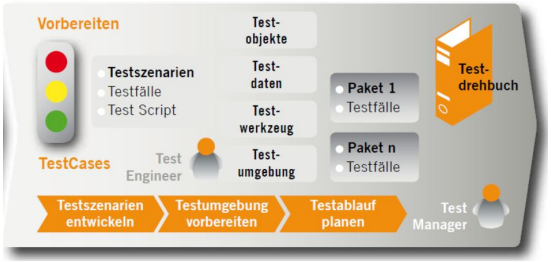

Abbildung 2.4: Phase 2: Vorbereiten.

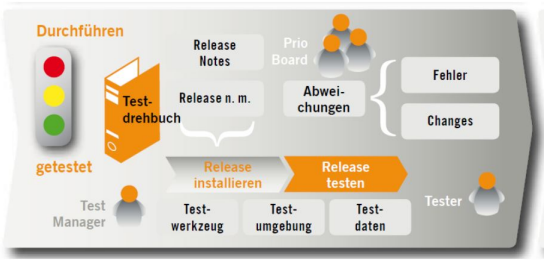

Abbildung 2.5: Phase 3: Durchführen.

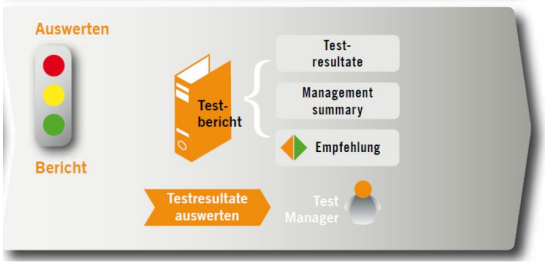

Abbildung 2.6: Phase 4: Auswerten.

Abbildung 2.7: Die 4 Phasen des Testprozesses von Noser [2].

Abbildung 2.8: Phase 1: Abdeckung Analysieren.

2.6 Teststrategie

Grundsätzlich bestimmt eine Teststrategie, ob, was und wie tief getestet werden soll. Die Teststrategie bestimmt das Test-Ziel und weist dessen Nutzen aus. Mit der Teststrategie setzt man Prioritäten, welche Bereiche wichtig sind zu testen. Prioritäten setzt man, da man weder unendlich Zeit noch unendlich Geld hat.
Aus folgenden Fragen kristallisiert sich die Teststrategie:

- Was kann im Fehlerfall passieren? (Menschenleben, Gesundheit, Umwelt, Reputation)
- Frage der Haftung und Garantie? (Nach EU Recht hat man 2 Jahre Garantie auf SW!)
- Wie lange «lebt» das Projekt / Produkt?
- In welchem Umfeld wird es eingesetzt? (Bsp: Billetautomaten Regen + Touch Screen / Sonneneinfall + Sichtbarkeit von Screen)
- Wie «komplex» ist das Produkt?
- Was kostet das Produkt? (Testen muss im Verhältnis sein zum erwarteten Umsatz)

- Gibt es Gesetze , Vorschriften oder Normen? (Datenschutz) (Kann man typischerweise nur mit Tests belegen)
- Welchen «Standard» haben vergleichbare Produkte? (tieferer Standart als die Konkurrenz erschwert den Markteinstieg)
- In welcher «Liga» wird das Produkt lanciert? (Budget vs Premium)
- Wie «sicher» soll oder muss das Produkt sein?

2.7 Übung: Test-Strategie in 15 Minuten

Abbildung 2.9 zeigt eine Hilfestellung um eine Test-Strategie in 15 Minuten zu entwickeln.

Abbildung 2.9: Hilfestellung für eine Test-Strategie in 15 Minuten [2].

Beispiel Teststrategie für Game:

- Welche Testaktivitäten sind notwendig? Das Spiel soll flüssig (NFT) und ohne Game Play beeinträchtigende Fehler spielbar sein. Daraus sind folgende relevanten Testaufgaben abgeleitet:
  - Performance Test auf handelsüblichen Geräten
  - Game Play Tests (Alpha /Beta Tests) um allfällige Fehler zu finden
- Wie intensiv muss getestet werden? Das Risiko ist sehr tief, einzig die Gefahr besteht das aufgrund von zu vielen Fehlern die Motivation zum entwickeln reduziert.
  - Beim entdecken von Bugs zuerst Tests schreiben welche diese Bugs Beschreiben und Ihn und ähnliche Bugs entdecken
  - Komplizierte Konzepte und Anordnungen vorgängig (TDD) testen, da man sicher sein kann, viele Bugs im Code zu haben
- Wie viel Zeit/Budget steht zur Verfügung? Da dies ein Freizeit Projekt ist, hat man weder Zeit noch Geld. Die einzig verfügbare Ressource ist die Motivation, welche die resultierende Zeit bestimmt. Eine hohe Motivation kommt von einem Funktionierenden Prototypen und neuen Features.

LITERATUR

[1] *sixsigmablackbelt.de Fehlerkosten 10er Regel*. https://www.sixsigmablackbelt.de/fehlerkosten-10er-regel-zehnerregel-rule-of-ten/. Accessed: 2021-02-26.

[2] Dominique Portmann. „The Noser Way Of Testing“. In: *Booklet Noser Engineering AG* (2016).

[3] *scrum-events.de Definition of Ready*. https://www.scrum-events.de/was-ist-die-definition-of-ready-dor.html. Accessed: 2021-03-05.