# Qualitätssicherung und Kundenwirklichkeit

## oder kurzes Lob des Fehlerteufels

1 Die Fallversuche am Schiefen Turm von Pisa haben nicht stattgefunden, weil Galilei seine These von der gleichförmig geradlinigen Bewegung nur im Weltall hätte beweisen können. Mit dem Versuch am Turm würden die atmosphärischen Einflüsse seine Thesen empirisch widerlegt haben. Ähnlich verhält sich es sich mit jedem Softwaretest. Im schwerelosen Testlabor kann ein Programm in einer definierten Umgebung auf ein gewünschtes Verhalten hin getestet werden. Sobald aber Kunden das Testlabor betreten oder das Programm beim Kunden zum Einsatz kommt, ändern sich (fast) alle Parameter („Unschärferelation“ eines Softwaretests).

**Abbildung 1:** Specht an einem Außentank des Spaceshuttles STS-70 kurz vor dem 21.ten Flug ins All. Der Specht hackt 205 Löcher in die äußere Schutzhülle. Quelle: **http://ohioastronaut.com/sts-70** (11.12.2021)

2 Das Verhalten eines Programms ist abhängig von einer unbekannten Anzahl von Bedingungen durch Updates, Betriebssysteme, Hardware und auch Anwenderverhalten. Der Softwaretest reduziert die Anzahl der Bedingungen auf eine ideale, gewünschte und gemeinte Konstellation. „*Genau wie in der Simulation*“ heißt es in der Schlussszene von Enterprise „Der Aufstand“.

3 Ein Theaterstück typisiert, dramatisiert und wirkt deshalb zugleich durch die Überzeichnung und die Verengung des Blicks. Der Softwaretest gleicht einem Theaterspiel, das immer nur die Szenen übt, die uns gefallen oder die wir für wichtig halten. Jeder Software-Test ist deshalb ein Spiegel unserer Vorstellungskraft, aber die Wirklichkeit liebt es, uns in unzähligen Varianten gegenüberzutreten und das auch mit schwarzem Humor.

4 Nach dem Test ist vor dem Test oder *„man steigt nicht zweimal in denselben Fluss“* (Heraklit). Die Wirklichkeit, in der sich ein Programm bewegt und bewähren muss, ist kein statisches Bauwerk, sondern ein sich täglich wandelnder Fluss von Bedingungen. Vorführeffekte – oder was gerade noch ging, jetzt funktioniert es nicht oder doch. Jeder Softwaretest zielt auf eine statische Wirklichkeit.

5 „*Contra factum non valet argumentum*“. Für die Scholastiker galt der Grundsatz, dass gegen Fakten Argumente keinen Bestand haben. Der Softwaretest schafft eine eigene Testwirklichkeit. Das Programm scheitert aber in der Wirklichkeit des Arbeitsalltags und an „verrückten“ oder „kreativen“ Anwendern, nicht aber in der Konzeption des Entwicklers und nicht in der Testumgebung. Die Testwirklichkeit hat die Anmaßung einer Utopie, die dann der schlechten Wirklichkeit vorgehalten wird.

6 Die Wirklichkeit ist überwältigender, als unsere Vorstellungskraft es sich auszumalen vermag. Die Flutkatastrophe ist, wenn sie eintritt, höher als der Deich, aber die Kosten für den Deichbau haben eine wirtschaftlich sinnvolle Grenze. Der Aufwand für Testumgebungen und -strategien ist immer eine Grenzwertbetrachtung. Und dennoch muss es für den Fall der Sintflut eine Option geben.

7 Jede Software hat eine Vergangenheit („DOS-Box“) oder die Rückwärtskompatibilität eines Programms endet nicht, sofern es benutzt wird und im wirklichen Arbeitsalltag

tagtäglich im Einsatz ist. Deshalb wahrt die neue Version einer Software die Funktionalität der alten Version. Tastaturbelegungen wie F5 oder F10 können über Generationen nicht verändert werden, weil Arbeits- und Sehgewohnheiten eine erstaunliche Trägheit aufweisen. Die Abschaltung oder Änderung einer altbewährten Bedienfunktion nimmt der Kunde als Fehler wahr.

8 Das Neuschreiben einer Software löst keine Probleme, sondern schafft andere Probleme, weil dieselben Personen immer wieder ähnliche Ergebnisse liefern und die Summe der Probleme erhalten bleibt. Gute Software wird letztlich vom Kunden geschrieben, der aber ist in der utopisch idealistischen Sicht ein Störfall.

9 Du „*sollst dem Ochsen, der da drischt, nicht das Maul verbinden*“ oder du sollst „*das Feld nicht bis zum äußersten Rand abernten*“ und die Reste der Ernte „*dem Armen und dem Fremden überlassen*“. Die mosaische Gesetzgebung weist darauf hin, dass das Streben nach vollständiger Effizienz Grenzen haben muss, weil Reibungen und Friktionen wesentlich sind für einen dauerhaften anpassungsfähigen Betrieb. Totalitäre Qualitätssicherung führt zum Stillstand der Systeme oder eliminiert letztlich den Kunden.

10 Wenn ein Fehler lange genug im Einsatz ist, kann er sich – dank der Alltagskreativität der Kunden – in ein Feature verwandelt haben, weshalb dann auf die Beseitigung eines Fehlers im Programm – das Ticket war schon älter – der Kundenrückruf folgt, warum die bisher gute Funktion abgeschaltet wurde.

11 Je mehr der Programmierer vor Kundenkontakt geschützt ist, desto größer ist die Gefahr, dass Ideal und Wirklichkeit aneinander vorbeireden, weil die Übermittlung der Berichte von der Hotline an den Programmierer nur eine andere Form des lustigen Spiels von der „stillen Post“ darstellt.

12 Fehleranalyse basiert oft auf dem Impuls, dass das richtige Konzept nicht entschieden genug umgesetzt wurde. Ins Bild übertragen: Fährt ein Auto mit Tempo 50 vor die Wand, dann führt die genannte Denkweise zu der ersten Vermutung, dass das Auto zu langsam gewesen sein muss. Der Vorgang wird mit höherer Geschwindigkeit wiederholt. Deshalb: Fehleranalyse darf den Gesamtzweck sowie das Verhältnis von Zweck und Mittel nicht aus dem Blick verlieren.

13 Kleinste Störungen können in komplexen Umgebungen zum Chaos führen oder umgekehrt für den stabilen Betrieb erforderlich geworden sein. Auch können Fehler gestapelt vorliegen, so dass die Beseitigung eines Fehlers gefährlichere Störungen aktiviert, als zuvor beseitigt wurden (Hydra-Effekt).

14 Fehler haben Witz und entziehen sich einem einfachen Ursache-Wirkungsmuster. Fehler können sich verstecken und kommen nur hervor, wenn der Testende unaufmerksam oder missmutig ist, also nicht genau hinsieht. Je tiefer im System und je niedriger in der Technologieschicht der Ursprung des Fehlers liegt, desto diffuser sind seine Auswirkungen an der Anwenderoberfläche.

15 Ein Autor kann seinen eigenen Text nicht korrekturlesen, weil er den Inhalt sieht und nicht die Form, weshalb der Programmierer ein schlechter Softwaretester ist. Für den Programmierer gibt es Fehler, die nicht möglich sind, eigentlich. Der Tester berichtet aber von ihnen und der Kunde redet schon seit einiger Zeit von diesen Programmgeistern.

16 Die Bereitschaft, Störungen und Fehler zu sehen, ist in Präsentationen am höchsten und in Testsituationen niedriger, weil die Testsituation auf den Nachweis des guten Gelingens zielt, weshalb die innere Autokorrektur des Testenden die kleinen und kleinsten Effekte im Augenwinkel nicht wahrzunehmen geneigt ist.

17 Eine gute und wegweisende Software kann durch ihre Fehlerhaftigkeit Millionen von Menschen Arbeit und Aufgabe geben, weil die Unvollständigkeit eines Programms zur Mitarbeit auffordert. Deshalb können Fehler eine Existenzberechtigung haben und Anzeichen für das Werden eines Projektes sein. Perfektion ist aus dieser Perspektive fertig, am Ende, aus.

Die Existenz von Fehlerteufeln lässt sich nicht wegdiskutieren oder wegtesten. Ein fehlerteufelfreies System wird es nicht geben können, denn das Streben nach vollständiger Fehlerfreiheit ist utopisch, führt zum Entwicklungsstillstand oder schlimmstenfalls zum Gesamtversagen des Systems. Fehlerteufel aber lassen sich in die Schranken verweisen. Wir können sie eindämmen und ihre Wirkmächtigkeit einschränken. Darin steckt implizit die Anerkennung des Existenzrechtes der Fehlerteufel, aber dafür helfen uns die kleinen Geister und Kobolde, indem sie uns daran erinnern, dass da immer noch ein Rest zu tun bleibt.

Köln, den 11.12.2021

Dr. Uwe Eissing