

# Programmierkonzepte in den Naturwissenschaften

## 9. Bugverwaltung

PD Dr. Till Biskup  
Physikalische Chemie  
Universität des Saarlandes  
Sommersemester 2021





- ❏ Software ist (fast) nie fehlerfrei. Je früher Fehler entdeckt (und behoben) werden, desto besser und billiger.
- ❏ Die Wahrscheinlichkeit von Fehlern sollte aktiv minimiert werden (Wahl der Programmiersprache, Werkzeuge, ...).
- ❏ Fehler sollten grundsätzlich in Tests verwandelt und auch in Entwicklungsversionen behoben werden.
- ❏ Eine Bugverwaltung hilft Anwendern und Entwicklern durch Strukturierung und Automatisierung.
- ❏ Eine Bugverwaltung sollte fest in die restliche Infrastruktur und Abläufe bei der Projektentwicklung eingebunden sein.

# Zur Herkunft des Begriffs


Als Insekten noch Rechner zum Absturz brachten...



9/9

0800 Antan started  
1000 " stopped - antan ✓  
13<sup>00</sup> MC (032) MP-MC ~~1.982147000~~  
(033) PRO 2 2.130476415 ~~(03)~~ 4.615925059(-2)  
convct 2.130476415  
Relays 6-2 in 033 failed special speed test  
in relay " 11.00 test.  
Relays changed

1100 Started Cosine Tape (Sine check)  
1525 Started Multi Adder Test.

1545  Relay #70 Panel F  
(moth) in relay.

First actual case of bug being found.

~~1630~~ Antan started.  
1700 closed down.

Relay 2145  
Relay 3376

1.2700 9.037 847 025  
9.037 846 995 convct

Logbuch-Seite des Mark II Aiken Relay Calculator mit dem ersten Bug (1947)  
NH 96566-KN courtesy of the Naval History & Heritage Command

Ausgangspunkt: Software ist (fast) nie fehlerfrei

Umgang mit Fehlern in Software

Motivation: Warum eine Bugverwaltung?

Handhabung: Fehlerberichte, Feedback, Integration

## Bug

Programmfehler oder Softwarefehler,  
allgemein ein Fehlverhalten von Computerprogrammen

„Weinberg's Second Law“

“ *If builders built buildings  
the way programmers wrote programs,  
then the first woodpecker that came along  
would destroy civilization.*

– Gerald Weinberg

## ▶ Validierung

- „Wird das richtige Produkt entwickelt?“
- Prüfung der Eignung einer Software für ihren Einsatzzweck auf Grundlage eines vorher erstellten Anforderungsprofils

## ▶ Verifizierung

- „Wird das Produkt richtig erstellt?“
- Feststellung, ob ein Computerprogramm seiner (formalen) Spezifikation entspricht (korrekt ist)
- Aufspüren vorhandener Fehler

- ☞ Validierung ist nicht formalisierbar, da die Anforderungen nicht formalisiert vorliegen.
- ☞ Eine formale Verifizierung ist im Allgemeinen nicht möglich.

## Satz

Erreichen formale Systeme einen bestimmten Grad an Komplexität, so lassen sich Aussagen angeben, die man weder beweisen noch widerlegen kann.

- ▶ vereinfachte Formulierung des (ersten) Gödelschen Unvollständigkeitssatzes
- ▶ weitreichende Implikationen für Mathematik und Informatik
- ☛ Die Fehlerfreiheit eines Programms lässt sich in der Regel nicht beweisen.

K. Gödel, *Monatshefte für Mathematik und Physik* 38:173-198, 1931

A. Turing, *Proceedings of the London Mathematical Society* 42:230-265, 1936

# Software ist (fast) nie fehlerfrei

Fehler in Software sind der Normalfall



- ▶ Fehler treten oft erst beim Anwender auf.
  - Fehlerfreiheit i.d.R. nicht beweisbar
  - Fehlersuche nur so gut wie die Suchenden
- ▶ Eine transparente und strukturierte Möglichkeit, Fehler an die Entwickler zu melden, hilft allen Beteiligten (Anwendern wie Entwicklern).
  - Formalisierung und Automatisierung von Abläufen
  - *Bugtracker* essentieller Aspekt einer Projekt-Infrastruktur
- ▶ Entwickler sollten ein großes Interesse daran haben, möglichst früh von Fehlern in ihrer Software zu erfahren.
  - je früher ein Fehler gefunden/behoben wird, desto billiger
  - Bugs können mitunter massive Konsequenzen haben.



# Software ist (fast) nie fehlerfrei

Bugs sollten ernstgenommen werden



Jungfernflug der Ariane 5 am 04.06.1996; Copyright ©: ESA

## Drei mögliche ernsthafte Konsequenzen

- ▶ Kosten
    - Beispiel Ariane-5-Jungfernflug:  
ca. 370 Mio. USD und nachfolgende Verdienstauffälle
  - ▶ Menschenleben
    - Beispiel Therac-25-Bestrahlungsgerät:  
3 Tote, 3 Schwerverletzte
  - ▶ Reputation
    - hauptsächliche Konsequenz in den Wissenschaften
- ☛ Bugs können immense Folgen haben.
  - ☛ Der Umgang mit Bugs kann entscheidend sein.  
(zügig, umfassend, Ursachen beheben)

Ausgangspunkt: Software ist (fast) nie fehlerfrei

Umgang mit Fehlern in Software

Motivation: Warum eine Bugverwaltung?

Handhabung: Fehlerberichte, Feedback, Integration

### Strategien im Umgang mit Fehlern

- ▶ vermeiden bzw. Wahrscheinlichkeit minimieren
  - Strategien während der Entwicklung
- ▶ frühzeitig entdecken und beheben
  - sowohl bei der Entwicklung als auch im Betrieb
  - Faustregel: je früher erkannt, desto billiger behebbar
  - ernst nehmen
- ▶ Fehler in Tests verwandeln
  - sowohl bei der Entwicklung als auch im Betrieb
  - setzt Verwendung automatisierter Tests voraus
- ▶ Fehler überall beheben
  - Hotfix muss in Entwicklerversionen einfließen
  - Regression durch Weiterentwicklung vermeiden

## Vermeiden bzw. Wahrscheinlichkeit minimieren

- ▶ *Coding Conventions*
  - keine „cleveren Tricks“
  - sprechende und konsistente Benennung von Variablen etc.
- ▶ geeignete Programmiersprache
  - Mechanismen zur Fehlerdetektion (z.B. starke Typisierung)
  - formale Spezifikationen: „*design by contract*“
  - ggf. einfach zu erlernende/verwendende Sprache
- ▶ Werkzeuge zur Fehlerdetektion
  - z.B. statische Code-Analysatoren
- ▶ lesbarer Code
  - vereinfacht das Finden von Fehlern (bei Code-Reviews etc.)
- ▶ Disziplin

## Frühzeitig entdecken und beheben

- ▶ Anforderungsanalyse
  - Was soll eine gegebene Routine/Software machen?
  - Voraussetzung für die Validierung
- ▶ klare Kriterien für korrektes Verhalten von Programmteilen
  - Voraussetzung für die Definition von Tests
  - setzt tiefes Verständnis der Zusammenhänge voraus
- ▶ Definition von Tests (zur Verifizierung)
  - Voraussetzung: Modularisierung und Testbarkeit
  - Automatisierung der Tests
- ▶ Qualitätssicherung bei der Entwicklung
  - regelmäßige Code-Reviews
  - Kultur des Umgangs mit Fehlern und Unzulänglichkeiten

## Fehler in Tests verwandeln

### ▶ Voraussetzung

- Existenz von (möglichst) automatisierten Tests
- modularer, testbarer Code
- strukturierter Ablauf für den Umgang mit Fehlern

### ▶ Idee

- Ein einmal aufgetretener Fehler kann wieder auftreten.
- Automatisierte Tests kosten nichts und geben Sicherheit.

### ▶ Umsetzung

- Testsuiten laufen automatisiert, häufig und regelmäßig.
- Jeder bekannte Fehler wird in einen Test verwandelt.

☛ Sowohl in der Entwicklung als auch bei Produktivsystemen

## Fehler überall beheben

### ▶ häufige Situation

- 1 Fehler wurde in veröffentlichter Version gefunden.
- 2 Hotfix zur Fehlerbehebung
- 3 Hotfix fließt nicht in Entwicklerversionen ein.
- 4 Nächste Veröffentlichung enthält alte Fehler (Regression).

### ▶ Lösungsansatz

- Fehler in Test verwandeln
- Test gegen Entwicklerversionen laufen lassen
- Entwicklerversionen ggf. anpassen

### ▶ Voraussetzungen

- Existenz von (möglichst) automatisierten Tests
- Versionsverwaltungssystem
- strukturierter Ablauf für den Umgang mit Fehlern



Ausgangspunkt: Software ist (fast) nie fehlerfrei

Umgang mit Fehlern in Software

Motivation: Warum eine Bugverwaltung?

Handhabung: Fehlerberichte, Feedback, Integration

### Gründe für die Nutzung einer Bugverwaltung

- ▶ Strukturierung und Vereinfachung von Abläufen
  - beschleunigend, befreiend, strukturierend
  - klare Referenz für jeden Fehler
- ▶ Dokumentation aufgetretener Fehler
  - vermeidet Doppelungen
  - wichtig für Nachvollziehbarkeit/Überprüfbarkeit
- ▶ Entkopplung von Nutzern und Entwicklern
  - keine direkte Interaktion notwendig
  - wichtig bei mehreren Entwicklern/größerer Nutzerbasis
- ▶ Diskussion der weiteren Entwicklung
  - *Bugtracker als Issue Tracker*
  - *Feature Requests*

## Strukturierung und Vereinfachung von Abläufen

- ▶ ein zentraler Ort
  - bewahrt den Überblick
  - einfacher Zugriff
  - einheitliches Format
- ▶ Automatisierung und klare Abläufe
  - Kontakt zwischen Berichter und Entwickler ausschließlich über Bugtracker
  - Andere Entwickler können einspringen/übernehmen.
  - Zuweisung, Rückfragen, Benachrichtigungen möglich
- ▶ Transparenz
  - Berichter ist jederzeit über Entwicklung im Bilde.
  - keine Doppelungen dank Referenz auf bekannte Fehler

## Dokumentation aufgetretener Fehler

- ▶ vermeidet Doppelungen
  - Referenz auf andere Fehler möglich
  - Fehlerberichte können als Duplikat markiert werden.
- ▶ bewahrt die Übersicht
  - Nicht jeder Fehler wird gleich behoben.
  - Manche Fehler ähneln sich.
  - Identifizierung häufiger Fehlerursachen und ggf. Anpassung der Entwicklungsstrategie
- ▶ wichtig für Nachvollziehbarkeit/Überprüfbarkeit
  - insbesondere bei Auswertungssoftware
  - im Zusammenhang mit Versionsverwaltung und Versionsnummern
  - Fehler nach betroffenen Versionen anzeigbar

## Entkopplung von Nutzern und Entwicklern

- ▶ Entkopplung von Auftreten und Behebung
  - Nutzer kann Fehler berichten
  - Entwickler kann später den Fehler beheben
  - Jeder weiß jederzeit, welche Fehler noch offen sind.
- ▶ erleichtert die Übernahme von Projekten
  - Fehlerberichte zentral an einem Ort
  - komplette Historie der Diskussionen verfügbar
  - Fehlerberichte durchsuchbar nach vielen Kriterien: offene Bugs, Zuweisung zu Meilensteinen, ...
- ▶ skaliert
  - kein Flaschenhals wie bei Emails an den Hauptentwickler
  - funktioniert auch noch mit vielen Entwicklern/Nutzern

## Diskussion der weiteren Entwicklung

- ▶ *Bugtracker* dienen dem Austausch
  - ähnlich einem Forum, aber fokussierter
  - klare Referenzen zu Programmverhalten
  - fokussiert auf Verhalten des Programms:  
keine Programmierkenntnisse notwendig, um beizutragen
- ▶ *Feature Requests*
  - Nutzer können weitere Funktionalität anfragen.
  - Lassen sich genauso kategorisieren, suchen, zuweisen, ...
- ▶ Begutachtung von Code
  - Jeder Entwickler kann mitdiskutieren.
  - *Patches* und *Pull Requests*
  - Verknüpfung mit Versionsverwaltung

Ausgangspunkt: Software ist (fast) nie fehlerfrei

Umgang mit Fehlern in Software

Motivation: Warum eine Bugverwaltung?

Handhabung: Fehlerberichte, Feedback, Integration

## Verantwortung der Nutzer: brauchbare Fehlerberichte

- ▶ Möglichst präzise Fehlerbeschreibung
  - Was wollte der Nutzer erreichen?
  - Was war stattdessen das Verhalten des Programms?
  - Ist der Fehler reproduzierbar?
- ▶ Details zum verwendeten System
  - Betriebssystem
  - Programmversion
  - ggf. Versionen abhängiger Bibliotheken
- ▶ Schwere des Fehlers
  - *blocker, critical, major, minor, enhancement*
  - hilft bei der Einschätzung der Wichtigkeit und Dringlichkeit
- 👉 Wichtig: keine sensiblen Informationen in Fehlerberichte!



## Verantwortung der Entwickler: zeitnahes Feedback

- ▶ Nutzer ernst nehmen
  - zeitnahe Antwort auf einen Fehlerbericht
  - wichtig für die Motivation, weiter Bugs zu berichten
  - Fehler immer zuerst in der Software vermuten
- ▶ Fehler lokal reproduzieren
  - ggf. fehlende Informationen vom Nutzer erfragen
  - Diskussion *nur* im/über den Bugtracker (Dokumentation!)
  - sensible Informationen (z.B. Daten) ggf. auf anderem Weg
- ▶ Wichtigkeit und Dringlichkeit einschätzen
  - kann sich ggf. ändern
  - ggf. einem Entwickler zuweisen oder selbst übernehmen
  - wenn weniger dringend: Zuordnung zu Meilenstein etc.

“ *I've found the root cause of our problems.  
It's people.  
They're buggy.*

– Dilbert

- 👉 Nutzer sind immer für Überraschungen gut.
  - oft vollkommen andere Herangehensweise
  - Nutzer sind i.d.R. eindeutig in der Überzahl.
- 👉 Nutzer sind auch (nur) Menschen.
  - reflektierter und geduldiger Umgang (Kant)
  - Fehler immer erst bei sich suchen

## Integration in bestehende Infrastruktur und Abläufe

- ▶ Versionsverwaltung
  - bei kritischem Fehler in veröffentlichter Version: Hotfix und in Entwicklerversion(en) einpflegen
  - Bugfixes i.d.R. von neuen Features trennen
- ▶ Versionsnummern
  - nach Bugfix inkrementieren
  - *Changelog*: Verweis auf behobene Fehler (mit Nummer)
- ▶ Tests
  - Fehler in Tests verwandeln
- ☛ Plattformen wie GitHub integrieren Bugtracker, Wiki etc.
- ☛ Separate Werkzeuge haben Vorteile



- ❏ Software ist (fast) nie fehlerfrei. Je früher Fehler entdeckt (und behoben) werden, desto besser und billiger.
- ❏ Die Wahrscheinlichkeit von Fehlern sollte aktiv minimiert werden (Wahl der Programmiersprache, Werkzeuge, ...).
- ❏ Fehler sollten grundsätzlich in Tests verwandelt und auch in Entwicklungsversionen behoben werden.
- ❏ Eine Bugverwaltung hilft Anwendern und Entwicklern durch Strukturierung und Automatisierung.
- ❏ Eine Bugverwaltung sollte fest in die restliche Infrastruktur und Abläufe bei der Projektentwicklung eingebunden sein.