



```
*  
* DESCRIPTION :  
* Sprint documentation, code comments and bug-reporting.  
*  
* NOTES :  
* These slides are a part of API;  
* See https://github.com/TUBSAPISS2017 for detailed description.  
*  
* Copyright Institute of Flight Guidance 2018. All rights reserved.  
*  
* AUTHOR : Paul Frost  
*
```

# Dokumentation und Bug-Reporting

Prof. Dr.-Ing. Peter Hecker, Dipl.-Ing. Paul Frost, Andreas Dekiert M. Sc.,  
29. Mai 2018

# Agenda

- 03. April Einführung
- 10. April Softwareprojektmanagement
- 17. April Entwicklungstools
- 24. April GitHub
- 08. Mai Einführung Arduino/Funduino
- 15. Mai Dateieingabe und -ausgabe
- 22. Mai **Exkursionswoche**
- 29. Mai Dokumentation und Bug-Reporting**
- 05. Juni Einführung von Qt
- 12. Juni GUI-Erstellung mit Qt
- 19. Juni Anleitung erstellen
- 26. Juni **Projektarbeit**
- 03. Juli **Vorbereitung der Abgabe**
- 10. Juli Abgabe**



# Lehrziele

## Dokumentation und Bug-Reporting

Als Teilnehmer soll ich am Ende dieser Übung...

- die Dokumentation eines Sprints erstellen können
- den Quelltext übersichtlich halten können
- den Quelltext sinnvoll kommentieren können
- Bug-Reports anfertigen können



## Sprint-dokumentation



Planung

Meeting

Abschluss



# Dokumentation der Sprintplanung

## Dokumentation

- Schriftliches Verfassen der Aufgaben in ganzen Sätzen
- Verknüpfung mit Zielkriterium herstellen
- Kontext bereitstellen (Zugehörigkeit zu User-Story / Zielkriterien)
- Darstellung der Abhängigkeiten (in Textform oder als Schema)

## Beispiel

**Lfd. Nr.:** A4.1  
**Bearbeiter:** Paul Frost  
**Zugehörigkeit:** User-Story 1, Zielkriterium 4 (oder US1/Z4)  
**Aufgabe:** Den Restbetrag auf dem Display anzeigen.  
**Abhängigkeiten:** A3.2 (oder keine)



Gibt es Fragen oder Anmerkungen zu dem Unterthema  
**Dokumentation Sprintplanung?**



# Dokumentation eines Sprint-Meetings

1/2

## Dokumentation

- Nach Teammitgliedern getrennt
- Auflistung der bearbeiteten Aufgaben der vergangenen Woche
- Zwischenstände der bearbeiteten Aufgaben:
  - abgeschlossen
  - nicht begonnen (, weil...)
  - zu 50 % erledigt
  - kann nicht weiter bearbeitet werden, weil...
- Auflistung der zu bearbeitenden Aufgaben für die folgende Woche



# Dokumentation eines Sprint-Meetings

2/2

## Beispiel

**Meeting:** 15.05.2018

<b>Teilnehmer:</b>	Paul Frost
<b>Bearbeitet:</b>	A4.1, A4.2
<b>Zwischenstände:</b>	A4.1 kann aktuell nicht weiter bearbeitet werden, weil das Guthaben noch nicht ermittelt werden kann A4.2 wurde begonnen
<b>Geplant:</b>	A.1 und A4.2
<b>Teilnehmer:</b>	Andreas Dekiert
<b>...</b>	



Gibt es Fragen oder Anmerkungen zu dem Unterthema  
**Dokumentation Sprint-Meeting?**



# Dokumentation des Sprint-Abschlusses

1/3

## Dokumentation

- Teammitglieder übergreifend
- Projektstatus
  - Aktuelle Revisionsnummer
  - Status der User-Stories, der Zielkriterien
    - Betrachtung derselben Fragestellungen wie im Sprint-Meeting
  - Sind Änderungen am Backlog erforderlich? Wenn ja, welche?
- Aufgabenstatus
  - Zusammenfassung der Status aller Aufgaben
    - vgl. Aufgabenstatus im Sprintmeeting
  - Zusätzlich soll die Revision angegeben werden, mit der die Aufgabe erfüllt wurde.



# Dokumentation des Sprint-Abschlusses

2/3

## Dokumentation

- Bewertung der Sprintplanung
  - Wurden neue Technologien ausprobiert?
  - Welche Höhen und Tiefen gab es?
  - Was hat genervt?
  - Was gab es zu feiern?
  - Welche Punkte können in der Sprintplanung verbessert werden?
  - Ist ein Fortschritt zu den vorangegangenen Sprints erkennbar?



# Dokumentation des Sprint-Abschlusses

3/3

## Beispiel

**Sprint 1** 28.05.2018

**Aktuelle Revision:** b9ebc02c892f92b33a7a626b55b553bfc7792d94

**Projektstatus:** US1 begonnen - Z1 abgeschlossen, Z2 begonnen...

US2 begonnen - Z1 begonnen; Z2 begonnen; ...

Änderungen am Backlog sind nicht erforderlich

**Aufgaben:** A4.1 begonnen

A4.2 abgeschlossen (b9ebc02c892f9...)

...

**Bewertung:** In diesem Sprint konnten nicht alle Aufgaben abgeschlossen werden, weil die Einrichtung der IDE mehr Zeit in Anspruch genommen hat als geplant. Die Aufgaben waren nicht kleinteilig ...



Gibt es Fragen oder Anmerkungen zu dem Unterthema  
**Dokumentation Sprintabschluss?**



# Abgehakt

## Dokumentation und Bug-Reporting

Als Teilnehmer soll ich am Ende dieser Übung...

- die Dokumentation eines Sprints erstellen können
- den Quelltext übersichtlich halten können
- den Quelltext sinnvoll kommentieren können
- Bug-Reports anfertigen können



Quelltext-  
dokumentation

// ...

Einordnung

Programmier-  
richtlinien

Kommen-  
tierung

Doxygen



# Hintergrund

Für den Kunden

- Nachweis des Arbeitsfortschritts

Für das Projektmanagement

- Risikobewertung
- Projektstatus

Für andere Entwickler

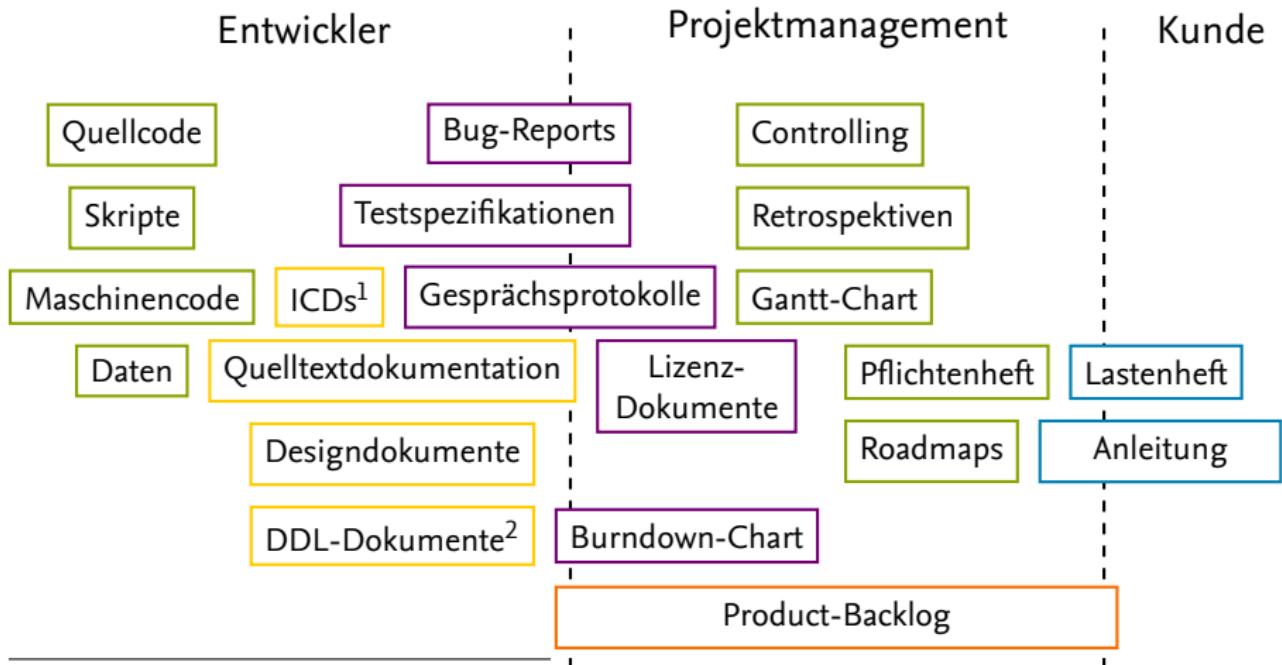
- Kommunikation zwischen Mitarbeitern ermöglichen/erleichtern
- Definition von Schnittstellen zu anderen Modulen

Für den Entwickler selbst

- Nachvollziehbarkeit auch nach längerer Zeit



# Klassifikation nach Nutzergruppen



<sup>1</sup>Dokumentation zur Schnittstellenansteuerung

<sup>2</sup>Datendefinitionssprache

# (Software-) Designdokumente

- Erlaubte Bibliotheken eingrenzen
  - Weniger Lösungswege → homogenere Herangehensweise
  - Weniger Probleme bei der Portierung auf andere Plattformen
- Programmierrichtlinien
  - Geben die Schriftform vor
  - Einrückungsstil, Namenskonventionen
  - (Un-) Erwünschte Konstrukte
- Softwarearchitektur
  - Algorithmen schematisch beschreiben
  - Beschreibung verwendeter Prinzipien & Design-Muster (Pattern)
  - Definition von Modulen und deren Abhängigkeiten
  - Darstellung erfolgt unabhängig der Programmiersprache



# Hintergrund

Listing 1: Beispiel für schlechte Code-Formatierung

```
class E{public: void shE () {  
Serial.print("Eins"); Serial.println();}  
}; class Z{public: void shZ ()  
{Serial.print("Zwei"); Serial.println();}  
}; E e; Z z; void setup () {  
Serial.begin(9600); } void loop () {  
e.shE(); z.shZ(); delay(1000); }
```

- ⇒ Keine Nachvollziehbarkeit des Quelltextes.  
Weder für Teammitglieder, noch den Programmierer selbst.



# Programmierrichtlinien

Was zeichnet guten Code aus?

- Funktionalität
- Verständlichkeit
- Wartbarkeit & Erweiterbarkeit
- Effizienz & Eleganz

Welche Mittel stehen den Entwicklern zur Verfügung?

- Formatierung
- Bezeichner
- Komplexität
- Kommentare



# Formatierung

# Klammern und Separatoren

- Bessere Wartbarkeit
- Geringere Fehleranfälligkeit

Listing 2: Ohne Klammern

```
while (true)  
wiederholung();
```

Listing 3: Mit Klammern

```
while (true) {  
wiederholung();  
}
```



# Formatierung

# Einrückungen

- Verdeutlichen die Zugehörigkeit
- Programmablauf wird nachvollziehbarer
- Syntaxfehler werden schneller ersichtlich

Listing 4: Mit Einrückung

```
while (true) {
    if (quit)  {
        break;
    }
    wiederholung ();
}
```



# Formatierung

# Einrückungen

Listing 5: Ohne Einrückung

```
if ((bedingungA && bedingungB)
|| bedingungC
|| bedingungD) {
    machEtwas ();
}
```

Listing 6: Mit Einrückung

```
if ((bedingungA && bedingungB)
    || bedingungC
    || bedingungD) {
    machEtwas ();
}
```



# Bezeichner

# Schlechtes Beispiel

Listing 7: Schlechte Bezeichner

```
const int varA = 42600;
const float varB = 35.8f;
const float varC = 122.4;

char* einvielzulangervariablenname = "LH321";
char* Varmitgrossbuchstabeamfang = "EDVE";

double _ = 52.15648;
double __ = 9.5614;

float test = 54600.64f;
```



# Bezeichner

# Beispiel

Listing 8: Sinnvolle Bezeichner

```
const int A320_EMPTY_WEIGHT_KG = 42600;
const float A320_WING_SPAN_M = 35.8f;
const float A320_WING_AREA_M2 = 122.4;

char* flightNr = "LH321";
char* departureAirport = "EDVE";

double latitude_deg = 52.15648;
double longitude_deg = 9.5614;

float currentMass_kg = 54600.64f;
```



# Bezeichner

# Namenskonvention

Variablen	Kleinbuchstabe am Anfang Trennung bei Großbuchstabe	date, statusLed
Konstanten	Großbuchstaben, Trennung durch Unterstriche	MAX_WIDTH, MIN_WIDTH
Methoden	Kleinbuchstabe am Anfang, Verb, Imperativ Getter-Methoden  Setter-Methoden	calcDistance ()  getSpeed (), speed ()  setSpeed ()
Klassen	Großbuchstabe am Anfang	Klasse



# Bezeichner

# Vorschlag

Listing 9: Der Namenskonvention folgende Beispielklasse

```
class EineKlasse {
    static int s_statischeVariable;

public:
    void setMemberVariable(int memberVariable);
    int memberVariable() const;

private:
    int m_memberVariable;
};
```



# Komplexität

# Funktionen

- Beschreibende Funktionsnamen
  - Funktionen möglichst klein halten
    - Richtwert: Funktion sollte ohne Scrollen überblickt werden können
- ⇒ Quellcode ist leichter zu verstehen und wieder zu verwenden

Listing 10: Beispiel für portabel Funktionen

```
double calculateX(double lat, double lon) {...}
double calculateY(double lat, double lon) {...}

void calculateXY(double lat, double lon)
{
    double x = calculateX(lat, lon);
    double y = calculateY(lat, lon);
    printPoint(x, y);
}
```



Gibt es Fragen oder Anmerkungen zu dem Unterthema  
**Programmierrichtlinien?**



# Kommentare

# Quellcode

- Programmiersprachen erlauben das Erstellen von Kommentaren
  - Die Programmerstellung wird nicht beeinflusst
  - Erhöhung der Verständlichkeit des Codes durch Anmerkungen
- C/C++/Java

```
// Diese Zeile ist ein Kommentar
/* Dieser Block
ist ein Kommentar */
```

- XML/HTML
- ```
<!-- Dieser Block
ist ein Kommentar -->
```

Anmerkungen im Quelltext IMMER als Kommentar hinzufügen!



# Beschreibung öffentlicher Funktionen

- Extern verfügbare Funktionen & Objekte sollten dokumentiert werden
  - Nutzer muss oft nur wissen, was gemacht wird, aber nicht wie
- Schnittstellen-Dokumentation in Header-Dateien (.h, .hpp)

## Listing 11: Deklaration (Header)

```
// Fuehrt eine Division aus.  
// Bitte Divisor niemals 0 setzen!  
float divide(float dividend, float divisor);
```

## Listing 12: Implementierung (Code-Datei .cpp)

```
float divide(float dividend, float divisor)  
{  
    return dividend / divisor;  
}
```



# Beispiel für vollständige Kommentierung

Listing 13: Sinnvoller Kommentar (mehr oder weniger)

```
/** \brief Mathe-Konstante: Kreiszahl Pi */
const float M_PI = 3.14159265359;

/** \brief Berechnet den Umfang eines Kreises
 * \param r Radius
 * \return Umfang des Kreises als float
 */
float calcCircumfence(float r)
{
    return 2 * M_PI * r;
}
```



# Dokumentation im Rahmen von API

## Dokumentation

- Jede eigene Routine soll mindestens ein kurzer Kommentar begleiten.  
Beispiel: `divide()`
- Standard-C++-Methoden müssen nicht kommentiert werden
- Zusätzlich soll jedes Gruppenmitglied 2 Funktionen mit Ein- und Ausgabeparametern implementieren und kommentieren.  
Beispiel: `calcCircumfence()`
- Diese sollen im Wiki zur Bewertung deklariert werden (siehe unten)

## Beispiel

**Bearbeiter:** Paul Frost

**Funktion 1:** `restbetragAnzeigen()` (<https://github.com/.../main.cpp>)



Gibt es Fragen oder Anmerkungen zu dem Unterthema  
**Quelltextkommentierung?**



# Schnittstellen-Dokumentation

- Programmierer benötigt Informationen zur korrekten Verwendung einer Schnittstelle oder Bibliothek
  - Was bewirkt welche Methode/Funktion?
  - Welche Parameter erwartet die Methode/Funktion?
  - Was gibt die Methode/Funktion zurück?
- Quellcodekommentare entstehen während des Entwickelns
  - Nicht jeder möchte Quellcodedateien nach Kommentaren durchsuchen
  - Oft keine gute Gliederung, kein Index und keine Möglichkeit der Verlinkung durch einfache Quellcodekommentare gegeben
- Ansatz: Dokumentation (bspw. als HTML-Dokument) aus Quellcode-Kommentaren erzeugen



# Doxxygen

- Programm zur automatisierten Beschreibung von APIs
  - GPL-lizenziert, Entwicklung seit 1997
  - Unterstützung mehrerer Programmiersprachen (C, C++, C#, Java)
  - In vielen Entwicklungsumgebungen integriert
  - Lauffähig unter vielen Betriebssystemen (Linux, Windows, MacOS)
- Programmablauf von Doxygen
  - Automatische Ermittlung der Quelltext-Struktur
  - Einlesen und Verarbeitung von speziellen Doxygen-Kommentaren
  - Ausgabe der API-Beschreibung in verschiedenen Formaten (HTML, CHM, PDF, etc.)



# Struktur- und Abhängigkeitsgraphen mit Doxygen

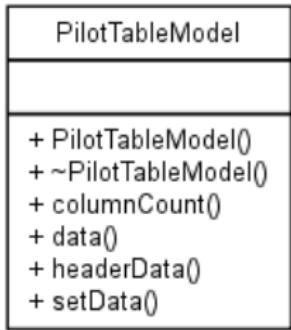


Abbildung 1: Klassendiagramm

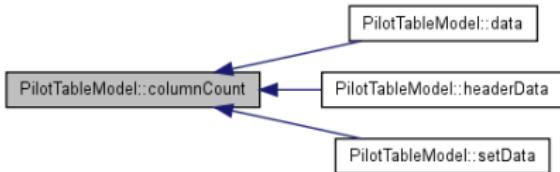


Abbildung 2: Callergraph

# Doxxygen-Kommentare

- Beschreibungen werden direkt in dem Quelltext der jeweiligen Klasse/Methode geschrieben
- Doxygen-Anweisungen innerhalb von Block-Kommentaren

|             |                                          |
|-------------|------------------------------------------|
| \brief      | Kurzbeschreibung der jeweiligen Funktion |
| \param name | Beschreibung des Parameters name         |
| \return     | Beschreibung des Rückgabewertes          |
| \author     | Liste der Autoren                        |
| \warning    | wichtige Hinweise zur Verwendung         |
| \bug        | Beschreibung eines bekannten Fehlers     |
| \version    | Aktuelle Version                         |
| \sa         | Siehe auch...                            |



Gibt es Fragen oder Anmerkungen zu dem Unterthema  
**Doxygen?**



# Abgehakt

## Dokumentation und Bug-Reporting

Als Teilnehmer soll ich am Ende dieser Übung...

- die Dokumentation eines Sprints erstellen können
- den Quelltext übersichtlich halten können
- den Quelltext sinnvoll kommentieren können
- Bug-Reports anfertigen können



## Bug-Reporting



Herkunft

Berühmte  
Fälle

Testing

Bug-  
Reporting



# Verwendung des Begriffs Bug

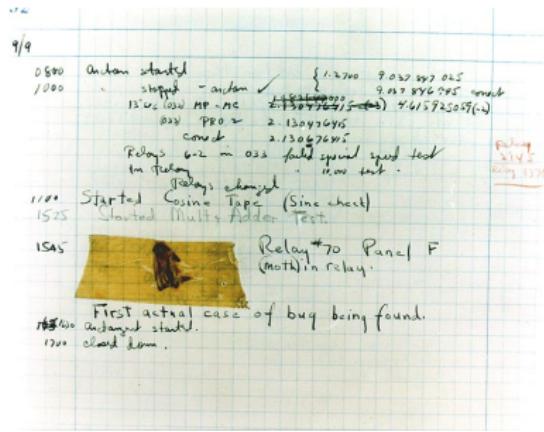


Abbildung 3: Erster echter Bugreport  
 Public Domain

<http://commons.wikimedia.org/wiki/File:H96566k.jpg>

- Schon Ende des 19. Jahrhunderts gebräuchlich
- Vorstellung von kleinen Tieren, die an Telefonleitungen knabbern
- Vorfall am Mark II Aiken Relay Calculator
- Verbreitung der Bezeichnung durch Grace Hopper



# Softwarefehler

## Definition

Ein Softwarefehler führt während des Programmablaufs zu einem Abweichen von der Spezifikation.

- Komplexe Software ist in der Regel nie fehlerfrei
- Mathematische Verifizierung ist nur bei einfachen Programmen praktikabel
- Fehlerfreiheit der Software geht von fehlerfreien Spezifikationen aus  
⇒ Qualitätsmanagement hilft, Softwarefehler zu reduzieren



# Kategorisierung von Softwarefehlern

## Zeitpunkt

- Zur Kompilierzeit
- Zur Laufzeit
- Zeitpunkt während der Laufzeit



# Kategorisierung von Softwarefehlern

## Zeitpunkt

- Zur Kompilierzeit
- Zur Laufzeit
- Zeitpunkt während der Laufzeit

## Ursache

- Syntax
- Softwaredesign
- Nutzereingabe
- Netzwerk-synchronisation
- Parallelität



# Kategorisierung von Softwarefehlern

## Zeitpunkt

- Zur Kompilierzeit
- Zur Laufzeit
- Zeitpunkt während der Laufzeit

## Ursache

- Syntax
- Softwaredesign
- Nutzereingabe
- Netzwerk-synchronisation
- Parallelität

## Auswirkung

- Bedienung erschwert
- Nicht-Erfüllung der Anforderungen
- Endlosschleife
- Programmabsturz
- Einfrieren



# Mars Climate Orbiter

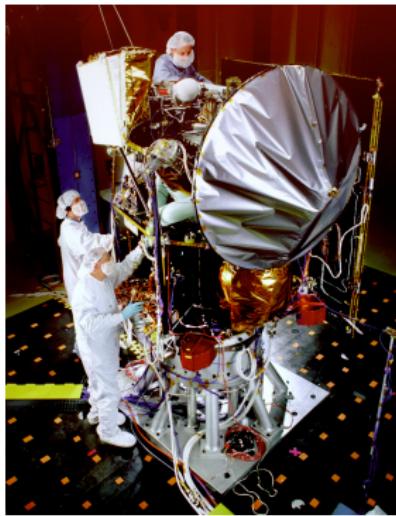


Abbildung 4: Mars Climate Orbiter  
Public Domain  
[http://commons.wikimedia.org/wiki/File:Mars\\_Climate\\_Orbiter\\_during\\_tests.jpg](http://commons.wikimedia.org/wiki/File:Mars_Climate_Orbiter_during_tests.jpg)

- Marssonde der NASA im Rahmen des Discovery-Programms
- Kurskorrekturen bei Marsannäherung
- NASA-Berechnung in [Newton · Sekunde]
- Lockheed-Martin-Annahme in [Pound-force · Sekunde]

## Ergebnis

Totalverlust der Sonde bei Eintritt in die Marsatmosphäre

# Ariane V88



- Erstflug der Ariane 5 (ESA-Trägerrakete)
- Komponenten aus der Ariane 4 wurden ungeprüft übernommen
- Speicherüberlauf führte zu Versand von Statusdaten statt Messdaten
- Steuersystem misinterpretiert Statusdaten

## Ergebnis

Selbstzerstörung der Rakete

Abbildung 5: Ariane V88  
Phrd/de:user:Stahlkocher, CC BY-SA 3.0  
[http://commons.wikimedia.org/wiki/File:Ariane\\_501\\_Cluster.svg](http://commons.wikimedia.org/wiki/File:Ariane_501_Cluster.svg)



# Heartbleed

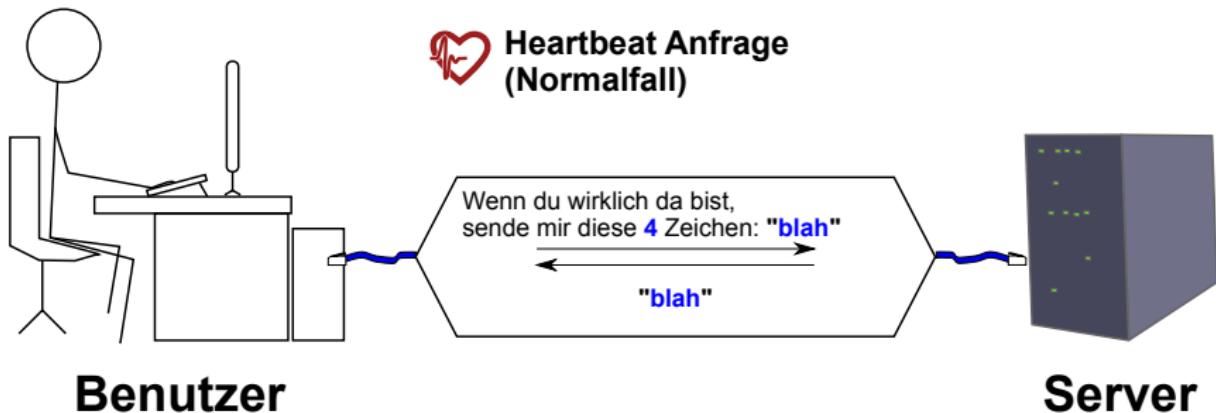


Abbildung 6: vgl. Heartbleed Bug  
SomeUser953, Patrick87, CC BY-SA 3.0

[http://commons.wikimedia.org/w/index.php?title=File:Heartbleed\\_bug\\_explained.svg&lang=de](http://commons.wikimedia.org/w/index.php?title=File:Heartbleed_bug_explained.svg&lang=de)



# Heartbleed

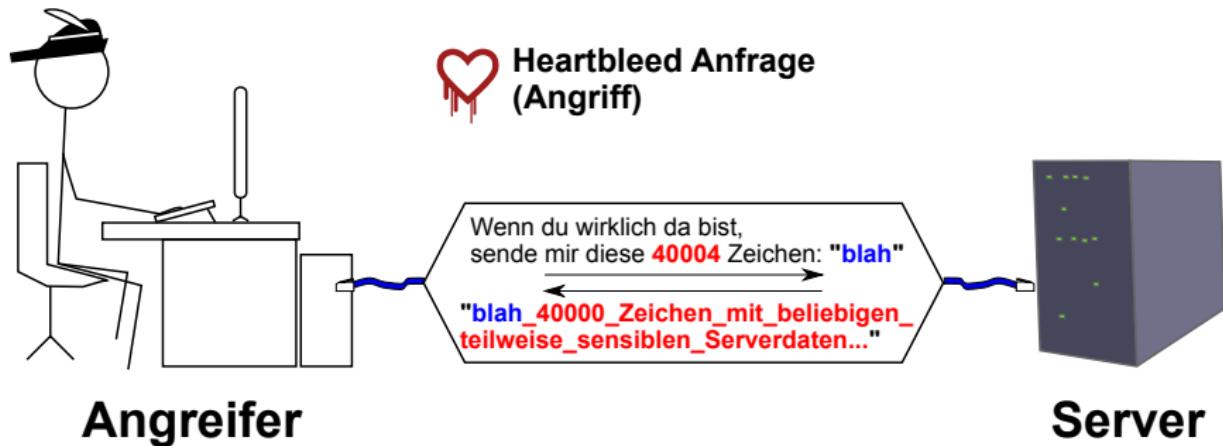
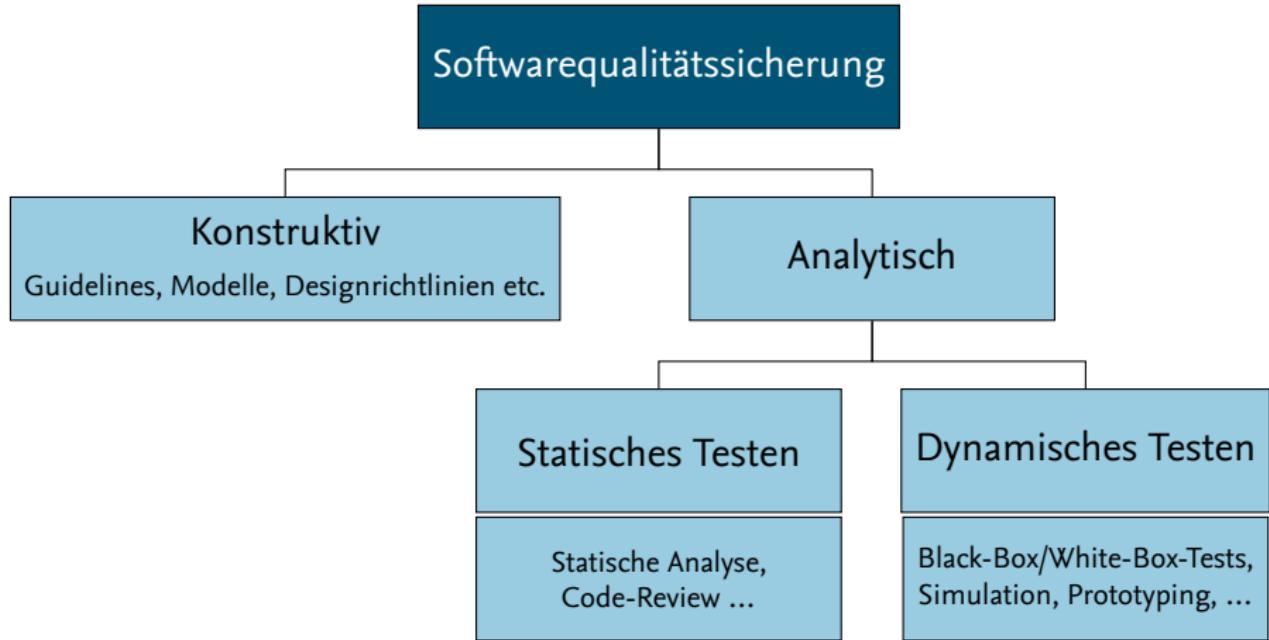


Abbildung 6: vgl. Heartbleed Bug  
SomeUser953, Patrick87, CC BY-SA 3.0  
[http://commons.wikimedia.org/w/index.php?title=File:Heartbleed\\_bug\\_explained.svg&lang=de](http://commons.wikimedia.org/w/index.php?title=File:Heartbleed_bug_explained.svg&lang=de)

# Welche Methoden helfen mir Fehler frühzeitig zu erkennen?



# Qualitätssicherung



# Modultests

- Testen abgeschlossener Einheiten
  - Einzelne Klassen
  - Einzelne Funktionen
- Überprüfung, ob Eingabewerte zu definierten Ausgaben führen
- Black-Box-Testing
  - Testen der Außenwirkung, keine Kenntnis der Umsetzung nötig



# Black-Box-Test

# Beispiele

## String-Funktion capitalize()

Alle Buchstaben eines Strings sollen in Großbuchstaben umgewandelt werden. Andere Zeichen bleiben unverändert.

| Eingabe | Ausgabe |
|---------|---------|
| abcdef  | ABCDEF  |
| aBcDeF  | ABCDEF  |
| !AbCd!  | !ABCD!  |

## String-Funktion replace()

In einem String (E1) sollen Zeichen (E2) durch andere Zeichen (E3) ersetzbar sein.

| E1     | E2 | E3 | Ausgabe |
|--------|----|----|---------|
| abcdef | a  | z  | zbcdef  |
| aBcDeF | a  | z  | zBcDeF  |
| !AbCd! | a  | z  | !AbCd!  |



# Manuelles Testen

- Statisches Testen möglich (Programm muss nicht lauffähig sein)
- Manuelle Interaktion mit dem Programm
- Ergebnis des Tests sollte ein Protokoll sein  
Review-Protokoll, Bug-Report, Feature-Report

## Vorteile

- Überprüfung des Quelltextes möglich
- Einhaltung von Richtlinien kann überprüft werden

## Nachteile

- Hoher personeller/manueller Aufwand
- Manuelle Interaktion birgt Probleme (Unregelmäßigkeiten)
- Ergebnis evtl. subjektiv beeinflusst



# Automatisiertes Testen

- Definierte Aktion im Rahmen eines Programmablaufs
- Automatisierter Ablauf von Programmschritten
- Ergebnis wird automatisiert ermittelt

## Vorteile

- Schnelle automatisierte Durchführung möglich
- Ergebnisse sind objektiv und eindeutig

## Nachteile (Schwierigkeiten)

- Tests müssen sinnvoll und nützlich definiert werden
- Tests können trügerische Sicherheit vermitteln (z. B. bei Lücken)
- Testumgebung muss eingerichtet und konfiguriert werden



# Was mache ich, wenn ich einen noch unbekannten Fehler entdeckt habe?



# Bestandteile eines Bug-Reports

- Fortlaufende Nummerierung
- Zusammenfassung (meist als Überschrift)
- Genaue Beschreibung des Fehlers
  - Betroffene Komponente/n
  - Erwartetes Verhalten der jeweiligen Komponenten
  - Screenshots
- Schritte zur Reproduzierung des Bugs
  - Genaue Software-Version (z.B.: Revision, Buildnummer)
  - Eigene Plattform (Betriebssystem, Rechnerarchitektur,...)
- Schweregrad und Priorität des Fehlers
  - Kritisch, wichtig, unwichtig, Verbesserung
- Status der Bearbeitung des Bugs
  - Neu, angefangen, behoben, geschlossen, wieder geöffnet



# Dokumentation des Bug-Reports in API

1/2

## Dokumentation

- Angabe der Revisionsnummer
- Schritte zur Reproduzierung des Bugs
- Genaue Beschreibung des Fehlers
- ggf. das erwartete Verhalten des jeweiligen Moduls

Die Dokumentation erfolgt auf GitHub im Reiter Issues oder auf der entsprechenden Wiki-Seite.



# Dokumentation des Bug-Reports in API

2/2

## Beispiel

|                            |                                                                                                              |
|----------------------------|--------------------------------------------------------------------------------------------------------------|
| <b>Lfd. Nr.:</b>           | #1                                                                                                           |
| <b>Revision:</b>           | 7aa8c84d                                                                                                     |
| <b>Bearbeiter:</b>         | Peter Pauly                                                                                                  |
| <b>Fehlerbeschreibung:</b> | Strings mit Umlauten oder ß lassen das Programm abstürzen, das Fenster schließt sich sofort nach Bestätigung |
| <b>Reproduktion:</b>       | Öffnen des Programms, Auswahl der Eingabezeile, Eingabe eines Strings mit Umlaut (klein/groß) oder ß         |



## Gibt es Fragen oder Anmerkungen zu dem Thema Bug-Reporting?



# Abgehakt

## Dokumentation und Bug-Reporting

Als Teilnehmer soll ich am Ende dieser Übung...

- die Dokumentation eines Sprints erstellen können
- den Quelltext übersichtlich halten können
- den Quelltext sinnvoll kommentieren können
- Bug-Reports anfertigen können



# Aufgabe



1. Herunterladen des fehlerhaften Konvertierungsprogramms API-Materialien/Bug-Report
2. Die zwei eingebauten Fehler finden
3. Regel zur Reproduzierung identifizieren
4. Bug-Report anfertigen
5. Bug-Report in #uebunginteraktiv veröffentlichen



# Beispiellösung

# Bug 1

## Bugreport #1: Falsche Konvertierung Meilen

**Revision:** 0c51bbe9fba8b1b3dc1e12fffa73a67528089feb

**Bearbeiter:** Peter Pauly

**Beschreibung:**

**Erwartet:** 1 mi wird in 1609 m / 0.87 NM / 5279 ft umgerechnet

**Beobachtet:** 1 mi wird in 1852 m / 1 NM / 6076 ft umgerechnet

**Reproduktion:** Start des Programms

Eingabe von „cDist“ zur Wahl des Modus

Eingabe der Eingangseinheit „mi“

Eingabe einer beliebigen anderen Ausgabeeinheit

**System:** Arch Linux, GCC 8.1.0-1

**Mögl. Lösung:** Überprüfung der Umrechnungsformel von Meilen in andere Einheiten.



# Beispiellösung

# Bug 2

## Bugreport #2: Absturz bei Konvertierung von Fahrenheit

|                      |                                                                                                                                                                                                    |
|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Revision:</b>     | 0c51bbe9fba8b1b3dc1e12fffa73a67528089feb                                                                                                                                                           |
| <b>Bearbeiter:</b>   | Marc Illic                                                                                                                                                                                         |
| <b>Beschreibung:</b> | Runtime Exception „Division by zero“ und Programmabsturz bei der Konvertierung von Fahrenheit.                                                                                                     |
| <b>Reproduktion:</b> | Start des Programms<br>Eingabe von „cTemp“ zur Wahl des Modus<br>Eingabe der Eingangseinheit „F“<br>Eingabe einer beliebigen anderen Ausgabeeinheit<br>Eingabe einer beliebigen Eingangstemperatur |
| <b>System:</b>       | Windows 7                                                                                                                                                                                          |



Jetzt besteht die Möglichkeit, das Sprintmeeting durchzuführen.

Protokolliert bitte

- die bearbeiteten Aufgaben der Vorwoche.
- die Zwischenstände der geplanten Aufgaben.
- die in der kommenden Woche zu bearbeitenden Aufgaben.



# Ende

Vielen Dank für eure Aufmerksamkeit!

