



```
*
```

```
* 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
```

```
*
```

Software-Dokumentation und Bug-Reporting

Prof. Dr.-Ing. Peter Hecker, Dipl.-Ing. Paul Frost, Andreas Dekiert M. Sc.,
7. Mai 2019

Agenda

- 09. April Einführung
- 16. April Softwareprojektmanagement
- 23. April Entwicklungstools
- 30. April GitHub
- 07. Mai Software-Dokumentation und Bug-Reporting**
- 14. Mai Einführung Arduino
- 21. Mai Dateieingabe und -ausgabe
- 28. Mai Einführung von Qt
- 4. & 11. Juni **Tag der Lehre und Exkursionswoche**
- 18. Juni GUI-Erstellung mit Qt
- 25. Juni Serielle Kommunikation
- 02. Juli API-Anleitung und Projektarbeit
- 09. Juli Vorbereitung der Abgabe
- 16. Juli **Fragen**
- 12. August 10:00 **Abgabe**



Lehrziele

Software-Dokumentation und Bug-Reporting

Als Teilnehmer soll ich am Ende dieser Übung...

- den Quelltext übersichtlich halten können
- den Quelltext sinnvoll kommentieren können
- Bug-Reports anfertigen können



Software-
Dokumentation

// ...

Einordnung

Programmier-
richtlinien

Kommen-
tierung



Hintergrund

Für den Kunden

- Nachweis des Arbeitsfortschritts

Für das Projektmanagement

- Projektstatus
- Risikobewertung

Für andere Entwickler

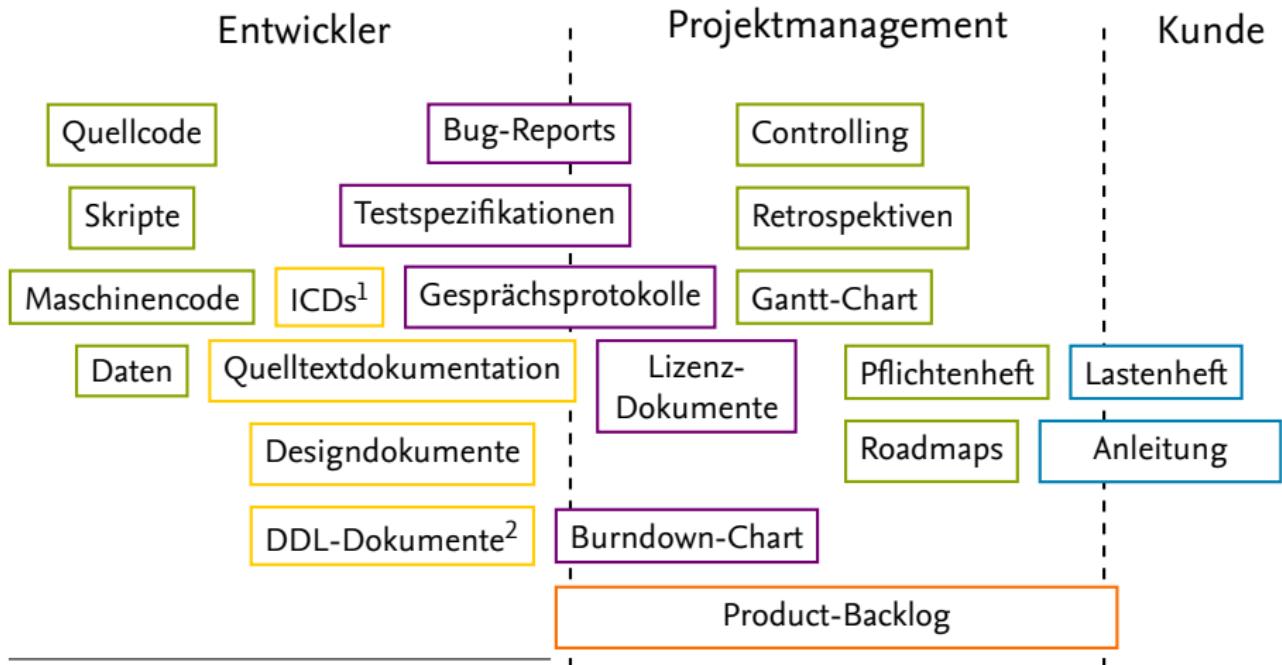
- Definition von Schnittstellen zu anderen Modulen
- Kommunikation zwischen Mitarbeitern ermöglichen/erleichtern

Für den Entwickler selbst

- Nachvollziehbarkeit auch nach längerer Zeit



Klassifikation nach Nutzergruppen



¹Dokumentation zur Schnittstellenansteuerung

²Datendefinitionssprache

(Software-) Designdokumente

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



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 portable 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

- Teammitglied muss oft nur wissen, was gemacht wird, aber nicht wie
- Für andere verfügbare Funktionen werden im Header dokumentiert

Listing 11: Deklaration (Header-Datei .h)

```
// 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;
}
```



Doxxygen-Kommentare

| | |
|-------------|--|
| \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... |

Doxxygen

Programm zur Generierung einer HTML-Dokumentation



Dokumentation im Rahmen von API

Dokumentation

- Standard-C++-Methoden müssen nicht kommentiert werden
- Jede eigene Routine soll mindestens ein kurzer Kommentar begleiten.
Beispiel: `divide()`
- Zusätzlich soll jedes Gruppenmitglied eine Funktion mit Ein- und Ausgabeparametern implementieren und kommentieren.
Beispiel: `calcCircumfence()`
→ Verweis im Wiki zwecks Bewertung (siehe unten)

Beispiel

Bearbeiter: Andreas Dekiert

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



Gibt es Fragen oder Anmerkungen zu dem Unterthema
Quelltextkommentierung?



Abgehakt

Software-Dokumentation und Bug-Reporting

Als Teilnehmer soll ich am Ende dieser Übung...

- 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 1: 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



Mars Climate Orbiter

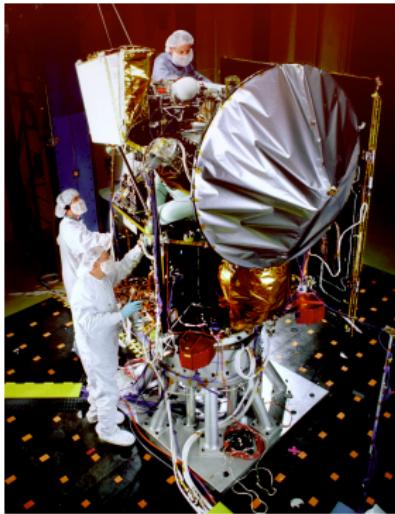


Abbildung 2: Mars Climate Orbiter
Public Domain
http://commons.wikimedia.org/wiki/File:Mars_Climate_Orbiter_during_tests.jpg

- 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



Abbildung 3: Ariane V88

Phrd/de:user:Stahlkocher, CC BY-SA 3.0
http://commons.wikimedia.org/wiki/File:Ariane_501_Cluster.svg

- 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



Heartbleed

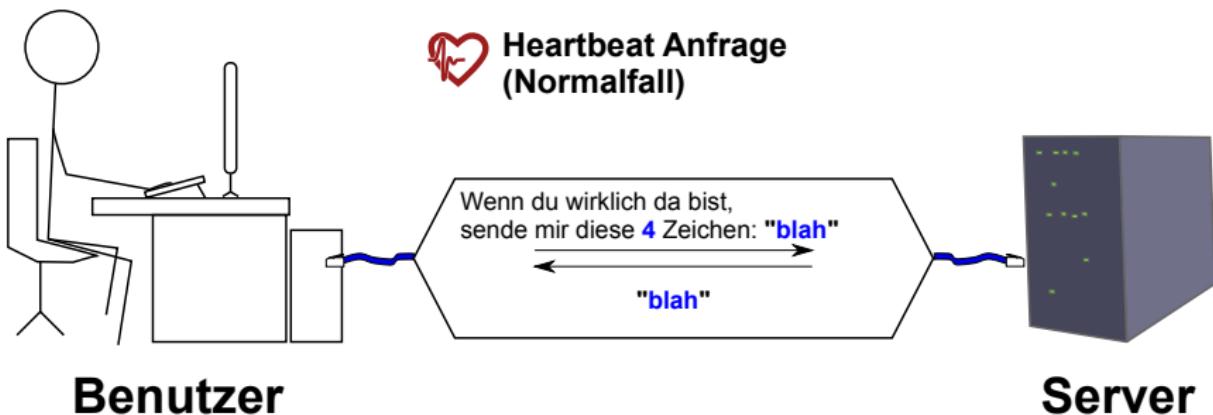


Abbildung 4: vgl. Heartbleed Bug
SomeUser953, Patrick87, CC BY-SA 3.0
http://commons.wikimedia.org/w/index.php?title=File:Heartbleed_bug_explained.svg&lang=de

Heartbleed

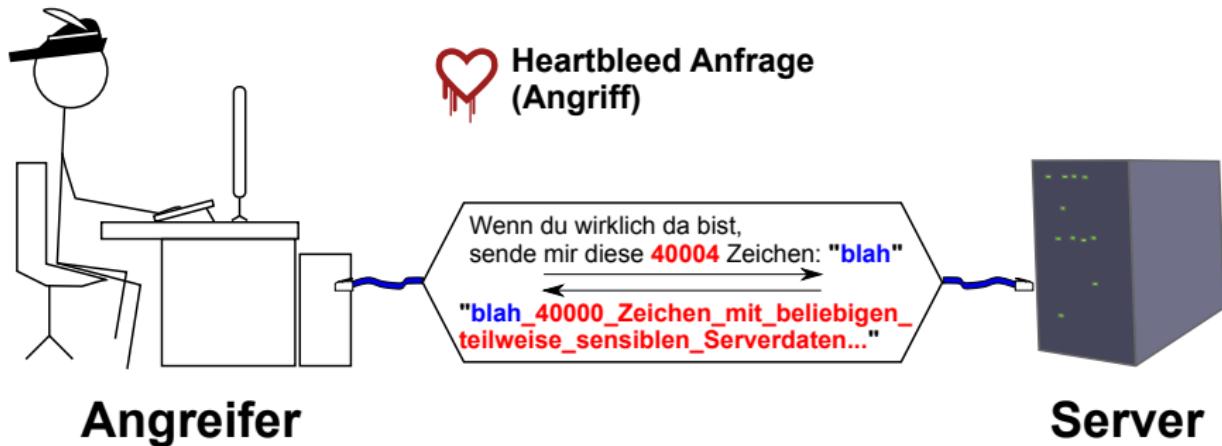


Abbildung 4: vgl. Heartbleed Bug

SomeUser953, Patrick87, CC BY-SA 3.0

http://commons.wikimedia.org/w/index.php?title=File:Heartbleed_bug_explained.svg&lang=de

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

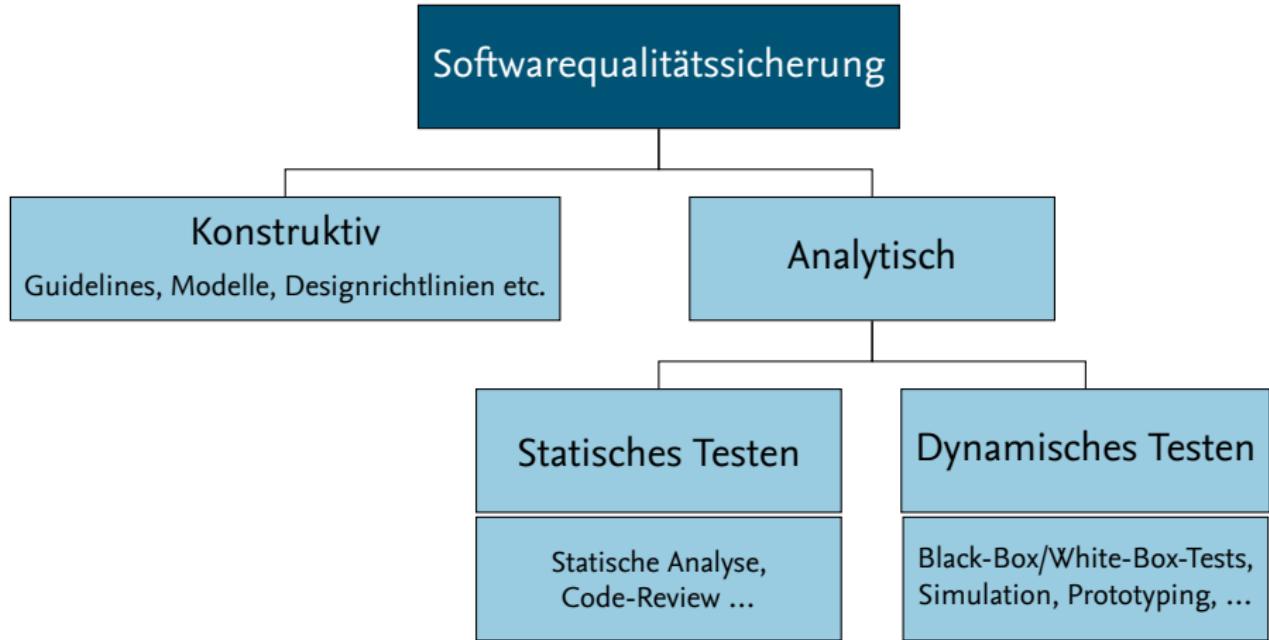
| Zeitpunkt | Ursache | Auswirkung |
|---|---|---|
| <ul style="list-style-type: none">▪ Zur Kompilierzeit▪ Zur Laufzeit▪ Zeitpunkt während der Laufzeit | <ul style="list-style-type: none">▪ Syntax▪ Softwaredesign▪ Nutzereingabe▪ Netzwerk-synchronisation▪ Parallelität | <ul style="list-style-type: none">▪ Bedienung erschwert▪ Nicht-Erfüllung der Anforderungen▪ Endlosschleife▪ Programmabsturz▪ Einfrieren |



Welche Methoden helfen mir Fehler frühzeitig zu erkennen?

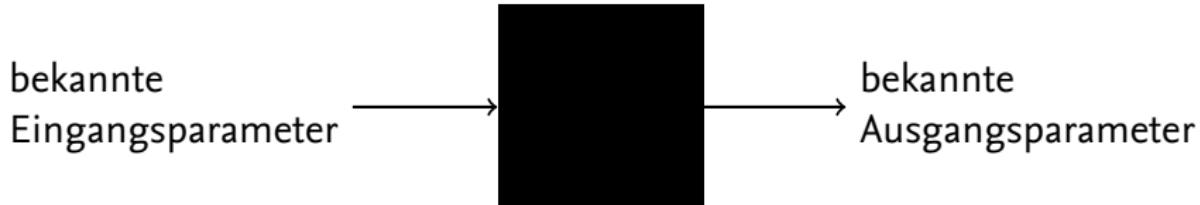


Qualitätssicherung



Dynamisches Testen

- 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) ersetzt werden.

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



Manuelles Testen

- Manuelle Interaktion mit dem Programm oder Quellcode
- 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?



Bug-Report

Antwort: Bei noch unbekanntem Fehler, Bug-Report erstellen.

Protokollierung

- Teil der Qualitätssicherung und des Projektmanagements
- „Problem wird nicht vergessen“

Hilfe für den Programmierer

- Kann Fehler nachvollziehen und reproduzieren
- Produktivitätssteigerung
- Behebung von Fehlern die der Programmierer nicht bemerkt hätte
- „Bessere Software“



Bestandteile eines Bug-Reports

- Eindeutige Nummer
- 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



Dokumentation des Bug-Reports in API

1/2

Dokumentation

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

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



Dokumentation des Bug-Reports in API

2/2

Beispiel

| | |
|----------------------------|--|
| Lfd. Nr.: | #1 |
| Revision: | 7aa8c84d |
| Bearbeiter: | Peter Pauly |
| Fehlerbeschreibung: | Strings mit Umlauten oder ß lassen das Programm abstürzen, das Fenster schließt sich sofort nach Bestätigung |
| Reproduktion: | 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

Software-Dokumentation und Bug-Reporting

Als Teilnehmer soll ich am Ende dieser Übung...

- 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. Zwei Fehler finden
3. Regel zur Reproduzierung identifizieren
4. Bug-Report anfertigen
5. Bug-Report in #fragen 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

Revision: 0c51bbe9fba8b1b3dc1e12ffffa73a67528089feb

Bearbeiter: Marc Ilic

Beschreibung: Runtime Exception „Division by zero“ und Programmabsturz bei der Konvertierung von Fahrenheit.

Reproduktion: Start des Programms

Eingabe von „cTemp“ zur Wahl des Modus

Eingabe der Eingangseinheit „F“

Eingabe einer beliebigen anderen Ausgabeeinheit

Eingabe einer beliebigen Eingangstemperatur

System: 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!

