QA Summary

Table of Content

QA Management QM

Def: Comprises all activities that are necessary to fulfil the achieve the quality requirements for a (software) product

Subdivision into:
- Constructive quality management: during development
- Analytical quality management: analyzing the product

Constructive QM

JavaDoc

- **Erste Zeile**
  - Beginnt mit begin-comment (`/**`)
  - Kurze Zusammenfassung der Methode
  - Automatisches übernehmen in Methodentabelle
- **Inline HTML-Links** erzeugen externe Links und können in allen Kommentaren verwendet werden.
- **Referenzen** auf andere Klassen, Methoden etc. können mit dem @see erzeugt werden.
- Erzeugt **HTML-Dokumentation**
- **Letzte Zeile**
  - Endet mit end-comment (`*/`)

```
/**
  * Calculates the volume of a sphere.
  *
  * @param r The radius of the sphere; {@code r >= 0}
  * @return Volume of the sphere with given radius
  * @throws IllegalArgumentException
  *         if the radius is negative
  * @see Sphere
  * @see <a href="https://en.wikipedia.org/wiki/
  *         Sphere_volume">Sphere Volume</a>
*/
public static float calculateSphereVolume(float r) {
    if (r < 0) {
        throw new IllegalArgumentException("…");
    }

    return 4f/3f * r * r * r * Math.PI;
}
```

- **Konstrukte von Javadoc**
  - <p> trennt Absätze (HTML-Tags erlaubt)
  - Leerzeile zwischen Beschreibung und Liste von tags
  - Erste Zeile, die mit einen "@" beginnt, beendet die Beschreibung
- Wichtige **Tags**:

`@author`	(Nur Klassen, Interfaces)
`@version`	(Nur Klassen, Interfaces)
`@param`	(Nur Methoden & Konstruktoren)
`@return`	(nur für Methoden)
`@code`	(für Codefragmente in der Dokumentation)
`@exception`	(@throws funktioniert auch)
`@see`	
`@since`	
`@serial`	(or @serialField or @serialData)
`@deprecated`	(sollte nicht mehr verwendet werden)
`@link`	(Referenz auf andere Klasse, z.B. {@link Client})

Exception Handling

- Ausprobieren: **try**
- Auffangen: **catch**
- Exceptiones auslösen: **throw**
- Seiteneffekte beseitigen: **finally**

```
try { /* Folge von Anweisungen */}
catch(MyException e)     {/* optionale Reaktion */}
catch(OtherException e) {/* optionale Reaktion */}
...
finally {/* optionale abschliessende Massnahmen */}
```

```
public static void main( String args[] ) {
   ...
   try {
         calculateSphereVolume(-0.1f);
   }
   catch ( IllegalArgumentException e )  {
        System.out.println(e.getMessage());
   }
   ...
}
```

Logging

Requirements:
- Scope of logging can be controlled centrally for each process and decentrally in the code using suitable log levels.
levels (log level)
- Can be switched off centrally for runtime measurements
- Can be archived/long-term storage (log file) with timestamp
- Logging in log file and console (real-time tracking) with own log level
- Tools for filtering, searching, jumping, highlighting
- Optional process interruption/user interaction (message window) in the event of critical errors
- Connection of system logging (e.g. event display in Windows)

Development / Team Culture

This includes the following elements, among others:
- The defined and practiced processes
- Interaction and communication within the team (respect, fault tolerance, tone / style)
- Commitment / perceived commitment of developers to software quality
- Commitment / perceived commitment of developers to defined processes (schedule, checklists, communication)
- Responsibility for improving processes and product

Responsibility Assignment Matrix (RAM)

Aufgabe	Loris	Raphael	Rahel
QA-Vorlesung			X
Meilenstein Abnahmen	X	X	X
Repository Aufsetzen			X
Discord Aufsetzen		X	

Responsible, Accountable, Consulted, Informed (RACI)

Describes the roles of the team members in the project:
- Responsible: responsible
- Accountable: accountable
- Consulted: consulted
- Informed: informed

Task	Alice	Bob	Alan	Eve
Nickname	R A C I	R A C I	R A C I	R A C I
Chat	R A C I	R A C I	R A C I	R A C I
CLI	R A C I	R A C I	R A C I	R A C I

Analytic QM

Various manual checks carried out during software development:
- Review in commenting technique
- Fast, cheap, flexible, often not thorough
- Informal review using the structured walkthrough technique
- Medium use of resources
- Formal inspection using the meeting technique (Fagan inspection)
- Expensive and time-consuming, but thorough
- Checklists as part of project management and before submission (consequence RACI)
- Mental load is shared, medium use of resources

Code Review

- Manual procedure for analytical quality assurance
- Code is "proofread" by a person who was not directly involved in its creation and checked for
previously defined criteria/checklists
- The following questions can be asked, for example
- Do I understand this code?
- Does the code adhere to the agreed programming style?
- Would I expect this code at this point in the project?
- Does it already have code with similar functionality elsewhere?
- Can the readability of this code be improved?
- Could I maintain this code and make changes?

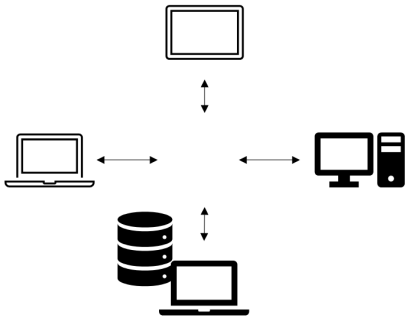
Testing Procedures

- Compiled, executable programme provided with input values as part of the test activities and executed
- Test programme in a real environment:
- Random sampling of input parameters à Correctness not proven
- "Testing may only show the presence of bugs but never their absence" (E. Dijkstra)
- Black box test:
- Function-orientated procedures, without knowledge of the implementation
- e.g. tests through equivalence class formation
- White-box test:
- Structure-orientated procedures, with knowledge of the implementation
- e.g. control flow-orientated tests

Black box process

Testing the completeness of the implementation of a specification by forming equivalence classes
- Values from equivalence class:
- Lead to identical functional behaviour
- Testing identically specified programme function
- Formation of equivalence classes based on specification
à All specified programme functions are tested

Create test cases after forming equivalence classes by selecting test data from many (ideally: all) valid equivalence classes.
(ideally: all) valid equivalence classes
- Conversely, perform tests for invalid equivalence classes to check correct system behaviour for such cases.
behaviour for such cases



Beispiel im Projekt: Ping/Pong
- Verbindungsverlust eines Clients
- Verbindungsverlust des Servers

Wie testen?
- Stecker ziehen
- Wi-Fi abschalten

Black-Box-Testing:
- Keine Kenntnis der Implementation
- Bei allen Gruppen gleich

•Netzwerkprotokoll:

- Blick «in die Box»
  - Was ist der Input?
  - Was wird verschickt?
- Implementation ist relevant und bekannt

Stabilität:

- Äquivalenzklassenbildung & Test Cases aufgrund des Netzwerkprotokoll
- Testing via z.B. telnet oder EchoClient
- Einfache Tests für alle gleich
  - z.B. leerer String schicken
- Weiteres Testen mit Implementationswissen
  - Beispiel:
    - Separator $
    - Chat-Implementierung chatallhi
    - $$$hi an Server geschickt wird?

Application of testing techniques

- Module test (unit test)
- Checking the individual modules/classes (separately from each other)
- Testing the correct functioning of a module/class in relation to the specification
- Integration test
- Checking the interaction of the modules
- Step-by-step combination of the modules to form the overall system
- The focus is on checking the correct interaction of the modules
- System/acceptance test
- Checking the functionality and performance of software against the requirements defined requirements
- Advantage of testing in different phases:
Reduction of the respective complexity of the testing activities to a manageable level

Unit Test

Each unit (function, class, ...) and application are tested separately by replacing the layer layer above (using the unit) with a test frame and the layer below (units used by it) with "pseudo modules".
(units used by them) by "pseudo modules"
- Test programme:
- Executable programme consisting of test object and test environment
- Test environment:
- Consists of modules that simulate the environment of the test object
- Components are test frames and pseudo modules
- Test frame (driver):
- Programme that simulates the use of the test object
- e.g. a test frame simulates a module that imports the test object
- Pseudo modules (stubs):
- Simulate modules (or units) used by the test object
- Pseudo modules are used, for example, to simulate all modules from which a module to be tested is imported.

- **JUnit** ist ein Framework zum Testen von Java Programmen mit Fokus auf Unit Test
- Weitere Dokumentation ist unter https://junit.org/junit5/ zu finden

```
public class SphereVolumeTest {

    @Test
    void radiusPositiveVariousHappyCases(){
        assertEquals(calculateSphereVolume(1f), Math.PI * 4f / 3f, 0.0001f);
        assertEquals(calculateSphereVolume(5f), Math.PI * 500f / 3f, 0.0001f);
        assertEquals(calculateSphereVolume(12f), Math.PI * 2304f, 0.0001f);
    }

    @Test
    void radiusNegativeThrowsIllegalArgumentException(){
        assertThrows(IllegalArgumentException.class,
            () -> calculateSphereVolume(-1f));
    }
}
```

Test-driven Development

«Testgesteuerte Entwicklung»:

- Verfahren in Softwareentwicklung
- Software-Tests (in der Regel Unit Tests) konsequent vor testenden Komponenten entwickeln

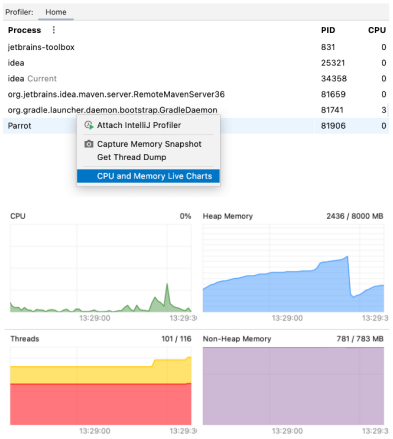
Profiler

Anforderungen:

- Zeit je Methodenausführung
- Feststellen von Bottlenecks
- Speicherverteilung und Auslastung
- Finden von Memory Leaks

Beispiel: IntelliJ Profiler

- https://www.jetbrains.com/help/idea/profiler-intro.html



Quelle: https://www.jetbrains.com/help/idea/cpu-and-memory-live-charts.html

Bug Report

- **«Bug»** = spezifisches Fehlverhalten einer Software
- **«Bug Report»** = spezifischer Bericht des Bugs, damit Entwickler Fehler beheben können

Ich habe gerade einmal das Spiel ausprobiert, bei mir hat es aber nicht richtig funktioniert. Könnt ihr euch das mal anschauen?

Problembeschreibung: NullPointerException in server.ClientHandler.java wenn ein Client versucht zu Würfeln wenn er nicht am Zug ist.
Version: 2eec824
Schritte zur Reproduktion:
1. Server + 3 Clients starten
2. Alle Clients einem Spiel beitreten
3. Spieler 1: Würfeln und Ziehen
4. Spieler 2: Würfeln
5. Spieler 1: Würfeln

Guter Bug Report **beinhaltet folgendes**:

- Welche Version(en) des Programms (commit hash / branch) sind betroffen?
- In welcher Umgebung (Betriebssystem, Java Version, etc.) tritt das Problem auf?
- Wie genau äussert sich das Problem?
- Was sind Ausgaben/Logs vor, während und nach dem Problem?
- Welche Schritte sind nötig, um das Problem herbeizuführen?
- Ist das Problem zuverlässig reproduzierbar?

Guter Bug Report **beinhaltet folgendes nicht**:

- Vom Problem unabhängige Information
- Wertungen/Anschuldigungen
- Redundante Information

Debugging

- Finding and correcting programming errors
- Subdivision into steps:

1. reproduce the problem: Can described error be reliably reproduced and if so, how?

2. isolate the problem: Which parts of the source code are executed when the problem occurs?

3. find the bug: How should the affected code behave and how does it behave instead?

4. fix the bug: Correct the above discrepancy in behaviour

«Quietschentchen» Debugging

- Quellcode Zeilenweise einem Hilfsobjekt (traditionell einem Quietschentchen) erklären
- Ziel ist keine externe Rückmeldung, sondern durch genaue Auseinandersetzung Fehler selbst zu finden

Quelle: https://adoreme.tech/rubber-duck-debugging-511ce81140b1

Pair Programming

- Working technique in software development in which two people work simultaneously on one computer
working on the creation of source code.
- Two roles:
- Person who writes code
- Person who checks code directly and provides feedback
- Swap roles regularly
- Often better and more readable code
- Increases information exchange between team members

Metrics and Quantifikation

Quantifizierung von Codequalität:

- Lines of Code pro Klasse
- Lines of Javavadoc / Lines of Code pro Klasse
- Code Coverage für Tests
- Lines of Code pro Methode
- Halstaead Complexity Measures
- Cyclomatic Complexity
- …

Aggregation von Metriken

- Average
- Max
- Min
- Mean
- …

Jacoco

Java-basiertes Tool für **Codeabdeckungsanalysen**

- Misst wieviel Quellcode durch Tests ausgeführt wird
- Erfasst und visualisiert Anweisungen, Zweige, Methoden und Klassen
- Dokumentation Online: https://www.eclemma.org/jacoco/

Installation:

- Einbindung in build.gradle

```
plugins {
        id 'jacoco'
}
```

Verwendung:

- Reports generieren mit `./gradlew test jacocoTestReport`
- Reports sind unter build/reports/jacoco/test/html