Qualitätssicherungskonzept

Synchronisation und Auswertung von Roboterfußballvideos

tj18b

Mitglieder
Dan Häßler
Robert Wagner
Sirk Petzold
Alex Eichhorn
Erik Diener
Jonas Wrubel
Tomas Daetz Chacon

Betreuer
Hans-Gert Gräbe
Tobias Wieprich
Tobias Jagla
André Köhler

Inhaltsverzeichnis

1 Dokumentationskonzept

Eine verständliche und konsequente Dokumentation ist von großer Bedeutung. Auch wenn sich die Dimension unseres Projekts in Grenzen hält, verhilft uns die Dokumentation dazu, den Überblick zu bewahren und erleichtert somit die Zusammenarbeit im Team. Auch zeitlich betrachtet darüber hinaus ist dies relevant, da diese Software auch dafür vorgesehen ist, nach dem Release von anderen Entwicklern gewartet und ausgebaut zu werden.

Die gesamte Dokumentation erfolgt auf Englisch.

1.1 Code-Dokumentation

1.1.1 Interne Dokumentation

Es wird angestrebt, dass der Quellcode so gut verständlich ist, so dass Kommentare überflüssig sind und der Code die beste Dokumentation für sich selbst ist. Interne Dokumentation sollte also sparsam eingesetzt werden und nur dann, wenn es nicht vermeidbar ist, dass der Quellcode ungeklärte Fragen hinterlässt.

1.1.2 Quelltextnahe strukturierte Dokumentation

Die quelltextnahe interne Dokumentation dient im Wesentlichen zur Beschreibung der implementierten Klassen und Funktionen. Als Minimum muss jede als public deklarierte Klasse und jeder als public oder protected deklarierte Teil davon (Attribute oder Methoden) dokumentiert sein. Ausnahmen bilden Methoden, deren Funktion als selbsterklärend angesehen werden kann, wie bei set- und get-Methoden. Auch Methoden, die von einer Superklasse überschrieben werden, müssen nicht zwingend dokumentiert werden.
Da diese Software ausschließlich in Java programmiert wird, nutzen wir als Dokumentationswerkzeug Javadoc.

1.1.3 Entwurfsbeschreibung

Die Entwurfsbeschreibung ist ein eigenständiges Dokument, welches alle relevanten Informationen zu den Struktur- und Entwurfsprinzipien dieser Software darbietet und die getroffenen Entscheidungen begründet. Mithilfe dieser Dokumentation wird zukünftigen Programmierern an diesem Projekt die Einarbeitung vereinfacht und das notwendige Fachwissen übermittelt. Die Entwurfsbeschreibung wird erst im Verlauf der Implementierungsphase entstehen, da sämtliche Modellierungs- und Design-Entscheidungen aktuell noch variabel sind und daher erst mit der Zeit eindeutig dokumentiert werden können. Dieses Dokument wird auf Deutsch erstellt.

1.2 Code

In diesem Abschnitt werden ein paar Vereinbarungen vorgestellt, die sich positiv auf den Dokumentationsaufwand auswirken sollen.

Guter Code muss folgende Eigenschaften besitzen:

- geringe Komplexität, andernfalls Anzeichen dafür, dass man den Code refaktorisieren sollte
- nicht mehrere Statements in einer Zeile
- eine Methode erledigt nur eine funktionale Aufgabe
- keine “hartcodierten” Zahlen (Magic Numbers) und Strings, d.h. falls in einer Funktion unbekannte Zahlen oder unbekannte Strings auftauchen, werden diese an eine Variable gebunden

1.3 Coding Standard

Ein einheitlicher Coding Standard ist sinnvoll um bestmögliche Lesbarkeit und Verständnis des Quelltextes zu garantieren.
Für dieses Projekt wird der offizielle Google Java Style Guide befolgt, weil er modern, konsequent und überaus verbreitet ist. Eine Ausnahme gehen wir bei dem Thema Einrückungen: Der Google Java Style Guide empfiehlt hier 2 Zeichen (4 bei Folgezeilen), wir legen uns allerdings auf 4 Zeichen fest.

1.3.1 Code Layout

Im Folgenden werden die wichtigsten Konventionen unseres Coding Standards erläutert.

Klammersetzung

if, else, for, do, while-Statements werden immer von Klammern umschlossen, auch wenn nur ein einziger Ausdruck enthalten ist.
Klammern werden stets am Ende der gleichen Zeile geöffnet.

Einrückungen

Für Einrückungen werden stets 4 Leerzeichen oder Tabs mit der Länge von 4 Zeichen verwendet.

Leerzeichen

Binäre Operationen sollten mit einem Leerzeichen umgeben sein. Dies gilt ebenfalls für den "–>"- und Ternary-Operator.
Direkt nach öffnenden Klammern, direkt vor schließenden Klammern und vor Kommas, Semikolons und Doppelpunkten sollten sie vermieden werden.

Kommentare/Javadoc

Kommentare beziehen sich stets auf die darunter liegende(n) Zeile(n) oder sie stehen am Ende der zu erläuternden Zeile.
Sie sollten immer aus ganzen natürlichsprachlichen Sätzen bestehen und trotzdem so prägnant wie möglich sein.
Natürlichsprachlich bedeutet, dass es zu vermeiden ist Code-Fragmente mit einzubeziehen.

Namenskonventionen

Klassen und Interfaces werden im UpperCamelCase deklariert (z.B. ControllerPane).
Methoden, Attribute und Parameter werden im lowerCamelCase deklariert (z.B. setMaxWidth).
Konstanten werden nur in Großbuchstaben deklariert und jedes Wort wird mit Unterstrich vom nächsten separiert (z.B. DURATION_IN_MS).
Wenn es sinnvoll erscheint können Abkürzungen und Akronyme verwendet werden. Beispielsweise können Variablen, die von irgendetwas die Anzahl zählen, mit # beginnen (z.B. #frame statt numberOfFrame).

1.3.2 Tools

Um sicherzustellen, dass gepushte Commits im Einklang mit unseren gewählten Richtlinien stehen, werden wir das Tool Checkstyle in die IDE integrieren. Dadurch werden

Commits nur akzeptiert, wenn der Code den Style Guide nicht verletzt, andernfalls wird der Commit mit einer Fehlerbeschreibung zurückgewiesen und der Entwickler muss den Code entsprechend überarbeiten.

2 Testkonzept

2.1 Einführung

Das Testkonzept berücksichtigt die Qualität der einzelnen Teile Ihrer Entwicklung (Komponententests) sowie die Qualität der Zusammenführung der Teile (Integrations- und Systemtest).
Hierbei halten wir uns grob an die IEEE829 (Testnorm) und an das Continuous Integration Prinzip.

2.2 Zu testende Eigenschaften

- Komponententests

  Videopane-Komponente, Log-Parsing, Setzen der Marker, Laden eines Videos

  Konfigurationsdateien, Schreiben der .mlt-Dateien, Tonspur, optionale Features

  Schaltflächen

- Integrationstests

  Zusammenarbeit der Komponenten (z.B. Tonspur und VideoPane)

  Synchronisation

  Parallele Wiedergabe mit Offsets

- Systemtest

2.3 Testansatz/-strategie

Grundsätzlich ist jede/s Klasse/Funktion/Feature der Software zu testen.
Zu jedem abgeschlossenen Issue wird ein Issue für dessen Tests angelegt.
Am Namen der Tests muss klar erkennbar sein, was getestet wird.

Um eine geordnetes Projekt zu gewährleisten, wird eine Ordnerstruktur angelegt, welche das Zuordnen der Tests vereinfachen soll.
Diese folgt folgendem Schema:

Package→Ordner (z.B. Test)→Ordner (z.B. Videopane)→...→KlassennameTest.java

Beim Bauen des Projekts sollen die Tests durchlaufen. Ein Test ist bestanden, wenn die zutestende Funktion den Anforderungen entspricht.

2.4 Zuständigkeiten

Die Testklasse wird von einem Teammitglied geschrieben, welches nicht die Klasse geschrieben hat. Hiermit soll gleichzeitig getestet werden, ob der Code verständlich geschrieben ist. Das Anlegen der Testissues obliegt dem Bearbeiter des Features.

2.5 Testrelevanz

Testrelevanz hat jede Funktion, welche Werte ausgibt, die in der Funktion bearbeitet wurden (Simple Getter-Funktionen müssen z.B. nicht getestet werden).

2.6 Risiken

Ein Risiko ist das planlose Durchführen der Tests. Außerdem ist der Zeitfaktor beim Schreiben der Tests kritisch. Deshalb ist das Schreiben der Tests im Fall der Fälle nieder priorisiert als das Schreiben der Features. Es muss darauf geachtet werden, dass Funktionen, die klar keine Testrelevanz haben, ausgelassen werden.

3 Organisatorische Festlegungen

Benutzung eines iterativen Entwicklungsmodell; das System wird als Folge von Release-Versionen zu vereinbarten Terminen entwickelt werden, dabei werden die Funktionalitäten Schritt für Schritt erweitert. Dabei werden zu den unten genannten Abgabetermine ein aktuelles Releasebündel bestehend aus Quellcode, Testmaterial, Entwurfsbeschreibung, Testbericht, Releaseplan, Aufwandanalyse sowie eine Demoversion des aktuellen Releases auf der Website.

Die Abgabedatei ist eine zip-Datei, wobei Quellcode, Beschreibungen und Testmaterial jeweils in eigenen Verzeichnissen abzulegen sind.

3.1 Abgabetermine

Termin	**Releasebündel**
7.01.2019	R1
21.01.2019	R2
04.02.2019	R3
04.03.2019	R4
18.03.2019	R5

3.2 Validierung

Zur Softwarequalitätssicherung ist Validierung unumgänglich, daher ist ein wöchentliches Treffen mit dem Auftraggeber vereinbart zur Validierung der Software. Dabei werden die aktuellen Releasebündel und den aktuellen Stand im Prozess diskutiert. Beginnend mit dem 10. Januar werden die Treffen Donnerstags um 15 Uhr durchgeführt. Dabei stehen folgende Räumlichkeiten zur Verfügung:

Termin	**Raum**
31.01.2019	P801
07.03.2019	P702
Sonst	P901