Qualitätssicherungskonzept

Inhaltsverzeichnis

1. Dokumentationskonzept

Bei der Implementierung sollten folgende Prinzipien eingehalten werden:

1.1 Prinzip der Verbalisierung

Ideen und Konzepte des Programmierers sollen im Programm möglichst gut ersichtlich sein.
Richtlinien und Konventionen für Bezeichner in Java nach Balzert:

1. Bezeichner(Identifier) sind natürlichsprachliche und problemnahe Namen oder verständliche Abkürzungen solcher Namen
2. Jeder Bezeichner beginnt mit einem Buchstaben: der Unterstrich (_) wird nicht verwendet.
3. Bezeichner enthalten keine Leerzeichen.
4. Generell ist Groß-/Kleinschreibung zu verwenden
5. Zwei Bezeichner dürfen sich nicht nur bezüglich der Groß-/Kleinschreibung unterscheiden
6. Es wird die deutsche Sprache verwendet
7. Auf Umlaute, wie z.B. “ß” ist zu verzichten
8. Ein Bezeichner aus mehreren Wörtern, dann beginnt jedes Wort mit einem Großbuchstaben
9. Jeder Befehl wird auf eine separate Zeile geschrieben.
10. Klassennamen beginnen immer mit einem Großbuchstaben, bestehen aus einem Substantiv im Singular, zusätzlich kann ein Adjektiv angegeben werden und die, die für eine GUI-Klasse stehen, enthalten das Suffix GUI.
11. Objektnamen beginnen immer mit einem Kleinbuchstaben, enden in der Regel mit dem Klassennamen, z.B. einKunde, beginnen bei anonymen Objekten mit ein, erster, a usw., z.B. aPoint, einRechteck und von GUI-Interaktionselementen beginnen kleingeschrieben mit zugeordneten Attributnamen der Fachkonzeptklasse gefolgt von dem Namen des Interaktionselements, z.B. nameTextfeld, nameFuehrungstext, speichernDruckknopf usw.
12. Attributnamen beginnen mit einem Großbuchstaben und sind detailliert zu beschreiben, z.B. ZeilenZähler (in UML),, WindGeschw, Dateistatus.
13. Operationsnamenbeginnen immer mit einem Kleinbuchstaben und in der Regel mit einem Verb, gefolgt von einem Substantiv z.B. drucke, aendere, zeigeFigur, leseAdresse, verschiebeRechteck, heißen getAttributname, wenn nur ein Attributwert eines Objektes gelesen wird, setAttributnamen, wenn nur ein Attributwert eines Objektes gespeichert wird, isAttributname, wenn das Ergebnis nur wahr (true) oder falsch (false) sein kann, z.B. isVerheiratet, isVerschollen
14. Einheitlicher Aufbau einer Klasse Attributdeklarationen, die für die gesamte Klasse gelten, Konstruktoren, Operationen, Operationen mit schreibenden Zugriff und Operationen mit lesendem Zugriff
15. Leerzeichen
    Bei binären Operationen und nach Schlüsselwörtern werden Leerzeichen verwendet. Keine Leerzeichen bei Punktnotationen und Klammerausdrücke.
16. Einrücken und Klammern von Strukturen
    Paarweise zusammengehörende geschweifte Klammern { } stehen immer in derselben Spalte untereinander. In der Zeile der geschweiften Klammer steht sonst nichts mehr. Alle Zeichen innerhalb eines Klammerpaars sind um einen Tabulatorsprung nach rechts eingerückt.

1.2 Prinzip der problemadäquaten Datentypen

Daten- und Kontrollstrukturen sind möglichst unverfälscht einzusetzen. Das Angebot an Konzepten einer Programmiersprache ist optimal zu verwenden.
Folgende Regeln sind zu beachten:
- Können die Daten durch Basistypen beschrieben werden, dann ist der geeignete Basistyp auszuwählen
- Der Typkonstruktor Feld ist zu verwenden, wenn man gleiche Datentypen zusammenfasst, Zugriff dynamisch berechnet wird, hohe Komponentenzahl oder/und Feldgrenzen existieren.

1.3 Prinzip der Verfeinerung

Das Programm bzw die Operationen werden durch Abstraktionsebenen (abstract) weiter spezifiziert. Alle Verfeinerungen sind substituiert.

1.4 Prinzip der integrierten Dokumentation

Wichtiger Bestandteil jedes Programms muss eine geeignete Dokumentation sein.
Bestandteile: Kurzbeschreibung des Programms, Verwaltungsinformationen, Kommentierung des Quellcodes
Die ersten beiden Angaben können in einem Vorspann zusammengefasst werden:
Programmname, Aufgabe: Beschreibung des Programms einschließlich der Aufgabe, ob es sich um ein GUI-, ein Fachkonzept- oder ein Datenhaltungs-Programm bzw. eine entsprechende Klasse (bei OOP) handelt, Zeit- und Speicherkomplexität des Programms, Name der Autors, Versionsnummer

1.4.1 Dokumentationskommentare mit Javadoc

Aufbau: /** Kommentar */

Kommentierung der Klasse erfolgt gleich in der ersten Zeile von vor den import-Dateien und enthält folgende Informationen: @param, @author, @version, @see, ausführliche Beschreibung der Klasse und deren Funktion, eventuelle Erklärung von Vererbungen

Kommentierung von Methoden erfolgt direkt über der Methode mit folgenden Informationen:@param, @return, @exception, kurze Beschreibung der Funktion der Methode und der enthaltenen Variablen HTML -Befehle können eingefügt werden, ebenso Hyperlinks.

1.5. Benutzerhandbuch

Während der laufenden Entwicklungsphase wird für jede Benutzerschnittstelle ein Kapitel zur Erklärung der Funktionen in ein Benutzerhandbuch geschrieben.

1.6 Entwurfsdokumentation

Das ist die erste Dokumentation, die sich an die anderen Projektteilnehmer richtet und die endgültige Dokumentation und Programmierung unterstützen soll. Es wird auf die Darstellung und Begründung aller wichtigen Entwurfsaspekte genauer eingegangen.

1.7 Dokumentation der Templates

Die Templates werden nach den obigen Regeln kommentiert. Da Sie nicht in der Sprache JAVA geschrieben sind gelten die Regeln für Javadoc nicht für Sie.

2. Testkonzept

2.1 Allgemeines

Heutzutage ist die Anzahl der Fehler bei der Entwicklung einer komplexen Software sehr hoch. Aus diesem Grund ist ein Testkonzept für den Entwurf guter Software notwendig, denn das ermöglicht Fehler aus einem Softwaresystem zu verringern. Des Weiteren ist es wichtig während der Entwicklung einer Software kontinuierlich zu testen. Jedes Projekt soll am besten in Teilprojekte geteilt werden. Dies erleichtert die Suche nach möglichen Fehlern und deren Korrektur. Die folgenden Testphasen sind bei dem Testablauf vernünftig und zu berücksichtigen:

- Komponententest
- Integrationstest
- Systemtest
- Abnahmetest

2.2. Komponententest

Der Komponententest spielt eine wichtige Rolle bei der Qualitätssicherung einer Software, da bei dieser Testphase einzelne Klassen und Methoden auf ihre Funktionalität, Korrektheit und Vollständigkeit getestet werden, um Fehler zu erkennen und diese zu beheben, bevor sie zu einem gesamten System hinzugefügt werden. Jede einzelne Komponente soll isoliert von anderen Komponenten des Systems mit Hilfe geeigneter Testbeispiele geprüft werden. Dadurch kann man verhindern, dass Einflüsse auf die Komponenten die Ergebnisse beeinflussen und jeder Fehler kann zu der entsprechenden Komponente zugeordnet werden. Bei jedem Test werden alle gefundenen Fehler oder falsche Ergebnisse dokumentiert.

2.3. Integrationstest

Der Integrationstest ist die Testphase, in der das Zusammenarbeiten einzelner Komponenten überprüft wird. Dieser Test geschieht nach dem Komponententest und vor dem Systemtest.
Der Zweck des Integrationstests ist es, die funktionalen Anforderungen sowie die Leistung und die Zuverlässigkeit des Systems zu überprüfen. Es ist möglich Fehler, die bei der Kommunikation zwischen den Komponenten auftreten, zu finden. Wenn Fehler auftreten, sind sie zu beheben und der Integrationstest wird widerholt bis der Test erfolgreich absolviert wird.

2.4. Systemtest

Bei dem Systemtest werden alle Komponenten des Programmes zusammengefügt und das gesamte System wird geprüft. In dieser Phase wird nicht aus der Sicht der Entwickler getestet sondern aus der Sicht des Anwenders, ob alle Anforderungen umgesetzt wurden.

2.5. Abnahmetest

Der Abnahmetest ist die letzte Testphase, die zu berücksichtigen ist. Die Software wird dem Auftragsgeber vorgestellt und dieser überprüft, ob die Software allen seinen Anforderungen entspricht. Ist das der Fall, dann wird die Software von dem Auftragsgeber genommen und das gesamte Projekt gilt als erfolgreich entwickelt worden.

2.6. Dokumentation

Alle Fehler, die während des Test auftreten, sollen gut dokumentiert werden. Es ist sinnvoll die Ursache der Fehler und deren Lösung zu dokumentieren, denn dies bietet bei zukünftigem Auftreten ähnlicher Fehler die Möglichkeit, diese schnell zu finden und gleichzeitig zu beheben.

3. Organisatorische Festlegungen

Alle Gruppenmitglieder müssen sich an die hier im Dokumentations- und Testkonzept getroffenen Vereinbarungen halten. Dies sollte vor Abschluss eines jeden Arbeitspaketes von den entsprechenden Verantwortlichen (für Dokumentation und Test) zumindest stichprobenartig überprüft werden.
Um das weitere erfolgreiche Fortschreiten des Projektes zu garantieren sind regelmäßige Gruppentreffen durchzuführen. Diese sollten wöchentlich stattfinden, am besten in Form eines persönlichen Treffens aller Gruppenmitglieder, oder bei zu hoher räumlicher Distanz mindestens als Telefonkonferenz (über Skype). Auf den Gruppentreffen sind die abgeschlossenen Arbeiten jedes Gruppenmitgliedes zu präsentieren und anschließend von allen zusammenzuführen. Außerdem muss auf jedem Gruppentreffen das Thema das nächsten Treffens festgelegt, dazu notwendige Arbeiten verteilt und zugehörige Deadlines vereinbart werden.
Falls ein Gruppenmitglied seine zu bearbeitenden Aufgaben aus persönlichen Gründen oder wegen unterschätzten Aufwands nicht rechtzeitig zur Deadline fertigstellen kann, dann ist frühestmöglich die Projektleiterin darüber in Kenntnis zu setzen. Diese ist dann verpflichtet den zusätzlichen Arbeitsaufwand auf andere freie Kapazitäten umzulagern oder wenn möglich neue Deadlines/Termine für Gruppentreffen festzulegen. Gleiches gilt falls ein Erscheinen zum vereinbarten Gruppentreffen nichtmehr möglich ist.
Der Stand der Arbeit jedes Gruppenmitglieds ist für die gesamte Gruppe verfügbar zu machen. Dies sollte über die momentan eingerichtete Facebookgruppe/Dropbox Ordner erfolgen und über das Git Repository. Gruppenmitglieder sollen die laufende Arbeit am Projekt gegenseitig kontrollieren und verifizieren. Dies ermöglicht einen einheitlichen Kenntnisstand der Gruppe sowie das rechtzeitige Beheben von Fehlern vor Deadlines/Gruppentreffen. Entsprechend empfehlenswert ist eine Bearbeitung der zugeteilten Aufgaben direkt nach einem Gruppentreffen.