University of Zurich[UZH]

s.e.a.l. software evolution & architecture lab

Software Praktikum 2012, Übung 4

Incremental Development, Integration Testing

1 Informationen

1.1 Kontext

Sie sollten inzwischen eine spielbare Version von *Asara* implementiert und JUnit-Tests erstellt haben, welche die wichtigsten Funktionen abdecken. In der letzten Phase geht es nun darum letzte Implementierungsarbeiten abzuschliessen, das Zusammenspiel sämtlicher Komponenten ausgiebig zu testen und die Qualität des Codes zu verbessern. Sozusagen der Feinschliff für die Schlusspräsentation.

1.2 Bestandteile der Abgabe

Eine aktualisierte ausführbare JAR-Datei Ihrer *Asara* Implementierung. Die JAR-Datei soll folgenden Inhalt haben:

- Die ausführbaren Klassen.
- Der dokumentierte Quellcode.
- JUnit Tests.
- Dokument welches Ihr Vorgehen beim Integration Testing dokumentiert.
- Aktualisierte Dokumentation, die sich mit der eigentlichen Implementierung deckt.

1.3 Formales

Ihre Abgabe hat in Form einer **einzigen** JAR-Datei (oder ein Download-Link zu der JAR-Datei) per Email an Ihren Betreuer und als Kopie an **hert@ifi.uzh.ch** zu erfolgen. Jede Source-Code-Datei muss zu Beginn die Namen und Matrikelnummern aller Gruppenmitglieder enthalten. Das Spiel muss durch einfaches "Doppel-Klicken" auf die JAR-Datei funktionieren und spielbar sein. Die Abgabe hat **spätestens bis um Mitternacht vor der Schlusspräsentation** zu erfolgen. Diese Vorgaben **müssen** eingehalten werden!

1.4 Gruppen

Die Übungen sind ausschliesslich in den gebildeten Gruppen zu lösen. Im abgegebenen Dokument muss ersichtlich sein, wer welchen Beitrag geleistet hat. Ausserdem muss jedes Gruppenmitglied die Abgabe insgesamt verstanden haben und in der Lage sein darüber Auskunft zu geben.

2 Aufgabenstellung

2.1 Finishing the Implementation

Verpassen Sie Ihrer *Asara* Implementierung nun den letzten Schliff und stellen Sie sicher, dass sämtliche geplanten Features korrekt umgesetzt sind.

2.2 Integration Testing

Testen Sie das Zusammenspiel Ihrer Komponenten, indem Sie für jeden Ihrer Use-Cases einige konkrete Szenarien durchspielen. Protokollieren Sie diesen Vorgang in geeigneter Weise und beheben Sie allfällige Unzulänglichkeiten.

2.3 Cleaning-Up: Refactoring and Documentation

Erfahrungsgemäss wird Ihre Implementierung noch einige sog. *Code-Smells* enthalten. Bringen Sie ggf. in einem ersten Schritt die Dokumentation auf den neuesten Stand (entspricht die Implementierung dem Entwurf? Sind alle Klassen und deren Schnittstellen gemäss den Vorgaben dokumentiert?). Refaktorisieren Sie anschliessend problematische Stellen in Ihrem Programmcode unter Verwendung der in der IDE integrierten Refactoring-Werkzeuge. Beachten Sie ausserdem die Ergebnisse der Qualitäts-Analyse *Sonar*, eine *"rules compliance"* von mind. 75% sollte erreicht werden. In [1–3] finden Sie zusätzliche, nützliche Ansätze, wie Sie Ihre Implementierung verbessern können. Dank Ihren JUnit-Tests sollten Sie sicherstellen können, dass Ihre Verbesserungen keine neuen Fehler einführen.

References

[1] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*. Pearson Education, 1999.

[2] B. Meyer. *Object-Oriented Software Construction*. Prentice Hall, 1997.

[3] A. J. Riel. *Object-Oriented Design Heuristics*. Addison-Wesley, 1996.