5. Praktikumstermin: Systematisches Testen

Bisher wurde der Code "nur" mit sogenannten "statischen" Qualitätssicherungsmaßnahmen analysiert. Das heißt, der Code wurde dazu noch nicht ausgeführt. Heute werden dynamische Qualitätssicherungsmaßnahmen ergänzt. Das heißt, der Code wird analysiert, während er ausgeführt wird. Dazu setzen wir automatisierte Tests und Anweisungsüberdeckung ein.

5.1 Vorbereitung

- Die gewünschten Refactorings sind durchgeführt und Sie können die Maßnahmen erläutern.
- Es gibt keine rot oder gelb markierten Methoden mehr im CCCC-Report.
- Die vorgegebenen Bugs sind identifiziert und korrigiert. Jedes Teammitglied kann die Fehlerursache und die Korrektur erläutern

5.2 Aufgaben

5.2.1 Kurzpräsentation "Statische Code-Analyse"

Sie präsentieren kurz welche Probleme die statische Code Analyse mit CCCC in Ihrem Code entdeckt hat und was Sie dagegen getan haben. Welche Refactorings haben Sie durchgeführt?

5.2.2 Diskussion der (behobenen) Bugs

Sie erläutern die herausgefundene Ursache für die Fehler.

- Das System stürzt ab, wenn man den Cocktail "Planters Punch" mischt
- Der Cocktail "Martini James B" wird aussortiert, weil er angeblich nicht mischbar ist. Allerdings sind alle benötigten Zutaten vorhanden.
- Es sollen nur die benutzten Dosiereinheiten gereinigt werden.
- Das Gerät akzeptiert die Eingabe 0 ohne Fehlermeldung, stürzt dann aber ab.

Überlegen Sie was man hätte tun können, um diese Probleme in der Wartung leichter auffindbar zu machen. Wie könnte man diese Fehler in Zukunft durch einen Test erkennen?

5.2.3 Automatisierte Tests mit "Google Test" in Jenkins

"GoogleTest" (gTest) ist ein Test-Framework für Unit Tests. Damit können (unter anderem) einzelne Klassen automatisiert getestet werden. In Ihrer Jenkins-Umgebung wird GoogleTest bereits ausgeführt. Vielleicht haben Sie in Ihrer Jenkins-Oberfläche schon den Eintrag "Test Results Analyzer" und die Grafik "Trend der Testergebnisse" entdeckt?

- Betrachten Sie die Reports, die beim Ablauf der Tests von GoogleTest erzeugt werden

5.2.4 Automatisierte Tests mit "Google Test" in der lokalen IDE

Damit Sie selbst automatisierte Tests erstellen können, müssen Sie in Ihrer lokalen IDE eine Testkonfiguration erstellen. Diese darf nicht die normale interaktive main-Funktion kompilieren, sondern eine andere Mainfunktion, welche die Tests aufruft.

- Normalerweise gibt es in CLion bereits die Konfiguration "CocktailProTest". Falls nicht, legen Sie unter RUN->Edit_Configurations eine neue Konfiguration für GoogleTest an. Wählen Sie "CocktailProTest" als Target. Anschließend können Sie die Konfiguration in der Leiste über dem Hauptfenster aktivieren, wenn Sie die Tests durchführen wollen.
- Analysieren Sie den zugelieferten Code im Verzeichnis "Test" und versuchen Sie die Funktionsweise der Tests zu verstehen.

5.2.5 Erstellung von ersten eigenen Tests

Arbeiten Sie sich zusammen mit einem anderen Teammitglied in die Funktionsweise von GoogleTest ein. Dazu erstellt je ein 2-er Team die Tests für die Klasse Recipe bzw. RecipeStep.

- Legen Sie eine neue cpp-Datei mit der Testklasse <Klassenname>_Test im Ordner Test an um "Ihre" Klasse zu testen und stellen Sie diese Datei unter Versionsverwaltung.
  Wenn Sie CLion verwenden, selektieren Sie den Test-Ordner und wählen Sie im Rechte-Maus-Menu: New C/C++-Source File. Dann können Sie die neue Datei bequem in Ihre Test-Konfiguration und in das Git-Projekt übernehmen.
- Prüfen Sie zuerst, ob Ihr lokales Projekt die leere Testklasse korrekt einbindet und kompiliert. Anschließend schreiben Sie die eigentlichen Tests.
- Passen Sie Ihre Testklasse so an, dass Sie exemplarisch eine Methode testet (z.B. getNoOfRecipeSteps() bzw. getZutat()).
- Wenn die Tests lokal erfolgreich laufen, laden Sie die Testklasse und die geänderte Projektkonfiguration in das zentrale Repository.
- Prüfen Sie mit Jenkins, ob die Tests korrekt ausgeführt werden.

5.2.6 Erstellung von weiteren Tests

Erstellen Sie weitere Tests, um die Klassen nach bestem Wissen gründlich zu testen.

Achtung: Testen Sie aber noch nicht die Klassen CocktailPro und CocktailZubereiter!

- Legen Sie neue Testdateien mit Testklassen im Ordner Test an und stellen Sie diese Dateien unter Versionsverwaltung
- Passen Sie die Testklassen so an, dass sie für jede Methode der zu testenden Klassen mindestens (!) einen Test enthält. Das heißt, zu jeder Methode in der .h-Datei der zu testenden Klasse erstellen Sie in der Testklasse einen "TEST_F-Block", der die zu testende Methode aufruft und mit einem oder mehreren EXPECT_xx oder ASSERT_xx prüft, ob das Ergebnis so ist wie es sein sollte.
- Überlegen Sie, welche Fälle auftreten können und wie Sie durch Tests sicherstellen können, dass die Funktion korrekt funktioniert. Testen Sie nicht nur den Erfolgsfall der Methoden, sondern auch das Verhalten bei Problemen.

Hinweis:

- Die zugelieferten Testklassen demonstrieren unter anderem,
  - wie man eine Bildschirmausgabe testen kann, indem man den Standard-Stream für cout temporär umleitet
  - wie man eine Bildschirmeingabe testen kann, indem man den Standard-Stream für cin temporär umleitet
  - wie man private-Methoden testen kann[11]
  - wie man prüft, ob eine Exception geworfen wird
- Falls Sie trotz ernsthafter Bemühungen einen Teil des Codes nicht testen können, überlegen Sie, ob Sie durch eine Umstrukturierung des Codes die Testbarkeit erhöhen können ("Refactoring"). Dokumentieren Sie solche "Problemstellen" für die Diskussion im nächsten Praktikumstermin.

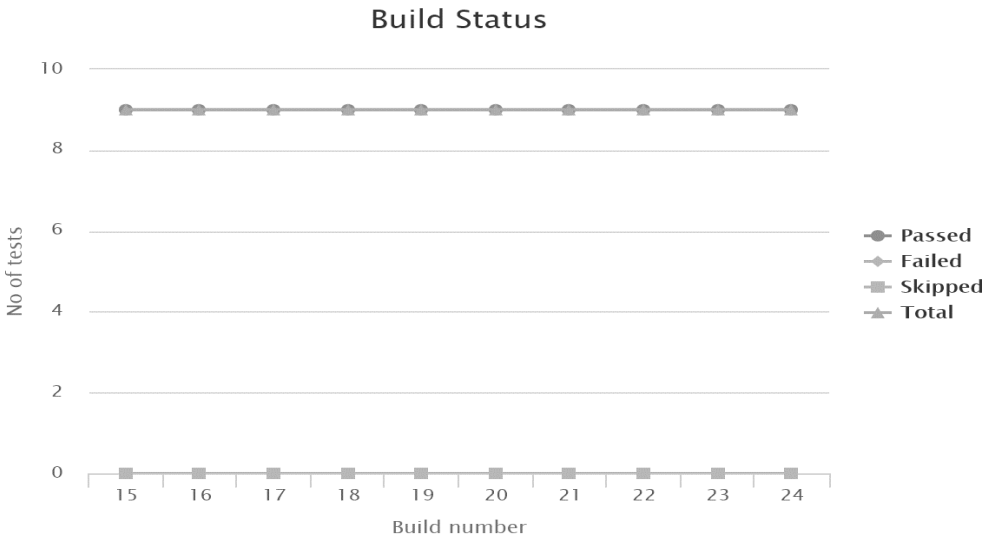
[11] Der Trick, über den Preprozessor private und protected durch public zu ersetzen, ist eine hässliche, aber schnelle Lösung. Man kann lange darüber philosophieren, ob man private-Methoden überhaupt testen sollte. Aber darum geht es hier nicht.

- Das Mischen von Cocktails dauert recht lange und nervt beim Testen. Haben Sie "schon" herausgefunden, wie die Methode set_Turbo() der Klasse Timer funktioniert? Verwenden Sie dieses Methode um Ihre Tests zu beschleunigen!

5.2.7 Testauswertung

Eine Testauswertung zeigt an, wie viele Tests für welche Klasse existieren und welche Tests erfolgreich verlaufen sind.

Chart	Package/Class/Testmethod	Passed	Transitions	24	23	22
☐	⊖ (root)	100% (100%)	0	0.001	0.001	0.000
☐	⊖ RecipeBookTest	100% (100%)	0	0.000	0.000	0.000
☐	deleteRecipeRemovesCorrectRecipe	100% (100%)	0	0.000	0.000	0.000
☐	getNumberOfRecipesReturnsValueOfAttribute	100% (100%)	0	0.000	0.000	0.000
☐	getRecipeReturnsCorrectRecipe	100% (100%)	0	0.000	0.000	0.000
☐	⊕ VorhandeneZutatenTest	100% (100%)	0	0.000	0.000	0.000
☐	⊕ gTestDemoTest	100% (100%)	0	0.001	0.001	0.000



- Wie viele Tests gibt es insgesamt?
- Wie viele Tests gibt es für die Klasse RecipeBook?
- Werden Ihre neu erstellten Tests aufgelistet?

5.3 Nachbereitung

Teilen Sie die zu testenden Klassen innerhalb Ihres Teams auf, um den Umgang mit Tests zu üben. Folgende Abnahmepunkte müssen bis zum Abgabetermin für den nächsten Praktikumstermin erfüllt sein:

- Es gibt für jede (testbare) Methode (außer für die Klassen CocktailPro und CocktailZubereiter) im Quellcode mindestens einen Test
- Die Laufzeit der Tests darf nur wenige Sekunden betragen. Aktivieren Sie den Turbo-Modus!
- Ein testbarer und lauffähiger Entwicklungsstand ist in Git eingecheckt. Prüfen Sie lokal, ob Ihr Cocktailautomat wirklich noch so funktioniert, wie es Ihr Auftraggeber wünscht.