4. Praktikumstermin: Metriken und Code-Qualität

Bei diesem Praktikumstermin soll die technische Infrastruktur erweitert werden, um systematisch und nachvollziehbar die Qualität des Codes zu verbessern. Dazu wird die bisher verwendete automatisierte Build-Umgebung, welche die bisher betrachteten Tools Doxygen und CppCheck integriert, um weitere Tools ergänzt.

4.1 Vorbereitung

- Ein lauffähiger und weiterhin vollständig kommentierter Entwicklungsstand ist in Git eingecheckt und Jenkins kann das Projekt bauen.
- Es gibt keine Compiler-Warnungen mehr.
- Die Doxygen-Dokumentation wird ohne Fehler erzeugt und enthält sinnvolle Kommentare. Der Inhalt ist sinnvoll und ansprechend.
- Alle Probleme, die CppCheck bemängelt hat, sind behoben. Sie haben dokumentiert, welche Maßnahmen Sie durchgeführt haben. Sie können zu jedem Befund erklären, was das Problem ist und warum Ihre Maßnahme das Problem beseitigt.

4.2 Aufgaben

4.2.1 Codequalität und Metriken

Jenkins hat unter anderem auch eine Codeanalyse mit "CCCC - C and C++ Code Counter" durchgeführt (http://cccc.sourceforge.net/). Suchen Sie den Report von CCCC in der Jenkins-Oberfläche.

Betrachten Sie den Bereich "Procedural Metrics Summary".

- Rechts sehen Sie die Spalte "L_C": Diese zeigt das Verhältnis von Codezeilen zu Kommentarzeilen. Wegen der vielen Doxygen-Kommentare dürften hier keine roten oder gelben Markierungen mehr auftreten. (Das war im ursprünglichen Projekt ganz anders.)
- Weiter links gibt es die Spalte "MVG": Darin wird für jede Klasse dargestellt, wie viele Verzweigungen die Methoden der Klasse beinhalten (vgl. McCabe Metrik / zyklomatische Zahl). Es ist allgemein bekannt, dass eine häufige Verwendung von Verzweigungen den Code schwerer verständlich macht. Scrollen Sie weiter runter (Procedural Metrics Detail), um die Werte für die einzelnen Methoden zu sehen. Schauen Sie sich die 5 Methoden mit den höchsten Einträgen für MVG an. Erkennen Sie einen Zusammenhang zwischen schwer verständlichem Code und dem MVG-Wert? Überlegen Sie, wie Sie die Anzahl der Verzweigungen reduzieren könnten.

4.2.2 Refactorings

Überarbeiten Sie den Quellcode in Ihrer IDE mit Refactorings. Die IDE hilft Ihnen dabei. Achten Sie darauf, dass der Code lauffähig ist, bevor Sie ihn wieder in das zentrale Repository hochladen.

Nutzen Sie einfache Refactorings (https://refactoring.com/), um Ihren Code weiter zu verbessern:

- Benennen Sie Variablen, Attribute und Methoden um, deren Namen nicht hinreichend aussagekräftig sind. Verwenden Sie dabei einheitlich eine CamelCase-Notation.
- Ersetzen Sie "Magic Numbers" im Code - also alle festen Zahlen außer der 0. Erzeugen Sie dazu entsprechende static Konstanten:
  Ersetzen Sie z.B. die `10` durch `SPEED_FACTOR` und definieren Sie die Klassenvariable

`static const int SPEED_FACTOR=10.`
Wenn es sinnvoll ist, ersetzen Sie bitte analog auch Strings durch entsprechende Konstanten.

- Reduzieren Sie die Anzahl der Verzweigungen in den Methoden aus der vorherigen Aufgabe, indem Sie Teile der Methode als neue (private?) Methode extrahieren. Dokumentieren Sie die vorgenommenen Änderungen so, dass Sie beim nächsten Praktikumstermin das Problem und Ihre Lösung präsentieren können. Denken Sie auch daran, dass neue Methoden gemäß der Doxygen-Regeln kommentiert werden müssen.

Wenn der Code läuft und die Anforderungen erfüllt, laden Sie das Ergebnis in das zentrale Repository und prüfen mit Jenkins, ob die Ergebnisse im CCCC-Report jetzt besser sind.

4.3 Nachbereitung

Teilen Sie die folgenden Aufgaben innerhalb Ihres Teams auf, um den Umgang mit Git und mit Jenkins zu üben. Folgende Abnahmepunkte müssen bis zum Abgabetermin für den nächsten Praktikumstermin erfüllt sein:

- Setzen Sie Refactoring ein, um alle von CCCC rot oder gelb markierten Methoden so zu verbessern, dass Sie eine akzeptable Qualität aufweisen.
- Setzen Sie die Refactorings in Ihrer IDE ein, um die Namen von Methoden und Attributen zu verbessern
  Tipp: Verwenden Sie regelmäßig die Pull-Funktion von Git, um sicherzustellen, dass Sie mit den aktuellen Dateien arbeiten und keine Konflikte mit ihren Teamkollegen entstehen
- Verwenden Sie wie üblich Tickets in GitLab für die Koordination der Tätigkeiten
- Korrigieren Sie bekannte Bugs im CocktailPro[10]:
  - Das System stürzt ab, wenn man den Cocktail "Planters Punch" mischt
  - Der Cocktail "Martini James B" wird nicht angeboten, obwohl alle benötigten Zutaten vorhanden sind.
  - Das Gerät reinigt nach dem Mischen alle Dosiereinheiten. Das verschwendet Reinigungsmittel und Zutaten. Es sollen nur die benutzten Dosiereinheiten gereinigt werden.
  - Das Gerät akzeptiert die Eingabe 0 ohne Fehlermeldung, stürzt dann aber ab.
- Randbedingungen:
  - Jedes Teammitglied hat in einem Branch mindestens zwei Refactorings vorgenommen, um die zyklomatische Zahl einer Methode zu reduzieren.
  - Bereiten Sie sich darauf vor, das Problem und Ihre Lösung zu präsentieren.
  - Die Bugs sind korrigiert und ein lauffähiger Entwicklungsstand ist in Git eingecheckt. Sie können (und sollten) dies mit Jenkins leicht überprüfen.
  - Die Reports in Jenkins (Doxygen, CppCheck, gcc-Warnings und CCCC) zeigen keine Befunde.

[10] Sie sollen die Fehler suchen und korrigieren und keine Workarounds basteln! Eine Änderung der Zutaten oder Rezepte ist keine gültige Lösung!