Bericht Software-Qualitätssicherung - Edgy

Projektziel

Das Ziel unseres Programmierprojektes ist es ein Multiplayerspiel namens Edgy mit Java zu programmieren. Das Spiel soll eine möglichst einfach gehaltene und für dieses Spiel optimierte Client-Server Struktur besitzen. Ein Spieler wird sich anfangs des Spiels mit seinem eigenen Benutzernamen und Passwort anmelden beziehungsweise registrieren können. Das Regelwerk soll so kompakt wie möglich implementiert werden. Die grafische Oberfläche wird in 2D gestaltet und beinhaltet zusätzlich eine aktuelle Rangliste sowie einen Overall-score der beteiligten Spieler. Damit wir diese Ziele möglichst effizient erreichen können werden wir einige Massnahmen anwenden.

Massnahmen

1. Wir haben *sieben* Massnahmen ergriffen, die die Qualität unseres Programmes sicherstellen. **Javadoc** macht unseren Code bestmöglich, anhand einer ausführlichen Dokumentation, für jeden verständlich und nachvollziehbar. Javadoc wurde wöchentlich aktualisiert und mit einem Vermerk «Javadoc» commited. Jedes Teammitglied ist angehalten seinen Code selbst zu kommentieren. Der QM-Manager generiert die JavaDocs vor jedem Gruppenmeeting am Montag und weist auf Verstösse hin.
2. Damit unser Code einheitlich gegliedert ist, werden wir alle im «Google style» programmieren und haben Richtlinien definiert wie wir unseren Code gestalten. Um dies regelmässig überprüfen zu können, haben wir in IntelliJ IDEA die Plugins Checkstyle sowie google-java-format Settings installiert, um alle kritischen Meldungen zu beheben. Dies wird wöchentlich überprüft und mit einem zugehörigen commit gepusht. Der QM-Manager überprüft den Code style und weist am Montag auf Verstösse hin.
3. Mit der Absicht die Lesbarkeit und unser Verständnis des Codes zu verbessern, wurden kontinuierlich Code-Reviews durchgeführt, dem Partner konstruktive Feedbacks geliefert und Code-Richtlinienverstösse im Plenum diskutiert. Diese werden jeweils mit einem zugehörigen Protokoll im File «ProtocolCodeReview.txt» im Quality Management Ordner dokumentiert und ins Repository gestellt. Zudem führt der Quality Manager jedes Wochenende einen Review des Codes durch und kommentiert Code Breaking Conventions, sowie unübersichtliche Code Abschnitte und ungenügend kommentierte Stellen. Diese Reviews werden mit «review» im Text kommentiert und mit demselben Vermerk ins Repository gestellt**.**
4. In unserem Code werden wir an allen nötigen Stellen ein Exception-Handling implementieren und somit ein gutes Feedback über den auftretenden Fehler erhalten. Damit wir in unserem Programm sehr spezifisch nach Bugs suchen können werden wir einen mehrstufigen Server-Logger verwenden**.** Die folgenden Log-Stufen wurden in unserem Code eingebaut: log.info(«msg»); printed Informationen, log.warn(«msg»); Warnungen, die den Verlauf des Programms nicht beeinflussen, log.error(«msg»); Fehler die nicht auftreten sollten, jedoch das Programm nicht stoppen und log.fatal(«msg»); fatale Fehler bei denen das Programm nicht weiterlaufen darf.
5. Zwecks eines möglichst gut lesbaren Codes werden wir regelmässig in Zweiergruppen Code schreiben, um so auch voneinander profitieren und lernen zu können. Ebenfalls im Quality Management Ordner werden sie in einem File «ProtocolPairprogramming.txt» protokolliert.
6. Ausgiebige Test-Routinen werden mit Hilfe der JUnit library vorgängig implementiert und unser Code anhand der Testvorgaben fortlaufend programmiert**.**

7. Damit Bugs oder nicht optimierte Codelines möglichst vermieden und reduziert werden können, wurde die externe Bibliothek Spotbugs verwendet.

Um die Einhaltung unserer Produktqualität zu gewährleisten wurden Verantwortlichkeiten und Deadlines im Projektplan definiert. Damit die Code Qualität für die Milestones erneut überprüft werden kann, wurde zudem eine Checkliste «Checklist.txt» erstellt. Diese wird vor der Milestone-Abgabe abgearbeitet und vom Quality-Manager mit einem «Milestone approved» commitet.

Report

Nach nun etwa drei Monaten und einigen Stunden Arbeit ist es an der Zeit ein Fazit bezüglich unseres Quality Managements (QM) zu ziehen. Im Allgemeinen wurde uns bewusst, dass das QM ein sehr wichtiger Bestandteil jedes Projektes ist und nur mit ausreichenden Massnahmen das erwünschte Ziel erreicht und teilweise sogar viel Zeit eingespart werden kann. In unserem Projekt wurden sieben QM Massnahmen verwendet und woraus wir hier nun ein kurzes Fazit ziehen.

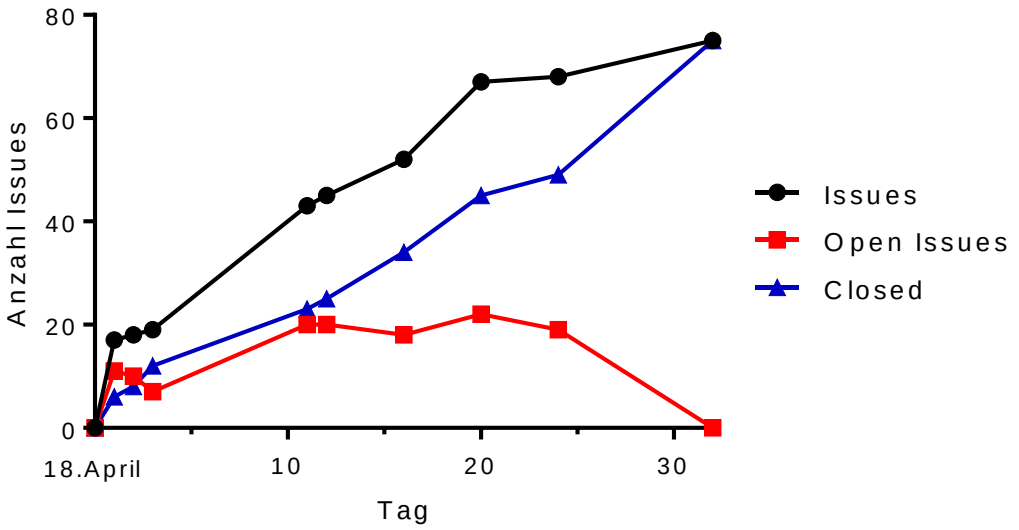

Abb. 1 Die Anzahl der Git Hub Bug Tracker issues nahm im Verlauf des Projekts zunehmend zu und konnte auch stetig abgearbeitet werden.

Anfänglich hier eine Statistik bezüglich unseres Issue Trackers auf Git. Sie veranschaulicht den kontinuierlichen Anstieg der Issues der ziemlich parallel mit den bearbeiteten Issues verläuft. Dies zeigt die Regelmässigkeit unserer geleisteten Arbeit. Pünktlich auf Milestone fünf konnten alle Issues, bis auf einige optionale, abgearbeitet werden. Das zeigt, dass der Projektplan gut ausgearbeitet und durchdacht sein musste.

Unsere erste Massnahme ist die im Code eingebaute Javadoc. Javadoc ist ausgezeichnet, wenn ein Aussenstehender einen Code verstehen will, für uns persönlich jedoch war die Javadoc nicht besonders hilfreich, da wir alle hauptsächlich in einem Teilgebiet des Codes tätig waren und somit ausserodentliche Kenntnisse über unseren Code hatten. In Meetings oder während Code Reviews wurde der Code zusätzlich besprochen, was das Gesamtverständnis des Codes beim ganzen Team

erhöhte. Bei auftauchenden Problemen konnte man sich meist einfacher untereinander helfen, ohne, dass die Javadoc von grossem Nutzen war.

Der Google style und besonders die installierten Plugins google-java-format und Checkstyle waren sehr hilfreich um den Code einheitlich zu formatieren. Dadurch konnte die Übersichtlichkeit unseres Codes verbessert werden. Durch das Plugin konnten auch fehlende Javadocs lokalisiert werden.

Die Code Reviews wurden eher selten durchgeführt, da aufgrund sehr regelmässiger Pairprogramming-Einheiten dies auch kaum mehr nötig war. Jedoch haben wir uns gegenseitig regelmässig unsauber programmierte Zeilen mit einem ToDo-Kommentar vermerkt, sodass der Urheber dadurch profitieren konnte.

In unserem Code haben wir häufig den Logger verwendet. Dieser stellte sich als sehr hilfreich heraus, besonders beim Lokalisieren von Bugs dadurch konnten einige Stunden an unproduktiver Zeit eingespart werden. Die Logging-Funktion erlaubte zudem die Erkennung von ineffezientem Versenden von Netzwerkpaketen und erlaubte die Erhöhung der Performance durch optimale zeitliche Abstimmung beim Versenden von Paketen.

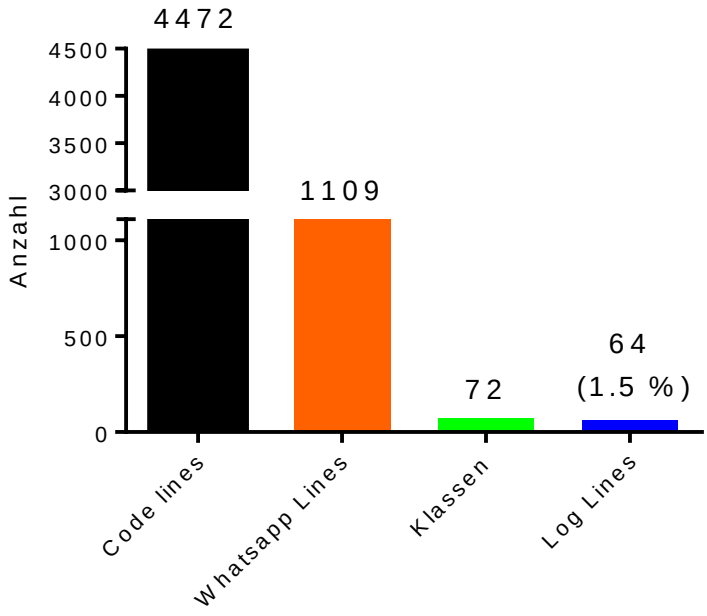

Abb. 2 Allgemeine Statistik zum Umfang von Edgy. Auf einen zeitlichen Verlauf wird aufgrund der Unübersichtlichkeit verzichtet.

Diese Statistik veranschaulicht die Anzahl Code Lines verglichen mit den geschriebenen Whatsapp Linien sowie den Klassen und den Log Zeilen. Sie zeigt die regelmässige Verwendung des Loggers auf und lässt auf eine gute Struktur des Programms hindeuten, da sehr viele Klassen erstellt wurden. Die vielen Messages lassen auf eine optimale Kommunikation zwischen den Teammitgliedern schliessen.

Wie wir feststellen mussten, ist Pairprogramming eine sehr nützliche Massnahme um Zeit einsparen zu können. Anfang des Projekts haben wir uns oft zu lange an einem Problem versucht, es jedoch dann gleichwohl nicht immer lösen können. Wir bemerkten, dass wir viel mehr als Team arbeiten und bei auftretenden Fragen sofort Hilfe bei den Teammitgliedern einholen mussten, denn zumeist konnte das Problem auf diese Weise wesentlich schneller behoben werden. Daraus entwickelten sich auch immer fortlaufend mehr Pairprogramming-Sitzungen.

Mithilfe der JUnit Library haben wir einen grossen Teil der Mechanik mittels Tests auf deren Funktionskorrektheit überprüft. Während dieser Testphase konnten mehrere Fehler in der Mechanik behoben werden. Jedoch war das Erstellen der Tests auch ein ziemlich langwieriger Prozess. Somit war JUnit zwar eine zeitaufwändige Massnahme, für die sich jedoch die investierte Zeit schlussendlich ausbezahlt hat.

Die externe Library Spotbugs ist eine sehr gute Möglichkeit Bugs im Code zu finden. Durch sie konnten wir diverse Fehler erkennen, die wir ansonsten wahrscheinlich übersehen hätten. Ebenfalls ermöglicht sie teilweise unsauber programmierte Zeilen und besonders falsche Zugriffe auf Variablen zu identifizieren. Somit war Spotbugs für unser Projekt überaus hilfreich.

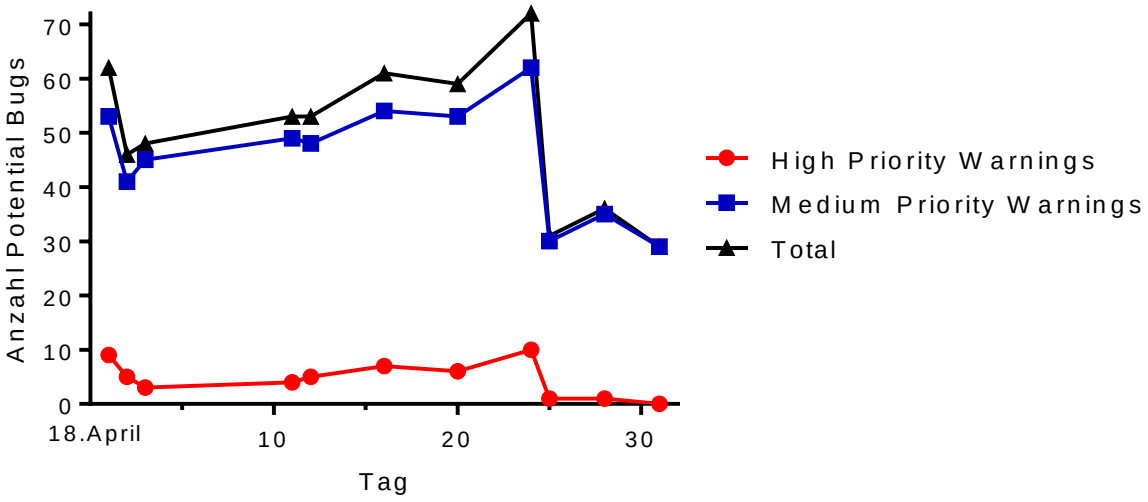

Abb. 3: Die Spotbugs-Warnungen haben im alltäglichen Programmierprozess zwar zugenommen, aber konsequentes Abarbeiten von Warnungen hat die Code-Qualität erhöht. Die verbleibenden Warnungen sind mehrheitlich irrelevant oder false-positives.

In dieser Grafik lassen sich die Bugs-Warnungen von Spotbugs ablesen. Sehr gut erkennen lassen sich die drei Milestones, wovor die Anzahl Bugs stets gesunken ist. Besonders auf den letzten Milstone konnten noch einige Bugs gelöst werden und sogar alle High Priority Warnings konnten behoben werden.

Unsere erstellte Milestone-Checkliste konnte sich ebenfalls bewähren, da wir durch sie stets noch einmal die Gewissheit erhielten nichts übersehen oder vergessen zu haben.

Fazit

Abschliessend bleibt zu sagen, dass das QM auf keinen Fall vernachlässigt werden darf, trotz der oft weniger interessanten Arbeit. Jedoch sollte man im Voraus entscheiden welche Massnahmen sich für das jeweilige Projekt als am sinnvollsten herausstellen könnten. Dies verlangt allerdings im Voraus eine klare Zieldefinition, sowie einen, alsbald möglich, ausgereiften Projektplan.

Der Quality-Manager

Tobias