Softwareengineering Projekt, 2016

Hashi

Projektentwicklung mit Scrum

Teammitglieder:
Ghenzi Gabriel
Hüppi Joel
Mazenauer Philippe
Messmer Martin
Mueller Chris
Schmid Christian

6.6.2016

INHALTSVERZEICHNIS

ÜBERBLICK

AUFGABE

Das Logikrätsel Hashi soll in einer Gruppenarbeit implementiert werden. Der Ablauf des Projekts erfolgt nach dem Prozessmodell Scrum.

PROZESSMODELL SCRUM

Bei Scrum werden zuerst die Projektrollen (Scrum Master, Product Owner, Scrum Team) definiert.
Der Product Owner definiert die Produktanforderungen in Form von User Stories, priorisiert diese und legt sie im Product Backlog ab. In einer Sitzung, dem Sprint Planning Meeting, werden die nächsten in Angriff genommenen User Stories in den Sprint Backlog übernommen und während dem Sprint vom Team abgearbeitet. Der Product Backlog wird ständig aktuell gehalten und erweitert.
Am Ende eines Sprints wird ein Sprint Review durchgeführt. Hier entscheidet das Scrum-Team was im Sprint erreicht wurde und was als abgeschlossen gilt. Anhand des Sprint Reviews wird der Product Backlog überarbeitet und neu priorisiert. Dieser Vorgang wird dauernd wiederholt, bis man am Schluss das fertige Produkt in den Händen hält.
Der ganze Ablauf wird während der Entwicklung stetig vom Scrum-Master kontrolliert.

SCRUM ROLLEN

Wir haben die Scrum Rollen wie folgt aufgeteilt:

Rolle	Name
Scrum Master	Chris Müller
Product Owner	Joel Hüppi
Team (Entwickler)	Gabriel Ghenzi
	Joel Hüppi
	Philippe Mazenauer
	Martin Messmer
	Christian Schmid

ENTWICKLUNGSSCHRITTE

USER STORIES

Eine User Story beschreibt Anforderungen und Eigenschaften, welche ein Kunde an sein Produkt stellt. Mit Hilfe der User Stories werden die im Product Backlog verwendeten Einträge formuliert.
Die User Stories werden vom Product Owner festgelegt. Wird eine User Story umgeschrieben oder ergänzt, werden diese Änderungen auch Product Backlog festgehalten.

PRODUCT BACKLOG

Anforderungen und Funktionalitäten welche in einem Produkt enthalten sein können, werden in einer geordneten Liste, dem Product Backlog abgelegt. Die Einträge werden Anfangs aus den User Stories abgeleitet und hinsichtlich ihrer Priorität bewertet. Der Product Backlog ist niemals vollständig. Er entwickelt sich ständig weiter.

StoryID	ItemID	Backlog Item	Category	Priority (1-5) 1: low, 5: high	Sprint	Status	Datum erledigt	Verantwortlicher
12	1	Scrum-Rollen verteilen	Planung	5	1	erledigt	19.04.2016	Team
12	2	Userstories definieren	Planung	5	1	erledigt	19.04.2016	Team
12	3	GitHub einrichten	Programmierung	4	1	erledigt	19.04.2016	Mazenauer
12	4	Domänenmodell erstellen	Planung	4	2	erledigt	26.04.2016	Team
12	5	Aus Domänenmodell grobes Klassendiagramm erstell	Planung	4	2	erledigt	26.04.2016	Team

Abbildung 1: Ausschnitt aus dem Product Backlog

SPRINT BACKLOG

Vom Entwicklungsteam werden Einträge aus dem Product Backlog ausgewählt, welche im nächsten Sprint umgesetzt werden sollen. Diese werden im Sprint Backlog abgelegt.
Die Sprintdauer haben wir am Projektanfang auf eine Woche festgelegt, da wir zur Umsetzung des Projektes nur ca. 2 Monate zur Verfügung hatten. Dabei kam es aber einige Male vor, dass gewisse Einträge/Funktionalitäten im Sprint Backlog nach einer Woche nicht fertig implementiert werden konnten, weil sie einfach zu aufwendig waren. Diese Einträge wurden in den nächsten Sprint übertragen. Die kurze Sprintdauer hatte allerdings auch einen positiven Aspekt, wir hatten jederzeit einen guten Überblick, wie der Entwicklungsstand des Projektes war.

Hashi: Sprint Backlog Woche 01

TeamSG: Philippe Mazenauer, Martin Messmer, Christian Schmid, Ghenzi, Joel Hüppi, Chris Müller

Datum: 19. - 25. April 2016

StoryID	ItemID	Backlog Item	Category	Verantwortlicher
11	1	Scrum-Rollen verteilen	Planung	Team
11	2	Userstories definieren	Planung	Team
11	3	GitHub einrichten.	Programmierung	Mazenauer

Abbildung 2: Beispiel Sprint Backlog Woche 1

SOFTWARE

KLASSENDIAGRAMME

PACKAGE-STRUKTUR

Die ganze Software wurde unterteilt in drei Packages.

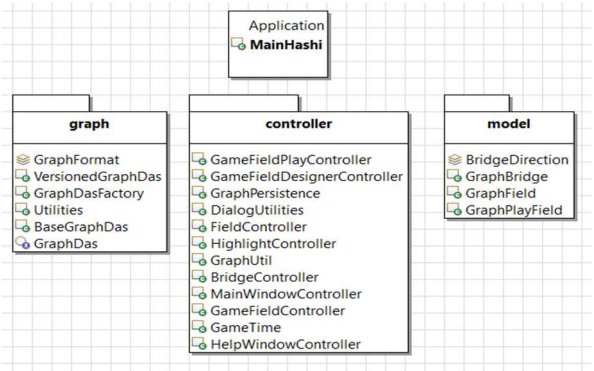

Abbildung 3 Package-Struktur Software

PACKAGE MODEL

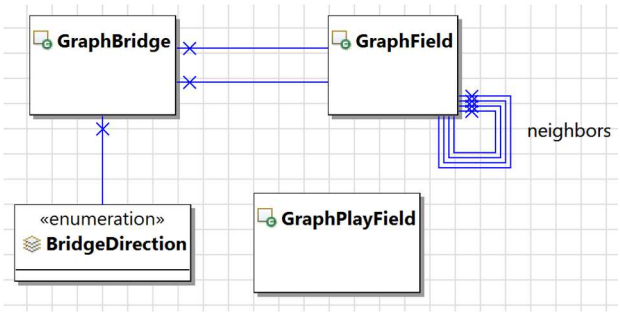

Abbildung 4 Package Model

PACKAGE GRAPH

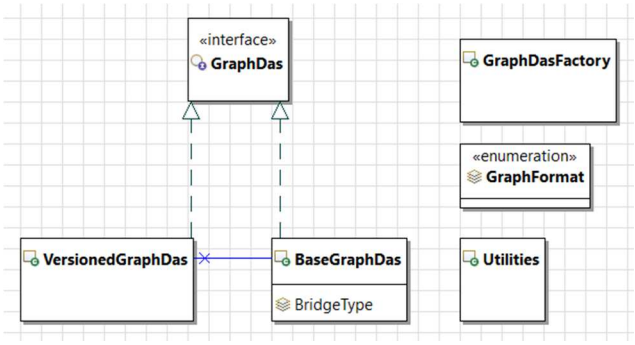

Abbildung 5 Package Graph

PACKAGE CONTROLLER

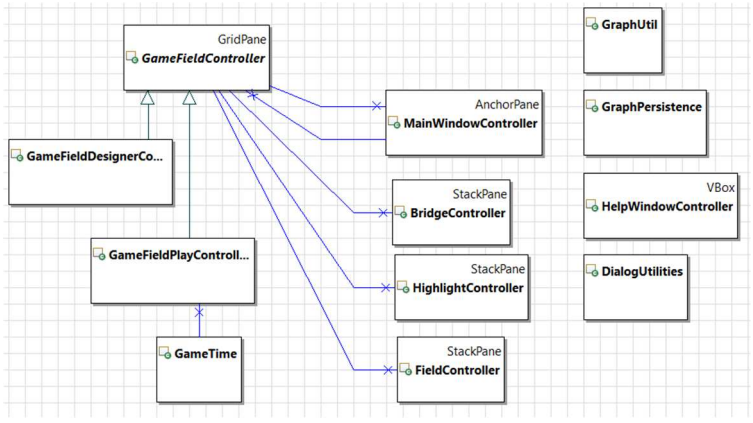

Abbildung 6 Package Controller

VERWENDETE PATTERNS

MVC (MODEL-VIEW-CONTROLLER)

Das MVC-Pattern trennt das Game in die Komponenten Model (Verarbeitung/Datenhaltung), View (Anzeige Spielfeld) und Controller (Eingaben im Spielfeld). Dies repräsentiert sich auch im strukturellen Aufbau der Packages im Projekt: Controller und Model. Für die Anzeige, das View im MVC, haben wir JavaFX benutzt. Die Oberfläche kann als Java-Klasse programmiert werden, jedoch ist es als FXML-File gespeichert und einiges einfacher, siehe

FXML. FXML-Files werden nicht in Packages zugeordnet, weshalb auch ein entsprechendes Package fehlt. Alle Views sind jedoch in einem eigenen Ordner abgelegt.

DECORATOR

Das Decorator-Pattern wird für die Undo/Redo Funktion verwendet. Die Klasse «VersionedGraphDas» erweitert die Klasse «BaseGraphDas» um die Undo/Redo Funktionalität.

COMMAND

Die Klassen «AddBridgeOperation», «RemoveBridgeOperation», «AddSolutionBridgeOperation» und «RemoveSolutionBridgeOperation» sind Kommandos und implementieren jeweils eine undo() und redo() Methode. Die ausgeführten Kommandos werden dann in der Klasse «VersionedGraphDas» in zwei separaten Stacks gespeichert. Ein Stack beinhaltet alle Undo-Kommandos, der Andere alle Redo-Kommandos. Wird nach dem undo eine andere Brücke gezeichnet, kann einfach der redo-Stack geleert werden. Ob undo und redo Funktionen überhaupt möglich sind, lässt sich einfach durch Abfrage der isEmpty()-Funktion klären.

FACTORY

Für die Klasse «GraphDas» benötigten wir eine Factory «GraphDasFactory». Diese gibt beim Aufruf der Methode «getGraphDas» eine Instanz von GraphDas, sofern eine existiert, zurück. Gibt es noch keine Instanz, wird eine erstellt. Dies stellt sicher, dass nur eine GraphDas-Instanz existiert, die das Datensystem des aktuellen Spielfelds repräsentiert. Zudem kann über die Factory auch die aktuelle Instanz beendet und eine Neue erzeugt werden. Diese Funktion wird beim Erstellen einer neuen Spielvorlage gebraucht.

TEMPLATE

Die abstrakte Klasse «GameFieldController» hat zwei abstrakte Methoden «removeBridge» und «addBridge». Diese zwei Methoden werden von den erbenden Klassen «GameFieldPlayController» und «GameFieldDesignerController» überschrieben. Dadurch ist ein dynamisches Wechseln der Implementation der beiden Methoden möglich.

SPIEL SPEICHERN/LADEN

Das ganze Back-End des Spiels basiert auf neo4j, einer Graphischen Datenbank. Diese Bibliothek umfasst alle notwendigen Tools, um unsere Spieldaten zu verwalten.

Die Funktionen Speichern und Laden sind auch integriert. Hiermit kann die gesamte Datenbank im XML oder JSON Format abgespeichert oder wieder geladen werden.

METRIKEN UND TESTS

TESTS

Zum Testen der Software haben wir das JUnit-Testframework benutzt. Dabei werden Methoden mit vordefinierten Parametern aufgerufen und die Rückgabewerte auf Richtigkeit überprüft. Aus Zeitgründen haben wir nur die für die Funktion des Spieles wichtigen Komponenten überprüft. Zudem stellte sich die Prüfung des GUI's als Herausforderung dar. Diese Aspekte spiegeln sich auch im Code-Coverage Test wieder: Beim Testlauf werden die Controller-Klassen sehr viel schlechter abgedeckt als die Model-Klassen.

Beim Testen der Software sind keine gravierenden Fehler aufgetreten. Allerdings konnten wir mit Hilfe der Tests den Code besser strukturieren und teils nicht benötigte Methoden und Klassen aufräumen.

METRIKEN

Die Metriken haben wir mit dem Software-Tool «Borland Together» überprüft. Der erste Testlauf sah schon ziemlich akzeptabel aus. Einzig die Werte CBO (Coupling Between Objects) und DOIH (Depth of Inheritance Hierarchy) waren im roten Bereich.

Metrics Test on 4th June 2016

Resource	Attribute Inheritance Factor (AIF)	Coupling Between Objects (CBO)	Cyclomatic Complexity (CC)	Comment Ratio (CR)	Depth Of Inheritance Hierarchy (DOIH)	Lack Of Cohesion Of Methods 3 (LCOM3)	Lines Of Code (LOC)	Method Inheritance Factor (MIF)	Number Of Child Classes (NOCC)
hashi	33	33	12	0	7	100	1824	95	2

Abbildung 7 Ausschnitt aus metricTests/metricTest-04062016.html

Aufgrund dieser zwei Werte haben wir die Klassen nochmals ein wenig umstrukturiert. Allerdings mussten wir feststellen, dass sich nicht grossartig was geändert hatte. Die hohen Werte konnten wir aber akzeptieren, da sie aus folgenden Gründen auftreten: Der hohe CBO Wert kommt von der Klasse «MainWindowController» die als zentraler Punkt im Programm fungiert. Sie erbt von der JavaFX-Klasse «AnchorPane», welche für diese hohe Kopplung verantwortlich ist. Der hohe DOIH Wert kommt auch von den Controller Klassen, welche jeweils von JavaFX Elementen mit hoher Hierarchie erben. Das Schlussresultat sah wie folgt aus:

Metrics Test on 6th June 2016

Resource	Attribute Inheritance Factor (AIF)	Coupling Between Objects (CBO)	Cyclomatic Complexity (CC)	Comment Ratio (CR)	Depth Of Inheritance Hierarchy (DOIH)	Lack Of Cohesion Of Methods 3 (LCOM3)	Lines Of Code (LOC)	Method Inheritance Factor (MIF)	Number Of Child Classes (NOCC)
hashi	35	38	15	0	7	100	2172	95	2

Abbildung 8 Ausschnitt aus metricTests/metricTest-06062016.html

Hier sehen wir, dass das CBO noch schlimmer wurde. Wir haben in diesen beiden Tagen jedoch auch ca. 350 Zeilen mehr Code generiert (siehe LOC). Das CBO wurde schlimmer, weil wir aus Performance gründen einige JavaFX Code-Zeilen aus dem XML File direkt in die Klassen genommen haben.

ERFAHRUNGEN

SCRUM

Aller Anfang ist schwer. Das haben wir auch bei der Abwicklung des Hashi-Projektes mit dem Scrum-Prozessmodell erfahren. Was sich in der Theorie simple anhörte, war bei der Umsetzung zum Teil schwierig. Trotzdem konnten wir einige Eindrücke und Erfahrungen sammeln, welche uns im späteren Berufsleben vielleicht wieder weiterhelfen können.

Obwohl man bei der Realisierung eines Projektes mit Scrum relativ frei ist, war eine grobe Grundplanung unverzichtbar. Nachdem die User Stories definiert wurden, sassen wir zusammen um die Grundstruktur der Software zu planen. Wir erstellten ein Domänenmodell und leiteten daraus ein Klassendiagramm ab. Dadurch war der Aufbau der Software von Anfang an klar. Gewisse Unklarheiten und Meinungsverschiedenheiten wurden somit aus dem Weg geräumt. Nach mehreren Diskussionen, wie wir die ganze Spiellogik programmieren möchten, ging es flott zur Sache. Das Entwicklerteam verstand sich super und arbeitete effizient zusammen. Jeder konnte seine Stärken und Fähigkeiten gezielt einsetzen, was auch den Lerneffekt unter den Teammitgliedern förderte. Hier sehen wir einen klaren Vorteil von Scrum.
Etwas unklar war die Erstellung des Product und Sprint Backlogs. Wir wussten nicht genau, wie exakt die verschiedenen Items ausgearbeitet werden müssen, somit waren sie anfangs recht schwammig definiert. Dadurch konnten die ersten Sprints nicht ganz fertig abgearbeitet werden. Diese Items haben wir dann feiner unterteilt. Der Product Backlog hat sich dadurch stetig vergrössert.
Wenn man Scrum konsequent anwenden würde, hätte man jeden Tag das sogenannte «Daily Scrum Meeting», in dem das Entwicklerteam seine Aktivitäten synchronisiert und an der Planung für die nächsten 24 Stunden arbeitet. Da wir nicht täglich an unserem Projekt arbeiteten und unser Team aus Vollzeit und Teilzeit Studenten bestand, führten wir nicht täglich ein Meeting durch.

Rückblickend war die Projektabwicklung mit Scrum sehr interessant. Wir denken, dass die Projektentwicklung mit Scrum noch verbessert werden kann, wenn man intensiv geschult wird und das erlernte auch tagtäglich im Berufsleben anwenden kann.

GIT

Für unser Projekt verwendeten wir GIT als verteiltes Versionskontrollsystem. Als Server benutzten wir github.com, welches für opensource Projekte gratis zur Verfügung steht. Wir haben festgestellt, dass das System sehr gut funktioniert. Auf die Webseite kann einfach mit dem Browser zugegriffen werden. Den Quellcode kann man daher auch ansehen, ohne Ihn auschecken zu müssen. Das ein- und auschecken ist einfach und erfüllt seinen Zweck. Allerdings muss auch hier beachtet werden, dass das Mergen (Zusammenführen zweier verschiedener Dateiversionen) unpraktisch ist und zu Fehlern führt. Deshalb haben wir darauf geachtet, dass nicht zwei Personen gleichzeitig an einer Datei arbeiten. Github stellt noch viele Zusatzfunktionen zur Verfügung, wie z.B. Issues verwalten mit denen wir unsere Sprints anfänglich definiert haben. Zusätzlich sind einige Grafische Darstellungen der Entwicklung des Programmes möglich sowie ein Wiki, welches sich sehr gut für die Dokumentation eignen würde. Aufgrund des Zeitdruckes konnten wird jedoch letzteres nicht nutzen, da uns die Zeit für die Einarbeitung fehlte.

FXML

JavaFX kann wie schon erwähnt als normaler Java-Code ähnlich wie Swing programmiert werden oder als FXML. FXML ist eine in XML aufgebaute Oberflächenbeschreibung. Die Oberfläche wird ähnlich wie HTML beschrieben, ohne die Funktionalität zu implementieren. Dies erlaubt eine komplette Trennung der Oberfläche von der Logik welche sich dahinter befindet. Mit dem sogenannten ScreenBuilder können Oberflächen einfach und mit wenigen Klicks erstellt werden und das ganze FXML-File wird im Hintergrund automatisch erstellt. Somit sind keine FXML-Kenntnisse vorausgesetzt.

Leider haben wir festgestellt, dass beim Einlesen der FXML Datei, welche die Oberfläche beschreibt, starke Geschwindigkeitseinbussen auftreten. Nach jedem Schritt wird die Oberfläche neu gezeichnet, damit verbunden ist das Laden aller Brücken, Felder und die Darstellung am Bildschirm. Daher haben wir den kritischen Teil in unseren Code eingebettet. Die Beschreibung erfolgt aber in der gleichen Art wie in der XML Datei.

Ein grosser Vorteil ist, das bei FXML die ganze Optik des Programmes mit einem eigenen Stylesheet realisiert werden kann. Dieses geschieht mit einem css-File welches aus der Webprogrammierung bekannt ist. Ein weiterer Vorteil ist die Portierung von JavaFX Oberflächen. So könnte das Programm einfach auf Android oder iOS portiert werden. Ebenfalls lassen sich JavaFX Programme auch ohne grossen Aufwand in einem Browser darstellen und somit auf einem Server betreiben. Das Spiel kann dann bequem vom User über den Browser gespielt werden.

TESTING

Wir haben den Aufwand, Testklassen zu schreiben stark unterschätzt. Zudem haben wir erst gegen den Schluss angefangen, die Tests zu implementieren. Aus Zeitdruck haben wir dann nur die wichtigen Funktionen getestet, und erreichen somit eine eher schlechte Code-Coverage.

VERGLEICH DER PROZESSE

Scrum bietet viele Vorteile wie einfache Regeln, wenige Rollen, Selbstorganisation und Eigenverantwortung in Teams und ist speziell geeignet für hochkomplexe Projekte mit unklaren Anforderungen. Die Projekte werden in mehreren Inkrementen entwickelt. Das macht Scrum sehr flexibel.
Im Vergleich zu anderen Prozessmodellen, wird bei Scrum auf einen Grossteil der Planung verzichtet. Es wurden keine Use-Cases, Systemsequenzdiagramme, Sequenzdiagramme, usw. erstellt. Das minimiert den Planungsaufwand enorm. Das könnte aber in grösseren Projekten auch zum Verhängnis werden.
Da unser Projekt relativ klein war, sahen wir es als ausreichend an, nur ein Klassendiagramm zu erstellen. Viele Funktionalitäten und Ideen zur Umsetzung der Software, folgten aus Diskussionen unter Teammitgliedern.
Abschliessend können wir sagen, dass die Projektabwicklung mit Scrum, bis auf einige Kleinigkeiten, erfolgreich war.

ABBILDUNGSVERZEICHNIS

LITERATURVERZEICHNIS

Scrum Guides. (2016). Von http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-DE.pdf#zoom=100 abgerufen

Scrum Master. (2016). Von http://scrum-master.de/Scrum-Einfuehrung abgerufen

Wikipedia. (2016). Von https://de.wikipedia.org/wiki/Scrum abgerufen

ANHANG

Die User Stories, der Product und Sprint Backlog sind im File Product_Sprint_Backlog_Hashi_TeamSG.pdf ersichtlich.