Alarm Clock: Arbeitsprozess

Jonathan Hyams
Pascal Schmalz

16. Juni 2017

Inhaltsverzeichnis

1 Zweck des Dokument

Dieses Dokument dient dem Leser zur Klarstellung, wie wir das Projekt angegangen sind, wie wir es geplant haben und wo wir Schwierigkeiten hatten.

1.1 Planung

Als erstes sind wir zusammengesessen und haben uns überlegt, was unser Projekt braucht, um einen anständigen AlarmClock zu werden. Dafür sind wir vor einer Wandtafel gestanden und haben Stichworte aufgeschrieben, die wir dann in den Code einbauen wollen. Darunter waren Stichwörter wie E-Mail, Datenbanken, Shellscripts und Cronjobs eingeflossen. Daraus haben wir dann ein UML designed, das aufzeigt, wie wir das Ganze aufbauen wollen (Abbildung 1).

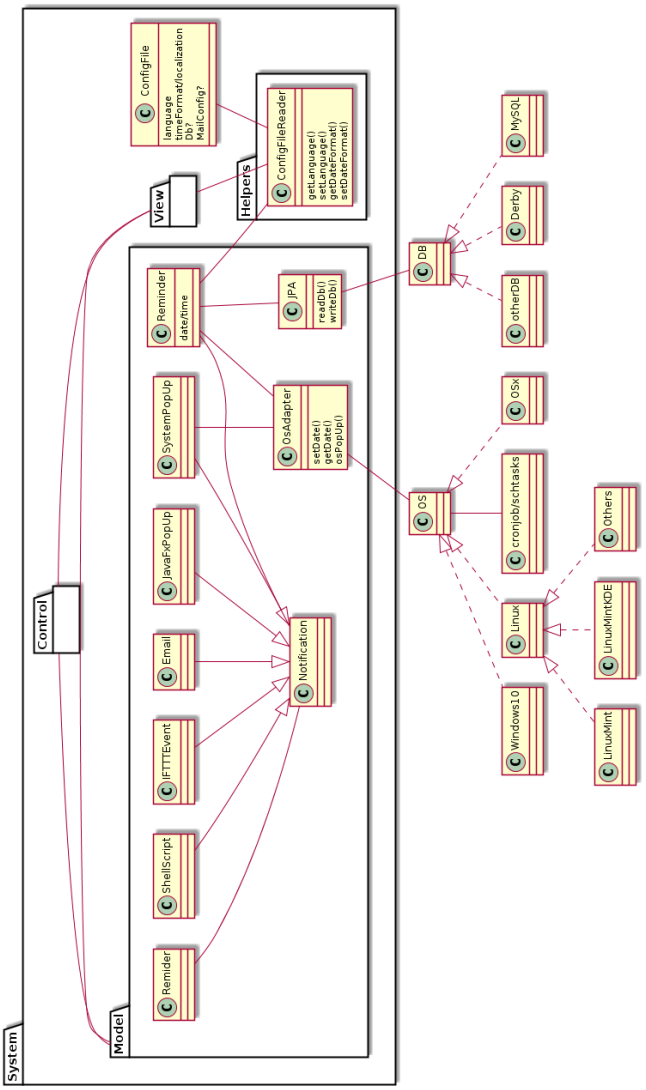


Abbildung 1: Systemübersicht Version 1

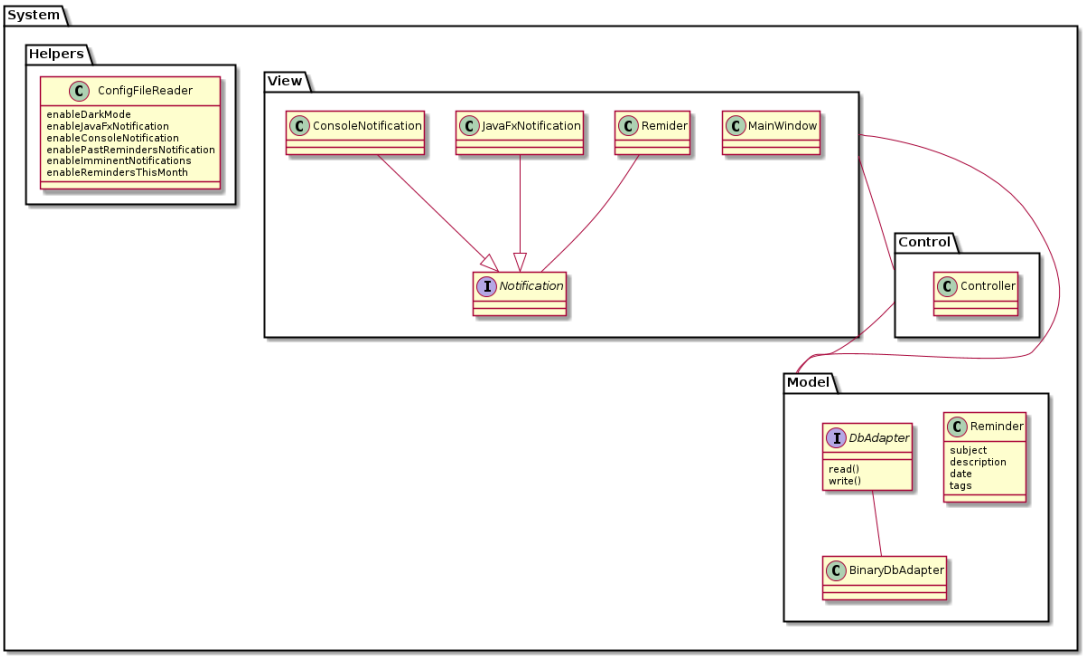


Abbildung 2: Systemübersicht Version 2

Dieses UML-Diagramm haben wir dann auch Herrn Prof. Fuhrer gezeigt, welcher es zwar eine gute Idee fand, aber sagte, dass wir das nie in der gegebenenen Zeit schaffen würden. Sachen wie Cronjobs, Shellscripts und verschiedene Datenbanken daran zu hängen wäre für uns unmöglich. Also haben wir nochmals von vorne angefangen und sind mit einem überarbeiteten UML vorangegangen (Abbildung 2).

Danach haben wir uns überlegt, was für Meilensteine wir uns setzen wollen:

1. Ein GUI mit einer Tabelle von Reminders
2. Abspeichern der Daten mittels einer Datenbank (wir haben uns für ein binäres Serialisieren entschieden)
3. Ein Poller, der immer wieder überprüft ob ein Reminder eingetroffen ist
4. Notifications, die den User über die Konsole zeigen, das ein Reminder eingetroffen ist
5. JavaFX Popups, die dem User zeigen, dass ein Reminder eingetroffen ist

Anschliessend haben wir begonnen, uns hinter das Codieren zu setzen. 2 Wochen vor der Abgabe des Codes haben mit der Dokumentation angefangen.

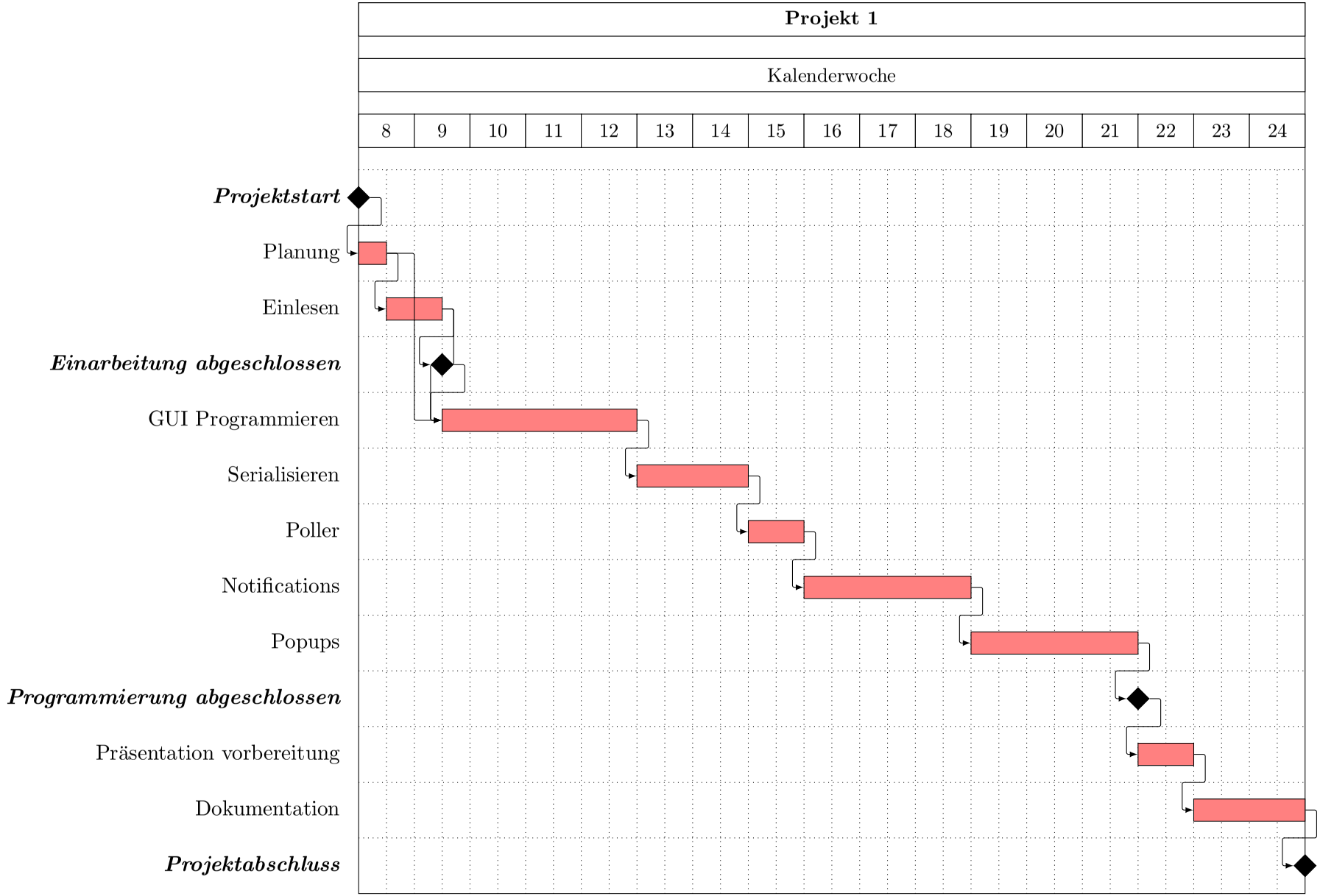
1.2 Deliverables

Von uns wurde verlangt, dass wir einen Betriebssystemunabhängigen Open Source Alarm Clock bauen, der wie kAlarm aufgebaut ist. Es sollte

- Wiederkehrende Events haben
- Persistenz
- Externes konfigurations File
- Datenbank unabhängig
- Betriebssystem unabhängig

Wiederkehrende Events und Externes Konfigurations File haben wir leider nicht mehr geschafft. Jedoch eine interne Konfigurations Klasse haben wir gebaut.

1.3 Gantt-Chart


Projekt 1
Kalenderwoche
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
Projektstart
Planung
Einlesen
Einarbeitung abgeschlossen
GUI Programmieren
Serialisieren
Poller
Notifications
Popups
Programmierung abgeschlossen
Präsentation vorbereitung
Dokumentation
Projektabschluss

Wir haben uns nicht im voraus überlegt was wie lange dauern wird. Wir haben uns einfach aufgelistet was im Code alles drin sein muss.

2 Meilensteine

2.1 GUI

Zuerst wollten wir ein brauchbares GUI als Interface haben, da es oftmals einfacher ist Ideen zu diskutieren, wenn man eine bildliche Diskussionsgrundlage hat. Unser Ziel war es das GUI als optischen Prototypen benutzen zu können.

2.2 Serialisieren

Der nächste Meilenstein war die Serialisierbarkeit. Wir wollten, dass wir die Reminders speichern können. Damit kann man sie später wieder von der Festplatte laden.

2.3 Poller

Mit dem Poller wollten wir sicherstellen, dass die Notifications auch abgesendet werden, wenn das Programm “geschlossen” wurde.

2.4 Notifications

Mit den Notifications wollten wir sehen, ob es funktioniert nach einer vom User bestimmten Zeit eine Notification auf der Konsole zu erhalten.

2.5 Popups

Danach wollten wir, wenn die Notification eintrifft, ein JavaFX Popup erhalten.

3 Stolpersteine

3.1 Gitignore

LaTeXgeneriert beim Kompilieren recht viele Dateien neben dem gewollten PDF. Diese Dateien wollten wir nicht über Git synchronisieren. Es wäre sonst möglich, dass die Dateien nicht zusammenpassen und beim erneuten Kompilieren zu einem Fehler führen. Wir wollten die Dateien also ins Gitignore file reinnehmen, in welchem man definieren kann, welche Dateien nicht mit Git verwaltet werden. Der Gitignore Syntax ist aber etwas speziell, man kann nicht einfach ein Verzeichnis ignorieren und dann mittels einer whitelist bestimmte Dateien wieder in die Versionierung mit einbeziehen. Will man ein solches Verhalten erreichen, darf man lediglich den Inhalt der Verzeichinsse ausklammern, und dann mittels Whitelisting wieder in die Versionskontrolle mit einbeziehen. Diesen Sachverhalt genau zu eruieren hat viel Zeit verschlungen. Als er einmal erkannt

wurde, mussten wir trotzdem noch genau darauf achten, dass wir nur genau die Dateien synchronisieren, welche wir auch synchronisieren wollten.

3.2 JavaFX serialisieren

JavaFX Komponenten lassen sich nicht Serialisieren. Bis wir das herausgefunden haben, hat es auch einige Stunden gedauert. Um dieses Problem zu umgehen, haben wir herausgefunden, dass, wenn man JavaFX Komponenten nicht direkt in der Klasse abspeichert, sondern sie nur zurückgibt, dann funktioniert es. Ein kurzes Beispiel: Die Table im GUI verlangt nicht Strings, sondern SimpleStringProperties. Das Problem ist aber, dass diese SimpleStringProperties nicht serialisierbar sind.

Falsch:

```
private SimpleStringProperty subject = new SimpleStringProperty
    ();

public SimpleStringProperty getSubjectProperty() {
  return subject;
}
```

Richtig:

```
private String subject = "";

public SimpleStringProperty getSubjectProperty() {
  return new SimpleStringProperty(subject);
}
```

3.3 Platform.runlater

Ein weiteres Problem das wir hatten, war das verspätete Erscheinen von Popups. Wenn man ein Popup z.B. eine Minute nach Programmstart aufruft, hat man effektiv mehrere Dutzend Exceptions bekommen. Der Grund war, dass Threads mit JavaFX sehr mühsam sind. Das erste Problem, das wir überhaupt hatten, war, die richtige Exception zu finden. Die IllegalStateException hat uns dann auf den richtigen Pfad gebracht. Die Exception lautete: ÏllegalStateException: Not an FX application thread; currentThread = Thread-4 [...]". Auf Stackoverflow haben wir dann gesehen, dass man um das neu erstellte GUI ein

```
Platform.runlater(() -> {...});
```

wirft. Das neue GUI erstellt man dann in den geschweiften Klammern. Dieses Problem zu beheben hat ca eine Woche gedauert, da wir praktisch jede Exception genau analysiert haben und selten wirklich etwas Brauchbares darauslesen konnten.

3.4 PopUps

Ein sehr merkwürdiges Problem war, dass ab und zu einige Popups einfach nicht erschienen sind. Zum debuggen haben wir ConsoleNotifications gebraucht, die gleichzeitig gelaunched wurden wie die Popups. Das Komische war, dass die ConsoleNotifications immer erschienen sind, jedoch die JavaFX Popups manchmal nicht. Nach langem Suchen hat sich herausgestellt, dass das erstellte GUI, welches im Platform.runlater(() -¿ ...); aufgerufen wird, nie ausgeführt wurde. Das heisst, jedes Stück Code darin wurde ignoriert, egal ob ein normales Print Statement oder das Erstellen eines GUI darin ist. Der Grund dazu war, dass sobald alle Stages, also alle JavaFX Fenster geschlossen sind, wird vom Compiler automatisch Platform.exit() aufgerufen. Dies führt dazu, dass alles was im Platform.runlater(() -¿ ...) ist, einfach ignoriert wird. Um dies zu umgehen, braucht man nur eine Zeile Code:

```
1   Platform.setImplicitExit(false);
```

Auch das herauszufinden hat sehr lange gedauert, da uns nie in den Sinn gekommen ist, dass so etwas überhaupt notwendig wäre. Wir haben mehr an unserem Code gezweifelt als an eine implizit aufgeworfene Methode von JavaFX.

3.5 DateTimePicker

Eines der grössten Probleme war, die vom User ausgewählte Zeit auszulesen und diese von einem String in ein DateTime Objekt umzuwandeln. Das Problem war, dass JavaFX zwar einen DatePicker zur Verfügung stellt, aber keinen DateTimePicker oder etwas ähnliches. Als erstes haben wir versucht, ein weiteres Textfield hinzuzufügen in welches der User eine Uhrzeit schreiben kann. Dieses wurde dann mittels eines Regex überprüft. Dies zum laufen zu bringen hat schon länger gedauert als wir gehofft haben. Wo wir dann stecken geblieben sind, war, als wir diesen String in ein DateTime Objekt umwandeln wollten. Dies wollte einfach nicht funktionieren. Beim Googeln, wie wir das machen können, haben wir zufällig das TornadoFX-Controls Plugin gefunden, welches eben einen DateTimePicker zur Verfügung stellt. Das hat dann unsere Probleme gelöst.

3.6 Maven

Ein kleineres Problem, aber dennoch ein Problem für uns war Maven. Unser Dozent wollte eine Möglichkeit, die das Launchen des Programms sehr einfach macht. Da wir bereits im Software Engineering and Design mit Maven gearbeitet haben, haben wir uns dafür entschieden. Es aufzusetzen gab bei uns aber immer wieder Probleme. Wir hatten zuerst einfach nur den Java Code im unserem Programm Folder ohne Maven. Um das Maven hinzuzufügen, mussten wir das Ganze auseinander nehmen, Maven aufsetzen und den Code wieder an den richtigen Ort tun. Jedoch hat dann die ganze Zeit der Code, der vorher noch erfolgreich kompiliert hat, nicht mehr funktioniert. Wir haben es dann noch zwei mal neu versucht aufzusetzen, danach hat es funktioniert. Leider wissen wir bis jetzt nicht, was wir anders gemacht haben.

4 Fazit

Wir haben unterschätzt, wie viele kleinere Probleme wir hatten und wie lange es dauert diese zu beheben. Vor allem das Arbeiten mit Threads in JavaFX hat sich als sehr unangenehm und mühsam ergeben. Wir haben dort mit den obigen Problemen sehr viel Zeit verloren. Wir haben dort auch immer wieder neue Ansätze probiert. Beispielsweise als das Serialisieren nicht funktionierte, haben wir uns angefangen in JPA einzulesen. Während dem Einlesen kam uns dann eine Idee, wie wir das Serialisierungsproblem lösen konnten. Bei den JavaFX Problemen gab es dann leider keine anderen Möglichkeiten diese zu umgehen, also mussten wir solange damit kämpfen, bis wir es dann endlich zum Laufen gebracht haben. Da wir an solchen kleineren Problemen so viel Zeit verloren haben, hat es nicht mehr gereicht, wiederkehrende Events einzubauen oder ein schönes Konfigurationsfile zu entwickeln. Trotzdem würden wir das Projekt jedenfalls teilweise als Erfolg betrachten, da wir nun einen funktionierenden, wenn auch nicht perfekten Alarm Clock gebaut haben.

5 Eigenständigkeitserklärung

Ich/wir bestätige/n, dass ich/wir die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der im Literaturverzeichnis angegebenen Quellen und Hilfsmittel angefertigt habe/n. Sämtliche Textstellen, die nicht von mir/uns stammen, sind als Zitate gekennzeichnet und mit dem genauen Hinweis auf ihre Herkunft versehen.

Pascal Schmalz

Jonathan Hyams

6 Versionskontrolle

Manuelle Version: 1.0.1

Automatische Versionierung: Last compiled: Fri 16 Jun 20:07:57 CEST 2017
Git HEAD Version: 209