

Alarm Clock

Jonathan Hyams
Pascal Schmalz

16. Juni 2017

Inhaltsverzeichnis

1	Zweck des Dokument	3
1.1	Planung	3
1.2	Deliverables	3
1.3	Aufwandschätzung und Kontrolle	3
1.4	Überarbeitung des Planes	3
1.5	Gantt Chart	3
2	Meilensteine	3
2.1	GUI	3
2.2	Serialisieren	3
2.3	Poller	3
2.4	Popups	3
3	Tests	3
4	Stolpersteine	3
4.1	Gitignore	3
4.2	JavaFX serialisieren	4
4.3	Platform.runlater	4
4.4	PopUps	5
4.5	Maven	5
5	Versionskontrolle	5

1 Zweck des Dokument

1.1 Planung

1.2 Deliverables

1.3 Aufwandschätzung und Kontrolle

1.4 Überarbeitung des Planes

1.5 Gantt Chart

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 zu 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 Popups

Pascal

3 Tests

4 Stolpersteine

4.1 Gitignore

L^AT_EX generiert beim compilieren 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 compilieren 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 Verzeichnisse 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.

4.2 JavaFX serialisieren

JavaFX Komponenten lassen sich nicht serialisieren. Bis wir das herausgefunden haben, hat 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:

```
1 private SimpleStringProperty subject = new SimpleStringProperty  
    ();  
  
public SimpleStringProperty getSubjectProperty() {  
    return subject;  
5 }
```

Richtig:

```
1 private String subject = "";  
  
public SimpleStringProperty getSubjectProperty() {  
    return new SimpleStringProperty(subject);  
5 }
```

4.3 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: IllegalStateException: Not an FX application thread; currentThread = Thread-4 [...]. Auf Stackoverflow haben wir dann gesehen, dass man um das neu Erstellte GUI ein

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

wirft. Das neue GUI erstellt man dann in den geschweiften Klammern.

4.4 PopUps

4.5 Maven

5 Versionskontrolle

Manuelle Version: 1.0.1

Automatische Versionierung: Last compiled: Fri 16 Jun 10:27:03 CEST 2017

Git HEAD Version: 199