Entwicklerdokumentation

PUML

Inhaltsverzeichnis

I Prozess- und Implementationsvorgaben 2
I.1 Definition of Done 2
I.2 Coding Style 2
I.3 Zu nutzende Werkzeuge 3
II Was wird wie gemacht? 3
II.1 Eclipse 3
II.2 LaTeX 5
II.3 GIT 5
III Best practice 7
III.1 GIT 7

I. Prozess- und Implementationsvorgaben

I.1 Definition of Done

XXX

I.2 Coding Style

Bitte die Datei javaCodeStyle.xml im specification-Verzeichniss in Eclipse importieren und verwenden. Hierfür in Eclipse unter „Window->Preferences->Java->Code Style->Formatter“ auf Import klicken und die XML-Datei auswählen.
Ist der passende Coding Style eingestellt kann der Quellcode mit „STRG+SHIFT+F“ automatisch formatiert werden. Wird dies vor jedem Commit gemacht, ensteht ein einheitlicher Code-Style und die Änderungen können gut mit GIT überprüft werden.
Des weiteren empfiehlt es sich bei größeren oder stark geschachtelten Code-Abschnitten die Zugehörigkeit der Schließenden Klammer mit einem Kommentar zu Kennzeichnen.
Sonstige Konventionen:

- Variablen und Instanzen beginnen kleingeschrieben
- Klassen und Interfaces beginnen mit Großbuchstaben
- Besteht ein Namen aus mehreren zusammengesetzten Wörtern, beginnen alle weiteren Wörter mit Großbuchstaben (keine Unterstriche in Namen verwenden)
- Aussagekräftige Namen verwenden
- Alle Namen auf Englisch
- Die Kommentare auf Deutsch

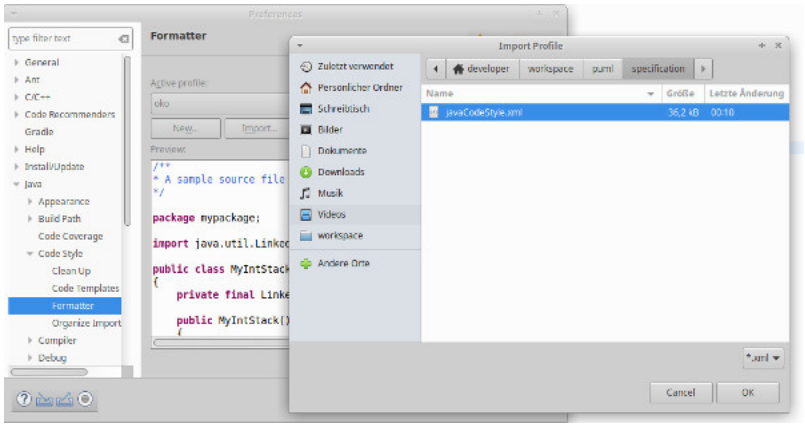


Abbildung 1: Code-Style in Eclipse importieren

- Lange Kommentare immer vor den Codeabschnitt
- Alle Methoden im Javadoc-Stiel dokumentieren

I.3 Zu nutzende Werkzeuge

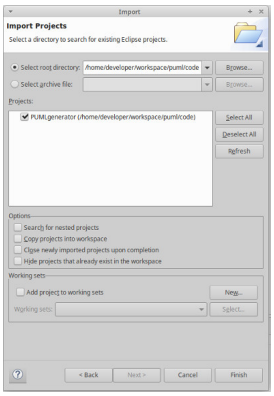
- Eclipse - Entwicklungsumgebung
- GIT - Dateiversionierung
- Meld - Unterschiede zwischen Dateien anzeigen
- Texmaker - Latex-Editor
- GIMP - Bildbearbeitung für das Editieren von Screenshots

II. Was wird wie gemacht?

II.1 Eclipse

II.1.1 Projekt in Eclipse importieren

In den workspace wecheln:
cd /workspace
Projekt Klonen:
git clone https://gitlab.imn.htwk-leipzig.de/weicker/puml.git
Benutzername und Passwort eingeben.
In Eclipse "File->Import->Existing Projects into Workspace"



Dann auf “Finish“ klicken.

II.1.2 WindowBuilder installieren

In Eclipse "Help->Install New Software..."
Unter work with “2018-09 - http://download.eclipse.org/releases/2018-09“ auswählen.
In der Section “General Purpose Tools“ die im Bild stehenden Häckchen anklicken
Dann auf “Finish“ und sich durch die Installation klicken.

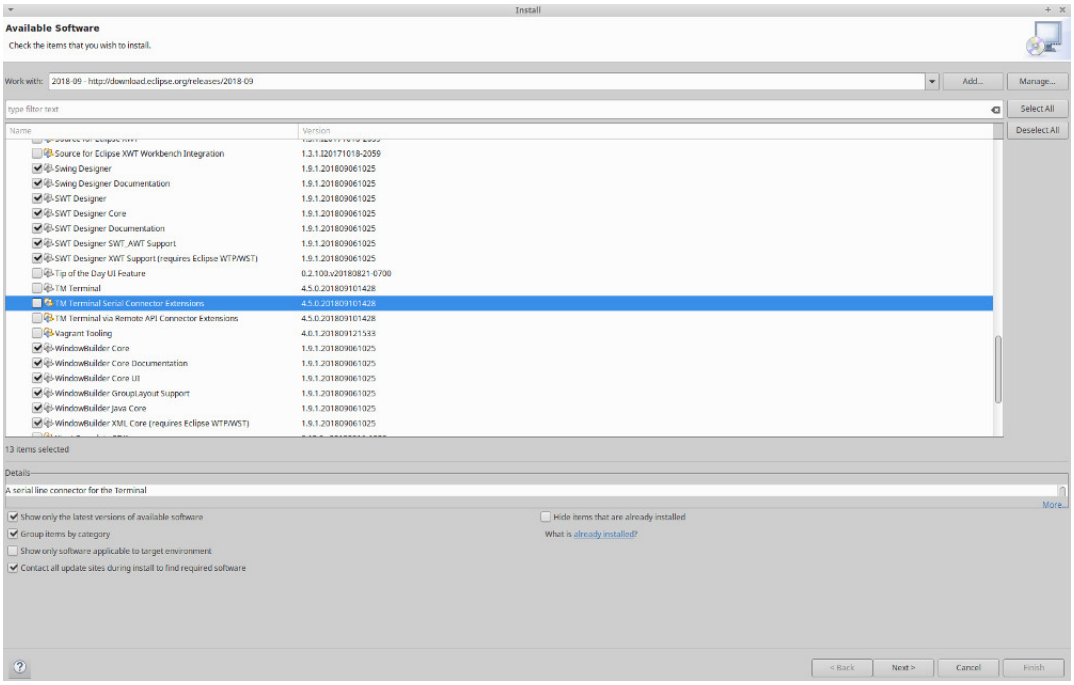


Abbildung 2: WindowBuilder installieren

II.1.3 GUI editieren

Es muss der WindowBuilder installiert sein. Dann auf die Datei die die Grafische Oberfläche implementiert (GUI.java) mit der rechten Maustaste klicken. Dann “Open With->WindowBuilder Editor“ auswählen.

II.2 LaTeX

II.2.1 Geschachtelte Überschriften

Durch die Makros:

- \nsecbegin{MeineÜberschrift}
- \nsecend

können geschachtelte Überschriften verwendet werden. Die Kapitel einfach in diese Makros einschließen. Somit muss nicht darauf geachtet werden auf welcher Ebene man sich im Moment befindet. Dies vereinfacht insbesondere das Auslagern von Text in andere Dateien.
Um die Makros in die Autovervollständigung des Texmakers aufzunehmen "Benutzer/in->Wortvervollständigung anpassen" wählen und dort die Makros hinzufügen.

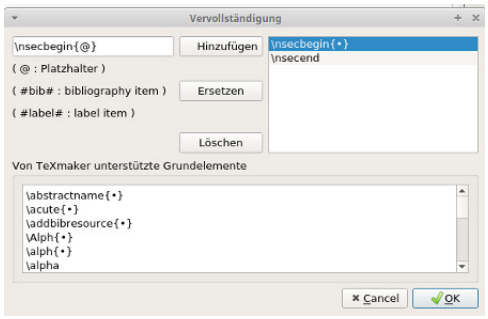


Abbildung 3: Autovervollständigung anpassen

II.2.2 Build-Dateien aufräumen

Beim erstellen des LaTeX-Dokuments werden jede Menge zusätzliche Dateien erstellt. Dank der entsprechenden ".gitignore-Datei" werden diese nicht in GIT hinzugefügt. Für den Fall dass man das Verzeichniss bei sich selbst bereinigen möchte, kann das "clean.sh"-Script ausgeführt werden.

II.2.3 Entwicklerdokumentation und Handbuch erstellen

Wenn etwas an der Entwicklerdokumentation oder am Handbuch geändert wurde, müssen diese Dokumente neu erstellt werden. Hierfür zunächst wie gewohnt die LaTeX-Projektdokumentation erstellen. Anschließend kann das "buildAllDocuments.sh"-Script ausgeführt werden. Dieses erstellt dann die entsprechenden Dokumente.
Weitere Informationen zum "multiaudience-Paket" unter https://www.uweziegenhagen.de/?p=3252.

II.3 GIT

II.3.1 Benutzername und eMail ins GIT eintragen

In Linux kann durch den Aufruf:
gedit ~/.gitconfig
die ".gitconfig-Datei" editiert werden. In dieser werden unter anderem auch Benutzername und eMail-Addressse des Benutzers gespeichert.

II.3.2 Basics

```
#Aktueller Zustand ausgeben
#Auf Welchem Branch bin ich?
#Gibt es Dateien die gendert sind?
#Wurden Dateien gelscht?
#Wurden neue Dateien hinzugefgt?
#Sind Aenderungen bereits fr den Commit vorgemerkt?
#Bin ich vor oder hinter dem Remote-branch?
git status

#Alle Aenderungen fr den Commit vormerken.
# Fr  den Punkt kann auch ein Pfad angegeben werden um bestimmte nderungen vorzumerken.
#Wenn die .gitignore-Datei richtig gepflegt wird, sollte immer die Variante mit dem Punkt
    verwendet werden knnen.
git add .

#Vorgemerkte nderungen Commiten
#Anschlieend muss die Commit-Nachricht im Editor eingetragen werden
git commit

#Alle commits auflisten
git log

#Unterschiede zwischen der aktuellen Version und einem lteren Commit anzeigen
git difftool hashDesCommits

#Unterschiede zwischen zwei aelteren commits anzeigen
git difftool hashDesErstenCommits hashDesZweitenCommits

#Zu einem lteren Commit wechseln
git checkout hashDesCommits

#Neuen Branch vom aktuellen Stand aus erstellen
git branch nameDesNeuenBranches

#Zu einem Branch wechseln
git checkout nameDesBraches
```

II.3.3 Mergen

```
#Einen anderen Branch in meinen aktuellen mergen
git merge nameDesBranches

#Bei Merge-Konflikt
git mergetool

#Fuer eine Datei direkt meine Version verwenden
git checkout --ours -- nameDerKonfliktdatei

#Fuer eine Datei die Remote-version verwenden
git checkout --theirs -- nameDerKonfliktdatei

#Nach dem mergen
git commit
```

II.3.4 Arbeiten mit dem Server

```
#Remote anzeigen
git remote -v

#Aktuelle Version eines Branches holen
git pull origin branchName

#Meine nderungen auf einen Branch hochladen
git push origin branchName
```

II.3.5 Eigenen Branch mit dem Master Synchronisieren

Wenn sich der Master wärend der Entwicklung am eigenen Branch weiter entwickelt hat, können die Änderungen des Master auf folgende weise in den eigenen Branch übernommen werden.

```
git status #Pruefen ob auf meinem Branch
#wenn nicht
git checkout myBranch
#Lokalen Master aktuallisieren
git pull origin master #sollte auch gleich in myBranch mergen
#wenn nicht
git merge master
#Wenn merge-konflikt
git mergetool
#Jetzt noch den Merge commiten
git commit
```

II.3.6 Lokale Branches aufräumen

ACHTUNG: Sollte nur gemacht werden, wenn alle Änderungen in den Master übernommen wurden und somit sicher sind!!

```
#Alle branches loeschen die es nicht mehr auf dem Server gibt
git remote prune origin
#Lokale branches die mit dem master gemerged wurden loeschen
git branch --merged master | grep -v '^[ *]*master$' | xargs git branch -d
```

III. Best practice

III.1 GIT

III.1.1 Keine nicht benötigten Dateien adden

Vor dem "git add ." immer mit "git status" prüfen welche Dateien hinzugefügt werden. Sollten nicht für das Projekt benötigte Dateien (z.B. übersetzte Binärdateien oder Dokumentation von Librarys) dabei sein, bitte die entsprechende ".gitignore-Datei" vervollständigen. Danach sollten die Dateien beim "git status" nicht mehr angezeigt und somit nicht mehr geadded werden.
HINWEIS: Die ".gitignore-Datei" ist (wie an dem führenden Punkt zu sehen ist) versteckt und wird nur nach dem setzen des entsprechenden Häckchens im Dateimanager oder beim "ls -a" angezeigt.

III.1.2 Branches nach aktuell zu bearbeitendem Thema benennen

Damit direkt aus dem Branchname ersichtbar ist was innerhalb des Branches bearbeitet wird, ist

es sinnvoll den Name entsprechend zu wählen (Z.B. GUIBranch). Da die Branches mit dem Gitlab-Server synchronisiert werden können, ist es auch ohne weiteres möglich dass mehrere Personen an einem Branch arbeiten.