



Softwaretechnik 2015/2016

PST Lehrstuhl

Prof. Dr. Matthias Hölzl

Joschka Rinke



- Übung 13:**
- 28.01.2016**
- **Fragen**
 - **Besprechung Blatt12**
 - **Klausur Tipps**

Aufgabe 1 – Verifizieren und Validieren



Was versteht man unter Validierung und was unter Verifizierung?

- **Validierung: Entspricht Programm den Wünschen**
- **Validierung: Entspricht Programm den Spezifikationen**
- **Verifizierung: Entspricht Programm den Wünschen**
- **Verifizierung: Entspricht Programm den Spezifikationen**

Validierung:

„Versuchen wir das Richtige zu machen?“ ...

...soll heißen, macht das Programm das, was es machen soll?

Entspricht das Programm den Wünschen des Kunden?

Verifikation:

„Machen wir das, was wir machen richtig?“ ...

...soll heißen, macht das Programm Fehler?

Entspricht das Programm den Spezifikationen?

Verifikations- & Validierungstechniken:

- **Statisch: Programm wird nicht ausgeführt**
 - Code Inspektion
 - formale Methoden
- **Dynamisch: Programm wird ausgeführt & Terminierungszustände oder Ergebnisse werden überprüft**
 - Tests
 - Fuzzing

(Validierung findet meist dynamisch statt)

Code Navigation:

Hilfreich für Code Inspektion

Code Navigation in Eclipse (kleine Auswahl):

Ctrl + **Shift** + **R** : open any resource

Ctrl + **E** : switch between open tabs; **Ctrl** + **Bild < >** : go to left/right tab

Ctrl + **T** : open type hierarchy of package

Ctrl + **O** : view all methods, press again to view all superclass methods, too

Ctrl + **1** : quick fix

Ctrl + **Shift** + **O** : organize imports

Ctrl + **L** : go to line

Ctrl + **Shift** + **F** : auto format

Ctrl + **Alt** + **H** : open call hierarchy

Alt + **Shift** + **R** : refactor

Hold **Ctrl** while clicking on a class/method/interface to open it

Ctrl + **Q** : go to last edited place

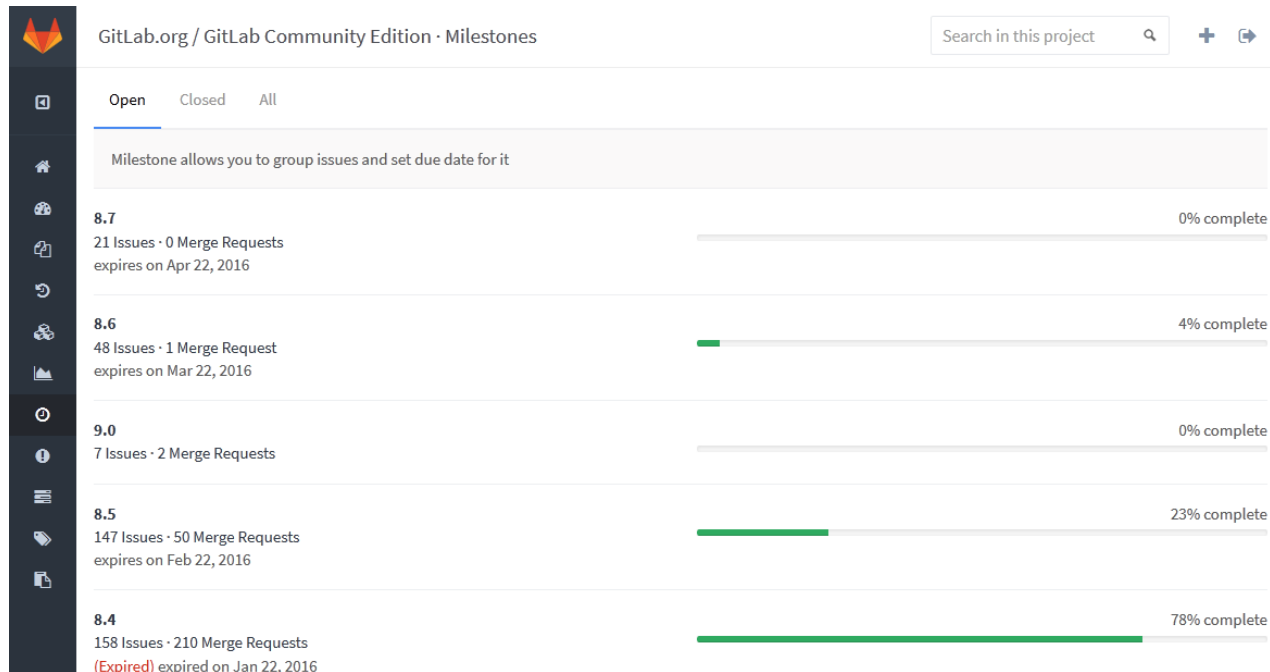
List of possible navigation in Eclipse: window → preferences → general → keys

Aufgabe 1 – Verifizieren und Validieren

Milestones: Legen Ziel für ein Projekt fest

- Oft Zusammenfassung von Issues und Merge Requests
- Liste der Issues, Merge Requests & Zieldatum
- In SCRUM als mögliche Sprintziele

Liste der Milestones eines Projekts mit Issues, Merge Requests & due date:



Aufgabe 1 – Verifizieren und Validieren

Issues: beschreiben konkrete Probleme/ Verbesserungsvorschläge

- Beschreibung
- Reproduzierbarkeit
- Klassifikation
- Verantwortlichkeiten
- open unassigned/ open assigned/ closed

GitLab.org / GitLab Community Edition · Issues

Search in this project

Open 2,021 Closed 3,013 All 5,034














Filter by title or description + New Issue

Author Assignee Milesto... Label Weight sort: Last created


| | | | |
|---|---|---|---------------------------|
| Build failed - you appear to have cloned a empty repository | #12728 · opened about an hour ago by Stefan van den Eertwegh | 0 | updated about an hour ago |
| [bug] MR with no change gets advertised as huge diff | #12727 · opened about an hour ago by Étienne Servais | 0 | updated about an hour ago |
| Deleting user should not delete their issues | #12726 · opened about 2 hours ago by Douwe Maan 8.6 | 1 | updated about 2 hours ago |
| Lines between file browser items (8.4.1, eb7f6690) | #12725 · opened about 2 hours ago by Jacob Vosmaer 8.4 Frontend | 3 | updated about 2 hours ago |
| Wrong sorting of commit order in MR view? | #12724 · opened about 3 hours ago by Artem Sidorenko | 0 | updated about 3 hours ago |

Aufgabe 1 – Verifizieren und Validieren

Issue:



GitLab.org / GitLab Community Edition · Issues Search in this project 🔍 + ↻




Open Issue #12717 · opened by  Sytse Sijbrandij · about 8 hours ago + New Issue


Allow the sidebar to be made wider on smaller screens

Report <https://about.gitlab.com/2016/01/15/making-gitlab-better-for-large-open-source-projects/#comment-2477979768>


When you collapse the sidebar manually on a large screen you get a **>** to expand it. Currently on a small screen it is collapsed automatically and you can't expand it. You should be able to collapse and expand on a small screen too.

/cc @jschatz1

 0  0 

 Jacob Schatz @jschatz1 · about 8 hours ago Master

[@sytse](#) If you were to expand the menu on a **iPhone 5 which is 320px** (points) wide in portrait (rendered at 2x) it would leave 90px left to display the content. iPhone 6 would leave 145px for content in portrait. Our content will not look good at 90px/145px wide. I suggest a menu that floats above the content and does not condense the content at that resolution.

 Étienne Servais @eservais · about 2 hours ago

This do not only affect smartphones but also two screens configuration. In my case, I have one vertical and one horizontal screen (both 1920*1080/1080*1920) and I have the **>** expand only on the landscape one. I don't say it's a bad choice but couldn't find any indication why this is not allowed. Is this based on the width of the screen or on the orientation?

Write Preview ↗ Edit in fullscreen

Markdown tip: Italicize words or phrases using `*asterisks*` or `_underscores_` 📎 Attach a file




Add Comment

Assignee: None

Milestone: **8.5**

Labels: Frontend bug

Weight: None

3 participants   

Notifications

Subscribe

You're not receiving notifications from this thread.

Aufgabe 1 – Verifizieren und Validieren

Verteilte Versionskontrolle:

- **Kein zentrales Repository, jeder Entwickler nutzt lokales Repository**
- **Es existiert aber für gewöhnlich ein offizielles Repository**

GitLab (Features):

- **Issue tracking: Issues werden kategorisiert, assigned, etc.**
- **Issue assignment: Issues sind assigned oder unassigned;
Entwickler assigned issue und bearbeitet issue dann (open assigned)**
- **Merge request: mehrere Branches in Repository → pull;
Meldung an andere Entwickler, dass ein Issue
abgeschlossen wurde; Anstoß zu Review und Diskussion**



Warum ist es sinnvoll innerhalb eines Projektes einen einheitlichen Workflow zu verwenden?

- **Automatisierte Erzeugen von konsistentem Code**
- **Vermeiden von Missverständnissen im Team**
- **Eindeutige Fehlerzuweisung zu Entwickler**
- **Erhöhte Motivation der Entwickler**

Aufgabe 1 – Verifizieren und Validieren

Workflow: Richtlinie zur Verwendung von DVCS

- einheitliches Vorgehen**
- Vermeidung von Fehlern**
- Zurückverfolgung von Changes etc.**

Aufgabe 1 – Verifizieren und Validieren

GitHub Workflow (vgl. GitLab Features):

- Create own branch from master:
 - master stays deployable
 - descriptive name for own branch
- Add commits:
 - keep track of development and add commit message
 - transparent history
 - separate unit → rollback possible
- Pull request (Merge request bei GitLab):
 - initiate discussion and reviewing
- Discussion:
 - questions can be answered
 - tests available?
 - according to code guidelines?
- Deploy:
 - discussion passed → deploy for verification under production
- Merge → into master branch

Continuous Integration (CI):

- Entwickler integrieren eigene Arbeitsversionen so oft wie möglich
- in das System um „integration hell“ zu verhindern
- Build und Tests laufen automatisiert auf einem Server
- hohe Toolunterstützung (Server-/ webbasiert)
- hohe Konfigurierbarkeit

Ablauf:

1. Entwickler nimmt Änderungen vor
2. Code funktioniert, Tests werfen keine Fehler
3. Code wird in VCS eingecheckt
4. CI Server bemerkt Änderungen
 - CI checks out Code
 - CI builds the whole system
 - CI runs all tests
 - Results are published
5. Entwickler werden über Probleme informiert



Was könnte in Zusammenhang mit Continuous Integration ein Nightly Build sein?

- **Eine häufig verwendete Form von CI in der Entwickler vornehmlich nachts entwickeln**
- **Eine Vorstufe von CI in der Build Prozess nachts und automatisiert erfolgt**
- **Nightly Build hat eigentlich nichts mit CI zu tun, das ist ein häufig auftretendes Missverständnis**



Formale Verifikation: Nachweis der Korrektheit eines Programms

- **Formales Modell als Spezifikation**
- **Mathematische Analyse des Modells**
 - **Modell (des Programms) in geeignetem Formalismus**
 - **Aussagen über Modell (das das Programm repräsentiert)**
- **Je ausdrucksmächtiger die Logik, desto mehr manueller Beweisaufwand**
 - **Werden Eigenschaften von Modell erfüllt**

(leichtgewichtige Methoden auf Programmcode möglich vgl. ACL2 aus VL oder C++ Checker, die problematische Stellen markieren)



Formale Techniken zur Verifikation:

- + erhöhte Systemsicherheit**
- + erhöhte Zuverlässigkeit des Systems**
- + Garantien können gegeben werden, die sonst nicht möglich sind**
- formale Spezifikation vs. tatsächliche Anforderungen**
- formale Beweise können Fehler enthalten**
- falsche Annahmen als Ausgangssituation**



Kontrollfluss-Graph (CFG): gerichteter Graph der Kontrollstruktur eines Programms repräsentiert

Testabdeckung: unterschiedlich, je nach Definition...

- **Zeilenabdeckung: Prozentsatz der getesteten ausführbaren Zeilen**
- **Anweisungsabdeckung: Prozentsatz der getesteten Anweisungen**
- **Zweigabdeckung: Prozentsatz der getesteten Kanten im CFG**
- **Pfadabdeckung: Prozentsatz der getesteten Pfade von Start- zu Endknoten**
- **Bedingungsabdeckung: Prozentsatz wird abhängig von Verfahren über Anzahl der getesteten Bedingungen ermittelt**

Es kann nie sichergestellt werden, dass ein System keine Fehler hat. Man kann nur sicherstellen, dass getestete Szenarien funktionieren.



Klassifikation von Tests:

- **Klassifikation nach Zielen: Validierungs- vs. Defekttesten**
 - **Validierung (Happy Path): prüfe ob Test gewünschte Funktion erfüllt**
 - **Defekt (Unhappy Path): versuche Fehlerzustände zu finden**
- **Klassifikation nach relativer Größe des SUT:**
Unit, Component, System (Integration)

Testarten:

- **Unit Test: isoliertes Testen einzelner Methoden/ Klassen**
- **Component Test: isoliertes Testen einzelner Systemkomponenten**
- **Integration Test: Testen des kompletten Systems (Entwickler)**
- **Akzeptanz Test: Testen ob Business Requirements erfüllt werden (Kunde)**

Aufgabe 3 – Testen



Weitere Testarten:

- **Regression Test:** behobene Fehler testen
- **Performance Test:** Verhalten des Systems unter Belastung
- **Usability Test:** wie gut kommen User mit System zurecht
- **Entwicklungstest:** Tests während der Entwicklungsphase durchführen
- **Release Test:** Vollständige Systemversion wird vor Übergabe getestet
- **Endbenutzertest:** Benutzer testen System in eigener Systemumgebung

Strategien:

- **White-Box:** Zugriff auf Systeminterna (inklusive Code)
- **Grey-Box:** Zugriff auf Designdokumente aber nicht auf Code
- **Black-Box:** kein Zugriff auf Systeminterna
- **Top-Down:** beginne mit Tests auf hoher Integrationsebene
- **Bottom-Up:** beginne mit Tests auf niedrigster Integrationsebene



Klassisches Fuzzing ist immer eine Form des...

- **...White-Box Testing**
- **...Black-Box Testing**
- **...Top-Down Testing**
- **...Bottom-Up Testing**



„Mocks“, „Stubs“, „Dummy-“ und „Fake-Objects“ werden in erster Linie bei...

- **...Unit Tests...**
- **...Component Tests...**
- **...Integration Tests...**
- **...Akzeptant Tests...**

...verwendet.



Unit Test: isoliertes Testen einzelner Methoden/ Klassen
Objekte anderer Klassen stehen nicht zur Verfügung,
werden aber häufig benötigt

Test Doubles:

- **Mock: Platzhalter, der Schnittstellen simuliert und überprüft ob erwartete Aufrufe erfolgen**
- **Stub: stellt pseudo Antworten für Aufrufe innerhalb des Tests bereit**
- **Dummy Object: Werden nicht genutzt, füllen lediglich Parameterlisten**
- **Fake Object (drivers): enthalten funktionierende Implementierung, die aber so abgespeckt ist, dass sie im eigentlichen Programm nicht verwendet werden**



Was ist der Unterschied zwischen Mocks und Stubs?

- **Mocks verifizieren simulierten Programmablauf, Stubs nicht**
- **Stubs verifizieren simulierten Programmablauf, Mocks nicht**
- **Ein Mock ist immer auch ein Stub**
- **Ein Stub ist immer auch ein Mock**