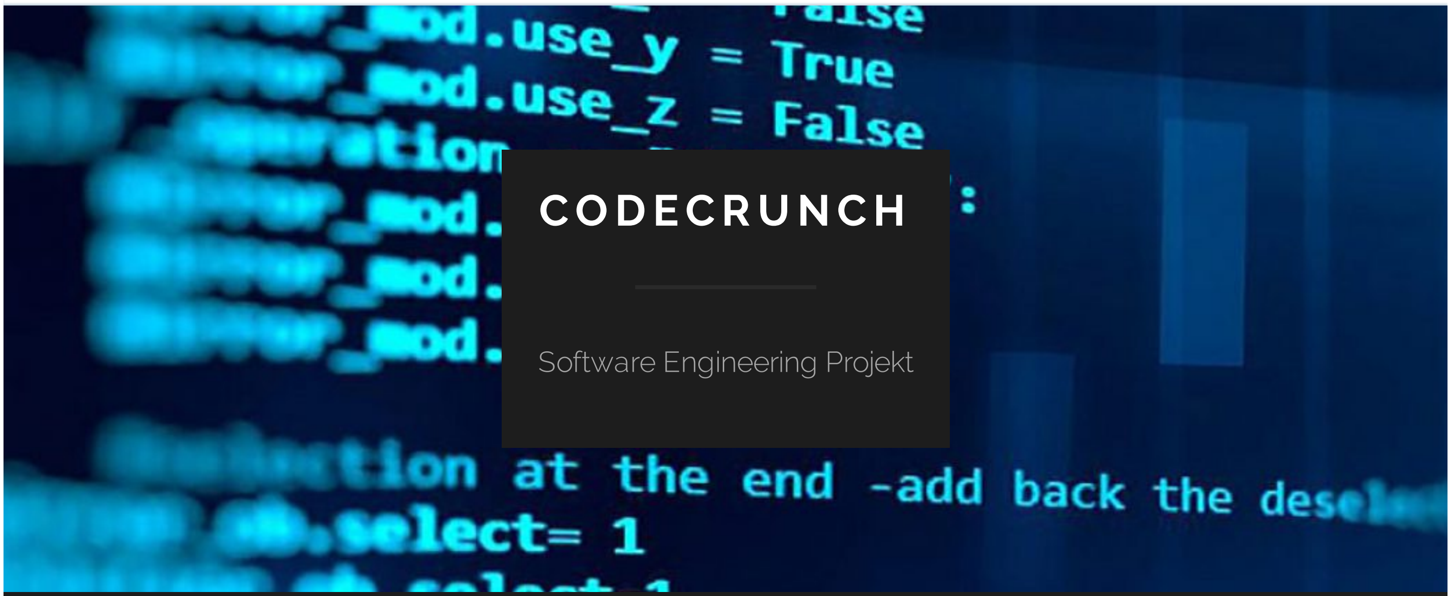


CODECRUNCH

Software Engineering Projekt

Week 22 – Our final Blogpost

17. JUN 2019 / CODECRUNCH / LEAVE A COMMENT

Hello everyone, this is our final Blogpost that can be seen as a main hub for our project.

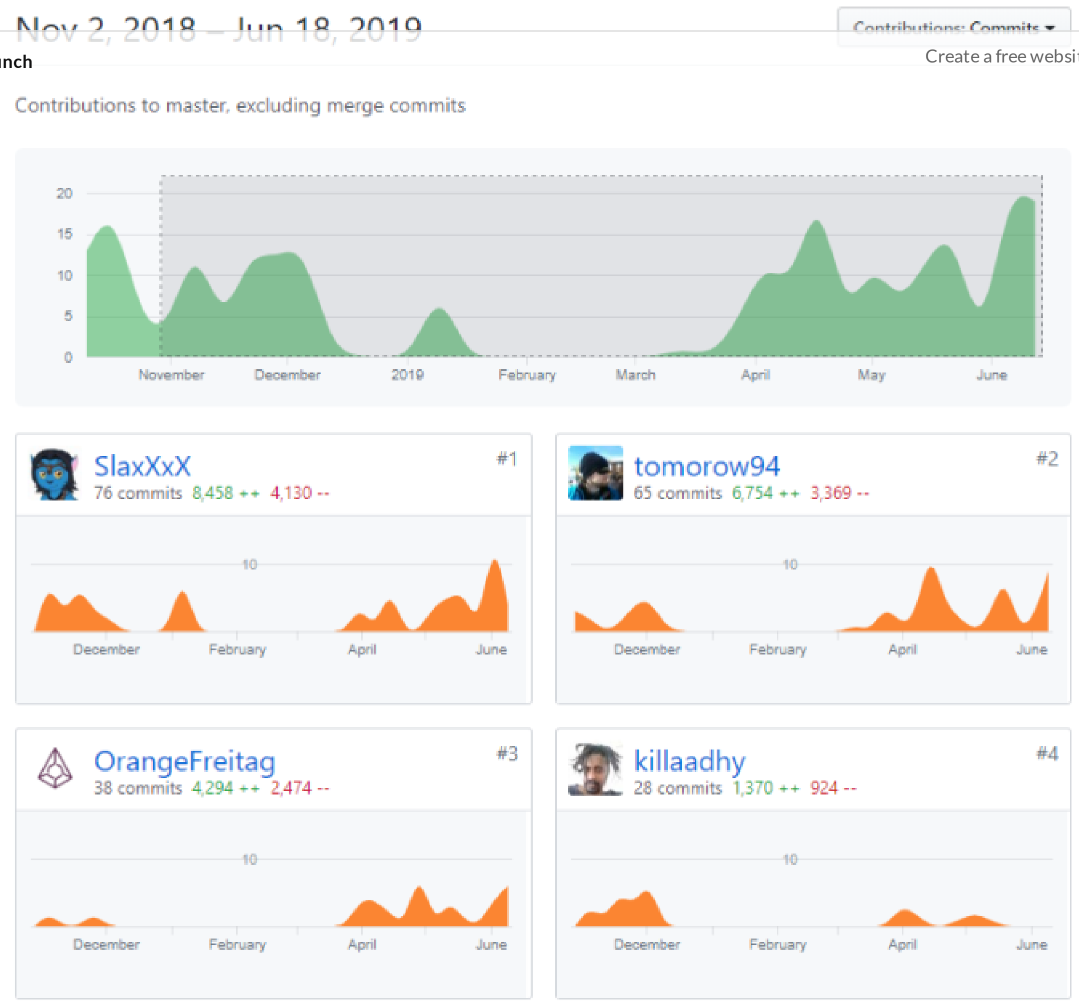
Below you will find all links, files and blogposts of this project.

Important Links:

GitHub

As in the first month we were occupied with getting the framework going,

all the code before november can be ignored. It was just auto generated over and over by our engine framework.


Nov 2, 2018 – Jun 18, 2019
Contributions: Commits
Contributions to master, excluding merge commits
20
15
10
5
0
November
December
2019
February
March
April
May
June
SlaxXxX #1
76 commits 8,458 ++ 4,130 --
tomorow94 #2
65 commits 6,754 ++ 3,369 --
OrangeFreitag #3
38 commits 4,294 ++ 2,474 --
killaadhy #4
28 commits 1,370 ++ 924 --

GitHub names:

Janis Schneider – SlaxXxX

Kai Schwark – tomorow94

Nirjan Thangaraja – killaadhy

Falko Reinert – OrangeFreitag

(This is also mentioned in our repositories readme)

Youtrack

SonarCloud

TravisCI

Important Files:

Class Diagram

SRS

SAD

Test Plan

OUCD

Our Video

Our Presentation

Blogposts:

Week 1 – Vision

Week 2 – Team Roles and Technologies

Week 3 – Software Requirements Specification

Week 4 – Use Cases

Week 5 – feature files

Week 6 – Class Diagram

Week 7 – Scrum

Week 8 – Scrum – Retrospective

Week 9 – SAD

WEEK 10 – Running Cucumber Test

WEEK 11: MIDTERM EXAM

Week 12 – Confirmation of Scope

Week 13 – Risk Management

Week 21 – Installation

12. JUN 2019 / CODECRUNCH / 2 COMMENTS

Hello everyone,

This weeks blogpost will show you how to install our game:

Visit our GitHub releases page

Download (ideally on your phone) the newest release APK and install it:

How to install APK files

That's already it! It's that simple. Have fun and see you next week to our final blogpost.

Week 20 – Deployment CI & Tech Stack

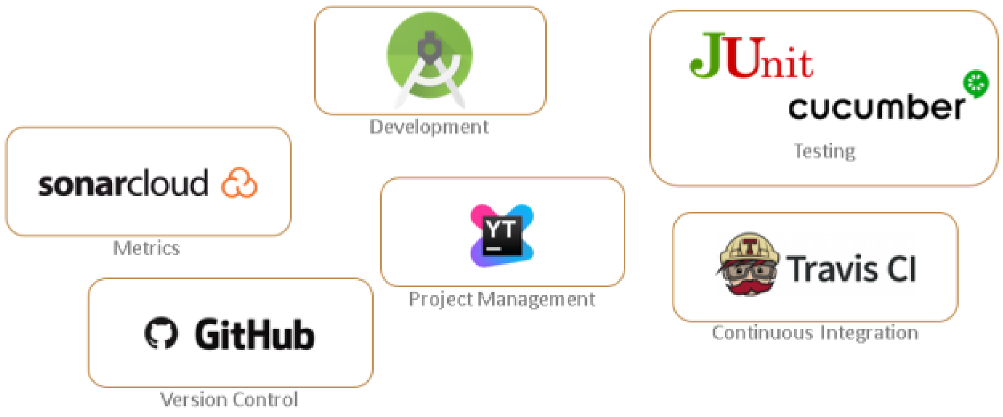
3. JUN 2019 / CODECRUNCH / LEAVE A COMMENT

Hello everyone,

in this blog post we would like to provide you with an overview of all the technologies we are using to develop our project and how we applied Continuous Integration Software to improve the efficiency of the project's development process.

Tech Stack

When developing software it is necessary to create an efficient software development infrastructure. This infrastructure usually consists out of a set of tools and technologies, which each individually fulfill a unique purpose by solving a characteric problem which occurs as a result of the development process for software, making the development of software easier to manage. Since software can take many different forms on a high-level, the set of tools used to develop software can vary significantly. In the case of our project, an android game, different tools are needed than when developing a web application. An example of a technology, which is unique to the type of project we are developing, is the libgdx game development framework. Libgdx provides a set of classes implemented in Java, which are frequently needed when developing android games. But our game is not a completely unique piece of software it does of course share a lot of similarities with other software, therefore we are also using a lot of technologies, which solve common challenges in software development processes. Examples of technologies which are not unique to our project, are Discord for communication, Anroid Studio for development, JUnit for testing, sonarcloud to provide metrics etc. The full set of technologies we are using is seen in the diagram below.



Continuous Integration with Travis CI

As the code of a software development project becomes increasingly complex, it is difficult to ensure that a project's codebase works as intended. As a result of this problem many different continuous integration tools have been developed . Travis CI which was released in August 2013 is an example of such a tool. Travis CI solves the codebase integrity problem by providing an easy way to run the code of software development projects in the cloud automatically at user specified timings.

In order for Travis CI to run a project's code as intended a configuration file has to be specified and placed into project's root directory. The needed configuration file goes by the name of "Travis.yml".

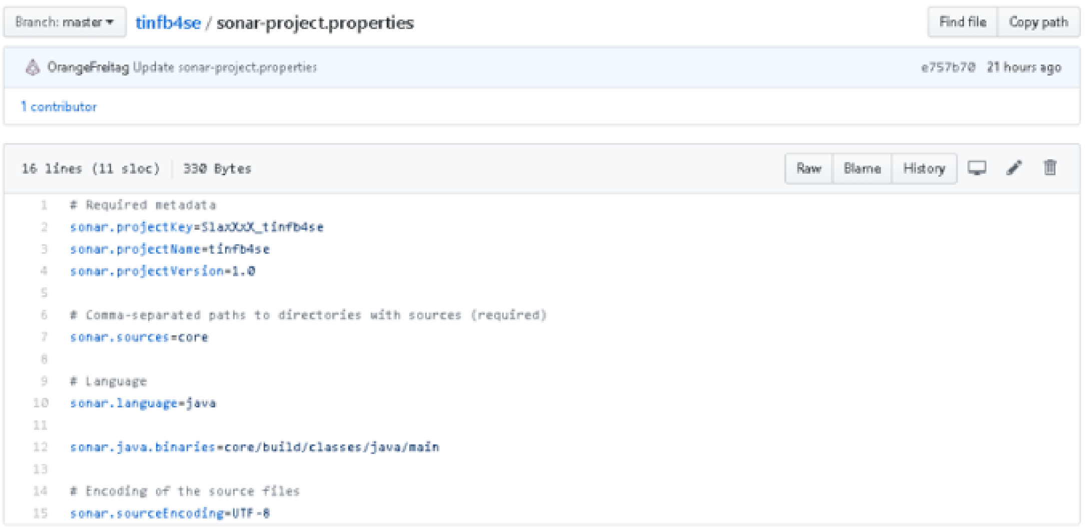
The following diagram shows the "Travis.yml" file of our project

```
language: android
sudo: false
android:
  components:
  - android-28
  - build-tools-28.0.3
  - android-22
  - sys-img-armeabi-v7a-android-22
addons:
  sonarcloud:
    organization: slaxxxx-github
    token:
      secure: 07330331709bf2b98e256c455067a8bf0d1ba5af
script:
- echo "starting script"
- "./gradlew clean build"
- sonar-scanner
```

As shown by the "travis.yml" content the main use-case for Travis CI in our project is the ability to trigger sonarcloud analyzes automatically through Travis CI builds.

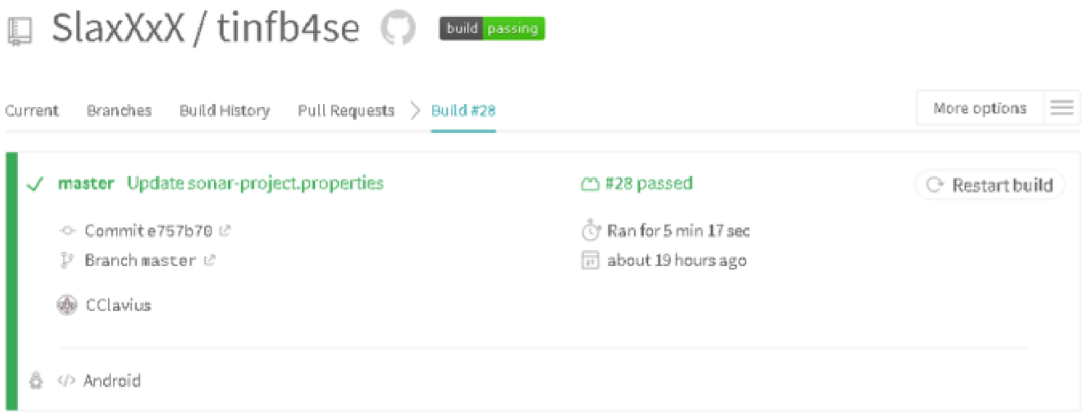
Apart from the configuration already shown within the "travis.yml", it is necessary to specify and include a "sonar-project.properties" file in the root directory of the project, where the sonarcloud analysis should occur, in order to let Travis CI trigger sonarcloud analyzes. The "sonar-project.properties" is required because it provides a Travis CI instance, running the automated builds, with the directory path at which code has to be analysed by sonarcloud.

The diagram below shows the "sonar-project.properties" file for our project.

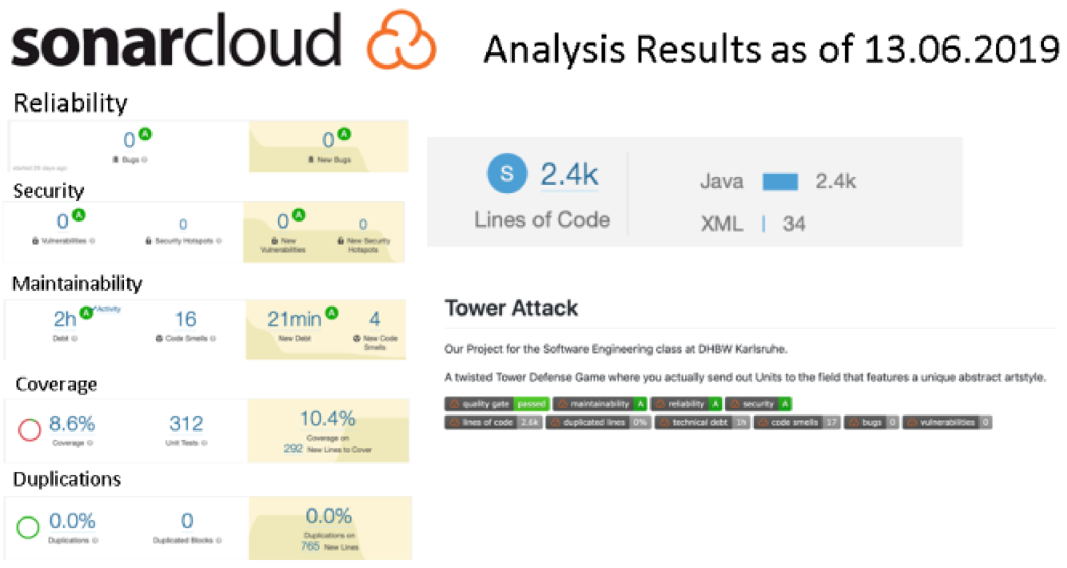

Branch: master
tinfb4se / sonar-project.properties
Find file
Copy path
OrangeFreitag Update sonar-project.properties
e757b70 21 hours ago
1 contributor
16 lines (11 sloc) | 330 Bytes
Raw
Blame
History

```
# Required metadata
sonar.projectKey=SlaxXxX_tinfb4se
sonar.projectName=tinfb4se
sonar.projectVersion=1.0

# Comma-separated paths to directories with sources (required)
sonar.sources=core

# Language
sonar.language=java

sonar.java.binaries=core/build/classes/java/main

# Encoding of the source files
sonar.sourceEncoding=UTF-8
```

If these configurations have been made the travis build belonging to the project should pass successfully and trigger a sonarcloud analysis.



As an example of how sonarcloud results look like here is an image of our recent analysis results.


Analysis Results as of 13.06.2019

Reliability

0 A Bugs | 0 A New Bugs

Security

0 A Vulnerabilities | 0 Security Hotspots | 0 A New Vulnerabilities | 0 New Security Hotspots

Maintainability

2h A Debt | 16 Code Smells | 21min A New Debt | 4 New Code Smells

Coverage

8.6% Coverage | 312 Unit Tests | 10.4% Coverage on 292 New Lines to Cover

Duplications

0.0% Duplications | 0 Duplicated Blocks | 0.0% Duplications on 765 New Lines

2.4k Lines of Code

Java 2.4k

XML 34

Tower Attack

Our Project for the Software Engineering class at DHBW Karlsruhe.

A twisted Tower Defense Game where you actually send out Units to the field that features a unique abstract artstyle.

quality gate passed | maintainability A | reliability A | security A

lines of code 2.6k | duplicated lines 0% | technical debt 1h | code smells 17 | bugs 0 | vulnerabilities 0

Conclusion

We hope that we were able to provide you with an overview of the tools we were using and an explanation how we implemented Travis CI in our project with this blog post.

Best Regards,

Falko@CodeCrunch

Week 19 – Metrics

26. MAY 2019 / CODECRUNCH / 4 COMMENTS

Hello everyone!

This week it's all about metrics. Our tool of choice for this task is Sonarcloud. We have decided to improve a code snippet in 2 different measures, which will be:

WMC (Weighted Methods Per Class): The weighted sum of all methods of a class.

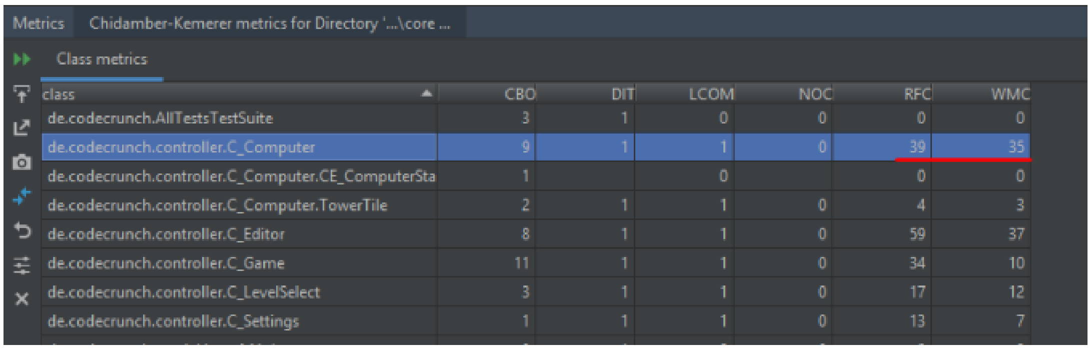
RFC (Response for a Class): The response set of a class that can potentially be executed in response to a message received by an object of that class.

Futher information about our chosen metrics: Chidamber & Kemerer object-oriented metrics

For a more detailed overview of our code metrics we used the Android Studio plugin "Metrics Reloaded". We chose to refactor the "C_Computer" class. Intelligent calculation and the simulated impression of human-like tower placement has put this class to the top of our metrics tool.

Before:

C-Computer on GitHub


Metrics | Chidamber-Kemerer metrics for Directory '...\core ...

Class metrics

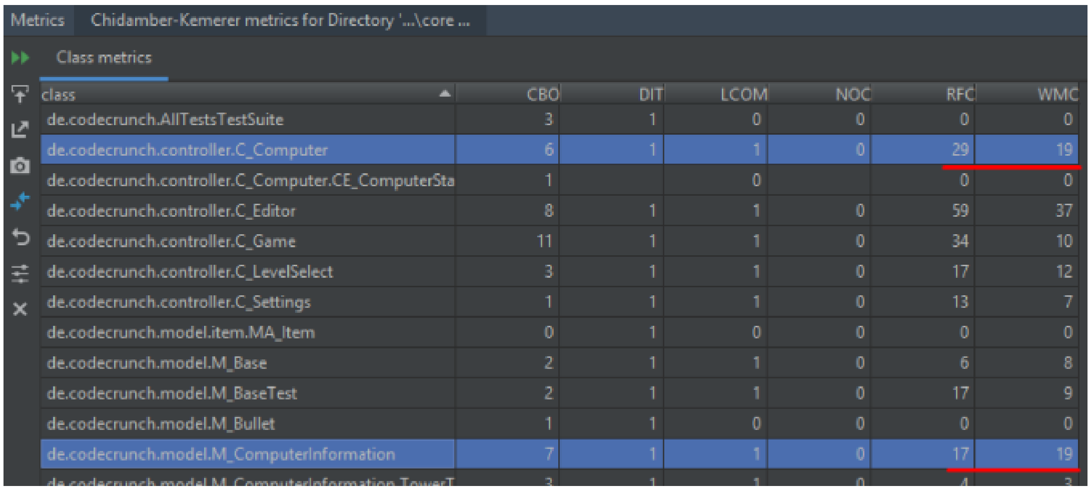
class	CBO	DIT	LCOM	NOC	RFC	WMC
de.codecrunch.AllTestsTestSuite	3	1	0	0	0	0
de.codecrunch.controller.C_Computer	9	1	1	0	39	35
de.codecrunch.controller.C_Computer.CE_ComputerSta	1		0		0	0
de.codecrunch.controller.C_Computer.TowerTile	2	1	1	0	4	3
de.codecrunch.controller.C_Editor	8	1	1	0	59	37
de.codecrunch.controller.C_Game	11	1	1	0	34	10
de.codecrunch.controller.C_LevelSelect	3	1	1	0	17	12
de.codecrunch.controller.C_Settings	1	1	1	0	13	7

For this refactoring process, we chose to split the tasks of this class into it's two main categories: Simulation of the computer "performing actions" in the game (C_Computer) and the storage and calculation of it's information about the field, mainly the best spots to place towers on (M_ComputerInformation).

After:

C-Computer on GitHub

M-ComputerInformation on GitHub


Metrics | Chidamber-Kemerer metrics for Directory '...\core ...

Class metrics

class	CBO	DIT	LCOM	NOC	RFC	WMC
de.codecrunch.AllTestsTestSuite	3	1	0	0	0	0
de.codecrunch.controller.C_Computer	6	1	1	0	29	19
de.codecrunch.controller.C_Computer.CE_ComputerSta	1		0		0	0
de.codecrunch.controller.C_Editor	8	1	1	0	59	37
de.codecrunch.controller.C_Game	11	1	1	0	34	10
de.codecrunch.controller.C_LevelSelect	3	1	1	0	17	12
de.codecrunch.controller.C_Settings	1	1	1	0	13	7
de.codecrunch.model.item.MA_Item	0	1	0	0	0	0
de.codecrunch.model.M_Base	2	1	1	0	6	8
de.codecrunch.model.M_BaseTest	2	1	1	0	17	9
de.codecrunch.model.M_Bullet	1	1	0	0	0	0
de.codecrunch.model.M_ComputerInformation	7	1	1	0	17	19
de.codecrunch.model.M_ComputerInformation.TowerT	3	1	1	0	4	3

This splits the "load" of the computer in both Metrics into the two new classes, decreasing the RFC value of C_Computer by 10 and WMC by 16.

A spot where we chose not to refactor, even though the class has an RFC of 37 is M_HealFountainTest, simply because it is a unittest and is allowed to have lots of methods and methodcalls.

Excelsior! Janis@CodeCrunch

Week 18 – Design Patterns

21. MAY 2019 / TINFB4SE / 4 COMMENTS

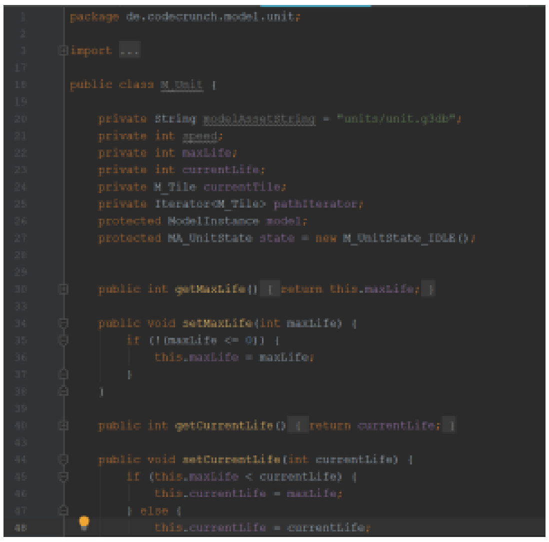
Hi everybody,

We are very late with our blog post because each of us had an immense workload this week. Sorry for that.

Our task for this week is focused on the topic Design Patterns.
We had to choose one of many Design Patterns and implement it into our project.
As we have already mentioned, we are using a Java Framework called libGDX, which has already some patterns implemented like MVC.

We decided to use the Template Method Pattern for our units.
The idea of this Pattern method is an algorithm in a base class using abstract operations that subclasses override to provide concrete behavior.

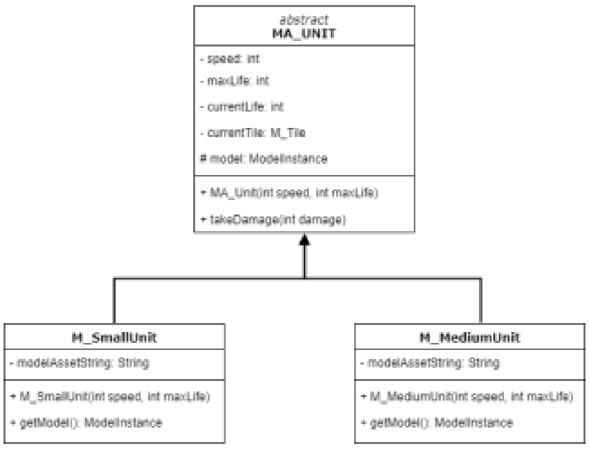
Before Refactoring



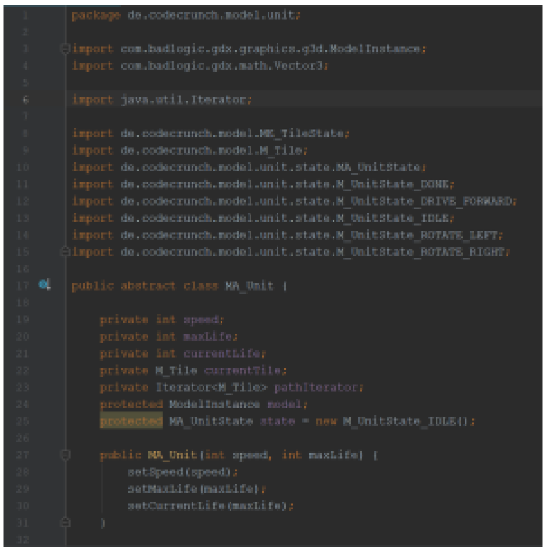
```java
package de.codecrunch.model.unit;

import ...

public class M_Unit {

    private String modelAssetString = "units/unit.g3db";
    private int speed;
    private int maxLife;
    private int currentLife;
    private M_Tile currentTile;
    private Iterator<M_Tile> pathIterator;
    protected ModelInstance model;
    protected MA_UnitState state = new M_UnitState_IDLE();


    public int getMaxLife() { return this.maxLife; }

    public void setMaxLife(int maxLife) {
        if (!(maxLife <= 0)) {
            this.maxLife = maxLife;
        }
    }

    public int getCurrentLife() { return currentLife; }

    public void setCurrentLife(int currentLife) {
        if (this.maxLife < currentLife) {
            this.currentLife = maxLife;
        } else {
            this.currentLife = currentLife;
```

At the beginning, we had only one unit for the game. The unit was realized in the class M_Unit. When creating an other unit, we noticed that this new class is almost a copy of M_Unit. Only unit-specific characteristic such as the speed have been adjusted.

After Refactoring

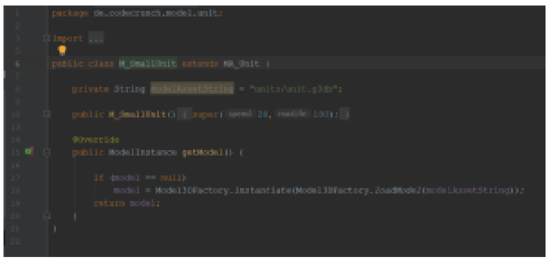

abstract
MA_UNIT
- speed: int
- maxLife: int
- currentLife: int
- currentTile: M_Tile
# model: ModelInstance
+ MA_Unit(int speed, int maxLife)
+ takeDamage(int damage)
M_SmallUnit
- modelAssetString: String
+ M_SmallUnit(int speed, int maxLife)
+ getModel(): ModelInstance
M_MediumUnit
- modelAssetString: String
+ M_MediumUnit(int speed, int maxLife)
+ getModel(): ModelInstance

After refactoring the code with the design pattern, there is an abstract unit class "MA_UNIT" which has all the methods for each unit.



```
package de.codecrunch.model.unit;

import com.badlogic.gdx.graphics.g3d.ModelInstance;
import com.badlogic.gdx.math.Vector3;

import java.util.Iterator;

import de.codecrunch.model.ME_TileState;
import de.codecrunch.model.M_Tile;
import de.codecrunch.model.unit.state.MA_UnitState;
import de.codecrunch.model.unit.state.M_UnitState_DONE;
import de.codecrunch.model.unit.state.M_UnitState_DRIVE_FORWARD;
import de.codecrunch.model.unit.state.M_UnitState_IDLE;
import de.codecrunch.model.unit.state.M_UnitState_ROTATE_LEFT;
import de.codecrunch.model.unit.state.M_UnitState_ROTATE_RIGHT;

public abstract class MA_Unit {

    private int speed;
    private int maxLife;
    private int currentLife;
    private M_Tile currentTile;
    private Iterator<M_Tile> pathIterator;
    protected ModelInstance model;
    protected MA_UnitState state = new M_UnitState_IDLE();

    public MA_Unit(int speed, int maxLife) {
        setSpeed(speed);
        setMaxLife(maxLife);
        setCurrentLife(maxLife);
    }

```

The new subclasses for each unit just override the unit-specific characteristic to provide concrete behavior. Each unit has specific characteristic like speed and life.



```
package de.codecrunch.model.unit;

import ...

public class M_SmallUnit extends MA_Unit {

    private String modelAssetString = "units/unit.g3db";

    public M_SmallUnit() { super( speed: 20, maxLife: 100); }

    @Override
    public ModelInstance getModel() {

        if (model == null)
            model = Model3DFactory.instantiate(Model3DFactory.loadModel(modelAssetString));
        return model;
    }
}

```

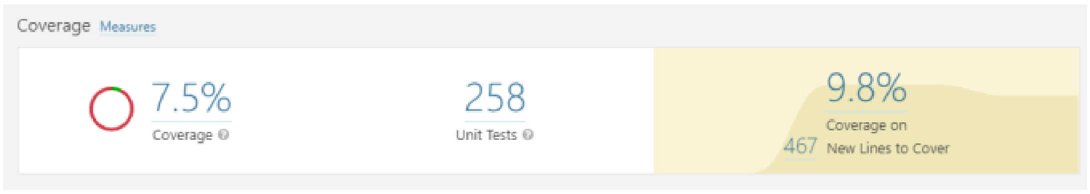
Week 18 – Coverage and Testplan

18. MAY 2019 / CODECRUNCH / 1 COMMENT

Code Coverage:

To calculate our code coverage, we use Jacoco.

The is then updated along with the other Metrics on SonarCloud by our Travis CI Tool.



As you can see, our code coverage is quite low, despite having quite a lot of tests.

This is because our game is at the most crucial points managed by our game engine, which we can't just mock in order to create tests. This resulted in very little area we actually had enough control over to test their function.

Testplan and our third Test:

We decided for our application being a game our third test had to be a user acceptance test (UAT). To gain useful feedback in a structured way we decided to set up a survey the testers had to fill out. The first questions were meant to be answered before testing our application and the last five afterwards. The results of the survey can be found here.

Our now complete testplan containing this and the two other tests can be found here.

Week 17 – Retro 2

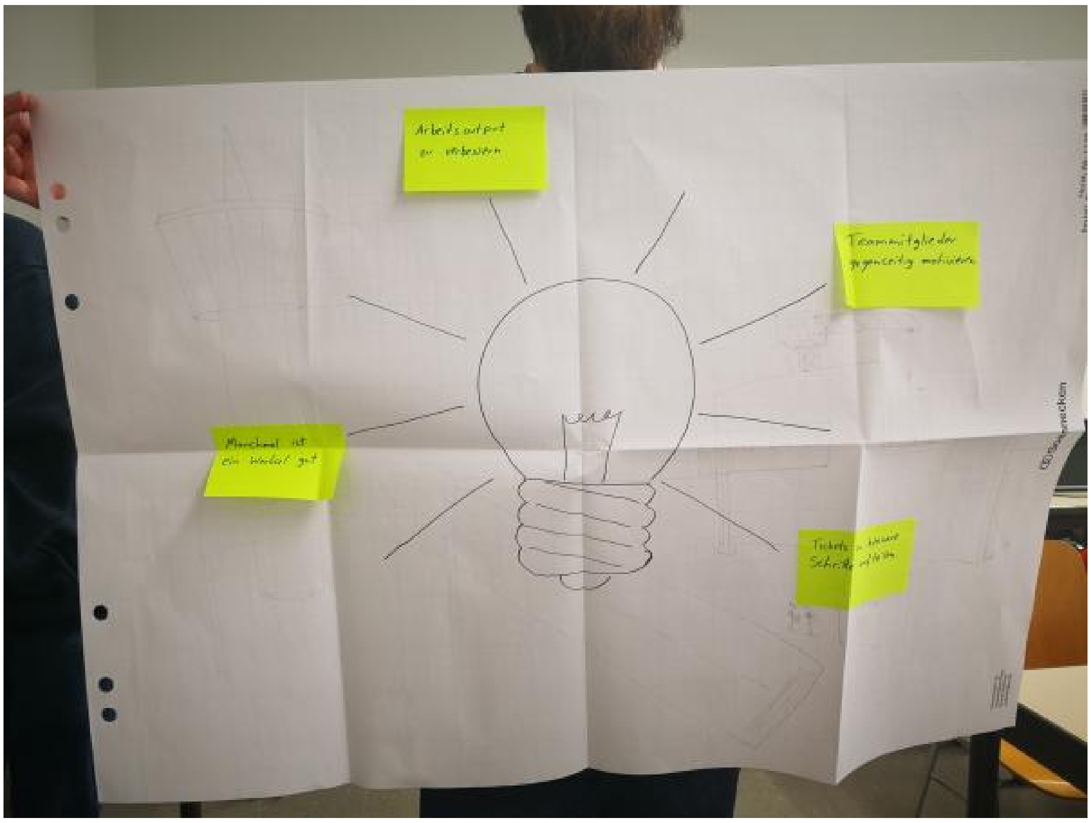
9. MAY 2019 / CODECRUNCH / 3 COMMENTS

Hi Everyone,

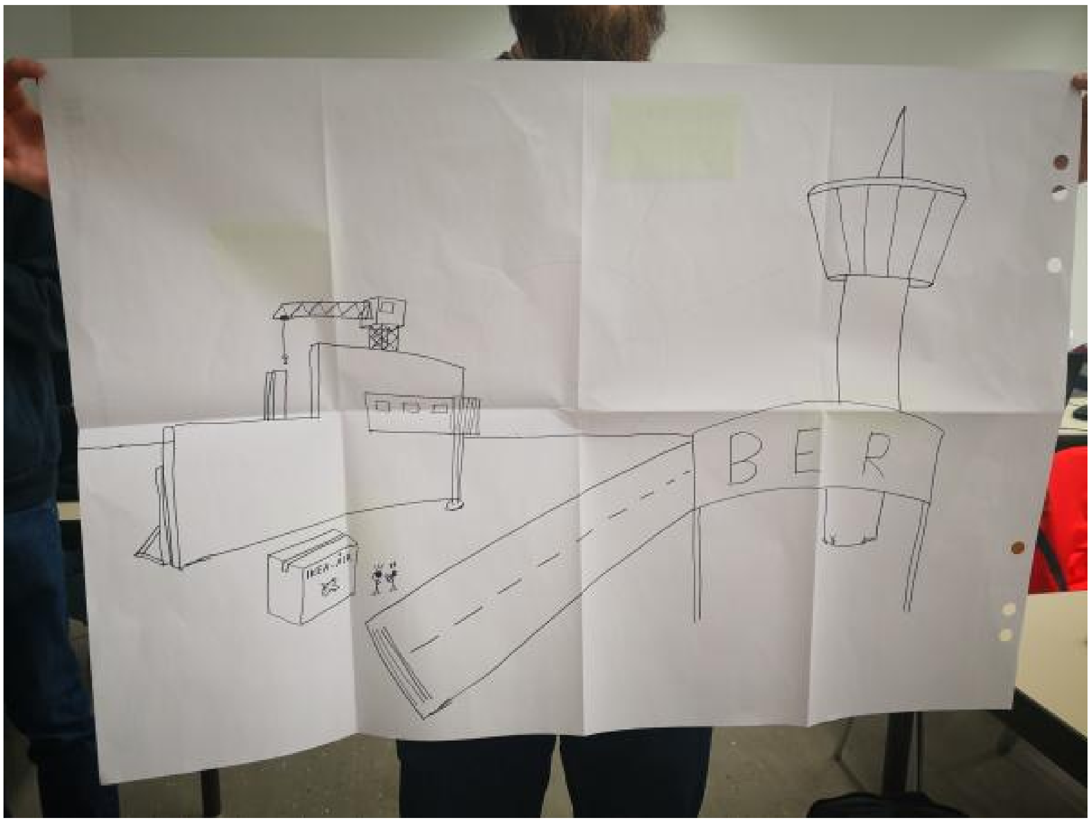
Today was the second time a SCRUM Master visited our class.

At first everyone had to chose a type of car that represents the own role in the team and tell why we choose it. There were many interesting results in the class. The members of our group identified with an ADAC-car, because of testing and writing tests most of the time, and a Police-car, because of directing everyone the right way in case of emergency. The type of car named the most in our group was a safari-car, that is heading out in unknown territory, discovering and accumulate knowledge.

Second we were to write down what we learned since the last retrospective we had in November 2018.


Arbeitsoutput zu verbessern
Teammitglieder gegenseitig motivieren
Manchmal ist ein Werkzeug gut
Tickets in kleinere Schritte aufteilen

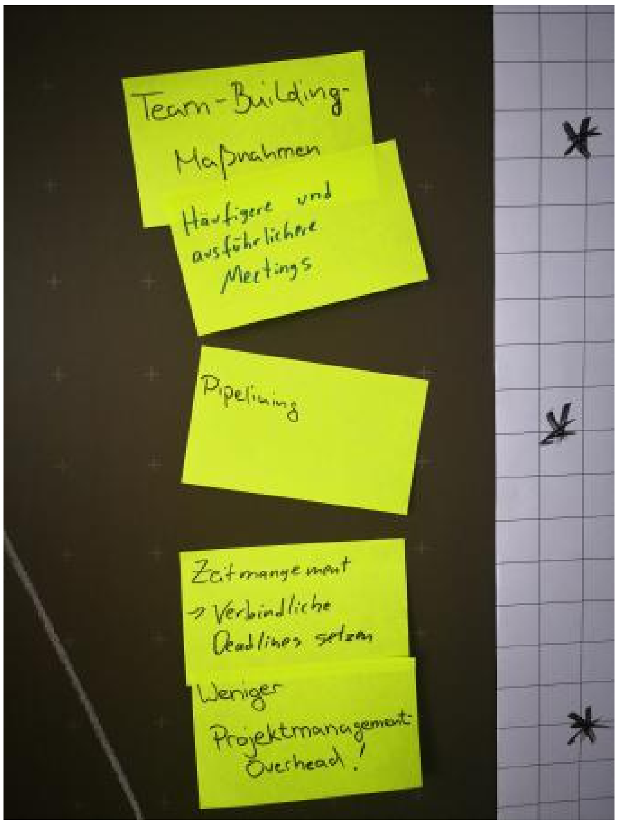
Next up we should paint a picture that represents our group onto a sheet of a flipchart. Thinking about our current state of our project and were we want to get we came up with the following picture:



We chose to draw the new airport of Berlin "BER" because we feel the same about our project. We already got the Menu, Settings and Editor done, what is represented by the finished flight tower. The runway stands for the finished maps and paths in our game. The towers are there but nothing more than a cardboard cutout like the building in the back. Last but not least the units are there, too. We just need to put all the parts at the fitting position. And we are standing in middle of the mess with more question-marks above our heads than we like.

The last task was to find things we can still improve in the last weeks. These are our ideas:

- Even more tests
- Team building
- Pipelining
- Time-management / less project management overhead


Team-Building Maßnahmen
Häufigere und ausführlichere Meetings
Pipelining
Zeitmangement → Verbindliche Deadlines setzen
Weniger Projektmanagement Overhead!