Chapter 1

Toolchain & Odroid

1.1 GCC/G++[JK]

In our initial presentation we held in front of the class, we talked about possibly using Clang as our compiler, due to some research we did during our C course (where we noticed that for example software interrupts aren't as good supported in gcc as they are in clang).
Due to the fact that everybody used gcc straight from the beginning and there was not really a discussion in the group about it, we stuck with it.

1.2 CMake[JK]

As proposed by the teacher, we are using CMake. It made the whole build process easier and more comfortable. Also the support for CMake of all modern IDEs allowed each member to use the IDE/development environment of his choice.
Another big advantage using the CMakeLists.txt is the overview we get. For example is it easy to find out the level of warnings we are using in that software project and the external software components (libraries, etc) we are linking the software with.
Notice that we mark all INCLUDE_DIRECTORIES as SYSTEM, to avoid warnings coming from these external software entities.

1.3 Jenkins[JK]

To automatically build the software components each time we push changes in their code to a certain branch, we introduced Jenkins to our toolchain.
To make this possible, Jenkins is connected to our GitHub repository with a so called webhook.
To ensure that also an overall build (on the Odroid) will work, we introduced an overall build behavior based on an overall CMake file as well. The server we are running the build on reflects the same build chain as it is installed on the Odroid (except the compiler, because of the hardware architectures are different). All necessary build tools and libraries are installed at the same paths.

1.4 pmccabe[JK]

We decided to integrate pmccabe directly into our Jenkins builds. Before each build starts, we run the pmccabe command to show us the ten most complex functions of the current project (see pmccabe man page).
To see the result, we just went to a specific build and looked at the console output.

1.5 Cppcheck[ME]

Analysing the code is a good way to get an overview about coverage and hidden bugs.
This tool in particular is a static analysis tool. That means that the code will be analysed in a static way, giving us an overview about static bugs as:

- Out of bounds checking
- Memory leaks checking
- Null pointer references
- Uninitialized variables
- Invalid usage of STL (Standard Template Library)
- Unused or redundant code
- Obsolete or unsafe functions

Unfortunately this tool will not give us any information about code coverage as the program is not being actively monitored while running, hence static analysis.

1.6 SonarQube[ME]

This is a open source platform for continuous inspection of code quality.
We use this in collaboration with Jenkins and Cppcheck, every build in Jenkins will be inspected and the results will be uploaded to this platform where it presents the Cppcheck results in a human readable and pleasant way.

When looking at the results all warnings/failures from Cppcheck will be transformed into a new “Issue” and it clearly states the bug and where it can

be found. All of the issues are also converted into Technical Debt according to predefined rules bundled with SonarQube. This is something while it's nice to have, we've not really been using.

1.7 Operating System[JK]

To have a lightweight operating system at hand, we decided to use a Debian 8.4 *Jessie* minimal image, available on the Odroid forums.
Most of our team members used a debian-based linux distribution before, so it seemed to be the most reasonable choice if there would be the need for them to work with the package manager, etc. The use of a minimal image made sure, that we would not have any unnecessary software, like a Desktop Environment, installed.

1.8 Connection[JK]

To connect to the Odroid, two of our team members were able to host a hotspot, which the Odroid could connect to.
This hotspot made sure, that the Odroid had an internet connection, so updating packages on it was possible, as well as pushing changes to the code made while testing and integrating could be pushed to the git repository (note that these changes were pushed with the user TacoServ).
We accessed the Odroid with an SSH connection subsequently.