

# Designing Infrastructure for an Online Teaching Assistant

## Bachelor project

Department of Computer Science at the University of Copenhagen  
(DIKU)

Truls Asheim <truls@asheim.dk>

June 21, 2015

# Contents

<b>Contents</b>	<b>1</b>
<b>1 Introduction</b>	<b>3</b>
1.1 Background and motivation . . . . .	3
1.2 Related work . . . . .	4
<b>2 Analysis</b>	<b>5</b>
2.1 Data overview . . . . .	5
2.2 Personas . . . . .	6
2.3 Assessment flow . . . . .	6
2.4 Submission lifecycle . . . . .	6
2.5 Submission constituents . . . . .	7
2.6 Executing untrusted code . . . . .	7
2.7 Design principles . . . . .	8
2.8 Handling user identities . . . . .	9
2.9 Interacting with the system . . . . .	10
2.10 Large Submissions . . . . .	11
<b>3 System Architecture Design</b>	<b>12</b>
3.1 General concepts and overall goal . . . . .	12
3.2 Users . . . . .	12
3.3 Components . . . . .	13
3.4 System flow . . . . .	15
3.5 Interactions . . . . .	15
3.6 Storage . . . . .	16
3.7 Interfaces . . . . .	17
3.8 Scaling and distributing . . . . .	17
<b>4 Measurements</b>	<b>18</b>
4.1 Metrics of submission assessment . . . . .	18
4.2 Performance thresholds . . . . .	18
4.3 System scaling requirements . . . . .	18
4.4 Estimating static requirements . . . . .	19

<i>CONTENTS</i>	2
<b>5 Discussion and conclusions</b>	<b>20</b>
5.1 Practical deployment considerations . . . . .	20
<b>Bibliography</b>	<b>21</b>

# Chapter 1

## Introduction

In this report, we will describe the design and implementation of an infrastructure for a system facilitating submission and automated correction of student assignments.

Unfortunately, some of the learning objectives proved overly ambitious for the time frame that we had available. In particular, we had hoped that our proof of concept implementation would be in a much more complete than it currently is. This impairs our ability to present firm conclusions but we do, however, expect that the design and testing methodologies described provides a solid starting point for such conclusions to be drawn at a later point.

A Master's project carried out in the spring of 2014 has laid the foundation for this work by discussing the intricate aspects of performing automated assessment of assignments[1]. This work builds upon the previous project by describing an infrastructure for a complete system supporting the submission and automated assessment of assignments

FiXme Note:  
Rewrite last  
part

### 1.1 Background and motivation

As a result of the study progress reform recently passed by the Danish parliament, students are required to finish their university studies faster than previously[2]. A consequence of this, is that many students can no longer find the time to act as TA's alongside their studies.

This has sparked a general interest at DIKU<sup>1</sup> to investigate implementing an automated assignment assessment system in order to reduce the workload for teachers and teaching assistants TA's.

Since the assignments of many Computer Science courses require students to complete programming homework, Computer Science finds itself in a unique position with regard to take advantage of automated assignment assessment systems.

---

<sup>1</sup>Department of Computer Science, University of Copenhagen

The system that we here will describe does not intend to replace TA's entirely, but rather automating the tedious of compiling and executing student programs. Human involvement is still needed to assess submissions as a whole.

An additional goal of this system is to be used as an Intelligent Tutoring System (ITS)[3] which can provide support for students while they are solving their assignments.

## 1.2 Related work

Tools for performing automated assessment of programming assignments is currently an actively researched topic. As a result of this, a large number of systems for performing this task has emerged over the years [4].

## Chapter 2

# Analysis

We will describe some of the aspects of handling student submissions in this analysis chapter of the report

### 2.1 Data overview

The system are required to handle several sensitive data items which must be protected from adversaries before the online assessment system can be put into production. The types of data are listed here in order of decreasing sensitivity.

1. The submissions of other students
2. The code for assessing an assignment
3. The completed assessments of assignments.

The first two items on the list are close in terms of sensitivity. We regard the submissions of other students as being the most sensitive data since they have a larger potential to enable plagiarism.

Since it would have less impact if an adversary gained unintended access to the code for assessing an assignment we don't assign it the same sensitivity. Getting access to the assessment code would only tell you what kind of output that your assignment should produce in order to pass. It would not tell you how to make the actual assignment. Thus, an assignment that is "tuned" to produce the output or contain the phrases that is checked for would not pass subsequent human inspection. A possible exception to this occurs should this system ever be used to correct assignments with a single correct answer. In this case, getting hold of the assessment code equates acquiring the answers for the assignment

Assessments generated by the system is the least sensitive data being handled since the, a) do not identify students by name and b) TODO

Because of the data that we handle, securing assignments from unauthorized access is our primary priority.

FiXme Note:  
Is it always  
the case that  
they do not  
identify  
students by  
name?

## 2.2 Personas

### 2.2.1 Student

TODO: We assume that students can not be trusted

### 2.2.2 Teacher

TODO: We assume that teachers can be trusted

## 2.3 Assessment flow

A submission goes through to phases of assessment; static and dynamic testing.

The static testing phase assess the static properties of submissions. In this phase, submitted code should never be executed

The dynamic testing phase will actually execute student submitted and compare its responses against a set of included reference answers.

### 2.3.1 Rationale

The reason for this split is that we can let two different security models govern the two phases.

In the static phase, evaluation programs are given complete access to the submitted source code. There is no technical limit in this phase preventing the testing scripts written by teachers from being executed. Furthermore, accidental student code invocation can also arise from vulnerabilities in utilities that is invoked as part of the assessment process. An example of such a vulnerability is the buffer overflow that was triggered in the `strings` utility by using it on certain ELF files [5].

Since this phase is sandboxed, no risk is posed to the wider system if malicious student submitted code is inadvertently run. but assess

Fixme Note:  
this makes  
no sense

In the dynamic phase both the student submitted code and the assessment code is run, but in two different sandboxes with the only interaction between them being pipes. This isolates the assessment code and

## 2.4 Submission lifecycle

In this section we describe the life cycle phases which an assignment goes through from submission to the assessment being returned to the student. The life cycle of a specific assignment depends on several factors. Particularly if we are dealing with an identified or anonymous assignment.

## 2.5 Submission constituents

## 2.6 Executing untrusted code

If the OnlienTA system is to fulfill its intended role, it is a necessity that it executes arbitrary code submitted by students. . For obvious reasons, we cannot trust student submitted code not to perform inappropriate actions, may these be unintended side-effects of programming errors or deliberate malicious attempts to exploit the system.

Fixme Note:  
rewrite

### 2.6.1 Sandboxing

We will handle this challenge by executing the student-submitted code in sandboxes where they have no chance of inducing side-effects on other parts of the system, gaining access to privileged data or affecting the assessment of other students assignments.

Sandboxing technologies range from fully-flgedged virtual machines to rule-based execution environments for programs. We will here focus on containers which is a type of OS-level virtualization and resource confinement which by enforcing separate namespaces for e.g. networking, filesystems and system permissions. The Linux kernel provide support or this kind of sandboxing through the feature sets namespaces[**namespaces** ], cgroups, rlimit and seccomp..

There are several well-known "packaged" implementations OS-level virtualization for Linux which is implemented using the namespaces features. In this section we will comparatively evaluate some of these and their suitability for our purpose.

#### Docker

Docker is a relatively recent addition to the family of application containerization tools. Opposed to most other OS-level virtualization technologies it focuses on application isolation rather virtualization of an entire operating system. It has popularized a new way of deploying and executing applications where, instead of applications being installed on a shared system, applications are deployed by starting a container which includes the application, its dependencies and its configuration. [6]

Docker comes with a complete set of tools for building and managing containers. These could be useful in supporting the TA's job of developing and testing assessment units for assignments.

On the other hand, the codebase of Docker is very large and contains a lot of features that are superfluous for our use case. A large codebase also has a higher chance of containing security critical bugs. The widespread adoption of docker for security critical applications, however, mitigates this concern somewhat.



Privilege escalation vulnerabilities has historically been more likely to arise from peripheral feature sets (as seen in the recent QEmu floppy driver vulnerability [CVE-2015-3456]), rather than the core containment functionality itself.

### Sandstone

As part of a related student project, a minimalist and modular interface to the Linux container APIs were developed. The philosophy behind this system, named sandstone, is to provide access to the sandboxing features of Linux as small individual building blocks which can be composed, providing the program executed with the desired level of isolation.

The building blocks of Sandstone is implemented as small programs written in the C language, each of which implements a part of the Linux container system. The programs are composed through command chaining. Exemplified, the composition works as follows: Let  $a$ ,  $b$ , and  $c$  be programs in Sandstone and let  $p$  be the program that we wish to execute in a sandboxed environment. We execute the process with the command line  $a\ b\ c\ p$ . Then  $b$  will be constrained by the isolation provided by  $a$ ,  $c$  will be constrained by  $a$  and  $b$  and  $p$  will be constrained by  $a$ ,  $b$  and  $c$  and will thus be executed with the level of isolation that we desired. [1]

There are good reasons to be wary of custom built security critical utilities. Security is hard to get right and custom built utilities that never see widespread use will never go through the public scrutiny offered to more publicly visible tools. A mitigating feature of Sandstone in this regard is its relative simplicity and the limited amount of code used in its implementation.

### Conclusions

The design that we propose later in this report will provide an

## 2.7 Design principles

We have devised a number of principles that our system design will be based on. These principles aims to simplify the design of our s

**Modularity** We want the components of our system to be modular and loosely coupled. Large monolithic components tends to foster unmanageable complexity which makes them harder to maintain and review for security problems. Modularity also adds flexibility to our system by enabling certain components to be swapped.

**Clearly defined minimal interfaces**

**Minimum permissions** A common "best practice" related to the security aspects of software development is to assign a minimalized set of permission to a component required by the work that it needs to perform. Traditional Unix environments only makes the distinction between a superuser (root) and a regular user. It's common for applications to be started with full superuser permissions and then, after performing privileged actions, resume operations as a unprivileged user.

The sharp separation between superusers and regular users no longer exists in model versions of the Linux kernel following the introduction of the *Capabilities* interface [7].

Several operations

**Data ownership separation** The most basic permission management primitive in Unix operating systems is the concept of a user. We wish to enforce fine grained user separation across the system such that a given user only have access to what is absolutely necessary. A common concern of executing applications inside a container is what happens in the unlikely event that a process successfully manages to escape the container. In this case, the attacker would usually end up having the same permissions as the process executing the container.... TODO

An important advantage of linux capabilities is that they follow a *process* and not the user. This means that an attacker would need to exploit, either So even if an attacker manages to escape our container, he would

### 2.7.1 Securing special capabilities

Despite our best efforts, we can never completely eliminate the possibility that an adversary will be able to take advantage of the capabilities given to our programs and force them to preform unintended actions. For instance, gaining arbitrary code execution in a process with the *chown* capability would allow one to change permissions of any file in the system. Since our security model to a large extent relies on users having access to a minimalized set of files, we would consider this to be a rather serious security incident.

Secure Computing (seccomp) [manpage] is a facility provided by the Linux kernel to provide application sandboxing through the filtering of syscalls. It allows an application to unidirectionally transition into a secure state where all of its syscalls are filtered by specified rules. Building on our example in the previous paragraph, we can use the seccomp to filter the *chown* syscall such that it only allows changing the owner of files to UID within an allowed range, e.g.  $(aaa000)_{36} - ((zzz999)_{36})$ .

We can use seccomp to build an extra layer of security around our programs which limits the effects of

FiXme Note:  
discuss  
problem of  
go spawning  
multiple  
threads per  
default  
somewhere

## 2.8 Handling user identities

We will support assessing submissions from two groups of user: anonymous and authenticated users. Submissions from these groups will be handled in two different

ways, both with regard to what services are offered to the user, but also how we represent their submissions internally. Submissions made by an authenticated user will be referred to as an *attributed* submission.

Anonymous submissions are removed after their life cycle has concluded. Only the assessment of the submission persists until it has been read by the submitting user.

### Identified submissions

For the time being, the scope of OnlineTA is limited to students at DIKU. Thus, we can take advantage of the standardized usernames assigned to everyone who is in some way associated with the University of Copenhagen (KU-usernames), both students and employees. KU-usernames are matched by the regular expression  $[a-z]\{3\}[0-9]\{3\}$ , with the exception that vowels aren't used as letters in generated usernames. We can use this username generation scheme to our advantage since we can interpret the KU-usernames as base 36 numbers and thus achieve a stateless and bidirectional mapping between Linux UIDs and KU-usernames, e.g.:

$$(\text{gfr534})_{36} = (993919360)_{10}$$

Since UIDs in Linux are represented as a 32-bit signed integer, actual implementations of this scheme needs to take into account that the (numerically) largest valid KU-username  $(\text{zzz999})_{36}$  is larger than  $(2^{31})_{10}$ .

**Verifying user identity** Another ongoing student project are designing and implementing a purpose-built GPG key server which are intended to be implemented into the OnlineTA system

## 2.9 Interacting with the system

In this section, we describe modes of interaction with the system for making submissions and receiving assessments.

Since this system is intended to be used even by freshmen submitting their first assignment, providing a simple, familiar, universal and immediately accessible interface for submitting assignments is an important goal. However, a secondary purpose of this system is, not just aiding TAs in their grading work, but also to provide students with continuous feedback while they are solving their assignments. It's therefore important that we meet students where they are and provide them with (from their perspective) the most seamless interaction possible.

The first goal is best achieved through a well-designed web interface for uploading assignments. The interface should, when possible, provide immediate feedback within the browser. As previously mentioned, we have based our user identification on verifying that an assignment has been signed with the GPG-key belonging

FiXme Note:  
We should  
hold off  
describing  
our imple-  
mentation of  
handling  
usernames  
through

FiXme Note:  
example?

FiXme Note:  
Talk about  
when system  
overload  
causes long  
submission  
times

to the user claiming to have submitted the assignment. This poses a potential obstacle when implementing authenticated submissions through the web interface. One option, is to require students to GPG sign their assignments prior to submission... Talk about WAYF, browser certificates, etc...

Many additional submission interfaces has been proposed since the conception of OnlineTA. The prevailing proposal has been to implement a command line interface which would allow a student to submit the current folder or a file for assessment. The Windows equivalent of this would be a Shell Extension adding tantamount functionality to the right click menu of files and folders. We are not qualified to provide equivalents for other platforms such as OSX, but in that particular case, the command line interface would also be feasible. A tangential advantage of this submission method is that it could be implemented to only transmit deltas between repeated submissions and thus significantly reduce upload time and internet bandwidth required for submission.

Another submission interface under consideration is Git. We consider the addition of

## 2.10 Large Submissions

Some courses may require large submissions which could saturate any feasible internet link through only a limited number of concurrent uploads. We propose that the upload bandwidth requirement can be reduced by applying a delta transfer algorithm on repeated submissions.

The rsync algorithm is an obvious candidate for solving this problem. However, the algorithm assumes that the receiving end has access to all files. Honoring our property of minimal user permissions the submission receiving process won't have access to the files of the previous submissions. We therefore propose a variation of the rsync algorithm built for our purpose. Since we cannot rely on being able to retrieve the state of the remote files we must be able to keep the difference between local and remote files locally. TODO

## Chapter 3

# System Architecture Design

In this chapter, we propose a design for our system. We will use the principles that we laid down in the previous chapter and present them as a part of an integrated system

### 3.1 General concepts and overall goal

With security as the, as previously stated, principal goal, we focus our design around several small processes, each with a well defined purpose and interactions. We furthermore enforce a principle of minimum permissions across our entire system.

#### 3.1.1 Container support

The design of the system enables the containerization model to be swapped, such that different containers can be used based on

### 3.2 Users

There are two levels of users in the OnlineTA system. One is the user owning the daemons responsible for handling the main assessment flow, and the other is the users owning the submissions. In the case of anonymous submissions

#### 3.2.1 Representing users on system

Name service lookup services in GNU/Linux systems are provided by the NSS (Name Service Switch) subsystem which is implemented in the Gnu glibc library. Lookup databases include users, groups and DNS names. The NSS system provides a unified interface for performing lookups in a database backed by multiple services. The NSS system is extensible through services which is called in the order specified by the `nsswitch.conf` file. In the OnlineTA system, we add a password mapping module which can be used for looking up user names to UIDs and vice-verca[8].

This allows us to represent our dynamically created users on the Linux system and thus avoid having to define them statically in the `/etc/passwd` file.

We perform user handling in two separate NSS modules for handling named and anonymous submissions respectively. The usernames of identified submissions are assigned by a NSS module which implements the bidirectional UID to KU-username mapping previously described. conversion scheme previously described.

We handle the usernames of anonymous submissions slightly differently through a different NSS module. This module works on lists of preallocated users and is autonomously responsible for maintaining lists of allocated and non-allocated users

FiXme Note:  
rewrite

### 3.3 Components

Our proposed design for the OnlineTA system is split onto a number of different components which are described in this section. Their relationship is shown in [figure 3.1](#).

**Frontend** The frontend is responsible for accepting submissions. In case of attributed submissions it is also responsible for verifying the identity of the user submitting the assignment. It will also perform the initial storage of the submission. Furthermore, the frontend is responsible for returning the assessment to the student.

It will go through the following steps in order to accept an assignment.

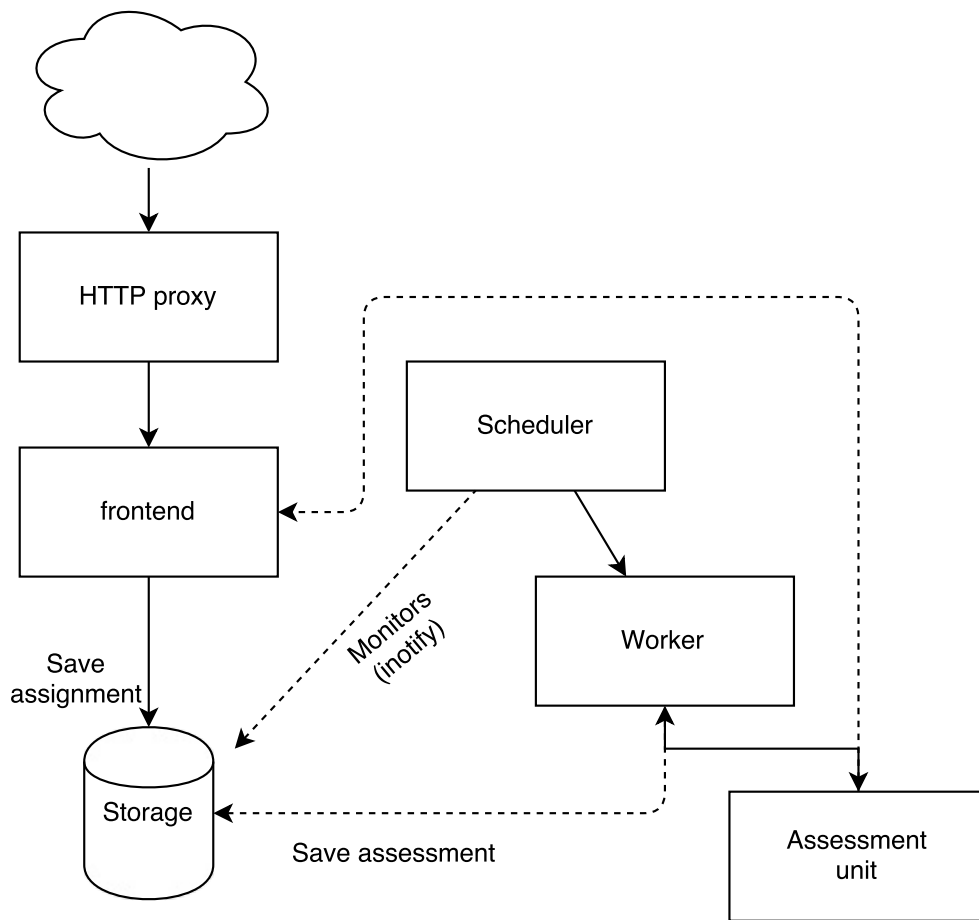
The frontend has the following roles

- Accepting submissions from the HTTP proxy
- Verify GPG signature of assignment to verify submission attribution
- Generating and assigning a UUID to the submission
- Writing the initial metadata file
- Creating directory for containing the submission and setting ownership of files

This component require the `CAP_CHOWN` capability.

**Scheduler** The scheduler will pick up the assignment after it has been saved to disk and pair it with a suitable assessment unit. Assignment scheduling is based primarily on a priority assigned to the submission by the frontend. Submissions with equal priorities are scheduled on a FIFO basis. If no priority is provided by the assignment, it could assign a priority based on a number of factors, e.g. time remaining before submission deadline. The scheduler directly spawns processes for executing assessment units.

The scheduler has the following roles:



**Figure 3.1** – Flowchart showing interactions between the components of the On-lineTA system

- Monitoring the incoming directory for new submissions
- Pairing submissions with an appropriate assessment unit
- Scheduling and prioritizing assessment of assignments
- Invoking workers for performing the actual assignment

**Worker** A worker process is spawned by the scheduler and its overall purpose is to set up and initiate the assessment process. It is initially started as the `onlineta` user but after the assessment environment has been configured and the containers has been created it will suid to the user owning the submission and shed its special capabilities.

Specifically, its roles are the following

- Setting up the environment for performing an assessment including mounting filesystems
- Setting up containers for assessment
- Relaying assessment to frontend

The special capabilities required by the Worker process are `CAP_SYSADM`, `CAP_SUID` and `CAP_SETPCAP`

**Assessment units** An assessment unit handles the actual assessment of an assignment... TODO

### 3.4 System flow

Describe what happens after a student submits an assignment, that is, what is the individual role of each of the components and what is the

### 3.5 Interactions

The scope of the current implementation is limited to supporting direct interactions via a web browser, however, the additional modes of interaction previously described can be implemented on top of the interface that we will describe in this section.

In the current design, all interactions with the system occurs over a secure HTTP connection. We use the first part of the request URI to determine which action is requested by the user while subsequent URI components are used as parameters for that request. The system supports the following actions:

/ This URI can be issued as a HTTP GET request and will return the main index page which will contain a list of courses and assignments available on that server

/submit/<course>/<assignment>/ When a HTTP POST request is issued to this URL it will be handled as an anonymous submission. If the submission is accepted, the frontend will respond with a page showing the assigned submission UUID. This UUID can then be used in order to retrieve assessments once they are completed

/submit/<course>/<assignment>/<KU-username> This request type is the same as the previous except that it will perform an attributed submission.

/query/<uuid> Used to query assessment status of assignment identified by given UUID. The status of an assignment can be the following ACCEPTED The





### **3.7 Interfaces**

### **3.8 Scaling and distributing**

A driving idea behind our design is to enable horizontal scalability. A key element to achieving this

## Chapter 4

# Measurements

A goal of this project is to determine the server capacity required by the OnlineTA system. Since the current implementation is not yet at a state where we can reliably perform such measurements, we will instead focus on describing measurements which can be performed on the final implementation in order to assess server capacity requirements.

### 4.1 Metrics of submission assessment

In order to know what benchmarks to conduct we need to establish the performance metrics that we can use to estimate the expected performance of our system.

### 4.2 Performance thresholds

In order to prevent making overestimated hosting recommendations it's important to establish acceptable thresholds for assessment throughput. A system that is scaled toward delivering assessments "as fast as possible" under any circumstance will have significantly different requirements than a system where we accept queuing of assignments and thus delaying assessments during peak load.

The perceived usefulness of the system and overall student satisfaction will depend greatly on striking the right balance between acceptable assessment throughput thresholds and cost of hardware deployments.

### 4.3 System scaling requirements

As our initial scope is limited to performing assessment of the assignments of DIKU courses, we can develop an idea of how large our assignment throughput needs to be fairly easily. E.g if we have 100 students enrolled in the introductory programming course, we can calculate an upper bound on the amount of resources consumed. This can be done by calculating the expected resource requirements for each assignment in the course. Our required throughput is then

## **4.4 Estimating static requirements**

We define static requirements as resources that scales linearly wuth an assignment, e.g. if we need to estimate the requiried

## Chapter 5

# Discussion and conclusions

We will end this report by presenting our conclusions and discussing the wider implications of our proposed design.

### 5.1 Practical deployment considerations

In this section, we will devote some space to consider the practical and economical implications of the possible hosting choices that we have for our design.

It may seem that a cloud hosting provider such as Amazon or Digital Ocean would provide the ideal environment for hosting OnlineTA,

# Bibliography

- [1] Oleksandr Sturmov. “Online TA”. Master Project. 2014. URL: <https://github.com/oleks/onlineta/raw/master/report/output/onlineta.pdf> (cit. on pp. 3, 8).
- [2] University of Copenhagen. *Fremdriftsreform (in danish)*. 2014. URL: <http://nyheder.ku.dk/fremdriftsreform/> (visited on 06/19/2015) (cit. on p. 3).
- [3] Reva Freedman, Syed S. Ali, and Susan McRoy. “Links: What is an Intelligent Tutoring System?” In: *Intelligence* 11.3 (Sept. 2000), pp. 15–16. ISSN: 1523-8822. DOI: [10.1145/350752.350756](https://doi.org/10.1145/350752.350756). URL: <http://doi.acm.org/10.1145/350752.350756> (cit. on p. 4).
- [4] Petri Ihantola et al. “Review of Recent Systems for Automatic Assessment of Programming Assignments”. In: *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*. Koli Calling ’10. Koli, Finland: ACM, 2010, pp. 86–93. ISBN: 978-1-4503-0520-4. DOI: [10.1145/1930464.1930480](https://doi.org/10.1145/1930464.1930480). URL: <http://doi.acm.org/10.1145/1930464.1930480> (cit. on p. 4).
- [5] Murray McAllister. Oct. 26, 2014. URL: [https://bugzilla.redhat.com/show\\_bug.cgi?id=1157276](https://bugzilla.redhat.com/show_bug.cgi?id=1157276) (visited on 06/21/2015) (cit. on p. 6).
- [6] Docker Inc. *What is Docker? An open platform for distributed apps*. 2015. URL: <https://www.docker.com/whatisdocker/> (visited on 06/10/2015) (cit. on p. 7).
- [7] The linux man pages project. *capabilities(7)*. May 7, 2015. URL: <http://man7.org/linux/man-pages/man7/capabilities.7.html> (cit. on p. 9).
- [8] GNU. *System Databases and Name Service Switch*. 2015. URL: [http://www.gnu.org/software/libc/manual/html\\_node/Name-Service-Switch.html#Name-Service-Switch](http://www.gnu.org/software/libc/manual/html_node/Name-Service-Switch.html#Name-Service-Switch) (visited on 06/18/2015) (cit. on p. 12).