Further development on the Moodle CodeHandIn Package

By Samuel T. Deane

Flinders University Masters Information Technology Thesis

Academic Supervisor:

Associate Professor Paul Calder

Commenced: 28th of July, 2014 Submitted: 10th of November 2014

Submitted to the School of Computer Science, Engineering, and Mathematics in the Faculty of Science and Engineering in partial fulfilment of the requirements for the degree of Masters of Information Technology at Flinders University – Adelaide, Australia

**Abstract**

The benefits of Automated Assessment of student programming assignments are clear - decreasing marking time, removing and hinting at common errors; thus freeing up tutors for more difficult questions; supplying speedy results to enhance and guide student learning. It is with these aims that an Automated Program Assessment System (APAS) for the Moodle Learning Management system called CodeHandIn was developed at Flinders University in 2013 by Jonathan Mackenzie, with the goal of introducing the module in future programming topics at Flinders University. This paper re-examines the literature with a focus on Moodle APASs, details how the project has been extend and refactored to be form the CodeHandIn Package and introduce a NetBeans Plugin for CodeHandIn as well as integration with an external Compiler controller.

Keywords: programming, Automated Assessment, Moodle

# Declaration

I certify that this work does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signed

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date

**Contents**

[Declaration 3](#_Toc403371464)

[Executive Summary 5](#_Toc403371465)

[1 Introduction 5](#_Toc403371466)

[2 Literature Review 6](#_Toc403371467)

[2.1 Early Automated Assessment 6](#_Toc403371468)

[2.2 Modern Automated Assessment 7](#_Toc403371469)

[2.3 Moodle Automated Assessment Programming Modules and Plugins 9](#_Toc403371470)

[2.4 Aspects And Tools Of Automated Programming Assessments 11](#_Toc403371476)

[2.5 Current Flinders Systems 15](#_Toc403371480)

[3 System Design 19](#_Toc403371489)

[3.1 Architectural changes over the previous project 20](#_Toc403371490)

[3.2 System Components 21](#_Toc403371491)

[4 System Development 29](#_Toc403371497)

[5 Remaining Work 30](#_Toc403371498)

[6 Conclusion 31](#_Toc403371499)

[7 References 33](#_Toc403371500)

[8 Appendix 36](#_Toc403371501)

[8.1 Use Case diagrams form the previous project 36](#_Toc403371502)

[8.2 New CodeHandIn Screenshots 38](#_Toc403371503)

# Executive Summary

This thesis is presented for the fulfilment of the Masters of IT at Flinders University, Adelaide South Australia, Australia. It includes a literature review covering an introduction and history to Automated Programming Assessment Systems (APAS)s with a specific focus on APASs developed for the Moodle Learning Management System. It also summarises the Moodle APAS prototype entitled CodeHandIn previously developed at Flinders University; the rational and aims of this project; the continued development and move to a plugin rather than module based development; its outcomes and remaining work. In addition this thesis included a brief user guide to aid installation and use of the system as it stands as well as information aiding any generic Moodle plugin development.

The latest version of the source code is available at <https://github.com/SamSE3>.

For additional information on the Moodle platform please see [https://Moodle.org/](https://moodle.org/)

# Introduction

Students learning to program are often instructed to write programs that produce a predictable output based on well-defined input. This enables tutors and assessors to quickly and efficiently evaluate student programs by comparing the produced output against the expected output. Whilst marking a program that uses predictable I/O makes it significantly easier to do, manually assessing even a single submission is an especially time consuming and tedious process.

As such, academic institutions around the world have employed and continue to employ the use of Automated Assessment System in their courses to evaluate the functionality of student programs. Furthermore, systems have evolved to allow students to repetitively submit their programs and to drive their development through the extended feedback provided from the Assessment System. This has had the net effect of freeing tutors from having to answer repetitive questions concerning common problems giving them more time to spend on the difficult ones.

These systems operate in a similar way to manual systems allowing students to submit their programs through a command line or GUI interface which then stores them on a file system that can be later accessed by the assessors. Rather than the assessors downloading the files to their machines, compiling them and testing them by manually feeding the program’s input and checking what they return. The system itself directs assessing of submissions using a sandboxed compiler and runtime service that protect against malicious, malformed and unresponsive code to compile, runs a series of test on each submission checking the output is equivalent to what is expected and lastly saving the sum of the marks of the tests that passed to a gradebook.

Currently there is no one system used at Flinders University for the automated assessment of student programs, creating a disparity between topics and adding an unnecessary learning curve for students. This is just one of the problems with the current system used at the University. As such, an honours project was conducted in 2013 by Jonathon Mackenzie that produced an initial prototype of an Automated Programming Assessment System (APAS) that could be used at Flinders University. The System developed entitled CodeHandIn, as in code-hand-in, integrated into the Moodle Learning Management System (LMS), the main LMS used at the University, called Flinders Learning Online or FLO.

This project continues the development of the CodeHandIn System, through:

* revised literate review that includes the general history of APASs and a focus on those specifically developed for the Moodle LMS,
* a new architecture that is more adaptable and in line with the Moodle Philosophy and
* a new Client program and NetBeans to connect to CodeHandIn externally and
* several functional improvements over the previous prototype.

# Literature Review

## Early Automated Assessment

The idea of testing the output of program based on some predefined input is not new, with some of the earliest teaching examples dating back to 1960 [1]. These early systems utilised punch cards to define an assembly language program that could be automatically tested using a separate grading program, called the ‘grader’, specific to each assignment. The grading program operated by returning ‘program complete’ if the program compiled and passed all the tests or ‘wrong answer’ if it failed a particular test, stopping further computation. Over the 15 months the testing method was run, the economic benefits were significant, reducing tutor time by a third whilst student computer time was reduced by an eighth. The latter being especially well received as all tests and labs were run on a single machine. This allowed enrolments at the school to increase, creating a larger student programming body, though it was noted that this resulted in less time for the students to interact with the computer.

A similar system developed at Stanford the following year used a grader that had predefined input, kept track of runtime and utilised an area to store grades in what would now be referred to as a grade book [2]. By 1969 the complexity of the grader program had jumped again with the grader program running on a modified operating system and compiler defining what could be called the world’s first sandbox [3]. These changes allowed greater reporting capability delivering more detailed test results including some limited feedback on plagiarism. Though perhaps the greatest benefit, as seen by the developers, was its sandboxing ability, to protect itself against malicious code.

These early systems were written in low level code often designed for a single purpose with a specific set of tests and as such required a high degree of expertise to develop and in some cases use. Though by the late 1980s a new type of automated programme assessment tool was being developed. These arose from the continued development of common utilities and toolsets often embedded in the operating system removing the high level of expertise required to write a testing program. Such systems were labelled as the first ‘tool oriented’ or ‘tool based’ systems allowed the teacher to both grade the functionality of the submission and the source code style though a GUI or command line interface [4, 5].The grader was script based, compiling submissions placed in various directories under different student names, running them against predefined sequences of input and outputting the results to individual files that the assessor could view. The grader system also, like its 1969 predecessor, included the ability to recover from bad or malicious code.

In the same year a system with a different take on automated assessment was introduced. The system, entitled TRY shifted the paradigm from one of teacher initiated automated grading assessment to one of student initiated automated progress assessment, creating a toolset that allowed students to test their own programs in their own time [4]. To ensure that students were really thinking about the content of their submissions the number of possible tests was limited for each student. Additionally it was noted again that running code on a live shared system could potentially be disastrous.

Another system that was designed around the same time, called the Ceilidh courseware system, expanded the paradigm of online learn even further handling courses, lecture notes, grade books and related data defining what is now called learning management system (LMS) or Course Management System (CMS) as it includes both content management and authoring systems [6]. This was in addition to its core purpose of providing a system to facilitate the assessing of student submissions, which it did from development, through execution to grading, commenting and administration.

1994 sore a novel development that defined testing facilities for Matlab and Maple that employed the use of socket connections as a means to map submissions to different processes. The system then evaluated submission in a similar manor to previously described systems [7].

The ASSYST system of 1997 took student based automated testing and AA testing one step further [8]. They implemented features such as weighted tests, automated style and complexity metrics as well as handling final submissions defining what is now called an assessment management system (AMS) that included inbuilt functionality for commenting, review and other administrative assessment functions.

## Modern Automated Assessment

With the Internet being the paradigm shifting force that it has been, many of the testing programs became web oriented, interfaced by GUIs to both display results and create assignments (doing away with the scripts and command line methods of their predecessors). This also fired the dramatic increase in development in Web technology and the companies surrounding it, giving AA developers increasingly better tools allowing them to design ever more sophisticated testing approaches.

One of these third generation Automated Assessment systems is CourseMaker, the successor of the Ceilidh courseware systems [6]. The system was a more complete LMS than anything used before defining users by privileges and expanding the idea of a managing all course related information and exercises. This includes wizards and other utilities to handle creation, editing and deletion of courses, users, administration statistics and assignments of various types. Of its many ground breaking features it included a multiple choice assignment type and a new assessment system entitled ‘diagram’. This tweaks the focus of previous AA systems to that of design rather than pure functionality.

It achieved this through analysis of a programs lexical structure, typographic layout, complexity and use of data structures, dynamic operation using test data, efficacy through its execution time and its feature set. This was achieved all the while supporting multiple languages from FORTRAN, SQL to Java and C. It achieved this using a mark-action file which like the script files of previous systems allowed the assignment setter to define the marking ratio for the assignment and how some of the aspects would be performed.

The feedback for a tested submission is also richer than previous systems, presenting a feedback tree to the student, outlining where marks were lost in the assignment and avenues for improvement including links to related information.

Another AA LMS similar to CourseMaker is the BOSS system [9, 10], though originally a local program, it added a web server to turn the traditional system into one of a web system. Initially this was focused on giving tutors access to submissions without having to access a machine locally but has since been extended to allowing students to externally submit and view their course information. Another notable change was the inclusion of limited analysis for plagiarism detection.

RoboProf was a Java-oriented assessment system that presented programming questions to students in a quiz style rather than a single assignment [11]. The system was not a LMS but single purpose system to give honours degree students an entry into programming providing 39 distinct problems of increasing difficulty or levels covering the use of primitives, variables, arrays, program flow to the use of data structures.

The Automated System for the Assessment of Programming (ASAP), developed at Kingston University in the United Kingdom took a more modular approach, itself part of a larger distributed system that included a proprietary gradebook for tutor review and a marking web service that acted as a go between [12].

ECAutoAssessment was similar to the Automated Programming Assessment in CourseMaker and the aforementioned ASAP integrated with an existing LMS, called Plone. Though technically an open source Content Management System, it was tailored to fill the role at Magdeburg University in Germany and in many other universities [13]. The ECAutoAssessment system was different from previous systems in that it aimed to support a wide range of programming languages and also integrate additional non-standard testing methods and types such as a short text based question and answer system. It was accessed through two separate interfaces added into the Plone LMS, one for teachers to create assignments, define tests and related testing schemas and the other one for students to submit and view the results of their submissions.

Another recent system was a plugin for Web-CAT automated programming test system [14] that was specifically designed to promote Test Driven Development (TDD) at Virginia Tech [15]. The reason being that TDD forms the backbone or is used in conjunction with several eXtreme Programming (XP) and Agile Devolvement Methodologies. Such methodologies utilise teams of programmers driving development forward with less documentation than a standard waterfall based development methodology. These development strategies have been shown to benefit documentation and maintenance, reduce the number of bugs (especially those that arise under more complicated conditions) as well as giving programmers a guide to follow, boosting their confidence, speed, accuracy and sense of accomplishment in the passing of tests.

TDD is a development methodology that specifies that one or more test should exists for each feature or functionality. This has the effect of mirroring or replacing part of a Software Requirements specification such that the tests drive the requirements rather than documentation. When TDD is strictly applied no functional code should be written before a test or if code is written. If code is written, it is assumed to be wrong until corresponding tests are written proving it is correct. As such, the idea of Automated Assessment that scales in complexity as the software develops is essentially the testing framework used in a TDD project.

The WebCAT plugin is based around an adaptive feedback system that uses the correctness of the solution code and the coverage of the test code to tailor a message to a student. If TDD practices are adhered to, the message acknowledges the use of good TDD practices and provides hints for additional progress in the form of hint credits. Otherwise the message encourages the student to go back towards a TDD approach without providing any additional hint credits.

Though the use of the system over a semester did not show significant improvement in adherence to TDD when compared to a previous non WebCAT semester for the same topic, the developers noted that the system still showed promise and could benefit from improving its persuasiveness and introducing TDD earlier in a course and sticking to it in a form of long term persuasion.

Another recent system used in MIT courses gives automated feedback about errors in student submissions [16]. The system does not aim to synthesis a correct program based on a rough outline submitted and use this solution to give further hints to match. Instead, it uses a rule-directed translation strategy where an error that meets a specific criteria will trigger a hint to be appended to the student’s feedback.

## Moodle Automated Assessment Programming Modules and Plugins

### Introduction to the Moodle Learning Management System

Several authors ([17-19]) have stated that any new Automated Program Assessment System (APAS) should be integrated into a Learning Management System (LMS) sometimes called a Virtual Learning Environment (VLE). This will provide a single common interface and integrate with any other resources already provided for the students. Taking this information on board, the Moodle LMS was chosen to be the target development platform as it is the predominate LMS used at Flinders University (called Flinders Learning Online or FLO).

Moodle stands for Modular Object-Oriented Dynamic Learning Environment. As such it is a single, robust, secure, modular, open source Learning Management System (LMS) that aims to make course resources accessible to as many students as possible. The system is written in PHP, underpinned with a common SQL database (Apache, PostgreSQL/MySQL/MariaDB) [20]. It currently has 54,372 registered sites in over 230 countries around the world and aims to be highly adaptable and configurable whilst being simple to use. [21]

It is thus relevant to cover the past and present system developed for Moodle that attempted to provide some form of Automated Assessment for a student program.

### CTPracticals

CTPracticals was a Moodle plugin that verified Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) assignments [22]. Submissions were made in groups through a Moodle interface, a verification engine checked the submission against the specification stated by the teacher and returned the mark, statistics and relevant information to the student. As such this is not a repeat submission type. There is only one submission for each activity made by one member for a group that will cover work detailed in one or more practical lessons. The student’s submission is made in the form of a zip file of CAD circuit schematic that is then checked to see if their tolerances and design are matched by the specification. The system was later upgraded to include support for MatLab code assessment that followed a similar architecture though it was more aimed towards submissions for a single practical rather than major assignments.

### Online Judge 1 & 2

Online Judge was one of the first Automated Programming Assessment plugins for Moodle. It is built as the name suggest, around the Online Judge library, which is a compile and testing service suite originally used to test submissions for online programing competitions. The other half of the plugin is a more conventional Moodle plugin that provides a user interface for students to submit their submissions through Moodle. Interestingly it, unlike other APA Moodle plugins, borrowed many of the features of the Moodle file assignment type. [23]

There are two main Judge sandbox engines:

* Libsandbox, an open source C/C++ and Python profile and watchdog program [24]; and
* Ideone, now called Sphere Engine, allows remote secure execution of source code for more than 40 programming languages. Ideone did supply 2000 free submissions per month but it appears they have since removed this offer. [25]

The system was used in C programming courses for Computer Science and Software Engineering students at the Harbin Institute of Technology, China. At the time of use the Institute had 2500 students enrolled in C programming topics, equating to approximately 400 students per lecturer who have 1-4 tutors. Prior to 2006, students submitted only a few programs and only a fraction of each assignment was marked by tutors to reduce their workload. The introduction of the Online Judge system greatly reduced the workload on tutors. One of the lecturers gave the students an infinite amount of assignments, awarding 100 marks to the student who completed the most over the semester.

In case studies, Online Judge brought significant improvements, with the average grade of 4 previous semesters without Online Judge 70.2 whilst after the introduction of Online Judge the average grade of the three following semesters was 89.6, 86.6 and 83.7 with the difficulty gradually increasing beyond that of the previous semesters. Unfortunately, plugin development or at least the international open source version appears to have been stopped, with no activity in the plugins main source code repository for the past two years.

### VPL, Virtual Programming Lab

Virtual Programming Lab, VPL, is perhaps the most significant APA module for Moodle to date. The system, besides being able to test user submitted assignments, allows: [26]

* In-browser editing of user programs removing the need for external editors;
* Interactive programs can be run in the browse through a console, enabling programming from any device with a web connection and internet browser;
* Integrated plagiarism detection through a system that checks for similarity. Code may include a watermark to mark authorship and can prevent copy and pasting of code from other source;
* Supports a wide range of programming languages (Ada, C, C++, C#, Fortran, Haskell, Java, Matlab/Octave, Pascal, Perl, PHP, Prolog, Python, Ruby, Scheme, SQL and VHDL);
* The uses of scripts and or programs to access submissions;
* Controlled program execution in terms of resource such as compile/runtime, memory size, file size and process count; and
* The use of a variable test description that can differ for teachers and students.

The system is open source under the MIT License, actively distributed and is still being maintained. Its most recent version has been developed to run with Moodle version 2.3 and higher and has older versions that date back to Moodle 1.4.5, indicating that it has been continually developed and improved over a period of several years.

Despite an extensive search, no papers detailing the VPL system written by the developers were found. The search did reveal some papers that mentioned the use of the system in relation to some other form of research [22]. There is also only a limited amount of documentation beyond that of the features list making it difficult to review the underlying system. The support is also not particularly good form the original authors so some users have taken upon themselves to write their own tutorials for several projects [27].

### CodeRunner

CodeRunner is a Moodle quiz question type that runs a serious of tests on a student-submitted program in a sandbox [28]. All code is written through the browser and tested based on its outputs for a series of inputs. The system, in conjunction with the past versions (pycode and ccode), has been used at the University of Canterbury, New Zealand, for the past four years. In that time it has tested tens of thousands of student submissions. The system supports C, Python, Matlab, Octave, Java and Coljure programming languages and, like other previously mentioned systems, is expandable.

CodeRunner provides a significant amount of Documentation on how to setup the plugin Setup and its general use making it more compelling than VPL [29]. For these reasons it is currently being looked into by NetSpot, the host of Flinders’ own Moodle, FLO, to be integrated into future releases.

## Aspects And Tools Of Automated Programming Assessments

From the systems described in the previous section and reviews and surveys conducted by [12, 17, 30-32] the features and elements of a well-developed Automated Programming Assessment System should include:

### Internal components

#### Testing paradigms

It is important to consider how the assignment it going to be tested. Historically this was done though the software interface defined by the student. This is a simple, effective solution but it relies on the student submissions all having the same interface, something that gets more difficult to resolve as more complex assignments are attempted. To achieve this, more often than not, a student will receive sample code when given the assignment that contains the interface and or stub functions. This gives structure and a framework to assignments which, though beneficial for an introductory topic, might be considered giving the student too much assistance or being restrictive for later topics.

A similar approach is to evaluate the program though some form of input/output interface. The program is compiled and the run through the use of streams such as file I/Os, pipes (i.e. sockets) or sinks as is the case with the Ceilidh system [33]. Unlike the software interface this has the advantage of abstracting the programing language allowing a broader support of languages without significant specification for a set language beyond running a program of that type. This gives greater versatility and the ability to use tests for multiple languages but it is difficult to make the expected vs seen I/O comparison robust enough to ignore irrelevant differences such as extra spaces in output.

These are both empirical approaches in that they only test the interface and not the code from which the interface is derived. There are more detailed measures of functional completeness than a purely interfaced model of evaluation. Additional approaches include some form of code inspection that does not check the functionality of the code but aspects of good programming such as runtime (Ceilidh system), use of resources, commenting and style (BOSS), code coverage (Scheme-Robo), multiplatform support and modular design [11, 34]. The downside is that these are generally more language specific methods of evaluation and require deeper specification than a generic I/O test engine. Furthermore, the subtlety of what defines good reusable software and changes in platform support are difficult to identify and thus evaluate without significant specification.

#### GUI over command line

A review of some of the systems presented in the previous two sections concluded that future AA tools should use a GUI as the predominate interface [19]. This approach being to present a system that is familiar to new students who have grown up using computers through the GUI only. Furthermore, many professional programming tools as Integrated Development environments also are GUI based. Test driven or Test First methodologies are becoming the norm these days and as such there is a need to introduce these development methodologies closer to the entry level - a process that will be substantially easier if a GUI is used.

#### Programming Language support

The system at its core must support compilation of an assignment and any relevant data related to that language. For the system to be fully viable, that is not specific to one topic, it should support more than a single language and include the ability for more languages to be added later on. The support should also not stop simply at understanding the language but provide relevant feedback on common errors with respect to that language i.e. stack overflows, out of range errors etc.

It would also be beneficial if the program could implement a testing program rather than just a script or series of tests. Instance such as testing a server’s functionality and responses using a client would be aptly suited to such a testing framework.

#### Sandboxing of submitted code

The code must be sandboxed. Unchecked code, malicious or otherwise, running on a production machine that students will depend on could lead to system failure stopping all students from checking their work. The idea of sandboxing was one of the first issues surrounding APA, introduced within a few years of the APA systems. Sandboxing is not trivial, as it attempts to provide a level of protection between a systems resource and the programs that run on it. There have been a number of methods of sandboxing from directory segmentation or chroot jail (changes the apparent root directory for a process) based sandboxes, container resource controlled sandboxes such as Docker, partial split virtual operating system such as FreeBSD jail to full virtualisation of operating system [35, 36].

#### Pedagogy: Tests and issues with testing

So far there has been a lot of consideration given to the internal components of what makes up an APAS but what truly makes the system useful is not the architecture but how it is used. To this end it is the tests themselves that will bring forth the benefits of the system. [12, 19]

It has been noted that the tests used in APAS are similar to those in Test drive development [14, 15, 37]. Hence the writing of such tests should employ the use of practices developed in TDD for writing tests. Such practices state that tests should follow the basic principle that they be simple. Simple to write, not taking much time to write, and simple to identify the location and cause of errors and flaws. [38]

It should be also noted that tests can be used beyond the aim of just ascertaining basic functionality but to test extend functionality of a program and introduce levels of functionality. This would allow a student who has put more effort into their submission and taken it to the next level to be rewarded with a higher mark. This will help to discern the better programmers from those who are just in it to pass. [39] Such tests can focus on corner cases and edge cases were transitions occur and subtle problems arise. [40]

This tiered hierarchy of testing will also introduce a guide for students to follow over the development of their program with the specific hints early in development to more broad hints later as the system develops. This model is similar to that of a judge assignment or computer game, giving students a sense of achievement as they progress.

It is important to remember that testing methods described so far are black box in that they do not test anything internal. As such a student could produce a program that returns values based on a decision tree rather than any internal processing creating what appears to be a valid solution to the testing program but is in no way a solution. To avoid such an occurrence, tests need be carefully considered possible with some dynamic component to avoid systems that use pre-determined output.

Another issues with the testing methods described so far is that they don’t test architecture or the commenting code. Such aspects are often overlooked by students but these become the driving factors that allows software to be developed in groups or maintained over long periods of time. To this end numerous sources have suggested adding a manually access ‘style’ component to the final mark. Sometimes the style mark can be as high as 26% to ensure that those getting a distinction or above are commenting their code and thinking about the architecture.

Such aspects tie in nicely with the results of few surveys that have stated that students using the automated assessment system do not mind them over traditional manually assessment but do state they prefer some small oversight from lecturers and tutors. As such oversight gives them real feedback on how well their submitted solution has been implemented.

It has been noted that one of the problems with open APAS system is that they give students the ability to submit an infinite number of times. This can lead students to a trial and error style of development, stopping them from taking the time to understand the program and come to grips with its subtleties. This is not a trivial issue to deal with as limiting the number of submissions moves students away from small change testing, an ideal for TDD.

### Plagiarism detection.

It is evident that by removing the human element from assessing and overseeing assignment submissions makes it easier for students to pass off work produced by others as their own. This is probably the second biggest issue with implementing an Automated Programming Assessment plugin, (the first being sandboxing). This is not just an issue for programming assignments but one for all submissions at an academic level. As such there already exists a vast amount of information on the subject that covers the intricacies in detail so only a brief overview will be covered here.

The prospect of plagiarism detection and code inspection are more complex than they first may appear. Just because two students submit code that has sections that are near exact does not mean that plagiarism has occurred. For instance a student may be given some level of code to begin or they all derive their code from a common source example given by their lecturer and not to mention that often there are only a few approaches to implement a particular solution. This is further complicated by the fact that it is becoming more commonplace for lecturers to encourage their students to seek open source solutions to their problems. In such cases, lecturers ask that students demonstrate their understanding of the supplied code even if they have not written it themselves [41].

A significant amount of plagiarism detection systems have been developed. In fact,nearly as many as the APASs themselves, each working on more less the same lines to perform a static analysis on the code, cleaned and with comments removed, against other submissions and external sources. Some of the more relevant systems are MOSS codesuite (formally Codematch), Sherlock (part of the BOSS system) and the plagiarism plugin for Moodle to name a few [42-45].

Thus plagiarism detection should be performed in three parts:

* Give students an introduction to the academic integrity documents authored by the university. Such documents should what constitutes fair use of references sources and materials provided throughout their course work;
* Ensure that students are not copying large chunks (as opposed to single lines or less) of their assignment that they have not been supplied with or have not duly referenced through comments and author tags;
* Revision of submission by a tutor or lecturer, though not in any high detail but to ensure that the guidelines are followed.

### Learning Management Systems integration

One of the significant drawbacks stated for many of the systems reviewed is that they were designed to be self-contained, single purpose systems. Building such systems quickly becomes non trivial as there are a lot of supporting components that may be overlooked such as [31]:

* Login and user management - for users to be able submit their own files there must be some form of login to distinguish and separate such files and submissions.
* Gradebook - to be of any use for bulk automation assessment individual grades must be recorded otherwise final tests might just well be done individually off system using unit testing. Even if gradebook is used most often, grades will have to be manually transferred to the official gradebook for the topic.
* Interface design - even if a command line UI is used a suitable interface must be developed that should be simple to use.
* Security measures to protect passwords and stop misuse of the system.
* Continued maintence, improvement and promotion of the system, though not a true component per se, continued maintence, improvement and promotion of system cannot be overlooked. This is what drives the use of a system - as a system that will not be used; is a waste of time to develop. This goes beyond the use by the designers of the system, to encourage other colleagues and academic institution to use and contribute to further system development. The later requires opening the source code up to the community so that more people can use it and add the functionality they require. To this end, in addition to user documentation, a greater emphasis should be applied to developer documentation for the developed system not just providing references to the platform on which it runs. Such documentation should also make information about the current state of the project freely available, to note its faults, future improvements and related versioning information.

Implementing these features and components properly can end up taking more time than the work on the core components that handle program compilation and testing. As such it is not surprising that most systems restrict themselves to a single UNIX environment that ties with existing student account systems. Such solutions are compromises as they create additional problems such as students having to login to a university machine which since most student’s, especially first year students, lack the ability to perform, means that local access is required. This necessitates the need to book computer labs and schedule tutors to assure students of access and in doing so decreasing the effectiveness and usefulness of the system.

This is combined with a push by most academic institutions to remove what little remains of the older non digital submission methods and to integrate and upgrade existing digital academic systems that are dated and no longer performing at their full potential. Flinders University is one such institution with is WebPET guideline where the aim is to provide a seamless consistent online learning support system across the university. [46]

For these reasons it is suggested that future APASs should be integrated into LMSs as they already provide many of the aforementioned features from account and file management through integrated gradebooks to facilities for plugin development. All the while offering a high degree of operating system independence with the web based architecture.

## Current Flinders Systems

Like most large academic institutions, Flinders University already employs the use of several Automated Programming Assessment Systems. Three of these systems are reviewed and the common problems associated with them identified.

### Computer Programming 2

Computer programming 2 or CP2 is a first year computer program topic, the second of two computer programming topics offered at Flinders University, the other being Computer Programming 1 (CP1). The topic introduces students to C/C++ programming building upon the Java programming language students learnt in CP1. Since students are not expected to have any experience in C/C++ programming the topic focus on the low level programming elements such as memory management, pointers and the formulation and use of basic data structures.

The Automated Programming Assessment Systems for this topic are small, command line based, programs used to assess small student programs that attempt to solve a specific problem. This is achieved by providing specific input and checking its produced output against a known answer. An indication of checkpoints is then stated for the more functionality the program fulfils towards the solution.

Once a student is happy with their solution, they zip their source files and submit it to Moodle. The lecturer then downloads the zipped source files, unzips, compiles and retests them against a larger more comprehensive test set before checkpoints are awarded and feedback is presented to the student.

### Application Development

Application Development is a second year topic that exposes students to elements of Application design and development. The topic teaches the use of the Java Programming language and the NetBeans IDE to develop applications in conjunction with design principles such as the Model View Controller methodology and good class design. Application Development uses a quiz based APAS to test students on a biweekly basis for which the students were graded. The quiz system combines short answer, multiple choice and small programming questions, presenting students with a randomised list of questions for each test to reduce the prospect of cheating and plagiarism.

The System also offered a practice test and exam that students could undertake on their own time. Though to access the system remotely meant installing additional software beyond the remote client to handle the GUI window the app used. This increases the difficulty of setting up the system but since is main function was testing students during tutes this isn’t necessarily a problem.

### Networks / Networks and operating systems

Another Topic of interest is Networks and Operating Systems that gives students an introduction into computer networking and the inner workings and functions of operating systems. For the past two years prior to Semester 2 2014, Networks and Operating Systems existed as two separate topics with two separate APASs. This year a new APAS was introduced that was created for a single project - to test a student’s Internet Relay Chat (IRC) server, the major programming assignment, by acting as one or more clients.

The APAS gives students a practical introduction to Connection handling, Concurrency and synchronisation. The system is based on Test Driven Development, informing students of the progress, successes and failures they make as they try to implement the functionality of the IRC server.

### Issues

Though these systems are well directed and have received good feedback from students they do have several notable drawbacks that include, but are not limited to:

* They are all university machine dependant – essentially forcing students to having to go to the university to produce their assignment so they can access write and test the software in the right environment. Whilst Remote login is an viable alternative it does require some level of setup knowledge, though often this is far complex for some students, a problem often compounded limited introductions to the approach by Lecturers.
* On the lecturer and tutor side they do reduce the workload of having to mark student assignments and test submissions but the system is still separate from the main grade books. That is, there is no integration with FLO. This means that lecturers and tutors must manually transfer grades from the student programs onto FLO.

Many of these problems can be solved by introducing a single common system that ingrates into flow.

## The Previous CodeHandIn System

For the reasons described in the previous section, a Moodle module for the Automated Assessment of student programs entitled CodeHandIn was developed by Jonathan MacKenzie as his main honours project in 2013. As this project continues this development, it is relevant to give a brief overview of this previous project.

In brief the project:

“[Proposed] a new system, where the IDE and LMS are connected, providing immediate feedback on the assessment test results. This should improve workflow for both students and teachers as it removes the need for students to manually zip and upload their code through the web interface of Moodle and give the assessor more time to focus on providing feedback to code instead of compiling and waiting for tests to run.”

To view a full description of this project, please see Mackenzie 2013 which can be located on the Flinders project web at <https://wiki.csem.flinders.edu.au/bin/view/CSEMThesisProjects/ProjectMack0242>

### Outline

**System**

Moodle

Codehandin module

* Basic settings
* Test results display
* Tests and Submissions via client only

Moodle web Interface

Client Side

Moodle Server Side

Client

Python

CodeHandIn

Client

CodeHandIn

Webservice

Local

Compilers

(controlled through a web service function)

Figure 1: Architecture of the CodeHandIn Module

The specification of the previous project defined the development and use of:

* A Moodle module that handled the definition of a CodeHandIn assignment, assignment instances and testing of student submissions.
* A Moodle Webservice that provided external access using XML-RPC to the Moodle module, facilitating support for:
  + Fetching assignments
  + Testing assignments
  + Listing files
  + Submitting an assignment
  + Creating a test
  + Creating a checkpoint
* A Python 2.7 client to access the Web service.

### Design of the system

The system design specified:

* The data requirements and data scheme the project used, outlining what constituted a test, checkpoint, CodeHandIn assignment and a CodeHandIn submission.

The updated data requirements can be found in section 3.2.1.2 Data requirements.

* Uses cases for the three main relevant systems operations, that of
  + CodeHandIn assignment creation (test, checkpoint and relevant info specification),
  + the submission and testing of assignments and
  + review of submissions

Since the use cases describe patterns of system use, that are essentially as is used in this version of development, the diagrams can be found in appendix 8.1 Use Case diagrams form the previous project

### Issues with the system

This first version of the system is more a prototype than a fully functionally polished system. As such there are few drawbacks, most of which are noted in the remaining and future work sections of the thesis submitted by Johnathon Mackenzie.

Some of the more notable issues with the previous system include:

* No sandbox support (though sandbox types were looked into, no specific type was chosen or developed);
* The absence of any form of plagiarism detection. Though this was probably due to the complexity of producing a plagiarism detection plugin combined with the fact that there already existed a separate project for developing of a plagiarism detection plugin;
* No support for updating, deleting or reordering of checkpoints or tests. It was however suggested that support for viewing a CodeHandIn be added to the Moodle module to solve this problem;
* Not fully integrated with the Moodle gradebook;
* The assessor cannot batch run the assignments for regrading;
* The Python script required details to be entered each time;
* No support for group assignments and collaborative work.

# System Design

The updated system extends and redefines the system specified by the previous project, to address many of the issues present in the previous system. This is predominantly achieved through a change in architecture for closer integration with the existing Moodle components. There are also several other improvements that address items specified in the future and remaining work sections of the previous project, such as better IDE support, more documentation and more robust coding of reworked sections.

The project ran over the course of 1 semester and as such development essentially occurred over a period of 12 weeks, so several of the initial aims such as the full testing, integration and an open review were not realised. As such the system design and interfaces are still evolving and are expected to change from when the project finally begins to be used in operation in the future.

This chapter defines the new architecture, the relevant changes in the package components that supported it and the introduction of a new NetBeans plugin and a new Java Client.

## Architectural changes over the previous project

Moodle

Moodle’s existing Assignment Module

Client Side

Moodle Server Side

Compiler

System

Codehandin

Submission

Plugin

Codehandin

Feedback

Plugin

Codehandin

Web service interface

Remote

Sandboxed Compiler controller

1

3

Shared lib

5

4

NetBeans IDE

Moodle web Interface

Python

Codehandin

Client

2

Java

Codehandin

Client (in IDE)

Local

Compilers

Local

Compilers

Figure 2: The new architecture for the CodeHandIn package

After familiarisation with the existing project and consultation with Netspot representatives (who host FLO, Flinders University’s version of Moodle) the architecture defined in the previous project (see Figure 1) was found not to be the best possible architecture for an Automated Programming Assessment System (APAS) for Moodle. The preferred architecture was to re-implement the CodeHandIn module as two separate plugins; one submission plugin and one feedback plugin, to extend Moodle’s existing assignment module.

This allows a CodeHandIn module to utilise the existing features of the Moodle Assignment module including support for group assignments, complex deadlines, better gradebook integration, the ability to grant personalised extensions and integration with other plugins such as the general text based feedback plugin. This takes most of the burden off the CodeHandIn plugins leaving the two new plugins to focus on the CodeHandIn specific components such as checkpoint and test creation and modification, submission feedback and manual assessment.

The architectural change meant that rather than having two components requiring use of the core functionality (the testing of submissions and the creation/modification of the tests) there were now three. To adapt to this change the code was refactored to use a centralised shared library attached to the web service as it uses most, if not all, of the core functionality methods.

Another change was the move to a more formal specification of an external compiler. Though it was stated in the previous project that a local compiler was only used for convenience, this project formally defines the use of an external sandboxed compiler. This includes a specific choice of compiler and support for its future implementation.

## System Components

The CodeHandIn package consists of five distinct components, as seen in Figure 2, these are

1. **A CodeHandIn Submission Plugin**, that handles the creation and modification of CodeHandIn Assignments through the definition of tests, checkpoints (groups of tests) and other information. See section *3.2.1 CodeHandIn Submission Plugin*;
2. **A range of Client plugins**, that provide an external interface for the management, submission and testing of CodeHandIn Assignments. See section *3.2.3 Java Client & NetBeans plugin;*
3. **A CodeHandIn Webservice**, that provides the functionality for the management, submission and testing of CodeHandIn Assignments. See section *3.2.2 Moodle Webservice*;
4. **A Remote Sandboxed Compiler Controller,** that allows Submitted programs to be compiled and tested remotely. See section *3.2.4 Remote Compile Controller; and*
5. **A CodeHandIn Feedback Plugin,** that handles Feedback and final grading of submissions. See section *3.2.5 CodeHandIn Feedback Plugin;*

Note that the entire data requirements for the CodeHandIn package are defined within the Codehandin Submission plugin, see section *3.2.1.2 Data requirements* for a complete specification.

### CodeHandIn Submission Plugin

The CodeHandIn Submission plugin has been given two primary responsibilities in the CodeHandIn system:

1. to define the database tables that support and represent the CodeHandIn assignment type and
2. to define the web interfaces for the declaration of global CodeHandIn options and the initialisation of CodeHandIn assignments.

#### CodeHandIn Submission Plugin Web interfaces

Currently the CodeHandIn Submission Plugin only defines one interface; an interface for handling of global options for the entire CodeHandIn assignment (as seen in Figure 3). At present there are only two global options for the CodeHandIn System, one that specifies the default programming language and another that defines if the option to compile/test before submission should be selected by default on the Assignment Creation form. However it is noted that there are several other options that would benefit from having a global default value set. Other possible global options include defaults for

* The style grade mark,
* The test limit per student,
* Runtime specific arguments such as max memory, max threads, compile time length and others

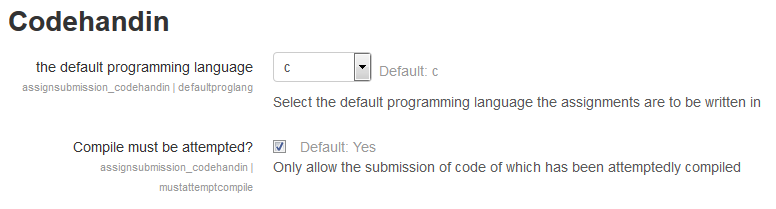
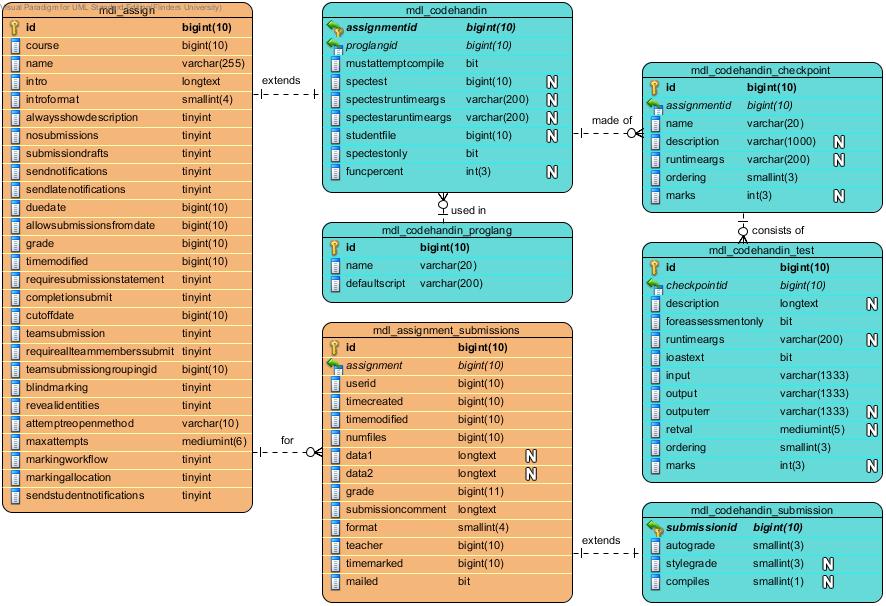
­­­­

Figure 3: Current Global CodeHandIn Options

The second type of web interface declared by the CodeHandIn Submission plugin used to aid in the initialisation of a CodeHandIn assignment.

This predominantly involves specifying the programming language to be used, other options like if the a test/compile must be attempted before submission and how submission will be tested i.e through a single test program i.e. one that may act as a client or through the use of tests and checkpoints (groups of tests). In total there are 23 options (5 high level, 5 a checkpoint and 13 for a tests) that have been placed onto a Moodle form and are fully functional with the underlying database.



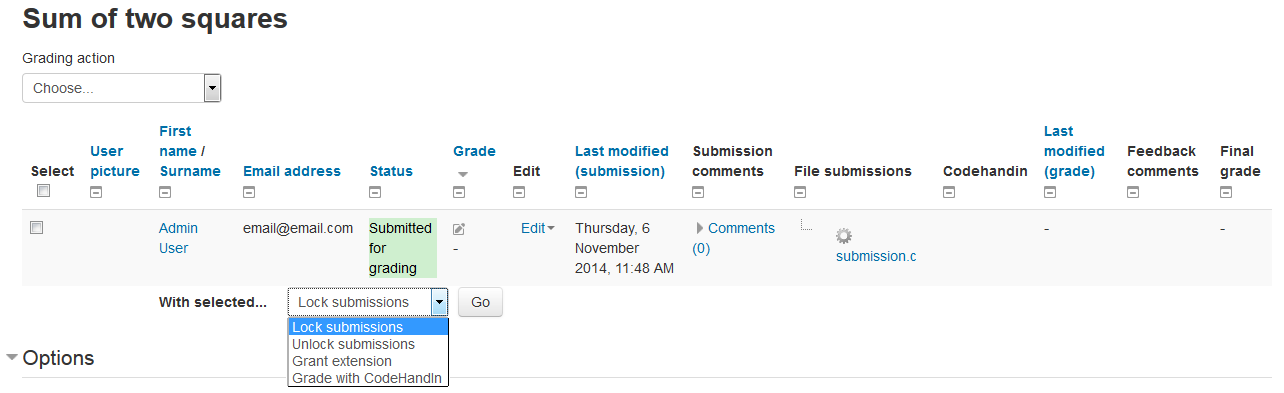
Tables for CodeHandIn

Core Moodle Tables

Figure 4: The integration of the CodeHandIn with the existing assignment moodle tables

Since this only gives the ability to define one checkpoint consisting of one test it is in no way functional or useful in any practical sense but does give some indication of how a future web based form would work. The reasons for this lax development is that the form API is more about simple options rather than describing complex entities and their relationships.

Significant effort was spent looking into the yahoo user interface (YUI) that is used to define the form elements but it was found that what was required was the extension of existing elements a process that would be time consuming. The alternative was to employ HTML elements but this would break the layout of the interfaces and would also require large effort to implement. As such a lot of time was wasted fiddling with UI trying to get it to look better but unfortunately the result did not get much passed the form presented in Figures Figure 11 and 12 in the Appendix.



The new CodeHandIn Option

Figure 5: Grading Option introduced by the codehandin feedback plugin

#### Data requirements

The data requirements of the entire CodeHandIn package are handled by the database tables defined in the submission plugin. The plugin introduces 5 new tables into the database (light teal coloured in Figure 4), two of which integrate directly into the existing Assignment module’s tables (mustard coloured in Figure 4)

Since these tables reflect the overall structure and options of the CodeHandIn assignment type, a brief description of each of the introduced tables and their components is presented below. It should be noted that these tables’ only store data that will be used more than once, information such as project compile/Test instances are stored in memory.

##### mdl\_CodeHandIn Table:

The mdl\_CodeHandIn table extends the definition of assignment (as defined by mdl\_assign table as seen on the left hand side of Figure 4) with specific features for the Moodle CodeHandIn package. This primarily specifies the programming language as well as handling for specific testing files (though the latter is yet to be implemented). The CodeHandIn table used in the previous development, supported an assignment style module, contained many of the fields present in the mdl\_assign table since the architecture now uses the assignment model (the replicated and facsimile fields were removed). The fields that support the definition of a CodeHandIn assignment are:

|  |  |
| --- | --- |
| assignmentid | the id of the assignment (references id in mdl\_assign) |
| proglangid | the id of the programming language (a ref to mdl\_CodeHandIn \_proglang) |
| mustattemptcompile | an option that lets assignments specify that they must attempt a compile/check before they can be submitted |
| spectest | Pointer to a specific test file that can be used to replace or in conjunction with checkpoints and tests assessment definition. |
| spectestruntimeargs | arguments for the specific test file for regular testing |
| spectestaruntimeargs | arguments for the specific test file for assessment testing |
| studentfile | pointer to sample test file that students can download |
| spectestonly | allows the use of a specific test file only |

Table 1: The mdl\_CodeHandIn table is used to hold CodeHandIn specific options, extending the assign table of the assignment module.

##### mdl\_CodeHandIn\_proglang

The mdl\_CodeHandIn\_proglang table stores the compile scripts for all the supported programming languages.

|  |  |
| --- | --- |
| id | the id of the programming language automatically assigned |
| name | the name of the programming language |
| defaultscript | the default script to use the language |

Table 2: The mdl\_CodeHandIn\_proglang table used to store information to support programing languages.

##### mdl\_CodeHandIn\_checkpoint

The mdl\_CodeHandIn\_checkpoint table represents a checkpoint or grouping of tests, its main function is to give each grouping a description that define the functionality it is testing.

|  |  |
| --- | --- |
| id | the id of the checkpoint automatically assigned |
| assignmentid | the id of the assignment of which the checkpoint belongs to |
| name | the name of the checkpoint |
| description | a description of the checkpoint (can include hints etc.) |
| runtimeargs | runtime arguments for the checkpoint |
| ordering | an integer that defined the order of the checkpoints for an assignment |
| marks | the marks assigned for this particular checkpoint |

Table 3: The mdl\_CodeHandIn\_checkpoint table defines a checkpoint or grouping of tests that define a common functionality.

##### mdl\_CodeHandIn\_test

The mdl\_CodeHandIn\_test table defines a single specific test. Each test should be written to validate a specific functionality (as defined by its checkpoint parent) under a specific set of conditions (input) using what it is expected to return under these conditions (output and or outputerr)

|  |  |
| --- | --- |
| id | the id of the test automatically assigned |
| checkpointid | the id of the checkpoint to which the test belongs |
| description | a description of the test (there is no name field) |
| forassessmentonly | if the test is for assessment only |
| runtimeargs | runtime arguments specific to this test |
| ioastext | if the input, output and outputerr fields are text rather than file pointers |
| input\* | the input for the test (text or file pointer) |
| output\* | the output for the test (text or file pointer) |
| outputerr\* | the outputerr for the test (text or file pointer) |
| retval | the return value of the test |
| ordering | an integer that defined the order of the test for the checkpoint |
| marks | the marks for this specific test (marks are equally divided with any remainder added to the last test) |

*Table 4: The mdl\_CodeHandIn\_test table that defines a single specific test.*

\* Fields that can accept values of 1333<= chars or files pointers which are later looked up in the file system by the web service.

##### mdl\_Codehandin\_submission

The mdl\_Codehandin\_submission table extends the assignment submission features with features specific to the Moodle CodeHandIn package.

|  |  |
| --- | --- |
| autograde | the grade from the automatic assessment |
| stylegrade | the grade for style supplied by the teacher |
| compiles | if the project compiles |

Table 5: The mdl\_Codehandin\_submission, that contains CodeHandIn specific options extending the mdl\_assignment\_submission table of the assignment module

##### mdl\_Codehandin\_codehandin\_runtime [deprecated]

Stored the runtimes for different languages, is now handled by the mdl\_CodeHandIn\_proglang table. Though some functionality was not carried over as it was not used.

### Moodle Webservice

The Webservice provides external services for the fetching of CodeHandIn information, the testing of assignments and the creation and modification of the CodeHandIn assignment specification. A complete list of functions provided by the Webservice are described in the following table.

|  |  |  |
| --- | --- | --- |
| **Fetching services** | | |
| **Service name** | **New/Updated** | **Service description** |
| fetch\_assignments | U | Return all or only the basic details of all or one of the available assignments for a specified user. |
| fetch\_assignment\_files | U | Return any files provided for an assignment |
| fetch\_uploaded\_assignment | N | Return files uploaded for an assignment |
| fetch\_file\_list | U | Return a list of submitted files and their information for a specific assignment. |
| **Testing services** | | |
| **Service name** | **New/Updated** | **Service description** |
| upload\_assignment | U | Upload a submission to Moodle with an option to test and or make the final submission. |
| **Creation/Update services** | | |
| **Service name** | **New/Updated** | **Service description** |
| create\_test | U | Creates a test for a given CodeHandIn checkpoint |
| create\_checkpoint | U | Creates a template for a given CodeHandIn activity with specified name and description', |
| update\_CodeHandIn | N | Updates a CodeHandIn to match the provided JSONObject and test files. |

Table 6: Webservice functions, N – new function U – existing Updated function

Many of the web services functions were already developed from the previous project but with the change to architecture most of these functions were rewritten to use the assignment module tables and adapted to the simpler CodeHandIn specific table. Options were also added for some of the existing functions, such as retrieving details for a specific assignment and retrieving a summary of all assignments as opposed to retrieving all details. As part of the refactoring the programming languages were moved to a separate table allowing languages to be introduced by adding them in the DB rather than having to hard code individual support for each language. See 3.2.1.2.2 mdl\_CodeHandIn\_proglang for a description of the table introduced.

Despite the architecture adaptions and other improvements, gaps in the desired functionality coverage were identified. To close these gaps two new functions were also developed – one to download any previously uploaded files for an assignment and another that allows the uploading of complete CodeHandIn assignments (as JSON objects) without the need repetitively enter checkpoints and tests through the create\_test and create\_checkpoint methods.

### Java Client & NetBeans plugin

To make the system more user friendly at least to students new to programming, a plugin for NetBeans that allows the downloading, testing and submission of student assignments was produced. The NetBeans uses a new native Java Client that allows it to interface with the CodeHandIn Webservice.

The Java Client is essentially an interface, implementing the web service functions as they appear at the web service (see 3.2.2 Moodle Webservice for functions list) with only minor checking so that incomplete and malformed requests are not made of any to the web service functions. Though there is the ability to extend the plugin in future so that CodeHandIn assignments can be made locally whilst being constructed and then uploaded to Moodle in one single function once completed this functionality has not been implemented yet.

The NetBeans plugin is designed to be an all-in-one plugin for handling of CodeHandIn assignments for students. It does this using a menu bar (see top of Figure 5) for the most generic operations that can be done in a single click and an options panel (see bottom of Figure 5) for more detailed operations that require login information and the displaying of more complex information.

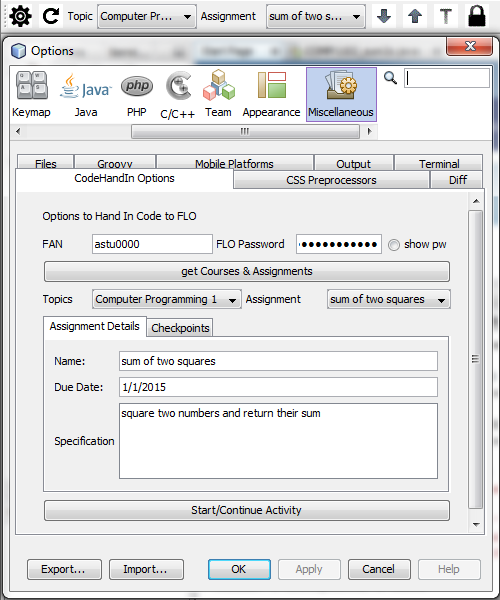


Figure 6: NetBeans Plugin Menu bar (TOP) and options form (MIDDLE) with CodeHandIn information retrieved using an internal Java client.

The menu bar currently can perform the following operations

|  |  |  |
| --- | --- | --- |
| icon | Operation | Description |
|  | View Settings | Opens the NetBeans Settings page |
|  | Check update | Downloads a version form Moodle if current version is not the newest |
|  | Download | Forces download from Moodle (asks if sample or other is preferred) |
|  | Upload | Forces upload (submission) to Moodle |
|  | Test | Uploads (submits) to Moodle which test the code and displays the results |
|  | Submit | Submits the code to Moodle for final grading |

Table 7: Menu bar Supported operations

The submit button was made separate to both the upload and test buttons as it would require a checkbox to avoid submission each time which would be both annoying and making it more likely that student would accidently press the submit button. The button icons were chosen for illustrative purposes and will probably be changed as they are somewhat dull and unintuitive in their current form.

The other part of the NetBeans plugin is the options panel that primarily aims to allow students to retrieve and display information about their CodeHandIn assignments. The students input their FAN and FLO password and press the Get Courses & Assignments button which uses the Java Plugin to get a token if it has not already be retrieved and gets the CodeHandIn assignments for topics the student is enrolled in. The web service sends this information as a JSON object which the NetBeans plugin uses populate the fields in the assignment details and checkpoints tabs.

The student, having selected the topic and the assignment they can view the general details of the assignment and the specific checkpoints and tests that will define the functionality of the project. Once they have identified the assignment they want to work on they can click Start/Continue Activity to begin or continue the assignment using functionality shared with the check update button.

When the student is ready to test their assignment they press the T button on the menu bar which uploads their submission to Moodle which then performs the check on it. The details are then returned as formatted text and written to an output window as seen in Figure 6.

Currently this process is not very quick and it might be advantageous in future to use a progress bar to give students some indication that testing will take a little bit of time. However the fact that tests are not near instantiations gets students to think more before testing. This is one of the reasons that support for limiting the number of tests per student was not implemented.

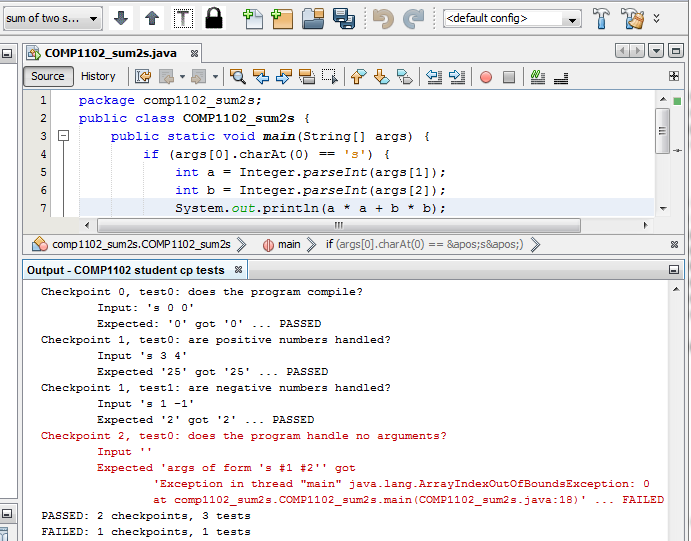


Figure 7: NetBeans Plugin test example

### Remote Compile Controller

As expressed in the literature review (specifically see section 2.4.1.4 Sandboxing of submitted code) and in the recommendations of the previous project, one of the most important components in APAS is the ability to execute code safely and securely. This project takes this on board and has implemented an external compiler controller called CompileBox [47, 48]. CombileBox was designed to be used as a tool for accessing programming submission for Job interviews but is also quite well suited to testing programming submissions.

CompileBox was primarily chosen as it is Docker based and has a higher degree of security and robustness than a traditional Chroot Jail System (changes the apparent root directory) where it is possible to break out of. Whereas a Docker system use self-enclosed containers that run on top of the operating system. The containers are completely separate, created and destroyed as needed with the effects of any malicious limited to the container [47].

To integrate Compilebox as a separate compiler, a client for Moodle was written. The client for the most part was taken from VPL with only a few modifications to change to use the CompileBox API. The system then works as before, sending files to be compiled and a test to compile box which returns the results. Though currently the system is a little laggy but this can be improved with multi testing and caching of test files in the future.

### CodeHandIn Feedback Plugin

Since most of the functionality is provided by the assignment module the CodeHandIn Feedback plugin only has one responsibility – to provide a web interfaces for the marking of final submissions, this includes the ability to:

1. batch test student submissions through the grade page and
2. Allow review of individual submissions for a style mark.

CodeHandIn batch grading of students submission is easily achieved through an entry added to the batch grading operations list on the grading form page (as seen in Figure 13 which can be found in the Appendix). However this is somewhat redundant as the system currently compiles and tests assignments when a student makes their final submission.

The second part of allowing review of individual submissions was equally easy to achieve by adding a few form elements to the user grading page. See Figure 14 in the Appendix.

As previously stated the actual functionality behind these interfaces is provided through the shared library in the web service.

# System Development

To help future contributors to the project and provide a record of the development carrier out in this project an overview of the System development for this project is provided.

It should be noted that the project was undertaken over a semester creating significant time pressure on getting up to speed with the state of the project and being able to contribute and improve it. Furthermore work on this project was not started until the middle of the second week of the semester. In retrospect this was a significant waste of time and projects information and guides should have been provided before the end of the previous semester with a deadline to choose a project in the mid semester break.

However despite this significant progress was made on the project. The development followed the following schedule:

**Week Activity**

2-4 Familiarisation with the project, commenting the code and reading relevant guides and instructions to Moodle programming and Automated Programming Assessments.

5-6 Original the aim was to add more features of the Moodle assignment system into the CodeHandIn module, but it became apparent that a twin plugin system (the architecture developed) would enable use of the existing assignment module without the need for replication. Work then went into redefining the database tables and rebuilding the queries that use the tables. Since most of the core functionality had to be rewritten, time was taken to centralise and rework the functions into a shared library attached to the web service

At around the same time a Skelton for a NetBeans plugin was developed, though was put on hold as not suitable example was found to create a project using the API.

6-7 The production an interface to create tests and checkpoints through the Moodle web interface (see section 3.2.1.1 CodeHandIn Submission Plugin Web Interfaces for further details). The initial goal was to make a dynamic table like list where checkpoints could be added by adding a new line and rearranged by moving rows up and down. But after extensive look into the Yahoo user interface (the GUI API used by the Moodle) the creation of such an interfaces was found to be too difficult for the timeframe or the project. Other work was done to try and improve the layout of the elements on the page but this also didn’t culminate in anything usable. As such the interface produced is the same as seen in section 8.2 New CodeHandIn Screenshots

8-11 The development of a java client, and continued development of the NetBeans plugin skeleton into a working prototype.

The Java client was developed for native integration with NetBeans and Codehandin modules. The client was originally written using the apache httpcomponents-client-4.3.5 library but it was found that NetBeans only has support for the 3.1.0, so it was rewritten but it was later found that had support for a fairly recent build of the jersey web client library and thus the client was rewritten again.

Whilst creating the NetBeans plugin some functionality was found to be missing so some existing functions were expanded with options and some new ones created, the results of which are expressed in section 3.2.2 Moodle Webservice.

The storing of files was also changed from using a separate CodeHandIn file area to using a assignment submission file area, in essence storing uploaded files in the submission box.

Some time also went into producing a PowerPoint presentation and rehearsing the associated speech.

11 The development of the Feedback plugin GUI.

Though templates of both the feedback and submission plugin were downloaded at the same time, it wasn’t until week 11 that anything was done on the feedback plugin. This proved to be fairly easy as it meant just adding only a few elements to the feedback forms and creating a batch function to grade multiple submissions.

12+ The rest of the time was spent working on the thesis and poster with some minor changes to project components

# Remaining Work

Despite the advance this project represent, a substantial amount of work still remains before the system can be introduced into a programming topic. This includes little things such as adding the fetching of grades to the web service function to creating a proper GUI for the submission plugin. In additional all of the components need to be tested and scrutinised thoroughly before the system is ready for some sort of trial. This section identifies the remaining and future work based for each component.

**Web service work**

* Add the fetching of grades to the fetch assignments function or perhaps introduce a new function.
* Use Moodle’s built in testing features to test the service.

**Submission plugin work**

* The CodeHandIn creation GUI for the web service needs to be completely rewritten using separate pages and a dynamic table like system as per the original aim as described under week 6-7 of the System Development chapter

**Feedback plugin work**

* Add grade on submit option such that assignments submitted for final marks are graded with graded entered into the submission table rather than being regraded through the option on the web interface.
* Add more global options such as default style mark percent

**Clients work**

* The Java plugin requires full testing of all functions, though most of the functions work, some have not been tested all that much as they were not used by the NetBeans plugin
* Little to no work was done on the handin.py script which needs to be updated to use the redesigned web service
* Split the clients in half creating one for CodeHandIn management (for staff) and one for submission and testing (for students)

**NetBeans plugin work**

* Create a project using the API, at the moment the plugin uses information based on the project of the current open file rather than the selected CodeHandIn project displayed in the menu bar.
* Real world examples of the plugin need to be attempted before it can be used

**External Compiler Service**

* Cache test files on the compile service side rather than sending them each time.
* Better handling of batch testing
* Rework client to use a different External compiler service … perhaps use the same as the Coderunner system when its introduced

# Conclusion

This project represents a significant contribution towards a fully integrated Automated Programming Assessment System for Moodle despite the short timeframe for development. The main impact of this project has been the introduction of a more modular architecture, reusing existing Assignment Module components making the system simpler and easier to maintain in the future. Other improvements include:

* A new GUI for the creation of CodeHandIn assignments has been made, though not fully functional, as only allowing the description of one checkpoint and one test provides a good start for further development;
* Similarly a GUI for the Feedback plugin is simple and will improve the tutors and lecturers ability of to mark submissions quickly;
* The redesign and integration of code in the web service has also made the project simple and will make future maintenance and improvements of the system much easier;
* The creation of a NetBeans plugin, though not as functional as initially aimed, does provide a simple interface to the system that students can use without having to use the command line which will hopefully encourage them to test more often;
* The introduction of an External Compiler service that will make the entire system more secure and open to use by the hosts of Moodle; and
* Smaller improvements such as a table for storing programing language information

These improvements have culminated in a more complete system that meets the initial aims of the project by making the marking of programming assignments simpler and freeing tutors and lectures form mundane and repetitive questions.

Despite the improvements, there remains a substantial amount of work to do before the system is ready for a trial run. As it is only in a trial run that the true value of the system itself and the contributions made in this project can by validated. However in the meantime the review of the CodeHandIn System as specified by this document has shown that this project takes CodeHandIn System closer to a useable system.

Given the difference the CodeHandIn system could make in future programming courses combined with the amount of work that has already been done to create the system it is highly recommended that the system continue development. It is estimated that another Masters project or equivalent is all that is required to get the CodeHandIn system up to the level of a trial candidate.

# References

1. Hollingsworth, J., *Automatic graders for programming classes.* Commun. ACM, 1960. **3**(10): p. 528-529.

2. Forsythe, G.E., *Automatic machine grading programs*, in *Proceedings of the 1964 19th ACM national conference*. 1964, ACM. p. 141.401.

3. Hext, J.B. and J.W. Winings, *An automatic grading scheme for simple programming exercises.* Commun. ACM, 1969. **12**(5): p. 272-275.

4. REEK, K.A., *The TRY system – or – how to avoid testing student programs.* SIGCSE Bull., 1989. **21**(1): p. 112-116.

5. ISAACSON, P.C.A.S., T. A., *Automating the execution of student programs.* SIGCSE Bull., 1989. **21**(2): p. 15-22.

6. Higgins, C., et al., *The coursemarker cba system: Improvements over ceilidh.* Education and Information Technologies, 2003. **8**(3): p. 287-304.

7. Matt, u.v., *Kassandra: the automatic grading system.* SIGCUE Outlook, 1994. **22**(1): p. 26-40.

8. Jackson, D. and M. Usher, *Grading student programs using ASSYST*, in *Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education*. 1997, ACM: San Jose, California, USA. p. 335-339.

9. Joy, M. and M. Luck, *The BOSS On-line Submission System.* Monitor, 1998(10): p. 27-29.

10. Joy, M., N. Griffiths, and R. Boyatt, *The boss online submission and assessment system.* J. Educ. Resour. Comput., 2005. **5**(3): p. 2.

11. Saikkonen, R., L. Malmi, and A. Korhonen. *Fully automatic assessment of programming exercises*. in *ACM Sigcse Bulletin*. 2001. ACM.

12. Douce, C., et al., *A technical perspective on ASAP–Automated System for Assessment of Programming.* 2005.

13. Amelung, M., K. Krieger, and D. Rosner, *E-Assessment as a Service.* Learning Technologies, IEEE Transactions on, 2011. **4**(2): p. 162-174.

14. Edwards, S.H. and M.A. Perez-Quinones, *Web-CAT: automatically grading programming assignments.* SIGCSE Bull., 2008. **40**(3): p. 328-328.

15. Buffardi, K. and S.H. Edwards, *Impacts of adaptive feedback on teaching test-driven development*, in *Proceeding of the 44th ACM technical symposium on Computer science education*. 2013, ACM: Denver, Colorado, USA. p. 293-298.

16. Singh, R., S. Gulwani, and A. Solar-Lezama, *Automated feedback generation for introductory programming assignments.* SIGPLAN Not., 2013. **48**(6): p. 15-26.

17. Ihantola, P., 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*. 2010, ACM: Koli, Finland. p. 86-93.

18. Moura, J.G., L.O. Brandao, and A.A.F. Brandao. *A web-based learning management system with automatic assessment resources*. in *Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports, 2007. FIE '07. 37th Annual*. 2007.

19. Douce, C., D. Livingstone, and J. Orwell, *Automatic test-based assessment of programming: A review.* J. Educ. Resour. Comput., 2005. **5**(3): p. 4.

20. Docs.moodle.org. *About Moodle - MoodleDocs*. 2014 5th October]; Available from: https://docs.moodle.org/27/en/About\_Moodle#Designed\_to\_support\_both\_teaching\_and\_learning.

21. Moodle.net. *Moodle.org: Moodle Statistics*. 2014 [cited 2014 5th October]; Available from: <http://moodle.net/stats/>.

22. Al-Khanjari, Z.A. and Y.M. Al-Roshdi, *Extending the Functionality of LMS to Support Computer Science Education Using Plug-in Tools*. 2014. Vol. 6. 2014.

23. Zhigang, S., et al. *Moodle Plugins for Highly Efficient Programmin Courses*.

24. Yu, L. *The Sandbox Libraries*. 2013 [cited 2014 31st October]; Available from: <http://sourceforge.net/projects/libsandbox/>.

25. ideone.com. *Sphere Engine™*. 2014 [cited 2014 31st October]; Available from: <http://ideone.com/sphere-engine>.

26. Rodríguez-del-Pino, J.C. *Features*. VPL - VIRTUAL PROGRAMMING LAB FOR MOODLE 2014 [cited 2014 31st October]; Available from: <http://vpl.dis.ulpgc.es/index.php/en/about/features:>.

27. Thiebaut, D. *Moodle VPL Tutorials*. 2014 [cited 2014 31 October]; Available from: cs.smith.edu/dftwiki/index.php/Moodle\_VPL\_Tutorials.

28. Lobb, R. *trampgeek/CodeRunner*. 2014 [cited 2014 5th October]; Available from: https://github.com/trampgeek/CodeRunner.

29. Lobb, R. *CodeRunner Documentation*. 2014 [cited 2014 5th October]; Available from: <http://coderunner.org.nz/mod/book/view.php?id=29>.

30. Amelung, M., et al., *Towards generic and flexible web services for e-assessment.* SIGCSE Bull., 2008. **40**(3): p. 219-224.

31. R, G., et al., *Enhancing learning management systems to better support computer science education.* SIGCSE Bull., 2008. **40**(4): p. 142-166.

32. Sant, J.A., *"Mailing it in": email-centric automated assessment.* SIGCSE Bull., 2009. **41**(3): p. 308-312.

33. Alstes, A. and J. Lindqvist, *VERKKOKE: learning routing and network programming online.* SIGCSE Bull., 2007. **39**(3): p. 91-95.

34. Joy, M., N. Griffiths, and R. Boyatt, *The boss online submission and assessment system.* Journal on Educational Resources in Computing (JERIC), 2005. **5**(3): p. 2.

35. Hykes, S. *What is Docker?* 2014 [cited 2014 31 October]; Available from: https://[www.docker.com/whatisdocker/](http://www.docker.com/whatisdocker/).

36. Riondato., M., *Chapter 15. Jails*, in *FreeBSD Handbook*. 2014, The FreeBSD Documentation Project.

37. Yingli, L., et al. *The Recent Development of Automated Programming Assessment*. in *Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on*. 2009.

38. Beck, K., *Test-driven Development: By Example*. 2003: Addison-Wesley.

39. Zhigang, S., et al. *Moodle Plugins for Highly Efficient Programmin Courses*. in *1st Moodle Research Conference*. 2012. Heraklion, Crete-Greece.

40. Forisek, M., *On the suitability of programming tasks for automated evaluation.* Informatics in education, 2006. **5**(1): p. 63-76.

41. Joy, M. and M. Luck, *Plagiarism in programming assignments.* Education, IEEE Transactions on, 1999. **42**(2): p. 129-133.

42. Mozgovoy, M., et al., *Fast Plagiarism Detection System*, in *String Processing and Information Retrieval*, M. Consens and G. Navarro, Editors. 2005, Springer Berlin Heidelberg. p. 267-270.

43. Aiken, A. *A System for Detecting Software Plagiarism*. 2014 [cited 2014 1st of November]; Available from: <http://theory.stanford.edu/~aiken/moss/>.

44. LE, T. *Plagiarism: Source-code Plagiarism Plugin*. Moodle Plugins 2014 [cited 2014 1st of November]; Available from: https://moodle.org/plugins/view.php?plugin=plagiarism\_programming.

45. safe-corp. *CodeMatch*. 2014 [cited 2014 1st of November]; Available from: <http://www.safe-corp.biz/products_codematch.htm>.

46. University, F. *WebPET*. University guidelines 2014 [cited 2014 1st November]; Available from: <http://www.flinders.edu.au/teaching/ict-in-education/university-guidelines/webpet.cfm>.

47. OsmanAli. *How We Used Docker to Compile and Run Untrusted Code*. 2014 [cited 1st of November 2014]; Available from: <http://blog.remoteinterview.io/post/89639823776/how-we-used-docker-to-compile-and-run-untrusted-code>.

48. Ali, O. *remoteinterview/compilebox*. 2014 [cited 2014 1st of November]; Available from: https://github.com/remoteinterview/compilebox.

# Appendix

## Use Case diagrams form the previous project

These are the Use Case diagrams developed in the previous project, since many of the operations remain the same these documents are included as a reference.

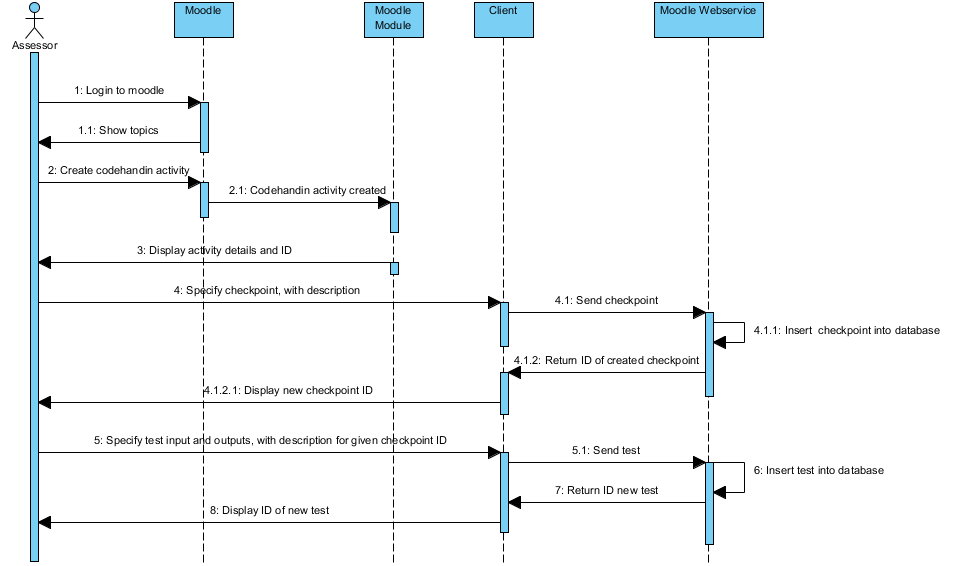


Figure 8: Teacher Creates Assignment, Checkpoints and Test Set Use Case

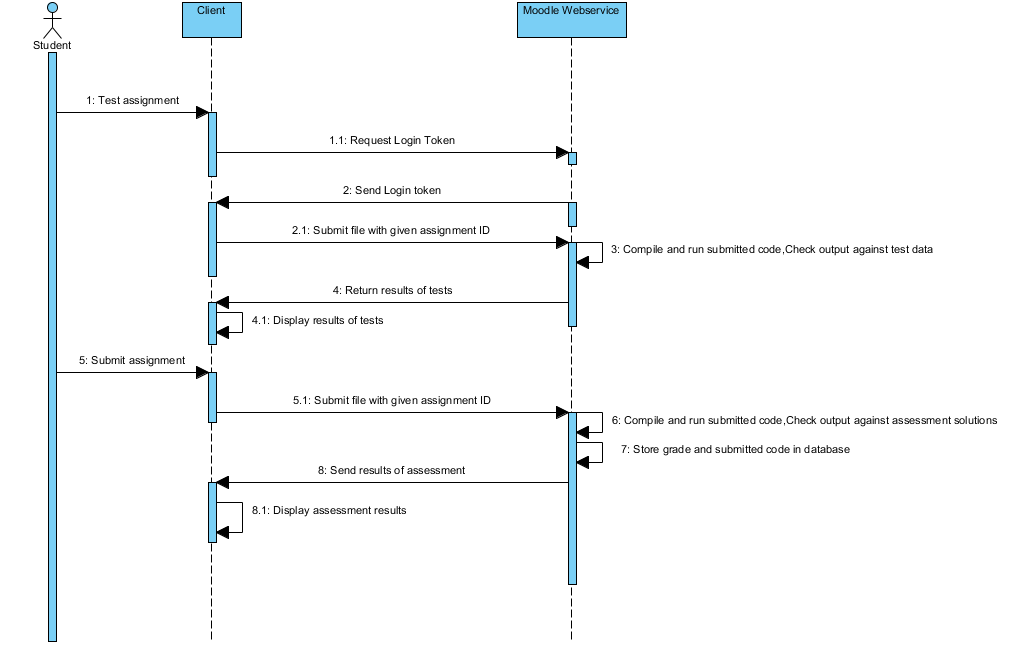


Figure 9: Student Tests and Submits Assignment

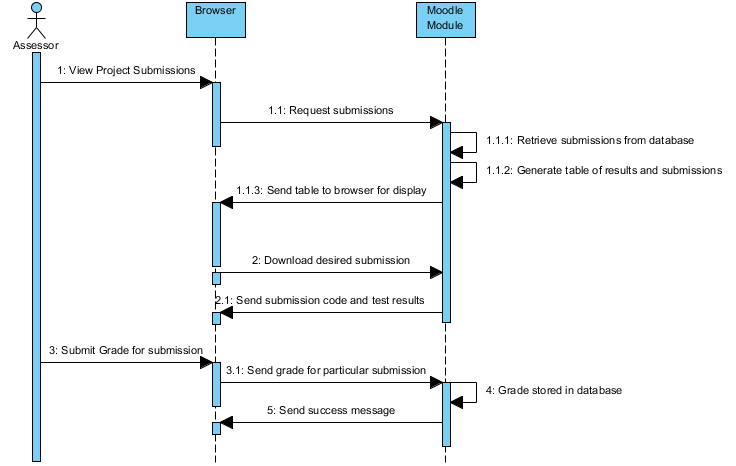


Figure 10: Teacher Reviews Submissions

## New CodeHandIn Screenshots



Other submission plugin options

Figure 11: The high level CodeHandIn options for creating a CodeHandIn through the web interface

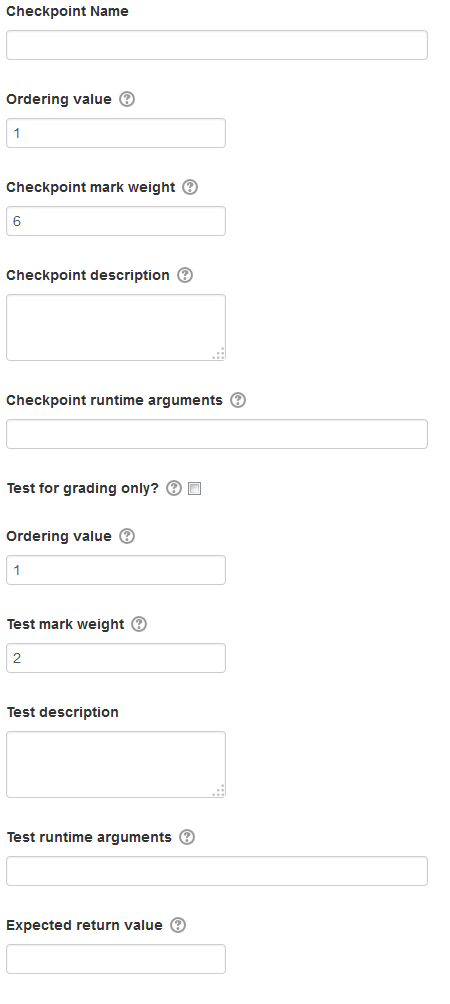


Figure 12: Continuation of Figure 11, Checkpoint and some test definition fields for creating a CodeHandIn through the web interface

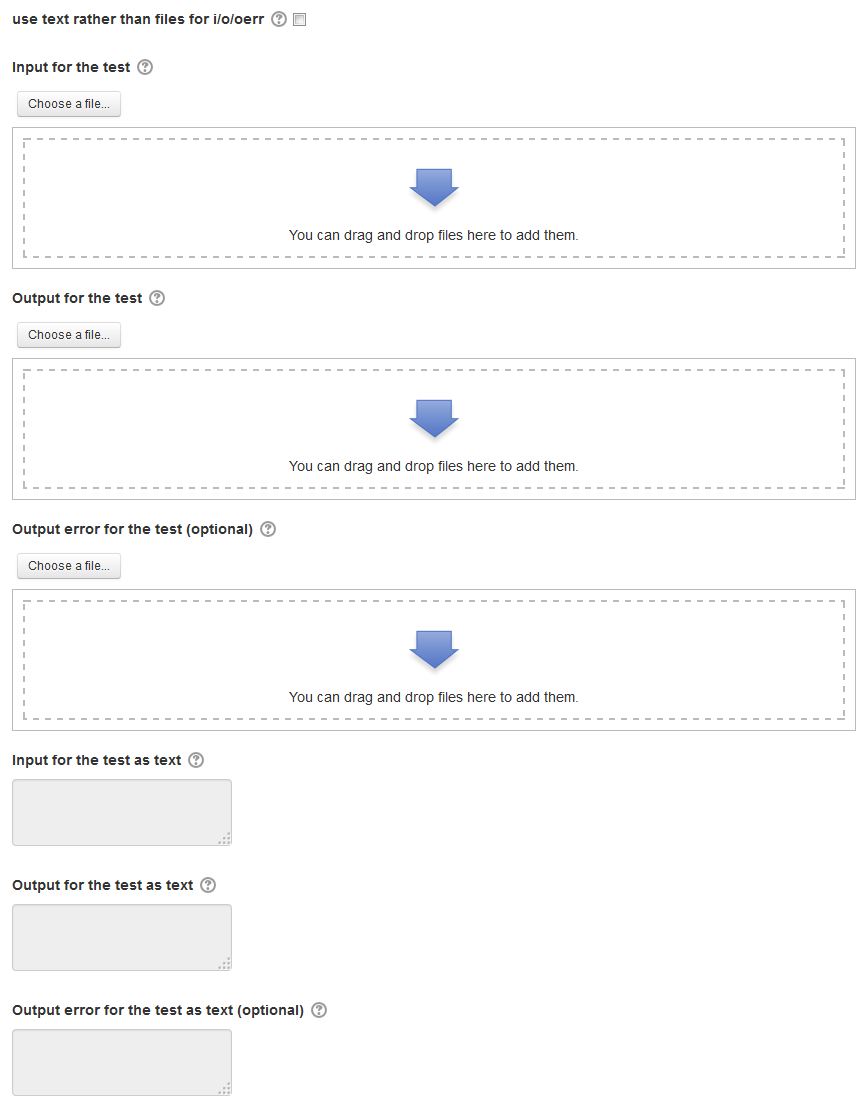
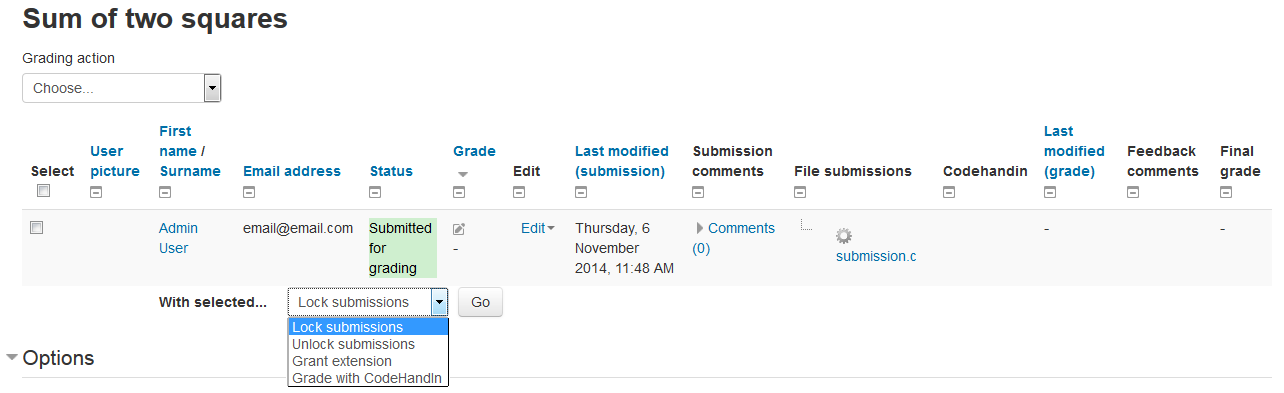
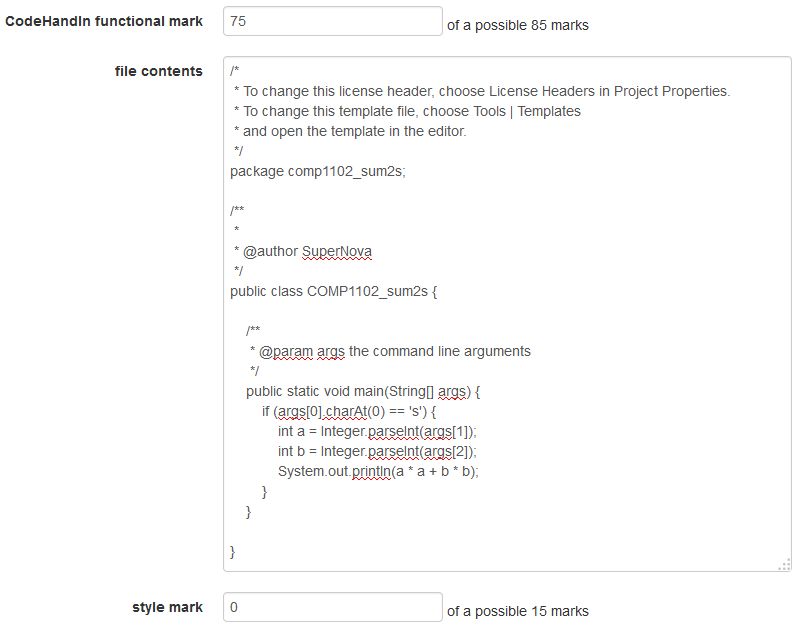


Figure 13: The Input, Output and Output error fields used in defining a tests for creating a CodeHandIn through the web interface



The new CodeHandIn Option

Figure 14: The feedback option for regarding CodeHandIn assignments through the web interface



Automatically calculated using the grader but can be manually overridden!

Figure 15: Style assessment fields for the Feedback plugin