OCAPE – Online Compiler and Performance Evaluator

Mihail Morosan

A thesis submitted for the degree of Master of Science in Advanced Computer Science

Supervisor: Mr. Keith Primrose
School of Computer Science and Electronic Engineering
University of Essex

August 2014

Abstract

Educational institutions, such as universities or high schools, often times organise programming competitions for their students. Researchers are always on the lookout for faster, more accurate or more memory efficient solutions to a wide variety of programming tasks. The tools available for performance analysis and feedback for these needs are limited to development tools, which often times do not have much automation or support for accepting code from a crowd.

By creating a tool for automated compilation, test generation and performance report generation, researchers or competition organisers can focus on tweaking the tasks and rewarding great results, rather than dedicating a lot of time to testing every individual solution one by one. By being able to immediately compare the performance of two or more solutions, programmers could exchange code to further improve their solutions.

This project aims to create such a tool, to allow the submission, compilation and testing of code in a variety of programming languages against a given programming task's requirements and constraints. It would offer the means for competition organisers to concentrate on the interesting tasks first, then simply accept submissions from the competitors and get the results in real time. Researchers would be able to maintain a database of solutions to their problems, with metrics on speed, accuracy and resource usage. All of this would be available on the web, with computation done server-side, for a great degree of mobility and stability.

Keywords

C++, competitions, research, compilation, server-side, online interface, Python, Java, C#, SQL Database, security, optimization, performance analysis.

Table of Contents

Abstract i
Keywords i
1. Introduction 1
1.1 Context 1
1.2 Project Renaming 1
1.3 Targeted Audience 1
1.4 Desired Outcomes 2
1.5 Document Details 2
2. Aims and Objectives 4
2.1 Complexity 4
2.2 Objectives 4
2.3 Use Case Scenarios 5
3. Project Design 6
3.1 Changes from proposal 6
3.2 Expected issues 7
3.3 Architecture 7
4. Project Implementation 8
4.1 Web Interface (Code-name WebFrontend) 8
4.1.1 Programming languages and tools used 8
4.1.2 Model types used 8
4.1.3 Code Controller 14
4.1.4 Scoring Submissions 15
4.1.5 Graph Generator 16
4.1.6 Visual Layout and Behaviour 17
4.1.7 Individual Pages 18
4.2 Submission Queue Component (Code-name SubQueue) 20
4.2.1 Programming Languages and Tools Used 20
4.2.2 Storage System 20
4.2.3 Active Behaviour 21
4.2.4 Program Design 21
4.3 Compilation Engine 22
4.3.1 Programming Languages and Tools Used 22

4.3.2 Program Design 22
4.3.3 C# Compilation System 23
4.3.4 C++ Compilation System 24
4.3.5 Java Compilation System 25
4.3.6 Python Compilation System 25
4.4 Evaluation Engine 26
4.4.1 Programming Languages and Tools Used 26
4.4.2 Program Design 26
4.4.3 Scoring Types 29
4.4.4 Test Files 29
4.4.5 Executable Evaluator 30
4.4.6 Python Evaluator 30
4.4.7 Java Evaluator 31
4.4.8 Raw Report 31
4.5 Report Generator 31
4.5.1 Programming Languages and Tools Used 31
4.5.2 Program Design 32
4.6 Security 32
4.6.1 User Account Security 33
4.6.2 Compilation-time Security 33
4.6.3 Security during Evaluation 34
5. Evaluation and Testing 36
5.1 Testing Methodology 36
5.3 Testing on Linux 37
5.2 Unexpected Results 38
6. Future Work 40
7. Conclusions 41
References 42
Table of Figures 43
Appendix 44

1. Introduction

This chapter serves as an introduction to the project and to this document.

1.1 Context

This project was born from the lack of publicly available comprehensive performance review tools for various programming languages. Everything available on the market only fulfils a part of what users need. Services such as Google Code Jam do not offer any sort of compilation performance evaluation beyond comparing outputs. Other tools, such as online compilers, do not give you any sort of performance evaluation or comparison between programs.

Furthermore, many of these projects are not available for offline needs. In a time when performance is extremely important and an objective way of evaluating different solutions to tasks, both in a research environment and in a competitive environment, is crucial, such a product was definitely required.

This project does not attempt to offer automated unit testing or similar testing, as these are not the performance evaluations it is interested in.

1.2 Project Renaming

During development, the project received a name: OCAPE. This is an acronym and it stands for Online Compiler and Performance Evaluator.

1.3 Targeted Audience

Parties interested in taking advantage of this project include researchers whose projects require performance fine tuning, programming competitions, and university computer science lecturers.

Researchers are always on the lookout for even minute improvements to existing algorithms for solutions. An objective tool that allows them to quickly compare different solutions on the same tests and on the same testing environment would save them a lot of time and would give further validity to their research.

Programming competitions on the other hand require an objective way of differentiating competitors' code such that great solutions can be found between the good.

Computer science courses also benefit from a product like this due to how lecturers could present performance improvements in a more visual manner to the students. By clearly showing the improvements of quick sort over bubble sort, for example, students will be able to visualise it a lot better.

Computer scientists in general could also benefit from this project due to its ability to compare the performance of various programming languages in a more casual environment with a minimal number of external factors coming into play.

1.4 Desired Outcomes

The final output of this project is a piece of software that could be given several pieces of code that all share the same desired output, and then objectively describe the performance given a predetermined set of tests. It would function in a headless manner with a web interface for controlling its functionality.

It should, if possible, run on both Windows machines and Linux machines.

It is my wish that this project be of high quality, as I would want to potentially commercialise it to interested parties. Possibilities include either licensing the software itself, or, alternatively, renting out servers with the software included.

1.5 Document Details

Chapter 1 serves as an introduction to the entire project and document.

Chapter 2 outlines some updates to the goals of the project, as described in the Project Proposal.

Chapter 3 presents changes to the design decided upon in the Project Proposal.

Chapter 4 offers an in-depth look at the actual development deliverables, as well as further design decisions done during implementation.

Chapter 5 deals with the various testing methodologies required to ascertain the quality, stability and predictability of the system.

Chapter 6 gives an overview of the work that might be done to the project in future iterations.

Chapter 7 has the personal conclusions at the end of this development run.

Introduction

Most figures presented in this report have been generated through the Visual Studio 2013 Architecture Diagram features available.

Readers interested in the technical implementation of the project should focus on chapters 3 through 5.

Readers interested in a commercial use-case description of the project will find more relevant information in chapters 2, 5 and 6.

Readers interested in the project’s unexpected programming language specific findings can read a selection in chapter 5.

2. Aims and Objectives

This chapter outlines general goals and objectives of this project, including commercial aims and quality targets.

2.1 Complexity

Due to the complexity of this project successfully creating a fully comprehensive solution is a really complicated task. There are many things one can evaluate about the performance of a program, including, but not limited to, how fast the program runs, how much memory it uses, how much space to store the resulting executables, and how many lines of code required. Beyond that, protecting the system from malicious programs can prove to be a lot of work.

There are also many programming languages that can be supported and it will be hard to argue that one programming language would deserve less support than others. By allowing objective comparison between different programming languages, users will be able to clearly see where their preferred programming language excels or where it fails.

2.2 Objectives

The main objective of this project is the creation of a commercial-quality product for evaluating the performance of third party submissions to various user-defined tasks. For that purpose, some important sub-objectives are required.

The user interface itself should also be able to present everything in a comprehensive manner while also maintaining user-friendliness. A good interface facilitates fewer problems during usage.

Security is also very important as allowing third-party source code to be compiled and run on a server could result in complete system breakdown. Users should be able to trust the server’s execution.

Last but not least presenting all the evaluation results in a decent manner can be beneficial to everyone. Statistics could be generated from all the results and a lot of insight can be gained. The project’s final aim is the creation of information and that information must be presented well.

The project proposal initially described the project as supporting only C++ and potentially Java. However this is not to say that the project should be limited to those two. Implementation has allowed for the addition of more supported programming languages at a minimal cost.

Aims and Objectives

Most of the goals and objectives from the project proposal have not changed. Please refer to that document for more information.

2.3 Use Case Scenarios

A typical scenario for this project is:

- Lecturer A wants to see how well their students understood the various sorting algorithms taught to them. He sets up a new task, called "Sort this array", describes the task and uploads a few test cases (combinations of inputs and their correct outputs).
- Lecturer A also sets that he allows participants to see other participants' results in the leader board.
- Student B thinks they have a good solution, written in C++, and uploads it to the task previously set up by lecturer A.
- After a few seconds, student B refreshes the page and sees the results of his code. They include some run times, some memory usages and how many tests he got right.
- Student C attempts the task, but their solution is in C#.
- After a few seconds, they have access to their report as well. Not only that, but both Student B and Student C can now compare their results. Student C notices how his solution is faster, but uses more memory, than student B's solution. Conversation ensues.

Other possibilities include researchers maintaining a constant testing bed for potential improvements to their existing algorithms. Or hosting programming competitions similar to the International Olympiad in Informatics. [1]

Some sample domains that would benefit from a tool like this include: machine learning, where newer algorithms could attempt to learn from data and put the knowledge in practice against validation data, gaining an accuracy value to compare against other submissions; comparing worst-case and best-case scenarios for algorithms with wide variance in behaviour; or the constant search for improvements in the speed of compression algorithms.

3. Project Design

This chapter describes design decisions done between the final version of the project proposal and this version of the final report. Most of the design decisions from the project proposal have remained unchanged. Please refer to that document for more information on the design.

3.1 Changes from proposal

There have been however some fairly important changes that will be described here. First and foremost on the changes to the compilation engine.in the initial design the compilation engine will only support C++ and possibly have a different implementation for Java. As a result of good coding practices I have decided during development that adding support for more languages was viable. The new design now allows for multiple supported languages, currently implemented being C++, Java, C# and Python.

Another addition to the original design is the way the programs are compared against a correct outputs. The feedback component is no longer responsible for ascertaining program performance. All of these responsibilities have been moved to the evaluation component. The feedback component is now responsible for creating a comprehensive JSON file and feeding it to the web interface for storage in the database.

An element of design that research on the types of tasks organisers might require deemed less useful was the generation of tests. Not only would most researchers or competition organisers already have tests ready from their own preparations, the wide variety of possible tasks makes a catch-all solution to test generation almost impossible. Methods to easily manage the tests to a task have been the focus instead.

While the project proposal only described binary files as outputs of the compilation engine, due to the addition of Python and Java interpreted and non-natively compiled files are also potential outputs of this component. This is important to note due to the fact that Python for example does not actually get compiled and its source code gets immediately sent to the evaluator.

A lot of changes were also made to the security side of the project. Due to further research into the area of sandboxing, it has been determined that implementing a dedicated sandbox is not viable project. As a result that feature has been scrapped and alternatives were researched. Said alternatives can be considered improvements.

3.2 Expected issues

Beyond the possibility of complete server takedown due to malicious code which should be handled by security measures implemented, many problems were not expected.

Porting the project from one server to another might prove problematic due to the many tools required for a successful run, but could most likely be solved through the use of a comprehensive installer. Such an installer could ensure that all required prerequisites are installed, that any PATH variables are set correctly and that all components are up to date.

3.3 Architecture

Nothing has changed from the initial project proposal. The same five modules have remained: the web interface, the server side queue engine, the compilation engine, the evaluation engine and the feedback component, now renamed the report generator.

The entire project is still done in an object-oriented manner where advisable, with some components requiring a single type (or even function) purely due to their simplicity.

Every single component, except for the Website Interface, is its own executable file. This is done to minimise potential failures and to quickly understand where problems are originating from. This also allows for some innovative solutions in sandboxing the behaviour of these programs, as well as allowing for a ‘plug-and-play’ method of swapping between modules should better ones arise. While some very small amount of time might be lost due to the overhead of starting up every single module every time, the flexibility this brings to the project is a lot more valuable.

Coding-wise, the design decision was to use as little code as possible for tasks that don’t require any particular level of complexity. If there is a clear task to be solved and said task does not get repeated in other parts of the project, a single iterative function should prove enough. This allows for the minimisation of resources used during runtime by eliminating function overheads, object overheads and function clean-up times.

More details will be available in the implementation chapter.

4. Project Implementation

This chapter describes the development stage and final deliverables of the project. Each subchapter is dedicated to a single component.

4.1 Web Interface (Code-name WebFrontend)

4.1.1 Programming languages and tools used

By weighing the various options available for developing the web-facing portion of the project, the following options remained viable: PHP using the Bolt framework [1] and C# ASP.Net MVC 5.

Both presented in-built authentication and user role management.

Bolt offered a more user-friendly approach to managing various object types, while ASP.Net offered greater stability and version control of object types.

ASP.Net, due to it being part of the .Net family of technologies, offered unparalleled access to the underlying operating system features, such as storage access and modification.

Mostly due to that last feature, as well as its support in Visual Studio, the platform chosen for development was ASP.Net MVC 5 using C# for logic and HTML + CSS + JavaScript for web development.

Figure 1 - Microsoft .NET MVC 5

The IDE of choice was, as mentioned above, Visual Studio 2013. No extensions to the base product were required.

4.1.2 Model types used

Beyond the base models provided by Visual Studio for defining users (and used in authentication), a number of other models were required for defining the product's functionality.

All of these models were defined and stored in a database using the Entity Framework and Code-First approaches to development. The Entity Framework allows a user to describe a model type in C# code, then automatically have said code generate the appropriate database structure, independently of what database the developer decides to use. Code-First allows for changes to the database to happen using a system similar to versioning software, by saving any modifications as a migration and also allowing for roll-backs to previous versions.

There are four project-specific models, as well as one supplied by Visual Studio by default.

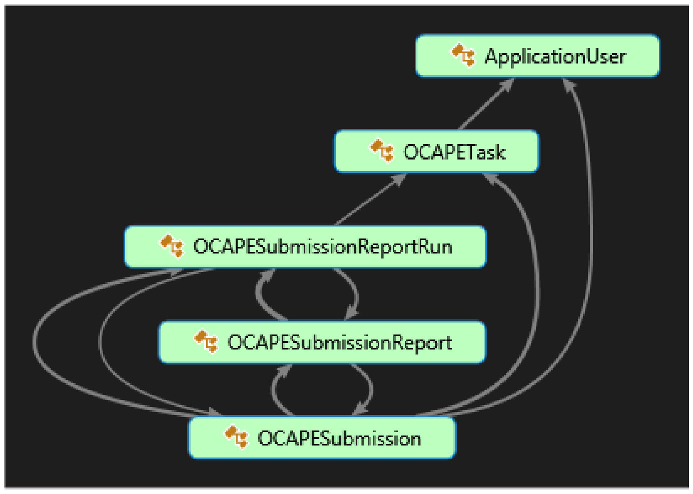


Figure 2 - Website Interface Model Interaction

4.1.2.1 OCAPETask

First and foremost is the Task type. This allows an organiser to define a new programming challenge, as well as set its constraints, scoring system and deadlines.

It holds a large array of information. Title, Description, Creation Date of the task, the task's Start and Deadline Dates, the Owner of the task, whether it is public or not, whether participants have access to the live leader board, evaluation constraints and scoring weights.

It also offers a single method, called GenerateTaskDetailsFolder(), that stores important evaluation information to the storage system for use with the other project components.

The evaluation constraints are the maximum memory usage permitted to the program, the maximum run time it is allowed to take, as well as how many times should a program be evaluated on the same test (for better averaging purposes).

The Task object's properties can be changed by the Task owner any time during run of the event. Any existing submission will not be re-evaluated, but they will have their score be updated to reflect any changes in weights or constraints.

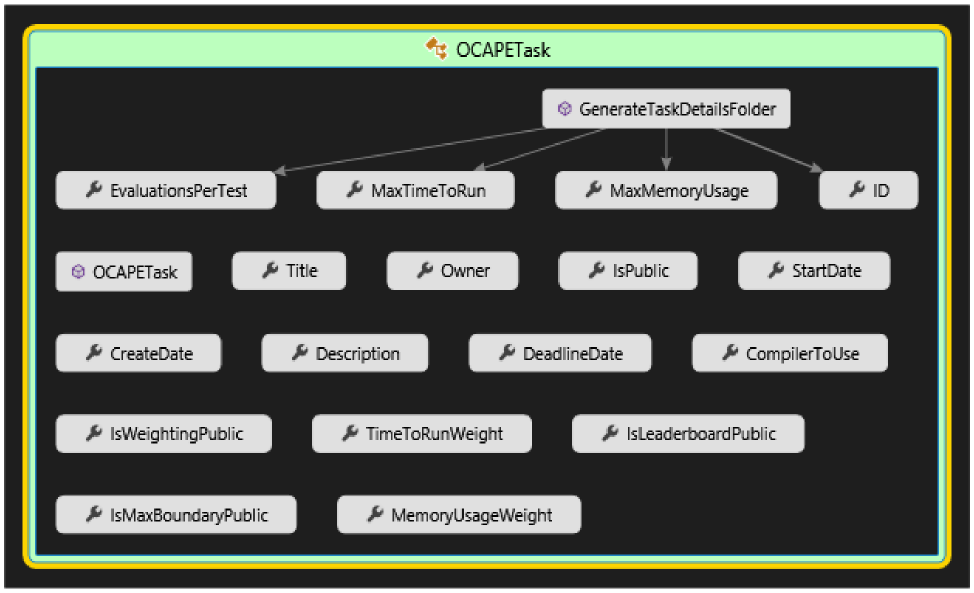


Figure 3 - OCAPETask Properties and Methods

4.1.2.2 OCAPESubmission

Any time a user decides they have a piece of code to run against a task, they upload it to the server and a Submission object is generated. This stores general information about the code: the task it has been submitted against, the code filename and its extension, the owner of the submission and any notes they may have added, as well as the submission date.

There is a single method, called UpdateScore(), and that is used to retrieve the task's scoring weights and evaluation constraints, then update the submission's score accordingly. It does nothing if the submission has not been evaluated yet.

Each submission is tied to one or zero Submission Reports. Zero if the evaluation has not completed yet, one if the evaluation has completed and a final report for the submission exists.

Every single file uploaded to create a submission is immediately renamed to "submission" followed by the ID value of the new OCAPESubmission generated for it, then followed by its original extension. This allows for smooth travel through the operational pipeline.

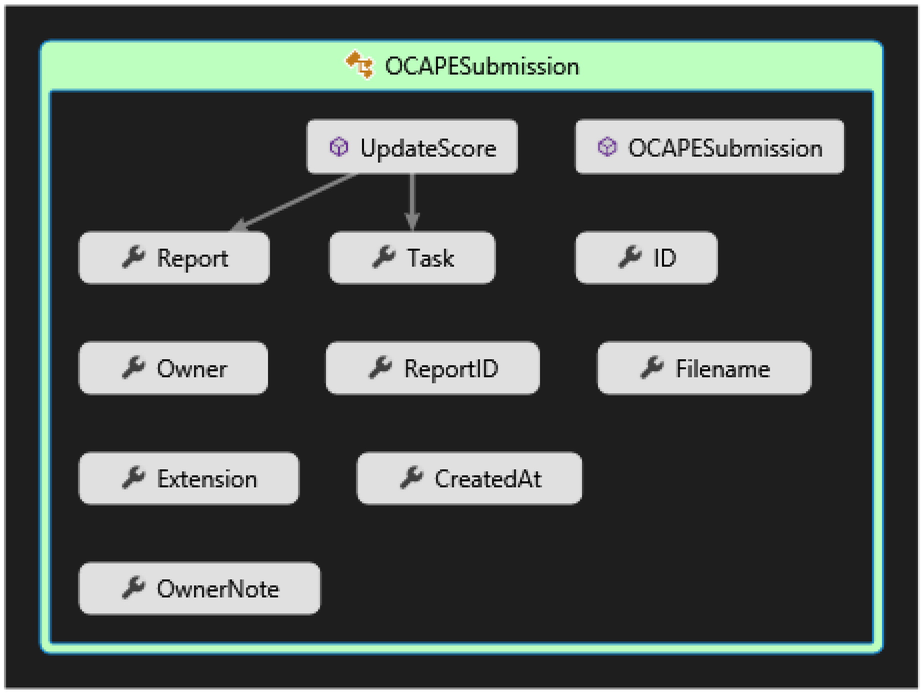


Figure 4 - OCAPESubmission Properties and Methods

4.1.2.3 OCAPESubmissionReport

Once a submission has been successfully evaluated, a Submission Report is generated. This submission report holds information on whether the code compiled successfully and its evaluation score. It also holds a list of Submission Report Runs, detailing individual results for each test, as well as a link to the submission object it describes.

There are two methods available: AverageRuntime() which gets the average run time of all submission report runs, as well as AverageMemoryUsage() which does the same, but for memory usage. These methods are very useful when generating statistics.

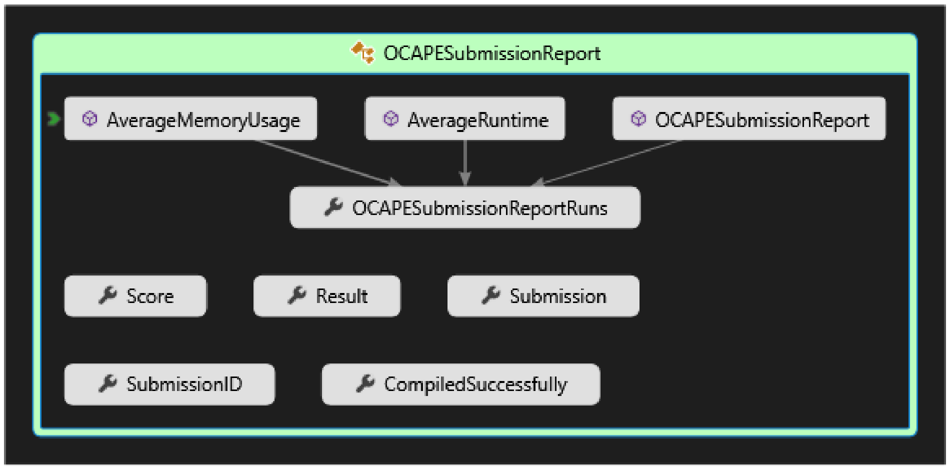


Figure 5 - OCAPESubmissionReport Properties and Methods

4.1.2.4 OCAPESubmissionReportRun

Each test the evaluation component goes through results in an object of this type. It stores independent test results, such as whether it resulted in the correct output, what its peak memory usage was and how long it took to run to its end.

There are two methods present in this object type: IsWithinConstraints() which describes whether this run was within the constraints set up by the task owner, and the static GetFromJSON(string) which translates a JSON object to an object of type OCAPESubmissionReportRun.

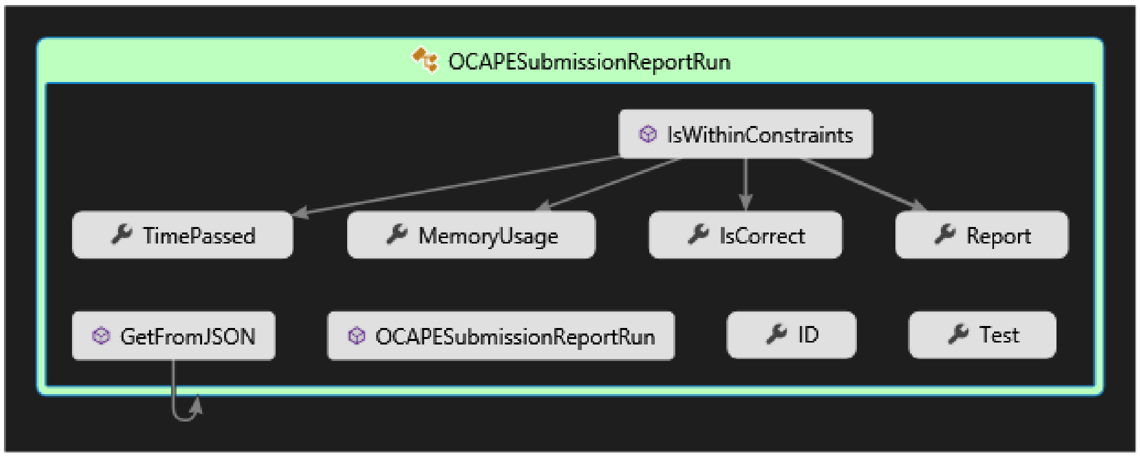


Figure 6 - OCAPESubmissionReportRun Properties and Methods

4.1.2.5 ApplicationUser

This object type is supplied by Visual Studio and describes a user of the system. It is the object type the authentication system checks against when attempting registration or logging in. It is also the object type that allows differentiation between Administrators, Event Organisers and Participants.

It is directly linked to OCAPETask and OCAPESubmission in order to describe the owners of a task and / or a submission.

More information on the user authentication system can be found in chapter 4.6.1.

4.1.2.6 Links between the object types

OCAPETask contains a link to the ApplicationUser that created it. This is to allow correct checking of ownership when accessing administrator functions, such as editing the task or deleting submissions.

OCAPESubmission contains a link to the OCAPETask it has been sent against, a link to the ApplicationUser that created the submission and a link to the OCAPESubmissionReport that will be generated once evaluation is complete (this is null until that happens). The ability to know the ownership of a particular submission is very important when wanting to reward improvements or good code.

OCAPESubmissionReport contains a backlink to the OCAPESubmission it has been generated for, as well as an entire list of OCAPESubmissionReportRun. This allows for comprehensive book-keeping of all submissions and their results.

OCAPESubmissionReportRun contains a single link back to the OCAPESubmissionReport it belongs to.

4.1.3 Code Controller

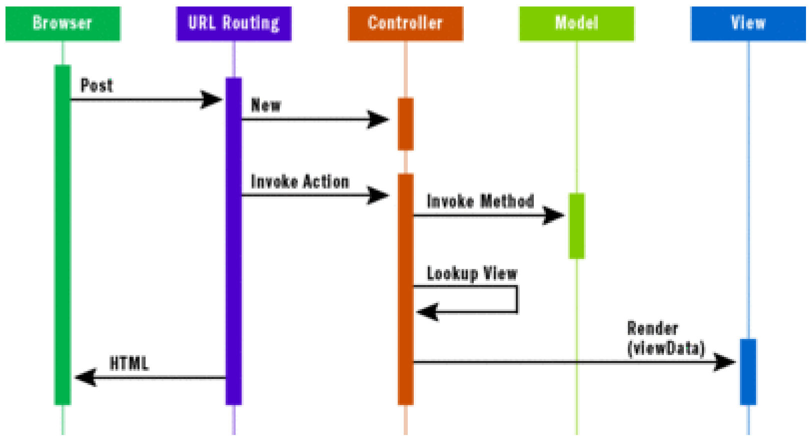


Figure 7 - MVC Request Flow [10]

The MVC framework relies on splitting any object type information from layout information and from business logic. The TasksController handles all the business logic in this module.

It has methods for redirecting incoming requests to the appropriate code behaviours. All website logic is handled by this single controller, thus any non-default code is under the {Base-URL}/Tasks/ URL.

These include, but are not limited to:

- Uploading code submissions for a task results in generating the required underlying storage structure for accepting new code files and associating them to the correct user, then saving the received files with correct filenames.
- Creating, Reading, Editing and Deleting of Tasks is handled in the standard MVC way, with views for both GET behaviour and POST behaviour.
- The leader board can be accessed by anyone with permission to do so.
- Old submissions without any reports or without any successful runs can be purged by administrators or task owners.
- Tests can be added by task owners to further improve on the testing phase.
- Submission Reports are generated when the system detects the fact that evaluation is completed and the user requests to see them.
- Graphical representation of statistics are also generated when requested through the custom-made graph generator.

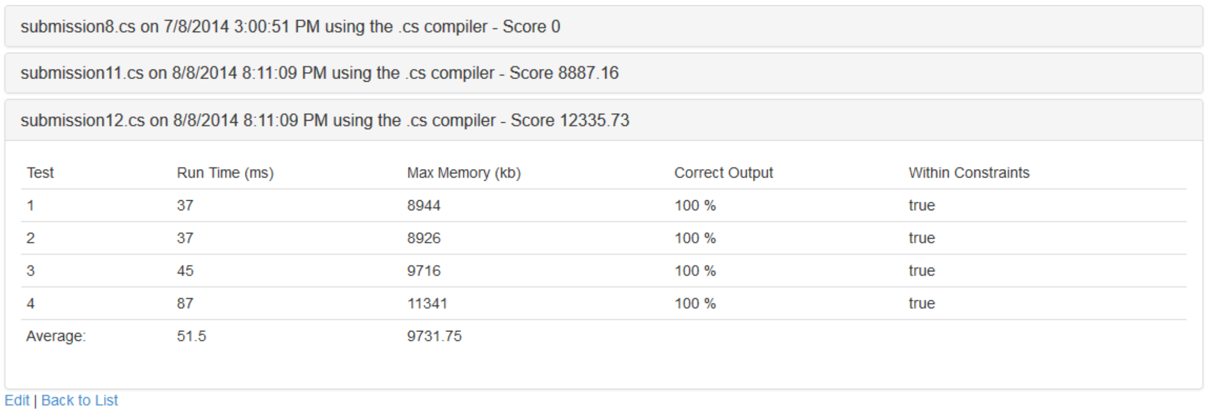
A feature of particular importance is the way in which the website communicates the evaluation parameters of a task to the evaluation engine. Whenever a task is created, or its settings are altered, a file is updated with the appropriate properties in the “Tasks” root-folder, under a sub-folder named after the task's ID value. This optimises the entire process, as reading from a file is a lot faster for a C# application than reading from a database.

4.1.4 Scoring Submissions

There are two scoring methods: absolute and partial. Absolute scoring requires programs have perfectly correct output. A single difference and the whole test is scored 0. Partial scoring allows errors in the output and scores the test on a similarity factor.

The scoring system makes use of weight values to decide what is more important to the organiser. There are two weights that can be modified: the run time weight and the memory usage weight. These make use of the constraints to calculate a final score for a submission. The formula is as follows: [(Maximum Runtime – Actual Runtime) * Runtime Weight + (Maximum Memory Usage – Actual Memory Usage) * Memory Usage Weight] = Final Score (assuming the output is correct).

Weights of 0 could be set to completely focus on accuracies rather than memory usage or runtime (unless a submission's runtime is higher than the task's limit). This is useful to emulate the scoring style of IOI-like programming competitions, where a number of tests (usually 10) are used and submissions can get 10 points for each test they solve within the memory and time limit given, up to a maximum of 100 points.



submission8.cs on 7/8/2014 3:00:51 PM using the .cs compiler - Score 0

submission11.cs on 8/8/2014 8:11:09 PM using the .cs compiler - Score 8887.16

submission12.cs on 8/8/2014 8:11:09 PM using the .cs compiler - Score 12335.73

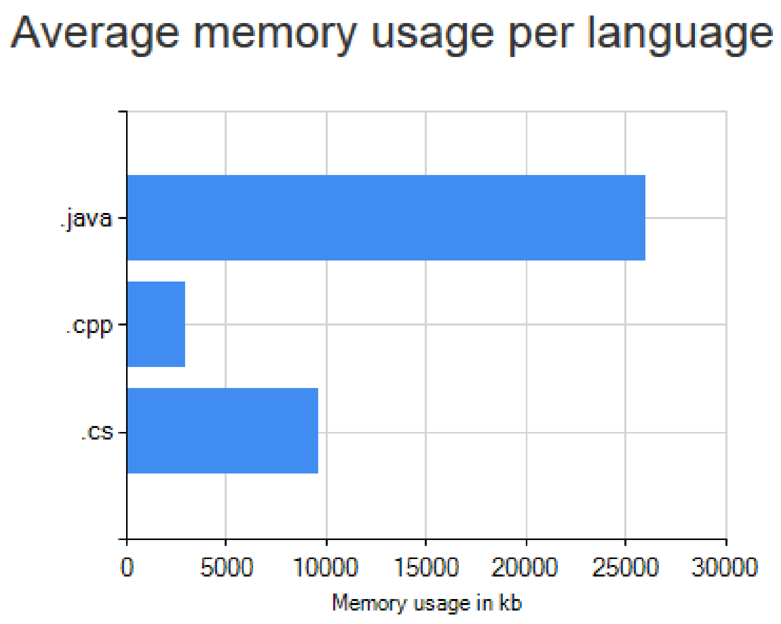
Test	Run Time (ms)	Max Memory (kb)	Correct Output	Within Constraints
1	37	8944	100 %	true
2	37	8926	100 %	true
3	45	9716	100 %	true
4	87	11341	100 %	true
Average:	51.5	9731.75		

Figure 8 - Example of a Submission with 100% Accuracy

Whenever a task's properties are edited, the scores of each submission are updated to reflect potential changes to weights and limits.

4.1.5 Graph Generator

There is an extension to the code that receives a list of Submission Reports and is able to generate a number of different visual graphs describing those reports. These allow for a better understanding of the various submissions' behaviour, as well as any issues some programming languages might be facing.



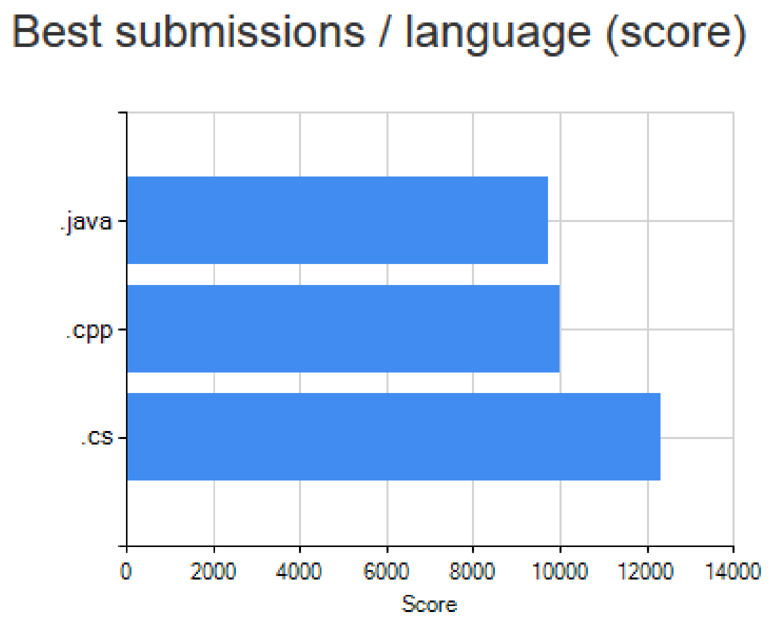


Figure 9 - Examples of Available Graphs

The graph types available are:

- Best runtime for each submission language. It presents the fastest submissions for each language in a horizontal bar graph.
- Average runtime for each submission language. It offers an overview of the average performance of all submissions, grouped by language.
- Average memory usage for each submission language. Does the same as the previous graph, but for memory usage.
- Best submissions for each language. Selects the best three submissions (based on their runtime) for each available programming language and arrays them in a bar graph.
- Best submissions for each submission language by score. Selects the best scoring submission for each programming language available.

These graphs can prove very useful when attempting to showcase improvements in an algorithm or advantages of one implementation over another.

4.1.6 Visual Layout and Behaviour

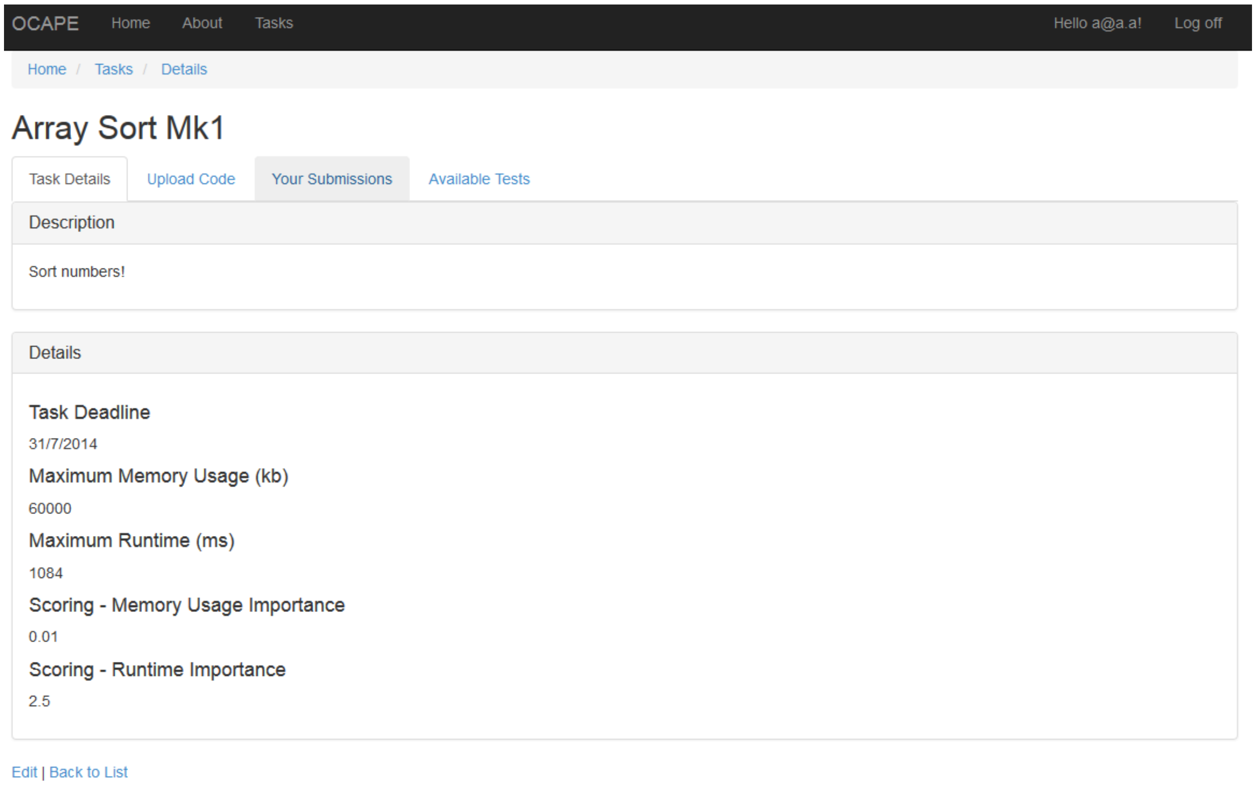
For presenting all this information to the user, the project makes use of HTML and CSS. A framework is employed, Bootstrap [2], that handles a lot of beautification of visual elements by overwriting the CSS code of elements, as well as adding a lot of new elements. An interesting side effect of using Bootstrap is the automatic support for mobile devices. When accessed from a smartphone or a tablet, the layout rearranges itself to look presentable.

JQuery is a JavaScript library that allows more interactions to be possible between users and the page, for greater control, greater interactivity and better rendition of information. It is used intensively for validating form controls and for organising information in a more readable fashion.

The resulting web pages can be accessed in most modern web browsers. JavaScript support is not compulsory, but it does help a lot in improving a user’s experience.

Accessing any pages that the user should not access will result in a 404 (page not found) error. This is to give the false impression that said page does not actually exist, rather than notifying the user of a hidden wall that is blocking them. Under normal behaviour, users should never encounter links that would send them towards restricted pages unless they have permission.

A lot of care has been put into presenting all available information in as clear a layout as possible. Users are able to quickly understand where their submission failed, how much they scored and where they are positioned in relation to other users.



Array Sort Mk1

Description

Sort numbers!

Details

Task Deadline
31/7/2014
Maximum Memory Usage (kb)
60000
Maximum Runtime (ms)
1084
Scoring - Memory Usage Importance
0.01
Scoring - Runtime Importance
2.5

Figure 10 - Example of the Details Page to a Task

The only element that was, from the design phase, chosen to not be visualised, is the current status of compiling and evaluating a submission. To speed up the process and to minimise the chance of failure due to probing for status updates, users simply see “No report available” until evaluation is complete.

4.1.7 Individual Pages

There is a number of pages used to create the user experience.

- The Home-Index page is used as a landing for new and returning users, offering quick access to some amount of information about new tasks, as well as links to logging in for unlogged users.
- The Home-About page has a short description of this project.

- The Tasks-Index page lists all the tasks a logged in user has access to. Should the user not be logged in, this redirects to the login page.
- The Tasks-Detail page offers all the information on a particular task, as well as offers the possibility of uploading a new submission against that task. Users with existing submissions to the given task can review the results on this page. All functionality (task information, upload submission and review results) is divided into several tabs. Each submission can be expanded to reveal results on a per-test basis. This allows for in-depth understanding of the submission run. Uploads a user submits are stored in a single folder under the “Uploads” root-level folder. More information on this structure is available in section 4.2.2. Should a submission not have a report generated yet, it offers a relevant message describing this situation.
- The Tasks-Edit page offers the task owner the chance to amend the details of a particular task. It offers input validation, as well as JavaScript controls to improve data input. It also allows the owner to upload new tests to the system, as well as remove any tests they no longer deem needed.
- The Tasks-Create page offers a select number of users the ability to create a new task. The form looks almost identical to the Tasks-Edit page, as the information entered is the same.
- The Tasks-Leaderboard page presents an overview of the best scoring submissions to a given task. All submissions are sorted by their total score. Each submission can be expanded to reveal results on a per-test basis, similar to the Tasks-Detail review page.
- The Tasks-Graphs page creates and presents a number of graphs for the given tasks. These graphs are described in section 4.1.4.
- The Accounts-Login page allows a user to login to the system. It also offer a link to the registration page. More information on the user authentication system can be found in section 4.6.1.
- The Accounts-Register page allows the creation of a new account for simple user access to the system. More information on the user authentication system can be found in section 4.6.1.

4.2 Submission Queue Component (Code-name SubQueue)

4.2.1 Programming Languages and Tools Used

The programming language used for the submission queue component is C# .Net. .Net languages offer great access and control over the underlying operating system (Windows in most situations. UNIX possible by using the Mono project). This project was developed in Visual Studio 2013, although it does not take advantage of any particular feature present in the IDE.

4.2.2 Storage System

For the test server, the underlying storage system is NTFS. However, there is no reason FAT32 or other storage architectures supported by Windows (potentially EXT3 or EXT4 if running under UNIX) would not work. Design choices ensure the usage of file-system independent functionality.

The underlying file system must, however, be directory-based, as that is how various submissions and outputs are arranged and distributed.

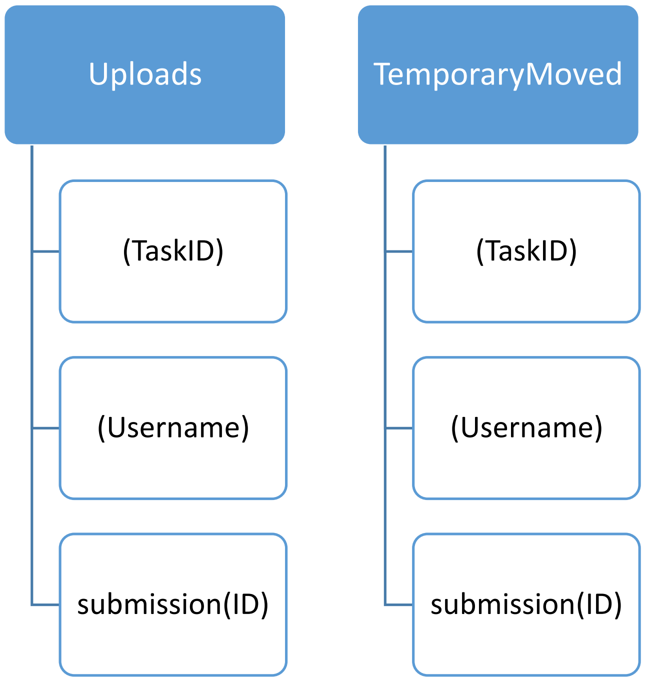


Figure 11 - File Storage for Submission Queue

The Submission Queue Component makes use of two root-level folders: “Uploads” and “TemporaryMoved”. Uploads is where folders for each individual task are stored. These folders are named after the ID of that task. Each task folder contains a folder for every user that has submitted code against it. “TemporaryMoved” has an identical structure to “Uploads”.

All submissions, at the end of an event, are available in the appropriate task subfolder in “TemporaryMoved”. This allows an event organiser to create an archive, or, alternatively, check source code of specific individuals.

4.2.3 Active Behaviour

The Submission Queue is the only module beyond the Web Interface that should always be started.

While running, the application polls the underlying file system for any files in the "Uploads" sub-folders. It does so by checking every single sub-folder within the root-folder for any files, then checking whether they are correctly named and stored. Any incorrect files are discarded.

There is a grace period of 1 second between checks, to preserve system resources. Faster checks are not required.

Once a legitimate submission is found, it is immediately moved to the appropriate "TemporaryMoved" sub-folder, deleted from the "Uploads" folder, then passed to the Compilation Engine. The Submission Queue then waits for the Compilation Engine (and every other component that starts as a result) to finish its run. All polling for new files is paused until the current submission is complete. This is as a result of a design decision: the number of submissions is unlikely to pile up in most scenarios, thus ensuring swift evaluation of all pending submissions in good time.

The behaviour could be changed in the future to allow multiple evaluations at the same time, up to the number of CPU cores available to the system. This would improve performance only in situations where too many submissions are being presented to the system.

4.2.4 Program Design

The whole program is designed as a single static function that contains a single loop. Every loop iteration is in charge of monitoring for new submissions, checking for their legitimacy, then sending correct submissions to the Compilation Engine.

All the file system checks are done through the System.IO library. This library offers access to the static Directory and File classes, each with functionality to checking for, copying, deleting and accessing directories and files respectively.

Sending a submission to the Compilation Engine is done by creating a new object of type Process, part of the System.Diagnostics .Net library. Two parameters are passed to the Compilation Engine process: the path to the correct file in the "TemporaryMoved" folder, as well as the ID of the task this submission is for.

Should the Compilation Engine fail from an unexpected issue, nothing is done to handle that scenario, as, in that case, there has been something very wrong to cause it. All non-critical errors are handled by the following components in the chain, starting with the aforementioned Compilation Engine.

The result of this code is a single executable file with no direct dependencies beyond .Net being installed. This executable file could be started by the website interface (given appropriate admin access), but it must currently be started manually.

4.3 Compilation Engine

4.3.1 Programming Languages and Tools Used

Similarly to the Submission Queue component, the Compilation Engine is written in C# .Net and developed using Visual Studio 2013. The comprehensive Process object type is extremely useful when invoking the various compilers required.

4.3.2 Program Design

The program behaviour is split in two major components: the general pipeline and the specialised compilation object. The general pipeline analyses the submission it receives, then decides on what compilation system to make use of.

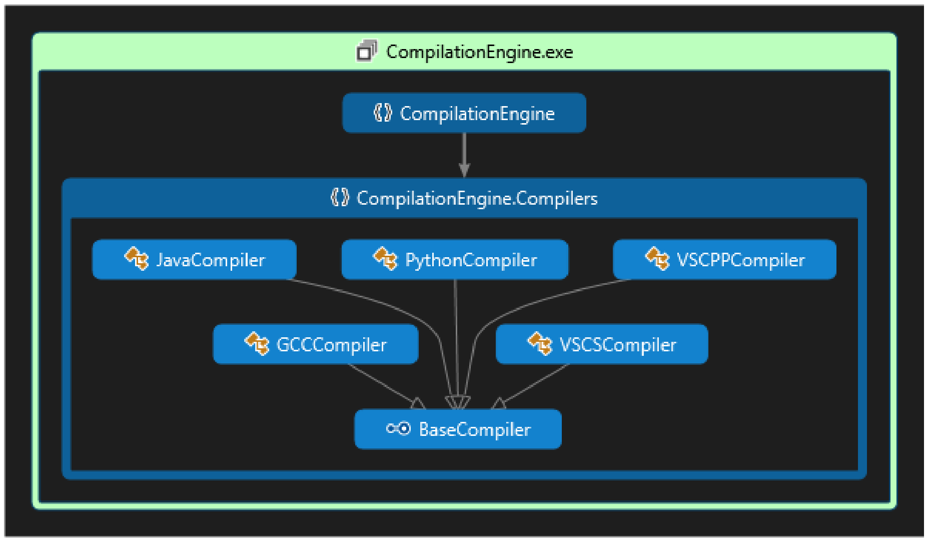


Figure 12 - Compilation Engine Object Types

All compilation objects are based on the BaseCompiler interface. This interface requires all compilation systems to implement a function to decide whether they accept a given extension, called "AcceptsExtension(string)", a function to retrieve the resulting output file's filename, called "GetOutputFile()", as well as a function to compile a given file, called "Compile(string, string, string)". This interface could also potentially allow third party compilation libraries to be developed and loaded by the Compilation Engine should such a feature be implemented.

All objects implementing the BaseCompiler interface make use of the Process object type for invoking the appropriate compilers and waiting for their completion. This allows for greater control over the compilation process, as well as better error-handling should the required software not exist.

A single compilation system is used for a single submission.

Once compilation is successful, the program passes the output file generated to the Evaluation Engine, alongside the appropriate task ID, and waits for it to complete its tasks. All output files are stored in the appropriate subfolder in the "ExeOutputs" root-folder. This folder's structure is identical to the "Uploads" and "TemporaryMoved" folders.

The result of this code is a single executable file with no direct dependencies beyond .Net being installed. This executable file does not need to be started manually at any time, as its behaviour is completely controlled by the Submission Queue component. Unless something unexpected happens, the application should always exit cleanly after attempting to compile code, evaluation successful code and generating the report.

4.3.3 C# Compilation System

C# code is passed on to an object of type VSCSCompiler. This compilation system allows files with the extension ".cs" and creates an executable file.

Under Windows, it makes use of the Visual Studio "csc" C# compiler by invoking the Visual Studio developer batch terminal script and passing the appropriate arguments to the csc program in a cmd.exe (Windows Command Line) process. Said batch terminal script is included with the OCAPE software and has been copied from a Visual Studio 2013 install. Under Linux, the Mono-Project "dmcs" compiler is used with similar parameters.

Successful compilation results in an error code of 0.

This compilation system requires that Visual Studio 2013 be installed on the server system. It does not require any particular pathing, as the VS developer batch terminal script handles all of this by itself.

4.3.4 C++ Compilation System

C++ code has two possible paths to take: either towards an object of type VSCPPCompiler, or towards an object of type GCCCompiler. Both compilation systems accept files with the extension ".cpp" and create an executable file.

The VSCPPCompiler also makes use of the Visual Studio developer batch terminal script, similarly to the C# compiler described in section 4.3.3. However, it does not invoke "csc". Instead, it requires "cl", the Microsoft C++ compiler. It makes use of several arguments to optimise the good in a reasonable fashion. These are: "/F268435456" for setting the stack size, "/EHsc" for setting the exception handling model to one relevant to the task, "/O2" to generate faster code through compiler optimisations, "/GA" to further optimise code for Windows environments, and "/W4" to enable warnings for debugging purposes. This compiler is not available for Linux systems.

The GCCCompiler, under Windows, makes use of the open source GNU C++ compiler [3] and its Windows implementation from the MinGW project [4]. All required executables and libraries for the compiler come with OCAPE by default, as a means of offering a default compiler to use out of the box. It functions by invoking "g++.exe" with several parameters, some including "-O2" for optimisation, "—stack=268435456" for setting the stack size and "-s" for static linking. Under Linux, it makes use of the same GNU C++ compiler, but through its native Linux version, supplying identical parameters.

Between the two available C++ compilers, the Microsoft one is recommended, as it has a history of having fewer behavioural bugs in its resulting executable files.

Successful compilation of either system results in an error code of 0.

Using the g++ compiler does not require any additional software be installed on the server system.

Using the Microsoft cl compiler requires that Visual Studio 2013 with C++ support be installed. No further pathing instructions are required.

4.3.5 Java Compilation System

Java code is sent to an object of type JavaCompiler. This system accepts single files with the extension ".java" and creates a Java object with the extension ".java".

Due to severe limitations in the way Java handles its classes, simply compiling the submission file would not result in a usable Java class file. Java requires that a file called "MyFile.class" hold a Java object type named "MyFile". While the user might have name their file "MyFile.java" with the type "MyFile" in it, the upload system will have renamed the file to something along the lines of "submission230.java". This file would result in "submission230.class", as the Java compiler, "javac", does not allow for the renaming of the inner object type, or renaming of the output file while preserving the inner objects.

As a result, the JavaCompiler object, before passing the submitted file further, must search through the file, replace any instances of the object definition with the expected new object type ("submission230"), then save the file and continue. For the previous example, "public class MyFile" would be replaced with "public class submission230".

By making this modification, the system ensures that the resulting file will be a valid Java byte-code program and will run as expected.

Once the source code submitted has been altered, it can then be passed on to the "javac" program for compilation into a ".class" object. No further parameters are supplied beyond the parameter overriding the default directory to compile to.

Behaviour is identical between Windows and Linux.

Successful compilation of the code results in an exit code of 0.

4.3.6 Python Compilation System

Python code is sent to an object of type PythonCompiler. This system accepts files with the extension ".py" and creates a file with the extension ".py".

Python is used in this scenario as an interpreted language. No binary file is created from the supplied source code, thus the system is simply left to copy the given submission file from its original location to the output folder, without any changes mid-way.

In the future, this compilation system could be renamed to “InterpretedCompiler” and be used for other similar languages, such as JavaScript, LUA or PHP.

Unless something unexpected happens during the copying of the file, this compilation system always returns an exit code of 0.

4.4 Evaluation Engine

4.4.1 Programming Languages and Tools Used

The evaluation engine, similarly to the other tools, is developed in C# .Net due to the plethora of tools available for running and checking resources used by other processes. It is developed in Visual Studio 2013.

As it will be shown in the Evaluation chapter, the choice of C# over other languages has proven to be most beneficial to the project, as it is currently extremely well optimised for running on the Windows platform, with some scenarios, particularly potentially time-intensive, such as probing a process for its current memory usage, being orders of magnitude faster than other languages.

Having as little overhead as possible is critical when attempting to evaluate the performance of a process, and this has translated in the design of the application.

4.4.2 Program Design

The entire application is run in a single thread, each test being done one at a time. The submission it is evaluating is run asynchronously, to allow for different data collection mechanisms to function.

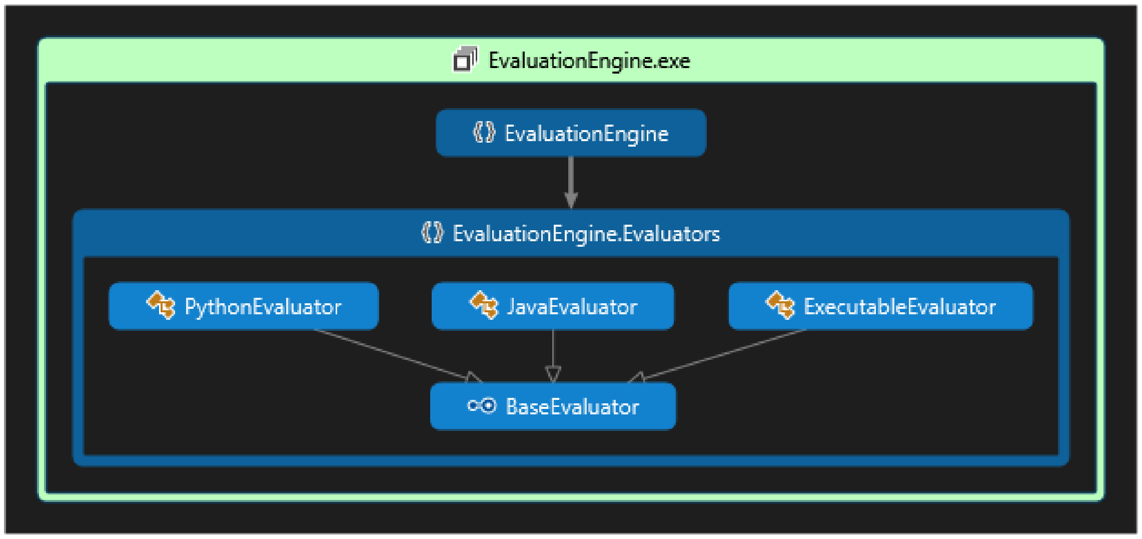


Figure 13 - Evaluation Engine Object Types

The application takes two arguments: the file path to the submission to evaluate, and the task ID of the task it has been submitted against. It then uses that task ID to search for the task's evaluation parameters. These are the ones saved during task creation (see section 4.1.3) in a file dedicated to said task. The relevant properties are:

- Maximum Runtime: how many milliseconds the submission is allowed to run for.
- Maximum Memory Usage: how much memory the submission is allowed to use during its run.
- Number of Evaluations: How many times the submission should be run against the same test, for averaging purposes.
- Scoring Type: 1 for absolute scoring. 2 for relative scoring.

All of these properties are described in-depth in this chapter.

Depending on the type of submission, an appropriate evaluator is chosen. At this time, there are enough evaluators to satisfy the available outputs from all compilation systems. These are: an Executable Evaluator that tests executable Windows binaries; a Python Evaluator that tests ".py" Python scripts; and a Java Evaluator that tests Java Byte-code stored in ".class" files. All of these evaluators are all implementing the BaseEvaluator interface, which requires them to implement the following functions:

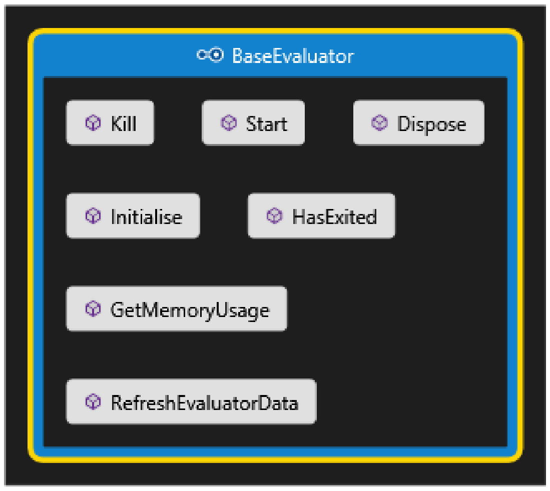


Figure 14 - Base Evaluator Interface

- Initialise(string) – takes the path to the file to test and sets the environment up.
- Start() – begins evaluation.
- RefreshEvaluatorData() – some evaluators require the invocation of a refresh function to update any data they have. This function allows an abstraction of it.
- GetMemoryUsage() – returns the amount of memory the submission is currently using, in kb.
- Kill() – forcefully stops the running of the submission.
- HasExited() – returns whether the submission is still running.
- Dispose() – clears any resources being used by the evaluator.

Within the same directory as the task parameter file, the Evaluation Engine searches for valid tests to run. Once it has found a new valid test to run the submission against, it copies the required files and starts the evaluation. Once the evaluation ends, the submission outputs are compared against the correct outputs (also stored in the same directory as the input files retrieved earlier) and various metrics are stored. This process is repeated a number of times equal to the Number of Evaluations parameter described above.

By running the same test multiple times and averaging the results out, the performance penalty of situations where a system failure stopped a submission from working optimally for a split second is minimised. It also helps smooth out submissions that might make use of stochastic algorithms and whose behaviour might vary wildly from run to run.

Once a test run has ended, the evaluator searches for the existence of the next test and repeats the above process. Once all tests have been expended, the stats are tallied up and the results saved to a raw Comma-Separated Values file (".csv") and passed on to the Report Generator.

The result of this code is a single executable file with no direct dependencies beyond .Net being installed. This executable file does not need to be started manually at any time, as its behaviour is completely controlled by the Compilation Engine component. Unless something unexpected happens, the application should always exit cleanly after attempting to evaluate the submission and generating the report.

4.4.3 Scoring Types

There are types of scoring available, which define how errors in the output files are treated.

The first scoring type, Absolute, will consider any difference (that is not whitespace) between the correct test output and the submissions output for that test as a failure of that test. The only two possible scores for a submission during an Absolute scored test are 0 (failure) and 100 (success).

The second scoring type, Relative, takes an accuracy-based approach. It also compares outputs line by line, but simply counts the number of differences between them and then calculates final accuracy as the ratio between correct number of lines (those that are identical between both the correct output and the submission output) and total number of lines. The final score for a Relative scored test can be anywhere between 0 (0% accuracy) and 100 (100% accuracy).

4.4.4 Test Files

Test files are stored using a particular pattern: "in<number>.txt" in the tasks "Tests" sub-folder, with their appropriate output file stored as "out<number>.txt" in the task's "Correct" sub-folder. The numbering system starts at 1 and goes up by 1 for each new test. This is to allow a predictable sequence of tests to be followed by the evaluation system, for clearer feedback and for simpler test management.

There are no limitations to what test files can contain, as they are in no way altered by any component of the project. It is up to the task creator to define the structure of an input file.

Output files, however, do have some limitations. In the case of Relative scoring, accuracy is calculated on a line-per-line basis. Task creators should take this in consideration when designing the outputs of their tasks. Absolute scoring should not care much for the output file's structure, as a single difference between the correct output and the submission's output will result in failure for that run.

4.4.5 Executable Evaluator

The evaluator used for testing executable files directly invokes said executable through the use of a Process object type (part of the System.Diagnostics .Net library).

The standard Process operations are also used in the appropriate functions. These are:

- Process.HasExited() to check whether a process has ended.
- Process.Refresh() to update memory usage metrics.
- Process.PeakWorkingSet64 to retrieve the peak memory usage the process has had.
- Process.Kill() to end a process.

The working directory is set as the appropriate sub-folder within the "ExeOutputs" root-folder.

This, luckily enough, works flawlessly under Linux environments due to having compiled native code.

No further optimisations are possible.

4.4.6 Python Evaluator

The evaluator used for testing Python scripts must first pass the script to the Python interpreter. Python must be installed on the system, with its binaries saved in the Windows PATH environment variable.

This evaluator then invokes "python.exe" on Windows or "python" on Linux, passing the appropriate filename as the argument.

The rest of the functions act in an identical manner to the Executable Evaluator due to the same Process object type being employed.

There is an overhead created by invoking an interpreter, as it has to allocate its own memory requirements before those of the submission. While runtime is not as affected by this, memory usage definitely is.

4.4.7 Java Evaluator

The Java Evaluator is the evaluator used when testing Java Byte-code class objects. The Java JDK (Java Development Kit) must be installed on the system, with its binaries saved in the Windows PATH environment variable.

Similarly to the Python evaluator, a different application is invoked, this time “java.exe” on Windows or “java” on Linux, with a class name as the argument. This class name is the same as the filename, but without the extension.

The rest of the functions are identical to the previous two evaluators’ functions.

In this scenario, the overhead created by the Java runtime is far greater. Memory usage spikes some orders of magnitude higher than even Python. While runtime is not affected much, memory usage constraints should be set appropriately if the task creator expects many Java submissions.

4.4.8 Raw Report

A raw report is generated in the “.csv” format before being passed on to the Report Generator. This file stores all the results of the evaluation in as few characters as possible. It can be opened in Excel or other data management tools and used in a variety of manners.

The following bits of information are stored for each run: the test run being described, its score, its memory usage in kilobytes and its runtime in milliseconds.

The file is stored as “submission<ID>.<submissionExtension>.raw.txt” in the appropriate sub-folder of the “ExeOutputs” root-folder.

4.5 Report Generator

4.5.1 Programming Languages and Tools Used

For consistency, the Report Generator is programmed in C# .Net and makes use of the same design paradigms as the previously described components. This component does not take advantage of any particular language features beyond the speed of C# file access. The Report Generator is developed in Visual Studio 2013.

4.5.2 Program Design

Following the pattern of maintaining simplicity, the deliverable is a single executable file with not dependencies beyond .Net being installed on the server.

The application takes three parameters: the path to the raw submission report generated by the Evaluation Engine, the ID of the task the submission was set against, as well as the full submission name.

The application reads through the ".csv" file and transforms it into a ".json" web object. JSON is an open standard format that attempts to present data in a readable manner and is often used in conjunction with Javascript. [5]

This extra layer on top of the raw report is useful for future-proofing the project. While the current output might seem simple enough to implement in the Evaluation Engine outright, possible future applications include the generation of ".pdf" files, of tabular ".xls" files and so on. This has been, however, beyond the scope of the project at this time.

This report is then imported by the Website Interface and its data stored in the appropriate model types (OCAPESubmissionReportRun and OCAPESubmissionReport).

Once the Report Generator finished its processing, the whole chain is ended and the Submission Queue can continue its monitoring of the "Uploads" root-folder. The Website Interface is not invoked in any way. Once a user attempts to access a submission's result, the underlying server looks for reports at that time and then processes the results. This saves on some resources that might be used by the server should the server be used on a different machine than the Submission Queue chain.

4.6 Security

One of the most important elements of this project for public use is maintaining a secure environment for the event organiser. Beyond the general stability concerns and protecting from loss of data, should an organiser not trust all of the potential code submitters, security measures must be placed to prevent server issues.

This sub-chapter describes the various security concerns and their solutions.

4.6.1 User Account Security

The Website Interface uses user-based security and, as a result, offers user registration and user login. The choices presented during implementation were a self-coded version of an authentication controller, or using the Microsoft-supplied version as part of the ASP.Net MVC 5 framework.

A custom, self-coded, authentication system would have, in the best case scenario, barely equalled the Microsoft offering, while taking a lot more time to develop and make use of. This way, security is handled by Microsoft's code rather than me, and advanced security measures, such as SSL, are supported out of the box.

The Account controller and its associated models have remained unchanged. This way, future updates to the ASP.Net MVC framework would be supported without losing functionality.

In practice the system works as desired.

4.6.2 Compilation-time Security

One of the main plans during development was to have some level of source code analysis and alteration, to prevent a malicious user from using a number of libraries to gain access to the underlying system. This has, however, proven to not be a good idea long-term for a variety of reasons:

- A malicious user is a lot more likely to find ways to obfuscate their attempted access to insecure libraries.
- Every single supported programming language would require its own set of accepted libraries.
- Some tasks might actually require access to a number of libraries that would be blocked.
- Compile-time would be lengthened due to the extra processing required.
- Incorrect processing might result in code that wouldn't compile.

As a result, an alternative was developed: manual removal of any libraries that might need to be blocked. Event organisers could manually remove the offending libraries from the programming language's development kit, thus rendering any submissions using them as incorrect before they have a chance to even compile, let alone run.

For C++, this would be as simple as deleting (or renaming) the appropriate header (".h") files from either the gcc (based on mingw) install supplied with the project, or from the Visual Studio VC install if using the Microsoft C++ compiler.

For Java, odds are nothing is required, as the language does not have any potential for system access due to its inexistent library of Windows-only functionality.

C# code does not require much library removal, as they require linking on top of defining their usage in the source code. No linking is done during compilation, thus submissions using non-standard and safe libraries would fail to compile.

Python libraries can be easily removed from the "Lib" folder of the Python install. Interpretation would fail should a non-existent library be used.

4.6.3 Security during Evaluation

Sandboxing the submission binary itself is not a viable solution due to the inherent complexities of catching and testing all the operations it could be doing. An application could attempt any number of malicious commands, some including editing the live memory contents of the system to gain control of it; accessing, editing and / or deleting files on the file system; or simply shutting down the system and causing inconvenience to the organiser.

The best alternative found to the lack of a custom sandbox is the creation of a virtual machine to host all the components except for the Website Interface. A small script could check for the virtual machine's availability, restarting it should it have crashed or been compromised in any way. Any malicious acts are kept in check within the virtual environment, resulting in only minor (if any) inconvenience to the host. While the best choice would be a Windows virtual machine, it is very unlikely that a Linux virtual machine wouldn't fulfil the task just as well via the use of the Wine software. Wine is a UNIX program suite that allows the running of Windows native applications in Linux environments. The only limitation would likely be the unavailability of the Microsoft Visual Studio C# compiler.

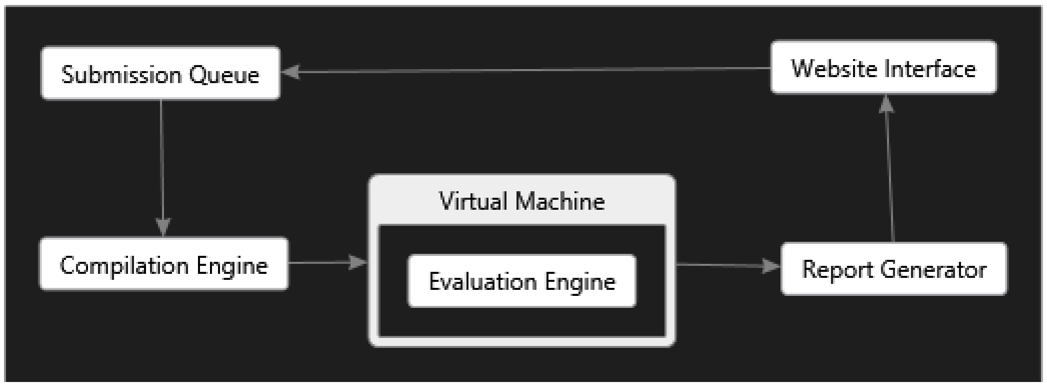


Figure 15 - Virtual Machine Sandboxing of Evaluation Engine

Another solution to the task of containing any submissions would be to run the Evaluation Engine under a third party sandboxing application. The one researched and tried was Sandboxie. [7] It intercepts all outgoing interactions an application attempts and redirects them to an internal file system it has defined. To ensure security, it attempts to just cancel many operations that it does not recognise. [8] The main issue with Sandboxie is its licensing. While it is free for personal use, any commercial applications require a paid license.

Particularly for C# code, an option for limiting the damage an application could do would have been the use of the AppDomain functionality offered by .Net. [9] This would not have been a great solution for the following reasons:

- Solution only available for applications written in C# managed code.
- Malicious users could simply use C# unamanaged code, or, alternatively, another programming language.
- It would have created an extra layer on top of evaluation that would have slowed down all executable files by a small amount.

The solutions previously described should cover most, if not all, the possible methods of attack available to malicious users. They require small amounts of further tool development, particularly in the Submission Queue component, but the benefits are obvious.

5. Evaluation and Testing

This chapter described the various methods employed in the testing of the project, both during and after development of any features.

5.1 Testing Methodology

The chosen method of testing the stability and correctness of the various features present in the project was a thorough manual testing after every non-minor feature update. Due to the heavily modularised design of all the components in the projects, any incorrect behaviour could immediately be pinpointed and fixes could be attempted. Not only that, but having less code for each component allowed for a predictable outcome in each scenario due to the fewer possible paths the program could take at every step.

While unit testing would have been definitely useful if more developers were involved in the process, manual testing proved to be the best in this one-programmer scenario. I was familiar with every function implemented in the project and had each possible outcome mapped well ahead of its implementation on paper.

Many developers fail to realise the benefits of visually describing a block of code's behaviour. By graphically mapping all the outcomes, many bad programming practices can be avoided, while dead ends in iterative processes can be dodged. Good planning can save a lot more time than building and using many tests for bits of code that might never need alterations, thus achieving their final form.

By creating a set of tasks with different ways of being solved, some considered optimal, some not, as well with mathematically calculating the most likely behaviours, testing could be controlled and features could be checked for correct outputs.

The standard tests included:

- Adding n pairs of numbers together, where n can be anywhere between 1 and 1.000.000. This is a straightforward task to solve, but the two chosen solutions had the following difference: the first one was optimal, thus always giving the correct result, while the second one had a 50% chance of giving the wrong answer. This test would highlight the absolute scoring system versus the relative scoring system.

- Calculating the first n prime numbers, where n can be anywhere between 1 and 2.000.000. There are many tweaks to improve the prime number generation system, with some of the solutions used taking advantage of them, while others not. This is a good test to highlight differences in runtime and memory usage between different languages.
- Sorting n given numbers, where n can be anywhere between 1 and 2.000.000. This is a task that has fast solutions (quick sort, merge sort, etc, all of these with an average complexity of O(n * log n)), as well as slow solutions (bubble sort for example, with a complexity of O(n ^ 2)). This test highlights the difference in runtime and memory usage between different algorithms.

For each task, an appropriate submission was written in each of the four supported programming languages, both for comparison and for testing the project compilation and evaluation pipeline. These submission attempt to solve the given task as efficiently as possible unless explicitly described for demonstration purposes. Some of them can be seen in the Appendix, alongside a short description of how they are supposed to function.

Given the ability for the system to accept multiple submissions at once, testing was made a lot easier. Multiple implementations could be processed and their results could be immediately checked both in the raw report files, as well as in the website interface submission details page.

5.3 Testing on Linux

For the purpose of running the components under a virtual machine, all components except for the Website Interface were tested to run under a UNIX system. For this purpose, a Linux Mint installation was prepared, with the appropriate WINE binaries installed. WINE is a piece of software that allows running of Windows native executables under Linux operating systems. Mono libraries were required as well. Mono is the open source implementation of C# .Net.

Once the required software was prepared, simply copy-pasting the executables generated by Visual Studio and running “wine SubmissionQueue.exe” was enough to have a functioning system. All tests described in the previous section were attempted and passed successfully. Alternatively, the code could be run by invoking “mono SubmissionQueue.exe” to the same result.

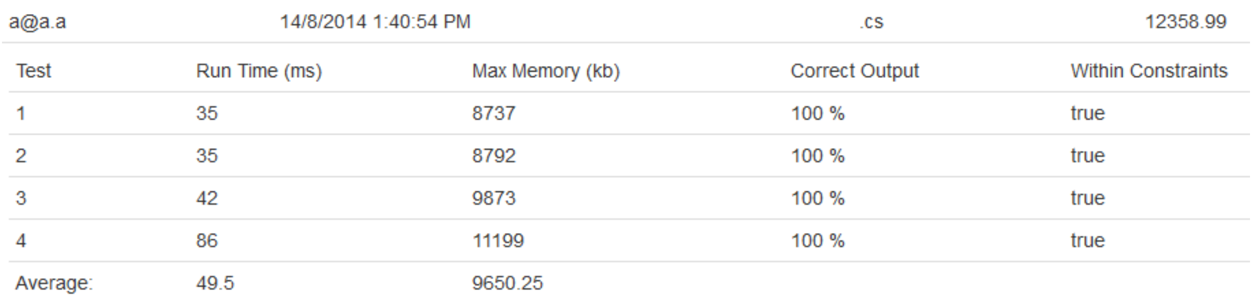
5.2 Unexpected Results

The most interesting and unexpected result of the various tests done during the development of the project was the comparison of speed between C# submissions and C++ submissions to the same task. While Python, due to it being interpreted in this project, and Java, due to it having to start the Java environment before running the code, were expected to be slower in most situations, C# was also predicted to be slightly slower than C++, due to it being a higher level language with dependencies to .NET libraries.

For the task of sorting numbers, using the same quick sort algorithm, the C# submission is faster than the lower level C++ submission. For higher values of n, the difference becomes clearer and clearer.

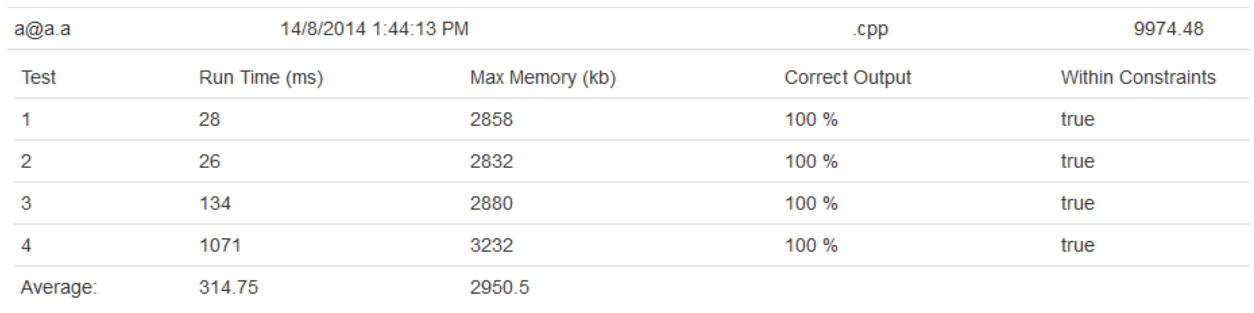
Test 1 has n = 5. Test 2 has n = 100. Test 3 has n = 10000. Test 4 has n = 100000.

The results of C# sorting and C++ sorting are as follows:



a@a.a	14/8/2014 1:40:54 PM		.cs	12358.99
Test	Run Time (ms)	Max Memory (kb)	Correct Output	Within Constraints
1	35	8737	100 %	true
2	35	8792	100 %	true
3	42	9873	100 %	true
4	86	11199	100 %	true
Average:	49.5	9650.25		

Figure 16 - C# Introsort results



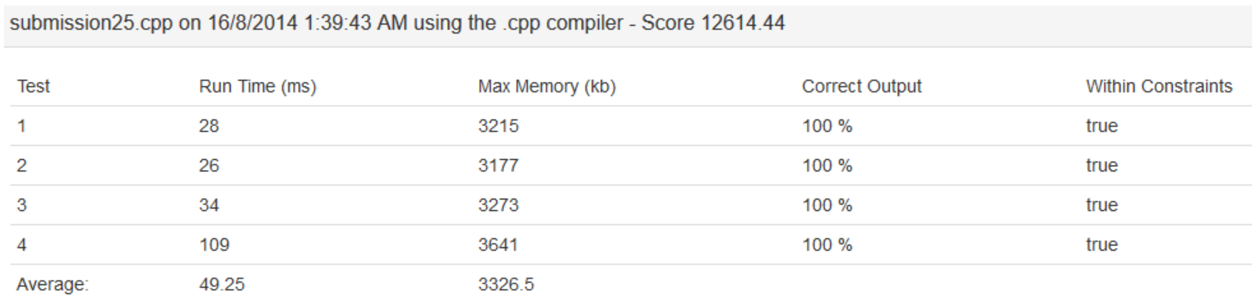
a@a.a	14/8/2014 1:44:13 PM		.cpp	9974.48
Test	Run Time (ms)	Max Memory (kb)	Correct Output	Within Constraints
1	28	2858	100 %	true
2	26	2832	100 %	true
3	134	2880	100 %	true
4	1071	3232	100 %	true
Average:	314.75	2950.5		

Figure 17 - C++ Introsort results – fstream input

While C++ does better with the low count of numbers to sort, it rapidly loses speed when n increases. Memory usage stays respectable, but when speed is of the essence, memory usage is not even close to as important.

A bit of research has shown that both the C++ STD library sort() function, as well as .NET's Array.Sort() function, use Introsort for sorting arrays [10] [11]. To make sure this isn't a standard library issue, a simple quick sort algorithm was implemented and used for both C++ and C# submissions. Results remained unchanged. The relevant code is available for your reading in the Appendix.

After a bit of research, as well as trial and error, the culprit was found to be the fstream C++ library. C++ uses this library to handle file input and output, and it has been developed to be a C++, object-oriented, alternative to the C-style stdio.h library. The std::endl function used to print a line delimiter uses a lot of time to flush the buffer to the output. Replacing fstream-based code with stdio.h-based code (essentially moving from C++ to C), or replacing std::endl with "\n", resulted in C# levels of speed, as exemplified in the results below. To some surprise, C# is still faster during the more intensive test. More information is available in an article I wrote. [12]


submission25.cpp on 16/8/2014 1:39:43 AM using the .cpp compiler - Score 12614.44

Test	Run Time (ms)	Max Memory (kb)	Correct Output	Within Constraints
1	28	3215	100 %	true
2	26	3177	100 %	true
3	34	3273	100 %	true
4	109	3641	100 %	true
Average:	49.25	3326.5		

Figure 18 - C++ Introsort results - stdio.h input

This could be an interesting area for further research, because C# should not be faster than C++. This is, however, beyond the scope of this project.

6. Future Work

There are many ways in which a project of this type could improve. Some include:

- Adding support for new languages, such as Haskell, Pascal, JavaScript, PHP, etc. There is a wide variety of programming languages, each with its own advantages and disadvantages. It is hard nowadays to say that one language is better in every way than another, with respect to modern languages. Freedom to use any code to solve a task is paramount in improving algorithms and discovering better solutions.
- More in-depth error reporting. Currently, errors are reported by a simple "Compilation failed" or "0% accuracy". Offering more information on what a user has done wrong with their submission, such as offering the compilation log, or telling them where exactly they got a wrong result, would help them improve their code.
- More statistics. When doing research, it is hard to say you have enough data to prove your suspicions or findings. The existing offerings are good, but more could definitely be available.
- Notifications. Should the product be used for a competition, notifying any victorious competitors of success via email could be a helpful feature for organisers to have. It could also be used to inform users of new tasks beginning or ending.
- A dedicated setup for installing the project. By preparing all the required systems to work with the project, as well as setting up any virtual machine environments, the project could be deployed to any organiser's machines.

None of these features are critical to the working of the product and I am greatly confident in the fact that, in its current shape, the project could be commercially viable. Performance is good, the user interface is helpful and, in most cases, self-explanatory, and all critical features are implemented.

7. Conclusions

The project proved to be a lot more promising that initially thought. The lack of such a useful tool is baffling, as the potential areas of use are amazing. I now know for a fact I will be continuing to improve this project, as well as use it in my future endeavours.

From a project planning point of view, I am very happy with my choice of methodology. I chose to go with the Agile method of organising tasks, jumping from task to task as believed best. This resulted in early prototypes, fast bug fixes and rolling updates towards the final versions.

Due to the early prototypes, testing could be started a lot earlier in the project timeline. Potential issues, not covered during design, were immediately uncovered and fixed. New features were also a lot easier to design around the existing code-base and implemented. Bad design decisions, such as the sandboxing of executables, were dropped accordingly.

I am also very happy to say that this project has uncovered new potential areas of interest, particularly in the difference in performance between programming languages on the same task. The differences between C# and C++ are something I have found very intriguing and will definitely look into more.

To end it, I am happy with the deliverables, but also with the potential for future development.

References

[1] "International Olympiad in Informatics," [Online]. Available: http://www.ioinformatics.org/index.shtml.

[2] "Bolt.cm," [Online]. Available: www.bolt.cm.

[3] "Bootstrap," Bootstrap Team, [Online]. Available: http://getbootstrap.com/.

[4] "GCC, the GNU Compiler Collection," [Online]. Available: https://gcc.gnu.org/.

[5] "MinGW | Minimalist GNU for Windows," [Online]. Available: http://www.mingw.org/.

[6] "JSON," JSON Group, [Online]. Available: http://json.org/.

[7] "Sandboxie - Sandbox software," [Online]. Available: http://www.sandboxie.com/.

[8] "Sandboxie - Privacy concerns," [Online]. Available: http://www.sandboxie.com/index.php?PrivacyConcerns.

[9] "How to: Run Partially Trusted Code in a Sandbox," Microsoft, [Online]. Available: http://msdn.microsoft.com/en-us/library/bb763046%28v=vs.110%29.aspx.

[10] "Array.Sort Method (Array)," Microsoft, [Online]. Available: http://msdn.microsoft.com/en-us/library/6tf1f0bc%28v=vs.110%29.aspx.

[11] "Introsort - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Introsort.

[12] M. Morosan, "C++ fstream - use "\n" instead of std::endl | Mihail Morosan," [Online]. Available: http://www.morosanmihail.com/home/post/cpp-fstream-use-n-instead-of-stdendl.

[13] "Cutting Edge: ASP.NET Presentation Patterns," [Online]. Available: http://msdn.microsoft.com/en-us/magazine/dd252940.aspx.

Table of Figures

Figure 1 - Microsoft .NET MVC 5 8
Figure 2 - Website Interface Model Interaction 9
Figure 3 - OCAPETask Properties and Methods 10
Figure 4 - OCAPESubmission Properties and Methods 11
Figure 5 - OCAPESubmissionReport Properties and Methods 12
Figure 6 - OCAPESubmissionReportRun Properties and Methods 12
Figure 7 - MVC Request Flow [10] 14
Figure 8 - Example of a Submission with 100% Accuracy 15
Figure 9 - Examples of Available Graphs 16
Figure 10 - Example of the Details Page to a Task 18
Figure 11 - File Storage for Submission Queue 20
Figure 12 - Compilation Engine Object Types 22
Figure 13 - Evaluation Engine Object Types 27
Figure 14 - Base Evaluator Interface 28
Figure 15 - Virtual Machine Sandboxing of Evaluation Engine 35
Figure 16 - C# Introsort results 38
Figure 17 - C++ Introsort results – fstream input 38
Figure 18 - C++ Introsort results - stdio.h input 39

Appendix

Task 1 – Sorting n numbers

C# Optimal Sorting:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Sorting
{
    class Program
    {
        static void Main(string[] args)
        {
            int N;

            StreamReader file = new StreamReader("input.txt");
            StreamWriter file2 = new StreamWriter("output.txt");

            Int32.TryParse(file.ReadLine(), out N);

            int[] intArray = new int[N];

            for (int i = 0; i < N; i++)
            {
                Int32.TryParse(file.ReadLine(), out intArray[i]);
            }

            Array.Sort(intArray);

            foreach (int i in intArray) {
                file2.WriteLine(i);
            }

            file.Close();
            file2.Close();
        }
    }
}
```

C++ Optimal Sorting:

```
#include <algorithm>
#include <stdio.h>
#include <stdlib.h>

using namespace std;
```

```
int main(){
    int n = 1;

    FILE * pFile;
    long lSize;
    char * buffer;
    size_t result;

    pFile = fopen("input.txt", "rb");
    if (pFile == NULL) { fputs("File error", stderr); exit(1); }

    fscanf(pFile, "%d", &n);

    int* v = new int[n];

    for(int i=0;i<n;i++)
    {
        fscanf(pFile, "%d", &v[i]);
    }

    std::sort(v, v+n);

    FILE * oFile = fopen("output.txt", "w+");

    for(int i=0;i<n;i++)
    {
        fprintf(oFile, "%d\n", v[i]);
    }

    return 0;
}
```

C++ Bubble Sorting:

```
#include <fstream>

using namespace std;

int main(){
    int n = 1;
    ifstream f("input.txt");
    ofstream f2("output.txt");

    f>>n;

    int* v = new int[n];

    for(int i=0;i<n;i++)
    {
        f>>v[i];
    }

    f.close();

    int c, d, swap;
    for (c = 0 ; c < ( n - 1 ); c++)
    {
        for (d = 0 ; d < n - c - 1; d++)
        {
            if (v[d] > v[d+1])
            {
                swap       = v[d];
                v[d]    = v[d+1];
                v[d+1] = swap;
            }
        }
    }

    for(int i=0;i<n;i++)
    {
        f2<<v[i]<<"\n";
    }

    f2.close();

    return 0;
}
```

Java Optimal Sorting:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class JAVA_Sort_Default {

    public static void main(String[] args) throws Exception {

        String inputFile = "input.txt";
        String outputFile = "output.txt";

        FileReader fileReader = new FileReader(inputFile);
        BufferedReader bufferedReader = new BufferedReader(fileReader);
        String inputLine;
        Integer n = Integer.parseInt(bufferedReader.readLine());
        List<Integer> lineList = new ArrayList<Integer>();
        while ((inputLine = bufferedReader.readLine()) != null) {
            lineList.add(Integer.parseInt(inputLine));
        }
        fileReader.close();

        Collections.sort(lineList);

        FileWriter fileWriter = new FileWriter(outputFile);
        PrintWriter out = new PrintWriter(fileWriter);
        for (Integer outputLine : lineList) {
            out.println(outputLine);
        }
        out.flush();
        out.close();
        fileWriter.close();

    }
}
```

Python Optimal Sorting:

```
f = open('input.txt', 'r')
f2 = open('output.txt', 'w')

n = int(f.readline())
array = []

for line in f:
    array.append( int(line) )

array.sort()

for s in array:
    f2.write(str(s) + '\n')

f.close();
f2.close();
```

Task 2 – Find the first n prime numbers

Python Solution:

```
def prime(i, primes):
    for prime in primes:
        if not (i == prime or i % prime):
            return False
    primes.append(i)
    return i

def find_primes(n):
    primes = list()
    i, p = 2, 0
    while True:
        if prime(i, primes):
            p += 1
        if i == n:
            return primes
        i += 1

f = open('input.txt', 'r')
f2 = open('output.txt', 'w')
```

```
n = int(f.readline())

for s in find_primes(n):
    f2.write(str(s) + '\n')
```

C++ Very Fast Solution:

```
//From http://www.troubleshooters.com/codecorn/primenumbers/primenumbers.htm

#include <fstream>
#include <string.h>
#include <stdlib.h>
#include <assert.h>

using namespace std;

typedef  unsigned long long bignum;

void findPrimes(bignum topCandidate)
    {
    ofstream f2("output.txt");
    char * array = (char*)malloc(sizeof(unsigned char) * (topCandidate + 1));
    assert(array != NULL);

    /* SET ALL BUT 0 AND 1 TO PRIME STATUS */
    int ss;
    for(ss = 0; ss <= topCandidate+1; ss++)
        *(array + ss) = 1;
    array[0] = 0;
    array[1] = 0;

    /* MARK ALL THE NON-PRIMES */
    bignum thisFactor = 2;
    bignum lastSquare = 0;
    bignum thisSquare = 0;
    while(thisFactor * thisFactor <= topCandidate)
        {
        /* MARK THE MULTIPLES OF THIS FACTOR */
        bignum mark = thisFactor + thisFactor;
        while(mark <= topCandidate)
            {
            *(array + mark) = 0;
            mark += thisFactor;
            }

        /* PRINT THE PROVEN PRIMES SO FAR */
        thisSquare = thisFactor * thisFactor;
        for(;lastSquare < thisSquare; lastSquare++)
            {
            if(*(array + lastSquare)) f2<<lastSquare<<endl;
            }

        /* SET thisFactor TO NEXT PRIME */
        thisFactor++;
        while(*(array+thisFactor) == 0) thisFactor++;
        assert(thisFactor <= topCandidate);
        }

    /* PRINT THE REMAINING PRIMES */
    for(;lastSquare <= topCandidate; lastSquare++)
        {
        if(*(array + lastSquare)) f2<<lastSquare<<endl;
        }
    free(array);
    f2.close();
    }

int main(int argc, char *argv[])
{
    bignum topCandidate = 1000;
    ifstream f("input.txt");
    f>>topCandidate;

    f.close();
    findPrimes(topCandidate);
    return 0;
}
```

C++ Standard Solution:

```
#include <math.h>
#include <fstream>

using namespace std;

bool isPrime (int num)
{
    if (num <=1)
        return false;
    else if (num == 2)
        return true;
    else if (num % 2 == 0)
        return false;
    else
    {
        bool prime = true;
        int divisor = 3;
        double num_d = static_cast<double>(num);
        int upperLimit = static_cast<int>(sqrt(num_d) +1);

        while (divisor <= upperLimit)
        {
            if (num % divisor == 0)
                prime = false;
            divisor +=2;
        }
        return prime;
    }
}
```

```
int main()
{
    ifstream in("input.txt");
    int n = 0;
    in>>n;
    ofstream f("output.txt");
    for(int i = 2; i <= n; i++)
    {
        if( isPrime(i) )
        {
            f<<i<<endl;
        }
    }
    f.close();
    in.close();
    return 0;
}
```

C# Standard Solution:

```
using System;
using System.IO;

namespace PrimeNumber
{
    class PrimeNumber
    {
        static void findPrimes(int topCandidate)
        {
            StreamWriter sw = new StreamWriter("output.txt");
            int candidate = 2;
            while(candidate <= topCandidate)
            {
                int trialDivisor = 2;
                int prime = 1;
                while(trialDivisor * trialDivisor <= candidate)
                {
                    if(candidate % trialDivisor == 0)
                    {
                        prime = 0;
                        break;
                    }
                    trialDivisor++;
                }
                if(prime == 1)
                    sw.WriteLine(candidate);
                candidate++;
            }
            sw.Close();
        }
        static void Main(string[] args)
        {
            StreamReader sr = new StreamReader("input.txt");
            String line;
```

```
                line = sr.ReadLine();
                findPrimes(Int32.Parse(line));
                sr.Close();
            }
        }
    }
```

Task 3 – Adding n numbers

C++ Optimal Solution:

```
#include <fstream>

using namespace std;

int main()
{
    ifstream f("input.txt");

    ofstream f2("output.txt");

    int a,b,n;

    f>>n;

    for(int i=0;i<n;i++)
    {
        f>>a>>b;

        f2<<a+b<<"\n";
    }

    f.close();
    f2.close();

    return 0;
}
```

C++ 50-50 Solution:

```
#include <fstream>
#include <stdlib.h>
#include <time.h>

using namespace std;

int main()
{
    ifstream f("input.txt");

    ofstream f2("output.txt");
    srand (time(NULL));

    int a,b,n;

    f>>n;

    for(int i=0;i<n;i++)
    {
        f>>a>>b;

        if(rand() % 2 == 0)
            f2<<a+b<<"\n";
        else
            f2<<a+b+1<<"\n";
    }

    f.close();
    f2.close();

    return 0;
}
```