KTH Computer Science
and Communication

Code Evaluation System in Microsoft .NET Framework

Evaluating the performance of different languages on the Microsoft .NET platform

WILLIAM LUNDIN FORSSÉN

Master's Thesis at NADA
Supervisor: Alexander Baltatzis
Examiner: Olle Bälter

TRITA xxx yyyy-nn

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus in arcu dui, ac pretium ipsum. Pellentesque iaculis tincidunt sapien in iaculis. Etiam eget justo eget risus luctus rutrum in ut neque. Nam id dolor vel nisl luctus vestibulum at nec nulla. Pellentesque sagittis, erat vitae imperdiet fringilla, nulla nunc sodales libero, id viverra justo odio id metus. Nam tortor libero, consequat ac imperdiet eu, aliquet in nisl. Aliquam erat volutpat. Vestibulum enim metus, dictum a fringilla in, commodo vitae sem. Vestibulum porttitor nisl vitae libero semper sollicitudin. Etiam rutrum laoreet quam vel tristique. Pellentesque turpis lacus, vestibulum in tempor eu, ullamcorper id nisi. Vivamus a molestie sem. Sed commodo mattis neque, ac convallis mauris tristique non. Donec dui purus, aliquet id pellentesque vel, porttitor at nunc. Praesent elit justo, porta tincidunt venenatis sit amet, sollicitudin quis sem.

Donec eu nunc orci, vel pretium lectus. Sed augue mauris, porta id convallis quis, consequat sit amet lacus. Nulla venenatis tortor at risus semper dictum. Nulla ut mi eu orci dapibus vulputate. Morbi tincidunt pharetra ultrices. In fringilla velit ut enim auctor luctus. Etiam venenatis blandit condimentum. Nam fringilla aliquam leo, a aliquam arcu malesuada vitae. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Pellentesque tempor nulla non dolor sagittis venenatis. Integer felis nibh, porttitor in condimentum ac, auctor quis dui.

Fusce eget tincidunt nibh. Quisque sed velit felis. Suspendisse in sagittis lorem. Fusce posuere ipsum id leo facilisis sit amet sollicitudin enim egestas. In augue massa, tincidunt ac blandit faucibus, ultricies nec tortor. Fusce et dui diam. Cras nec eros id dui laoreet lacinia. Proin malesuada est vel erat pharetra rutrum faucibus nulla porta.

Referat

System för kodevaluering i Microsoft .NET Framework

Donec eu nunc orci, vel pretium lectus. Sed augue mauris, porta id convallis quis, consequat sit amet lacus. Nulla venenatis tortor at risus semper dictum. Nulla ut mi eu orci dapibus vulputate. Morbi tincidunt pharetra ultrices. In fringilla velit ut enim auctor luctus. Etiam venenatis blandit condimentum. Nam fringilla aliquam leo, a aliquam arcu malesuada vitae. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Pellentesque tempor nulla non dolor sagittis venenatis. Integer felis nibh, porttitor in condimentum ac, auctor quis dui.

Foreword

This report is my Master Thesis project which was carried out at Valtech in Stockholm, Sweden. This thesis is part of my last course in the Masters Programme in Computer Science at The Royal Institute of Technology (KTH). The most difficult part of this thesis was finding out what subject to research. I wanted to evaluate something in Microsoft .NET Framework, but I didn't know what. Then one day some words from my supervisor Alexander Baltatzis echoed in my head from one of his lectures "Write code in any language and run it on .NET". That sentence sparked my interest in finding out whether it was actually possible or not, since most people who write programs on the .NET platform do so using C# or Visual Basic, you barely ever hear about someone using any other language (even though many others are supported by the .NET platform).

Glossary

Term	Description
CES	Code Evaluation System, also known as Automatic Grading System, a system for evaluating or grading code.
CIL	Common Intermediate Language, the lowest level human-readable programming language in .NET Framework.
CLI	Common Language Infrastructure, a specification that describes the runtime environment of Microsoft .NET Framework.
CLR	Common Language Runtime, is the virtual machine of Microsoft .NET Framework.
CLS	Common Language Specification, a set of rules and features that a .NET language must implement and understand.
GUI	Graphical user interface, an interface that is image oriented rather than text oriented.
IKVM	An implementation of the Java for Mono and Microsoft .NET Framework.
IL	Intermediate Language, a language of an abstract machine.
IronPython	An implementation of the Python programming language targeting the Microsoft .NET Framework and Mono.
Mono	An open source project created to enable .NET Framework cross-platform compatability.
MSIL	Microsoft Intermediate Language, a synonym for CIL.

Contents

Chapter 1

Introduction

1.1 Problem Statement

A common challenge for Valtech and other consulting companies is to recruit the right people. The recruitment process in the IT business often involves several steps and can be very time consuming. This is due to the fact that knowledge and skill differ greatly from programmer to programmer and therefore a need to interview many people arises (just to find one that is deemed suitable for the job).

The recruitment process usually involves some kind of test to assert the skill of the programmer. The test is sometimes performed on paper (questionnaire) or on a white board. Both of these environments arent FIXME really suitable to program in, hence most programmers tend to do their work on computers with the assistance of a web browser and a suitable IDE.

These two issues present a challenge in which the optimal solution would result in less time spent recruiting and at the same time asserting the skills of the programmers before an interview is considered.

1.2 Goal

The main goal was to evaluate a system which could satisfy the demands stated above. Thus, since no such system was present at Valtech, there was a need to first build the system and then evaluate it through test cases with performance and memory consumption in mind.

The system would allow users to submit solutions to common programming problems and then have their code evaluated. Their code would have to have a limit on execution time since a "good" programmer would be able to solve problems efficiently. The results of this evaluation would be used to estimate how much time the different programming languages would need in order to solve specific problems, as all of these languages would be run in the Microsoft .NET Platform.

1.3 About Valtech

Valtech is an independent, global IT-consulting company with offices in Europe, United States and Asia. The company was founded 1993 in France and has over 1600 employees worldwide. The Swedish branch is primarily oriented towards web solutions. Valtech's most well-known award winning projects include the websites 1177.se, Antagning.se and Riksbank.se.

All work concerning this thesis was carried out at Valtech in Stockholm, Sweden.

Chapter 2

Background

2.1 What is automatic code evaluation?

Automatic code evaluation, or automatic grading, is a computer system that has the ability to judge code. This is usually done in the following steps:

1. A user is given a programming task or problem.
2. The user attempts to solve this problem by writing code.
3. The user sends this code to the automatic code evaluation system.
4. The system compiles the code (if needed).
5. The resulting program is then run by the system with some test data as input.
6. The system verifies that the program output is correct (or incorrect).
7. An answer is then returned to the user indicating the status of the code which he or she submitted (i.e. "Accepted" or "Wrong answer").

There are some variation to the process above, sometimes more verbose feedback than "Wrong answer" is used, often indicating exactly which test went wrong and why (referens ISECON.2006). This is usually done in order for university students to learn more about debugging their code.

2.2 History

Evaluating code automatically is nothing new. The earliest known system was built in 1960 by Hollingsworth (referens Hollingsworth). This system was used in an introductory course in Algol programming. The results from using this student-system approach rather than the traditional student-teacher was that it cut costs considerably for the staff since the time they needed to grade the students work was reduced by as much as one third. The students themselves also spent less time,

since they were able to have their work graded immidiately instead of waiting for a teacher to do it. This system also made it possible to considerably increase the number of students taking the course. It did, however, have some shortcomings. For instance, a student's program could modify the grader program, making cheating possible.

An article from 2005 (referens p1-douce) describes three generations of automatic graders. The first generation systems were those regarded as being built and/or used in the 1960's and 1970's. Unsurprisingly, they used code that were close to pure machine code and some even used punched cards. In order to make them work, it was necessary to modify both compiler and operating system.

The second generation systems (1980-2000) introduced script-based tools. These involved various verification schemes and also asserted that the code was written in a certain way/style (decided by the teacher). Typically these graders involved command-line GUIs. Languages like C and Java were used extensively.

The third generation (2000-) differ from the second generation systems primarily in two ways. One is that they mostly use web based GUIs. The other is that they often included a plagiarism detection system, since students sometimes shared code amongst each other. There were some minor issues among these detection systems (referens ISECON2006)(referens p1-douce). If the programming task was too simple or if a lecturer had been excessively thorough when describing the homework, the submissions would tend to be very much alike and thus picked up by the plagiarism detection system. This made it somewhat difficult to distinguish between real plagiarism and the false positives.

2.3 Todays systems

Todays modern systems (ref Kattis)(ref p219-amelung)(ref p1-douce)(ISECON.2006) (considered to be third generation systems) commonly contain the following features:

- A web based front-end.
- A language specific back-end.
- A seperate back-end for each programming language the system supports.
- An SQL database used to store submissions.
- Sandboxing of submitted code to maintain a solid system state.

These systems mainly differ in their support for different programming languages.

Chapter 3

Method

This chapter explains how the system was implemented. It contains details about the work-flow, the GUI, how multiple languages are supported, how the security works and some of the difficulties and limitations encountered.

3.1 System Description

The system was named CELINE, an abbreviation that comes from the descriptive phrase "**C**ode **E**va**L**uation **I**n .**NE**t" (the uppercase bolded letters forming the abbreviation).

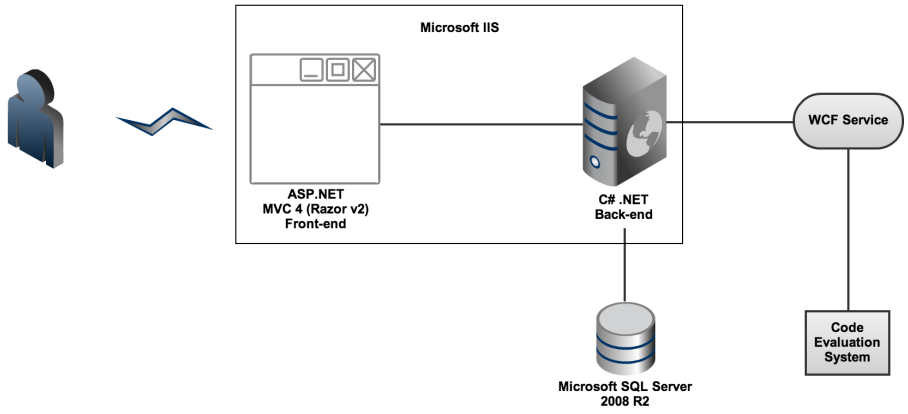


Figure 3.1. Overview of CELINE.

Figure 3.1 describes an overview of the system. The system uses a web based GUI for listing problems and submitting solutions to them. Going through this figure from left to right we can see that a user connects to the website through a web based GUI, which is built using a combination of ASP.NET MVC4 (Razor v2) technology and JavaScript. The system uses Microsoft IIS to enable this web support, which the standard web server software used in .NET. When the user

submits code to solve a problem, that submission is saved to a database which is a Microsoft SQL Server 2008 R2 database. The submission is then forwarded to a WCF Service which in turn creates a new instance of the code evaluation system (built using C#) using the submission as input. It then returns the status code generated from the submission, this is described in section 3.1.3.

3.1.1 The Web GUI

The GUI is built using the ASP.NET MVC4 template. This saved both time and effort while still giving the website an easy to understand simplistic style.

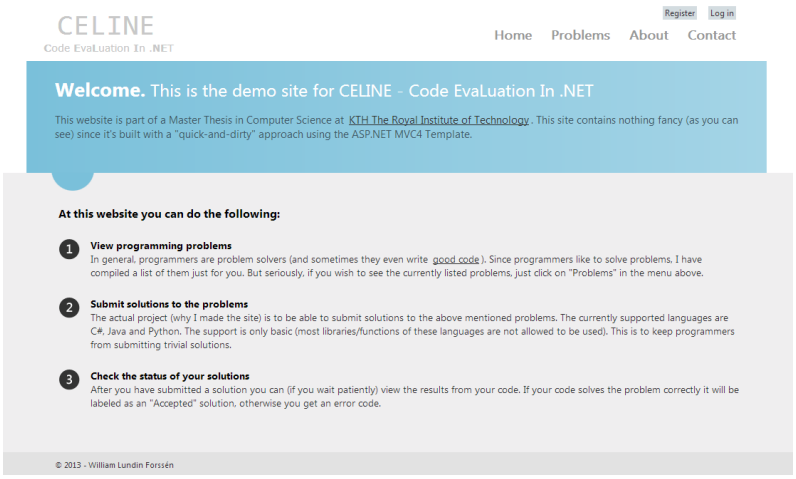
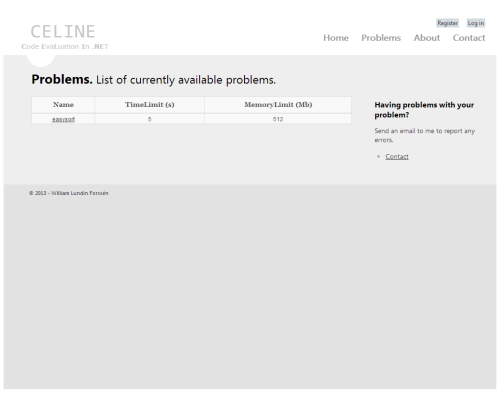
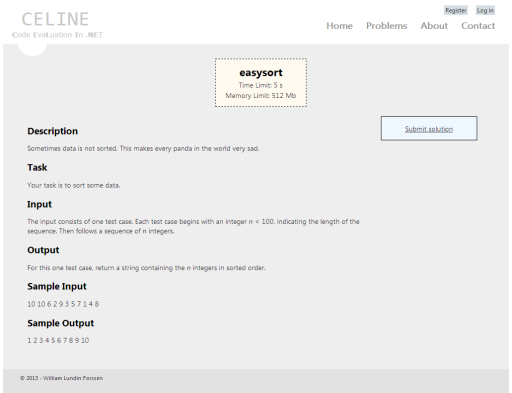


Figure 3.2. The start page of CELINE.

In Figure 3.2 we see the start page. The middle area contains some informative text, the top right contains the site menu, the register and login buttons.



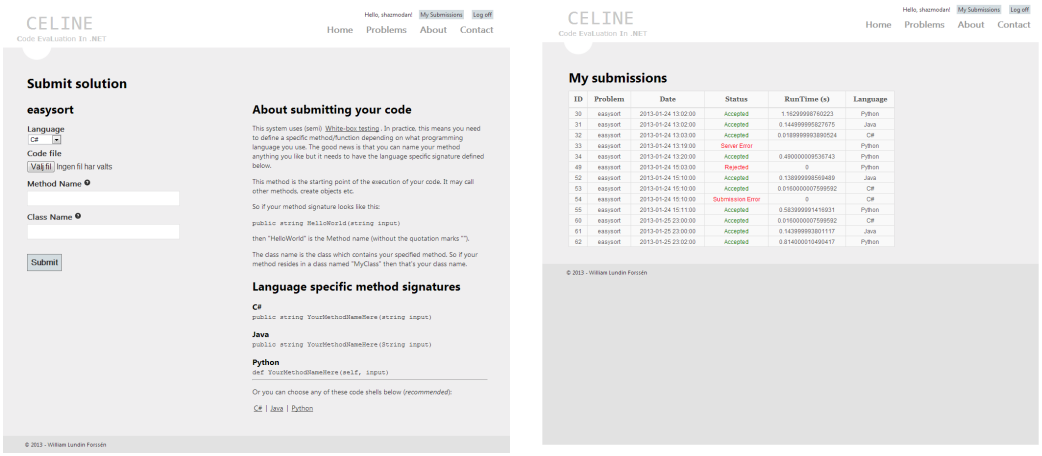
(a) List of available problems.



(b) One problem called “easysort”.

Figure 3.3. Navigation in the Problems menu.

Figure 3.3(a) shows what happens if you click on “Problems” in the menu. A list of currently available problems is displayed. The list contains details such as maximum run time and maximum memory usage. Clicking on one of the problems in this list will generate a page with more details of that particular problem, this is displayed in Figure 3.3(b). To the right of the details-text is a button for submitting a solution. Clicking it will generate a page containg a form for submitting code, illustrated in Figure 3.4(a). Apart from the form itself on the left side, this page contains some useful information on the right side.



(a) Code submission page.

(b) List of all submission made by the current user.

Figure 3.4. Navigation when submitting code.

Figure 3.4(b) shows the page listing all of the current users submissions. The user is sent here after successfully submitting a solution (by filling out all the fields of the Submission page illustrated in Figure 3.4(a)) or by clicking on the button in the top right corner. This page contains information about each of the users submissions. Each line details what problem the user attempted to solve, a timestamp, total runtime, the programming language used and the returned status code.

3.1.2 The code evaluation system

When a user has submitted his or her code, it eventually reaches the CES. Figure 3.5 describes the flow of the most common paths for a submission through the system.

Depending on the programming language, a compiler is chosen. The code is then compiled into an assembly file and loaded into a seperate secure (sandboxed) Application Domain (AppDomain). The CES then invokes the user specified method with a string containing the test data. The CES waits for the code to complete or for the problem to timeout. It then compares the string returned by this method with another string containing what should be the correct output. If the strings

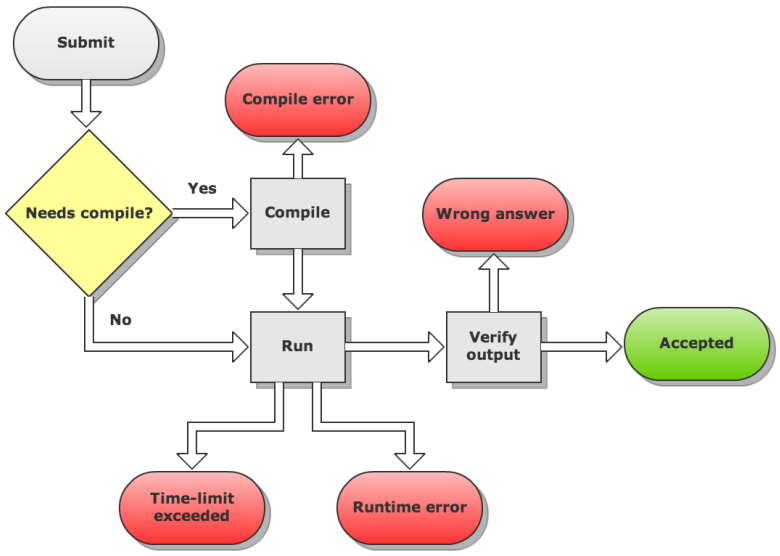


Figure 3.5. Common paths of problem flow in the CES.

match, the submission is regarded as being a success, otherwise it's regarded as a failure. The CES then returns the appropriate status code.

3.1.3 The status codes

The status code is a simple message indicating if the submission was successful or not. This system attempts to be more verbose on errors than other modern systems (described in section 2.3) since it uses white-box testing which is more prone to submission related errors than black-box testing. These concepts are described more thoroughly in section ???. FIXME

- Accepted - The code has compiled, run and gave the correct answer.
- Wrong Answer - The code has compiled and run but gave the wrong answer.
- Server Assembly Error - The CES failed to build an assembly from the codefile, thus making the code unable to run.
- Submission Error - Occurs if the code tries to reference a code library that isn't available/allowed.
- Illegal Operation - Occurs if the CES detects a forbidden system call (i.e. accessing files, using the network etc...).

- Class or Function Error - Occurs if the class or method name doesn't correspond to the name provided by the user, thus resulting in the CES being unable to invoke the method.
- Time Limit Exceeded - The code ran longer than the limit for this particular problem.
- Rejected - This is a general error, it indicates that the CES has been unable to determine why the submission failed.
- Server Error - This indicates that the CES instance has crashed.

3.2 Supported languages

3.2.1 C#

The support for C# was implemented with the use of the Microsoft CSharp library (ref http://msdn.microsoft.com/en-us/library/microsoft.csharp.csharpcodeprovider.aspx) and the System CodeDom Compiler library (ref http://msdn.microsoft.com/en-us/library/system.codedom.compiler.icodecompiler.aspx).

3.2.2 Java

The Java support comes from using javac (ref http://www.oracle.com/technetwork/java/javase/tech/java
137034.html), the standard Java compiler that comes with the Java Development Kit (JDK) (ref http://www.oracle.com/technetwork/java/javase/overview/index.html). Javac compiles Java code to bytecode. This bytecode is then converted to CIL using ikvmc (ref http://sourceforge.net/apps/mediawiki/ikvm/index.php?title=Ikvmc), a tool that is part of IKVM.NET (an implementation of Java for Mono and the Microsoft .NET Framework) (ref http://www.ikvm.net/).

3.2.3 Python

Python is supported through the use of IronPython (ref http://ironpython.net/). IronPython provides a library which contains a ScriptEngine from which python code can run without the need for compiling. Some issues encountered with the IronPython implementation are described in section 3.3.2.

3.3 Difficulties and limitations

3.3.1 Security

Executing unknown code can be dangerous for several reasons, the code might alter the core program or even do damage to the computer. Thus it was necessary to avert this danger by running the code in a safe and secure sandboxed

environment. In .NET this is handled through the use of AppDomains which act as a layer of isolation between applications (ref http://msdn.microsoft.com/en-us/library/cxk374d9.aspx). AppDomains make it possible to protect the core application from malicious code and exceptions that might occur during execution of the unknown code. This protection comes at the cost of performance since AppDomains are able to load assemblies but cannot unload them. To get rid of a loaded assembly the entire AppDomain must be unloaded, which is the case with each new submission. CELINE creates a new AppDomain for each submission and then uses proxy objects called "Marshals" (ref http://msdn.microsoft.com/en-us/library/aa720494(v=vs.71).aspx) to enable communication between the core-AppDomain and the untrusted-AppDomain, in order to get the output that the unknown code produced.

There is an exception to the rule about AppDomains being safe from each other. That is when the code that is executing in the untrusted-AppDomain generates a stack overflow exception. This exception will travel back into its parents AppDomain (in this case the core-AppDomain) since it's no longer possible to catch this exception as from .NET 2.0 and above. In order to avoid crashes caused by stack overflow exceptions the CES is instantiated with each new submission by the WCF Service.

3.3.2 IronPython limitations

IronPython does not have the same performance as the other languages for a number of reasons. The first reason is that it's 61.6% slower on average than normal Python (version 2.7) (ref http://ironpython.codeplex.com/wikipage?title=IP27A1VsCPy27Perf). The startup time is also significantly slower since this language is scripted using a ScriptEngine which adds another layer of interpretation. It is however important to mention that IronPython does have the capability to compile the code into an assembly, but with a serious drawback, the assemblys methods and main entry point cannot be invoked like other assemblies since the MSIL is not CLS-compliant (ref http://msdn.microsoft.com/en-us/library/system.clscompliantattribute.aspx) and therefore cannot be directly accessed from other .NET languages (ref

- IronPython limitation - Python ScriptEngine thing

3.3.3 White- and black-box testing

- White-box testing and black-box testing

Chapter 4

Testing Methodology

4.1 Aspects to tests

4.1.1 Execution speed

4.1.2 Memory usage

4.2 The chosen tests

4.2.1 Language overhead

4.2.2 Common Operators

4.2.3 Insertion Sort

4.2.4 Merge sort

Chapter 5

Testing Results

5.1 Language overhead

5.2 Common operators

5.3 Insertion sort

5.4 Merge sort

Chapter 6

Discussion

Chapter 7

References

Appendix A

RDF

And here is a figure

Figure A.1. Several statements describing the same resource.

that we refer to here: A.1