STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume LXIV, Number 2, 2019
DOI: 10.24193/subbi.2019.2.07

AUTOMATION AND GAMIFICATION OF COMPUTER SCIENCE STUDY

IMRE ZSIGMOND

Abstract. A large part of the job of a university teacher in computer science is to verify student exercises. The task of verifying correctness, coding style, conventions, and grading is laborious and if done correctly leaves little time for questions. Automating aspects of this work helps all parties involved by freeing time up for questions. The solution detailed in the paper automates a number of these tasks and provides near instant feedback for the students, together with a platform to gamify student learning.

1. Introduction

Computer science teachers at a university level face laborious work verifying the correctness, coding style, conventions, and predefined technical details of student exercises [2]. Very little time is left to answer questions, help struggling students, to tackle complex coding situations, and adapt new teaching techniques. In addition, the institution needs experienced and trained personnel to do the job. Extra care must also be taken to check for plagiarism. All the while, the study of computer science lends itself to automation in core areas.

In the last decade there has been increased research into the gamification of learning, as a response to the ever-increasing challenge to motivate students and increase their engagement [14]. The current best accepted definition of gamification is, the use of game design elements in non-game contexts [4]. Literature reviews suggest that these techniques tend to have a positive effect most of the time [9]. Some of these elements in their turn come from

Received by the editors: November 15, 2019.

2010 *Mathematics Subject Classification.* 68Q60.

1998 *CR Categories and Descriptors.* D.2.11 [**Software engineering**]: Software Architectures – *Domain-specific architectures*; D.2.4 [**Software engineering**]: Software/Program Verification – *Model checking.*

Key words and phrases. system architecture, education, automated verification, gamification .

psychology, entertainment [18]. The goal is to maximize engagement through capturing the interest of learners, and inspiring them to continue learning.

If parts of the work arguably should be automated and there is an increasing need for gamification elements, the question remains how much to automate? how many game design elements to add? and how to imbue the two?

In the solution detailed in the paper the approach to these questions was to make an assignment validator platform with configurability and expandability in mind. The various options and the easy expandability allow for fine tuning of features for a given course material. The platform also serves as a gamification workbench to aid experimentation.

This paper is organized as follows: related work on assignment automation and gamification in section 2, main contribution in section 3, conclusion in section 4, discussion in section 5 and bibliography in section 6.

2. Related work

Automated grading of computer science assignments at college level goes back to 1965 [8]. Back then it was used for numerical analysis courses, written in BALGOL a dialect of ALGOL, and handed in on punch cards. A decade later physicists still use porta-punch cards but assignments are generated randomly and evaluated in batch [12]. A decade after that we see automated grading of multiple-choice tests in [17]. By the early 1990's complex assignment graders for programming and Matlab are developed and used [19]. As late 2000's arrive we have student websites graded with the help of Ruby scripts, as well as requirement assignment, although it relies on exact HTML naming on student's side [5]. The base problem is still being researched and different solutions are being developed [15].

JFDI Academy's solution is approaching ours, they were able to provide students with timely feedback. It had auto-grading, in which once a student submitted the answer to a question, (s)he was automatically given feedback on whether it was correct. Instructors were also able to receive feedback on a student's progress. Students also had the opportunity to raise questions or concerns regarding the assignment, by posting comments which enabled instructors to help them in a timely manner [3].

In parallel, although much recently gamification experiments are being conducted on students in various settings. For example, the promotion of risk taking in language learning [10]. Sometimes the two are combined, for example in the case of [6] a computer engineering master's level college course was gamified with automatic grading. This resulted in forum engagement increase of 511% to 845% versus the previous year.

3. GamifyCS

3.1. **Solution description.** The question of, how much teaching assisting tools, as well as gamification elements to add, is a non-trivial one. The answer tends to depend on the needs of various courses and the aims of the professors teaching said courses. With the goal to provide an automating tool and gamification workbench we identified the following priorities to implement first: individualized exercise assignment, correctness check, semi-automated anti-plagiarism test, near instant feedback, story elements. These features allowed for large scale experimentation, and more time helping students, the details of which will be presented in a future paper. The following features were prototyped and remain to be used in the next semester: fully-automated anti-plagiarism test, static code analysis checks, achievements. While the prototypes worked a subset of them were prioritized to experimentally validate them before more were added.

While the tool is both an assignment validator and a gamification workbench, the core validation is detailed in Figure 1, and there is a more technical view in the next section. While the processes shown are all optional, they do represent the typical needs of a teacher. The process in general starts with an upload on the student's part, the archive is saved locally, extracted, cleaned, and repackaged for MOSS. [1]. If the language warrants compilation the sources are compiled and linked to executables. Then separate instances are run for each predefined test. Test scenarios are evaluated, and the results are saved to be displayed later. Meanwhile on the coding style side StyleCop analyzer is invoked, the resulting report being saved to the database. In parallel the repackaged source files are uploaded to MOSS. for anti-plagiarism checks, and the results saved to the database.

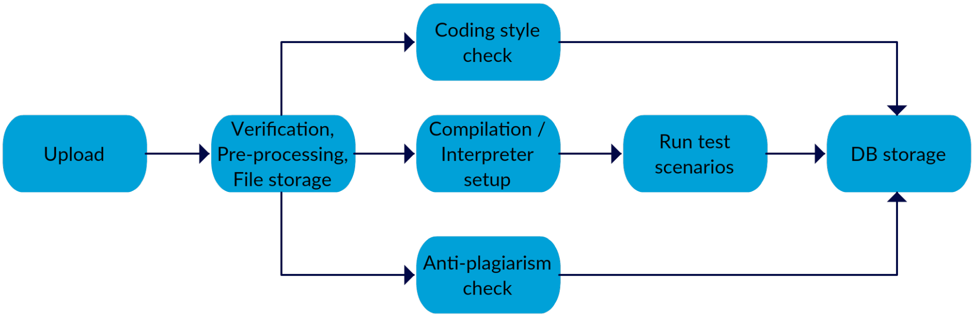

Coding style check
Upload
Verification, Pre-processing, File storage
Compilation / Interpreter setup
Run test scenarios
DB storage
Anti-plagiarism check

Figure 1. Validation flow

For this tool to be useful it had to support multiple programming languages, so it can be applied for many courses and needs. Supporting any number of programming languages is quite a challenge in itself. The solution was to be language agnostic from the code's point of view and just swap in different modules that had the job of dealing with specific programming language features. The module can decide to try to resolve the specifics with code, or calling external tools. Thus, support for any new language has to be coded in separately.

Correctness check for any given problem is its own separate branch of computer science, and entirely out of scope for this paper and solution. To help with solving this problem each assignment has a well-defined input and output sequence. This sequence is run on each hand-in of a specific assignment, thus testing it. This limits the type of assignments that can be tested though. It is an acceptable first step since a significant number of assignments, in an academic setting have console-based UIs. The proposed solution is useful for full courses, and ultimately can be expanded for more specialized correctness checks, while retaining their other benefits.

Static code analysis of programming assignments has been the focus of computer science research, all the while the specifics of code quality is not a universally agreed upon subject [13]. The debate becomes more varied when different languages are concerned, and it also changes as time goes on. Even on a principle level there are various disagreements, and few universal rules are agreed by most people. For example, memory leaks are universally considered bad while variable naming changes wildly even within one language. What rules to require remains the discretion of the teacher of a specific course.

Plagiarism check is one of the basic functionalities that is also one that cannot be completely automated [11]. An example of technical difficulty is the project files that are generated by their IDE, that should not be checked. The tool sometimes leads to false positives which would be detrimental to learning if students were falsely accused. There are many special cases and situations in practice that would lead to false positives. Our solution was to take the reports generated by Stanford's MOSS API and check them manually. It should be mentioned that sometimes teachers can and do discover certain forms of plagiarism. While software based solutions may not work better then experienced staff, together with the increased time per student, because of the lack of need to check for correctness, increases the change for cheaters being caught.

3.2. **Architecture.** For an illustration of the architectural decisions a UML communication diagram is used in Figure 2. The main application is a website developed using ASP.NET MVC in C#. It runs on a windows server with an

SQL Server installed locally. Structurally it can be considered a standard MVC site, at around 14000+ lines of code. Logging was twofold, with various activities either saved to the database or to logfiles. The hub for the more interesting parts is the TestRunnerService. As mentioned in section 3.1 the process of verification starts with an upload on the student's part, the archive of each attempt is saved locally to a unique path, then extracted, cleaned of non-source files, and repackaged for the MOSS API's convenience.

From this point on the verification thread, source code is compiled and linked if compilable, or supplied to the interpreter otherwise. Separate instances of the resulting executable are run for each test to ensure clean runs. Communication with the executable on StdIO redirects had to be run on separate threads. Then test scenarios are evaluated, the instances closed, and results are saved to be displayed later. The running executables are monitored by a windows service called Monitor Service. The reason for a separate service is that in certain scenarios the executables would not terminate from the context of the verification thread, and remain active in the system. Such programs if caught in an infinite loop would quickly consume large amounts of processor resources, and had to be killed externally. The monitor service does just that by comparing process StartTimes to local time, for running processes of a given name.

On the coding style thread, the StyleCop analyzer is invoked asynchronously, the resulting report being saved to the database, and displayed as part of the test results. Meanwhile on the anti-plagiarism thread the repackaged source files are uploaded to the MOSS API, and the results saved to the database.

3.3. **Features.** Compilation of the uploaded project was one of the central issues of automation on the project while supporting several potential languages. External tools were necessary because real life programming languages evolve as do their compilers. The solution decides how to prepare the executables based on the language in question. The switch is done through the strategy design pattern. Supporting C# required only .NET framework calls and it produced the required executable from code. Python being an interpreted language required the execution of the interpreter with the " -i + pathToSource" supplied as parameters. The resulting execution was the target for testing. By far the most complicated was support for C++, where minor differences between compiler parameters between student side compilation and sever side compilation lead to subtle but significant differences in test output. After much experimentation the following parameters yielded the best results:

```
"cl /permissive- /GS- /analyze- /w /Zc:wchar_t /ZI /Gm- /Od /sdl
/Zc:inline /fp:precise /D \"WIN32\" /D \"_DEBUG\" /D \"_CONSOLE\"
/D \"_UNICODE\" /D \"UNICODE\" /D \"_CRT_SECURE_NO_WARNINGS\"
```

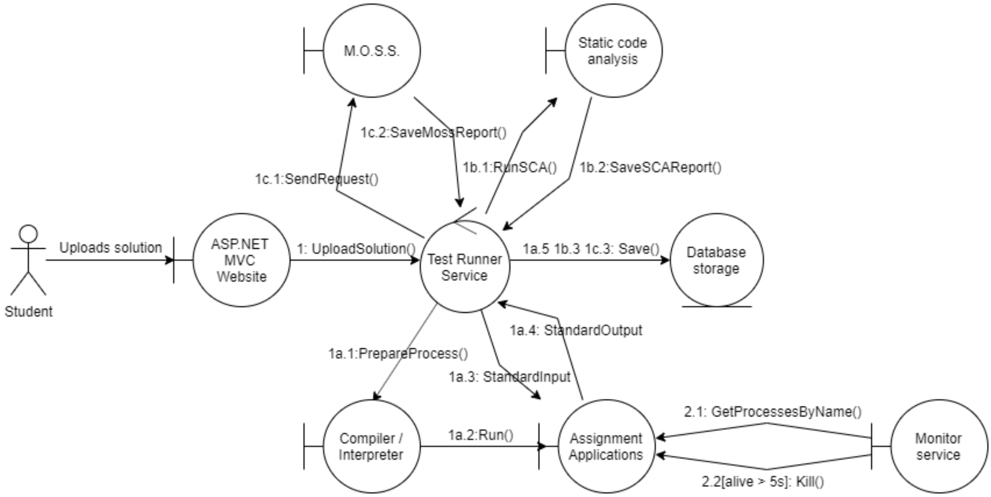


Figure 2. Communication diagram

```
/errorReport:none /WX- /Zc:forScope /RTC1 /Gd /Oy- /MDd /FC /EHsc
/nologo /diagnostics:column [listOfFullPathFilesInProject]
/Fa.\\ /Fe:.\\program.exe /Fo.\\"
```

When executing the compiled code, the main application redirected the standard input and output of the assignment to the main program, which in turn fed, then read the predefined sequences. While redirection is a just a flag at startup, the two being streams separate threads needed to be started to communicate with the new process. To facilitate the situations where the program being tested stopped to respond or hang in an infinite loop, timeouts on the threads were used. In addition to timeouts the new process's state was monitored both from the testing program itself and a separate windows service. The monitor service was to query running processes with certain names and terminate those with more than 5 s runtimes. To give more leeway on correctness checks for the students, regular expressions were used by default on their output that has been compared to the expected output on the I/O test. An unexpected and welcome side effect was that students could upload their solutions multiple times during the practical exams serving as working, testable backups in case of software/hardware failure.

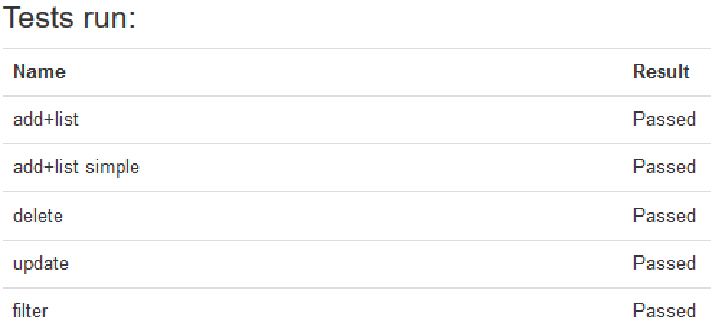
To illustrate with an example: one of the assignments had among other requirements the need to catalogue old maps, the "add" and "display" console commands were specified to take the form:

add mapCatalogueNumber, stateOfDeterioration, type, yearsOfStorage
list

One test aiming to check the ”add” command with a valid input, started with sending the application: ”add 1234, new, political, 15”, then sent: ”list”, finally the regex that ran against the output was: ”.*(1234)*.*new.*political.*15”.

Code quality checks were prototyped with StyleCop an open source static code analysis tool which has been used to success with .NET projects [16]. The StyleCop.Analyzers project was used for the analysis. It may be called with a configurable list of rules and it generates a report of violations. Example of a rule would be: “FieldNamesMustBeginWithLowerCaseLetter” the violation of this rule happens when the name of a field in a class begins with an upper-case letter. The various rules can be configured per course.

User interface was specifically designed to be minimalist and responsive. The goal was to remove distractions and appeal to current generation’s standards. It also focused the work to deliver higher quality rather than more features. The UI consisted mainly of data manipulation screens for the administrators. An example of test results the students saw can be found in Figure 3.



Tests run:

Name	Result
add+list	Passed
add+list simple	Passed
delete	Passed
update	Passed
filter	Passed

Figure 3. Test Run Example

Among the possible gamification mechanics, two were chosen on the basis of what experiments were scheduled. The first two were instant feedback and narrative. Instant feedback is one of the most basic gamification mechanics. It shortens the work-reward cycle and it is easier to do on modern computers. In computationally intensive situations the feeling of instant feedback can be achieved by making time for static code analysis to complete by showing the I/O test results one by one with small, artificial delays. If the instant feedback option was used on a group then they could upload solutions any time of the day, otherwise they could only upload during their individual laboratory times.

Narrative is one of the strong suits of R.P.G.s [7], which is where some of the gamification mechanics originate from. It aids in immersion and gives a sense of purpose to solving exercises. The aspect of tying all exercises together in a narrative was one of the interests while working on this system. With the aim of finding out, if adding cross assignment narrative adds to the enjoyment of solving programming puzzles. If a group was selected for narrative than they saw an alternate text for their assignments that took part of an overarching story. Regular assignments and examinations both took part of the story. There was also support for different languages for the exercises for the same course thought in a foreign language.

For login and authorization by role, standard ASP.NET MVC controls were used, which are deemed safe enough. Any Id used in the various URLs are GUIDs as to make it impossible to guess any other ids. Students had the possibility to re-upload new exercises solutions until they were happy with the results. All code variants were saved on the server both code and test result for data-mining purposes but after the semester the student names and emails were anonymized.

4. Discussion

Taking into account previous research [2] the issue of a secured environment came up. Allowing random code to run without a sandbox would expose the server to vulnerabilities. A sandboxed environment would be ideal but this remains a future development. Another potential issue mentioned in the same research that was noticed is the temporary overwhelming of the hardware when too many students upload at the same time. Their solution is to add waiting queues which would be a sensible choice here as well, but it has to be prototyped and UI has to be changed to manage student expectations.

An unfortunate side effect of having the server accessible to the internet was a persistent, distributed attempt to gain control from at least one botnet. While unsuccessful the several tens of thousands of requests did tie down a portion of the server's resources.

5. Conclusions and further work

The system developed has great potential for experimentation, since it allows for features to be switched off and can be freely extended. Various experiments can be designed around the tool. Since we store code and usage data, data mining methods can be used in lockstep with traditional experimental setups.

The tool was used for a semester for an object oriented programming course. Aside from bug-fixes the tool was unchanged during this period. The gamification experiments conducted is the subject of an upcoming paper. The automated correctness check did noticeably increase the code review time per student. In the future this will be the default tool for that course.

Assignment validator features should be expanded with full blown sandboxes to run the executables in, the support for running queues to balance load and improve measurements, the support for more programming languages and UI languages, unit test code coverage calculator, and also data mining tools would be a useful addition. There are a number of small changes that should be made for example messaging students directly.

Since the field of gamification is vast, and so far under researched, there are many techniques that should be added and tried. Mechanics for pacing might be added for example leveling, and achievements. Also, mastery might be targeted with leaderboards.

References

[1] A. Aiken. Measure of software similarity: Plagiarism detection system. Technical report, Computer Science Division of University of California, 2002.

[2] A. L.-W. O. B. Cheanga, A. Kurniaa. On automated grading of programming assignments in an academic institution. *Computers & Education 41*, pages 121 —- 131, 2003.

[3] Y. L. B. Leong. Application of game mechanics to improve student engagement. In *International Conference on Teaching and Learning in Higher Education*. Citeseer, 2011.

[4] S. Deterding, D. Dixon, R. Khaled, and L. Nacke. From game design elements to gamefulness: Defining gamification. *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments, MindTrek 2011*, 11:9–15, 09 2011.

[5] X. Fu, B. Peltsverger, K. Qian, L. Tao, and J. Liu. Apogee: automated project grading and instant feedback system for web based computing.

[6] J. J.-D. G. G. Barata, S. Gama. Engaging engineering students with gamification. *2013 5th International Conference on Games and Virtual Worlds for Serious Applications, VS-GAMES 2013*, pages 24–31, 09 2013.

[7] D. A. G Gygax. *Dungeons and dragons.* TACTICAL STUDY RULES, 1974.

[8] N. W. G.E. Forsythe. Automatic grading programs. *Communications of the ACM, 8(5)*, pages 275–529, 1965.

[9] J. Hamari, J. Koivisto, H. Sarsa, et al. Does gamification work? - A literature review of empirical studies on gamification. In *HICSS*, volume 14, pages 3025–3034, 2014.

[10] S. Itamar. Using gamification and gaming in order to promote risk taking in the language learning process. *MEITAL National Conference*, pages 227 — 232, 2015.

[11] N. v. V. J. Hage, P. Rademaker. A comparison of plagiarism detection tools. *Technical Report UU-CS-2010-015*, 2010.

[12] D. D. J. Taylor. Constructed-response, computer-graded homework. *American Journal of Physics, 44*, pages 598–599, 1976.

[13] M. M. Striewe. A review of static analysis approaches for programming exercises. *Computer Assisted Assessment. Research into E-Assessment.*, pages 100–113, 2014.

[14] K. J. Majuri, Jenni and J. Hamari. Gamification of education and learning: A review of empirical literature. In *Proceedings of the 2nd International GamiFIN Conference, GamiFIN 2018*. CEUR-WS, 2018.

[15] M. Poženel, L. Fürst, and M. Viljan. Introduction of the automated assessment of homework assignments in a university-level programming course. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 761–766. IEEE, 2015.

[16] B. A.-H. Q. Zoubi, I. Alsmadi. Study the impact of improving source code on software metrics. *2012 International Conference on Computer, Information and Telecommunication Systems (CITS)*, 2012.

[17] H. H. R. M. Rottman. Computer grading as an instructional tool. *Journal of college science teaching, 12*, pages 152–165, 1983.

[18] B. F. Skinner. Two types of conditioned reflex and a pseudo type. *Journal of General Psychology 12*, pages 66–77, 1935.

[19] U. von Matt. Kassandra: The automatic grading system. *ACM Special Interest Group on Computer Uses in Education Outlook*, 22, 03 2001.

Faculty of Mathematics and Computer Science, Babeș-Bolyai University, 1 Mihail Kogălniceanu, RO-400084 Cluj-Napoca, Romania

Email address: imre@cs.ubbcluj.ro