Basic feedback on JavaScript code

Bram Nieuwenhuize (851514132)

May 5, 2015
version 1.1

1 Fundamentals of providing feedback

This article contains the outcome of a limited study towards elucidating the fundamental aspects of providing feedback on JavaScript code. The foundation of this research is a project which goal is the development of a software tool called JavaScript-examiner. The aim of this tool is support students to learn JavaScript in an academic distance learning environment, in particular while studying the course Web applications: the Client Side. This research is one of the artefacts to realize some context and generate content, to extract requirements and recommendations for the remainder of the main JavaScript-examiner project, from the perspective of providing feedback. From more or less general feedback as starting point, quite soon the scope is narrowed to the specific field of interest: Providing automatic feedback on JavaScript code. At first, a theoretical foundation is outlined, to create an overview. Some thoughts are given to Distance Learning, E-learning and learning a (program) language. This overview is used to penetrate the rich field of (freely available) existing online learning platforms to learn JavaScript, in enquiring the presence of didactics as proposed in academic writings and discovering some characteristic features. In addition, a tutor with experience in examining the mentioned course is interviewed to get more insight in difficulties and commonly made mistakes while programming in JavaScript. The results of these two field explorations are comprehended, to determine success factors of such a platform as is on the one hand and, more importantly, elucidate requirements and features for the JavaScript-examiner, that will contribute to mastering the art of programming in JavaScript.

2 Feedback in learning a programming language

When developing software it is often quite important to get familiar with the environment where the software will be used. In this case, developing for students like ourselves (even within the same discipline), it might be argued this knowledge is already present. To some extent this is true, like our experience in learning programming languages and even learning them aided by software designed for this purpose. At the same time, there is a lack of knowledge and expertise in teaching, providing feedback and motivation techniques. In assuming these aspects are important for the success of the project, as supported in the conclusion in [Alomari, 2009], some related articles are consulted.

2.1 Distance Education & E-learning

Next to the support of students, another goal of the project is to reduce the amount of time spent by the tutor to correct the (in JavaScript written) solutions to exercises. According to [Guri-Rosenblit, 2005] it is hard to achieve this goal using e-learning. This study focuses on the E-learning and distance education, showing examples where e-learning is not helpful in this aspect, and even increases the workload. This is something to keep in mind, as the bias of ICT automatically leading to a decrease of labour-intensity is often quite strong. Later on, some potential preventive measures to this concern are presented.

Another article, [Ypsilandis, 2002] , about feedback and distance education outlines some interesting observations. A shift from teaching towards learning is recognized in contemporary education. Feedback is more and more recognized as important assistance mechanism. This feedback should not only take place from tutor to student, but also among students themselves and even the student reflecting on the own work. According to (mentioned in the article) Ross(2000), there are three characteristics of feedback quality: timely, helpful, developmental. Thus, it should be provided at the right time within the learning process, have some instrumental aspects to really help the student to reach a next level and it should contribute in motivating to continue studying. Besides these general key issues, the field study points out two specific requirements. It should be possible to provide feedback on the exercises and the platform itself, in a way the student experiences some kind of dialogue. For example, the student should be able to make suggestions to improve an exercise, or utter their difficulties while using the tool. The other recommendation is to provide feedback in a print friendly format, as a lot of students still prefer reading from paper when studying, and store the feedback with the other, in hard copy provided, study materials. This last point is connected to the conclusion of [Shepherd, 2011], namely the demand of well written (full sentences) feedback instead of just some key points or audiovisual indicators.

2.2 Similar Projects

Now, with some general context about the field of use and feedback in general, it's time to move on and start to narrow the scope. To start a project like the JavaScript-examiner can be hard, if there is no guidance. Luckily there have been similar approaches. One of these approaches is a project with a similar goal, with the significant difference being the programming language, that is Java instead of JavaScript. This project [O'Brien et al., 2014] has an interesting approach, for using a supposedly known language (Python) to support the learning process of the language to be learned (Java). Besides the didactic aspects of this method, it's a way to reduce the, earlier mentioned, labor intensity for the tutor. Creating exercises by putting the solution more or less down in another well known programming language is likely to be far less challenging and time saving, compared to creating a a set of delimited solution requirements and problem description in well written natural language. The goal to reduce time of the tutor is even stretched further in this project. Students are supposed to write test cases themselves. The tutor only writes a very basic test suite. Students are challenged to add test cases to this suite. To enforce the students to write these test cases, the feedback functionality is only enabled if the student has already submitted one or more test cases. This seems to be a little too much, as it undermines the primary goal, and the skill of writing such test cases requires knowledge of the language in the first place. Still, the idea of rewarding the student for a contribution like adding a test case is something to keep in mind, as writing test cases is an important aspect in programming, and part of the course as well. To avoid a lot of similar test cases, a Code Coverage Library is used. In general, the tool in the mentioned project needs a problem statement written in a known program language and a template for the language to learn, a (at first basic) test suite.

Another project by [Watson et al., 2011] focusses on the form of the offered exercises. Their concern is the drawback of little and thoroughly delineated assignments. From a maintainability and developmental perspective these might be in favour to complex and realistic quests. According to their research, there is a demand for interesting and realistic problems to solve. This challenges the student to really grasp the fundamental skills of programming, and motivates the student as they experience there is really something to gain, in solving such a puzzle. In order to motivate students, their goal is creating a model to enable development of adaptive e-learning systems to provide individualized guidance and feedback. From a presupposed link between learning and playing, the model is inspired by the concept of Serious Games: games with educative purpose. According to them, games are appropriate because of their nature in developing skills through repetition and extending knowledge step by step. But, compared to most serious games, more comprehensive feedback is required, due to the complexity of learning a programming language. The proposed model is quite comprehensive. In order to meet the requirements for such a platform, exercises are represented by learning scenes (like levels in games), provided with a set of learning concepts, pre-required scenes (to create gradual complexity), a test suite and code similarity analyse (functional solution), required fragments and constraints (to be able checking semantic aspects as well). In comparison with the project mentioned before, the amount and complexity of artefacts needed for an exercise is much greater. But in creating more complex exercises, the amount of exercises needed to cover the material might be reduced to some extent. The model includes an approach for step by step feedback as well. This feature might be interesting for later stages or probably even subsequent projects to extent the JavaScript-examiner. In relation to this last point, [Burckhardt et al., 2013] might be interesting as well.

3 Existing Online JavaScript education platforms

Possessed with the background information, now it is possible to review some current platforms that offer learning programming languages like JavaScript. Though, before jumping in, it is important to have some plan to review the platforms. Based on the research in the previous chapter, the following characteristics and features will be reviewed:

- Platform characteristics (web-based, command shell, client, user interface)
- Exercises
  - Form (i.e. natural language, programming language, game)
  - Extent
  - Structure (i.e. gradual difficulty)
  - Challenging?
- Input format (i.e. template, empty text area, command-line)
- Feedback
  - General characteristics
    - Timely
    - Helpful
    - Developmental
  - Form (i.e. Well written, bullet points, model solution, print friendly)
  - Extent (i.e. on syntax (of JavaScript code), on functionality (of JavaScript code), on semantics (of JavaScript code))
  - Contributors (i.e. Creators of platform, users)
- Possibilities to send feedback to the platform and exercises

Of course, any not yet distinct, but possible interesting features, will be examined as well. As the goal is to get an overview of the aspects and facets rather then the specifics of each platform, the review is ordered by the distinct facets. Subsequently, the specific platforms are only explicitly mentioned if a present feature is considered useful to examine more closely later on.

3.1 Platform

All platforms are web based, except Nodeschool.IO [1] , which uses the Node.JS Command Line and NPM. One platform is visually presented as a game [2]. Most platforms offer learning modules for several languages. Some platforms use cookies, others require registration and login to keep track of progress. Login

[1] http://nodeschool.io/
[2] http://codecombat.com/play

is sometimes required to start learning. Some of the platforms enable logging in with Google or Facebook credentials.

In general, the user interface consists of a big area to write and / or show code, a quite small area with instructions, an area to show output and some indication of location and direction (the place of the current exercise within the course). On top of this, KhanAcademy [3] shows responses and related discussions about the current exercise as well. Most platforms require a mouse click to submit the solution, and some clicks again to move on to the next exercise. In Codeschool [4] it's possible to navigate through exercises by using keyboard short-cuts or keywords.

3.2 Exercises

All platforms have the exercises divided in several sections. The introduction starts often with a classical challenge: print Ḧello Worldẗo the console. KhanAcademy, Code Combat and Make Games With Us [5] have chosen a different introduction, in providing some functions that should be called to generate a visual action on the screen. For example, in Code Combat you have to collect Gems by moving the Hero through calling *moveRight()* on the *this* keyword.

In some cases, there is a clear learning curve, and it's required to complete a section to move on to the next one. Others, like Nodeschool.IO, just present different sections where distinct features within a section are linked to an exercise. While examining the platform, I felt more attracted to the platforms with a learning curve. But Nodeschool.IO enables the user to quickly learn something about a specific topic, without the need to complete any other sections. A solution that have both benefits, might be the possibility to enter a section any time, but mention the pre-required knowlegde for each section anyway.

The extent of covered programming aspects depends slightly, but the greatest part is still about core programming within a single module. Advanced software engineering practices (covered in this project within the Domain research by Boris Arkenaar) like modularity, design patterns, code elegance or performance indicators (i.e. use of memory, running time) are hardly covered. Exceptions are KhanAcademy (sections Object Oriented Design, Documentation, Readable code), and Nodeschool.IO (requiring for example recursive functions, restrictions on using specific constructions or functions).

The exercises itself range from one liners like print something to console, to (pseudo) realistic application like building or playing a little game. The scale of the latter is still quite small and still written within one module. Often, a section starts with the smallest exercises, and finishes with a big one where all aspects of the section come together. These big exercises often are more challenging and really demands the user to make use of the newly acquired skills. Sometimes it is hard to remember required facets from earlier sections, when demanded in a different one. Here, it would have been useful to make use of an (optional)

[3] https://www.khanacademy.org/computing/computer-programming/programming/

[4] http://JavaScript-roadtrip.codeschool.com/

[5] http://www.makegameswith.us/build-flappy-bird-in-your-browser/

refresh. Almost all platforms with an explicit learning curve have divided the material in the following sections: Introduction, Strings, Logic / Loops, Arrays, Functions, Objects.

3.3 Input

As already mentioned, NodeSchool.IO is somewhat different to the other platforms, and this goes along for the Input as well. It's the only platform where any text editor can be used. To provide some guidance, all exercises come with a template that can be copied to the newly created text file. All other platforms use a rich text area, often already provided with a template consisting of declarations, methods and commented instructions. The text areas are rich in the sense it's possible to undo or redo actions, cut, copy, select (multiple lines as well). Udacity [6] has also some simple input textfields for the purpose of submitting answers in natural language. KhanAcademy has a nice feature, where some information about the exercise is provided dynamically in the same textarea where the user can create the solution. This works quite nice, as there is no explicit switch from passive to active learning.

3.4 Feedback on solutions

While exploring the several platforms, an interesting fact came to surface. As long the user satisfies the assignments by providing correct answers , the feedback is quite nice. The points you receive, the next level that unlocks or an information box with Ẅell Donëbrings satisfaction and motivates to go on. The correctness of the answers is in almost all cases determined by the outcome of the executed code; passing the test suite is sufficient. Codecombat provides bonus points in case the code doesn't exceed a specified number of lines, but if you just enter all code on one line, it's easy to collect the bonus with any code that passes the test suite. *Make Games With us* has I nice visualization of the just created solution. Every time a user successfully solves an exercise, it's possible to use the functionality in the little game you are building.

Of course, the cases where the answer doesn't suffice is more interesting when looking at feedback. This is the moment a platform can transcend from being a mere JavaScript interpreter to an active education tool. Submitting code that couldn't be interpreted results in the kind of feedback you'll expect from a regular interpreter. Working hard on a solution that results in feedback like *Unexpected token ILLEGAL* is not very stimulating. Unfortunately, no tool seem to passively check the submitted code, in order to detect possible problems, before automatically interpreting that results in some kind of error that can't be easily interpreted by a JavaScript rookie.

When starting to submit code that perfectly well executes but doesn't pass, there is some variation in the feedback. Some platforms seem to perform feedback based on the actual test case whereon the solution failed. This is an

[6] https://www.udacity.com/

interesting method to determine more closely what goes wrong. *NodeSchool.IO* has another interesting feature, by checking for certain restraints. If the assignment states not to use a for-loop or a specific method, it really checks whether it's used or not in the solution, and provides accurate feedback in this matter.

3.5 Feedback on platform and exercises

KhanAcademy has a nice functionality in this aspect as well. All exercises are linked to a topic that is shown under the exercise. Within this topic, users are helping each other to solve the problems. Some other platforms have some kind of forum or q & a as well, accessible through a link that opens a new window. This shift of view is quite disorientating so the barrier to make use of this feature is quite high. The possibilities to provide feedback towards the platform itself is invariably limited, only a general contact function is provided.

4 General difficulties in learning JavaScript

The main topic of this article —how to develop a JavaScript-examiner— should be examined with help of a true J̈avaScript examiner.̈ Fortunately, M.S. Stuurman was willing to share her expertise. As author of the material and primary attendant of the mentioned course, she has a lot of experience with tutoring JavaScript. In a short interview she outlined the general goals of the course, the common difficulties for students, possible benefits gained with the tool for both students and tutors and some specific subjects that deserve special attention.

Webapplications: the Client Side is a course that really aims on elegant client side programming in JavaScript. Instead of offering a encyclopaedia consisting all possible constructs, it's based on the principle of active learning, whereby the student is supposed to learn by doing.

Examining the final exams often show ugly code, but code that does meet the functional requirements. Until now, these answers sufficed to pass the test, as it is hard to adequately prepare students with focus on elegance as well. It would be a great improvement when the tool obviates this emission. Another reoccurring flaw in the answers is redundancy. This makes the code hard to interpret and ever harder to maintain. Declaration of variables seem to be difficult as well. Often the variables and constants are declared here and there, instead of declaration at the top of the document. On top of this, the *var* keyword is not properly used, which leads to undesired global instances. The common *constructor* convention, declaring the method starting with an upper case character and not returning a value is also something to pay attention to.

The assignments in the course material are representative for the ones to use in the tool. Specifically the chapters 4,5 and 6 contain many good questions. Eventually, chapter 7 about *Object Oriented Programming in JavaScript* may be appropriate as well. The tutors of the course are willing to add the exercises to the tool, and it would be nice to be able to make alterations and additions after uploading the exercise. In this way, adjustments can be made to provide students with more effective support. Features to monitor the need of such modifications to the exercises, are desired as well. At the same time, the anonymity of the student should be guaranteed. This to avoid reluctance for a student using the tool, being anxious this may reflect negatively on their final result. The tool should explicitly state and effectively enforce this anonymity. The monitoring functions could for example contain data about commonly made mistakes, exercises that are time-consuming and the satisfaction rate in using the tool and specific exercises. The method as proposed in chapter 2, where students are challenged to write test cases themselves, may be interesting as well. Writing test cases is seen as an important skill to acquire.

5 Conclusion

When analysing the results, it's possible to extract some recommendations for the JavaScript-examiner, especially in relation with the feedback features. The need to really grasp the domain and context has been pointed out and the call to establish a critical attitude towards the desire and possible solutions to reduce the labour intensity came across. The possibilities to encourage feedback among students themselves is tempting, and really something to consider.

While zooming in and discovering things more closely related to our project, suggestions for print friendly, timely, elegant and well written feedback came up. The idea of students that can write test cases themselves to gain skills and at the same time improve the tool is very appealing, especially with the goal of reducing the time spent by the tutor in mind. In this way we may be enabled to create a tool that is capable of facilitating -the desirable- more realistic and interesting assignments. The review of several platforms mainly showed the limitedness of learning programming languages with help of a tool, when considering the feedback capabilities. Still, it can be very useful, now we have more insight in very specific pro's and con's in usability and usefulness of such tools. Basal considerations like the need to login and possible extensions like a discussion forum, ways of input and submitting both assignments and solutions, rewards that encourage to continue, forcing a sequence or let the student choose the order of completing the assignment and examples of nice ways to visually present the tool. The experience of getting feedback that makes no sense, has been productive for the research as well. Now, we know it makes little sense to present feedback that is just a thrown runtime error. In general, this part of the research also showed the relevance of the main project: there is much to win.

Conquer ugly code, that is a noble cause. A much needed one, according to the interview. Redundancy, variable declaration and writing proper constructors are focal points. The desire for more or less dynamically maintainability came up, preferably with the possibility to monitor usage in order to improve the assignments and meet the needs of the student. In relation to this aspect, the importance of the user anonymity was touched.

Of course, it's not likely the tool will meet all recommendations and conditions that came across. Still, when considering all these features in this stadium, we supposedly have more leads to properly design the infrastructure, that enables further project-groups to expand the JavaScript-examiner, in order to reach and maybe even transcends some of these quite appealing features.

Bibliography

Alomari, A. M. (2009). Investigating online learning environments in a web-based math course in Jordan. 5(3):F1.

Burckhardt, S., Fahndrich, M., de Halleux, P., McDirmid, S., Moskal, M., Tillmann, N., and Kato, J. (2013). It's Alive! Continuous Feedback in UI Programming. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 95–104. ACM.

Guri-Rosenblit, S. (2005). 'Distance education' and 'e-learning': Not the same thing. 49(4):467–493.

O'Brien, C., Goldman, M., and Miller, R. C. (2014). Java Tutor: Bootstrapping with Python to Learn Java. In *Proceedings of the First ACM Conference on Learning @ Scale Conference*, L@S '14, pages 185–186. ACM.

Shepherd, M. (2011). Graduate student preference for instructor feedback in MBA distance education. 14:1.

Watson, C., Li, F. W. B., and Lau, R. W. H. (2011). *Learning Programming Languages through Corrective Feedback and Concept Visualisation*, pages 11–20. Number 7048 in Lecture Notes in Computer Science. Springer Berlin Heidelberg.

Ypsilandis, G. (2002). Feedback in Distance Education. 15(2):167–181.