"Digital Show-How": Extreme Active Learning for Introductory Programming

David Stotts
Dept. of Computer Science
Univ. of North Carolina, Chapel Hill, NC, USA
stotts@cs.unc.edu

Abstract— **For the past ten years we have employed active learning methods in introductory programming classes at UNC Chapel Hill. In this report we present experience with one such method we call "digital show-how", in which students learn to program apprentice-fashion, by doing programming in class. Digital show-how is extreme active learning. Students do *a lot* of programming, in real-time, in class as the instructor shows them how. They get instant feedback on their work, and so have constant low-stakes assessments. We describe the goals for the classes, the tools and methods we have developed to support digital show-how, and the results we have measured on how well the goals are being met.**

Keywords—active learning, programming education, low-stakes evaluation, automated evaluation, pedagogy, apprentice, teaching at scale, digital show-how

I. Overview

For the past 10 years we have been employing an active learning approach in introductory programming classes at UNC Chapel Hill. In the early incarnations we taught JavaScript using web browsers and text editors as the technical infrastructure. Class "lecture" was replaced with the instructor writing code in class while the students did the same on laptops. While this approach was preferable to traditional lecture format for both the students and the instructor, students identified several difficulties in trying to follow an in-class programming format, especially as programs grew in complexity and length.

To overcome these issues, and to more directly support this "show-how" technique, we developed the Bricks teaching environment. Bricks embodies what we call "extreme active learning" – active learning taken to extremes. Bricks has enabled effective teaching of larger programming classes with fewer resources, has increased class attendance dramatically, and has decreased the load on TA office hours. Use of Bricks has been enthusiastically received by the students and has led to increased student performance on evaluations.

In the following sections we first give the objectives we have for the Bricks project, as well as our measured outcomes from teaching with digital show-how. Then we describe the student and instructor experiences in a typical class with Bricks. The extreme active learning pedagogy embodied in Bricks is then outlined, and finally we give an architectural description of the Bricks software. We end with a future studies section.

This project is supported by grant funds from and the Center for Faculty Excellence at UNC Chapel Hill, and the UNC System General Administration.

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

II. Objectives and Educational Outcomes

The methods we embody in Bricks have been tested in 12 offerings of COMP 110 at UNC Chapel Hill, with class sizes ranging from 15 to 140. The course is entitled "Introduction to Programming using JavaScript"; it is introductory in that no prior programming experience is assumed or needed. The course is also populated mostly with students who will not become active programmers, but who do want some exposure to the workings of their digital world.

Bricks and its pedagogy are designed to help meet the following personal, departmental, and campus goals. Many of these goals arise from rapidly growing demand for seats in programming classes, but non-increasing resources such as faculty and TA instructional hours:

- engage the students in active learning methods during class meeting time
- increase class attendance levels and keep them high for the duration of the course
- reduce the load on TA office hours
- provide more consistent and timely (rapid) evaluation and feedback to students, since new assignments build on previous one
- increase content comprehension and improve mastery levels of the students
- ameliorate the "technical mystification" first time programmers often experience with more complex programming languages and IDE tools
- make success at programming more attainable by students not likely to choose the subject as a career
- cut down on "getting lost"… falling behind on content understanding and then never recovering or catching up due to the cumulative nature of the subject
- serve more students with limited instructional resources

We have evaluated the "show-how" approach in each offering of the course. The UNC Center for Faculty Excellence (CFE) designed and executed these evaluations, and we used the feedback from them to improve the system and approach after each offering. The methods were shown to be effective at improving student comprehension of the subject, based on exam performance, up a half letter grade on average. Class attendance increased to above 90% per meeting in the larger classes (and high 90's for smaller ones). Student content

understanding is also better, as is shown by considerably fewer of them spending excessive time with the TAs during office hours. We think increased understanding is partly due to greatly increased class attendance (learning takes place in class), and partly due to the constant practice that using Bricks gives to students (extreme repetition).

Class discussions are lively and participation is not dominated by a few. Bricks has the ability for the instructor to project student code for class discussion. Often a student will ask why the approach they use seems to work when it is different from the one demonstrated in class; the instructor will project the code and ask class members to offer pros and cons for each approach. The students are very enthusiastic in participating in these discussions.

The in-class undergrad TA has been crucial for maintaining good in-class programming pace without stopping for a lot of small syntax issues. This is, in turn, is made possible by the on-line real-time program help request feature of Bricks.

Finally, the "show-how" approach has been very well received by the students. In CFE exit interviews, many describe the class as being fun, with some saying that the course is one of the best they had taken. In comparing these interviews with a baseline taken in pre-Bricks offerings, the perceived value of class attendance is up considerably post-Bricks.

III. "A DAY IN THE LIFE"

A class meeting with Bricks is far from a traditional lecture. Students spend the entire time "doing" rather than sitting and only listening. Each student is expected to read the supporting materials (on-line text, PowerPoint slides) prior to attending a class. The class meeting then consists of 4 or 5 "type what I type" (TWIT) program developments that illustrate the language topic of the day. The students log on to Bricks, and see the problems for the day. A problem is selected by the instructor (and the student), and the problem specification is displayed in Bricks. The instructor begins to solve the problem (write code for it) using the same student interface that each student uses.

In the code editor window the students literally type the same things that the instructor types as the problem is solved before the class; the instructor's work is projected overhead, and as an instructor codes they engage the class in discussion about strategy, methods, language features to choose, etc. When done the students submit their programs to the Bricks server for instant grading, and receive credit for getting it functionally correct (proper output) and stylistically correct (good code structure and language usage).

Bricks has an online help feature that allows a student to submit a question about a program that is not working (more details in following sections). This on-line help is critical to the success of the digital show-how methods. Undergraduate TAs in class receive and answer student questions in real-time as code is being written (during "lecturing"). The instructor need not to stop for questions such as "why does it say semi-colon expected at line 5?", but the students who are confused with syntax are able to keep up with the progress of the programming.

Programs guided by the instructor in class receive full credit (albeit small credit -- many small stakes assessments). If a student chooses not to attend class, they can still work the "type what I type" (TWIT) programs later, but after class they receive half credit; thus, attending class is rewarded. Each day's module contains 4 to 6 "do it yourself" (DIY) problems that are similar to those done with the instructor in class. Students work on DIY programs as homework, and receive credit via instant auto-grading just as they do in-class. They may also submit on-line help requests for DIY programs and receive assistance asynchronously.

All student program submissions (TWIT, DIY, WALL, EXAM) are kept in a database, correct or incorrect. They are available for students to review at any time. Any submission can be loaded directly into the coding editor, and so students are encouraged to construct new programs by starting with their previous ones. All help requests are saved as well, and are reviewable at any time by students through the Bricks interface.

In summary, a student learns by doing in-class what the instructor shows them how to do, and then later applying that practice to new (but similar) problems on their own. As the learning of smaller concepts accumulates, larger programming assignments (WALLS) require assembling these structures into larger codes; these are worked on by students over a time-frame of a couple weeks each.

IV. PEDAGOGY: EXTREME ACTIVE LEARNING

Traditional old-school programming classes have depended on lecture for explaining or enhancing material presented in detail in text books (or web pages). The student experience in such a class is that learning happens as they work through 5 or 6 large programs (not in class). Many students traditionally struggle with how to combine the many smaller syntax-based features (loops, function calls, conditional branches, etc.) into larger structures to solve larger, more complex problems. To gain this understanding, they attend TA office hours, where they get hands-on demonstrations of the concepts that were talked about in class. With Bricks, we make this hands-on demonstration *be* the class content and experience.

This traditional approach is analogous to teaching house carpentry by asking a person to read a book on framing and then telling them to build a 3-bedroom 2-bath cottage with screened porch. A more obviously effective method would be first to have them build a box or two, then a dog house, then a child's playhouse, then perhaps a one-car garage… and then perhaps a 2-car garage with stairs and an upper room. Only then can something more complex like the full cottage be attempted with real hope of student success.

We think our apprentice approach is the right one to employ for teaching programming as well (and our measured outcomes support this belief). If you want to learn to program, you should

find someone who knows what they are doing, and do what they do. If you want to learn to write code, you must write code -- *a lot of code.* You must practice the small things, repeatedly, and then learn how to assemble the small things into larger things. This is the hands-on apprentice-style pedagogy we encode and use in the Bricks environment. Hands on coding is effective, for example, in tutorial applications (Codecademy [8], for example). In Bricks, we combine this with the advantages of in-class instructor guidance and peer discussions.

The extreme active learning approach used in Bricks has been developed over about ten years of informal practice, with the last five years directly supported by the Bricks software system. Bricks encapsulates our pedagogy and effectively manages ever larger class sizes with minimal resources. Bricks supports active learning via apprentice-based methods, some class inversion, instant feedback, numerous low-stakes evaluations, and peer review/discussion.

We trace some of the ideas we have built on to two texts: “Fluency with Information Technology” [1], and “Learn Python the Hard Way” [2]. With older, traditional intro classes (in Java) many students have noted that Java was mystifying to them. For example, consider the simplest “Hello, World” program in Java (Table 1, left side). In addition to the complexity of the text for such simple output, a student must download a cumbersome IDE like Eclipse and spend no small amount of time deciphering the many panes, menus, file hierarchies, project structures, and other incantations needed just to get “Hello, world” to print out.

Fifteen years ago Larry Snyder noted [1] that JavaScript allows teaching programming with little mystery; a student needs no tools other than those already understood: a text editor and a web browser. Compare the previous Java code to the equivalent intro program in JavaScript (Table 1, right side). It has no unexplained parts, no IDE that won’t run efficiently on half the students’ laptops, and no complexity. Since it runs in a Web browser, a friend or parent can execute it as well over the Internet (“*Look, Mom and Dad, at what I coded up*”).

TABLE I. JAVA VS. JAVASCRIPT

Java program	JavaScript program

`public class FirstProg {`
`public static void main(String[] args)`
`{`
`System.out.println(“Hello, world”);`
`} }`
	`alert(“Hello, world”) ;`

For this reason, we began teaching COMP 110 in JavaScript over ten years ago, and we used a form of in-class coding to augment in-class lectures. This was well received by students, but once programs reached a certain level of complexity it was difficult for students to keep up in their editing. The solution was to create Bricks to directly guide students to keep up, and also to give credit for participating. Shaw [2] uses a “type what I type” TWIT approach to learning the syntax and usage patterns of Python, engaging muscle memory through typing and directly exploiting apprenticeship in teaching. Combining these, and adding in low stakes evaluations, instant feedback (auto-grading of programs), and a database of submission history produces the “show-how” capability of Bricks for classroom extreme active learning

Kent Beck, creator of Extreme Programming (XP) [6,7], explains using “extreme” in naming his method: if a practice is known to be useful for creating good software, then let’s see if we can multiply the effectiveness by taking that practice to an extreme. We use the same reasoning in our extreme active learning practices. If you want to write code, then we will write an extreme amount (compared to traditional classes). Students write about 150 programs over the course of a semester. These range from 1 or 2 lines (the smallest fundamentals) up though 10-15 lines, on to 30-50 lines, then up to more traditional assignments of 100 to 200 lines. You need feedback, so we give indicators of success early and often – constantly – with numerous small-stakes evaluations and instant grading of each program submission.

Bloom’s Taxonomy of learning stages [3,4] shows that the foundational level of any learning experience is “remember” or basic fact acquisition. Extreme active learning in Bricks supports this directly with the daily “type what I type” exercises in which students practice applying smaller language features and syntax concepts appropriately in small coding (guided by the instructor). The instructor provides explanations while coding of why the demonstrated methods work, supporting the second level of Bloom’s -- the “understand” level, where concepts and ideas are fleshed out (using basic facts as support). Bloom’s third level – apply ideas to new situations – is then supported in the DIY problems in Bricks, where students practice their in-class learning on similar “homework” programming problems. Finally, Bloom’s fourth level – analyze and draw connections among ideas – is supported in Bricks with the larger, multi-week WALL programs. All smaller programs in Bricks are reviewable and editable by a student, so the larger WALLS can be built on, and composed from, prior TWIT and DIY work. It can even be argued that Bloom’s fifth level (evaluate – justify a stand or decision) is supported in that student code is often projected for class discussion of methods and comparisons to other approaches. This is a form of peer evaluation as well.

V. THE BRICKS ENVIRONMENT ARCHITECTURE

The Bricks system is built as a client-server Web application. Two separate browser-based clients interact with the server:

- a student coding interface, for program development and feedback review (Fig. 1)
- an administrator dashboard, for tracking student/class progress and for creating content (Fig. 3)

A. Student Interface

The Bricks student interface is shown in Fig. 1. Student use only this client, both in class and out. In class, a student will log on and interact with the Bricks problems for the day via this

interface. It allows coding, receiving scores on code, and getting real-time assistance on code. Students select and read a problem description (program specifications), write JavaScript code in an editor window, browse their prior submissions for the problem, test the code locally (with the JavaScript engine in the browser), and when satisfied that the problem is solved correctly, submit the program to the server for scoring. Instructors also use the student interface when working in class, acting as a student when demonstrating how to code solutions to the basic instructional TWIT problems.

Outside of class, students use the student interface to check their scores and to work on homework -- the "Do-It-Yourself" (DIY) coding problems. They also read the on-line text through Bricks, and submit program help requests if needed (and review replies they receive).

Fig. 2 shows the view a student has when a help request is answered. The left panel shows the code submitted, the error messages from the server about that program, and the student's question. The right panel is the reply from the instructor. It shows the text of the answer, and it also may show annotated code. One help request can be made per submission; all are saved in the database for review by students at any time.

B. Administrator Dashboard

The administrator dashboard for Bricks is shown in Fig. 3. The instructor uses this dashboard outside of class to define problems, solutions, and grading criteria, and to organize content. In class, the instructor uses the dashboard to monitor student performance on each program being demonstrated. As students are working in class, typing what the instructor types, the instructor will view the dashboard "matrix" for the problem. The matrix is an array of color-coded student icons that show progress (a progression from not started through several stages until completely done). Matrix icons also show which students have requested help, and which wish to have their code shared.

Help requests can be answered through the dashboard both in class (synchronously) and out of class (asynchronously). We have in-class TAs that use the dashboard to answer on-line help requests in real time during TWIT programming. This keeps the instructor demo flowing along smoothly, with very few interruptions for simple syntax problems.

When a student shares code, the instructor may then use the dashboard to project, edit, and execute that code for the class to discuss. The projection screen is an active editor, so the instructor can add and execute student code suggestions for all to view and discuss. Finally, the dashboard provides easy viewing of the database for all saved student submissions; correct code as well as incorrect code is saved and viewable. Each student sees only their own work (through the student client) but instructors can view all work for all students. The work is sliced as all students on one problem, or all problems from one student; scoring summaries are also shown.

C. Bricks System Architecture

The architecture of the Bricks system is block-diagrammed in Fig. 4. The dotted boxes outline the components of the student client, the administrator client, and the common server. The two client side components (student and administrator) each run in standard browsers and use Bootstrap [9] for the UI look and feel. The server side is a *node.js* [10] server with embedded JavaScript engine; the server interacts with MongoDB [11] as a persistent store for user information, problem definitions, student programs, help requests, and scores. The server subsystem is structured and launched using the Sails MVC framework [12] on a Linux platform, and listens to specific ports for HTTP requests from the student and admin clients.

We have structured Bricks to scale to larger classes. Analysis of student programs for style correctness is done on the client-side, on the student laptops. Results of this analysis are sent to the server as a metrics object along with the program for grading. The server still must execute each student program for checking functional correctness but is not burdened with style analysis. The matrix graphics in the administrator dashboard can presents information about several hundred students at a glance in compact form.

VI. Future Research Directions

Though Bricks is continuing to be used for in-class instruction, a version has been developed for use in on-line courses as well. Over 100 screen-capture videos have been added to provide the on-line equivalent of instructor demonstration coding. This version is under experimentation in on-line courses at UNC Chapel Hill and is so far showing student acceptance and teaching effectiveness results similar to Bricks in-class. Another version of Bricks has been developed that allows group formation and collaborative editing of code, for pair programming and other collaborative development experiments. Finally, through Brick as reported here is for teaching using JavaScript, a version is being developed that instructs in TypeScript.

References

[1] L. Snyder. Fluency with Information Technology: Skills, Concepts, & Capabilities. Pearson, Boston, 2013 (5th ed.).

[2] Z.A. Shaw. Learn Python the Hard Way: A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code. Addison-Wesley Professional, 2013 (3rd ed.).

[3] B.S. Bloom, M.D. Engelhart, E.J. Furst, et. al. "Taxonomy of Educational Objectives: The classification of educational goals," in Handbook I: Cognitive Domain. New York, David McKay Company, 1956.

[4] L.W. Anderson and D.R. Krathwohl (eds.). A Taxonoy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy and Educational Objectives. New York, Longman, 2001.

[5] P.D. Stotts, "Active methods for teaching programming," at Frontiers of Engineering Education Symposium, National Academy of Engineering, Irvine, CA, Sept. 25-8, 2016. https://www.naefoee.org/symposia/currentsymposium.aspx

[6] K. Beck, "Embracing change with Extreme Programming," IEEE Computer 32(10), pp. 70-77.

[7] K. Beck. Extreme Programming Explained. Addison-Wesley, 2004.

[8] Codecademy: Learn to Code. https://www.codecademy.com/

[9] "Bootstrap 3 Tutorial," W3 Schools Tutorials, https://www.w3schools.com/bootstrap/

[10] "Node.js Introduction," W3 Schools Tutorials, https://www.w3schools.com/nodejs/nodejs_intro.asp

[11] "Learn Mongo DB," Tutorials Point: Simply Easy Learning, https://www.tutorialspoint.com/mongodb/

[12] Sails.js – Realtime MVC Framework for Node.js, https://sailsjs.com

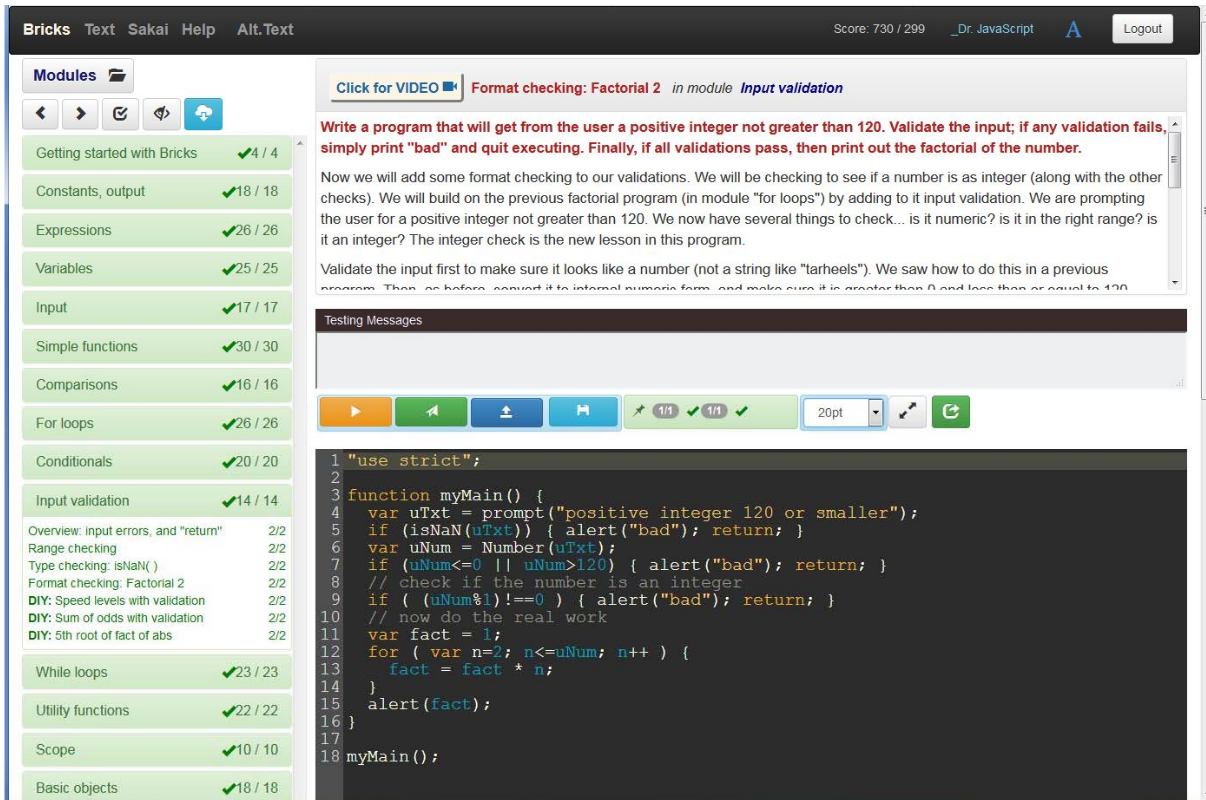

Fig 1: Bricks student view

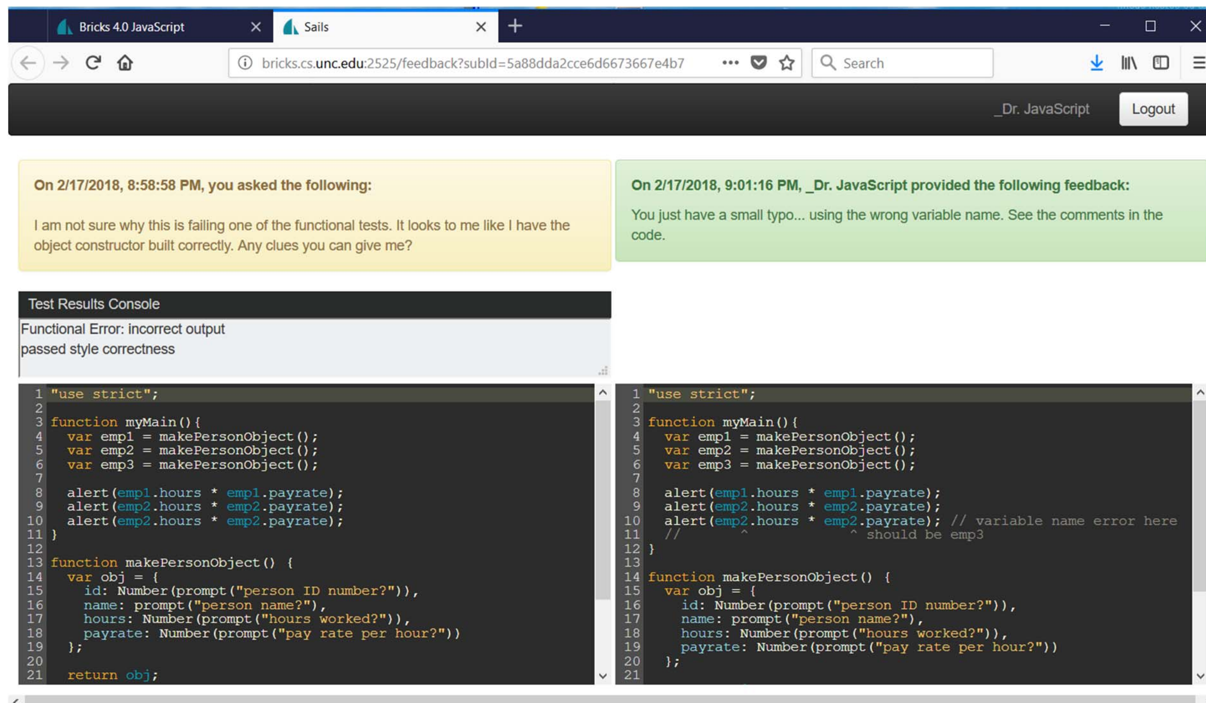

Fig 2: Student view of reply to help request

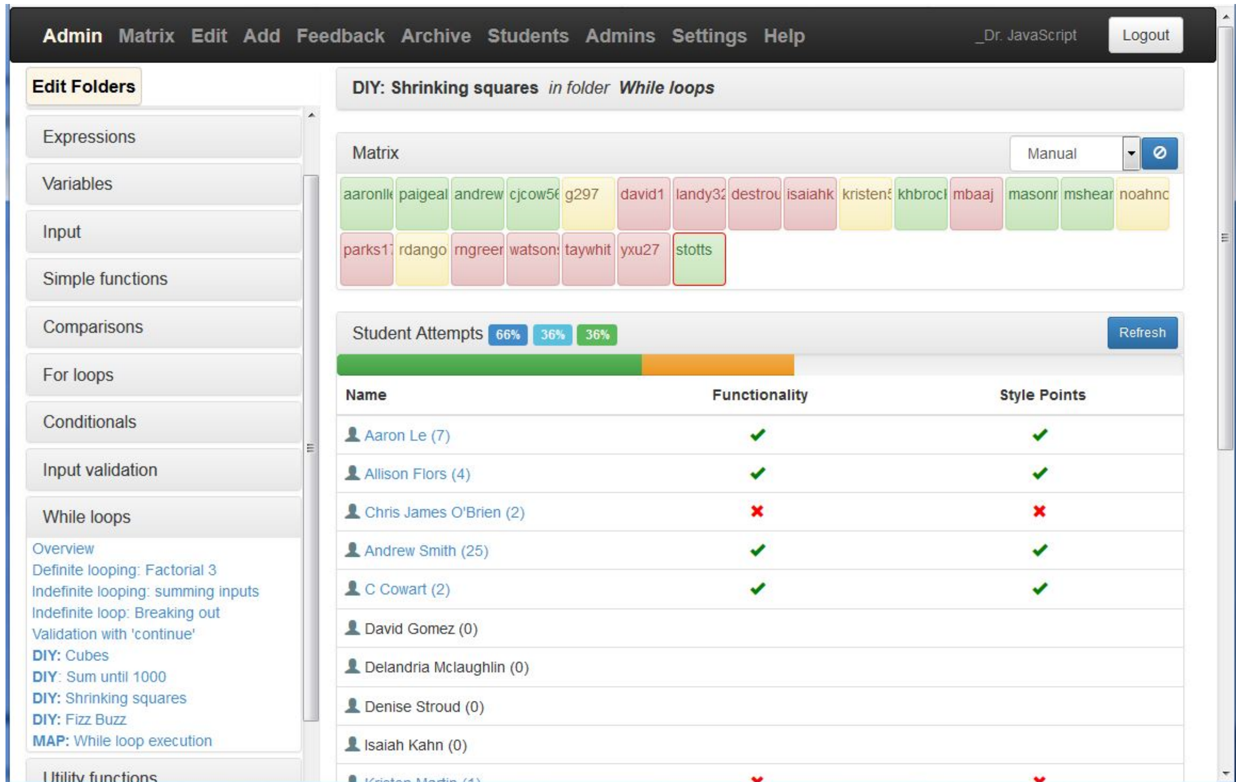

Fig 3: Bricks administrator view: the Matrix

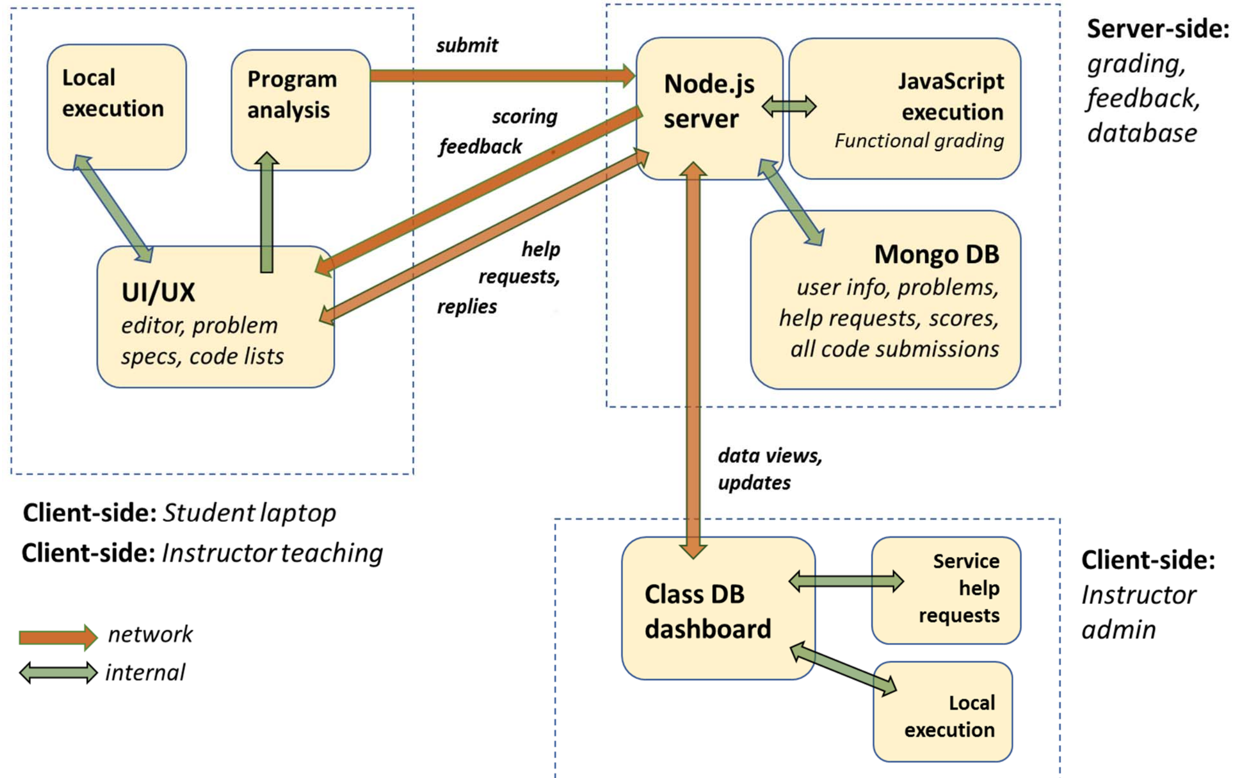

Fig 4: Bricks architecture diagram