EPHP – A VISUALISATION TOOL FOR LEARNING SERVER-SIDE WEB DEVELOPMENT: INITIAL WORK AND PILOT STUDY

Nick Whitelegg and Olufemi Isiaq
Southampton Solent University
United Kingdom

Abstract

Over the course of more than 10 years teaching PHP, we have come to recognise that many students find the architecture of web applications confusing and in particular often confuse the programming constructs to read user data from the front-end with those used to obtain data from the back-end database.

Our *EPHP* software aims to address these problems via a visual simulation of the operations of a PHP-based web application. A prototype has been used already by students on our second-year unit "Developing for the Internet," and feedback via a survey obtained. We discuss points for enhancement identified from the survey and our own observations of the students using the software.

Keywords: visualisation, learning tool, web development, learning programming

Background

The Challenges of Teaching and Learning Web Development

For many years, teaching and learning programming has been recognised as a significant challenge. A number of reasons have been put forward for this. Two major international, multi-institutional studies were done in the early years of the 21st century by McCracken et al. (2001) and Lister et al. (2004). The former found that many students lacked the ability to solve problems, while the latter found (via a series of code-based multiple-choice questions) that many students had an inability to comprehend the semantics of small sections of code. Other studies (Radošević, Orehovački, & Lovrenčić, 2009) suggest basic syntax is a problem for many students.

This has been backed up by our own experiences at university on a range of programming units: students frequently struggle with a range of aspects of programming, ranging from basic syntax errors, confusion with object-oriented concepts, and problem-solving skills.

Web development, as pointed out by Zhang and Dang (2015), raises a number of unique problems not seen in other types of programming, due to the multitude of different technologies (e.g., HTML, CSS, JavaScript and a server-side language) and due also to the distributed, client-server nature of web applications. A typical web application consists of three components: the web browser; the web server, such as Apache (The Apache Software Foundation, 2016), which executes server-side scripts written in a language such as PHP; and the back-end database system, such as MySQL or Oracle.

The role of PHP, the focus of this study, in such applications is to receive user input from the browser, look up data in a database, and format that data for presentation to the end user, which is sent back to the browser as a response. This system is more complex than a simple desktop application, increasing the scope for confusion on the part of the students (see Figure 1).

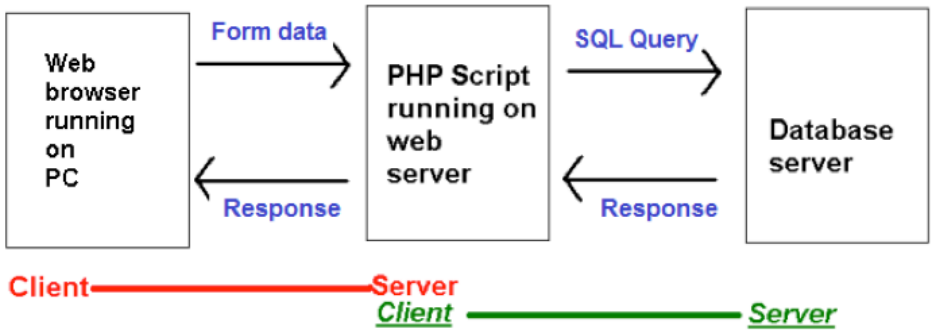

Figure 1. The components of a typical PHP web application.

The principal author has been running and teaching a second-year undergraduate unit entitled "Developing for the Internet" for 14 years. This unit focuses largely on server-side web development using PHP. During this time, we observed several common problems encountered by students. Frequent issues come from students appearing not to understand that PHP runs on a server (a common problem is opening the PHP directly from the hard drive of their machine in their browser, rather than requesting it from a server by typing in a valid URL), and confusion between the three components of the system described above (students frequently confuse data being sent to a PHP script from a web form and data being received by a PHP script from the database).

Previous studies back up our own observations that web development is a particular challenge. Park and Wiedenbeck (2011) made a "study of help-seeking activity in an introductory Web development course" (p. 125). The problems included the code itself but also other, ancillary issues such as how to transfer files to a server.

Alston, Walsh, D., & Westhead (2015) interviewed five lecturers as part of their study, who gave feedback on what they believed to be the main issues causing problems for their students. One participant identified that problems in web development come from the details being behind the scenes; giving the quote "a lot of it is hidden from the students" (p. 2). This is certainly the case: for example, a beginning student may not realise that when entering a URL in a web browser, a request is sent to a web server via a request which delivers the requested page as a response. Another example is the fact that a PHP-based web application requires a web server, such as Apache, to run, as described above – which is a likely source for the misunderstanding we have observed where students attempt to run PHP scripts directly.

Park and Wiedenbeck (2011) also raise the issue of "decomposition" (p.131) in which they propose that students have difficulty breaking down what they see as a single whole (a web page) into its components (HTML and CSS); Alston et al. (2015) raise similar points. By inference the same principle could lead to students being unable to break down a PHP application into its three components (HTML, running in a browser; PHP, running on Apache; and a back-end database).

A further study on the difficulties encountered by students in web development was done by Zhang and Dang (2015). These authors researched (by means of a survey) factors affecting students' issues when learning web development with C# and ASP.NET. Findings from this research are also enlightening: for example, "code behind" (p. 3), a specific feature of .NET development (Microsoft, 2009) was seen as difficult and other comments made by students included difficulties understanding "why things go where they do" (p. 3) and "knowing what file structure does what, how to program the code behind file" (p. 3). These observations appear to support those of Alston et al. (2015), who, as described above, obtained the comment from one of their interviewees that "a lot of it is hidden from the students" (p. 2) – and (with the caveat that .NET web development is different in style to PHP) also support our own.

In summary, then, learning web development is a significant challenge when the problems with learning any type of programming are added to the difficulties in comprehending the relatively complex, multi-language client-server system that is a database-driven PHP server-side web application.

Existing Interactive Learning Tools for Web Development

A number of online environments exist to help students learn web development involving interactive exercises, including Codecademy (Codecademy, 2017) and CodeAvengers (Online Education Ltd, 2016). Codecademy consists of a series of interactive exercises for learning various web development technologies, including PHP. Exercises involve writing code in the browser or filling in blanks in existing code. Codecademy includes a split view in which the output from the code is shown on the right hand side of the screen (including syntax errors). Detailed feedback is given for errors, and badges given for completing exercises.

Codecademy covers language features such as loops, arrays, conditionals, and variables. However it does not cover the client-server aspect of PHP, such as receiving form data, understanding HTTP requests and responses, and database interaction.

In addition, some IDEs, such as PHPStorm (JetBrains, 2017) include full debugging tools such as breakpoints and watches. Nonetheless, none of these tools really address the problem of effectively presenting the distributed, multi-component, multi-language nature of a server-side web application and allowing students to understand how it works as a whole.

Visualisation

A good deal of work has been done in using visualisation tools to help students learn general programming, using languages such as Java or C++. Tools include EVizor (Moons & Backer, 2013), JEliot (Ben-Ari et al., 2011; Ben-Bassat Levy, Ben-Ari, & Uronen, 2003; Moreno & Joy, 2007) ViLLE (Kaila, Rajala, Laakso, & Salakoski, 2009), and Verificator (Radošević et al., 2009). However, little or no work appears to have been done on using such tools for learning web development.

A common approach made by these tools is to present an animation, involving stepping through the code, highlighting the current line of code, and showing users what is going on graphically. This is done by means of depicting code constructs such as variables and their contents, objects and their contents, loops, or function or method calls and their parameters. These graphical depictions are typically in different colours for emphasis.

Success in improving student learning with these tools has varied somewhat but trends towards positive. For instance, with EVizor (Moons & Backer, 2013), students were asked questions on the operation of a recursive program. Results indicated significantly more understanding after using the tool, and furthermore, answers given were at higher levels of the SOLO taxonomy (Biggs & Collis, 1982, as cited by Moons & Backer, 2013). The design of EVizor was informed by learning theories including constructivism and cognitive load theory – in the case of the latter, not overloading someone with too much information to "reduce load on working memory" (p. 370). Guidelines from cognitive load theory incorporated in the tool included placing "related concepts (of similar or different media, such as pictures and explanatory text) closely together" (p. 373) and that "information should be kept concise, and irrelevant information should be hidden" (p.374).

ViLLE (Kaila et al., 2009) is another such tool. It steps through code and includes separate windows for program output, current values of variables and visualisation of techniques such as recursion. It includes play, fast forward and rewind buttons and controls for speed. A study using ViLLE revealed that it was of particular benefit to "novice programmers" (p. 25).

JEliot, which has had several, progressively enhanced versions, has been the subject of the most studies amongst this group of tools, with mixed results. Moreno and Joy (2007) used JEliot 3 as the subject of their study, with mixed results: on the one hand, students gave positive feedback in that they believed it helped them understand concepts such as "object creation" (p. 55), but on the other hand it did not appear to help them actually write the code which uses such concepts.

Ben-Bassat Levy et al. (2003) performed a study with the earlier version, JEliot 2000, amongst a group of Israeli "10^{th}-grade" (p. 2) school students. Concrete results were more encouraging than the later study here; students were able to articulate programming concepts more clearly. It was found to helped medium-range students to a greater extent than either strong or weak

students; the investigators believed that stronger students "did not really need it" (p. 11) and that the "weakest students are overwhelmed" (p. 11).

Naps et al. (2002) derived a taxonomy for student engagement with visualisation tools based on a survey of educators plus analysis of the existing literature. This taxonomy involves 6 levels of interaction with a visualisation tool: "no viewing," "viewing," "responding" (answering questions using the tool), "changing" (making the visualisation work differently), "constructing" (changing the visualisation itself) and "presenting" (to an audience) (p. 142). With the Naps taxonomy it is proposed that visualisations need to be at levels three or higher to have benefit, i.e., students should not just watch a visualisation but should at the very least be able to interact with it, at the "responding" level (answering questions presented by the tool). The ViLLE (Kaila et al., 2009) study confirmed that the tool was only of use when students interacted with it at level 3 or higher.

A number of findings can be summarised from this analysis of the literature:

- The visualisation tools examined present programming constructs graphically, typically by stepping through the code and at the same time visualising the constructs (such as simple variables, or objects, and their values) in adjacent windows.
- Visualisation tools generally have reasonable success, but only when students interact with them (Naps level 3) as opposed to passively watching them (Kaila et al., 2009).
- There is a need to place related information together (such as graphics and their explanation) and hide unnecessary information (Moons & Backer, 2013) to avoid information overload and thus confusion (Ben-Ari et al., 2011; Ben-Bassat Levy et al., 2003)
- Tools appear to benefit beginner or intermediate programmers more than experts (Ben Bassat-Levy et al., 2003; Kaila et al., 2009), presumably because experts are happy learning programming without the aid of a visualisation tool.

EPHP – The Learning Tool

We now present the initial prototype of our visualisation tool for learning web development, *EPHP* (the "E" standing for "easy"; the hope being that it will ease the process of learning PHP web applications).

EPHP is a web-based application with the server-side component itself written in PHP; the client-side is written in JavaScript with an HTML5 and CSS-based interface. In order to visualise the client-server nature of a web application, the screen is split into three windows. The left-hand window represents the client (the browser) while the right-hand window represents the server. In between is the "Network" window, which represents the communication between client and server.
The client window consists of two tabs, *Develop* mode and *Simulation* mode. In Simulation mode, the client window contains a "browser within a browser"

in which the user can enter a URL and visualise the HTTP request being sent to the server. This is shown in Figure 2:

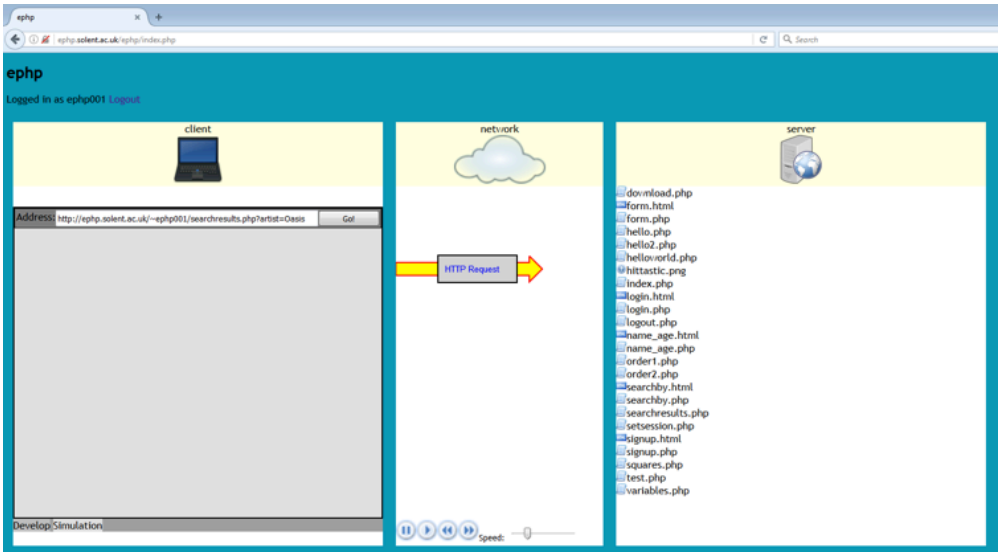

Figure 2. The EPHP user interface.

Furthermore, the user can roll the mouse over the "HTTP request" box and view – and edit – the HTTP request being sent to the server. In this way, EPHP aims to help students visualise the client-server communication that takes place behind the scenes when a user enters a URL in a web browser.

When a request is received, the requested file is highlighted server-side, and sent back to the client as an HTTP response that can, like the HTTP request, be viewed and edited by the browser. Finally the requested page is loaded into the "browser within a browser" in the client window.

Develop mode allows users to enter HTML or PHP code using an embedded Ace editor (Cloud9, 2017) and upload to the server using FTP, allowing users to make use of EPHP as a single tool to develop, upload and test their code.

As well as visualisation of client-server communication, EPHP aims to help students understand the relationship of PHP code to data from web forms, and to results from a database query. A user can enter a search term in a web form (which sends its data to a PHP script) in the "browser within a browser," visualise the HTTP request, and see the PHP code stepped through on the server window. PHP variables are highlighted, and a user can roll over them to see their value. When they are highlighted, the corresponding form field is also highlighted. Thus, EPHP shows the relationship between the PHP code and the form fields. This is shown in Figure 3.

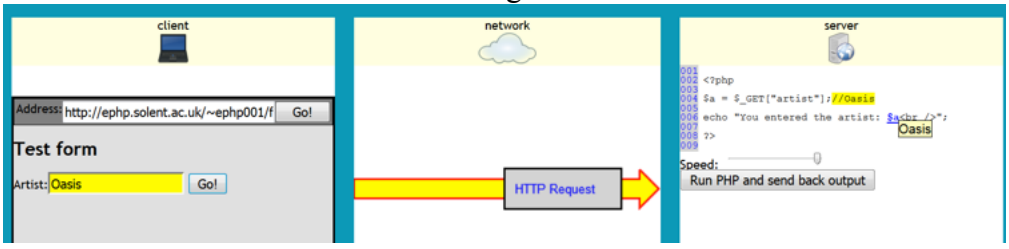

Figure 3. Reading form data in EPHP.

This example shows a simple PHP script to read in the artist the user entered in the form into a variable, **$a**, and then display a message confirming to the user which artist they entered. Note how the form field is highlighted in yellow, and a comment, also in yellow, has been added to the line of code which reads in that form field (i.e., `$a = $_GET["artist"]`). If there are multiple form fields, each will be highlighted in a different colour, and the corresponding line of PHP will have a comment of matching colour added to it, explicitly linking the PHP variables to the corresponding form data. Note also how rolling over the variable **$a** shows its value, i.e., Oasis.

This approach is informed by findings from the research above, which suggests that visualisations should be annotated with accompanying text and that unnecessary information should be hidden until needed (Moons & Backer, 2013), for example the variable $a is accompanied by its value (the user's chosen artist) but only when you roll over the variable.

EPHP also helps students understand the workings of a typical “while” loop to iterate through database results, which in our experience is a common source of difficulty. Figure 4 illustrates this:

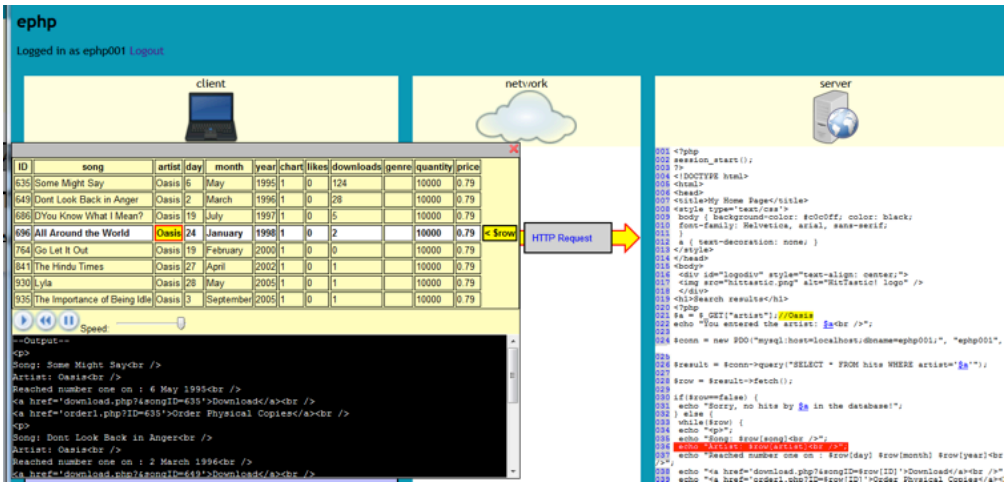

Figure 4. Looping through database results.

The simulation shows the "while" loop being executed multiple times (one for each row from the database results); each line is stepped over and highlighted. At the same time, the window on the left shows the entire set of database results, highlighting the current row being processed. Thus students can see the relationship between the code and the actual results. Note also how the artist (Oasis) is emphasised within the current row, indicating that the current line of PHP being processed (the one highlighted in red) is specifically the line that displays the artist from the current row. Finally, note the black output window below the database results, indicating the HTML being generated from the PHP code within the loop, providing further insight to students. EPHP allows students to have a degree of control; for instance “video player” style controls (Play, Pause, Fast-forward, Rewind) have been added so that students can pause, resume and repeat animations, which Moons & Backer

(2013) suggested leads to "a better understanding of the causal relations between the code and the actions caused by the code" (p. 373).

Deployment of EPHP to Students

During the academic year 2016/17, EPHP was deployed as a learning tool within our second year unit "Developing for the Internet." Students used EPHP for six of the first seven lab activities, covering HTTP requests and responses, basic PHP, PHP and databases, and passing information between pages in multi-page PHP applications. These topics were done over the course of four weeks, with typically two topics per week. Later topics (session variables, AJAX and prepared statements) did not use EPHP, as the tool cannot handle those technologies at present.

During the course of the laboratory sessions, we qualitatively observed students' interaction with the tool, and immediately following the four-week period obtained feedback from students via an online survey with 14 questions (11 multiple-choice and three freeform text).

Results

A total of 126 students registered for a login to the EPHP server, but nine students had no code uploaded, and furthermore the timestamp of their web directory suggested that it had not been modified since creation. This suggests that 117 students actually used it.

Of these 117, 36 responded to the survey, a 31% response rate. Most of the students in the sample rated themselves as beginner or intermediate programmers: only 10% rated themselves as "strong" or "advanced." Thus, the sample is representative of EPHP's target group, as it is expected from our own experience that stronger programmers should be able to understand web development without the need for a visualisation tool.

The results suggest that, amongst the group that responded, EPHP has generally been perceived as useful. In particular:

- 83% of students rated EPHP's HTTP request and response animation "useful" or "very useful" to visualise how information is sent between client and server, with 42% rating it "very useful."
- 78% rated EPHP "useful" or "very useful" in helping them understand the relationship between user input in HTML form fields and PHP variables containing that input, with 33% rating it "very useful."
- 77% rated EPHP's database loop animation "useful" or "very useful" to understand the PHP code to display the database results, though only 17% rated it "very useful," with one student not answering.
- 85% of students answered "totally agree" or "agree" to the statement "the visualisation feature of EPHP helps me understand the communication between client and server in a PHP application."

- 71% of students answered “totally agree” or “agree” to the statement “EPHP has helped me learn the basics of PHP,” with one student not answering.
- 81% of students answered “totally agree” or “agree” to the statement “Overall, do you agree that EPHP has been beneficial to your learning in this unit?”

Two of the questions elicited slightly less favourable responses:

- 58% of students found EPHP’s animation of the database results “useful” or “very useful” in helping them understand how to pass database data through to another script, though another 31% answered “no strong opinion” to this question.
- 60% of students answered “totally agree” or “agree” to the statement “EPHP has made learning the basics of PHP more enjoyable,” though a further 23% answered “neither agree nor disagree” to this question.

The responses to the questions above appear to indicate that the visualisation of communication between client and server, in which the HTTP request and response are illustrated by arrows moving from the client window to the server window and back, were particularly beneficial, with 80% positive responses in both questions relating to this topic.

Also seen as useful was EPHP’s ability to visualise the relationship between form fields and the PHP variables used to read their contents, with a 78% positive response to this question. Furthermore, when asked “what features of EPHP are most useful to you?” (with students able to select more than one feature), this feature was rated most favourably at 72%, with the HTTP communication animation coming second at 61%.

Visualisation of the database results, arguably EPHP’s most advanced feature, was rated somewhat less favourably. While 77% of students rated EPHP’s database loop animation useful or very useful to understand the corresponding PHP code, only 42% selected it in the “what features of EPHP are most useful to you?” question – suggesting that it was often seen as secondary to the two features above.

The two features seen as least useful, according to this question, were the integrated code editor and FTP upload features, with just 36% of students selecting each – features not critical to EPHP's role as a visualisation tool but included in order to make EPHP an “all-in-one” beginner development environment.
Three questions (11-13) invited students to contribute more specific, written-answer feedback, with Question 11 asking students to report bugs, Question 12 inviting comments on usability or user-friendliness, and Question 13 asking for suggestions for additional features or improvements. Some interesting responses come from these questions, which partly align with the multiple-choice answers given above. The following themes commonly occurred.

Make EPHP support other browsers. Due to some of the advanced JavaScript techniques used (HTML5 drag-and-drop being a particular issue), EPHP only works fully on the Firefox browser, with many respondents requesting Chrome support in particular. Four respondents mentioned this in Question 11, one in Question 12 and one in Question 13 – six independent respondents in all.

Apply the CSS to the rendering of the web pages in the "browser within a browser" in the client window. At present, only the HTML of requested web resources is rendered, with the CSS not applied. One student mentioned this in Question 11, one in Question 12 and two in Question 13 – four independent respondents in all.

Include functionality to delete files on the server. Six students mentioned this in Question 13.

Give an option to turn off the animation. One student mentioned this in both Question 11 and Question 12, and a further two students in Question 12 only – three students in all.

Allow the size of the coding window to be adjusted. Two students mentioned this in Question 13.

EPHP needs to be more flexible in how it interprets the code. Two students reported issues with EPHP when it was processing PHP code written differently to the style introduced in the lectures.

Discussion

The generally positive feedback obtained from the pilot study presented above gives encouragement that the early prototype of the tool presented so far shows promise, and validates its use in the unit as an aid to learning web development.

A number of issues have been raised, both by the students' responses and from our own observations. Firstly, the functionality to visualise the database results is perhaps not seen as quite as useful as EPHP's other features. This is in one way surprising as, from our own observations, students have previously struggled with the meaning of the PHP code to show database results, and EPHP aims to make this clearer through visualisation. On the other hand, this is the least mature and least tested feature in EPHP and had some inherent problems. It was desired to show the database results on the "server" side of EPHP – as database results are obtained on the server – but it was difficult to place the window here without obstructing the PHP code. Midway through the four-week period, we repositioned this window to the client side (as shown on Figure 4), which is conceptually incorrect but does allow students to see the code and the results concurrently. Thus it is quite conceivable that students experienced usability problems with this feature. We propose to seek more detailed feedback from students, via a focus group.

One observation we made ourselves was that students struggled to enter code in the "Develop" tab within the "Client" window of EPHP due to lack of

space. The integrated editor was too narrow to clearly see their code; the "Network" and "Server" windows occupied too much space. Furthermore, many students utilised a mode of working in which they had two browser windows open simultaneously side-by-side, one containing EPHP, and the other containing the lecture notes or the worksheet. This further reduced the amount of space available for students to enter code in the editor. In the end, most students resorted to writing the code in a separate editor (such as Notepad++), and copied and pasted it into the EPHP develop window before uploading. This observation is supported by the feedback for Question 13, with two students mentioning this issue. Thus it is proposed to allow a resizable "client" window, to allow it to occupy most of the screen while code is being developed.

A common theme of the feedback was that EPHP did not apply CSS to the web pages rendered in the "browser within a browser" in the client window; thus, this is on the priority list for new features. However, there is a potential conceptual problem here. EPHP aims to show users what is actually happening when a browser requests a resource from a server. CSS files are typically linked to HTML pages in this way:

```
<link rel="stylesheet" type="text/css" href="style.css" />
```

When the browser encounters the above tag, it sends another HTTP request to the server to receive the specified CSS file (style.css here). If EPHP is to apply all external stylesheets, to be consistent with its stated aim of showing users what actually goes on, it really needs to show another HTTP request animation to retrieve the stylesheet. This could potentially make the animation sequence too long and frustrate the user; is it worth doing this simply to apply the CSS? This is an interesting question and a potential further subject for a focus group.

A possible solution to this problem is to give the user more control to turn animations on and off. Three students requested the ability to turn off the animation, three students requesting this feature; with one student specifically volunteering the information that she/he found it frustrating once the concepts had been mastered.

It is of note that a few students have attempted to use EPHP during later topics within the unit, such as session handling and AJAX, when not specifically instructed to - indeed, when advised not to. EPHP is at present unable to handle these technologies so cannot be used in this context, but the fact that students have attempted to use it appears to indicate that these students are happy with it. In addition, a significant number of students have continued to use it as an upload tool (in preference to a dedicated FTP client recommended by ourselves), again suggesting they are comfortable with this aspect of it. This does contradict the fact that only 36% rated the FTP upload facility as a "most useful" feature – though this may have been due to being perceived as secondary to the visualisation.

There are some lingering concerns, however, with how much some students have learned and whether they have gained a clearer and deeper understanding of server-side web application development. After several weeks using EPHP,

we have observed that some students remain confused between form data and data obtained with a database, and make mistakes with referencing form fields and database columns correctly within their PHP code. This perhaps aligns with Moreno and Joy (2007) who observed that while students using JEliot appeared to give good feedback on the tool, they did not do appreciably better in actual coding problems. Care needs to be taken that EPHP is not merely "wowing" the students, but actually enabling them to achieve deeper learning. Indeed, Ben-Ari et al (2011) suggest that "teachers need to prepare lessons that use JEliot in order to obtain the maximal pedagogical advantages" (p. 382) – suggesting that benefits could be achieved by carefully planning the curriculum to make the most of EPHP, something that did not happen to a great extent this year; EPHP was introduced in a somewhat ad-hoc manner. For instance, adding features such as questions or quizzes, as suggested by Moons and Backer (2013) and Moreno and Joy (2007), is a possible additional step.

So what of the future? Obvious steps to take include incorporating the requests for additional features made by the students. In addition, we plan to overhaul the code "step-through" feature, which is currently coded in an inflexible way, and can only handle code written in a particular style. The principal author presented EPHP at the Hampshire PHP User Group in November 2016 and received useful suggestions from PHP professionals on improving this aspect of the software. At present the PHP Tokenizer API (The PHP Group, 2017) is used to interpret students' PHP code; more flexible and powerful alternatives include PHP-Parser (Popov, 2017) and possibly integration with the xdebug system (Rethans, 2017), which opens up the possibility of adding full debugging functionality.

At present, EPHP works with PHP only; however, its architecture, with distinct client and server components, potentially allows use with other server-side languages (by substituting the server component), thus widening its user base.

Thus, we plan to fully integrate an enhanced and improved version of EPHP into the "Developing for the Internet" unit in early 2018. The tool also has potential for the follow-on third-year unit "Web Application Development" which focuses on web services and their clients, a frequent source of confusion for students. Assuming appropriate work has been done, it is possible EPHP could be introduced for this unit this coming autumn. We also plan to conduct follow-on analyses of EPHP and its effects on learning in the coming academic year.

Conclusion

We have presented EPHP, our prototype PHP learning tool, and conducted a pilot survey that has produced generally positive feedback and indicated that the tool has promise. A number of concerns and issues have been identified through student feedback and our own observations, and we aim to address these with the next version of the software.

Acknowledgements

The authors would like to thank colleagues Prins Butt and Darren Cunningham for input into the design of the software and for discussion of ideas, and would also like to thank attendees of the PHP Hampshire user group for suggestions for improvement on the technical aspects of the software.

References

Alston. P., Walsh, D., & Westhead, G. (2015). Uncovering "threshold concepts" in web development: An instructor perspective. *ACM Transactions on Computing Education, 15*(1).

The Apache Software Foundation. (2017). *Apache HTTP Server Project* (Version 2.4.25) [Software]. Available from https://httpd.apache.org/

Ben-Ari, M., Bednarik, R., Ben-Bassat Levy, R., Ebel, G., Moreno, A., Myller, N., & Sutinen, E. (2011). A decade of research and development on program animation: The JEliot experience. *Journal of Visual Languages and Computing, 22*(5), 375–384.

Ben-Bassat Levy, R., Ben-Ari, M., & Uronen, P. A. (2003). The JEliot 2000 program animation system. *Computers & Education*, *40*, 1-15.

Biggs, J., & Collis, K. (1982). *Evaluating the quality of learning: The SOLO taxonomy.* New York, NY: Academic Press.

Cloud9 (2017). *Ace: The high-performance code editor for the Web* [Software]. Available from https://ace.c9.io

Codecademy. (2017). *Learn to code interactively, for free*. Retrieved from https://www.codecademy.com

JetBrains. (2017). *PHPStorm: Lightning-smart PHP IDE* (Version 2017.1) [Software]. Available from https://www.jetbrains.com/phpstorm/

Kaila, E., Rajala, T., Laakso, M-J., & Salakoski, T. (2009). Effects, experiences and feedback from studies of a program visualization tool. *Informatics in Education*. *8*(1), 17-34.

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., … Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, *36,* 119-150.

McCracken, M., Almstrum, V., Diaz, D., Gudzial, M., Hagan, D., Kolikant, Y., … Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, *33*(4), 125–140.

Microsoft. (2009). *INFO: ASP.NET code-behind model overview.* Retrieved from https://support.microsoft.com/en-gb/help/303247/info-asp.net-code-behind-model-overview

Moons, J., & Backer, C.D. (2013). The design and pilot evaluation of an interactive learning environment for introductory programming influenced by cognitive load theory and constructivism. *Computers & Education*, *60*, 368–384.

Moreno, A., & Joy, M. S. (2007). JEliot 3 in a demanding educational setting. *Electronic Notes in Theoretical Computer Science*, *178*, 51–59.

Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., …Velázquez-Iturbide, J. Á. (2002). Exploring the role of visualization and engagement in computer science education. *Working Group Reports*

from ITiCSE on Innovation and Technology in Computer Science Education, *35*(2), 131–152.

Online Education Ltd. (2016). *CodeAvengers: A better way to learn to code websites, apps and games*. Retrieved from https://www.codeavengers.com/

Park, T., & Wiedenbeck, S. (2011*)*. Learning web development: Challenges at an earlier stage of computing education. *Proceedings of the 7th International Workshop on Computing Education Research (ICER'11)*(pp. 125–132). New York, NY: ACM.

The PHP Documentation Group. (2017). *Tokenizer.* In *PHP Manual*. Retrieved from http://php.net/manual/en/book.tokenizer.php

Popov, N. (2017). *PHP-Parser* (Version 4.0) [Software]. Available from https://github.com/nikic/PHP-Parser

Radošević, D., Orehovački, T., & Lovrenčić, A. (2009). Verificator: Educational tool for learning programming. *Informatics in Education*, *8*(2), 261–280.

Rethans, D. (2017). *XDebug extension for PHP* (Version 2.5.4) [Software]. Available from https://xdebug.org

Zhang, Y., & Dang, Y. (2015). Investigating essential factors on students' perceived accomplishment and enjoyment and intention to learn in Web development. *ACM Transactions on Computing Education, 15*(1).

Author Details

Nick Whitelegg
nick.whitelegg@solent.ac.uk

Olufemi Isiaq
femi.isiaq@solent.ac.uk