TagStyle: Programming with a Purpose for Front-End Web Development

Alexander Hollenbeck[1], **Benjamin Rothman**[1], **Philip House**[1], **Sarah Lim**[1], **Haoqi Zhang**[2]
[1]Northwestern University
Evanston, IL, USA
{alexhollenbeck, benjaminrothman2014, philiphouse2015, slim}
@u.northwestern.edu

[2]Northwestern University
Evanston, IL, USA
hq@northwestern.edu

ABSTRACT
Programming students invest time into learning exercises to develop their skills, but the results often go unused. Meanwhile, many of the open issues in professional software projects could be solved by students, if not for the high barriers to entry. We introduce Programming with a Purpose, a framework for crowdsourcing software development tasks while providing students with authentic learning. We present TagStyle, a web application that applies Programming with a Purpose to front-end web development. Using TagStyle, students decompose the problem and write solution code. We conducted a user study evaluating learner output of TagStyle. Results show that incorporating student output into real projects is feasible. We analyze common errors and show how TagStyle informs future applications of Programming with a Purpose.

Author Keywords
Crowdsourcing; social computing

ACM Classification Keywords
H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous

INTRODUCTION
Novice programmers represent a fast-growing, eager, and untapped source of talent. In addition to skyrocketing enrollment in undergraduate computer science programs [24], a plethora of online systems for learning programming has flourished as aspiring coders spend their free time practicing and acquiring new skills [22]. Yet for all the effort students expend on these courses, very little of the code they write during the learning process will ever be used in real projects or applications. At the same time, the need for new talent in both the professional and open-source world is growing dire [20].

Released in the DTR Tech Report 2015.

The problem is that there is no easy way for a novice programmer to contribute useful code to a project without first proving him- or herself as an expert [19]. In reality, many coding tasks involved in a professional or open-source software project are relatively routine and could feasibly be done by programming students [12]. Students are eager to work on issues from real projects, as authentic practice increases engagement [21] and employers of computer science graduates frequently judge applicants based on their project portfolio [4]. Having students tackle the simpler problems in software development projects would have the double benefit of giving students real-world project experience while helping to meet the great need for programmers by routing simpler tasks to unpaid students.

To make effective use of learner efforts, we introduce *Programming with a Purpose*, or PWAP, as a framework for decomposing complex software development tasks into easy subtasks, serving those tasks to learners of an appropriate skill level, and combining the learners' output into useful code that is delivered back to the client. In this paper we present *TagStyle*, a prototype which applies PWAP to front-end web development.

TagStyle is a web application which allows a client to submit a mockup image for a new website. It returns a style guide for that website in the form of Cascading Style Sheets (CSS) that can be applied on top of Twitter Bootstrap. TagStyle breaks the problem down into two types of tasks that each provide authentic practice for programming students with a basic knowledge of HTML class structure and CSS. In the first stage, learners tag the Bootstrap elements in the mockup. The output from the first stage is routed to the second stage, where learners write CSS to style default Bootstrap elements to look like the ones identified in the mockup.

We conducted a user study to assess whether learners could use TagStyle to 1) successfully tag a mockup with the proper Bootstrap classes, and 2) successfully write CSS to style elements according to the desired appearance. We

found that in both stages, output quality varied significantly by learner and by the type of element being tagged or styled. Enough high-quality output was generated in both stages to support the hypothesis that learners are capable of producing useful code. However, mechanisms are needed to validate and combine learner output before a fully finished Bootstrap style guide can be returned to the client.

This paper makes the following contributions:

1. We introduce Programming with a Purpose (PWAP), a framework for crowdsourcing software development projects using novice programmers.
2. We present TagStyle, a first prototype of PWAP which applies the framework to the development of front-end web style guides built on top of Twitter Bootstrap.
3. We evaluate TagStyle through a user study of programming students and present our findings as a guide for future applications of PWAP.

The paper proceeds as follows. We first describe related work in crowdsourcing interfaces and teaching applications that use crowdsourcing to accomplish useful work. Next, we enumerate the design challenges posed by PWAP and how they informed the design of TagStyle. We describe the methodology of our user study and report our findings in the context of our hypotheses. Finally, we discuss the limitations of TagStyle and its implications on future PWAP applications.

RELATED WORK

Our conceptual contribution falls under the umbrella of *learning with a purpose*, a broad term we use to describe systems that help learners acquire skills while harnessing their efforts for useful work. For example, *Duolingo* sources language translations from learners to provide free language education [14]. *LevelUp* is a platform for helping Adobe Photoshop learners gain new skills through interactive tutorials while applying their knowledge to improve real-world images submitted by requesters. More recent work in *learnersourcing* provides opportunities for learners to enhance lecture and how-to videos while watching [6].

There are two main conceptual differences between PWAP and these previous works. First, PWAP consolidates the means and the ends of the learning process, as learners develop industry-specific skills through actual industry tasks. By contrast, in Duolingo, translation provides an effective means for learning, but does not necessarily correspond with learning ends insofar as most learners are not aspiring to become professional translators. Second, high-level problems in the programming domain are more complex than those in other domains, and consequently more difficult to decompose into constituent subtasks. For the purposes of learning and contributing useful work, the process of learning how to decompose and structure code is core to PWAP, but largely absent in these other systems.

In thinking about structuring and decomposing tasks, our crowd work process builds upon previous techniques designed for complex problem-solving in paid crowdsourcing systems. Like previous systems such as CrowdForge, Soylent, and PlateMate, TagStyle engages learners in decomposition by tagging a mockup into components [8, 1, 11]. As tagging decisions apply not only locally but also globally (e.g., tagging a button as *btn-primary* means that other similar buttons should also be tagged as such), having global context and views of related components is useful for consistency. Here TagStyle draws inspiration from *crowdware* interfaces for handling human computation tasks with global constraints by providing a single shared workspace and actionable feedback to direct users toward unsolved problems [23].

Recent work on CrowdCode seeks to crowdsource software development by engaging crowd workers in identifying and implementing areas of a program where new code is needed [9]. Their technical work focuses on collaboration and coordination issues for resolving problems such as spillover from higher-order program changes. Similar issues can occur in PWAP, wherein structural changes may require necessary revisions to the document model and styling. Our work takes on a different focus by considering the challenges in involving learners in programming tasks, and studying the challenges and opportunities in having learners produce useful work.

Online learning platforms such as Codecademy and Khan Academy are rapidly expanding the number of people exposed to programming. These platforms focus primarily on teaching basic syntax to help students start on personal projects, but do not provide avenues for authentic practice critical for one's continued deep understanding and development toward being a professional. Practice is limited to a small set of curated "toy" examples, most of which do not sufficiently resemble work in a professional software development setting. On Codecademy alone, over 24 million users have completed 100 million "toy" exercises [3, 5]. This collective learning effort is a valuable potential resource for contributing useful work while learning, which we seek to utilize through tools and methods for supporting PWAP.

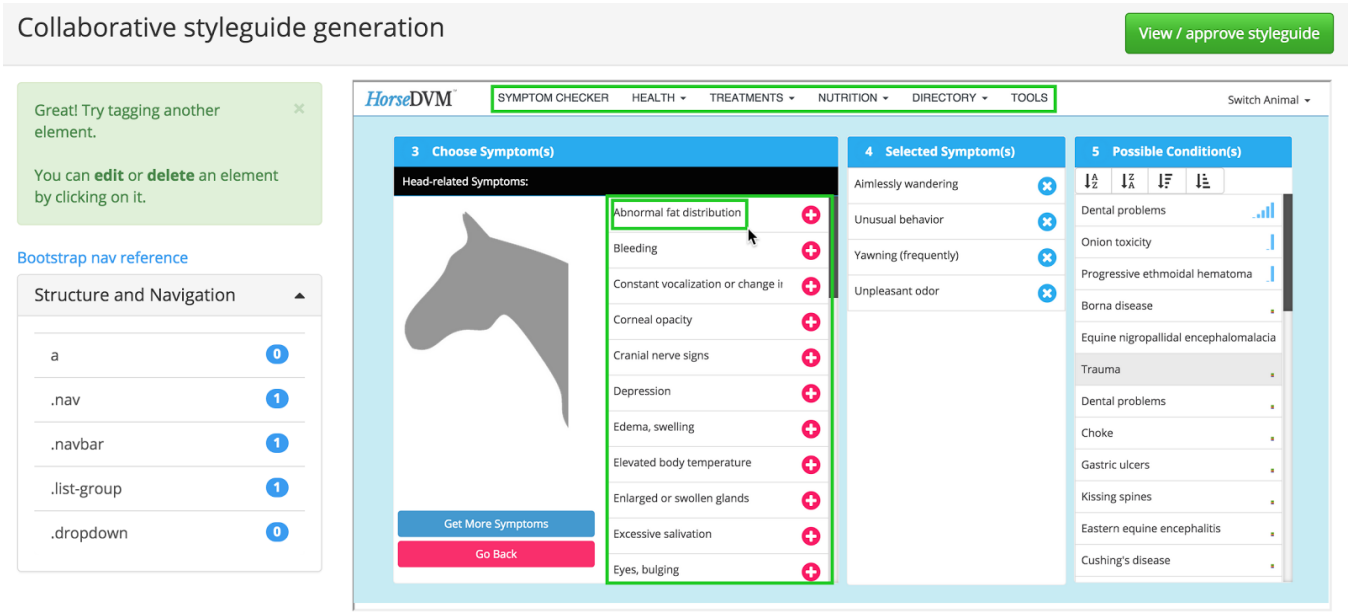

Figure 1. Tagging interface. This learner is in the process of identifying structure and navigation elements. Green rectangles indicate the areas they have identified.

DOMAIN CONSIDERATIONS

Learner considerations

We conducted user observation sessions with 12 undergraduate computer science majors with varying degrees of experience to determine an appropriate domain for learner tasks. Participants worked on isolated tasks in different areas of software development, including algorithm problem-solving, unit testing, code refactoring, and front-end styling. We interviewed participants regarding the ease and perceived educational value of completing each of the various tasks. Front-end styling tasks were manageable for users and conferred the most educational benefit for three reasons:

1. Tasks were relatively small and quick to complete.
2. Tasks had well-defined goal and clear measures of progress.
3. Tasks constituted authentic examples of applied CSS skills that learners cared to gain.

Based on these observations, we chose front-end development as the task domain for TagStyle. Other areas of software development could potentially yield tasks that meet the criteria identified above and function well in a PWAP application.

Client considerations

Besides meeting learner needs, tasks must yield output that is useful to clients. PWAP output code should be easy for clients to verify and incorporate. Output should require minimal refactoring, which is both costly and time consuming (ADD REAL AGILE SOURCE).

We chose to output a front-end style guide to the client. A style guide consists of CSS styles for element classes, to be reused across a website. Since most web elements are styled by class rather than individually (source: web style guide book), a style guide is more useful than isolated CSS rules for each element. For TagStyle we chose to output a style guide built on Twitter Bootstrap, a publicly available set of classes and styles commonly used as a starting point for a website's front end. A Bootstrap style guide suits clients for four reasons:

1. A Bootstrap style guide can be easily added to a project by simply appending the new stylesheets to existing ones.
2. Bootstrap's use of namespaced CSS classes cuts down on the burden of code refactoring.
3. The accuracy of the output is easily verified through a visual comparison of the desired style guide and the produced web page.
4. The client can easily indicate the desired output by submitting a static design (mockup) of the website they are developing.

Beyond front-end styling, any domain for a PWAP application should allow for output that can be easily verified by the client and quickly applied to a project.

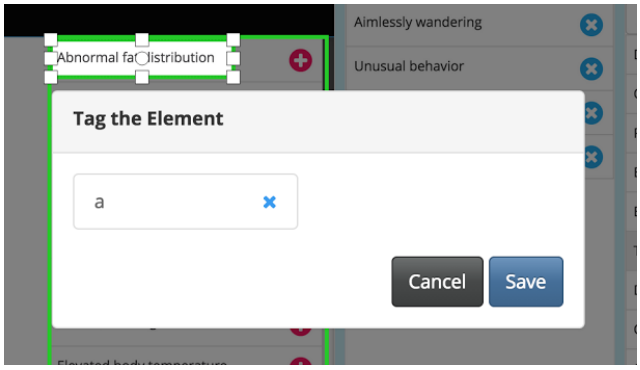

Figure 2. Assigning classes to identified elements

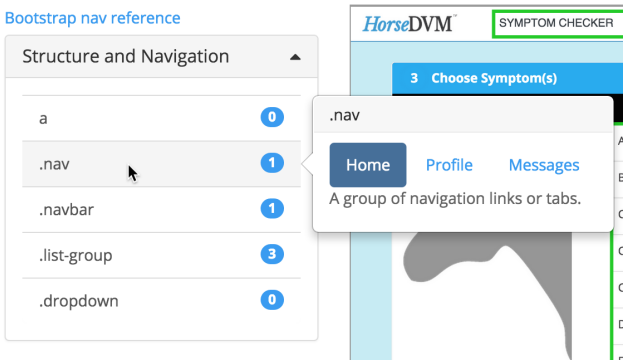

Figure 3. Affordances on the list of classes

SYSTEM DESCRIPTION

TagStyle, our current PWAP prototype, seeks to convert a client-provided mockup into a ready-to-use Bootstrap style guide. This problem can be broken down into two categories of learner tasks:

1. *Tagging*: identifying the Bootstrap elements in the mockup that need to be styled.
2. *Styling*: writing the CSS rules for each element to apply on top of the default Bootstrap styles.

These tasks meet learners' requirement for small, modular tasks. Bootstrap's pre-defined set of class definitions makes it easy to identify classes of elements in the mockup, and writing CSS for an individual class typically requires only a few lines of code. Furthermore, the learner can easily understand the goal and verify the completeness of both types of tasks through visual feedback. We describe each type of task in further detail.

Tagging

In the tagging stage, learners are presented with a mockup of a web page and are tasked with 1) identifying Bootstrap elements in the mockup and 2) tagging them with an appropriate Bootstrap class. These tags are used to splice the mockup image into smaller slices, each capturing an instance of a particular stylized Bootstrap component. By compiling images tagged with the same class, the system

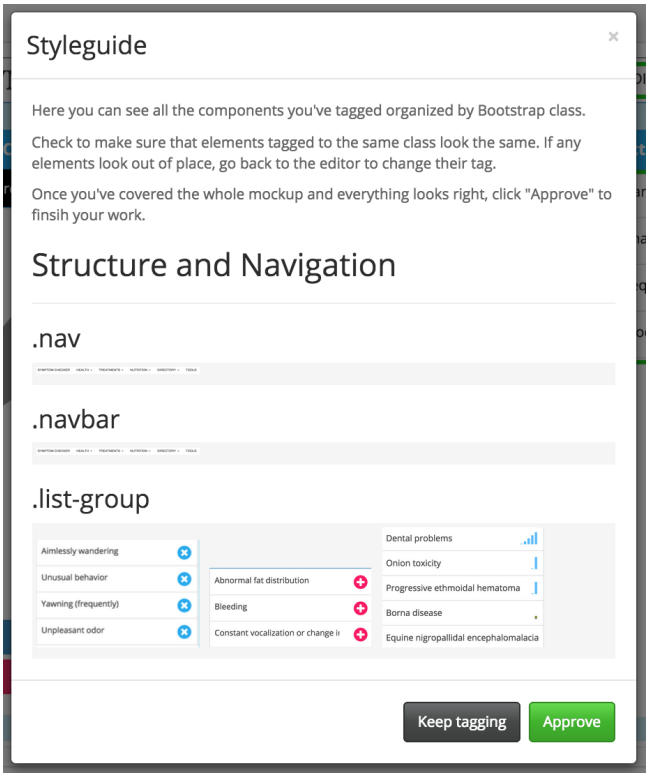

Figure 4. Style guide modal

generates a visual set of examples for each individual Bootstrap component.

A learner draws rectangles around instances of Bootstrap elements found in the mockup (Figure 1). Each learner is tasked with finding all instances of one particular category of element: buttons, typography, or structure and navigation.

After identifying an element as belonging to the target category, a learner must assign a specific Bootstrap class from a list of options (Figure 2). Only those classes which fall under the target category are available; for example, if the learner is identifying structure and navigation elements, the available classes are *a* (link), *.nav*, *.navbar*, *.list-group*, and *.dropdown*.

Learners receive a number of affordances to aid them in correctly tagging the mockup. Prompts appear in the upper left to guide the learner through the process. The learner may move, resize, delete, or change the tags on any identified element. The list of options supports three affordances: a link to Bootstrap documentation for the learner's reference, a visual example of each class that displays when the learner hovers over the class, and badges showing the number of elements the learner has tagged for each class so far (Figure 3).

After tagging several elements, the learner is prompted to use the "View / approve style guide" button in the upper right to check their work and gauge completeness. Upon clicking the button, a modal appears showing the visual

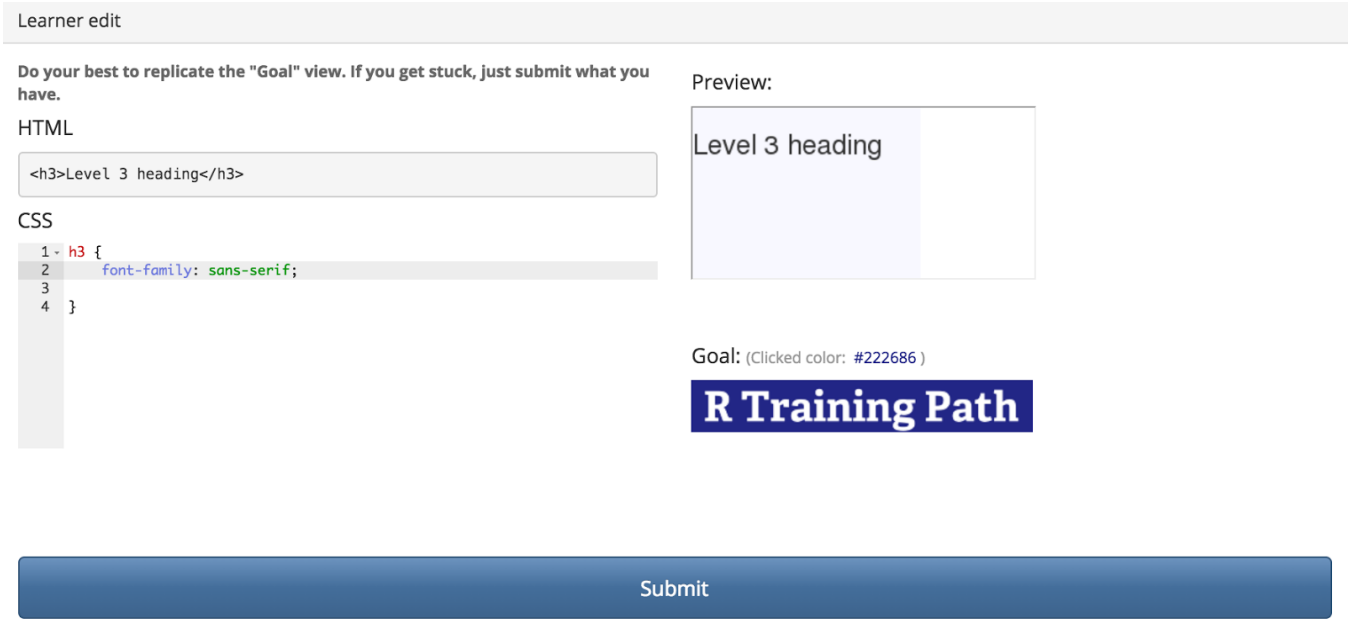

Figure 5. Styling interface. This learner is in the process of styling a level 3 heading.

style guide in progress, allowing the learner to visually verify that elements they have tagged as the same class are similar (Figure 4). The learner may either click "Keep tagging" if they wish to continue tagging, or "Approve" if they are satisfied with their work. Clicking "Approve" concludes the task.

The output of the tagging stage is a set of images spliced from the mockup, each tagged with the Bootstrap class that should be styled to produce that image. The collection of outputs from learners working on all categories of element forms a visual set of examples for the desired style guide. Each group of images corresponding to one individual Bootstrap class constitutes an input for the styling stage.

The tagging interface is designed to function as a shared workspace according to the crowdware paradigm. However, in this prototype our goal was to understand how individual learners approach the tagging tasks and assess the quality of their work. Therefore, each learner completes tagging tasks in isolation.

Styling

In the styling stage, a learner writes the CSS to style the elements identified in the tagging stage, creating an actual style sheet that can be applied on top of Bootstrap. An individual styling task consists of a set of goal images tagged as the same class in the same mockup by learners in the tagging stage. The resulting CSS can be applied on top of Bootstrap to create the style guide for the particular class that the learner is styling.

In this stage the learner writes CSS rules beneath the selector in the "CSS" field (Figure 5). The "HTML" field shows HTML that is automatically generated based on the class the learner is working on, and may not be edited. The "Preview" field shows the result when the CSS is applied to the HTML and rendered in a browser, and updates live as the learner types to show progress. The "Goal" field shows the mockup image or images produced in the tagging stage. The learner is tasked with writing CSS such that Preview matches Goal as closely as possible.

As with tagging, a number of affordances aid the learner in their task. Text prompts guide the learner through the task. The CSS field shows any syntax errors in the learner's CSS. Clicking any part of the Goal image displays the hex code for the color directly beneath the mouse, enabling the learner to precisely match the desired colors in the mockup.

Once the learner has matched Preview to Goal as closely as possible, they may click the "Submit" button to submit their work and complete the task. A single learner's output provides the CSS styling for one specific Bootstrap class in the style guide. The collection of outputs from all learners working on styling tasks for each class found in the mockup forms a complete Bootstrap style guide that can be applied to the web page.

USER STUDY DESIGN

In our user study, we evaluate whether the PWAP framework can be applied effectively to front-end web development in order to produce useful results for clients. Because viable crowdsourcing models rely upon the production of satisfactory output from individual users, we were primarily interested in assessing the quality of learner output in each stage of the platform. We thus tested two hypotheses:

Figure 6. From left: Mockup 1, Mockup 2, Mockup 3

1. *Tagging stage*: given a certain subset of Bootstrap components to look for, learners with some CSS background should be able to tag a mockup with the appropriate Bootstrap classes.
2. *Styling stage*: given images of custom-styled Bootstrap elements, learners with some CSS background should be able to implement those components by customizing the default CSS.

Methods

We recruited 17 participants from the undergraduate computer science program at our university via an email survey. Participants had different levels of front-end programming experience, but all had at least minimal CSS experience. Participants performed 3 tagging tasks and 3 styling tasks then completed an exit interview. Participation lasted 40 to 60 minutes and participants were paid $20.

We chose to provide the input for both the tagging and styling stages so as to prevent variability in input from affecting our measurement of learners' ability to complete each task. For the tagging task, we supplied input in the form of three mockups taken from Bootstrap's Expo, a showcase of live sites that are built entirely on Bootstrap with custom style guides applied. We chose sites that used a variety of elements in each of the three element categories (Figure 6). For the styling task input, we examined the source HTML for the sites and tagged the three mockups using the TagStyle interface according to the classes used by the sites' developers. The output of images or groups of images corresponding to particular Bootstrap classes was used as the input to the styling tasks. Participants in the study completed both tagging and styling tasks using these inputs, and their outputs were evaluated.

Tagging stage

In the first part of the study, users were asked to tag all of the components of one of typography, structure, or buttons. There was no previous tagging work done when they started, and they were allowed to finish whenever they thought they had tagged all of the elements that fit that group of components. They repeated this task for the two other mockups with randomly assigned components. Users were allowed to use any online resources to help them figure out what and how to classify.

Styling stage

In the styling stage, our goal was to test whether learners with some CSS knowledge could correctly style default Bootstrap elements to match a provided set of examples. During each task, participants were asked to style a default Bootstrap element to match the appearance of the examples provided. Participants were asked to complete one styling task from each of the three mockups, but were randomly assigned to an element category within each mockup.

RESULTS

17 users participated in at least one stage of the study. Three had very minimal experience with CSS, and ten had a moderate amount, having previously built or modified a few websites. Four had significant or professional experience. Seven of the participants had previously heard of Bootstrap, including one participant with very minimal CSS experience, two with moderate experience, and all four with significant experience.

For each mockup, one tagging task and one styling task were created for each of the three element types. Thus, each stage of the study had 9 total tasks that could be assigned to users. Each task was assigned to between 4 and 6 users who completed the task independently.

Tagging results

All 17 users completed three tagging tasks. Each task used a different mockup and a different category of element, so each user had exposure to all three mockups and all three categories.

Because the mockups were in fact screenshots of real websites built on Bootstrap with custom style guides, we evaluated the users' output by comparing their tags to the actual Bootstrap classes used by the websites' developers to render the page. We examined the source HTML for each page and categorized each element by its class, forming a ground truth to compare against user output.

For each task we counted the number of instances of two conditions:

- *Matched*: the user identified and tagged an element with the exact class that appears in the source HTML for that element.

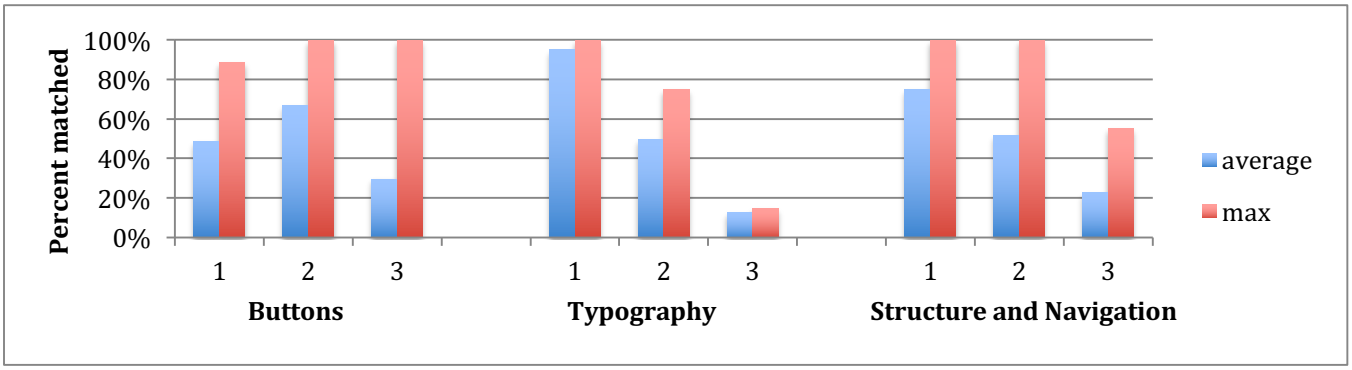

Figure 7a. Average and maximum percent matched in tagging tasks.

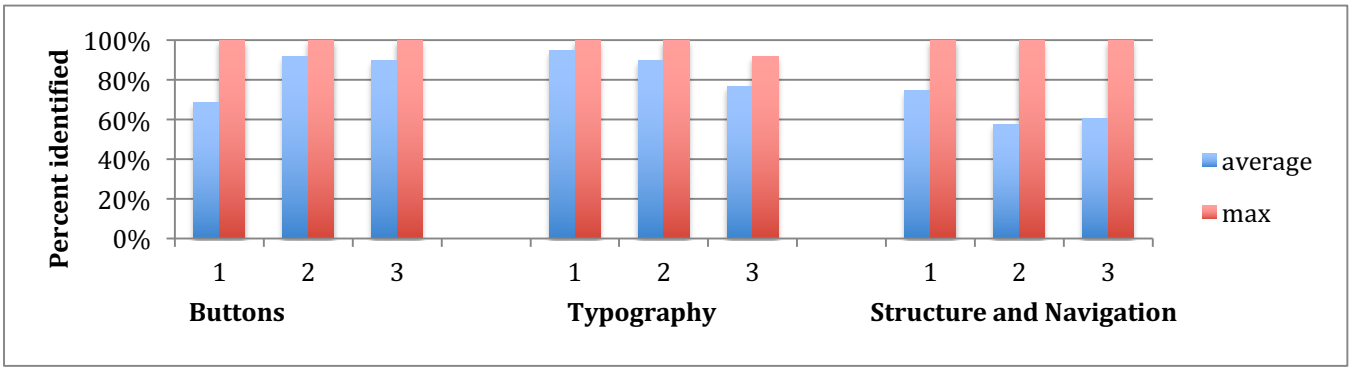

Figure 7b. Average and maximum percent identified in tagging tasks.

- *Identified*: the user identified the element in the proper category but used a different class than the source HTML. For example, if the user tagged an element as *btn-primary* but the class in the source HTML is *btn-default*, that element is counted as identified.

We calculated percent matched and identified by dividing the number of instances of each by the total number of elements of that class present in the source HTML. The matched percentage indicates how closely the user matched the original developer's class names for the elements in the mockup. The identified percentage indicates how many elements the user correctly identified as a particular type (buttons, typography, or structure and navigation). The average score and best score for each task are shown in Figures 7a-b.

On average, users scored 40% for matching and 72% for identification, though individual scores varied widely by user, element type, and mockup. The overall incidence of matched elements was low: on average, 37% of buttons, 33% of typography, and 50% of structure and navigation for a total average of 40%. The incidence of identified elements was higher: on average, 83% of buttons, 68% of typography, and 64% of structure and navigation elements for a total average of 72%. Percent matching ranged from 0%-100%, and percent identified ranged from 20%-100%.

Analysis of the top performing users for each task shows that for the majority of tasks, high quality output was produced. Of the 9 tasks given to users in this stage of the study, 4 had at least one user whose tags perfectly matched the class names in the mockup's source HTML, and all but 2 tasks had a user score at least 75%. All but one task had a user who perfectly identified all instances of their type of element in the mockup. Between 4 and 6 users worked on each task.

Styling results

Of the 17 participants from the tagging stage, 15 completed the styling stage. Tasks fell into three different categories based on the element being modified: button, typography, and structure. Each of the three mockups included one task from each category. With the exception of one participant, who completed tasks from Mockups 1 and 3 only, each participant completed one task from each mockup. Element categories were randomly assigned, and participants collectively completed 15 button tasks, 15 typography tasks, and 14 structure tasks.

We evaluated user performance using qualitative assessments of visual output in comparison to the original style guides. Elements were scored for accuracy in context, color, and typography; Table 1 details the specific criteria used. While these scores do not comprise standalone metrics of visual accuracy, they do provide useful heuristics for output comparison across tasks and user groups.

Table 1. Criteria used to evaluate styling output. The reference images served as the ground truth for comparison.

Variable	Comparison Criteria
Isolation	Element correctly isolated from surroundings/background in reference images
Padding	Ratio between content size and padding approximately matches reference
Foreground Color	Foreground/content color matches reference
Background Color	Background/element color matches reference
Font Family	Same CSS generic font family as reference text
Font Appearance	Text appearance matches reference
Extra Criteria	Border or border radius applied, if applicable

Across the three tasks, the average score was 0.759, suggesting that individual students with some CSS knowledge can implement custom Bootstrap elements according to provided examples. Figure 7 displays the average user performance, according to mockup and task condition.

Users performed noticeably better on button versus typography tasks. Users averaged the highest scores on button tasks (0.785), followed by structure (0.726) and typography (0.659). The marked discrepancy in performance on button and typography tasks is independent of participants' CSS backgrounds. In an ordinal ranking of users according to average composite score, Structure tasks were distributed evenly across participant performance ranges. However, of the 21 tasks completed by the seven above-median scorers, button and typography tasks were represented at frequencies of 11 and 4 respectively. Conversely, of all tasks completed by below-median users, 10 out of 20 were typography, with only 3 button tasks represented.

Several participants had trouble determining the boundaries for an individual element. Due to the bounding box technique used to tag mockups, style guide images typically displayed the goal element in context, i.e. against the page background or with borders intact. Several users interpreted these artifacts as parts of the goal element, writing CSS rules to implement backgrounds as element borders, for example (see Figure 9). These observations suggest the importance of clearly demarcating element boundaries when producing and compiling style guide images.

User feedback

User interviews revealed a consensus that TagStyle's authentic practice was an improvement over existing learning platforms. Even users with significant CSS experience felt the system helped them "learn and practice well." In particular, users liked the "tight feedback loops," "preview button," and "live-updating" editor in the styling interface, which felt "similar to Codeacademy" but allowed for "more freedom and creativity."

Users stressed the importance of having multiple, clear visual examples for each task. During the tagging stage, those unfamiliar with Bootstrap naming conventions were "unsure which tag to use," particularly when "class naming was different from [their] personal style." However, those who consulted the included Bootstrap reference felt the "pop-up examples helped a lot" and "clear[ed] up mental models of classes." Another user expressed the importance of providing sufficient examples during the styling stage:

> I realized that the button shouldn't be fixed width but did it anyway to match the preview. On the second set of buttons, I didn't code a fixed width because the two preview images were different widths, so obviously there shouldn't be a fixed width. If this were actually being used to generate stylesheets, having multiple examples is necessary to avoid overfitting the preview.

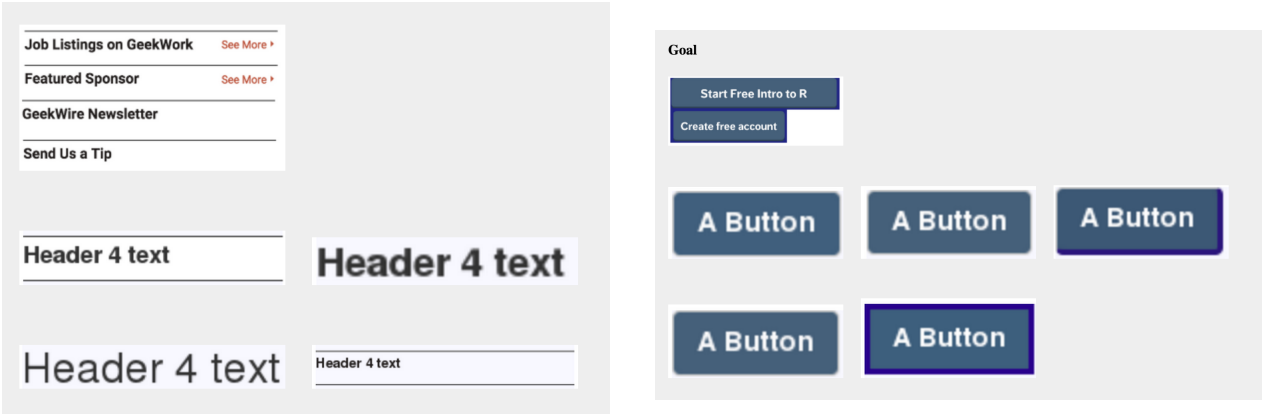

Figure 8. Users had difficulty isolating the goal element from neighboring context.

During the tagging stage, users had trouble identifying elements when necessary contextual information was missing from the mockup. For example, users had trouble differentiating between static text and links, since the static mockup image did not allow for interaction. On a more granular level, many users correctly identified buttons as such, but didn't know which subclasses (e.g. *btn-default*, *btn-main*) to apply. Others found certain stylistic attributes somewhat arbitrary: "*btn-sm* and *btn-xs* were ambiguous."

Participants found the styling tasks to be manageable in size and scope. One user with a professional web development background praised the tasks for being "small and easy," and several users with moderate experience found the subdivision of styling tasks helpful. These observations indicate that our system design successfully divided the code generation process into concrete and close-ended subtasks.

Authentic practice was most beneficial for users with moderate prior CSS experience. One such user found the first styling task difficult and "had to look up a lot of things," but noted that "subsequent [tasks] made more sense…[it was] very satisfying to get to the end goal." Meanwhile, the three participants with very minimal CSS experience expressed unanimous frustration and confusion. "It was difficult to tag because you're not sure which tag to use." During the styling task, these users had "no idea where to begin" or "[didn't] know how much to change," with one leaving an element completely unstyled. These findings corroborate our hypothesized niche for the PWAP system, providing a crucial bridge between introductory tutorials such as Codeacademy and Dash General Assembly, and professional web development.

DISCUSSION

While TagStyle is a specialized tool for creating style guides, it addresses a more general problem: crowdsourcing code development in the form of an authentic learning tool. We identify four major takeaways from TagStyle and its accompanying user study that inform future PWAP systems.

Learners are capable of properly decomposing a software development problem.

Despite low scores overall for the tagging portion of the study, almost all tasks had at least one learner with perfect or near-perfect output. This result shows that a relatively small group of learners (4 to 6 per task) is sufficient to produce a correct decomposition provided the incorrect output can be ignored. Future PWAP applications should seek a minimum number of users per decomposition task in order to achieve this result.

Learners with moderate skill in a programming domain can contribute useful code.

12 out of 15 users achieved a composite accuracy score of 0.67 or higher across their three styling tasks. In the context of our scoring rubric, which evaluated output appearance according to criteria roughly corresponding with CSS properties, this score suggests that the majority of users can individually style any given element to at least two-thirds completion. Given that TagStyle had no mechanisms for verifying output quality, these results suggest a strong baseline for output quality that future PWAP applications can build upon.

Context is important for both decomposing and solving tasks.

Results from the tagging stage showed that learners struggled to tag elements whose function and interactivity could not be conveyed by a static mockup. For example, there was a great discrepancy between the percent of buttons identified (83%) and matched (37%) in comparison to the other element types. In many of these cases the element's exact class had no correct answer, since choosing between an info button and a success button, for example, depends in part on what happens when the button is a clicked, as well as what relation the button has to the purpose of the site. Neither piece of information was available to the learner.

Both qualitative results and user feedback from the styling stage demonstrate the importance of providing users with sufficient visual context through example images. When styling elements to match goal images, some users interpreted the bounding box as the exact boundary of the element, causing errors. For example, several users implemented a colored border around a button element, when in reality the color was part of the page's background included in the bounding box. Potential solutions to this problem include providing new users with specific visual examples of this mistake, as well as having users style elements from mockups they previously tagged.

In both stages, users made many errors by misunderstanding the context of their task. Future PWAP applications should provide affordances to frame tasks in their appropriate context.

Learners can use PWAP to get authentic practice.

PWAP offers an effective means for learners to obtain authentic, industry-applicable programming experience. In the TagStyle prototype, the tagging interface allows users to easily visualize the process of breaking down a mockup image into its constituent elements, and the styling interface asks users to translate a mockup into code piece-by-piece. The contribution to a real project is clear to the learner at both stages. Furthermore, PWAP tasks are identical to those required by professional development settings. Whereas the canned exercises on existing platforms can feel contrived, the PWAP learning experience feels authentic and applicable.

LIMITATIONS

The major limitations of TagStyle are a lack of mechanisms for 1) validating the quality of the learner output and 2) combining the output of learner tasks into a form which is useful to the client.

Output quality

While results showed that many learners were able to successfully tag the mockups and write high-quality CSS under certain conditions, enough errors were made in both stages that if the prototype were run end-to-end, the amount of usable code returned would likely be small. Too much error would be introduced if outputs from tagging were sent directly to styling. A mechanism is needed to 1) verify that the inputted mockup is built on Twitter Bootstrap, 2) only permit correct tags to be forwarded on to the styling stage, and 3) verify that the output from the styling stage is correct before returning it to the client.

Combining output

Assuming the quality of output from both stages can be assured, there still needs to be some way to combine the individual stylesheets outputted by learners in the styling tasks into a single stylesheet that the client can apply directly to their website. Simply appending all returned stylesheets together might in theory produce a website that looks like the mockup, but such a product would require extensive refactoring before it could be incorporated into the site's codebase in a maintainable way. A mechanism is needed to combine styling output into a standardized style guide with consistent, maintainable CSS.

FUTURE WORK

Results from the study show that the development of front-end web style guides built on Twitter Bootstrap is a feasible goal for PWAP applications, provided the two barriers discussed in Limitations can be overcome. Future applications with this goal should incorporate mechanisms from other crowdsourcing applications to solve those barriers. We discuss three potential mechanisms in this section.

Iterative refinement

Some crowdsourcing systems use *iterative refinement* where users build upon each other's solutions over time to produce output more thorough or correct than each user could produce individually. PWAP applications could use iterative refinement as an output validation mechanism. As an example, in TagStyle, the output from styling or tagging tasks could be sent to other learners for improvement and error correction. Once no learners identify areas for improvement, the output would be considered correct and returned to the client.

Combination as a learner task

In the partition-map-reduce paradigm, the final reduce stage involves learners combining outputs from the map stage in a way that solves the original problem. PWAP learners could reduce the output from many other learners to solve the original software problem, making combination a learner task. In TagStyle, a learner could collect all output from the styling stage for a particular mockup and synthesize a style guide by aggregating the best CSS for each element and standardizing code practices. This task could be reserved for learners with higher experience or who have completed many other tasks well.

Crowdware

A key feature of crowdware systems is a shared global workspace where learners can see each other's work as they progress toward a solution. PWAP applications could use a crowdware interface to collectively decompose a software problem into proper tasks. TagStyle's tagging interface was designed to function as crowdware. Multiple learners working to tag a single mockup on a shared interface could check each other's tags and produce a task breakdown with greater accuracy and consistency.

REFERENCES

1. Bernstein, M., Little, G., Miller, R., Hartmann, B., Ackerman, M., Karger, D., Crowell, D., Panovich, K. Soylent: a word processor with a crowd inside. In *UIST* (2013), 313-322.
2. Chilton, L., Kim, J., Andre, P., Cordeiro, F., Landay, J., Weld, D., Dow, S., Miller, R., Zhang, H. Frenzy: collaborative data organization for creating conference sessions. In *CHI* (2014), 1255-1264.
3. Colao, J.J. With 24 million students, codecademy is bigger than you thought. *Forbes.com*, 2014.
4. Greenberg, Saul. Embedding a Design Studio Course in a Conventional Computer Science Education. *International Federation for Information Processing*, 289 (2009), 23-41.
5. Kafka, Peter. Codecademy rounds up $10 million for web lessons. AllThingsD, 2012.
6. Kim, J., Nguyen, P., Weir, S., Guo, P., Miller, R., Gajos, K. Crowdsourcing step-by-step information extraction to enhance existing how-to videos. In Proceedings of the 32nd annual ACM conference on Human factors in computing systems (2014), 4017-4026.
7. Kim, J., Zhang, H., Andre, P., Chilton, L., Mackay, W., Beaudoin-Lafon, M., Miller, R., Dow, S. Cobi: a community-informed conference scheduling tool. *UIST* (2013), 173-182.
8. Kittur, A., Smus, B., Khamkar, S., Kraut, R. Crowdforge: crowdsourcing complex work. In *UIST* (2011), 43-52.
9. LaToza, T., Towne, W., Adriano, C., and van der Hoek, A. Microtask Programming: Building Software with a Crowd. In *UIST* (2014), 1-12.
10. Lynch, P., and Horton, S. *Web Style Guide* (3rd. ed.). Yale University Press, 2008.
11. Noronha, J., Hysen, E., Zhang, H., Gajos, K. Platemate: crowdsourcing nutritional analysis from food photographs. In *UIST* (2011), 1-12.
12. Samer, F., and Sproull, L. Coordinating Expertise in Software Development Teams. *Management Science*, 46, 12 (2000), 1554-1568.
13. Turk, D., France, R., Rumpe, B. Limitations of agile software processes. In *Third International Conference*

on Extreme Programming and Flexible Processes in Software Engineering, XP (Alghero 2002), 43-46.

14. Von Ahn, Luis. Duolingo: learn a language for free while helping to translate the web, IUI. In *Proceedings of the 2013 international conference on intelligent user* interfaces (New York 2013), ACM, 1-2.
15. Von Ahn, L., and Dabbish, L. Designing Games with a Purpose. *Communications of the ACM*, 51, 8 (2008), 58-67.
16. Von Ahn, L., Liu, R., and Blum, M. Peekaboom: a game for locating objects in images. In *CHI* (2006), 55-64.
17. Von Ahn, L., Kedia, M., and Blum, M. Verbosity: a game for collecting common-sense facts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York 2006), ACM, 75-78.
18. Von Ahn, L., and Dabbish, L. Labeling images with a computer game. In *CHI* (2004), 319-326.
19. Von Krogh, G., Spaeth, S., Lakhani, K. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32, 7 (2003), 1217-1241.
20. Westervelt, Eric. A Push To Boost Computer Science Learning, Even At An Early Age. *NPR*, 2014.
21. Williamson Shafer, D., and Resnick, M. "Thick" Authenticity: New Media and Authentic Learning. *Journal of Interactive Learning Research*, 10, 2 (1999), 195-215.
22. Wortham, Jenna. A Surge in Learning the Language of the Internet. *The New York Times* (March 27, 2012), 2012.
23. Zhang, H., Law, E., Miller, R., Gajos, K., Parkes, D., Horvitz, E. Human Computation Tasks with Global Constraints. In *CHI* (2012), 217-226.
24. Zweben, Stuart. 2012. Computing Degree and Enrollment Trends. Retrieved March 20, 2015 from http://cra.org/govaffairs/blog/wp-content/uploads/2013/03/CRA_Taulbee_CS_Degrees_and_Enrollment_2011-12.pdf.