Accessibility in an educational software system: Experiences and Design Tips

Laura Fathauer
IT Services
Miami University
Oxford, OHIO 45056. USA
fathaulg@miamiOH.edu

Dhananjai M. Rao
CSE Department
Miami University
Oxford, OHIO 45056. USA
raodm@miamiOH.edu

Abstract—Accessibility has been recognized as a key aspect of using technology for education. Yet educational software, including web-based systems, and even the main web-pages of top institutions often do not meet accessibility standards. When addressing accessibility issues, some creators of educational technology attempt to merely retrofit accessibility. Such an approach of "accessibility as an afterthought" often leads to low quality or even ineffective user interfaces (UI) and degraded user experience (UX) for all users. In order to avoid the aforementioned common pitfalls, we pursued an agile development process with an accessibility expert integrated into the process. The agile method has been used to develop a web-based automatic testing and grading software system called CODE. This paper presents our agile method and experiences with designing and developing an accessible software system. We discuss the software design and the some of the accessibility improvements recommended to meet accessibility standards. We present statistical analysis of 3,300 submissions from 47 students and their experiences. Our experience shows that even if software tools are able to identify accessibility issues, they are not effective in suggesting appropriate solutions to address them. A key "lesson learned" is that integrating an accessibility expert into software development teams is an effective approach to ensure effective and accessible UI/UX for educational technologies.

Keywords—Accessibility, Section 508, WCAG, Auto-grader, Programming, Agile Process

I. Introduction

Accessibility has been recognized as a key aspect of using technology for education [1]. In the United States 54.4 million people have a disability (18.7 percent of the overall population in 2005), while the number of persons with disabilities worldwide is estimated to be more than 550 million. Accessibility requirements are mandated by federal regulations as part of Section 508 of the 1998 Rehabilitation Act and will remain a critical aspect of future educational technologies. UNESCO also deems accessibility to be a key requirement to enable basic and inclusive education for all. The standards for accessibility adopted nationally and internationally are the Web Content Accessibility Guidelines (WCAG), and many commercial and freely available tools have been developed to test compliance for compliance [2].

A. Accessibility pitfalls and challenges

There is a clear need for accessible software in education. However, even the main web-pages of top institutions often do not meet accessibility standards [1,2]. In 2017, Acosta-Vargas et. al. analyzed the primary websites of 51 top Universities worldwide for compliance with WCAG guidelines using the WAVE browser plug-in [1]. They found that out of the 51 Universities, 44 of them did not even meet the minimum levels of WCAG compliance. The gap in accessibility extends to software and systems used in the classroom and online educational environments.

When addressing accessibility issues, developers of educational technologies often attempt to merely retrofit accessibility [12]. Such an approach of "accessibility as an afterthought" often leads to low quality or even ineffective user interfaces (UI) and degraded user experience (UX) for all users. Moreover, accessibility issues can manifest themselves in different ways. Bigham, Lin, and Savage discuss accessibility challenges in web-pages for blind users, where users are unable to even distinguish between inaccessible information versus information simply not being present [3]. It has been identified that addressing accessibility issues, particularly in educational technologies, requires "constructive disruption" in the software development life cycle (SDLC) [6].

B. Proposed alternative approach

The agile process was used to develop an automatic testing and grading plug-in called CODE, an extension for a course management system. Accordingly, to avoid the need to retrofit accessibility, we integrated an accessibility specialist into the agile software development process. Section 2 of this paper presents our agile process and development experiences and the need for a specialist to facilitate software development. Section 3 presents pertinent details on CODE. We discuss accessibility design recommendations, in Section 4. Section 5 and 6 discuss pertinent educational theories and related works. Section 7 presents the results from statistical analyses from 3,300 submissions by 47 students in support of the educational outcomes of this study. Section 8 concludes the paper, highlighting the importance of integrating accessibility early in

software design to improve experiences for all students in a classroom setting.

II. Integrating Accessibility in Agile Process

Agile software development methodology is widely used for developing software in an iterative manner. Each iteration, called a "sprint", involves a complete front-to-back life cycle often including delivery of partial subset of features to solicit customer feedback. Agile approaches have shown to be effective for developing software, particularly the first generation of systems.

We have used an agile process, specifically a slightly modified version of scrum, to develop the CODE plug-in introduced in Section III. We chose the agile process for the following reasons: ① a preliminary prototype for the plug-in had already been developed and we had a good idea about the anticipated design and outcomes; ② we had experts to support different aspects of the life cycle. Having experts is a key ingredient for success of an agile process; and ③ we had a relatively short time (i.e., 2.5 months of summer) to get key features operational and agile process was favored to enable all team members to be involved. We integrated an accessibility specialist from our University IT department into the process. The added cost was minimal, as the specialist was already working full time, dedicating about 2 hours per week (on average) working on this project.

The key phases in each sprint in our agile process are illustrated in Figure 2. Each sprint focuses on addressing one or more features or "stories". The plug-in has been developed in PHP programming language using the Laravel framework. The development phase also utilized the WAVE browser plug-in for identifying and resolving accessibility issues. The objective was to operate with the mindset that — "accessibility is an integral functionality of the software." In addition, it enabled us to address as many accessibility issues as early as possible.

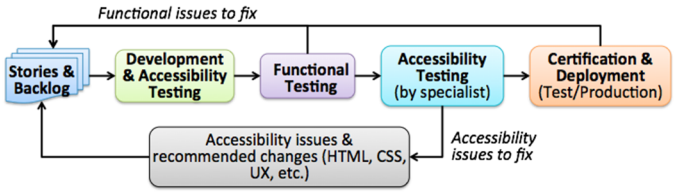


Figure 2: Overview of phases in each sprint in our agile process

A. Integrating an accessibility specialist

A distinguishing aspect of our agile process is that we integrated an accessibility specialist into the team. The role of the accessibility specialist was to enhance the software development process by providing iterative, incremental assessment of accessibility issues. A key objective is to address accessibility issues at the earliest opportunity to reduce development time and refactoring overheads.

More importantly, having an accessibility specialist involved in the agile process enabled us to realize several key benefits, namely:

1) *Addressing structural aspects of accessible interface:* An important consideration for accessibility is using semantic HTML, such as an appropriate use of heading tags. One of the issues with using software tools for accessibility assessment is that they were unable to identify semantic inconsistencies. This issue was particularly evident in the structural organization of web-pages, particularly for section dividing content missing heading markup (`<h1>`, `<h2>`, `<h3>`, etc.)

2) *Identifying missing components in UI:* Tools only identify issues with aspects of the interface that are present or are required. However, tools cannot identify missing information or inconsistencies, which were identified by our accessibility specialist. These included consistency in the information displayed in headings, informational messages, keyboard tabbing order, inconsistency in notations for expanded/collapsed panels, etc. Such user experience issues required a human to understand the context to identify issues and suggest solutions to address them.

3) *Focusing on the "why?":* While the Web Content Accessibility Guidelines (WCAG) do include technical recommendations, the primary focus is on supporting the use of technology by persons with disabilities. Accessibility experts bring this perspective to the software evaluation and can identify the impact of accessibility issues beyond simply technical compliance with the guidelines.

4) *Exploring alternative solutions:* In many scenarios the accessibility specialist was able to provide several alternative approaches to address accessibility issues. Discussing the alternative approaches enabled the team to identify effective solutions that balanced development overheads, future extensibility and maintainability, accessibility, and user experience.

5) *Improved user experience:* In addition to visual checks, the accessibility specialist used different screen readers (primarily JAWS and NVDA). While the recommendations developed arose out of accessibility testing, all users can benefit from the changes in structuring and ordering information and providing additional visual cues in the interface.

6) *Guidance for other manual checks of accessibility:* One of the issues with using software tools for accessibility assessment is that they currently only check for a subset of accessibility issues. Many accessibility tests must be done manually; for example: identification of issues with tabbing order of elements, missing ARIA tags on some elements, and inconsistent alternative text on image elements.

These experiences collectively helped the team realize shortcomings of tools and the importance of integrating an accessibility specialist in the development process. Experiences gained via the short sprint cycles enabled the team to quickly

build trust in the process, which is a key aspect emphasized in the agile manifesto.

B. *Outcomes and example recommendations*

Our agile process involving an accessibility specialist has been successful in enabling development and deployment of the CODE plug-in. Each sprint cycle lasted for 2 or 3 weeks, depending on the feature being developed. Sprints that involved changes to user interface were organized into several stories, with each one focusing on a specific interface. This enabled the team to stay focused on functional and accessibility testing.

Overall development of the plug-in was completed in about 4 months, with another month invested in testing and certification by Miami's eLearning committee and Miami IT. We have already successfully used the plug-in to teach a systems course with 47 students generated with over 3,300 submissions.

III. Overview of CODE Plug-in

The CODE plug-in has been designed to facilitate teaching and learning in programming-centric courses. It operates as a plug-in within the Canvas Learning Management Systems (LMS) from Instructure Inc. The CODE plug-in has been developed using the Learning Tools Interoperability (LTI) features of Canvas. CODE has been designed to retain all the familiar Canvas features, including assignment creation, assignment management, solution submission, grading via speedgrader, providing feedback, and managing grades via the gradebook. The objective is to enable students and instructors to operate within a familiar environment while providing additional features.

Figure 1 illustrates typical interactions with the CODE plug-in from both faculty and student perspectives. The process commences with the instructor configuring an assignment. Assignment configuration includes setting the programming language, style checking options, functional tests to be run. Instructors are encouraged to submit example solutions to verify that various settings and functional tests are correctly configured. Once an assignment has been configured and published, students may submit solutions to Canvas. Students upload their submissions and initiate grading. The grading process involves compiling, style checking, and testing student programs. The grading process is performed in independent Docker containers for improved security, isolation, and responsiveness. Typically, the grading process takes about 20 to 30 seconds. The results from the grading process are provided back to the students. Students use the feedback to enhance or bug fix their programs and resubmit their updated solutions for grading. Once students are satisfied with their solutions, they submit for grading. Instructors review student submissions and provide additional feedback either by annotating source code (supported by CODE plug-in) or providing feedback via the speedgrader feature of Canvas.

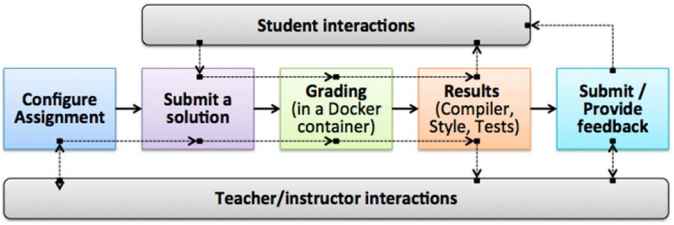


Figure 1: Typical interactions with code by faculty & students

A screenshot of the CODE plug-in results screen is presented further on in this paper. Detailed video demonstrations illustrating the use of our plug-in from both instructor and student perspectives are available online at https://code.cec.miamioh.edu/code.

IV. Design considerations for Accessibility

Below are example recommendations from the accessibility specialist that were integrated into CODE. These recommendations included explanations of the "why" of the recommendations. Moreover, these were issues that tools did not identify. These recommendations are based on the Web Content Accessibility Guidelines (WCAG) that address issues that are important for several types of disabilities, including blind/low vision, color-blindness, deaf/hard of hearing, and mobility impairments. Though many of the items in WCAG can help people with disabilities better understand software and content, issues for people with cognitive disabilities are not extensively included.

1) Updates within the embedded iframe: Some One of the ways blind and low vision users can interact with technology is through a screen reader, which converts text to speech. These users should receive some kind of indication of the results of their actions, such as a response to a successful submit action. The CODE plug-in is displayed within an HTML iframe within the Canvas. The plug-in periodically refreshes this content to update its status, so the plug-in acts like a single-page application. One strategy to let users know that their action has been successful is to set the keyboard focus on the top element of the new content when it loads. In this situation, the CODE plug-in sets the focus back to the top-element when content is reloaded; screen reader users would hear the newly focused content.

2) An accessible countdown timer: The CODE plug-in page refreshes every 10 seconds to update the status from the asynchronous grading process running in a Docker container. It uses a progress bar to visually provide indication as to when the information will be refreshed. Correspondingly, there is a "progress bar" design pattern that provides updates to screen reader users. The details on the progress bar design pattern are available from the ARIA-1.1 document.

The code section for implementing a progress bar is shown in figure 3. The div element shown includes the relevant ARIA attributes to communicate to screen reader users the changing

values. The attribute "aria-valuenow" communicates the current value of the progress, and is updated as the timer counts down.

Figure 3: Sample HTML from CODE implementing the progress bar recommendation.

3) Use of semantic HTML: The results screen in the CODE plug-in includes complex content that needed to be properly organized to ease access to the information. Rather than using simply visual styling or listing data in paragraphs, the accessibility specialist recommended to have a heading and have the details on a row below that. Another recommendation was to use an HTML list to explicitly call out each specific submission requirement. The list streamlines the experience for both regular and assistive technology users who can get a quick summary of the number of requirements.

4) Consistent headings: Adhering to structured use of HTML heading levels improves navigation for screen reader users and improves understanding the structure of the content for all students. Reusing LTI specification recommendations for headings streamlines readability.

A portion of the CODE results screen is shown in Figure 4. The results screen includes major sections that contain the results along with nested subsections. The accessibility recommendation was to mark the blue "Testing Result Summary" section with a Heading 2 element (h2), and to add a "results" section heading at the same H2 level. These headings provide the structure of the results screen, as well as providing screen reader users a way to more easily locate information.

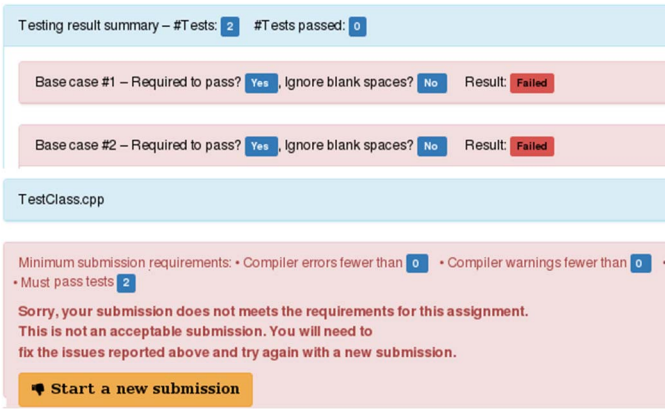


Figure 4: Screenshot of CODE result screen with recommendations for heading markup

5) The "Add file" button text is confusing: The CODE plug-in provided a button for users to add multiple source files for a program to be included in a single submission. The development team used an "Add file" button that adds a second button to upload a file, which is used to actually upload the file. However, the accessibility specialist identified this as a source of confusion and inconsistency. Instead the functionality was streamlined by removing the first button. This style of operation is also consistent with the approach used by Canvas LMS, which helped to further streamline experience for all users.

IV. Pertinent Education Theories

The focus of this work is on exploring the effectiveness of integrating accessibility assessment into an agile process. For the CODE plug-in, the focus on accessibility stems from the interplay of two key educational theories underlying its philosophy, namely: behaviorism and constructivism. Behaviorism is a learning theory that focuses on eliciting desirable behaviors, typically through repetitive conditioning [7]. Specifically, Computer-based Training (CBT) builds on behaviorist theory to facilitate learning particularly at knowledge, comprehension, and application cognitive levels of Bloom's taxonomy. Our CODE plug-in falls into the category of behaviorist tools because it is designed to elicit several positive behaviors, including: ① maintaining good coding style via its style checkers, ② improve structure and cohesion by limiting methods to 25 lines ③ regular testing via automated tests.

The CODE plug-in also embodies the essence of constructivism in which students actively learn by solving a problem iteratively. The CODE plug-in runs functional tests and provides immediate feedback to the students. The feedback enables the students to observe issues, form mental models, and enhance their solutions to correct reported errors. Combined with the aforementioned behavioral outcomes, the plug-in also facilitates learning at the application, analysis, and synthesis levels in Bloom's taxonomy.

Accessibility is a vital requirement for success in both behaviorism and constructivism [6,7] because both learning theories rely on the learner effectively utilizing the feedback. Poorly designed feedback or inaccessible user interfaces (UI) negatively impact student learning. Moverover, accessible user interfaces (UI) have become critical for students with visual impairments, learning, and other disabilities. Providing accessible materials is federally mandated. Consequently, to avoid the pitfalls discussed in Section 1.A, we have pursued an agile process with an accessibility specialist integrated into the software development life cycle as discussed in Section IV.

VI. Related Works

Accessibility of educational software is an active area of research and development. Several tools, processes, and philosophies (such as Universal Design), have been proposed to facilitate the design and development of accessible technologies. The focus on this work is accessibility in a web-based, educational software system and hence we focus only on this category of recent scholarly works.

Bai et. al. discuss the importance of including accessibility testing in an agile process [9]. They argue that developers,

testers, and designers should share responsibility for accessibility. They present results from 2 case studies and perform a cost-benefit analysis of 9 different methods to identify accessibility issues. From their analysis, they conclude that a combination of several methods, involving tools and an accessibility specialist, yields the best results for identifying issues in software [9]. In a more recent work focused on universal design [12], they present results from a survey of 89 members of an agile team including developers, testers, and project leaders. Their survey results show that accessibility and university design was largely ignored by almost half of the team members. Only 13% of the team members had universal design integrated into their activities. This observation stresses the need for a dedicated accessibility expert to be involved in agile teams, similar to our proposed agile process.

Similar to our work, several recent accessibility investigations also use the accessibility checking tool called WAVE. WAVE (https://wave.webaim.org/) is a popular web accessibility evaluation tool available in the form of a browser extension. It checks conformance of websites with the Web Content Accessibility Guidelines (WCAG) from the Word Wide Web Consortium (W3C). Acosta-Vargas *et. al.* analyzed the primary web-site of top 51 Universities worldwide for compliance with WCAG using the WAVE browser plug-in [1]. They also report a similar study involving the use of the WAVE browser-plug-in, but focusing on just the universities in Latin America [8]. Their work also includes reviews of similar investigations and we refer readers to references therein for research involving the use of the WAVE browser-plug-in. In contrast, in our work, we use the WAVE browser-plug-in as an integral part of our development and accessibility-testing processes.

Marsi and Lujan-Mora propose a combined method for evaluation of the accessibility of web-applications based on WCAG guidelines [11]. They propose the use of a Web Accessibility Barrier (WAB) metric that quantitatively summarizes results from different types of accessibility issues on a given website. In a recent publication with Sanchez-Gordon [10], Lujan-Mora proposes a 5-step process for accessibility testing of web-applications in the agile process. Their work focuses on important details to the testing phase and emphasizes the use of multimodal testing via tools, simulators, and experts.

These investigations [9,10,11,12] emphasize the importance and effectiveness of using both tools and an expert to identify accessibility issues. They essentially add support for our process involving the use of both tools and an accessibility expert in an agile software development process. Unlike the earlier works that aim to be generic, this paper is focused on the details of using an extended agile process to develop an accessible, educational software system.

VII. Assessments and Results

The proposed Code Assessment Extension (CODE) plug-in has been used to teach a systems course (1 semester, *i.e.*, 15 weeks) with heavy emphasis on programming in C++. The topics in the course include core Operating Systems (OS) concepts, including multiprocessing, multithreading, and virtualization. This one semester-long course (15 weeks) consisted of 54 students with junior standing from 3 specializations, namely: computer science (42), software engineering (5), and computer engineering (7).

The CODE plug-in was used for submission and assessment of student submissions for 6 homework assignments. In total, students submitted over 3,300 submissions as part of the 6 homework, with different number of submissions per-student for each homework (see Figure 5). The following subsections analyze pertinent characteristics of student submissions to draw inferences about usability and accessibility of the plug-in. In addition, we also present data from end-of-course anonymous student survey.

A. Assessment of student learning curves

The use of the CODE plug-in was introduced in the second week of the course as part of a short lab exercise. A video demonstration (about 9 minutes) illustrating the use of the CODE plug-in was played in class. The students were directed to utilize the video demonstration and submit solutions for the lab exercise. All of the students were able to successfully complete the lab exercise and submit solutions via the plug-in without any issues. The short learning curve is reflective of a design that is both intuitive and accessible to students. Moreover, the students did not solicit any further assistance or clarification regarding the use of the plug-in for completing the assigned homework. This observation adds further credence in support of the effectiveness of universal design philosophy underlying the CODE plug-in.

B. Analysis of plug-in usage volume

Having successfully completed the introduction of the CODE plug-in, it was used for assessment of 6 separate homework in the course. Homework was not assigned during weeks around 2 midterm exams and during Thanksgiving break. The students had 7 days to complete each homework that was due at midnight. However, the complexity of the assignments varied throughout the course, with complexity generally increasing throughout the course.

The histograms in Figure 5 summarize the number of submissions used by students for each of the 6 homework. As illustrated by the histograms, most students required several submissions for each homework. It must be noted that students were strongly encouraged to fully test their programs in their Integrated Development Environment (IDE) prior to uploading their programs to the plug-in. The average number of submissions generally increased from 6.13 to 13.95, reflecting the trend in increasing complexity of homework. However, HW #4 was a larger term project and consequently, the average number of submissions (Mean: 18.77) was higher.

The data in the histograms shows that the students did not shy away from using the plug-in. They actively used the plug-in for testing their programs and repeating submissions to ensure successful completion of homework. The usage patterns

suggest that the students found the plug-in relatively straightforward to use — *i.e.*, if the plug-in was not accessible or easy to use, we would have observed a much lower volume of usage (average of 3-to-4 submissions), particularly given that students were strongly encouraged to fully test their programs in their IDEs prior to submission. Instead, the students actively used the plug-in thereby indirectly establishing the effectiveness of its universal design.

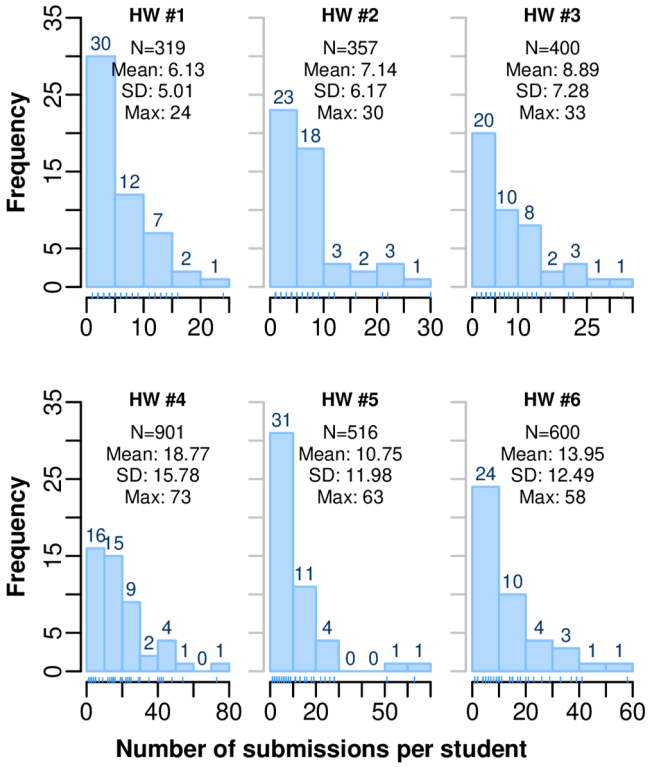


Figure 5: Histogram of submissions by students

C. *Analysis of temporal-patterns of usage*

The previous subsection analyzed the volume of plug-in usage by the students. In this subsection, we analyze the temporal characteristics of the plug-in usage that was observed across all 6 homework assignments. Specifically, we observe the number of submissions received with respect to the homework deadlines. The histogram in Figure 6 plots the distribution of number of submissions with respect to the deadline. As illustrated by the histogram, most of the submissions for homework were received in the last few hours prior to the deadline. Across all 6 homework, over 62% of the submissions occurred in the last 8 hours. The submission patterns were consistent immaterial of the increasing complexity of the homework. Recollect that students had 7 days (including weekends) to work on each homework.

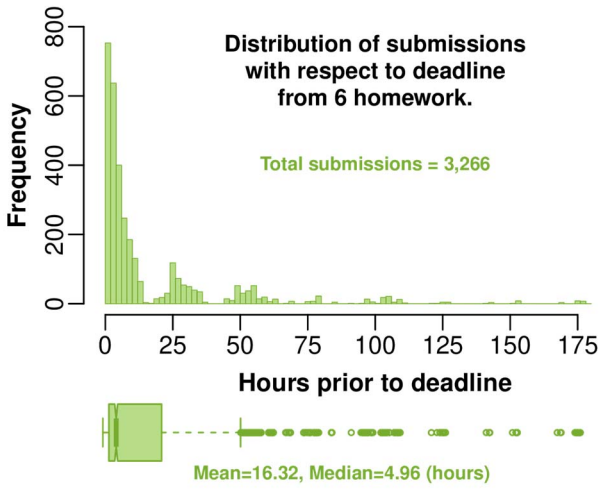


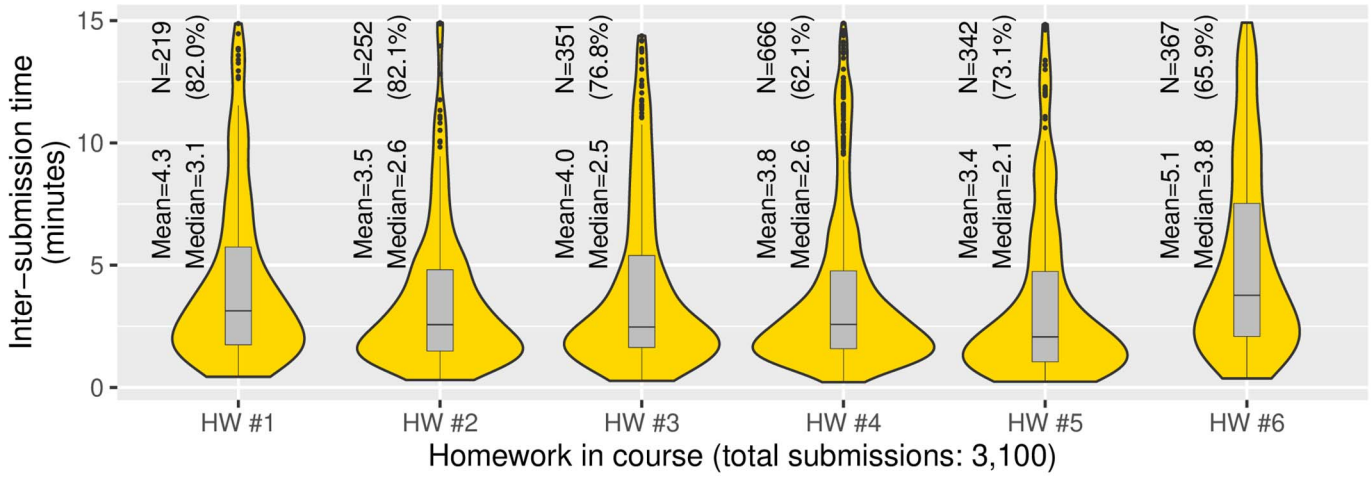
Figure 6: Histogram of number of submissions received with respect to homework deadlines.

The data in Figure 6 strongly suggests the students actively work on homework, rushing to finish their work just before deadlines. During such a high activity the usability, and accessibility of the plug-in plays a critical role — *i.e.*, pertinent information needs to be readily accessible to the students.

Accordingly, to assess the accessibility and effectiveness of our design, we have conducted a more detailed analysis of inter-submission times. Inter-submission time is defined as the duration between the time when results were presented to a student and the time when the student started another submission. The inter-submission time essentially indicates the time a student took to observe an error report on the CODE plug-in, fix the issue reported, and start another submission. Here, we use the inter-submission time as an indirect measure of usability and accessibility of our system.

The violin plots in Figure 7 illustrate the distribution of inter-submission times (in minutes) observed for the 6 homework. Note that Figure 7 only shows submissions whose inter-submission time is less than 15 minutes. This threshold is used to characterize the high-volume submission timeframes close to deadlines (see Figure 6). Interestingly, this 15-minute range covers a majority of submissions, with > 80% for HW#1 and HW#2.

The median inter-submission time ranged from ~2 minutes for HW #5 up to about ~4 minutes HW #6. That is, about 2-to-4 minutes were sufficient for students to review the report from CODE plug-in, modify their programs, test it, and resubmit their solution. As indicated by the long-tailed distributions of the violin plots, most of the submissions occurred in relatively quick successions. This suggests that students were able to quickly glean information from the plug-in and utilize it in a short timeframe. This observation strongly supports the inference that the CODE plug-in is effectively designed to



present pertinent information, the information is readily accessible, and the plug-in straightforward to use.

D. Analysis of anonymous end-of-course student feedback

The statistics in Table I present student responses to questions on an end-of-course anonymous survey. The survey focused on student perceptions about the CODE plug-in and their use of it. Overall student perception about the use and effectiveness of the CODE plug-in was positive. Responses to the first question in Table I indicates that about 80% of the students agreed that automatic testing helped to reduce errors in their program. Similarly, responses to the 4th question in Table I show that 77% of the students agree that using the plug-in has helped to improve their solutions. Importantly, responses to the 3rd question in Table I shows that nearly 80% of the students have come to rely on some of the automated testing to verify their solution. The positive experiences and the reliance on the CODE plug-in add further support to the inference that the plug-in has been effectively designed.

In conjunction with the inferences drawn from the statistical analyses, the results strongly support the inference that the plug-in provides a good user interface, enhances user experience, and presents information in an accessible format for all students. Collectively, the quantitative and qualitative results strongly support successful application of universal design principles underlying the CODE plug-in.

TABLE I
Student feedback from anonymous end-of-course survey. Legend: SD: Strongly Disagree, D: Disagree, N: Neutral, A: Agree, and SA: Strongly Agree

Survey question (N=47)	SD	D	N	A	SA
I think automatic testing has helped reduce errors in my program.	0 0%	4 8%	6 13%	28 60%	9 19%
I liked getting early feedback whether I have met most (if not all) program expectations via automated testing.	0 0%	1 2%	10 21%	20 43%	16 34%
I have come to rely on some of the automated testing to check if my program actually works as expected	1 2%	8 17%	10 21%	21 45%	7 15%
Overall I think the automatic testing has helped me improve my programs.	1 2%	6 12%	4 9%	23 49%	13 28%

VIII. Conclusions

Technology and web-based software will continue to permeate and disrupt education. Software systems, particularly educational software, need to meet accessibility requirements so that they can be effectively used by millions of learners with a variety of disabilities. However, most web-pages of even top institutions fall short on WCAG accessibility standards [1,2]. Moreover, software engineering teams often do not include accessibility and universal design considerations in their development processes and often retrofit accessibility [12]. Such an approach of "accessibility as an afterthought" often leads to low quality or even ineffective user interfaces (UI) and degraded user experience (UX) for all users.

A key challenge is that accessibility is a complex spectrum of issues that requires a specialist to be involved in the Software Development Life Cycle (SDLC), specifically in: design,

development, and testing. Accordingly, we integrated an accessibility specialist into an agile software development process. Section 2 of this paper presents our enhanced agile process with two key changes. First, we have integrated an accessibility specialist into the development team. Second, we have an explicit accessibility testing phase in each one of our sprints.

The agile process was used to develop an automatic testing and grading plug-in called CODE. It has been developed to streamline teaching and learning in programming-centric courses. CODE has been developed as a plug-in to our existing Canvas LMS to take advantage of its features and retain the familiar environment for instructors and students. Section 2 discussed pertinent details about CODE and its usage.

Section VII presented statistical analysis of pertinent characteristics of submissions to indirectly assess the accessibility, user experience, and overall effectiveness of the plugin. Collectively, the quantitative and qualitative results strongly support successful application of universal design principles underlying the CODE plug-in.

Our experiences highlight the following key benefits that were realized through our enhanced agile process involving an accessibility specialist. First, we were able to address structural aspects of our system early in the design cycle thereby improving our ability to address accessibility issues. Next, accessibility testing was integrated into the development process using tools like WAVE to regularly check and report issues to be addressed. This helped to reduce many routine issues and facilitated the specialist to focus on higher-level issues. The accessibility specialist focused on identifying and providing recommendations based on guidelines and best practices. Finally, and importantly, the specialist was able to help the team to focus on the underlying reasons which helped the team to design better solutions. The accessibility specialist was instrumental in exploring design alternatives to address the issues, steering the team towards universal design.

Our experiences suggest that integrating an accessibility specialist into agile teams is an effective strategy to develop accessible software systems. In our instance accessibility was a mandatory requirement, and hence identifying and addressing issues early in the design cycle helped to complete our project in a timely manner. We estimate that accounting for accessibility early eliminated making large structural changes to our software later, thereby reducing overall development time by about 15%. In addition, our specialist was working on multiple projects, which amortized the cost of the added agile team member. Given the time savings and only incremental cost to including an accessibility specialist, we maintain that our enhanced agile process can be successfully reused for other projects. Our experiences strongly suggest that our approach will facilitate development of educational software that will meet -and exceed- accessibility requirements while providing improved experience for all.

ACKNOWLEDGMENTS

We would like to thank faculty and staff in eLearning Miami and Miami University IT for their help and support in certifying and deploying the CODE plug-in.

REFERENCES

[1] P. Acosta-Vargas, S. Luján-Mora, and L. Salvador-Ullauri, “Web accessibility policies of higher education institutions,” in 2017 16th International Conference on Information Technology Based Higher Education and Training (ITHET), July 2017, pp. 1–7.

[2] J. P. Bigham, I. Lin, and S. Savage, “The effects of “not knowing what you don’t know” on web accessibility for blind web users,” in Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility, ser. ASSETS ’17. New York, NY, USA: ACM, 2017, pp. 101–109.

[3] T. Calle-Jimenez, S. Sanchez-Gordon, and S. Luján-Mora, “Web accessibility evaluation of massive open online courses on geographical information systems,” in 2014 IEEE Global Engineering Education Conference (EDUCON), April 2014, pp. 680–686.

[4] Y. Mohamad, C. A. Velasco, N. Kaklanis, D. Tzovaras, and F. Paternò, “A holistic decision support environment for web accessibility,” in Computers Helping People with Special Needs, K. Miesenberger and G. Kouroupetroglou, Eds. Springer International Publishing, 2018, pp. 3–7.

[5] S. W. van Rooij and K. Zirkle, “Balancing pedagogy, student readiness and accessibility: A case study in collaborative online course development,” The Internet and Higher Education, vol. 28, pp. 1 – 7, 2016.

[6] S. N. Elliott, R. J. Kettler, P. A. Beddow, and A. Kurz, “Accessibility progress and perspectives,” in Handbook of Accessible Instruction and Testing Practices. Springer, 2018, pp. 3–7.

[7] B. Bakhshinategh, O. R. Zaiane, S. ElAtia, and D. Ipperciel, “Educational data mining applications and tasks: A survey of the last 10 years,” Education and Information Technologies, vol. 23, no. 1, pp. 537–553, Jan 2018.

[8] P. Acosta-Vargas, T. Acosta, and S. Luján-Mora, “Challenges to assess accessibility in higher education websites: A comparative study of Latin America universities,” IEEE Access, vol. 6, pp. 36 500–36 508, June 2018.

[9] A. Bai, H. C. Mork, and V. Stray, "A Cost-Benefit Analysis of Accessibility Testing in Agile Software Development Results from a Multiple Case Study," International Journal on Advances in Software, vol. 10, no. 12, pp. 96-107, 2017.

[10] S. Sanchez-Gordon and S. Lujan-Mora, "A Method for Accessibility Testing of Web Applications in Agile Environments," 7th World Conference for Software Quality (WCSQ 2017), Lima (Peru), March 20-22, 2017.

[11] F. Masri and S. Lujan-Mora, "A Combined Agile Methodology for the Evaluation of Web Accessibility," IADIS International Conference Interfaces and Human Computer Interaction 2011 (IHCI 2011), pp. 423-428, Rome (Italy), July 24-26 2011. ISBN: 978-972-8939-52-6.

[12] A. Bai, H. Mork, and V. Stray, "How Agile Teams Regard and Practice Universal Design during Software Development," In "Transforming our World Through Design, Diversity and Education," Studies in Health Technology and Informatics, vol 256, pp. 171-184, 2018. doi: 10.3233/978-1-61499-923-2-171