

MASLO: A MOBILE LEARNING DEVELOPMENT SYSTEM

Rovy Branon¹, Moses Wolfenstein², & Cathrin Weiss²

¹University of Washington; ²University of Wisconsin—Extension

The Mobile Access to Supplementary Learning Objects (MASLO) is an open source software kit developed by the Advanced Distributed Learning Co-Laboratory (AADLC) at the University of Wisconsin-Extension. The MASLO kit is designed to provide the components for content authoring and delivery on Apple iOS® and Google Android® smart phones. This project built on work from an earlier AADLC effort to develop a high school test preparation mobile application called Revu4u (Review for You). In that project, AADLC team members successfully created a technical framework for delivering smart phone enabled instructional content. While the technical components functioned as intended, virtually no time was spent in Revu4u on creating usable interfaces. Additionally, the Revu4u project was limited in scope to only deliver multiple-choice questions and feedback. The MASLO project focused on usability rather than technological capability and was designed as a more comprehensive instructional content authoring environment. In this article, the authors will describe MASLO as a design case. The purpose is to clearly describe the kit itself, critical design decisions, and the context and situations relevant to understanding the decisions made. Relevant Revu4u processes and outcomes are briefly described as a precedent for MASLO. MASLO is a living, open source project, and this case describes the development of the kit up to August 2012.

Rovy Branon is the Vice Provost for Continuum College at the University of Washington. Previously, Rovy was the Associate Dean and Executive Director of the Academic Advanced Distributed Learning Co-Lab at the University of Wisconsin-Extension. His research includes expanding access to higher education through technology.

Moses Wolfenstein is the Co-director of the Academic Advanced Distributed Learning Co-Lab at University of Wisconsin-Extension. Moses' research interests include game-based learning environments and design.

Cathrin Weiss was previously a Senior Software Engineer at the Academic Advanced Distributed Learning Co-Lab at the University of Wisconsin-Extension.

INTRODUCTION

This design case describes the development of a platform for mobile learning rather than a single instructional intervention. The first section details the context of the case, including goals, location, philosophy, precedent work, time, budget, and personnel. The second section describes the MASLO system, including components of the system and key design decisions. The final section is a brief discussion of the current state of MASLO and the relevance of MASLO as a design case.

DESIGN CONTEXT

In 2011, The Academic Advanced Distributed Learning Co-Laboratory (AADLC) at the University of Wisconsin-Extension was awarded a research contract to develop a mobile learning platform by the U.S. Department of Defense (DoD). The contract was funded through a Broad Agency Announcement (BAA) call for research. In that call (BAA: W91CRB-08-R-007, 2010) the DoD described an interest in the following mobile learning capabilities:

Mobile devices free learners from traditional learning environments and allow greater knowledge sharing in social environments. There is a need at ADL to determine the practical relationship between distributed learning and knowledge sharing, and how mobile devices factor into that relationship, especially if SCORM is required. Of particular interest is how structured learning content that is based on predetermined product design requirements is maintained, updated then distributed to mobile devices at the time of need.

A subsequent series of conversations between the AADLC and the DoD program manager indicated that a proposal

Copyright © 2016 by the International Journal of Designs for Learning, a publication of the Association of Educational Communications and Technology. (AECT). Permission to make digital or hard copies of portions of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page in print or the first screen in digital media. Copyrights for components of this work owned by others than IJDL or AECT must be honored. Abstracting with credit is permitted.

to extend a previous AADLC mobile system project would fit within these requirements. The new effort was called Mobile Access to Supplemental Learning Objects (MASLO). The name was created to reflect a belief that mobile phones might be best used to support supplemental instruction rather than serving as a primary mechanism for course delivery. Additionally, small, bite-sized course elements were the cornerstone of the learning object approaches being developed by the Advanced Distributed Learning (ADL) initiative. MASLO is an ongoing open source project, but this design case will describe the funded period of performance in the research contract. Work began in March 2011 and the initial prototype was completed in August 2012.

Location and Environment

The Academic Advanced Distributed Learning Co-Laboratory (AADLC) is a learning technology research group at the University of Wisconsin-Extension. The AADLC was originally formed in 2000 as a collaborative effort between the University of Wisconsin System and the Wisconsin Technical Colleges System. It was chartered as an official node in the U.S. Department of Defense ADL Initiative. Multiple labs representing different sectors were created at other locations (e.g., the Workforce ADL Co-Laboratory at the University of Memphis). In the early 2000's the ADL Initiative focused on the development of technical standards to support e-learning. The most well-known of these efforts is called the Sharable Content Reference Model or SCORM. As the academic lab in the ADL effort, the AADLC at the University of Wisconsin advocates the use of e-learning standards in K-20 settings. In the mid-2000s the role of the AADLC began to include research on other emerging learning technologies including the use of games, mobile systems, and augmented reality. These efforts were funded under various grant efforts and were often led by faculty at the UW-Madison in collaboration with other institutions (e.g., MIT, Harvard, etc.). The U.S. Department of Defense also continued to provide funding to the AADLC for a variety of technical tasks and white papers. For several years, the AADLC also operated under a funding agreement with the Florida Virtual School as a research and development partner. It is relevant to note that while projects were often instantiated through training and education on a topic, most of this work is related to the development or design of technologies to support learning rather than on specific instructional interventions (e.g., development of courses). In other words, the AADLC is not an instructional design production shop but is focused on testing emergent technical standards and developing platforms to support instructional design work. Some examples of these platforms include technical standards (e.g., SCORM, xAPI), mobile applications, and educational games.

In the latter part of the decade, the AADLC became part of the University of Wisconsin-Extension's Continuing Education, Outreach, and E-learning (CEOEL) division. The

connection to the U.S. DoD ADL Initiative remains, and the majority of work for AADLC continues to be software development. One significant difference, however, is an increased effort to connect research projects to instructional needs of the University of Wisconsin-Extension, which includes online degree development, agricultural outreach, and community education. About the same time the AADLC started to change its direction, the U.S. DoD ADL Initiative launched new research efforts on mobile learning. MASLO is one of those new efforts.

Team Members

The team that worked on MASLO included several positions. More about the team member views, impact of turnover, and decisions are included throughout the case. When the project began, all team members were men and from the United States. When the mobile software developer left the research lab, a woman, originally from Germany, replaced him. Team members ranged in age from 25 to 45 years and all had at least a college education. The principal investigator and the additional researcher both hold a Ph.D. (in Instructional Systems Technology and Educational Leadership and Policy Analysis, respectively). The following list provides an overview of the project positions and their respective time allocations:

- Mobile software developer/programmer—allocated at 100%.
- A second programmer—allocated at 50%.
- Principal investigator—committed 25%.
- Interface designer (graphic artist)—allocated at 25%.
- Additional researcher/usability role—was added outside of the project budget and worked approximately 60% of his time for the duration of the project.

MASLO Project Goal

The primary goal of this project was to develop a prototype mobile learning publishing platform that included the following components and features. Some of these features were related to client contractual requirements and others were the result of precedent work, both of which are described later in the case.

- An authoring tool for creating mobile-friendly content.
- iOS (Apple) and Android (Google) applications for delivering the content.
- A separate server for storing content so that content could be authored once and then downloaded and formatted by the mobile applications.
- Offline capability: All content (once downloaded) must be available to the learner even if no internet connection is available.

- Open source: the code and documentation must be made available for free download and be open to modification.
- Ensure a “dual purpose” use for both military training and K-20 education.

Design Philosophy

The design philosophy was not explicitly stated or discussed by the team at the beginning of the project but is implied through the project goals and outcomes. Later sections of this design case describe the team members’ roles and views. One emergent philosophical view held by all the team members is that technology should help improve access to educational opportunities. The word “access”, in this case, means more than the ability for an end user to acquire educational materials or instructional experiences. Access also means that those with knowledge have access to easily usable tools to enable sharing among a global audience of learners depending on the topic area.

More broadly, improving access includes a myriad of other issues related to modern computing technology, including an awareness of network access constraints for those unable to afford broadband for economic reasons and those disenfranchised by simple lack of network capability in many rural areas. Access also includes access to the designed product itself through open source licenses rather than locking it down through a patent or pay licensing process (Branon & Wolfenstein, 2013). Evidence of the team’s belief that access is a primary requirement is found in many of the MASLO design decisions.

A parallel philosophical commitment that was implemented implicitly was the view that parsimony in design was essential to the success of the project. Although the team did not consciously invoke the tradition of minimalist design (e.g., Carroll, 1990), the specific history of the project combined a commitment to usability principles (e.g., Nielsen, 1999; Norman, 1988) and a general inspiration from contemporary website and software design (e.g., Maeda, 2006) to advance an aim of simplifying interaction and user experience as much as possible. Late in the project the team arrived at the conclusion that parsimonious design dovetailed with the notion of access as an essential objective in the creation of a new technology. By designing parsimoniously, the final product was more accessible to more users.

Precedent Work: Revu4u

MASLO was funded and conceived on the basis of previous mobile technology work at the AADLC. Revu4u is a high school test preparation app the AADLC developed for Florida Virtual School (FLVS). The goal of Revu4u was to help FLVS test the “mobile learning application development waters” on a small budget. Limiting the scope to test preparation kept Revu4u financially viable. Revu4u was also the first mobile development project for the AADLC and we wanted to test several technical possibilities. Revu4u was successful as a technical test bed and demonstrated that content for a native iPhone® application could download and run content stored in a third-party “cloud” database (see Figure 1). While such architectures are common today, this was a relatively novel approach in 2009 (Branon, Wolfenstein, & Raasch, 2012).

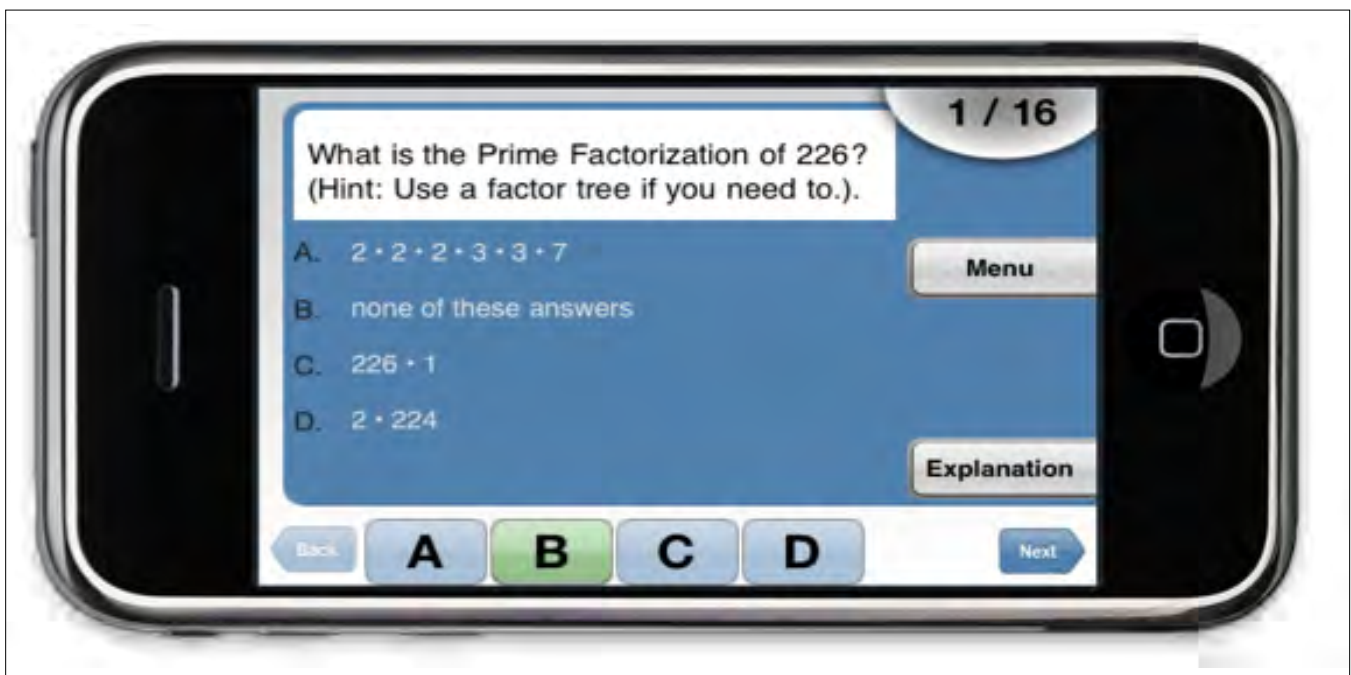


FIGURE 1. An example of a typical Revu4u multiple-choice question screen.

The complete details of Revu4u are beyond the scope of this article but that effort directly influenced numerous MASLO design decisions. Specific Revu4u influences are described throughout this case.

Other 2010 Mobile Learning Technologies

In early 2010, while the team was developing the MASLO Broad Agency Announcement proposal, learning management system providers were beginning to offer mobile applications allowing students access to online courses through their smart phones. These apps were often less-than-full-featured versions of the learning management system but were designed to support full courses.

Other stand-alone proprietary mobile learning products available at the time (e.g., LearnCast.com) were also developed with the idea that people wanted to create full courses for delivery to mobile phones. In early team discussions, the authors did not want to replicate this full course model. The team believed that there was a need for a lightweight application that could supplement face-to-face or online instructional environments. In other words, we did not want to shrink a course and put it on a mobile device. We wanted instructors to think about parts of courses or even unique additions to courses that might be well-suited for delivery on a mobile device.

This desire to focus on a subset of instructional delivery was based, in part, on the personal experiences the design team had with their own mobile technology use. Team members felt that mobile apps worked best when they focused on a small number of features. Recognizable apps available in 2009-2010 included Facebook, Evernote, Dropbox, and Yelp. The team discussed how these mobile apps were often more limited in scope than their more full-featured web-based versions. In some cases it was clear to the team that the apps were only limited for technical reasons and not by design but some cases, like Todo, were intentionally sparse. The reasoning was that mobile devices could have any number of applications and therefore each individual app could be relatively simple. The design of MASLO would allow rapid development of small mobile learning applications, rather than a comprehensive learning content system.

Additionally, many of the available proprietary mobile learning platforms locked content into a particular service or device. The developers wanted to separate content from a particular mobile application and ensure portability of content out of a MASLO database. The team believed these considerations were differentiators for an open source system in a growing field of mobile learning applications.

Project Timeline

Originally proposed in May 2010, the MASLO project was approved by the U.S. Department of Defense in September

2010. Contracts were signed and executed between the University of Wisconsin and the U.S. Federal Government in April 2011. At that time, the project was approved as a one-year research and development effort with an expected end date of March 2011. A turnover in personnel required a six-month no-cost extension and the final deliverables were completed by August 30, 2012.

While a one-year timeline might seem relatively short for the development of a fully functional prototype content creation platform, the changes in the mobile technology landscape over the total two years from proposal writing to completion were dramatic. The design team felt constrained to stay within the project guidelines. Some of these tensions will be evident in MASLO design decisions.

Budget

MASLO was funded with a \$308,000 contract award. Most of the money was allocated to pay for software developers (35%), other funding paid a partial buyout of the principal investigator's time (10%), an interface designer (20%), and student time (5%). The balance of the grant was for required institutional administrative overhead, travel, and technology acquisition. There was no requirement for matching funds or effort, but additional university human resources were used on an ad hoc basis because the software was viewed as potentially beneficial to the institution.

MASLO: CASE OVERVIEW AND CORE DESIGN FEATURES

MASLO is best described as a kit for developing mobile learning applications. The MASLO kit contains three major pieces: a content development tool that runs on a personal computer, a web or "cloud" storage component, and mobile applications for learners to consume content on iOS® and Android® devices (see Figure 2). MASLO is not a single app or service. The raw code for the apps, the web server, and the content tool are available for developers to download and then modify for their own institution. The code is available here: <http://academiccolab.org/maslo>.

Intended Audience and Contexts of Practice

MASLO is intended to be a platform for creating content rather than instructional content for a particular audience. The intended audience is content experts, not professional instructional designers or developers. While nothing excludes designers from using the tool, the lack of sophisticated tools may create frustration for power users.

While the context for the use of MASLO is open, the system requires access to a laptop or desktop computer and a mobile device for testing. Requiring both of these limits the use to those with access to these tools. The authors are aware that this might limit use in places where users do not

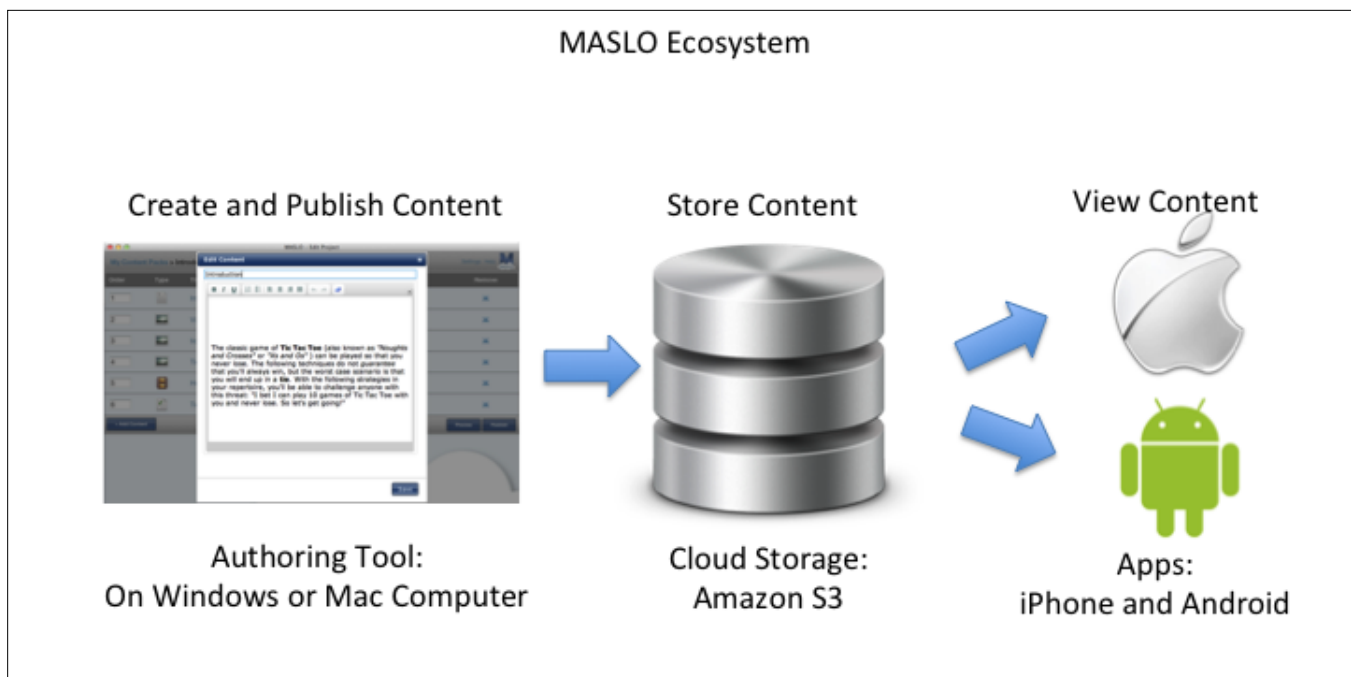


FIGURE 2. This diagram shows the three components of the MASLO ecosystem.

have access to computers but, at the time the system was in development, creating an authoring tool that could run on a mobile device was not within the scope.

How MASLO Works

Content authors (faculty, teachers, and trainers) use a simple computer-based authoring tool to create instructional content packages. The authoring tool, which will be described later in the case, is more simplified than the online learning tools in a learning management system or even a word processor. Content authors upload these packages to a web host so that they are available for download by the learners. A learner downloads a mobile application to his or her device. Once in the application, learners see a list of downloadable content packages stored on the web server. The learner selects which content packs to download to their device and then accesses them at any time, whether connected to the internet or not.

The possible types of content inside a package include text, video, audio, images, and multiple-choice quizzes. The quizzes are not scored or graded but are intended to provide the learner with a way to test their knowledge.

High-Level Platform Design Decisions

The MASLO contractual requirements dictated several design decisions. These included:

- Open source
- Author once and view on multiple devices
- Offline content access
- Simplified mobile content authoring environment

Open source

The decision to make all of the code available as open source was both pragmatic and driven by a decision to make the product as accessible as possible. For projects funded through the U.S. Federal Government, there is an increasing bias to fund software that is developed under open source licenses. While not a specific requirement, the idea that the original software is GOTS (Government off-the-shelf) means that the Federal government can use the software without additional licensing. Such preferences do not preclude patenting parts or the entire product, but declaring in the research proposal that the code would be released as open source eliminated a number of contractual complexities.

Another pragmatic aspect of choosing to make the product open source is that the design team was able to use other products available under similar license arrangements. As one brief example, the developers embedded the open source video player called VLC into the MASLO authoring tool. If MASLO was not open source, the VLC license would not have permitted free use within the tool (VideoLAN, 2015).

Beyond the more pragmatic aspects of creating the product as open source, the authors believe that giving full access to the code allows others to take joint ownership, breaks down

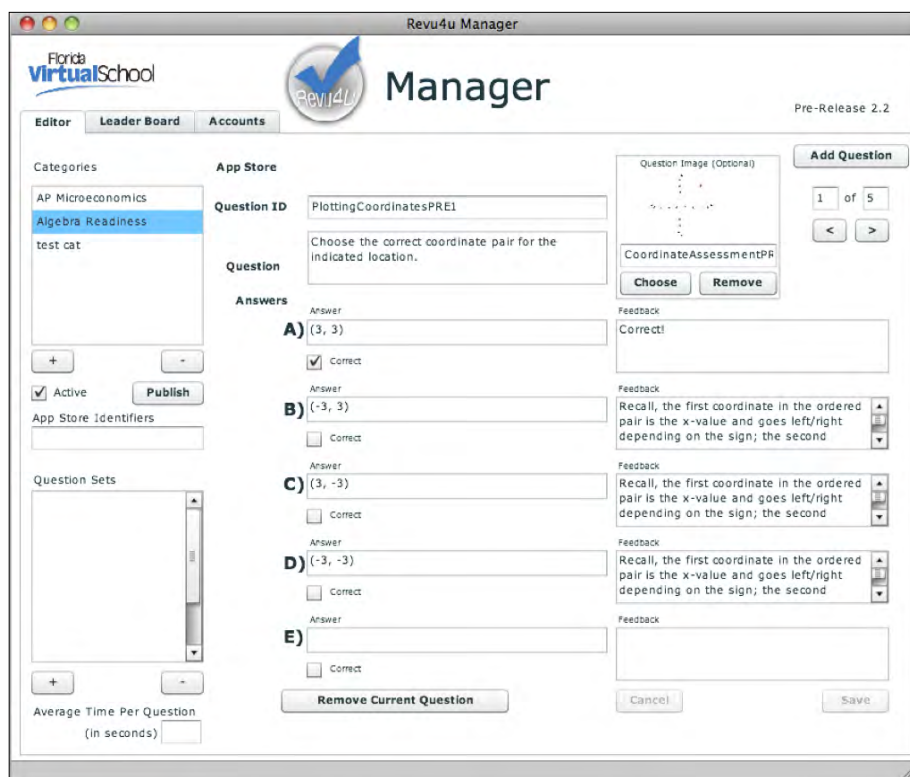


FIGURE 3. A screen capture of the Revu4u multiple-choice authoring tool (precedent work).

some economic barriers for its use, and encourages broader adoption and therefore survival beyond the funding period.

Author once and view on multiple devices

According to the British Broadcasting Corporation, the U.S. DoD is the largest employer in the world with 3.2 million service members and civilian support personnel (BBC News, 2012). This number does not include all of the associated contractors. Training solutions for DoD employees need to scale and work with as many devices as possible. As a part of the contract requirements, the AADLC stated that the content would be stored in such a way that it could be rendered to different devices without having to create the content multiple times. This same requirement was also helpful in meeting MASLO's dual mission intent to support K-20 educational settings.

Offline content access

Another requirement was the need to make all content available on the mobile device itself. Content could be stored on a central server but once downloaded to the mobile device it had to be fully functional without requiring a connection back to the server. The nature of service members' jobs to work in remote areas was a key driver for this requirement, but the AADLC team also saw this need for educational institutions serving remote students in the U.S. and more

specifically in rural Wisconsin (Branon & Wolfenstein, 2013).

Simplified mobile content authoring environment

This requirement was less clearly defined because it was not a technical requirement per se. Both the AADLC and the DoD wanted a way to get content to mobile devices that did not require the subject matter expert, professor, or teacher to have programming experience. Efforts to meet this requirement are described in detail in the rest of the case.

MASLO: Authoring Tool

The development team began the MASLO project by designing the authoring tool. Choosing to start with that part of the three-component system was a *key design decision* driven by work on the previously described Revu4u project. The Revu4u effort was a proof-of-concept to help the client, Florida Virtual School, determine whether the use of mobile technology was feasible. Developing an authoring tool was out of scope for a technological proof-of-concept and all content in Revu4u was hand-coded by programmers. The Revu4u technology worked well enough that the client decided to move immediately from prototype to

Implementation (see Figure 3). The lack of resources to expand the project scope, however, meant that an authoring tool had to be hastily assembled. No user testing was done on the Revu4u authoring tool and only a modicum of quality assurance was completed. Revu4u still exists in the Apple app store but little content was added after the prototype work ended. While no formal Revu4u failure analysis was conducted, anecdotal conversations with the client and some of the users indicated that the incomplete authoring tool was a limitation to widespread adoption.

One key benefit of the Revu4u project was that the technological approach *did* work. This approach included separating content from the mobile application (iPhone only) by storing it on a hosted service (the service was Amazon Web Services). This functioning prototype was sufficient to gain funding through the U.S. Department of Defense for the project which would become MASLO. Since the Academic ADL Co-lab knew the technology architecture worked, we decided to focus the MASLO design effort on the greatest area of weakness. We started by developing the authoring

tool and then worked our way back into the mobile device applications and storage system creation. This decision was further reinforced when the lead mobile programmer departed the lab for an outside position.

The first MASLO authoring tool prototype

The first prototype for a MASLO authoring tool was created by the lead programmer on the project. This interactive prototype was not designed to facilitate user testing but was intended to help the design team grapple with the features and basic concepts for the design. There were long debates about the placement of certain features and the numbers of features the authoring tool should offer. The previously developed Revu4u application was a multiple-choice test preparation tool and the first inclination by the programmer was to make sure that he built an equally robust multiple-choice test tool into MASLO. The first limited prototype for a mobile content authoring tool is shown in Figure 4 with the working name of “MilkShed.” After the MilkShed prototype was created, the team took a step back and decided to take a different approach. There were two situational factors that led to the decision to rethink the design. One was that the lead mobile programmer took a new job (a factor examined further in the next section of this case). The second factor was a realization that this interface, while simple, was already more complicated than the team originally envisioned. An additional impetus to rethink the authoring tool interface arose during a design team meeting. During that meeting, attempts to discuss all of the MilkShed buttons created a sense of confusion about nomenclature and function. The conversation started in that meeting continued for over a week. A key outcome of this discussion was a general concern that if the design team was confused about the overall goals of the system, it would be challenging to keep the product simple for end users.

There was internal conflict about the tension between potential technological capabilities of the system and the need to provide a simple and accessible user experience. Our software developer could see far more possibilities for features in both the authoring tool and the mobile devices, and wanted to ensure that the interface accounted for these potential uses. One team member shared the following blog and video by Stanford professor BJ Fogg (2008): <http://www.behaviormodel.org/ability.html> with the team via email to attempt to bridge some of the divide between possible

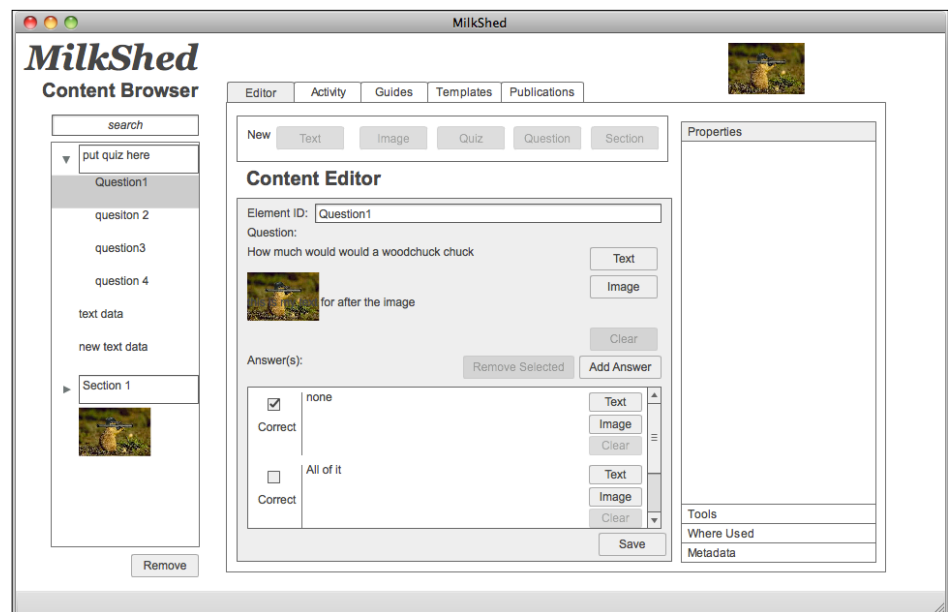


FIGURE 4. The prototype of the first MASLO authoring tool, temporarily codenamed “MilkShed.”

technological capabilities and simplicity. While never formally resolved, the conflict lessened when the original software developer accepted a new position in the private sector.

Employee turnover

Six months before the project began, a new associate director of research joined the AADLC team. His background was primarily design-oriented and significantly less technical. In the initial stages of the project, the senior software developer worked primarily in isolation consulting with the associate director of research and the rest of the team on nomenclature, and periodically presenting the result of his early stage interactive prototype to the other team members. The result was that the senior software developer’s concept of the project was fairly well defined before any other member of the group had an opportunity to weigh in on interface considerations, or general perspectives on the end user.

Just before the official contracted start date, the senior software developer accepted a new position outside the university. The process impact of this turnover was that the timeline of the project had to be lengthened to 18 months through a “no-cost” extension (meaning that the U.S. DoD would allow the longer time but provide no additional funding). The scope and overall architecture of the project remained unchanged, but many design decisions changed significantly. Some of the technical design changes are discussed later in this case.

The loss of the programmer meant a substantial delay in code development, especially for the iOS and Android apps. However, it gave an opportunity for the new associate director to focus on the user interface design of the product rather than the program code. The senior software

developer approached the project under the assumption that front-end interface concerns could be considered after all back-end capabilities were defined. While this perspective on software development was valid, it trended towards the development of a system that would have more capabilities than would be required by the end users.

From the associate director's perspective, leading with expansive system capabilities also ran the risk of pushing the authoring tool towards an interface that supported expert users with a high level of computational fluency, but had too high a bar for many potential users who were less digitally savvy. While it is certainly possible to start with a complex system and then develop an interface that can meet users where they are, this approach also tends to invest significant project resources in developing features that only a small portion of real users actually access. Refocusing the project from a user's perspective meant leading with an analysis of the end users and development of the interface concept first, then working to establish a back-end for the system that could handle the necessary content types in relation to the front-end workflow.

Approximately six months after the lead developer left the lab, a new software developer was hired to continue the project. By that time, the associate director and remaining programmer had established a new workflow and approach to the project. That approach included simplifying the interface and conducting multiple rapid prototypes and usability tests which will be discussed later in this case.

Back to the drawing board—Rethinking the authoring tool audience: teachers, faculty, and trainers, not instructional designers or programmers.

Numerous whiteboard sessions were conducted by the AADLC staff with ideas for the authoring tool interface generated and quickly discarded even before mockups were created for usability testing. Figure 5 shows an interface with tools that would slide off and on screen as they were selected. This version was erased from the whiteboard shortly after it was conceived and never made it to a paper prototype.

MASLO authoring tool: The second concept prototype

The next version of the authoring tool interface was initially sketched up through a series of whiteboard sessions. General usability heuristics were employed to drive rapid iteration across these sessions. Once a reasonable baseline interface concept was achieved, the interface was mocked up in a variety of states using a web-based tool called MockFlow, which allowed the team to extend the rapid prototyping process beyond whiteboarding and related approaches for sketching interface concepts. Learning MockFlow was a bit of a challenge primarily due to its workflow which presented some problems to the team. First released in September of 2009, MockFlow was a relatively new product with some evident limitations. At the time the team was using it in early 2011, it lacked a few capabilities common to comparable software such as quickly moving elements between layers, or easily adding custom graphics to a mock-up. Nonetheless, it was workable for generating early, testable paper prototypes quickly. The result was a second concept prototype built with an understanding that the team would use it as an instrument for usability testing.

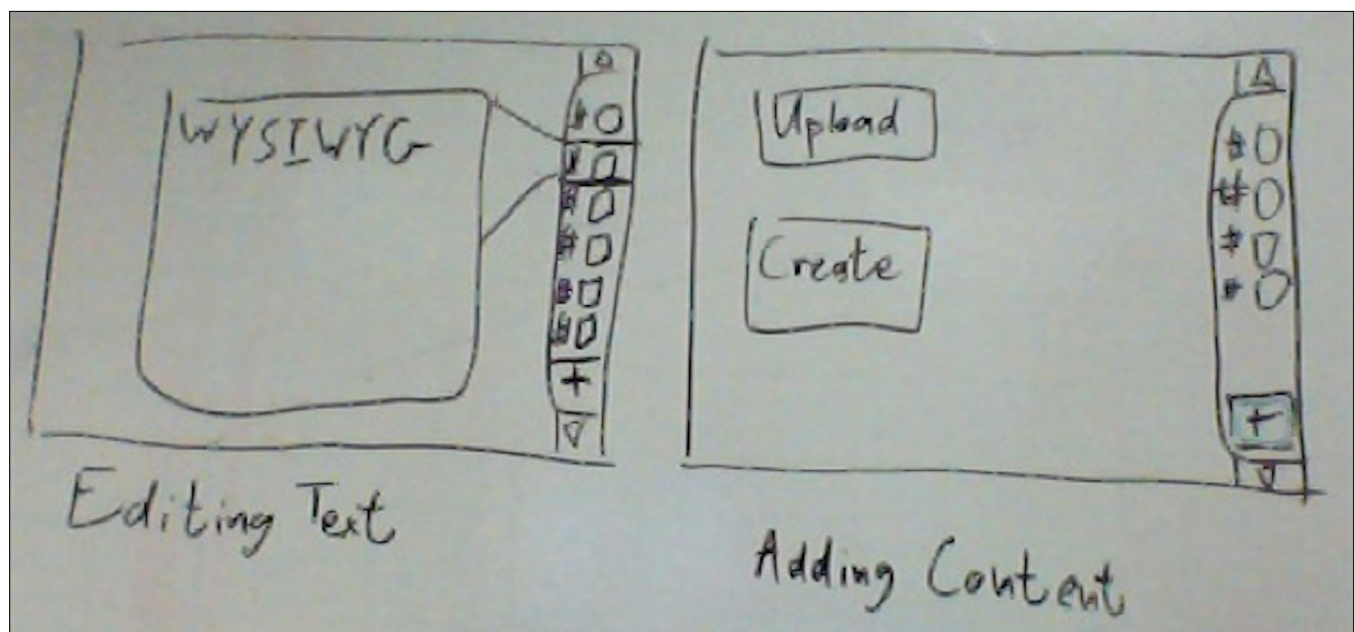


FIGURE 5. Example of an early whiteboard draft version of the authoring tool that never made it to a paper prototype.

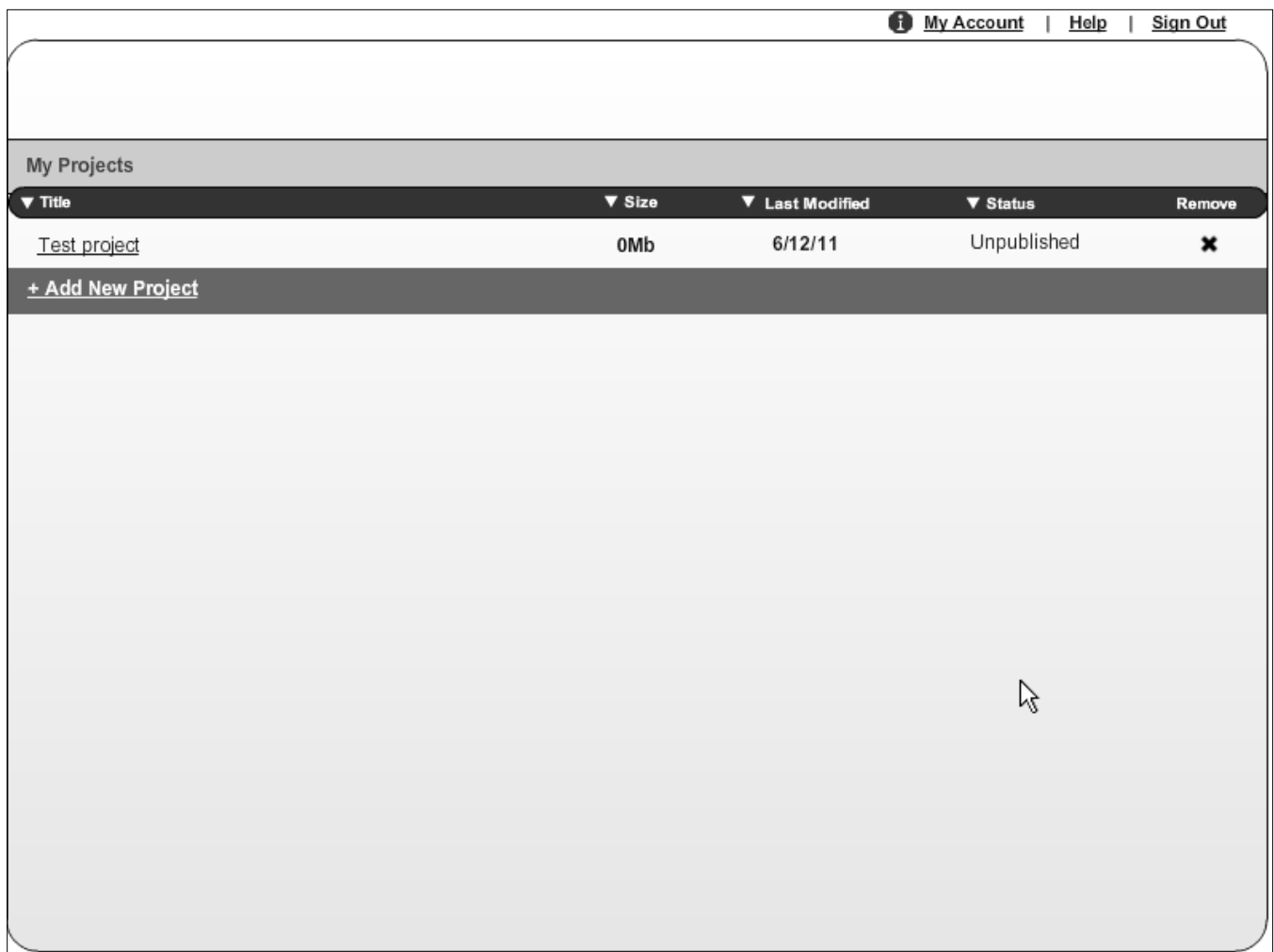


FIGURE 6. This is an early rapid prototype mockup of the authoring tool. The MASLO authoring tool project screen: This is the main screen for initiating a new project. Note the simplification of this interface compared to the original MilkShed concept.

The use of paper prototypes was new for several team members and the value was not immediately clear for all members compared to the use of digital prototypes. Once the first tests began, however, there was a general recognition of how paper prototypes could provide valuable data. Figure 6 shows one of the next-generation MASLO interfaces, which was greatly simplified from the MilkShed version.

Usability testing: Examples of changes large and small over multiple iterations

Figures 7 to 13 are some examples of how paper prototypes and rapid iterative development changed the interface. They represent the progression of the design over the course of the project once the transition was made from initial whiteboard sketching to paper prototype development

using MockFlow. As is evidenced in these examples, design began with a more complex interface that drew on several common features from other applications, and became progressively simplified over the course of several iterations. Users were asked to write on the paper prototypes during testing when it felt natural to do so, and some of their feedback is shown in these examples. The samples selected illustrate how users provided feedback through paper prototype testing. Feedback came in the form of comments written on the prototypes and comments recorded by testers. All of this user input fed into the evolving interface as it progressed towards an increasingly streamlined style with simplified elements.

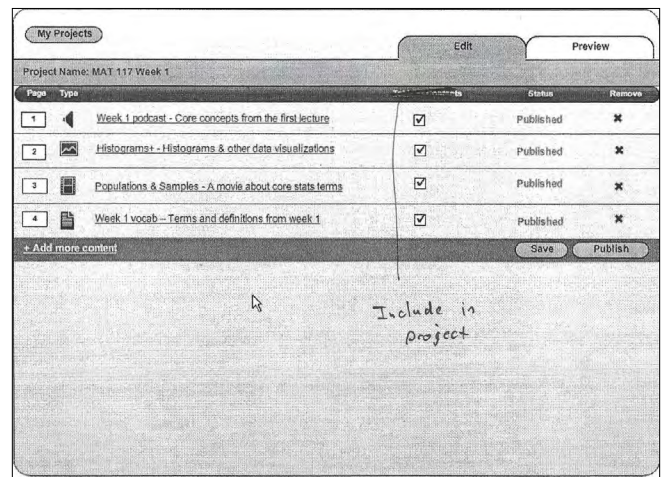
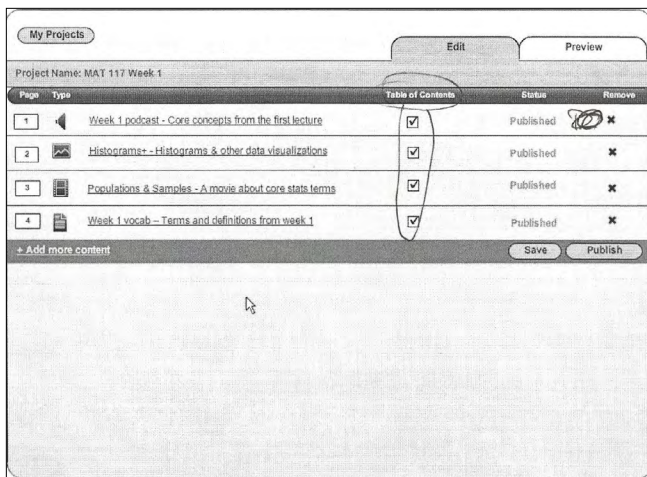


FIGURE 7. Paper prototype data: multiple users indicated confusion over the check boxes that allowed content to show on a table of contents page. This feature was dropped as a direct result of this confusion and the entire application was simplified by eliminating a separate “table of contents” page in favor of a direct listing of all content pieces.

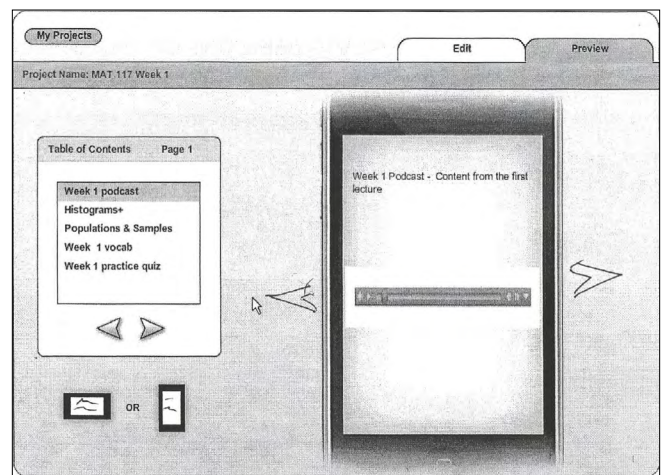
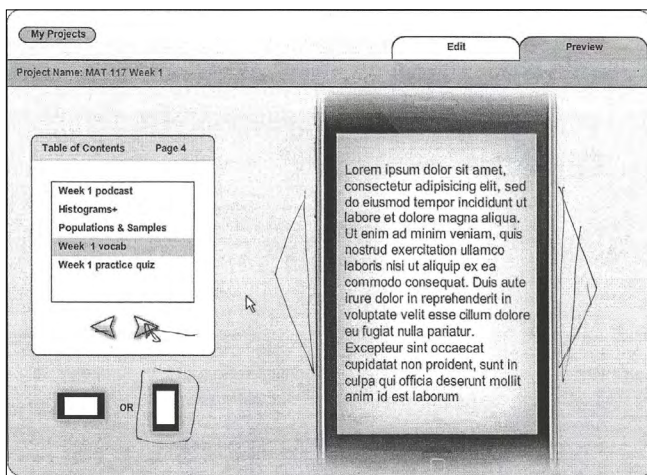


FIGURE 8. In the MASLO preview tool, multiple users drew arrows to indicate that they wanted to simulate swiping from screen to screen, even though there were arrows in the content selection box on the left. While this initially seemed redundant, it was easy to add and improved the user experience. The current version of MASLO features arrows on either side of the preview area.

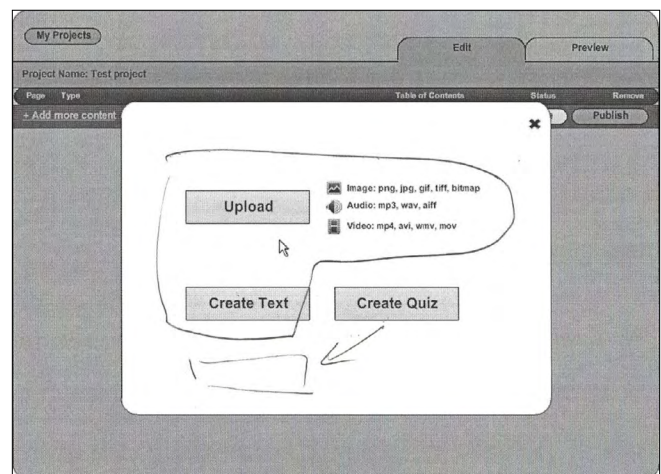
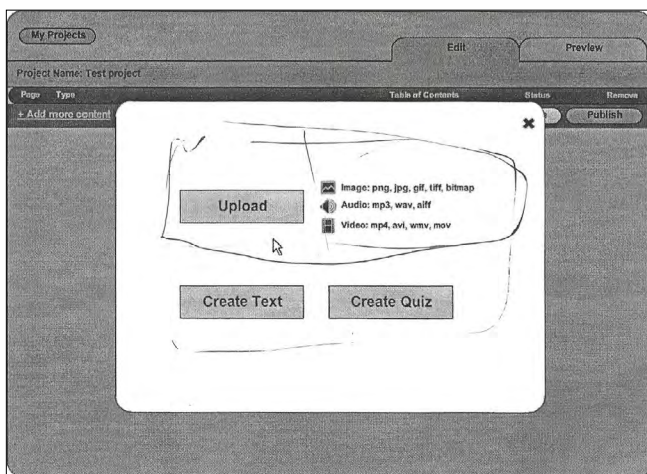


FIGURE 9. Two users noted that the arrangement of this screen was not entirely clear. The user’s markings on the upper right image suggest that simply placing all major content types in vertical alignment would eliminate confusion.

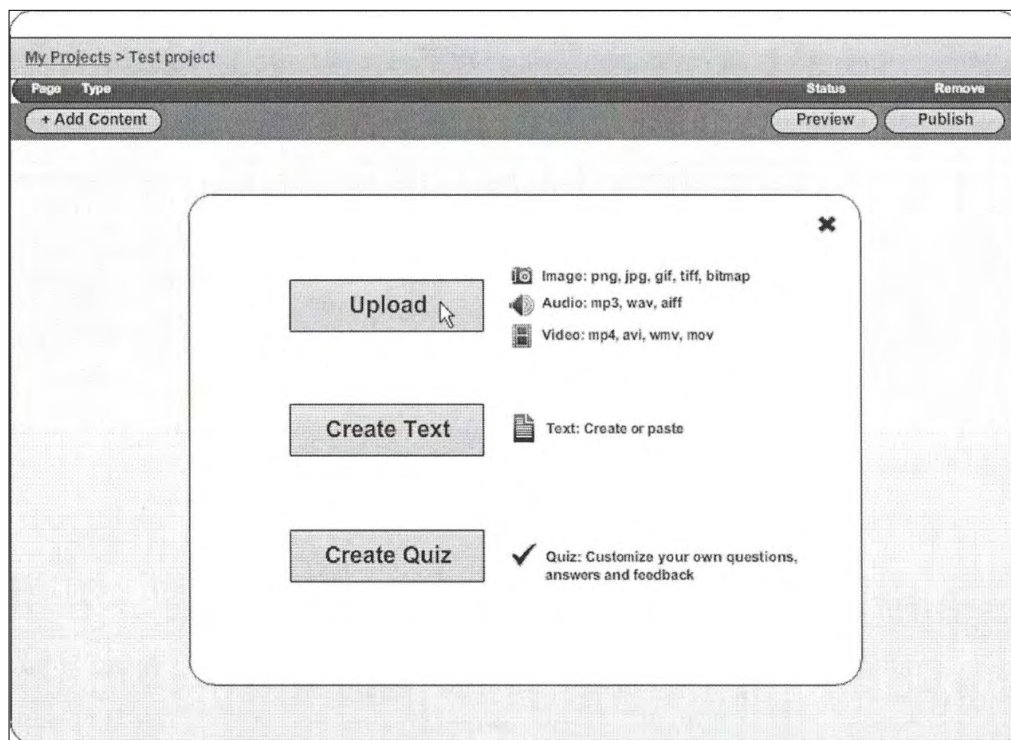


FIGURE 10. A vertical alignment of the three buttons was easy to change at the paper prototype stage.

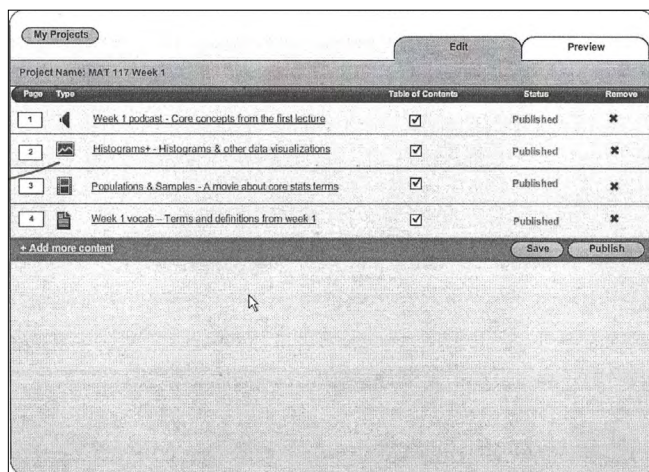


FIGURE 11. A version of the authoring tool that included tabs: Tabs were part of the MilkShed concept and allowed multiple screens to be “hidden” when they were not needed. The simplified version of the authoring tool reduced the number of tabs to two: an edit tab and preview tab (upper right of image). Unfortunately, this placement of the preview tab meant that many users never saw this feature. The last vestiges of the tabbed interface died during user testing and a simple button beneath the working area highlighted this feature in a more direct way (see Figure 12)

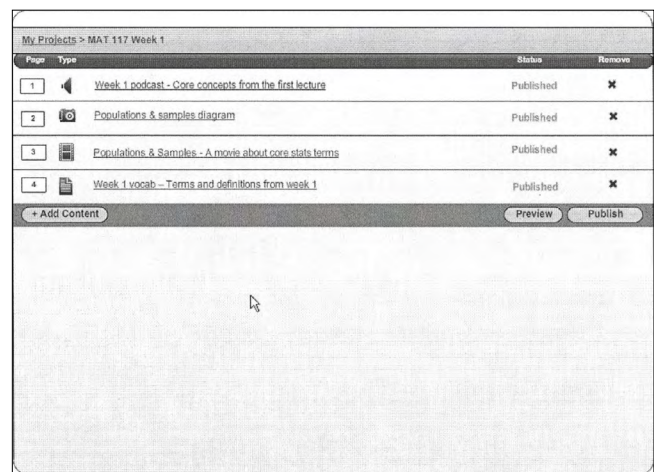


FIGURE 12. Death of tabs: The later version of the authoring tool with tabs removed.

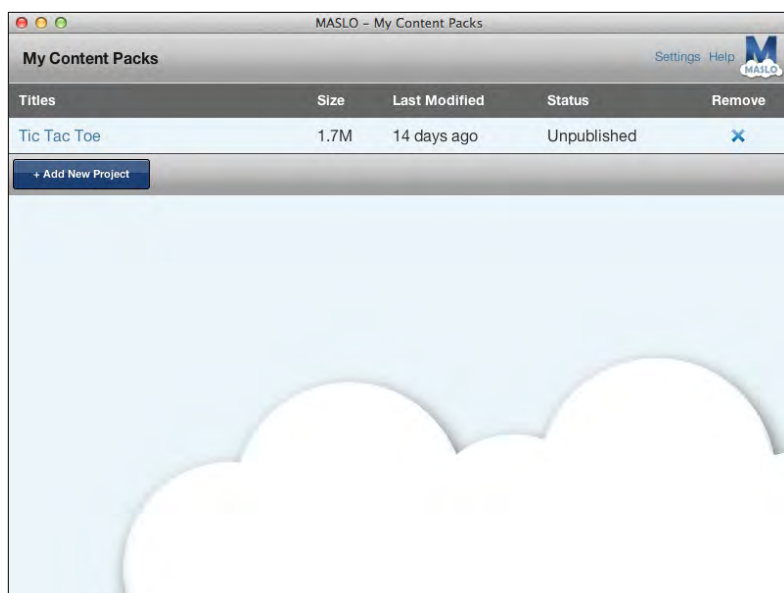


FIGURE 13. A screenshot of the release version of the MASLO authoring tool after all of the testing and code development was completed.

The examples in Figures 7 to 13 were only a handful of the changes made as a result of user testing but they are some of the more explicit cases that influenced development. There were other design decisions that were the subject of internal debates about the extent of control to impose on authors and some of those are discussed in the next section.

User Constraint Versus Guidance

One persistent area that emerged through the project was the degree to which authors should or should not be enabled to make bad design decisions in their content packages. While the design team agreed that it was futile to attempt to “save users from themselves”, they also determined that looking at the system in terms of distributed knowledge meant working to build some elements of best practices into the software.

Content package size

One debate the team had was whether to limit the size of content packages authors could create. At the time, Apple had a hard limit of 20 megabytes for in-application downloads if the phone was not on a Wi-Fi network (i.e., using cellular data transfer). Additionally, the devices themselves had more limited memory to store content. The team debated whether authors should be prevented from creating packages that would be difficult for learners to download in potentially restrictive situations.

The decision was to provide a warning for packages exceeding 20 megabytes but not to set a hard limit for authors. While a hard cap prevents suboptimal designs by novice

content developers, it also restrains authors who might understand this limit, but who need to develop media rich packages. One key factor in our decision is that all packages are designed to operate offline so there is some expectation that people might download large multimedia packages when they have broadband access.

Formatting text

In approaching the design of the WYSIWYG (what you see is what you get) editor, the team made a more or less unanimous decision to limit the array of tools available for formatting text. Despite feedback from some of the test users indicating a desire to have a greater array of text formatting capabilities, the team realized that this was one area in which the benefits of giving more control to advanced users were outweighed by the consequences of giving too much control to users who had less knowledge about designing instruction for mobile screens.

Elements like font, size, and color were not included in the WYSIWYG editor as the potential to create poor experiences using these elements was too great. Elements such as bold and italic styles, for emphasis, and numbered and bulleted lists were retained, as these tools can be essential for creating good instructional content.

Instructional guidance

In early conversations, the team discussed providing instructional design guidance as well as simple content guidance. Merrill (1999) notes that most content development tools provide this kind of structural guidance but no instructional guidance. Wary of this criticism, one original idea was to give authors a series of templates that would guide them in ways to take advantage of mobile devices as they created their packages.

This idea was discarded for two reasons: (a) it was out of scope for the project and programming these templates would have far exceeded the budget; and (b) based on potential constraints imposed by such templates, the team felt that authors might see themselves bound to a particular instructional approach with a tool that was largely intended to provide open options.

The result of this decision is unclear but the current version of the authoring tool tends to lead people to develop extremely linear content (in the very limited uses we have seen so far), even though other types of instruction are possible. This is a feature that might be revisited in future versions of the authoring tool.

Peer Coding

One unique design process that emerged during the course of the authoring tool development was one we called “peer coding.” In Agile software design, pair programming is a term that is sometimes used to describe two programmers working together (Williams & Kessler, 2002). In that definition, the role of the second programmer is often to catch errors while the first programmer is still coding. For this project, the roles were very different. The two “peers” in this case were a designer; the associate director and the programmer. Rather than checking code per se, the designer would ensure, in real time, that the programming work was yielding the appropriate interface outcomes. In addition, the designer functioned as a sounding board for the programmer. The programmer would walk the designer through the code as it had been written up to a point where he was encountering a challenge. The designer would then prompt the programmer with a series of questions from his intelligent novice perspective (Halverson, Wolfenstein, Williams, & Rockman, 2009), and in the process of answering them, the programmer would determine one or more possible solutions to the development problem. The two would then establish the best path forward based on the previously established design goals.

MASLO: The Mobile Applications

The MASLO kit contains the code for an iOS (Apple) application and an Android (Google) application. Interfaces for these two applications are almost identical. Team members referred to these applications as “players” because they allow users to view and interact with learning content but do not permit authoring of content.

Usability testing for the mobile applications

User testing for the mobile players was not as extensive as it was for the authoring tool. The rationale for less testing was functional and pragmatic. The functional reason for less user testing was that the app was far less complicated than the authoring tool. Users could download and view content and make a couple of setting changes but could do little else. Pragmatically, the project was running behind schedule due to employee turnover and there was no time for multiple rounds of testing. Testing was completed on the first mockup and some changes were made.

Much of the focus for the mobile app usability testing was on labels and navigation names. One challenge was to help users understand that they had to download content packs onto their devices before they could access the content. One screen in the app represented all of the possible content available (stored on the server) and another screen showed

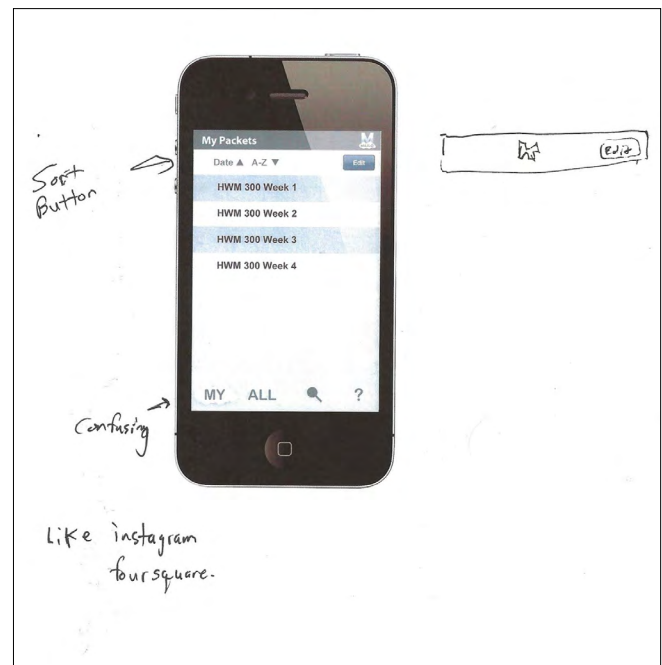
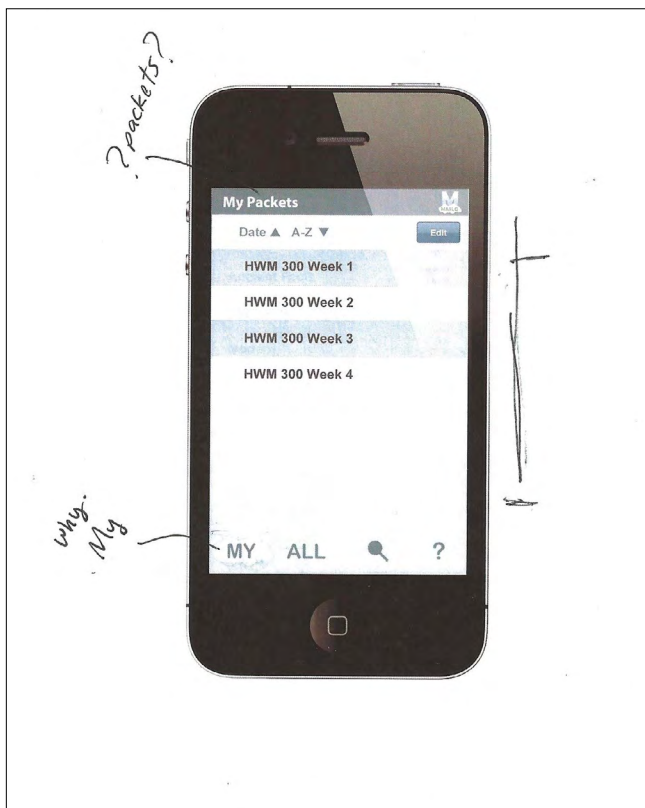


FIGURE 14. These are images of paper prototypes for mobile after user testing. Note the “Why ‘My’?” in the first image and “Confusing” in the second.



FIGURE 15. Examples of the release versions of the MASLO “Home” screen (left) and the “Store” screen (right).

the packages that had been downloaded to the device. Both screens looked the same (lists of content packages) and differentiating between them was difficult. When combined with the limited real estate of a mobile phone, finding the right words became a challenge.

The team debated different terms based on how other mobile applications were addressing this problem. For example, e-book reader apps, such as Amazon’s Kindle or Apple’s iBooks, had similar models (download books and read them on the device). At the time, the word “cloud” was still new and Amazon addressed this through “My Kindle” and “Store.” The team debated terms like “store” and “library” for naming the storage area and finally decided to try “My” and “All.”

Using the simple terms “My” and “All” as shorthand for content downloaded to “my” device or “all” content available on the cloud proved to be confusing to the usability testers (see Figure 14). After asking users what they would understand, the team settled on “Home” and “Store.” The app opens to the home screen and, if empty, this page contains a message directing users to the store screen to download content (see Figure 15). Given the impending deadline to complete the project, the team felt this was a reasonable solution.

Controls on the home screen are minimal and only include the ability to sort by title and to edit or delete content packs. The store screen includes a button next to each pack with four different possible states: (a) Install, indicating that

content is not downloaded; (b) Installed, indicating that content is already on the device; (c) Updated, indicating that the pack is installed on the phone but the content in that pack has been changed since it was downloaded to the device; and (d) Price (e.g., \$2.99), indicating that the pack is premium content and that the user will be charged to download that content to the device.

In addition to these and other user interface issues, the team had to make a number of critical decisions about how to support at least two mobile operating systems and leave open the possibility that other systems could be supported in the future.

Native Applications Versus Web Applications for MASLO

In the world of mobile application development, there are a couple of basic choices in terms of how an app is created. One is to use the programming language of the proprietary software systems (i.e., native apps), and the other is to develop “web apps” or mobile-enabled websites with programming sophisticated enough that they can handle more complex functions. There are pros and cons to each approach. Native development requires creating different application code for each device while a web app will generally run on any smartphone that supports web browsing. Web applications, however, only support limited



FIGURE 16. Examples of two types of basic MASLO content. An edited text page in a MASLO pack (left) and a sample quiz (right).

functions that many smartphones can deliver, especially when operating offline, and so the possible feature set of an application is reduced.

The types of content and display functions of MASLO are simple and could be handled through a web app (see Figure 16). One of the key factors that led the team to select native application development over web application development is the commitment to deliver the content when the phone or mobile device is offline. While web applications do allow caching (storing web content offline), iPhone limits this cache in such a way that larger content packages would not be able to download or run on the device. At the time MASLO was in the latter stages of development, this web application cache limit was 20 megabytes. Exceeding 20 megabytes was likely for any package using video and would limit many potential instructional uses. In addition, since caching did not appear to be an option during the earlier stages of development the file structure was not developed to work with a solution that leveraged caching content. As such, creating a web app version of MASLO would not have simply meant modifying the existing software to create a third version of the player. It would have meant a major overhaul of data structure and the authoring tool. Nonetheless, the team discussed this option thoroughly before discarding it, because if it had been possible to create a web app version of the player at low cost it could have extended the reach of MASLO to include learners who did not have smartphones.

Using PhoneGap

One way the software developer decided to manage the complexity of keeping applications for two different platforms (iOS and Android) in alignment was by using a tool called PhoneGap. While some back-end elements for the two MASLO applications ultimately needed additional native code for each respective platform, PhoneGap helped to keep the core code base for front-end and back-end interface the same. The main goal was to keep software maintenance for subsequent mobile operating system upgrades as convenient as possible. PhoneGap, however, did not remain as backwards compatible as developers had initially hoped. Rapid release cycles with numerous adaptations of new features and bug fixes regularly required significant changes in the PhoneGap-interfacing code in addition to necessary changes to keep up with API (application programming interface) updates in the native code for Android and iOS. In hindsight, it is questionable whether the decision to use PhoneGap versus going entirely native truly saved development and maintenance time.

PhoneGap was originally developed by the independent shop Nitobi Software. However, Nitobi and PhoneGap were acquired by Adobe in late 2011 while MASLO was in an active stage of development. While this has ultimately led to a broader scale of support for PhoneGap, it was an

unanticipated development for the MASLO team. It is ultimately unclear what effect, if any, the Adobe acquisition had on support for PhoneGap while MASLO was in development. But as with the significant changes to mobile operating systems that took place over the course of the project, it draws attention to the dynamic and, at times, unstable nature of mobile technology development over the course of the project.

MASLO: Cloud Storage

When the new software developer joined the project, she came into a situation in which a number of design decisions had already been made. While many conceptual elements were in place, there were many details that were not determined. One of the significant components that had yet to be specified was the cloud storage for content packages. The team assumed that the previous Revu4u project provided the groundwork for this part of the project, but numerous design changes meant that the cloud storage had to be engineered as if it were a new project.

Amazon

In the precedent Revu4u application, AADLC developers used Amazon's schema-less database system, SimpleDB, to store all data in small, but flexible increments (Branon, Wolfenstein, & Raasch, 2012). The reasons for this decision resulted from the specific requirements of that project. For MASLO, however, the database aspect of Amazon's services added complexity, especially in relation to the file structure and the potential size of MASLO content packages. The new software developer on the team believed that using Amazon web services would be sufficient for the needs of MASLO and would simplify development.

Security

The design team had given little thought to the security features required for managing the content storage. While security or authentication is not necessarily needed for providing free content to end users, it can become an issue for content deployment. In general, it is rarely desirable to give upload access to just everybody because this can lead to a number of problems, ranging from simply providing too much content, to creating workflow challenges, to providing content containing malicious or illegal material. Therefore, the cloud storage component required at least a basic upload limitation feature.

Limiting content provision can be done in multiple ways. One way is to designate a central administrator who has to review and approve every uploaded content piece before it is allowed to go "live". This method, however, potentially requires a lot of attention and in-depth knowledge, from the administrator, about which content is appropriate. A different way to limit content provision, and the method

MASLO implemented, is to maintain a database of authorized users. Before users can upload content, they have to authenticate with the cloud storage system. A central administrator still has to add users to the system but once a content author is in the system they can freely upload content, which can be provided to end users right away.

Administrative interface

By the time the Amazon components were being constructed, there was no time for usability testing of any of the administrative interfaces. Given that the first apps were likely to be developed by the AADLC, and the clock was ticking, these interfaces were an afterthought. While the functions were simple, there was no time to ensure all the elements were easy to follow. The team realized that anyone working on the administrative side of MASLO would need a technical background. The authoring tool and the applications had to be extremely simple because non-technical users were the audience. In the case of setting up cloud storage, the audience would be someone with the experience to at least set up and manage a web server.

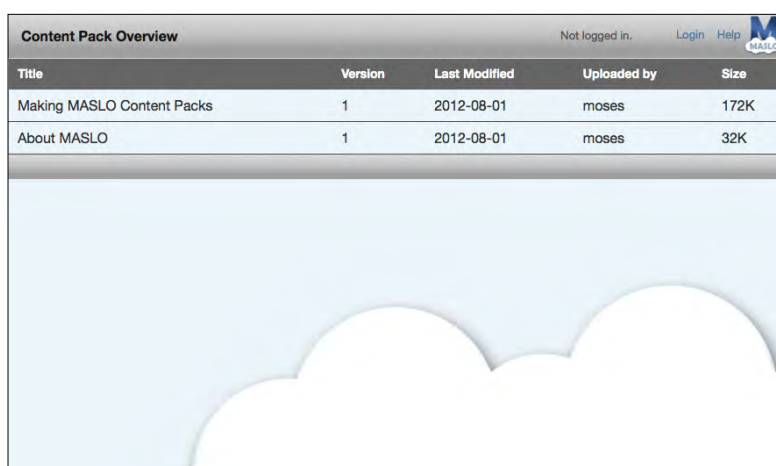
Since the cloud administrator panel was an afterthought, the initial version was completely based on terminal scripts without any graphical user interface. To allow test content authors to see their uploaded content while the mobile client was still under submission with the Apple App Store, a basic web front-end was added, whose appearance very closely resembled the look and feel of the MASLO authoring tool. The initial front-end allowed users to see an overview of the uploaded content packs, to log in, and for authenticated users to delete content packs (see Figures 17 and 18).

DISCUSSION

The primary goal for the MASLO design effort was to build a fully functional prototype mobile learning authoring tool and delivery platform.

Outcomes and Relevance

Two known apps exist that are built with the MASLO platform (the open source nature of the project means that others are possible). One is the prototype MASLO Setup Guide application. The first prototype application was created with the MASLO kit as a proof of concept but also as an instructional tool about the MASLO kit itself. It can be found in the Apple iOS app store with the decidedly unappealing but literal name “MASLO Setup Guide.” This guide has seen



Title	Version	Last Modified	Uploaded by	Size
Making MASLO Content Packs	1	2012-08-01	moses	172K
About MASLO	1	2012-08-01	moses	32K

FIGURE 17. A screen capture of the cloud storage administrator interface.

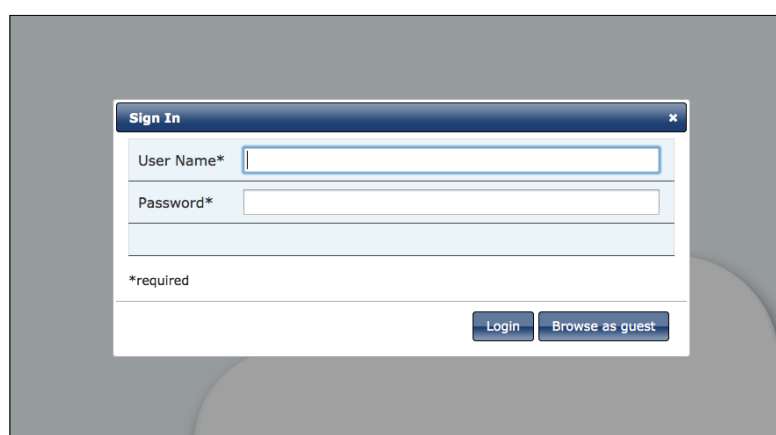


FIGURE 18. A screen capture of the administrative login interface. FIGURE 18. A screen capture of the administrative login interface.

limited downloads and is of use primarily to developers looking to implement the application.

The second application represents a new fork in the MASLO code that allows multiple content stores to exist within a single app. Explanation of the changes in MASLO go beyond this design case and may form the basis for a future case. It is mentioned here to show that the MASLO code is still in use as of the writing of this case. The name of that application is the University of Wisconsin-Extension Learning Portal (search the app store for UWEX LP). Originally, the AADLC was developing two applications for educational projects at UW-Extension using the platform. One of these was a labor education application with a faculty member at the UW-Extension School for Workers. The faculty member developing the content for that application is an expert in migrant farmworker issues and plans to distribute it through a national labor organization serving more than two million workers. The other application was called “History on the Bay” with a UW-Extension Cooperative Extension professor (B. H. Huff, personal communication, January 19, 2012). Because multiple faculty members wanted these applications, a

version of the MASLO mobile app was created to accommodate multiple authors.

In 2012, there were other indications that the MASLO code was being used beyond our projects. All of the code is still available for download for free from the Academic ADL Co-Lab website (<http://www.academiccolab.org>). An email distribution list was also made available for those interested in following the continued development of the MASLO platform. One hundred and fifty people signed up to be notified of updates to the platform but the email list was ended in 2014. The team did receive email from several developers across the U.S. with questions. One developer in Hungary submitted bug reports to the GitHub site for the project (a repository for open source software projects). We do not require people to notify us about MASLO uses, so it is not clear how many of these projects are moving forward or what these people are doing with the software, but it is an indication that it was explored by a number of people.

Despite these MASLO use cases and early interest, the team is continuously reminded of the challenges that non-technical application creators face in the mobile space. While the MASLO authoring tool and mobile interface went through extensive testing and have proven easy to use, creating an app, setting up server storage, and managing the constant technical changes to mobile platforms still requires deep technical skills. When presenting on MASLO or having discussions, team members have repeatedly heard that this technical knowledge requirement remains a barrier for more widespread MASLO adoption. The team discussed the potential to streamline the entire MASLO application creation component, but the constantly changing technical requirements of the space mean that such a tool set would be cost prohibitive. The team investigated one solution of offering an “open source business model” in which the tools were free for those with technical skills, but that plan did not come to fruition.

Design Reflections

While MASLO is a mobile learning platform and not a specific instructional intervention and the design decisions are specific to this instance, it represents systems design in the emerging space of mobile learning technology. Because this is work within a rapidly evolving area, this design case is deeply situated in the time in which the work occurred. Changes to mobile devices, their operating systems, massively distributed (i.e., cloud) computing services, contractual language, and terms of service agreements mean that some of the decisions in this case will appear dated much more quickly than might be true in other design cases. Despite the imminent loss of relevance for some choices and even because of it, the team felt that capturing this project’s design decisions creates precedent knowledge beneficial to the field. Three of the major decisions the authors faced

when developing MASLO are summarized in the following sections.

Design decision reflection: MASLO architecture (Computer authoring, mobile delivery, and cloud storage)

When MASLO was first proposed in 2010, the idea of using separate servers to store content was only beginning to get some traction. Just two years later, in 2012, such offerings were common and, at the time of the final revision of this design case in 2015, may even seem quaint to many software designers. The ability to store content separately from a mobile application is customary and ever more sophisticated. In 2010, the Android and iOS application environments were far less compatible. While substantive differences remain, the ecosystem of developer tools allowing easier cross-platform compatibility has rapidly grown. Both are still valid considerations but the technical barriers are far lower in 2015.

What has changed dramatically and was even changing while the project was actively forming was the decision to limit content authoring to desktop or laptop computers. The decision to use Adobe AIR meant that content authoring could be done from either an Apple Mac or a Windows-based computer, but mobile devices (tablets or phones) could not manage the file storage or file formats in an effective way. Users wanted that functionality but the limitations of 2011 mobile operating systems did not make that easily doable. Of all the design decisions the team made in 2011-2012, those related to architecture are the most noticeably dated just a couple of years later.

Design decision reflection: Access versus parsimony

One of the most critical and ongoing set of design decisions was related to the tension between maximizing the capabilities of technology and the desire to create software with a low technical knowledge requirement. The first software developer was a computer scientist with an advanced knowledge of software design. He could see possibilities and capabilities for MASLO that the team and users could not envision. The project owner, coming from a more user-centered design background wanted to make sure that the software was not so complex that it was unusable by the target audiences. The initial tension was resolved when the first software developer left the university for a private sector position.

While the early technical work remained, the team used the time between the departure of the first developer and the hiring of the second to conduct interface user-testing with paper prototypes. The tension emerged in several other aspects of the design. For example, the team had to decide how much to prevent users from uploading content that might create a “bad” mobile experience for their learners. This meant choosing whether to restrict file types and file sizes so

that only optimal files could be uploaded. One example of compromise was to restrict video files to those which would run on both iOS and Android but not to restrict file size. Instead, a warning would pop up to let users know that the size of the file might impact learner experience. Part of the rationale in making these decisions was to stop users when a particular action would not work at all (in the case of file formats), and allow actions that might be detrimental but could ultimately work (such as very large file sizes).

Even the word “access” became a loaded term for debate. For example, making the entire project open source meant that no educator would be prevented from experimenting due to burdensome licensing costs, but that also placed a higher burden of technical knowledge on the user. Access to extra editing features for more technical users was generally limited to preserve access for non-technical users. The team used a principle of parsimony over feature capability but one user’s simplicity was another user’s barrier to create the content they wanted. Parsimony was a constant part of the conversation and one that was only partially resolved and even then it was often resolved due to other project constraints (i.e., employee turnover, time/budgetary requirements).

Design decision reflection: Open source

The decision to make the application and code open for modification and non-commercial use was both a contractual requirement and a design decision to improve access. The intent was to keep the costs low and have a thriving community to take over the project after the initial grant funding ended. At the time, the U.S. Department of Defense was placing a higher value on open source projects so that they could use and modify code without seeking additional licensing expenses. Like so many open software projects, however, the critical mass of developers needed to sustain such an effort did not materialize.

While the code is still publicly available, no known work outside the team has materialized. As more time passes, it appears that the code is withering because the original team has disbanded and moved to new endeavors. One reflection is that open source is ultimately dependent on the creation of a strong support community. The project budget and constraints on the development team did not take into account the need to develop such a strong community.

CONCLUSION AND A POSTSCRIPT ON THE FUTURE OF MASLO

This design case represents a moment captured in time up to the 2012 release of the first iteration of the MASLO platform. As of 2015, the MASLO team has disbanded and moved on to other professional endeavors. The code remains available for modification and use but, without almost continuous updating and upgrading, the platform has failed

to keep pace with the times. The projects mentioned at the beginning of the discussion section have mostly stagnated or chose to use newer software. The authors believe MASLO remains a worthy design case because the challenges of using emerging technologies are part of educational technology’s past and future.

Designers creating software platforms will all contend with shifting standards, changing platforms, and rapidly evolving user needs. Forging a path through a foggy future and making design decisions while changes are happening are likely the norm for future educational software designers. It is the authors’ hope that the decisions made in this specific case provide designers with additional knowledge they can pull apart and recombine to create the next generations of software.

ACKNOWLEDGEMENTS

The MASLO project described in this design case was funded through a Broad Agency Announcement grant (W91CRB-08-R-007, 2010) through the United States Department of Defense.

REFERENCES

- BBC News. (2012, March 20). Which is the world’s biggest employer? Retrieved from <http://www.bbc.com/news/magazine-17429786>
- Branon, R., & Wolfenstein, M. (2013). MASLO: An open source platform for mobile development. In W. Kinuthia & S. Marshall (Eds.), *On the move: Mobile learning for development* (pp. 69-84). Charlotte, NC: Information Age Publishing.
- Branon, R., Wolfenstein, M., & Raasch, C. (2012, February). iOS® and Amazon SimpleDB: Connecting mobile learning to the cloud. *eLearn Magazine*. Retrieved from <http://elearnmag.acm.org/archive.cfm?aid=2151676>
- Carroll, J.M. (1990). *The Nurnberg funnel: Designing minimalist instruction for practical computer skill*. Cambridge, MA: MIT Press.
- Halverson, R., Wolfenstein, M., Williams, C., & Rockman, C. (2009). Remembering math: The design of digital learning objects to spark professional learning. *E-Learning and Digital Media* 6(1), 97-118.
- Maeda, J. (2006). *The laws of simplicity: Design, technology, business, life*. Cambridge, MA: MIT Press.
- Merrill, D. M. (1999). Instructional transaction theory (ITT): Instructional design based on knowledge objects. In C. M. Reigeluth (Ed.), *Instructional-design theories and models: A new paradigm of instructional theory* (Vol. 2, pp. 397-424). New York, NY: Lawrence Erlbaum.
- Nielsen, J. (1999). *Designing web usability: The practice of simplicity*. Indianapolis, IN: New Riders Publishing.
- Norman, D. (1988). *The psychology of everyday things*. New York, NY: Basic Books.
- VideoLAN legal concerns. (2015, April 15). Retrieved from <http://www.videolan.org/legal.html>
- Williams, L., & Kessler, R. (2002). *Pair programming illuminated*. Boston, MA: Addison-Wesley Longman Publishing.