Design, Development, and Implementation of the Instructional Module Development System (IMODS)
Division: NSF Grantees Poster Session

Abstract

There is a growing demand and interest in faculty professional development in areas such as outcome-based education (OBE), curriculum design, and pedagogical and assessment strategies. In response to this demand, a number of universities have established teaching and learning centers to provide institution-wide, and sometimes program specific support. A team of researchers is engaged in a User-Centered Design (UCD) approach to develop the Instructional Module Development System (IMODS), a software program that facilitates course design. IMODS will be an open-source web-based tool that will guide individual or collaborating STEM educators, step-by-step, through an outcome-based education process as they define learning objectives, select content to be covered, develop an instruction and assessment plan, and define the learning environment and context for their course(s). It will contain a repository of current best pedagogical and assessment practices, and based on selections the user makes when defining the learning objectives of the course, IMODS will determine and present options for assessment and instruction that align with the type/level of student learning desired. To this end, the project addresses the following two research goals:

1. Develop the instructional module development system (IMODS) using user-centered design methodology
2. Assess the effectiveness, efficiency, and usability of IMODS in creating outcome-based course design

This paper describes our efforts in the High-level (Conceptual) Design phase of the UCD methodology. This phase follows the collection of data from potential users on what they currently do and will need in the future. The main goal of high-level design is to create an early blueprint of the system. We have identified 2 tools that will be most suitable for this phase of the project: Navigation Model and Prototyping. The navigation model will illustrate how all user interface screens should be connected. Ideally this should reflect the user's mental model to facilitate intuitive navigation between screens to accomplish the task of instructional design. A high-fidelity prototype is currently under development that will provide details of all key screens and a number of auxiliary screens with appropriate navigation between them. The expected outcome of the high-level design phase is a validated prototype of IMODS. This poster describes the design and development of the prototype that will be used in usability testing of the software to elicit feedback from potential users on the effectiveness, efficiency, and usability of IMODS for outcome-based course design.

1. Introduction

Most science, technology, engineering, and mathematics (STEM) educators have little training on developing a course based on factors such as learning objectives of the course, the time frame in which the objectives are to be met, the content to be covered in the course, instructional and assessment techniques to be used. There are tools in the market that enable instructors to build courses based on some of these factors, for example, scheduling classes, managing learning content, etc. However, there is no tool that takes into consideration the complexity of educating students effectively based on a set of objectives, corresponding assessment and instructional

techniques, and given a set of resources. There is a growing demand and interest in faculty professional development in areas such as outcome-based education [1], curriculum design, and pedagogical and assessment strategies. IMODS aims to solve this complex problem and provide instructors an easy-to-use software interface that will allow them to design their courses.

The IMODS is an open-source web-based course design software that:

- Guides individual or collaborating users, step-by-step, through an outcome-based education process as they define learning objectives, select content to be covered, develop an instruction and assessment plan, and define the learning environment and context for their course(s).
- Contains a repository of current best pedagogical and assessment practices, and based on selections the user makes when defining the learning objectives of the course, the system will present options for assessment and instruction that align with the type/level of student learning desired.
- Generates documentation of course designs. In the same manner that an architect's blueprint articulates the plans for a structure, the IMOD course design documentation will present an unequivocal statement as to what to expect when the course is delivered.
- Provides just-in-time help to the user. The system will provide explanations to the user on how to perform course design tasks efficiently and accurately. When the user explores a given functionality, related explanations will be made available.
- Provides feedback to the user on the fidelity of the course design. This will be assessed in terms of the cohesiveness of the alignment of the course design components (i.e., content, assessment, and pedagogy) around the defined course objectives.

IMODS is currently being developed using a user-centered, as opposed to technology focused, methodology. This approach is well suited for the project given the high cognitive nature of outcome-based course design tasks, and the high levels of interactions required between the user and the system to not only facilitate the development of course designs, but to help users build an enduring foundation of knowledge, skills and habits of mind about curriculum development.

2. User-Centered Design Methodology

User-Centered Design (UCD) methodology is being used for the development of IMODS. UCD is a systematic approach that is typically divided into 5 main phases.

- Phase 1 – User Research
- Phase 2 – High-level design
- Phase 3 – Detailed design
- Phase 4 – Development and development support
- Phase 5 – Testing and Installation support

User centered design focuses on software development using a top-down holistic approach. Traditional software design methodologies focus on a typical requirements-design-development approach in which user interface and user interactions are deferred to later stages and are typically added near the end or in a parallel track to the development process. While this approach allows us to tackle large and complex software applications to the completion point, they often result in less usable applications. One of the main reasons is that the internal structure

and organization of software features is determined by the developers. While this might not be a visible problem in simpler software applications with few features – the type of applications we develop in small class projects, or simple mobile apps-, it becomes a significant concern with larger, more complex applications that involves hundreds of features and complex tasks to execute. From a user perspective, an application is only what it offers as its user interface (UI). Without any prior knowledge of the actual application's internal structure, the user attempts to build a mental model of what the application might look like using the available interface. A perfect user mental model is an accurate representation of what the application actually is; an ideal, but rare case. In reality, the user mental model has some resemblance of the actual application model, and it gets better with continuous use, marking the difference between a beginner and an expert user.

The build up of experience is gained initially with the user's initial attempts to interact with the application. This interaction is typically done by the user trying to execute specific tasks for their daily work. The execution of those end-to-end tasks can be mathematically represented using graph theory where the nodes represent task steps extracted from the use cases, and the edges represent transition between those steps. A complete task will correspond to an end-to-end path on the graph. The graph is considered to represent a navigation model, and should be easy to cross, or navigate" by the user without prior knowledge. In traditional software development approaches, this navigation structure is often ignored or only partially addressed, and ends up as being a by-product of the application. In this case, the navigation model is "inferred" by the user upon initial use attempts, and is used as the underlying core component of the user mental model.

If this inferred navigation structure corresponds to the actual application structure, we tend to call this application "intuitive to use". The user ends up liking the application and having better experience in using it. A "user experience" term is often used to mark this pleasant interaction. If the user mental model, or the navigation structure of it to be specific, does not correspond to the actual internal structure of the application, a discrepancy between the user mental model and the actual application emerges. In this case, the user will have harder time in attempting to use the application to complete their daily tasks, and will need to go deeper into understanding the actual internal structure of the application via training sessions, experts- or help support. This will work as an additional source of information for the user besides their interaction with the application UI, and will help them improve their mental model to closely match the actual internal structure of the application. While this will help build a correct mental model, or remove any discrepancy of an existing one, it could take longer time and requires some training, adding to the cost of using the software. This is often referred to as a less-intuitive application, and is marked by a lower user experience.

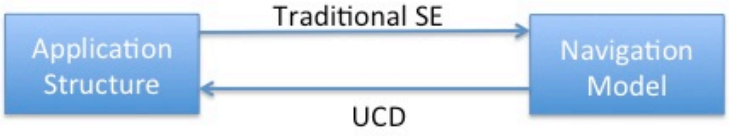

Figure 1: Traditional SE vs UCD

The main goal of UCD is to start by building a user navigation model first using multiple UCD tools and techniques. The second major step will be to use this navigation model to define and design the application structure using multiple prototyping techniques (Figure 1). As we can see, this explains the main departure from the traditional software development where

we start with application structure.
The main focus of UCD is on different starting points:

1) Who the users are
2) What the users currently do
3) What the users want from the new application

Several tools have been devised by the UCD methodologists to execute those initial steps. Tools in general allow us to manage more complex amount of information, giving us leverage over complexity. The first step of identifying the users start by meeting with different people affiliated with the education domain. While the initial and main focus is on teachers, two main aspects need to be considered:

i. Besides teachers, other people could be involved, for example administrative workers, students, and technical support. Those other types of users are researched and identified as "User Roles".
ii. Even the main category of users (teachers) need to be looked at in more details. In a typical situation, teachers have different backgrounds, different experience, and different context to teach in. These differences are looked at using "User Personas" as a common UCD tool that is growing in popularity.

The second part, "what the users currently do" is done via different UCD tools. We followed "Brainstorming sessions" and met with several users to ask them about their daily work practices. Besides brainstorming, we also had a series of interviews and questionnaires to supplement our work. The collected work was categorized and analyzed. The third part, "what the users want" is being done with deeper analysis on the findings extracted in the previous part. This is where the navigation modeling is used as an innovative technique to envision the user needs in terms of a navigation structure that can be translated into an actual application structure.

After successful completion of the User Research phase, we worked on the high-level and detailed design phases of the project. We are currently in Phase 4 of the project that involves software development. The following sections describe our efforts in the various phases of UCD.

3. Phase 1 – User Research

During the User Research phase of the project we conducted 3 focus groups with 5 engineering and computing systems faculty members in each session. The aim of these sessions was to understand the course design process used by the participants. At the beginning of each session all participants were asked to fill an electronic background survey that collected demographic information, primary areas of interest in teaching and research, time spent on teaching, number of courses taught per year (at both undergraduate and graduate levels), and number of new courses developed (both at undergraduate and graduate levels). Participants were also asked to fill an electronic questionnaire about curriculum design tools that they currently use to create and manage their courses (e.g. preparing syllabi; communicating with students; developing teaching materials; preparing, assigning, and delivering grades, etc.). The results of this phase were published in ASEE 2014 and FIE 2014 [2], [3]. Based on this research, a list of 10 most commonly used tools were identified. Data collected from the focus groups about course design process was categorized into inputs, processing and decision-making, and output artifacts. Consolidated data from the 3 focus groups that were conducted was presented in ASEE 2014 [2].

4. Phase 2 – High-level Design

After the user research provided a relatively clear idea and understanding of domain- and user

needs, this initial design phase provides a high-level design with concepts identification, conceptual modeling and early prototyping. The main goal of high-level design is to plot down schematic ideas and steps into visual graphs and models; an early blueprint. We started by investigating different options and provide design alternatives to make sure we have a broad view before identifying a good design. Doing this early on, at high-level, sketchy, paper-based only, and without going into details help provide several solution alternatives at a very low cost. We then chose between multiple good ideas instead of focusing on only one early on. The high-level design sketches were discussed with the users to make sure what they said in unstructured dialogs and vague ideas and imaginations can now be concretely captured in design artifacts for further validation and clarifications [4, p. 220]. We have identified 2 tools that are most suitable for this project in this phase.

***Tools**:*

a) **Navigation Model** is one of the essential methods of design that we used. A significant challenge in complex software is not the contents of each screen, but how the user mentally build a mental view of how all screens are connected (like a city road map), and how to navigate between hundreds of screens to accomplish their task. In this regard, we have developed an effective technique, elastic prototyping, an implementation of a participatory design to help designers and users build a navigation model together, greatly reducing time and effort needed. Figure 2 shows the navigation model for the primary application.

b) **Prototyping (PT)** is extensively used in UCD to visualize and validate all otherwise vague ideas and unclear expectations at low cost and high effectiveness. We focused on three main categories of prototyping: Paper (low-level) PT, low-fidelity electronic (medium level) PT, and high-fidelity, detailed PT [5, p. 188]. Paper prototypes are very inexpensive and help us capture several initial ideas and concepts, and validate them. After explaining their needs, users often change their minds when they see them on paper. Therefore multiple paper PT sessions gives a head start in validating what users actually mean and need. After initial concepts, design ideas and directions were identified, we moved into a medium fidelity prototyping stage where we provided a sketchy visualization of key screens without contents and gradually validated them and added initial contents.

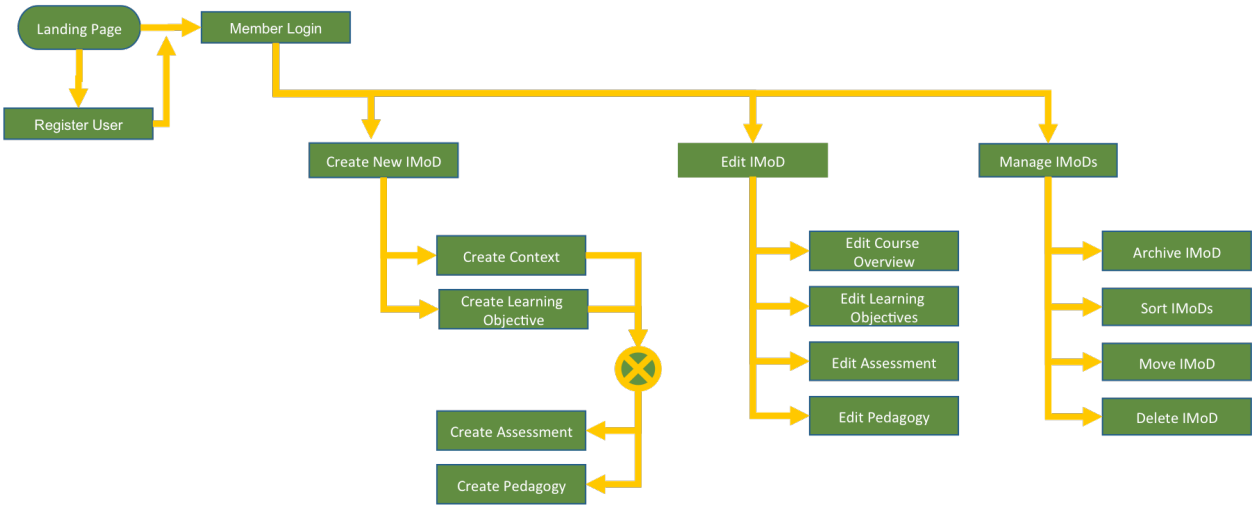

Figure 2: Primary Application Navigation Model

5. Phase 3 – Detailed Design

At this stage, we focused on the main high level solution, and started looking into details from different perspectives including main application features, auxiliary features, concrete navigation models, detailed screens (all screens, with all contents), menu options, visual and interaction consistency across all screens, exceptions and error massages and recovery, reliability assurances and, help. This phase goes in parallel with development phase as more details are uncovered and technical problems arise. The design was based on the theoretical model for outcome-based course design called the PC^3 model [6]. User interface mockups were created with details of various user inputs that will be solicited through the course design process. Figure 3 shows sample user interface mockups of the course overview page and the learning objectives page.

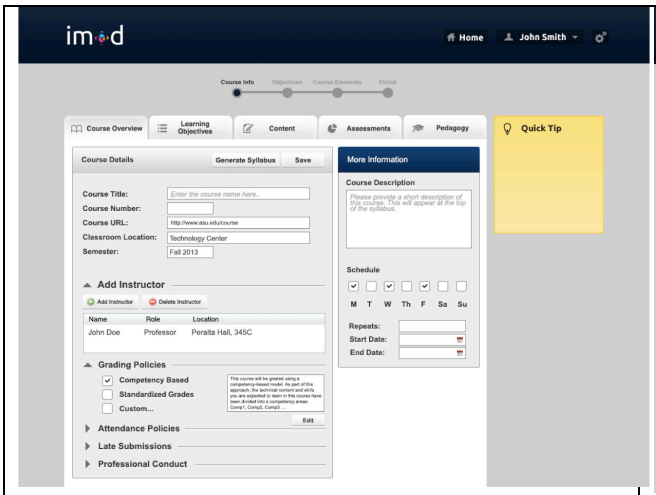

Figure 3a: Course Overview Page mockup

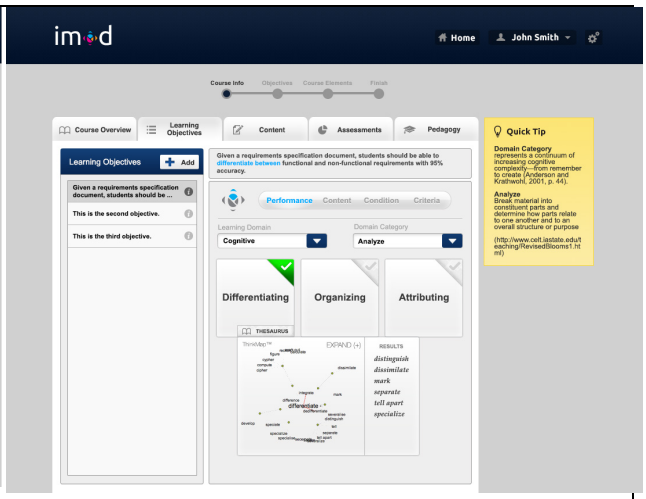

Figure 3b: Learning Objectives page mockup

6. Phase 4 – Development and Development Support

At this stage, we considered different technology options, and started experimenting with platforms and initial software architecture. As implementation of essential features started, close collaboration between designers and software engineers (software architects and developers) was essential to ensure the consistency of design and to prevent any deviations. Several technical problems require careful reconsideration of detailed design and even high-level design options. Iteration is a fundamental design approach that is extensively being used across the UCD process. Therefore, UCD is highly iterative and most of its phases are heavily overlapping to ensure design and development decisions are aligned at all times with the actual user needs.

This phase of the project included identifying appropriate technologies to be used for the development of the IMODS semantic web application, design of the back-end database schema, installation and configuration of the server-side and client-side technologies, and development of the user interface screens for login, registration, index, and creation of an instructional module and the connectivity of these web pages with the backend database. An Agile software development methodology called Scrum is being used for the development of this project. Scrum is an iterative and incremental framework for managing product development. A sprint (or iteration) is the basic unit of development in Scrum. The sprint is restricted to a specific duration, two weeks in case of the IMODS project. Each sprint is started with a planning

meeting. The aim is to define a sprint backlog where the tasks for the sprint are identified and an estimated commitment for the sprint goal is made. Each sprint ends with a sprint review-and-retrospective meeting, where the progress is reviewed and shown to stakeholders and improvements for the next sprints are identified.

6.1 Analysis of Technologies

The purpose of analyzing various technologies during this phase of the project was to ensure rapid development with the latest technologies in the field of software development and use open source technologies wherever feasible. Towards this end, an analysis of web application frameworks, version control systems, server side technologies and client side technologies was performed. Table 1 shows all of the technologies considered and the analysis that informed the final selection.

Table 1: Analysis of Technologies for IMODS development

Key Architecture Functions	Possible Solutions	Analysis Comments	Final Solutions
Framework	Django, Grails, WebApp2	• Django and WebApp2 are written in Python and have Google App Engine support • Django has request handler, template engine and form processor • WebAp 2 has request handler	Grails
Version Control	Bitbucket, Github, Gitlab, Gitlolite, SVN	• Bitbucket - free private repositories • Github - free public repos + paid private repos • SVN is centralized, Git is decentralized • All work with Unix, Linux and Windows systems	Github
Server-side technologies	Java based technology	• As it is compatible with semantic web technologies	Groovy
Databases	SQL, NoSQL, JSON, Google Datastore, MySQL, PostGreSQL	• CouchDB stores data as "documents", as one or more field/value pairs expressed as JSON • App Engine Datastore provides a NoSQL schema-less object datastore, with a query engine and atomic transactions • IMODS data is expected to have numerous relations and hence schema-less store is not being chosen	PostgreSQL
Client side scripting	ExtJS, jQuery, JavaScript, CoffeeScript, AngularJS, BackboneJS, HTML5, CSS3, Twitter Bootstrap	• Backbone.js requires more Boilerplate code, but is smaller than Angular.js • ExtJS does not provide good support	jQuery, HTML5, CSS3, possibly Bootstrap
Semantic Web Technologies	OWLite, Protege, Apache Jena	Apache Jena • API for reading, processing and writing RDF data in XML, N-triples and Turtle formats • Rule-based inference engine for reasoning with RDF and OWL data sources • Stores to allow large numbers of RDF triples to be efficiently stored on disk • Query engine compliant with the latest SPARQL	OWLite, Protege, Apache Jena

		specification	
Licensing		• GPL, MIT, BSD - It is not permissible under the GPL to use GPL in proprietary software while keeping that software closed source • MIT and BSD: Because you cannot restrict others from simply obtaining the source code, selling open source licensed software as is makes for a difficult proposition	TBD
Cloud/web technologies	Google App Engine, Amazon Web Services (AWS)	• Google App Engine: Free within quota, help and tutorial • AWS: Free usage for a year	None for now
Login			CAS

The following subsections provide a detailed description of the selected technologies.

A. Groovy on Grails:

Groovy on Grails is an open source, full stack, web application framework for the Java Virtual Machine. It takes advantage of the Groovy programming language and convention over configuration to provide a productive and streamlined development experience. The main features of this framework are as follows:

- Rapid: The Grails framework is one the fastest ways to get a web application up and running.
- Robust, proven technologies: Grails is based on Spring and Java – technologies that are tried and tested by millions.
- Interoperability with Semantic Web technologies: One of the major requirements of this project was having the ability to use the Apache Jena API in the future. Groovy runs on JVM, hence, it is compatible with Apache Jena API that is written in Java.
- Simplicity: Groovy has a clean and simple syntax, and is easy to learn and understand.
- Easily defined relationships: In Grails, it is easy to define relationships between classes using Grails Object Relational Mapping (GORM). These relationships could be one-to-one, one-to-many or many-to-many, and may be unidirectional or bidirectional.
- Scaffolding: Relationships specified in the domain classes can be used to create, read, update and delete database entries.
- Model, View, Controller Architecture: Once domain classes are generated, controllers and views can be generated using grails commands.
- Plugins are available for most things any web application would need, for example, the SpringSecurity plugins for login.
- Support is available on all modern browsers automatically.
- Groovy on Grails has a very active development community, with ample documentation and other resources.

B. Groovy/Grails Tool Suite:

The Groovy/Grails Tool Suite™ (GGTS) provides the best Eclipse-powered development environment for building Groovy and Grails applications. GGTS provides support for the latest versions of Groovy and Grails, and comes on top of the latest Eclipse releases.

- Included with GGTS is the developer edition of vFabric tc Server, the drop-in replacement for Apache Tomcat.

- GGTS is freely available for development and internal business operations use.

C. PostgreSQL database management system:

PostgreSQL is a powerful, open source object-relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness. Following are some features of PostGreSQL:

- It runs on all major operating systems, including Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), and Windows.
- Its closest competitor, MySQL, is now owned by Oracle, making its future uncertain.

D. Git for version control:

Git is a distributed revision control and source code management (SCM) system with an emphasis on speed. The advantages of using Git for source code management are as follows:

- Git is decentralized. The local copy is a repository with ability to commit to it and get all benefits of source control on the local system. This is useful when there might be connectivity issues. Local commits can be pushed to remote repositories any time the connectivity is reestablished. This makes it better than centralized version control systems like SVN.
- Git is well suited for open source projects – a project can be forked and changes can be made in the forked project, one can then ask the original project maintainer to pull code from one's fork.
- Github is free and allows developers to connect and review code.
- Git is now supported on Windows as well – initially it was only supported on *nix platforms

6.2 System Architecture

Grails uses Spring Model–View–Controller (MVC) architecture as the underlying web application framework. MVC is a software architecture pattern which separates the representation of information from the user's interaction with it. The Grails architecture can be described as follows:

- The foundation of Grails is the Java Virtual Machine (JVM).
- There is a separation between the Java language and the JVM. This is important in Grails because in the next level up from the JVM, both the Java and Groovy languages are used.
- The final layer of the architecture is the application layer. This layer follows the Model-View-Controller (MVC) pattern.
- Grails uses Spring MVC as the underlying web application framework.
- A controller handles requests and creates or prepares the response. A controller can generate the response directly or delegate to a view.
- A controller can have multiple public action methods, each of which maps to a URI.
- A model is a Map that the view uses when rendering information on the web page. The keys within that Map correspond to variable names accessible by the view.

7. Future Work

The development of IMODS prototype is still ongoing, and will be further described in future publications. The next steps will include completing development and testing of the key features

of the IMODS system. Usability testing of this prototype will be conducted at the Co-PI's institution followed by testing at a leading Engineering Education conference to get user feedback that will be incorporated into the design and development of the IMODS system. The scope of this project will also include the evaluation of its novel approach to self-guided web-based professional training in terms of: 1) user satisfaction with the documentation of course designs generated; and 2) impact on users' knowledge of the outcome-based course design process.

Acknowledgments

The authors gratefully acknowledge the support for this project under the National Science Foundation's Transforming Undergraduate Education in Science, Technology, Engineering and Mathematics (TUES) program Award No. DUE-1246139.

REFERENCES

[1] G. C. Furman, "Outcome-Based Education and Accountability.," *Education and Urban Society*, vol. 26, no. 4, pp. 417–437, 1994.

[2] S. Bansal, O. Dalrymple, A. Gaffar, and R. Taylor, "User Research for the Instructional Module Development (IMOD) System," in *American Society for Engineering Education Annual Conference (ASEE)*, Indianapolis, IN, 2014.

[3] O. Dalrymple, S. Bansal, A. Gaffar, and R. Taylor, "Instructional Module Development (IMOD) System: A User Study on Curriculum Design Process," in *Frontiers in Education (FIE)*, Madrid, Spain, 2014.

[4] J. Lazar, J. H. Feng, and H. Hochheiser, *Research methods in human-computer interaction*. John Wiley & Sons Inc, 2009.

[5] W. Lidwell, K. Holden, and J. Butler, *Universal principles of design: 125 ways to enhance usability, influence perception, increase appeal, make better design decisions, and teach through design*. Rockport Pub, 2010.

[6] K. Andhare, O. Dalrymple, and S. Bansal, "Learning Objectives Feature for Instructional Module Development System," presented at the PSW American Society for Engineering Education Conference, San Luis Obispo, California, 2012.