Mendix as a solution for present gaps in Computer Programming in Higher Education

Emergent Research Forum (ERF)

Michael Litman
Concordia University Wisconsin
Michael.Litman@cuw.edu

Daniela Field
Mendix | MIT
Daniela.Field@mendix.com

Abstract

Traditional Computer Science education introduces computer programming very early in the curriculum. While appropriate, this does create difficulties given that related topics are not introduced until much later. While it is important to familiarize students with computer programming concepts as early as possible, it is equally important to introduce them to related concepts such as persistent storage, API integration, user experience implementation and user design as early as possible at a high level. Traditionally, these concepts are not introduced until far later in the curriculum creating disconnect in a student’s education. This disconnect is most noticeable upon graduation when many students are unfamiliar with common integrations used in industry. The Mendix platform allows these concepts to be introduced early on at an applied level, allowing students to benefit from their function prior to being introduced to their underlying theory.

Keywords

Programming languages, education, rapid application development

Introduction

While all software applications require programming logic, many require technologies beyond one’s skills with a programming language. In many cases, a reasonable user interface, localized or cloud based persistent data, third party API integration, and security concerns play a role in the ultimate solution. It is the position of this paper that modern Computer Science approaches in Higher Education failed to adequately deliver these competencies. Students are bombarded with syntax and logic in their first several courses, only to be introduced to topics like database and security later in their curriculum. These topics are often taught from the theory perspective, giving students little, if any, practical experience in the context of software application development. Topics such as team collaboration tools, API integration, user interface and user experience are largely ignored or barely touched upon outside the context of a larger software application. This approach produces Computer Science graduates who have a strong programming and problem-solving skillset, but who lack the tools necessary to bring the technologies to bear that are required to be productive in the industry without additional training. (Bissyandé et al. 2013) In order to understand the students and their skills we designed a survey to allow students to self-assess their skills and knowledge in the 5 main identified skill categories needed. The goal of this research paper is to identify the gaps and further continue the research in addressing the student’s skill gaps.

Current State of Computer Programming Education

Computer programming education tends to emphasize the difficulty of learning syntax over other, arguably, larger barriers (Stefik et al. 2013). This is evidenced by the transition to interpreted languages like scheme and python, which present lower barriers (Prechelt 2000) of entry than historically complex languages like C/C++. Computer programming involves telling a computer what to do. The job of a computer programming language is to give a human the tools necessary to solve problems in a way that is similar to how they traditionally solve problems in daily life (Lahtinen et al. 2005). As an abstraction, we will assume
that humans solve problems by using their memory, asking questions, and repetition. All programming languages have facilities which emulate these abstractions. Traditional programming languages emulate memory through the use of variables, asking questions through the use of conditionals, and repetition through the use of loops and functions. Providing a usable representation for these ideas is the job of the language syntax and is independent of the computer programming skillset necessary to wield any programming language.

Learning to program a computer can be thought of as being similar to becoming proficient in a sport. Practice, repetition, and time are all part of becoming a proficient computer programmer. One can academically understand a programming concept, but putting that concept into practice in the context of a large problem requires significant experience (Bhattacharya et al. 2011). One reason that computer programming education is so difficult initially is that humans are so proficient at solving problems that they have difficulty articulating how they solve problems. The skill of breaking the steps of a solution down into smaller pieces requires a significant time commitment by the student, which in turn requires the educator to delay or omit the introduction of related technologies for fear of overwhelming the student. This is exacerbated by the traditional approach of using a high level, syntax-based programming language to practice this new skillset (Bissyandé et al. 2013; Ghezzi et al. 1997). A new computer programming student is faced with the daunting task of learning computer programming as a skillset, while also learning the syntax of a new language, as well as the associated tools required by that language. This can lead to a disconnect regarding competence in computer programming offering only part of the required solution to a modern software application (Harrison et al. 1996; Meyerovich et al. 2013; Nanz et al. 2014).

We conducted a survey to assess students’ familiarity on five important development skills and the results confirm theories proposed by this paper. There were 112 student survey takers primarily from 2 universities: Concordia University Wisconsin and Massachusetts Institute of Technology. 75% of the students were juniors, seniors and graduate level students in computer science and information science field. 17% were freshman and 7% were sophomore. While the initial survey group is small, our goal is to expand the survey across more students and universities.

![Chart showing student familiarity with different programming skills.](image)

**Figure 1**

Based on the survey, the students reported higher levels of programming fundamentals with scores of 4 to 5. Conversely, user roles and security, third party APIs, databases and user interfaces are lagging behind - with the majority of students reporting scores of 1 to 3, meaning no experience to limited exposure.
Over the last 20 years, the state of technology has changed significantly, yet the state of Computer Science education has changed very little. Students are not only graduating without the practical experience described above, but they also lack the context concerning where their various theory classes fit into the software development process. (Dahlgren et al. 2005)

It would be advisable to revisit how computer programming is taught. We propose a tool that abstracts the problem-solving component from the syntax to allow students to focus on proficiencies related to the various components found in modern software applications. A tool that allows students to explore user interfaces and persistent storage and allows them to tap into external data sources through APIs and to target various platforms would be beneficial. A tool that empowers students to tackle problems similar to those faced in industry very early in their education by focusing on applied concepts rather than the underlying theory would provide students with the practical foundation necessary for their theory courses to fit into a framework where their usefulness is evident. Concepts like MVC, Singleton, and the Observer design patterns could be covered much earlier in the curriculum where they could be introduced in a practical sense, rather than as a footnote in a later course. (Miliszewska et al. 2007; O’Kelly et al. 2006; Schulte et al. 2006)

**Mendix as a tool to bridge the gaps**

Mendix is a full stack platform designed to rapidly develop applications. The platform gives developers control of the various app development layers from front end to back end. It abstracts and automates the main app development components, such as the data structure, business logic, and user interface.

**Collaboration**

In the industry, collaboration between business and IT is critical for a company's success. The business understands business problems and needs technical and digital solutions to fix those problems. The IT department needs to support the business by delivering solutions that work. Collaborating and speaking the same language is essential. In order to deliver applications rapidly - in weeks vs months - the iterative process within the agile methodology is the best fit (Frydenberg et al. 2017). The agile methodology allows for iterative development and for the business to provide input and shape the product before it is delivered. When students create a Mendix project, the collaboration workspace is automatically created with built-in agile process features. Students can capture their user requirements as user stories and place them into weekly or bi-weekly development cycles called Sprints. Throughout the semester, they can capture the requirements and build a solution and iterate and improve it until there is final presentations.

**Database Structures**

The database structure is central to the architecture of an application. It is where data is stored. In the Mendix platform, the data model describes the information in an application in an abstract way. It is based on Unified Modeling Language (UML) and composed of 3 main components: entities, attributes, and associations. The platform translates entities into database tables, attributes into table columns, and associations into database tables that store the primary keys of the related entities. Associations are the relationships between entities based on cardinality - one to many, one to one, or many to many.

**Business Logic**

In the platform, the business logic is designed using the principles of Business Process Modeling Notation (BPMN). In order to add logic such as generating documents, validations, calculations, third party integrations, and more, the developers will build microflows. Microflows are a visual representation of the logic and steps, and its main building blocks are start and end points, activities, exclusive splits, input parameters, and so on.
The above microflow shows the business logic required for registering a student into a class. The pathways and activities can be understood by students of all levels from freshman - students without any technical background - to seniors.

**User interface and user experience**

The user interface and user experience can make or break an application. If the application is archaic, and the end user has to click too many times to accomplish essential tasks, they will not have a pleasant experience, and most importantly, the rate of adoption will drop. Thus, an application that delights end users, and provides a great flow and experience, is essential. The user interface is designed as a what you see is what you get (WYSIWYG). Students can drag and drop user interface components like viewing of lists of objects or changing single records. They can design and change the pages to fit their project needs.

**Security**

Most applications have some form of user authentication and limited permissions based on user roles. As a developer, user roles are created from the user stories. Based on the user role, an end user of the application has to be authenticated and given access to parts of the application. The developer can limit the access of forms, microflows, and specify the create, read, update, and delete (CRUD) permissions on the entity level. Through a number of clicks the students can assign user interface, logic and database level security and permissions with ease.

**Conclusion**

In conclusion, there is a lack of real world experience for the students based on their own self-assessed survey results. The goal of our research is to expand the survey to further students and universities. The Mendix platform bridges the gap between computer science education and the demands of the industry. We propose a modification to modern computer science curriculums to introduce students to a tool like Mendix early in their degree programs, in order to assist their understanding of the various building blocks in common industry software development and computer science theory. In order, to assess how effective a tool like Mendix is, we will be conducting the same survey to the next group of students who have learned the platform in their curriculum. We will compare the data and assess how the student’s skills are changing and continue the research on best ways to address the industry skill gaps.

**Acknowledgements**

We would like to thank Mendix and survey participants from Concordia University Wisconsin, MIT, and Western Illinois University.
REFERENCES


