Experiences in Enhancing Functionalities in Apps Using a Customer-Focused Agile-Oriented Approach

Ethan L. Baker
Department of Computing Sciences
SUNY Brockport

May 13, 2023

Faculty Advisor
Sandeep Mitra
Professor
Department of Computing Sciences

Table of Contents

Abstract

We describe our experience in developing a student assessment data capture and management app for a campus-based committee. The primary goal was to develop the app to be customized to the specific needs of the customer. The code from a similar app developed for another campus customer during the previous year was available to us. While this app had some features that were also replicated in our project, and that code could be reused as is, much of the functionality had to be adapted to the new customer's requirements, and a number of features were completely new to this project. To implement this application, we took an Agile-oriented approach in which we stayed heavily engaged with the customer throughout, seeking to meet the customer's goal of a user-friendly and intuitive GUI. Throughout the project, we encountered various obstacles. First, in a small university with no graduate program, there were few qualified students available for the project team, so the team was relatively small compared to the size of the project. In addition, we had a hard time limit dictated by the academic calendar and graduation dates. The ability to reuse code from the previous project helped, and the time spent in understanding this code was reduced considerably by the previous project's adherence to appropriate design patterns and that our project used the same technology tools. To succeed in this environment, we also observe that it is necessary to work with the customer to identify the project scope, take a Scrum-based approach to track progress, and conduct code reviews to keep everyone "on the same page". It is also critical to ensure that we get a team with the right skills, and ensure adherence to coding standards, for which we note that faculty member involvement in identifying the right students (rising juniors, for example), and *especially* in ensuring coding standards are met, is necessary.

Introduction

The purpose of the project was to develop an application that assists with data management for general education assessment for the college. It would allow users to easily input and modify student assessment data, as well as generate reports based on that data. The General Education Assessment Committee (GEAC) of SUNY Brockport that the app was developed for had previously recorded and stored student assessment data in Microsoft Word documents, which proved to be inconvenient and slow. Additionally, there were some calculations that professors had to perform by hand, which caused errors at times. This project was meant to alleviate these difficulties by streamlining data entry and report generation, and to provide one centralized location to store the student assessment information.

The application was developed by a team of two students with the assistance of the faculty advisor. This setup was not ideal, as the scope of this project warranted a larger team of three or four students. However, there were no other interested students who also had taken the necessary classes that would prepare them for a project of this size. A student would need to have taken both software engineering classes CSC 427 and CSC 429 (as they simulate a similar project) and also have at least one year before graduating in order to join the team, which significantly reduced the pool of candidates. There were multiple students that were interested in joining the team, but they lacked the technical knowledge required to meaningfully contribute, as they were still early in their academic careers.

An important aspect of this project was learning how to correctly adapt existing code to work within a new application. We note that in order to be successful in this endeavor, it is imperative that both applications utilize the same design patterns, otherwise they would be too

different and code from one would be incompatible with the other. The patterns we primarily relied on were model-view-controller, observer, and factory, all of which were present across both applications. The role of the faculty advisor is critical here, as they are more familiar with the correct design principles than the student team, and so it is up to them to make certain that these principles are being upheld. A loose adherence to the aforementioned design patterns would have made our system too dissimilar from the old application such that reusing any aspects of it would have been nearly impossible.

Overview of the Agile Software Development Methodology

The approach we took was based on the Agile software development methodology, which promotes design flexibility and heavy customer involvement. The Agile methodology was born out of a need to develop software faster ("Agile Alliance"), which, given our time constraints, made it a near-perfect fit. The Agile Manifesto states four key principles: individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan (Manifesto for Agile Software Development). Not every principle was relevant to the project, but Agile was still better than other well-known methodologies. In particular, "customer collaboration over contract negotiation" was something we wanted to keep in mind throughout the project, because while there may not have been contracts to negotiate, our focus on heavy customer involvement fit the spirit of that principle. In addition, "working software over comprehensive documentation" perfectly matched our philosophy of delivering the product first and foremost, and then working on other deliverables like a user manual second. And finally,

“responding to change over following a plan” is a cornerstone of good software development, especially when the customer is highly involved, since they might request new features or alterations at any point in the development process. Flexibility is key in the professional world, and we strove to emulate real techniques as much as we could within the bounds of our constraints.

Another concept we borrowed from real-world software development is Scrum. Scrum is an Agile framework that aims to get results through incremental progress and continuous feedback (“What is Scrum?”). Scrum is a fairly straightforward methodology that offers enough customizability that a software team is often able to implement it without much issue, which is exactly how it turned out in our case. Scrum is based around Sprints, which are intervals of time in which the team works toward a specific goal (or goals) that are delineated at the start of the Sprint. We chose a length of one week for each Sprint with weekly or biweekly meetings depending on the workload. In a Scrum Team, there are three roles: the Product Owner, the Scrum Master, and Developers (Schwaber and Sutherland). The Product Owner serves as the representative of the product, meaning they understand it best and decide what features should and should not be included. In order to change an aspect of the product, one would have to petition the Product Owner to accept the change. In our team, the faculty advisor held the role of Product Owner, due to their increased understanding of the customer’s needs and experience with directing teams of students. The next two roles, Scrum Master and Developer, go hand in hand. The Scrum Master personally oversees the team of Developers and ensures they are following the Scrum process and producing steady results. The Developers are the ones that are responsible for programming and other tasks related to directly creating the product. In the two-student team, we switched which student took on the role of Scrum Master and which one was

only a Developer every Sprint, although we made a number of changes to the Scrum Master position to enable it to work cohesively with the project. In most professional software development teams, the Scrum Master does not actually do any programming, but they might review code and suggest changes. With only two students, we could not afford to have only one actively coding at any given time, especially since most of the programming was done independently and thus not under the supervision of the Scrum Master. The student that took on this role was instead primarily responsible for updating the Product Owner about any new additions or changes to the application, as well as deciding what each student would work on at the start of a Sprint, while still doing an equal amount of the coding. The position of Scrum Master also rarely rotates around a development team, but we made this change in order to give both students experience with this role. Overall Scrum proved to be a very effective tool that significantly helped us stay on track and imparted invaluable experience onto the two students.

We ran into a few restrictions that hindered our ability to work in an Agile manner. The first was our strict deadline dictated by the team's graduation at the end of the Spring semester. The Agile methodology deemphasizes adhering to deadlines as a way to promote iterating over the product and continuously incorporating customer feedback in order to create the best product possible. This is different from many other software development methodologies, where the goal is to finish the product before a set deadline, without worrying as much about the quality. One such methodology is Waterfall, named for the idea that development phases progress linearly, and once a phase is completed, one cannot go back to it, much like traveling over a waterfall. While we still wanted to be as Agile as possible, we incorporated some ideas from Waterfall in order to gear us toward finishing the application before the deadline.

Another restriction on our ability to be Agile was simply the limited number of students on the team that worked on the project. The flexibility that Agile encourages was difficult to adhere to when our team did not have the manpower to quickly implement customer suggestions. If we had a larger team, even by one or two students, then the time constraint would have been much less problematic and we would have been able to incorporate more feedback from the customer. In a more ideal version of this project, there would have been a larger team, and we would have completed the main coding of the application much earlier. This would allow us to then give the customer some time with the system, and have them give us feedback, which we would use to alter the application to the new specifications. This setup is much closer to Agile standards than we were able to achieve, so we had to try our best to adapt. This restriction was more difficult to overcome than the deadline constraint, since software development methodologies that deal with small team sizes are almost nonexistent. Even less Agile methods like Waterfall still assume a reasonable number of people in the development team, so we had to improvise a solution, which turned out to be straightforward, but difficult. The student team simply had to do more work each Sprint to ensure they were ahead of schedule. This was especially taxing due to the class workload they had, as both were full-time students for the duration of the project, so work on the application would fluctuate heavily during midterms, finals, and other work-heavy periods.

Designing the System: Keep the End-User Experience in Mind

An early challenge we ran into was the size of the main screen. On the older application, there were 14 primary buttons which were all reasonably small. On our system, we had 19

buttons, and many had fairly long names. This proved to be problematic, because while we did not want the main GUI to be too densely packed, we also wanted the button names to be descriptive enough that users would understand what they did. To solve this issue, we spoke with the customer extensively to figure out names for the buttons that would both be understandable to a user and small enough to make the main screen compact. One of the ideas we considered was to have each main screen button lead to another screen with more button choices. This way we could put fewer buttons on the initial screen but still have each use case represented by a button. We did not end up following through with this however, due to the additional time it would take to implement the added screens. In the end, we made some compromises and created a main screen that was a bit more crowded than we would have liked, but it accomplished our goal of communicating the purposes of the buttons to the users. We grouped buttons in rows, such that each row would include related functions. For example, the top row had three buttons: Add a New Gen Ed Area, Update Existing Gen Ed Area, and Delete Existing Gen Ed Area. All of the functions that dealt exclusively with Gen Ed Areas were in that top row. We also positioned the rows so that a user would generally start at the top row and work their way down in most cases. There would rarely be an instance where a button higher on the main screen would require data from a button lower on that screen.

Let us now provide some more details about the functionality of the application. This is necessary for the reader to be able to understand the relevance of the descriptions of our reuse experience that we provide later in this document. It will also serve as a good piece of documentation for future students who may sign up to be maintenance engineers on this product.

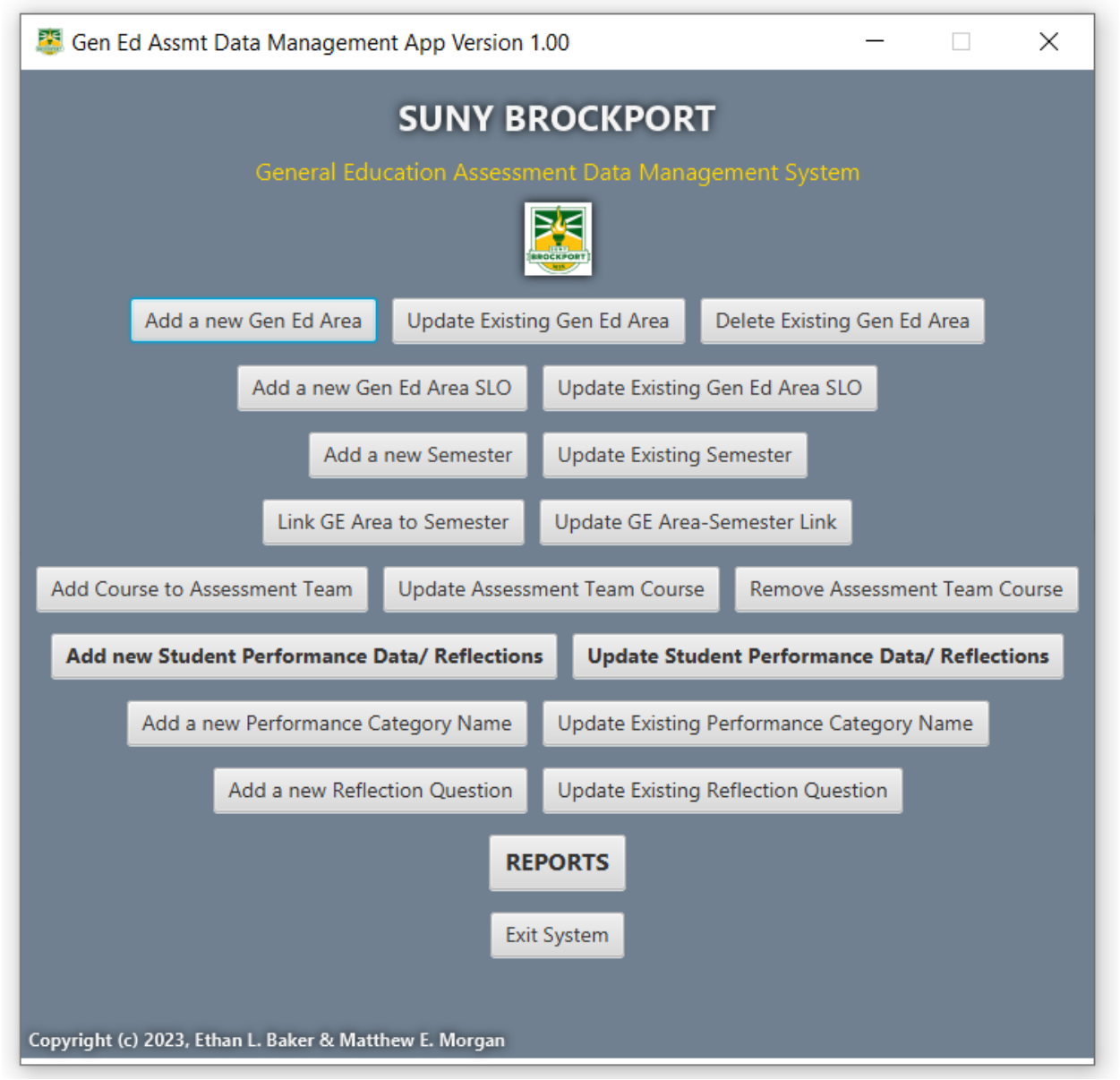

Figure 1: Main Screen

In the general use of our application, the user will be entering all of the relevant data for a given semester. This need not happen all at once, but the button progression will be roughly the same regardless of the time it takes the user to enter the data. They will start by adding a new Gen Ed Area, if they have to. There are a fixed number of Gen Ed Areas that are rotated through, so eventually there will be no need to add new ones. Once the required Gen Ed Area is in the

database, all of the Student Learning Outcomes (SLOs) associated with that Gen Ed Area must be entered as well. Next, the user must add the semester that they are performing the evaluation in, and then link the required Gen Ed Area to the semester. This creates what is called an "Assessment Team," which is just a Gen Ed Area and Semester combination that represents the group of professors that will be performing the evaluation. The Assessment Team is then used to add all of the relevant courses that the SLOs will be assessed in. All of these functions can be performed early on in the semester, since they do not yet use student performance data. The next functions will begin with entering the assessment data, and thus will be done much later. The user can enter the student categorization data and the instructor reflection data in any order they wish. Student categorization data is the numerical data that shows how well the students in the relevant courses did in each SLO area, sorted by student level. There are four performance categories: Exceeds, Meets, Approaches, and Does Not Meet; and each student falls into one of these categories for each SLO. Instructor reflections are the assessment team's answers to each of four prompts regarding student performance in the current semester and as compared to previous semesters. The student categorization and instructor reflection data is the most important information we store in our database, since it is difficult to store and synthesize without an automated system. The two rows of buttons beneath the categorization and reflection row are something of an exception to our design philosophy, as they deal with information that the user has already entered on previous screens, but they will be used so rarely, if ever, that it is not impactful. One row allows the user to add or modify the names of the performance categories, and the other supports addition and modification of the prompts for the instructor reflections. It is unlikely that these buttons will be used, but we created them just in case.

The final button on the main screen is for reports. We allow the generation of three types of report, which will be discussed in sequence. The first utilizes the student categorization data that the user has entered to display performance by SLO in a similar manner to how the user originally entered the data into the system, but with a few additional options. First, it displays what percentage of students were in each performance category per SLO. In the past, these percentages were calculated by hand by the professors, and there were often errors present in these values. This will no longer be the case once our application is implemented, since it calculates these percentages automatically. Another useful feature is that it displays the totals and percentages for both the “Exceeds” and “Meets” categories summed together, which is useful data for the evaluation team. And importantly, there is an option to select the student level of the data to be in the report. This way, the assessment teams can easily view the results for freshmen, sophomores, juniors, seniors, or all of the students aggregated together. This report will be significant, because it allows the GEAC to see how Brockport students performed at SLOs over time by viewing the data throughout each semester. The second kind of report that our application can generate is the instructor reflection report. This allows users to print out the four reflection prompts alongside the respective instructor reflections for a given Gen Ed Area/Semester pair. And finally, the assessment team report simply prints each course number and course code for the assessment team that the user selects as an easy way to identify which courses were evaluated for a given Gen Ed Area.

	A	B	C	D	E	F	G
1	Report for Gen Ed Area: Natural Science assessed in Semester: Fall 2023						
2	Class Level: All						
3							
4		SLO 1	SLO 2	SLO 3	SLO 4	SLO 5	
5	Exceeds	112 (41.64	113 (43.30	126 (46.49	62 (50.41%	98 (37.84%)	
6	Meets	105 (39.03	64 (24.52%	67 (24.72%	38 (30.89%	76 (29.34%)	
7	Approache	32 (11.90%	55 (21.07%	43 (15.87%	19 (15.45%	62 (23.94%)	
8	Does Not I	20 (7.43%)	29 (11.11%	35 (12.92%	4 (3.25%)	23 (8.88%)	
9							
10	Meets and	217 (80.67	177 (67.82	193 (71.22	100 (81.30	174 (67.18%)	
11							
12	Report created on 04-12-2023 10:51 AM						

Figure 2: Gen Ed Basic Data Report

Implementing the System: How Can We Effectively Reuse Similar Code?

In the previous year, an application similar to ours was developed for the Institutional Student Learning Outcomes (ISLO) Assessment Committee at SUNY Brockport. This application collected and aggregated assessment data, and allowed users to create reports that utilized the data in the system. The major difference between the two applications is that the existing app was built for ISLO data, while ours is for General Education Assessment data. From the data management point of view, the manner in which the data was collected and organized for Gen Ed was different from that for the ISLOs. For example, the raw data for Gen Ed is collected on the basis of student level (freshmen, sophomore, junior, and senior), whereas for ISLOs the data had no such distinctive characteristics, but was aggregated across all students in the classes from which it was collected.

While the majority of our application was coded from scratch, some of the functionalities of the earlier system could be reused in their entirety. For example, both apps required semester data, using the season (fall or spring) and the year, to be stored in their databases, and the code in the prior app for adding and modifying semesters could be reused completely. There were also functionalities that were similar across the two apps, but not identical. The adding and modifying of the principal assessment categories is one such example. In our system, we do not have ISLOs as the principal assessment category; instead, we utilize Gen Ed Areas as the principal category for which we aggregate the information in the database. ISLOs and Gen Ed Areas are similar enough that we could import the code for ISLO features and adapt them for Gen Ed Areas with minimal changes. Similarly, the way we collect and update student performance data and instructor reflections draws heavily from those functionalities in the old system, with a few alterations. Our student categorization and instructor reflection data is much more complex than the corresponding data from the other application, so while that one was able to condense both functions into one screen, we had to separate them for the sake of user-friendliness. So while we had to write plenty of additional code in order to handle the more complicated data, the code that we were able to reuse still provided us with a solid groundwork to build upon due to the underlying similarities.

The report generation feature of the old system turned out to be somewhat tricky to reuse. In that application, only one type of report could be generated, while ours supported three different types. This meant that while some of the general-purpose code could be modified to work with our reports, much of it had to be created ourselves. We decided to use the comma-separated values (CSV) file format in order to be consistent with the old application and to facilitate the reuse of certain elements of its “write to file” function. We also inherited its use of

tables to preview the data that would be written into the report file for two of our reports, but the student categorization report proved to be too complicated for us to fit all of the necessary data into one table.

Reusing code in this manner can be beneficial, but there are some potential hazards that come with utilizing code not built for the current system. If the two applications are extremely different, then it may be more trouble to utilize existing code than it is worth, since it would have to be heavily modified to fit with the new system. However, there are often subtle changes between programs that are difficult to detect, and they can have unanticipated effects with reused code. Even small changes can require repurposed code to undergo extensive rewrites, so it is important to recognize what can be reused and what is better off being created from the ground up.

One of the most important aspects of the previous system that we decided to repurpose was the graphical user interface (GUI), or the screens in the system. There were a few reasons we made this decision, with the first being simply that it would cut down on work in an already time-constrained environment. We could not import the old screens in their entirety of course, but adapting their formatting and design would save us time and allow us to work on more important aspects of the system. However, another important reason for this decision was because, if the two applications looked similar to users, it would help those that were already trained in the old system to understand the new one. Our system would seem less like an entirely new application and more like a continuation of the old one. This would reduce the time it takes for the customer to learn how to use our system, and ease their transition into using it.

Observations From Our Experience

Due to our strict adherence to our design principles and development methodology, as well as the exceptional work ethic of the student team, we were able to finish the coding aspect of the project at the end of May, ahead of schedule. The rest of the time was devoted to other important tasks, such as deploying the product at the customer's site, training the customer in its use, and creating a user manual. A user manual is a document that gives instructions and advice for those that use the system. It is especially important in our case because the people that understand the application the best (the student team) are both graduating shortly after the project is finished. This means that any questions a user might have cannot readily be answered by the two programmers, so a user manual has to serve as a stand-in. Thus it is critical that the manual contains detailed enough instructions that a user will be able to operate the system without guidance, as well as answers to common questions they might have. The user manual was our primary focus once we finished coding, but we made sure not to neglect the other tasks either. Deploying the product is as simple as setting up the application on the customer's computer, but training them is a difficult aspect for many projects. Fortunately, our customer was already familiar with the application from the previous year, so our system was straightforward for them to learn. As noted earlier, we intentionally developed our application to be visually and logically similar to the previous one, and this is one of the areas where that decision paid off. The simple similarities such as identical button placement and the use of the same fonts greatly aided the user in reducing confusion when learning our system. The more they understood the similar aspects, the more they were able to focus on learning the new ones.

Conclusion

This project had its fair share of complications, many of which were unique to the academic environment it took place in. The strict deadline limited our ability to integrate Agile principles into our workflow and prohibited us from meaningfully incorporating large changes to the product as requested by the customer. In addition, being limited to two students on the development team also reduced our flexibility and required them to work much more quickly and flawlessly than other academic programming projects. And the fact that both students were full time meant they only had so much time to work on this project between classes, assignments, studying, and exams. All of these factors—combined with a large scope and actual customer—demonstrates that it was no easy feat for the team to deliver the product when they did, especially to the satisfaction of SUNY Brockport's GEAC. Of course, there were some elements that worked in our favor, namely our intelligent reuse of code from a prior application.

The system developed for the ISLO Assessment Committee mirrored ours in many ways, or more accurately, our design was modeled after that of the previous system. Both used semester data, principal assessment categories (ISLOs and Gen Ed Areas), and allowed the generation of reports. There were enough similarities that we could use the ISLO system as a framework to build off of when designing ours, in order to reduce the code that had to be written from the ground up. Developing an application from scratch is a difficult process, as it is not always simple to recognize what order the different components should be created in, and there is often much backtracking to connect existing elements to newly created ones. With such a framework, we were able to alleviate these problems and focus intently on programming the various functions our application performs. However, it was not as simple as copying the old

code into our system, as we had to ensure that we utilized the same design patterns that the previous team did. This is where the faculty advisor is exceptionally important, since they understand not only the concepts behind the proper design patterns, but also how to correctly implement them. If they are not careful to make sure that the student team is following these patterns, then there will be wasted time later on fixing these flaws. In a project of this size, taking a coding shortcut will result in problems down the road, as each component relies on many others working correctly. Proper code reuse and adherence to design patterns were two of the major factors that allowed us to be successful despite the many constraints. Other contributing factors were our use of Agile development techniques along with concepts from Scrum and Waterfall, as well as selecting students that had the necessary skills to take on such a project. Our recommendation is for related projects to adopt a similar philosophy in order to see the same success we did.

References

"Agile Alliance." Agile Alliance |, 26 Apr. 2023, https://www.agilealliance.org/.

Manifesto for Agile Software Development, http://agilemanifesto.org/.

Schwaber, Ken, and Jeff Sutherland. "The 2020 Scrum GUIDE." Scrum Guide | Scrum Guides, 2020, https://scrumguides.org/scrum-guide.html.

"What Is Scrum?" Scrum.org, https://www.scrum.org/resources/what-scrum-module.

Appendix

Code of the entire application is available at:

https://github.com/Ethandevil/BakerHonors