The Viability of UML Models in Small Web Applications

Jason Dudley and Devon M. Simmonds
Department of Computer Science
University of North Carolina at Wilmington
{jcd8852, simmondsd}@uncw.edu

Abstract

As the use of online business systems increase so does the need to produce more efficient and less error prone web based systems. Modeling is a core abstraction mechanism used to manage complexity, drive implementation and save resources by facilitating the identification of potential problems during systems design. While there is a prevailing intuition of the benefits of using modeling languages such as the Unified Modeling Language (UML) in the development of very large systems, research is needed to delineate the viability of modeling languages for small and medium sized applications. This paper reports on a case study that examined the viability of using UML to model web based applications that require 20 - 50 hours to produce for developers new to UML. The goal of the research is to answer three questions. What are the benefits and costs of using UML diagrams? Will UML diagrams increase the customers understanding of the applications being developed? Will UML diagrams decrease error prone code? Results indicate that use of UML diagrams may be more appropriate for larger applications.

Keywords: UML, web-based software development, software engineering.

1 Introduction

UML [17, 12] has become the de facto modeling standard for object-oriented software systems. Use of UML is particularly appealing due to the increased complexity of software systems which make it more difficult to express software requirements design and architecture using procedural languages [8].

There are several perceived benefits of UML diagrams in software development [1]. UML diagrams can serve as a source of documentation in which large systems can be simplified by raising the level of abstraction [2, 14]. UML modeling may facilitate the discovery and correction of flaws early in the software lifecycle before implementation has begun [16] and in addition, use of UML diagrams is expected to enable external stakeholders to better understand the functionality of the system resulting in improved stakeholder communication of issues and changes [16].

However, these perceived benefits must be corroborated with actual research data if widespread use of UML is to be promoted and sustained. While some research results indicate that by providing visual representations of the modeled system before development, use of UML diagrams can decrease the maintenance costs of software [2], other research are ambivalent about the benefits of UML modeling [7, 13]. Clearly, further research is needed to assess the viability of UML in all facets of software development including Internet-based software systems. Indeed, there is increasing demand for integrating business applications into the web [5, 9] and this has created a niche for programmers looking to enter the web development field. This increasing demand will attract many developers with varied software development maturity and experience, and as a result some quality concerns such as the cost of development and the productivity of the developers will arise.

To determine if productivity and cost concerns can be mitigated by using UML diagrams a small to medium sized web application was developed and use and effectiveness of UML was assessed. The goal of the research is to determine if UML diagrams are a viable solution to mitigating these quality concerns. Section 2 outlines the research approach, section 3 describes our case study and section 4 presents the research results. Discussion and conclusion are presented in section 5.

2 Method

There are fourteen diagram types in the UML 2.2 specification [17]. Of these, class diagrams, sequence diagrams and use cases are three of the most popular. To this end class diagrams, sequence diagrams and use case diagrams were selected as the diagram types to be assessed in the study. A
web based application was developed and measurements were recorded of the time taken to: learn each diagram type, produce the UML diagrams, and write code. Examples from previous work were used as a template [1]. The capacity of UML diagrams to aid customer understanding of the application was evaluated by presenting the customer with each diagram and recording the diagram type that the customer indicated provided the greatest clarity. At a later date after the program has been released to the customer, the amount of bugs will be tracked so that resulting defect data may be compared to results from previous projects.

3 Case Study

The initial requirements for the application were mapped out by direct verbal interaction with the customer. Subsequently, ambiguities and inconsistencies were clarified through telephone conversations with the customer. The application requirements are described in the paragraphs that follow.

The target application will create a full access administrator to manage a point of sale (POS) web-interface. The program will allow the administrator to assign accounts to specific sales staff and to manage sales staff user names and passwords. This administrator will be responsible for assigning each account to the appropriate sales staff. A single full
Access administrator will be able to add, modify and delete accounts; add, modify and delete sales staff; add, modify and delete sign shops; and add, modify and delete available products and product categories. A class diagram for the application is shown in Figure 1, a use case diagram for the application is shown in Figure 2 and the database diagram is shown in Figure 3.

Accounts include fields for name, account number, and address. Sales staff profiles include name, user name, and password. Each staff member will be assigned to a specific Branch. The administrator is responsible for ensuring that the sales staff is assigned to the appropriate branch. Sign shop profiles include name, user name, and password. Available products include fields for name, size, description, supplier, brand, type and quantity. Sales staff are responsible for placing the product orders through the POS. Products will also be marked either active or inactive. Product categories are a mechanism for organizing products. Each product will be assigned to a specific category. Figure 1: Use Case Diagram

3.1 Managing Sales and Sign Shops

A sales staff must have the ability to access the web interface using the assigned user name and password. When a salesperson logs into the system, they will see only the accounts that the full-access administrator has assigned to them. Sales staff must be able to filter and organize accounts by name and account number as well as view past point-of-sale orders for accounts. Additionally, they will have the ability to place a new order by selecting merchandise available from the Sign Shop. The full-access administrator is responsible for ensuring that all necessary products are available in the system. When the salesperson completes an order, this is saved in the account and an email, with the order, is sent to the Sign Shop administrators.

There must be multiple sign shops as determined by the customer. Each Sign Shop will be responsible for completing the orders sent via email from the sales staff and will be able to access the web interface via a specific user name and password. The Sign Shop administrators will assign orders to their specific shop. Upon completion of an order, the Sign Shops will be responsible for marking orders as complete. When an order is marked as complete, the web-interface will send a notice via email to the appropriate sales staff noting the completion of the order.

3.2 Learning UML

The first step in this project was to source documentation to learn the syntax and semantics of UML class, sequence and use case diagrams. While this step is not normally addressed in research articles, learning is often part of the development process and should be accounted for in such cases. The approach taken to learning what the diagrams represent was to find examples in published literature. However, different publications presented the UML diagrams in slightly different ways. Analysis shows that UML is used rather loosely in practice, with model defects such as incompleteness and inconsistency resulting in miscommunication and misinterpretations, and exacerbating the need for quality improvement [11]. The diagrams for this application were produced using Microsoft Visio 2007 [6] and examples found in research articles [1].

4 Results

4.1 Use Case Diagrams

Figure 4 shows the time taken to learn use case syntax and semantics and to develop the use case for each actor in the diagram. Overall, the data show that seven hours were spent of which about forty percent was spent learning use cases. The customer indicated that the use case diagram was more useful but did not provide any additional information about what the software did. The implementation of code did not benefit from this diagram because of the same reasons. This may be
because the application being developed was so simple that this diagram did not represent a higher level of abstraction.

4.2 Class Diagrams

A class diagram represents the structure of the application by showing the application’s classes, their attributes, and the relationships between the classes see Figure 1. Performance data for class diagrams is shown in Figure 5. It took approximately two hours to learn the syntax for the class diagrams. The time taken was much shorter than the other diagrams probably because class diagrams are more commonly used. The customer found the class diagram (Figure 1) to be useful in that it showed what fields would be stored within each entity. Customers were informed that forms representing fields would be provided to them to enter in the appropriate information.

The class diagram was very useful when implementing the code because it allowed the implementor the ability to develop each class separately without omitting any class. In the past, the developer would create applications without the use of any model and so everything was in the developer’s mind, which is much more amenable to omissions. However, for this application, the way the diagram breaks the application down into separate classes made it easy to reduce the overall complexity by creating methods within the classes. It is the opinion of the developer, that the class diagram was the most useful model overall.

4.3 Sequence Diagrams

The project proposal described earlier was used to create sequence diagrams. The sequence diagrams were modeled to describe in detail the interactions Sign Shops and Sales Staff would have with the application. This was required mainly because the detail behind an action was not revealed in the project proposal or use case and therefore required further clarification to the user. The sequence diagrams showed the exact actions that would need to take place and what responses would occur as a result of the actions. Sequence diagrams were created for login into the system, placing an order, assigning an order and updating an order. Figure 6 shows the update order sequence diagram. Performance data for sequence diagrams is shown in Figure 7. The time taken to learn and create the sequence diagrams was about seven hours of which three hours were dedicated to learning and reading about the syntax of sequence diagrams. When presented to the user they were minimally effective because the user did not find the syntax of the diagram user friendly. However, the diagrams were a significant asset when creating the action event code.

4.4 Code Implementation

The application was developed in classic ASP [15] using an object oriented approach. This particular language was chosen because of the reduced complexity versus ASP.NET [18, 15]. It is simpler because all the code is visible to the programmer and it is not based on a framework. Another reason ASP Classic was used is because the web applications that this application will be compared to were also developed in ASP Classic. Developing in ASP.NET could skew
the results because the code implementation would have been much faster and less error prone due to the framework.

It took 27 hours to develop the code resulting in about 3000 lines of code. In comparison, it took 22 hours to develop all the diagrams for the application. Education time for UML diagramming contributed largely to the total diagram time at eight hours total.

5. Discussion and Conclusion

The results indicate that for small web applications, where a developer chooses to model the application before coding, in cases where the modeling language is unfamiliar to the developer, learning the syntax and semantics of the diagrams to be used can significantly affect the overall cost of developing the application. In the present study, the learning component accounted for almost fifty percent of the overall cost of development.

This research suggests that UML diagrams in small sized web applications may not be viable due to the simplicity of the applications. In our study, the time taken to create the diagrams increased the project cost by some 82%. Even without the time for learning UML, modeling itself increased the overall cost by 70%. However, before a consensus can be realized as to the benefit or lack thereof of UML, defect data for this application need to be compared with similar applications developed without modeling. It is well established that minimizing of lost productivity and the time to correct errors is a fundamental goal of all software stakeholders. If code developed from design models results in less defects than code developed without models, then the used of models may be a viable option even for small applications. On the more positive side, the customer’s understanding of the application was enhanced by using the diagrams and there was a noticeable
increase in customer confidence resulting from use of models which is certainly a good thing from a developer and marketing standpoint.

Even without defect data for this application, a fair estimation is still possible in terms of the cost savings resulting from a potential lack of defects in the code. For example, five bugs per 1000 lines of code (LOC) is normal in software applications [4]. Therefore, with 3000 LOC, this application may have some 15 remaining bugs. Based on in-house information gathered from previous applications of a similar size, it takes 15 to 30 minutes to fix each bug. Based on this information, the average time to fix each bug would be around 22 minutes. In total then, it would take around six hours to fix these bugs. The important question then, is whether the time spent learning UML and modeling the application could have resulted in a smaller defect density, if the time were spent testing the software instead. Of course, even if use of UML for smaller applications increases the cost of development, many organizations may be willing to accommodate the increased cost if the resulting code has a smaller defect density, since a smaller defect density may correlate with smaller productivity loss due to software failure resulting from defects.

Other factors may also have an impact on the question of the viability of UML. For example, different types of small applications will vary in complexity and so may yield different results. Some studies have shown similar findings but they are not as focused on small sized web applications as this research. For example, another study has shown that UML diagrams do not reduce overall effort in software maintenance when updating the diagrams are taken into account [3]. Another important fact to consider since this research is based on software quality is that software organizations invest about 80% of their development resources for issues related to productivity [10]. When this is taken into account, the UML diagrams in this application do seem viable.
References


