Teaching Object-Oriented Software Design within the Context of Software Frameworks

Zoya Ali, Joseph Bolinger, Michael Herold, Thomas Lynch, Jay Ramanathan, Rajiv Ramnath
The Ohio State University, aliz@cse.ohio-state.edu, bolinger@cse.ohio-state.edu, herold@cse.ohio-state.edu, lynch.268@osu.edu, jayram@cse.ohio-state.edu, ramnath@cse.ohio-state.edu

Abstract – **Object-oriented software design and programming is an essential part of a computer science curriculum. We have observed that novice software developers, such as fresh college graduates who have been taught object-oriented design, are able to apply good design principles in theory. However, this rarely extends into their professional practice, when they are asked to design software intended to run inside a software framework. In fact, we observe that even advanced software developers abandon good design practices when developing software while using a framework, and focus on simply "making it work." This paper presents and discusses a methodology developed for designing software in the context of frameworks to overcome these issues. We show how *design patterns* can serve as the bridge between the paradigms imposed by the framework and the ideal, unconstrained design of the system. We also suggest an evaluation method for observing the results of using this methodology when used by the students.**

Index Terms – Design patterns, Mobile applications development, Object-oriented design, Software frameworks

INTRODUCTION

Object-oriented (O-O) software design and programming is an essential part of a computer science curriculum. The idea behind O-O design is that programs are intended to solve problems in the real world; thus basing software components on real world objects will make the software easier to both figure out (analyze) and design. In all the CS curricula that we have examined we have found that O-O design is taught with the intention of using it to develop software using an O-O language – such as C++, Java, C# and so on.

However, little software of any consequence is developed directly using only a programming language. Most commercial software is being developed using a software *framework* by extending and customizing the default, generic functionality that it provides. We have observed that novice software developers, such as fresh college graduates, who have been taught object-oriented design are able to apply good design principles in theory, but rarely in professional practice. When they are asked to design software intended to run inside a software framework, such as .NET, J2EE, or the Android SDK, they often fail to apply O-O design principles. In fact, we observe that even software developers who are not novices abandon good design practices when developing software while using framework, and focus almost all of their energy on simply "making it work."

The rationale behind using object-oriented design when developing an application is to conform the program to a real world problem. Basing software components on real world things makes the software easier to understand as well as design. Further, since the components have been modeled around the real world, they are not only more likely to be stable (as the real world is not changing quickly), they also are more capable of capturing the evolving nature of the application components.

Frameworks provide means for developing applications where functionality and architecture can be reused across multiple applications. While an object-oriented framework is a set of collaborating object classes that embodies an abstract design and provides solutions for a family of related problems, a framework specific to an environment, such as mobile devices, provides design solutions around device capabilities and services. A framework typically consists of a mixture of abstract and concrete classes. Abstract classes usually reside in the framework, while the concrete classes reside in the application. A framework, then, is a semi-complete application that contains certain fixed aspects common to all applications in the problem domain, along with certain variable aspects unique to each application generated from it [1].

We can say that design patterns and frameworks not only help in understanding and constructing real world applications, but also provide additional reusability of both design and implementation. We can selectively override or modify certain functionalities of the framework to provide specific functionality. This is achieved by using a set of object-oriented techniques to achieve "inversion of control". Inversion of control means that the program or, more precisely, the architecture of the framework enables customization of the application processing by event handler objects invoked through frameworks mechanisms. For example, a user's touch event to indicate his action will be handled by a concrete event listener that interfaces with the framework. When an event occurs, the framework dispatches control to the respective handler class by invoking its methods, which perform application-specific processing on the events. Inversion of control allows the framework, rather than each application, to determine which set of application-specific methods to invoke in response to external events [2].

In this paper, we present and discuss a methodology we developed for designing software in the context of frameworks. This methodology is intended to be taught to the students at our university. We show how *design patterns* can serve as the bridge between the paradigms imposed by the framework and the ideal, unconstrained design of the system. The methodology is aimed to help students come up with better designs for their applications when using any framework. We want students to not only learn the implementation of frameworks, but also learn to use frameworks to their advantage by exploring the framework itself.

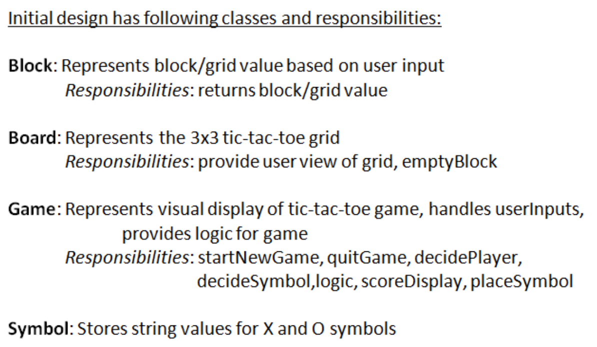
APPROACH

In this paper we explore how to teach students in colleges to write good software. In particular, we want to teach them to write software written within the Android SDK framework. The main reason we chose the Android framework was to let students develop applications which they can not only develop, but also publish them in the Android market for others to use. This is different from traditional curricula, as students get to see the progress of their application in a real market. Also, since Android is an open development platform, it enables users to develop rich and innovative applications that take advantage of both device capabilities and framework APIs [3]. However, simply using the framework guidelines does not help build a well-designed application, as frameworks provide only generic guidelines. We need to use design patterns that result from application design to overcome complexities and serve as a bridge between paradigms imposed by the frameworks and the ideal and unconstrained design of the applications.

To come up with the methodology we used one developer who was new to the Android framework. The developer was asked to develop a simple Tic-Tac-Toe game using the Android framework. We observed that, being a first time user of the framework, he found it to be complicated to fit their design easily into the framework. This led him to abandon good design principles and encouraged him to just fit their application into the framework. When we evaluated the code, the program was complex and difficult to understand, had tight coupling between classes and took lot of time to debug. Figure1. shows the initial design with classes and responsibilities.

Based on these issues in the design we decided that we needed to explain the framework in terms of design patterns. The initial difficulties faced were: (i) to understand the applicability of the framework to the application (ii) mapping of generic Android functionalities to specific application functionalities.

Thus, our instruction became a three-step process. First, we would have the students design their application through the use of object-oriented design. Second, we would have the redesign the application through the use of design patterns. Third, we have them adjust their use of the design pattern to match that within the framework.


Initial design has following classes and responsibilities:

Block: Represents block/grid value based on user input
Responsibilities: returns block/grid value

Board: Represents the 3x3 tic-tac-toe grid
Responsibilities: provide user view of grid, emptyBlock

Game: Represents visual display of tic-tac-toe game, handles userInputs, provides logic for game
Responsibilities: startNewGame, quitGame, decidePlayer, decideSymbol,logic, scoreDisplay, placeSymbol

Symbol: Stores string values for X and O symbols

FIGURE 1
INITIAL DESIGN

I. Designing with Object-Oriented Design

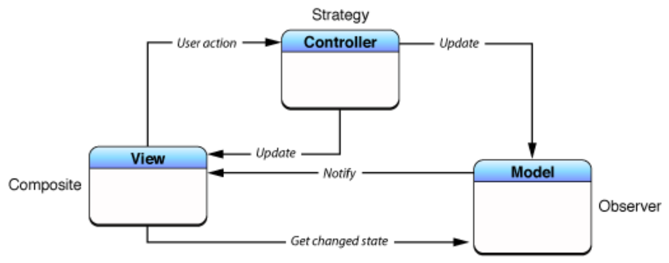
By using the problem statement, students are encouraged to think about it in the standard object-oriented way and use their previous experience to come up with objects, classes, responsibilities, collaborators, methods, etc.

II. Designing with Design Patterns

Then we introduce the common design patterns found within the Android framework. One such pattern is the model-view-controller (MVC) pattern [4]. MVC is a design pattern originally developed in the 1970s for Smalltalk. Today, it is predominantly seen in mobile and web applications. The idea behind this pattern is to isolate the domain logic of the application from the way the data is presented to the user (i.e. the application's user interface), so that these two very important components of any application can be designed, implemented and maintained separately. This can be seen in Figure 2.

As we observed in our example, the logic of playing the game Tic-Tac-Toe based on its rules forms the domain logic for the application. This is the *model* of the application. A visual representation of the Tic-Tac-Toe grid represents the *view* with which the users interact. The *controller* is a component, interposed between the model and the view, that receives the user actions and translates these commands to actions on the model. Then, the controller takes the resulting

FIGURE 2
THE MODEL-VIEW-CONTROLLER DESIGN PATTERN [5]



updates to the model and notifies the view that it has to update.

After teaching the students this design pattern, we can show the advantages of the technique. It is can be explained that by simply swapping out the view, one could easily change the entire presentation of the game without changing any of the backend of the controller or the model. By following this pattern, any redesign of the view can be easily done without affecting the underlying rules of the application. Thus letting the student think about the application in terms of the MVC pattern.

III. Resultant Pattern or Solution

Next, in order to better understand the framework and smoothly amalgamate our design of the application with the framework, we need to educate the students about the design pattern as dictated by the framework. In order to integrate the application into the framework, one would need to understand the framework and compare it with a standard design pattern. In our case of using MVC, we describe the implementation of the pattern within the Android framework and how one can develop a mapping from the MVC reference to the version implemented within the framework.

Showing the students how the pattern works within the Android framework helps them learn two independent things. First, how they can leverage the pattern within the framework, which is an important part of learning Android development. Second, gaining complete understanding of the pattern by seeing how it is implemented within a larger system. By comparing the Android MVC implementation with their own, it shows how the Android developers think differently about the pattern than students do. This is an important part of internalizing the use of design patterns.

METHODOLOGY

The Android developers have developed a distinct set of patterns for use by application developers. These are: Activities, Services, Broadcast Receivers, and Content Providers [3]. We will briefly describe these patterns to provide background for this section.

- **Activities** correspond to a particular "screen" and they are invoked directly or indirectly through Intents.
- **Intents** are a mechanism for calling a method, passing data, or receiving results.
- **Services** are application components that run in the background and do not require a user interface or interaction with the user. Two examples are playing music in background and getting user location from GPS hardware.
- **Broadcast Receivers** enable an application to interact by receiving data from other applications. For example receiving call while the application is running.

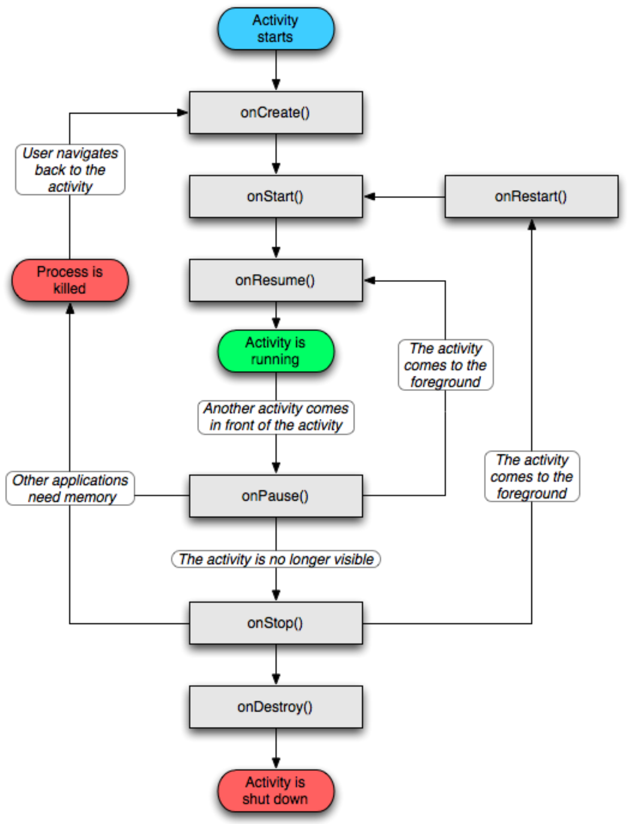

Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
Other applications need memory
onPause()
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

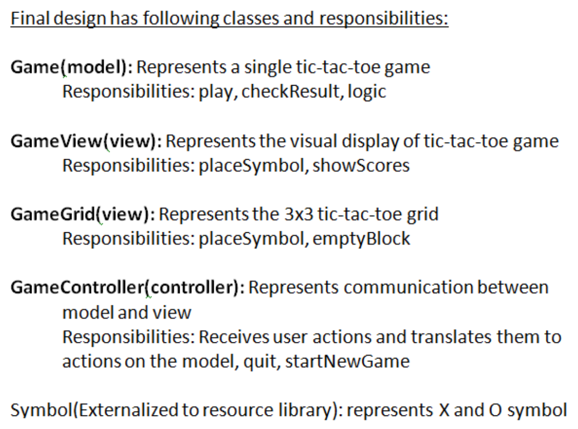
FIGURE 3
THE ACTIVITY LIFECYCLE WITHIN AN ANDROID APPLICATION [3]

- **Content Providers** globally expose and update persistent data as a simple table on a relational database.

Given these patterns, developing an application for the Android framework is very different than developing a generic application. Thus, we needed to provide a methodology to the students for converting their object-oriented design into an Android application design.

We came up with the following methodology to be used by students in order to take advantage of the framework in their design implementation:

1. Paraphrase the problem statement and extract all the nouns and verbs from it. The nouns serve as candidate objects, classes and attributes, while verbs serve as responsibilities.
2. Merge the extracted nouns into classes. This may require discarding irrelevant nouns or nouns representing the same thing.
3. Merge extracted verbs into classes, instances and responsibilities.
4. Assign responsibilities by identifying required methods to complete those responsibilities.



Final design has following classes and responsibilities:

Game(model): Represents a single tic-tac-toe game
Responsibilities: play, checkResult, logic

GameView(view): Represents the visual display of tic-tac-toe game
Responsibilities: placeSymbol, showScores

GameGrid(view): Represents the 3x3 tic-tac-toe grid
Responsibilities: placeSymbol, emptyBlock

GameController(controller): Represents communication between model and view
Responsibilities: Receives user actions and translates them to actions on the model, quit, startNewGame

Symbol(Externalized to resource library): represents X and O symbol

FIGURE 4
FINAL DESIGN AFTER USING THE METHODOLOGY

5. Walk through the scenario to ensure that each scenario is supported by methods and identify the collaborations between them.

These steps are the same as standard object-oriented design. To explain the compromises made with respect to framework and design, we gave them three additional steps.

1. *First*, once the classes have been identified, developers need to identify each activity corresponding to a certain view and set of actions involved in that view. By referencing the lifecycle of an activity within the Android framework, as seen in Figure 3, the developer needs to divide the define classes and methods from their object-oriented analysis and integrate them into the classes, as described in the activity lifecycle.
2. *Second*, map view classes with controller classes and model classes with controller classes.
3. *Third*, externalize resources such as images and strings from the application code in order to maintain them independently. Externalizing your resources also allows you to provide alternative resources that support specific device configurations, such as different languages or screen sizes. This becomes increasingly important as more diverse Android-powered devices become available. This step has a design pattern inherent in it. The framework directs you to organize resources in your project's res/ directory, using various sub-directories that group resources by type and configuration. [3] Figure 4 shows the final resulting classes and their responsibilities after using the methodology.

PROPOSED EVALUATION

We plan to evaluate the usefulness of this methodology by teaching it in a senior level software engineering class at The Ohio State University. With class size ranging from 30 to 35 students, application development will be done as individual assignments. We will use the Tic-Tac-Toe game application and the Android framework to teach the methodology. Later we will use a more complicated application to evaluate students' understanding of the framework and the methodology.

We will use questionnaires, individual implementation reports and a final exam to evaluate their understanding of the methodology and the implementation. We will record their learning curve by evaluating their second project implementation. We will use questionnaires to learn their previous experiences with frameworks and how it helped them in the mobile application development. We will record these twice: once prior to teaching the methodology and again after the methodology is taught. We will use individual reports to understand their implementation. Lastly, exam results will be used to know how well they understood the methodology.

CONCLUSION

In this paper, we illustrated how adaptation to a new framework can be a challenge for novice developers and how implementation of standard object-oriented designs into a new framework may lead to bad design and misuse of the framework.

Because of this, we developed a methodology that guides the students on how to productively adapt the generic guidelines as offered by the framework into the specific functionalities as required in their application. This methodology is intended to help the students overcome problems like overly complex design, tight coupling, a rigid controller function and low exploitation of the framework.

In this methodology, we encourage the students to use design patterns as described in their object-oriented application design and compare their designs to the framework. We use this comparison as a bridge to map the two. This not only resulted in better design of the application, but also gave students a new insight in how to approach a new framework and amalgamate their application into it.

We do not say that simply following framework design would result in a good application, as it requires compromising and adapting at both application level and framework level. These compromises allow a developer to come up with a final application design that is easier to understand, easier to debug, and allows code to be more maintainable and reusable.

In order to evaluate the effectiveness of this methodology, we proposed a method for evaluation. We will use several qualitative approaches, including project evaluations and questionnaires. We will also use quantitative approaches, including a study of exam results from the course.

REFERENCES

[1] Srinivasan, S., “Design Patterns in Object-Oriented Frameworks”, *Computer*, Vol. 32, No. 10, February 1999, pp. 24-32.
[2] Burbeck, S. “Applications Programming in Smalltalk-80: How to Use Model-View-Controller”, Softsmarts, Inc., 1987.
[3] Android Developer’s Guide. http://developer.android.com, retrieved Febuary 2011
[4] Fayad M., Schmidt C.D., “Object-oriented application frameworks”, Communications of the ACM Magazine Volume 40 Issue 10, Oct. 1997
[5] Apple Developer’s Guide. http://developer.apple.com, retrieved Febuary 2011

AUTHOR INFORMATION

Zoya Ali, Graduate Student, Department of Computer Science and Engineering, The Ohio State University, aliz@cse.ohio-state.edu

Joe Bolinger, Ph.D. Candidate, Department of Computer Science and Engineering, The Ohio State University, bolinger@cse.ohio-state.edu

Michael Herold, Ph.D. Student, Department of Computer Science and Engineering, The Ohio State University, herold@cse.ohio-state.edu.

Thomas Lynch, Ph.D. Student, Department of Computer Science and Engineering, The Ohio State University, lynch.268@osu.edu

Rajiv Ramnath, Director, C.E.T.I., Associate Professor of Practice, Department of Computer Science and Engineering, The Ohio State University, ramnath@cse.ohio-state.edu

Jayashree Ramanathan, Director, C.E.T.I., Associate Research Professor, Department of Computer Science and Engineering, The Ohio State University, jayram@cse.ohio-state.edu