Evaluating Domain-Driven Architectural Designs and Non-Functional Architectural Attributes for Windows Phone 8 Mobile Applications

Bachelor of Science Thesis in Software Engineering and Management

MOZHAN SOLTANI
RETTA SHIFERAW SIYOUM

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, June 2013

Evaluating Domain-Driven Architectural Designs and Non-Functional Architectural Attributes for Windows Phone 8 Mobile Applications

Examiner: MICHEL CHAUDRON

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2013

Evaluating Domain-Driven Architectural Designs[*] and Non-Functional Architectural Attributes for Windows Phone 8 Mobile Applications

Mozhan Soltani
Software Engineering and Management
Dept. of Computer Science and Engineering
Chalmers University and University of Gothenburg
mozhan.soltani@gmail.com

Retta Shiferaw Siyoum
Software Engineering and Management
Dept. of Computer Science and Engineering
Chalmers University and University of Gothenburg
rsabitirta@gmail.com

ABSTRACT

Most IT companies are interested in investigating new mobile technologies as mobile devices are noticeably prevalent these days. In this study, we collaborated with Volvo IT to identify the extent of conformance to domain-driven architectures when developing Windows Phone 8 (WP8) applications, as well as to find out what non-functional attributes can be applied to these applications.

We took the Action Design Research (ADR) strategy to develop a purchasing order system prototype and investigate the applicability of security, performance, and maintainability to Windows Phone 8 applications. We found that while the Model-View-ViewModel (MVVM) pattern brings high maintainability to WP8 applications, these applications can still conform to a domain-driven architecture with, at least, the user interface, domain, communication, and service components, and fulfill high levels of security as well as performance.

Due to the limited time frame of the study, we did not consider other non-functional attributes, such as integrability, robustness, and simplicity, for WP8 applications. Those attributes can be investigated in future research.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures, Patterns, Model View View-Model pattern*
; D.4.6 [**Operating Systems**]: Security & Protection
; D.4.8 [**Operating Systems**]: Performance

[*]Domain-driven architecture is an architectural design that is focused on the core domain and domain logic. In this archutecture, the design is based on a model of a domain and the model is a system of abstractions that describes certain aspects of a domain. We will elaborate on domain-driven architectures in section 3, Architectural Designs.

Keywords

Windows Phone 8, Domain-Driven Architecture, Model View View-Model Architecture, MVVM

1. INTRODUCTION

In recent years, mobile devices have become noticeably ubiquitous and the number of mobile operating systems has increased. Therefore, mobile devices have become a prevalent platform for business applications. Many companies adopt mobile technologies in order to increase their responsiveness to meet a wider range of customer demands [13, 18, 20, 21]. To adopt the mobile technologies, companies need to investigate certain technical aspects so that the new mobile technologies would conform to their standards and principles. For instance, these companies typically have particular development standards and architectural principles to conform in their software applications in order to be highly consistent and time efficient.

Similar to other IT companies, Volvo Group IT Governance promotes a software architectural framework that consists of two parts:

- A generic domain-driven architectural design called Volvo Group Target Architecture (VGTA)
- A set of non-functional attributes, including security, performance, and maintainability

Due to the generic structure of this framework all the applications at Volvo Group are supposed to conform to it, regardless of the domain or platform they are intended for. Therefore, it is essential to realize the extent to which new mobile technologies can conform to this framework.

In this study together with Volvo IT, as the industrial partner of the research, we intend to find out the extent of conforming to domain-driven architectural frameworks while developing Windows Phone 8[1] applications. We also intend

[1]Windows Phone 8 (WP8) is the second generation of the

to identify the non-functional attributes that can be applied to these applications. Therefore, we ask the following research questions:

1. To what extent is it feasible to conform to a domain-driven architectural framework, which includes the user interface, domain, communication, and service components, while developing a Windows Phone 8 application?
2. What non-functional attributes can be applied to Windows Phone 8 applications?

To answer the questions, we took the action design research strategy [4, 16] to develop a Windows Phone 8 application, and evaluate it against VGTA. The application conforms to the MVVM[2] architectural pattern and provides the main functionalities of a typical purchasing order system; create, update, read, and delete order.

Due to the limited time frame of the study, we considered a subset of the architectural attributes that were included in the architectural framework. We looked into the applicability of security, performance as well as maintainability, and left the rest for the future studies.

The report is structured as follows: In section 2, we will explain why and how we followed the action design research methodology to develop the prototype and obtain the intended knowledge. In section 3, we will present knowledge about domain-driven architectures and the Volvo Group architectural framework, as well as explain the MVVM architectural pattern and the architectural advantages that it supports. Research findings will follow, in which we will explain the architectural design and attributes of the application prototype as well as the architectural attributes that Microsoft supports for Windows Phone 8 applications. In section 5, we will answer the research questions, and at the end, in section 6 we will provide our conclusions.

2. RESEARCH METHODOLOGY

In the following sections, we motivate the choice of the Action Design Research (ADR) strategy (Section 2.1) and present the research process (Section 2.2) that we used, following ADR, to frame the research problem, develop the purchasing order system prototype, and draw the final conclusions.

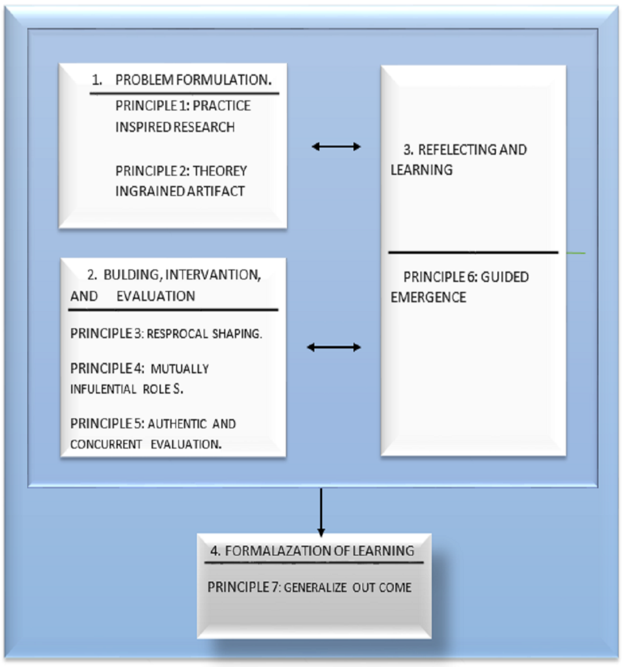
2.1 Motivation for ADR

To perform the qualitative evaluations [14] and answer the research questions, we found Action Design Research (ADR) [4, 16] to be a suitable strategy through which we could conduct iterative developments and evaluations, as well as contribute to a general knowledge about the Windows Phone 8 applications. The following two main incentives formed the basis for choosing ADR:

- ADR requires the researchers to conduct iterative and inseparable building, intervention, and evaluation during the second stage of the research process. This phase leads to gaining an in-depth understanding of organizational values, expectations, and assumptions. In addition, the researchers can benefit from the iterative reshaping practices to validate their findings and increase the credibility of the study [16].
- ADR requires the researchers to generalize the outcomes of the study, during the fourth stage of the research process, and contribute to the knowledge about a class of field problems, rather than challenges that are specific to an organization [16].

Windows Phone mobile operating system from Microsoft. It was released in October 2012, and is among the popular mobile platforms nowadays [24].

[2]MVVM is an architectural pattern used in software engineering that originated from Microsoft as a specialization of the presentation model design pattern [7, 5]. It facilitates a clear separation of the development of the graphical user interface from the development of the business logic or back end logic known as the model. This pattern is recommended for developing Windows Phone 8 applications.



1. PROBLEM FORMULATION.
PRINCIPLE 1: PRACTICE INSPIRED RESEARCH
PRINCIPLE 2: THEOREY INGRAINED ARTIFACT
2. BULDING, INTERVANTION, AND EVALUATION
PRINCIPLE 3: RESPROCAL SHAPING.
PRINCIPLE 4: MUTUALLY INFULENTIAL ROLE S.
PRINCIPLE 5: AUTHENTIC AND CONCURRENT EVALUATION.
3. REFELECTING AND LEARNING
PRINCIPLE 6: GUIDED EMERGENCE
4. FORMALAZATION OF LEARNING
PRINCIPLE 7: GENERALIZE OUT COME

Figure 1: Stages and Principles of ADR

2.2 Research Process

Following ADR, we took the following four steps: 1) Problem Formulation, 2) Building, Intervention, and Evaluation, 3) Reflection and Learning, and 4) Formalization of Learning. We elaborate on these steps in the following sections.

2.2.1 Problem Formulation

Typically IT companies expect a large number of requests from their customers with respect to new technologies. Therefore, they are required to investigate the new technologies in order to stay responsive and fulfil the customer and market requirements. The need for this investigation provided the impetus for the present research.

At the stage of the problem formulation, we collaborated with a mentor from Volvo IT to realize the problem scope, frame the research problem, and identify the solution possibilities. This stage drew on two principles: 1) practice-inspired research, which emphasizes on viewing the field problem when it comes to defining the research problem, and 2) theory-ingrained artifact, that emphasizes that ensemble artifacts, created and evaluated using ADR, are informed by theories [4, 16]. Conforming to these principles, we specified the research aims as well as the research questions (presented earlier in section 1) and formed the initial design theories. The design theories comprised of the following:

- Using the Windows Phone 8 emulator on which we could debug and run the prototype
- Conforming to the MVVM architectural pattern for developing the prototype
- Implementing the basic functionalities of create order, read order, update order, and delete order in the prototype

2.2.2 *Building, Intervention, and Evaluation*

At this stage, we perceived the formulated problem as well as the theoretical premises that were adopted in the former stage and began implementing the application prototype. We used secondary data sources [1] (books and articles on MVVM) to realize the MVVM architectural pattern and follow it in the prototype. This stage drew on three principles of 1) reciprocal shaping, 2) mutually influential roles and 3) authentic and concurrent evaluation, which together emphasize the inseparability of building and evaluation in this strategy [4, 16]. Following these principles, in order to develop an in-depth understanding of expectations from Volvo IT and apply them appropriately, we delivered multiple versions of the prototype iteratively. In each iteration we, together with our mentor from Volvo IT, evaluated the application and planned for further reshaping until all four functionalities of create, read, update, and delete order, as well as communication with a backend for saving and receiving the orders were implemented properly.

2.2.3 *Reflection and Learning*

As figure 1 illustrates, this stage occurs in parallel with the first two former stages. It moves conceptually from building a solution to a particular problem to applying that specific knowledge to a broader range of problems [4, 16]. Reflection and learning draws on the guided emergence principle which emphasizes that the ensemble artifact reflects both the preliminary design and its ongoing reshaping. At this stage, we used secondary data sources about VGTA (official documents from Volvo IT) to gain an understanding about specific architectural qualities that it imposes on the applications. In addition, we studied the architectural attributes that Microsoft supports for WP8 applications. We ensured that the contribution to knowledge is identified by conscious reflection on our research questions and design theories, in terms of MVVM, VGTA, and the associated architectural principles, as well as evaluating the emerging prototype against the reflected knowledge.

2.2.4 *Formalization of Learning*

At this stage, we used secondary data sources (mainly articles) to gain knowledge about domain-driven architectural frameworks. We then generalized the study and contributed to the field knowledge about architectural perspectives of Windows Phone 8 applications. We conducted the generalization by casting VGTA to a domain-driven architectural framework and answering the first research question on the extent of conformance to such frameworks when it comes to developing Windows Phone 8 applications. In addition, we answered the second research question on the applicable non-functional attributes by identifying the architectural attributes that we could address in the application prototype as well as those that Microsoft supports for WP8 applications.

3. ARCHITECTURAL DESIGNS AND ATTRIBUTES

This section is divided into two main parts. In the first part, we explain domain-driven architectures and the Volvo Group architectural framework, which consists of VGTA as an instance of a domain-driven architectural design as well as a number of non-functional attributes. In the second part, we explain the MVVM architectural pattern and the advantages that it brings to Windows Phone 8 applications.

3.1 Domain-Driven Architecture

Domain-driven architectures are used to develop software for complex needs and enhance specific domains. In the process of developing these architectural designs, design and development work together to create a better solution. Technical and domain experts collaborate in an iterative manner to promote a conceptual model that addresses particular domain problems. The primary focus of the technical and domain experts is on the core domain and domain logic. Good design accelerates the development, while feedback from the development process enhances the design. In this approach, domain is the subject area to which the user applies a software program, and model is a system of abstractions that represents the target domain and can be used to solve the domain-related problems [7, 9].

The model is an essential part of designing a domain-driven architecture to deal with the complexity of the project. All thinking processes about the domain are synthesized in the model, therefore the model must be communicated to the technical and domain experts precisely and completely so that the collected knowledge is shared among them. To express the domain model, team members should develop ubiquitous language around it and use diagrams, pictures, use cases, etc to spread the gathered knowledge in the model appropriately [7, 9].

For a successful design process, it is important that the project has access to domain experts, and the project team has experience and interest in object-oriented design and programming. In addition, communication is paramount for the success of the project when building a domain model. A core principle in the process of domain-driven design is to use a language based on the model. Since the model is the common ground, it is appropriate to use it as the building ground for this language. This language is ubiquitous and appears consistently in all forms of the communication. It will create the premise for the design team to function well [7, 9].

In the process of building the model, patterns that are commonly used are referred to as building blocks of the domain-driven design [7, 9]. These patterns include layered architecture, aggregate, service, repository and factory. Moreover, typically domain-driven designs include, at least, user interface, domain, communication, and service components which have their own responsibilities in the software design.

In what follows (Subsection 3.1.1), we will explain the Volvo Group architectural framework and elaborate on Volvo Group Target Architecture (VGTA), which can serve as a generic domain-driven architectural design for the applications of Volvo Group. In addition, we outline the architectural principles that are associated to the non-functional attributes of the framework.

3.1.1 *Volvo Group Architectural Framework*

Volvo Group IT Governance promotes a software architectural framework that consists of a generic domain-driven architectural design as well as a set of non-functional attributes[3] that includes security, performance, and maintainability. In this section, we will explain the architectural design and the components that it embraces, as well as the architectural principles that are associated to the non-functional attributes of the framework.

3.1.1.1 Volvo Group Target Architecture.

The architectural design is called Volvo Group Target Architecture (VGTA) and is comprised of six types of components. The components are either technical or business-related logical constructs and consist of:

- User interface component
- Workflow component
- Domain component
- Gateway component
- Proxy component
- Utility component

In what follows, we explain the responsibilities and communication rules that are defined in VGTA for these components.

User Interface Component The user interface (UI) component contains UI presentation and UI workflow functionality. It uses services provided by the domain and workflow components. Services to be consumed by the user interface should be specifically designed to meet both performance and usability requirements of the user interface. Therefore, services provided to the user interface should be tailor-made

[3]Considering the time frame of the study, we decided to focus on a subset of these attributes which we found the most addresable.

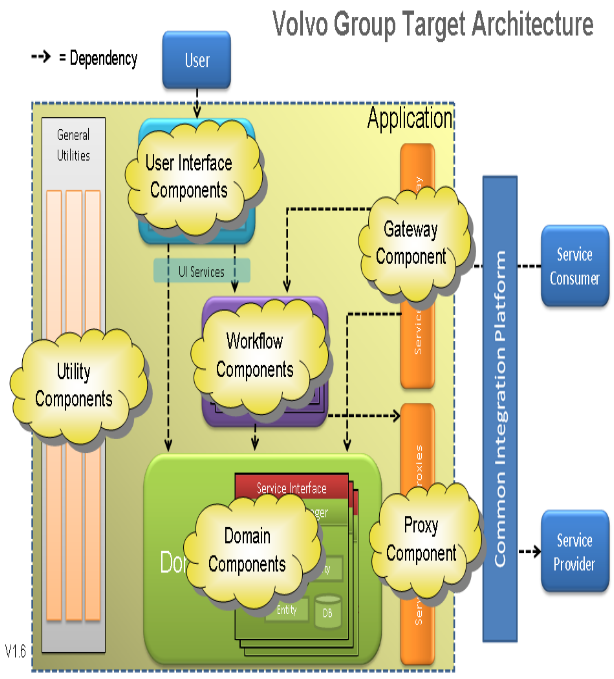


Figure 2: Volvo Group Target Architecture

for this component to avoid a talkative protocol and make the development process more efficient.

Workflow Component A workflow component is merely responsible to coordinate a sequence of actions that are needed to move a workflow forward. This component spans over multiple domain components to complete a business action, however it must not contain any business logic itself. The functionality of a workflow component involves calling a number of different services from domain components. It can also call services from other workflow components depending on the required action to be performed. To maximize scalability and promote simplicity, it is recommended to use asynchronous messaging when designing a workflow component.

Domain Component A domain component represents a set of business concepts, such as order and customer. The internal information model of a domain component is detailed in a domain model, e.g. an order domain model. Each domain component owns it's data by having separate databases or separate data schemas. Data schemas contain the tables corresponding to the domain model entities. The databases and the domain entities can only be reached through the service facade of the domain component.

It is important that the business logic within a domain component can change without affecting it's users. Therefore, the functionalities of a domain component should only be reachable through a service interface. The service interface can simply consist of a class interface that hides the internal behavior of the domain component. The following summarizes the rules to consider when designing a domain component:

- Each domain component can only be accessed through its service interface which hides the domain's internal structure.
- The services of a domain component are accessed through synchronous calls.
- Transformation of external information models into internal format should not be done in the domain component, it is the responsibility of gateway and proxy components.
- Only the domain component can change it's own data.
- A domain model shall not directly operate on another application's information model.
- Within the same application, no information model transformation is required.

Gateway Component Typically in an application only one gateway component is used which is responsible for receiving service calls and events from external sources, such as external applications. A gateway component does the necessary transformations between the messages received from external sources and the internal services of domain and workflow components. The information model of external sources should be isolated from the information models that are used internally by the domain and workflow components in the application.

Proxy Component A proxy component is responsible for accessing the external services outside of the application. While this component isolates it's consumers, workflow and domain components, from the external information model, it should not contain any business logic. A proxy component makes the necessary transformations between the internal information concepts and the information concepts of the consumed external services. The external data models are data descriptions, e.g. XML declarations, that provide the information of how to interpret data from an external application.

Utility Component The utility component delivers general functionalities and application-wide services that do not directly belong to any domain component within an application, e.g. security, and log. It can also contain base classes, utilities, constants and exceptions that should be shared application-wide and do not fit into any of the other component types. A utility component is a library of generic functionalities and should not contain any business logic.

3.1.1.2 Non-functional attributes. Non-functional attributes of the Volvo architectural framework address security, performance, and maintainability. In what follows, we outline the architectural principles that are associated to these attributes.

Security

- The platform to keep the sensitive information must be secured from malware attacks.
- Applications must be isolated from each other and can not access the memory used by other applications.
- It should be possible to develop and distribute the applications within an organization privately.
- Confidentiality and integrity of data must be protected from unauthorized data access or unintended information disclosure.

Performance

- Applications must have the least talkative communications with each other.
- Applications must have optimized memory usage.
- Applications must respond to the user within an acceptable time.

Maintainability

- Applications must contain autonomous components and support continuous functional modifications and improvements.
- Applications must support unit testing by promoting modularity.

3.2 MVVM Architectural Pattern

Model-View-ViewModel (MVVM) is a software architectural pattern that originated from Microsoft as a specialization of presentation model design pattern, and is recommended for developing Windows Phone 8 applications [2, 5]. MVVM helps cleanly separate the business and presentation logic of the application from its user interface (UI). This can make the application easier to test, maintain, and evolve. It can also greatly improve code re-use opportunities and allow developers and UI designers to more easily collaborate when developing their respective parts of the application [2, 5, 6]. Using the MVVM pattern, the UI of the application and the underlying presentation and business logic is separated into three separate classes:

- **View** encapsulates the UI and UI logic. It is a visual element, such as a window, page, user control, or data template that defines the controls contained in the view as well as their visual layout and styling. View is responsible to define the structure and appearance of what the user sees on the screen. It references the view model through its DataContext property. The controls in the view are bound to the properties and commands exposed by the view model. The code behind the view can define UI logic to implement visual behavior that is difficult to express in XAML or that requires direct references to the specific UI controls defined in the view [2, 5, 6].
- **ViewModel** encapsulates presentation logic and state, as well as relies on the binding system for communicating with the view. The view-model takes care of

moving data from the model to the view and communicating user gestures to the model from the view. It has no direct reference to the view or any knowledge about the view's specific implementation or type. The view model implements properties and commands to which the view can data bind and notifies the view of any state changes through change notification events. The properties and commands that the view model provides define the functionality to be offered by the UI, but the view determines how that functionality is to be rendered. Typically, there is a one-to many-relationship between the view model and the model classes. The view model may choose to expose model classes directly to the view, so that controls in the view can bind data directly to these classes. In this case, the model classes will need to be designed to support data binding and the relevant change notification events. The view model is testable independently of the view and the model [2, 5, 6].

- **Model** encapsulates the application's business logic and data. Business logic is defined as any application logic that is concerned with the retrieval and management of application data and for making sure that any business rules that ensure data consistency and validity are imposed. To maximize re-use opportunities, models should not contain any user task-specific behavior or application logic. Typically, the model represents the client-side domain model for the application. It can define data structures based on the application's data model as well as any supporting business and validation logic. The model may also include the code to support data access and caching, though typically a separate data repository or service is employed for this [2, 5, 6].

MVVM is largely based on Model View Controller (MVC) and Model View Presenter (MVP) patterns, which were the two most common architectural styles before MVVM emerged. This pattern addresses some of the limitations of MVC and MVP and combines some of their strengths [23]. In what follows we elaborate on MVC and MVP architectural patterns and explain the advantages of using MVVM over them.

3.2.1 Model View Controller

Model View Controller (MVC) is a software architecture pattern which separates the representation of information from the user's interaction with it [23]. This pattern consists of three components:

- **View** is responsible for displaying data and collecting user input. The view gets its data from the model including notifications that data has been updated and needs to be refreshed. These notifications are implemented using an observer pattern. When the user interacts with the view through gestures, the view is responsible for collecting those gestures and forwarding them along to the controller for processing.
- **Model** contains business entities and the data that UI displays. The model is responsible for notifying the view of changes in state which is generally done with an observer pattern.
- **Controller** is responsible for taking user input and communicating it to the model for processing. The controller typically implements a use case and alters the model depending on the user actions.

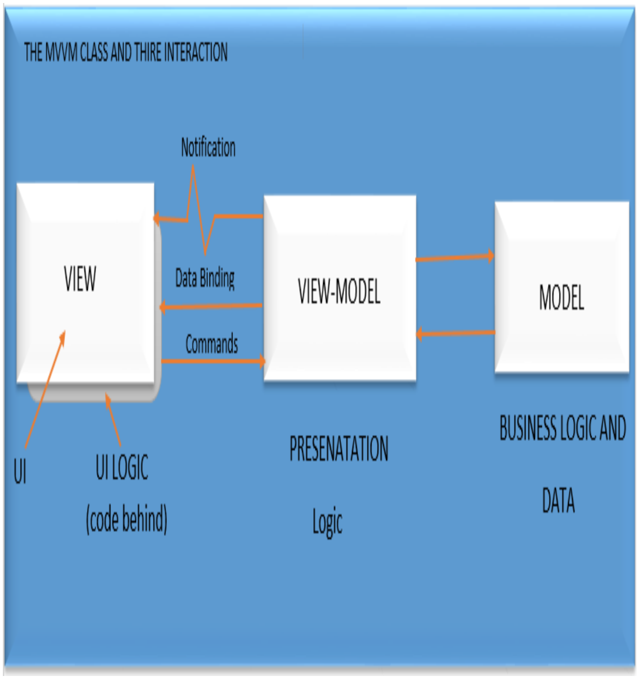


Figure 3: Model-View-View Model Architectural Pattern

In this pattern, the view can be synchronized with the model through using observers and subscribing to changes in model [23]. As figure 4 shows, the controller does not know anything about the view, however the view can switch between controllers and a single controller can be used by multiple views. This form of coupled relationship among the components of MVC brings disadvantages to this architectural style. Following MVC,

- the view logic and the view state are both tightly coupled in the view, making them difficult to test or share,
- only the code for the model can be re-usable, and
- memory leaks might occur when using .Net events. This is because in .NET, events only support using strong references and not weak references. When an observer object, view, subscribes to an event on a subject object, model, the subject keeps a reference in the form of a delegate (or function pointer) to the observer. In .NET, memory management is handled by the garbage collector and the garbage collector will not collect any object as long as another object has a strong reference to it. This means that the view subscribing to the model's events will cause those models to hold strong references to the views. These strong

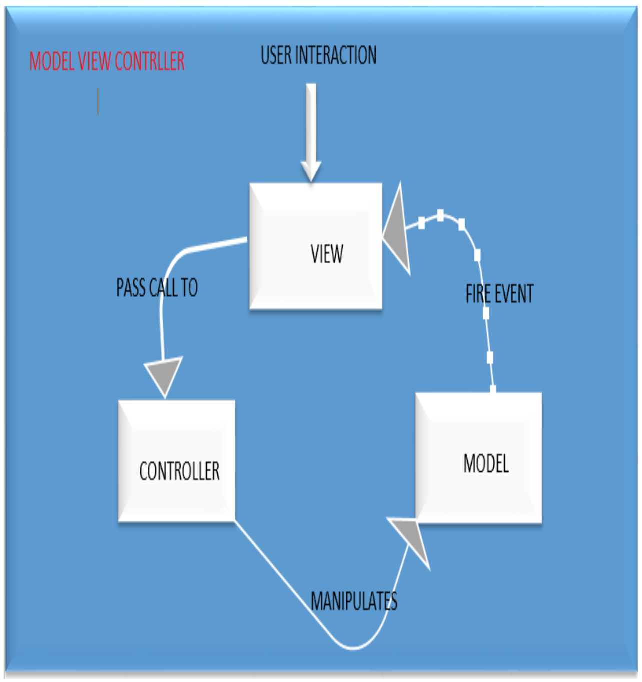


Figure 4: Model View Controller Architectural Pattern

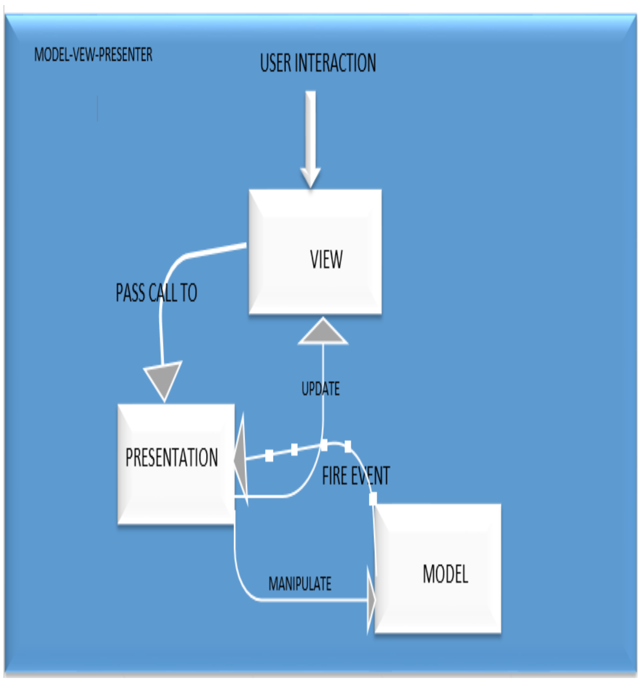


Figure 5: Model View Presenter Architectural Pattern

references will prevent the garbage collector from collecting the views, causing the views to leak memory.

3.2.2 Model View Presenter

The MVP pattern is a derivative of MVC that was engineered to facilitate automated unit testing and improve the separation of concerns in presentation logic [23]. MVP consists of three components:

- **View**, that is the user interface (UI) of the software. program
- **Model**, that is the actual data that the presenter will request and get displayed in the view. It is responsible for obtaining the data so it is the one that reads files or connects to a database.
- **Presenter**, which is an entity that can manipulate the model as well as present the data to the view and update it.

The goal of using the MVP design pattern is to separate the responsibilities of the application in such a manner as to make the application code testable and maintainable [23]. Moving view state and view logic to the presenter makes it much easier to factor logic and code out of the UI layer for more streamlined, and reusable code that is easier to test.

The main difference between MVC and MVP is that, in MVP the presenter refers back to the view while in MVC the controller does not do so. It is important to note that the presenter has no knowledge of the actual UI layer of the application. The presenter knows it can talk to an interface, but it does not know or care what the implementation of that interface is [23]. This promotes reuse of presenters between disparate UI technologies and makes it easier to mock the views and run unit tests. However, two shortcomings of MVP still remain:

- MVP uses a lot of events, and as we discussed for MVC, events can cause memory leaks.
- A lot of code remains untested in the view.

3.2.3 Advantages of MVVM

As we mentioned earlier, MVVM is largely based on MVC and MVP architectural patterns. In comparison with MVC and MVP, MVVM can have a number of advantages, including:

- **Increased testability**: Testability is improved as all view logic is now testable from unit tests [2].
- **Less code**: The amount of code required to manage the view decreases, as the developer no longer has to deal with boilerplate code behind code. This code involves a lot of casting and error checking in production quality code. Less code means fewer bugs, less code to maintain, and fewer unit tests to write [2].
- **Increased decoupling**: When using the pure approach, the developer no longer needs to have the view and mediator (view model, presenter, or controller) be explicitly aware of each other. The view does have a reference to the view model, however, under pure MVVM, it is not necessary for the view to be aware of the type of the view model [2].

- **Allows for streamlined development processes**: Developers and designers can work independently on the same application views. This is because of the decoupling in this pattern and also because the developer can create a view model that exposes the needed data points and have the view model properties return design-time data. This allows designers to work on the look of the application while the view model and model are being built [2].

4. RESEARCH FINDINGS

This section is divided into two main parts. In the first part, we explain the prototype in terms of its architectural design and the architectural attributes that it fulfills. In the second part, we explain how Microsoft supports security and performance attributes for Windows Phone 8 applications.

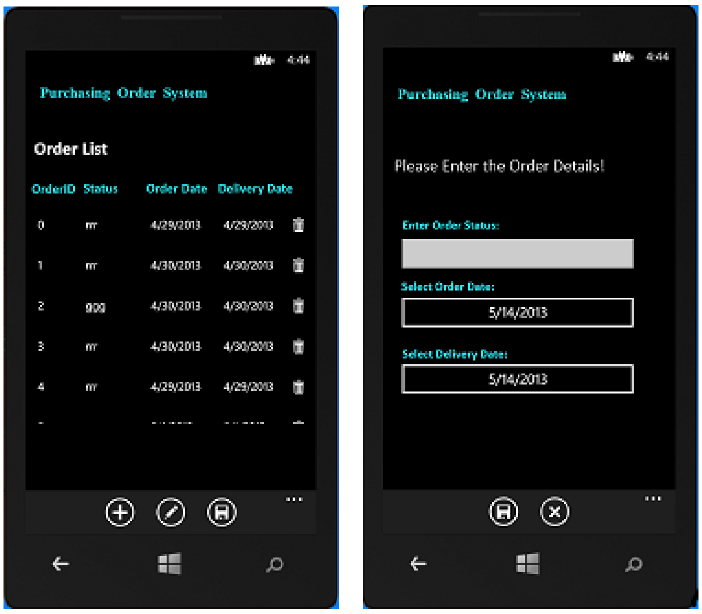
4.1 Purchasing Order System Prototype

A purchasing order system is designed to help manage the purchasing and receiving of stock for a business, and also prevent unauthorized purchases from taking place. To evaluate the extent of conformance to a model-driven architectural design and identify the applicable architectural attributes to a Windows Phone 8 application, we developed a Windows Phone 8 application called Purchasing Order System (POS). The POS application provides the functionalities of create order, read order, update order, and delete order, as well as communication with a backend to save and retrieve the data about the created orders. When the application starts, it communicates to the backend to retrieve the orders that the user might have already saved. The user can then see the list of orders, update them, or add a new one. To add a new order, the user should specify order date, delivery date, and order status. In addition, each order can be associated to multiple parts that the user can specify at the time of creating or updating the order item. To specify the parts, the user should provide the part name and part quantity either at the time of creating or updating an order. When the user terminates the application, the list of orders that he has made will be sent to the backend to be saved.

4.1.1 Prototype Architecture

To develop the application, we used the Visual Studio 2012 development environment, from Microsoft, which supports different programming languages such as C, C++, and C# [17]. We followed the MVVM architectural pattern, due to the certain architectural advantages that it supports for Windows Phone 8 applications, and separated the application UI from application logic. We made the application UI in *XAML*[4], using the Microsoft Blend tool, and wrote the application logic in C#. In addition, we used the Windows Phone 8 emulator[5] to debug and run the application. The prototype consists of the following components, from the MVVM perspective:

- **View**: This component is responsible for the application presentation and contains the seven pages of the prototype. The pages are xaml files that include UI controls and are associated to xaml.cs files. Xaml.cs files are responsible for UI logic as well as the visual behavior of the controls, and include the code-behind of the pages. In addition, the controls of these pages are data bound to the properties that the view model exposes from the model of the application.



Order List

OrderID	Status	Order Date	Delivery Date
0	rrr	4/29/2013	4/29/2013
1	rrr	4/30/2013	4/30/2013
2	ggg	4/30/2013	4/30/2013
3	rrr	4/30/2013	4/30/2013
4	rrr	4/29/2013	4/29/2013

Purchasing Order System

Please Enter the Order Details!

Enter Order Status:

Select Order Date: 5/14/2013

Select Delivery Date: 5/14/2013

Figure 6: The Order List and Create Order Pages of the POS Application

- **ViewModel**: This component is responsible for transferring the order data and part data to the pages of the application. Using C# code, it implements the *INotifyPropertyChanging* and *INotifyPropertyChanged* interfaces to update the controls of the pages when the data is changed in the model component. It also communicates user actions, for creating, reading, updating, and deleting the orders or their associated parts, to the model of the application. Using interfaces, the view model is loosely coupled to the other components of the application, therefore, it is simple to unit test, modify, and re-use the code of this component.

[4]Extensible Application Markup Language (XAML) is a declarative XML-based language created by Microsoft [3]. XAML elements map directly to Common Language Runtime object instances, while XAML attributes map to Common Language Runtime properties and events on those objects. XAML files can be created and edited with visual design tools like Microsoft Expression Blend, Microsoft Visual Studio, and the hostable Windows Workflow Foundation visual designer. They can also be created and edited with a standard text editor, a code editor like XAMLPad, or a graphical editor like Vector Architect.

[5]Windows Phone Emulator is a desktop application that emulates a Windows Phone device [22]. It provides a virtualized environment in which one can debug and test Windows Phone applications without a physical device. It also provides an isolated environment for application prototypes.

In addition, the view model of the POS prototype includes a separate component, called Order Service. This component includes the services for communicating with the backend and uses the HttpClient[6] library to send HTTP requests and receive HTTP responses asynchronously. This library supports portability across the Microsoft platforms, therefore, using this library adds to the code re-use possibilities of the application.

- **Model**: This component contains the application data, related to the created orders and parts, and is responsible for managing this data, using C# code. It uses the LINQ to SQL[7] framework to map the data objects to SQL relational tables. The model includes two data schemas, order and part, that define the properties of order and part entities. The following code snip shows part of the order data schema in which "orderId" is defined as a primary key:

```
namespace SamplePOS.Model
{
 [Table(Name = "Orders")]
 public class Order :
 INotifyPropertyChanged ,
 INotifyPropertyChanging
 {
  public Order()
  {
   partts = new EntitySet<Part>(
   rev =>
   {
    NotifyPropertyChanging("Partts");
    rev.Order = this;
   },
   rev =>
   {
    NotifyPropertyChanging("Partts");
    rev.Order = null;
   }
   );
 }

// Define order id
private string orderId;
[Column(IsPrimaryKey = true ,
DbType = "nvarchar(20)",
CanBeNull = false ,
AutoSync = AutoSync.OnInsert)]

public string OrderId
{
 get { return orderId; }
 set
 {
  if (orderId != value)
  {
   NotifyPropertyChanging("OrderID");
   orderId = value;
   NotifyPropertyChanged("OrderID");
  }
 }
}
            ...
 }
}
```

[6]HttpClient is a part of .NET Framework 4.5 and Windows Store apps that provides developers an extremely easy way to connect with services across the internet including REST-based services [8].

[7]In LINQ to SQL, the data model of a relational database is mapped to an object model expressed in the programming language of the developer. When the application runs, LINQ to SQL translates into SQL the language-integrated queries in the object model and sends them to the database for execution. When the database returns the results, LINQ to SQL translates them back to objects that the developer can work with in his own programming language [15].

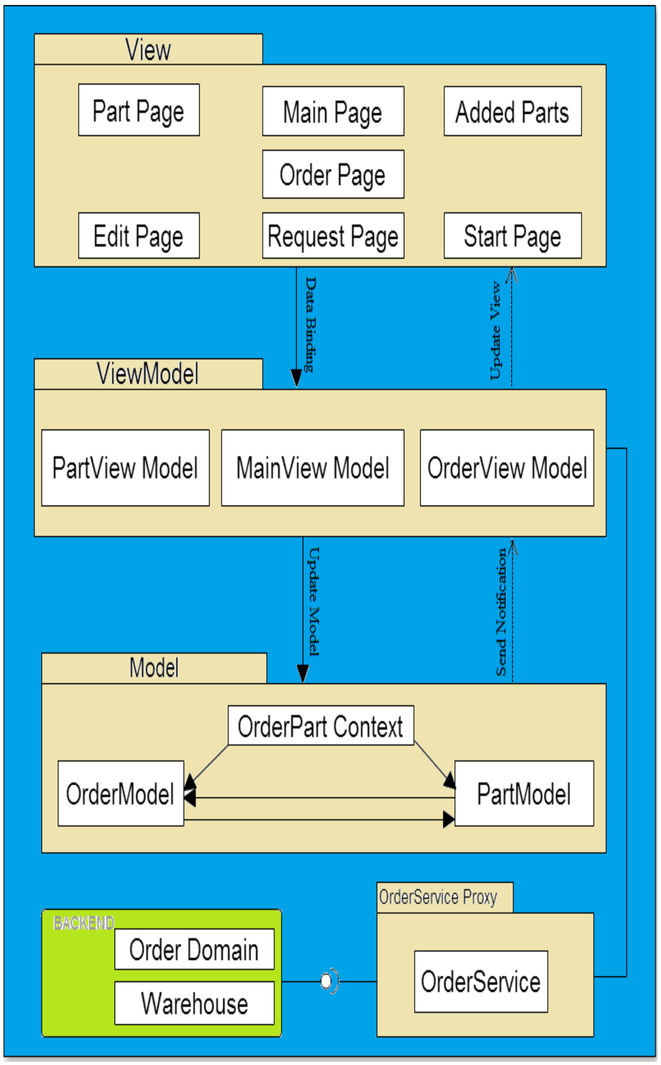


Figure 7: POS Prototype from the MVVM Perspective

4.1.2 Prototype Architectural Attributes

4.1.2.1 Maintainability. As we explained the architectural structure of the POS prototype, the view model component of the prototype uses interfaces for communicating with the view and model components, and therefore it is loosely coupled with them in terms of data format and logic. This separation makes it simpler to unit test, modify, and re-use the view model as well as the model components, and leads to a high level of maintainability.

4.1.2.2 Performance. Since the Order Service of the view model uses asynchronous HTTP communication with the backend, the application becomes loosely coupled with the backend in terms of time and therefore, the availability of the backend can have less influence on the performance of the prototype. However, during the runtime the application uses a local database to save and retrieve the orders as well as their associated parts. This also increases the performance of the prototype as accessing data through the local database takes place faster than accessing it through communication with the backend.

In addition, as Microsoft Developer Center recommends [11], the prototype uses JPG images since the JPG decoder is much faster than the PNG decoder[8]. The prototype uses Tap and DoubleTap event handlers, whenever the user selects an order to update or see the associated parts, and displays the list of orders using LongListSelector control. These event handlers and the list control are recommended in Windows Phone 8 applications for better performance [11].

4.2 Supported Non-Functional Attributes in Windows Phone 8

4.2.1 Security

Windows Phone 8 is designed with a holistic and defense-in-depth approach to security design to help protect against malware, data leakage, and other threats [12, 19]. In what follows we bring the approaches and techniques that Microsoft has utilized to support a high level of security in Windows Phone 8:

4.2.1.1 System Integrity. Windows Phone 8 employs a secured boot process and code signing to help assure platform integrity. Applying these features, only validated software products would be allowed to execute, therefore, the operating system will be protected from malware attacks. Using secure boot technology, all boot components are assigned to digital signatures so that only authorized code can execute to load the Windows Phone operating system.

The next layer of defense is provided when a boot manager component takes over to complete the boot process. The boot manager requires all code in the operating system, including drivers and applications, to be signed by Microsoft before the device becomes ready and the user can start using the phone [12, 19].

4.2.1.2 Chambers and Capabilities. Windows Phone introduces a chambered security model to provide the advantages of attack surface reduction, and isolation. The chambered model uses isolation to achieve the principle of least privilege. Under this principle, each chamber provides an isolation boundary within which a process can run, as well as a security policy which defines the operating system capabilities that the running process within the chamber can call [12, 19]. This technique leads to a number of security advantages:

- Each application receives only the needed capabilities to perform it's use cases.
- Applications become isolated from each other and can not access the used memory or stored data of other applications.

4.2.1.3 Enterprise Line-of-Business Applications. Organizations may require the possibility to develop and directly distribute their custom applications to employees. With Windows Phone 8, organizations can register with Microsoft to obtain the tools to develop, package, sign, and distribute applications to employees. They can use a validated process, without having to submit their applications to the Windows Phone Store [12, 19].

4.2.1.4 Data Protection. Windows Phone 8 includes several features to help protect against unauthorized data access or unintended disclosure. These features follow:

- **Phone PIN:** Access to a Windows Phone can be controlled through a PIN, that users can set through the Lock Screen Settings [12, 19]. In addition, additional security functionalities can be configured using Exchange ActiveSync (EAS) policy for password length, password complexity or other parameters.
- **Remote Wipe:** If a Windows Phone becomes lost or stolen, a remote wipe of the device can be done by using either the Exchange Server Management Console or Outlook Web App [12, 19]. In addition, an EAS policy can be set to wipe a phone after a configurable number of unsuccessful PIN attempts.
- **Device Encryption:** To help keep everything, such as documents or passwords, safe, Windows Phone 8 can be configured to use Bitlocker technology to encrypt all internal storage, including operating system and data partitions. When device encryption is turned on, any file saved to the phone is also encrypted automatically. If a PIN-protected Windows Phone is lost or stolen, the combination of device lock and data encryption makes it difficult for an unauthorized party to retrieve sensitive information from the phone [12, 19].
- **Data Leak Prevention:** Windows Phone 8 offers native support for Information Rights Management (IRM), which allows content creators to assign rights to Microsoft Office documents or email messages they send [12, 19]. When IRM is employed, the data in rights-protected documents or email messages is encrypted so that it can only be viewed by authorized

[8]Windows Phone supports JPG and PNG image formats.

users. IRM can also be used to limit other rights to a document or message, such as preventing the document or message from being copied or printed.

4.2.2 *Performance*

4.2.2.1 Windows Phone 8 Core. Windows Phone 8 is the first mobile operating system from Microsoft that uses Windows NT kernel [9], which is the same kernel that runs Windows 8. Using the NT kernel, Windows Phone 8 can now support multi-core CPUs of up to 64 cores, as well as 1280x720 and 1280x768 resolutions, in addition to the base 800x480 resolution already available on Windows Phone 7 [24]. Developers can take advantage of this feature and create high performance applications in terms of responsiveness and resource usage.

4.2.2.2 Application Performance Considerations. It is important to consider performance when creating Windows Phone 8 applications since it has a limited central processing unit and graphics processing unit, compared to a desktop or laptop PC [10, 11]. To optimize performance of apps on Windows Phone 8, several changes were made to the way that XAML processes graphics and other objects. In what follows, we explain the techniques to consider and tools to use for developing high performance Windows Phone 8 applications:

- **App Monitoring tool:** This tool helps developers identify issues such as slow startup time, slow response time to input, and high battery drain.
- **EnableRedrawRegions tool:** In the page construction, the EnableRedrawRegions property can be set to true so that a developer can visually see what regions of the application are being drawn. Developers can use this feature to performance tune the applications. Using the EnableRedrawRegions tool, when a region is completely drawn, it is shaded with a color. The colored regions indicate that the CPU and not the GPU is used to perform the drawing. When the CPU is used to draw, it is called software drawing. Software drawing is normal, because everything must be drawn by software the first time it is displayed, however, developers should be careful about excessive software drawing.
- **Images:** There are several considerations for selecting and including proper images for a better performance.
  - There are two supported image formats for Windows Phone 8: JPG and PNG. In general, the JPG decoder is much faster than the PNG decoder and should be used for images that are fully opaque. Images that use transparency should be stored as PNG because JPG does not support transparency.
  - In Expression Design, developers can create complex visuals and export these visuals as XAML or as image files. When the visuals are static, the developers should consider storing them as an image instead of XAML. In contrast to decoding and rendering an image, XAML can potentially require more processing. Using XAML for a visual requires parsing the XAML, creating the object in the visual tree, and rendering the object.
  - Due to the limited screen resolution of Windows Phone, another way to optimize performance is to limit the image size to 2000x2000 pixels, which is the size limit of images in the Windows Phone environment.
- **User Input:** User input in Windows Phone includes manipulation events, gesture events, mouse events, and touch events. Unless there is a specific need, manipulation events and gesture events, Tap, Double Tap, and Hold, are recommended for better performance.
- **Progress Controls:** It is recommended to use a progress indicator when performing a time-consuming operation to indicate to the user that the application is working. For this purpose, ProgressIndicator and ProgressBar are the optimized controls that are recommended to be used.

4.2.2.3 Performance Requirements. Before Windows Phone 8 applications are published in the Windows Phone Store, they must meet certain performance requirements set by Microsoft. These requirements include:

- **Application launch time:** Applications must render the first screen within 5 seconds after launch and then they must be responsive to the user input within 20 seconds.
- **Application responsiveness:** If an application performs an operation and it is unresponsive for more than three seconds, then the application must display a visual progress bar or busy indicator.

5. DISCUSSION

This section is divided into two main parts. In the first part, we map the POS architecture to VGTA and answer the first research question on the extent of conformance to domain-driven architectures when developing Windows Phone 8 applications. In the second part, we consider the architectural attributes that POS fulfills as well as those that Microsoft supports for Windows Phone 8 applications, and answer the second research question on the applicable architectural attributes to Windows Phone 8 applications.

5.1 Extent of Conformance to a Domain-Driven Architecture

From the perspective of a domain-driven architecture, the POS prototype may consist of the following components:

[9]Windows NT is a family of operating systems produced by Microsoft, the first version of which was released in July 1993. It was a powerful high-level-language-based, processor-independent, multiprocessing, multiuser operating system with features comparable to Unix [24].

- **Presentation Model:** contains the view and view-model components, and can be corresponded to the user interface component of VGTA.

- **Domain Model:** contains the model component which includes the order domain component. The model can also include a warehouse part which can hold the information about the stored items. The domain model can be corresponded to the domain component of VGTA.

- **Order Service Proxy:** handles the communication with the backend to send and receive the order items. The order service proxy can be mapped to the proxy component of VGTA.

- **Order Service Gateway:** could be responsible for providing services to other applications. Due to the limited scope of POS, the order service gateway was not implemented in the prototype. However, in case that it is implemented, it can be mapped to the gateway component of VGTA.

- **Utility:** could be responsible for general functionalities, e.g. providing username and password. Due to the limited scope of the prototype, the utility component was not implemented. However, in case it is implemented, it can be corresponded to the utility component of VGTA.

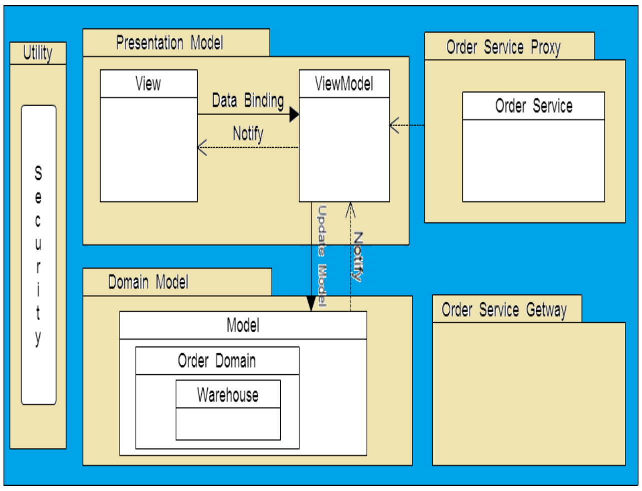


Figure 8: POS Prototype from the VGTA Perspective

POS Prototype	**VGTA**
Presentation Model	User Interface Component
Domain Model	Domain Component
Order Service Proxy	Proxy Component
Order Service Gateway	Gateway Component
Utility	Utility Component

Table 1: Mapping POS prototype to VGTA

As we explained, domain-driven designs use common architectural patterns, such as the layered pattern, and include, at least, the conceptual components of user interface, domain, communication, and service. We consider VGTA as a generic domain-driven architecture, in which the communication components are called gateway, and proxy, and the service component is referred to as utility.

As we pointed in section 3.2, it is recommended to follow the MVVM architectural pattern for developing Windows Phone 8 applications due to the architectural advantages that it supports. Our study shows that following MVVM, it becomes simple to run unit tests, re-use code, as well as modify the applications, and therefore, the applications will fulfill maintainability to a high extent.

We argue that while it is a right practice to follow the MVVM architectural pattern for developing Windows Phone 8 applications, it is feasible to conform to a domain-driven architectural design that includes the user interface, domain, communication, and service conceptual components.

5.2 Applicable Non-Functional Attributes to Windows Phone 8 Applications

Considering the findings of the study, that we presented in the previous section, we explain how Windows Phone 8 applications can fulfill security, performance, and maintainability:

- **Security:**
  - The secured boot process of Windows Phone 8 helps the platform the keep the sensitive information secured from malware attacks.
  - The chambered security model leads to isolated applications which can not access the memory and data of each other.
  - It is possible to develop and distribute custom applications in organizations by registering to Microsoft and obtaining the required tools.
  - The techniques that Microsoft promotes for phone PIN, remote wipe, data encryption and data leak prevention lead to the possibility of protecting data from unauthorized data access or unintended data disclosure.
- **Performance:**
  - Using local databases in Windows Phone 8 applications can lead to less talkative communications among them.
  - Memory usage can be optimized in Windows Phone 8 applications as these applications can have default caps on the amount of memory, which varies depending on the type the mobile device.
  - The performance requirements on Windows Phone 8 applications from Microsoft, the application launch time and responsiveness requirements, enforce the applications to render the first screen within 5 seconds, respond to the user input within 20 seconds, and in case that an operation takes more than three second to execute, the applications must display a progress indicator. Therefore, as long as an application is certified and published in Windows Phone Store, it is guaranteed that

the application will have proper interaction with users.

- **Maintainability:**
  - Following the MVVM architectural pattern, the components of Windows Phone 8 applications will be loosely coupled and separated from each other in terms of logic. Therefore, they will be autonomous and it is possible to fulfill continuous functional modifications.
  - Following the MVVM architectural pattern leads to significant separation of concerns and therefore, it will be simple to re-use code, and run unit tests in Windows Phone 8 applications.

Considering the study results and the aforementioned knowledge about the supported architectural attributes, we argue that Windows Phone 8 applications can fulfill security, performance, and maintainability to a high extent.

6. CONCLUSIONS

In this study, we collaborated with Volvo IT to identify the extent to which Windows Phone 8 applications can conform to domain-driven architectures, as well as to realize what non-functional attributes can be applied to these applications.

We considered a software architectural framework that Volvo Group IT Governance promotes. This framework consisted of Volvo Group Target Architecture (VGTA) and a set of non-functional attributes from which we chose only security, performance, and maintainability to look into, due to the limited time frame of the study. We developed a purchasing order system application prototype in which we followed the Model View View-Model (MVVM) architectural pattern because of the non-functional qualities that it supports for the Windows Phone 8 applications. In addition, we looked into the non-functional qualities that Microsoft has considered for Windows Phone 8 applications.

We realized that while it is a right practice to implement the MVVM pattern for Windows Phone 8 applications, these applications can still conform to a domain-driven architectural design such as VGTA that include, at least, user interface, domain, communication, and service components. Moreover, we found out that Windows Phone 8 applications can fulfill high security, performance, and maintainability, due to the significant support that Microsoft provides in terms of security and performance, as well as the significant separation of concerns that MVVM supports for these applications.

7. ACKNOWLEDGMENTS

We would like to kindly thank Volvo IT and our industrial supervisor, Johan Westlund, for providing us with this research topic and supporting us during the development of the POS prototype with invaluable feedbacks. We would also like to thank our university supervisor, Morgan Ericsson, for his constructive guidances and feedbacks throughout the research. We cordially thank Jon Kristensen for providing us with the opportunity to use a communicating backend for the POS prototype.

References

[1] W. C. Booth, G. G. Colomb, and J. M. Williams. *The craft of research.* University of Chicago Press, Chicago, 2008.

[2] B. Brumfield. *Developer's guide to Microsoft Prism 4 : building modular MVVM applications using Windows Presentation Foundation and Microsoft Silverlight.* Microsoft, 2011.

[3] M. Dalal and A. Ghoda. *XAML Developer Reference.* O' Reilly Media, Inc., 2011.

[4] A. Debbiche, A. Treptow, and Y. He. Towards a generic reference architecture for mobile applications. *Unpublished bachelor's thesis, University of Gothenburg, Sweden.*

[5] A. Ghoda. *Windows 8 MVVM Patterns Revealed: covers both C# and JavaScript approaches.* Apress, 2012.

[6] J. Gossman. Introduction to model/view/view-model pattern for building wpf apps. `http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx`, 31 May 2013.

[7] K. B. Hoffmann. Domain driven design in action. *Unpublished master's thesis, University of Kopenhagen.*

[8] I. Landwerth. Portable httpclient for windows phone. `http://blogs.msdn.com/b/bclteam`, 18 May 2013.

[9] F. Marinescu and A. Avram. *Domain-Driven Design: Quickly.* LULU Press, 2007.

[10] Microsoft. App memory limits for windows phone 8. `http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj681682(v=vs.105).aspx`, 10 May 2013.

[11] Microsoft. App performance consideration for windows phone. `http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff967560(v=vs.105).aspx`, 10 May 2013.

[12] Microsoft. *Windows Phone 8 : The Right Choice for Business.* Microsoft Corporation, 2013.

[13] Y. Natchetoi, V. Kaufman, and Y.Karabulut. Service-oriented architecture for mobile collaboration. *International Conference on Collaborative Computing: Networking, Applications and Worksharing,* pages 371–375, 2008.

[14] M. Q. Patton. *Qualitative Research Evaluation Methods.* SAGE Publications, Inc, 2001.

[15] J. Rattz and A. Freeman. *Pro LINQ: Language Integrated Query in C# 2010.* Apress, 2010.

[16] M. K. Sein, O. Henfridsson, S. Purao, M.Rossi, and R.Lindgren. Action design research. *Mis Quarterly,* 35:37–56, 2011.

[17] M. Snell and L. Powers. *Microsoft Visual Studio 2012 Unleashed.* Sams Publishing, 2012.

[18] C. C. Teng and R. Helps. Mobile application development: Essential new directions for it. *Information Technology: New Generations,* pages 471–475, April 2010.

[19] P. Thurrott and R. Rivera. *Windows 8 Secrets*. John Wiley & Sons, Inc., 2012.

[20] S. TorkAbadi. Towards a generic reference architecture for mobile applications. *Unpublished bachelor's thesis, University of Gothenburg, Sweden.*

[21] B. Unhelkar and S. Murugesan. The enterprise mobile applications development framework. *IT Professional*, 12:33–39, 2010.

[22] D. Vaughan. *Windows Phone 8 Unleashed*. Sams Publishing, 2013.

[23] R. Vice and M. S. Siddiqi. *MVVM Survival Guide for Enterprise Architectures in Silverlight and WPF*. Packt Publishing, 2012.

[24] A. Whitechapel and S. McKenna. *Windows Phone 8 Development Internals*. Microsoft, 2013.