SMART – An Architecture Framework for Web Applications

THESIS

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in
the Graduate School of The Ohio State University

By

SaranyaDevi Ganesan

Graduate Program in Computer Science and Engineering

The Ohio State University

2013

Master's Examination Committee:

Dr. Rajiv Ramnath, Advisor

Dr. Jayashree Ramanathan
Abstract

Web applications have become the primary source of information and transactions over the internet. This has led to high demand for continuously evolving web applications. The web industry foresees frequent shifts in the requirements and architecture of web applications throughout their lifecycle. This demand has become two fold with the advent of “learn to adapt” websites in the research industries.

SMART is a Scalable, Maintainable, Adaptable, Recyclable, and Testable web application framework that can be used to create modifiable web applications, with reusable design components. It adds a variation to the established MVC pattern and gives a new perspective on modularization. It introduces a new navigation model and a new facet “type” to the MVC pattern. This doublet provides web applications with the ability to adapt to any changes in the requirements easily. At its low-level, the framework acts as a classic MVC and at its high-level, it evolves into a Model View Controller Type (MVCT) Pattern. This dynamicity is influenced by the web application’s demands. By providing a flexible framework, SMART reduces the time spent on reasoning over framework options and, in turn, channels the saved time into actual development.

SMART framework aims to define a new viewpoint on the abstraction strategies that has been followed in web applications, thus elevating reusability and introducing the most demanded recyclability.
I dedicate this thesis to my mom who believed in me so much than I ever did.
Acknowledgments

First, I would like to express my heartfelt gratitude to my advisor Prof. Dr. Rajiv Ramnath for his continuous support in my Master’s study and research work. With complete confidence, he endowed me with big responsibilities in which I had no prior experience. I would like to commemorate his help in every phase of my transition from a learner to a researcher.

Besides my advisor, I am much obliged to my mentor Michael Herold, Ph.D. student under Dr. Rajiv Ramnath, who guided me through all aspects of my web application project SWSS with the School of Nursing. This project is the base of this thesis. His design principles and methodologies have encouraged me to focus on the design and architecture of the web application.

I would like to thank my project initiator Vicki Von Sadovszky for this wonderful opportunity. I would also like to acknowledge other collaborators with whom I worked closely in this project: Prof. Dr. Maria Palazzi and her student Allison Biesboer from the Design School, William Matcham, and system administrators from the School of Nursing.

I would like to express my gratitude to Dr. Jayashree Ramanathan, who agreed to be my committee member and review my thesis. Thank you for taking time to evaluate my contributions.
I would like to thank State Bank of India, who funded my education and Ms. Padmapriya Chandramouleswaran, my colleague at Symantec, who wholeheartedly signed a legal document to help me during a period of financial distress in the course of my Master’s education. Without them, I could not have started this intriguing journey in a new country. Thanks to all my friends and classmates, who stood by the thick and thin of my life, which helped me overcome culture shock and home-sickness, who delighted me with compliments when I achieved and who taught me valuable new challenges, which helped me view the simplicity in complexity.

I would also like to thank my colleagues from Symantec and my Manager Badri C.V., who helped me take my first step in the professional life and who always insisted that I could do more and better.

Last but not the least; I would like to elicit the presence of my family in my spirit and actions, even though they are 8000 miles away. I thank my mom and dad who encouraged me all the while, giving me enough confidence to face any challenge. I treasure the deeds of my little sister, whose tantrums I miss every day. She never fails to surprise me with easy solutions whenever I face the toughest problems. I would also like to thank my two year old nephew, who constantly showers his ignorant love on me. Finally, I am overwhelmed with the support that I got from my boyfriend, Siva, who gave me all the reason to love and innovate. He channeled me into focusing what I wanted to pursue by showing boundless affection and a readiness to bridge any distance. Thank you for accepting my dreams as yours. I could not have wished for a better dream partner.
Vita

Jun 2004 to Feb 2009 ............................................. B.Tech, Anna University

Sep 2011 to Present ............................................. Masters Graduate Student, Department of

                   Computer Science and Engineering

                   The Ohio State University

Feb 2012 to Mar 2012 ............................................. Student Software Engineer, Department of

                   Computer Science and Engineering

                   The Ohio State University

Apr 2012 to Present ............................................. Graduate Research Associate, Department

                   of Computer Science and Engineering

                   The Ohio State University

Fields of Study

Major Field: Computer Science and Engineering
# Table of Contents

Abstract ...................................................................................... ii

Acknowledgments ........................................................................ iv

Vita .......................................................................................... vi

1.1. Terms and Definitions .......................................................... 2

1.2. Research Background .......................................................... 2

1.2.1. History of World Wide Web ........................................ 3

1.2.2. Problems in Web Development ...................................... 4

1.2.3. Demands for WAFs .................................................... 5

1.2.4. Evolution of Web Application Frameworks .................... 7

1.2.5. What is SoC? .......................................................... 9

1.2.6. Current problems ..................................................... 9

1.3. Research Motivation ............................................................ 10

1.4. Research Problem Statement ........................................... 11

1.5. Research Goals and solution ............................................ 12

1.6. Blueprint of the Thesis ...................................................... 14

2.1 Architecture of Web Applications ...................................... 15
5.1 Goal and Objectives Evaluation ................................................................. 87
5.2 SMART Evaluation Tree ............................................................................ 87
  5.2.1 Scalability ......................................................................................... 88
  5.2.2 Maintainability .................................................................................. 89
  5.2.3 Adaptability ...................................................................................... 90
  5.2.4 Recyclability ...................................................................................... 90
  5.2.5 Testability ......................................................................................... 91
5.3 Case Study with SWSS ............................................................................. 91
5.4 Case Study with .Net Experts ................................................................ 93
  5.4.1 Large Scale Application .................................................................... 93
  5.4.2 Medium Scale Application: ............................................................... 94
  5.4.3 Small Scale Application: ................................................................. 95
  5.4.4 Reusability Percentage of MVCT Pattern ........................................ 96
  5.4.5 Development Time: MVC Vs MVCT ................................................ 97
References ....................................................................................................... 101
List of Tables

Table 1. Percentage of framework Code in the SWSS MVCT Application.................. 92
Table 2. Large Scale Application's ActionResult Count and View Count .................. 93
Table 3. Medium Scale Application's ActionResult Count and View Count.................. 94
Table 4. Small Scale Application's ActionResult Count and View Count .................. 95
Table 5. Estimations of code reuse in large, medium, and small web applications........ 96
Table 6. Development Cost for Large, Medium, and Small Scale Applications............. 97
List of Figures

Figure 1. Web Application Development Life Cycle .......................................................... 10
Figure 2. Simple Canonical web application Architecture .................................................. 16
Figure 3. General web application Architecture .................................................................. 18
Figure 4. The 4+1 View Architecture Model ....................................................................... 19
Figure 5. 3-Tier Architecture .............................................................................................. 21
Figure 6. Typical MVC fit in web applications ..................................................................... 22
Figure 7. Hut framework Analogy ....................................................................................... 24
Figure 8. Cocoon Pipelining Sequence Diagram ................................................................. 29
Figure 9. Yii framework Handling User Request .................................................................. 32
Figure 10. ASP.NET MVC Communication Model .............................................................. 34
Figure 11. ASP.NET MVC Activities Illustration .................................................................. 35
Figure 12. Page Controller Pattern MVC Architecture ....................................................... 38
Figure 13. Grouping Common Actions into the Base Controller ......................................... 39
Figure 14. Abstracting web Dependent and web Independent Code .................................... 40
Figure 15. Front Controller Pattern MVC Architecture ..................................................... 43
Figure 16. Front Controller Pattern's Internal Architecture ............................................... 44
Figure 17. Front Controller Pattern Sequence Diagram ..................................................... 45
Figure 18. ASP.NET MVC Control Flow Architecture ....................................................... 46
Figure 19. MVC HTTP GET Request Sequence Diagram .......................................................... 49
Figure 20. MVC HTTP POST Request Sequence Diagram ....................................................... 51
Figure 21. ASP.NET MVC Low Level Process Flow Architecture ......................................... 53
Figure 22. Controller and ActionResult Structure in MVC Application .............................. 55
Figure 23. View Mapping inside MVC Application .................................................................. 55
Figure 24. Database First, Model First and Code First Development Methodologies .......... 57
Figure 25. Develop a MVC Application Use Case ................................................................. 58
Figure 26. View1 with PartialView1 and Partial View2 .............................................................. 61
Figure 27. View2 with Partial View2 ....................................................................................... 61
Figure 28. View3 with PartialView1 and Partial View3 .............................................................. 61
Figure 29. View4 with PartialView1 .......................................................................................... 62
Figure 30. Inheritance in Animal Hierarchy ........................................................................... 66
Figure 31. Variation in Animal Inheritance Hierarchy ............................................................. 67
Figure 32. MVCT Pattern in SMART Framework ................................................................. 69
Figure 33. SMART MVCT Control Flow Architecture ......................................................... 71
Figure 34. SMART Conceptual Architecture ........................................................................ 74
Figure 35. Page Repository Design in SMART ...................................................................... 76
Figure 36. Type table, Question Mapped to Page Table via QuestionPaging ...................... 78
Figure 37. HTTP GET Request to SMART Framework's MVCT SoC .................................... 79
Figure 38. HTTP POST Request to SMART Framework's MVCT Pattern ............................ 80
Figure 39. MVCT Low Level Architecture ............................................................................ 81
Figure 40. Application Structure for MVCT Pattern .............................................................. 83
Figure 41. Use Case to Develop Navigational Model in MVCT.................................................. 84
Figure 42. MVCT Use Case to Develop Type Controllers......................................................... 85
Figure 43. Integrate Page Engine and the Type Engine in MVCT............................................ 86
Figure 44. Evaluation Tree for web application Frameworks .................................................... 88
Figure 45. Increasing Order of framework % in MVCT Application ....................................... 92
Figure 46. Comparison of ActionResults and Views in Large Scale Application
MVC/MVCT........................................................................................................................................ 93
Figure 47. Comparison of ActionResults and Views in Medium Scale Application
MVC/MVCT........................................................................................................................................ 94
Figure 48. Comparison of ActionResults and Views in Small Scale Application
MVC/MVCT........................................................................................................................................ 95
Figure 49. Reuse percentage across large, medium, and small MVCT applications........... 96
Figure 50. Man Hours Comparison between MVC and MVCT in a Large Scale
Application........................................................................................................................................ 97
Figure 51. Man Hours Comparison between MVC and MVCT in a Medium Scale
Application........................................................................................................................................ 98
Figure 52. Man Hours Comparison between MVC and MVCT in a Medium Scale
Application........................................................................................................................................ 98
Sample Code

Sample Code 1. View Model Sample from MVC Application .................................. 47
Sample Code 2. View Code for a Registration Model ........................................... 48
Sample Code 3. Account Controller with LogOn ActionResult .............................. 49
Sample Code 4. LogOn View ........................................................................... 50
Sample Code 5. ViewModel Object being used for evaluation inside the Controller ..... 52
Sample Code 6. MVC Application's Routing Module ....................................... 54
Sample Code 7. Question Type View Model Sample ......................................... 71
Sample Code 8. Question Type ViewModel Sample ........................................... 72
Sample Code 9. Choice ViewModel inside the QuestionViewModel ..................... 72
Sample Code 10 Question Controller as a Type controller ............................... 73
Sample Code 11. Navigational Decisions in SMART ......................................... 77
Sample Code 12. Register Module for MVCT Pattern with the New Navigational Model ................................................................. 82
Preface

This thesis presents a web application framework based on the ASP.NET Model-View-Controller (MVC) framework and adds new dimensions of “adaptability” and “recyclability” to it. Essentially, the concept of separation of concerns (SoC) has been applied in a unique manner within the .NET framework in order to build reusable web application components. Web application developers started using object oriented analysis and design (OOAD) in combination with SoC to enhance reusability. However, the SoC concept has typically been applied in a manner that limits OOAD from being applied across web application layers. In fact, balancing SoC and OOAD is a challenging design problem. This thesis presents a solution towards achieving this balance.

The idea to develop a more reusable framework for web applications emerged when I started to work on the Strong Women Stay Safe (SWSS) website to help with the research work at the School of Nursing. The SWSS project is a joint venture among CETI, ACCAD, and the nursing school. One of the main requirements was to develop a website with easily customizable page flows. The idea to bring a new “concern” to the manner SoC has hitherto been applied to MVC emerged in CETI group meetings on Persuasive Systems with my advisor Dr. Rajiv Ramnath. The key insight was that in a web application code reusability is usually measured across projects and not within a project. This key insight is the primary driver of this thesis.
CHAPTER 1: INTRODUCTION

The World Wide Web (WWW) is one of the greatest technological novelties existing today. It began as a medium of communication with static content in the early 90s and later became the medium of rendering software applications to users. In doing so, it has evolved through many generations and has made the world smaller and provided several benefits to it.

Concurrent development of technologies like JavaScript, Ajax, JSON, and Document Object Model (DOM) has made web applications interactive. Web applications have become one of the significant capital investments in every company. This huge growth has opened it to a myriad of demands.

To standardize the wide area of applications, web application frameworks (WAFs) were introduced. WAFs define a standard development paradigm for web applications by implementing factors like scalability, reliability, and extensibility. Web applications reached global population to serve business needs. This globalization of web applications has led to a shift in the way web applications need to be developed.

This section describes the evolution of websites and web application frameworks. Section 1.1 introduces the World Wide Web to the audience and discusses the problems in web applications development. Demands on WAFs are described in detail along with
the consequent evolution of WAFs. In Section 1.2, the motivation for this thesis is presented to the readers. Section 1.3 discusses in detail the various problems that a developer incurs during web development and how the business needs are affected. With the problem and motivation as the base, Section 1.4 presents the goals and solution outline of this thesis. Section 1.5 contains the blueprint of this thesis.

1.1. Terms and Definitions

Few non-quantitative terms are used throughout this thesis to explain SMART WAF. The definition of these terms with respect to web applications, WAFs and this thesis are discussed below.

Recyclability: Recyclability is the ability of a product to undergo recycling and emerge as a completely new product. For a WAF, it is defined as the ability to accommodate change in any part of the web application. Change can be in the data, design, functionality, structure, or navigation between the pages. The term Moldability is often used interchangeably with recyclability throughout this thesis.

Collaborative Development: Collaborative development is the ability to allow different teams to concurrently develop a web application with each team focusing on different aspects of the system.

1.2. Research Background

There has been a rapid transformation in the development standards of WAFs from the simple conventional methods to the more advanced conceptualization methods
Several software engineering practices have been applied and integrated into web applications development in the past few years (Ousterhout, 2009). One such practice is the SoC software paradigm. This is similar to object oriented programming (OOP). This paradigm has helped WAFs evolve with more reusability and extensibility.

E-commerce websites have incorporated different technologies with each one being used for a different purpose within a website. Two different suites of technologies grew to fulfill these demands, with both the technologies succeeding for common business and trading applications. One was Java and the other was Microsoft® ASP.NET (M. Keith Mortensen, Infusion Development; Rob McGovern, Infusion Development; Charles Liptaak, Microsoft® Corporation, 2003). Both of these frameworks are based on the industry standard programming languages, Java, and C#. Both the frameworks allow little room for change in the framework level to let developers explore other options. Nevertheless, the demand for the “right” framework continues and the vacuum caused by missing pieces is yet to be bridged (Clarke, Harrison, Oscher, & Tarr, Subject Oriented Design: Towards Improved Alignment of Requirements, Design and Code, 1999).

1.2.1. History of World Wide Web

In the last decade, information transformation in the internet has been tremendous and the evolution of business and trade in it has been rapid. Several web frameworks have sprung up to aid this evolution that give priorities to business needs. When the internet became the foundation of business and trade, websites were built to establish online businesses and to reach out to global populations beyond physical boundaries. The
demand for complex functionalities led to the demand for reliable, scalable, secure, robust, and fast web applications. With so many conditions for a successful web application, it became difficult to settle on a single technology.

In addition to this, web applications have entered the research arena too where they are the primary data collection tools. Researchers use the “learn and adapt” strategy on web applications, which has required much better extensibility and modifiability (An Approach to User-Behavior-Aware Web Applications, 2005).

1.2.2. Problems in Web Development

Though there has been a significant breakthrough in the web development tools and WAFs, the initial intent of web protocols was to support content delivery onto web documents and not for complex communication and inheritance. The first standalone website did not have any dynamicity data, process, or control. The addition of databases, scripts, styles, frameworks might have enhanced the functionality and the presentation of web applications, but they have complicated the protocols and have obstructed the productivity of web developers. Some of the reasons, why these communication patterns might be a huge hindrance to web development are as follows.

Individual web developers face issues with their sub systems and the communication between the subsystems. They use online forums and try to reinvent the wheel repeatedly. Since there is no standardization, some solutions are correct and some are incorrect. These incorrect solutions might temporarily resolve the leaf issue, yet the root issue might be now exacerbated because of the incorrect fix.
The languages used in a typical web application development could be heterogeneous. The code behind the web pages could be written in Java, C++ or C#, the presentation layer could be written in HTML, CSHTML or ASPX, the database layer could be written in a procedural language or a simple query language and there are other set of documents like CSS (Cascading Style Sheets) for styling, XML (Extensible Markup Language) for configurations and scripting languages for custom scripts. This heterogeneity makes code reuse between languages not possible. For example, consider a simple text field where the user enters data. This data has to be validated to follow certain rules. When validations are written both on the client side and on the server side, integrity has to be checked such that they follow the same semantics (Lhotka, 2006).

When components in a web application are loosely coupled, modularity and code size increases because a glue code has to be written to hold the loosely coupled components together (Software Practice and Experience, 2001). Glue code adds to both structural and visual commotion and so it makes testing or debugging difficult by veiling potential bugs (Davis, Gamble, Hepner, & Kelkar, 2005). Conversely, when a developer wants to reduce the glue code, modularity decreases (Balasubramanian, Schmidt, Molnár, & Lédeczi). Presence of glue code in a web application indicates that the languages used for web development does not embrace modularity and it is high time to pick a standard solution to resolve this issue (Clint, Garin R.; Thomas, Richard J.).

1.2.3. Demands for WAFs

Standardizing is a very important procedure in the software industry (LU & YEUNG, 1998). Especially, when inexperienced or non-technical people use the
software, standard methods will resolve the problems easier. For example, when many
ASP.NET developers face a problem with the membership class, standardizing the best
solution will prevent reinventing the wheel. Later, versioning this standard with more
contributions could make the standard better.

In addition to creating standards, references play a very important role to extend
the life of a standard (Ousterhout, 2009). When standards lack references, the technique
devised goes astray and interpretations become divergent. For example, web standards
like cascading style sheets (CSS) and document object model (DOM) did not document
references and hence when new web browsers were developed, they all had varied
solutions resulting in incompatibility (Koch, 2013).

One way to standardize a web development process is to develop WAFs that will
attempt to resolve common problems encountered during web applications development.
Nevertheless, these issues could also be resolved by any software framework, which can
treat web applications as a special type of software application and try to resolve the
problem. Then why do we need WAFs?

The problem with migrating Object-Oriented (OO) application frameworks to
web applications is that web applications are not written with a single language. They are
the epitome of cross platform technologies and so migration of any application
framework to web applications will require many modifications. The need for WAFs
arises from the fact that websites contain HTML pages, scripts, database procedures,
objects, navigational structures, and mapping between the different components. A WAF
defines component behaviors and facilitates mapping between components.
Applications development has evolved to produce hybrid software solutions in a shorter development cycle. In web applications, an array of technologies and a myriad of entry points influence the complexity as well as encourage collaborative development. Collaborative development might require involvement from different domain experts like back end developers, front end developers, integration engineers, database engineers, network security specialists, web server managers, designers and deployment coordinators. Successful commercial web applications invest both time and money for collaborative development. Every web application owner is looking at a limited budget efficient web development techniques. Several research projects in industry and academics are also looking for web development solutions at a faster production rate. This helps them to solve their research issues faster (Pabiniak, Chester, 2007).

1.2.4. Evolution of Web Application Frameworks

The web application features present today in WAFs emanated from a series of feature developments in the WAF standard (Fayad, Schmidt, & Johnson, 1999). This section discusses the evolution of web applications and WAFs.

The first generation web applications used only static web pages. The second generation used common gateway interface (CGI) protocol. The CGI scripts tied the universal resource locator (URL) to applications. Once the URL was selected, a sub process initiates the application and generates HTML content for the output device. This output is redirected to the browser and is rendered as HTML on the web page. The file based or database based implementation of stateful CGI protocols was a simple feature added to the stateless CGI. Even after the advent of stateful CGI protocols, stateless
remained as the signature of CGI web applications because statelessness is easy to implement and maintain (Ousterhout, 2009).

The first generation WAFs came after CGI protocols. Two such frameworks were the PHP frameworks (Tatroe, Lerdorf, & MacIntyre, 2006) and the frameworks that used Java servlets (Coward, 2003). In this generation, web specific library packages were included to parse the URL and to execute the corresponding code. The library took care of database connections and dealt with information sources like the file systems. Session management was introduced in this generation of WAFs. These first generation frameworks also enabled HTML generation and modification, by coming up with annotations that are replaced with the HTML content for the web page. This replacement code is customizable by the developer. It could range from being a simple static HTML content to complex dynamic code.

The second generation of WAFs concentrated primarily on business objects and databases. Object Relational Mapping (ORM) was invented and it facilitated web application development with relational databases. Some of the famous ORM frameworks were the Rails framework and the Django. This generation also saw the influence of SoC on web applications. The MVC pattern enhanced collaborative development and reusability. Thus, the market was filled with many MVC-patterned frameworks (Reenskaug, Models-Views-Controllers, 1979) (Reenskaug, Thing-Model-View-Editor, 1979). In the same generation, JavaScript libraries came into existence. The aim of JavaScript was to veil the incompatible view on browsers. It dominated the market
with additional functionalities like DOM, Ajax, advanced User Interface (UI) controls, animation, and styling effects via rich text editor.

These two generations of WAFs have elevated the web application from CGI approach to have more features and enable better server-side implementation.

1.2.5. What is SoC?

The principle of SoC played a very important role in the evolution of software design. It has been the one main rule that software designers used to define new design strategies. The driving force behind most of the software design evolution is to achieve a better SoC. The famous computer scientist Dijkstra has proclaimed that the best way to achieve SoC in a program or software would be to modularize it into components and focus on the individual modules (Adam, 2007).

1.2.6. Current problems

One of the challenges in developing a WAF is to accommodate collaborative development. A good framework should allow at least two or more teams to participate in the development activities. This is a very important consideration to reduce the development lifecycle of a web application. However, most WAFs only allow partial collaborative development leading to a huge time being spent on figuring out a way to integrate contesting code. In the worst case, contesting code might have to be rewritten. A good WAF not only allows collaborative development, but also allows smooth integration between these independently developed components.
Adapting web applications to meet the changing demands of users can yield better results both in e-commerce and in research. As a result, learning and adapting strategies have become quite popular in web development. Although, WAFs have given web developers the flexibility to develop full featured applications in an expedited manner, it is very time consuming to reconstruct the entire website in a different way with a different set of features. While, a WAF might have reduced the man hours spent in initial development, it might not be flexible enough to give developers the ability to restructure the website or its content.

1.3. Research Motivation

In a web development life cycle (WDLC), a WAF is chosen in the design phase after careful consideration of the requirements and the tradeoffs.
Once the website framework is chosen, technical specifications are written down and prototyping is done to evaluate the trade offs. A great number of man hours are spent in understanding the framework with respect to its documentation, popularity, features, and fit into the web application. Most of the time adjustments have to be made along the way of weighing options.

Every research work undergoes an evaluation phase and it might require sensitive data collection from the public or target users. The demand for unbiased data had led researchers to prepare data collection tools that could be presented to a common user without any persuasion. Persuasion research recommends using the learning and adapting strategy to avoid persuasion. Given this, it is clear that research industries will require data collection tools, which can continuously adapt based on the learning.

1.4. Research Problem Statement

The research problem of this thesis originated from the problems faced in the SWSS web application. The requirements of SWSS is as follows,

“Designing and developing an interactive survey website for Strong Women Stay Safe, an incremental learning and adaptive health research application, with the allowance for structural, functional, visual, and content changes at any time during the research with little or no development cost incurred.”

SWSS is a website survey application, which wants to educate women on sexual wellbeing irrespective of their race and culture. The challenge is to present the website
with information and games tailored to the individual user so that the user feels a strong connection to the personalized information. The researchers designed a data collection tool to collect data about the user’s personal values and culture with the help of avatars and a few carefully selected survey questions.

The initial data collection would be from a small group of subjects. This attempt will help the researchers understand the target population and to restructure the data collection tool to sieve this target population to get to the core sample. This method presents an incremental adaptive website, which keeps changing based on the information learnt from the targeted population in each try until the researchers learn no more. To assist this incremental learning, data collection tools i.e., survey website in the case of SWSS has to undergo dynamic recycling. This will involve various levels of modification in structural, functional, visual, or content data to tailor it to the target population better.

So the synthesized problem statement would be, “To develop an enterprise WAF based on .NET MVC3, driven by structural, functional, visual and content data, which could generate a web application when characterized with custom structure, layout and content data. The framework should be Scalable, Maintainable, Adaptable, Recyclable, and Testable.”

1.5. Research Goals and solution

The goal of this thesis is to resolve all the research problems described above via providing a way to integrate OO with SoC principles.
Developing patterns and new architectures on top of the existing architectures is a common methodology in the software industry to reduce redundancy and increase reusability and extensibility. A good software developer has to identify the point where his framework is generating a lot of redundant code and strategically design the application to improve reusability. When this attempt fails, the situation calls for a new framework, which should be an add-on to the previous framework, in the sense that it should be able to deliver the full functionality and features of the previous framework and at the same time void its negatives. The framework chosen for this thesis solution was the ASP.NET MVC, which follows the classic MVC Pattern. The intended WAF caters to the unique need of the current web applications as follows:

- **Scalability:** In MVC, scalability of a web application is achieved by improving the code size as well as by tuning database performance. With SMART, database tuning alone would suffice, as the framework is data driven.

- **Maintainability:** The amount of reusable code in a MVC web application is very limited. The intended framework resolves this problem by following a data-driven approach.

- **Adaptability:** Since SMART framework uses OO concepts to abstract data, no special training is required to use this framework.

- **Recyclability:** When a website has to be completely recycled into a new form, SMART provides a customizable engine for managing this recycling process. Thus, SMART provides web applications with the ability to evolve continuously.
• Testability: Since SMART focuses on designing reusable components within a web application, testing is made easy (Harrold, 2009).

1.6. Blueprint of the Thesis

This thesis is organized as follows. In Chapter 2, the research background is reviewed in detail along with the description of literature works studied during the implementation of SMART. Chapter 3 focuses on the architecture of the ASP.NET MVC framework on which SMART is based. In Chapter 4, different dimensions of SMART framework are presented. In Chapter 5, evaluation of the framework is presented using an evaluation tree. This section compares SMART with the ASP.NET MVC while evaluating using the evaluation tree.
CHAPTER 2: BACKGROUND REVIEW

This section reviews the background work that was done before starting the actual implementation of SMART framework. Section 2.1 describe about the architecture of web applications to understand the hotspots where reusability could be implemented. Section 2.2 describes WAF and its characteristics to understand the underlying standards in constructing a framework. Section 2.3 introduces two components of a web framework, the conceptual model, and the navigational model. In Section 2.4, different approaches to web frameworks development are discussed, with reference to different generations of framework. Section 2.5 unrolls work that was done in similar lines.

2.1 Architecture of Web Applications

Architecture has its roots in civil engineering. Civil engineers lay down blueprints before starting construction of a building. The importance of architecture to web applications is the same as it is to a building (Grady Booch, Chief Technical Officer and Vice-President, Catapulse, 2001). A good architecture is the essential predictor for a cost effective, scalable, and continuously evolving website.
2.1.1 Canonical web architecture:

The canonical web architecture follows the basic server-client architecture. It has an application server and database interacting with each other in the same way as in any server software application. The web server and the web clients belong to the web space and they interact with both the application server and the file system. The simple canonical web architecture can be presented as follows.

![Simple Canonical web application Architecture](image)

Figure 2. Simple Canonical web application Architecture

Though the architecture diagram resembles traditional client-server architecture, there are several architectural differences when the elements interact with each other. For example, web application has to decide between maintaining a state and not maintaining
a state and this affects all elements in the architecture unlike the traditional client-server architecture. There are several challenges in the implementation of web application architecture.

Different communication protocols are used in different layers of a web application. For example, communication between a web client and a web server would follow an Internet Protocol (IP) whilst a file server and an application server will communicate like a normal software application. Integration between these different communication protocols is a challenge in web application.

The decision between thin client and fat client is a very important architectural tradeoff for web applications. A web application developer should understand the advantages and disadvantages of each of these architectures. Thin clients are simple to use and easy to implement. Hence, sophisticated solutions cannot be expected out of it, whereas, thick clients offer locality of reference, sophistication of distribution and interactivity. However, the implementation and communication overhead with the thick client might not be suitable for many web applications. Therefore, understanding the demand of the web application and choosing between the thick and the thin client requires a learning curve.

Considering all these challenges, detailed common web architecture could be derived. The Patterns and Practices Application Architecture Guide 2.0 discussed about one such common framework and this section refers to this more standard common framework. The representation of this framework is as follows (Meier, et al.).
2.1.2 Architectural Views:

When designing a web application architecture, it is essential to consider different views as described in the 4+1 View Architecture model (Kruchten, 1995) (Shklar, Rosen, & Jones).
The design view of the WAF focuses on the actual design of different elements in a web application and the interactions between them. This includes the functional requirements of a system.

The process view is mainly used for scalable web applications, which requires manual thread implementation. If the underlying architecture framework does not have a mechanism to implement threads, then the web application architecture should be designed to accommodate concurrency and synchronization mechanism and therefore avoiding deadlocks or starvation.

The implementation view is the configuration setup in a web application. Apart from coding the functionality of a web application, proper setup has to be done for the
communication protocols to work as expected. In addition, files and components integrated with the system should be assembled and marked for release. Effectively, the final product should have a standalone code with a complete functionality.

The deployment view maps the architectural components to the system hardware. This mostly involves the operational engineers’ design document. The topology has to be understood and coordinated with the document before preparing the web applications for release. This ensures distribution, delivery, and successful installation of a web application on the server.

While designing a web application, use case view aids in understanding different perspectives of different users in the system. The system is used internally by testers, analysts and externally by various end users. Understanding their perspectives helps the developer to build a modular component oriented web application.

2.1.3 3-tier Architecture:

In a 1-tiered architecture, presentation, business and logical layers are inside a single hardware and in a 3-tiered architecture, these layers are in different physical platforms.

The initial tiered application was a 1-tier architecture, which had all the code in a single machine. The presentation, data and the logical layers were all tightly coupled and were indiscernible. It caused many problems with respect to scalability. Even with a multiprocessor system, the system was not scalable enough to cater the needs of increasing users, due to the connection limitations to a single server. Even if the server was powerful enough to accept many connections, code has to be changed in all the three
layers to make the web application scalable. In 1-tier architecture, porting the application to another machine means rewriting the code from the scratch.

The 2-tier architecture was a separation of presentation layer and logic layer from the data layer. This made it easy for developers to forget the backend implementation while writing logic or html. Yet, problems that were described in the 1-client architecture remained between the logical and database layer.

The 3-tier architecture has three tiers with each tier belonging to one layer, presentation, logical and database.

![3-Tier Architecture](image)

**Figure 5. 3-Tier Architecture**

It has been one of the famous WAFs for quite a long time. As the names suggest, presentation layer deals with the HTML, static content, styles and all the front end applications, business layer deals with the business logic and the data objects, which will be responsible to act as an interface to the database layer. The database layer has the actual data objects, which are updated, based on the business logic. The implementations
of these three layers should be independent. When connected, these tiers should communicate smoothly without glitch.

2.1.4 Model View Control Architecture:

The MVC pattern was first implemented in Smalltalk and was called as the Smalltalk MVC Framework. It was primarily used for standardizing the UI of Smalltalk-80. Three types of objects evolved out of this MVC Paradigm. The Model models the real world entity, which includes the entities, characteristics of the entities, state of the entities, application domain data and mapping between the application state and the entity state. The view is responsible for the visual content, from text to high level graphics. The controller reacts to the user inputs and manages both the view and the model. MVC is successful because of the division of responsibilities that lets an individual work on individual concerns. The figure below illustrates a typical .NET MVC.

![Figure 6. Typical MVC fit in web applications](image-url)
The model holds the business logic of a web application and encapsulates all the system actions. The view represents the presentation layer and the controller focuses on control flow between view and model. It ensures appropriate actions are triggered when an event occurs.

2.2 Web framework and its characteristics

*Website Definition:* A website is a collection of web pages served from a registered web domain via Hyper Text Transfer Protocol (HTTP). A website is hosted on a web server where both content and code resides and is accessed via a URL from anywhere on the internet. The web pages are either plain text or Hyper Text Markup Language (HTML) or Extensible Hyper Text Markup Language (XHTML).

*Web application Definition:* A web application is the dynamic version of the static websites. They have run able application scripts along with HTML and these scripts execute when the web pages are accessed via URLs.

*Web application framework definition:* A WAF is a software framework that gives flexibility of filling customizable code and generating web applications.

2.2.1 Analogy for WAF:

WAF is like a blueprint or frame for construction. Based on the requirements document, a WAF is chosen by the developer. Generally, WAF contains the template code and pre-implemented control flow. For example, WAF is like the frame for a hut,
which cannot be used to build a multi-storey building, but can be used to build a thatch roof hut, brick hut and so many other variations of hut.

![Figure 7. Hut framework Analogy](image)

The constructor need not worry about how to place the wood one after another or how to define an outline. The framework takes constructor off the responsibility to redefine the shape of a hut, which is already been defined. Similar to the above analogy, a WAF is a framework chosen by the web developer to create web applications whose outline conforms to WAF standards.

### 2.2.2 WAF Characteristics:

The primary purpose of WAF’s existence is to promote reusability (Schwabe, Rossi, Esmeraldo, & Lyardet, 2001). Web design frameworks are conceptual approaches to avoid redundancy and to maximize code reuse. The importance of abstracting and reusing such design components and structures are explained in detail in this paper (Schwabe, Rossi, Esmeraldo, & Lyardet, 2001). Any WAF should be able to provide the
four primary goals of web applications: scalability, maintainability, availability, and reliability. In addition, every WAF should have goals of its own. Different web applications will need different architecture and hence different WAF implementations. Some might have decoupled components of navigation, design and UI and some might not focus on the navigation completely.

Web developers choose different web design and development frameworks based on the web application that they are going to develop and this requires a learning curve. The learning curve adds up every time a different type of web application is being developed. Apart from spending time in actual design and development, there is an interval spent in discussing and deciding which web design or development framework will best suit the application.

Web applications are different from conventional software in the sense that they should have fewer bugs, quick development time, and continuous evolvement. Therefore, improving the development of web application should not only focus on the code development but also on the testing and debugging strategies. Very few frameworks enhance the testing and debugging factors of web development. A framework that could enable decoupling of all components will be ideal for component and unit testing.

2.3 Components of WAF

A web framework has two main components. One is the actual concept of the web framework where the application coding resides and the other is the navigation code. The conceptual model of the web application contains annotations, which are placed as
placeholders for the developers to fill in the web application code, and the navigational model is the control hand offs that happen within a web application. For example, MVC framework defines navigational model within each controller where the controller is the manager handing off control to the next controller, essentially the next URL and hence allowing navigation within a site. These two components are the backbone of a WAF.

2.4 Approaches to web Frameworks

Websites were initiated with static pages, images, and content in the first generation. Today, web applications have lot more complex structures and components entailed to them. For example, enterprise web applications show dynamically created pages, use relational databases to store enterprise data, implement database transactions, use the content management systems, and handoff controls between components. If a web developer has to go in detail into all of these components then it might be difficult to create an enterprise web application. A WAF separates SoC from the development cycle to keep web development process simple (Yaldiz) (J, 2002).

OOP has been one of the biggest advents in the software industry. Most programmers used OOP within each SoC layer. However, the development of relational databases and Object Relational Mapping (ORM), has given rise to OO frameworks designed for web applications. ORM frameworks like Object Relational Bridge (ORB) and hibernate have become quite popular in software development. They give developers the flexibility to develop software applications that are less error prone and less deviating from the standard practices.
2.5 Literature Review

Several papers and news articles relevant to WAFs were studied during the course of SMART WAF development. This section describes some of the notable research works that were referred for the SMART WAF development. In addition, the section describes similar WAF development to justify the need for such a WAF.

2.5.1 DotNetNuke:

“DotNetNuke is an open-source framework that started in 2002 and is currently on version 07.00.04 (Pabiniak, Chester, 2007). There is a large community of developers involved in the core features and there are over 350,000 register users. Third party vendors supply many add-ons to the system including messaging, forums, user profiles, and skins.” This WAF is aimed for research projects, which require smaller budget web development and produce fully fledged features. DotNetNuke has its own membership code and a content management system. The framework has a skinning engine, which gives developer the flexibility to change the page design easily and not make any significant changes in the code. The advantage of allowing different add-ons to be combined with the web applications includes features that are very popular in websites. This will help web developers reap cost and time benefits and web owners to have customizable and pluggable features. DotNetNuke was primarily started for health research projects and hence it eases data entry and manages the side effects effectively.

Since this is a new attempt in the WAFs, the learning curve of this framework is high. A developer has to understand the structure and the organization of this framework to work with it. A static web application might be built easily with this framework where
the programmer’s job is to understand how to instantiate the web framework and give inputs to it. A dynamic web application is very common and to develop such a web application the developer has to understand the entire architecture. This higher learning curve is the primary limitation of the DotNetNuke.

2.5.2 Apache Cocoon: A Web Framework for E-Health System:

This paper talks about developing a framework that is focused on SoC (Ottaviano, Mora, & Waldmeyer, 2006). SoC is defined as a way to design layers within web application architecture independent of the content and style factors. Especially developing a web application involves several stages of development. Separating or abstracting those components out will aid in “Separation of Concerns.” For example, a web application has business logic, model data, database, content, style, process flow, and data flow.

The framework described in this paper, cocoon is designed to provide “Separation of Concerns” for applications that has different levels to process. These levels are devised together by pipelining their completion. The logic behind this implementation is to hook the right component with the right connection components. For example, let us take a simple application web page. One pipeline might involve in generating content sequentially and another pipeline might involve in generating styles for these components. Now connecting these two pipelines together at the right time is important. Because, it is inappropriate to connect styles for a questions page to a text page and vice versa. The components are made aware of their connection via SAX events.
2.5.3 Object Oriented Hypermedia Design Method (OOHDM):

Object Oriented Hypermedia Design Method (OOHDM) was primarily focused on viewing websites as navigational views (Schwabe, Rossi, Esmeraldo, & Lyardet, 2001). They had contexts, indexes for user interface design unlike the object model. With OOHDM, any object oriented programming model can be used and at the same time gain the advantages of OOHDM being a navigational model. Research has taken place to take advantage of this advantageous framework and apply code reuse on it. Especially this
paper deals with e-commerce websites and most of them follow the same navigational structure and similar functions to the users. The research starts with finding such similar recurring patterns of navigations and functions. They have recorded these recurring patterns with mixture of the GOF (Gamma, Helm, Johnson, & Vlissides, 1995). According to this paper, it is possible to reuse Micro-Architectural details in a website, if one modularizes the website development process to decouple that micro-architecture component. This paper describes the Object-Oriented frameworks and their standard way of enabling reuse of software.

Some of the information discussed in this paper applies to S.M.A.R.T framework as well. So the comparison is detailed in this section. Object-Oriented application frameworks are a classic way of building high quality software applications in a specific domain. Every domain has an abstract design defined in this Object-Oriented application framework. So for that domain, the corresponding design is picked for use (Fayad, Schmidt, & Johnson, 1999). Object-Oriented application frameworks define a set of concrete as well as abstract classes and models. These could be used alone or could be used in collaboration too. The application could be divided into parts that can later use the model classes described in these frameworks. Therefore, Object-Oriented application frameworks act as a skeleton, which could be fleshed by an application developer without understanding the underlying implementation details. This set of applications is grouped together as templates and developers are given “handles.” The challenge with the Object-Oriented application frameworks is that, the application developer should be
knowledgeable about the framework to apply the abstract models defined in each domain to individual part of the application (Marchetto & Trentini).

Several attempts to migrate regular software development frameworks to web frameworks were made. This paper (Marchetto & Trentini) is an attempt to apply Object Oriented framework to web application development. Apart from applying the metrics of an Object Oriented framework, it also focuses on the quality of the framework and hence includes factors like reliability and testing into the framework. The paper describes a UML model that describes the mapping between the web components and the Object Oriented components in a software application. Once the meta-model is customized according to different web applications, the model is injected with lots of quality factors. They have done extensive literature review to define quality rules and implement those rules into the framework. The quality metrics measured are testability, error proneness, reliability and fault tolerance. These metrics are later normalized in the range 0 to 1. They have also conducted studies to identify the relationship between the quality metrics.

2.5.4 **Yii Framework:**

The Yii framework was developed for PHP web applications. It follows the MVC pattern. It is a component based framework specifically designed for high-performance large scale web applications (The Fast, Secure and Professional PHP Framework). It stands for easy, efficient, and extensible framework. It was developed as a generic framework that can be used as a website, portal, content management system, or forums. Yii adds a front-controller to the existing MVC design pattern. This front-controller takes care of the section dispatching and context processing.
Most of the frameworks described in this section are applicable for special domains like health care or enterprise applications, whilst the SMART WAF is targeted at both general purpose and domain specific web applications.
CHAPTER 3: .NET MVC3 FRAMEWORK

The Microsoft® ASP.NET MVC framework incorporates the classic MVC pattern. Several frameworks have been developed based on the MVC pattern due to the features it offers. MVC is considered as one of the best examples to use the SoC principles effectively (Lhotka, 2006). MVC pattern is a 3-tier architecture model for web applications, where the data layer is the model, the logical layer is the controller, and the presentation layer is the view. However, the distinction between the MVC and other 3-tier architectures is in the control flow.

This section describes the Microsoft® ASP.NET MVC framework on top of which SMART is built. In section 3.1 SoCs in Microsoft® ASP.NET MVC framework is briefed. Section 3.2 goes into details of how these concerns are architected as a framework together. In section 3.3 variation of the basic MVC architecture is described. Section 3.4 describes the control flow in the Microsoft® ASP.NET MVC framework. Section 3.5 describes the process flow in the framework. In Section 3.6, we look into the detailed routing mechanism of the process flow. Section 3.7 gives an overview of how the web development with MVC framework takes place. In section 3.8, hotspots where improvements are possible are described in detail.
3.1 Separation of Concerns

The architecture of .NET MVC follows the traditional architecture of the MVC pattern. The communication among the model, view, and controller in .NET MVC is illustrated below.

Figure 10. ASP.NET MVC Communication Model

The controller renders corresponding views and constantly updates model based on user actions. The model might or might not be closely tied to the view. The only reason .NET allows interaction between a view and a model is to allow intermediate models called as ViewModels corresponding to different views, and a single model.

3.2 .NET MVC Architecture

ASP.NET MVC framework has grouped web application responsibilities into model, view, and controller. The communication among these components is similar to the communication in the MVC pattern.
The Model is responsible for the data objects and real world entities. In ASP.NET, it has additional responsibilities to maintain the state of the entities. The model implements a repository of handles, which are used by the views to query the state of a web application. Whenever there is a change in the state, the model updates the repository object, which is then used by the rest of the application.

![Diagram of ASP.NET MVC activities]

Figure 11. ASP.NET MVC Activities Illustration

The view is responsible to show data entities in a presentable way to the user. It also takes note of any changes to the presented data entities and posts back data to the controller, which will update the data objects behind the scene. The view represents the
entire web application to the user and is a critical part of the web application where collaborative development from design and development team is needed. The view queries the model repository for any state changes while rendering a webpage.

The controller is the entry point into a web application and it interacts directly with user actions. Since a controller deals primarily with user actions, it is filled with ActionResult code. Each ActionResult might invoke another action or another controller or render a view. Internally in ASP.NET ActionResults are closely tied to views. The controller also manages the model updates by updating the data objects via the repository.

3.3 Architecture Variations

The .NET MVC framework became popular because it is based on the well known MVC pattern. It involved open standards like HTML, XML, and SOAP (M. Keith Mortensen, Infusion Development; Rob McGovern, Infusion Development; Charles Liptaak, Microsoft® Corporation, 2003). This new paradigm for web services and web applications made web development easy for web developers. According to the article (M. Keith Mortensen, Infusion Development; Rob McGovern, Infusion Development; Charles Liptaak, Microsoft® Corporation, 2003), ASP.NET MVC has reduced the development cycle of web applications quite well.

3.3.1 Centralized Page Controller and Front Controller Pattern:

Though the entire control mechanism is hidden from the developers, Microsoft provides users with customizing control flow in one of the two ways. One is by adapting a centralized page controller method and the other is by following the front controller
pattern in ASP.NET. Web applications can have a base class and application controllers can inherit from this class. This method of inheriting a page controller is called as the centralized page controller method. The other alternative to this method is to have a single application level controller, which processes the user requests similar to the struts controller.

3.3.2 Problems with MVC Pattern:

The MVC pattern handles separation between the model, view, and controller. However, this separation is not very important. In fact, this separation is often considered less critical or even omitted (Fowler, 2003). MVC is often used for thin client development, and these thin clients inherently separate the view and the controller. The controller usually runs on a server and the view runs on a client browser.

The traditional MVC pattern does not group actions. For example, consider a user record website. When a user edits his or her profile, the resulting action would be to redirect the user to his or her profile page. When the user registers in the web application, the resulting action would be to redirect the user to his or her profile page too. The same is the case when the user deletes his record. These actions could be grouped together.

Placing common code in a common area would reduce duplication. However, in traditional MVC, controllers are so closely tied with actions that partial actions can be repeated in one or more controllers. Therefore, grouping these partial actions into a base controller and invoking it based on application logic would be more appropriate.

If one adapts the ASP.NET MVC that achieves the separation of concerns for model, view and controller there are quite a few problems.
• When the application becomes complex, the ability to reuse disappears eventually. MVC usually has redundant code when the application is more complex.

• If a developer prefers to use scripted server pages like ASP.NET, it would be difficult to group similar actions because the controllers have to be separate to handle these scripted server pages. In addition, debugging and testing scripted server pages in large scale web applications becomes difficult.

3.3.3 **Page Controller Pattern:**

ASP.NET MVC uses the page controller pattern to overcome all the mentioned issues above. The page controller pattern of ASP.NET MVC uses an interception and dispatch logic.

![Page Controller Pattern MVC Architecture](image)

**Figure 12. Page Controller Pattern MVC Architecture**
When there is a page request, the corresponding action on the model is invoked, and the corresponding view is dispatched. This mechanism is hidden from the developers and the developers are given handles to handle these events inside the view. In each view, the developer can look into the code behind and decide the corresponding action. The page controller method, is one way of separating the dispatch logic from the view logic in the controller. By abstracting out the page controller, code duplication is reduced significantly and the consistency and testability of the application are increased significantly. The common ways to design a reusable web application is not to have a one to one mapping between the controllers and every URL in the web application. This will lead to duplication of code for every request. Instead, similar actions should be grouped together into a base class.

Figure 13. Grouping Common Actions into the Base Controller

For example, in the above diagram, session management, validating a user for each page, query strings common for all pages, might all go into the BaseController, whilst, Page1 specific query strings can go into Page1Controller. This will keep the
application’s design simple and the changes migrate in a top down fashion comfortably, without needing to have different testing points. This method works very well for web applications, which have functionalities in common. Sometimes, developers might have to figure out common functionalities in different levels of the hierarchy.

### 3.3.4 Problems with the Page Controller Pattern:

Sometimes, trying to group similar functionalities could cause bottle necks for changes in the system. In such cases, dropping page controller pattern and using either front controller pattern or helpers would be much better. Since the page controller is closely coupled with the HTTP-based web request, it has reference to the query strings and other view related parameters. To avoid this, the developer could simulate these view parameters for testing purposes and later integrate it with the WAF. Again, this will result in a buggy code and it is time consuming. To avoid this page controller can further be divided into view-dependent and view-independent code.

![Diagram: Abstracting web Dependent and web Independent Code](image)

**Figure 14. Abstracting web Dependent and web Independent Code**
In the above diagram, the BaseController has the controller logic and the rest of view dependent code is in the AspNetController. When the web request comes into the system, the AspNetController can parse all the view related web parameters and pass them to the BaseController by injecting it via a viable data structure. Now, this SoC has made more room for reusability as well as testability. We test the individual controllers as well as reuse the base controller for other set of completely different views.

Some of the other common problems with the page controller pattern are described below. Most of them are the limitations of inheritance concept.

- **Dependencies**: Intermediate controllers result in redirects and it takes time for the request to be processed by the corresponding controller. So keeping this intermediate layer of controllers as thin as possible will result in a good tradeoff between a flexible architecture and reduced dependencies.

- **One controller per page**: The key constraint of a page controller method is to create one controller for each web page, which increases the code redundancy multifold.

- **Deep inheritance trees**: As the inheritance trees grow deeper, it follows the classic dependency problem. To maintain the inheritance factor, a developer has to lose flexibility to design each page differently.

### 3.3.5 Advantages of Page Controller Pattern:

Page controller pattern makes developing the web application quite easier. Since each page ties to a controller, the architecture portion of the web application in the logical layer is voided. The mapping between controllers and web pages are clear, two or more
developers can work on individual pages and this parallel development increases productivity. To scale an application with page controller pattern, new controllers has to be added and the existing application remains untouched without any need for change.

Page controller pattern attained popularity in the past few years and most of the frameworks, which evolved, associated this pattern with server pages like ASP, JSP, and PHP. The problem with this interpretation is that the server pages are not clear in separating the concerns. Therefore, action code and presentation code go hand in hand making it difficult to abstract out the common controller functionalities. Hence, it was deemed wrongly that the page controller pattern was a bad design and the front controller pattern was a good design. However, this perception evolved from the specific implementations, which were in turn faulty and hence both these patterns are viable choices for good and appropriate architectures (M. Keith Mortensen, Infusion Development; Rob McGovern, Infusion Development; Charles Liptaak, Microsoft® Corporation, 2003). However, using them interchangeably does not let the developers reap the benefits of these patterns and in turn makes their application more complex for designing, developing and testing.

3.3.6 Front Controller Pattern:

For complex applications, which result in deep inheritance hierarchies, front controller design pattern is the most suited. This pattern curbs the redundancy in code and increases the flexibility for complex applications. Similar to the page controller pattern, the front controller pattern requires grouping the common logic into a single location. This basic level of abstraction would later enhance the testability and usability of the
code. Until this portion, both the page controller and the front controller abstract the logic in the same way. However, a front controller pattern is more sophisticated in that it does not add this common logic into a base class and fatten it over time. In addition, a page controller has single object per page and hence applying a common action across all pages is not possible. In addition, implementing a common action by requiring coordination across several pages is a recipe for bugs. This is especially true for the most common and important cross-cutting actions such as security and fault-tolerance. This can be eliminated with the front controller pattern (Reenskaug, Models-Views-Controllers, 1979).

Figure 15. Front Controller Pattern MVC Architecture
In the front controller pattern, the entire input request to the web applications are funneled through the front controller. The front controller in turn contains portions of command and handler.

![Figure 16. Front Controller Pattern's Internal Architecture](image)

The handler is the manager within this front controller and it receives input via the front controller. This action sequence is similar to the action sequence between the controller and the view. The commands inside the front controller follow a pattern as described in this reference (Gamma, Helm, Johnson, & Vlissides, 1995). The command execution is the last step in the front controller’s execution. Once that completes, it naturally renders the appropriate view.
3.3.7 Advantages:

One of the primary advantages of using a front controller is that it resolves the decentralization problem. A well coordinated action could be performed using a front controller pattern without any issues. Centralizing the dispatch concept helps in maintaining a single configuration file. Front controllers not only simplify the configurations in a web server, but also allows dynamic addition of commands (Fowler, 2003) (Alur, Crupi, & Marks, 2001).

3.3.8 Disadvantages:

In a front controller pattern, the performance degrades heavily, because a single controller handles all the requests. This becomes a bottleneck when the application has many requests to serve. One of the reasons why the front controller did not become quite famous in ASP.NET development is that the front controller is a special type of controller.
and it cannot be built with the default built-in controller of the framework. It has to be
developed independently and then integrated into the application, which will take quite a
lot of time for development and maintenance.

3.4 Control Flow Architecture

Control flow architecture defines an order in a series of function calls, statements,
or commands executed in the system. In MVC, the control flow starts from the controller,
which is invoked by the routing modules inbuilt in the ASP.NET MVC architecture.

![ASP.NET MVC Control Flow Architecture](image)

Figure 18. ASP.NET MVC Control Flow Architecture

The controller invokes the ActionResult requested by the URL and executes it.
The execution of ActionResult is a two-step process. In the first step, the model is
updated. If the URL calls for any change in the repository objects, they are persisted. If
the URL tries to retrieve the model data then, the controller is responsible for retrieving this data. In the second step, the controller invokes the ActionResult’s view as per the mapping defined in the MVC architecture. The model objects are passed onto the view for the user to see. The view is tied with the model for maintaining the state of the system. Views can query the model for state information and represent themselves with the updated state. The model object in the MVC is further divided into database, ViewModels, and the repository. Database contains the original data entities of the system. The repository interfaces the database with the rest of the system. ViewModels are model representations of the view elements. For example, in a simple registration view, the ViewModel will contain the registration class as follows,

```csharp
namespace Application.Models.ViewModels
{
    public class RegisterModel
    {
        public string UserName { get; set; }
        public string Email { get; set; }
        public string Password { get; set; }
        public string ConfirmPassword { get; set; }
    }
}
```

Sample Code 1. View Model Sample from MVC Application

The registration view corresponding to the ViewModel is as follows,
When the view queries the state of the system, validations for these fields can be added in the model or in the view to make it stricter or lenient respectively. When the username field is marked as the required field in the model, irrespective of the client side validation, any view implementing the model always validates the user name field.

3.5 Process Flow Design

This section describes the high level GET and POST request flow through the system and how the process flows between the major components of the system.

When the client issues a GET request to the controller, the controller requests the model objects that are related to the page being requested. The controller creates a ViewModel object and instantiates the database access interface. The repository returns updates to the controller via ViewModel objects.
For example, the Account Controller has an ActionResult LogOn as follows,

```csharp
namespace Application.Controllers
{
    public class AccountController : SessionController
    {
        //
        // GET: /Account/LogOn
        
        public ActionResult LogOn()
        {
            return View();
        }
    }
}
```

Sample Code 3. Account Controller with LogOn ActionResult
The Account controller can be invoked with this URL: http://<Application Name>/Account/Logon. There is no ViewModel associated with this corresponding GET, so the view is invoked with empty parameters. Even though the view is invoked with empty parameters, the view code still has a mapping to model as shown below.

```csharp
@model Application.Models.ViewModels.LogOnModel
@using (Html.BeginForm()) {
    <div class="login">
        @Html.ValidationSummary(true, "Please correct the errors and try again.")
        <fieldset>
            <div class="editor-field">
                @Html.TextBoxFor(m => m.UserName)
                @Html.ValidationMessageFor(m => m.UserName)
            </div>
            <div class="editor-field">
                @Html.PasswordFor(m => m.Password)
                @Html.ValidationMessageFor(m => m.Password)
            </div>
            <div class="editor-label">
                @Html.CheckBoxFor(m => m.RememberMe)
                @Html.LabelFor(m => m.RememberMe)
            </div>
            <button type="submit" class="btn">Log In</button>
        </fieldset>
    </div>
}
```

Sample Code 4. LogOn View

The view is always tied to the ViewModel to maintain or update state changes. In addition, this mapping is required in the post action when the user actions affect the model state.
When the user completes the required action on the web page, the submit button generates a POST request to the controller.

Figure 20. MVC HTTP POST Request Sequence Diagram

The POST request inherently has the model updates from the view and the controller accesses these updated ViewModel objects. The controller has a post ActionResult that has ViewModel objects as its parameters. Now these updated objects’ properties are sent to the model repository and the repository in turn executes queries to persist the data entity properties. For example, in the Account controller class, the
parameter LogOn model indicated that the model was passed from the view to the controller inherently when the POST was invoked in the view.

Properties like username and password can then be used inside the controller to invoke repository’s functions like ValidateUser(). This function in turn has queries to validate the user with the given properties and return a boolean value based on the results.

namespace Application.Controllers
{
    public class AccountController : SessionController
    {
        // POST: /Account/LogOn
        [HttpPost]
        public ActionResult LogOn(LogOnModel model, string returnUrl)
        {
            if (ModelState.IsValid)
            {
                if (Membership.ValidateUser(model.UserName, model.Password))
                    return RedirectToAction("Index", "Flow");
                else
                    ModelState.AddModelError("", "Error.");

            // If we got this far, something failed, redisplay form
            return View(model);
        }
    }
}

Sample Code 5. ViewModel Object being used for evaluation inside the Controller

A success message is returned all the way from the database to the controller. Once the Controller receives the success message, it either calls another controller for further processing or redirects to another controller based on the application demands.
3.6 Process Flow Routing Design

The process flow routing design is the low level explanation for the process flow described above. It explains the components are mapped to call one after another and maintain a navigational flow in the system. Let us consider a scenario where the user tries to access the URL: \[http://<server>/home/index/2\].

![Diagram of ASP.NET MVC Low Level Process Flow Architecture]

Figure 21. ASP.NET MVC Low Level Process Flow Architecture
When the MVC Application starts, it always goes to the Global.asax file and looks for the routing information. This routing module is responsible for setting a default redirect point when the application does not have an initial state. The routing module looks something like this.

```
namespace MvcApplication
{
    public class MvcApplication : System.Web.HttpApplication
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                "Default", // Route name
                "{controller}/{action}/{id}", // URL with parameters
                new { controller = "Home", action = "Index",
                        id = UrlParameter.Optional }
            );
        }

        protected void Application_Start()
        {
            RegisterRoutes(RouteTable.Routes);
        }
    }
}
```

Sample Code 6. MVC Application's Routing Module

The ApplicationStart() function registers all the routes and the routing module, RegisterRoutes() is responsible for mapping the URL with the components inside the architecture. As you can see, the RegisterRoutes() uses the following pattern “{controller}/{action}/{id}” on the URL “/home/index/2”, which means the home
corresponds to the controller name, index corresponds to the action name and the 2 corresponds to an optional id. The structure inside the application will be as follows.

Figure 22. Controller and ActionResult Structure in MVC Application

Now the Controller class has two methods Index() and About() both of which calls the function View().

Figure 23. View Mapping inside MVC Application
The internal implementation of this function, ties each of these ActionResults to their corresponding views even though the function call is the same.

The view renders the HTML and uses either a master layout or a shared layout inside the application to format the HTML. This is the entire process flow when the application gets into a new state because of URL invocation.

### 3.7 Web Development life cycle

This section describes the development style with MVC architecture from a developer’s perspective. Once the requirement collections phase is complete, the developer decides on the WAF. There are three different approaches to code a web application in MVC Framework: model first, database first and code first. Each of these approaches is used for different requirements in the model layer.

Model First: In this model first approach, the ViewModels are written to represent the tables in the database. There is a close coupling between the model and the database in this methodology. This is similar to the MVC pattern in Java and Ruby on Rails. Once the models are written, the database is generated from the model. This method is suitable when the developer wants to control the relationship between the tables and hand code the normalization. This is also suitable for developers who are comfortable with design, because ASP.NET provides a design interface where the developer can design his application and then generate databases from the design. The model first approach gives
huge flexibility in the entity class being generated. The developer can edit or modify the entity class file when needed.

Figure 24. Database First, Model First and Code First Development Methodologies

Database First: In the database first approach, the developer creates the data repository for the applications and uses a SQL server management tool to design the relationships and constraints. This is more suitable for developers who like to visualize the system from a database point of view. In addition, this approach makes it very easy to use existing database. This was the first approach released by ASP.NET in the MVC pattern. There are several database providers, which take care of the incremental updates for the database first implementation.
Code First: In the code first approach, everything from the table to the model is hand coded. No design tool is used in the process. This is suitable for developers who are not comfortable with design tools and who would like to control the code flow. Initial release of code first approach did not let developers deploy the database incrementally which made it inadequate when deploying web applications in production environment. However, the current release has enabled migrations similar to Rails MVC.

Here are the steps required to build a web application using the ASP.NET MVC framework.

Figure 25. Develop a MVC Application Use Case
The developer chooses one of the methods described above to code the model layer. The model layer should have both the database and the model design.

The next step would be to create the ViewModel, which ties the essential elements from the model to the view. What goes into the ViewModel is decided based on what state information the model wants to communicate to the view.

Once the ViewModel entries are decided, the developer has to create a repository interface, to update the ViewModels and the object properties in the database. The repository interface can follow a Unit of Work Pattern or a Repository Pattern depending on the demand of the web application. This interface has queries for Create, Read, Update, and Delete (CRUD).

Once the Model is created, the developer has to group actions within his application. For example, registering, change password, login, and logoff might all belong to the account action as they all act upon the account of a user. Therefore, a controller named “Account” is created and actions within the controller will correspond to the activities, register, login, logoff, change password and so on. Every controller inside a web application should inherit from the controller of the MVC Framework. This is done to inherit the hidden implementation details of ASP.NET’s MVC. The developer might choose to have a session controller implementation too. With this step, the logical layer is complete for use.

For the presentation layer, both the designers and developers collaborate to work together on the styles and the HTML content. Every view should correspond to an ActionResult in the controller except for the shared views, which can contain shared
layouts for all the views. With this collaboration, successful integration of the styles and HTML is possible and the presentation layer is complete when all the required views for the applications are created.

Now the application is ready to be launched for further improvisations.

3.8 Analysis of Hotspots

This section describes the throttles in the development of an ASP.Net MVC web application. The default MVC implements the page controller pattern, as the front controller is difficult to maintain. The page controller pattern has been refined in the recent days and the one controller per view exists no more. With the advent of partial views, it has become possible to tie an ActionResult to generate multiple views.

For example, say that they developer wants to develop four Views:

- View 1 shows a calendar control, map and the user profile information.
- View 2 shows the user profile information only.
- View 3 shows the calendar control, map and traffic update control.
- View 4 shows the calendar control and the map.

Now the developer can take advantage of the partial views and create the following partial views,

- Partial view with calendar control and map
- partial view with user profile information
- Partial view with traffic update control

Now the views are mapped to the partial views as shown in the diagram below.
Figure 26. View1 with PartialView1 and Partial View2

Figure 27. View2 with Partial View2

Figure 28. View3 with PartialView1 and Partial View3
Therefore, instead of four views, there could be three partial views. Irrespective of having partial views, the page controller pattern generates a lot of redundancy in the controller code. For demonstration purposes, let us consider a scenario of an interactive website. The website might have sections like questions about lung cancer, skin cancer and stomach cancer. Each of these sections might have data about these cancers and a small quiz at the end. For such a small-scale application, developers come with a better enumeration of controllers, but for the purpose of this research let us assume that this is a large scale application and emulate how controllers would be designed for such a scenario. The way ASP.NET MVC page controller pattern works is that, the developer would want to treat these sections individually, even though they contain similar elements like “data” and “quiz.” Therefore, the controllers enumerated for this application with the traditional MVC would be as follows.

- LungReportController.cs
- LungQuizController.cs
• SkinReportController.cs
• SkinQuizController.cs
• StomachReportController.cs
• StomachQuizController.cs

Each of the report controllers is going to display HTML content from the corresponding views and each of the quiz controllers is going to display questions to the user and update answers into the table. The entire code repeats itself in each of the above sections. This redundancy is caused because of how SoCs work. The architecture’s logical layer has potential room for abstraction and reusability. Another factor that limits the developers from using inheritance in these situations is how the navigation model is implemented in this framework. The ASP.NET’s MVC navigational model is integrated into the controller and it applies to all controllers. This means, each controller has the logic to redirect to the next controller and so on. Since the controllers are written based on a class of actions and aiding the navigational model in the web application, classifying the controllers like lung, skin and stomach makes it difficult to get better reusability over these classifications. Unless there is a generalized pattern to organize these classifications with another top-level hierarchy, it would be difficult to devise a new navigational model to improve the reusability and decrease the redundancy factor.
CHAPTER 4: SMART FRAMEWORK

Software reuse involves writing reusable functional code snippets as library modules. With respect to web applications, it is widely believed that the possible places for reuse could be the interface templates, database components and small plug-ins (Pree, 1994). Apart from these, there are several places in web application where code reuse could contribute significantly to reduced development time, lower number of bugs, increased reusability, and increased productivity.

This section describes the SMART framework, which allows extensive reusability by defining a new navigational model for the MVC and by integrating object oriented abstraction into the MVC Separation of Concerns. In section 4.1 problems with single dimension, separation of concerns is explained in detail. In section 4.2, the SoC pattern derivation for the SMART framework is described in detail. In section 4.3 Separation of concern in SMART, MVCT framework is briefed. Section 4.4 describes the control flow in the Microsoft® ASP.NET MVC framework. Section 4.5 describes the conceptual framework of SMART and introduces the navigational model for SMART. Section 4.6 describes the process flow in the framework. In Section 4.7, we look into the detailed routing mechanism of the process flow. Section 4.8 gives an overview of how the web development with MVCT framework takes place.
4.1 SoC Vs Redundancy

SoC software paradigm is aimed at division of responsibilities and building reusable constructs. However, will increasing SoC decrease redundancy within a web application? In effect, it does, but the separations as well as the concerns are relative. For example, in MVC SoC is applied in a single dimension and hence we have Model, View and Controller. By introducing a second dimension, we can in turn abstract each of those layers further. When do we start applying new dimensions depend on the need for one (Gurp & Bosch, Separation of Concerns: A Case Study). It is futile to keep increasing the concerns in the single dimension, because this would in turn render more and more redundancy beyond a certain point.

The SoC community is trying to address problems with the concerns and their separations. Significant research work has resulted in extensive products like Aspect Oriented Programming (AOP) (Kiczalez, et al., 1997), multidimensional SoC concept (Tarr, Ossher, & Harrison, 1999) and Subject Oriented Programming (SOP) (Harrison & Ossher, 1993). Several research authors have talked about identifying patterns and grouping them by hierarchy, which could result in a modular SoC (Buschmann, Meunier, Rohnert, Sommerlad, & Stal, 1996) (Clarke, Harrison, Osher, & Tarr, Subject Oriented Design: Towards Improved Alignment of Requirements, Design and Code, 1999).

4.2 Soc Pattern

SoC or OO abstraction allows the developers to visualize objects as data and function. Most of the time modularity is focused on the functional level. A software
application data is different from a web application. For example, let us consider the picture below. This is a natural inheritance hierarchy for a software application.

![Inheritance in Animal Hierarchy](image)

Figure 30. Inheritance in Animal Hierarchy

Now let us emulate the same example as a MVC application. Let the web application have two web pages, dog, and the sheep. Each of these web pages can have functionalities like write, run and attributes like yellow. The structure of the MVC application for this example would be as follows,

- Dog and Sheep views
- Animal, Dog and Sheep controllers
- Dog and Sheep models
This design reduces the redundancy to a great level and increases the abstraction and reusability by taking advantage of the inheritance concept.

Now consider the following example, where there is not much difference between the dog and sheep lass. However, MVC architecture would need separate controller classes for dog and sheep, just to support the navigational model in the architecture.

![Figure 31. Variation in Animal Inheritance Hierarchy](image)

Since we are emulating this example as a web application, we should also consider presentation of dog and sheep on the web page, which means there are functionalities like drawing pixels and lines. In the first picture, the sheep and dog are of different colors and hence separate functionalities could be written for them, but this would result in redundancy in the second diagram where both the dog and the sheep are white in color. In the MVC layer, there is no abstraction for abstracting out this color.
information. The traditional MVC has Model, Controllers, Views and ViewModels, but there is no way to abstract controllers based on their respective view categories. Here, strictly separating the concerns controllers and views would increase redundancy. Now, visualize the same application with over 15 different animals having eight different colors and they will have much in common. The redundancy factor increases as the complexity of the web application increases, because the web pages will have both similarities and differences between them. “Taking advantage of the similarities within a ‘Concern’ while tending to the differences within the same ‘Concern’ is the need of the hour.”

Lowering the strictness between the controller and the view, allows abstractions in the second dimension over the controller with respect to the view. This can result in a better reusable and constructive architecture.

4.3 MVCT Architecture

With the discussions above, it is clear that there is still room for more abstraction in the MVC architecture. However, what could be abstracted has to be identified from an object oriented perspective. Let us consider common web applications and look into what they deliver to the consumers. Most of the content inside a web application is text, graphics, audio, video, and plug-ins. Within content, graphics, audio and video most of the WAFs store these media information as files and render the URLs in html tags. In a way, these could be considered as text too. Especially Microsoft® ASP.NET gives the ability to render decoded HTML from database onto the view. So, a controller that maps to a text view to generating text, questions, graphics, audio and video could inherit the
from a single text controller class. To generalize, when a form with input elements is
displayed in the view it is considered a write typed view, when the same form has only
content the view is considered to be read only typed views. Within the read and write
typed content view, a developer might see more “types” and might want to have more
classifications. This concern belongs to the second dimension of the MVC framework
and it is applied over the combination of controller and views.

![MVCT Pattern in SMART Framework](image)

Figure 32. MVCT Pattern in SMART Framework

Since view and controllers are closely tied to the model object, the
implementation of the SMART architecture framework includes type concern even in the
model layer. However, a developer who chooses not to use this implementation can do so
flexibly. Using “types” in the model layer makes the framework more data driven and behaves differently to different data inputs.

For example, the SMART framework can have a text controller and a text view, which processes text objects and displays them. The text content is either stored in the database or in a file for rendering. In either of these ways, the application will have a text table, which maintains all the text content of the application or the location of the text content file. If the developer chooses to not have a single text table, but have a more meaningful database layout like the lung cancer text table, skin cancer text table, he or she has the flexibility to follow that design inside this architecture as long as the developer is able to visualize the connections between the controller and the view.

4.4 Control Flow Architecture

The SMART framework has identified three general types of content being served in web pages so far. There could be more types for individual applications and this framework does not aim to integrate all such types, but only to give the design perspective to the developer with these three basic types. It is a developer’s discretion to use zero to infinitive types as the developer sees fit. The typed controller calls the corresponding typed View to display the typed model object. The view, which is closely tied with the model object, should be of the same type too, so that they can share the state of the application. The view sends back POST data for the writable types to the controller and the controller updates this data into the database.
For simplicity, let us consider a Question type model.

```csharp
namespace Application.Models.ViewModels
{
    public class QuestionListView
    {
        public List<QuestionView> Questions { set; get; } 
    }
}
```

Sample Code 7. Question Type View Model Sample
The Question model itself contains an id, question text, a list of choices, and the selected answer for the questions.

```csharp
namespace Application.Models.ViewModels
{
    public class QuestionView
    {
        public int QuestionId { get; set; }
        public string QuestionText { get; set; }
        public List<Choice> Choices { get; set; }
        public string Answer { get; set; }
        public string SelectedAnswer { get; set; }
        public QuestionView()
        {
            Choices = new List<Choice>();
        }
    }
}
```

Sample Code 8. Question Type ViewModel Sample

Each choice in the system can be treated as a data element in the choice model and hence it contains text and id.

```csharp
namespace Application.Models.ViewModels
{
    public class Choice
    {
        public int ChoiceId { get; set; }
        public string ChoiceText { get; set; }
    }
}
```

Sample Code 9. Choice ViewModel inside the QuestionViewModel

Once the model is defined, the Question controller takes care of bundling these objects together and invoking the view with this bundled object.
The controller now sends this list of questions to the view, which refers to this model object as described in the previous sections.

4.5 Conceptual Architecture

The conceptual architecture of the SMART framework describes about the navigational model that has been developed in this application. Unlike the page controller and the front controller pattern, the SMART framework’s navigational model tries to overcome all the potential issues by creating a hybrid pattern from the page controller as well as front controller pattern. This conceptual framework diagram shows how the navigational model works in SMART.
The SMART framework has a flow controller, which acts similar to a front controller. However, the disadvantage of the front controller is that it is not the same as the inbuilt MVC controllers. Instead, developers have to develop this controller
individually and hence maintenance is difficult. Instead, the flow controller in the SMART framework handles all the flow patterns in the application with the MVC’s built-in controller.

Every type controller in the application hands over the control to the flow controller after rendering its view. This separation of navigation model helps in restructuring the navigational flow of the application at any time during development or maintenance. The flow controller is fed with the page information. The page information can either be stored in the database or as a flowchart, which will then be loaded as xml into the application.

In SMART, it is implemented in the database for simplicity. The page data repository has all the page names and entries for the next and previous pages. Few web applications might want to group the pages into sections and hence section information is stored within the page repository. The developer can choose to add any level of hierarchy in the page repository.

Apart from maintaining the page navigational model, the repository also maintain what type the page is, this is where type enters the Model layer. A simple page repository can be implemented as follows.
The default page in the SMART framework is always the flow controller. The flow controller makes a navigational decision based on two types of data. One is the page name and the other is the flow type.

Figure 35. Page Repository Design in SMART
Once the next page is decided, the page type information is obtained using the database repository. Since the controllers are also typed, the flow controller can just call the TypeController.cs with the page name.

Now the TypeController.cs, which can be a QuestionController.cs, knows the page name and hence looks into the database for the content of that page. Since the page model is being used in all the controllers, a hybrid model TypePageViewModel could be created for all the types.

The developer might also choose to combine the two models as TypePageViewModel <TypeViewModel, PageViewModel> and process it from inside the controller. Once the controller completes its execution this tuple ViewModel can be passed to the view and the view handles the data in a usual manner. In the database, every type table is linked to the page table as follows.

```csharp
if (isAlt)
    nextPage = _dbrepo.getPage(currentPage, "nextalt");
else if (isCurr)
    nextPage = currentPage;
else if (isNext)
    nextPage = _dbrepo.getPage(fcurrentPage, "next");
```
4.6 Process Flow Design

The process flow design of the SMART is similar to the MVC process flow, except for the flow controller, which is handled before any other controller is handled. The flow controller still follows the same process as the default MVC controller. This section describes high level GET and POST request flow.

When the client issues a GET request to the application the first controller being hit is the flow controller, the flow controller requests the model objects that are related to the page being requested. The flow controller creates a PageViewModel object and then
instantiates the database access interface (repository pattern, in this case) and calls the appropriate functions on this object.

Figure 37. HTTP GET Request to SMART Framework's MVCT SoC

Once the flow controller receives the page information from the database, it invokes the type Controller. Once the type view is generated, the user can perform actions on the web page. When the user completes the required action on the web page, the submit button generates a POST request to the controller.

The POST request inherently has the model updates from the view and the controller accesses these ViewModel objects, which were updated by the users. The
controller has a post ActionResult that has the TypeViewModel as well as the PageViewModel objects as parameters. Now these updated objects’ properties are sent to the model repository and the repository in turn executes queries to persist the data entity properties.

Figure 38. HTTP POST Request to SMART Framework's MVCT Pattern

Since the flow controller is responsible only for navigational purposes, it is not involved in any of the user actions POST request. However, when the user navigates into the system, the type controller sends the PageViewModel Object to the flow controller to enable navigation.
4.7 Process Flow Routing Design

This section describes the navigational model in the low level architecture.

Figure 39. MVCT Low Level Architecture

Here, the readers can visualize how the URL routing takes place in MVCT in contrast to MVC. Since MVCT uses type objects which in turn has pages, the URL structure for MVCT will have “Type/Action? Page=""&id="". Let us consider a scenario where the user tries to access a simple MVCT application’s home page. The URL for this page is  
http://<server>/Text/index?page=home. When the MVCT application starts, it
invokes the routing module in the Global.asax file and looks for the routing information, which is edited from the default MVC as follows.

```csharp
namespace MvcApplication
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
        routes.MapRoute(
            "Default", // Route name
            "{controller}/{action}/{id}", // URL with parameters
            new { controller = "Flow", action = "Index",
                id = UrlParameter.Optional }
        );
    }
}
```

Sample Code 12. Register Module for MVCT Pattern with the New Navigational Model

As you can see, the RegisterRoutes() still uses the same MVC route pattern "{controller}/{action}/{id}" on the URL "/Text/index?Page=home", which means the text corresponds to the controller name, index corresponds to the action name and the action has query string named page whose value is home. Therefore, the structure inside the application will be as follows.
The flow controller now has the page information of the current page and will know what to invoke next from the page repository.

4.8 Web Development life cycle

This section describes the stages of development with the SMART framework. The first stage is the page engine feed step and the second stage is deciding the type step, where the controllers are typed based on the content in the application and the last stage is the integration between the page engine and the type engine.

To develop a page engine, the developer has to list all the pages in the order of navigation. In fact, developing a flowchart for such representation would make the design easier. Once the design is in place, start the implementation by filling in the page tables with their current, next and alternate page details. Once the page repository is ready, we
can start creating the page models, which describes the properties of the page which you want to maintain throughout the application.

For example, SMART framework implementation uses pagename, pageid, page type as some of the properties to describe the PageViewModel in the entire application. Now that the page model is ready, flow controller can be created. The flow controller should involve logic to check the attributes like the current page name, current page type and the type of navigation (previous or next) and then calculate where the navigation is going to lead based on the results from the database. The flow controller is also

Figure 41. Use Case to Develop Navigational Model in MVCT
responsible for creating the default application’s home page or error pages. In SMART framework even, the error state of the system is described as the “text” type.

Once the page engine is all set to go, the application starts the second stage of development, which involves typing the application content into categories and developing controllers for each of them.

Figure 42. MVCT Use Case to Develop Type Controllers

The control flow between these controllers is in the same way as described in the previous sections. Each controller should have a memory of what page it is serving, because this is a coordinated property until the application terminates.

Once the type controllers are in place, integrate them with the page engine that was created in the previous step.
Figure 43. Integrate Page Engine and the Type Engine in MVCT

The type controllers connect to the flow controller at the end of each POST request. The TypeViewModels use the PageViewModel to emulate a tuple data inside the controller. This tuple can then be sent to the view like any variable.
CHAPTER 5: EVALUATIONS AND USEFULNESS

The framework was evaluated against a reference of standard WAF requirements and the web applications it generates were evaluated against standard web applications. The evaluations were on social acceptability, cost incurred during development, maintenance, scalability, technical viability and operational feasibility. Measurements were taken before using SMART and after using it. The significant differences in the measurements favoring the S.M.A.R.T framework will portray the advantages of this framework.

5.1 Goal and Objectives Evaluation

The first step towards evaluation of a WAF is to check if the framework fulfils its goals and objectives. As discussed in the goals section of this thesis, SMART framework takes advantage of the page controller pattern and the front controller pattern and maintains a navigational model in a separate controller.

5.2 SMART Evaluation Tree

Based on the features offered in SMART, the following evaluation tree was constructed. Each of the leaf nodes in the tree is used for evaluation of the SMART framework against the MVC.
5.2.1 Scalability

Scalability of a web application is estimated based on two factors. One is the complexity and the other is the scalability effect. Complexity is the estimation of how complex a scalable design would be. Scalability effect is the estimation of how much code has to be changed in the current application to make it scalable.

MVC can be scaled quite well without any complexity. However, there is a limitation in the number of controllers. The controllers grow as the application grows and this increases the code size. There is no scalability effect in MVC. As new controllers are written for scalability, they are added to the application without any significant changes.
In contrast, MVCT has neither the complexity factor nor the scalability effect. Since it uses type controllers, newly added pages are classified or new classifications are creates. The scalability effect might slightly be present if the inheritance has to be modified due to newly added types.

Both the frameworks have their pros and cons for the scalability factor. The developer can choose to use the MVCT as MVC if he wants nil scalability effect.

*Hence, there is a NO clear winner for scalability factor*

### 5.2.2 Maintainability

Maintainability of MVC pattern is quite simple. If there is a change in the logic, the developer goes to the targeted layer and makes changes to the targeted controller, or a model or a view. Teams can maintain their individual components individually. However, if there is glue code change, then collaboration between teams are needed.

The maintainability factor of the MVCT pattern is better in that if there are 20 pages that need maintenance then all that has to be done is to modify the corresponding entry in database. In a similar fashion, the page flow could also be changed in a MVCT pattern by changing entries in the database.

Evaluation of maintenance also involves flexibility factor in maintaining individual components inside the framework as shown in the evaluation tree. To maintain leaf node attributes, a MVC framework will need most of the application code rewritten. With MVCT framework, it is easy because information flow in the application is customizable including process and control flow.

*Hence, MVCT wins an upper hand in Maintenance.*
5.2.3 Adaptability

Both the frameworks are compared for usability and learn ability factor. There are many resources for a new comer to understand the MVC architecture and to understand how the control pattern is implemented between the concern layers. To learn how the MVCT framework works, the developer should start with learning the MVC framework and then read the MVCT section of this thesis. At the time of writing, there is no public documentation prepared for the MVCT architecture.

However, there is one other major downside to the learning curve of MVC, which is not associated with the MVCT. When the developer tries to improve the reusability, a lot of time is spent on learning about good architecture practices, as MVC does not provide these by default. However, the MVCT framework takes a jump and is already there with the architecture details to provide the best reusable web application design.

*Hence, MVCT is a clear winner in Adaptability.*

5.2.4 Recyclability

Recyclability is completely not possible in MVC framework. When a developer develops a continuously evolving website with MVC, he has to keep trash ing portions of website that will have to change. However, using MVCT could provide recyclability for applications quite easily. For example, let the website be updated to have new content in all the pages, with MVCT just the content repository has to be changed and the website code is untouched. A non-technical person can fill the content repository with the new content. If the website has to be completely trashed and a new one has to be created, still
the developer can save the type controllers and the page controllers that created and reuse these in the newer application.

*Clearly, MVCT wins in recyclability*

### 5.2.5 Testability

MVC application is testable both in component level and in system level. MVCT provides flexible component level testing and sub group integration testing. The process flow is complex in the MVCT framework and hence testing the process flow might require more time.

*There is NO winner with respect to testability.*

### 5.3 Case Study with SWSS

SWSS was developed with SMART framework. It is a medium scale interactive health web application. So far, the architecture of the application has been changed 5 times as the requirements of the research changed. All the changes requested were implemented within 24 man hours effectively.

Here is the statistics from the SWSS web application using the MVCT pattern SMART framework. The table shows the percentage of framework code in the overall system. The table also shows, which parts of the application has significant increase in the percentage, as expected, the increase was in the controller and the model, because that is where the page and type engines influence a lot.
The table clearly shows how less custom code is required when SMART framework is used. The pie charts below are arranged in the order of decreasing percentage of custom code in the SWSS web application.

![Pie charts showing code distribution]

Figure 45. Increasing Order of framework % in MVCT Application

This arrangement shows that controllers are the most benefitted and views are the least benefitted in a MVCT Pattern. Consider 20 pages of text type views in an application. Within these pages, the requirement might call for different styles. Thus irrespective of inheritance, number of views in a system might increase.
5.4 Case Study with .Net Experts

.NET Experts from Microsoft’s .NET forum and within the research group where consulted for various sized ASP.NET MVC web applications. Design of those web applications were studied in detail. Interesting trends showed up in the reusability pattern.

5.4.1 Large Scale Application

A large scale application developed by Rion Williams, a lead software developer at Structure Co., was considered for evaluation. It was a health application, primarily containing text, audio, video, and images. When the application was redesigned with the MVCT framework the code size decreased significantly and it had reusable constructs.

<table>
<thead>
<tr>
<th>No of Action Results</th>
<th>No of Views</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVC</td>
<td>68</td>
</tr>
<tr>
<td>MVCT</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 2. Large Scale Application's ActionResult Count and View Count

Figure 46. Comparison of ActionResults and Views in Large Scale Application

MVC/MVCT
The chart above shows the significant gap between MVC and MVCT patterns for large scale applications.

5.4.2 Medium Scale Application:

For a medium scale application, SWSS was used and the statistics is as shown below.

<table>
<thead>
<tr>
<th></th>
<th>No of Action Results</th>
<th>No of Views</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVC</td>
<td>27</td>
<td>153</td>
</tr>
<tr>
<td>MVCT</td>
<td>9</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 3. Medium Scale Application's ActionResult Count and View Count

![Comparison of ActionResults and Views in Medium Scale Application](image)

Figure 47. Comparison of ActionResults and Views in Medium Scale Application MVC/MVCT

There is almost 1/3 reduction in the number of controllers in a medium scale application with varied types. The reason why the view count has gone down in this graph is that the SWSS MVCT does not use varied styles. Most of the text pages look
similar in styling, questions pages differ in styling and these were managed via partial views, and hence there is a reduction in View counts.

5.4.3 Small Scale Application:

A small scale application was designed based on the following scenario. Consider the following simple web application "user health risk assessment website.” This website will have a series of health related questions to the users and then some information is provided to the user based on how they answered those questions. Let the application have 100 pages and let these pages be grouped into sections (for e.g., lung health, heart health, skin health, bones health). The website will calculate a health risk score based on how users answered the questions from these four sections. For this application, the statistics were as follows.

<table>
<thead>
<tr>
<th>MVC</th>
<th>MVCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4. Small Scale Application's ActionResult Count and View Count

Figure 48. Comparison of ActionResults and Views in Small Scale Application

MVC/MVCT
For a small scale application, there is not much difference between the MVC and MVCT, because the framework instantiates with very less number of type controllers. Hence, at a small scale level MVCT behaves like a MVC application.

**5.4.4 Reusability Percentage of MVCT Pattern**

For the three types of the applications described above, estimations of code reuse were measured and the results were as follows.

<table>
<thead>
<tr>
<th></th>
<th>Large Scale</th>
<th>Medium Scale</th>
<th>Small Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overall System</strong></td>
<td>6503</td>
<td>2761</td>
<td>372</td>
</tr>
<tr>
<td><strong>Framework</strong></td>
<td>5678</td>
<td>2055</td>
<td>342</td>
</tr>
<tr>
<td><strong>Custom</strong></td>
<td>825</td>
<td>706</td>
<td>30</td>
</tr>
<tr>
<td><strong>% of framework code</strong></td>
<td>87.31354759</td>
<td>74.42955451</td>
<td>91.93548387</td>
</tr>
</tbody>
</table>

Table 5. Estimations of code reuse in large, medium, and small web applications

![% of Framework code](image)

Figure 49. Reuse percentage across large, medium, and small MVCT applications
As the graph shows, large scale, and small scale application framework are quite reusable, whilst a medium scale application does not have significant level of reusability. The reason for this skew is that the medium scale application has varied types and the large scale application has repeated similar types. For a small scale application, the custom code closely follows the framework code, hence it adds very little effect.

5.4.5 Development Time: MVC Vs MVCT

The three scale applications described in the above section were analyzed for development cost and the number below are in man hours.

<table>
<thead>
<tr>
<th>Size of the web application</th>
<th>MVC</th>
<th>MVCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large</td>
<td>1500</td>
<td>480</td>
</tr>
<tr>
<td>Medium</td>
<td>240</td>
<td>144</td>
</tr>
<tr>
<td>Small</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 6. Development Cost for Large, Medium, and Small Scale Applications

Figure 50. Man Hours Comparison between MVC and MVCT in a Large Scale Application
For a large web application, MVCT reduces more than half of the development time, because the newer pages have more probability to have been classified already.

![Man Hours Comparison between MVC and MVCT in a Medium Scale Application](image)

**Figure 51.** Man Hours Comparison between MVC and MVCT in a Medium Scale Application

For a medium scale application, the trend decreases but not as much as the large scale. Again, the same reasoning applies. The medium scale application is yet to evolve and the new pages might introduce newer types.

![Man Hours Comparison between MVC and MVCT in a Medium Scale Application](image)

**Figure 52.** Man Hours Comparison between MVC and MVCT in a Medium Scale Application
A small scale application is pretty much the same as the MVC framework and there is not much advantage in using the MVCT pattern, because all the initial pages might be classified as a new type.
CHAPTER 6: CONCLUSION AND FUTURE WORK

In this section, summary of what was achieved and what more can be achieved with SMART framework will be discussed.

As we proceeded through the thesis, we suggested various demands that the web application domain had. We suggested solutions to supply to these demands. We also constructed a SMART WAF along the way of describing solutions.

The framework’s ease of use and reduced learning curve which was discussed in the evaluation section proves that SMART is most suitable for a developer who prefers to use MVC with flexibility. When compared with MVC, the MVCT pattern was a clear winner in all aspects and this thesis would have given the readers a different perspective on web applications.

Future work on SMART would be to estimate its use on other frameworks and evaluate the improvement. Extend SMART into a generalized architecture with the MVCT pattern, which could be used in combination with any framework. In addition, the page engine in SMART suffers from throttling and hence an effort to distribute the flow controller would make the web application developed with SMART more robust.
 References


101


102


Lhotka, R. (2006 March 1). *Should validation be in the UI or in business objects*. From http://www.lhotka.net/weblog/ShouldValidationBeInTheUIOrInBusinessObjects.aspx


*See how technologies are trending across the web.* (2013 йил 09-April). From BuiltWith: http://trends.builtwith.com/framework/


