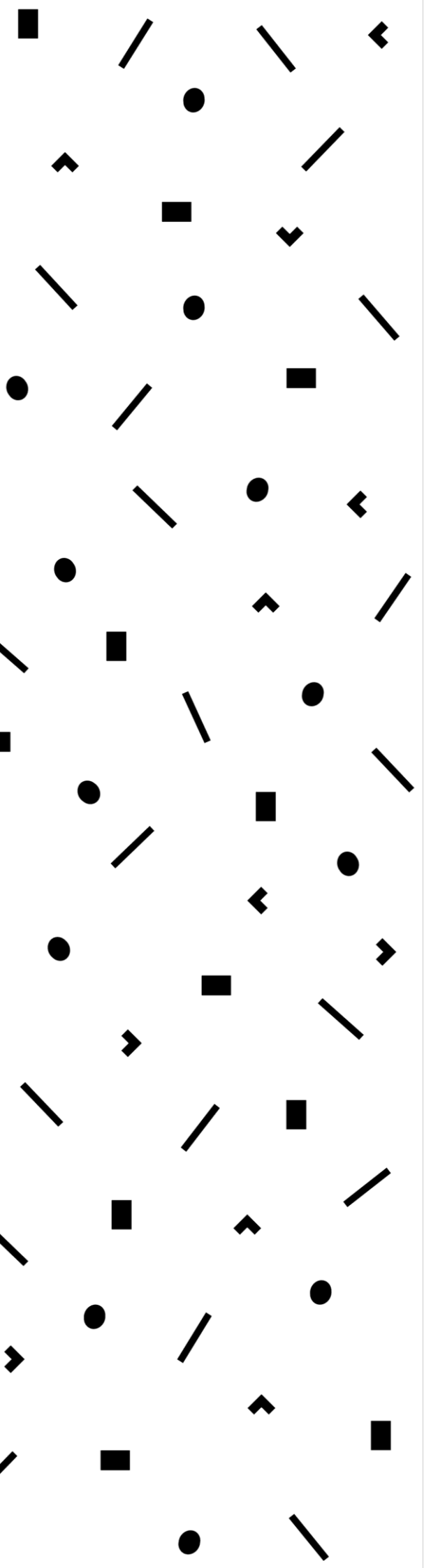


LAMK Lahden ammattikorkeakoulu
Lahti University of Applied Sciences

Migration from ASP.NET Web Forms to ASP.NET MVC

Case study: K Company– Promotion Email Web Application

LAHTI UNIVERSITY OF APPLIED SCIENCES
Degree programme in Business Information Technology
Bachelor's Thesis
Spring 2016
Ngoc Duc Nguyen

Lahti University of Applied Sciences
Degree Programme in Business Information Technology

Nguyen, Ngoc Duc:

Title: Migration from ASP.NET Web Forms to ASP.NET MVC

Case study: K company– Promotion Email Web Application

Bachelor's Thesis in Business

Information technology

41 pages, 0 page of appendices

Spring 2016

ABSTRACT

According to rumors, the soon-to-be released version of ASP.NET, which is vNext, will no longer suppor the commonly used ASP.NET Web Forms framework. On the other hand, the ASP.NET MVC, another ASP.NET framework, offers many more advantages compared to Web Forms. There has never been a greater urge to transfer existing ASP.NET Web-Form-based web applications into a new environment where ASP.NET MVC dominates**.**

The aim of this thesis was to create creating a detailed and straightforward guideline for the migration process from the old Web Forms framework to the new MVC one.

As a result of this study, a new artefact was created by combining some already available solutions for the migration. The newly created artefact is supposed to be simpler and more transparent for ASP.NET developers of various competence levels. From the author's perspective, the new artefact is applicable for both small to medium and large-scale projects. However, the migration method suggested by the author is more suitable for small- and medium-scale projects. For large-scale projects, the author will recommend another solution in the final parts of the thesis.

Key word: ASP.NET Web Forms, ASP.NET MVC, ASP.NET Web API, Object-Relational Mapping, Migration.

***Note to reader**: The case company's name has been changed to K company due to confidentiality required by the company

CONTENTS

1 INTRODUCTION

According to the daily updated report by w3techs.com about the top 10 million websites in the popularity ranking presented by Alexa (an Amazon company) at the time this thesis is written, 15.9% of websites responing to the report.are using ASP.NET as their server side technology.

ASP.NET is a web framework for building extraordinary sites and web applications utilizing HTML, CSS, and JavaScripts. With the availability of ASP.NET, it has never been easier to build dynamic and data-driven applications. Even better, such applications are compatible with various browsers without the need for developers to re-customize their applications according to each browser (Liberty & Hurwitz, 2003, p. 3). In addition, ASP.NET provides two frameworks for developers to start developing web applications: ASP.Net Web Forms and ASP.NET MVC.

ASP.NET Web Forms is the oldest framework for creating ASP.NET web applications which was released on January 16, 2002. On the other hand, ASP.NET MVC is a younger framework which was first released on March 13, 2009 with the ASP.NET MVC 1.0. By the time ASP.NET MVC is introduced, the MVC pattern has become one the most used design patterns for web development. Moreover, as the release day of the next ASP.NET version is getting closer, there are a lot of rumors that ASP.NET Web Forms will be abandoned in the next release. Hence, there is a bigger need for moving from ASP.NET Web Forms to ASP.NET MVC than ever.

Migration to a new system is the progress that is highly time-consuming and extremely risky. Therefore, the planning phase is an essential requirement for the success of the migration process. A good planning phase should be able to reduce delays and minimize costs (Jacob, 1991, p. 37). Consequently, effective planning or guideline is a must before making any migration between the pair frameworks.

Therefore, this study is conducted in order to give readers clear requirements and detailed guidelines for the migration from ASP.NET Web Forms to ASP.NET MVC.

2 RESEARCH TASK

2.1 Research problem

Until now, the latest version of ASP.NET is 4.6 which was released on July 20, 2015 and it is still supporting both ASP.NET MVC and ASP.NET Web Forms. As mentioned above, rumors have that Microsoft will cease to support Web Forms in the new ASP.NET which are destined to be introduced in the year 2016. Developers, though, can still continue to build Web Forms products in Visual Studio 2015 using .NET 4.6 framework. However, for sure, newly developed Web Forms applications will not be able to use the interesting new features of ASP.NET 5 (Walther, 2015). ASP.NET 5 stack employs MVC as the sole paradigm to develop HTML. Current Web Forms pages won't be available in the next ASP.NET vNext (Esposito, 2016). In addition, when the author of the thesis was trying to create a new web application in Visual Studio by using ASP.NET 5 preview version, there was no option for starting a new ASP.NET Web Forms application. This means, up to the time this thesis is being written, there is a high probability that ASP.NET Web Forms will not be supported in the next released version.

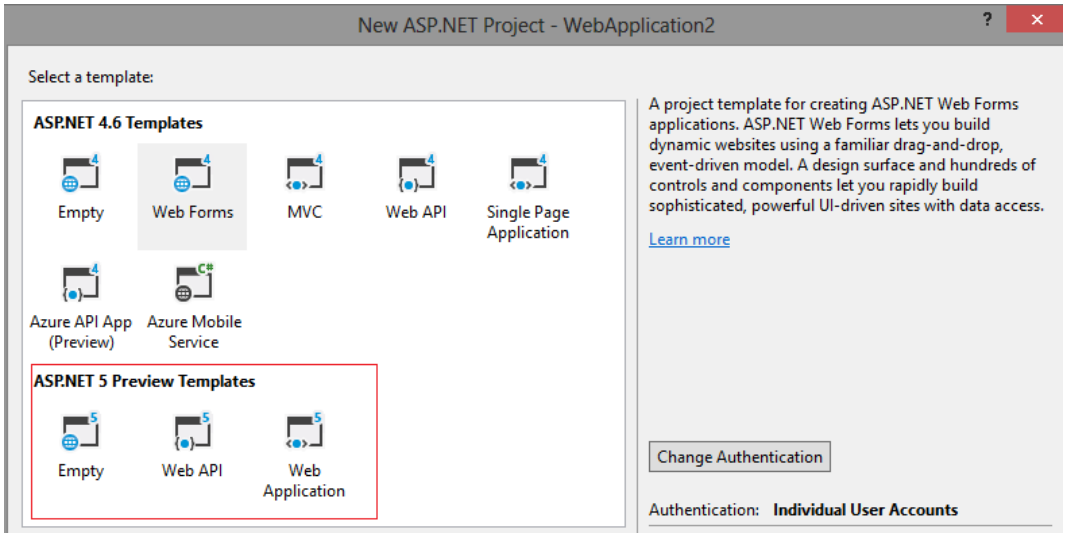


FIGURE 1: No ASP.NET Web Forms option in ASP.NET 5 Preview Templates

Even though all the information about the next version of ASP.NET are still an illusive prediction since there is nothing official from Microsoft yet, considering the many advantages ASP.NET MVC brings, it is highly advisable by the thesis author to move from a legacy technology to a new and more powerful one.

During the time of study as well as collaboration with a Startup company that uses ASP.NET as a server side technology, the thesis author has had the chance to work with both ASP.NET MVC and ASP.NET Web Forms. At the same time, the writer has perceived the weaknesses and strengths of the pair. From the writer's point of view, there is a need for adapting ASP.NET MCV to a new project and moving legacy ASP.NET Web Forms project to ASP.NET MVC.

Consequently, transferring to a new platform would result in the expense of considerable amount of effort and precise planning. However, the result, if satisfactory, would be worthy of the invested resources.

2.2 Value and contribution of the study

Since there is yet any official guideline for transferring between ASP.NET frameworks, the writer has a strong desire to study the progress of transferring frameworks, aiming at creating a general plan for the moving from ASP.NET Web Forms to ASP.NET MVC.

With the primary goal is to provide a clear structure guideline for transferring progress and the requirements for that, this thesis will be a guideline for ASP.NET developers to refer to during the planning phase when aiming to perform a migration from ASP.NET Web Forms to ASP.NET MVC.

2.3 Thesis objective and reaserch question

Despite the importance and the complexity of the transferring between ASP.NET frameworks, there has not been a guideline as well as the

acknowledgement for project managers and developers before starting the migration progress. In addition, the final aim of conducting research is to uncover answers for problems with the employment of scientific techniques and protocols (Kothari, 1990, p. 2). Therefore, the writer comes up with the following research question that will be answered by this thesis.

Research question: How to transfer from ASP.NET Web Forms to ASP.NET MVC?

Thus, the type of research question is descriptive question and in order to answer the research question, this thesis is following the descriptive study approach. According to Robson's study, the descriptive approach portrays a complete and accurate profile of a certain person or situation, either of flexible or fixed design (Robson, 2002).

2.4 Thesis structure

In general, the thesis contains seven chapters. In chapter one, the thesis writer mentions the current situation of the pair frameworks and the purpose of the thesis. Chapter two focuses on the research design of the study, the chapter has seven sub-chapters which describe the research design in details. Next, chapter three provides the theory and information needed in this study. The third chapter includes theory about ASP.NET Web Forms, ASP.NET MVC, Object-Relational Mapping – Entity framework, ASP.NET Web API and solutions from previous studies.

Chapter four provides information about the new artefact combined by the thesis writer first in a nutshell and then in details. In the chapter, each of the step of the new artefact will be described in detail and provided with example. Chapter five is about case study and artefact evolution from the author's point of view.

Chapter six will be the summary of the whole thesis and will answer the research question. Finally, in chapter seven the thesis writer will discuss

the limitation, reliability and validity of the thesis. Also in chapter seven, information about further study can be found.

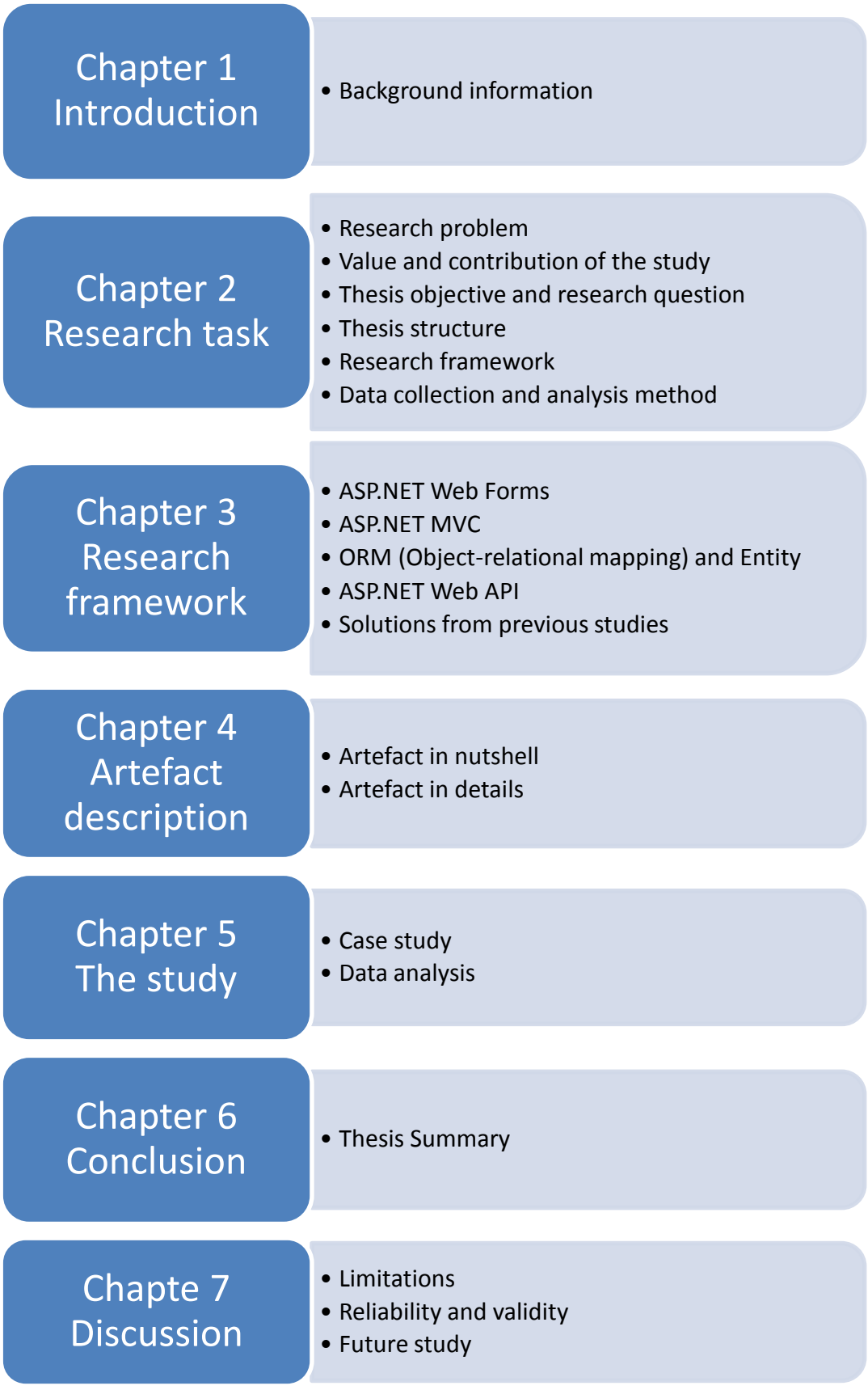


FIGURE 2: Thesis structure

2.5 Research method

Research methods can be understood as a series of protocols, planning and algorithms that serve the research purposes. By using research methods, researchers employ a clear course of actions for sample data acquisition and selection that should lead to acquiring a solution to our problem (Rajasekar, S., P.Philominathan & V.Chinnathambi, 2016). Hence choosing a suitable research method before starting a research is a very critical step. In addition, there are two research approaches: inductive and deductive. By following the inductive research approach, a researcher starts by selecting relevant data to his research, going from data to theory, from a specific view to a more general one. On the other hand, a researcher skilled in the inductive approach begins with a theory and tests that theory, moving from a general point of view to a more specific approach (Blackstone, 2012).

With the nature of the research question, the writer can choose to follow either inductive or deductive approach. Nevertheless, within the limited time as well as knowledge, the writer is not interested in creating a new theory or solution to answer the research question. Instead, the writer will make a research based on already existing studies or solutions then apply it to a real case study in order to answer the research question. Therefore, this thesis will employ deductive as its research approach.

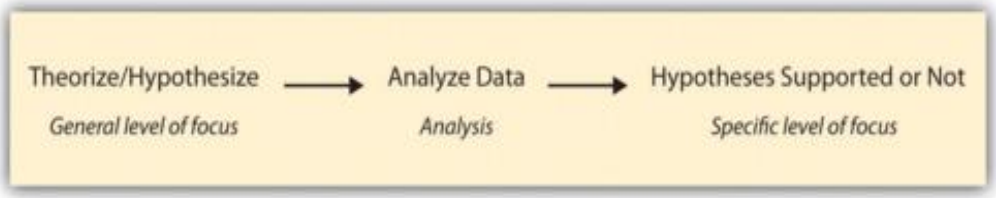


FIGURE 3: Deductive research

Moreover, this thesis is controlled by using design science research method. The thesis writer chooses this method because it allows the writer to test, evaluate and also improve the artefact. According to “A Design Science Research Methodology for Information Systems Research“, the

design science research is defined as followed: Design science… produces and evaluates IT artifacts in order to solve identified organizational problems. The study itself contains the rigorous process to design artifacts to tackle observed problems, to contribute to the research problem, to evaluate the designs, and to explain the outcome to targeted audience (Peffers, K., Tuunanen, T., A.Rothenberger, M. & Chatterjee, S., 2008).

2.6 Research framework

Since this study is the Design Science research, the research framework in this study will follow Design Science research framework from Hevner, March, Park and Ram (Hevner, A. R., March, S. T., Park, J. & Ram, S., 2004). The framework is visualized below.

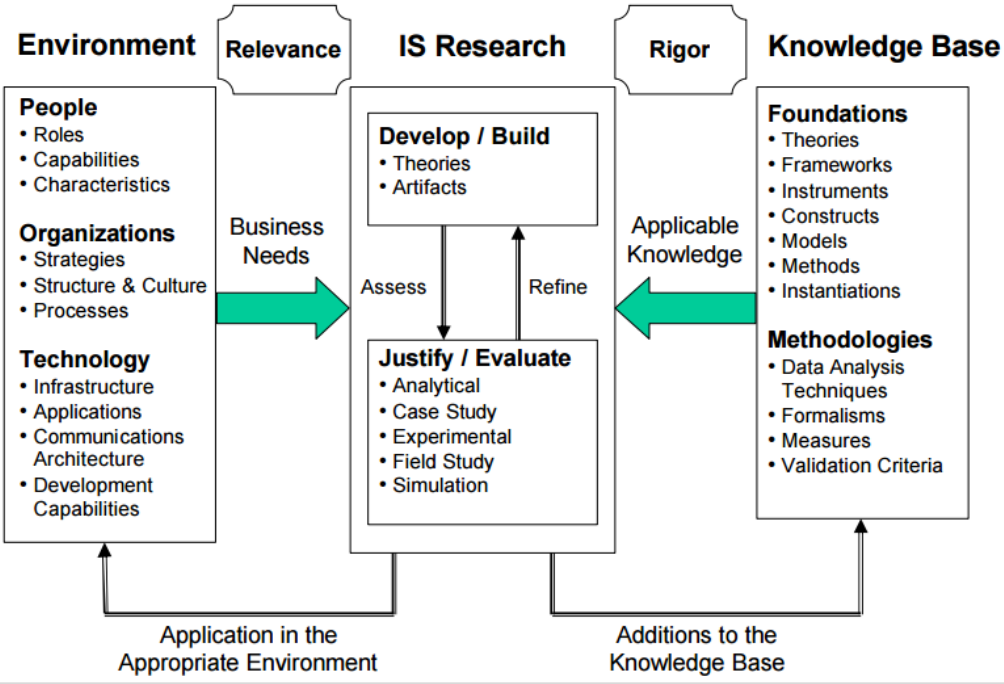


FIGURE 4: Design science research framework (Hevner, et al., 2004)

According to Hevner A.R (Hevner, et al., 2004), in order to conduct the research, there will be a need for two components: Environment and

Knowledge Base. Each of the components will be described in detail in the following parts of this chapter.

2.6.1 Environment

The environment element sets the range of problems where the phenomena of interest reside (Hevner, et al., 2004). In the environment, people, organization and technology are the three aspects that need to be concerned. Hence, the company (1) having an ASP.NET Web Forms web application and (2) aiming to make a migration to ASP.NET MVC as well as (3) having a full-time junior ASP.NET developer whose time allowance is enough to finish the migration would be the best business environment for this study.

2.6.2 Knowledge Base

The Knowledge Base element delivers information about which information needed and which method used in order to accomplish the research. In addition, a typical knowledge base contains: Foundations and Methodologies (Hevner, et al., 2004).

Foundations are theories, abstractions and results from previous studies. On other hand, methodologies provide a guideline on how to justify/evaluate the study including how the data is collected, how to analyze data and which criteria need to be taken into account when performing data analysis.

Consequently, in this thesis, the characteristics and the behaviours of both ASP.NET Web Forms and ASP.NET MVC will be studied as foundations. In addition, depending on the nature of the artefact, the foundations also require information about different components that will be used during the migration progress. Finally, each foundation also includes existing solutions for the research problem.

Regarding the definition of methodologies, the data for this study will be collected from writer's reflection during the time the writer applies the artefact into a case study. Moreover, in order to have an accurate data analysis result, the writer comes up with four different criteria that will be used during the analysis phase including: complexity, time cost, labor cost and performance which will be discussed later in this thesis

2.6.3 Design science in this study

The purpose of this study is researching on the current existing solutions then making an aggregation in order to come up with a detailed and straightforward guideline for the migration from ASP.NET Web Forms to ASP.NET MVC progress. In addition, the new artefact that will be created by the writer will be tested by applying the artefact to a real case study. Consequently, the design science research in this study is a mix of Design as an Artifact and Design Evaluation.

2.7 Data collection and analysis method

2.7.1 Data collection

In this thesis, K-company will be the case company. The company has a promotion-email-sending platform which is built with ASP.NET Web Forms that anh which is in need of moving into the new ASP.NET framework. With the real case study, it is a great chance for the writer to apply and observe the artefact in real life progress.

Furthermore, the data will be collected by using reflection collection methods. The unit of data of this study will be the writer's daily working diary during the time performing the the migration task. For more details about what and how the data is collected, the below table is presented (tasks will be described in detail in chapter 4).

Actors	The thesis writer who acts as an ASP.NET developer
Roles	Making a migration from ASP.NET Web Forms to ASP.NET MVC.
Settings	The environment is in the case study's company office with all the working facilities belonging to the company in order to assure the confidentialilty of the case study product.
Processes	The developer starts implementing the migration step by step based on the artefact description. Working time is 7.5 hours a day which is the same as a normal working day.
Activities	Task 1: Review the migration plan. Task 2: Get familiar with the product structure and database. Task 3: Start implement Models according to the database structure. Task 4: Setting priority for all the .aspx files in the ASP.NET Web Forms project which will be migrated first. Task 5: Create an MVC Controller for the chosen .aspx file. Task 6: Create an API Controller and action methods returning JSON data which will be assigned into server side components in the next step. Task 7: Assign value to server-side components with the data obtained from JSON objects by using JavaScript as well as handle the interaction between users and server side components. Task 8: Create API Controller and action method which handles POST request from Views (.aspx files).

	Task 9: Test from the browser to make sure that the project works fully without any bugs or errors. *Note: repeat from Task 5 to Task 9 until the project is fully migrated to new framework.*

As mentioned in part (2.6.2) above, with the purpose of evaluating the artefact, the writer comes up with four criteria: complexity, time cost, labor cost and performance.

For the first criterium: complexity. There will be three questions involved while doing the data analysis: How hard it is for a junior developer to understand the artefact plan? How deep and wide knowledge the developer needs to have in order to start implementing the migration? How hard each step of the artefact when apply it to real case study is?

Secondly, for the time cost, the two questions below will be concerned: How long it takes to migrate one .aspx file in average? How long in total to migrate the whole working project in case of small and medium project like case study project?

Thirdly, the labor cost for fully transferring the case study will be calculated by using the formula: total amount of hours * average salary of junior developers in Finland * 1.5 for tax and extra costs. The second and third criteria will provide an approximate cost of the transferring progress.

Last but not least, the performance of the system after migration will be considered as well. The aspects regarding to performance will be speed, security and the complexity of maintains progress.

Failing to consider each of the four criteria precisely will result in excessive costs and lengthy delays in workflow.

2.7.2 Analysis method

During the applying artefact progress, the writer will note down carefully any difficulties, extra efforts and working hours in order to conduct accurate conclusion of the artefact.

The data will be analysed after the author has finished the transferring progress. The writer will use code techniques to process the working diary and to point out the difficulties and find the answer for all the considered criteria mentioned in previous chapter. Additionally, few technical tests will also be performed after the transferring progress has been completed in order to evaluate the success level of each performance criterion.

Finally, the writer will provide a conclusion of the artefact regarding four criteria mentioned.

3 RESEARCH FRAMEWORK

In this part, the writer will describe (1) the definition of ASP.NET Web Forms and (2) ASP.NET MVC, (3) ORM (Object Relation Mapping) Database & Entity Frameworks, (4) ASP.NET WEB API Controller and finally (5) previous studies.

3.1 ASP.NET Web Forms

ASP.NET (former name of ASP.NET Web Forms before ASP.NET MVC was released) is a data driven web application framework. ASP.NET comes with huge amounts of server side components. With ASP.NET, an accurate HTML code will be sent to each user's circumstance or request. (Cox, 2008).

For more details, at the point when there is a request from users, the request is compiled and executed on the server by the framework and afterward, the HTML markup structure is created by the framework which browsers can render. (Microsoft, 2016).

In addition, developers can develop ASP.NET Web Forms web applications with Visual basic, C#, Managed C++, J# or JScript.Net programming language.

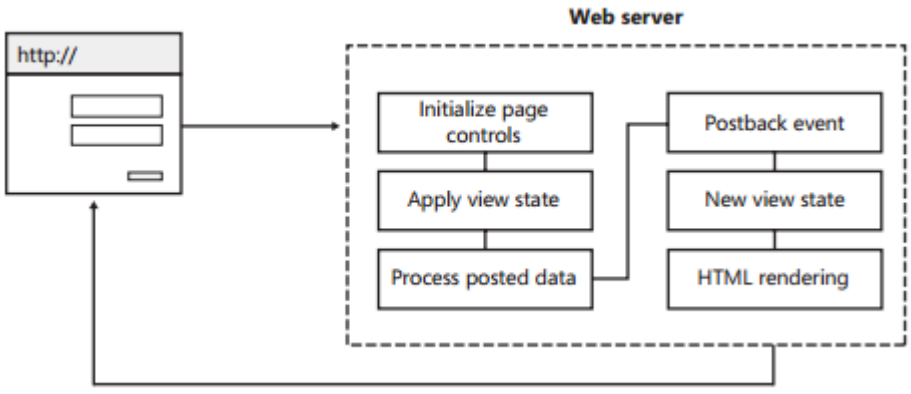


FIGURE 5: Web Forms model in action (Esposito, 2011)

3.2 ASP.NET MVC

ASP.NET MVC is one of the frameworks provided by ASP.NET that allows developers to build web applications in a rapid way. ASP.NET MVC was released in 2008 and has made a great impact on the way developers create ASP.NET web applications.Unlike the ASP.NET Web forms, ASP.NET MVC separates the user interface (HTML, CSS) from the business logic behind by following the MVC (Model – View – Control) design.

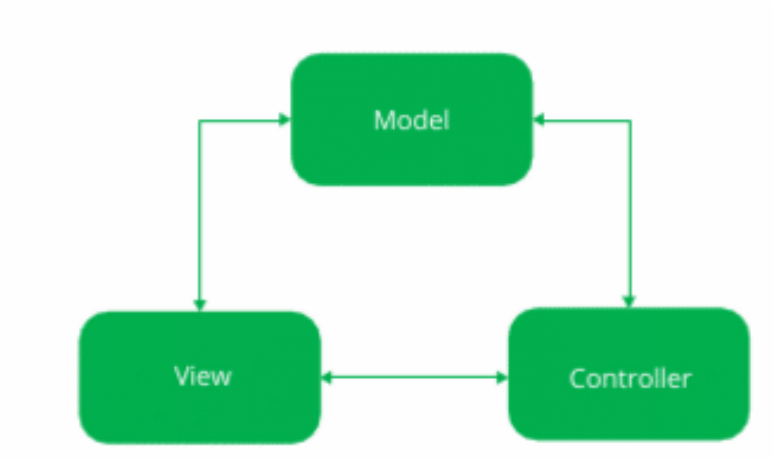


FIGURE 6: MVC pattern (Perkins, 2012)

Model: Model is a class which is used to communicate with the database and inclues various actions, for example: update or retrieve information from the database.

View: View is the web application user interface created from pure HTML with data from Model.

Controller: Controller controls which view will be displayed to the user with the correct Model data. In addition, Controller also handles the interaction between the end user and the web application.

(Perkins, 2012)

3.3 ORM (Object-Relational Mapping) and Entity Framworks

The principle of ORM is to assign to third party libraries or frameworks the task of creating a correspondence between objects and tables. Classes and attributes will be mapped to relational databases with corresponding tables containing rows and columns. With the mapping technique, it gives developers a chance to communicate with the database via objects instead of pure SQL (Goncalves, 2013).

With the development of ORM principle, many third-party libraries or frameworks have been released for different programming platforms. ASP.NET is not an exception, the Entity Framework which is an ORM framework for ASP.NET was first released in 2008.

For more details, according to Rahul Rajat Singh, Entity Framework is an ORM (short for Object Relational Mapper) framework built on top of ADO.NET. Entity Framework and allows developers to write data access codes as models instead of as SQL queries. This characteristic makes producing data access layers much easier and less effort-consuming (Singh, 2015).

3.4 ASP.NET Web API

3.4.1 ASP.NET Web API

ASP.NET Web API is a web framework provided by Microsoft which supports .NET 4 and above. ASP.NET Web API implements the HTTP specification and allows developers to build or use HTTP services and makes it extremely easy to build RESTful services. ASP.NET Web API is inspired by the ASP.NET MVC. Hence, developers who work with ASP.NET MVC will find themselves familiar with ASP.NET Web API. With ASP.NET API, a traditional ASP.NET project will turn into a powerful HTTP API (Ugurlu, Zeitler, Kheyrollahi, 2013).

3.4.2 ASP.NET Web API Controller vs ASP.NET MVC Controller

Since ASP.NET Web API is inspired by ASP.NET MVC, the framework itself contains some components that can be found in ASP.NET MVC, one of them is the Controller. Meanwhile, the artefact which will be evaluated in this study will contain both ASP.NET MVC Controller and ASP.NET Web API Controller, therefore having knowledge about differences between the pair Controllers is necessary in order to perceive the artefact.

By default, The ASP.NET MVC Controller is usually used for reacting to users' inputs or updating models with information from users. In brief, the Controller working with data coming in and transferring data to specific view (Galloway, Haack, Wilson, Allen, 2011). By contrast, ASP.NET Web API action methods will serialize the return values into the JSON with the help from popular Json.Net library (Block, Cibrano, Felix, Dierking, Miller, 2014).

Despite the differences of the default action behaviour, ASP.NET MVC Controllers and ASP.NET Web API Controllers can still generate the same result. However, extra effort is required for different scenarios. The figure below will show the differences of the syntax between the pair Controllers, when the requirement is to return a JSON object when there is an AJAX call hits the action method.

ASP.NET MVC

```
public class TweetsController : Controller {
  // GET: /Tweets/
  [HttpGet]
  public ActionResult Index() {
    return Json(Twitter.GetTweets(), JsonRequestBehavior.AllowGet);
  }
}
```

ASP.NET Web API

```
public class TweetsController : ApiController {
  // GET: /Api/Tweets/
  public List<Tweet> Get() {
    return Twitter.GetTweets();
  }
}
```

FIGURE 7: Differences between the pair Controller syntax

3.5 Solution from previous studies

With the aim of conducting a guideline for migrating from ASP.NET Web Forms to ASP.NET MVC, the writer has done some research on current existing solutions. Unfortunately, there is no official guideline from Microsoft. However, there are still a few solutions which are created by developers and published in the form of blogs. Therefore, the writer picks the two most straightforward solutions into consideration. Each of the solutions has its own advantages and disadvantages.

3.5.1 Integrate MVC into existing Web Forms

In 2013, Rachel Appel provided a course: Migrating ASP.NET Web Forms to ASP.NET MVC (Appel, 2013) in WintellectNow platform. The course contained many useful information including: when the migration from ASP.NET Web Forms to ASP.NET MVC is needed, what can be kept from old ASP.NET Web Forms when doing the migration and introduction in detail about how to migrate from the old frameworks to the new one.

Within the scope of the solution. The writer collects information on how to enable new friendly URL routes (which follow ASP.NET MVC route patterns), how to create Models with Entity frameworks, how to create Controllers and Views which follow the convention of configuration of ASP.NET MVC.

According to Appel's solution, old .aspx files will be reused as a View in the new ASP MVC web applications. This is understandable since reused .aspx files will save a lot of time and effort in re-designing user interfaces. However, this point reveals the flaw of this study. The solution only gives the suggestion to reuse the view, but does not mentione how to get rid of code behind in .aspx files which is a complicated process.

3.5.2 Migrating ASP.NET Web Forms to the MVC Pattern with the ASP.NET Web API

Peter Vogel who is a specialist in ASP.NET development has a blog in Microsoft magazine blog. The blog provides a solution for the migration from ASP.NET Web Forms to ASP.NET MVC which is Migrating ASP.NET Web Forms to the MVC Pattern with the ASP.NET Web API (Vogel, 2013). As can be seen from the name of the solution, if using this instruction, the new system will only follow the MVC pattern but will not work fully as an ASP.NET MVC web application.

From this study, the writer gets an idea how to use ASP.NET Web API as a tool to get rid of the code behind in .aspx files. This solution, if applied successfully, will fix the flaw in the previous solution provided by Appel.

During the time spent on the second solution, the writer has had a chance to learn how to work with ASP.NET Web API, especially in how to create Web API Controllers which return JSON objects or perform business logics depending on each specific case. In addition, the thesis writer has also acquired the knowledge of how to create AJAX requests with JavaScript thanks to this solution.

As mentioned above, this solution only migrates the old ASP.NET Web Forms into MVC Pattern. Hence, the new system is not fully an ASP.NET MVC application especially in URL routing.

4 ARTEFACT DESCRIPTION

According to chapter 3.5, currently, there is no official guideline for making the migration between two frameworks. Therefore, the thesis author has made an aggregation of two existed solution mentioned in chapter 3.5 previously. The new solution will solve the research problem and answer the research question.

The basic concept for the artefact is: **Integrate MVC into existing Web Forms code with the help from Web API**. As the nature of the artefact, the progress starts with the existing Web Forms project, during the project, adding packages or refactor code is required in order to finish the transferring progress. Finally, during this chapter, the artefact will be described in detail.

4.1 Artefact in nutshell

Theoretically, in the ASP.NET Web Forms web application, the HTML and business logic are tightened within one .aspx file. The server side code does not only control the business logic but also govern the HTML code partly via server side components. By contrast, the ASP.NET MVC web application tends to separate the logic and HTML code. Hence, the Views and Controllers are created. Consequently, the main focus point when commencing the migration will be isolating the business logic code and HTML code in the old ASP.NET Web Forms web application into Views and Controllers.

In order to achieve the main focus point, there is a need to get rid of all ASP.NET Web Forms server side components or at least, make them independent from the code behind. Notwithstanding, removing all the server side components will destroy the user interface of the web application and lead to the expense of unnecessary effort for re-writing pure HTML code. Hence, keeping the components and make them work independently from the code behind is an ideal solution. But since the HTML code is created dynamically in the server whenever there is a

request to the route (to .aspx file in Web Forms) and in some cases the value of the component is also assigned by the data that come from the business logic, finding a way to create a components on the server side and assign the data (if needed) without interacting with the code behind inside the .aspx file is required. In this case, Web API Controller and AJAX techniques are the key. In order word, the server side components are only created in .aspx files (server side) but the value of the components to be attached when the HTML is loaded (client side) via AJAX calls.

Let us now turn to more details about the artefact, as the result of moving to MVC design patterns, any request will not hit .aspx files anymore. Instead, the Controller with correct action methods will be triggered. After the Controllers and action methods are called, the view will be rendered, in order to keep the old HTML design, the old .aspx file will be used as a view.

Notwithstanding, the old .aspx needs to be modified by adding AJAX technique which will send the request and receive data from the Web API Controller then assign the value to corresponding HTML components which have already been created on the server side. Still, rendering the view and displaying it on the user's browser are not enough, there is also a need of knowing how to control the behaviour of server side components when there is an interaction between them and users. Since the purpose of the artefact is trying to get rid of code behind in the .aspx files, the server side components no longer post back to the target .aspx file but they will send the request to Web API Controllers as a substitute. Hence, in the Web API Controller, different actions could be performed, for example: business logic that update databases, getting more data or just redirecting to another controller (URL). The figure below will give a visual view of the artefact in brief.

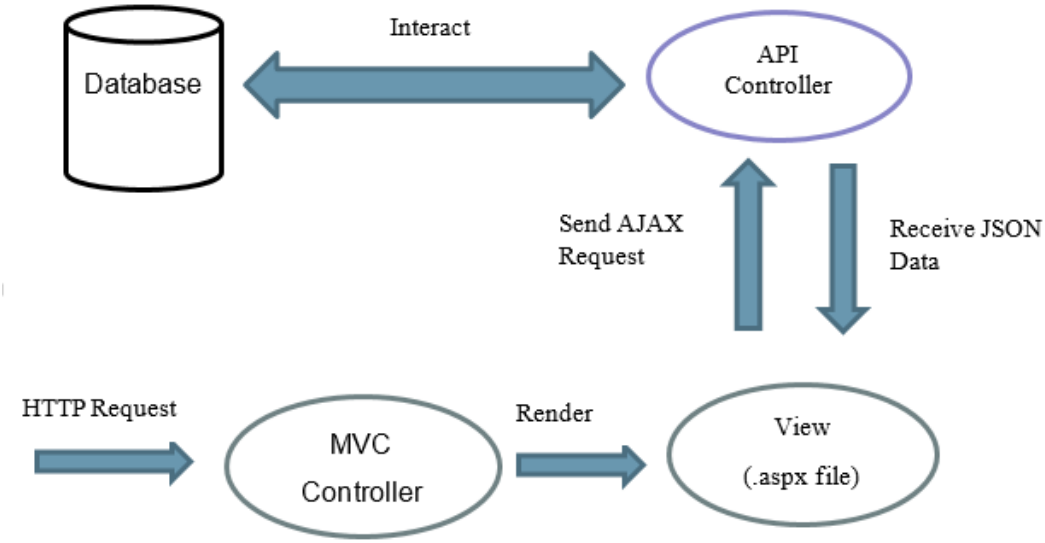


FIGURE 8: Artefact's workflow

4.2 Artefact in details

In this part, the artefact will be described precisely along with examples. In general, the artefact has seven main steps, small unexpected steps might be needed.

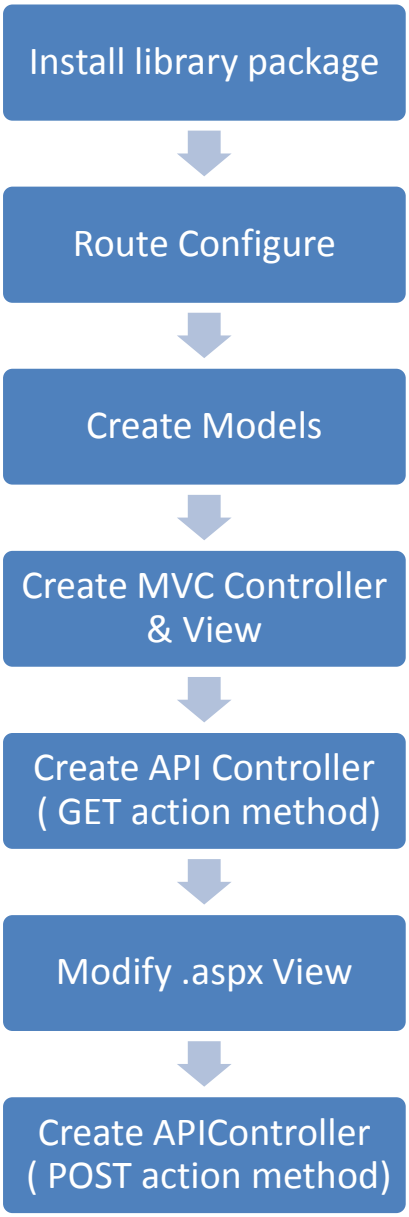


FIGURE 9: Artefact step by step

4.2.1 Install library package.

Even though ASP.NET MVC and ASP.NET Web Forms are both based on ASP.NET library provided by Microsoft. However, the characteristic of the pair is clearly different. This fact leads to the dependence on different third party libraries. In addition, the API Controller which is one segment of the artefact also requires other packages that haven't been referenced yet in already existed ASP.NET Web Forms web application.Therefore, installing

ASP.NET MVC and ASP.NET Web API is the initial step of the migration progress. Installing all the requirement packages and also manage all the configuration are quite a huge amount of workload. Luckily, inside Visual studio – Integrated development environment which is most used to develop ASP.NET web application, all the required packages can be installed only with one click within the NuGet package manager. From Visual Studio, click on Tools, select NuGet package management then search and install ASP.NET MVC and ASP.NET Web API packages

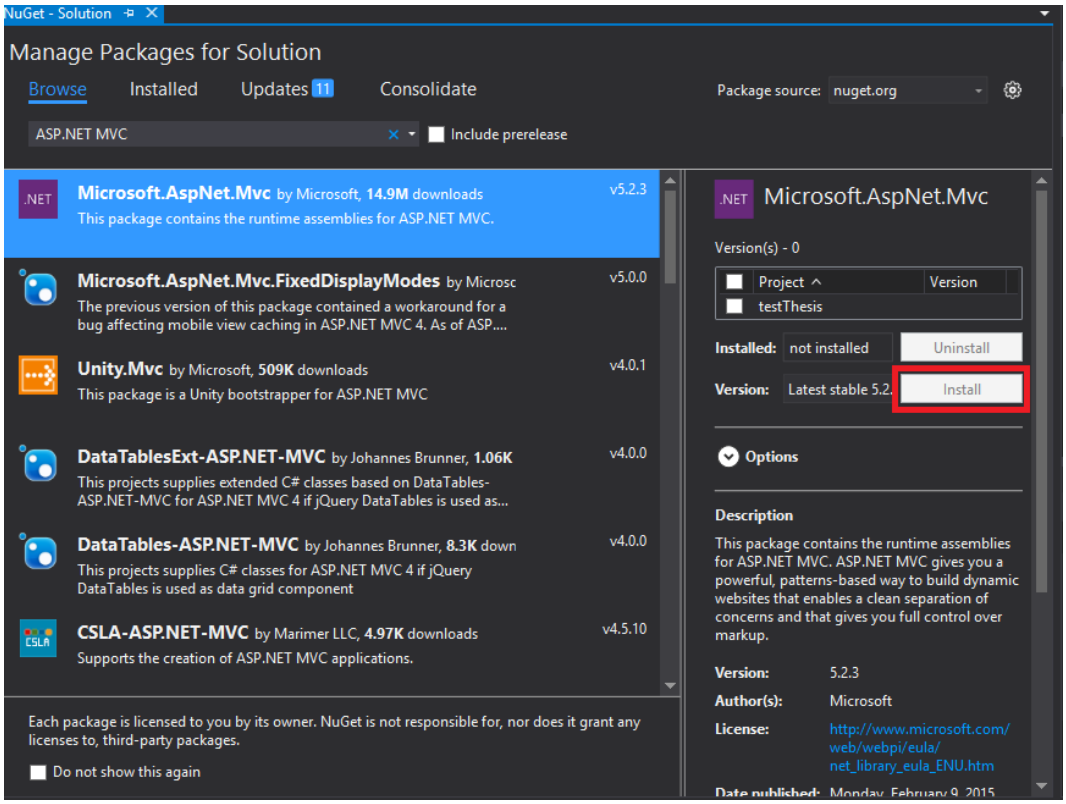


FIGURE 10: Install packages via Nuget inside Microsoft Visual Studio

After doing various steps required by NuGet, new packages are referenced and can be found under the References category of the solution.

4.2.2 Route configuration

Since the routings in ASP.NET Web Forms and ASP.NET MVC are totally different, configuring a new URL routing format is a next step in the

migration progress. Firstly, the developer must create an App_Start folder in the solution, then the RouteConfig.cs file is created inside the App_Start folder. Within the newly created file, adding the noreturn function with the name RegisterRoutes which takes a RouteCollection object as a parameter. The RegisterRoutes function itself will enable the friendly user URL and declare a new routing route for the web application.

```
public static void RegisterRoutes(RouteCollection routes)
{
    var settings = new FriendlyUrlSettings();
    settings.AutoRedirectMode = RedirectMode.Permanent;

    routes.EnableFriendlyUrls(settings);

    RouteTable.Routes.MapHttpRoute(
        "API Default",
        "api/{controller}/{id}",
        new { id = RouteParameter.Optional }
    );

    routes.MapRoute(
      name: "Default",
      url: "{controller}/{action}/{id}",
      defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
    );
```

FIGURE 11: How to enable user-friendly and declare the new rules of routing which include MVC Controller and API controller

4.2.3 Create Models

In ASP.NET Web Forms, usually, the interaction between the web application and database is managed by pure SQL language which might lead to the SQL injection security issues if the SQL parameters are not used. In contrast, in ASP.NET MVC, the communication with the database is handle by Entity framework which an ORM (Object-Relational Mapping). Thus, next step in the process is creates Models and setting up the connection between Entity Frameworks and Database. Before starting to create models, the folder Models is required to be created first. Models are normal C# classes. However, the number of Models classes is equal to the number of tables in the current existing database. Moreover, every property of the Model class represents one column in the database table.

Thus, adequate knowledge about the Database structure and the preciseness in naming class property during creating Model class is highly recommended.

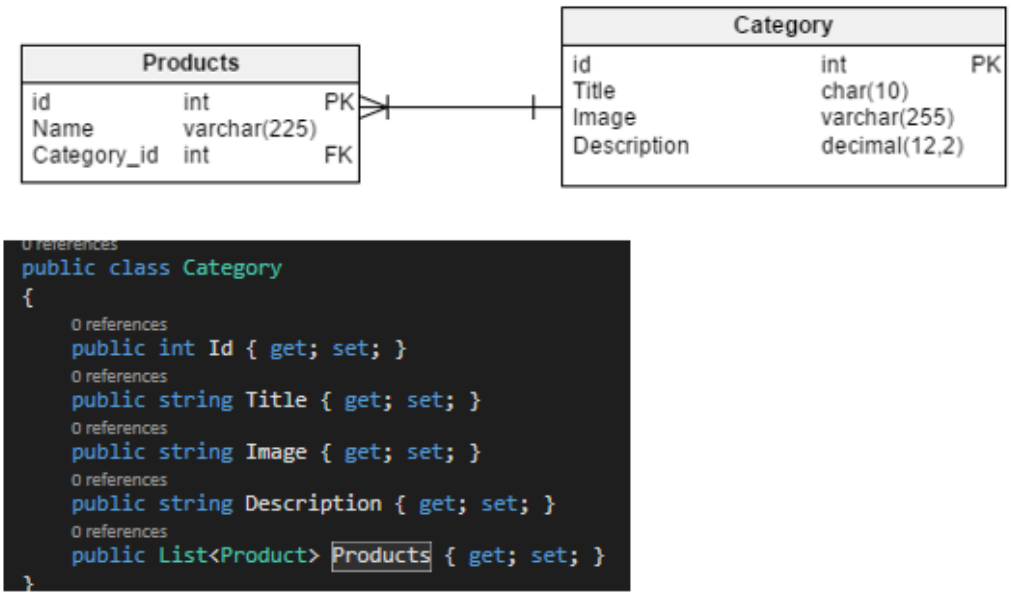


FIGURE 12: Category Model Class

After creating models based on tables in the database, in order for Entity framework to understand which one is C# model class and which one is just a normal one as well as setting up the connection with database, the DbContext class is required. The recently created class must inherit from the DbContext class and also inherit and override the constructor of DbContext class with the name of the connection string which specifies information about the database connection. In Entity framework, the DbSet type property corresponds with database tables, so the name of each DbSet type property needs to be the same with the corresponding table.

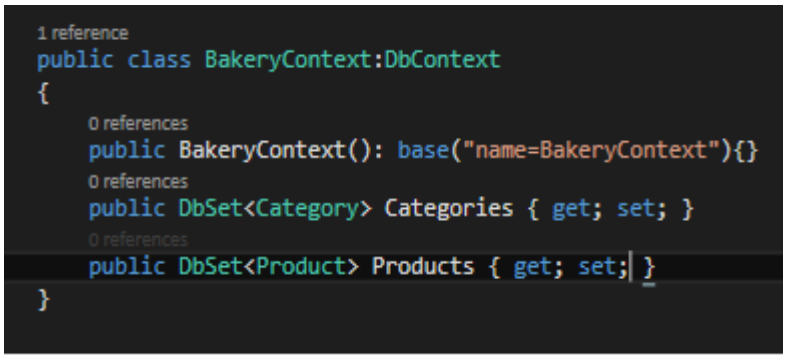


FIGURE 13: Setting up DbContext

Normally, the data in the existing database will be overridden when the web application resets. However, since the progress is migrated from existing working web applications, keeping and working with already existing data is a must. Hence, setting the data initializer to null is needed. The data initializer will be set inside the Global.asax file and inside the Application_Start method.

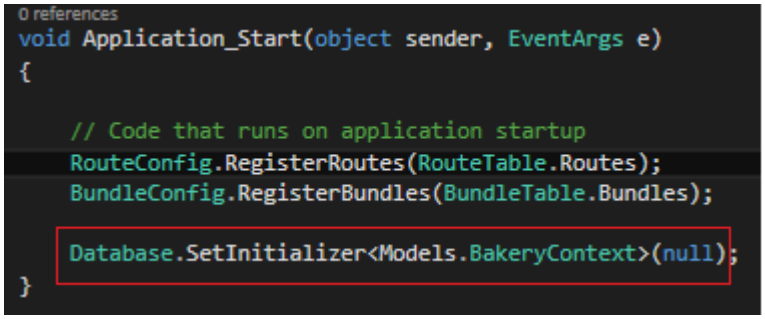


FIGURE 14: Setting initial data

4.2.4 Create MVC Controllers & Views

The MVC Controller in the artefact does not manage the business logic itself (the ASP.NET Web API will handle this) but the Controller still manages which view will be shown when there is a user request. Therefore, next stage of the artefact is creating MVC Controllers.

In order to fulfil the convention of configuration of ASP.NET MVC, the new folder Controllers will need to be created inside the current solution. Controllers will be stored within the newly created folder.

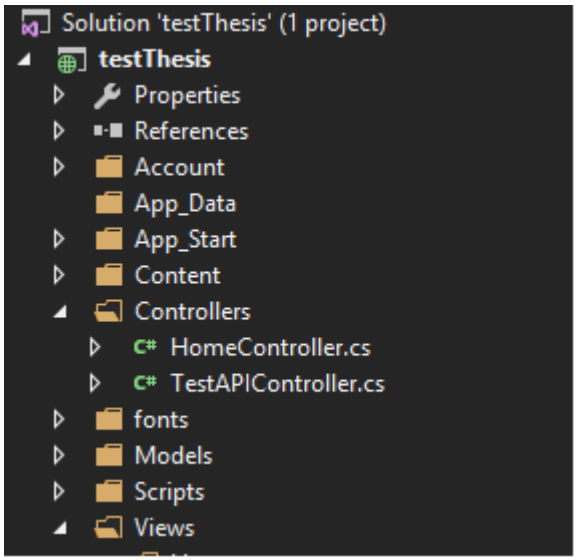


FIGURE 15: Controller folder in the solution

Creating Controllers based on existing .aspx files will require a precise plan, the suggestion here is that related aspx files can be grouped into one Controller, and MVC Controllers only handle GET requests which return the .aspx file as a view to users, any POST request will be handled by Web API Controller.

After creating Controllers, there will the need for creating View files for each of the action methods inside Controllers. Since old .aspx files will be reused as Views for new Controllers. Nevertheless, the convention of configuration still needs to be fulfilled. The process requires that new Views folder be created in the solution. In addition, inside a Views folder, different folders need to be created and named after different Controllers. Views (old .aspx file) belong to specific Controllers will be stored in the corresponding folder.

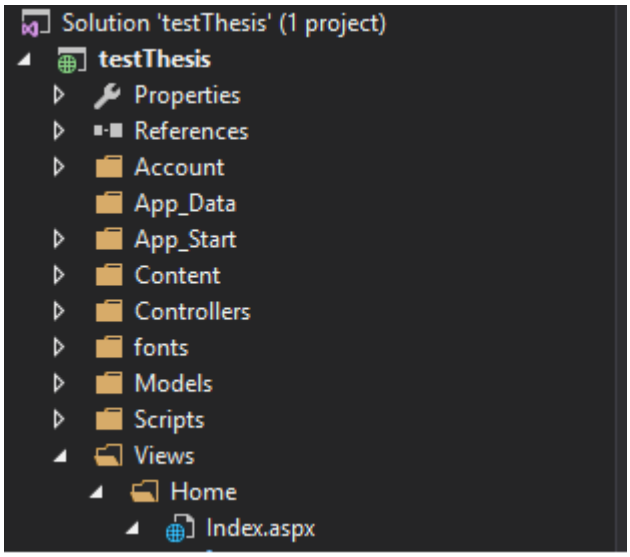


FIGURE 16: Views folder in the solution

4.2.5 Create Web API Controller (GET Action Method)

Web API Controllers play a vital role in the artefact. Within the artefact, ASP.NET MVC Controllers control the routing but ASP.NET Web API Controllers can be considered the brain of the whole system since they provide data for displaying purposes as well as updating the database. Like ASP.NET MVC Controllers, ASP.NET Web API Controllers are also stored inside the Controllers folders. In addition, API Controller must inherited from the ApiController class.

Web API Controller action methods can be divide into two different type: action method handles GET request and action method handles POST request.

For this stage of the artefact, action methods which handle GET request will be created. The Action Method which handles GET request will return data after performing different business logics based on user request. In some cases, the GET Actions Method might contain an optional parameter (mostly an Id)

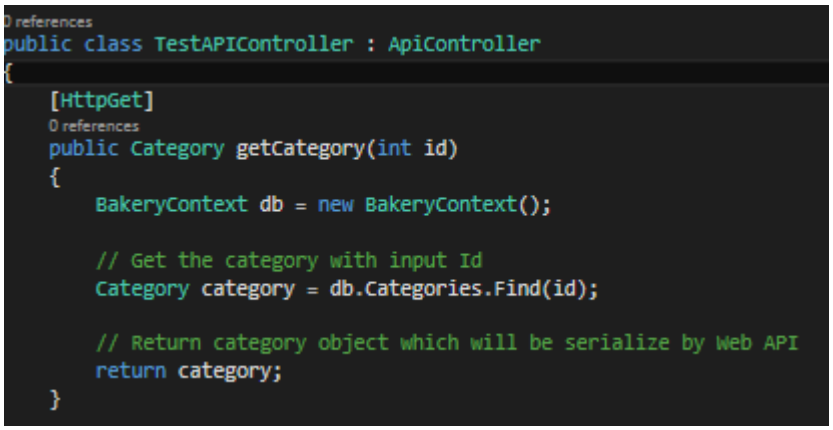


FIGURE 17: Web API Controller with Action Method return a specific category with a Category Id as a parameter

4.2.6 Modify .aspx View

During this stage, an old .aspx file will be modified by adding an AJAX technique with JavaScript codes. Views (.aspx files) are stored in corresponding subfolders inside the Views folders created in previous stage.

In .aspx files AJAX technique will be implemented for two purposes: assigning data to server side components and handling the interaction between user and server side components. The syntaxes for both of the purposes are quite similar. Yet, there are still some slightly differences.

For assigning data purpose, the AJAX with GET request will be implemented. A GET request will be sent to specific Web API Controllers and Action Methods. The request, if success, will return a piece of data in JSON format. Finally, with the returned data, extra JavaScript is required to assign data to components dynamically. Usually, the GET request will be sent when the page is loaded.

```
<body>
    <form id="form1">
    <div>
          <p>Here the category details</p>

          <asp:Label ID="Name" runat="server" ></asp:Label>
          <asp:Label ID="Id" runat="server" ></asp:Label>
    </div>
    </form>
</body>

</html>


<script>
    $(document).ready(function () {
        $.ajax({
            url: "/api/TestAPI/getCategoryDetails", success:function(data, status) {
                $("#Name").text(data.name);
                $("#Id").text(data.Id);
            }
        });
    })
</script>
```

FIGURE 18: AJAX request which get the data from Web API and assign the value into server side components

For handling the interaction between users and the server side components, the AJAX with POST request will be implemented. Before implement an AJAX request, the runat=”server” needs to be get rid of in the <form> tag, this action will make sure that the form will never post back the code behind in the .aspx file. Also, various JavaScript events will need to be added to the server side components, for example: onClick or onChange. After the mentioned modify, when there is an interaction between the user and the server side components, the components themselves will trigger the corresponding JavaScript function, instead of posting back to the code behind as does the original behaviour.

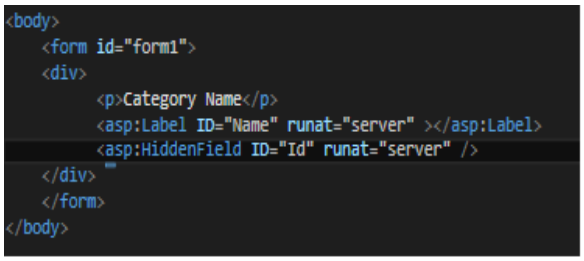
```
<body>
    <form id="form1">
    <div>
                    <p>Category Name</p>

                    <asp:TextBox ID="name" runat="server"></asp:TextBox>

                   <asp:Button ID="Update" runat="server" Text="Update" OnClick="updateCategory()" />
    </div>
    </form>
</body>

</html>


<script>

    $(document).ready(function () {
        $.post("/api/TestAPI/updateCategory", $('#form1').serialize())
        .success(function () {
            /// Do something if the request success
        });
    })
</script>
```

FIGURE 19: AJAX POST request in order to update the category

4.2.7 Create Web API Controller (POST Action Method)

As mentioned above, beside the GET action method, the POST action method also needs to be created in order to complete the transferring progress. The POST action method will be triggered when there is an interaction between the user and the server side components in the old .aspx files.

For POST request, the progress is little bit more problematic since the action method which handles POST request will need to accept a JSON object as its parameter. Fortunately, the Web API coding convention will do most of the heavy lift, what left to be done is only creating Data Transfer Object (DTO) classes to hold the value from .aspx files. The DTO class itself will contain properties with names which match the names associated with the server side components that hold the data needed to be posted to the Web API Controller.



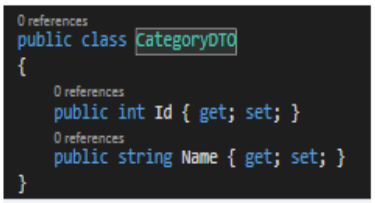


FIGURE 20: Category DTO class associate with .aspx server side components

After defined DTO class to handle parameter, the business logic will be implemented inside action methods. The Action method will perform the action to update database as well as some extra actions after the database is updated successfully, for example: redirecting to specific ASP.NET MVC Controller or returning the HttpResponse.

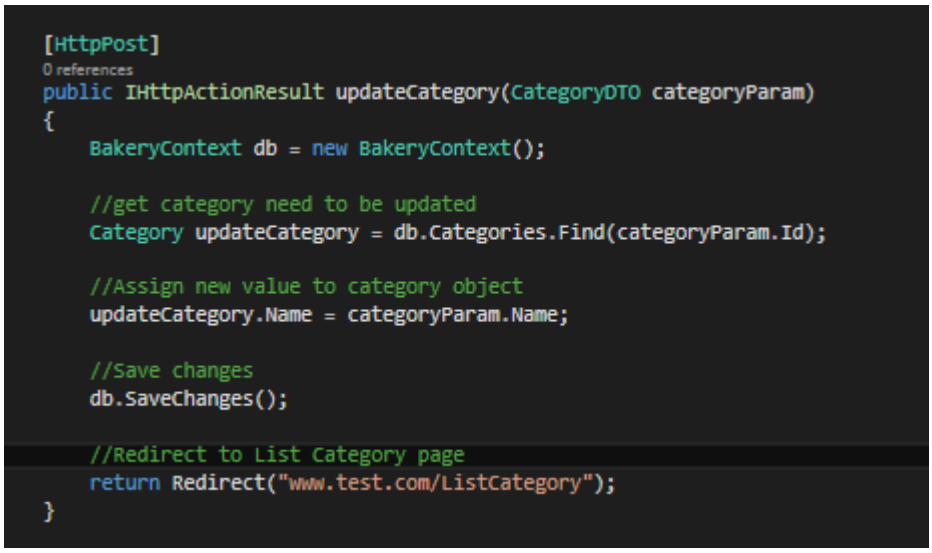


FIGURE 21: updateCategory action method which update database and redirect to ListCategory page

5 THE STUDY

5.1 Case study

As an e-commerce company, K-company embraces marketing online in general and marketing via email in detail as a vital part in its business model. Therefore, developers in K-Company have come up with the idea of the 24Email application which allows the sale department to send promotion emails in the most efficient way and to the accurate customer targets.

Email24 was created in 2011 with ASP.NET Web Forms. The web application itself could be considered a clone of Mailchimp when it was created.

The main functions of the web application include: sending emails with pre-defined templates, sending text emails or HTML format emails, sending email within a group of receivers and also blacklisting receivers whom the emails are not meant for.

After doing some investigations on the project, the thesis writer can give a brief summary of the web application structure.There are 22 .aspx files, the average number of lines of code in the code behind of the each .aspx file is fewer than 100 lines. For the database of the system, there are 18 tables created with MySQL. Also, the system includes a class library for coordination between ASP.NET Web Forms and the database. The application could be considered a traditional web application with three layers: presentation (ASP.NET Web Forms), business logic (class library), and data access (same class library with business logic; MySQL)

In summary, Email24 can be considered a medium scale web application with medium-level business logic behind.

5.2 Data analysis

5.2.1 Complexity

Before starting to implement the migration, the writer has been working in the programming field for two years and can be considered a junior developer. In addition, the writer also has basic knowledge of both ASP.NET Web Forms and ASP.NET MVC. However, the knowledge about ASP.NET Web API is missing from the writer's skill set. Consequently, the laid back costs the writer about two hours to learn a new knowledge about ASP.NET Web API and understand the artefact.

The artefact itself is straightforward and easy to understand. Nevertheless, the writer faced the first problem in the fourth step when creating MVC Controllers and Views. In order to finish this step, the writer needs to spend extra time to get used to the flow and structure of the web application in order to create Controllers in a most immaculate and efficient way.

Another bottleneck that prevents the writer from finishing the migration occurred in the sixth stage which involves modifing the Views (.aspx files). The HTML server side components which is created by .aspx files has an unpredicted HTML structure (especially the GridView component) which leads to unexpected extra effort and time cost to investigate on the HTML web page structure created by the server side.

Beside the mentioned problems, the rest of the artefact, from the junior developer's perspective is easy to follow and understand.

5.2.2 Time cost

The writer started implementing the migration on Wednesday 23th of March 2016 and partly finished the migration on Friday 1 of April. The migration progress costs the thesis writer eight working days which means sixty working hours. Additionally, it takes the thesis writer from one hour up

to four hours to migrate one .aspx file, depending on the size and the complexity of the file. In different circumstances, the time cost may vary.

5.2.3 Labor cost

According to the Collective Agreement – IT Service Sector (1 November 2013 to 31 October 2016) created by the federation of Finnish technology industries, federation of professional and managerial staff –YTN and association of IT sector employee, the minimum salary for level 1(junior) design/development job is 2166 euro/month (Collective Agreement – IT Service Sector, 2016) which means around 14.44 euro/ working hour.

With the the formula:

> *Labor cost = salary per hour * total working hour * 1.5 (for tax and extra costs ...)*

The labor cost for one junior developer to finish the migration for medium size project is around 1300 euro.

5.2.4 Performance

Despite the extra work, the result so far from the migration at the end is from twelve.aspx files, the new ASP.NET MVC application contains only four Controllers which is an idea project structure for future maintenance.

On the other hand, the writer also implementes the speed test between the old ASP.NET Web Forms web application and the new ASP.NET MVC web application. The test is operated by using the service from webpagetest.org with the server located in Stockholm, Sweden with Google Chrome as a browser.

The result from the speed test shows that, with the same page, it takes 1.7 seconds for the ASP.NET Web Forms web application and 2.2 seconds for the new ASP.NET MVC to load the web page. Hence the new web application is 0.5 seconds slower compare to the old one. However, the

time gap is acceptable and understandable since the new ASP.NET MVC web application needs to send an extra request Web API Controllers in order to display the web page on the browser. In addition, the thesis writer also conducts test on different browsers. The result is optimistic: the web application is combatible with most common browsers such as Firefox, Google Chrome and more.

6 CONCLUSION

The main purpose of this study is to create a detailed guideline for the migration progress from already existed ASP.NET Web Forms web applications to new ASP.NET MVC web applications. Therefore, a new artefact has been created by combining two already existed solutions with some modification.

The artefact contains seven steps in general (some steps need to repeat until the project is fully migrated). Each of the step in the artefact is described in detail with example in chapter four. In addition, knowledges required to breakthrough the artefact are presented in chapter three of the thesis.

Moreover, the artefact created in this thesis is also evaluated by the thesis writer by applying it into a real case study. With the result from the data collection and data analysis progress, the writer also provides some information for readers to consider before deciding to follow the artefact or not. The information can be found in chapter five.

In summary, with the artefact “Integrate MVC into existing Web Forms code with the help from Web API”, this study has provided a new solution to answer the research question of the thesis. From the thesis writer’s perspective, with medium-level of complexity, acceptable time cost, labor cost and performance, it highly advisable to apply the migration to small and medium size projects. Notwithstanding, with some bottlenecks requiring extra effort, it could be too complicated, time-and-labor-consuming to perform the artefact in projects of bigger scales. Hence, for future studies, the writer will suggest another solution that might fit better for large scale projects.

7 DISCUSSION

7.1 Limitations

Firstly, since the artefact is combined by a junior level developer, there could be some high level aspects in the migration progress that are ignored by accident due to the limitation of skill and knowledge.

Secondly, the artefact is only evaluated with the medium scale project and mid-level business logic. Therefore, there is no guarantee the artefact is the most efficient solution for big scale projects.

Finally, with the rocket development of technology, new versions of frameworks are released frequently. Hence, there is a high potential that the artefact will not be compatible with later versions of ASP.NET or will require some extra steps not covered in this study.

7.2 Reliability and validity

Any study about web technologies can be outdated easily due to the fact that web technologies themselves change rapidly. New ASP.NET version vNext is currently being developed and will be released soon in 2016, therefore the material and information of this study will require a frequent update in order to maintain the reliability and validity of the study.

7.3 Future study

7.3.1 Secure Web API

According to the artefact, a new component which is ASP.NET Web API will appear in the project after performing the migration. By default, anyone who knows the endpoint (URL) of methods in the API Controller can send a request and get the data. Consequently, there is a need of securing the Web API in order to prevent data leaking. The suggestion from the thesis

writer is implementing the authorization mechanism for the Web API and data encrypted when responding to requests.

7.3.2 Implement test driven development (TDD)

Nowaday, with the popular of continuous software development, testing in general and unit testing in detail is a must in every IT project. With the new MVC pattern, it is now easier than ever to implement TDD in the ASP.NET web application. Therefore, there is a need to implement unit testing for all the functions in the project.

7.3.3 Running ASP.NET MVC and ASP.NET Web Forms at the same time

As mentioned above, there are still some bottlenecks in the artefact that could make it impossible to apply the artefact into big scale projects. Hence, another solution that could be taken into account is allowing the old code (build with ASP.NET Web Forms) to run at the same time with the new code (build with ASP.NET MVC). This solution can save time and labor cost since there is no need for a transferring progress. However, it requires a research on how to let two frameworks run at the same time, especially in the route configuration.

8 REFERENCES

Appel, R., 2013. *Migrating ASP.NET Web Forms to MVC (course).* [Online]
Available at: https://www.wintellectnow.com/Videos/Watch?videoId=migrating-aspdotnet-web-forms-to-mvc
[referred on 1 April 2016].

Blackstone, A., 2012. *Sociological Inquiry Principles: Qualitative and Quantitative Methods.* Washington: Flat World Education, Inc.

Block, G. ym., 2014. *Designing Evolvable Web APIs with ASP.NET.* s.l.:O'Reilly Media.

Cox, K., 2008. *ASP.NET 3.5 For Dummies.* Hoboken: For Dummies.

Esposito, D., 2011. *Programming Microsoft ASP.NET 4.* s.l.:Microsoft Press.

Esposito, L., 2016. *What ASP.NET 5 Means to a Technical Manager.* [Online]
Available at: http://www.codemag.com/article/1501071
[referred on 15 February 2016].

Federation of Finnish Technology Industries, Federation of Professional and Managerial Staff –YTN , Association of IT Sector Employees, 2016. *TES_2013_en_web-version.pdf.* [Online]
Available at: http://tietoala.fi/rauta/wp-content/uploads/2015/11/TES_2013_en_web-version.pdf

Galloway, J., Haack, P., Wilson, B. & Allen, K., 2011. *Professional ASP.NET MVC 3.* s.l.:Wrox.

Goncalves, A., 2013. *Beginning Java EE 7.* New York : Apress.

Hevner, A. R., March, S. T., Park, J. & Ram, S., 2004. *Design Science in Information Systems Research.* s.l.:s.n.

Jacob, W., 1991. System Migration. *Information Today*, July, 8(7), p. 37.

Kothari, C. R., 1990. *Research Methodology: Methods and Techniques (Second Edition).* 2nd toim. New Delhi: NEW AGE INTERNATIONAL (P) LIMITED, PUBLISHERS.

Liberty, J. & Hurwitz, D., 2003. Programming ASP.NET. Teoksessa: *Programming ASP.NET.* Sebastopol: O'Reilly, p. 3.

Microsoft, 2016. *Introduction to ASP.NET Web Forms.* [Online]
Available at: http://www.asp.net/web-forms/what-is-web-forms
[referred on 1 March 2016].

Peffers, K., Tuunanen, T., A.Rothenberger, M. & Chatterjee, S., 2008. A Design Science Research Methodology for Information Systems Research. Teoksessa: *Journal of Management Information Systems .* Abingdon: M.E. Sharpe, Inc..

Perkins, B., 2012. *Windows Azure and ASP.NET MVC Migration.* s.l.:John Wiley & Sons.

Rajasekar, S., P.Philominathan & V.Chinnathambi, 2013. *Research Methodology.* [Online]
Available at: http://arxiv.org/pdf/physics/0601009.pdf
[referred on 20 February 2016].

Robson, C., 2002. *Real World Research.* 2nd toim. United Kingdom: Blackwell Publising.

Singh, R. R., 2015. *Mastering Entity Framework.* s.l.:Packt Publishing Ltd.

Ugurlu, T., Zeitler, A. & Kheyrollahi, A., 2013. *Pro ASP.NET Web API.* New York: Apress.

Walther, S., 2015. *Top 10 Changes in ASP.NET 5 and MVC 6.* [Online]
Available at: http://stephenwalther.com/archive/2015/02/24/top-10-changes-in-asp-net-5-and-mvc-6
[referred on 15 February 2016].

Vogel, P., 2013. *ASP.NET - Migrating ASP.NET Web Forms to the MVC Pattern with the ASP.NET Web API.* [Online]
Available at: https://msdn.microsoft.com/en-us/magazine/jj991978.aspx
[referred on 1 April 2016].