A MONITORING FRAMEWORK FOR SOFTWARE ARCHITECTURE DEGRADATION IN DEVOPS PRACTICE

Gonagala Withanage Piyumi Sampath Gunarathna

158216D

Degree of Master of Science

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

March 2019

Declaration

I declare that this is my own work and this thesis does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: Date:

The above candidate has carried out research for the Masters thesis under my supervision.

Name of the supervisor: Dr. Indika Perera

Signature of the supervisor: Date:

Acknowledgement

I am deeply grateful to my supervisor Dr. Indika Perera from the University of Moratuwa Department of Computer Science and Engineering whose help, motivation, suggestions and encouragement helped me in all the time to complete this work. Dr. Indika Perera inspired me greatly to work in this project.

I would like to thank my family and friends for their understanding during the period of the project and gave me support. Without their help I would face many difficulties in the time of the project. And finally my grateful thanks to all the people who help me during the project.

Abstract

This is a research report that desired to carry out to find the software architecture issues and possible monitoring techniques to overcome those issues in DevOps practice**.** DevOps is a new philosophy that helps software organizations to innovate faster and to be more responsive to business needs, it promotes collaboration between developers and operations which improves quality of software development and more frequent software releases.

Continues delivery in shorter development iterations and deploy a software system faster are the key practice of DevOps. And also because of that faster practice, there is huge risk of occurring a software failure unless the continuous monitoring. Therefore the DevOps teams use automated tools for monitoring functional criteria and the performance. But still, most of the teams are not monitoring the architecture of the application.

Unless there need to be done a change intently, the architecture of the system need to pertain its initial designed and finalized architecture in DevOps culture, in shorter development iterations. Therefore it is more important to monitor the software architecture without leading to a software drift or erosion. Therefore this research objective is to build a Monitoring framework for architectural degradation in DevOps practice.

The end goal is Continuous Testing and Continuous Monitoring. Testing and Monitoring are what will prove that the new built is the right required application, that functions and performs as designed and desired.

As the main research objective it identified a missing area of software architecture monitoring methodologies and analyzed and identified a way to prevent software architecture erosion using that. This research is more focused on unconventional usability of the solution and project file contents and how it can be leveraged to capture the architecture of the application and how it can be used as an effective architecture design monitoring framework.

This research states a methodology which uses project file and solution file content to detect the architecture specific information from the code base and a mechanism to capture them and compare them with a pre-defined architecture rule set. An empirical and theoretical evaluation has been done to prove this concept actually works in real life scenarios. It opened up a new area of architecture conformance checking to the future researchers of the field of software architecture.

Keywords

DevOps, Continuous integration, Continuous deployment, Defect warnings, Continuous monitoring, Software development lifecycle, Quality assurance, SDM

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

This chapter offers an outline to the main topics and the motivation behind the project a brief overview of the problem, project objectives, expected outcomes and the outline of the overall structure of the report are also provided.

1.1 Background

Today's business challenges have been pushed traditional delivery approaches to new levels. Therefore, “the benefits of a DevOps approach far outweigh any potential difficulties in aligning the two transparency-limited silos. It delivers systems to the business faster and reduces risk of production changes through automated non-functional testing and shorter development iterations”. Unfortunately, in the traditional approach there has been little working partnership between the development and operations silos. Development and operations teams might work in different buildings, or even continents.

DevOps focus to break down the difficulties and conflicting issues that usually exist between development and operations teams, such as functional requirements, application performance and project spend. Devops concept leads development and operations teams to work together, delivering reliable and safe systems into production very rapidly, and to operate and maintain them more efficiently and effectively Development and operations team work better together, thought more alike, broke down silos, and shared responsibilities.

“DevOps integrates developers and operations teams in order to improve collaboration and productivity by automating infrastructure, automating workflows and continuously measuring application performance”. Devops team tries to automate everything. Automate code testing, automate workflows, automate infrastructure. They would write software as small chunks that are integrated tested monitored and deployed usually in hour’s verses the traditional way of writing large chunk of software over weeks or months and do weeks or months of testing. And also the development and the production environment are identical based on the configuration. Write small chunk of code will allow them to increase frequency of deployment and improve the time to deploy new code. It also enables them to adopt the iterative process to monitor measure and improve the code and operations every day. Improve the ability to response market needs or other things that impact software. They automate software instead of manually building, configuring software and infrastructure. Consequently developers have the ability to build infrastructure at scale to large number of servers in multiple locations using various types of hardware. Further alteration that DevOps oriented team does is using source control system so as to support for managing tracking and documenting all of the changes both the configuration management code and the application code. The changes that implement is to adopt a discipline of application performance m0nitoring and optimization and almost real-time. This will allow developers to understand the impact for the performance due to their changes. The vital goal is to have a production environment that gives their customers a great user experience.

DevOps oriented teams give a lot of benefits towards a company. It potentially allows company to grow on the rate of software delivery and improves the time to market from months and weeks to days and hours. Certainly this would be a massive competitive advance. And also, by automating their infrastructure, this allows companies to maintain improved places. Therefore they can more focus on things such as improving the business and the online contact. They were usually spending more time on these activities to improve the organizational values. When company is able to build and offer better products that means they have happier customers and happier developers. Because of these reasons most of competitive companies have a tendency towards this new philosophy devops, which is changing in mindset of the two groups that need to work closer together and getting write automated tools which allows them to build and test code continuously.

1.2 Importance and novelty of the problem

DevOps is basically focused on following four basic processes:

• Continuous Integration

• Continuous Delivery

• Continuous Testing

• Continuous Monitoring

The end goal is Continuous Testing and Continuous Monitoring. Testing and Monitoring are what will prove that the new built is the accurate required application, which functions and performs as desired.

“The DevOps team ensures that the system is performing as desired way via continuous monitoring”. In this practice they not only monitor the environments and systems but also the application. They ensure that the applications are performing at optimal levels. This requires that DevOps teams use tools that can monitor application performance and issues.

Since there are continues delivery in shorter development iterations, the application is being developed faster. And also there is huge risk of occurring a software failure unless the continuous monitoring. Therefore the DevOps teams use automated tools for monitoring functional criteria and the performance. But still most of the teams are not monitoring the architecture of the application. Since there are shorter development iterations, it is very essential to monitor the software architecture without leading to a software drift or erosion.

Software architecture erosion or a drift can be occurred due to the new or changing requirements and technologies, different architectural knowledge of developers or decision making and project management problems

Software Drift

Drift violation arises when the latest development of software goes away from the pre-defined conceptual architecture. In actual fact, the implemented software is unable to satisfy all the client requirements but the most of it.

Software Erosion

The most severe violation which could occurs in the implemented software which does not meet any client's requirements. Simply the system cannot support the desired new changes due to the code structure. Software drift also could be lead to software erosion in future.

There should some sort of communication between the developers and system designers for preventing software architectural drift or erosion. That can be done by developers or implementers such that where designers follow the implementation progress of the software or in the other hand where the implementers are documenting their work carefully. But, since these implementation iterations are shorter iterations, developers and designers cannot spend more effort on mentioned tasks. Therefore DevOps always try to automate almost everything, this verification scenario also should be automated to place in DevOps practice.

1.3 Research Problem

Monitoring is much more significant factor in architecture and design, in order to meet operational requirements. There are tools that analysis the code architecture. But still there is no proper automated continuous software architecture monitoring in industry. New tools are needed to this fast-paced world. The essential monitoring tool should compare and analysis the current architecture with the pre-designed architecture and should identify and warn about the architectural changes which comes up continuously.

1.4 Objectives

By considering the background of this problem and challengers my expectation is to formulate several methodologies and approaches in this report to address some of those concerns.

The main objective of this research is to monitor the architectural degradation in DevOps practice by detecting software drift and erosion so as to keep the software system in right track. To achieve the mentioned objectives, initially desired formula is to build a monitoring framework which continuously analyzes the changing architecture of the software system. The delivery process can be improved by making this new framework part of it.

- This framework will give the analysis reports of the architecture.

- There will be architectural design analysis with each release.
- Whenever there is a new development release, this framework would identify the architectural changes and would warn about them.
- This will keep the architectural changes and implementing code architectural history as in source control history.
- Where there is an architectural drift or erosion, the code deployment out to production can be restricted until that fixed.
- This would improve the way how the development and the production handoff happen.

By helping to identify and to prevent architectural degradation issues, this framework would enhance the benefits of the DevOps practice, the product and software company growth.

1.5 Overview of the Document

This document consists of six chapters. The first chapter comprised the introduction to the research by presenting a background to the underline domain of software architecture and software architecture degradation. It will present a brief overview to the problem which the research trying to find a solution. The motivations to solve the stated problem and the objectives and milestones set to achieve that objectives are also listed in the introduction chapter.

The second chapter includes the findings of the related literature. It includes descriptions regarding the software architecture degradation and the impact to the industry level software. The causes of erosion and the available solutions to prevent it also focused on this chapter. Findings of this chapter helped this research to identify the possible methodologies and areas that are not yet considered to solve the problem of architecture erosion.

The third chapter discuss about the recognized methodology of solving the problem of software architecture degradation. It will include all of the possible and identified architecture detecting and generating mechanisms which will be the scope for this research. The concepts also talks about the mechanism to develop a proof of concept to the desired solution.

Fourth chapter included of the information regarding the solution architecture and the implementation of the proof of concept. This chapter also contain in depth details about finding a solution for the problem of software architecture degradation using software solution file and project files. At the end of this chapter there is a section that describes implementation of the prototype for the monitoring tool Solution Design Monitor (SDM).

Fifth chapter has the information concerning to the evaluation of the research methodology that consists of automated and manual evaluation along with the performance testing.

Sixth chapter gives the conclusion of the research by stating all the contributions along with the limitations of the research scope.

CHAPTER 2

LITERATURE REVIEW

There are similar researches that conducted on the topic of architectural erosion and various methods for dealing with those issues. In this section, existing researches and techniques for controlling architecture erosion and drift are discussed.

Pollet, D. [1] present a “taxonomy that includes goals, processes, inputs, techniques and outputs for reconstructing eroded architectures”. In this paper authors have presented a state of the art in software architecture reconstruction and restoration approaches.

“The reflexion models technique [2] compares an extracted model of the implemented architecture and a hypothetical model of the intended architecture using an intermediary mapping defined by a human evaluator. The hypothetical architecture, in the absence of documented architecture specifications, is usually modeled by observing external system behavior. The computed outcome is the reflexion model that identifies places in the implemented architecture where there are deviations or omissions from the hypothetical design. The comparison process is mostly a human task although tool support is available for refining mappings and visualizing reflexion models. This technique depends on other tools such as call graph analyzers or dependency checkers to build a model representative of the implemented architecture.”

Postma [3] also presents a method same as reflexion models. But authors has been used “relationship constraints among architecturally significant modules to verify conformance between implementation and its intended architecture”. They have implemented the architecture derived from source code. “But the mappings and constraints are automatically generated by tools using formal models of the intended architecture and input from architects”.

Likewise there are set of researches that done to reduce erosion and drift issues. And also there are a few researches that discuss of error warning techniques.

2.1 DevOps Practice.

DevOps has done a major change in to continuous system development. The gap between developers, operations and the end user has reduced allowing for earlier problem detection by DevOps. In early days, each sprint item of software were just specifications, those weren't validated with the end user until it goes to the production. With DevOps, The continuous development and frequent releases of the software to the end user has converted to an easy task. DevOps allows developers and operations to work together more efficiently and effectively.

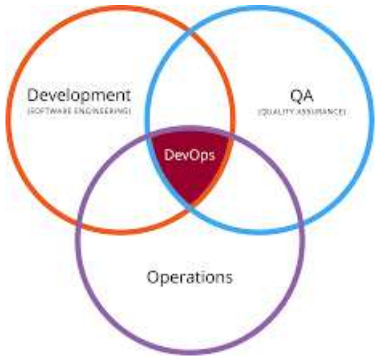


Figure 2.1.1: DevOps

According to the most of the references there is no DevOps process or methodology. "DevOps is a conceptual framework [4], comparable with Agile Software Development. Organizations need to incorporate DevOps principles and practices in their processes."
To get DevOps benefits, software organizations should restructure to incorporate DevOps principles and practices in their software company processes. Automation is the main thing that should be done to incorporate DevOps in a software company. There are two important conclusions related to automation. "First, People should be hired with the right knowledge of automation is important to support DevOps. [4] And the second, there are a lot of opportunities for automating steps in the software development process. Organizations should trust employees to make the right decisions and make them responsible for automating the process.

2.2 Continuous Integration in DevOps

"DevOps describes techniques for automating repetitive tasks within the software development lifecycle (SDLC) [21], such as software builds, testing, and deployments, allowing these tasks to occur more naturally and frequently throughout the SDLC."

When set of developers in a team doing new implementation of the code, they test the modification locally and then check-in the code changes into the central repository. Therefore developers must focus on frequent code check-ins every day, to stay away from multifarious merge troubles. After code changes are checked in to the repository, CI system takes the control of building the latest system. It monitors the source control repositories for all the configured projects. When it detects a new code check-in, then it pulls an updated version of the code and the goes through the configured build pipelines. The CI server compiles and builds the new code when the project is written in a compiled language. The CI server also runs the associated unit test, regression test and UI test suites for the project to check on latest changes. When these configured build pipeline succeed, the server then run through the release pipeline to deploy the application to a staging environments. If any of these configured steps fails, the CI server will fail the build and immediate after the failure, it stops the build process and sends failure notifications to the entire project team. Keep the build passing through the build pipeline at all times is the team's goal, therefore a developer who breaks the build should immediately get actions to fix the issue and get it back on track. In explained manner, the CI server helps the habit of thoroughly testing of code before check-in it to the repository, to avoid build breaks and troublemaking towards the productivity and efficiency of other developers in the team.

In real development environment, without a QA process also a developer can check broken code into the source control repository. Other developers in the team may make changes that depend on this latest broken code, or attempt to merge new changes with it. When this kind of situation arises, the control of the team can be lost about the system's working state, and undergo in a loss of energy when they are forced to revert back the changes from many developers to return the software back to at least previous functional stage.

“CI servers automatically compile, build, and test every new version of code committed to the central team repository, [22] ensures that the entire team is alerted any time the central code repository contains broken code. This severely limits the chance for terrible merge issues and loss of work built upon a broken codebase.” Ensuring the Agile dream of a consistent working version of software, the CI server also automatically deploys the build latest application to a quality assurance (QA) environment or staging environment.

All the above described steps and actions are performed based on the automated configuration in the build pipeline and deployment scripts in release pipeline which collaboratively written by development and operations engineers. The most important thing is team collaboration because of the fact that it ensures that the operations expertise in deployment needs. And these automated scripts are understandable and can be used for enhancement by all team members. This team collaboration which comes with DevOps practice sets the stage for use of the same scripts to finally deploy the system into production environments with high confidence, a process known as continuous deployment.

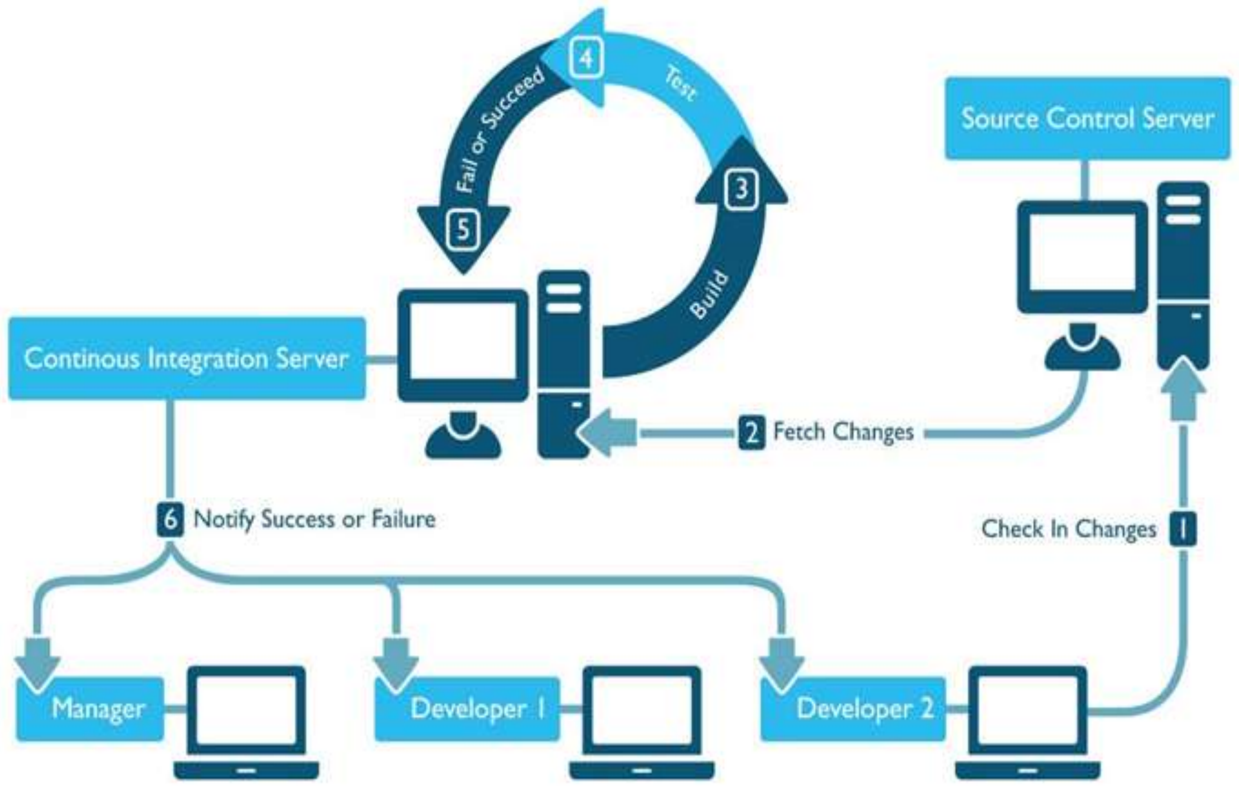


Figure 2.2.1: Continuous Integration Flow

Above figure illustrate the CI process of a company. At first the CI server checks out latest code changes from central repository. Then it compiles the build according to the builds pipeline and then run the testing for the code. At this testing stage it primarily runs the unit tests, and also possible though static code analysis. After these task completed when the latest code is tested, the CI server deploys the latest system it to staging environment (QA). In this stage, the CI server can also perform other tests requiring such as user interface testing, integration testing and advanced security testing. This ensures the quality of latest running version of the software. Reliable with ever changing agile requirements which emphasize the persistently proper working version of the desired software, ability of revert back automatically to the last successful working version of the software and keep a working stage of system existing even if integration tests failed are the advantages of continuous integration.

2.3 Software Architecture in Practice, Architecture Issues, Software Erosion and Software Drift

Software change is unavoidable. All software systems need to be progress with always expanding and changing user requirements. Therefore, it is very important for Software Company to perform maintenance of software in such a way as to diminish complications that arising from changes and the possibility of occurring new bugs to be with the latest changes.

Architectural erosion usually occurs when a program's code is initially less than optimal, contains "hacks" to quickly add functionality or simply cannot support the desired new changes due to the code structure. The problem it presents is an ever increasing maintenance cost due to the complexity of the system as a result of the accumulation of various design decisions. The only viable solution to fix or prevent architectural erosion is to rewrite the entire code base from scratch and try to anticipate future developments in order to accommodate them.

“Architectural drift is when the implementation of a program diverges from the initial design and purpose.[6] The problems it brings are similar to erosion; it will be increasingly difficult to further develop, maintain or even understand the code because design decisions are not always apparent when only looking at the code.“ To prevent architectural drift, there needs to be some kind of communication between the system designers and the implementers such as having the designers follow the progress of the implementation or having the implementers carefully document their work.

Software erosion would cost more money to fix because drift can be corrected earlier on in the development stage whereas erosion is usually left until much later, a situation which can be likened to prevention of a problem versus curing the problem after the fact. Because erosion is usually dealt with after the development of a program is complete it may involve many more stakeholders than drift would.

Software architecture erosion can be caused by a number of problems associated with the way the software is commonly developed.

- Traceability of design decisions

It is very important to know the prescriptive architecture of the system in order to change the system when dealing with new requirements. Problems can occur when the notations commonly used to create software lack the expressiveness needed to express concepts used during design. There are no or lack of proper documentation about the implemented functionality or the design, which eventually leads to making guesses about the system.

- Maintenance cost is increasing

During the software evolution the maintenance task becomes increasingly effort consuming due to the fact that the complexity of the system keeps growing. Those task can be both time consuming and costly. This may eventually cause the developers to take suboptimal design decisions either because they do not understand the architecture or because a more optimal decision would be too effort consuming

- Accumulation of design decisions

Due to the hierarchical nature of design decisions high level architectural decisions are followed by many low level architectural design decisions. The design decisions are accumulated and interact in a way such that revision of one would force reconsideration of all of the others. When a programmer decide to change a design for any reason, then they must consider the system as a whole and take a optimal strategic decision which eventually consider all other decisions affected by that or they must work with a system design which is not going to be optimal

- Iterative methods

A primary goal of the system architecture design is to create a design that can accommodate future changes to the system easily. This conflicts with the iterative nature of many software development methods (ex: in agile) since these methodologies typically incorporate new requirements that may have an architectural impact, during development where a proper design requires knowledge about these requirements in advance.

- Lack of continuous refactoring

Refactoring should be done regularly, if not then small design or implementation issues, architectural smells or decision inconsistencies will be accumulated, and consequently the software qualities will degrade

- Uncertainty about the evolution of the system

Most of the times when the creation of prescriptive architecture takes place the designers are uncertain about the possible future goals of the system. What are the future extensions, the possible future integrations and migrations are not visible during the primary phases due to various reasons. This may lead the prescriptive architecture and hence the descriptive architecture hard to maintain

- Release pressure

With the increasing number of change requests and the tight schedules the developers are forced to complete the tasks assigned to the as soon as possible. Though the task completed without a problem and passed the user acceptance tests that don't mean the fix or the change made to the system was the optimal one. The developer may be unintentionally change the system architecture and cause the system to early eroded state

- Changing requirements

This cannot be stopped. Requirements are in their nature are subjected to change. What we can do is to build the system so that it could withstand the changes in the future. Knowing the possible changes or the possible requirements can help the designers to do a better job

- Lack of knowledge about early design decisions

This happens due to both lack of documentation and staff turnover. If the code base is not self-explanatory (when the system is growing cannot expect it to be self-explanatory always) then there should be a proper documentation or the developers needs to be in touch with the production and maintenance. If not it's hard to understand and maintain the intended purpose of the system

When a software system getting large and getting mature with the time it will tend to degrade from the original architecture and erosion will eventually happen. Identifying the software erosion with the time as early as possible will help to recover from it or delay it. There are symptoms of a deteriorating system [16]

Code quality problem s of a source code may include unnecessarily complex or lengthy functions, abuse of language features, wrong use of infrastructure features etc. When an experience developer feels that the code is too complex for its intended purpose or there are so many boilerplate codes here and there it might be too late to recover it. Sometimes a well-developed code base may have violated a major design decision with its latest change. So with a review it can be identified as early as possible and take actions to solve the problem

Uncertainty about specifications Most of the times undocumented changes added to the system effectively making the existing design specifications obsolete. When it feels like there is a great deal of uncertainty about the system specification and the architecture design it might be a good time to take time to resolve those problems and after that move forward

Regressions Fixes for defects often introduce new problems. Some of them are visible or produced new bugs as soon as the new deployment goes and some of them will remain few years to show symptoms

Deployment problems since the original design of the system was developed aiming to cater a certain set of deployment configuration steps it might not be hold on with a new and changed set of deployment steps. This changing of the environments keep happening and we cannot stop that. These deployment problems can be identified when the system show symptoms regarding deployment issues

Several key areas that must be addressed to improve the software change process

- Change understanding and architecture analysis
- Build historical baseline of software change data
- Group changes based on impact/difficulty
- Facilitate discussion among developers

- Facilitate change difficulty/complexity estimation

The output of the review provides insight into the architectural change process and describes the effects that changes can have on architecture. Software is being changing rapidly because of the nature of its continuing changes, increasing complexity and continuing growth. As a result of the review findings, "the Software Architecture Change Characterization Scheme (SACCS) was created. [7] The attributes of the proposed scheme were extracted from change taxonomies and associated change characteristics identified during the review".

SACCS has been designed as a decision tree where choices made for the high-level characteristics affect decisions that can be made at the more detailed level. The specifics of the relationships among the attributes of the scheme will evolve as additional constraints and dependencies are identified. The characterizations are made using an electronic form. A developer can use this form to record their selections individually along with a rationale for the selections. The developer's characterization of the change can then be used to facilitate a discussion among other developers about the proposed impact. The goal is to determine whether the change can be made, given development constraints and architectural complexity.

This systematic review serves a starting point and provides the preliminary framework for a model of application-dependent change difficulty prediction for a change decision support system.

2.4 Source Code to Architecture Mapping Tools.

There are most of available software and tools to assist and to maintain and to understand a software system. These range of software which act as aids of maintenance or comprehension to full software development life cycle, for instance McCabe, Logiscope [Meek88] and SNiFF+1. All these tools supply graphical representations of the few areas of the software. Those analyses are basically a class inheritance hierarchy, function call graph, or activity flow graph. "The graphs produced by these tools [23] are unfortunately not very well presented and some use layout criteria which are not suitable for the data being displayed, such as symmetry and regular spacing of nodes.” The control flow graphs produced by the tools are more aesthetically pleasing due to the inherent tree-like structure of these graphs. One important feature which is present in these tools is the ability to move from the visualizations to the source code fairly easily, thus allowing the maintainer to switch between the low-level detail and the abstract visualizations.

As mentioned above there are some of tools that create the software architecture from the source code mapping. Among them following two covers almost all the areas of converting source code to multiple types of architecture design diagrams.

2.4.1 Doxygen

This is a tool that generates architecture design diagrams from source code. This supports almost all popular programming languages such as C, Objective-C, C#, PHP, Java, Python, FORTRAN, VHDL, Tcl, and to some extent D.

It can generate an on-line documentation browser and/or an off-line reference manual from a set of documented source files. In this software it has functions to generate output in various formats such as hyperlinked PDF, RTF (MS-Word), UNIX man pages, PostScript and compressed HTML. [21] The design documents are extracted directly from the sources. This helps to keep constant tracking by documenting along with the source code.

Doxygen can be configured to generate the architecture designs, code structure from the source files. Therefore this is very useful tool to visualize all the relation between all the elements exists in a project. And this helps to understand large source distributions. This software also have features to visualize the relationships between the a range of components and elements by means of including collaboration diagrams, inheritance diagrams and dependency graphs which are generated automatically. And also Doxygen can be used for creating normal documentation. “Doxygen is developed under Mac OS

X and Linux, but is set-up to be highly portable. [18] As a result, it runs on most other UNIX flavors as well. Furthermore, executables for Windows are available".

2.4.2 Architexa

Architexa is a very smart tool that help to fully understand a complex codebase by documenting it to visual diagrams. This gives following benefits to its users.

- Easily understand the code
- Maintain effortlessly
- Transfer knowledge between employees
- Increase code quality with architectural review
- Architectural challenges for Agile projects

Architexa lets you create diagrams that work as an intelligent sketchpad. If you do something often on a piece of paper and there is a way to make it easier to do on a computer, we want to support it.

Constantly changing code is the main difficulty when working in an Agile projects. Most of the UML tools are used to provide architectural support, but it require significant time to get its benefits. When the constantly code change, that is very hard for developers to create and update all the diagrams related to a modification. Rapid code modifications in agile projects [20] prevent teams from benefitting from a well-defined architecture and clear module boundaries resulting in a number of problems. This tool support development team by directing to undetected missing requirements, reducing difficulty when shared components are used and reducing the possibility of architecture erosion and code overlap.

"The Architexa suite has been designed to support developers in making architectural decisions and allow them to benefit from UML diagrams. [20] Understanding code architecture and preventing boundaries erosion. Diagrams built straight from the code allow developers to be connected to the code, and high-level overview diagrams like Layered Diagrams show developers' major code modules for them to dive into. It can be easily identify that where a newly implemented feature should be placed and how to prevent architectural erosion or a drift maintaining well-defined architecture and module boundaries"

This tool allows software developers to easily generate high-level design diagrams from the core code components. "Developers can go into the code step-by-step using the module dependencies, definitions, and capabilities, to identify similar components and ensure that the architecture [20] is kept up-to-date with the code". This tool beneficial to maintain the software system architecture with continues code changes. Developers can share the generating system architecture by Architexa with added notes and comments along with it. Therefore software design diagrams are common reference point for a team and it allow for developers to access required document when needed.

2.4.3 Code-maps

Making a code map essential to the user interface of the development environment promises to answer common information needs, reduce disorientation, and also to anchor team conversations. "Spatial memory and reasoning are little used by software developers today. In a lab-based evaluation of a previous version of our code-map design, developers form a reliable spatial memory of a code map during 90-minute sessions of programming tasks. Code maps [23] allow developers to be better grounded in the code, whether working solo or collaboratively". That helps to fundamentally change and improve the software development experience.

2.5 Defect Warnings Techniques.

Software quality is important, but often imperfect in practice. Including testing, code review, and formal specification, there can be used many techniques to try to improve quality, Static-analysis tools evaluate software in the abstract, without running the software or considering a specific input. Relatively this tool looks for code violations of recommended or reasonable programming practice than it trying to confirm that the code fulfills all its specification. Current trend is using statics analysis tool for defect detection in software, as it very useful it’s coming to practice very quickly.

In software development, there are so many defect warning techniques. Almost all tools and techniques are tightly coupled with the developing IDEs. Those tools analysis the software applications so as to identify potential defects, such as race conditions, deadlocks, wrong use of APIs and security issues. And those will show as violations and defects.

And also it is important to track that defect occurrences for the built versions. That helps to fix the pervious warning and defect in later versions. And there are one more advantages of keep versioning the defect, as if same issue occurs in multiple versions previous decisions can be used again. And also studying the lifecycle of defect warnings provides an interesting new perspective on code evolution. Tracking defects and warnings through a series of software versions disclose where potential warnings and defects are introduced and fixed, and the duration of the defect persistence, exposing interesting recovering trends and patterns.

With the evolution of software, it need to branch repositories by developers into separate development stages or merge separate repositories into a single development stream. However, when a branch is created for a new version of software, maintenance on the old version of branch also continues in parallel for some time duration, may be months or year. In the same way, maintenance can also continue on a module or a feature after it is merged into another branch or a repository. In this kind of situation, a developer who evaluate the system and find a bug warning needs to know whether the same issue is persist or not in other branches of the development process. Because there can be scenarios like those issues may already have been fixed or can be marked as a false positive on a different branch.

It can be handle this kind of situation differently by the techniques for warning signatures, pairing warnings and matching warnings with advantages and disadvantages to each approach. “The algorithm for pairing warnings used by FindBugs was designed with a linear sequence of software versions in mind. Pairing warnings is more fine-grained than warning-signatures in that it can determine, for example, which potential null-pointer dereference in a method was fixed between two versions. This type of fine-grained information [17] is very useful to a developer actively working on a linear branch who needs detailed information about bug warnings in order to best focus his resources”. However, the pairing implementation currently assumes that the lifespan of a warning is defined by the version in which it is introduced and the last version in which it still exists. Consequently, it would be difficult to apply the pairing approach to capture equivalent warnings between code branches of software as it is currently implemented.

“The warning-signatures approach used by Fortify Software [17] computes a unique hash for each instance of a warning based on a string representation of the name of the bug pattern and the name of the method and class file in which it occurs. If there are collisions, the string value is simply re-hashed to resolve the collision”. A major benefit is that the hash can easily be used to find warnings in earlier versions of a linear development stream as well as in separate branches of parallel development. Thus, the warning-signatures approach is more appropriate for developers who need to fix bugs across branches of software, or who need to integrate branches of software together.

2.6 Layered diagrams

Logical division of components of software and functionality are considered as layers in software. And the physical location of components are not considered in these layered concept whereas all layers illustrate the physical distribution of the software functionality and components on separate computers ,servers, networks, or remote locations.” Layers can be located on different tiers, [14] or they may reside on the same tier. Application model shows several layers - presentation layer, services layer, business, data, and cross-cutting layers. All layers are represented as UML models”.

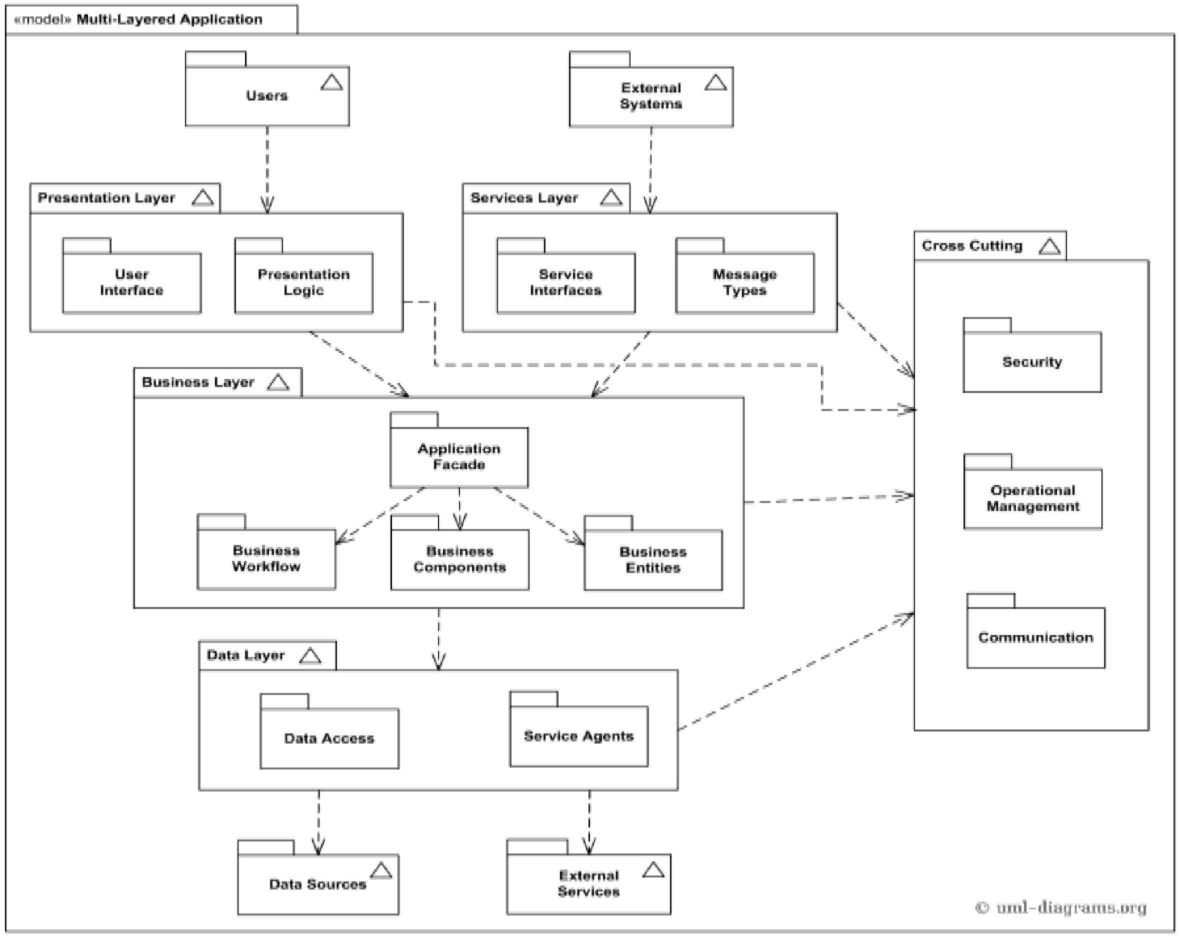


Figure 2.6.1: Layered Diagram

“Users and external systems are also represented as models, and communicate with presentation and services layers, correspondingly. Diagram also shows data sources such as relational databases and web service agents that provide access to data and external or remote services that are consumed by the application [17]”.

Layered diagrams can be used to design architecture diagrams and make sure the code stay consistent with its design in Visual Studio latest versions. We can track work associated with our models by linking team foundation server work items model elements. “This provides traceability between the models the IDE and the code [14] whether you designing a new design or updating the existing one. When refactoring an application layered diagram can be used to check whether it is consistent with design”.

A layered diagram shows the permitted dependencies within the solution which are mapped to layers. They are made up with shapes and relationships between those shapes. Following figure illustrate the layered diagrams in visual studio.

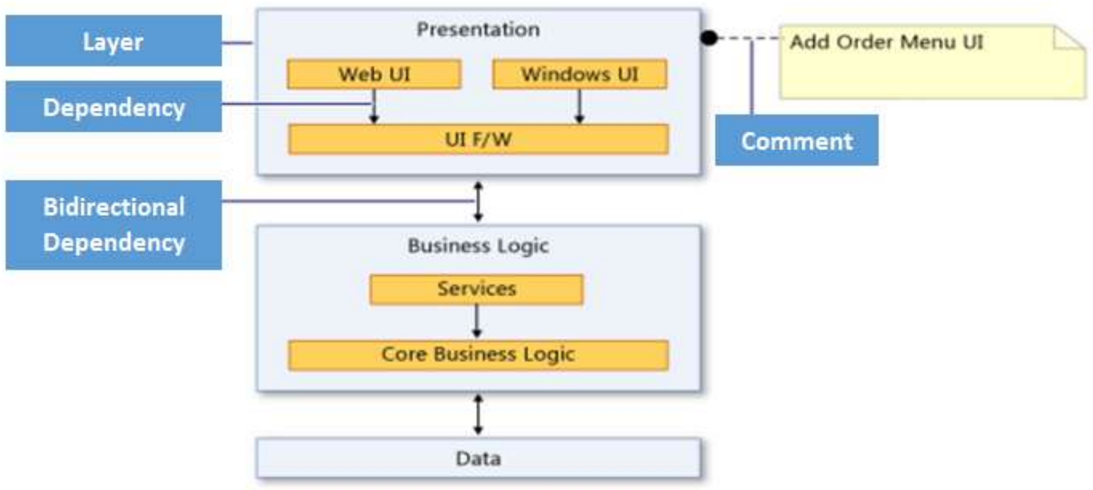


Figure 2.6.2: Layered Diagram in Visual Studio

Layer diagram allows lot more than the explanation of the intended architecture. Validation also can be done code against the layered diagram to determine whether an unacceptable dependency exists. By including this step is a part of check in and build process the changes can be more easily realized in the code that violates the architecture design.

Each layer can be linked to artifacts in the solution, such as projects, classes, namespaces, project files, and other parts of your software. The numbers of artifacts that are linked to the tiers are showed by the number on a layer.

CHAPTER 3

METHODOLOGY

Ultimate objective of this research is to monitor the architectural degradation in DevOps practice by detecting software drift and erosion so as to keep the software system in right track. So as to achieve the mentioned objective, desired formula is to build a monitoring framework that continuously analyzes the changing architecture design with the continuous implementation of the software system. The delivery process of the project also can be improved by incorporating this new framework into the continuous integration and continuous deployment pipelines.

3.1 Identify the Possible Solution

The desired framework will be beneficial for the currently available systems which needed to preserve the consistent architecture design of a software system. At this initial stage, the framework will only available for the projects which done in .Net framework. Reverse engineering and Visual Studio layered diagrams to be used for the solution.

Reverse engineering is the intended methodology for creating the design diagram. Reverse engineering produce and update UML models from all source files in a project. In this desired methodology, identifying all the projects and its dependencies always this will create a layered diagram. This generated layered diagram will visualize the high-level, logical architecture of a software system. All new added references will be highlighted to easily identify the new additions and changes. This separation is the main advantage of this automatically generated layered design diagram. Since all the newly added projects references are highlighted in the diagram it would be very convenient for the technical persons to validate all the changes by one sight.

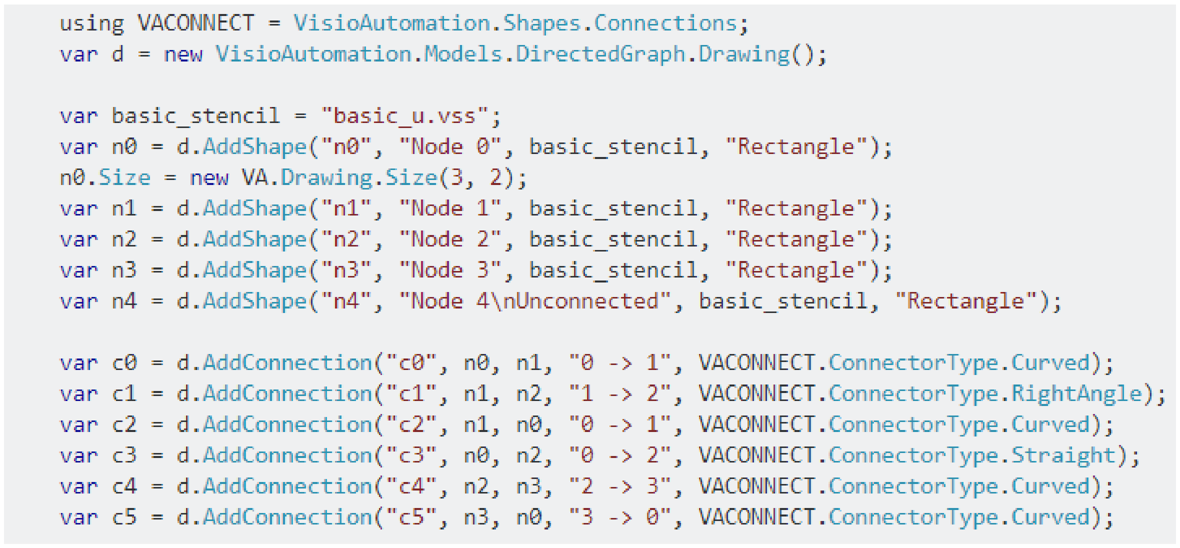
The newly added or existing dependencies between layers can be specifying a constant validated architecture. These dependencies, are represented as arrows, indicate which layers can use or currently use the functionality represented by other layers. By organizing a software system into layers that describe distinct roles and functions, a layer diagram can help make it easier to understand, reuse, and maintain the code.

3.2 Developing a Proof of Concept

In order to prove that the concept of this research actually matters and can actually contribute the software product line and software development life cycle, a tool was implemented. A project which uses a layered architecture style was used as the project under analysis. The tool was used to detect the architecture violations induced to that source code.

First approach is to observe the project dependencies from solution file and project file and create layered diagram using the observed details. Architecture changes notification using the .net layer diagram azure pipeline and create a framework improving specification of dependences. And the second approach is to add notification system and architecture design document comparison in to open source tool such as Architexa.

Following Visio tools will be used to do the implementation of the design diagram document.



```
using VACONNECT = VisioAutomation.Shapes.Connections;
var d = new VisioAutomation.Models.DirectedGraph.Drawing();

var basic_stencil = "basic_u.vss";
var n0 = d.AddShape("n0", "Node 0", basic_stencil, "Rectangle");
n0.Size = new VA.Drawing.Size(3, 2);
var n1 = d.AddShape("n1", "Node 1", basic_stencil, "Rectangle");
var n2 = d.AddShape("n2", "Node 2", basic_stencil, "Rectangle");
var n3 = d.AddShape("n3", "Node 3", basic_stencil, "Rectangle");
var n4 = d.AddShape("n4", "Node 4\nUnconnected", basic_stencil, "Rectangle");

var c0 = d.AddConnection("c0", n0, n1, "0 -> 1", VACONNECT.ConnectorType.Curved);
var c1 = d.AddConnection("c1", n1, n2, "1 -> 2", VACONNECT.ConnectorType.RightAngle);
var c2 = d.AddConnection("c2", n1, n0, "0 -> 1", VACONNECT.ConnectorType.Curved);
var c3 = d.AddConnection("c3", n0, n2, "0 -> 2", VACONNECT.ConnectorType.Straight);
var c4 = d.AddConnection("c4", n2, n3, "2 -> 3", VACONNECT.ConnectorType.Curved);
var c5 = d.AddConnection("c5", n3, n0, "3 -> 0", VACONNECT.ConnectorType.Curved);
```

Figure 3.2.1: Visio Tool Usage for Design Documents

And also this framework would keep a software architecture evaluation report for each version. Therefore, any problems with the application can be detected and analyzed in a global context, and if problems are well annotated, different log sources can be correlated to learn even more about the state of the application and the project.

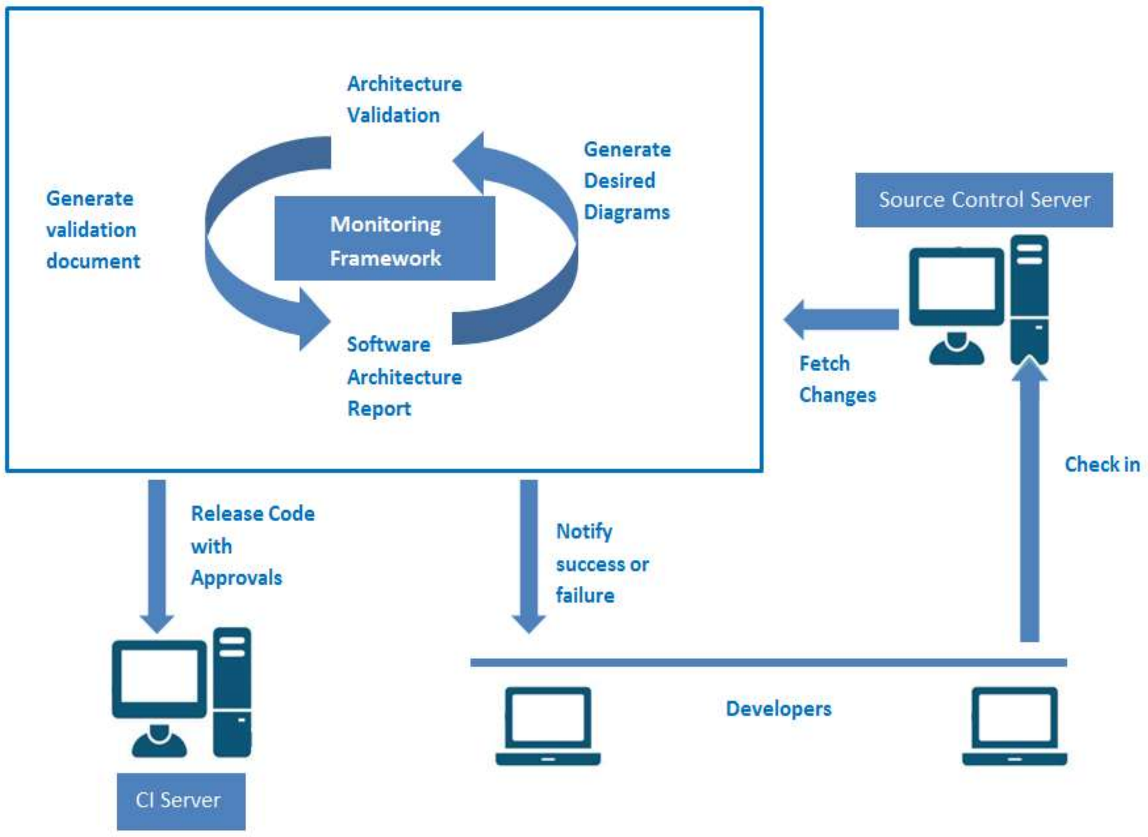

Architecture Validation
Generate Desired Diagrams
Generate validation document
Monitoring Framework
Software Architecture Report
Source Control Server
Fetch Changes
Check in
Release Code with Approvals
Notify success or failure
Developers
CI Server

Figure 3.2.2: High Level Architecture Diagram of the Monitoring Framework

The implementing framework is to be combined with the source control system to get latest software architecture evaluation report. At the first level, monitoring system should have the accurate software architecture design configured with it, both UML diagram and layered diagram. With the software evolution, if the software architecture diagrams need to be changed that should reflect to monitoring framework by adding modified architecture diagrams. The desired software architecture design evaluation can be done by configuring when to do it. There will be functionalities such as evaluation report for each code check in, code release, sporadic evaluation and etc.

The figure 3.1 illustrates the desired monitoring framework high level architecture. Whenever developers check in their code changes to source control server that changes to be fetched with the monitoring framework. With those newly added changes, latest software architecture diagrams would be generated through the monitoring framework such as UML diagrams, layered diagrams and etc. So as to identify the changed and the continued features created diagrams should be compared with the configured architecture diagrams. According to that comparisons software architecture evaluation report will be generated and whenever architecture change occurs that will be notified to development team. Development team could correct the occurred software drifts and then the code can be released to the Continuous Integration (CI) Server.

- This framework will give the analysis reports of the architecture.
- There will be architectural design analysis with each release.
- Whenever there is a new development release, this framework would identify the architectural changes and would warn about them.
- This will keep the architectural changes and implementing code architectural history as in source control history.
- Where there is an architectural drift or erosion, the code deployment out to production can be restricted until that fixed.

This would improve the way how the development and the production handoff happen. Helping identify and prevent architectural degradation issues, this framework would enhance the benefits of the DevOps practice, the product and software company growth.

3.3 Evaluation of the Proof of Concept

The implemented initial version was evaluated to check and verify its performance and capability of detecting the architecture violations accurately. Most of necessary evaluations have been done on the implemented initial version to verify the error detection. Improvement and enhancements were identified for the initial implemented version of the monitoring framework and appropriate conclusions were made according to the results.

CHAPTER 4

IMPLEMENTATION

Source or the code base is the most important artifact in the implementation of this motoring framework. The implanted solution must establish a way to extract the architecture information from the project source and compare the obtained architecture details with the prior defined architecture. Although there are most complex and costly ways to generate designs as mentioned in literature review, in this research it’s mainly focused to identify proper and beneficial way to achieve the same objective.

One of the identified solutions is introducing source code changes that need to be done by the developer or the architect so as to comply on the design during the development phase. This concept requires an initial training to use the appropriate tool and write codes which comply with that tool. According to the given facts by previous researchers there are several other ways of doing the same thing. In this research, the main focus is on two concepts out of them.

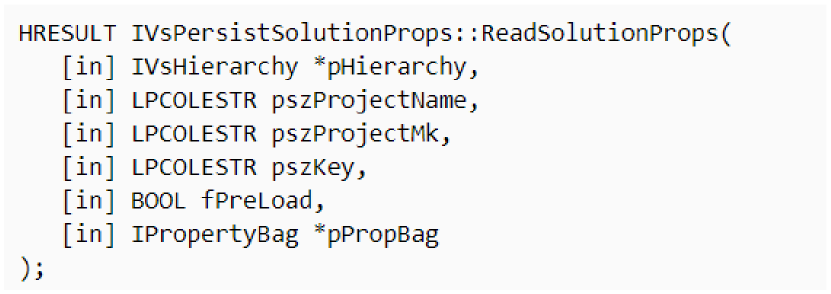
- Identify projects, components and its dependencies evaluating the solution file and project file.
- Generate layered architecture elaborating components inside layers.

4.1 Understanding the solution file.

A solution is the basic structure of the way how projects are organized in Visual Studio. The solution file maintains all the information of all state for each project in .sln in text-based format.

When solution projects are created and built in Visual Studio IDE, Visual Studio uses MSBuild to build each project in the solution. Each and every .Net project includes an MSBuild project file which has a file extension that reflects the project details. So as to build projects in solution, MSBuild processes the project file and it is associated with the project. The project file is an XML document which contains all the information and instructions for build the project using MSBuild. The file content includes database or web server settings, platform requirements, tasks that must be performed, versioning information and etc.

The project file which is an XML document is the main and easiest point of obtaining the all project details and its dependencies.

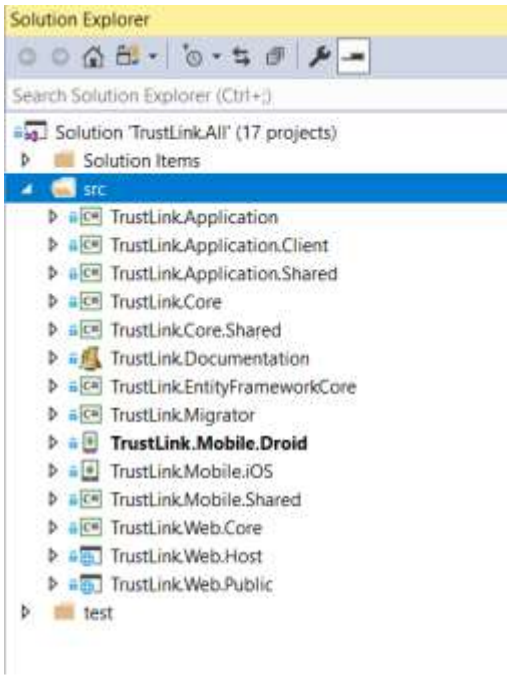


```
HRESULT IVsPersistSolutionProps::ReadSolutionProps(
   [in] IVsHierarchy *pHierarchy,
   [in] LPCOLESTR pszProjectName,
   [in] LPCOLESTR pszProjectMk,
   [in] LPCOLESTR pszKey,
   [in] BOOL fPreLoad,
   [in] IPropertyBag *pPropBag
);
```

Figure 4.1.1: Properties of a Solution

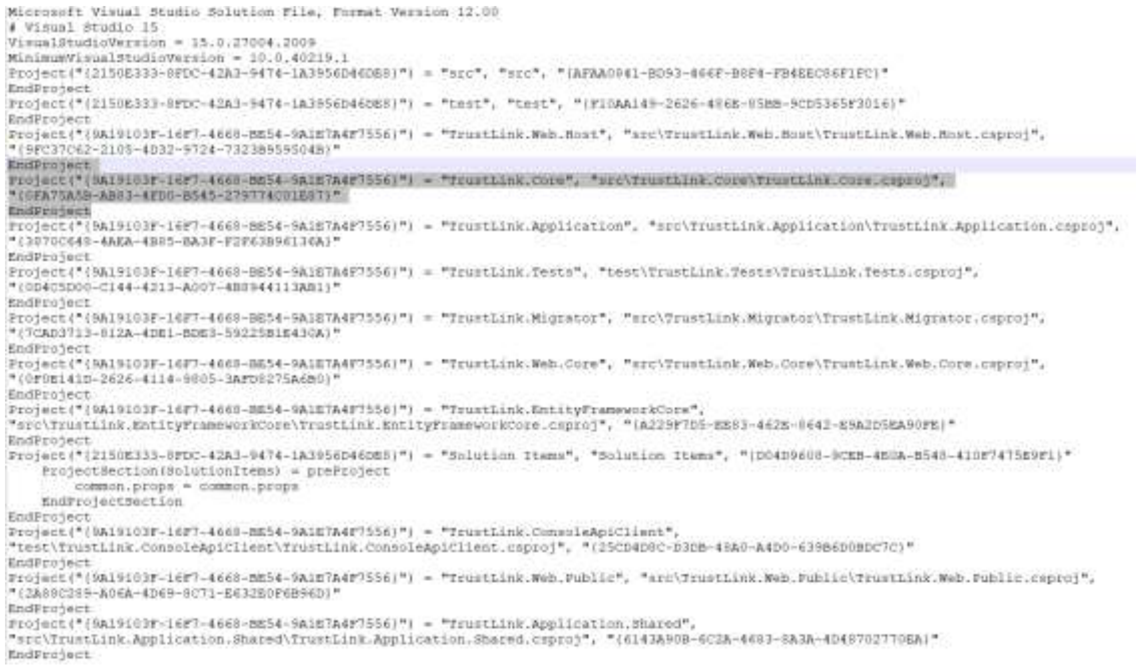
4.1.1 Solution File Contents.

Each and every project in a solution persists with a unique instance ID therefore other projects in the solution can retrieve the details as needed in an ideal manner. When the solution and projects are under source code control, the path of the project must be relative t0 the path of the solution. Having the solution file and project file stored on the server relative to the solution file, is relatively easy to find and copy the project file to the user's machine. Considering all those details, all the included projects, project file stored locations and project identity key and lot more details can be identified through the solution file (.sln file). This same methodology is used for this framework to identify the project architecture.



```
Solution 'TrustLink.All' (17 projects)
  Solution Items
  src
    TrustLink.Application
    TrustLink.Application.Client
    TrustLink.Application.Shared
    TrustLink.Core
    TrustLink.Core.Shared
    TrustLink.Documentation
    TrustLink.EntityFrameworkCore
    TrustLink.Migrator
    TrustLink.Mobile.Droid
    TrustLink.Mobile.iOS
    TrustLink.Mobile.Shared
    TrustLink.Web.Core
    TrustLink.Web.Host
    TrustLink.Web.Public
  test
```

Figure 4.1.1.1: Sample Solution View

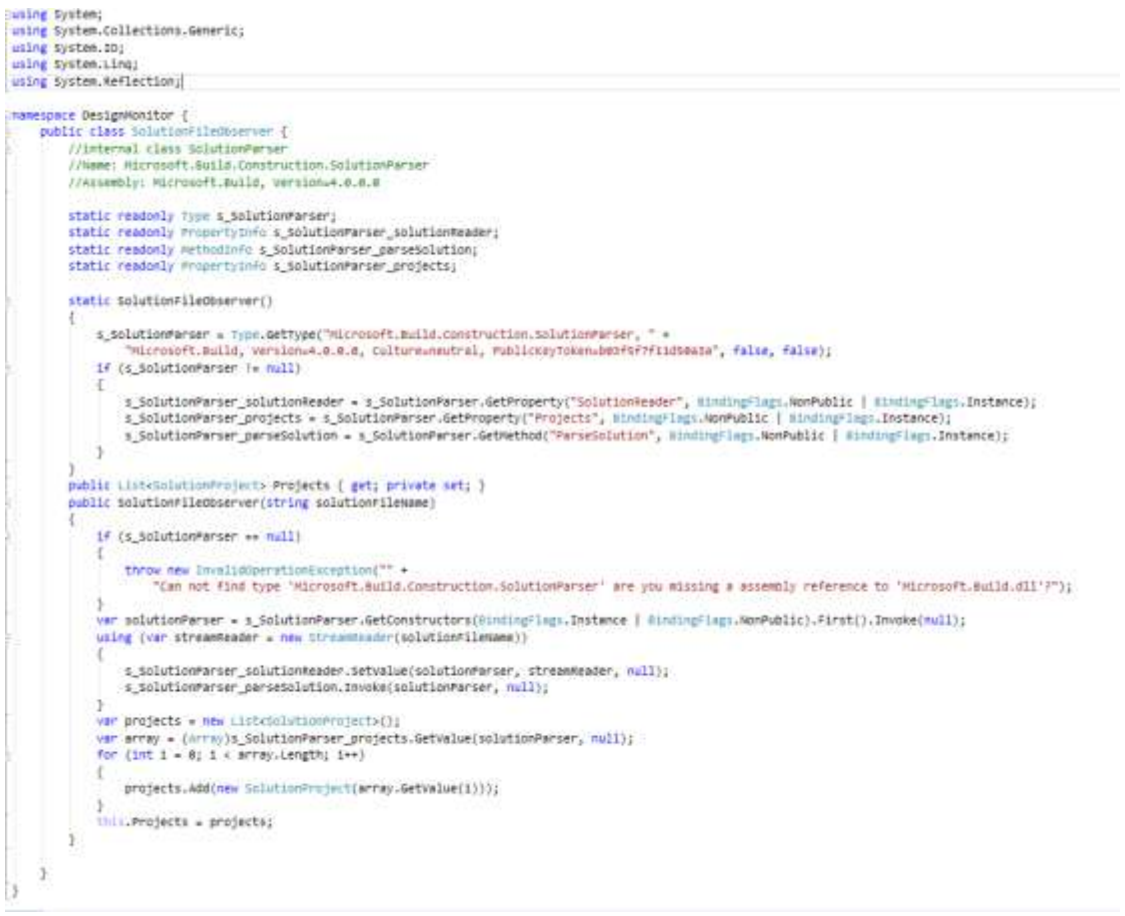


```
Microsoft Visual Studio Solution File, Format Version 12.00
# Visual Studio 15
VisualStudioVersion = 15.0.27004.2009
MinimumVisualStudioVersion = 10.0.40219.1
Project("{2150E333-8FDC-42A3-9474-1A3956D46DE8}") = "src", "src", "{AFAA0041-BD93-466F-B8F4-FB4EEC86F1FC}"
EndProject
Project("{2150E333-8FDC-42A3-9474-1A3956D46DE8}") = "test", "test", "{F10AA149-2626-486E-85BB-9CD5365F3016}"
EndProject
Project("{9A19103F-16F7-4668-BE54-9A1E7A4F7556}") = "TrustLink.Web.Host", "src\TrustLink.Web.Host\TrustLink.Web.Host.csproj",
"{9FC37C62-2105-4D32-9724-7323B959504B}"
EndProject
Project("{9A19103F-16F7-4668-BE54-9A1E7A4F7556}") = "TrustLink.Core", "src\TrustLink.Core\TrustLink.Core.csproj",
"{0FA75A5B-AB83-4FD0-B545-279774C01E87}"
EndProject
Project("{9A19103F-16F7-4668-BE54-9A1E7A4F7556}") = "TrustLink.Application", "src\TrustLink.Application\TrustLink.Application.csproj",
"{3870C648-4AEA-4B85-BA3F-F2F63B96134A}"
EndProject
Project("{9A19103F-16F7-4668-BE54-9A1E7A4F7556}") = "TrustLink.Tests", "test\TrustLink.Tests\TrustLink.Tests.csproj",
"{0D4C5D00-C144-4213-A007-4B8944113AB1}"
EndProject
Project("{9A19103F-16F7-4668-BE54-9A1E7A4F7556}") = "TrustLink.Migrator", "src\TrustLink.Migrator\TrustLink.Migrator.csproj",
"{7CAD3713-812A-4DE1-BDE3-59225B1E430A}"
EndProject
Project("{9A19103F-16F7-4668-BE54-9A1E7A4F7556}") = "TrustLink.Web.Core", "src\TrustLink.Web.Core\TrustLink.Web.Core.csproj",
"{0F0E141D-2626-4114-9805-3AFD8275A6B0}"
EndProject
Project("{9A19103F-16F7-4668-BE54-9A1E7A4F7556}") = "TrustLink.EntityFrameworkCore",
"src\TrustLink.EntityFrameworkCore\TrustLink.EntityFrameworkCore.csproj", "{A229F7D5-EE83-462E-8642-E9A2D5EA90FE}"
EndProject
Project("{2150E333-8FDC-42A3-9474-1A3956D46DE8}") = "Solution Items", "Solution Items", "{D04D9608-9CEB-4E0A-B548-410E7475E9F1}"
	ProjectSection(SolutionItems) = preProject
		common.props = common.props
	EndProjectSection
EndProject
Project("{9A19103F-16F7-4668-BE54-9A1E7A4F7556}") = "TrustLink.ConsoleApiClient",
"test\TrustLink.ConsoleApiClient\TrustLink.ConsoleApiClient.csproj", "{25CD4D0C-D3DB-48A0-A4D0-639B6D0BDC7C}"
EndProject
Project("{9A19103F-16F7-4668-BE54-9A1E7A4F7556}") = "TrustLink.Web.Public", "src\TrustLink.Web.Public\TrustLink.Web.Public.csproj",
"{2A89C289-A06A-4D69-8C71-E632E0F6B96D}"
EndProject
Project("{9A19103F-16F7-4668-BE54-9A1E7A4F7556}") = "TrustLink.Application.Shared",
"src\TrustLink.Application.Shared\TrustLink.Application.Shared.csproj", "{6143A90B-6C2A-4683-8A3A-4D48702770BA}"
EndProject
```

Figure 4.1.1.2: Sample .sln File

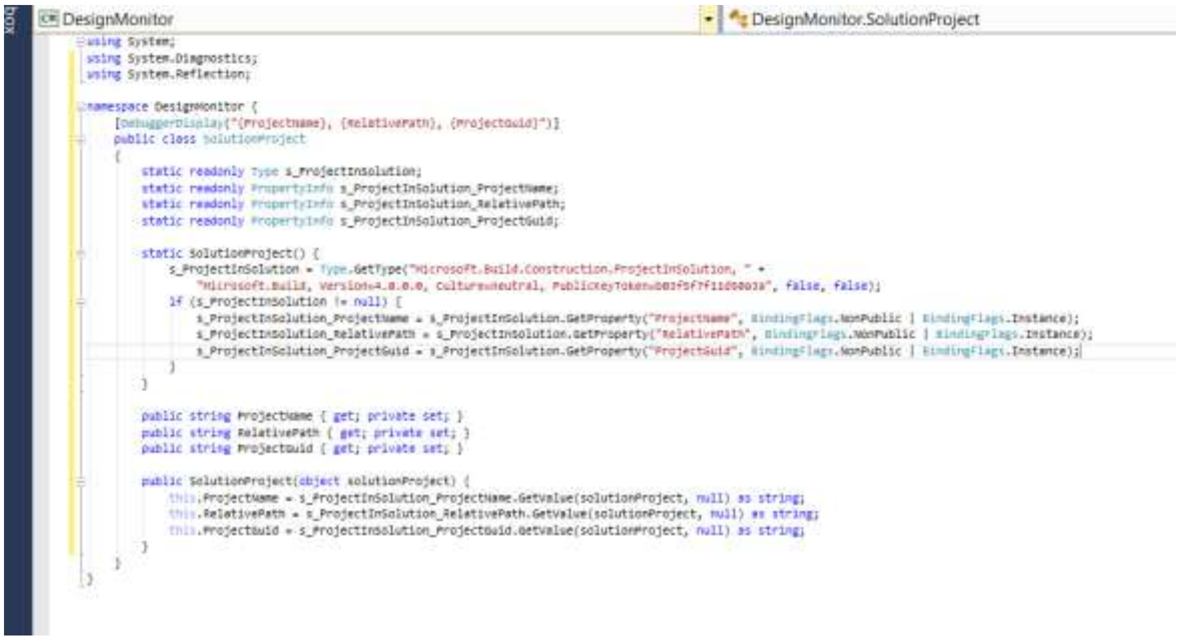
4.1.2 Methodology of Observing Solution Content

For the initial implementation of the framework is to identify the entire project detail summary by examining and reading the .sln file. In order to achieve that following methodology has been used in the basic implementation. It gets the solution file path and read the included project details iteratively. There we can obtain the list of project details along with the .csproj path details for further evaluation of architecture.



```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Reflection;

namespace DesignMonitor {
    public class SolutionFileObserver {
        //internal class SolutionParser
        //Name: Microsoft.Build.Construction.SolutionParser
        //Assembly: Microsoft.Build, Version=4.0.0.0

        static readonly Type s_SolutionParser;
        static readonly PropertyInfo s_SolutionParser_solutionReader;
        static readonly MethodInfo s_SolutionParser_parseSolution;
        static readonly PropertyInfo s_SolutionParser_projects;

        static SolutionFileObserver()
        {
            s_SolutionParser = Type.GetType("Microsoft.Build.Construction.SolutionParser, " +
                "Microsoft.Build, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a", false, false);
            if (s_SolutionParser != null)
            {
                s_SolutionParser_solutionReader = s_SolutionParser.GetProperty("SolutionReader", BindingFlags.NonPublic | BindingFlags.Instance);
                s_SolutionParser_projects = s_SolutionParser.GetProperty("Projects", BindingFlags.NonPublic | BindingFlags.Instance);
                s_SolutionParser_parseSolution = s_SolutionParser.GetMethod("ParseSolution", BindingFlags.NonPublic | BindingFlags.Instance);
            }
        }
        public List<SolutionProject> Projects { get; private set; }
        public SolutionFileObserver(string solutionFileName)
        {
            if (s_SolutionParser == null)
            {
                throw new InvalidOperationException("" +
                    "Can not find type 'Microsoft.Build.Construction.SolutionParser' are you missing a assembly reference to 'Microsoft.Build.dll'?");
            }
            var solutionParser = s_SolutionParser.GetConstructors(BindingFlags.Instance | BindingFlags.NonPublic).First().Invoke(null);
            using (var streamReader = new StreamReader(solutionFileName))
            {
                s_SolutionParser_solutionReader.SetValue(solutionParser, streamReader, null);
                s_SolutionParser_parseSolution.Invoke(solutionParser, null);
            }
            var projects = new List<SolutionProject>();
            var array = (Array)s_SolutionParser_projects.GetValue(solutionParser, null);
            for (int i = 0; i < array.Length; i++)
            {
                projects.Add(new SolutionProject(array.GetValue(i)));
            }
            this.Projects = projects;
        }

    }
}
```

Figure 4.1.2.1: Implementation of Solution File Observer



```
using System;
using System.Diagnostics;
using System.Reflection;

namespace DesignMonitor {
    [DebuggerDisplay("{ProjectName}, {RelativePath}, {ProjectGuid}")]
    public class SolutionProject
    {
        static readonly Type s_ProjectInSolution;
        static readonly PropertyInfo s_ProjectInSolution_ProjectName;
        static readonly PropertyInfo s_ProjectInSolution_RelativePath;
        static readonly PropertyInfo s_ProjectInSolution_ProjectGuid;

        static SolutionProject() {
            s_ProjectInSolution = Type.GetType("Microsoft.Build.Construction.ProjectInSolution, " +
                "Microsoft.Build, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a", false, false);
            if (s_ProjectInSolution != null) {
                s_ProjectInSolution_ProjectName = s_ProjectInSolution.GetProperty("ProjectName", BindingFlags.NonPublic | BindingFlags.Instance);
                s_ProjectInSolution_RelativePath = s_ProjectInSolution.GetProperty("RelativePath", BindingFlags.NonPublic | BindingFlags.Instance);
                s_ProjectInSolution_ProjectGuid = s_ProjectInSolution.GetProperty("ProjectGuid", BindingFlags.NonPublic | BindingFlags.Instance);
            }
        }

        public string ProjectName { get; private set; }
        public string RelativePath { get; private set; }
        public string ProjectGuid { get; private set; }

        public SolutionProject(object solutionProject) {
            this.ProjectName = s_ProjectInSolution_ProjectName.GetValue(solutionProject, null) as string;
            this.RelativePath = s_ProjectInSolution_RelativePath.GetValue(solutionProject, null) as string;
            this.ProjectGuid = s_ProjectInSolution_ProjectGuid.GetValue(solutionProject, null) as string;
        }
    }
}
```

Figure 4.1.2.2: Implementation of Solution Project

All included projects, relative paths and other needed details are obtained in above state. And then next step is to identify project specific details which are mandatory for doing relational mapping of existing projects and its components.

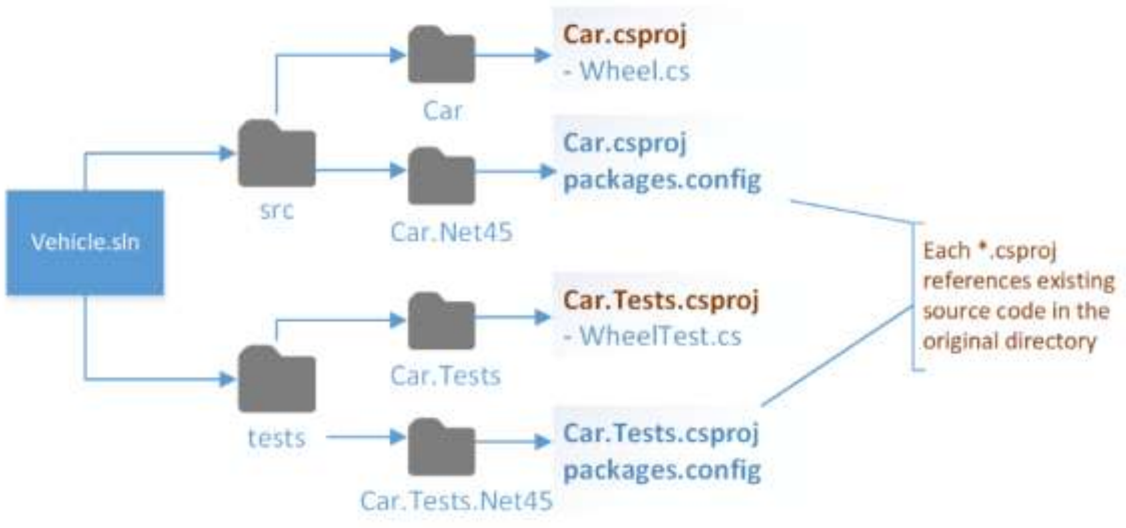


Figure 4.1.2.3: Solution Directory Hierarchy

4.2 Understanding the Project File

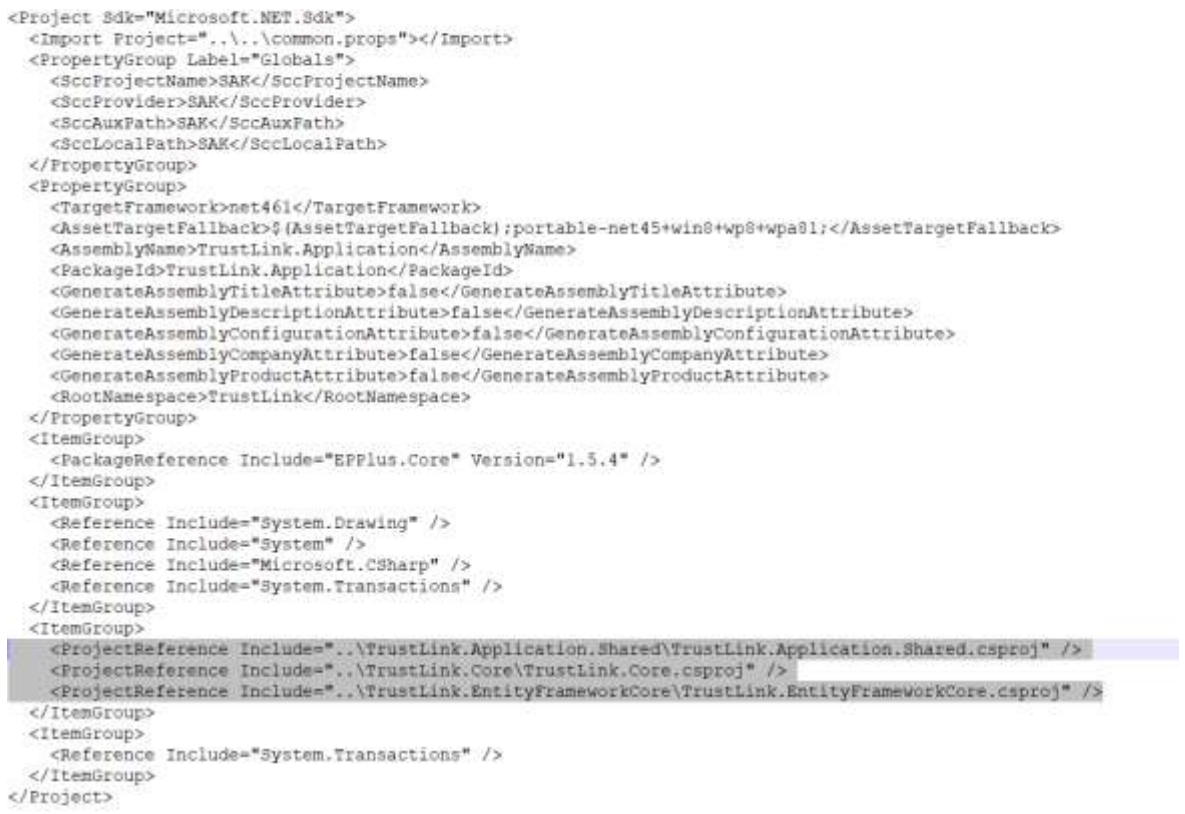
When the solution is created and built in Visual Studio IDE, it uses MSBuild to build each and every project in the solution. Each and every project in a solution includes an MSBuild project file, with a file extension that reflects the type of project. For example, a Visual Basic.NET project (.vbproj), a C# project (.csproj), or a database project (.dbproj). In the process of building a project, MSBuild need to process the project file associated with the project. The project file is an XML document that contains all the information and instructions that MSBuild needs so as to build the particular project. Since this is the initial state of the implementation, only the C# projects (.csproj) have been considered to continue on the implementation and evaluations.

4.2.1 Project File Contents.

Included MSBuild XML schema is the main artifact that used for identifying all the reference and connection with outside projects, in order to get the list of dependencies to map in the design document. In addition to identifying the XML schema for the project file, the Project element can include attributes to specify the entry points for the build process. The project element is the root element of every project file. Element by element can be observed here to get all the project specific details in to an object that includes all the properties.

In order to achieve the main objective, observing all project references are the main part of this .csproj file evaluation. Basically, ProjectReference element is the key for getting all these project dependencies programmatically.

Following figure elaborates how .csproj looks like and the way elements are organized. The highlighted area defines the outside project references and dependencies.



```xml
<Project Sdk="Microsoft.NET.Sdk">
  <Import Project="..\..\common.props"></Import>
  <PropertyGroup Label="Globals">
    <SccProjectName>SAK</SccProjectName>
    <SccProvider>SAK</SccProvider>
    <SccAuxPath>SAK</SccAuxPath>
    <SccLocalPath>SAK</SccLocalPath>
  </PropertyGroup>
  <PropertyGroup>
    <TargetFramework>net461</TargetFramework>
    <AssetTargetFallback>$(AssetTargetFallback);portable-net45+win8+wp8+wpa81;</AssetTargetFallback>
    <AssemblyName>TrustLink.Application</AssemblyName>
    <PackageId>TrustLink.Application</PackageId>
    <GenerateAssemblyTitleAttribute>false</GenerateAssemblyTitleAttribute>
    <GenerateAssemblyDescriptionAttribute>false</GenerateAssemblyDescriptionAttribute>
    <GenerateAssemblyConfigurationAttribute>false</GenerateAssemblyConfigurationAttribute>
    <GenerateAssemblyCompanyAttribute>false</GenerateAssemblyCompanyAttribute>
    <GenerateAssemblyProductAttribute>false</GenerateAssemblyProductAttribute>
    <RootNamespace>TrustLink</RootNamespace>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="EPPlus.Core" Version="1.5.4" />
  </ItemGroup>
  <ItemGroup>
    <Reference Include="System.Drawing" />
    <Reference Include="System" />
    <Reference Include="Microsoft.CSharp" />
    <Reference Include="System.Transactions" />
  </ItemGroup>
  <ItemGroup>
    <ProjectReference Include="..\TrustLink.Application.Shared\TrustLink.Application.Shared.csproj" />
    <ProjectReference Include="..\TrustLink.Core\TrustLink.Core.csproj" />
    <ProjectReference Include="..\TrustLink.EntityFrameworkCore\TrustLink.EntityFrameworkCore.csproj" />
  </ItemGroup>
  <ItemGroup>
    <Reference Include="System.Transactions" />
  </ItemGroup>
</Project>
```

Figure 4.2.1.1: Overview of Project File

4.2.2 Methodology of Observing Project Content

Project file is a file which is written in xml format including all the required details. XML stands for Extensible Markup Language file format. This is used to create common information formats. And these formats can be shared including both the format and the data on the World Wide Web, intranet, etc.

Following figure illustrates element organization of the xml file.

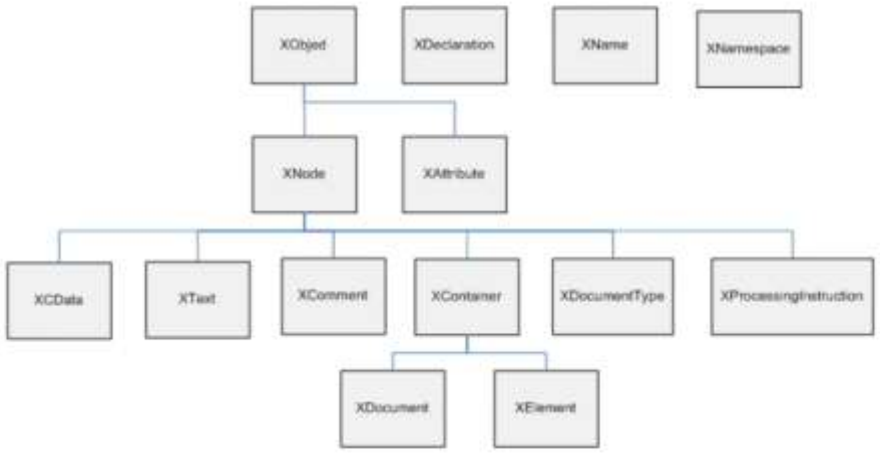
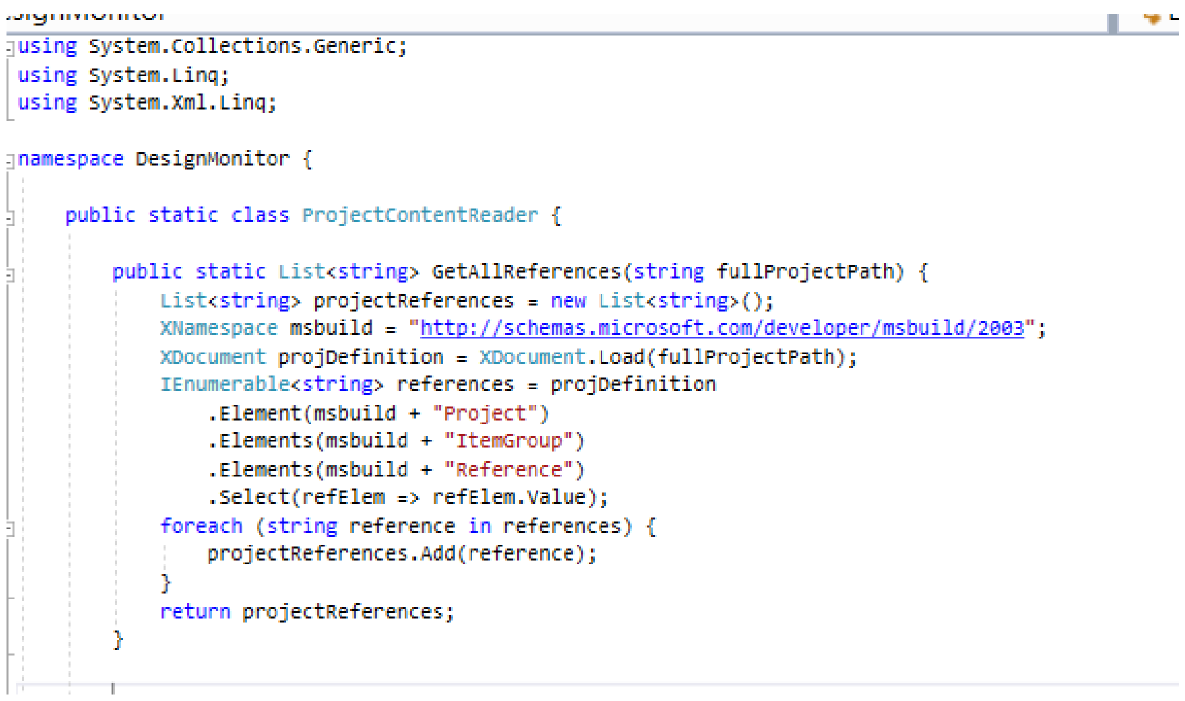


Figure 4.2.2.1: Overview of XML File Element Hierarchy

Since mostly used XML data manipulation is XDocument and XMLDocument class. In this project also same technology has been used, the following code snippet explains how the project dependencies are extracted from the .csproj file.



```
using System.Collections.Generic;
using System.Linq;
using System.Xml.Linq;

namespace DesignMonitor {

    public static class ProjectContentReader {

        public static List<string> GetAllReferences(string fullProjectPath) {
            List<string> projectReferences = new List<string>();
            XNamespace msbuild = "http://schemas.microsoft.com/developer/msbuild/2003";
            XDocument projDefinition = XDocument.Load(fullProjectPath);
            IEnumerable<string> references = projDefinition
                .Element(msbuild + "Project")
                .Elements(msbuild + "ItemGroup")
                .Elements(msbuild + "Reference")
                .Select(refElem => refElem.Value);
            foreach (string reference in references) {
                projectReferences.Add(reference);
            }
            return projectReferences;
        }
```

Figure 4.2.2.2: Implementation of Project Content Reader

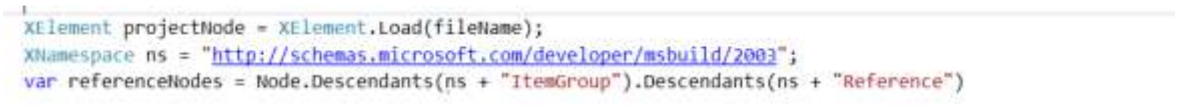

```
XElement projectNode = XElement.Load(fileName);
XNamespace ns = "http://schemas.microsoft.com/developer/msbuild/2003";
var referenceNodes = Node.Descendants(ns + "ItemGroup").Descendants(ns + "Reference")
```

Figure 4.2.2.2: Implementation of Project File Node Observer

At this stage we have observed list of projects and then the relationships of projects to one another, evaluating the .sln file and .csproj file. All the details that need to draw the layered design have been observed. Therefore these extracted layers, projects and its references are store in a new .xml file in order to compare with future changes. Generated xml is used to identify changes time to time. Therefore this file is stored with each code build version in repository.

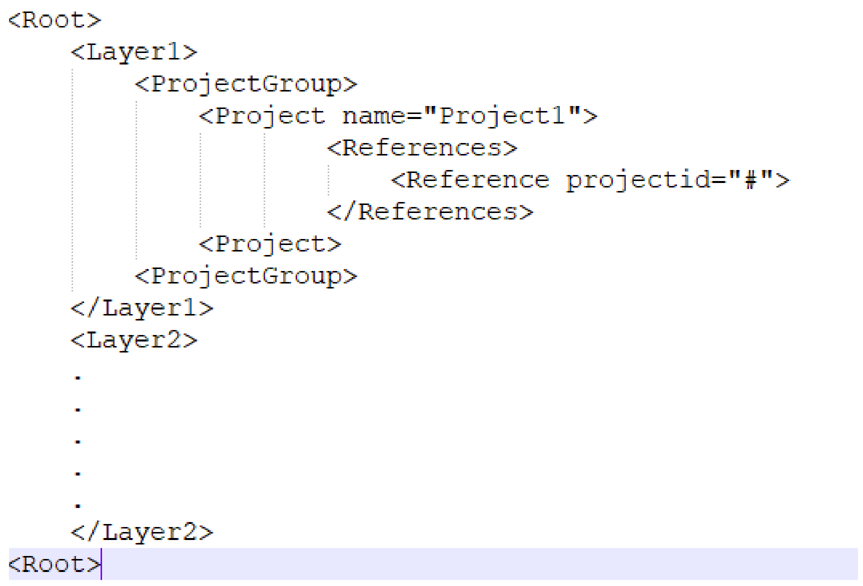

```
<Root>
    <Layer1>
        <ProjectGroup>
            <Project name="Project1">
                <References>
                    <Reference projectid="#">
                </References>
            <Project>
        <ProjectGroup>
    </Layer1>
    <Layer2>
    .
    .
    .
    .
    .
    </Layer2>
<Root>
```

Figure 4.2.2.3: Generated XML Structure

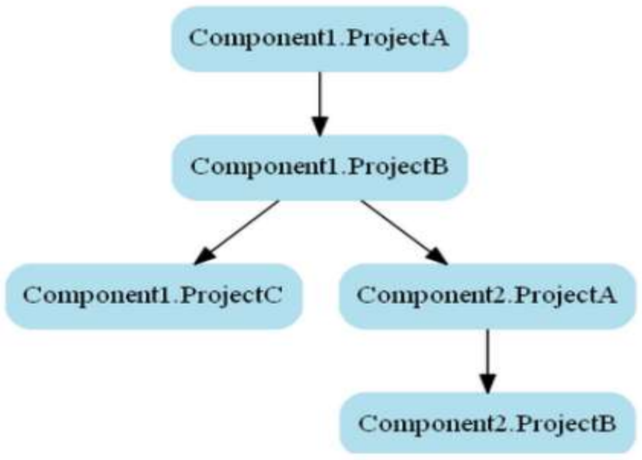
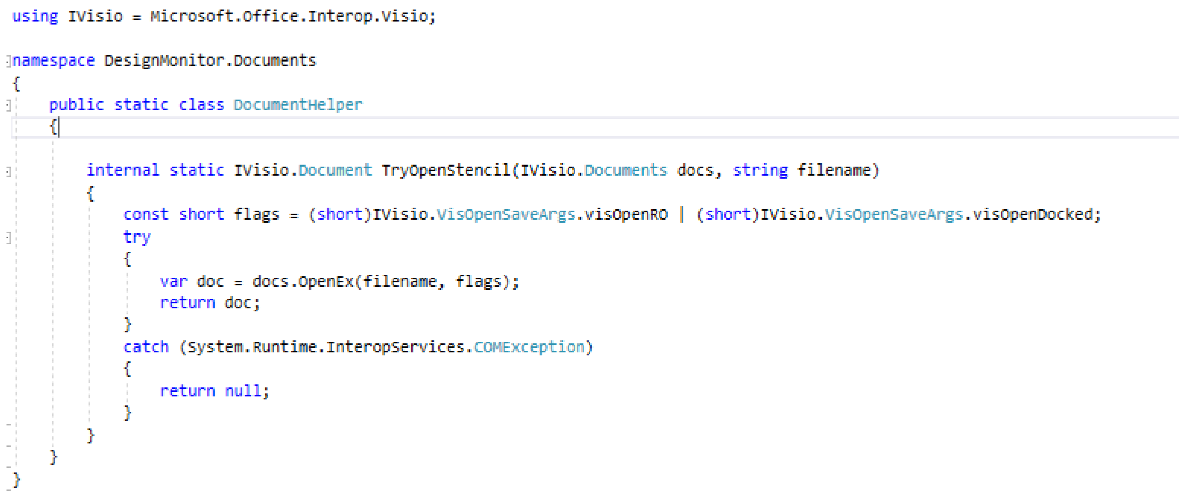


Figure 4.2.2.3: Observed Project Detail Hierarchy

4.3 Generate Architectural Design

This section discusses how the design document is generated programmatically. A design document is the basically the solution structure of the projects and their hierarchical relationships. This section discusses how to dynamically generate a VISIO file containing the layered architecture design for a given particular solution reading data from a data source.

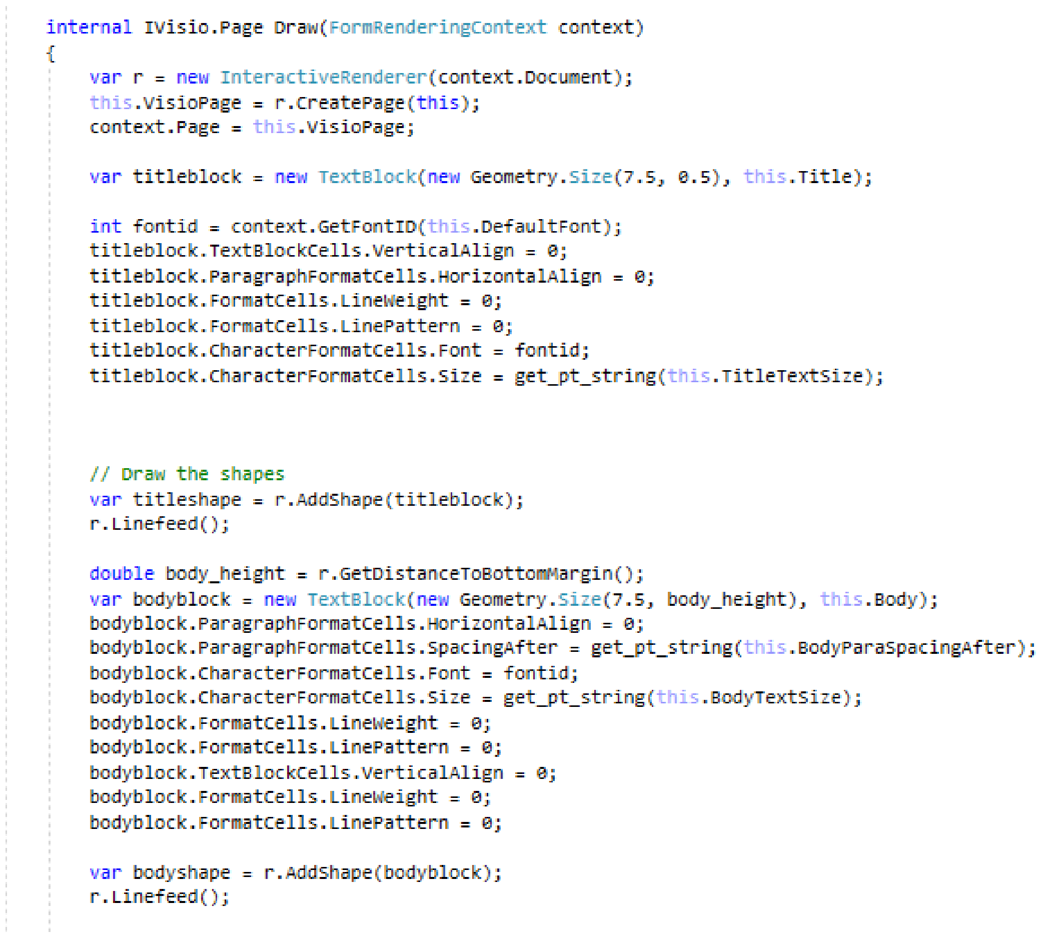


```
using IVisio = Microsoft.Office.Interop.Visio;

namespace DesignMonitor.Documents
{
    public static class DocumentHelper
    {

        internal static IVisio.Document TryOpenStencil(IVisio.Documents docs, string filename)
        {
            const short flags = (short)IVisio.VisOpenSaveArgs.visOpenRO | (short)IVisio.VisOpenSaveArgs.visOpenDocked;
            try
            {
                var doc = docs.OpenEx(filename, flags);
                return doc;
            }
            catch (System.Runtime.InteropServices.COMException)
            {
                return null;
            }
        }
    }
}
```

Figure 4.3.1: Implementation of Design Helper

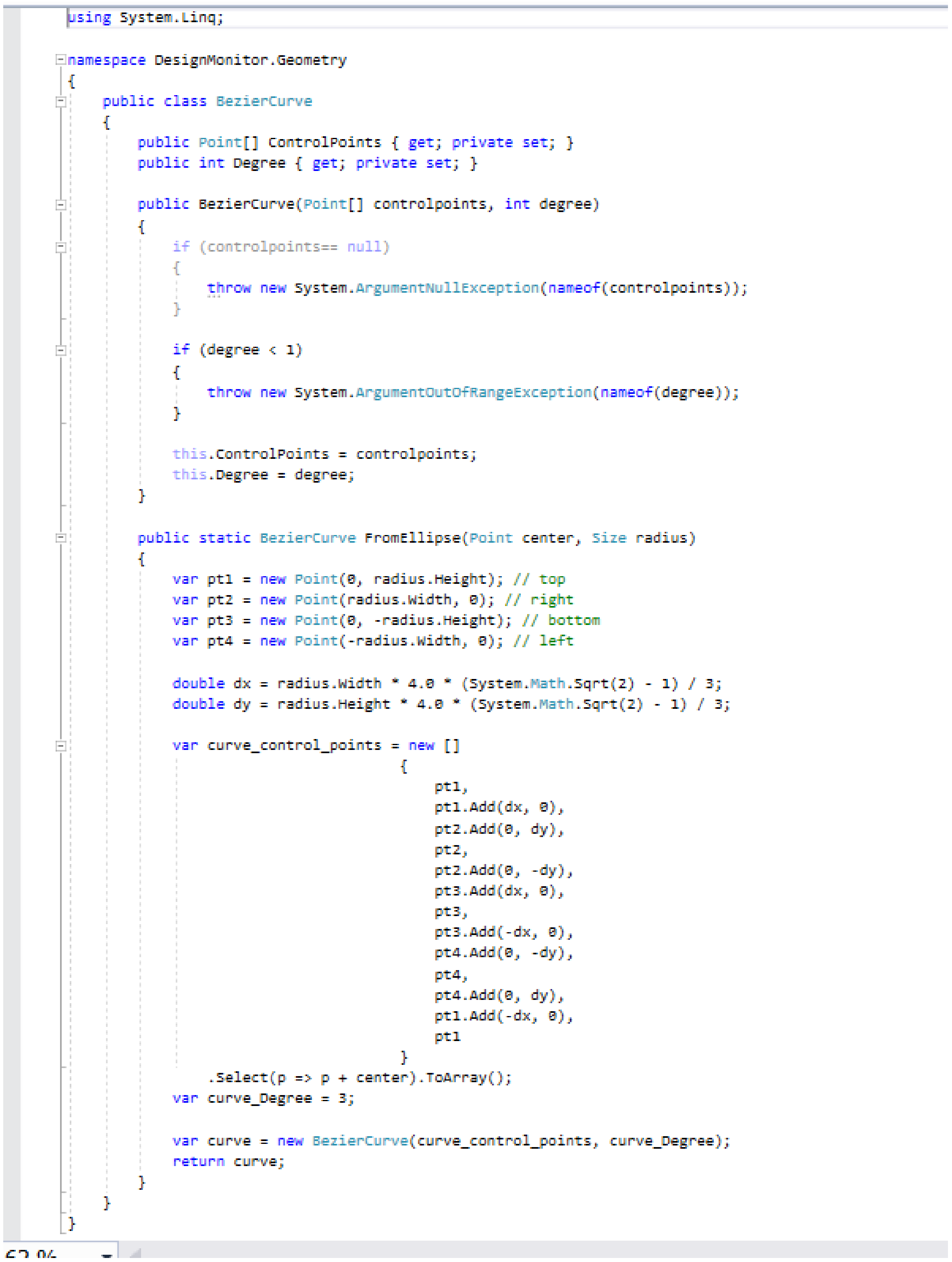
4.3.1 Component Draw and Placement in Document

Achieving the task of drawing the components of the solution with accurate placing is the next massive target. For that here we have to obtain the information about the sheet width and height to know where to place the components. Then it is putting the root boxes in the middle of the sheet by dividing the sheet width by the number of roots. Also it is subtracting from the yPosition the level number so that the boxes with increasing level number will get a lower position on the design chart.



```
internal IVisio.Page Draw(FormRenderingContext context)
{
    var r = new InteractiveRenderer(context.Document);
    this.VisioPage = r.CreatePage(this);
    context.Page = this.VisioPage;

    var titleblock = new TextBlock(new Geometry.Size(7.5, 0.5), this.Title);

    int fontid = context.GetFontID(this.DefaultFont);
    titleblock.TextBlockCells.VerticalAlign = 0;
    titleblock.ParagraphFormatCells.HorizontalAlign = 0;
    titleblock.FormatCells.LineWeight = 0;
    titleblock.FormatCells.LinePattern = 0;
    titleblock.CharacterFormatCells.Font = fontid;
    titleblock.CharacterFormatCells.Size = get_pt_string(this.TitleTextSize);



    // Draw the shapes
    var titleshape = r.AddShape(titleblock);
    r.Linefeed();

    double body_height = r.GetDistanceToBottomMargin();
    var bodyblock = new TextBlock(new Geometry.Size(7.5, body_height), this.Body);
    bodyblock.ParagraphFormatCells.HorizontalAlign = 0;
    bodyblock.ParagraphFormatCells.SpacingAfter = get_pt_string(this.BodyParaSpacingAfter);
    bodyblock.CharacterFormatCells.Font = fontid;
    bodyblock.CharacterFormatCells.Size = get_pt_string(this.BodyTextSize);
    bodyblock.FormatCells.LineWeight = 0;
    bodyblock.FormatCells.LinePattern = 0;
    bodyblock.TextBlockCells.VerticalAlign = 0;
    bodyblock.FormatCells.LineWeight = 0;
    bodyblock.FormatCells.LinePattern = 0;

    var bodyshape = r.AddShape(bodyblock);
    r.Linefeed();
```

Figure 4.3.1.1: Implementation of Design Drawer

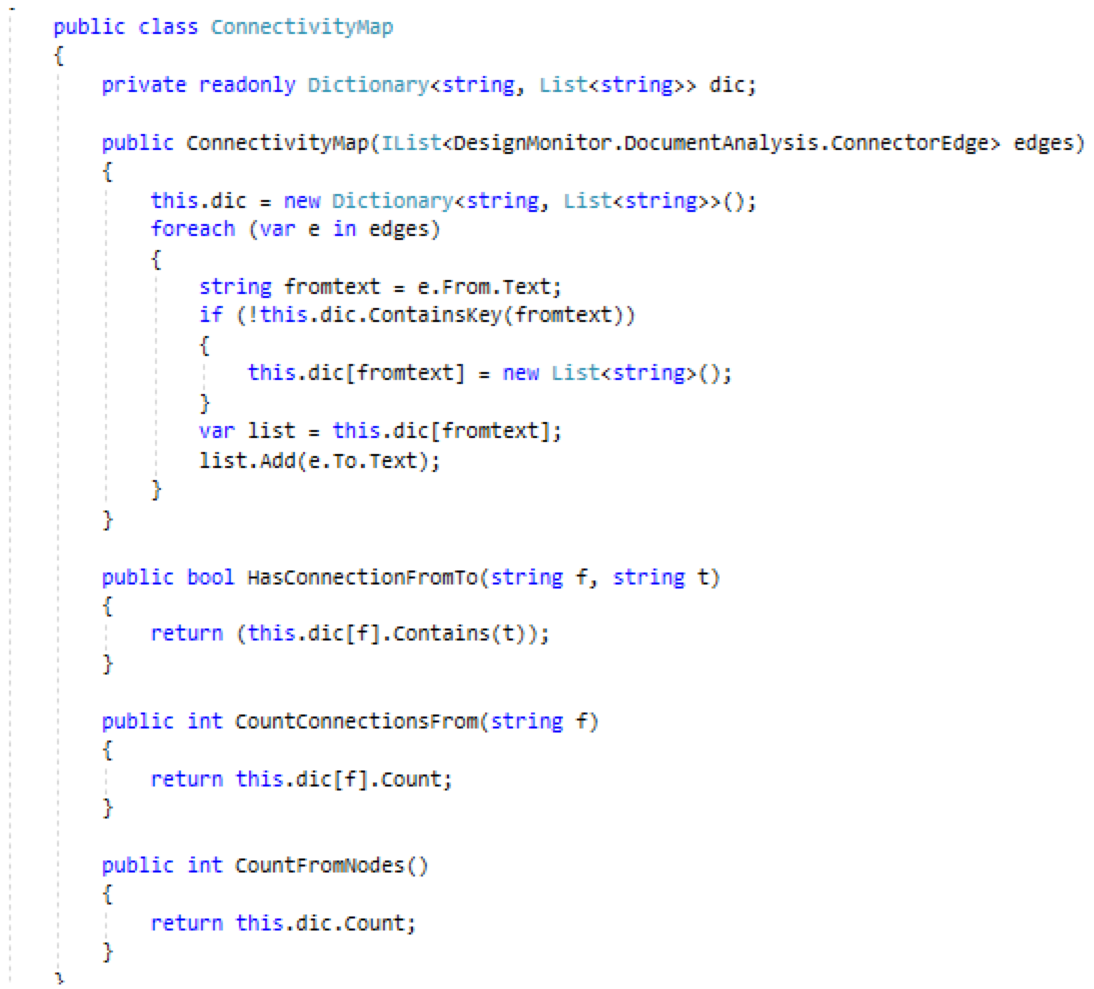


```
using System.Linq;

namespace DesignMonitor.Geometry
{
    public class BezierCurve
    {
        public Point[] ControlPoints { get; private set; }
        public int Degree { get; private set; }

        public BezierCurve(Point[] controlpoints, int degree)
        {
            if (controlpoints== null)
            {
                throw new System.ArgumentNullException(nameof(controlpoints));
            }

            if (degree < 1)
            {
                throw new System.ArgumentOutOfRangeException(nameof(degree));
            }

            this.ControlPoints = controlpoints;
            this.Degree = degree;
        }

        public static BezierCurve FromEllipse(Point center, Size radius)
        {
            var pt1 = new Point(0, radius.Height); // top
            var pt2 = new Point(radius.Width, 0); // right
            var pt3 = new Point(0, -radius.Height); // bottom
            var pt4 = new Point(-radius.Width, 0); // left

            double dx = radius.Width * 4.0 * (System.Math.Sqrt(2) - 1) / 3;
            double dy = radius.Height * 4.0 * (System.Math.Sqrt(2) - 1) / 3;

            var curve_control_points = new []
                                {
                                    pt1,
                                    pt1.Add(dx, 0),
                                    pt2.Add(0, dy),
                                    pt2,
                                    pt2.Add(0, -dy),
                                    pt3.Add(dx, 0),
                                    pt3,
                                    pt3.Add(-dx, 0),
                                    pt4.Add(0, -dy),
                                    pt4,
                                    pt4.Add(0, dy),
                                    pt1.Add(-dx, 0),
                                    pt1
                                }
                .Select(p => p + center).ToArray();
            var curve_Degree = 3;

            var curve = new BezierCurve(curve_control_points, curve_Degree);
            return curve;
        }
    }
}
```

Figure 4.3.1.2: Implementation of Geometry for Design Document

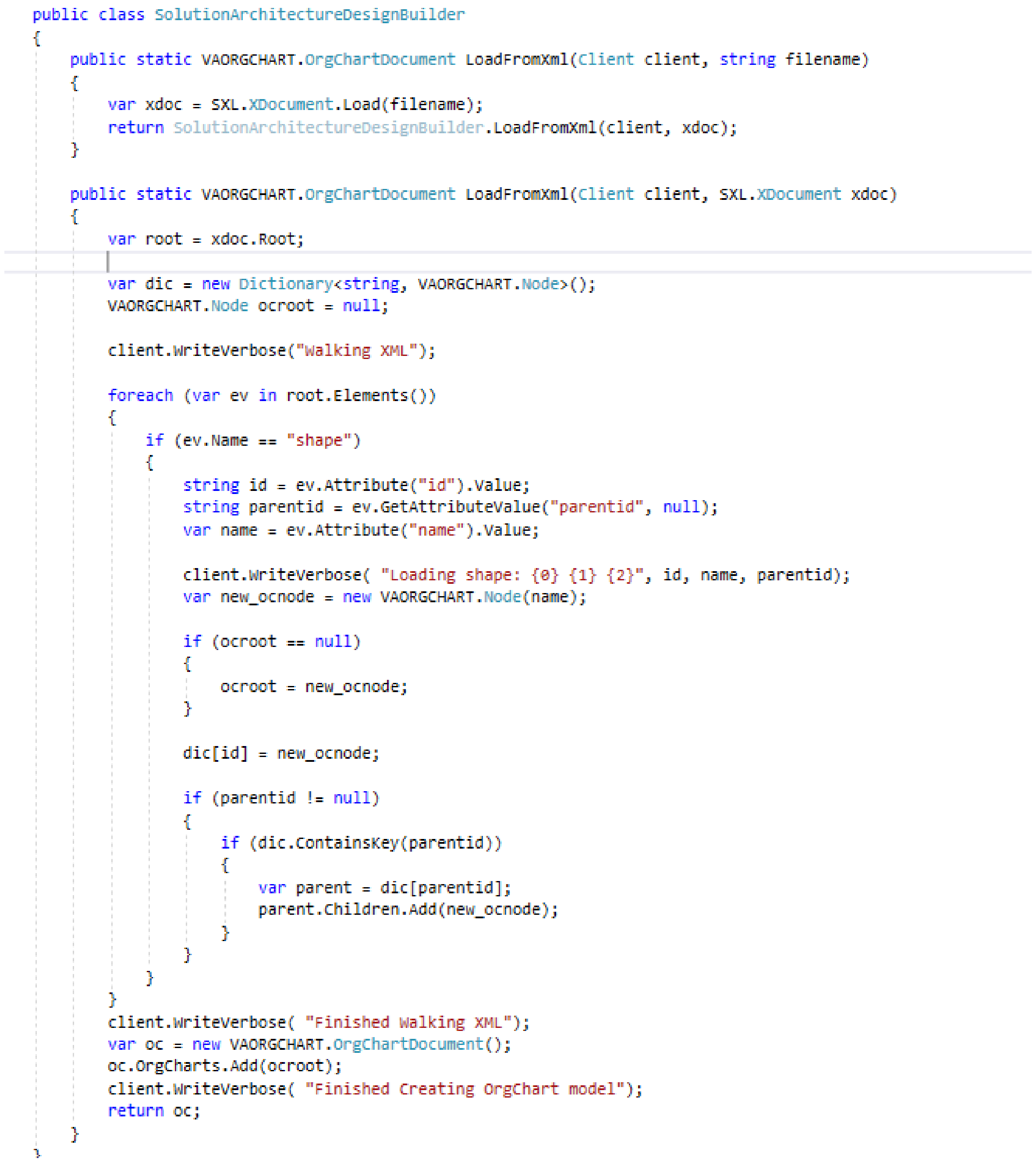
4.3.2 Map Relationships of Project Components

This connectivity map is used for the specify relations between layers, projects and components. After drawing of all the components have been done, it iterates through the dependencies and connects the edges of components. And then coloring is done for the components to distinguish from one another. Created xml file including all the required information is being processed through following solution architecture design builder in order to generate the desired design document.



```
public class ConnectivityMap
{
    private readonly Dictionary<string, List<string>> dic;

    public ConnectivityMap(IList<DesignMonitor.DocumentAnalysis.ConnectorEdge> edges)
    {
        this.dic = new Dictionary<string, List<string>>();
        foreach (var e in edges)
        {
            string fromtext = e.From.Text;
            if (!this.dic.ContainsKey(fromtext))
            {
                this.dic[fromtext] = new List<string>();
            }
            var list = this.dic[fromtext];
            list.Add(e.To.Text);
        }
    }

    public bool HasConnectionFromTo(string f, string t)
    {
        return (this.dic[f].Contains(t));
    }

    public int CountConnectionsFrom(string f)
    {
        return this.dic[f].Count;
    }

    public int CountFromNodes()
    {
        return this.dic.Count;
    }
}
```

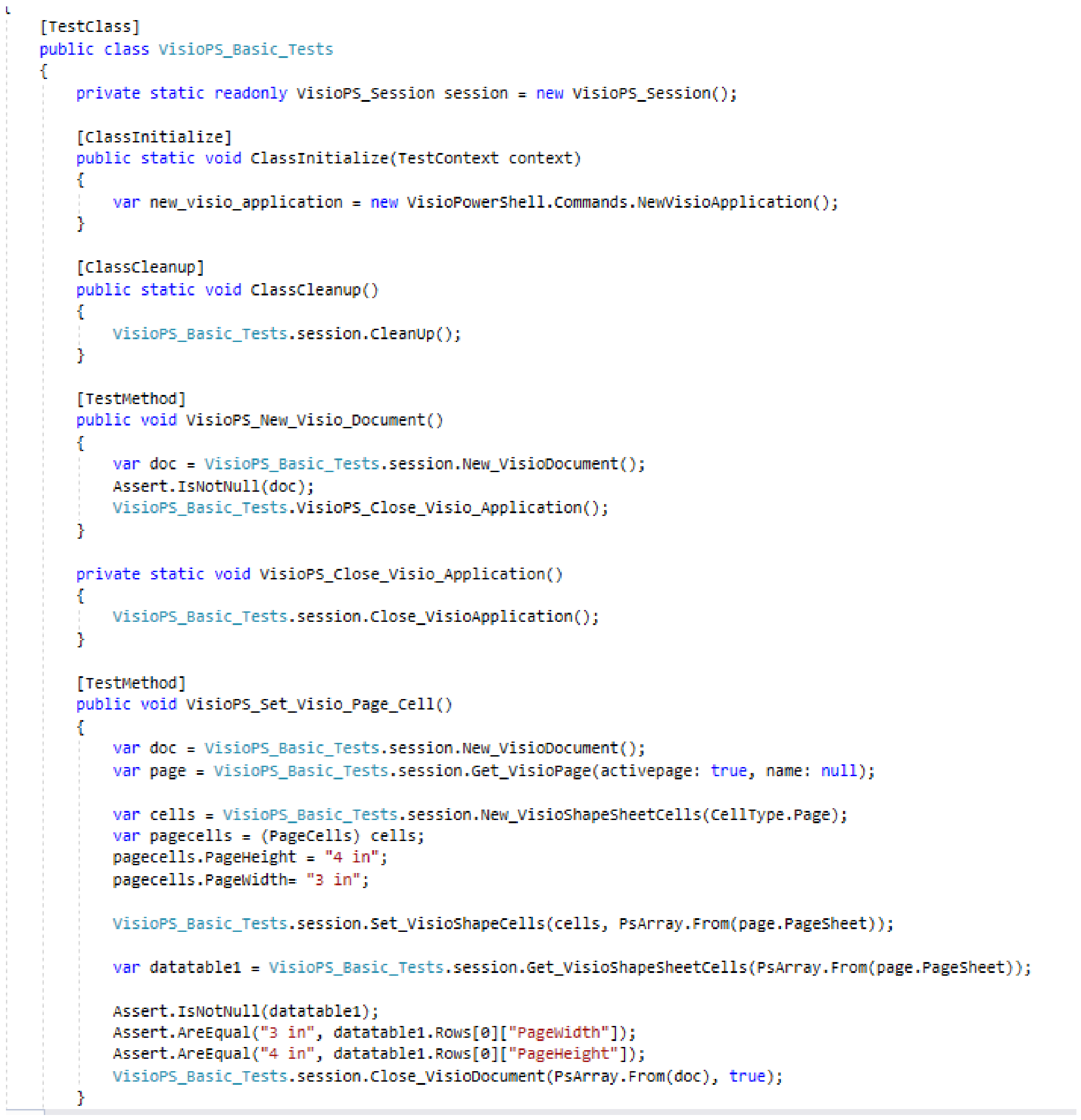
Figure 4.3.2.1: Implementation of Mapping Relationship



```
public class SolutionArchitectureDesignBuilder
{
    public static VAORGCHART.OrgChartDocument LoadFromXml(Client client, string filename)
    {
        var xdoc = SXL.XDocument.Load(filename);
        return SolutionArchitectureDesignBuilder.LoadFromXml(client, xdoc);
    }

    public static VAORGCHART.OrgChartDocument LoadFromXml(Client client, SXL.XDocument xdoc)
    {
        var root = xdoc.Root;

        var dic = new Dictionary<string, VAORGCHART.Node>();
        VAORGCHART.Node ocroot = null;

        client.WriteVerbose("Walking XML");

        foreach (var ev in root.Elements())
        {
            if (ev.Name == "shape")
            {
                string id = ev.Attribute("id").Value;
                string parentid = ev.GetAttributeValue("parentid", null);
                var name = ev.Attribute("name").Value;

                client.WriteVerbose( "Loading shape: {0} {1} {2}", id, name, parentid);
                var new_ocnode = new VAORGCHART.Node(name);

                if (ocroot == null)
                {
                    ocroot = new_ocnode;
                }

                dic[id] = new_ocnode;

                if (parentid != null)
                {
                    if (dic.ContainsKey(parentid))
                    {
                        var parent = dic[parentid];
                        parent.Children.Add(new_ocnode);
                    }
                }
            }
        }
        client.WriteVerbose( "Finished Walking XML");
        var oc = new VAORGCHART.OrgChartDocument();
        oc.OrgCharts.Add(ocroot);
        client.WriteVerbose( "Finished Creating OrgChart model");
        return oc;
    }
}
```

Figure 4.3.2.2: Implementation of Component Placement in Document

Following test formula elaborates the Visio document creation verification that done with unit testing for the described scenarios.



```
[TestClass]
public class VisioPS_Basic_Tests
{
    private static readonly VisioPS_Session session = new VisioPS_Session();

    [ClassInitialize]
    public static void ClassInitialize(TestContext context)
    {
        var new_visio_application = new VisioPowerShell.Commands.NewVisioApplication();
    }

    [ClassCleanup]
    public static void ClassCleanup()
    {
        VisioPS_Basic_Tests.session.CleanUp();
    }

    [TestMethod]
    public void VisioPS_New_Visio_Document()
    {
        var doc = VisioPS_Basic_Tests.session.New_VisioDocument();
        Assert.IsNotNull(doc);
        VisioPS_Basic_Tests.VisioPS_Close_Visio_Application();
    }

    private static void VisioPS_Close_Visio_Application()
    {
        VisioPS_Basic_Tests.session.Close_VisioApplication();
    }

    [TestMethod]
    public void VisioPS_Set_Visio_Page_Cell()
    {
        var doc = VisioPS_Basic_Tests.session.New_VisioDocument();
        var page = VisioPS_Basic_Tests.session.Get_VisioPage(activepage: true, name: null);

        var cells = VisioPS_Basic_Tests.session.New_VisioShapeSheetCells(CellType.Page);
        var pagecells = (PageCells) cells;
        pagecells.PageHeight = "4 in";
        pagecells.PageWidth= "3 in";

        VisioPS_Basic_Tests.session.Set_VisioShapeCells(cells, PsArray.From(page.PageSheet));

        var datatable1 = VisioPS_Basic_Tests.session.Get_VisioShapeSheetCells(PsArray.From(page.PageSheet));

        Assert.IsNotNull(datatable1);
        Assert.AreEqual("3 in", datatable1.Rows[0]["PageWidth"]);
        Assert.AreEqual("4 in", datatable1.Rows[0]["PageHeight"]);
        VisioPS_Basic_Tests.session.Close_VisioDocument(PsArray.From(doc), true);
    }
```

Figure 4.3.2.3: Automates Unit Tests

4.4 Design Diagram

Before delivering the application architecture and design team to use in the designing phase, business process analysts have to process descriptions in text that need to be converted t0 UML layered diagrams. UML design diagrams are the easier and clearer for architects and developers to understand and compare with one another. It can take a long time and significant effort for business analysts to analysis all the business processes and descriptions and create design diagrams for each check-in. Having a tool that automates generation of the models from solution file is certainly a great help. The SDM tool provides the capability to create such a valuable artifact whenever needed.

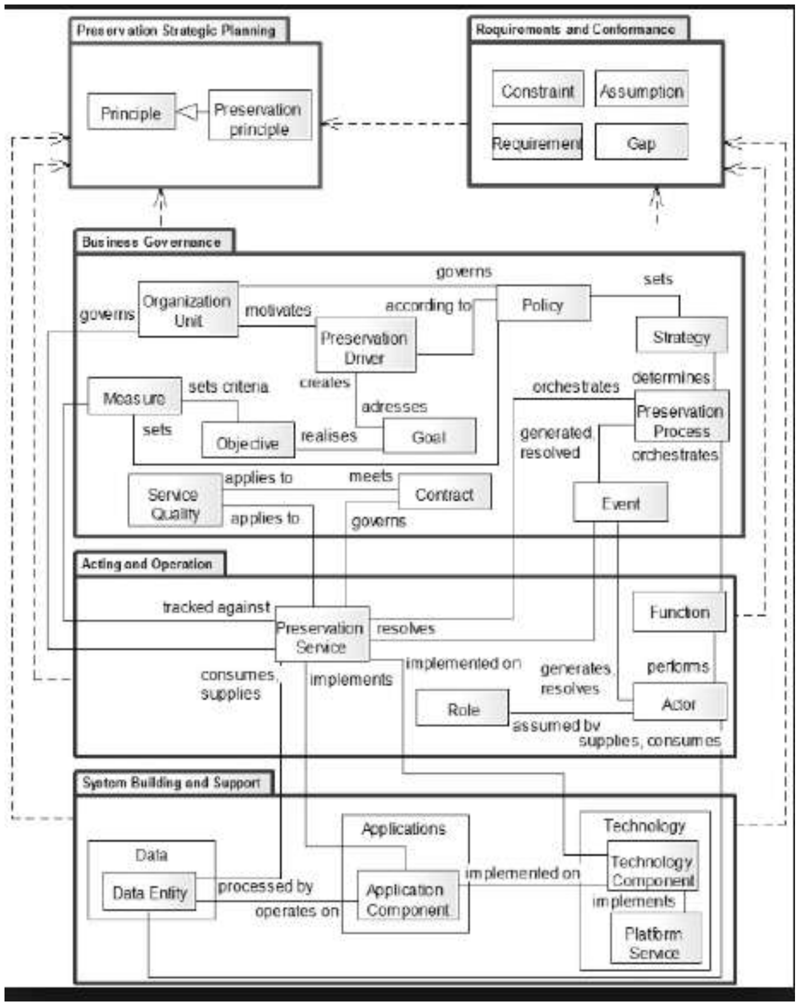


Figure 4.4.1: Generating Architecture Document

CHAPTER 5

EVALUATION

The evaluation of this concept and the actual implementation of it basically depend on few factors. Few of them are the skill and technical level of human resources who is going to test this manually and the complexity of the implemented solution which the Solution Design Monitor (SDM) is applied to generate and evaluate the design.

The main reason of this research is to identify and produce a solid concept of methodology to decrease architecture degradation of an enterprise level application so as to achieve company project level success through a continuous integration process. Many researchers have been done related to this specific area and this research will focus on an innovative ways of identifying architecture degradation. The research was done for identifying how the solution and project files (.sln and .csproj) can be used to reduce the violations of the architectural design when developing a complex system.

In addition to above information, there can be many reasons that lead to violation of the architecture design by a developer. As per this research it is narrowed down to check only for some of the pre-defined architecture components relationships are violated or not. Relationship variances between layers and projects are the fundamental rule that evaluated through the given solution to comply with design throughout the development process.

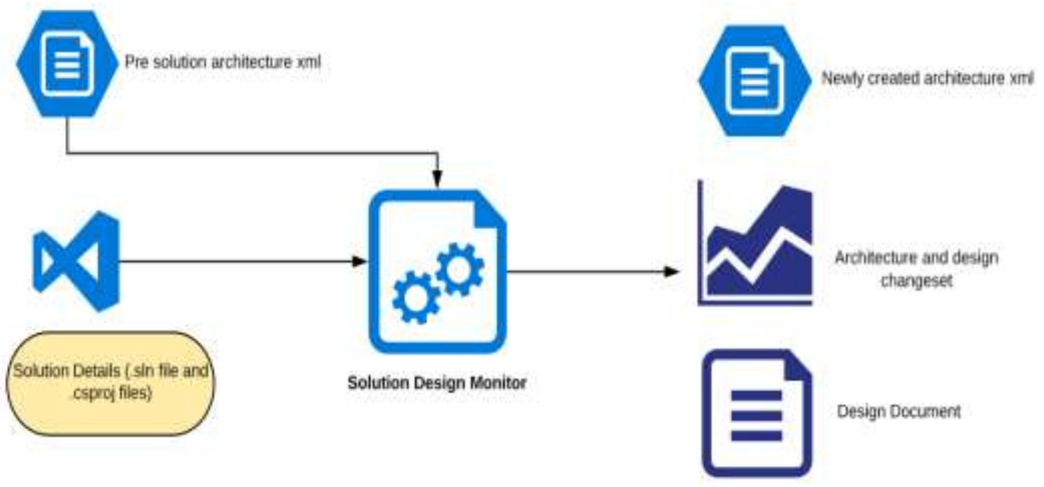

Pre solution architecture xml
Newly created architecture xml
Architecture and design changeset
Solution Details (.sln file and .csproj files)
Solution Design Monitor
Design Document

Figure 5.1: Basic Flow of the Solution Design Monitor

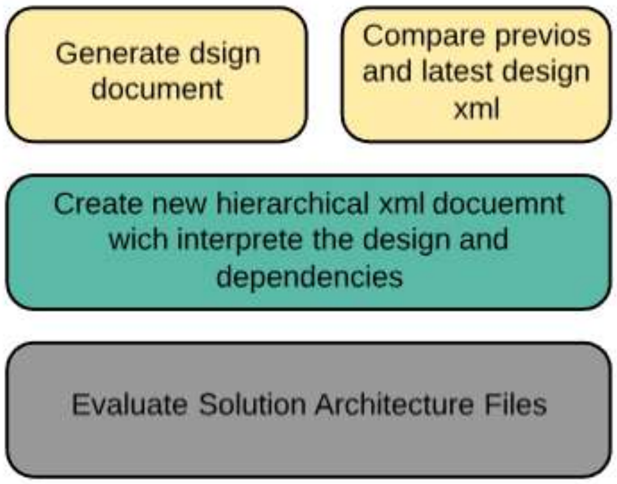

Generate dsign document
Compare previos and latest design xml
Create new hierarchical xml docuemnt wich interprete the design and dependencies
Evaluate Solution Architecture Files

Figure 5.2: Basic Components of Solution Design Monitor

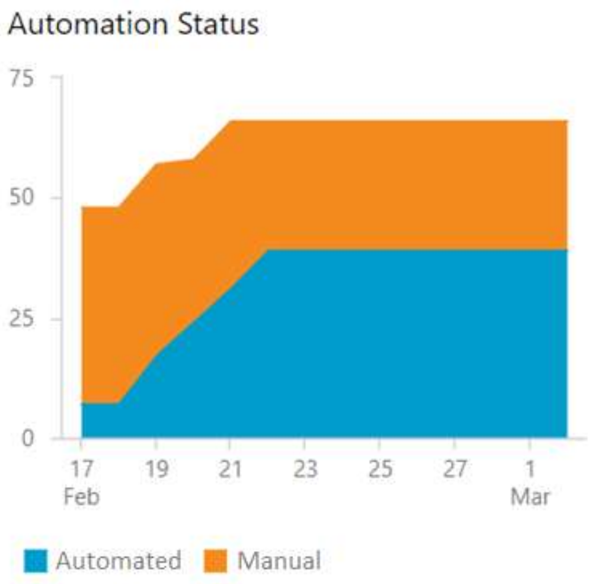


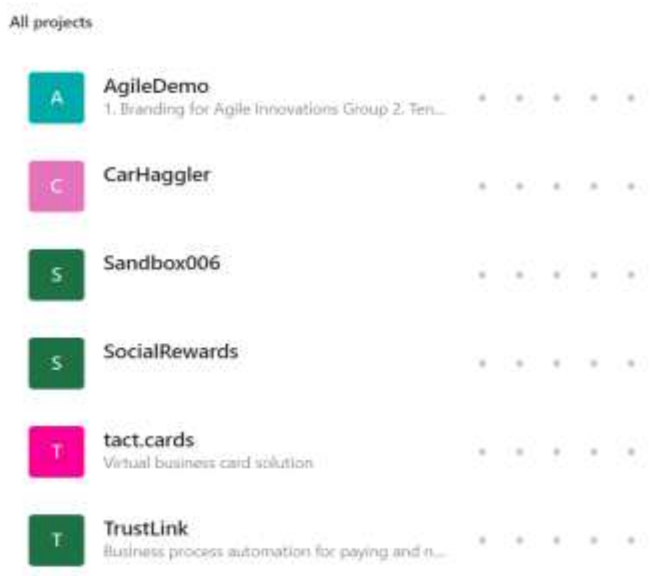
Figure 5.3: Unit Test Summary

5.1. Evaluate the Accuracy of the Analysis of Solution Architecture by SDM

For this we needs to evaluate the accuracy of the generated architecture documents by processing solution file and project files. The SDM access and process the both solution file and project files so as to generate the new xml which includes list of component and all project dependencies. Because of the fact that in SDM, we obtain and store all the specific project details, which need to compare with the actual pre-defined architecture only from the solution files, this must be evaluated and verify to continue further scenarios of architecture design validation using SDM.

5.1.1 Accuracy Validation

The solution architecture generation evaluation which was a manual accuracy validation has been done for solution files of following projects.



All projects

- AgileDemo
  1. Branding for Agile Innovations Group 2. Ten...
- CarHaggler
- Sandbox006
- SocialRewards
- tact.cards
  Virtual business card solution
- TrustLink
  Business process automation for paying and n...

Figure 5.1.1: List of Used Projects

Validation process:

- Select the set of projects.
- Explain the SDM behavior to tech leads who mainly involve in taking design related decisions of each project.
  - Changes detecting technique that have been used.
  - The design diagram that interpret all the dependencies of each projects.

- Generate the layered architecture diagram for each of projects.
- Verify the change set given by the SDM and the latest modifications.
- Manually check both change set results and the generated diagram accuracy compared to extract project architecture and latest changes.
- Get the percentage of the accuracy for the generated artifacts by the SDM from the architect or the tech lead of each project.

Project	**ETicketing**	**TrustLink**	**CarHaggler**	**SocialRewards**	**tact.cards**	**AgileDemo**
Accuracy	**80%**	**95%**	**75%**	**70%**	**85%**	**80%**

Table 5.1.1: Architecture Evaluation Results

5.2 Performance Testing of SDM

The evaluation of the performance of the implemented POC, SDM was done using set of workload factors and parameters. Those are the complexity of the solution and project architecture, the size of the code and the number of project files in the solution.

This performance testing was done in personal computers which have following configurations.

- Random Access Memory : 8.00 GB
- Processor : Intel(R) Core(TM) i7-2630QM CPU @ 2.00GHz
- Operating System : Windows
- System Type : 64-bit Operating System

5.2.1 Performance Testing By the Complexity of the Solution Architecture and Dependency Projects

For this testing project that used layered architecture has been used. The number of projects included in a solution kept constant to 5 projects. The complexity of the solution was changed by adding few additional dependencies which are relevant to the layered architecture and the some relevant to project customizations, which includes dependencies relevant to development best practices.

Average Execution Time of SDM with the Complexity of the Dependencies.

Complexity of dependencies (Number of references)	Average Execution Time (Milliseconds)
2	10
3	10
5	11
6	13
8	15

Table 5.2.1.1: Average Execution Time of SDM with the Complexity of the Project Dependencies

Here we can see that the SDM execution time tends to increase in small numbers with the complexity of the number of project included in a solution. That variation is not significant. According to the figure 5.2.1.1, it shows small liner increment and with that liner increment we can predict that the execution time will increase with the number of the projects in a solution when goes into project count with much higher complexities.

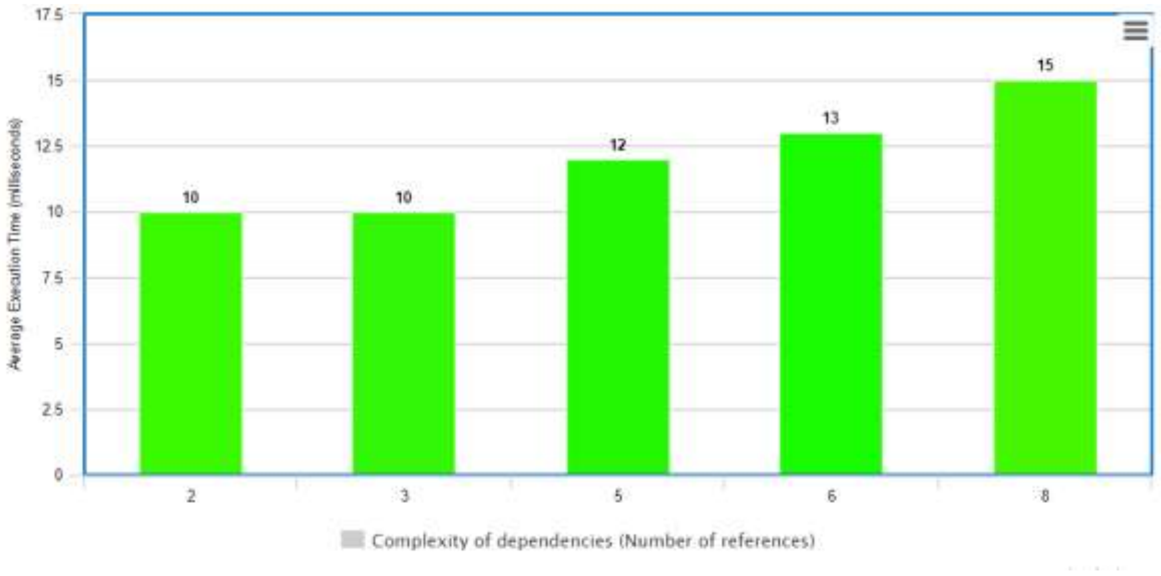


Figure 5.2.1.1: Average Execution Time of SDM with the Complexity of the Project Dependencies

5.2.2 Performance Testing By Number Of Project Included In The Solution.

Evaluating the performance of the SDM against the number of projects is very important since this kind of a monitoring tool is very useful in fairly large and complex applications. For this testing also same set of projects has been used with systematically increased code base with number of projects. SDM will load every project file under the considering solution into the memory at some point and do the analysis. So when the number of projects increased a significant amount of increase in the execution time was expected. It was not increased by a very large number but small linear increment can be observed from the data.

The test was again conducted on the scenarios where the complexity of the solution architecture is differed due to project size. This was done to check how the complexity of the project architecture affects the run time along with the increased project count. For that we used five different solution architectures.

Average Execution Time of SDM with the Complexity of the Number of Projects in a Solution.

Complexity of Solution Content(Number of Projects)	Average Execution Time (Milliseconds)
3	8
5	10
8	13
10	15
13	18

Table 5.2.2.1: Change of the Average Execution Time with the Number of Projects in a Solution.

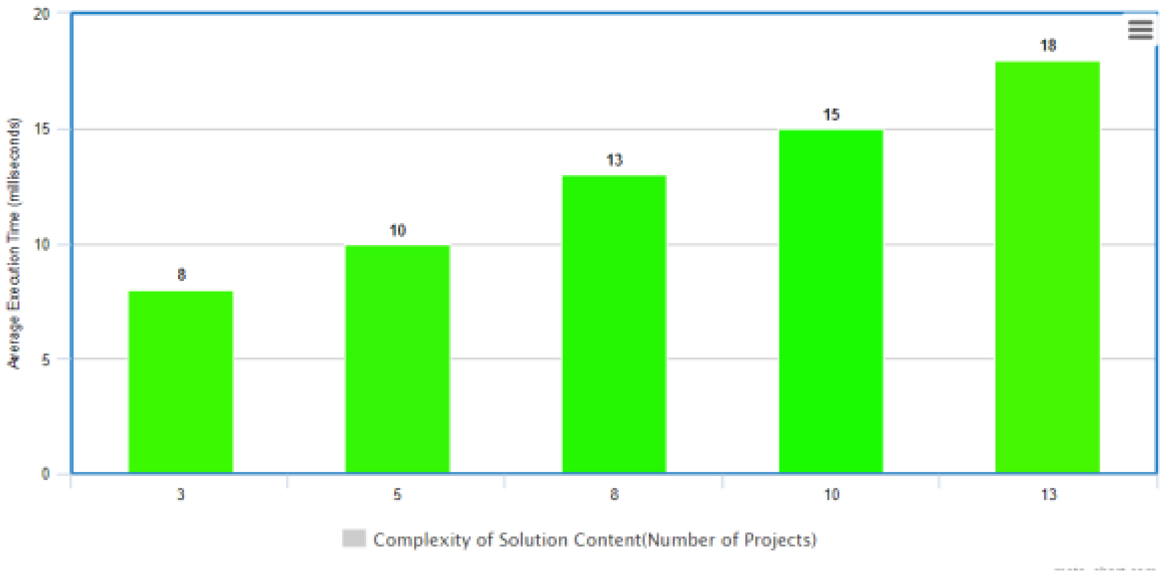
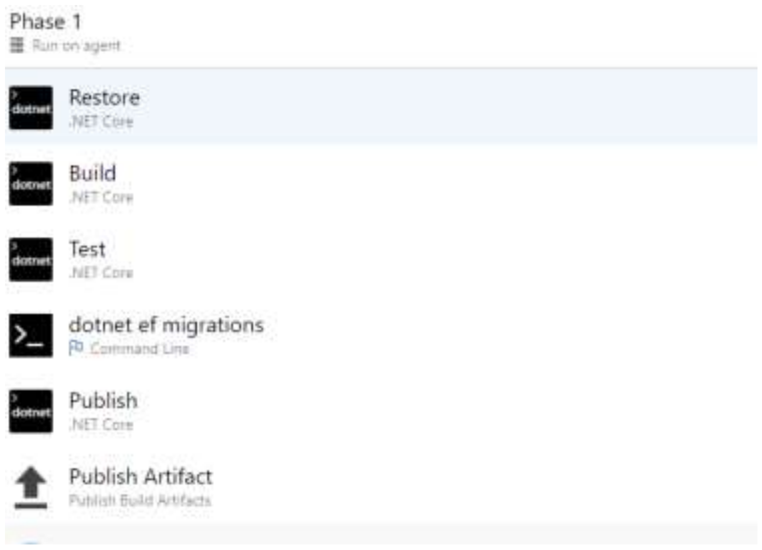


Figure 5.2.2.1: Change of the Average Execution Time with the Number of Projects in a Solution.

5.3 Analytical Evaluation of the Impact of SDM in Continuous Integration Flow

Because of the fact that SDM is developed to practically detect the architecture violations with the code base changes it is ideal to plug-in this to a proper continuous integration and continuous deployment flow. SDM is ported as an Azure plug-in and the developer can do a local analysis for his changes to the development code base. Then after the code check in to the central repository the code will again get scanned for the architectural violations in the development and release pipeline.



Phase 1
Run on agent

Restore
.NET Core

Build
.NET Core

Test
.NET Core

dotnet ef migrations
Command Line

Publish
.NET Core

Publish Artifact
Publish Build Artifacts

Figure 5.3.1: Build Pipeline before Adding SDM

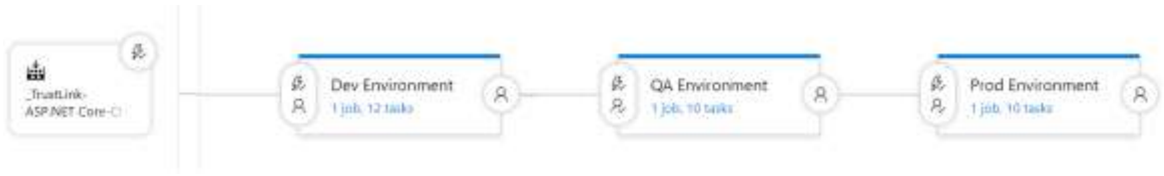


Figure 5.3.2: Release Pipeline before Adding SDM

Before SDM is introduced to the continuous integration flow it was like in the figure 5.3.1 and figure 5.3.2. There the evaluation of code done manually by a tech person and this process might take hours and days with the availability of the reviewer resulting long waiting time to deploy the code in pre-production environment. Results of that review also depends on the technical and development language skill level of the reviewer and his way of the understanding about the architectural design through codebase of the system, but this solution could overcome the person skill level of evaluating design architecture in a minimum time.

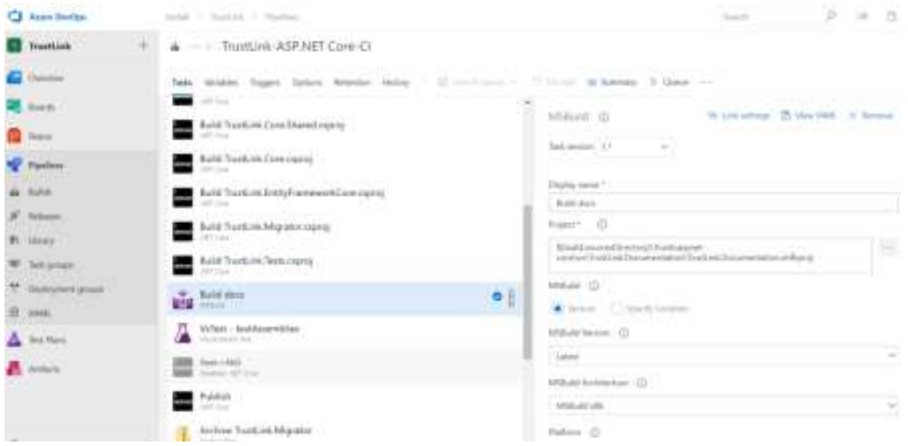


Figure 5.3.3: Build Pipeline after Adding SDM

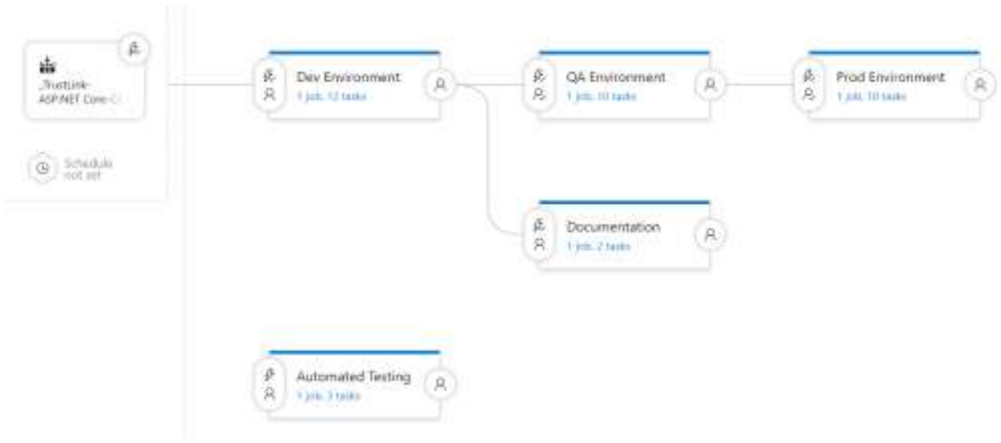


Figure 5.3.4: Release Pipeline after Adding SDM

After Introducing the SDM into pipeline (Figure 5.3.3 and Figure 5.3.4), the manual work of architectural design reviewing can be enhanced or completely removed. SDM have a comprehensive way of checking and identifying whether the pre-architecture design match with the modified and newly created architecture. It won't have the problems like when it is done by another reviewer which is a greater advantage of SDM.

CHAPTER 6

CONCLUSION

In conclusion, DevOps is a new philosophy that can helps software organizations to innovate faster and to be more intrinsic to business needs since it promotes collaboration between developers and operations which improves quality of software development and more frequent software releases. Since there are shorter development iterations, it is more important to monitor the software architecture without leading to a software drift or erosion. And this research objective is to build a Monitoring framework for architectural degradation in DevOps practice.

Development of a maintainable and durable software solution is a challenging task for any development organization. Achieving nonfunctional quality goals such as protecting the system architecture throughout the software development cycle, and protecting the organization wise quality standards and fast time to market or deployment is directly affecting the above mentioned goal. There have been many researches carried out towards this area focusing on developing concepts and methodologies to preserve the systems by retaining and properly managing the non-functional quality goals.

6.1 Research Contribution

This research basically focuses on detecting an approach to evaluate the prescriptive architecture using solution and project files and the changes done by the developers throughout the development life cycle. Using this approach toward an organization, it can develop a standard to follow throughout implementation process within the organization, and that includes developers continuously verifying the code changes to align with the high level project design architecture without any hazard violations. Then they have to develop a way which processes the code base against the prescriptive architectural information. This monitoring framework can be applied in to the systems for identifying changes throughout the development process. The predefined prescriptive architecture information can also be configured through in file which is XML format.

The implemented proof of concept is the Solution Design Monitor (SDM). As a version of POC, SDM create a solution and project specific XML file with an observed set of information which contain all the layers, components and its relationships. There is an component analyzer to analyze the design architectures and compare it with the predefined architecture design. Then it produces the results (Charts, Graph and documents) which can be used in the decision making process. This POC was evaluated both automated and manual ways to prove that this kind of a simple concept can actually work in real software development scheme.

Furthermore, using this concept in a real critical and complex software project is a challenging task up to some level. The reason is not the technical barriers (some testing was done for large and complex project also so that it can actually be done using a SDM like monitoring tool) but because of the needed changes for software development life cycle in reality is critical. If an organization willing to adopt this kind of approach for a project then they firstly need to create design architecture and the relational mapping in to an xml file. Then derive an architecture design set to be checked or define the architecture style invariants, and then identify how to enforce those design mapping within the xml document. That is basically defining the required dependencies and components where to use them in order to provide the codebase changes into architecture document with additional information. After that each and every member of the development team needs to adhere to those guidelines through the development phase. If an organization can preserve on above it is obvious fact that these concept are in place and they can actually get the real benefit applying into development life cycle.

6.2 Research Limitations

Identified few limitations for this are this research concept and the implemented POC was not tested in extreme conditions such as in real life complex software applications in organizations. In order to test this in real environment and conditions above mentioned barriers need to be cleared. And also a considerably massive team of development is necessary. The individuals of development team members need to be trained so as to understand the scenarios that needs to be done and how to properly use this monitoring framework. Getting actual progress over considerable time duration, we need to use this POC in a project team for a fairly long time such as months. That is not feasible because of the fact that we have to align with the time constraints of the module. For this research, only the .net solution files were considered. Since most developers and organizations use .Net framework as the preferred programming language for their development tasks due to its ease of use. Therefore these research concepts were obliviously limited to the resources and tools which supports only for the .Net.

6.3 Future Work and Conclusion

In this research an implemented fundamental tool called Solution Design Monitor (SDM) was developed as a proof of concept. It can be improved in few ways to add extra benefits for using this monitoring approach. If we can establish a pipeline job for this, then it would be much easier for the developers to assess the design with each build with auto generated document for each code check in.

Furthermore the future work related monitoring framework, precise to each project with the prearranged architecture then it would be much easier to use this approach in real life complex projects. Since Solution Design Monitor uses the solution files of a software project, a devoted framework can actually insert some of the default architectural design information into the code base so the work needs to be done by the developer will be condensed.

At the present stage, this SDM monitoring framework support projects written in .Net framework only. This framework can be extended to evaluate other implementation languages like java, python, php and etc. The architecture design capturing part and the evaluation needs to be done in that language specific project files.

This research was intended to find out the feasibility of identifying software architecture degradation through basic project artifacts that available for all projects. Using a method which assess the solution files and project files and then use a predefined design document of components and relationships to compare the high level design architecture with the continuously changing architecture was a successful one. According to the evaluation results, we can conclude that augmenting the project information which are less complex such as solution files and project files can actually be used to decrease the architecture degradation up to a considerable extend.

REFERENCES

[1] Ducasse, S. and Pollet, D., 2009. Software architecture reconstruction: A process-oriented taxonomy. *IEEE Transactions on Software Engineering*, *35*(4), pp.573-591.

[2] Murphy, G.C., Notkin, D. and Sullivan, K.J., 2001. Software reflexion models: Bridging the gap between design and implementation. *IEEE Transactions on Software Engineering*, *27*(4), pp.364-380.

[3] Postma, A., 2003. A method for module architecture verification and its application on a large component-based system. *Information and Software Technology*, *45*(4), pp.171-194.

[4] Erich, F., Amrit, C. and Daneva, M., 2014. Report: Devops literature review. *University of Twente, Tech. Rep*.

[5] Burd, E.L., Chan, P.S., Duncan, I.M.M., Munro, M. and Young, P., 1996. Improving visual representations of code. *University of Durham, Technical Report*.

[6] Zhang, H.Y., Urtado, C. and Vauttier, S., 2010, July. Architecture-centric development and evolution processes for component-based software. In *Proc. of 22nd SEKE Conf., Redwood City, USA (July 2010)*.

[7] Ayewah, N., Pugh, W., Morgenthaler, J.D., Penix, J. and Zhou, Y., 2007, June. Evaluating static analysis defect warnings on production software. In *Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering* (pp. 1-8). ACM.

[8] De Silva, L. and Balasubramaniam, D., 2012. Controlling software architecture erosion: A survey. *Journal of Systems and Software*, *85*(1), pp.132-151.

[9] Che, M. and Perry, D.E., 2014, April. Architectural design decisions in open software development: A transition to software ecosystems. In *2014 23rd Australian Software Engineering Conference* (pp. 58-61). IEEE.

[10] Shahin, M., 2015, September. Architecting for DevOps and Continuous Deployment. In *Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference* (pp. 147-148). ACM.

[11] Garlan, D. and Schmerl, B., 2004, May. Using architectural models at runtime: Research challenges. In *European Workshop on Software Architecture* (pp. 200-205). Springer Berlin Heidelberg.

[12] Stark, G., Skillicorn, A. and Ameele, R., 1998. An examination of the effects of requirements changes on software releases. *CROSSTALK The Journal of Defence Software Engineering*, pp.11-16.

[13] Chen, L., 2015, May. Towards Architecting for Continuous Delivery. In *Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference on* (pp. 131-134). IEEE.

[14] http://www.uml-diagrams.org/multi-layered-application-uml-model-diagram-example.html

[15] Hammer, D.K. and Ionita, M., 2005. SCENARIO-BASED SOFTWARE ARCHITECTURE EVALUATION METHODS An overview.

[16] Van Gurp, J., Brinkkemper, S. and Bosch, J., 2005. Design preservation over subsequent releases of a software product: a case study of Baan ERP. *Journal of Software Maintenance and Evolution: Research and Practice*, *17*(4), pp.277-306.

[17] Williams, B.J. and Carver, J.C., 2010. Characterizing software architecture changes: A systematic review. *Information and Software Technology*, *52*(1), pp.31-51.

[18] Knodel, Jens, Dirk Muthig, and Matthias Naab. "Understanding Software Architectures by Visualization--An Experiment with Graphical Elements." *2006 13th Working Conference on Reverse Engineering*. IEEE, 2006.

[19] Murphy, Gail C., and David Notkin. "Reengineering with reflexion models: A case study." *Computer* 30.8 (1997): 29-36.

[20] Hochstein, Lorin, and Mikael Lindvall. "Diagnosing architectural degeneration." *28th Annual NASA Goddard Software Engineering Workshop, 2003. Proceedings.*. IEEE, 2003..

[21] Knodel, Jens, et al. "Static evaluation of software architectures." *Conference on Software Maintenance and Reengineering (CSMR'06)*. IEEE, 2006.

[22] Bandara, Vidudaya, and Indika Perera. "Identifying Software Architecture Erosion Through Code Comments." *2018 18th International Conference on Advances in ICT for Emerging Regions (ICTer)*. IEEE, 2018.

[23] Meyer, Mathias. "Continuous integration and its tools." *IEEE software* 31.3 (2014): 14-16.

[24] Smith, Tony. "Protecting the process [source code management]." *Engineering & Technology* 5.4 (2010): 51-53.

[25] Venkitachalam, Hariharan, et al. "Automated Continuous Evaluation of AUTOSAR Software Architecture for Complex Powertrain Systems." *INFORMATIK 2017* (2017).

[26] Vasilescu, Bogdan, et al. "Quality and productivity outcomes relating to continuous integration in GitHub." *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 2015.

[27] Kim, J. and Garlan, D. (2010). Analyzing architectural styles. Journal of Systems and Software, 83(7), pp.1216-1235.69

[28] Monroe, Robert T., et al. "Architectural styles, design patterns, and objects." *IEEE software* 14.1 (1997): 43-52.

[29] Mehta, Nikunj R., and Nenad Medvidovic. "Composing architectural styles from architectural primitives." *ACM SIGSOFT Software Engineering Notes*. Vol. 28. No. 5. ACM, 2003.

[30] Vatanawood, Songpon Thongkumand Wiwat. "An Approach of Software Architectural Styles Detection Using Graph Grammar."

[31] Katte Darshan, A., and P. Suganya. "Json is Efficient over the XML in Native Application." *International Journal of Computer Applications* 165 (2017): 545-586.

[32] Pandey, Mrinal, and Rajiv Pandey. "JSON and its use in Semantic Web." *International Journal of Computer Applications* 164.11 (2017): 10-16.

[33] Kövesdán, Gábor, Márk Asztalos, and László Lengyel. "Architectural design patterns for language parsers." *Acta Polytechnica Hungarica* 11.5 (2014): 39-57.

[34] Lyon, Douglas A. "Semantic annotation for Java." *Journal of Object Technology* 9.3 (2010).

[35] Buitrago Flórez, Francisco, et al. "Changing a generation's way of thinking: Teaching computational thinking through programming." *Review of Educational Research* 87.4 (2017): 834-860.

[36] Mei, Hong, and Jun-Rong Shen. "Progress of research on software architecture." *Ruan Jian Xue Bao(Journal of Software)* 17.6 (2006): 1257-1275.

[37] Shaw, Mary, and David Garlan. *Characteristics of Higher-Level Languages for Software Architecture*. No. CMU-CS-94-210. CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1994.

[38] Weyuker, Elaine J., and Filippos I. Vokolos. "Experience with performance testing of software systems: issues, an approach, and case study." *IEEE transactions on software engineering* 26.12 (2000): 1147-1156.

[39] Marshall, Adam C. "A Conceptual model of software testing." *Software Testing, Verification and Reliability* 1.3 (1991): 5-16.

[40] http://theagileadmin.com/what-is-devops/

[41] https://insights.sei.cmu.edu/devops/2015/01/continuous-integration-in-devops-1.html

[42] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.51.5526&rep=rep1&type=pdf

[43]https://www.thinkhdi.com/~/media/HDICorp/Files/White-Papers/whtppr-1112-devops-kim.pdf

[44] http://www.sparxsystems.com/enterprise_architect_user_guide/9.2/software_engineering/reverseengineersourcecode.html

[45] http://www.codingthearchitecture.com/2014/06/24/software_architecture_as_code.html

[46] http://www.architexa.com/

[47] http://www.stack.nl/~dimitri/doxygen/

[48] http://softarch.usc.edu/projects/automated-recovery-of-software-system-designs/