Software Architecture in Web Development

Mikhail Gospodarikov
University of Greenwich, United Kingdom
mail@mig0s.tk
Website: http://mig0s.tk
GitHub: https://github.com/mig0s
LinkedIn: https://sg.linkedin.com/in/mig0s

Abstract

This article intends to present a technology review providing a critical and unbiased evaluation, perspective, and opinion based on scholarly reviews of fundamental concepts of software architecture and stages of software development which are preceded before the actual implementation of the application.

World Wide Web is a great environment for creating secure applications rich in functionality, capabilities and performance. The Web is the only platform that offers ultimate range of supported client platforms; all you need is a browser, whether you have an Android, Linux, Mac, Windows or any other operating system and internet connection – the web welcomes you.

The software made for web is able to perform gracefully regardless of form-factor or screen resolution of client devices, but it takes a certain level of expertise and skill to develop truly good applications and keep up with new trends and standards. The article covers the very core of a software product – its architecture.

Portions of this article will be reproduced and submitted in the Undergraduate Project Report form in fulfilment of the requirements for the BSc Hons Computing Degree for Mikhail Gospodarikov from University of Greenwich.

Purpose

The purpose of this article is to explore and inform software developers about the methodologies applied in the University of Greenwich's Computing degree project with an educational industry project work. This article intends to give a definition of Software Architecture; highlight its importance; identify stakeholders of architecture and discuss their needs; present requirements to architecture comparing industrial and academic approaches; as well as discuss benefits of use of architecture in software engineering. The article presents the narrative review in the context of Web Development.

Research methodology

The author used a narrative literature review technique to search the relevant online databases by using pre-defined keywords. This article is based on relevant information which was extracted from the retrieved articles and books, and the ensuing discussion focuses on providing the reasons for the use of software architecture as a basis for choosing software framework and development pattern for the industry based project work.

Findings

From the online searches across journal articles and books studies that matched the criteria were retrieved, and the information extracted from each resource includes the authors, publication year, assessment method used, target audiences, coverage of project and industry based project goals.

Practical Implications

This article is the author's perspective review offering opinion emphasising the significance of software architecture and frameworks in developing Web Applications that bring value to clients through performance, secure, well-structured and well-documented maintainable solutions satisfying various business needs.

Originality

The review of the retrieved articles indicated that no previous software production was implemented in this educational institute with use and/or consideration of architectural patterns as predecessors of designing and implementing a web application.

Keywords: *web application development, PHP, MVC, Yii2, MySQL, software architecture, relational database.*

1. INTRODUCTION

"Good design is good business" - *Thomas J. Watson Jr. (a core belief of IBM Studios)*

This article is focusing on software architecture in context of cross integration of multiple computer systems, specification of system components and its relations, as well as compares academic and industrial view of software production.

Having this endless field of opportunities such as the Web however, requires decent knowledge of software development architectures and techniques. As systems grow and evolve, comes a need for integration of data, services and APIs into different environments such as subsystems (i.e. Customer Relation Management into Enterprise Resource Planning) and platforms (i.e. mobile interfaces or applications). Without prior expertise and structure implementation of such enhancements becomes extreme in time, resource and effort consuming.

This review discusses software engineering architectural pattern known as Model-View-Controller (MVC), its components and interactions. In this part, the following important features of a software project are discussed in details: object-oriented code and relational data structure.

2. THE DEFINITION OF SOFTWARE ARCHITECTURE

The definition of Software Architecture can be described as a discipline of creating high level structures of a software systems and its documentation. It is the overview and structure, like the outline plans and sketches of a large building. (Lunn, 2003) Basically, it is a set of blueprints for a software project using which you can see how everything is put together in a software system, whether it is web-based Customer Relationship Management System, standalone Enterprise Resource Planning System or an embedded POS. Software architecture is an important part of Information and Information Technology Planning and Management in an organization; it “represents a concrete artefact that can be used for discussion with and between stakeholders” (Bosch, 2000).

However, it is not only a tool for translating the IT magic to the management and non-technical stakeholders, but also a disciplined guidance for developers.

3. WHY SOFTWARE ARCHITECTURE IS IMPORTANT

“Programming without an overall architecture or design in mind is like exploring a cave with only a flashlight: You don’t know where you’ve been, you don’t know where you’re going, and you don’t know quite where you are” - Danny Thorpe

Since the mid-50s of the past century the concept of software architecture has been widely and vigorously debated in the community of IT professionals. It was due to the fact that architecture was initially considered only as a necessary basis for information systems. At the moment the software architecture could be understood as a purely technical concept, i.e. a set of software and hardware modules optimally interacting with each other using predefined links and protocols to ensure efficient operation of a certain system. And the systems served for narrow tasks such as scientific and military purposes.

Over time, with drastic revolutionary development of the IT sector, subsequently scaling on all areas of human activity, the software architecture has entirely become a product of software developers and started to be considered as a part of Software Development Life Cycle (SDLC). This happened because there were less and less resources given to develop software, and certain stages were spliced in one, and results were expected earlier.

The modern stage of human history teaches our nature to desire constant reflexion and rethinking of our current position. As a result of this process, there should be an understanding that with no architectural basis for further maintenance, support and evolution of a software product, requirements from such system shall not be fulfilled, and business expectations shall never be satisfied.

Architectural design was conceived as a scientific discipline. Without it, it was impossible to start writing algorithms and implement business logic. After all, the software product acquired the necessary shape and formed into a complete system in the minds of elected engineers at this exact stage.

The current situation on the IT market is determined by entirely different conditions. Engineers, developers, analysts do not have so much time and “human capital” to develop a product entirely by the book. We are in a state of constant time pressure when the success of a project is determined by the time it hits the market, but not by the quality of the solution itself, and there are no prerequisites for changing these principles of the market.

In such environment, architecture helps to generate competitive advantage of a product, not only due to rapidity of its creation, but due to reasonable and optimal structure of entities, its capabilities and interactions, which makes further development process flexible and the product – maintainable, without compromising time, quality or functionality.

4. STAKEHOLDERS OF SOFTWARE ARCHITECTURE

Architecture should harmonize all needs of software product stakeholders. The purpose of architectural software is to meet the complex needs of conflicting groups of the most important stakeholders. Very often it is rather difficult to fulfill all the requirements. Reaching a compromise solution that will suit the majority of stakeholders is one of the key aspects of a successful architectural design.

We can identify the following groups of stakeholders and their needs:

I. ***End users:*** interested in an intuitive interface, logical and predictable behavior, performance, reliability, usability and accessibility.

II. ***System Administrators:*** interested in an intuitive and logical functions, simple management and monitoring tools.

III. ***Marketer / Product Promotion Specialists:*** interested in a competitive software product features, its fast time to market, positioning and allocation among other products.

IV. ***Support Specialists:*** interested in a clear, unambiguous, and well-documented maintainable and changeable product.

V. ***Developers:*** interested in clear and simple requirements, a consistent design principle and/or paradigm.

VI. ***Clients:*** interested in adequate price, stability, return on investment, etc.

Many of the requirements stated above are non-functional in nature, as they do not directly affect the functionality of the system (easy maintenance). Thus, such requests define capabilities and limitations of a system expressed in ways the product should fulfill its purposes, but it is not the essence of this purposes. The main stakeholder of an architecture is the architect of the system who is responsible for designing and implementing the right solution.

5. REQUIREMENTS TO THE ARCHITECTURE

“Marry your architecture in haste, repent at leisure”- Barry Boehm

First and foremost, a software architecture should be easy to understand. The set of requirements should be simple and accessible for any employee involved in a project at any stage of the lifecycle. If it is hard to explain the idea, there is a possibility that someone in your team will silently nod in agreement, but will not do the right thing later.
The requirements to certain components should be easy to use. If the requirements are not detailed a level which allows its use without extra attention and thinking, we risk that the application will not be developed as designed. The differences between theoretical

schemes and design and real structure of the implemented software may cause misunderstanding between the development team members and users.

The set of requirements in an architecture should provide attributes and functions acceptable by the client. It is truly an important point if the system is taking care of online work among thousands of employees. If something does not work in such a system, the architecture is considered unsuccessful.

The architecture should not be only simple enough for the client to understand, but also represent a clear map for components implemented and used to make sure that the developers are comfortable with further development of the product.

The development team should agree on the architecture used. If there is a person who stands against suggested implementation approach (someone who, for instance, does not like or does not have experience with the database engine selected), and if you not localize his or her influence on the development process by making this person follow everyone else's way, the project may end up in a disaster.

Common principles should be used for all professional activities within a certain project and certain technology the team works with. If a decision is made, it should be followed everywhere and by everyone. This principle is not only applicable for defining requirements but also to many processes in the domain of information technologies. If a decision is not followed or its use causes difficulties, the reason for it should be found. If the situation is truly exceptional, the rule (decision) should be revised and reproduced in a way that covers such exceptions. However, it is better (and easier) to avoid such complications.

Thus we come to the universalization of the operating principles, and we can gradually achieve a complete architectural design for an organizational system, but it is important to remember that in any case you should know the limits. The system must be flexible and adaptive, but solid in its core. This will sustain a lot of tests, coming in a variety of tasks during the creation of software architecture. A rule should be a rule. If it does not work - it will be bad for all the stakeholders.

In the development of all types of requirements for an architecture of a software it is also necessary to consider the organizational and functional structure of the companies / divisions for which the software is being developed. The architecture of the product provided should not contradict the essence of the company's workflow, the structure of cooperation in the implementation of software development processes, work practices, etc. If the automated process of software product creating will involve other departments, divisions or a company, responsibilities and areas of influence of modules of the system should be clearly and unambiguously defined.

6. ACADEMIC AND INDUSTRIAL VIEWS ON SOFTWARE ARCHITECTURE

If you look at traditional software development done by a company with various clientele, you may find that such firms produce one system at a time due to such limitations as lack of manpower to fulfil all the roles in a project; production line is focused on delivery and evolution is not considered. The priority here is given to such issues as time, budget, maintenance cost and decreasing competitiveness. The quality of the software and documentation produced is not as important as getting the job done. The product here has to hit the market on time with requested features at minimal cost.

Hence, we can highlight four main objectives of a software project:

I. Development cost
II. Time-to-market
III. Maintenance cost
IV. Quality

Industrial view of software architecture is mostly conceptual with minimal explicit definition and no first-class connectors for runtime binding and glue code for adaptation assets. Programming languages (e.g., Java, C++) and script languages (e.g., Make, JavaScript, PHP) used to describe the configuration of the complete system (Bosch, 2000). The industry focuses on the solution for a certain set of issues itself, and on the first three objectives highlighted above.

On the other hand, there is an academic view where architecture is explicitly defined, consists of components and first-class connectors for project assets. From this point of view, Architectural Description Languages (ADLs) describe architectures and are used to automatically generate applications (Bosch, 2000). This attitude is focused on quality and purity of the solution and has no cost or time constraints.

According to Jan Bosch's comparison we may think that academic approach seems to be "the right way" of software development which provides clear structure of a system and delivers quality. However, this approach is not applicable in business world where such constraints as money and time exist. To continue with what has been said, there is no limit to perfection, and an academic project may never have a release, but industry projects often compromise quality. Hence, we should find the right balance between these two approaches that fit our requirements and resources.

7. SOFTWARE COMPONENTS AND FRAMEWORKS

To reduce time and cost of development, software development firms and individual developers may maintain a repository of components or use software frameworks. Such way to store ready-to-use source code provides an opportunity to reuse software units written within a project over subsequent versions, in other projects, and in different organizations. The first two levels of source code reuse are more suited for component-based development within a company, and frameworks may fit a wider range of requirements from different companies for back-end (e.g., Microsoft Foundation Class Library, Oracle Application Development Framework) and front-end (e.g., Qt, Twitter Bootstrap).

Jan Bosch compares academic and industrial views of reusable components as follows: reusable assets are black-box components in academic approach, these assets have a narrow interface through a single point of access, they have a few and explicitly defined variation points which are configured during installation, and standardized interfaces implemented, so they can be traded on component markets. Academia focuses on high reuse of asset functionality and on the formal verification of functionality.
The industry, however, consider reusable assets as large pieces of software with a complex internal structure and no enforced encapsulation boundary (e.g., object-oriented frameworks). Interfaces of such assets are provided through classes and

models of the asset, and they have no significant differences to non-interface entities. Variation requirements in the industry are covered by replacement of entities in the asset, and sometimes a component has multiple versions used for different purposes. Due to high possibility of discovering differences in standards, reusable assets are often being adapted to match existing product-line requirements. Quality attributes such as performance, reusability, maintainability, reliability, and functionality have equal importance here.

Academic and industrial approaches may also be represented as component-based approach and framework-based approach. Same as in the previous section, there is no winner and software production unit has to determine the right fit for their business. To do so, it is important to understand the key advantages of each approach.

Component-based development is a long-term strategy that increases the up-front cost of development, with the pay-off once components are repeatedly reused. Production process is split into component development and application assembly (Lunn, 2003).

Framework-based development has common aspects, functions and features of applications grouped together in abstract classes, thus decreasing the time and cost needed for development, as well as increasing quality of code, and helps development novices to improve their skills and knowledge by forcing them to write proper code that fits framework's logic and paradigm. Figure 1 roughly shows the amount of code needed to implement new feature during the time in a project based on reusable components (or a OOP framework), and Figure 2 shows the same on a project with no reuse (procedural approach). On these graphs you can clearly see the advantage of generating reusable code. As you use framework like Zend of Symfony, it is rather tedious to start with basic functionality for your application, but through the time being, the amount of code significantly decreases due to solid base of existing code, models, objects, controllers, etc. that you can reuse.

Frameworks and components play one of the primary roles in creating a good software architecture for enterprise applications, side by side with standardization. An application may be standalone, but more often it is a subsystem, or a part of a bigger system, or has subsystems within. These systems are integrated with each other and exchange data transactions, so if there will be no standard for data structure and components, the system will not perform well, it will be tedious to reuse data from one subsystem in another, the application will be hard to maintain. Thus, it will lack certain functionality, and any extent may compromise existing functions and even cause a reason to reengineer the whole infrastructure of enterprise software.

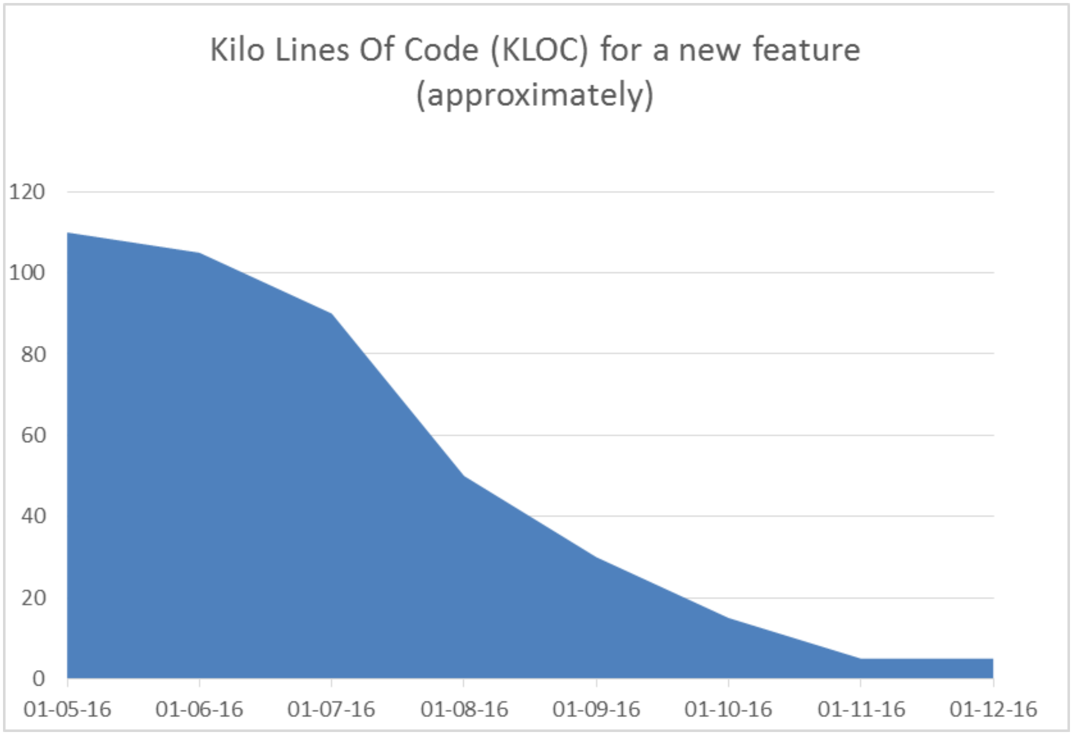

Figure 1: approximate amount of KLOC needed for a new feature (project based on reusable code)

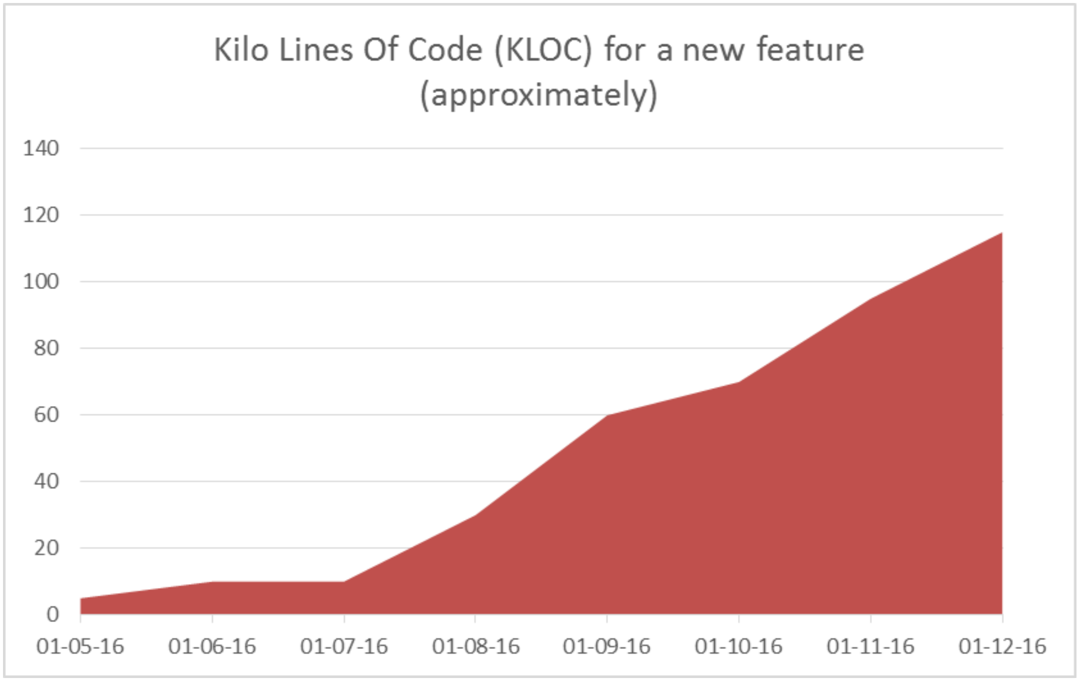

Figure 2: approximate amount of KLOC needed for a new feature (no reusable code)

8. BASIC INFRASTRUCTURE AROUND WEB APPLICATIONS

Now, we discuss the infrastructure around web applications which consists of client and server parts, thereby realizing the "client-server" technology. The client part implements a user interface, generates requests to the server and then processes the responses from the client. The server part receives a request from a client, performs the calculations, then creates a web page and sends it to the client over a network using HTTP protocol.

The web application itself is able to act as a client to other services, such as a database or another web application which might be located on another server. A striking example of a web application is MediaWiki - a Content Management System for Wiki articles: a plurality of participants can take part in the creation of a network encyclopedia, using the browsers of its operating system (whether it is Microsoft Windows, GNU / Linux or any other operating system) and without downloading additional executable modules work with the database of articles.

Currently a new approach to web application development called Ajax is gaining popularity. When you use the Web application Ajax does not reload entire pages, but only pulls the necessary data from the server, making the pages interactive and the work float productive, reducing the time needed to go between pages.

Also recently gaining increasing popularity Web Socket technology that does not require constant requests from the client to the server, and creates a bi-directional connection where the server can send data to the client without a request from the latter. Thus it is possible to dynamically control the content in real time.

To create a web application on the server side, there is a variety of technologies that we as developers can use with any programming languages or server platforms (PHP, Node.JS) which can give output to the standard console or a browser.

9. MVC AS THE CHOICE ARCHITECTURE FOR WEB APPLICATIONS

Moving forward to justification of selected structure for our application, let us review current trends in web development with PHP. Figure 3 provided by SitePoint in 2015 represents people’s preference in frameworks, and we can clearly see, most of the popular choices are MVC frameworks, and they overweight such options as “We use a CMS for everything”, “No Framework” and “Company Internal Framework” which may or may not be based on MVC pattern. According to w3techs.com, “*PHP is used by 82.1% of all the websites whose server-side programming language we know.*” Thus, the absolute majority of the web industry professionals are using MVC and frameworks.

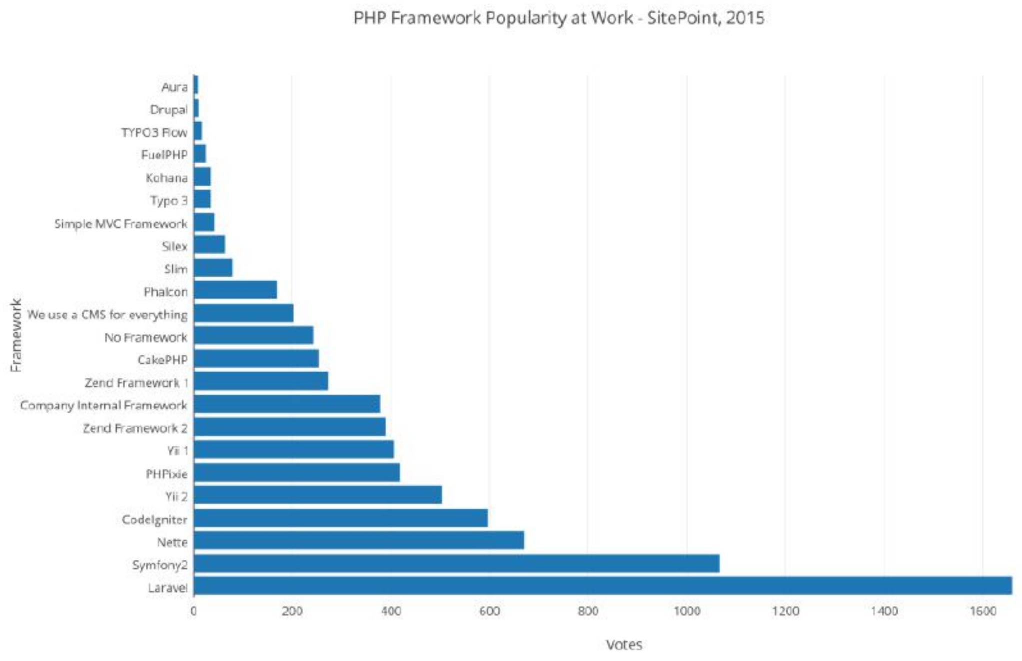

Figure 3: PHP Framework Popularity at Work – SitePoint, 2015

The Model-View-Controller pattern was used in software engineering for a long while. Back in 1979 it was described by Trygve Reenskaug “as an obvious solution to the general problem of giving users control over their information as seen from multiple perspectives” in “Models-Views-Controllers”. MVC has created a surprising amount of interest, says Trygve in his work “The original MVC reports” dated 2007. In the present MVC architecture has become very popular among web developers.

The idea behind the structure of MVC template is rather simple: you need to clearly identify responsibilities for different operations performed by your application. In MVC, the application is divided into three main components, and each of the components is responsible for various tasks. Models are responsible for interactions with data and contains business logic. Views are used as a presentation layer for presenting data to users in any format compatible with the client-side environment (i.e. HTML, XML, JSON). Controllers handle user requests and calls for certain resources and services of an application. Thus, MVC is structuring the components of an application clearly.

Let us review these components in depth:

<u>Controller</u>, as stated before, manages user requests (received as HTTP (HTTP/2) GET or POST requests which are generated when user clicks on the elements of an interface to perform various actions. Its main purpose is to call and coordinate the actions of requested resources and objects needed to perform actions specified by the user. Typically, the controller interacts with an appropriate model, passing or retrieving data and/or commands (create, read, update, delete) and provides an appropriate view for the output.

<u>Model</u> is a container of rules and functions used to manage data, which represents the concept of application management. In any application, the entire structure is modeled as data that can be processed in a certain way. What data do you give to an application – a message or a book? It takes only the data described in pre-defined rules (i.e. the date of publication cannot be in the future, e-mail cannot contain more than one "@" symbol, a name of the author cannot be longer than N symbols, and so on).

The model gives a certain representation of data according to user's request to the controller (i.e. a message, a page of a book, personal details of an author and so on). The model of this data will be the same regardless of ways we want to present it to a user. Therefore, we may choose any of the available views for displaying this data.

Model contains the most important part of the logic of an application – logic that provides a solution to designed function we are currently dealing with (a forum, a shopping cart, a list of reviews, etc.), and has validation rules that have to be followed during input validation. The controller, however, is more like housekeeping – handles basic organizational logic of our application.

View provides various ways to display the data obtained from the model. It may be a template that is filled with data. There may be several different types of templates and the controller selects the most appropriate view for the current situation. Often, frameworks use a certain view generator of a template engine such as Twig.

Web application typically consists of a set of controllers, models and views. A controller may be arranged as a core, which receives all requests and causes other controllers to perform other actions as appropriate. The most obvious advantage we can get from using the Model-View-Controller architectural pattern in web development is clear separation of presentation logic (User Interface) from application logic (data processing).

Supporting various range of client platforms, from different operating systems to screen resolutions and form-factors is a common issue nowadays. The request made from a phone or a tablet should be differentiated from a request made from a personal computer or another application. And MVC helps here: a model will return the same data for each of these requests, however a controller may select an appropriate format for the output format such as JSON or XML if the request comes from another application, or HTML if the request comes from a web browser.

In addition to isolating views from business logic and data constraints, MVC significantly reduces the complexity of large applications. The code becomes much more structured and readable. Thus making testing and maintenance of an application easier.

10. LIMITATIONS OF MVC ARCHITECTURE AND HMVC AS THE SOLUTION

Looking perfect in theory, MVC faces several challenges in practice. Let us start with recalling the main issue this architecture solves: divide an application into three different components – Models, Views and Controllers, and achieve minimal dependency between these components, as well as between different parts of a software system. In academic sources and examples, it works out quite well. There is a model for some data, a view for it, and a controller to perform certain operations to let user interact with this data.

However, in the real world the application has to simultaneously operate several models such as articles, users, comments. It is not tedious to implement, and MVC provides appropriate structure, but implementation of such interactions increases the number of dependencies – a view and a controller start to depend on more than one model, and one model provides data for more than one view and/or more than one controller.

To display data from different models, it is more logical to use views designed particularly for them. For instance, displaying comments for articles and items in the exact same way looks logical. The classic MVC structure does not provide such capability, but developers bypass it using templates. We use one view that gets data from models and a combination of different templates serves to display the output. And we increase the number of dependencies again.

Sometimes it is necessary to perform an action on more than one model at the same time. For example, we need to delete a user from a content management system and all of his or her comments and posts. We end up creating a controller that describes not only the operation of the model it refers to (in this example – users), but also to models which have no direct relationship with it (posts and comments). Thus, we generate more and more non-obvious dependencies.

How do we solve these issues? Since the problems arise from the fact that instead of an application structured according to MVC pattern we create something like MMMVVVCCC, where each model, view and controller can relate to different subsystems, the solution is obvious – get back to MVC with only one model, view, and controller for each component. Hence, we can identify the core principle of HMVC: the application should use only rigidly fixed triads of Model-View-Controller subsystems that interact with each other through its controllers only.

RESULTS

As we can see, software architecture is not only a compulsory discipline in software engineering, but also a handy tool for structuring software products, and simplifying not only development but also integration into existing IT environment, and maintenance. Before starting the actual production, a software architect shall review business processes, consider surroundings of a required application and create the design that fits requirements in a certain pattern apprehensible by stakeholders.

MVC is not a silver bullet in software architecture. It has its limitations and there are ways to bypass or solve appearing issues. And HMVC is not a perfect solution either because it requires a lot of low-level design, very detailed and clear structures of data inputs and outputs, so sometimes it is unnecessary time- and effort-consuming and makes the system “over engineered”.

LIMITATIONS OF STUDY

This work does not include discussion of the actual implementation of discussed pattern and approaches. The software framework chosen is stated in the Conclusion with decent justification.

The article not discuss similarities and differences from MVC of such architectures as MVP (Model-View-Presenter) or PAC (Presentation-Abstraction-Control) and others as well as advantages and disadvantages of its use due to low rate of popularity of these patterns in the web industry and certain limitations.

RECOMMENDATION

The author recommends to review the reference list to understand the process of developing a software architecture that fits requirements of a project and other topics in depth. The sources in the list contain broad discussion of the role of an architecture in software project management, design and documentation.

Another recommended article is “Scaling web applications with HMVC” by Sam de Freyssinet, developer of a PHP framework known as Kohana. In this article, Sam provides readers with diagrams and code examples that help to comprehend advantages, implementation techniques and structure of HMVC.

CONCLUSION

Software Architecture affects every aspect of software production and lifecycle. Starting from the planning stage of a project, it is important to reach a certain level of clearance on requirements to the software product, its surrounding within an IT infrastructure of an organisation. An architect should consider expectations of different stakeholders, and design the system in way that fits all of these different aspects, and then pass the design to the development team. A project benefits from its architecture during the development through clearly stated low-level requirements, and then after the release, maintainers are happy with comprehensive structure of the source code and detailed documentation.

One of the goals of my project was making the solution accessible from different desktop and mobile platforms such as Windows (both NT and RT), Linux, Android, Mac OS, iOS and others; thus, the product is a web application with server-side scripting done with PHP and is based on Yii2 framework. The framework is using Model View Controller (MVC) architecture and DB-first approach. Hence, it requires well-structured relational database which in this project is built on MySQL. The framework resolves some of the MVC limitations through “lazy loading method”, and I use quite a number of dependencies which is not a perfectly optimized approach. However, the scale of the project does not require the system to handle high load.

References

Avison, D.E., Fitzgerald, G. and Fitzgerald, G. (2006) *Information systems development: Methodologies, techniques and tools*. 4th edn. London: McGraw Hill Higher Education.

Bosch, J. (2000) *Design and use of software architectures: Adopting and evolving a product-line approach*. Reading, MA: Addison-Wesley Educational Publishers.

Brooks, F. 1986. "*No Silver Bullet*" Information Processing 1986, Proceedings of the IFIP Tenth World Computing Conference, H.-J. Kugler, ed. Amsterdam: Elsevier Science B.V.; pp. 1069–1076.

Humphrey, W.S. (1994) *A discipline for software engineering*. 6th edn. Reading, MA: Addison-Wesley Educational Publishers.

Lunn, K. (2002) *Software development with UML*. Basingstoke: Palgrave Macmillan.

Pressman, R.S. and Maxim, B.R. (2014) *Software engineering: A practitioner's approach*. 8th edn. New York: McGraw Hill Higher Education.

Pty2016SitePoint (2015) *The best PHP framework for 2015: SitePoint survey results*. Available at: https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/ (Accessed: 30 August 2016).

Q-Success (2009) *Usage statistics and market share of PHP for Websites, august 2016*. Available at: https://w3techs.com/technologies/details/pl-php/all/all (Accessed: 30 August 2016).

Reenskaug, T.M.H., 1979. The original MVC reports.

Sam de F. (2010) *Scaling web applications with HMVC*. Available at: https://inviqa.com/blog/scaling-web-applications-hmvc (Accessed: 30 August 2016).

Whitten, J.L., Bentley, L.D. and Randolph, G. (2006) *Systems analysis and design methods*. 7th edn. Boston, MA: McGraw Hill Higher Education.