Corvinus University of Budapest

Faculty of Business Administration

Department of Computer Science

The technology trends in web development and their effects on businesses

Author: Dávid Naményi

BSc in Business Informatics

2016

Advisor: Blanka Láng, PhD

I. számú melléklet

NYILATKOZAT SAJÁT MUNKÁRÓL

Név: **Naményi Dávid**

E-mail cím: **dnamenyi@gmail.com**

NEPTUN kód: **kd00kg**

A szakdolgozat címe magyarul:

A webfejlesztés technológiai trendjei és azok üzleti hatásai

A szakdolgozat címe angolul:

The technology trends in web development and their effects on businesses

Szakszeminárium-vezető (vagy konzulens) neve: **Dr. Láng Blanka**

Én, **Naményi Dávid** teljes felelősségem tudatában kijelentem, hogy a jelen szakdolgozatban szereplő minden szövegrész, ábra és táblázat – az előírt szabályoknak megfelelően hivatkozott részek kivételével – eredeti és kizárólag a saját munkám eredménye, más dokumentumra vagy közreműködőre nem támaszkodik.

Budapest, **2016. május 2.**

_____

hallgató aláírása

TÉMAVEZETŐI NYILATKOZAT

Alulírott **Dr. Láng Blanka** konzulens kijelentem, hogy a fent megjelölt hallgató fentiek szerinti szakdolgozata benyújtásra alkalmas és védésre ajánlom**.**

Budapest, **2016. május 2.**

…….

konzulens aláírása

II. számú melléklet

NYILATKOZAT
A SZAKDOLGOZAT NYILVÁNOSSÁGÁRÓL

Név: **Naményi Dávid**
Alapszak, szak neve: **Gazdaságinformatikus, BSc**

Dolgozatom elektronikus változatának (pdf dokumentum, a megtekintés, a mentés és a nyomtatás engedélyezett, szerkesztés nem) nyilvánosságáról az alábbi lehetőségek közül kiválasztott hozzáférési szabályzat szerint rendelkezem.

TELJES NYILVÁNOSSÁGGAL

A könyvtári honlapon keresztül elérhető a Szakdolgozatok/TDK adatbázisban (http://szd.lib.uni-corvinus.hu/), a világháló bármely pontjáról hozzáférhető, fentebb jellemzett pdf dokumentum formájában.

KORLÁTOZOTT NYILVÁNOSSÁGGAL

A könyvtári honlapon keresztül elérhető a Szakdolgozatok/TDK adatbázisban (http://szd.lib.uni-corvinus.hu/), a kizárólag a Budapesti Corvinus Egyetem területéről hozzáférhető, fentebb jellemzett pdf dokumentum formájában.

NEM NYILVÁNOS

A dolgozat a BCE Központi Könyvtárának nyilvántartásában semmilyen formában (bibliográfiai leírás vagy teljes szöveges változat) nem szerepel.

Budapest, **2016. május 2.**

……

a hallgató (szerző) aláírása

Table of contents

1. Preface

In this paper my goal is to demonstrate the evolution and the current situation and challenges of web development, showcase a modern day web developer's set of tools and scrutinize the technology shifts' effects on businesses.

Thanks to the rapid growth in its economic significance, the web has grown into a massive industry with plenty of areas and different aspects that you need to take into consideration when talking about web projects. For instance there is product development, project management, information architecture, user experience design, user interface design, system operation and software development too, not to mention the human factors and the general business areas that also apply (e.g. finance, accounting, marketing, sales, branding, HR, etc.).

This paper is far from being exhaustive and focuses mainly on the technical side, i.e. development and operations. These are still very broad terms and cover numerous different areas, each of which would deserve a book on its own due to the complexity and depth of the questions and challenges. This is why I decided to give only a bird's eye view of a few subjectively selected areas within the field of web development.

I aim to demonstrate briefly what the most important technologies, tools and practices are nowadays and how the recent advancements are changing the landscape from the business perspective. Hopefully I will also be able to shed some light on the direction in which the industry is moving.

Some of the things I write about can be discussed in general and apply to all fields of software development, but web development is somewhat different and unique in a number of aspects. I myself have only gained experience in the field of web development, so everything I write about refers to the web development environment.

Although I discuss the trends and technologies mostly in general, I write about some personal experiences and opinions too which I have gained through the web development projects I have been involved in at JóSzaki (the biggest Hungarian handyman finder service), at Divide By Zero Australia (a technology and branding agency in Sydney) and at the Digital Team of The CyberInstitute (Brisbane, Australia).

2. The evolution of the web

I am not going to go into the specific details of the web's history, but I am going to mention a few important concepts and will show how closely tied its advancement is to the advancement of the internet itself.

So first, let's clarify what the difference is between the internet and the web. We often use these terms interchangeably, but actually they mean different things. To make it simple, the internet is the actual computer network that connects billions of digital devices together globally, allowing them to transfer information with each other via certain protocols. The web is just one of the many ways of accessing information on the internet. It uses the HTTP protocol to allow communication and transmission of data between applications. Using browsers is the most common way of accessing web documents (web pages) over the internet. In this paper I'm focusing on the software development environment in which the web pages (websites) are built and operated. When discussing the web's evolution, however, I need to talk about the web and the internet at the same time because they cannot be easily separated from each other in this context.

It is very interesting to know and see that the internet started as a limited network of interconnected computers used primarily for military and scientific purposes and then evolved into something way larger, the open World Wide Web, and through countless changes and shifts finally became an essential part of our everyday lives. I must also note that the word 'finally' might be misleading in this context, because the rapid advancement of internet technology is far from reaching its peak. It appears to be conquering new territories and revealing new ways to utilize the technology.

Kim Morrow in her 2014 UX Booth article writes that "the internet has become an integrated, seamless, and often invisible part of our everyday lives. (...) The only thing that seems certain is that the Internet is changing rapidly". I think she makes a really good point here, because it articulates not only the ever-changing nature of the internet, but also the extent to which we have embraced it and have built our lives upon it.

A good way of demonstrating how the level of embracement is changing is to take a look at the different generations' lives. Based on Harry Wallop's article published in 2014 in The Telegraph, Generation X people were born between the early 1960s and

the early 1980s, Generation Y people were born between the early 1980s and the mid-1990s and the members of Generation Z are those who were born like after 1996 or so. Now, how a friend of mine explained it, Generation X know exactly when they are using the internet and what for, Generation Y cannot distinguish between using the internet and not using the internet, because it is such an integrated and invisible part of their lives, and Generation Z does not even understand the question.

This process was and is being further enhanced by the fast spread of smartphones and the rise of Internet of Things (IoT) devices, alongside with the emergence of big data technologies and artificial intelligence solutions.

The web itself started off as simple, static HTML pages referencing each other (Web 1.0) and then evolved into the Web 2.0 era which brought dynamic, complex websites, extended social networks and lots of user generated content, all fostering collaboration, communication and sharing. The definition of the third generation (Web 3.0) is way less clear, some even argue whether it has already started or we are still in the time of Web 2.0.

Based on Nova Spivack's research at the Lifeboat Foundation, the technologies and concepts that build up the foundation of Web 3.0 are "semantic web, microformats, natural language search, data-mining, machine learning, recommendation agents, and artificial intelligence technologies — which emphasize machine-facilitated understanding of information in order to provide a more productive and intuitive user experience." This generation is often referred to as 'the Semantic Web' and 'the intelligent Web'. Spivack also uses a diagram to illustrate the direction of advancement in terms of 'Semantics of Social Connections' and 'Semantics of Information Connections':

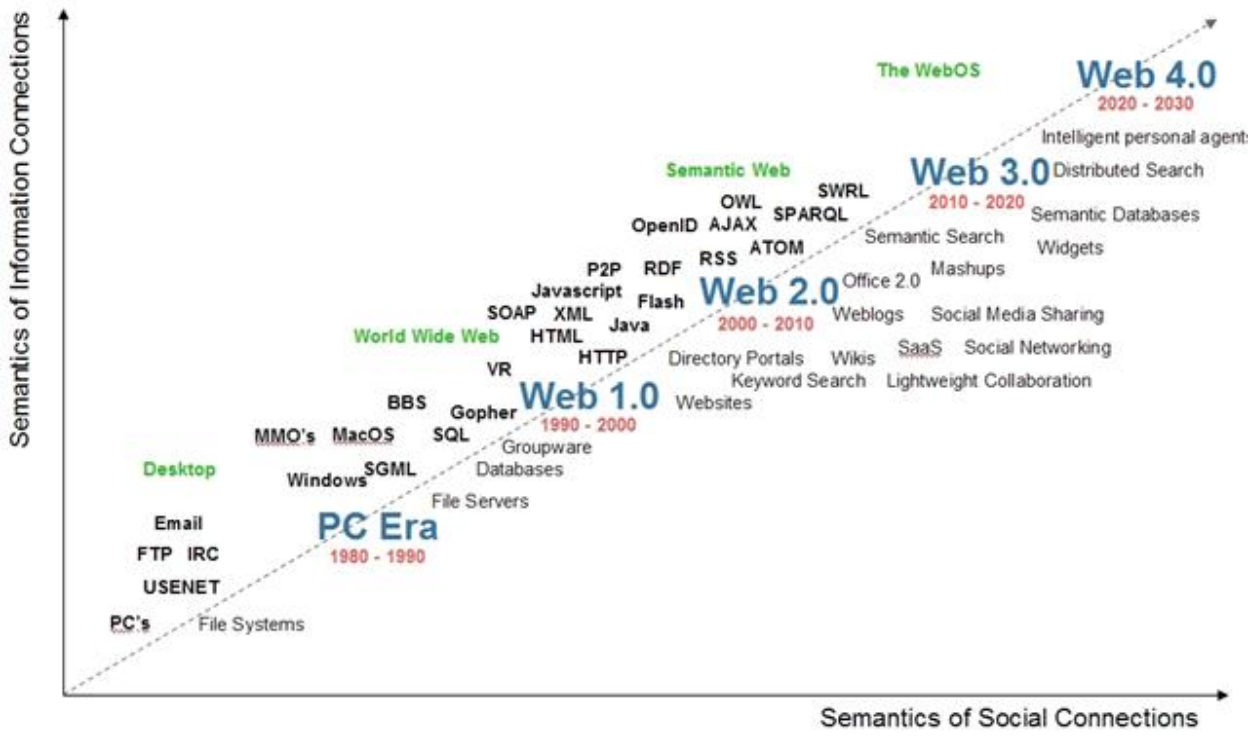

Figure 1. 'Semantics of Social Connections' and 'Semantics of Information Connections'. Source: Spivack (2016)

In 2008 Mills Davis conducted a profound, 720-page study on the evolution of the web, and in his short executive summary there is a great diagram that aims to capture the essence of the web generations. The horizontal axis shows the extent of social connectivity and the vertical axis shows the extent of knowledge connectivity and reasoning. The Web (1.0) connected information, The Social Web (2.0) connects people, The Semantic Web (3.0) connects knowledge and The Ubiquitous Web (4.0) is going to connect intelligence.

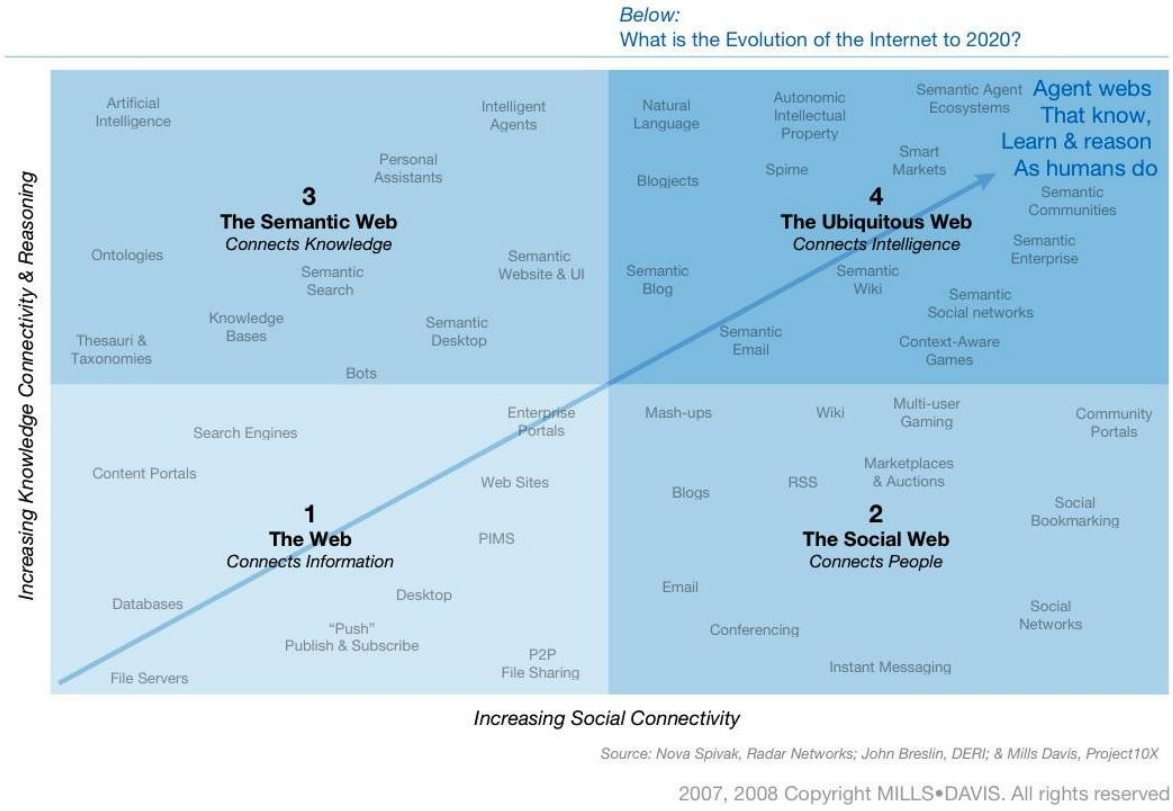

Figure 2. What is the Evolution of the Internet to 2020? Source: Davis (2008)

According to the statistics of www.internetlivestats.com, as of 17 April 2016 there are approximately 3 350 500 000 internet users in the world which means that around 40% of the world population has an internet connection today and the total number of websites in the world is around 1 015 600 000. In Hungary the internet penetration is estimated to be around 80.2% (in 2000 it was only 7%), while one of the most popular Hungarian news & media website (index.hu) has about 40.6 million visits a month (www.similarweb.com estimate, March 2016).

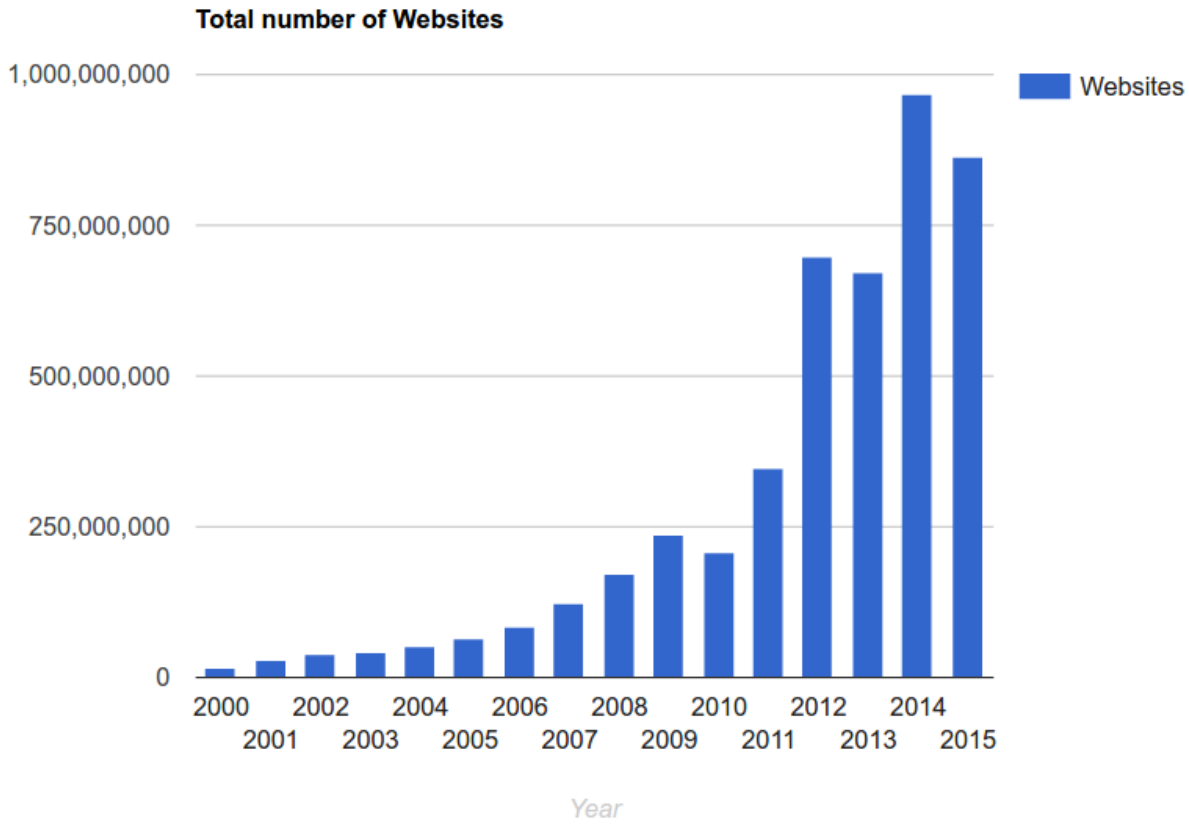

Figure 3. Total Number of Websites. Source: NetCraft and Internet Live Stats (2016)

"Website" means unique hostname (a name which can be resolved, using a name server, into an IP Address). It must be noted that around 75% of websites today are not active, but parked domains or similar. Periodic drops in the total count can depend on various factors, including an improvement in NetCraft's handling of wildcard hostnames.

Personally I think that responsive web design techniques (which make many companies choose a mobile-friendly web UI over a native smartphone application) and cloud computing (which moved lots of the traditionally desktop-based applications to the web) are certainly among the many drivers of the leap in the number of websites.

This overview of the trends sort of explains why web development has become so vital in today's economy. Through this huge transformation the whole web industry has boomed - both the demand and supply sides are getting stronger and stronger. On the one hand, almost all existing companies need either a simple online presence or advanced web based systems and there are countless new start-ups and businesses

that are formed specifically to exploit the opportunities that the web provides. On the other hand, many companies deliver web development service, so in order to stay ahead of the competitors, they need to adjust their skillset, technology and efficiency to the increased needs (both in terms of speed of delivery and quality of implementation).

Due to the increasing expectations regarding the speed of delivery and the quality of implementation, developers also face new challenges. Complex problems usually call for complex solutions, and developers need new approaches, skills and tools to maintain speed and quality.

In the following chapter I am going to showcase a slice of the recent trends, technologies and best-practices that are shaping the modern web development environment and then later on I will also write about how they affect the business.

3. The technology trends

In this chapter I am going to elaborate on a number of important aspects that are necessary to consider when someone wants to run successful and high quality web development projects. I aim to give an overview on a broad spectrum of tools and techniques which all contribute to an effective and efficient environment. I will focus on the concepts and advantages of each of these, rather than on the exact details of the implementation.

3.1. **Cloud computing**

Since the move to cloud computing is such an apparent phenomenon in the IT world and I am going to mention cloud solutions many times in this paper, I find it important to clarify and define the cloud right at the beginning.

“In the simplest terms, cloud computing means storing and accessing data and programs over the Internet instead of your computer's hard drive.” (Griffith, 2015) To be a bit more specific and in-detail, here is the definition of cloud computing that we can find in the book called Cloud Computing Bible (Sosinsky, 2011): “Cloud computing refers to applications and services that run on a distributed network using virtualized resources and accessed by common Internet protocols and networking standards. It is distinguished by the notion that resources are virtual and limitless and that details of the physical systems on which software runs are abstracted from the user. (...) Cloud computing makes the long-held dream of utility computing possible with a pay-as-you-go, infinitely scalable, universally available system. With cloud computing, you can start very small and become big very fast. That's why cloud computing is revolutionary, even if the technology it is built on is evolutionary.”

Cloud has become mainstream in the field of web development too - almost every major website and web service utilizes the cloud in one way or another. Taking a look at the analysis of the RightScale report published by Ben Kepes (2015), we can see that Amazon Web Services (AWS) continues to dominate in public cloud by 57% adoption, followed by Microsoft Azure, Rackspace and Google.

Throughout the paper I am going to demonstrate several cloud services and their benefits from both the IT and the business perspective.

3.2. **Programming languages and technologies**

A good number of different languages and technologies are used for web development nowadays. I am not going to detail them or compare them, especially because such comparisons and disputes usually result in senseless flame wars without actually finding a winner, because obviously they all have advantages and disadvantages, similarities and differences which make it impossible to simply benchmark them. The goal of this section is just to give an overview of the most commonly used ones.

3.2.1. Front-end

In front-end development (also called the client side - which is responsible for the presentation layer, the user interfaces) the situation is quite simple in terms of programming languages. Since XHTML and Flash are mostly out of business, there are only 3 major languages used, each in its own domain: HTML is used as the markup language to describe the content and structure of the website, CSS is used as the presentation language to describe the look of the website through style sheets, and Javascript is used as the programming language to make the interfaces interactive and provide ways for asynchronous communication with the server. Even though the map of front-end development languages seems to be clear and simple at first glance, the ever-growing number of libraries, extensions, tools and frameworks have made it quite messy and complex.

In the CSS world, the situation is still relatively easy to understand: on the one hand, there are numerous CSS frameworks and standards to make naming and usage conventions standardized and thus the CSS code sustainable, reliable and scalable. On the other hand, the interpreted scripting languages that can be compiled to pure CSS, like SASS and LESS are becoming widespread. These extend CSS by providing mechanisms available in more traditional programming languages, like variables, functions (called mixins), logical nesting, loops and inheritance.

In the field of Javascript development, there are a lot of libraries and frameworks that are widely used. Each has its advantage and use case. They offer things like easier handling of the DOM, dynamic views, quicker implementation of user interfaces, advanced animations, enhanced event handling, versatility, extensibility and the list goes on. There are also some which provide their own syntax and can be compiled into

Javascript. To mention a few, the most popular ones include jQuery, AngularJS, React, Babel, Coffeescript, TypeScript, ExtJS, Impress.js, Backbone.js and D3.

When a huge set of Javascript tools and libraries are used for a project, managing these external libraries and dependencies becomes a major headache for developers. RequireJS and Browserify have gained ground because they can effectively make it less of a hassle by their file and module loader solutions.

Frameworks that give both CSS and Javascript solutions and components are also mainstream. Bootstrap and Foundation are the two dominant players in the market. They are constantly developed to enable developers use out-of-the-box, easy-to-use solutions to the real world challenges.

Regarding HTML, templates engines are often taken advantage of in order to reduce code duplication and have a more maintainable HTML code base. The most popular template engines operate on the server side (e.g. Smarty, Twig, Blade) but there are also some client side templating systems, like Mustache.js and Handlebars.js.

As the last discussed piece of the front-end development technologies, it's worth mentioning Grunt and Gulp, which are task managers that are responsible for a lot of the compilation and compression tasks and play a crucial part in the build processes. They can compile SASS and LESS into CSS along with many extra functionalities (like adding browser prefixes and compressing the resulting CSS files) and are also used to validate, compile, concatenate and compress Javascript files.

3.2.2. Back-end

In back-end development (also called the server side - which is responsible for the data manipulation and business logic) the selection of widely used programming languages is a lot larger. PHP, Javascript (mainly NodeJS on the server side), Ruby, Python, Perl, Java, C#, Scala and Go are all examples of popular languages that can be used for web development.

Gathering reliable data on their respective popularity is extremely hard, because most of the surveys are not representative and also quite a few of the languages are used for multiple purposes, which makes measuring the actual usage for web development impossible. Still I would like to show some statistics that demonstrate the situation broadly.

Stack Overflow, the huge question and answer site for programmers has been conducting an extensive developer survey each year since 2013. These surveys reveal a lot about the technology trends. This year (2016) more than 56.000 coders in 173 countries answered their questions, and the collected data shows that JavaScript is by far the most popular programming language, but since its main domain is still the client side, it does not say much about its ranking among the back-end technologies. The following graph shows the results:

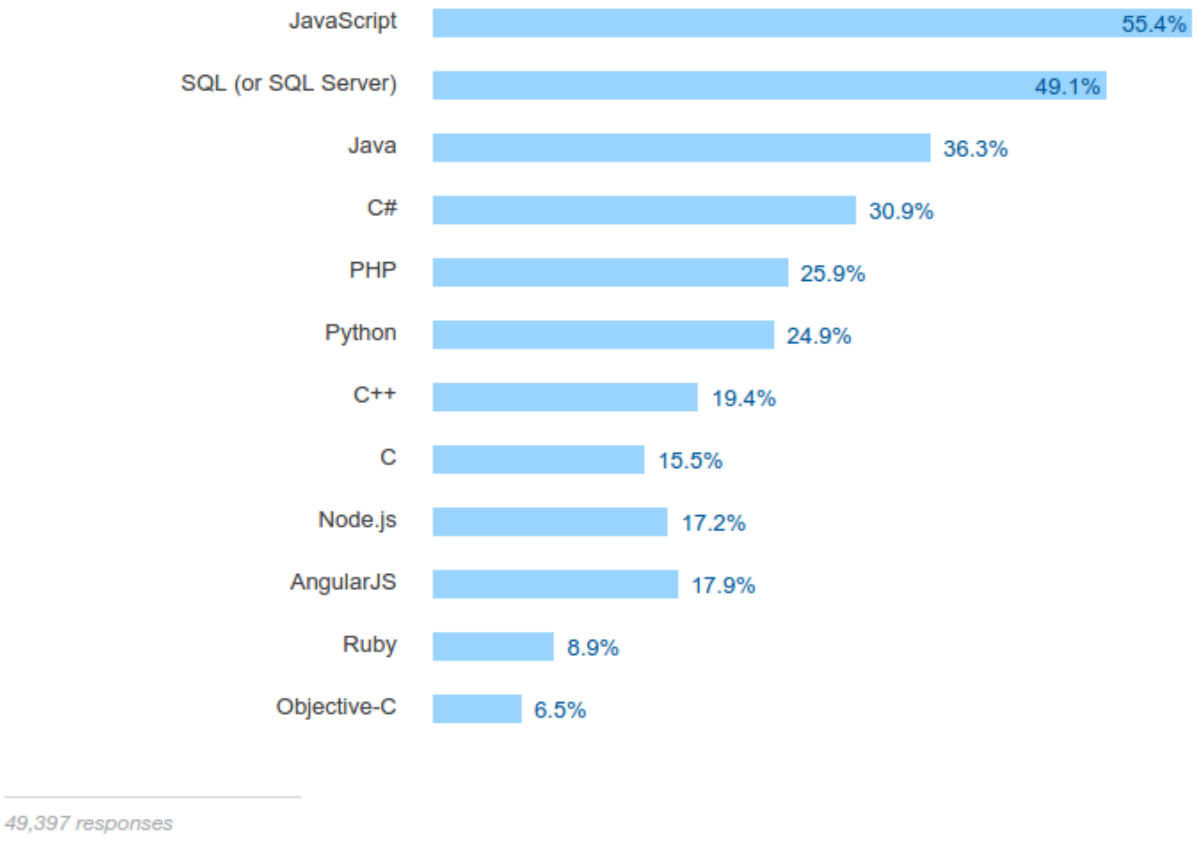

Figure 4. The most popular programming languages. Source: Stack Overflow (2016)

This data is actually not very precise regarding web development, because as I have already mentioned, the first ranked Javascript mostly covers front-end development, SQL is for databases, Java and C# are extensively used outside the web development area, while C++, C and Objective-C are never or rarely used for web development. Still, it gives a notion of the prevalence of the languages.

Taking a look at previous year's data, the biggest change regarding the web development languages is that PHP's 34.8% result in 2013 shrinked to 25.9% by 2016 in favor of emerging languages.

Gerard Millares in an article published in 2015 writes that "more than 75% of the top websites use PHP as their server side programming language". The source of the data is not clear so we should have reservations about this information, but it's interesting to see it along with his following statement: "even though PHP is the by a far margin, the most used server side programming language, it is amusing that when it comes to websites that attract high traffic, Java and Javascript are the clear winners. While around 82% of websites with Java as their server side language attract high traffic, the value drops to below 15% for websites having PHP as their server side language". You can see more information about this on his diagram:

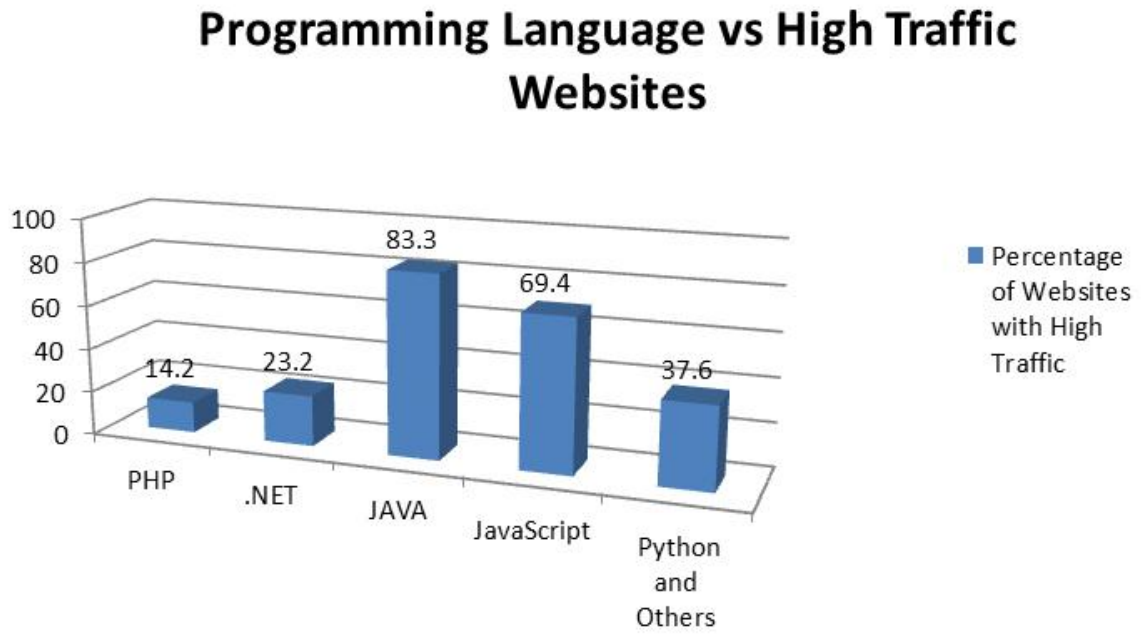

Figure 5. Programming Language vs High Traffic Websites. Source: Millares (2015)

In addition to the presented facts that PHP is gradually losing ground and is mainly used for low-traffic sites, in my personal opinion PHP is also a language that is generally perceived as a not "trendy", not "sexy" programming language. This notion might have been somewhat improved by the new versions of the language that introduced very

important new features and language constructs and also addressed the performance issues and made PHP programs run way faster.

Those topics in my paper that are programming language specific assume that PHP is used for the back-end because that's what I am most familiar with.

Unlike in the previous front-end development section, I am not going to write about frameworks and tools here, because they are going to be discussed later on, for instance in Section 3.3.1.

3.2.3. Databases

There is a breadth of options when it comes to databases, and most of the key players have been around for long and keep their positions in the market. However, as I see it, there are some major trends happening there that are reshaping the landscape. I am going to reveal some statistics first and then elaborate on these trends.

The Vienna-based SolidIT Consulting & Software Development GmbH has developed an advanced algorithm, called DB-Engines Ranking, that measures the popularity of the database systems based on publicly accessible data by using a combination of metrics. Here is their latest data from April 2016:

303 systems in ranking, April 2016

Rank			DBMS	Database Model	Score		
Apr 2016	Mar 2016	Apr 2015			Apr 2016	Mar 2016	Apr 2015
1.	1.	1.	Oracle	Relational DBMS	1467.53	-4.48	+21.40
2.	2.	2.	MySQL	Relational DBMS	1370.11	+22.39	+85.53
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1135.05	-1.45	-14.07
4.	4.	4.	MongoDB	Document store	312.44	+7.11	+33.85
5.	5.	5.	PostgreSQL	Relational DBMS	303.73	+4.10	+35.41
6.	6.	6.	DB2	Relational DBMS	184.08	-3.85	-13.56
7.	7.	7.	Microsoft Access	Relational DBMS	131.97	-3.06	-10.22
8.	8.	8.	Cassandra	Wide column store	129.67	-0.66	+24.78
9.	9.	↑ 10.	Redis	Key-value store	111.24	+5.02	+16.69
10.	10.	↓ 9.	SQLite	Relational DBMS	107.96	+2.19	+5.67
11.	11.	↑ 14.	Elasticsearch	Search engine	82.58	+2.41	+17.92
12.	12.	↓ 11.	SAP Adaptive Server	Relational DBMS	73.32	-3.33	-13.37
13.	13.	13.	Teradata	Relational DBMS	72.26	-1.81	+2.00
14.	14.	↓ 12.	Solr	Search engine	66.02	-3.35	-15.98
15.	15.	15.	HBase	Wide column store	51.49	-0.92	-9.65
16.	16.	↑ 17.	Hive	Relational DBMS	49.08	-1.43	+6.33
17.	17.	↓ 16.	FileMaker	Relational DBMS	46.10	-1.83	-5.72
18.	18.	18.	Splunk	Search engine	42.35	-1.38	+4.32
19.	19.	↑ 21.	SAP HANA	Relational DBMS	40.35	+0.36	+7.01
20.	20.	↑ 22.	Neo4j	Graph DBMS	31.91	-0.44	+3.50
21.	↑ 22.	↑ 25.	MariaDB	Relational DBMS	31.58	+1.70	+9.19

Figure 6. DB-Engines Ranking. Source: SolidIT (2016)

The traditional relational database managements systems (RDMS) enjoy clear dominance and will most probably remain strong, because they are a perfect choice in many cases and what is more, most developers are used to working with them. The 3 top SQL-based databases currently are Oracle, MySQL and MS-SQL based on the DB-Engines Ranking, but personally I think that specifically for web development purposes MySQL and PostgreSQL are the two winners.

The first major trend we can see though is the rise of alternative database technologies. In the past few years, NoSQL (also called nonrelational) databases became very popular and widely used. But what exactly does the term mean? In the preface of NoSQL Distilled (2012), Sadalage and Fowler explains it this way: “The term ‘NoSQL’ is very ill-defined. It’s generally applied to a number of recent nonrelational databases (...) They embrace schemaless data, run on clusters, and have the ability to trade off traditional consistency for other useful properties. Advocates of NoSQL databases claim that they can build systems that are more performant, scale much

better, and are easier to program with." MongoDB, Cassandra, Redis, Riak, Neo4j, CouchDB, HBase / Hadoop, Google Cloud Bigtable and Amazon DynamoDB are all examples of NoSQL databases, even though they employ a number of different logical models. A lot of, mainly large-scale, web development projects have decided to utilize the advantages of NoSQL technologies.

Although three years ago there was only one NoSQL database in the top 10 of the aforementioned list, now there are three (all of which is "schema-less") which shows that they're gaining momentum. This graph, based on the same DB-Engines Ranking data, captures the trends quite well:

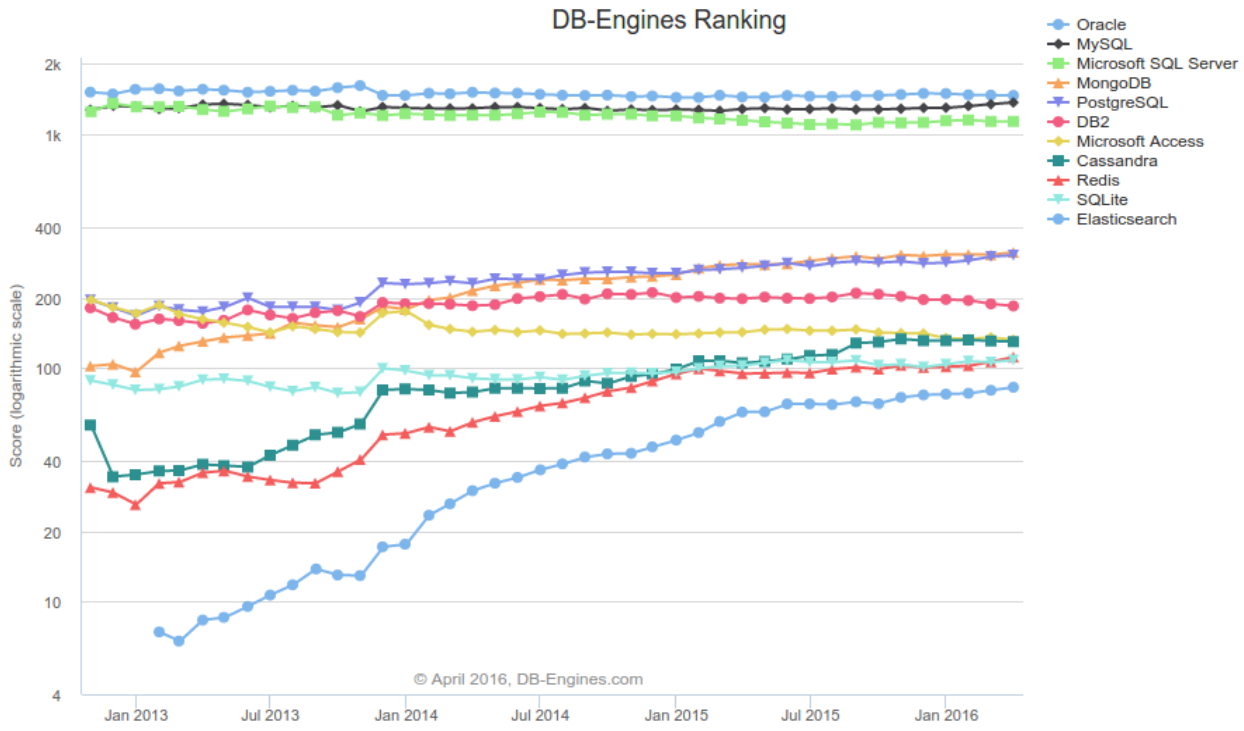

Figure 7. DB-Engines Ranking. Source: SolidIT (2016)

It is worth noting about these statistics that the calculation takes into account metrics like search volume in Google or mentions in forums and social media. This can lead to distorted results, because just that something is searched for or talked about does not mean that it is actually loved and used. For example, people probably talk a lot more about the issues of a particular system than about the happy moments, or maybe the topic is the struggle to migrate to another, better database. Nonetheless, these figures give a glimpse of the database market and are good enough estimations, especially

because it is virtually impossible to gain data about the number of active installations or other relevant metrics.

The second major trend is pointed out very clearly by Pramod J. Sadalage in an article published in 2014: “There is also movement away from using databases as integration points in favor of encapsulating databases with applications and integrating using services”. Instead of many applications relying on the same enormous database to share information, the big systems are often broken down into smaller web services with their own smaller databases, and the information sharing happens through the services’ communication. It is very closely tied to microservices (about which I am going to write later in Section 3.6) and also to the so-called Polyglot Persistence concept, which is a very important result of the rise of NoSQL databases.

Polyglot Persistence means that “it is best to use multiple data storage technologies, chosen based upon the way data is being used by individual applications or components of a single application. Different kinds of data are best dealt with different data stores.” (Serra, 2015) To explain both Polyglot Persistence and the encapsulation of the database with services, I am referencing NoSQL Distilled again, because the authors created a really good set of charts to demonstrate the process of moving from a large relational database that is responsible for all the data (Step 1) to using different database engines for different purposes (Step 2) and then finally wrapping those datastores into services (Step 3):

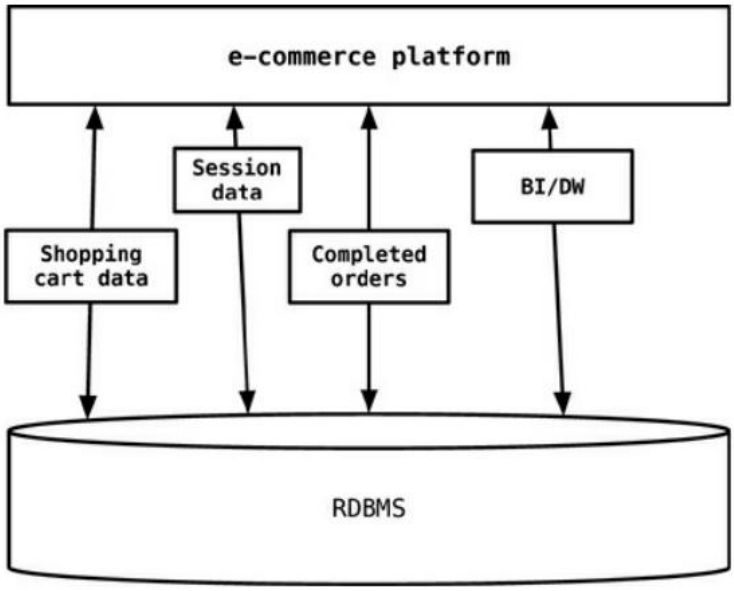

Figure 8. Step 1: Use of RDBMS for every aspect of storage for the application. Source: Sadalage - Fowler (2012)

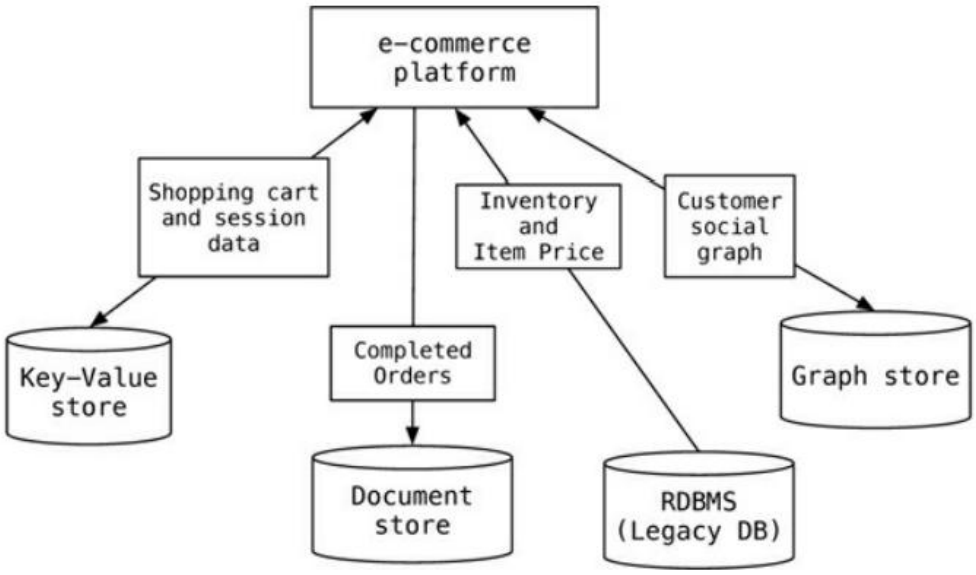

Figure 9. Step 2: Example implementation of polyglot persistence. Source: Sadalage - Fowler (2012)

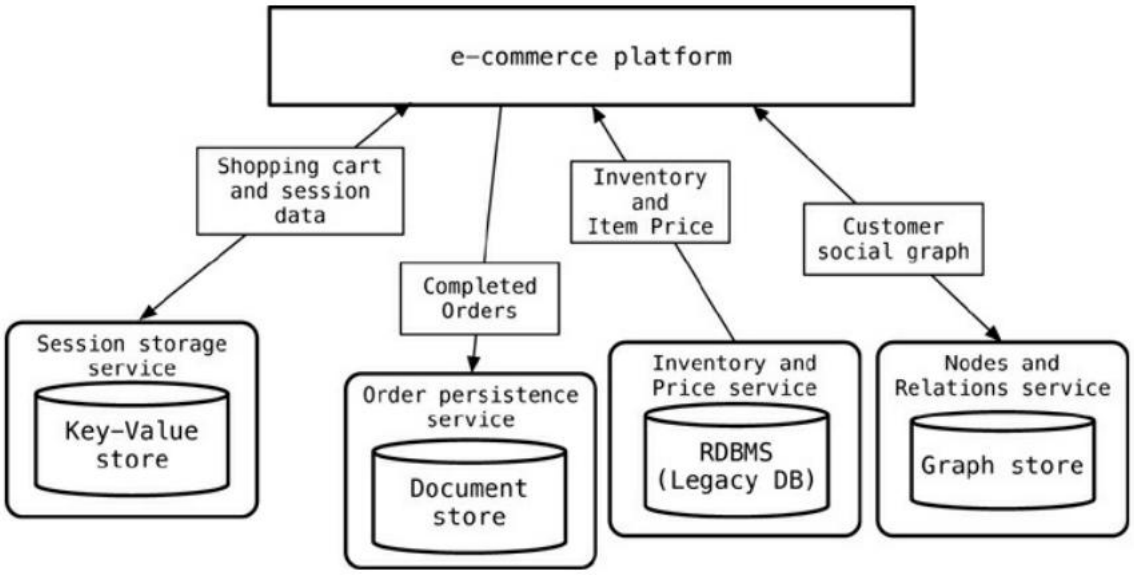

Figure 10. Step 3: Using services instead of talking to databases. Source: Sadalage - Fowler (2012)

The third trend that I would like to mention is that of moving databases to the cloud. There is no big surprise here, it is just a part of the whole cloud trend, but beyond

simple database hosting, companies like Amazon and Google provide cheap fully-managed cloud database servers that take a lot of burden off the shoulders of web developers and operation people. To see its benefits it is enough to mention a few of them, e.g. automated backups, automatic failure detection and recovery, software patching, security and access control, scalability, high availability and speed.

Having seen this impressive list of benefits we can say that these services are an ideal choice for both traditional relational and nonrelational databases. This is true and they are widely used for lots of web development projects indeed - I also have experience with these and I am absolutely satisfied with them so far.

However, I must note that relational and nonrelational databases have one very important difference at this point. While a relational database is typically supposed to run on a single machine and thus can only be scaled vertically, most of the NoSQL databases are designed to run on a cluster of machines which makes it easy to scale them horizontally (I am going to provide more information about the types of scaling later in Section 4.1). It means that the cloud database services can provide an even bigger advantage for distributed NoSQL databases by making it possible to automatically spin up or terminate nodes based on the load.

3.3. **Build your own or use an existing solution?**

This is a very important question, and the answer is, of course: it depends. I'll start with discussing the most common case and then I'll also mention the case when it can make sense to act differently.

In my opinion, it is essential to be familiar with and know how to use third-party services, libraries and tools, because we need to provide a large set of functionality while maintaining fast and agile development. Nowadays web products need to satisfy a broad spectrum of needs, and most of the time there is just not nearly enough time and know-how to develop all those features.

Usually developers would rather build everything from scratch, but implementing your own solutions to common problems takes a whole lot of time. And it's not just the time it takes to write the actual code, but also the great amount of time that you need to invest in gaining the domain specific knowledge that you need for that particular piece of software. What's more, you can never perform as thorough testing and

refinement as widely used software packages have already got, because they have a broader user and developer base that they can rely on.

As the saying goes: *“Don’t reinvent the wheel!”*

Package managers (for instance: Composer, Npm, Bower, Rubygems) enjoy outstanding popularity nowadays, because they have made it easy to download, install and update packages from external sources, and they enhanced the trend to move in this direction. What you need to consider is whether you always want to get the latest version of the code automatically or you want more predictability by making sure that the underlying codebase does not change between releases. It depends on the situation, but I think that in most cases the best way to go is manually updating the external packages regularly to get the bug fixes and improvements and having tests that ensure that the new version behaves the same way the previous version did.

There are many cloud services available out there that you can integrate to your workflow by connecting to it through an API. If you need to have full control over the service, in many cases you can choose to install it on your own server instance instead of using the cloud version. By hosting it, you take over some of the pain of operations but it can be necessary for some network security and availability concerns.

It is strongly recommended to use only properly tested, reliable, well-documented third party libraries, tools, services and APIs. I have had some bad experiences with services that were in an early stage and had not been used and tested out in the wild yet. It means that we wasted lots of time and money both on the integration (poorly documented endpoints, a good number of emerging issues), on the bug fixing and on moving on to a more stable service and doing a major clean-up and refactor in our code.

Choosing a good library or service in and of itself will not necessarily take you ahead of the curve because many others use them. Utilizing these in the right combination, however, might give you a competitive advantage. It is important to understand what each of these tools are designed for, what their strengths and weaknesses are, how they integrate with the packages that are already in use and how you can tailor them to your specific needs.

Above a certain size and complexity, it might make sense to build our own version of the third party libraries and tools to make it a perfect fit for our needs and decrease

dependency on others. I have seen it happening at bigger projects, that in the beginning they decided to use a third-party solution for something, but then later on, as the project grew and they wanted to be more independent and wanted to have even better performance, they started building their own tools. Obviously, it can only happen when you can afford it both in terms of financials and know-how.

3.3.1. Frameworks

This topic is also part of the “Build your own or use an existing solution?” question. Just as libraries, modules, tools and external services, nowadays frameworks are also widespread and play a substantial role in web development projects. How do they contribute to the success of a project?

“Frameworks are about efficiency and effectiveness. They save you time. By forcing common conventions, a framework helps solve common issues like view rendering, asset generation, security, application configurations -- things that happen in every web application. This is good. It brings consistency to decisions. Instead of implementing a feature by writing a number of custom modules, all we have to do is implement it the way the framework wants us to. This saves us time and headaches, and makes the development process easier.” (Megyesi, 2012) And this list of benefits is far from being exhaustive, it could go on with numerous challenges and problems that they provide built-in solutions for. There is also another important factor which I will write about in the People as Single Points of Failure section (Section 4.4.3).

Megyesi mentions an interesting disadvantage too that is worth bearing in mind: “because frameworks are so good at making decisions for us, we get lazy. Instead of thinking hard about how to build a clean system with crisp abstractions, we think about what the framework would want us to do, regardless of whether the resulting code is clean.”

Talking about web development frameworks, the MVC pattern (Model - View - Controller) is a crucial concept as it has been mainstream for a long time and lies under a very big chunk of the world’s websites. When used properly, it can simplify the development and increase the quality and maintainability of the code base. Krzysztof Rakowski in a 2011 Smashing Magazine article explains the pattern briefly by saying that “MVC is a software architecture that allows for the separation of business logic from the user interface. In this architecture, the user sees and interacts with the view

that, in the case of Web applications, is generated HTML code (along with JavaScript, CSS, images, etc.) User actions are passed (as HTTP requests, GET or POST methods) to the controller. The controller is a piece of code that handles and processes user input and then reads and makes necessary changes to the model, which is responsible for the storage and modification of data. (In simple terms, the model consists of the database structure and contents, and the code used to access it.) Then, the controller generates the proper view that will be sent and displayed to user. “

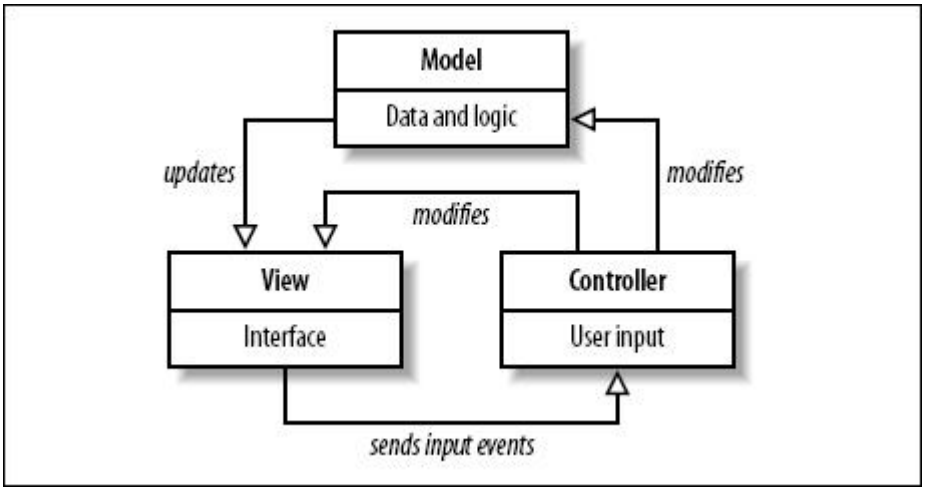

Figure 11. MVC pattern (Model - View - Controller). Source: Moock.org (2016)

Regarding PHP, there are a couple of very strong competitors in the field of frameworks. Obviously, just the same way as for all the other topics in this paper, it is practically impossible to gather reliable data on the actual usage and popularity of the key players, but I have found two sets of data that are worth presenting. The first is the result of the 2015 edition of the annual SitePoint framework popularity survey which Bruno Skvorc published at the end of March 2015:

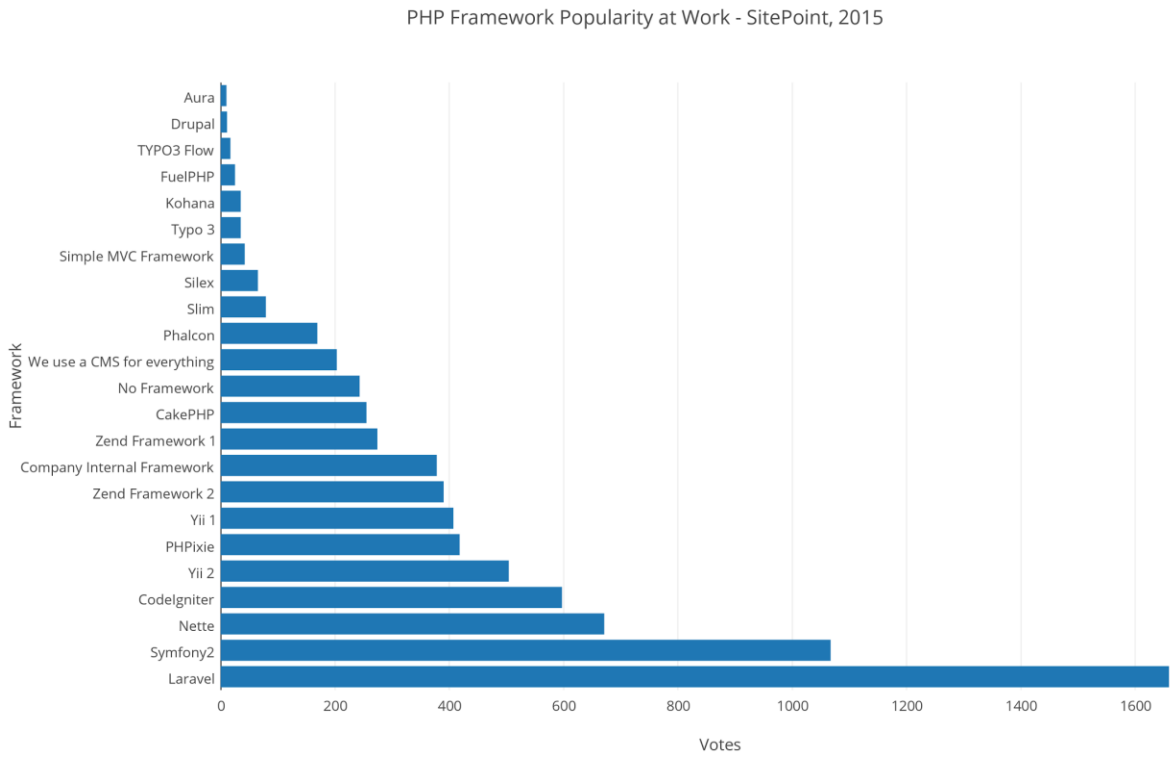

Figure 12. PHP Framework Popularity at Work. Source: Skvorc (2015)

The impressive ranking of Nette and PHPixie is a bit surprising and might be down to the relatively small sample size and the attention of their communities at that given time which might have led them to active participation in the survey. The order of Laravel, Symfony2, CodeIgniter and Yii 2 appears to be a more realistic result.

I also collected data using Google Trends and visualized the results. The graph simply shows the web search interest over time for the given keywords between 1 January 2006 and 23 April 2016:

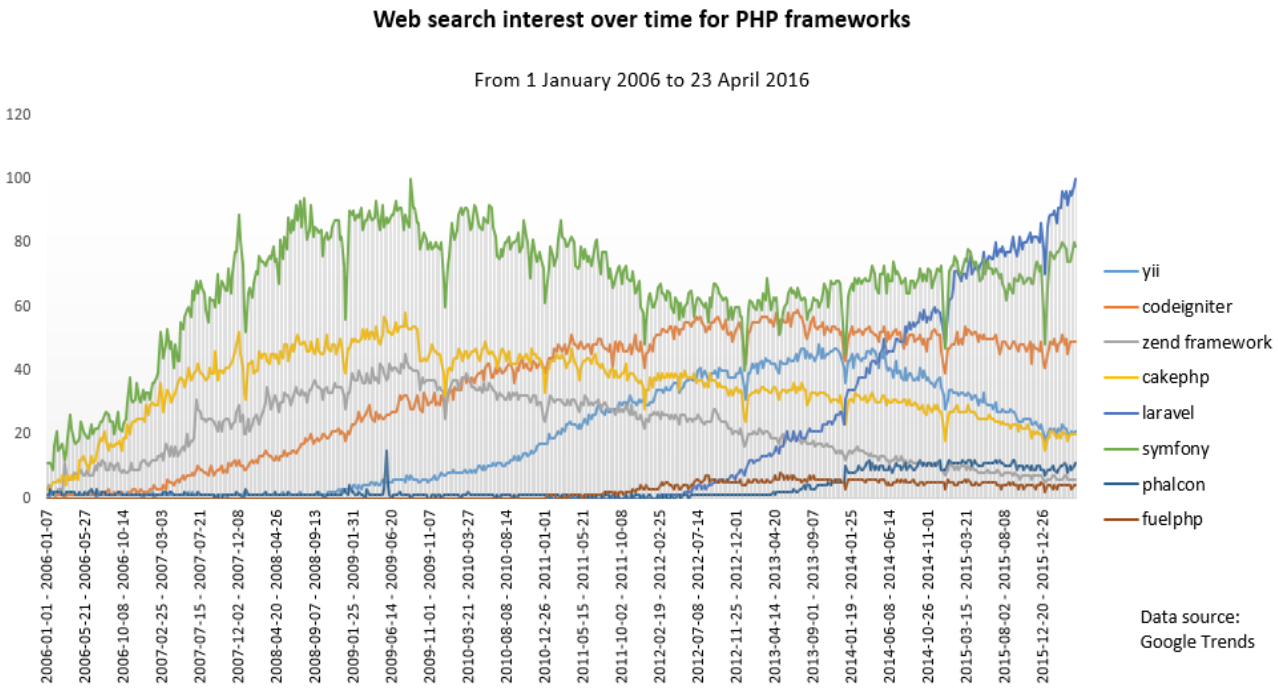

Figure 13. Web search interest over time for PHP frameworks. (My own work, 2016)

This chart does not take the different versions of these frameworks into consideration, but still shows the major trends in the market: Symfony and CodeIgniter have been strong for a quite long time while Laravel appeared from nowhere and took the lead by an extremely fast increase in popularity. Yii, CakePHP and Zend Framework, on the contrary, are gradually losing ground in favor of others.

3.3.2. Content management systems

As the last point in this section, I would like to show the current landscape in a crowded niche market: the content management systems (CMSs), which are widely used for various purposes. Even though web developers who prefer crafting bespoke software tend to despise the common content management systems, it is clear that both the demand and the supply remains strong in the CMS market and they mean a perfect solution for lots and lots of projects.

As usual, there is no precise usage data available due to the complexity of measurement, but the statistics are good enough to point out the key players in the market. The data collected by W3Techs.com shows that as of April 2016 Wordpress has approximately 59.4% market share and interestingly it powers over 26% of all the websites in the world.

© W3Techs.com	usage	change since 1 March 2016	market share	change since 1 March 2016
1. WordPress	26.4%	+0.4%	59.4%	+0.2%
2. Joomla	2.7%		6.1%	-0.1%
3. Drupal	2.2%		4.9%	
4. Magento	1.3%		2.8%	-0.1%
5. Blogger	1.2%		2.7%	-0.1%

percentages of sites

Figure 14. Most popular content management systems. Source: W3Techs.com (2016)

The global OpenSource CMS information provided by the Wappalyzer browser add-on which covers roughly 1% of the entire web leads to the same conclusion that, as of April 2016, Wordpress, Joomla and Drupal are the major competitors, and Wordpress is enjoying significant dominance.

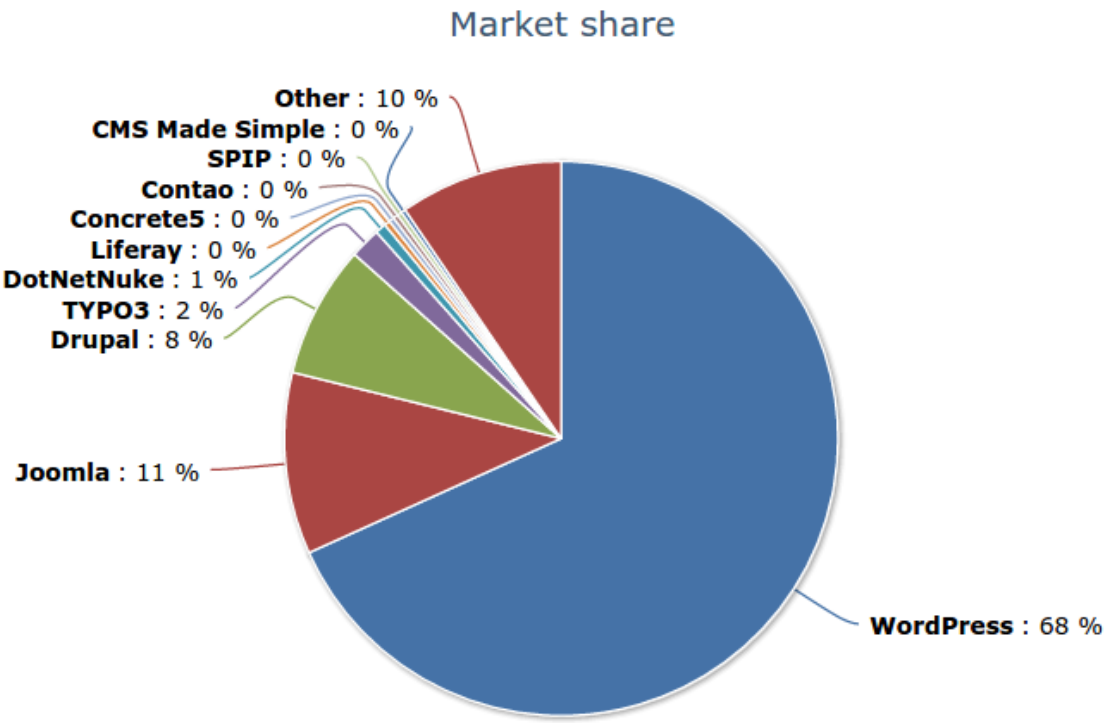

Figure 15. Market share of CMS systems. Source: OpenSource CMS (2016)

3.4. **Version control**

Version control and Git are common buzzwords in the developer scene. I don't think many would argue with saying that nowadays version control is an absolute must for every web development project, and fortunately even amateur developers and developers of small-scale projects tend to use it as a foundation for the code base. But what does version control mean?

"Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. For the examples (...) software source code as the files being version controlled, though in reality you can do this with nearly any type of file on a computer. (...) It allows you to revert files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. Using a VCS (Version Control System) also generally means that if you screw things up or lose files, you can easily recover. In addition, you get all this for very little overhead." (Chacon - Straub, 2014)

There are centralized (e.g. CVS, Perforce, SVN) and distributed (e.g. Git, Mercurial) VCSs. "The main difference between the two classes is that Centralized VCSs keep the history of changes on a central server from which everyone requests the latest version of the work and pushes the latest changes to. On the other hand, on a Distributed VCS, everyone has a local copy of the entire work's history. This means that it is not necessary to be online to change revisions or add changes to the work." (Vergara, 2012).

I personally prefer using Distributed VCS, Git to be specific, and as I see, the trends are all going in this direction, especially because the majority of the open-source projects use Git.

Git is the actual command line tool that we can use locally to version control our files. We can also set it up on a server and use it as a remote repository where we can push our changes or, not surprisingly, we can also use a cloud repository as the remote.

What makes the well-known cloud repository services (e.g. Github, Gitlab, Bitbucket) particularly interesting is that not only do they provide a repository hosting service but they also provide an extensive service package built around the Git technology. Most of them have collaboration and communication features, wikis and issue tracking

capabilities, plugins for integration with other software products, continuous integration tools and even free or cheap hosting of websites. What's more, Github has managed to build a huge and active community that contributes to the success of lots of open-source projects.

3.5. **Server environment**

Having discussed how cloud is gaining ground in the industry, it is no surprise that in the past few years the biggest part of the server infrastructure that is behind the majority of the websites in the world has moved to the cloud too. Hosting a website on a cloud server has numerous advantages over traditional solutions and thus has become mainstream.

There are many popular providers that make a broad variety of of Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) services available to developers. I am not going to elaborate on all these different kinds of services and providers, but I would like to mention Amazon Web Services (AWS), Microsoft Azure, Rackspace, Google Cloud Platform, Heroku, DigitalOcean and OpenShift to name a few of the most popular providers.

To illustrate the wide range of solutions briefly, I am going to write a few examples about what is possible with Amazon Web Services. I am sure that these things can also be achieved by other providers, but AWS is the one that I personally have experience with.

Instead of buying and maintaining your own server, you can use the virtual private servers available in the AWS Elastic Compute Cloud (EC2) service. You can spin up new instances in minutes and can also tailor them to your specific needs (e.g. configuration of memory, CPU, instance storage and boot partition size). You have full control over the virtual server instances and can configure custom security groups and access levels. What is more, the Amazon EC2 Service Level Agreement ensures 99.95% availability which would be quite hard and expensive to achieve on your own server. You are not even bound to a single physical location because you can launch instances in quite a few different locations (so-called regions) around the globe. Domain name and DNS management is not a problem either thanks to the AWS Route 53 service.

AWS also makes it easy to scale up and down the performance of our instances and even to launch or terminate instances automatically based on a set of rules. They also

have a great load balancing service as well as monitoring and reporting services. Scalable content storage and content delivery network are also available in the set, as well as an email sending service (which takes a burden of installing a mail server off your shoulders), notification service, queue service, continuous deployment, container, database and cache services.

Even without going into details we can get a glimpse of how the various challenges can be solved with relative ease and low costs in a modern web development environment.

3.5.1. Containers

Beyond the virtualization and cloud dominance, containerization is also a very prominent trend nowadays. This technology is not new at all, but it wasn't very widely used before 2013 when Docker, Inc announced, launched and open-sourced its container engine.

"Container virtualization is often called operating system-level virtualization. Container technology allows multiple isolated user space instances to be run on a single host. (...) Docker is an open-source engine that automates the deployment of applications into containers. (...) So what is special about Docker? Docker adds an application deployment engine on top of a virtualized container execution environment. It is designed to provide a lightweight and fast environment in which to run your code as well as an efficient workflow to get that code from your laptop to your test environment and then into production. (...) Docker aims to reduce the cycle time between code being written and code being tested, deployed, and used. It aims to make your applications portable, easy to build, and easy to collaborate on." (Turnbull, 2014)

Since it is a great and useful technology, companies started adopting it rapidly and within less than 3 years it has become extremely popular.

3.5.2. Operating systems and web servers

Finally, let us take a look at what operating systems run typically on servers used for web development and which web servers are in use.

According to the data collected by W3Techs, as of 30 April 2016 67.8% of the websites using Unix-based operating systems (among which Linux is said to be the most

popular), 32.2% of the websites use Windows and Apple's OS X is used by less than 0.1% of the websites.

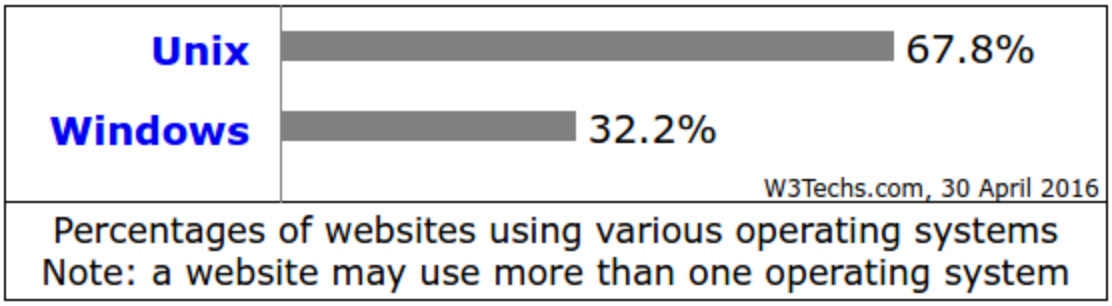

Figure 16. Usage of operating systems for websites. Source: W3Techs (2016)

From Netcraft's March 2016 Web Server Survey we can see that 32.4% percent of all the websites in the world were running on Apache webserver, not much ahead of Microsoft IIS. Nginx and Google Web Server (GWS) are also in the top 4.

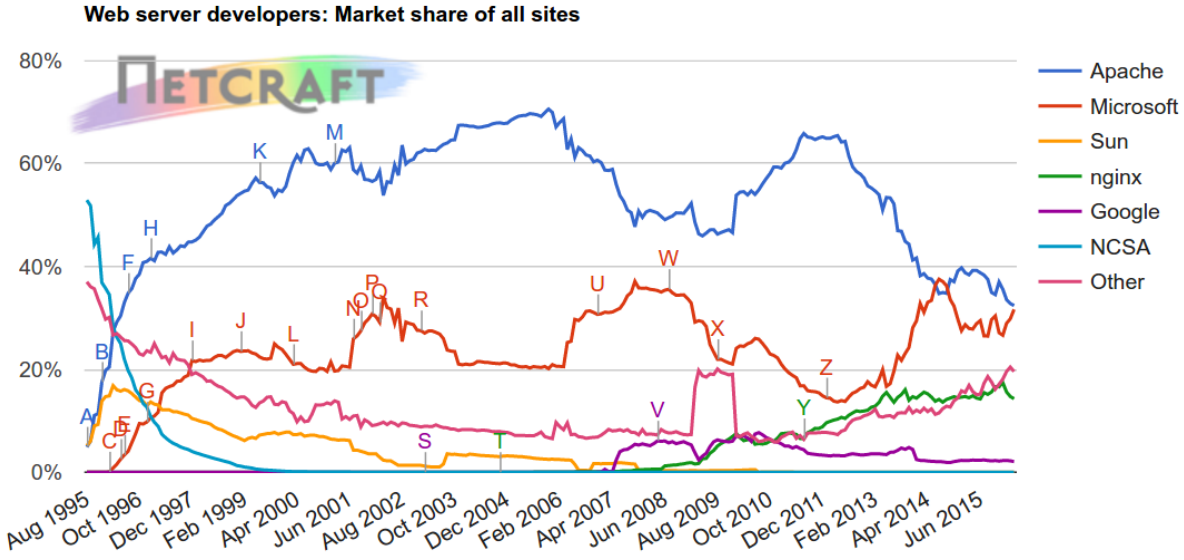

Figure 17. Web server developers: Market share of all sites. Source: NetCraft (2016)

Developer	March 2016	Percent
Apache	325,285,185	32.40%
Microsoft	317,761,318	31.65%
nginx	143,464,293	14.29%
Google	20,790,767	2.07%

Figure 18. Web server developers: Market share of all sites. Source: NetCraft (2016)

What is interesting is that these results are significantly different for the top million busiest websites:

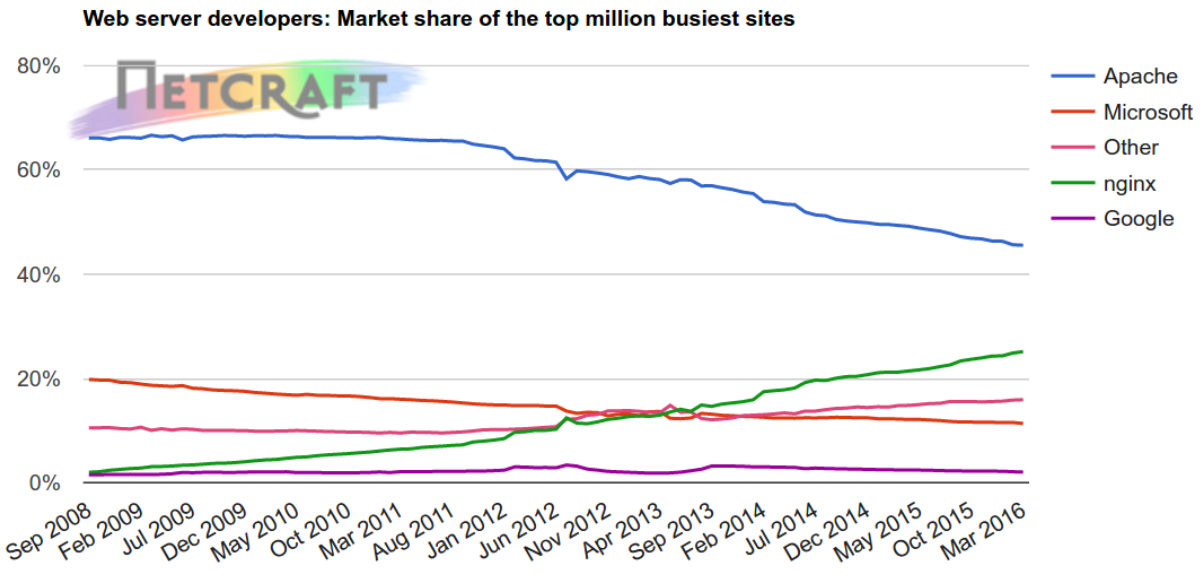

Figure 19. Web server developers: Market share of the top million busiest sites. Source: NetCraft (2016)

Developer	March 2016	Percent
Apache	455,428	45.54%
nginx	251,440	25.14%
Microsoft	113,585	11.36%
Google	20,266	2.03%

Figure 20. Web server developers: Market share of the top million busiest sites. Source: NetCraft (2016)

The chart also shows that nginx's market share is massively increasing among the top performers of the industry.

3.6. **Microservices**

The term microservices is the latest buzzword in web development that you hear about every day multiple times and there is no conference or meetup event where it is not mentioned or talked about. However, just that the term is overused, does not mean that it has lost its power - in fact, countless development teams are considering or are actively working on moving to a microservice-based architecture, not to mention the companies that had already embraced the idea years ago.

The concept is very much the same as what I have already written about in the Databases section (Section 3.2.3) in relation with Polyglot Persistence. The main idea is splitting the big monolith systems into smaller, independent modules (services) that communicate with each other (typically via RESTful interfaces).

"Microservices are an approach to distributed systems that promote the use of finely grained services with their own lifecycles, which collaborate together. Because microservices are primarily modeled around business domains, they avoid the problems of traditional tiered architectures. Microservices also integrate new technologies and techniques that have emerged over the last decade, which helps them avoid the pitfalls of many service-oriented architecture implementations. (...) Microservices are small, autonomous services that work together." (Newman, 2015)

Benjamin Wootton in a 2016 article writes about various benefits of microservices. For example, "in a Microservice architecture, you should be able to deliver new functionality and iterate on the system faster than you would be able to on a more monolithic architecture. (...) We can modify just one system component, test it, and then push it to production outside of any centrally mandated release cycle. This is a much faster and more agile way to ship new software features". In addition to the polyglot persistence, "because of the isolation and independence of the Microservices, individual services can be polyglot in terms of programming language [too], giving us the ability to use 'the right tool for the job'". He goes on saying that "in a Microservice world we can be much more flexible and scale individual services up and down as

necessary, giving the system a much more dynamic property that is well suited to an elastic cloud environment". Last but not the least, "both the Microservice architecture and the way it is usually approached typically gives us a high degree of resilience as a property of the system".

Needless to say that the rise of microservices brings about a breadth of new problems and challenges too, for instance network and operational complexity, data consistency and distribution issues and monitoring and debugging difficulties. Obviously these also drive the growth of new services and tools aiming to ease the pains of the developers and operation people.

3.7. **The API world**

When discussing the most important aspects of modern-day web development, the term API surely needs to be mentioned. APIs are ubiquitous and power so many of the websites and services that we use daily. And not only are they present on the web but in fact, "APIs aren't at all new; whenever you use a desktop or laptop, APIs are what make it possible to move information between programs". (Proffitt, 2013)

"API stands for application programming interface. (...) An API is a way for two computer applications to talk to each other over a network (predominantly the Internet) using a common language that they both understand. (...) There are APIs that are open to any developer, APIs that are open only to partners, and APIs that are used internally to help run the business better and facilitate collaboration between teams. An API, then, is essentially a contract. Once such a contract is in place, developers are enticed to use the API because they know they can rely on it. The contract increases confidence, which increases use. The contract also makes the connection between provider and consumer much more efficient since the interfaces are documented, consistent, and predictable." (Jacobson - Brail - Woods, 2012)

It probably helps understand a bit more what I wrote about in the 'Build your own or use an existing solution?' and 'Microservices' chapters. Developers can incorporate the extra knowledge or features provided by external services by sending requests to and receiving responses from their APIs and microservices also communicate with each other through APIs.

Today the web is an ever-growingly dense network of interconnected applications that rely on one another for pieces of information that they themselves do not possess.

To name a few, some widely used examples include the APIs of Google, Facebook, Twitter and Dropbox or the wealth of integrations available in the major project management, communication, customer relationship management, ERP, online payment, billing and continuous deployment software products.

Regarding the means to access a web service, there are two main approaches: SOAP (Simple Object Access Protocol) and REST (Representational State Transfer). As I see it, REST has overtaken and left SOAP behind by miles and is used far more often nowadays. Steve Francia in a 2010 article wrote that "the general rule of thumb I've always heard is 'Unless you have a definitive reason to use SOAP use REST'. (...) RESTs sweet spot is when you are exposing a public API over the internet to handle CRUD operations on data. REST is focused on accessing named resources through a single consistent interface. SOAP brings its own protocol and focuses on exposing pieces of application logic (not data) as services. SOAP exposes operations. SOAP is focused on accessing named operations, each implement some business logic through different interfaces. (...) Since REST uses standard HTTP it is much simpler in just about every way. Creating clients, developing APIs, the documentation is much easier to understand and there aren't very many things that REST doesn't do easier/better than SOAP."

He also notes that "REST permits many different data formats whereas SOAP only permits XML. (...) JSON usually is a better fit for data and parses much faster. REST allows better support for browser clients due to its support for JSON". What is more, "REST has better performance and scalability. REST reads can be cached, SOAP based reads cannot be cached".

One of the trends that I have personally seen in the PHP world is that microframeworks optimized specifically for APIs (e.g. Lumen, Slim, Silex) get more and more exposure due to the increasing demand for APIs.

The other one is that at some projects, beyond the MVC pattern's separation of concerns, there is a complete separation of the front-end and the back-end. How it works is that the back-end system knows nothing about view rendering or anything of that kind, it just simply provides a RESTful API. This way the front-end can be completely independent, developed by a different team and even located elsewhere. In a common case it would consist of a Javascript framework (e.g. Angular) that communicates with the back-end API.

3.8. **Software testing**

Back in 2008, Paul Ammann and Jeff Offutt in the book called 'Introduction to Software Testing' wrote that "not very long ago, software development companies could afford to employ programmers who could not test and testers who could not program. (...) Software testing in industry historically has been a nontechnical activity. Industry viewed testing primarily from the managerial and process perspective and had limited expectations of practitioners' technical training. As the software engineering profession matures, and as software becomes more pervasive in everyday life, there are increasingly stringent requirements for software reliability, maintainability, and security. Industry must respond to these changes by, among other things, improving the way software is tested".

This improved way of testing prefers automated, scripted testing over manual testing (although manual testing is still present and necessary in many cases) and also moves the responsibility of writing the testing code from a tester to the developer.

Software testing has gradually become more and more prominent in the web development scene too. Fortunately in the past few years the idea and practice of testing became mainstream and today we can say that the majority of professional web developers, especially the ones working on big projects write tests every day, as an integral part of their job. What is more, the test-driven development (TDD) approach, which means that first you must write a test that fails before you write the actual code, gets significant attention and popularity in web development too.

Profound software testing is crucial everywhere for quality assurance, but in my opinion automated testing is particularly important in the field of web development, because unlike other kinds of software products that ship their new versions in certain, relatively rare intervals, the deployment of code changes should be continuous in a web environment. I am going to write more about continuous deployment later on, but it is important to note that without proper automated testing, it would mean an incredibly high risk to roll out new versions not having some extent of confidence that every part works as it should. If our test suites cover the whole application (in an ideal world) and all the tests passed then we can be fairly sure that there no major drama is about to happen.

Writing tests takes up a big chunk of a developer's time and can seem tedious sometimes, but actually it saves a lot of time, effort and stress when it comes to altering complex parts or performing refactoring, not to mention the confidence and peace of mind during releasing code changes and the decrease in the number of bugs.

There are several testing frameworks available for PHP which make testing easier and more effective. The best-known solutions include PHPUnit, Codeception, Behat, PHPSpec and Selenium. There are also some great cloud services which make automated cross-browser website testing possible.

3.9. **Continuous Integration and Deployment**

Continuous Integration and Continuous Deployment are key terms in today's web development environment. The concepts do not mean the same thing but they are very closely related to each other and usually go together.

3.9.1. Continuous Integration

The great minds at ThoughtWorks explained Continuous Integration (CI) in an article as "a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early". Martin Fowler, Chief Scientist at ThoughtWorks adds that "Continuous Integration doesn't get rid of bugs, but it does make them dramatically easier to find and remove".

At first glance setting up and maintaining a process like that might seem like an overhead, but in reality, it can save a lot of time, energy and headache. I have learnt it the hard way through countless occasions of spending hours resolving conflicts that emerged during merging long-lived Git branches back to the main branch. Now, at JóSzaki, we push most of the commits right into the common develop branch and even if we need to create a separate branch for some reason, we keep it up-to-date and merge it back as soon as possible. Also whenever a new commit is pushed to the repository, the build and the tests are executed automatically and we can see straightaway if something goes wrong (we even get notifications to a Slack channel).

I love how the authors of the previously mentioned article put it into words: "Continuous Integration is cheap. Not continuously integrating is costly. If you don't follow a continuous approach, you'll have longer periods between integrations. This

makes it exponentially more difficult to find and fix problems. Such integration problems can easily knock a project off-schedule, or cause it to fail altogether."

Martin Fowler in his 2011 article uses a funny but self-explanatory chart to demonstrate how the frequency of integrations reduces the difficulty of each integration:

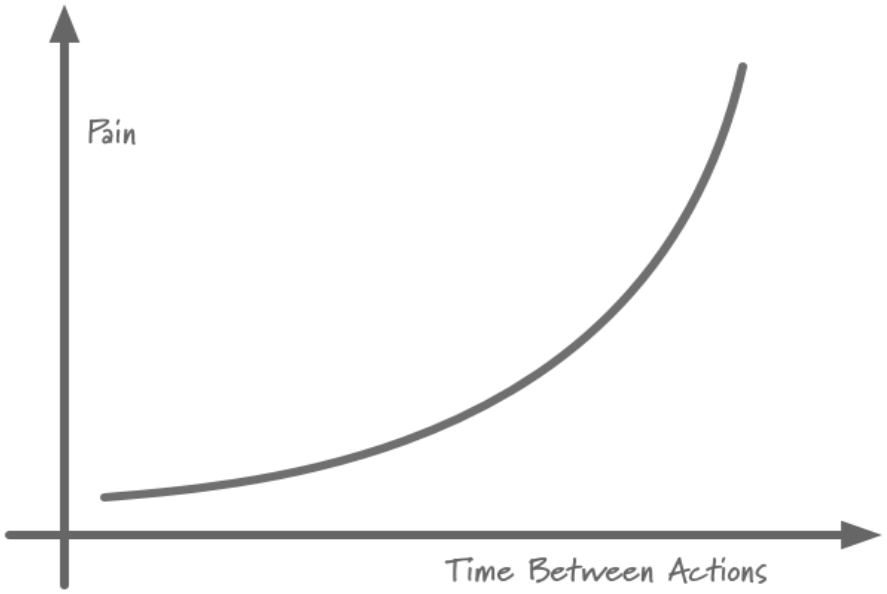

Figure 21. Frequency Reduces Difficulty. Source: Fowler (2011)

"If you have this kind of exponential relationship, then if you do it more frequently, you can drastically reduce the pain. And this is what happens with Continuous Integration - by integrating every day, the pain of integration almost vanishes. It did hurt, so you did it more often, and now it no longer hurts". (Fowler, 2011)

3.9.2. Continuous Deployment

Continuous Deployment (CD) is only about taking one more step in the same direction: it basically means that every commit pushed into the shared repository that passes the automated tests and produces a successful build gets automatically and instantly release (deployed) into the production environment. Here is a bit more in-depth explanation that reveals the background and significance of such an automated process:

"Most modern applications of any size are complex to deploy, involving many moving parts. Many organizations release software manually. By this we mean that the steps

required to deploy such an application are treated as separate and atomic, each performed by an individual or team. Judgments must be made within these steps, leaving them prone to human error. Even if this is not the case, differences in the ordering and timing of these steps can lead to different outcomes. These differences are rarely good. (...) Over time, deployments should tend towards being fully automated. There should be two tasks for a human being to perform to deploy software into a development, test, or production environment: to pick the version and environment and to press the “deploy” button. (...) The automated deployment process must be used by everybody, and it should be the only way in which the software is ever deployed. This discipline ensures that the deployment script will work when it is needed. One of the principles that we describe in this book is to use the same script to deploy to every environment. If you use the same script to deploy to every environment, then the deployment-to-production path will have been tested hundreds or even thousands of times before it is needed on release day. If any problems occur upon release, you can be certain they are problems with environment-specific configuration, not your scripts.” (Humble - Farley, 2010)

One of the reasons why I really like this approach is that the quick fixes, minor improvements and small features that used to wait in the queue for days or weeks can now be deployed almost immediately and beyond its positive implications for quality, in some way it even gives the developers a sense of progress and impact - they can see the result and consequences of their work right away in production. On the one hand, it eases the pressure to get everything perfect because you can make very short iterations and these “not-yet-perfect” versions give you instant feedback and reveal the accidental mistakes. On the other hand, developers are urged to ensure consistent quality and releasable code at all times, because poorly written code fragments are likely going to bring about issues straightaway.

These are quite a few popular continuous integration tools available that make it very easy to setup a highly automated process. Most of these tools can observe certain events in our cloud Git repository and respond with custom actions like running the test cases or starting the deployment process, access our cloud server instances and perform changes necessary for deployment and even communicate with a number of external services (e.g.: send the new version number to Sentry, update tickets in the project management systems, send notifications to a Slack channel, etc…). Some of the

most popular continuous deployment software are Jenkins, Travis CI, Codeship, CircleCI and GitLab CI.

We can see that this topic brought together many previously discussed things, for instance the cloud technologies, server infrastructure, version control, APIs and software testing. These can all come together very nicely and can thus result in a modern and highly efficient environment. The key question is how you can integrate these tools and services in order to get a powerful combination of them. It is all about synergy, when the sum of the power of the parts is less significant than the actual power they produce when they all come together. This positive synergy is also called the 2 + 2 = 5 effect (Reference for Business, Encyclopedia of Management). This means that it is not sufficient to choose good tools and services, but we also need to understand what they are designed for and how they will cooperate with all the other things that are in use in our project. The ultimate goal is to achieve an environment that enables web developers and web developer teams to maximize their potential.

3.10. **DevOps culture**

Now, at the end of the 'The technology trends' chapter, I would like to mention a phenomenon that is less of a technical and more of a cultural and organizational matter by nature.

The evolution of the so-called DevOps culture has a significant effect on a web developer's life and daily tasks. Traditionally development used to be somewhat separated from operations, maintenance and deployment activities, but this situation is being transformed into a new set-up where either there is a strong and active collaboration between development and operations or developers even take over a part of or all of these tasks.

"An attitude of shared responsibility is an aspect of DevOps culture that encourages closer collaboration. It's easy for a development team to become disinterested in the operation and maintenance of a system if it is handed over to another team to look after. If a development team shares the responsibility of looking after a system over the course of its lifetime, they are able to share the operations staff's pain and so identify ways to simplify deployment and maintenance (e.g. by automating deployments and improving logging). They may also gain additional observed

requirements from monitoring the system in production. (...) DevOps culture blurs the line between the roles of developer and operations staff and may eventually eliminate the distinction.” (Wilsenach, 2015)

4. The business perspective

So far I've been writing mostly about what technology is available, what it looks like and what it is capable of doing. Although these solutions can be really exciting from a developer's point of view, we must also see that even if an ideal, perfect software and server environment existed, in and of itself it wouldn't be worth much without delivering value to the business. This is why I find it important to take a close look at what the main factors are from a business perspective and what the business can gain and benefit from the technology shifts.

4.1. **Scalability**

Scaling has been mentioned many times in this paper already, and it is a very important topic indeed. In this context scaling up basically means keeping the behavior of a website or web service unchanged despite the increasing load (which usually means the number of users). It comes into play either when the traffic is growing over time or when unusually high traffic hits the web server, for example thanks to a marketing campaign or some other event that causes a swift increase in interest. In most cases, increasing the computing capacity of the server (e.g. CPU, memory) solves the problem. Scaling down, on the other hand, means lowering the performance of the system when less computing power is enough.

There are two distinctively different types of scaling: vertical scaling (also called scaling up) and horizontal scaling (also called scaling out). A 2014 David Beaumont article explains it this way: "Vertical scaling can essentially resize your server with no change to your code. It is the ability to increase the capacity of existing hardware or software by adding resources. Vertical scaling is limited by the fact that you can only get as big as the size of the server. Horizontal scaling affords the ability to scale wider to deal with traffic. It is the ability to connect multiple hardware or software entities, such as servers, so that they work as a single logical unit. This kind of scale cannot be implemented at a moment's notice."

I drew a chart to illustrate the difference:

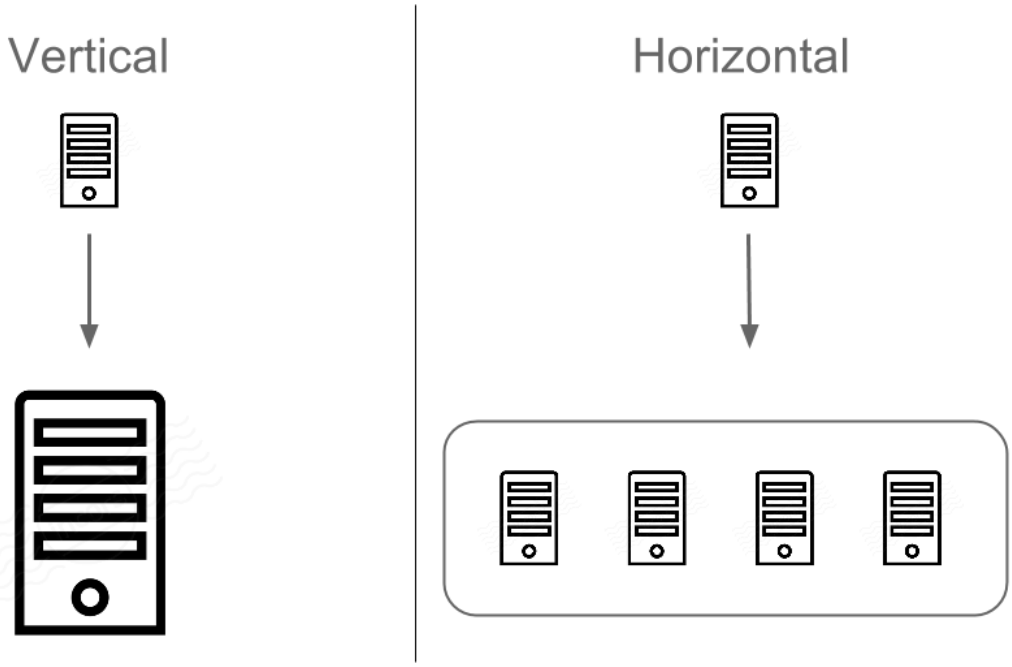

Figure 22. Vertical and Horizontal Scaling. (My own work, 2016)

According to Amir Shevat (2008), for “small scale application scaling up [vertical scaling] might be cheaper and faster to develop and implement” but “it is a costly and not an infinite solution, (...) there is a physical limitation to the computing power and memory you can have in a single computer”. For large applications horizontal scaling is better, because it “offers infinite scalability, when you need to support more users you just add more low cost computers to your server farms. On the other hand, this is not a straightforward solution. You need to design, architect, and develop your application to be ready to scale out”.

The great news here from a business perspective is that on a cloud server infrastructure (that I described in Section 3.5) horizontal scaling can be achieved easily (even automatically) and most importantly, cost efficiently, instead of spending a large sum on scaling up the application vertically. If the application is built on a microservice infrastructure, it is enough to launch new nodes of the services which are performance bottlenecks, as opposed to scaling up the entirety of a monolithic application. The performance, resources and costs can be tailored to the actual demand and needs. For example, during marketing campaigns or in times of the year when we expect higher traffic we can scale up the application while in less busy periods we can scale it down.

It makes it easy to introduce new services or features or to expand the service to new areas or even to new countries or continents. In case of a geographic expansion a great solution could be to spin up new instances in another availability zone which is closer to the newly targeted areas to provide their users with low latency.

4.2. **Costs**

For businesses, operational costs are always of very high importance, so I find it worthwhile to take a look at how the costs are affected and changed by the direction in which web development is moving.

Very much the same way as in other industries, we can say that automation in general frees up human resources and thus saves money. It is worth the effort to provide the developers with an environment where they can effectively and efficiently collaborate with each other and can create outstanding value. This can be achieved by enabling the developers to focus on the things that they truly care about and that they are best at – instead of spending their expensive and valuable time on tedious, repetitive tasks that could be automated or outsourced.

The wave of cloud servers and other cloud services has managed to diminish the maintenance costs by making in-house servers, devices, tools, network and data storage unnecessary. By that I do not only mean the actual cost of the hardware but also the cost of the needed software and the cost of the workforce who is responsible for configuring and maintaining the infrastructure.

“Cloud-based services can help you save money on many fronts, including server maintenance, power and cooling costs, and software licensing and upgrade expenses. (...) Rather than spending money to maintain hardware that often goes unused, subscribing to software and services for a low monthly fee can help small businesses stretch their budgets further.” (American Express Company, 2011)

On top of the changes in cost efficiency, the cost structure has also changed. Some of the typical fixed costs have turned into variable cost due to the rise of ‘pay as you go’ cloud services. To clarify: “fixed costs [are] not tied to production [while] variable costs fluctuate according to how much you produce”. (Thompson, 2015) What it means for web development projects is that a significant portion of the total costs depend on the traffic and performance of the application. If your website or web services generates money for you also on a per user or per performance basis, then in very simple terms

we can say that you pay less when you have less income and you pay more when you have higher income. Its advantage is that the barrier to entry goes down in the industry and new players can start building their projects with relatively low costs and they only need to pay more when their projects turn out to be successful.

Cloud computing and the use of third-party tools and services is ideal for companies big and small because they save cost and give extra convenience and allow you to focus on what you do best. Where I personally see a distinction though is the use of proper automated testing, code quality assurance, reporting and monitoring tools and automated deployment solutions. I feel that in the field of web development teams developing their own products used to be more likely to incorporate these practices because for them the long-term effects are more important and therefore they were willing to invest time, money and learning into these to reach better quality and efficiency. On the contrary, teams working on client projects tended to have the “get it done as quickly as possible” attitude due to the deadlines and the lack of emotional attachment to the projects. In my opinion thanks to the rapid growth and advancement in the field, these practices are no longer considered to be a luxury and more and more companies in diverse fields can afford to have them and realize that the investment breeds actual competitive advantage. What I am trying to say is that the necessary investment both in terms of money and knowledge has dropped and now practically anyone has access to these services either for free or at a low price and it has become so easy to start using them that even novice developers can utilize them.

4.3. **Quality**

Even though price, speed of delivery and other factors often get higher priority than quality, I still insist that quality is very valuable to the right customers and high quality is a great value that the business can sell. This is why it is important for the business that the web developers create a quality product that in return creates value for the customer and thus to the business.

I have already written about how automated tests reduce the number of times when something unexpected happens and makes it easier to spot faulty parts, how using third-party software that is widely used, tried and tested can decrease the number of bugs and leaks and how we can achieve more consistency by automating complex processes are extremely prone to human error. These all help the business to have a strong value proposition based on quality that appeals to the customer.

Sommerville collected a number of software quality attributes in the ninth edition of the Software Engineering book (2011), and these apply to web development and web products too:

Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learnability

Figure 23. Software quality attributes. (Sommerville, 2011)

We can see that many of these attributes, like robustness, reliability, testability, resilience, security, modularity, portability, reusability and efficiency are addressed and improved by the technology shifts discussed earlier in this paper.

4.4. Internal processes

4.4.1. Metrics

The new set of tools and services that I demonstrated in the technology trends chapter make it possible to monitor, measure, track and analyze systems and processes a lot more accurately. This is an important factor because in order to understand the actual performance of a system, team or company, you need good and reliable metrics. In recent years many companies have started consciously sticking to data-driven decisions, and obviously they apply this approach to the web development teams and projects too.

A commonly practiced way of setting goals and measuring results is defining KPIs (Key Performance Indicators). With these advanced tools and services a lot of data is available automatically and thus gathering data for the metrics is not a big burden any longer. To name a few examples, deployment speed, deployment success rate, deployment frequency, rollback / regression frequency, test coverage, number of tests, test execution time, code duplication rate, availability, average response time, number

of reported bugs, number of runtime exceptions and development velocity (based on story points) can all be used as KPI of the web development team and the figures can easily be retrieved from the various continuous integration, monitoring and analytics tools.

We use several of these metrics at JóSzaki and they provide us with a really good base for comparison both to our previous results and to other teams, and they serve as indicators on our development speed and quality. Knowing our velocity, for instance, helps us with resource planning and also helps us draw conclusions about the efficiency of our methods and processes. What is more, we define team goals based on these metrics and seeing how we are improving adds an extra incentive and motivation to our daily work.

One last aspect I would like to mention in this topic is that web service providers can commit themselves to more realistic availability and performance criteria when they have reliable metrics about every part of their systems. Not only does it help when defining an SLA (Service Level Agreement) but it also helps to spot the bottlenecks and weakest links that need immediate attention.

4.4.2. Flexibility and velocity

Automation, cloud technologies and third-party solutions have all contributed to an increase in speed of development (velocity), and the flexibility is also improved by things like continuous integration and deployment. This increased agility is important for the business because it can mean shorter release cycles, flexible release options, the ability to satisfy urgent business needs almost instantly and the ability to fix business critical bugs and issues quickly.

In today's constantly changing and volatile environment it is essential to have the ability to run quick and short experiments and iterate easily on new features and improvements. The business and product development teams can come up with new ideas and can rapidly go through the "build - measure - learn" methodology introduced by the book called The Lean Startup (Ries, 2011). They can get their experiments to production in a minimum amount of time and can change their minds based on the results and can rollback or modify things.

On top of these advantages, the short cycles and frequent deployments also reduce the risks of a change. What is more, these often expose hidden inefficiencies and costs which can then be addressed and fixed.

4.4.3. People as Single Points of Failure

In the IT world the 'single point of failure' term is often used to refer to an "element or part of a system for which no backup (redundancy) exists and the failure of which will disable the entire system." (Business Dictionary, 2016) Obviously a single point of failure is a thing to be avoided because it threatens the security and stability of the whole systems.

Where is becomes interesting is that it does not only apply to computer systems but to organizations too. A person can easily be a single point of failure if their absence or departure would paralyze the organization or the actual computer system.

In 2013 Tomas Kucera wrote that "your responsibility as a leader is to identify key people and plan for their unexpected demise. The goal is not to have key people at all."

As I see it, many web development projects have transitioned from a state of using completely custom and unique, complex and unclean code bases to a state of using well-known frameworks even for bespoke software, relying on properly documented and widely used libraries and cloud services, adapting common principles and coding conventions, having extensive test suites and automated deployment processes. These somewhat standardized, documented and industry standard processes and tools result in an environment where workforce is a little bit easier to replace and dependency on a particular developer is reduced.

It is important to note that I do not think developers should be perceived and treated as cogs in a machine that can easily be replaced anytime, because they carry an incredible wealth of experience, knowledge and relationships specific to the organization. I like how Amy Rees Anderson in a 2013 article wrote that "great employees are not replaceable (...) it is great people that make a great company". I totally agree with it but I also know that what she writes later on is also true: "there will be some life events that take great employees away from a company, which cannot be stopped". This thought and all the cases when a developer is absent for a shorter or longer time amplify the need for an environment where people are less likely to be single points of failure.

4.4.4. Employer branding

I think this is a really important aspect but the meaning of the term might not be clear at first sight. This is how a recruitment guide on Realstaffing.com defines it: “An employer brand refers to the perceptions key stakeholders, and more specifically current and potential employees, have of your organisation. It is about how they view the company; from how you conduct yourselves in the market, through to what they think it would be like to work for your organisation. An effective employer brand presents your organisation as a good employer and a great place to work and can, as a result, help with recruitment, retention and generally affect market perception of your company.” (Realstaffing.com, 2016)

I believe that having a great environment can make the company more appealing and attractive to developers. What I mean by great environment in this case is that the developers can use advanced, cutting-edge technologies and tools, they are encouraged and supported to follow the trends and keep learning and experimenting, and their mundane tasks are automated so that they can spend their time with exciting challenges.

As a consequence, it will be easier to attract and hire talent and retain existing employees. It is of particularly high importance nowadays as companies are faced with scarcity of quality workforce in the field of web development. As far as I know this is happening all over the world in the whole IT industry, but what I am sure about is that this is the case in Hungary.

According to an index.hu article published in March 2016 the Hungarian companies could employ 22.000 more people in IT positions while in the European Union there is a need for an additional 600.000 - 700.000 software developers. I haven’t found any data about the web developers in particular, but based on my own experiences, the situation is just about the same and companies find it difficult to hire good developers.

This is the end of ‘The business perspective’ chapter. We can see that not only does the technology change rapidly but businesses are also strongly affected by the wind of change. New opportunities and advantages emerge while businesses need to face new challenges and difficulties as well, such as privacy, security, responsibility and dependency concerns.

5. Summary

The few selected areas that I covered in this paper gave an overview the current state of web development and also gave a glimpse into what is in the toolbox of a modern day web developer. The rapid improvement and transformation of the technologies, best practices and tools make web development both a challenging and an exciting occupation.

As I see it, the main threads of the advancement are automation, cloud computing, distributed and scalable systems, and last but not least open-source software and extensive collaboration that joins forces, connects knowledge and experience and creates ambitious, far-reaching and valuable products and services.

We could also see that the technology shifts that we are experiencing have affected businesses too in numerous ways. Those who can adapt to this ever-changing environment and can take advantage of the opportunities have a great chance of success as there is still a wealth of unleashed potential on the web. This is where this paper can help – by becoming familiar with the trends and becoming well-versed with the available technology, both individuals and businesses can gain a deeper understanding of the web development environment and will be able to spot and exploit opportunities.

Since I did not go into the specific software engineering details, the technology overview is suitable even for non-developer participants of web development projects. Knowing how today's technology works and how the different areas relate to each other can help them identify with the challenges and struggles of developers and can also help them originate innovative ideas.

I am fairly sure that the presented evolution of the web is going to continue. We are just about to go deeper into the era of the Semantic Web (Web 3.0) which connects knowledge and then over time we are going to get closer to the Web 4.0 era, the era of The Ubiquitous Web (4.0) which is forecasted to connect intelligence to an extent that is yet unknown.

I also expect that Artificial Intelligence (AI) solutions are going to become even more common and useful and will be present in almost every segment of the web development landscape. Two examples of its manifestation are communication and

integration bots (which are already apparent in many web application, like Slack and Facebook Messenger) and intelligent software robots that are likely to replace human workforce in trivial, repetitive web development tasks.

6. List of figures

7. References

American Express Company (2011): OPEN Insight Guide - Running Your Business in the Cloud
https://c401345.ssl.cf1.rackcdn.com/pdf/OPEN_Savings_Cloud_Insight_Guide.pdf
Downloaded on 22 April 2016

Ammann, P. - Offutt, J. (2008): Introduction to Software Testing, Cambridge University Press

Anderson, A. R. (2013): Great Employees Are Not Replaceable
http://www.forbes.com/sites/amyanderson/2013/02/13/great-employees-are-not-replaceable/#16db46407230
Downloaded on 7 April 2016

Beaumont, D. (2014): Thoughts On Cloud
http://www.thoughtsoncloud.com/2014/04/explain-vertical-horizontal-scaling-cloud/
Downloaded on 20 April 2016

Business Dictionary: single point of failure
http://www.businessdictionary.com/definition/single-point-of-failure.html
Downloaded on 7 April 2016

Chacon, S. - Straub, B. (2014): Pro Git, Second Edition, Apress Media LLC

David, M. (2008): Summary of Project10X's Semantic Wave 2008 Report: Industry Roadmap to Web 3.0 & Multibillion Dollar Market Opportunities
http://www.itu.dk/people/cmmm/Mills%20Davis%20Web%203.0.pdf
Downloaded on 17 April 2016

Griffith, E. (2015): What Is Cloud Computing?
http://www.pcmag.com/article2/0,2817,2372163,00.asp
Downloaded on 19 April 2016

Fowler, M. (2011): FrequencyReducesDifficulty
http://www.martinfowler.com/bliki/FrequencyReducesDifficulty.html
Downloaded on 3 April 2016

Francia, S. (2010): REST Vs SOAP, The Difference Between Soap And Rest
http://spf13.com/post/soap-vs-rest
Downloaded on 28 April 2016

Humble, J. - Farley, D. (2010): Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley Professional

Jacobson, D. - Brail, G. - Woods, D. (2012): APIs: A Strategy Guide, O'Reilly Media

Kepes, B. (2015): New Stats From The State Of Cloud Report
http://www.forbes.com/sites/benkepes/2015/03/04/new-stats-from-the-state-of-cloud-report/#4797721126f9
Downloaded on 19 April 2016

Kucera, T. (2013): How to avoid "single point of failure" situations in your team?
https://thegeekyleader.com/2013/07/28/how-to-avoid-single-point-of-failure-situations-in-your-team/
Downloaded on 7 April 2016

Megyesi, M. (2012): Why Frameworks? https://blog.8thlight.com/myles-megyesi/2012/09/12/why-frameworks.html
Downloaded on 21 April 2016

Millares, G. (2015): Top 5 Programming Languages Used In Web Development
http://blog.stoneriverelearning.com/top-5-programming-languages-used-in-web-development/
Downloaded on 14 April 2016

Moock.org: Model/view/controller design pattern ("MVC")
http://www.moock.org/lectures/mvc/
Downloaded on 19 April 2016

Morrow, K. (2014): Web 2.0, Web 3.0, and the Internet of Things
http://www.uxbooth.com/articles/web-2-0-web-3-0-and-the-internet-of-things/
Downloaded on 15 April 2016

Netcraft (2016): March 2016 Web Server Survey
http://news.netcraft.com/archives/2016/03/18/march-2016-web-server-survey.html
Downloaded on 30 April 2016

Netcraft - Internet Live Stats (2016): Total number of Websites
http://www.internetlivestats.com/total-number-of-websites/
Downloaded on 15 April 2016

Newman, S. (2015): Building Microservices, O'Reilly Media

OpenSource CMS (2016): CMS Market Share
http://www.opensourcecms.com/general/cms-marketshare.php
Downloaded on 25 April 2016

Proffitt, B. (2013): What APIs Are And Why They're Important
http://readwrite.com/2013/09/19/api-defined/
Downloaded on 28 April 2016

Rakowski, K. (2011): Getting Started With PHP Templating
https://www.smashingmagazine.com/2011/10/getting-started-with-php-templating/
Downloaded on 21 April 2016

Realstaffing.com: Building a compelling employer brand
http://www.realstaffing.com/employers/recruitment-guides/building-a-compelling-employer-brand
Downloaded on 8 April 2016

Reference for Business: Synergy
http://www.referenceforbusiness.com/management/Str-Ti/Synergy.html
Downloaded on 2 April 2016

Ries, E. (2011): The Lean Startup, Crown Business

Sadalage, P. (2014): NoSQL Databases: An Overview
https://www.thoughtworks.com/insights/blog/nosql-databases-overview
Downloaded on 17 April 2016

Sadalage, P. - Fowler, M. (2012): NoSQL Distilled - A Brief Guide to the Emerging World of Polyglot Persistence, Addison-Wesley Professional

Serra, J. (2015): What is Polyglot Persistence?
http://www.jamesserra.com/archive/2015/07/what-is-polyglot-persistence/
Downloaded on 17 April 2016

Shevat, A. (2008): Scale out versus scale up – How to scale your application
http://spacebug.com/scale-out-versus-scale-up-html/
Downloaded on 1 May 2016

Skvorc, B. (2015): The Best PHP Framework for 2015: SitePoint Survey Results
http://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/
Downloaded on 21 April 2016

SolidIT (2016): DB-Engines Ranking
http://db-engines.com/en/ranking and http://db-engines.com/en/ranking_trend
Downloaded on 17 April 2016

Sommerville, I. (2011): Software Engineering, Ninth Edition, Pearson Education, Inc.

Sosinsky, B. (2011): Cloud Computing Bible, Wiley Publishing, Inc., Indianapolis

Spivack, N.: Web 3.0: The Third Generation Web is Coming
https://lifeboat.com/ex/web.3.0
Downloaded on 15 April 2016

Stack Overflow (2016): Developer Survey Results
http://stackoverflow.com/research/developer-survey-2016
Downloaded on 14 April 2016

Stubnya, B. (2016): Nem tüntetnek érte, de a jövő múlik rajta
http://index.hu/gazdasag/2016/03/04/informatikushiany_munkaeropiac_oktatas_informatika/
Downloaded on 1 May 2016

Thompson, M. (2015): Fixed and Variable Expenses: What Do They Mean for Production?
http://www.business.com/finance/fixed-and-variable-expenses-what-do-they-mean/
Downloaded on 7 April 2016

ThoughtWorks: Continuous Integration
https://www.thoughtworks.com/continuous-integration
Downloaded on 29 April 2016

Turnbull, J. (2014): The Docker Book
http://books.linuxfocus.net/files/books/James.Turnbull.The.Docker.Book.Containerizati

on.is.the.new.virtualization.B00LRROTI4.pdf
Downloaded on 29 April 2016

Vergara, D. (2012): Version Control Systems: Distributed vs. Centralized
http://oshyn.com/software-development/version_control_systems_distributed_vs_centralized
Downloaded on 24 April 2016

Wallop, H. (2014): Gen Z, Gen Y, baby boomers – a guide to the generations
http://www.telegraph.co.uk/news/features/11002767/Gen-Z-Gen-Y-baby-boomers-a-guide-to-the-generations.html
Downloaded on 15 April 2016

Wilsenach, R. (2015): DevOpsCulture
http://martinfowler.com/bliki/DevOpsCulture.html
Downloaded on 30 April 2016

Wootton, B. (2016): The Benefits Of Microservices
http://sendachi.com/2016/microservices/the-benefits-of-microservices
Downloaded on 30 April 2016

W3Techs (2016): Most popular content management systems
http://w3techs.com/
Downloaded on 29 April 2016

W3Techs (2016): Usage of operating systems for websites
http://w3techs.com/technologies/overview/operating_system/all
Downloaded on 30 April 2016