Estimating Effort for Cross-platform Web Application Development

Abstract

The ubiquity of ever faster Internet connection and the democratization of Internet of Things (IoT) devices have significantly contributed to the proliferation of web applications. Such applications transcend software and hardware boundaries allowing user interaction in innovative ways that were unheard just a decade ago. Modern web applications are aware of their surroundings through the use of Global Positioning System (GPS), gyroscope and accelerometer sensors; are human friendly and communicate naturally using gesture, voice and touch; understand if the user is accessing the application while driving a car, sitting at home or jogging outdoors.

Understanding the particulars of different platforms and the limitations from a multiverse of devices requires discipline, technical knowhow and toil. It requires methodology that comprises the tools to correct identify, classify and calculate effort. This paper proposes a better model for cost and time estimation for web applications, as well as discussing characteristics that are often unaccounted using established effort estimation methods. Special focus is placed on cross-platform and user context requirements, this paper argues that the cost and time spent on these two areas are often underestimated due to the lack of a complete framework in the current literature.

Three established effort estimation methods are compared; Web Model (WEBMO), Cost Estimation, Benchmarking and Risk Analysis for Web (Web-COBRA) and Object Oriented Method Function Points for Web (OOmFPWeb). Using OOmFPWeb as the base methodology this paper demonstrates how it can be extended to account for cross-platform requirements by adding a new model into OOmFPWeb framework. This model is called Platform Model and is used to quantify cross-platform effort into measurable units.

Keywords

Effort Estimation, Mobile Applications, Internet of Things, Object Orientation, Software Measurement
Contents

1. Introduction
2. Unique Challenges of Web Application Projects
   2.1 Disambiguation
   2.2 Cross-platform availability
   2.3 Browser Fragmentation
   2.4 User Interaction
   2.5 Graphical User Interface
   2.6 Hardware constrains
   2.7 Network constrains
   2.8 Storage constrains
   2.9. Web application development effort
3. Review of available web effort estimation methods
   3.1 Reviewed method: WEBMO
   3.2 Reviewed method: Web-COBRA
   3.3 Reviewed method: OOmFPWeb
4. Proposed Solution
   4.1 Defining the Chosen Model
      4.1.1 Identifying Functional Requirements
      4.1.2 The OOmFPWeb Conceptual Models
      4.1.3 The Platform Model
   4.2 Measuring Functional Requirements
      4.2.1 Identifying application boundary and scope
      4.2.2 Identifying data and Transaction candidate functions
      4.2.3 Counting data and transaction functions
5. Applying the Chosen Model
   5.1 Defining a Hypothetical Web Application
   5.2 Identifying Application Boundary
   5.3 Identifying Transaction Functions
   5.4 Identifying Data Functions
      5.4.1 Identifying Object Model Units
      5.4.2 Identifying Dynamic Model Units
      5.4.3 Identifying Functional Model Units
      5.4.4 Identifying Navigation and Presentation Model Units
      5.4.5 Identifying Platform Model Units
   5.5 Counting Data and Transaction Functions
6. Conclusion and Further Work
Appendix
References
1. Introduction

It is hard to estimate the total number of Internet connected devices with a browser and the ability to access web applications. Ericsson estimates that there will be by 2021 around 28 billion connected devices [1], many of those devices will be able to access applications via a web browser. As the IoT ecosystem matures and diversifies, the way we interact with those devices will also evolve. We already see today a trend in the use of voice controls [2] where the user can interact with a web application using only voice commands.

The global trend towards more connected devices will likely correspond to a rise in complexity for web application development, which should increase in parallel to the number of platforms that offer Internet access via web browser. These new applications compel methodology that takes in consideration web software cross-platform characteristics and offer new tools that can quantify the complexity of implementation. Available software effort estimation methods were conceived decades before the popularization of mobile and IoT devices, consequently those methods have not been initially designed with modern cross-platform requirements in mind. This paper argues that web applications require a different effort estimation method, a method that can translate cross-platform and user context requirements into numerical quantifiable units.

Cross-platform requirements are all the logical evaluations, storage, or data presentation, that are necessary to enable an application be accessed on different platforms. This paper argues that those requirements are unique to modern web software, and that available effort estimation methods are unable to comprehensively account for those peculiarities and limitations. User context requirements are all the logical evaluations the software performs to understand what the user is doing while interacting with the application. This context awareness can then be used to personalize how the software interacts and displays information. This paper argues that context awareness is mostly overlooked by the current literature and that context artefact units are unaccounted.

Hence, the main objective of this paper is to present a proposal for an estimation model for web applications, as well as discuss the characteristics that are unaccounted for using available effort estimation methods. Chapter 2 discusses in details the technical requirements and limitations web application projects need to overcome when creating cross-platform and context aware software. Chapter 3 evaluates three effort estimation methods, Reifer Web Development: Estimating Quick-to-Market Software [17], Ruhe et al. Cost Estimation for Web Applications [18], and Abrahão et al. Evaluating a Functional Size Measurement Method for Web Applications: An Empirical Analysis [4]. Chapter 4 introduces the Platform Model to OOmFPWeb effort estimation framework. Chapter 5 demonstrates how to apply the new OOmFPWeb framework using a hypothetical web application. Finally, in chapter 6, are the final considerations and conclusions of this paper.
2. Unique Challenges of Web Application Projects

In this chapter, some of the more challenging requirements of web application projects will be discussed. Many of these requirements did not existed a decade ago, they reflect the present consumer market were software is omnipresent, and interaction can happen in many forms from countless devices. Each section concentrates on a specific cross-platform development challenge that highlights the difficulties software developers and project managers must overcome when estimating development effort.

The potential of building once and deploying everywhere independent of hardware or operation system (OS) allowing organizations to directly reach customers is the driving benefit of web applications. Although web applications currently underperform devices native applications, the lower cost and cross-platform availability might prove essential for organizations trying to support large number of customers without writing platform-specific implementations of their application [6]. The popularity of web applications has continuously growth through the years. According to a global online survey commissioned by Sencha (2015) with 120 senior leaders in North America and Western Europe, there is a shift away from native application development as 27% of organizations plan to move from native to web technologies [7].

While cross-platform software has clear benefits, the development and delivery challenges are more unclear due to the speed technology evolves. Existing platforms and future platforms that will be created in the upcoming years, require a multitude of platform-specific software optimizations. A well-designed cross-platform application, takes into account hardware and software limitations while offering an optimal user experience (UX) that is tailored for each platform. Mobile devices have distinctive particularities like touch screen and limited display resolution, while Smart TVs have entire opposite characteristics.

2.1 Disambiguation

The web application term is often broadly used to categorize all applications that are available via the Internet. This paper distinguishes between two groups of web-based applications, the first called web applications, and the second cloud applications. Web applications are exclusively designed to be accessed via web browser making then platform independent and fully self-contained inside a browser window. The technology used is a combination of server-side and front-side scripting. In the server-side, programming languages like PHP, C#, Ruby or Java are commonly used, while in the front-side the technologies applied are a combination of Javascript, HTML and CSS. The web browser acts as a thin client performing lightweight computation, just-in-time compilation (JIT), and information presentation. Complex computation is performed by the web server or several web servers, thus using less hardware and software computation from the end user device. Using this architecture, the web application can be available to a broader number of devices with hardware limitations such low processing power, limited memory, and restricted local storage.
Cloud applications have similar characteristics as web applications and can provide the same features. Anyhow cloud applications offer more advanced services such as multi-tenancy. They can be equally accessed via a web browser or any other application using Hypertext Transfer Protocol (HTTP). Cloud applications do not necessary have a front-side graphical user interface (GUI), instead data Input and output (I/O) is performed via an application programming interface (API).

2.2 Cross-platform availability

Cross-platform availability is the biggest differentiator between traditional software and web software. To achieve seamless user experience on multiple devices the developer must overcome series of platform specific hardware and software limitations. He must understand the different contexts the user interacts with the application, and then, dynamically tailor the experience according to the device.

An exhaustive list of all different platforms able to access web applications via web browser would be cumbersome to compile because new devices are continuously being released into the consumer market. Table 1 illustrates how diverse the connected device ecosystem is [5], the more popular devices are mobile phones, tablets and desktops. Anyhow as hardware component costs decrease and Internet connection becomes common place, web applications will exponentially be used from other platforms. While web software is promptly available on any device through a web browser, it has also unique challenges. Handling the peculiarities of each device can be complex and time-consuming. Developers need to test different devices, understand distinctive packaging details, evaluate diverse network performance, optimize I/O, and the context in which the application is used.
Table 1. Existing platforms that offer a web browser

<table>
<thead>
<tr>
<th>Platform</th>
<th>Network</th>
<th>Input/output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop computers</td>
<td>WiFi, Ethernet, 2G, 3G, 4G, LTE</td>
<td>keyboard, pointing device, voice, gesture, touchscreen</td>
</tr>
<tr>
<td>Mobile phones</td>
<td>WiFi, 2G, 3G, 4G, LTE</td>
<td>touchscreen, voice</td>
</tr>
<tr>
<td>Tablets</td>
<td>WiFi, 2G, 3G, 4G, LTE</td>
<td>touchscreen, voice</td>
</tr>
<tr>
<td>Smart TVs</td>
<td>WiFi, Ethernet</td>
<td>keyboard, pointing device, voice</td>
</tr>
<tr>
<td>Game Consoles</td>
<td>WiFi, Ethernet</td>
<td>Keyboard, voice</td>
</tr>
<tr>
<td>Wristwatches</td>
<td>WiFi, Bluetooth, 2G, 3G, 4G, LTE</td>
<td>touchscreen, voice</td>
</tr>
<tr>
<td>Ebook readers</td>
<td>WiFi, 2G, 3G, 4G, LTE</td>
<td>touchscreen</td>
</tr>
<tr>
<td>Digital cameras</td>
<td>WiFi</td>
<td>touchscreen</td>
</tr>
<tr>
<td>Cars</td>
<td>WiFi, 3G, 4G, LTE</td>
<td>touchscreen, voice</td>
</tr>
<tr>
<td>Home appliances</td>
<td>WiFi, Ethernet</td>
<td>touchscreen, voice</td>
</tr>
<tr>
<td>Smart glasses</td>
<td>WiFi, Bluetooth</td>
<td>gesture, voice, touch</td>
</tr>
</tbody>
</table>

Software developers will gradually rely more on methodology that correctly takes in consideration cross-platform requirements, even if the application only target a specific platform. The end-user is not always aware that a web application has been designed and optimized only for a limited set of devices. He is unaware of development complexity and probable assumes that the application works on any device. If the user interacts with the web application from a device that was not tested by the developer, the user experience can be suboptimal as the application may fail to correct present information that should fit the device screen or only offers keyboard input where voice input would be ideal.

2.3 Browser Fragmentation

The browser main purpose is to present web resources [9], when combining these resources a developer creates a web application. The browser abstracts the complexities of the operation system (OS) while providing a standardized platform for the web application to be accessed. The ability of creating software for one platform and using on multiple platforms is more cost effective [6], but not easily achieved. Writing portable applications that accurately work across platforms is even more challenging due to multiple browser vendors and versions. Different browsers compile the application slightest different [8] due to variations in rendering engines, and even different versions of the same engine can lead to different results. As August 2016, there are 9 popular browsers globally [Figure 2]. The browser market leader is Google Chrome that on desktop alone has 4 different popular versions [Figure 3].
Web browsers and web standards are constantly evolving. The latest versions of modern browsers offer very consistent support for web standards [10]. Anyhow while web applications are hosted remotely server-side and are dynamically upgraded without end-user action, the browser application itself does require the user to manually upgrade the locally installed software. Because of this manual requirement, is common to find outdated versions of a vendor browser being used. Web developers are well aware of such shortcomings and apply different techniques to mitigate the problem. One of those tools is a web application that scores different browsers implementation of HTML5 technologies to a maximum of 555.
points [Figure 4]. From looking at the scoring board is clear that not all vendors and browser versions offer the same support for HTML5, the test does provide a breakdown detail of all evaluated features, anyhow understanding how all these variables will affect a specific web application often requires trial and error.

**Figure 4. Desktop browser HTML5 support comparison on Nov 2016 (Source: html5test)**

<table>
<thead>
<tr>
<th></th>
<th>Chrome</th>
<th>Opera</th>
<th>Firefox</th>
<th>Edge</th>
<th>Safari</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current</td>
<td>492</td>
<td>461</td>
<td>460</td>
<td>370</td>
<td></td>
</tr>
<tr>
<td>Older</td>
<td>492</td>
<td>489</td>
<td>466</td>
<td>377</td>
<td>364</td>
</tr>
<tr>
<td></td>
<td>482</td>
<td>479</td>
<td>466</td>
<td>354</td>
<td></td>
</tr>
<tr>
<td></td>
<td>456</td>
<td>309</td>
<td>312</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>394</td>
<td>907</td>
<td>266</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.4 User Interaction

Interaction with web applications often happens in a broader context than traditional applications. While traditional on-premise software has a well-defined user target and is confined to a specific platform, web applications are used in a wider context by many platforms. A good way to visualize this multi context interaction is by imagining a calendar application that can be accessed from a smart watch during outdoor running, or from a desktop computer when sitting at home. While running, the user has no access to a keyboard and the small screen resolution in the watch makes it impractical for typing, the user instead interacts with the application using voice commands. Anyhow when the same application is accessed from home on a desktop computer, the interaction is performed using mouse and keyboard. Only by implementing both I/O mechanisms the application can be fully user friendly and deliver an optimal experience.

Modern browsers and the latest web standards provide a variety of APIs capable of collecting different user inputs, this data is then transmitted to the web application that can dynamically detect the device type and optimal output mechanism. The latest version of HTML5 gathers data from a broad array of sensors present on mobile or non-mobile devices including compasses, gyroscopes and accelerometers [11]. An application that offers multiple types of interaction requires the developer to carefully evaluate the platforms, the browsers and the context in which the end-user accesses its services. Because not all browsers offer exactly the same technology parity [Figure 5] extra testing and troubleshooting is often required.
**2.5 Graphical User Interface**

Developing user friendly web applications requires careful planning due to the fragmented mobile technology landscape where device capabilities such as screen size, input types and display functionalities can vary dramatically from vendor to vendor [12]. There are different ways to design web application interfaces, the simplest approach would be enabling the web application UI to be responsive. Graphical elements would dynamically scale and resize to the device screen resolution, that approach works similar to the way traditional software operates. The second most expensive approach, is in creating multiple interfaces that are optimized to each platform where the application is being used. Popular web applications often combine resizing on desktop and a redesigned UI on mobile devices.

Device screen sizes vary greatly between platforms [Figure 6]. Tablets and mobile phone screens also rotate according to the way the user is handling the device, on these cases, extra testing is also required to guarantee GUI elements adjust according on both vertical and horizontal positions. Mobile devices are also known to be slow and often take more time to load web software [13], those differences in performance can be mitigated by creating multiple assets that are optimized for some of the most common screen resolutions. Hardware intensive UI elements such images, animations and video are transcode into multiple versions ranging from low to high quality, then logic in the web application associates the correct asset resolution to a specific device. In the case of a Smart TV assets can be presented in full high definition (HD), while on a mobile phone images are rescaled down to a lower resolution.

Where: greener the color highest the support, and red means no support.
2.6 Hardware constrains

A connected device hardware performance can play a significant part on how web applications perform, specially regarding the components associated with the rendering and processing of information. Studies have demonstrated that more powerful hardware reduces browser delays by accelerating OS services and network stack [13].

Web applications targeting broader consumer market penetration must consider how page loading times will potentially affect the user experience. Consumers tolerable waiting time for information is approximately two seconds [15], testing and adjustments of the web application components may be required to stay below the accepted waiting threshold. Those adjustments may change depending on device hardware. In mobile devices alone there is a significant difference in performance [Figure 7]. When adding more platforms, e.g. Smart watch, Smart TV and automobile the complexity in correct estimating how a web application performs increases exponentially.
Figure 7. Differences in JavaScript engine performance between mobile phones. Benchmark performed on Google Chrome and Internet Explorer browser. Higher is faster. (source: Google Octane)

The lifecycle of a connected device also need careful consideration. Consumer expectancy is that most electronics will last an average of 5 years, anyhow expectancy is different between products. Flat panel TVs have the longest expected life, reaching 7.4 years, while mobile phones have the shortest life expectancy at 4.7 years [14]. This disparity in the pace consumers upgrade to newer devices eventually leads to some platforms being much faster and using newer technology.

2.7 Network constrains

Web applications performance are highly dependent on network speed as the application components are hosted in the cloud. Underlining technology such 3G, 4G or DSL significantly impact user experience. Differences in network latency can lead to application issues such timeout of critical components and asynchronous failures. To mitigate these issues, when designing web applications is important to have good means for measuring existing latencies as well as simulating new latencies throughout all parts of development. Developers are required to spend considerable time trying to reproduce bandwidth and web application performance bottlenecks on diverse network technologies.

Network performance not only differs between devices, but also by geographic location. In European countries alone there is a pronounced difference in Internet speeds, from the fastest average connection in Sweden scoring 17.4 Mb/s to Italy being the slowest with a speed average of 6.5 Mb/s [Figure 8].
2.8 Storage constrains

As web applications become more sophisticated, information in the form of personal data (consumer-driven) and big data (enterprise-driven) needs to be secure handled and stored. Much of this data is stored in the cloud and accessed on demand from the application. Anyhow part of this data need to be stored locally by the browser. Storing locally non-sensitive data helps reduce Internet data usage, and in some cases, significantly improve performance. Using HTML5 persistent storage APIs can be particularly helpful when dealing with mobile Internet. Mobile phone contracts in many countries have restricted amount of Internet data transfer, wireless mobile technologies such 3G or 4G are limited and expensive.

Previous to HTML5 developers were limited by the lack of APIs for persistent storage. Today those tools are available, one of the most broadly used technologies is called WebStorage. WebStorage has a persistent storage component called LocalStorage, and temporary storage called SessionStorage which will be cleared after the browsing session expires. All modern browsers have implemented HTML5 and WebStorage standards, but the allowed amount of persistent storage varies [Figure 9]. Inconsistencies in data storage limits need careful evaluation and testing on multiple browsers, and even within the same browser on different versions the storage threshold vary.
Figure 9. Available Local Storage on different browsers. (source: [16] Kitamura, Eiji)

<table>
<thead>
<tr>
<th>Browser</th>
<th>Version</th>
<th>Available storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chrome</td>
<td>40</td>
<td>10 MB</td>
</tr>
<tr>
<td>Chrome Mobile</td>
<td>40</td>
<td>10 MB</td>
</tr>
<tr>
<td>Android Browser</td>
<td>4.3</td>
<td>2 MB</td>
</tr>
<tr>
<td>Firefox</td>
<td>34</td>
<td>10 MB</td>
</tr>
<tr>
<td>Firefox Mobile</td>
<td>34</td>
<td>10 MB</td>
</tr>
<tr>
<td>Internet Explorer</td>
<td>9-11</td>
<td>10 MB</td>
</tr>
<tr>
<td>Opera</td>
<td>27</td>
<td>10 MB</td>
</tr>
<tr>
<td>Firefox</td>
<td>34</td>
<td>10 MB</td>
</tr>
<tr>
<td>Firefox Mobile</td>
<td>34</td>
<td>10 MB</td>
</tr>
<tr>
<td>Safari</td>
<td>6-8</td>
<td>5 MB</td>
</tr>
<tr>
<td>Safari Mobile</td>
<td>6-8</td>
<td>10 MB</td>
</tr>
</tbody>
</table>

2.9. Web application development effort

There are clearly several differences and obstacles when developing cross-platform software, some of those challenges can be mitigated by effective planning while others are more ambiguous. Challenges such continuously testing one application on multiple platforms, each containing multiple devices, and each device containing unique software and hardware specifics; presents enormous challenges when trying to estimate effort, time and costs. There is a myriad of methodologies and tools currently available that help better plan and administrate those challenges. Some established methods have evolved to incorporate mobile devices into planning, anyhow as consumer market relentlessly matures from desktop to IoT and beyond, those methodologies are failing to keep pace.

Effort estimation Methodologies that have evolved to accommodate web application requirements will be discussed in the next chapter, those publications developed by different authors provides the framework this paper will use to highlight the current short comes and use as foundation for an improved methodology.
3. Review of available web effort estimation methods

A number of papers have been published regarding the necessity to upgrade traditional effort estimation methods for modern web-based software. This chapter discusses three publications that better fulfils the addressed problem. The selection was based on three criteria: the paper has been cited on multiple peer-reviewed publications, it is built upon a recognized standard, and embrace web application characteristics.

Among the selected papers, the most cited publication was Reifer Web Development: Estimating Quick-to-Market Software [17], it has been cited 233 times. The second most cited paper was Ruhe et al. Cost Estimation for Web Applications [18] with 123. The paper by Abrahão et al. Evaluating a Functional Size Measurement Method for Web Applications: An Empirical Analysis [4] was only cited 27 times, but scored high on the other two criteria.

Being built upon a recognized standard is of paramount importance to assure quality and reliability. Both Reifer WEBMO and Abrahão et al. OOmFPWeb measure systems are an extension of Albrecht Function Points [3]. A function point is a “unit of measurement” used to compute a functional size measurement (FSM) of software. Function Point is one of the most widely adopted methods and offer numerous ISO/IEC standards. The most widely used FSM method is the International Function Point Users Group (IFPUG) Function Point Analysis certified as ISO/IEC 20926:2009 [19]. Ruhe et al. Web-COBRA does not appear to be provided by any ISO certified organization.

The last criterion is that the measure system embraces web application characteristics [Table 2]. These characteristics are based on Reifer (2000) paper, but still valid today, the only addition is the cross-platform category. All methods scored equally in the 9 categories, anyhow all methods failed the cross-platform category. Reifer Web Objects was custom built to deal with these characteristics, while both Ruhe et al. Web-COBRA and Abrahão et al. OOmFPWeb methods cite Reifer work and develop their publications to deal with the same challenges.
Table 2. Characteristics of web development projects. (source: Reifer [17] pp. 2)

<table>
<thead>
<tr>
<th></th>
<th>WEBMO</th>
<th>Web-COBRA</th>
<th>OOmFPWeb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary objective is bringing quality products</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>to market as quickly as possible</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Typical project size is small (3-5 team members)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Typical timeline is 3-6 months</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Development approach employed is rapid</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>application development, gluing building</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>blocks together, prototyping</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primary engineering technologies used are</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>component-based methods, 4th and 5th generation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>languages (html, Java, etc.) visualization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(motion, animation), etc.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processes employed is ad hoc</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Products developed are object-based systems,</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>many reusable components (shopping carts, etc.),</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>few external interfaces, relatively simple</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>People involved are Graphic designers, less</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>experienced software engineers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cross-platform. Take into account the context</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>and device where that the application is being</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>used</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.1 Reviewed method: WEBMO

The WEBMO measurement system was developed by Donald J. Reifer in 2000 from his personal experience in traditional and web software projects. WEBMO enhances an earlier used algorithmic software model called The Constructive Cost Model (COCOMO II) and introduces a new metric called Web Objects. The COCOMO II model uses Source Line of Code (SLOC) as its underlining size metric, while Web Objects computes size by taking into account each element that make up a web application. Web Objects are a modification of Function Point counting tailored for Intranets and Internet environments, it combines the five traditional Functional Point predictors (internal logical files, external interface files, external inputs, external outputs, external inquires) plus four web-specific predictors (number of XML, HTML, and query lines, number of multimedia files, number of scripts, number of Web building blocks) into a single size metric.

The WEBMO cost model does not require precise knowledge about of cost drivers during development process, instead generic cost drivers can be defined in order to estimate in the early phase of the development. As the projects progresses and influential features mature, those are then better defined in the cost model. The main advantages of this model are the use of a mathematical formula for effort prediction and the presence of an extension.
mechanism allowing for the addition of new types of operands and operators [4]. WebMO mathematical formula is defined as follow:

\[
\text{WEBMO Model}
\]

\[
\text{Effort} = \Lambda \prod_{i=1}^{8} (\text{Size})^{P_i} \\
\text{Duration} = B(\text{Effort})^{P_2}
\]

Where: \( A \) and \( B \) = constants \( P_1 \) and \( P_2 \) = power laws \( cd_i \) = cost drivers \( \text{Size} \) = # web objects

WEBMO is a good attempt of predicting effort and cost of web-based applications, anyhow when it was first conceived the consumer device ecosystem was very limited. Back in the year 2000 web applications were restricted by technology and Internet connection. Between 2000-2015, global Internet penetration grew 7-fold from 6.5% to 43% and mobile broadband increased 12-fold since 2007 [21]. Web Objects, the underlining metric of WEBMO, is limited by implementation technology and the fact that it cannot be applied in the early stages of the web application development lifecycle.

Reifer model has demonstrated benefits when applied into non-complex web hypermedia systems [17], anyhow those systems are easier to predict than web applications, because web hypermedia systems are usually much smaller in terms of the expended development effort and the user do not have the ability to affect the state of the system on the web server [18]. Web application requirements today look exceptionally different and more complex than 15 years ago when desktop PC was the solo platform. WEBMO dated formula does not offer a clear mechanism for adding multiplatform operands or handle the complexity of web applications.

3.2 Reviewed method: Web-COBRA

Web-COBRA is an adaptation to the Web domain of the COBRA (COst estimation, Benchmarking and Risk Analysis) method developed by L. C. Briand, K. El Emam, F. Bomarius in 1998. To create a COBRA model the method combines experts knowledge and past project data, gathered in a controlled way, then use this information to adjust the estimations coming from a model that exploits a size measure as cost driver [22].

In order to obtain a COBRA model, the person calculating the effort, need information about three project factors. First, the project characteristics such as project type, application domain, and so on. Second, the size measure calculated in a consistent way among all projects. And third, the cost drivers describing the resources expected to influence the development effort. Based on these three factors from the casual model, a cost measurement is extracted for the current project development. Web-COBRA mathematical formula is defined estimating the magnitude of relative error (MRE) as a percentage of the actual effort for a project [18]:
In addition, there is a prediction level (Pred). This measure is a proportion of observations for a given level of accuracy:

\[
\text{Pred} (l) = \frac{k}{N}
\]

Web-COBRA greatest advantages is the use of COBRA method for the web domain by combining expert knowledge with data from a small number of projects to develop cost estimation models. Unfortunately, empirical data on the effectiveness of Web-COBRA has only been assessed to a limited dataset of 12 web applications. Ruhe et al. Web-COBRA model has improved on Reifer’s WEBMO by concentrating its empirical research on more complex web applications. It also takes advantages of Reifer Web Objects to predict a project’s development effort using a composite estimation method that utilizes expert knowledge and a limited amount of past project data. Anyhow Web-COBRA also fail to take into account multiplatform requirements because in 2003 when the model was developed, mobile and other IoT devices were scarcely available in the consumer market.

### 3.3 Reviewed method: OOmFPWeb

Abrahão et al. OOmFPWeb measure model is the more recent method from the 3 reviewed, it was published in 2004 and was based on a laboratory experiment which evaluates OO-Method Function Points for the Web. The method was designed to conform to the IFPUG (International Function Point Users Group) counting rules for FPA (IFPUG, 1999) and intended to be used within the context of web-based systems. OOmFPWeb provides a measure of functional size for web applications that are produced using Object-Oriented Web Solutions (OOWS) (Pastor et al., 2001), this method integrates navigational and presentational design with a classical OO conceptual modelling approach. OOmFPWeb works by generating a value based on the amount of functionality a web system provides to a user. The method evaluates functionality of web systems based on user-defined requirements encapsulated in the conceptual model of web application projects. This is a breakthrough from Reifer 2000 WEBMO model that rather is based solely on implementation artefacts created once the application has been fully developed [24].

Empirically evaluation of OOmFPWeb comes from a 2009 study performed by S. Abrahão and G. Poels [23], the analyses based on four data sets from a family of experiments conducted in Spain, Argentina and Austria demonstrate the method efficiency when compared to existing industry practices. The method was also perceived as easy to use and useful by the participants [23], even if OOmFPWeb is believed to consume more time than established Functional Point Analyses (FPA) [24]. Unfortunately, because OOmFPWeb is a new procedure that is not yet widely used in practice, the authors conducted their research on an artificial laboratory setting using student participants rather than practitioners.
The OOmFPWeb functional size measurement requires two steps of abstraction, called the identification step and the measurement step. In the identification step a conceptual schema is built based on defined user requirement specifications (Data, Process, Behaviour, Navigation and Presentation). In the measurement step all relevant identified elements of the web application are defined as Base Functional Component (BFC) types, as in the ISO/IEC standard for functional size measurement [25]. Next, the functional size of each identified BFCs is quantified by rating their complexity and translating this rating into a Function Points value. Finally, the functional size values of the BFCs are summed to obtain the functional size value of the Web application [23].

Figure 10. The OOmFPWeb measurement procedure (source: Abrahão et al. 2004)

OOmFPWeb effort measuring representation offers greater visual cues and step-by-step procedures if compared to WEBMO or WEB-COBRA, its five conceptual models encapsulate each requirement domain into a measurable unit of Function Points values that facilitate the counting of effort units. Anyhow OOmFPWeb also fail to describe multiplatform requirements. The method Presentation Model only describes the user interactions with the application’s web interface, but it assumes that the context and the device are immutable. Modern web applications are built using a responsive interface, were the UI adapts to the interaction context (running, sitting, driving, so forth) and platform characteristics (phone, Smart TV, Smart watch).
4. Proposed Solution

In the previous chapter, three different methods have been compared based on specific criteria [Table 2]. One of these methods OOmFPWeb have been selected as the optimal model for complex web-based systems.

The main benefit of using the OOmFPWeb methodology is that it offers a modular conceptual structure that uses UML-compliant diagrams to describe the different views of an OO web application. This modular decoupled framework allows easier addition of new modules that can be retrofitted without having to amend the remaining domains. Additional models can then enhance the original framework while reusing the methodology established tools, mathematical formulas and counting rules. In OOmFPWeb the measurement of functional size is made at the conceptual schema level, so as a consequence, the application development is independent of implementation choices. That methodology characteristic, fits well with cross-platform development concepts were newer platforms and technologies are continually evolving.

Because OOmFPWeb lacks built-in cross-platform support, this paper will suggest a way of extending the original method by adding a new conceptual model called the Platform Model, this new model will complement the other existing models by calculating cross-platform effort. The Platform Model encapsulates web application context and platform-specific requirements that are unique for each platform [Table 3]. Modern multiplatform web applications typically require extra Source Line of Codes (SLOC) and Quality Assurance (Q&A) testing than single platforms applications, those requirements will be captured in the new proposed Platform Model.

Table 3: Characteristics Breakdown by Platform.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Context</th>
<th>Connection</th>
<th>*Example Display Resolution</th>
<th>I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phone</td>
<td>Walking, Commuting</td>
<td>WiFi, 2G, 3G, 4G, LTE</td>
<td>480x320</td>
<td>Touch, gesture, voice</td>
</tr>
<tr>
<td>Tablet</td>
<td>Sitting at home, Commuting</td>
<td>WiFi, 2G, 3G, 4G, LTE</td>
<td>768x1024</td>
<td>Touch, gesture, voice</td>
</tr>
<tr>
<td>Smart Watch</td>
<td>Running, walking</td>
<td>WiFi, Bluetooth, 2G, 3G, 4G, LTE</td>
<td>390X312</td>
<td>Touch, gesture, voice</td>
</tr>
<tr>
<td>Smart TV</td>
<td>Sitting at home</td>
<td>WiFi, Ethernet</td>
<td>1920x080</td>
<td>keyboard, pointing device, voice</td>
</tr>
<tr>
<td>Smart Car</td>
<td>Driving</td>
<td>WiFi, 3G, 4G, LTE</td>
<td>1280x400</td>
<td>Touch, pointing device, voice</td>
</tr>
<tr>
<td>Desktop</td>
<td>Sitting at home, at the office</td>
<td>WiFi, Ethernet, 2G, 3G, 4G, LTE</td>
<td>1680x1050</td>
<td>keyboard, pointing device, voice</td>
</tr>
</tbody>
</table>

*Display resolution among device vendors differs greatly, the select examples belong to some of the most popular devices currently available I the consumer market.
The ultimate goal while extending OOmFPWeb for cross-platform applications is in keeping it backward compatible with the original framework created by Abrahão et al, so the new proposed method can remain compatible with Function Points (ISO/IEC) and benefit from its original methodology for identification and measurement units.

4.1 Defining the Chosen Model

Using OOmFPWeb as the blueprint for the proposed new method, effort estimation is done in two steps. The first, is by identifying all functional requirements of a web application. Those requirements are classified into five original models created by Pastor et al., 2001 (Object Model, Dynamic Model, Functional Model, Navigational Model and Presentation Model) plus a new model introduced in this paper called Platform Model. The second step is when all relevant identified elements of the web application are measured and quantified. Most of the original conceptual framework [26] remains unchanged, except by the addition of the Platform Model into the Identification step [Figure 11].

Figure 11: OOWS Conceptual Models Plus Platform Model (source: O. Pastor and S. Abrahão, 2001)

4.1.1 Identifying Functional Requirements

In the identification step the elements that add functional size to the project are identified. The result of this first step is an abstract model of the relevant elements for functional size measurement, this model is described according to the FSM meta-model that uses units of measurement to count the identified elements. As OOmFPWeb is an extension of FSM method for OO web applications, mapping rules must be defined for the different perspectives of an OO system. These perspectives and their corresponding OO-Method conceptual model views
are categorized as: Object Model, Dynamic Model, Functional Model, Navigation Model, and Platform Model. The Platform Model encapsulates all requirements that are specific to cross-platform functionality and user context [Table 3]. It is defined at the same level as other conceptual models, and because OOmFPWeb is a modular framework, the new conceptual model can be added without having to amend the other models.

4.1.2 The OOmFPWeb Conceptual Models

Each OOmFPWeb conceptual model identifies a specific type of development effort. These are the components that consume web application development time and budget.

- The **Object Model** defines the data that is maintained by the application and the computations that the application performs, those are characterized by classes representation with shared events, agent relationships, inheritances and aggregation hierarchies [Appendix A.1].
- The **Dynamic Model** captures the behaviour of inter-objects within an application, those are characterized by events, actions, transactions, preconditions and control-conditions [Appendix A.2].
- The **Functional Model** captures the semantics attached to any change of state, those are characterized by the computations that the application performs along with object state changes [Appendix A.3].
- The **Functional Model** captures the semantics attached to any change of state, those are characterized by the computations that the application performs along with object state changes [Appendix A.4].
- The **Presentation Model** captures the semantics of how a user interacts with a web application, those are characterized by data input, manipulation of the web environment and the way information is presented [Appendix A.5].

The OOmFPWeb conceptual models provide a good framework to help identify web application requirements, most functional requirements are captured in one of the five conceptual models above. Anyhow OOmFPWeb does not explicitly deals with more abstract concepts like cross-platform and user context, for those requirements this paper introduces a new model called Platform Model.

- The **Platform Model** captures the semantics of how the application interacts with different platforms and within different contexts, those are characterized by different I/O functionalities, information presentation, network technologies and the context in which the application is being used.

4.1.3 The Platform Model

The Platform Model abstraction expresses the logical processing a web application performs when used on diverse contexts and platforms, cross-platform and user context requirements are translated into UML-like notations and quantified using OOmFPWeb counting rules.
Cross-platform requirements are all the logical evaluations, storage and data presentation, that are necessary to enable an application be accessed on multiple platforms:

- Testing and mitigating network issues, e.g. improving timeouts, presenting on screen feedback if a component takes long time to load, and so forth.
- Testing and mitigating device specific limitations, e.g. limited storage, no cookie or local data persistency, limited screen density, and so forth.

User context requirements are all the logical evaluations the software performs to understand what the user is doing while interacting with the application:

- Different I/O for improving user-experience, e.g. voice input on a smart watch and keyboard mouse on a laptop computer.
- Different information presentation, e.g. limiting information presentation to a minimum on a small device screen, but displaying any extra information on a large screen.

Figure 12 illustrates how the user is routed to a different (optimized) landing page based on device detection, if the application only rescales the UI for the specific device (watch, tablet or desktop) the text may be too small to read on the watch. In this case, the developer is required to carefully design the amount of information displayed on each device screen. The time and cost spent on such cross-platform requirement is encapsulated in the Platform Model. These logical procedures illustrate the effort units that are necessary to translate human requirements based on the platform (smart TV, Connected Car, and so forth) and context (sitting on the sofa, driving, running, and so forth) into programming instructions. Those instructions are the essential mechanisms that allow web applications delivering an optimal user experience on multiple platforms.

Figure 12: Platform Model Representation
4.2 Measuring Functional Requirements

In the measurement step the elements captured in each conceptual model (Object Model, Dynamic Model, Functional Model, Navigational Model and Presentation Model and Platform Model) are identified. As a consequence, the functional size of a Web application is calculated in the problem space by mapping the identified primitives of *Object-Oriented Web Solutions* (Pastor et al., 2001) into *Function Point Analysis* (Albrecht, 1979) counting rules. Calculating the functional size of a web application takes several steps that are breakdown into identification and counting.

4.2.1 Identifying application boundary and scope

These are the borders between the web application being measured and the external applications or user domain. The Agents that are outside the application and the conceptual models define the application functionality. The global view of the application can be represented by a navigational map, the application boundary corresponds to all navigational maps represented in the Navigational Model.

*Figure 13: Application Boundary Map (source: Abrahão et al, 2004)*

Where:
- ILFs are the internal logical files
- EIFs are the external interface files
- Els are the data the user enters into the system
- EOs are the data the system presents to the user
- EQs are the input query responses from other systems

The application boundary defines the borders of what is being measured (system inner-works and externals interfaces with other systems), while the scope defines the functionality which will be included in a particular measurement. Figure 13 defines the application boundary as every functionality inside the circle, while the scope includes all the information defined in the six conceptual schema views (Object Model, Dynamic Model, Functional Model, Navigational Model, Presentation Model and Platform Model).
4.2.2 Identifying data and Transaction candidate functions

Data candidate functions express in numerical terms the effort required to define and store system data, while transaction candidate functions express in numerical terms the effort required to manage the system data. OOmFPWeb method was designed to conform to the IFPUG (International Function Point Users Group) counting rules [Appendix A.6], those rules assign numerical values to express the functional size of the web application components. Once all elements are identified and quantified, the total effort size can be calculated.

4.2.3 Counting data and transaction functions

OOmFPWeb provides mathematical formulas for calculating both Data and Transaction Functions [Appendix A.9, A.10]. The result of those calculations provides the amount of functionality that the web application delivers, therefore using this value as an effort estimation.
5. Applying the Chosen Model

Applying the extended version of OOmFPWeb follows the same steps as the original methodology with the addition of the Platform Model. To better illustrate this process a hypothetical cross-platform web application is defined, the application is a simple video player that highlights some specific platform requirements defined in the application boundary.

5.1 Defining a Hypothetical Web Application

As an example of a modern cross-platform web application a simple HTML5 video player application is defined. The W3C HTML5 standard abstracts most of playback complexities in its API, so that makes the HTML5 \(<video>\) element [27] an ideal candidate to highlight only platform and use context requirements.

Figure 13: Simple Video Player User Interface Mock-up

![Image of a simple video player user interface mock-up]

The simple video player presents a short list of requirements [Table 4] that based on platform detection perform different logical instructions. The control bar should be hidden when opening the player on a phone because Google Android and Apple iOS, the two most popular smartphone OS vendors in the market, by default add their own control bar over the video being played. The title and author description should be hidden on mobile due to the limited screen size, this is not an issue on desktop or tablets. On a desktop computer, the video should start by default in High Definition (HD), but not on a tablet or phone because it may consume too much of the user Internet mobile data. Finally, on a phone the user should be able fast-forward the video by shaking the device, anyhow such user input is not available on tablet and desktop computer.

Table 4: Simple Video Player Requirements per Platform

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Tablet</th>
<th>Phone</th>
<th>Desktop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display custom control bar (play, pause and scrollbar buttons)</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Display title and author text menu</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Start playback using HD media file</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Enable shake to fast-forward</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>
5.2 Identifying Application Boundary

The functionalities inside the project boundary are all the logical functions necessary to select and playback video assets from a Content Delivery Network (CDN), query asset metadata from a database, present the user with asset information and a menu interface for user input commands [Figure 14]. In addition to these logical functions there is platform specific functionalities such platform detection, asset selection based on platform and information presentation based on platform.

Figure 14: Simple Video Player Boundary

Where:
- ILFs are the internal logical files
- EIFs are the external interface files
- Els are the data the user enters into the system
- EOs are the data the system presents to the user
- EQs are the input query responses from other systems

5.3 Identifying Transaction Functions

The simple video player performs a data transactions every time the user interact with the web interface [Figure 14]. Using FSM measurement rules to translate functional requirements into measurement units, below are the functional components required to perform each transaction, these have been simplified to 1 action and 1 attribute change.

[press play button] El
- 1 DET for an action
- 1 FTR for state change of attribute value

[display song title & author] EO
- 1 DET for an action
- 1 FTR for state change of attribute value

[query song title & author] EI and EO
- 1 DET for an action
- 1 FTR for state change of attribute value
5.4 Identifying Data Functions

Data and transaction candidate functions elements are the building blocks of FSM measurement model. In this stage, measurement units are applied to each requirement of the application by breaking it down into smaller functional components and identifying each unit as Object, Dynamic, Functional, Navigation, Presentation or Platform effort units.

5.4.1 Identifying Object Model Units

The simple video player logic is contained within three classes, one Class named Controller where all the processing logic is encapsulated, one Class called View where all the output representation of information is encapsulated, and one Class called Model that manages the data and data logics of the application [Figure 15]. The simple player View Class initiate other sub-classes that built the visual components displayed on the screen, anyhow for the purpose of this example we assume these classes are readily available outside our boundary scope.

**Figure 15: Simple Video Player Class Representation**

Based on the simple video player Class Representations above we can use FSM measurement rules to translate functional requirements into measurement units as below.

Class:

- [Model] 1 RET for the class
- [View] 1 RET for the class
- [Controller] 1 RET for the class
- [Model.title] 1 DET for data-valued attribute of the class
- [Model.author] 1 DET for data-valued attribute of the class
- [Model.platformName] 1 DET for data-valued attribute of the class
- [Model.mediaName] 1 DET for data-valued attribute of the class
- [View.textMenu] 1 DET for data-valued attribute of the class

Class Service:
- [View.model] 1 FTR for class reference
- [Controller.model] 1 FTR for class reference
- [Controller.view] 1 FTR for class reference
- [View.playButton] 1 FTR for class reference
- [View.pauseButton] 1 FTR for class reference
- [View.progressBar] 1 FTR for class reference

5.4.2 Identifying Dynamic Model Units

The simple video player has two logical state transitions, the first is detecting the platform type where the application is being loaded, while the second evaluate the best media type for the identified platform [Figure 16].

Figure 16: Simple Video Player platform detection and media playback diagrams

Based on the simple video player diagram representations above we can use FSM measurement rules to translate functional requirements into measurement units as below.

- 1 FTR [Controller.detectPlatform()] for a control condition
- 1 FTR [Controller.playMedia()] for a control condition
5.4.3 Identifying Functional Model Units

The simple video player keep track of how many times a media have been played, this change in state happens every time event `onComplete` is dispatched by incrementing variable `playCount` with one unit [Figure 17].

**Figure 17: Simple Video Player playback count diagram**

![Diagram showing the simple video player playback count](image)

Based on the simple video player diagram representations above we can use FSM measurement rules to translate functional requirements into measurement units as below.

- 1 FTR [Controller.playCount] for state change of attribute value

5.4.4 Identifying Navigation and Presentation Model Units

The simple video player user interface adjusts to the underlying platform, that allows some of its I/O components to adjust according to the context in which the user is accessing the application and select more user-friendly interaction mechanisms [Figure 18].

**Figure 18: Simple Video Player fast-forward action diagram**

![Diagram showing the simple video player fast-forward action](image)

Based on the simple video player diagram representations above we can use FSM measurement rules to translate functional requirements into measurement units as below.
• 1 DET for click action
• 1 DET for shake action
• 1 DET for press action

5.4.5 Identifying Platform Model Units

The simple video player application must support three different platforms, and dynamically select the best mechanisms for each platform and interaction context. That means the developer need to test and troubleshoot the application on each platform and design the application according to the best practices for each one.

• 2 extra DET [Controller.detectPlatform()] 1 for each platform that is the result of an IF condition
• 2 extra DET [Controller.playMedia()] 1 for each platform that is the result of an IF condition
• 2 extra DET per user context accessing the application (user can be both moving using a mobile device or stationary using desktop computer)
• 1 extra DET [Model.platfromName] for a record that deals with specifics of a platform

5.5 Counting Data and Transaction Functions

The amount of data handled by the application logical files and transactional functions determines the amount of functionality that the software delivers, hence its functional size. Calculating the simple video player total functional size takes two steps, first count all data and transaction effort units, secondly apply OOmFPWeb calculation formula.

Table 5: Simple Video Player Data Function Measure Units

<table>
<thead>
<tr>
<th>Data Functions</th>
<th>DETs</th>
<th>RETs</th>
<th>FTRs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Model</td>
<td>5</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Dynamic Model</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Functional Model</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Navigation and Presentation Model</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Platform Model</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>15</td>
<td>3</td>
<td>9</td>
</tr>
</tbody>
</table>

Applying the weights described in the table 5 the obtained functional size for data functions is:

\[
\text{OOmFP}_D = (\text{DETs} + \text{RETs}) + \text{FTRs}
\]

\[
\text{OOmFP}_D = 18 + 9
\]

\[
\text{OOmFP}_D = 27
\]
Table 6: Simple Video Player Transactional Function Measure Units

<table>
<thead>
<tr>
<th>Transaction Functions</th>
<th>DETs</th>
<th>RETs</th>
<th>FTRs</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILFs</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>EIFs</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>EIs</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>EOIs</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>EQs</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

By measuring all weights [Table 6], the obtained functional size for transaction functions is:

\[
\text{OOmFP}_T = (\text{DETs} + \text{RETs}) + \text{FTRs}
\]

\[
\text{OOmFP}_T = 5 + 5
\]

\[
\text{OOmFP}_T = 10
\]

Lastly, the partial functional sizes obtained by the data and transactional functions calculations are summed to obtain the functional size value of the system. Therefore, the total functional size for the simple video player web application is 37. Without the addition of the Platform Model effort units to account for testing and troubleshooting on different devices the final functional size would be underestimated by 7 units.

By adding the Platform Model to the remaining OOmFPWeb framework, cross-platform and user context functional requirements were explicitly defined beforehand. That forces the developer to think about those abstract requirements and use UML-like notation that can be quantified using OOmFPWeb measurement units. Without the Platform Model, some of those requirements may still be captured in the remaining models, even if those models do not specifically deal with cross-platform and user context characteristics. Nevertheless, because none of the remaining models explicitly define rules for identifying cross-platform and user context, some requirements may be unaccounted and eventually deliver an inaccurate time and cost estimation.
6. Conclusion and Further Work

This paper has proposed a better model for cost and time estimation for web applications. It has discussed the characteristics (cross-platform and user context) that are often unaccounted using established effort estimation methods. It has also demonstrated how the OOmFPWeb conceptual framework can be extended by adding a new model called the Platform Model. This new model is used to capture and quantify these missing requirements, therefore providing a better effort estimation result than the original methodology.

This paper concludes that modern web applications by default inherit the need to provide cross-platform and context functionalities. The user expects the application to interact and present its data according to the platform best practices and in context, failing to do so may result in a poor user-experience. Modern methodologies able to fully estimated all the hidden complexities of web software are uncommon and difficult to grasp, thus web developers may prefer using their own instinct to guess effort instead of correctly estimating it. Finally, as IoT connected devices grow in popularity and the way people interact with those devices diversify, effort estimation complexity will also increase making effort guessing even more unfit.

Further development and refining of the Platform Model for OOmFPWeb method is needed, specifically by testing the complete methodology using a real web project. In the example presented in Chapter 5 many components and real world complexities have been omitted or large simplified, thus applying the model into a real complex system may expose gaps or mismatches in some of the assumptions this paper makes.
Appendix A

A.1 Object Model Representations

The Object Model abstraction expresses the logical data that must be maintained by the application; insert new data, update data and remove obsolete data. Whereas the legacy views represent logical data that are only referenced by the application. Logical data refers to the data as known by the OO system, it does not refer to the physical representation of the data as stored on storage media. Figure 19 illustrates Class Name and its complex data computation relationships expressed as attributes reader_code, book_code and date; methods loan and return; relationships with Class Book and Class Reader; and other agent relationships that combined outlines the application computations.

Figure 19: Object Model Representation of Class Relationships (source: O. Pastor and S. Abrahão, 2001)

The components of the Object Model are:

- The signature of a class including attributes, methods and properties:
  - Integrity Constraints (static and dynamic) with state conditions that must be satisfied
  - Valuations that state how attributes are changed by event occurrences
  - Derivations that relate some attribute’s values to others
  - Preconditions that determine when an event can be activated
  - Triggers that introduce internal activity
- Inheritance:
  - Specialization deals with descendent inheritance or derivation of children classes from a parent class
- **Generalization** deals with ascendant inheritance or generation of parent classes collecting common properties of predefined classes
  - Specialization:
    - **Permanent** when an object belongs also to a given specialized class as soon as the object is created.
    - **Temporal** when an object act as the corresponding specialized class when a role creation occurs
  - Legacy View for pre-existing software systems
    - **Filter** placed on a class
    - **Aggregation relationships** with a class

### A.2 Dynamic Model Representations

The Dynamic Model abstraction expresses the application components dynamic interactions, those are represented in terms of conceptual visual diagrams that illustrates how these mechanisms interconnect. Therefor revealing the intricacies of OO applications interactions into quantifiable components. Figure 20 illustrates the sequence interaction between *Enter Code* and *Release Door* by evaluating the *Code* value, hence triggering the correct transaction.

**Figure 20: Dynamic Model Representation as a State Diagram (source: Wikipedia, 2016)**

The components of the Dynamic Model are:
- **State Transaction Diagram**
  - **Sequence of events** performed by object abstracts
- **Object Interaction Diagram**
  - **Interactions between objects** using triggers and global interactions
A.3 Functional Model Representations

The Functional Model abstraction expresses the OO system state changes. Figure 21 illustrates these changes in the Class reader attribute book_number were the state change can be either increased or decreased by one.

Figure 21: Functional Model Representation of State Change of a Class Attribute (source: O. Pastor and S. Abrahão, 2001)

<table>
<thead>
<tr>
<th>Category: push-pop</th>
<th>Class: reader</th>
<th>Attribute: book_number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ac. Type</td>
<td>Action</td>
<td>Effect</td>
</tr>
<tr>
<td>Incr.</td>
<td>REA: rent</td>
<td>book_number + 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>--</td>
</tr>
<tr>
<td>Decr.</td>
<td>REA: return</td>
<td>book_number - 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>--</td>
</tr>
</tbody>
</table>

A.4 Navigational Model Representations

The Navigational Model abstraction expresses the navigational components designed in the application. Those components can hence be classified, converted into FSM measurement units and quantified. Figure 22 illustrates the two navigational steps a user would require to navigate from Home to How to Order section, therefore these two steps are translated into two FSM measurement units. Figure 23 illustrates the two points of view a user may have, therefore two more FSM measurement units are counted. The components of the Navigational Model are:

- **Navigational Map** that represents the global view of the web application for an audience

  Figure 22: Navigational Map (source: O. Pastor and S. Abrahão, 2001)

- **Navigational Context** is the point of view that a user has on a moment
**Figure 23: Navigational Context** (source: O. Pastor and S. Abrahão, 2001)

- **Navigational Link** allows navigation from one context to another
- **Navigational Class** are the attributes and methods of a class specified in the Object Model

**Figure 24: Navigational Class** (source: O. Pastor and S. Abrahão, 2001)

- **Context Relationships** indicates the navigation direction, relationship and context between the classes

**Figure 25: Context Relationships** (source: O. Pastor and S. Abrahão, 2001)

- **Context Dependency Relationship** provides additional information in the current node without indicating any further navigation
A.5 Presentation Model Representations

The Presentation Model abstraction expresses the user interaction patterns with the system interface. Figure 27 illustrates the steps a mobile phone user would take to access the component menu by swapping to the right. One interaction step translates to one FSM measurement unit. On a desktop computer where the UI is optimized for larger screens and the use of pointing device and keyboard, these steps would be different, therefore producing more FSM measurement units.

A.6 OOmFPWeb Measurement Rules

Numerical values are used to the express the functional size of the web application components. The mapping of the software system to numerical values is accomplished by measuring the data and transaction functions according to FSM measurement rules. Measurement units are defined as Data Element Types (DET), Record Element Types (RET), and File Types Referenced (FTR). DETs correspond to attributes in the object classes and RETs correspond to object classes, while FTRs are data function referenced during the execution of a transactional function [23].
Object Model:
- **Class:**
  - 1 DET for each data-valued attribute of the class
  - 1 DET for each attribute in the IF of a class or legacy view referred to by a univalued aggregation relationship
  - 1 DET for each attribute in the IF of the superclasses of a class
  - 1 RET for the class
  - 1 RET for each multivalued aggregation relationship
- **Legacy View:**
  - 1 DET for each non-derived attribute of the legacy view
  - 1 DET for each attribute in the IF the class related to by a univalued aggregation
  - 1 RET for the legacy view
- **Class Service:**
  - 1 DET for each data-valued argument of the service
  - 1 DET for the capability to send messages
  - 1 DET for the action (Accept/Cancel) of the service execution
  - 1 FTR for the class
  - 1 FTR for each new class referenced in the object-valued argument of the service
  - If a value by default is defined, count one FTR for new class referenced in the formula
  - If the service is a destroy event, count one FTR for each new class accessed in the cascade formula
  - If the service is a transaction, count one FTR for each class referenced in the transaction formula
  - If a specialization by condition is defined, count one FTR for each new class accessed in the specialization formula
  - If a specialization by event is defined (carrier/liberator event), count one FTR for each new class for which the event is a carrier/liberator
  - If integrity constraints are defined, count one FTR for new class referenced in the formula
- **Legacy View Service**
  - 1 DET for each data-valued argument of the service
  - 1 DET for the capability to send messages
  - 1 DET for the action (Accept/Cancel) of the service execution.
  - 1 FTR for the legacy view
  - If preconditions are defined, 1 FTR for each new class referenced in the formula of a precondition definition
  - If integrity constraints are defined, count one FTR for new class referenced in the formula

Dynamic Model:
- 1 FTR for each new class referenced in the formula of a control condition, defined in the state transition diagram
- 1 FTR for each new class referenced in the formula of a trigger definition, defined in the interaction diagram
• 1 FTR for each new class referenced in the formula of a precondition definition, defined in the state transition diagram

Functional Model:
• 1 FTR for each new class referenced in the formula of a valuation definition
• 1 FTR for new class referenced in the formula of a control condition associated to the valuations.

Navigation Model and Presentation Model:
• 1 DET for each attribute in the display set
• 1 DET for each offered action
• 1 DET for each offered navigation
• 1 DET for the system capacity to display messages
• 1 FTR for each class or legacy view in the display set

Platform Model:
• 1 extra DET per platform for each IF condition that deals with specifics of a platform
• 1 extra DET per platform for each IF condition that deals with user context
• 1 extra DET per platform for each record that deals with specifics of a platform
• 1 extra DET per platform for each record that deals with specifics of user context

The Platform Model adds 1 extra unit of effort to all specifics of cross-platform and interaction context. Those extra units represent the effort to test and troubleshoot the same application on different platforms.

A.7 Data candidate functions

Data functions are classified as Data Element Types (DET) or Record Element Types (RET). A DET is a unique, non-repeated field, anyhow if one integer variable user_name is stored in multiple fields it is still counted as one DET. A RET is a subgroup of data elements within a logical file. For instance, in a Human Resources Application, information for an employee is added by entering some general information. In addition to the general information, the employee is a salaried or hourly paid employee. Then, two RETs are identified: salaried employee and hourly paid employee. RETs can be of two types, Internal Logical Files (ILFs) are the data internally maintained by the system, while External Interface Files (EIFs) are the data referenced in other systems.
Data candidate functions express in numerical terms the effort required to define and store system data. Figure 28 demonstrate how the variable name that is stored in multiple fields counts as one DET, while quantity has two RETs as it refers to one internal data element (ILF) and one external data element (EIF).

A.8 Transaction candidate functions

Identify the logical files and functions that perform transactions using or altering existing internal or external data. Transactional functions are processes of user interaction with the system that can manifest in three different ways: the user enters data into the system (EIs), the system presents data to the user (EOs), and an input requests an immediate response from the system (EQs) [23].
Transaction candidate functions express in numerical terms the effort required to manage the system data. Figure 29 illustrates the three OO functions necessary to input the name variable (EI), present name variable to the user (EO) and dispatch a response after an input request (EQ).

### A.9 Data functions Mathematical Formula

OOmFPWeb data function mathematical formula assign numerical values to the amount of data processed in the web application.

**Figure 30: Data Function Mathematical Formula (source: Abrahão et al, 2004)**

\[
OOmFP_{Web} = OOmFP_D + OOmFP_T
\]

\[
OOmFP_D = \sum_{c \in OM}^{c} OOmFP_{ILF}(DET_{class}, RET_{class}) + \sum_{l \in OM}^{l} OOmFP_{EIF}(DET_{view}, RET_{view})
\]

Where:
- \( OM \) denotes an Object Model
- \( c \) is a class in \( OM \)
- \( l \) \( v \) is a legacy view in \( OM \)
• DETs and RETs are elemental measurements that are calculated in the data functions to determine their complexity

A.10 Transaction functions Mathematical Formula

OOmFPWeb transaction function mathematical formula assign numerical values to the functionality provided to the user to process data in a web application.

Figure 31: Transaction Function Mathematical Formula (source: Abrahão et al, 2004)

\[
-\text{OOmFP}_T = \sum_{m \in OM} OOmFP_E(m, DETs_{method}, FTRs_{method}) + \sum_{pp \in PM} OOmFP_{EQ}(DETs_{pp}, FTRs_{pp}) + \sum_{nc \in NM} OOmFP_{EQ}(DETs_{nc}, FTRs_{nc})
\]

Where:
• OM denotes an Object Model, PM denotes the Presentation Model, NM denotes Navigational the Model, and PLM denotes Platform Model
• \(m\) is a method of the class or legacy view
• \(pp\) is a presentation patter
• \(nc\) is a navigation context
• DETs and FTRs are elemental measurements that are calculated in the transactional functions to determine their complexity
References


