Cristian Planas González

Generating social media for the movie world: TweetMovies

Helsinki Metropolia University of Applied Sciences
Bachelor of Engineering
Information Technology
Thesis
19 March 2012

Metropolia

Abstract

Author(s) Title Number of Pages Date	Cristian Planas González Generating social media for the movie world: TweetMovies 81 pages 19 March 2012
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Instructor(s)	Mr. Kimmo Saurén, Senior Lecturer

The study explores the possibilities for developers in the social media. This is done not only with analysing tools, but also with the development of a real project: TweetMovies.

TweetMovies is a web application/social network that makes it possible for users to share and read opinions about movies on the Internet. In TweetMovies the users generate business value. This makes TweetMovies a perfect example of 2.0 social media.

TweetMovies has, among other features, a user authentication system; interaction with external APIs like Facebook, Twitter or The Open Movie DataBase; extense use of script systems like Javascript and its framework JQuery; web layouts created using advanced mark-up languages like Slim and Sass; or unit and acceptance tests following the general BDD methodology. All this is used under the frame given by the framework Ruby on Rails.

The scope of this thesis is to analyse the present state of the Web, especially the Web 2.0 phenomenon and the most important movie applications. Some of the main tools used in the development of web applications are also reviewed and analysed. Finally, the main design decisions and concepts in the design of a 2.0 application that were applied in the creation of TweetMovies are also commented.

The final result of this thesis was the creation of the web application TweetMovies.

Keywords	Web 2.0, Ruby on Rails, social network, social media

Contents

1. Introduction

In the contemporary world, social networks' influence is growing more and more, and this tendency will probably continue in the next decades. Nowadays it is quite difficult to find a young person in the modern world who does not have an account in one or more of the most popular social networks: Facebook, Twitter or Google+. This trend demonstrate once more how the everlasting human desire to communicate uses and adapts to the new contexts generated by the innovations of technology.

However, even if Facebook and Twitter are present in the everyday life of most of the population of the Western countries, given that their popularity is more or less new, it is still possible to find plenty of space for new applications. These spaces will probably be a great source of jobs for the IT developers in the next years.

One of the most obvious fields where new IT applications will be created is the public's interests and hobbies. Humans already live in a society where individuals are not searching anymore for their own survival, since that is taken for granted. Most of an individual's efforts and time – apart the time dedicated to work – is dedicated to free-time activities. Those activities are, at the present time, an extremely big source of money.

In addition, the worldwide web has been a great place for people to practice their interests. Thousands of portals talk about almost anything anyone could desire to play, to collect or to learn. Since the beginning of the Internet this has been one of the main appeals of the Web. Further, cinema, as probably the most popular and universal of all the arts, has been totally involved in this process.

It is possible than cinema is one of the most usual topics on the Web. From a juggernaut source of data (Internet Movie DataBase) to tens of thousands of blogs, film fans love to watch, discuss and discover movies. The Internet has broken the typical domain on the taste of the public that the big American cinema companies had, giving the audience access to a new world of alternative cinema(s). A movie lover in the present day no longer depends on his or her town local cinema choices to watch new movies: the last Iranian auteur film or that forgotten noir movie of the 50s are

available in a few clicks. Nevertheless, as Jonathan Rosenbaum wisely argues in his book *Goodbye Cinema, Hello Cinephilia* [1, 3-9], this evolution has changed the critical discussion too. Now the ring where movie critics confront their positions is no longer restricted to the sphere of their own nation. With English as an international lingua franca, critics from around the world use the Web immediacy to create more appealing and comprehensive speeches about the seventh art.

Even though the cinema field is well known, there can still exist room for basic tools. It is really possible that society still has to understand and exploit fully everything that the Web 2.0 concept means: how users should interact constantly with the application, or even how they and their interactions build the real content of the application, being it only the field who defines the limits.

That essential miss that is being discussed is in the form in which the opinion of the users about movies is expressed. Even nowadays, if anyone surfs on some of the most popular movie portals (like IMDB, FilmAffinity or even RottenTomatoes) that user will see that most of the reviews are written in long texts. In addition, it is no secret that the audience, nowadays, loves to have the information in a brief and direct way: that is exactly the main point of Twitter. In this situation, the users tend to only check the number of stars that the movie has received, and, with some luck, to give their own score. That interaction is too short, especially when it is known that the user is willing to participate much more: all the social networks prove it.

That is exactly the subject of this thesis: the developing of an application that will provide a way for users to share their opinions about a given movie in a quick, easy way. This application will use the already established social networks, as the author of this thesis thinks that particularly Facebook and Twitter will be useful, at the same time, as functional features and as a way of advertising the application itself, as the comments written by the users in TweetMovies will appear in their public walls, making it possible to spread the use of TweetMovies at an unbelievable speed.

Given that what is in question is a web application, web tools to develop it will be necessary. And that is another reason for this choice, as the author of this thesis already has some experience in web developing. After working more than one year

using different web programming languages (PHP and Ruby on Rails), the author of this thesis thinks that he is able to develop fully a social web application with its most basic and important features. The author of this thesis must admit that the creation of a website of this kind will help him as a great cover letter even in such specialized labour world as the web developing scene (and Ruby on Rails in particular) is.

2. The Web

In less than 20 years, the Internet has passed from being an unknown science-fiction-like concept to a colossal reality, present in almost any human activity. How has that been possible? Which are the differences between Web 1.0 and 2.0? How does that matter to activities like businesses?

2.1 Brief history of the Web

There are two important events in the origins of the Internet: one, the development of point-to-point communication and, after that, packet switching. The main system that resulted based on research in these fields was ARPANET.

ARPANET was born as a military project of the United States Department of Defense to improve the communications between the research laboratories of some US schools. In its first version, created in 1969, it included four universities: University of California (UCLA), Stanford, University of California-Santa Barbara and University of Utah. The first email transmission was produced in 1971. [2] There were other networks of the same type during those years: NBC, Merit Network, and the French CYCLADES among others. However, ARPANET was the one in which the technological innovations went further.

The other main event in the creation of the Internet was the standardization of the TCP/IP protocol. In the mid-seventies, scientists faced an important problem: they already had established some different networks, but the different existing protocols made communication between them impossible. Some researchers like Robert E. Kahn and Vinton Cerf developed a solution: it was necessary to create a common internetwork protocol to communicate between all those subnetworks. The first specifications of that protocol, the name of which is TCP/IP, appeared in 1974, but it did not reach its final form until 1978. [3]

After that, especially in the eighties, these subnetworks between several public and private institutions of the US continued to develop until the nineties when several

commercial Internet Service Providers (ISP) replaced the public structure running the network.

With the formation of the private ISPs, the course of the history of the Internet dramatically changed. In 1992 the prohibition of using the network for commercial purposes was removed, thus, opening the Internet to the private sector and the general public. [4] It was then when the growth of the Internet started. Figure 1 shows the growth in Internet hosts and therefore users, since the mid-nineties.

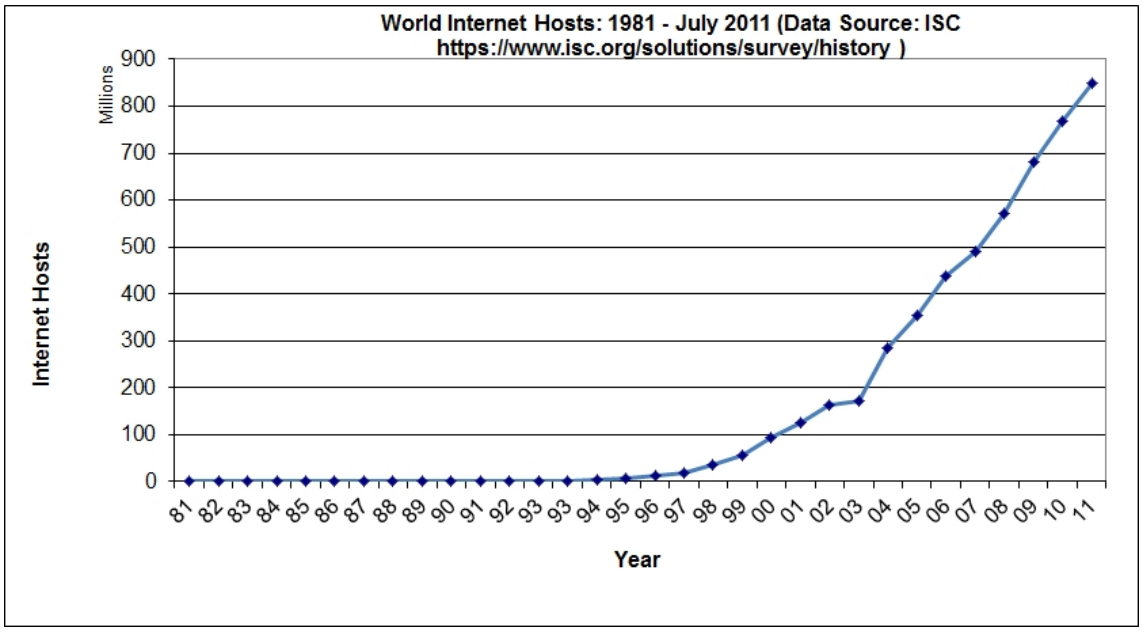

Figure 1. World Internet Hosts (1981-July 2011). [5;6]

In the 1990s the Internet started to reach the population of the Western world. In that time, the most popular services were email and the World Wide Web (www), services still essential in today's Internet; actually Internet is still called colloquially "the Web". In the 1990s Web, features like blogs, discussion forums or chats systems like IRC (Internet Relay Chat) started to show the dormant social potential that already existed on the Internet.

However, the Internet has changed and not all the features that were popular in that time maintained that popularity. A good example of that are Web portals, a concept that was central in the way people accessed the Web in the nineties and one that does

not exist anymore in the present day Internet, or at least, not with the same importance.

A web portal is a site that works as a central point of access to the information available on the Web. Nowadays, portals are usually around one topic or work like de facto newspapers, but in the nineties it was different. One important point to understand was that the Internet was an extremely new tool then: most of its users still did not understand its essentials. In this situation, Internet providers (usually phone companies) offered as a default navigation site a portal created by them: this way, not only did they help the user but they also controlled the main channel of information that the user could access. Nevertheless the business of web portals in that moment was quite temporary, as it was based on people's lack of skill with a new tool; a characteristic of the model of business that was clearly destined to change.

Some examples of web portals are AOL, MSN or Yahoo; however, every country had some portals of their own. For instance in Spain, the main one was Terra, owned by the main telecommunication company of the country, Telefónica.

2.2 Web 2.0

Nowadays, everyone talks about a shocking concept: the Web 2.0., or 2.0 in short. As a web developer, it is essential to know deeply the basic interactions in the networks. However what is exactly the 2.0? And how is it possible to use it to make TweetMovies more successful?

2.2.1 From 1.0 to 2.0

From 1995 onwards, companies started to invest in Internet. It was widely considered that the Web was the future and that increased the stock value of Internet-related companies in an unreasonable way. Actually, some authors talk about prefix investing, which is based on winning only adding an "e" or a ".com" to the name.

The bubble came to its peak in the 2000s. Lots of reasons have been argued to explain its burst: some inside the technology world, one example being users getting used to the Web and not needing web portals anymore or the United States against Microsoft case, where Microsoft was declared monopoly. The increase of interest rates clearly influenced the situation, too.

The burst of the dot-com bubble changed dramatically some of the most essential ideas around the Web. Web portals were deprecated, as plain users improved their IT skills. A new central point on the web navigation, one that still maintains its position nowadays, replaced them: searchers.

This change of habits was also caused by technological reasons: before the 2000s, searchers were not as reliable as they are today, and it was probably not enough for a normal user to ignore old web portals and use exclusively a searcher. In the preGoogle world, the algorithms behind web searcher engines like AltaVista or Lycos were less effective and slower than the one created by Sergey Brin and Larry Page.

However, the relationship between the 2.0 concept and searchers is only marginal. To understand 2.0, it is necessary to remember the situation of the Internet in the early 2000s. After the dot-com bubble burst, the future of the Web started to be doubted by some: maybe what had been seen so far was only a toy, a trend that would disappear quite quickly. It was pointed out that the Internet had failed to really affect the lives of its users.

This is when a new concept of the Web appeared. The term 2.0 was popularized after O'Reilly Media, a media company specialized in computer technology topics, organized the first Web 2.0 conference. In the opening remarks they famously attacked what they called the "Web 1.0", using Netscape business model as an example:

> Netscape framed "the web as platform" in terms of the old software paradigm: their flagship product was the web browser, a desktop application, and their strategy was to use their dominance in the browser market to establish a market for high-priced server products. Control over standards for displaying content and applications in the browser would, in theory, give Netscape the kind of market power enjoyed by Microsoft in the PC market. **Much like the "horseless carriage" framed the automobile as an extension of the familiar, Netscape promoted a**

> **"webtop" to replace the desktop**, and planned to populate that webtop with information updates and applets pushed to the webtop by information providers who would purchase Netscape servers. [7, bold added]

In 2005, Tim O'Reilly formalized all these ideas in a paper titled *What is Web 2.0?* [8] In the foreword to *Web 2.0: A Strategy Guide*, O'Reilly summarizes his ideas about 2.0 this way:

> I argued that Web 2.0 is ultimately about harnessing network effects and the collective intelligence of users to build applications that literally get better the more people use them. What's more, I argued that just as the personal computer rewrote the rules of the computer industry in ways that blindsided the giants of that era, so too, would the Internet upend all the power structures and business models of today's industry. [9]

However maybe the best and most compact definition of 2.0 was done by the same O'Reilly in a 2006 article:

> Web 2.0 is the business revolution in the computer industry caused by the move to the internet as platform, and an attempt to understand the rules for success on that new platform. Chief among those rules is this: **Build applications that harness network effects to get better the more people use them.** [10, bold added]

There are a lot of examples of differences between the old 1.0 model and the new 2.0 one. In 1.0, the user could only read quietly the information given to him by the webmaster, usually by a web portal; that made it easy to control the website, but it also made the experience much less exciting; in the end, 1.0 pages were like newspapers, magazines or books, but usually with much more mediocre authors. In the 2.0 Web users are able to give their opinions about any content of the page: the comments feature was one of the first to be popular and it started the paradigm change.

However, that philosophy was not limited to this particular concept: users should not only be able to give their opinion around one central piece of information created or chosen by the webmaster, they should also be able to produce their own pieces of data. Blogging started to be widely popular, thanks to platforms that made it extremely easy to create a blog.

The next step in the 2.0 advances probably has been the most exciting one: the rise of the social networks. Nowadays, the main ones are Facebook and Twitter but there are many more, such as local versions of Facebook (Tuenti in Spain, VKontakte in Russia, Renren in China) or others more focused on certain hobbies or activities, like LinkedIn, a professional-oriented social network, or FilmAffinity, an application about the movie world.

Nevertheless the most important ones, Facebook and Twitter, are considered nowadays as giants of the web, and it is normal. Only Facebook gets 100 billion hits per day. [11] They start to be considered not only as places where people can manage their friends and acquaintances, but also as social tools where people can show their opinions and beliefs.

2.2.2 Impact of 2.0 on the society

It is widely accepted that the social movement called "The Arab Spring" started in Twitter [12], where a great number of young people can organize themselves in an easy and quick way. This movement, or group of movements, started with a small incident: the self-immolation of Mohamed Bouazizi after being mistreated by the Tunisian police. Public outrage quickly grew over the incident, expanded by social networks like Twitter or Facebook. [13]

The regime of President Zine el Abidine Ben Ali had ruled Tunisia since 1987, and any form of protest was quickly and successfully oppressed and kept silent. However, with the international – or maybe *afternational* – nature of the social media, the Tunisian governments' efforts to silence the riots were pointless. Thanks again to the media, especially the social one [14], the demonstrations that started because of Bouazizi death, spread not only to his country but to all the Arab countries and even to the whole world. The Egyptian revolution, the Libyan Civil war and the Yemeni, Bahraini and specially the Syrian uprising [15], or, in the Western world, the 15-M protests in Spain and the Occupy Wall Street movement continued this new revolutionary way of citizen interaction: with this, it is obvious than 2.0 is a strong weapon for social change.

Nevertheless, not all the implications of social media are so politically meaningful. Social networks have changed too the way humans communicate in the private sphere. It is easier to chat or organize something with our friends, and finding people who share the same hobbies is also much more simple. However, even in the media world things have changed: now it is common to include a comments space in every piece of news, and even the major media comment what it is happening on Twitter, probably considering that Twitter reflects the opinion of the majority of the population.

As this social media revolution continues, professional profiles in the media and communication world like contents manager, interactive and relational marketing coordinator or community manager start to be common. Even in normal businesses, usually far away from the IT world, the social media has made a terrible impact. For example, a smart campaign in Facebook can change the fate of a given business, making success easier and quicker.

Social media has a lot of forms. In figure 2 it is possible to see which types have more users.

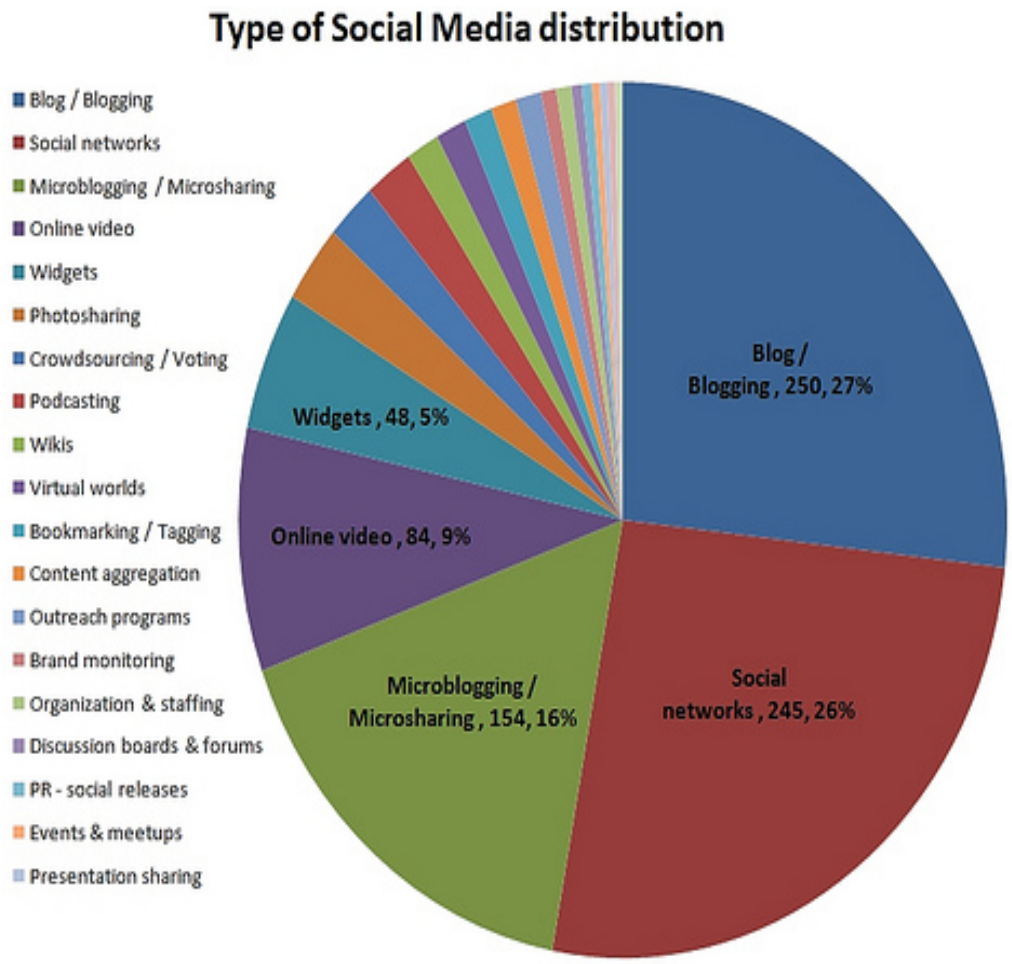

Figure 2. Type of social media distribution. [16]

As it can be observed, blogging and social networks are the most popular ones. The graphic is from March 2009. Right now (beginning of 2012) it is very probable that the distribution has changed: for example, microblogging is probably more popular now than in 2009.

2.3 Business on the Web

The evolution of the Web has also reached business. It is possible to define the differences between a 1.0 business and a 2.0 one stating that the first poured vast amounts of venture money into acquiring new customers and a (probably) unsustainable market share in the hopes of an initial public offering that would save their businesses. On the other hand, Web 2.0 companies try to figure out a profitable

path to growth and advertising-based monetization of network effects and the infinitely diverse needs of users worldwide.

The 1.0 business model was essentially flawed, as it did not target any specific market: it was a business not adapted to the special environment of the Internet, where distances did not exist and new economic concepts arose. On the other hand, successful 2.0 businesses exploit all the advantages available on the Web. In the next sections, some of the most important ones will be reviewed.

2.3.1 The long tail

The long-tail strategy is a business concept on which several 2.0 companies base their business model. Chris Anderson, the editor-in-chief of the technology magazine Wired, popularized the business model.

> The long tail is that of the demand curve of products versus sales. The best-sellers are all at one end, but as we move to the other sales drop off in a long slow curve that never quite hits zero. Traditional retailers draw a line only part-way along this curve, because **slow-moving items return less profit than the cost of stocking them. But online retailers backed by huge warehouses and fast stock deliveries can easily afford to keep them permanently available.** Helped by clever search engines that can suggest possibilities for customers with special interests, these niche items suddenly become profitable. **Amazon, for example, gets half its sales from outside its 130,000 top titles.** [17, bold added]

In other words, a typical business is limited by what Anderson calls ”the tyranny of physical space”. A normal store can only offer those products that are profitable given the clients that exist around the business: a product with a small market or even a big but spread one, is just not profitable, as the stock costs are bigger than the benefits of the sales.

But with the Web that situation changes. These kinds of items that before were only available in mail stores and that were widely considered as not very reliable can be bought in big online stores like Amazon. This phenomenon can be seen in figure 3.

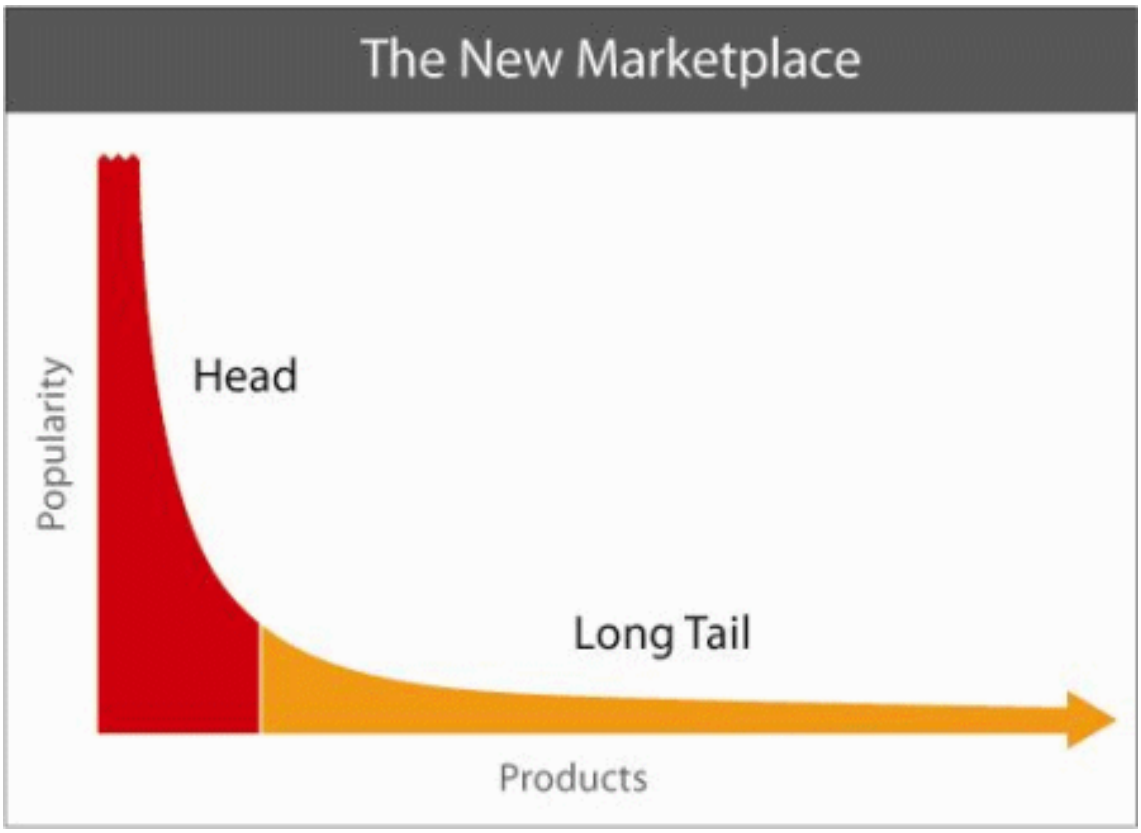

Figure 3. The New MarketPlace. [18]

Before the Internet was popular, selling not very popular products was not profitable because of space and storing reasons, and business was limited to the most popular products (the red part of the graphic). But with the web, that long tail (the orange part) is profitable for the first time. The long tail concept, illustrated in figure 3, was partially exploited by the 1.0 model, but for some reasons, one of which being the maturation of most of the Web users, the model reached a bigger success in the 2.0 era.

2.3.2 Reducing costs

Another essential concept of business on the Web is that it needs to get to a critical mass of customers: after that, more clients come naturally. The problem is that before getting to that number of users the company will have to spend money, more than a physical world business. That is called the burn rate [9,24]. Figure 4 shows the burn rate in the typical cash flow of a 1.0 business.

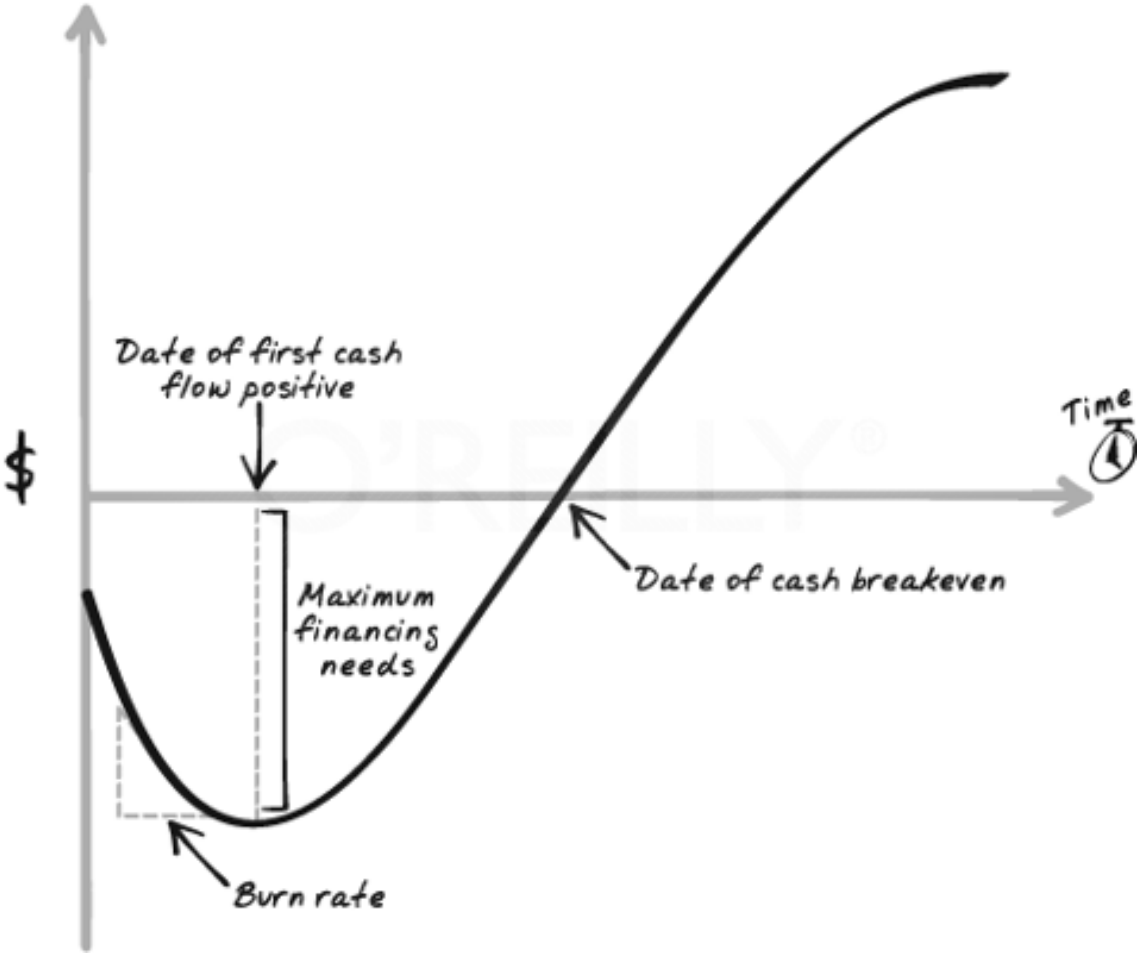

Figure 4. The burn rate. [9,25]

Essentially, new Web 1.0 businesses were trying to buy new customers and get market share with their investors' dollars. That acquisition was very expensive, suffocating businesses just when they were starting.

After the dot-com crash, entrepreneurs tried to avoid, or at least soften the cost curve. Modern 2.0 applications do that in certain ways:

- **Customer acquisition costs are smaller:** Building a community is different from attracting users. The initial investment is smaller, and then the community itself helps attract users.
- **Product costs are reduced:** 2.0 businesses usually do not have a physical product to sell, so they lack inventory. In some other cases, this advantage is only the on-demand creation of the inventory.

- **Development costs have shifted:** The typically huge initial investment in hardware and software for building a large-scale web site has declined thanks to cheaper hardware and specially free or open-source software that now is much more reliable than in the 1.0 era.
- **Iterations change the shape of the curve:** Being a small-scale community based business makes it logical to start a company and to increase the investments as the sales grow.
- **Exponential effects play a much greater role:** Achieving the critical mass before the competitors is even more important than in 1.0, as users create value. [9,34.]

With all these advantages, Web 2.0 business cost curve is not only better than of 1.0. As showed in figure 5, its initial burn rate even improves the curve of a classical, physical world business.

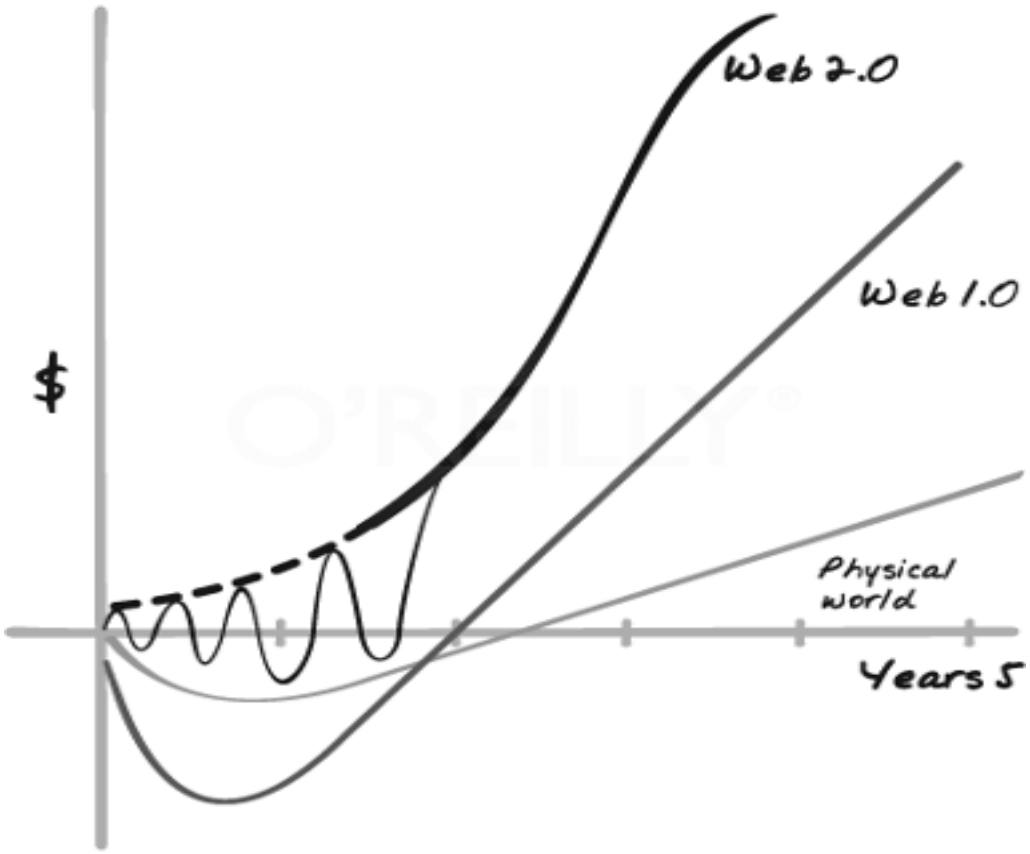

Figure 5. Comparison of the cost curve between physical world, 1.0 and 2.0 businesses. [9,33]

Most of the points made above relate with 2.0 businesses creating a community, a group of users that maintain relations among themselves. One of the ways that Web 2.0 applications use to attract new users are ”freemium” accounts. This term was introduced by venture capitalist Fred Wilson in his blog. He described it this way:

> Give your service away for free, possibly ad supported but maybe not, acquire a lot of customers very efficiently through word of mouth, referral networks, organic search marketing, etc., then other offer premium priced value added services or an enhanced version of your service to your customer base. [19]

In a 2.0 application all the users, even the freemium ones, add value to the applications, which can be monetized. For example, Flickr has been so successful at involving users that more than 85% of the photos on its service have human-added metadata, which would be extremely expensive if the company had to pay for it.

As stated by Fred Wilson, a great difference, maybe the main one, between all the 2.0 businesses is the way they monetize. On the Internet it is possible to find rather normal businesses that only use 2.0 to get added value for their products, like Amazon does with the users’ comments. Others, like Flickr, get their benefits from premium accounts and use the free users to increase the added value and to get to the critical mass of users. Finally, there is another example which is the most radical one, the one that only relies on its popularity and does not ask money from its users in any way: Facebook or Google are examples of that.

2.4 Search Engine Optimization (SEO)

In the modern Web, searchers occupy a central position in the typical workflow of any Internet user. When wanting to access some information, if the user does not know where to find it, or even if he does but needs new sites, he will go to Google and search it.

In this situation, the main way to reach users and customers on the Internet is improving the position of the site in Google searches related with the product or topic

that the site is about. For managing that, the Search Engine Optimization discipline was created.

SEO is an acronym for Search Engine Optimization. It is an Internet marketing strategy consisting of techniques to improve the indexation of a site in the main Internet searchers. For that, SEO has to foretell the behaviour of the Google algorithm, as the algorithm is kept secret by Google to avoid developers hacking the system to improve their sites' visibility. In other words, for increasing the traffic in a web using SEO, it is necessary to have a good idea about how Google works.

2.4.1 How Google works

When a user executes a search in Google, some factors define if a given site is going to appear in the results and in which position it is going to have. These elements are:

- Links pointing to the site
- Title of all the pages on the site
- URL [20,8.]

The last two only apply to deciding if the webpage is related to the topic searched using the keywords. Nevertheless, the first one (links) works in two ways: it does not only check the importance of a site, but it also helps to define the topic of a web. For example, if there are a lot of surf sites pointing to another site, it is probable that the site is also related with the surf world. Like this, they also generate the importance value of a web.

SEO experts think that the Google algorithm basically uses two variables for defining that importance rank of a site: one is public; the other one is secret. In this section both are going to be analysed.

2.4.1.1 PageRank

PageRank is an algorithm used by Google to define the importance of a website on the Internet. PageRank understands that the importance of a site only depends on the significance it has to other webs. PageRank uses links to define this popularity within the Internet; a page with links pointing to itself will be ranked better than others with no links. There is another element, too: the importance of the sites that are linking to a site also matter [21]. This process can be seen in figure 6.

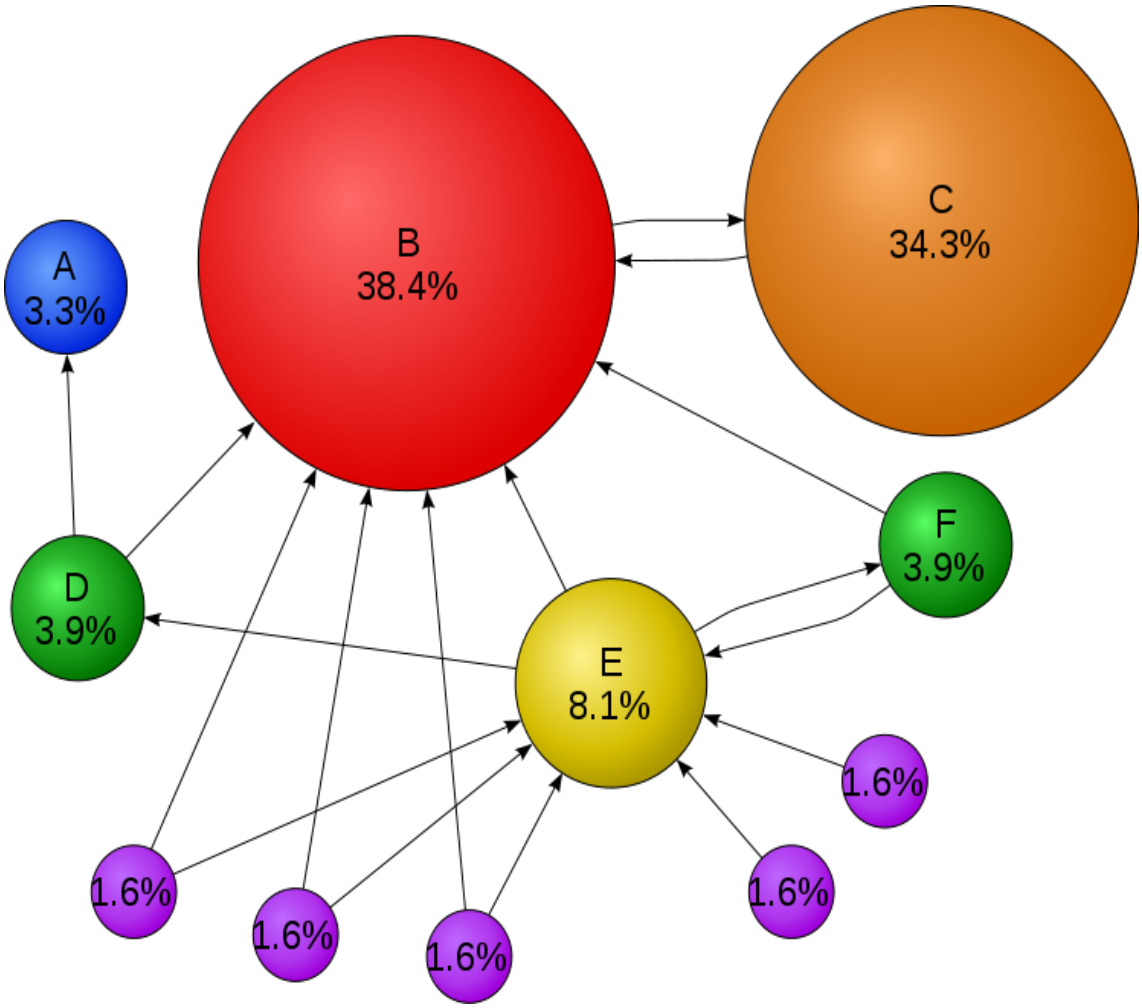

Figure 6. How PageRank works. [22]

In the figure above, B is the best-ranked site as most of the others are linking it. However, sites with only one link pointing to them (A, C, D, E and F) have different ranking, as the ranking of the owner of the link is different too.

This is still the essential concept of the Google algorithm. But the problem is that Google makes public an official PageRank that has a lot of flaws. For example, it is only updated every 60-90 days, when the real value is being updated everyday and it does not state the amount of trust that Google has on that site. For defining that trust and for talking about the real value that Google gives to websites, SEO professionals use the term TrustRank.

2.4.1.2 TrustRank

When the SEO professionals discovered Google's methodology, a new kind of industry started: the link industry. Sites with high PageRank had the power to pass some of their good ranking to any page they linked to, increasing this way the visibility of the linked page.

That, obviously, misleads search engines. To avoid this, searchers started to use another algorithm: TrustRank. TrustRank checks if a page deserves to be trusted, meaning that the page is not spam, does not feature mainly adult content or, is not selling its links. [20, 12-13] If a site loses its TrustRank this does not mean that its rank in Google searches will get worse: this only means that the site has lost the skill to pass PageRank. [20,12] From that moment, the links showed on that page will not be taken into account by the search engine.

The term TrustRank is also used in SEO terminology as a kind of true PageRank: a PageRank that is not public and is really counting the amount of trust that Google gives to the sites that are linking another site.

2.4.2 Applying SEO

SEO not only analyses the way that searchers work, but it has also has developed some techniques to improve a site indexation. If a developer knows the main elements of the indexation process, he or she can improve greatly a site visibility.

2.4.2.1 Keywords

It has already been stated that the Google algorithm works in two ways: one checking if a site is related with the topic searched and the other analysing the importance or ranking of the site. Keywords are extremely important for the first element. Keywords are the words that people usually type when they want to find something. For example, if a user wants to find information about Finland, the main keyword will obviously be ”Finland” but also others like ”snow”, ”Santa Claus” or ”cold”.

Finding correct keywords can be very important for a business. For example, in the case of an online grocery store, it is quite possible that for some months a given word gets trendy: words like ”gourmet” or ”Christmas food” during Christmas. Adding these keywords to the metadata, the links or the URL structure will make Google think that the site is a very good match for that keyword in particular.

There are some tools and techniques that permit developers to find the most popular keywords to their business or topics. Some of them are:

- **Making a survey:** It can be as simple as asking some people what they would write in Google if they wanted to find a given product. The most repeated words are probably the keywords for that product.
- **Using Google AdWords Keyword Tool:** It is a free application created by Google that lets any user analyse the quantity of search for any keyword and which other keywords are related with that.
- **Analysing the competitors:** It is important to check on the link, URL structure or specially the metapage title of competitor websites that now are in the top positions for a given market searches. If it is possible to find a repeated word that was not expected, the word is probably a new keyword. [20,23-26.]

Once the keywords have been found, they have to be added to the metapage title, links and URLs. Nevertheless, there are some techniques that will improve the effect of those keywords in the searches.

2.4.2.2 The metapage title

The metapage title is a field defined in the head of the webpage, using the <title> tag. It is also the text that appears in the upper part of the browser during the visit to a website and the blue underlined result that Google shows whenever a user makes a search. It is a very important element in the way that Google algorithm works and there are some rules to use it to improve the indexation:

- **Be short**: If the title is too long, it is possible that Google marks the page as spam. 100 characters should be enough.
- **Do not repeat keywords**: Google already makes all the combinations possible with the words inside a title, so it is not necessary to write the same keywords all the time. For example, if a site wants to sell football T-shirts, a good metatitle would be ”SportsStore – Madrid, Barcelona, Munich, Manchester T-shirts” and a bad one ”SportsStore – Madrid T-shirts, Barcelona T-shirts, Munich T-shirts”. Actually, the latter one would probably be marked as spam. [20,29.]

The best attitude when writing a metapage title is avoiding tricks and using keywords in a reasonable way.

2.4.2.3 Links

Links are important in defining the ranking and the similarity to the topic of a given website. Some good techniques to improve the index of a site using links are:

- **Mixing text links and image links**: Text links have a bigger effect on the keywords, but it is accepted that if an unnatural quantity of links are of text type, that will have negative effects on the ranking, as it is probably a spam.
- **Using keywords in the links carefully**: Until 2008, just adding the keywords to the text link or the alt field in the image link was enough, but Google engineers decided to change the behaviour of Google to define too many links with a given keyword pointing to the same site as ”suspicious”. Mixing the use of the keywords and the links is important: some links with only the keywords,

other with the keywords and other non-important words, other without the keyword inside the link but in the same paragraph. [20,31-36.]

Again, the general idea is to create as natural links as possible, for avoiding that the Google algorithm detects it as a hack.

2.4.2.4 URL structure

The URL is the text a user writes in the navigation bar to get to a given webpage (for example, 'www.google.com' or 'www.facebook.com'). Some other techniques to follow when adding the keywords to the URLs are:

- **Use a domain with the main keyword in it**: If that is not possible or is too expensive, the best option is to add another word. For example, if 'www.finland.com' is not available, an option would be 'www.yourfinland.com'.
- **Create friendly URLs**: URLs should be as readable as possible, and include the keywords in a natural way. [20,36-38.]

Nowadays there are tools that connect the URL with the data used in the content. These kind of tools can be very useful to improve the indexation of our site.

2.4.2.5 Time

Time is important because of two factors:

- **Changes in Google ranking are not immediate**: a web developer needs to wait until the SEO implementation he or she perfomed improve the ranking of the website in Google.
- **Google penalizes new websites**: a new page will not appear on the top of any meaningful keyword search in Google for some months. [20,39-40.]

It is generally accepted that a new site will not have a good PageRank until some time has passed.

3. The movie industry and the social networks

This section will give a short overview on the most important online applications that the general audience uses around the movie world. With this, it will be possible to determine which features are more successful among the movie lovers and which possible features have not been created yet. This kind of analysis will help the development of TweetMovies.

3.1 IMDB

Clearly, the most popular application about movies, IMDB (acronym for Internet Movie Data Base) receives 100 million requests every day. That makes it the most popular movie application on the Internet.

The purpose of the website was initially quite simple: the purpose was creating the most complete database about movies, a place where one could check data about any movie ever made. However, as years went by, IMDB tried to increase its scope with new features, using its great position among the audience as the best competitive factor. Some of these features are:

- **Support for professionals:** IMDB is not only working as a source for movie fans. One important part of its profits comes from cinema professionals that pay money to be and have access to their contacts list. As surprising as it may seem, for the movie world IMDB is what LinkedIn is for the business world.

- **Ratings:** This is a very popular feature, as the IMDB Top 250 has become one of the most important "best movies lists" in the world. The IMDB rating system uses its own special algorithm. This means that the rating of a movie is not the medium result of all the users' grades; the system also evaluates other variables such as the quantity of users that have seen the movie.

- **Lists:** In IMDB users can create lists of movies, such as "My favourite westerns". This feature is quite popular in movie applications, probably because cinema magazines love to publish this kind of lists. However, in the case of the

IMDB implementation, it has not been successful. The main problem probably is that even if the feature works well, it does not connect with anything: a user can only create his list as a reminder of some movies, but he is not able to access similar lists of his friends, people with similar movie tastes, or the general audience.

- **Integration with other social networks:** IMDB offers links for its users to tweet or change their Facebook state talking about a given movie.

In general, the social features of IMDB are not good. That happens not because of a problem in the code or a lack of publicity; the problem is that IMDB is still thinking of itself as a giant database, and all the other features are just add-ons. That is very easy to see from a usability point of view: all the features that are difficult to access are hidden behind small icons and underexplicative menus.

IMDB lacks a social plan and the real intention of creating a new interaction between the users and the application. IMDB needs a plan more careful about the things that users can do in the application and between themselves than about the data that the website will show to the user. In short, IMDB needs to evolve from 1.0 to 2.0. That can be seen even in the design, as figures 7 and 8 illustrate:

Figure 7. IMDB index page. [23]

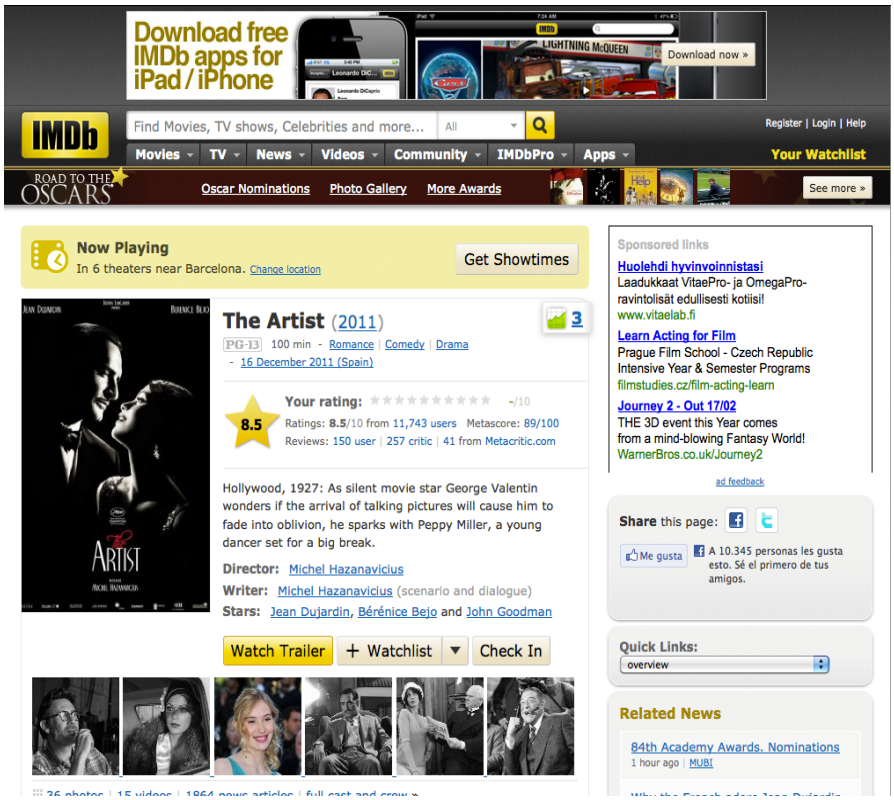

Figure 8. The page of a random movie from IMDB.com. [24]

The IMDB interface is full of text, and that makes it not very usable. Of course, that depends on the purpose of IMDB for users: if it is to have the biggest movie database on the Internet, IMDB is already complete; however, as a Web 2.0 application, it comes too short.

3.2 FilmAffinity

A Spanish web application about movies, FilmAffinity, hosts two versions: one in Spanish and another in English [25]. Despite that, the popularity of FilmAffinity is essentially limited to Spain and the rest of the Spanish-speaker countries.

Given that IMDB is a giant in everything related to movie data, any application that wants to introduce itself successfully on the Web has to find a different space: this concept was also important during the development of TweetMovies. In the case of FilmAffinity, its main feature is to be able to recommend movies based on the mark that other users give to them. There are two ways of doing this: first, checking the general list of best movies ('TOP Filmaffinity') where any user can see the medium grade given to a movie by all the users of the application; the other way is to generate a list based on these parameters:

1. Movies the user has not evaluated: it is assumed that he has not seen them.
2. Movies with a better medium mark by the 20 users with the most affinity with the user who is generating the list.

The affinity is a value created by the application itself: it is an attribute based on the similarity between the mark that two users give to the same movies. FilmAffinity assumes that two users giving similar grades have similar movie tastes and then recommends to one of the users the movies that the other has liked. To improve the chances, the application uses the medium grade of the users with most affinity.

Other important features of FilmAffinity are:

- **Information:** In the IMDB chapter it has already been stated that it is probably impossible to compete against IMDB about the quantity of

information. Thus FilmAffinity has taken the quality approach. Movie data is used to be user-friendlier: there are more complete and better-written plot synopses, critics' opinions and other features.

- **Critics:** In FilmAffinity, users can post their own critiques about movies; users can read and rate movies, and the most popular ones appear first, making it simpler for other users to access the most interesting information about the given movie.
- **Integration with other social networks:** FilmAffinity offers links for its users to tweet or change their Facebook status talking about a given movie. However, the links are poorly located, so rarely a user tweets or changes his Facebook status from the website.

In the opinion of the author of this thesis, FilmAffinity has an interesting concept and it has been implemented correctly. The main problem that FilmAffinity has, as a social network, is its lack of almost any update since its creation: the interface looks very old, as shown in figure 9.

Figure 9. FilmAffinity interface. [26]

The graphic design of FilmAffinity is that of 1.0: much of the information is on the index page, and it has a very small font size that does not attract the user. This also makes the application extremely difficult to use from a mobile phone: the links are very small and it is hard to click on them with the fingers.

Another big problem for FilmAffinity users is the lack of interaction between them: FilmAffinity, being an extremely popular social site about movies, its users would like to interact, but the only way to do so is using private messages that do not enable group discussions and that are difficult to find on the screen. This is another graphic design problem. This issue was so big that the users created, on their own, a message board called *El foro que surgió de FilmAffinity* [27] which means, literally, *The message board that came up from FilmAffinity*. Being small and completely amateurish (it is created on a free message board platform) the board is not interesting by itself, but what it means is that when users are creating their own platforms to express themselves on the application and without help, it is obvious that the application is not developing correctly in a 2.0 way.

4. Technologies

The most basic tool when developing an IT application is a powerful language that is adapted to the given environment where the application is going to be used; in this case, the Web. Nevertheless, the choices do not stop here: a group of technologies have to be chosen (from a testing suite to a deployment site) for making possible the correct development of the software. The purpose of this section is to present and explain the technologies chosen for the creation of TweetMovies.

4.1 Ruby

Ruby is an object-oriented programming language created by Yukihiro "Matz" Matsumoto. Its popularity increased after the creation of the web application framework Ruby on Rails.

One of the most repeated quotes by the Ruby developers is that in Ruby everything is an object. Classes are objects; modules are objects; even the Ruby starts already in an object (main, of the class Object).

Inspired by SmallTalk, one of the main characteristics of Ruby is its human-like syntax: in front of new circumstances, the beginner to Ruby can try to solve them with an unusually high rate of success just writing words (in English) as if the compiler was a human-like being.

However, what are the reasons behind Ruby's popularity in the web community? Is it popular only because of Rails? If this is the case, why did the Rails creators choose Ruby instead of PHP or Perl? In the next chapters, the main characteristics of Ruby are going to be explained. The chapteres are going to proceed from its user-friendly syntax to some of its more advanced features, powered by its capacity to modify the behavior of the program during runtime, which most of Ruby developers call it metaprogramming.

4.1.1 Syntax

Even before the creation of Rails, Ruby was identified with high usability and accessibility. This is because of the simple and highly readable names used in its most basic functions. Below, the creator of Ruby defines these features.

> Often people, especially computer engineers, focus on the machines. They think, "By doing this, the machine will run faster. By doing this, the machine will run more effectively. By doing this, the machine will something something something." They are focusing on machines. But in fact we need to focus on humans, on how humans care about doing programming or operating the application of the machines. We are the masters. They are the slaves. [28]

A developer can check the emphasis that Ruby syntax has in a good user interface even in the most basics interactions with the language. Some examples of this are given below:

Writing in the default output is shown in listing 1:

```
puts 'Hello World!'
```

Listing 1. A simple writing.

Calling a method with each of the values in a vector is shown in listing 2:

```
vector.each do |p|
  method(p)
end
```

Listing 2. Iterating in a vector.

Repeating an action any number (in the example, 5) of times is shown in listing 3:

```
5.times do
  action
end
```

Listing 3. Caling a method with an enumerator.

It is obvious that this kind of syntax makes the life of the developer much easier.

4.1.2 Array iterators

In the previous chapter the main method to iterate an array, *each*, was described. This method is simple and usable but some iterating features are missing. Thus, it is, for example, important to find out if is possible to change all the values inside an array.

Ruby provides the developer with some powerful methods to manipulate arrays. The most important ones apart from *each* are *each_with_index*, *map*/*collect* and *inject*. Some examples are given belos. How they work is also described:

each_with_index

An example of the use of each_with_index is shown in Listing 4.

```
numbers.each_with_index do |value,index|
  puts 'In the number '+index.to_s+'the value is 'value.to_s
end
```

Listing 4. Iterating with each_with_index.

The example above uses *each_with_index* to access to the index of an array at the same time than its value.

map/collect

An example of the use of map is shown in Listing 5.

```
numbers.map do |v|
  v+1
end
```

Listing 5. Iterating with map/collect.

The *map* and *collect* methods (which are synonyms) write on the given attribute the value that is processed in the block. In the example above, one unit is added to each of the values inside the array.

inject

The method *inject* is probably one of the more useful methods in Ruby; it saves a lot of code compared to the way of iterating an array in almost any other language. Listing 6 gives an example of *inject*.

```
res = [1,2,3,4].inject(0) do |r,v|
  r+=v
end
```

Listing 6. Iterating with inject.

When iterating with *inject*, two variables are passed to the block: the first (*r* in the example) is one that will be brought to each of the iterations, and it will normally be used for calculating the final result; the second (*v* in the example) is the value in one position in the array. After iterating, *inject* returns the final value of *r*; in this case, after executing the code, the value of *res* will be 10. [29, 29-42]

4.1.3 Open Classes

We will define open classes with an example. In a given project it could be decided to redefine an essential class of the language (String, for example). With most of the languages this would be extremely difficult, and usually developers try to avoid that. With Ruby it is extremely easy, as it is possible to add or modify methods of a given class in any moment, even during the execution of the code.

For example, a developer might want to change the method clear of the class String. Usually this method transforms the String to an empty String but in listing 7, the developer is going to change it to make it return an empty space (" "). It would be like this:

```
class String
  def clear
    return " "
  end
end
```

Listing 7. An example of Open Classes.

Editing an existing method of a given class is called MonkeyPatching [30,9]. Obviously it is quite a dangerous resource, but used wisely, it can be quite useful.

This feature, added to the RubyGems system enables any Ruby developer to easily modify the language behavior, and then share his changes with the community.

4.1.4 Dynamic Methods

Another very useful feature that the dynamic status of Ruby offers to the developer is that the Dynamic Methods enables the user to define methods after the coding phase, which means that they will be defined during execution.

For example, an application that provides the number of each type of vehicle in a given area has been developed. For doing that, the application calls a service that will send back the name of each type of vehicle and the number in a Hash. It is required to make a method for each type of vehicle that will return the number associated to it. The problem is that the API is still in development, and the types can still change. However, with Ruby it is not needed to depend on the external code anymore: it is possible to make a dynamic solution that will adapt automatically to the changes. In listing 8, an example of this is given [30, 41-48.].

```
vehicle_types.each do |v|
  define_method(v.name) do
    return v.number_of_vehicles
  end
end
```

Listing 8. An example of Dynamic Methods.

This solution takes the vector *vehicle_type* for each value (they are Hashes of two elements, *name* and *number_of_vehicles*) and uses the method *define_method* to create a new method with the name passed; in this case, each of the names inside of the vector.

4.1.5 Method_Missing

A brother feature of Dynamic Methods, *method_missing* is behind one of the most flashy things in the ActiveRecord management of data. In Rails, it is possible to find an object of a given class by any of its attributes, like this: *Model.find_by_nameoftheattribute(name)* (for example, *User.find_by_name('Mikka')*). The generation of the methods is dynamic, so it can be used with any attribute, even with one with a weird name. Most of Rails beginners are surprised at that.

This is because Rails makes possible to change any of the methods, even the system ones. And *method_missing* is exactly that: a system method. It is a method that, if it is impossible to find a method in the object, calls its superclass. If the object does not have a superclass, it raises an error. Changing *method_missing*, it is possible to take any call that does not find a method and check dynamically if it is something that the code can use. An example of how method_missing works can be seen in listing 9:

```
Module Kernel
    def method_missing(method)
      method = name.to_s
      if method.to_s.length(13)
        puts 'I don't like methods with 13 characters'
      end
   end
end
```

Listing 9. An example of *method_missing*.

This method writes a message for any non-existing method that is 13 characters long. The value of the arguments given to the method can also be checked, or even the block passed. This great flexibility gives a lot of power to the developer. [30, 48-67]

4.1.6 Alias

The alias macro makes a copy of a method with another name. With this it is possible to redefine a method using the old version of the same method in the new one. In listing 10, Open Classes and Alias will be combined to change the behavior of Ruby when adding numbers:

```
class Fixnum
  alias :newsum :+
  def +(arg)
    newsum(arg.newsum(1))
  end
end
```

Listing 10. Using *alias*.

Listing 10 redefines how Ruby adds Fixnum, a Ruby class that is very similar to Integer. First the adding method (+) is saved in newsum, and then it is redefined: the new result will be the same as the old one plus one unit: 1+1 = 3.

4.1.7 Attribute Access

In other languages, developers are used to having a long and boring job: to write access methods to each of the variables of a class. In place of that, Ruby provides three macros to avoid this: *attr_accessor*, *attr_writer* and *attr_reader*. As the names suggest, *attr_accessor* implements reading and writing methods, *writer* permits to change the value and *reader* to read it. An example of its use is given below.

```
Class User
  :attr_accessor :name
end
```

Listing 11. An example of *attr_accessor*.

Listing 11 creates the variable name with read and write accesses.

There are other macros in Ruby relative to attribute access: for example, *attr_accessible* and *attr_protected*. They permit or forbid to mass-assign a given attribute.

4.1.8 Blocks and Procs

Blocks and Procs are the ways that Ruby has to handle closures. A closure is a piece of code that is not executed just in the obvious order of the code. However, a closure is not only a function or method. A closure can be passed to another block of code, for the receiving part to execute it when it needs to.

There are two ways to build a block in Ruby:

- `do code end`
- `{ code }`

These blocks can be passed to other objects and executed there with other variables.

The next step is to show how the blocks work and how that can be useful for developing. The example in listing 12 is going to implement a method to execute the same block of code for all the values inside of an array:

```
array.iterate! do |a|
  a*2
end

class Array
  def iterate!
    self.each_with_index do |n, i|
      self[i] = yield(n)
    end
  end
end
```

Listing 12. An example of blocks. [31]

The main point of interest in listing 12 is the *yield* method that executes the passed block replacing the variable inside the // with a new value. This is quite useful: for example, Rails passes the entire HTML to a controller, and after working out the value of the needed variables, it executes it with *yield*.

However, what if the developer wants to have many different blocks at his disposal and use them multiple times? Should the code pass the same block again and again? It is important to remember that Ruby is fully object-oriented, so this can be handled quite cleanly by saving reusable code as an object itself. This reusable code is called a Proc. The only difference between regular blocks and Procs is that a block is a Proc that cannot be saved, and as such, it is a one-time use solution. The next example is the same as the latest, but using Procs:

```
array.iterate!(double)

double = Proc.new do |a|
  a*2
end

class Array
  def iterate!(code)
    self.each_with_index do |n, i|
      self[i] = code.call(n)
    end
  end
end
```

Listing 13. An example of Procs. [31]

There is a third way to declare a block in Ruby: Lambdas. They are very similar to Procs, and actually there are only two very subtle differences between them: Lambdas check the number of arguments passed to the block and they raise an error if the number is not right. Further, Lambdas execute themselves fully, that meaning that the block does not finish when it finds a *return*, but continues executing until the end. The differences, as it can be observed, are only important in very specific cases.

4.1.9 Conclusion

The methods described above are only some of the special techniques that can be applied while programming in Ruby. There are some other important features that have not been discussed, but Ruby is too different and fascinating to explain it fully in such a short space. In the last chapter of this thesis, References, the reader should find some books and blogs about Ruby where more information about the language can be found.

To end this chapter, it is important to remark that Ruby gives a lot of power to the developer: it is extremely flexible letting the developer to modify the behaviour of the language in almost any way he likes. However, it is important to remember that with great power comes great responsibility. When these special features of Ruby are used, they can also provoke great, almost impossible to track, errors. That is why developers tell that Ruby is a language for grown-ups.

4.2 Ruby on Rails

Despite all the features of Ruby, it was not a very popular language for a long time. It lacked having a big company behind it and it also was a bit too heavy and slow. However, everything changed when Rails appeared. The popularity of Ruby increased to the level of the most important dynamic languages, as it can be seen below.

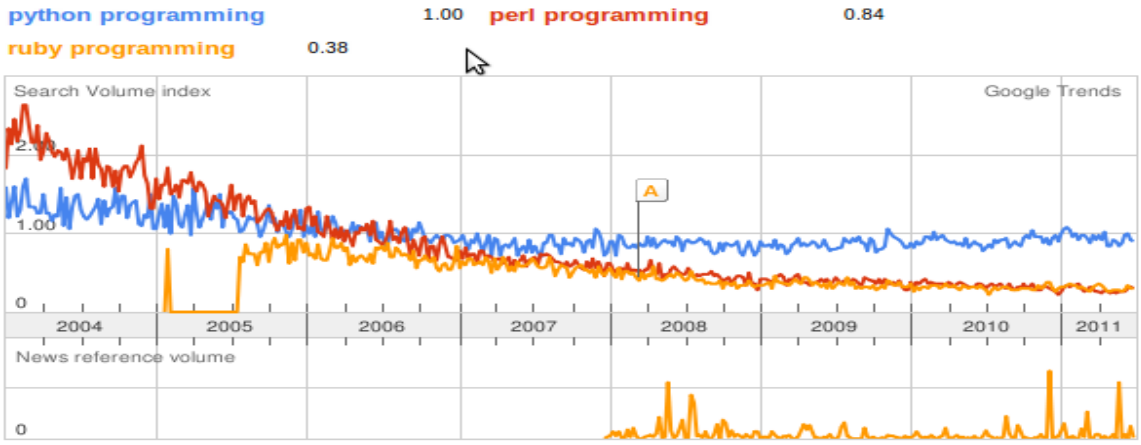

Figure 10. Popularity (calculated by number of searches in Google) of Ruby. [32]

Ruby on Rails is an open-source web application framework. David Heinemeier Hansson created it in July 2004 but he did not share commit permissions until February 2005. [33] In the figure above, it can be seen that the term "Ruby programming" was not popular at all until the dates of the creation of Ruby on Rails.

Arguably the most popular framework in the world, Rails presents a variation of the MVC architecture pattern, which is showed in figure 11.

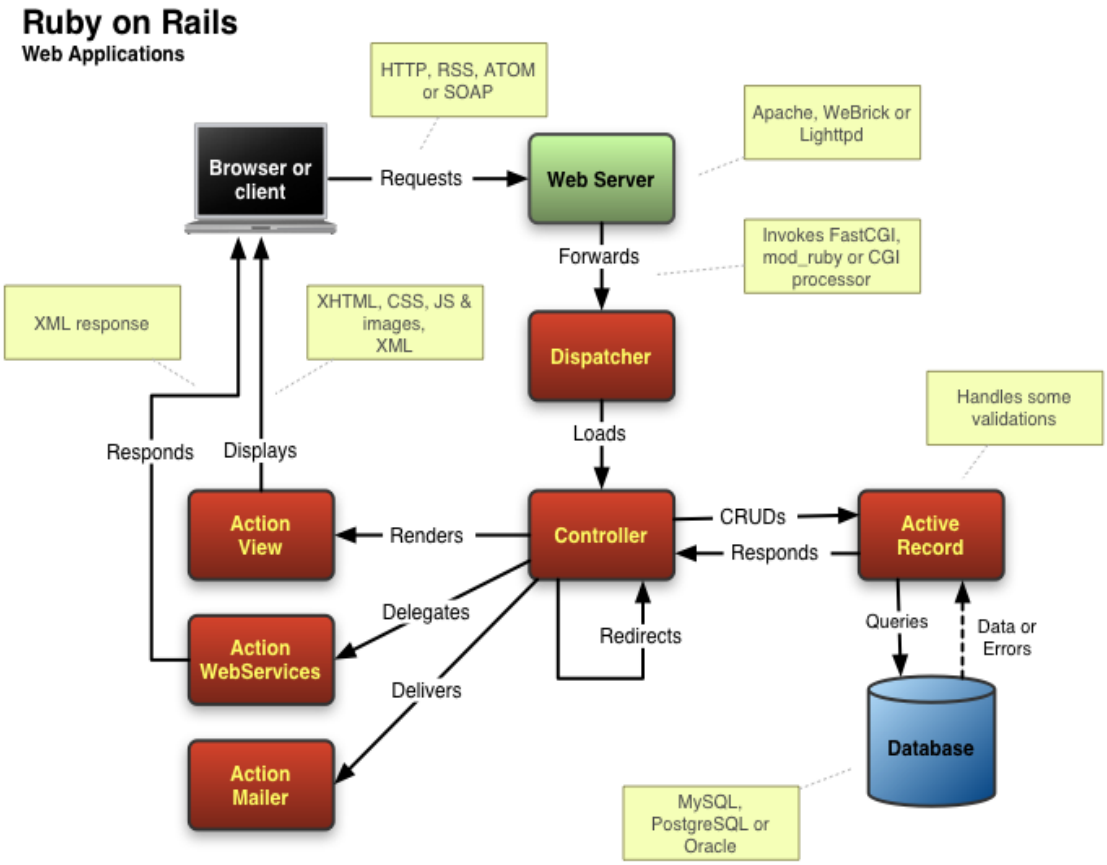

Figure 11. MVC implementation of Ruby on Rails. [34]

The main difference with the normal MVC implementation is ActiveRecord, a system that manages all the data of the application (the "models" in MVC). This feature is quite important and actually it has been cut as a gem of its own, meaning that a developer can use it even when not using a Ruby on Rails application. This will be described in the next chapter.

Development in Ruby on Rails follows some of the most popular paradigms in the Agile methodologies. Some of them are:

- **DRY (Don't repeat yourself):** This principle is stated as *"Every piece of knowledge must have a single, unambiguous, authoritative representation within a system"*. [35] It places special emphasis on avoiding the duplication and favouring the reuse of code.
- **CoC (Convention over Configuration):** This paradigm considers that it is better to decrease the number of choices that has to be made before starting to code a project. This does not mean to have less flexibility: it only means that the framework will take a choice for the developer until he changes it using configuration. For example, Rails supposes that a model called User will be based on a table with the name "Users".

4.3 ActiveRecord

ActiveRecord is a Ruby gem used by default by Ruby on Rails. ActiveRecord connects business objects and database tables to create a persistent domain model where the logic of the objects and the data inside the DB is only in one place. It is reasonable to say that it is the ultimate support for object-oriented software or an application.

ActiveRecord is an implementation of the object-relational mapping (ORM) pattern. This is the definition of that pattern given by its creator:

> An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data. [36]

ActiveRecord also adds inheritance and association to the ORM pattern, making it more simpler for the developer.

ActiveRecord provides the developer with some useful methods to access the database. Using some advanced features of Ruby like metaprogramming, it automatically generates finder methods for every attribute in every model of the data. For example, in a model User with attributes named name and birthdate, methods like

this can be used: *User.find_by_name(name)*, *User.find_by_name_and_birthdate(name, birthdate)* or even *User.find_by_birthdate_and_name(birthdate, name).* ActiveRecord also generates read and write methods for all the attributes: only for those found in the table, dynamic ones are not persistent, and therefore, they are not treated by ActiveRecord.

There are alternatives to managing databases and models in Ruby, the most popular one being Datamapper.

4.4 RubyGems

RubyGems is the most popular package manager of Ruby: with it, it is extremely easy to install Ruby programs and libraries. RubyGems is formed by a standard format, the "gem", for the Ruby code. It is a tool for managing the installation and updating of gems, and it can be called in bash using the "gem" command. It is also a server for distributing the gems.

In January 2011, more than 33 thousand gems had already been created (or as Ruby developers say, "cut") and there had been almost 450 million of downloads of gems [37]. Thus RubyGems proves that open-source projects created by a great number of developers are competitive.

However, maybe the most important thing about RubyGems is not its software, but how professionals use it: Ruby developers, when facing a typical coding problem, try to find and create a general solution that fits to the needs of their fellow colleagues. From big and complex utilities for registering and administering users and access levels (Devise) [38] to simple regular expression generators (EasyRegExp, created by the author of this thesis) [39] or even tools that automatize the authentication in third-party APIs such as Facebook or Twitter (Omniauth) [40], a Ruby developer can find easy and quick solutions to problems that in other languages would be hard to solve.

Rubygems and its success is part of the Web 2.0 phenomenon, the developers' ways of being social. The other main actors of this Web 2.0 revolution in the coding sphere are Git and Github.

4.5 Git

Git is a control version system, developed by Linus Torvalds for the development of the Linux kernel: its main point when it was created was speed. It is free software.

The main characteristics of Git that have made it so popular among Ruby developers is its strong support of non-linear development: creating a branch and merging a repository is really quick, and the structure of the repository being based on small pieces of changed code called "commits". Thus, a developer can even import only a part of a branch without having to use the rest of it. Nevertheless, it is probably easier to show the power of Git with an example of a normal project using it, like the one in figure 12.

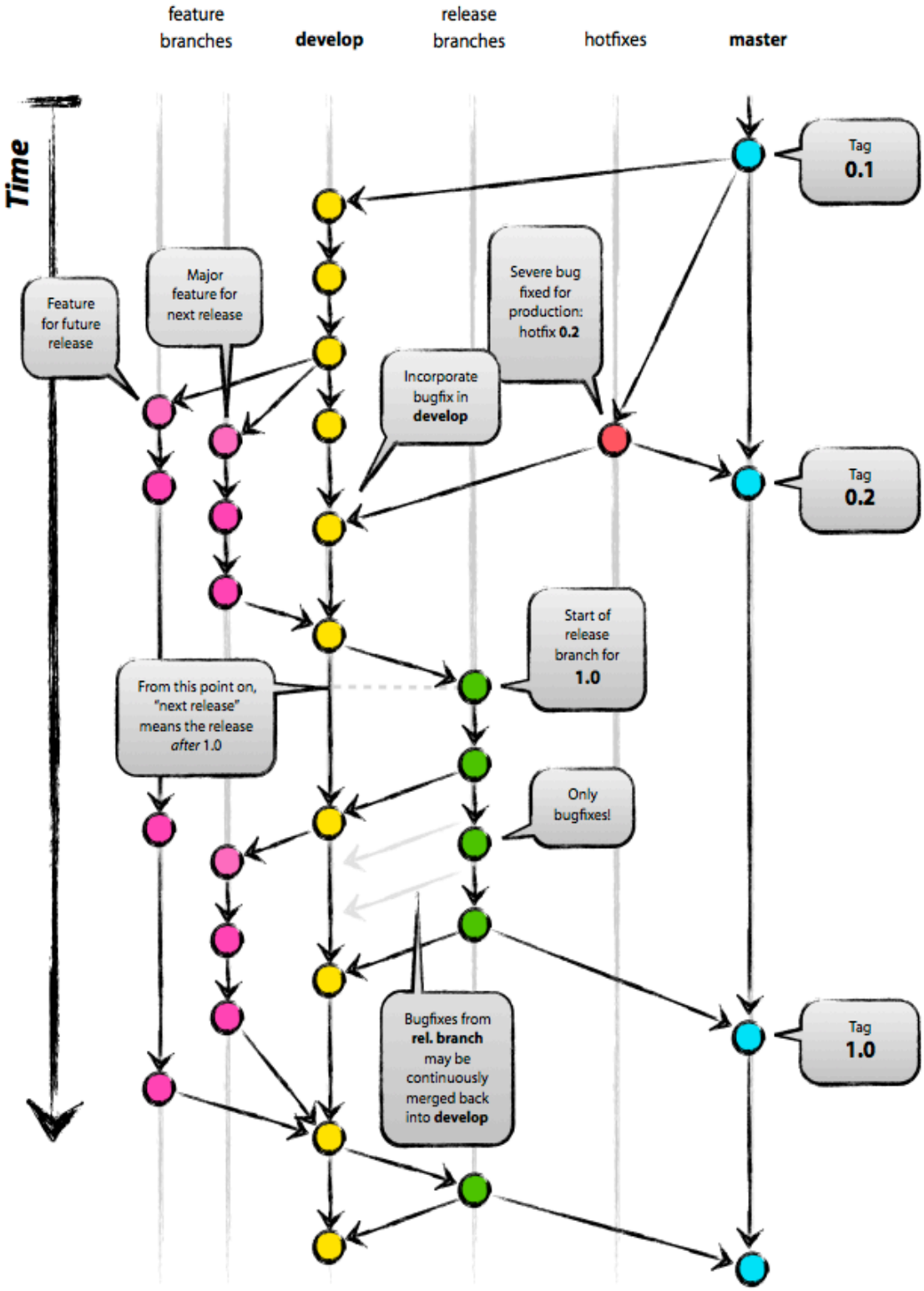

Figure 12. Structure of the branches in a Git project. [41]

In the project structure showed in figure 12, all the branches of the project can be seen (the vertical lines). They are:

- **Master:** The main branch of the project and the one that is created by default when the git repository is started. It is used as the public part of the project, as the one that the user has access to.
- **Hotfixes:** Branches of this kind are created when an error is found on the master branch: they are used to develop the patch for solving the issue and when that is done, the hotfix is merged with the master branch. Obviously, they have a short life in the development process. Some users do not use this kind of branches, repairing bugs in the master branch directly.
- **Release branches:** Branches that incorporate some hotfixes (the ones for minor bugs that do not need to be solved right away) and new features from the develop branch. A release branch is, in short, a pack of the changes that the new version of the project will include.
- **Develop:** This is the other main branch in a Git-driven project. It is the base camp for all the new features. A develop branch always marks the space for the next version of the software; when it is merged with the release, a stage (version) of the branch is finished and another starts.
- **Feature branches:** These are branches created from Develop. Each new feature for the project will need a feature branch for itself; the name of the branch will be a reference to this new feature (for example, in TweetMovies the author of this thesis had feature branches with names such as *new_login* or *hashtag*).

4.6 GitHub

GitHub is a web-based hosting service for software projects using Git, or, said in another way, a repository of Git projects accessible by Internet. So far, it looks like a traditional service. Nevertheless, Github it is not traditional at all.

Thanks to its friendly interface and social network features, developers can follow projects and other developers; they can also fork a project to their computer to add some functionality they think is missing or repair a bug and then ask the owner of the

original software to add the change to the main branch. Like this, software started by one developer can be changed by lots of others, and a project that had one original purpose can change and increase its scope thanks to the collaboration of the Ruby community.

However, Github is not only a place for sharing code: a user can also pay for a private space to save projects and make them available only for the people who are going to work in them. Usually, companies use the private space for the projects they develop for clients and the public space for personal projects of its employees: having created important open-source projects improves greatly the reputation of any company, and with that, its income.

4.7 Devise

Devise is a flexible authentication solution for Rails. It solves one of the most typical problems that a developer faces: creating a reliable authentication system.

Devise provides the developer with some helpers like *current_user* that returns the User object logged in the current session. Another useful helper is *before filter :authenticate_user!* that combines the *before_filter* macro with a devise method to limit access to controllers or even actions within controllers. The are also some generating methods accessible from the command line, like *rails generate devise MODEL*, that create models, controllers or views depending on the given options.

The authenticable models can be configured using the 12 devise models:

- **Database Authenticatable:** encrypts and stores a password in the database to validate the authenticity of a user while signing in. The authentication can be done both through POST requests or HTTP Basic Authentication.
- **Token Authenticatable:** signs in a user based on an authentication token (also known as "single access token"). The token can be given both through query string or HTTP Basic Authentication.

- **Omniauthable:** adds Omniauth support. OmniAuth will be described in chapter 4.9.
- **Confirmable:** sends emails with confirmation instructions and verifies whether an account is already confirmed during sign in.
- **Recoverable:** resets the user password and sends reset instructions to the given mail.
- **Registerable:** handles signing up users through a registration process, also allowing them to edit and destroy their account.
- **Rememberable:** manages generating and clearing a token for remembering the user from a saved cookie.
- **Trackable:** tracks sign in count, timestamps and IP address.
- **Timeoutable:** expires sessions that have no activity in a specified period of time.
- **Validatable:** provides validations of email and password. It is optional and can be customized, so a developer can define his own validations in the web formulary.
- **Lockable:** locks an account after a specified number of failed sign-in attempts. Can unlock via email or after a specified time period.
- **Encryptable:** adds support of other authentication mechanisms besides the built-in Bcrypt, the default one.

Each module adds some methods, filters and attributes to the model class. Adding a devise module to a class is as easy as adding a macro with the name of the module *:rememberable, :registrerable , :validatable* and the rest [42].

4.8 Haml, Slim and Sass

Since the creation of the Web, coding readable and validating HTML and CSS code has been a great objective for developers. For achieving that, there are some markup languages and extensions that make writing code much simpler. HAML and Slim are used for generating web templates; Sass, in its scss extension, is the default CSS template for Ruby on Rails.

4.8.1 Haml

HAML (HTML Abstraction Markup Language) is a lightweight markup language used to describe/generate web documents avoiding inline coding. HAML works like a replacement for inline page template systems such as PHP, ASP or, in the case of Ruby and Ruby on Rails, eRuby.

HAML follows the DRY (Don't repeat yourself) paradigm. It tries to erase all the repetitions that HTML and the usual page template systems have, and one of the major sources of duplication in web code is the closing part of tags.

Tags are the way that HTML uses to indicate nesting. For example, a code like *<body> Hello </body>* indicates that all the properties of the body apply to the "Hello" text; in other words, "Hello" is nested into the body tag. In HTML closing tags, like </body> are needed, to show where the nested code ends.

HAML uses another way to solve this problem: indentation. Indentation was already used as an informal way to improve web template readability, but it was not parsed. HAML parses it using a default indentation of two spaces; all the lines of code under a given tag that are moved two or more spaces to the right are nested in the tag. The last example in HAML would look like listing 14:

```
%body
  Hello
```

Listing 14. An example of HAML.

As it can be seen in the last example, HAML also tries to save some code using abbreviations. Some more examples can be seen in table 1. In the table, some typical expression are compared to their equivalent in HAML:

Table 1. HTML-HAML comparison.

HTML	HAML
<body>…</body>	%body
…	.main

…	#first_column

In a web template there is also a normal code; in this case, written in Ruby. A complete example of a web template should include not only HTML elements but also code. In table 2, two equivalent codes are given in ERB (the default page template of Rails) and HAML to compare their performances:

Table 2. An ERB document and its HAML equivalent. [43]

ERB	HAML

  <%= print_date %>
 	
  <%= current_user.address %>
 	
  <%= current_user.email %>
 	
  <%= current_user.bio %>
 	
	#profile
	  .left.column
	  #date= print_date
	  #address=
	current_user.address
	  .right.column
	  #email= current_user.email
	  #bio= current_user.bio

As it is obvious seeing the example above, HAML code is much shorter and simpler.

4.8.2 Slim

Slim is another lightweight markup language used to generate web documents. It is an evolution of HAML, making it even lighter; Slim essentially erases some of the symbols, meaning that the code is shorter, as can be seen in table 3.

Table 3. A table showing equivalent expressions in HTML, HAML and Slim.

HTML	<h1 id =’hello’>Hello! </h1>
HAML	%h1{id=’hello’} Hello!
Slim	H1 id=’hello’ Hello!

In the example, it is possible to see the main difference between HAML and Slim. It is the disappearance of the % symbol each time that a native HTML tag should be used. This improves readability and saves time for the developer.

Slim uses indentation for nesting exactly in the same way as HAML, being extremely easy to learn for any HAML user.

4.8.3 Sass

Sass (Syntactically Awesome Stylesheets) was the CSS twin of HAML, as the same developer, Hampton Catlin, created both. The same team developed both projects until they separated in 2010. In fact, the old syntax (.sass) used indentation to indicate nesting. The new syntax (.scss) is the default CSS template for Ruby on Rails, and it does not use indentation and it is compatible with vanilla CSS stylesheets.

Sass is a CSS3 extension. It adds new features to the usual way in which CSS stylesheets work, changing its nature to a more dynamic one.

4.8.3.1 Variables

In Sass, it is possible to define variables using the $ symbol to reuse them in the stylesheet.

```
$red: #AA0000;
$default_margin: 15px;
.column{
  color: $red;
  margin-left: 15px;
}
```

Listing 15. Using variables with Sass.

As it can be seen in listing 15, it is easier to redefine the general appearance of a website. It is easier and quicker to configure the colour palette, margins, paddings and

the rest of CSS features. After that, those values can be invoked quickly only by using the variables.

4.8.3.2 Nesting

Sass applies the DRY paradigm by making nesting selectors and properties within one another possible.

```
table.main{
  border: 1px;
  td{
    font-weight: bold;
  }
}
```

Listing 16. Using nesting in Sass.

In listing 16, *td* is nested inside the table with the class ”main” so the *font-weight: bold* property will only apply there.

4.8.3.3 Mixins

Mixins allow the developer to reuse whole groups of CSS properties; it is even possible to pass them arguments.

```
@mixin right($margin){
  float: right;
  margin-right: $margin;
}
.right_column{
  @include right(15px);
}
```

Listing 17. Using mixins in Sass.

In listing 17, the *div* with class *right_column* gets all the propierties of the mixin right, with the variable *$margin* with value 15px.

4.8.3.4 Selector Inheritance

Sass, trying to imitate the behaviour of object-oriented languages, makes possible that one selector can extend another.

```
.left{
  float: left;
  table{
    text-align: left;
  }
}
.menubar{
  @extend .left
}
```

Listing 18. Selector inheritance in Sass.

In listing 18, not only *.menubar* inherits the *float: left* property, but it would also inherit the *text-align: left* if there was a table inside that div.

4.9 Omniauth

On the Web, there are big amounts of data which can interconnect between themselves: for example, the usability of a site can be increased by the site taking the user's data from another application, like Facebook, instead of creating a register form of its own. For doing that, the site has to connect to an external API and get the data it needs.

However, as the Web has been developed by thousands of professionals, the APIs used to be very different from each other, so a developer had to study each instance before coding his connection to it. That was quite inconvenient, as most authentications have common points. To solve this issue, Omniauth was created.

Omniauth is an authentication framework. Its main goal is to let developers avoid the difficulties arising from using different authentication providers. Omniauth follows a general authentication philosophy: below there is a diagram describing the process. The figure includes a part of the process that changes depending on the API that Omniauth is trying to authenticate to: that changing part is called the strategy.

Right now, there are already more than 50 strategies developed; the most popular ones are Facebook and Twitter as most people already have accounts in them and they are, especially Facebook, the mainstays of 2.0. The strategies are not included in Omniauth by default, but they are easy to find and add to the main Omniauth process. Open-source collaborators develop more strategies constantly. [40]

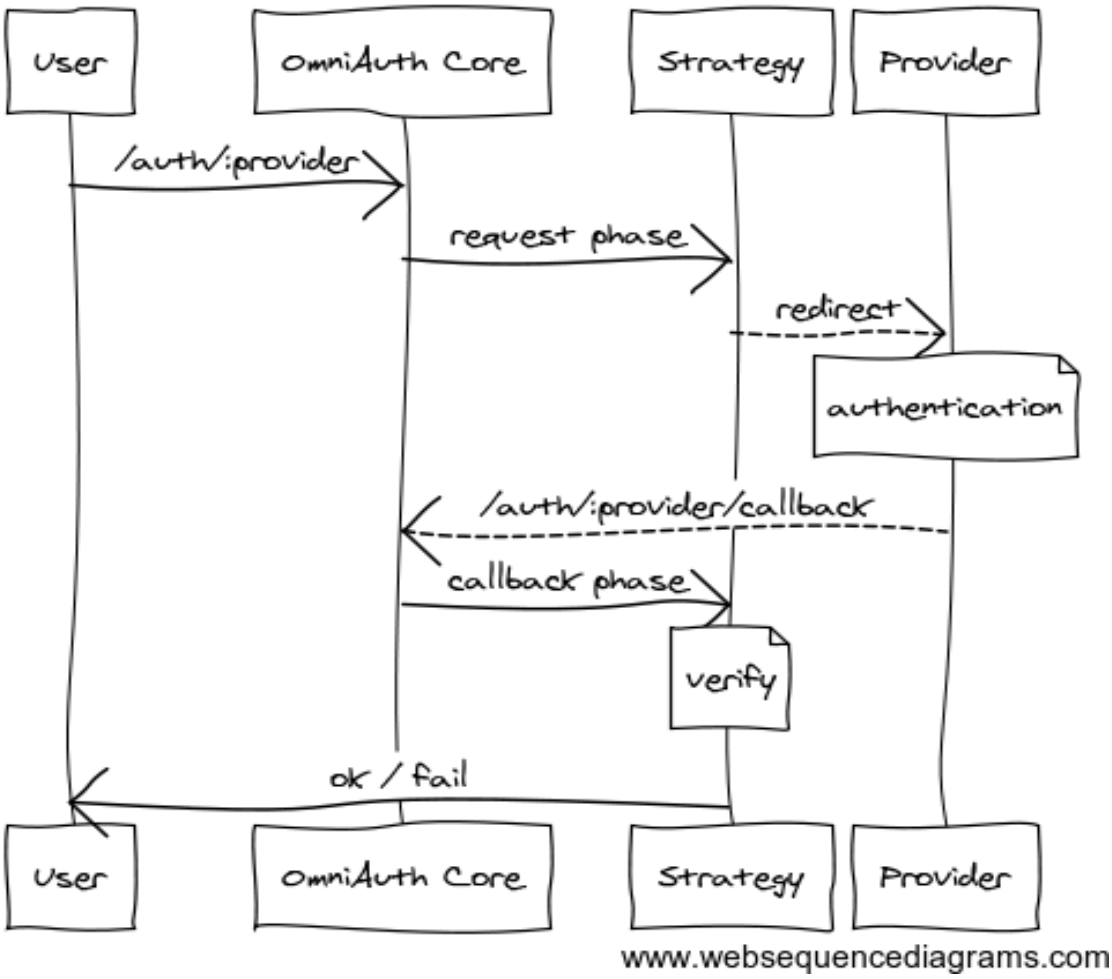

Figure 13. How Omniauth works [44]

The Omniauth authentication process is summarized in figure 13. In it, a user has to access to a given route; by default that is /auth/:provider (for example, for authenticating in Facebook, it would be www.myapp.com/auth/facebook). That route will activate Omniauth, passing the *:provider* as a variable. Then Omniauth starts the strategy for the given provider, or raises an error if the provider does not exist in the strategies list. The strategy redirects to the API. It is possible that if personal data is being requested, the API will ask the user to login. After this phase, the API sends data to the application, usually to a route specified before in the API configuration. There, Omniauth takes the data and sends it to the strategy algorithm that verifies if the format is correct: if it is right, the data is sent back to the application, to a route configured previously; if it is wrong, Omniauth raises an error. Each API returns very different data, from the path to the image avatar of a user (in Facebook or Twitter, for example) to a string with the companies where one person has worked (LinkedIn or XING). At this point, the developer has to check the information his application gets from the API, and in which format it has been sent to him, as he needs to access the data for his own purposes.

4.10 Rspec

Right now, in the Rails community, there are two extremely popular agile development techniques: Test Driven Development (TDD) and Behavior Driven Development (BDD). Both techniques are very similar: in fact, BDD is an evolution of TDD. Both of them will be described carefully in the chapter devoted to the development of TweetMovies.

For the moment, the idea can be reduced to this: modern developers need some powerful and flexible tools to test the application's features before deploying it. In Ruby, there are two very popular gems for doing that: Rspec for unit tests and Cucumber for acceptance tests.

Unit tests are methods that test individual pieces of code to determine if they are fit to use. Rspec creates an environment where developers can check the behaviour of only a function or object, supposing that everything else is working fine. Usually, an rspec test collection for a given application is divided in module and class types: controllers, models, helpers and even, if the developer thinks it is necessary, views. Then, every

module or class has its own suite of tests usually called *nameoftheclass_spec.rb*. It is like this because Rspec only loads automatically files finished in 'spec', as a way to identify its tests from all the other kind of files. A typical rspec file looks like listing 19.

```
require 'spec_helper'
describe Bet do
  describe "attributes" do
    it { should respond_to :price }
    it { should respond_to :multiple }
    it { should respond_to :game }
    it { should respond_to :past? }
  end

  describe "relations" do
    it { should respond_to :player }
    it { should respond_to :draw }
    it { should respond_to :shopping_cart }
  end

  describe "#from_club?" do
    context "if the bet player is a club" do
      it 'returns true' do
        subject.stub(:player).and_return
mock_model(Club)
        subject.should be_from_club
      end
    end
    context "if the bet player is a user" do
      it 'returns false' do
        subject.stub(:player).and_return
mock_model(User)
        subject.should_not be_from_club
      end
    end
  end
end
```

Listing 19. A typical rspec suite of tests.

First, *spec_helper* is included. It is a file that gives access to all the special methods of Rspec. However, the division of the test made by Rspec is more important: in the example, the next features of the class 'Bet' are being tested:

- Attributes: Usually, the test only checks that the model responds to the given attribute; in other words, this happens if an object of the class Bet has an attribute called price.
- Relations: This is quite similar to attributes. The test checks if the given object has a relation with another object by a given name; in this case, an object of the class bet should have a relation called player.
- Methods: This refers to the core of the test. In this part each and every method of the code is checked. Each method will have a different test for all of the actions that the method has to perform. For example, if in a method the content of two variables is changed in a method, two different tests will be needed, each one checking the change of one variable. Like this, it is really easy and quick to see what exactly makes a piece of code fail.

Another important part of Rspec is the methods it gives to the developers; the most interesting ones being *stub* and *mock*. As it has been stated before, a unit test has to check the correct behaviour of one individual object supposing that everything else works fine. The problem is that a lot of times it cannot assure that some other module is going to work properly: sometimes it is possible that the module still does not even exist. So what should developers do? Should they wait until they create the other modules to test the code? However, even like this, how can they know if they made an error that is going to change the whole application?

For this, Mock and Stub exist. Mock creates a phantom object of a given class, one that does not need to validate; for example, *mock(Bet)* creates a copy of a Bet object. Mock is a good way to avoid typical problems when creating objects for testing. The other method that is constantly used in Rspec is *stub*. Stub fakes the result of a call inside of an object: for example, *bet.stub(:paid?).and_return(true)* causes that the object bet always returns true when the method *paid?* is called.

4.11 Cucumber

Cucumber is, with Rspec, the most used testing tool in the Ruby on Rails scene. A typical test using Cucumber looks like the one in *Listing 20*:

```
#user_manages_tweets.feature
Feature: User manages tweets
  In order to manage my tweets
  As a user
  I must be able to create and delete them

  Background:
    Given I am logged in
    And There are already some movies in the system

  Scenario: User creates a new tweet
    When I write a new tweet about a movie
    Then the tweet should appear in the movie page

  Scenario: User deletes a tweet
    Given I have already written a tweet about the
movie
    When I delete it
    Then the tweet should not appear anymore

  Scenario: User likes a tweet
    Given There is a tweet I like
    When I express my love for it
    Then the score of the tweet must have improved
```

Listing 20. A typical cucumber suite of tests.

Under the Feature tag the general purpose of the test is described. In *Background*, the situation of the application and the user are described when the tests are executed. In addition, in each scenario there are three kinds of tags: *Given* (the situation before executing), *When* (the action performed) and *Then* (what should have happened).

Cucumber tests are extremely verbose and are designed to be read not only by developers but also by customers. Some agile methodologies state that this kind of test should replace specifications: in fact, Cucumber is the main application related to the BDD development process.

5. TweetMovies

So far, this thesis has described an advanced web developing technology (Ruby and all the software around it), analyzed the concept of Web 2.0 and the changes that it has brought to society, and investigated the movie world and its applications on the Net. Now the concept behind the application TweetMovies, is going to be explained. The thesis also explains how it is going to work, how the author of this thesis developed it and what is expected out of it when it is available to the public.

5.1 The concept

Cinema is the art the 20th century; it is the first art to be immediately popular. After the invention of the video and the expansion of movies beyond cinema screens, it is becoming more and more a social art. To ask about movie tastes is considered a polite introductory question; it is a bit like asking about work. In this changing era, it is obvious that people are demanding for tools to be able to socialize.

After the existing movie applications analysis, and after lots of years of using them, some failures and some free spaces appear. The author of this thesis thinks that most users ask themselves: what would I like to have? How would I like to socialize around my movie tastes? What are the social trends that still have not been covered by the Web applications? TweetMovies tries to be an answer to these questions.

As implied in its name, the main concept of TweetMovies is the tweet, a bit like "affinity" in FilmAffinity. This short way of communication (140 characters maximum) has changed radically the ways of communication, especially between young people. Despite their small size, tweets can be very different between themselves: they can talk about any kind of topics and be serious or funny, deep or foolish. The main property of a tweet is probably its short life. Good and interesting tweets can be retweeted but they are usually forgotten in a few minutes. However, is that correct? Should users not keep the tweets that they really like in the same way they put some blogs or blog entries in the markers of their browsers?

The main feature of TweetMovies is to be able to communicate an opinion about movies in a short format and to read the opinions of other users, organized and classified by the importance that the community think that they have. The short message format has already been explored by the movie world: lots of movie posters include a short comment about the film. With the speed of our contemporary society it is probable that this format should be more popular. It is the responsibility of web developers to create platforms for these possibilities to happen.

5.2 Main Features

The features that TweetMovies should have are, in order of importance:

1. Users should be able to read and create tweets about movies.
2. To expand quickly the use of the application, users should be able to resend tweets to their social networks (mainly, Twitter and Facebook).
3. As in any social network, users may have relations with other users, in this case, friends. Being friends with a user will change the behaviour of the application, giving easier access to the data of the friend user.

5.3 Implementation

The implementation of TweetMovies has its own peculiarities. The process is not going to be described step by step in this thesis, but it is important to explain the main differences between a project developed following the rules of the social coding and one with more regular rules.

5.3.1 Development methodology

During the coding, a development method called Behavior-Driven Development (BDD) was used. BDD is an evolution of Test-Driven Development (TDD). In this chapter these methodologies are described; some of the incidents that the author of this thesis found during its use are discussed too.

5.3.1.1 Test-Driven Development (TDD)

Test-Driven Development is one of the most popular methodologies in the agile software development scene. The philosophy behind TDD can be described with a sentence that is popular in the Ruby on Rails community: ”All code is guilty until proven innocent”. This means that it is supposed that a piece of code is not working until it is proven that it does. The only way to prove that is passing a test.

Test-Driven Development has changed radically the usual workflow in IT projects: usually the main phases of the software are specification, design, implementation and testing. In TDD, testing comes before implementation; and in more extreme variations of it like BDD, the testing phase replaces the specification and design stages. Figure 14 shows a typical TDD cycle.

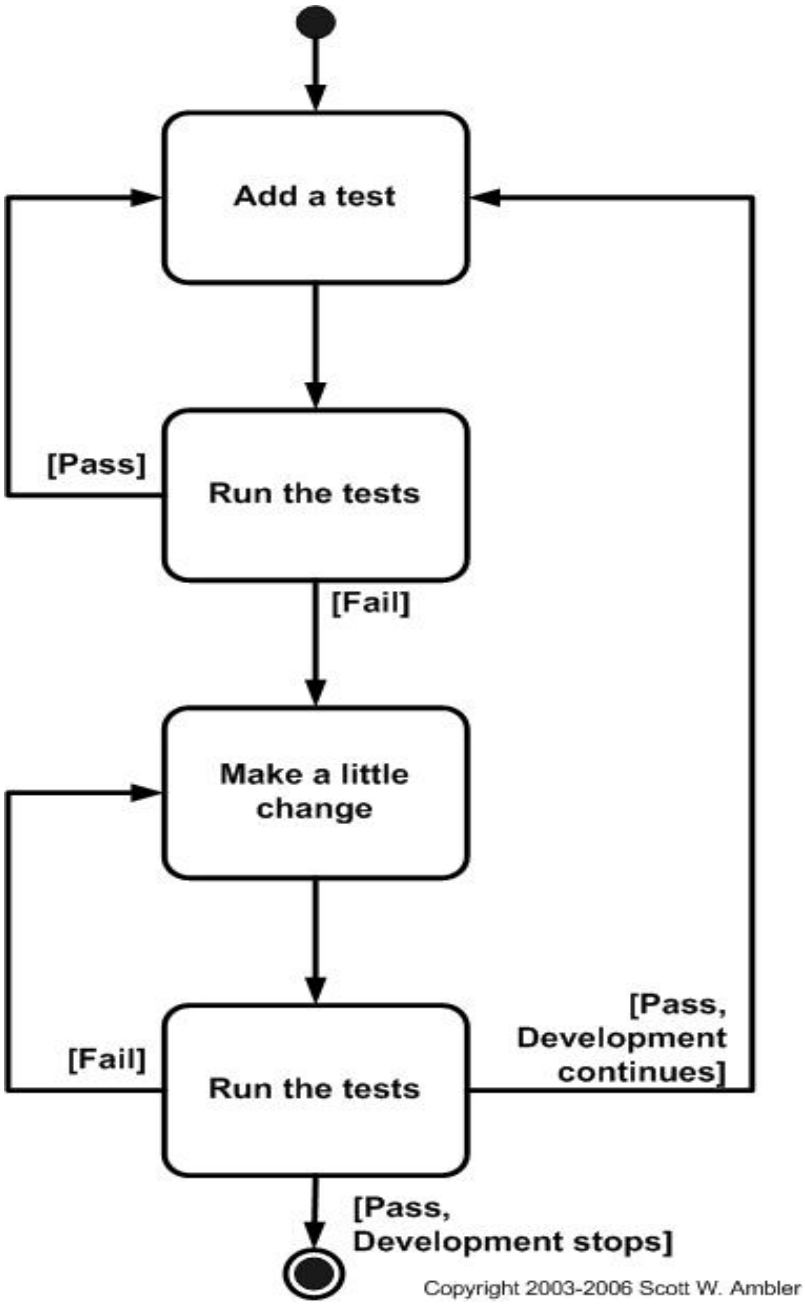

Figure 14. The Test-Driven Development Cycle. [45]

In a TDD process, first the test is created; then, the developer modifies the code of the application until it passes the code. In an advanced project, when a change is implemented into the code, all the tests are run, and not only the new one. Thus it is possible to detect quickly if a new piece of code is provoking errors in old features. It should also be considered that Test-Driven Development is only formed by unit tests.

5.3.1.2 Behavior-Driven Development (BDD)

Behavior-Driven development is the natural evolution of TDD. In this context, the word natural has a special meaning. BDD is a business-like, natural language user-approach to TDD.

This is achieved through some simple changes in the typical ways in which TDD tests are written:

- Unit test names are whole sentences starting with the word ”should”; or, at least, they must state clearly the objective.
- Unit tests are written in order of business importance.
- Acceptance tests are written using ”User stories”. The following is an example: "As a [role] I want [feature] so that [benefit]".
- Acceptance criteria are written in terms of scenarios and implemented as classes: Given [initial context], When [event occurs], Then [ensure some outcomes]. [46]

For most developers, the basic rules of BDD are the last ones: the main reason behind the success of this development technique is the revolution it implies in acceptance tests.

The most popular applications to implement TDD and BDD tests are Rspec and Cucumber: more information about how they work can be found in chapters 4.10 and 4.11.

Being a cyclical methodology, BDD can be explained in a simple way using diagrams. Figures 15 and 16 illustrate how BDD works:

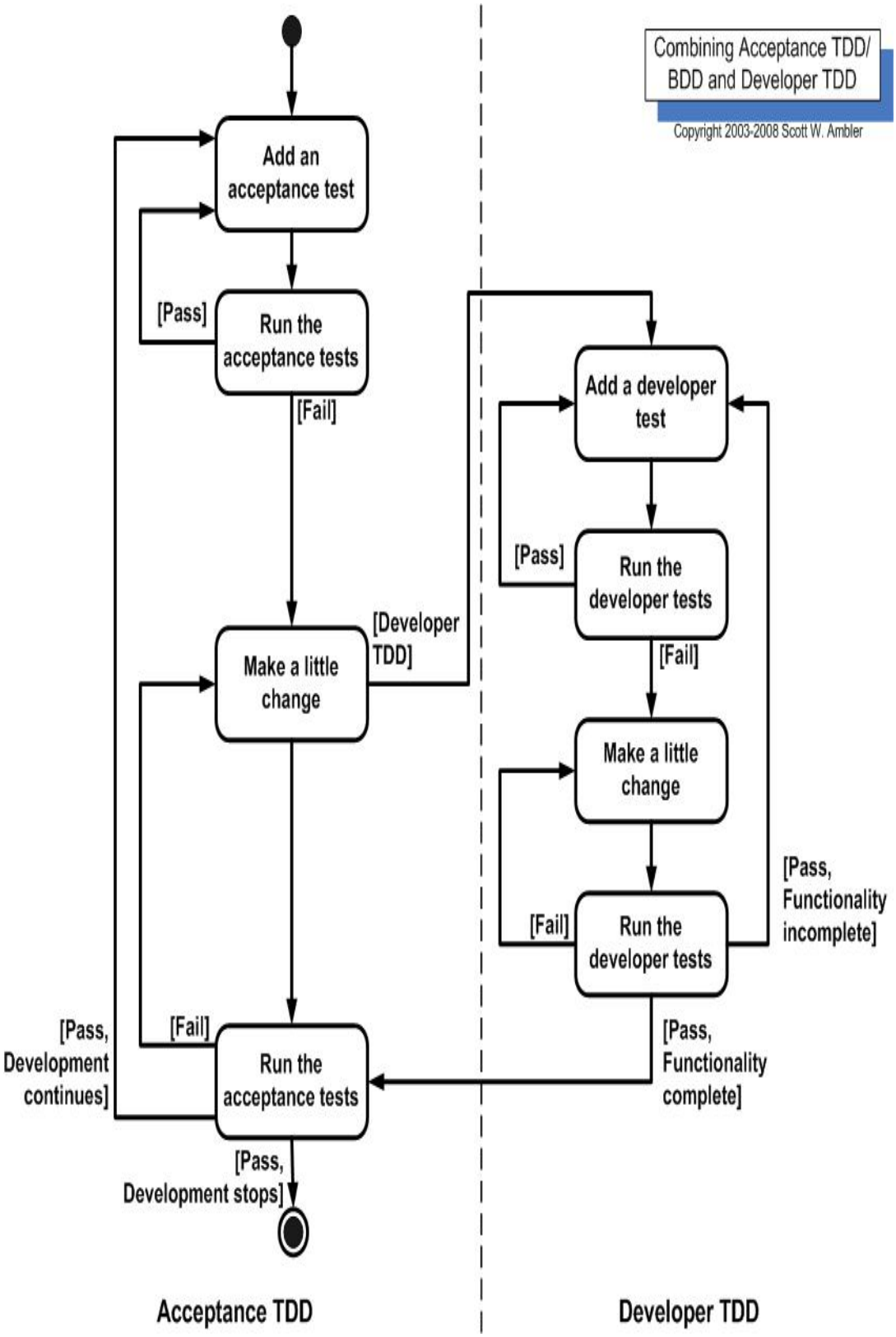

Figure 15. The Behavior-Driven Development cycle (2). [45]

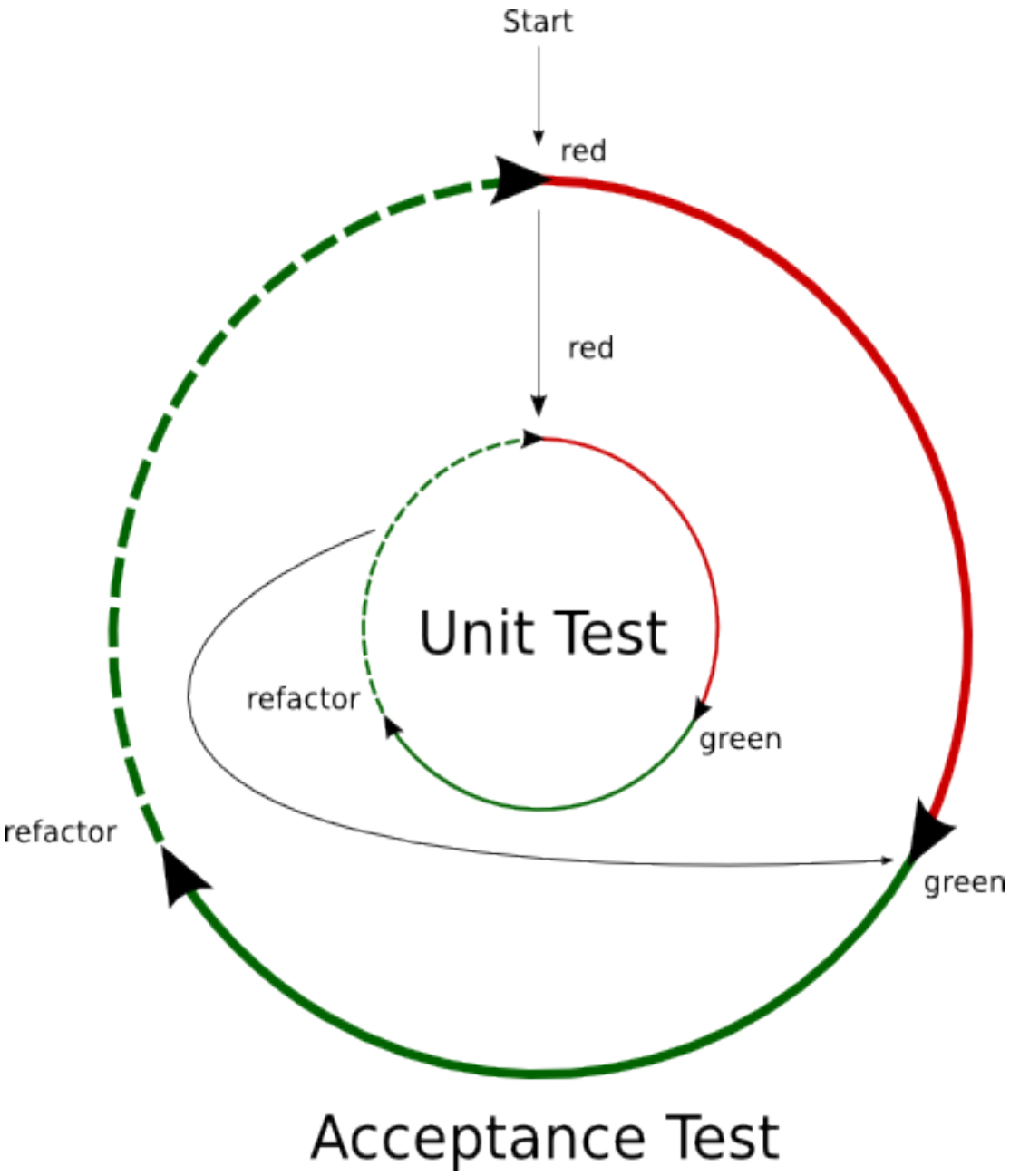

Figure 16. The Behavior-Driven Development cycle (1). [47]

As the figures above show, Behavior-Driven Development adds a behaviour dimension to the TDD cycle. By using integration and unit tests, BDD covers the entire scope of an application, helping to create more secure software from the developer and user point of view.

5.3.1.3 Incidents

TweetMovies was started as BDD, but at the end the author of this thesis just developed it normally. Further, there are some tests still missing. The main problem came with the acceptance tests created with Cucumber and Javascript. TweetMovies is full of JavaScript and JQuery, and testing it is slow and inaccurate.

5.3.2 Design

In this section the purpose behind the general design of TweetMovies is described through screenshots of the main sections of the application. With TweetMovies the author of this thesis tried to create a modern interface, with not much text except for the essential content of the web: the users' tweets. For all that, icons and Javascript/JQuery animations were used.

5.3.2.1 Menubar

In TweetMovies the main information has a lot of space: the user will interact with his/her own data only using the upper menubar.

The icons, when the user is not logged in, are:

- Home
- Search
- Login

The icons, when the user is logged in, are:

- Home
- Search
- Create movie
- Connected to Twitter
- Connect/Disconnect to Facebook
- My profile
- Logout

A typical state of the menubar can be seen in figure 17:

Figure 17. The menubar of TweetMovies. [48]

The buttons change their colour when the mouse is on them, making the interface more appealing. This effect was created using JQuery.

5.3.2.2 Index page

The index page of TweetMovies is organized in three main sections (the menubar and the footer are not counted, as they are present in all the application), as it can be seen in figure 18:

Figure 18. The index page of TweetMovies. [48]

The sections are described below:

- **Logo and slogan:** The logo is the main header on all the pages. The slogan is not only for advertising purposes: it tries to be a very short explanation of what the application is about and how it works.

- **Search:** Usually, the search bar will be accessible from the menubar, but in the index page it was decided to move it to the centre of the page. This made it easier to interact from a mobile phone. The default value inside the input ('Search movies!') explains the content of the web.

- **Trending movies and latest tweets:** The index will already show tweets about movies. The tweets will be the latest ones and the best tweet of a trending movie. Right now, as TweetMovies does not have much traffic, the trending movies are the ones with the most tweets; but it is expected that in the near future, when there are enough tweets created everyday, the trending movies will be the ones with more tweets from the last 24 hours.

5.3.2.3 Movie page

This page is the main space of the application. The users should spend most of their time there, as here the main interactions with the application will happen.

The structure of the movie page in TweetMovies can be seen in figure 19.

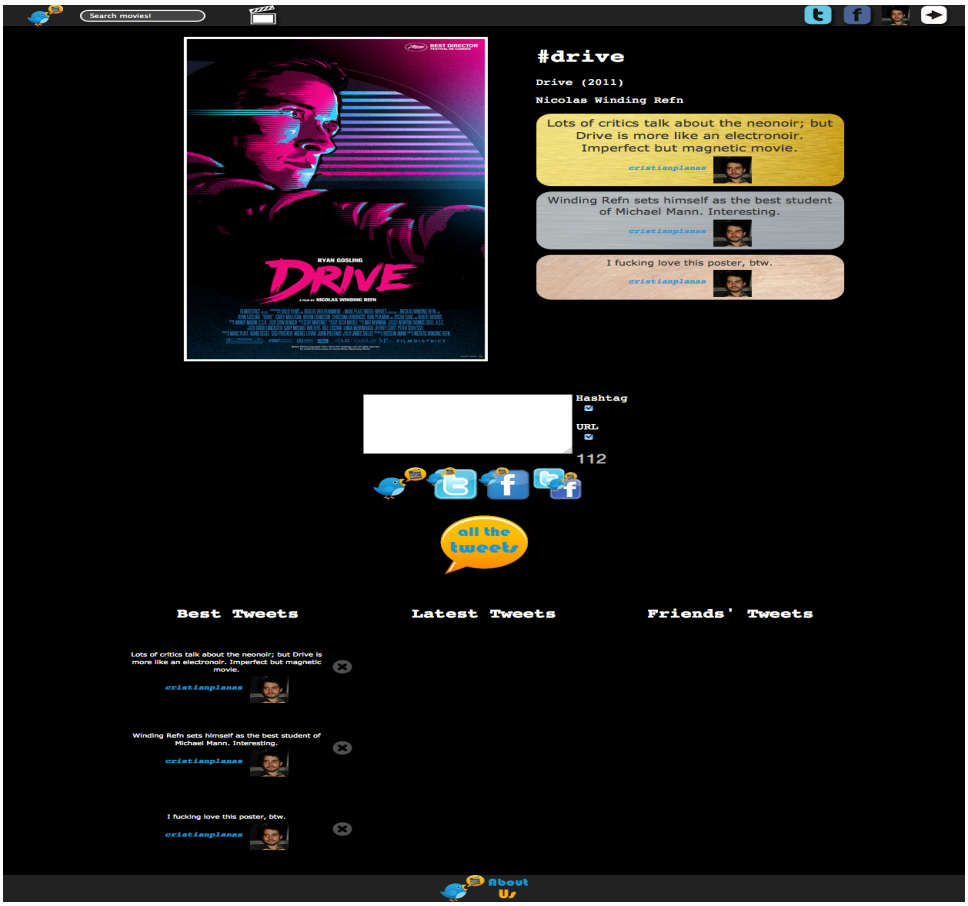

Figure 19. A movie page in TweetMovies. [49]

As it can be seen, the structure of the page is divided in these sections:

- **Movie information:** Some information is needed to identify the movie. The fields chosen include the title, the director, the year and the poster. There is a hashtag to identify the movie in Twitter, too.

- **Best tweets:** The top three tweets about the movie will show just next to the movie information, and with a bigger font size than the normal tweets. The top three tweets will be ordered in ”gold”, ”silver” and ”bronze” to introduce a competitive dimension in the tweeting.

- **Tweet form:** This is an input box for writing the tweet. Next to it, there are two checkboxes for automatically including the hashtag and the URL of the movie. This last option makes it easier for the Twitter friends of the user to vote for the tweet, and this also creates a viral effect. Then, regarding the

buttons, depending of the level of authentication, the user will be able to create the tweet only in TweetMovies and to resend it to Twitter and/or Facebook. There is a counter of the number of characters that disables the buttons when the tweet is longer than 140 characters. This protection is also triggered when the user adds the hashtag or the URL.

- **Tweets**: First, there is a link to all the tweets about this movie; then, there are three columns with different kind of tweets: best tweets, latest tweets and friend's tweets.

5.3.2.4 User profile page

The user profile page shows the main information about a given user. An example of it can be seen in figure 20.

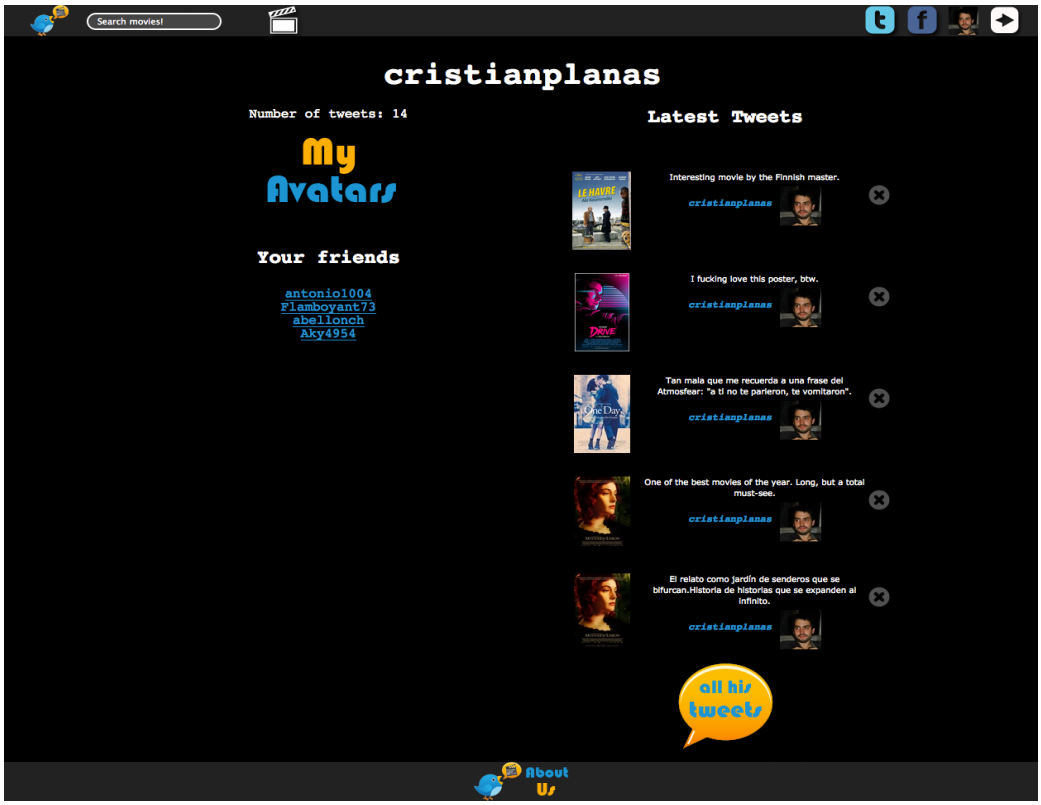

Figure 20. A user's profile page in TweetMovies. [50]

Figure 20 has the name of the user in the header, with an icon to add it as a TweetMovies friend next to it. Then it is roughly divided in two columns:

- **General information and options:** Right now, the data shown consist of the total number of tweets, the friends of the user and the option to choose the avatar between the Twitter one, the Facebook one, and none. In the future, it is expected that more options and information are added.

- **Tweets:** The five latest tweets appear in the right column. There is also a link to see all the tweets written by the user.

5.4 The future

Even if TweetMovies is right now an interesting application and even if it can be used by the public without any problems, it is still probably not prepared to go to the market and be successful.

The next chapters explain what will probably be the next steps in the development of TweetMovies.

5.4.1 Filling the database

For developing a successful movie application, one of the most basic needs is to have a complete movie database. This might not be like IMDB, but it should be big enough for most users.

Right now, a logged user can add the movie that he wants to tweet about if it does not exist already. However, users nowadays are lazy and like to be able to do whatever they want to do in the quickest way possible. Thus, in the near future this way of filling the database will be deprecated and a new source of movie data will be needed.

The objective is to solve this problem using the biggest movie database in the world: IMDB. Most of its data is public, and is offered in a plain text format: this way other

applications do not need a massive access to the IMDB website as that could hurt the web availability.

Using these plain text files a script will be developed using Ruby to take the needed data. It is assumed that IMDB is too complete for the purposes of TweetMovies, so only the information relative to the movies with more than 10,000 votes will be taken. Using different files, the following data of each movie will be needed: original title, international title, year and director; if it is available, the title translation to other languages will be taken too (mainly Spanish).

However, a part of the problem is left: this is the poster of each movie. In a 2.0 application, the graphic interface is very important, and having a high quality poster attached to its movie is essential. So far, the users upload a poster by themselves to the TweetMovies space in the Amazon S3 data storage service, but if the filling of most of the DB is going to be automatized, the same should be done with the images. In this situation, there are two options: to automatize the uploading or to use references to images on other sites.

Fortunately, there already exists a web service that provides users with high quality images of almost any movie: IMPAwards (Internet Movie Posters Awards). IMPAwards does not have an API, but an open-source developer has already coded a solution and packed it like a gem. [51]

In this way, taking profit of already exiting DBs and using open-source code, it is possible to have a complete database for TweetMovies.

5.4.2 Datamining Twitter with an AI algorithm

In the creation of TweetMovies, there were two options: the first one, the one that was chosen for the application, was for TweetMovies to be a platform for writing and reading about movies. The second one was for TweetMovies to be mainly an analyser of the data Twitter generates about movies. In fact, this second project is growing right now, created by other developers, under the name “MovieTweets”.

Users tend to be shy when using a new application, and achieving success is much easier when the site gets to a critical mass of users. In the near future, it is possible that a very simple AI algorithm is developed for filling the database with tweets of real users in Twitter. For that, some movies that would easily attract a lot of traffic will be chosen (for example: Inception, Harry Potter, or Twilight); then, the algorithm will take tweets with the hashtag #moviename using the Twitter API. As a result, the algorithm would filter and evaluate the tweets using a dictionary of movie-related words.

5.4.3 Improve using social information

One feature that will be added soon to TweetMovies is the option to automatically add Twitter friends to the TweetMovies list. It is possible to create an option for Facebook friends too, but that is technologically more complex as TweetMovies only uses the Twitter account to do the login.

In the future, it will also be possible to generate more useful information, like movie recommendations, datamining the database of tweets when it is big enough.

5.4.4 Launching the application to the market

When the author of TweetMovies considers that it is really complete, a commercial launch of the application will be prepared. The most important resource of TweetMovies will be their users and the information that they will create; so a quantity of users big enough is needed to be successful. There are some options to attract users:

- **Presenting the application on message boards:** TweetMovies is a 2.0 application, and because of that and the social tools it has, it can benefit of social networks. TweetMovies will be present in Twitter and Facebook from its very beginning, so the application will generate advertising totally for free.

- **Buying advertising space:** There already are some web applications which enable movie lovers to talk and learn about their favourite films; the author of

this thesis thinks that it could be really useful to put some advertising there: IMDB in particular. It is a better option than FilmAffinity as it has more visits and also the language of its users is mainly English. It will be important to have, at the beginning, an English-speaking audience, as it will be much easier to expand from here than from a Spanish-speaking (for example) public.

The author of this thesis is also interested in allying with a company with more experience: some medium-sized Ruby companies or even IMDB could give the business approach that TweetMovies is lacking.

6. Conclusion

In this thesis, two spaces have been described and analysed: first, the movie applications on the Internet and, secondly, some of the most powerful tools for web application development. With these two steps, two things have been demonstrated: first, that the market lacks a 2.0 application around the movie world; and second, that there already exist means for creating the application, even with a small group of developers.

After that, TweetMovies, the movie social network, was created and described; not only the final result and its features were described but also the development process.

The main conclusion of this thesis is that, as the TweetMovies project illustrates, agile methodologies let developers to create reliable software that can create a business, that the open-source way is fully working even in a company-like scenario, and that 2.0 is a change of social paradigm that still has a lot of possibilities.

The author of this thesis expects that this work can show the big opportunities that still exist on the Web and hopes that this project can inspire others to do the same.

References

1. Rosenbaum Jonathan. Goodbye cinema, hello cinephilia. University of Chicago Press; 2010.

2. Hauben Ronda. From the ARPANET to the Internet – A study of the ARPANET TCP/IP digest and of the role of online communication in the transition from the ARPANET to the Internet [online].
   URL: http://www.columbia.edu/~rh120/other/tcpdigest_paper.txt.
   Accessed on 12th February, 2012.

3. Kessler Gary C. An overview of TCP/IP Protocols and the Internet [online]. 9 November 2010.
   URL: http://www.garykessler.net/library/tcpip.html.
   Accessed on 12th February, 2012.

4. Scientific and Advanced-Technology Act of 1992, 42 U.S.C. § 1862 [online].
   URL: http://www.law.cornell.edu/uscode/text/42/1862i.
   Accessed on 12th February, 2012.

5. File:WIntHosts1981-July2011.jpg [online].
   URL: http://en.wikipedia.org/wiki/File:WIntHosts1981-July2011.jpg.
   Accessed on 12th February, 2012.

6. Internet Systems Consortium. Internet host count history – Number of Internet hosts [online].
   URL: http://www.isc.org/solutions/survey/history.
   Accessed on 12th February, 2012.

7. O'Reilly Tim, Battelle John. Opening welcome: State of the Internet industry [online]. October 5, 2004.
   URL: http://itc.conversationsnetwork.org/shows/detail270.html.
   Accessed on 12th February, 2012

8. O'Reilly Tim. What is Web 2.0? Design patterns and business models for the next generation of software [online].
URL: http://oreilly.com/web2/archive/what-is-web-20.html.
Accessed on 12th February, 2012.

9. Shuen Amy. Web 2.0: A strategy guide. Business thinking and strategies behind successful Web 2.0 implementations. O'Reilly Media; 2008.

10. O'Reilly Tim. Web 2.0 compact definition: trying again [online].
URL: http://radar.oreilly.com/2006/12/web-20-compact-definition-tryi.html.
Accessed on 12th February, 2012.

11. Johnson Robert. Scaling Facebook to 500 million users and beyond [online]. July 21, 2010.
URL: https://www.facebook.com/note.php?note_id=409881258919.
Accessed on 13th February, 2012.

12. Huang Carl. Facebook and Twitter key to Arab Spring uprisings: report [online]. The National; June 6, 2011.
URL: http://www.thenational.ae/news/uae-news/facebook-and-twitter-key-to-arab-spring-uprisings-report.
Accessed on 25th January, 2012.

13. Gardner Frank. Tunisia one year on: where the Arab Spring started [online]. BBC News Africa; December 17, 2011.
URL: http://www.bbc.co.uk/news/world-africa-16230190.
Accessed on 25th January, 2012.

14. Saletan William. Springtime for Twitter. Is the Internet driving the revolutions of the Arab Spring? [online]. Slate; July 18, 2011.
URL:http://www.slate.com/articles/technology/future_tense/2011/07/springtime_for_twitter.html.
Accessed on 25th January, 2012.

15. Casey Tina. Study: Twitter played pivotal role in Arab Spring [online]. TPM; TPMIdeaLab; September 23, 2011.
URL: http://idealab.talkingpointsmemo.com/2011/09/study-twitter-played-pivotal-role-in-arab-spring.php.
Accessed on 25th January, 2012.

16. Kim Peter. Analysis of a wiki of social media marketing examples [online]. Being Peter Kim; March 24, 2009.
URL: http://www.beingpeterkim.com/2009/03/smm-wiki-analysis.html.
Accessed on 13th February, 2012.

17. Quinion Michael. Long tail. World wide worlds [online].
URL: http://www.worldwidewords.org/turnsofphrase/tp-lon1.htm.
Accessed on 14th February, 2012.

18. Anderson Chris. About me [online].
URL: http://www.longtail.com/about.html.
Accessed on 14th February, 2012.

19. Wilson Fred. My favorite business model [online].
URL: http://avc.blogs.com/a_vc/2006/03/my_favorite_bus.html.
Accessed on 14th February, 2012.

20. Bailyn Evan, Bailyn Bradley. Outsmarting Google: SEO secrets to winning new business. Que Publishing; 2011.

21. Brin Sergey, Page Lawrence. The anatomy of a large-scale hypertextual web search engine [online]. 1998.
URL: http://infolab.stanford.edu/pub/papers/google.pdf.
Accessed on 15th February, 2012.

22. File:PageRanks-Example.svg [online].
URL: http://en.wikipedia.org/wiki/File:PageRanks-Example.svg.
Accessed on 10th December, 2012.

23. IMDB [online].
URL: http://imdb.com.
Accessed on 7th March, 2012.

24. The Artist [online]. IMDB.
URL: http://www.imdb.com/title/tt1655442/.
Accessed on 7th March, 2012.

25. Quiénes somos [online]. FilmAffinity.
URL: http://www.filmaffinity.com/es/meetus.php.
Accessed on 7th March, 2012.

26. FilmAffinity [online].
URL: http://www.filmaffinity.com/en/.
Accessed on 7th March, 2012.

27. El foro que surgió de FilmAffinity [online].
URL: http://filmaffinity.mforos.com.
Accessed on 7th March, 2012.

28. Venners Bill. The philosophy of Ruby. A conversation with Yukihiro Matsumoto, part I [online]. September 29, 2003.
URL: http://www.artima.com/intv/ruby4.html.
Accessed on 25th February, 2012.

29. Olsen Russ. Eloquent Ruby. Addison-Wesley Professional Ruby Series; 2011.

30. Paolo Perrotta. Metaprogramming Ruby: program like the Ruby rros. Pragmatic Bookshelf; 2010.

31. Sosinski Robert. Understanding Ruby blocks, procs and lambdas [online]. December 21, 2008.

URL: http://www.robertsosinski.com/2008/12/21/understanding-ruby-blocks-procs-and-lambdas/.
Accessed on 15th February, 2012.

32. Bullock Ben. Google trends [online]. July 1, 2011.
URL: http://blogs.perl.org/users/ben_bullock.
Accessed on 15th February, 2012.

33. Rails core team [online]. Rubyonrails.org.
URL: http://rubyonrails.org/core.
Accessed on 7th March, 2012.

34. Ruby – Niwatori – Picasa web albums [online].
URL: https://picasaweb.google.com/Dikiwinky/Ruby.
Accessed on 7th March, 2012.

35. Venners Bill. Orthogonality and the DRY principle. A conversation with Andy Hunt and Dave Thomas, part II [online].
URL: http://www.artima.com/intv/dry.html.
Accessed on 26th February, 2012.

36. Fowler Martin. Patterns of enterprise application architecture. Addison-Wesley Professional; 2002.

37. RubyGems. Your community gem host [online].
URL: https://rubygems.org/.
Accessed on 7th March, 2012.

38. Devise repository in GitHub [online].
URL: https://github.com/plataformatec/devise.
Accessed on 7th March, 2012.

39. EasyRegexp repository in GitHub [online].
URL: https://github.com/Gawyn/easyregexp.

Accessed on 7th March, 2012.

40. Omniauth repository in GitHub [online].
    URL: https://github.com/intridea/omniauth.
    Accessed on 7th March, 2012.

41. Driessen Vincent. A successful Git branching model [online].
    URL: http://nvie.com/posts/a-successful-git-branching-model/.
    Accessed on 20th January, 2012.

42. Devise repository and README in GitHub [online].
    URL: https://github.com/plataformatec/devise.
    Accessed on 7th March, 2012.

43. Haml [online].
    URL: http://haml-lang.com/.
    Accessed on 7th March, 2012.

44. Dumitrascu Irina. OmniAuth strategy for everything else [online]. November 30, 2010.
    URL: http://dira.ro/2010/11/30/omniauth-strategy-for-everything-else.
    Accessed on 12th February, 2012.

45. Ambler Scott W. Introduction to test driven development (TDD) [online].
    URL: http://www.agiledata.org/essays/tdd.html.
    Accessed on 8th February, 2012.

46. North Dan. Introducing BDD [online]. March, 2006.
    URL: http://dannorth.net/introducing-bdd/.
    Accessed on 8th February, 2012.

47. Chelimsky David, Astels Dave, Dennis Zach, Hellesøy Aslak, Helmkamp Bryan, North Dan. The RSpec book: behaviour-driven development with RSpec, Cucumber and friends. Pragmatic Bookshelf; 2010.

48. Tweet Movies – Use Twitter and Facebook to discover cinema [online].
URL: http://tweetmovies.net/.
Accessed on 30th January, 2012.

49. Drive (2011) – Tweet Movies [online].
URL: http://tweetmovies.net/movies/drive.
Accessed on 30th January, 2012.

50. Tweet Movies – Use Twitter and Facebook to discover cinema [online].
URL: http://tweetmovies.net/users/10.
Accessed on 30th January, 2012.

51. Maddox/impawards – GitHub. Simple library to find high quality posters for movies [online].
URL: https://github.com/maddox/impawards.
Accessed on 30th January, 2012.