Ingen titel än

Fredrik Jonsén
Linköping University
Linköping, Sweden
frejo105@student.liu.se

Alexander Stolpe
Linköping University
Norrköping, Sweden
alest170@student.liu.se

ABSTRACT
Coming later

INTRODUCTION
Today a lot of people are using social media in one way or the other and it is estimated that there will be around 2.67 billion social media users around the globe by 2018[6]. Most of these social networks have released Application Programming Interfaces (APIs) which developers wish to integrate these networks into their software.

Motivation
Social network usage is growing and has gone from 0.97 billion users in 2010 to 2.14 billion in 2015.[6]. This would account for approximately 29% of the earth's population in 2015, which was 7.347 billion 2015[9]. It is worth noting that this counts created user accounts and not unique users, one person can have multiple accounts over multiple networks, and accounts may not belong to an actual person, but rather companies or bots.

Because of the currently high, and still growing, number of social media users we find it highly likely that we will see an increasing number of applications that involve social media in their software in one way or the other. Out of all the social networks existing today there are twenty that have more than 100 million active accounts[7]. This means that if one would want to create an application that involves a lot of social networks we will have to do a lot of work just to implement all of these into our system.

Purpose
Because of this we see a need for a way to combine these social network APIs in some way to save development time and reduce the amount of duplicate code written in software. The purpose for this project is to create a library that combines the APIs in a modular fashion, where each API serves as a module in order to simplify adding new networks, and make it easier to involve social media in software.

A software library by definition is a set of pre-written code that a developer might add to a project in order to add more functionality or to ease the development process[8].

RESEARCH QUESTION
Is it possible to create a maintainable modular library for Social Network APIs?

Modularity
For the purpose of our study, we define modularity as the extent to which a program can be divided into modules, where[**Kiczales:2005:APM:1062455.1062482**]:

- Each module has a well-defined interface that describes how the system can interact with it.
- Changes to the underlying functionality of a module does not affect the modules interface.
- Modules can be put together with other modules in different ways to make a complete program

In our study modularity will mostly affect how easy it will be to integrate new social media APIs in the future.

LIMITATIONS
Because there are so many social networks existing today and we have a finite amount of time to complete this project we will focus on a smaller set of APIs to implement into our library. We have set up a few criteria for the APIs so we can find suitable candidates for our library, which are:

- The social network must be one of the 22 most popular[7].
- It must be relevant for our geographical location, in this case Europe.

Other than this we will judge the API itself by how good its documentation is, its functionality and its ease-of-use.

THEORY
Code Reuse
A lot of time and money can be saved by reusing code, of which libraries are one form. Although there are some issues with using libraries, the gains from avoiding reinventing the wheel makes writing and using libraries a common practice, in particular in open source software[2].

Writing code which can be easily reused requires a deeper analysis of the problem domain, which may increase the cost and time required compared to developing the same code without reuse in mind, but can drastically decrease the cost of developing systems in the future where the code can be reused [4]. This cost reduction is apparent foremost in terms of direct cost of development, but also in time-to-market, which can be argued to be even more important in the long term[1].

With code reuse there are also several potential issues which have to be kept in mind, both when it comes to the implementation itself but also when it comes to using the implementation. Backwards compatibility between versions is a major topic in itself[5]. There is also the risk of a library being abandoned by it's maintainer. This is especially true for proprietary libraries, where the source code may not be available. It this case, the library might have to be replaced, making all the effort to use the library wasted.

Library Interface Design

When designing our library we will want keep several things in mind. We want to design the library so it's easy and straightforward to use for a developer. As Henning points out in his article[3], about design of APIs, it is very easy to create a bad one, but very hard to create an API that feels natural and easy to work with. APIs, as we know, are a kind of interface for a program to gain access to another program without direct access, and can be compared to the interface of our library.

Henning continues to discuss guidelines for how an API should generally be designed. What feels most relevant to our work is how he describes how the APIs should be designed from the perspective of a user, because when it's done from the implementer's point of view the needs of the user are often forgotten. It's usually best to document first, because when it is done after the implementation the programmer, who wrote the functionality, will usually just dictate what he did, rather than make it is obvious enough for others who are not as familiar with the code.

Social Media APIs

As mentioned, one of the biggest issues with using libraries is the risk of the project being abandoned. This risk increases significantly when the library itself uses APIs which integrate oft-changing platforms, such as the case of Social Media. The rate of change differs between networks, in some cases on average three times a year[1], in other cases several times a month[2], although the impact of the changes varies greatly. At best, a lack of active development simply means missing out on new functionality. In other cases, such as unfixed security vulnerabilities[3], may render the library unusable.

Similiar work

Several similar works, Agorava[4], ASNE[5] and SocialMedia Abstractions[6]serve the same or similar purposes, but all were abandoned before reaching a stable release. In the case of ASNE, the project was abandoned explicitly due to a lack of free time. This shows the issue of a library having only a single maintainer, as the project risks being abandoned by it's developer as soon the project is no longer a priority. The others have no stated reason for the lack of continued development. There are also commercial services[7] which provide this functionality for some popular APIs, but charge money to sign up and use. As it is proprietary it's inner workings are completely opaque, and thus will not be examined by this study.

Encountered issues

Despite being abandoned, we hope to still learn from the problems the similar libraries encountered and solved. For this, we looked into each project's issue tracking (where available) and commits. This was somewhat complicated in that one project, SocialMedia Abstractions, simply had not used issue tracking. Another project, ASNE did use issue tracking, but much of the discussion regarding individual issues was largely in Russian, making it unusable in our case. The issues, despite the name, did not always regard bugs. In the vast majority of cases, issued stemmed from users misunderstanding the library documentation, requesting features, or suggesting refactoring of code to increase maintainability. In the case of documentation misunderstandings, these were often solved by simply adding examples. There was also a noticeable difference in the amount of issues pertaining bugs in ASNE, which included no automatic tests, compared to Agorava, which includes a large amount of automatic tests, and had almost no issues regarding logical errors, despite having a much larger code base.

[1]https://developers.facebook.com/docs/apps/changelog
[2]https://www.hitchhq.com/twitter/activities
[3]https://github.com/gorbin/ASNE/issues/107
[4]http://www.agorava.org/
[5]https://github.com/gorbin/ASNE
[6]https://github.com/socialsensor/socialmedia-abstractions
[7]https://cloudrail.com

REFERENCES

1. Griss, Martin L. Software reuse: from library to factory. *IBM systems journal*, 32(4), 1993: 548–566.
2. Haefliger, Stefan, Krogh, Georg von, and Spaeth, Sebastian. Code reuse in open source software. *Management Science*, 54(1), 2008: 180–193.
3. Henning, Michi. Api design matters. *Queue*, 5(4), May 2007: 24–36.
4. Lim, Wayne C. Effects of reuse on quality, productivity, and economics. *IEEE software*, 11(5), 1994: 23–30.
5. Raemaekers, Steven, Deursen, Arie van, and Visser, Joost. Measuring software library stability through historical version analysis. *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. 2012, 378–387.
6. Statista. Number of social network users worldwide from 2010 to 2020 (in billions). Address: `https://www.statista.com/statistics/278414/number-of-worldwide-social-network-users`.
7. Statista. Leading social networks worldwide as of january 2017, ranked by number of active users (in millions). Address: `https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/`.
8. The Linux Documentation Project. Shared libraries. Address: `http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html`.
9. World Bank. Population, total. Address: `http://data.worldbank.org/indicator/SP.POP.TOTL`.