iOS Smart Mobile

TECHNICAL RESEARCH

##### Emil and Tudor

Application building

horizontal line

# Placeholder image

# 

Table of Contents

[**Introduction**](#_849cejguhnab) **3**

[**API connections**](#_feibjpiovq56) **4**

[Hypothesis 1:](#_dg9bmppmfodl) 4

[**Version control system operation**](#_842kxd6ygsud) **6**

[Hypothesis 1:](#_qu6t06mp4c9h) 6

[Hypothesis 2:](#_dtg5yimaz7tq) 7

[**UX means a user friendly and intuitive application**](#_skdogdzh8kbp) **9**

[Hypothesis 1:](#_x0v2a33wcv0i) 9

# 

### Introduction

This research paper will show, what are the first steps of a programmer to create a application and some other components with which an application can’t go without. One of the first steps is the design component. The users musn’t struggle with the GUI of the application. In order to complete this step a good programmer can use different APIs to make their application more interactive in short time or even help to develop the back-end. Programmers often work in teams, so they need to work without interruptions and technical difficulties, thats why version control systems are developed.

### API connections

#### Hypothesis 1:

API usage simplifies app developing.

The purpose of an API as a GUI is that it makes it easier for people to use programs, and it also makes it easier for developers to use certain technologies in app building. “By abstracting the underlying implementation and only exposing objects or actions the developer needs, an API simplifies programming.”

A good example is a GUI for an email client that provides the user with a button which does all the steps from highlighting new emails to fetching them, a file input/output API can give the developer a function that lets him copy a file from location Y to location X without requiring that the developer to understand the file system operations occurring behind the scenes.

The design of an API has significant impact on its usage. The hiding information principle describes the role of interfaces as enabling modular programming by hiding the implementation details of the modules. Because of this module users do not need to understand the complexities inside them. APIs are designed to only provide tools that a user would expect. The design of interfaces is an important part of software architecture and organization of a piece of software that is complex.

API classification based on their usage:

1. Internal/Private

These APIs are only used in within companies or organizations. Even though RESTful APIs usage is growing strong, some of the old ones like .NET or SOAP/HTTP still have a big presence.

2. External/Public

These APIs are available to the public. At this stage of maturity, most of the external APIs are written based on REST/JSON technologies. “They provide access and integration capabilities that are easier to use than the more industrial-strength capabilities leveraging web services.”

3. Partner

These APIs are designed for partners only. They are made in order for them to be able to access business functions in relation to the business relationship of the partnership. Some examples can be online catalog, reconciliation and ordering.

There are many types of APIs. One of the most common types of APIs are Web APIs,they provide an interface for web applications, or applications that need to connect to each other via the Internet to communicate. There a lot of public APIs that can be used to do everything from checking traffic or weather, to updating your social media status, or make payments.

There are a lot more private Web APIs. These APIs are not available to the general public. They are used by companies to extend their services and capabilities across a broad range of use cases.

<https://www.mulesoft.com/resources/api/types-of-apis>

<https://en.wikipedia.org/wiki/Application_programming_interface>

<https://developer.ibm.com/apiconnect/documentation/api-101/types-apis/>

### Version control system operation

#### Hypothesis 1:

We need to use a version control system to handle changes which occur and to organize our work.

horizontal line

References:

<https://en.wikipedia.org/wiki/Version_control>

<https://en.wikipedia.org/wiki/Git>

<https://en.wikipedia.org/wiki/Apache_Subversion>

<https://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows>

<http://www.differencebetween.net/technology/software-technology/difference-between-git-and-svn/>

horizontal line

A component of [software configuration management](https://en.wikipedia.org/wiki/Software_configuration_management), version control, also known as revision control or source control, is the management of changes to documents, [computer programs](https://en.wikipedia.org/wiki/Computer_program), large web sites, and other collections of information. Changes are usually identified by a number or letter code, termed the "revision number", "revision level", or simply "revision". For example, an initial set of files is "revision 1". When the first change is made, the resulting set is "revision 2", and so on. Each revision is associated with a [timestamp](https://en.wikipedia.org/wiki/Timestamp) and the person making the change. Revisions can be compared, restored, and with some types of files, merged.

The need for a logical way to organize and control revisions has existed for almost as long as [writing](https://en.wikipedia.org/wiki/Writing) has existed, but revision control became much more important, and complicated when the era of computing began. The numbering of [book editions](https://en.wikipedia.org/wiki/Edition_(book)) and of [specification revisions](https://en.wikipedia.org/wiki/Specification_(technical_standard)) are examples that date back to the print-only era. Today, the most capable (as well as complex) revision control systems are those used in [software development](https://en.wikipedia.org/wiki/Software_development), where a team of people may change the same files.

It is know for the programmers that there will be always changes to the source code of an application. Therefore the programmers need control of their application’s code. To revert changes or demonstrate progress in the form of a commit. There are different types of version control systems like git and svn.

The version control systems answers all of these questions, which a normal drop box can not or at least cannot control.

* Made a change to code, realised it was a mistake and wanted to revert back?
* Lost code or had a backup that was too old?
* Had to maintain multiple versions of a product?
* Wanted to see the difference between two (or more) versions of your code?
* Wanted to prove that a particular change broke or fixed a piece of code?
* Wanted to review the history of some code?
* Wanted to submit a change to someone else's code?
* Wanted to share your code, or let other people work on your code?
* Wanted to see how much work is being done, and where, when and by whom?
* Wanted to experiment with a new feature without interfering with working code?

#### Hypothesis 2:

Subversion diversity comparison between SVN and Git

horizontal line

References:

<https://en.wikipedia.org/wiki/List_of_version_control_software>

<https://en.wikipedia.org/wiki/Comparison_of_version_control_software>

horizontal line

**Repository model** describes the relationship between various copies of the source code repository in the data tree. In a **client–server model**, users access a master repository via a [client](https://en.wikipedia.org/wiki/Client_(computing)). Typically, their local machines hold only a working copy of a project tree. Changes in one working copy must be committed to the master repository before they are propagated to other users. In a **distributed model**, repositories act as peers, and users typically have a local repository with version history available, in addition to their working copies. In [software development](https://en.wikipedia.org/wiki/Software_development), distributed version control (also known as distributed revision control) is a form of [version control](https://en.wikipedia.org/wiki/Version_control) where the complete [codebase](https://en.wikipedia.org/wiki/Codebase) - including its full history - is mirrored on every developer's computer. This allows [branching](https://en.wikipedia.org/wiki/Branching_(version_control)) and [merging](https://en.wikipedia.org/wiki/Merge_(version_control)) to be managed automatically, increases speeds of most operations (except for pushing and pulling), improves the ability to work offline, and does not rely on a single location for backups.

Git - the distributed repository model

SVN - the client-server model

**Concurrency model** describes how changes to the working copy are managed to prevent simultaneous edits from causing nonsensical data in the repository. In a lock model, changes are disallowed until the user requests and receives an exclusive lock on the file from the master repository. In a merge model, users may freely edit files, but are informed of possible conflicts upon checking their changes into the repository, whereupon the version control system may merge changes on both sides, or let the user decide when conflicts arise. Distributed version control almost always implies a merge concurrency model.

Git - merge

SVN - merge or lock

These are the difference of some very popular source control applications - Git vs SVN:

|  |  |
| --- | --- |
| Git | SVN |
| distributed | client-server |
| centralized server and repository | stores files of content |
| easier branch system | Harder to manipulate branches |
| no global revision feature | global revision feature |
| better content protection every single item is encrypted | no encryption |
| distributed under GNU and is maintained | Apache Subversion, or SVN, is distributed under the open source license. |

1. Git is distributed under GNU, and its maintenance overseen by Junio Hamano; Apache Subversion, or SVN, is distributed under the open source license.

### UX means a user friendly and intuitive application

#### Hypothesis 1:

A programmer must be a user experience designer as well

horizontal line

References:

<https://hackernoon.com/ux-ui-design-for-mobile-apps-d4e079adce37>

horizontal line

A programmer must make an application design which must be understandable and easy to use. If a person requires to go from point A to B, there must be a maximum of 3 clicks to reach B. However, this isn’t the only job of an GUI developer, he must must create the application interactive in addition to understandable. This addition makes the process of creating an user experience design very hard. The person behind the the user interaction design must keep up with modern trends. Make test to simulate the user's behaviour and adapt or improve suggestion received from his testers. This whole information has to be processed and implemented.

The reality of every developer is that they’re completely immersed in the back end, but they’re never face-to-face with the end user. The user doesn’t interact directly with their work—the user interacts with what the designer creates.

This difference in perspective keeps developers from seeing the whole picture. They’re missing the most important part: the user.

Developers, learning design makes it possible for you to assemble the entire system in your mind—from how the user interacts with your product, down to the last link where you store the information in your database. This complete trajectory allows you to develop a product that’s better than the competition.

<https://www.invisionapp.com/blog/developers-should-learn-design/>