

# Mercurial review

---

## Software Architecture

**Sjors Otten (3558010)**  
**Kevin van Ingen (3561593)**  
**Mark Rouhof (3553507)**  
**Robert Vroon (3440516)**

**26-1-2011**

Mercurial is a Distributed Version Control System. This document describes a requirement change with small impact and a requirement with a large impact. In addition, this document will answer whether the Architectural overview of Mercurial (2011) is enough to define the impact of those requirements.

**Table of contents**

Introduction..... 2

Requirement with little impact ..... 3

    Relative differences between timestamps..... 3

    Networked time synchronization upon commit ..... 3

    Impact of the proposed change ..... 3

Requirement with large impact..... 4

    Impact of the proposed change ..... 5

Architectural description review of trade-off ..... 6

Conclusion ..... 7

References..... 8

## Introduction

Zhao et al define change impact analysis as *“Change impact analysis is the task through which programmers can assess the extent of the change, i.e. the software component that will impact the change or be impacted by the change. Change impact analysis provides techniques to address the problem by identifying the likely ripple effect of software changes and using this information to re-engineer the software system design.”* (Zhao, 2002) Within their research Zhao et al address change impact analysis on the architectural level of a system. *“To reduce the complexity of code-level change impact analysis and to allow a maintenance programmer to assess the effect of changes of the system at the architectural level so that software evolution actions can be made earlier.”* (Zhao, 2002) Their method works by finding backward, forward, and unified architectural slices as well some architectural chops of the specification by starting from a component and/or a connector of interest.

The method given by Zhao et al gives a clear process for change impact analysis but cannot be used due to time restrictions. Bohner gives a better phased process of change impact analysis as they identify two classes of impact analysis, traceability and dependency (Bohner, 1996). Within traceability links between requirements, specifications, design elements, and tests are captured, and these relationships can be analyzed to determine the scope of an initiating change. This review concentrates on the change impact of requirements within the architecture of Mercurial.

## Requirement with little impact

This section describes an use-case for the Mercurial versioning system that requires a small change in the architecture. The use-case is defined as follows’:

**“The ability to use concurrent timestamps for file meta data.”**

Mercurial uses system timestamps for file creation and modification. Whenever a user merges or its branch with another Mercurial instance the metadata of the files is committed as well. This could mean that whenever the system time differs between users the perceived time modified or created can be displayed incorrect. Difference between time zones is one of the most frequent causes of this problem. These *subjective timestamps* can be misleading for users that compare timestamps of file, and potentially make bad decisions.

In general this would not be a problem occurring by all Mercurial users. Linux users that are connected to the Internet would probably use the Network Time Protocol (NTP) to synchronize the system clock (Mills, 1991). The reason that time stamping still causes problems is because the distributed nature of Mercurial facilitates working in disconnected environments.

Mercurial supports extensions which make it possible to add or modify system behavior without making changes to the base architecture (Mercurial, 2011). This change would potentially inhibit two types of added behavior.

### Relative differences between timestamps

The extension can offer the possibility to add relative timestamps on synchronization. In this case the revision null serves as a reference point. The extension saves the time difference for each revision. This way it is possible to compute metadata for each file so relative differences match reality.

### Networked time synchronization upon commit

The extension can offer the possibility to use timestamps received through a network upon synchronization. With this solution the system time is left unattended.

### Impact of the proposed change

Both extensions to the architecture of Mercurial would solve the time stamping problem without making adjustments to the Mercurial core.

In the report that is subject of this review, the problem described above is not addressed. Extensions are only described for front-end extensions. Extensions of core functionality are not addressed.

## Requirement with large impact

The second use-case we chose to investigate, changes requirements in such a way that the actual architecture of Mercurial fails. We investigate a use-case with something the system should sensibly be able to do, but can only be accomplished with much additional re-architecting.

The use case is defined as follows:

**“On the fly editing together at the same time in one file (like Google docs spreadsheet, Google Wave).”**

In present days we seldom come across decentralized repositories in our day to day work, like Mercurial. Software patterns are more and more geared towards a server-client pattern due to advances in technology (Garlan & Shaw, 1993). Traditional server-client patterns at this time are subjective to a significant change called “cloud computing”. Cloud computing and in particular the ‘Cloud’ can be defined as *“a type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumer”* (Buyya, Yeo, & Venugopal, 2008).

For this use case we introduce the requirement to Mercurial that it has to support on the fly editing by multiple users in the same file, at the same time and storing it in the cloud. As the architecture of Mercurial only supports decentralized storage this should cause the architecture to fail. In Mercurial it is not possible to work simultaneously on one document without having to first pull the latest revisions with the command *hg pull* and after that merging those with the command *hg merge*. With the command *hg pull* a full copy (clone) is created on the client’s machine (fetched from another client’s repository) and with *hg merge* the latest changes that the client made are being merged with the latest revision of the file available in the newly created repository. After the changes are committed to the repository by executing the command *hg commit*, a new change set is written to the appropriate change log. Before another user can work on this file, the user has to go through the whole process of pulling (*hg pull*), merging (*hg merge*) and committing (*hg commit*) his changes before his document is up-to-date again. This is a drawback in Mercurial’s architecture when it comes to introducing the requirement mentioned above.

There is no support available in the architecture that can cope with this requirement due to its decentralized nature, which results in an increase of time and recourse behavior. In order to accomplish an implementation of this requirement several functions are in need of implementation:

- Connection possibility to a cloud-server;
- User interface which allows on-the-fly editing in a document without it being locked when multiple users are simultaneously working on it;
- Introduction of a server-client pattern.

### **Impact of the proposed change**

The implications of this requirement result in an architectural failure. With the introduction of this requirement we showed that the architecture of Mercurial does not suffice in handling simultaneous editing in the same document and storing it in a cloud. Although Mercurial allows for expansion with so-called extensions, not one is available that introduces cloud-support or simultaneous editing of files by users due to its architectural limitation.

## Architectural description review of trade-off

We first of all doubt that a system trade-off could have been described with a high level of clarity and preciseness within one and a half page. Secondly there are no trade-offs described within the trade-off analysis. The authors describe some quality aspects which are independent. Take for example *“The trade-off we are going to investigate is centralised versus decentralised. In a decentralised system, recoverability is better, as each clone of the repository has the full history and can be used to recover other clones. In a centralised system, the server can fail, in which case each user is unable to use the system.”* (Alkemade T.P., 2011) A trade-off would describe that one system trades recoverability for security, which happens in this case but is not described.

Another issue within the trade-off analysis of “Architectural overview of Mercurial” is the clarity and preciseness of their statements. Take again for example *“Each clone of the repository has the full history and can be used to recover other clones. In a centralised system, the server can fail, in which case each user is unable to use the system.”* (Alkemade T.P., 2011) We could argue that if a server fails, it does not necessarily mean that the version history file gets corrupted. So what is the acceptable time for a centralised system to be offline, practice learns us that this would not be long. Another example *“The local repository can grow quite large”* (Alkemade T.P., 2011) this sentence is not really clarifying anything, what is quite large? 20 years ago 25 megabytes was quite large, nowadays figures within terabytes are quite in common.

Another discussion could be made on the authors comparative description of centralized and decentralized paragraph “the Hackathon” in regard to preciseness. They state that programmers need to work weekends and due to a closed office need to work within a café with no internet access. Later on, they state that when a programmer is done with his work his colleges will pull data from his repository. But how would this be possible without an internet connection? You could argue that the programmers then communicate through Bluetooth or create their own WIFI hotspots, but this is not done by the authors.

The research would benefit from practical foundations like usage statistics of the Mercurial system.

## Conclusion

In this document we described two different new requirements for Mercurial. A requirement with little impact on the architecture of Mercurial and one with large impact on the architecture of Mercurial. We used the “Architectural overview of Mercurial” (Alkemade T.P., 2011) to verify the impact on the system. Finally, we reviewed the architectural description of Mercurial on preciseness and clarity.

The first requirement is *“The ability to use concurrent timestamps for file meta data.”*. Mercurial uses system timestamps for file creation and modification. Whenever a user merges or its branch with another Mercurial instance the metadata of the files is committed as well. This could mean that whenever the system time differs between users the perceived time modified or created can be displayed incorrect. Difference between time zones is one of the most frequent causes of this problem. These *subjective timestamps* can be misleading for users that compare timestamps of files, and potentially make bad decisions. Both extensions, relative differences between timestamps and networked time synchronization upon commit, to the architecture of Mercurial would solve the time stamping problem without making adjustments to the Mercurial core.

The second requirement is *“On the fly editing together at the same time in one file”*. In present days we seldom come across decentralized repositories in our day to day work, like Mercurial. Software patterns are more and more geared towards a server-client pattern due to advances in technology (Garlan & Shaw, 1993). Traditional server-client patterns at this time are subjective to a significant change called “cloud computing”. There is no support available in the architecture that can cope with this requirement due to its decentralized nature, which results in an increase of time and recourse behavior.

The architectural description of Mercurial consists of a trade-off analysis, but there are no trade-offs described within this section. The authors describe some quality aspects that are independent. Another issue within the trade-off analysis is the clarity and preciseness of their statements. Statements like *“The local repository can grow quite large”* (Alkemade T.P., 2011) are very subjective and vague. Another discussion could be made on the authors’ comparative description of centralized and decentralized paragraph in regard to preciseness.

Finally, the “Architectural overview of Mercurial” (Alkemade T.P., 2011) was enough to come up with the chosen requirements and to verify the impact. There was no need to use other literature about Mercurial to verify the impact of the chosen requirements. The trade-off analysis is short and not a real trade-off analysis but the rest of the document was very useful.

## References

- Alkemade T.P., V. A. (2011). *Architectural overview of Mercurial*. Utrecht.
- Bohner, S. (1996). Software change impact analysis. *Citeseer*.
- Buyya, R., Yeo, C. S., & Venugopal, S. (2008). Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications* (pp. 5-13). IEEE.
- Garlan, D., & Shaw, M. (1993). An introduction to software architecture. *Advances in software engineering and knowledge engineering, 1*, 1-40.
- Mercurial. (2011, January 25). *UsingExtensions*. Retrieved January 26, 2011, from Using Mercurial Extensions: <http://mercurial.selenic.com/wiki/UsingExtensions>
- Mills, D. (1991). Internet Time Synchronization: The Network Time Protocol. *IEEE TRANSACTIONS ON COMMUNICATIONS*, 1482-1493.
- Zhao, J. a. (2002). Change impact analysis to support architectural evolution. *Journal of software maintenance and evolution: research and practice*, 317-333.