**Concordia University**

**SOEN 6611**

**SOFTWARE MEASUREMENT**

**Title: Effect of Code Review Time on Release Time**

**Team Members:**

Asifali Dauva: 6756239

Ankita Mishra: 6776973

Nikita Parashar: 6766994

Pradeep Samuel: 6934153

# Executive Summary

Chromium is a large open source project hosted by Google. As a project manager of Chromium, I am concerned about the delays in the release of upcoming versions of the project. The time spent for a particular release involves developing new features and fixing previously existent issues. But that is not the only area where time is spent, a significant amount of time is also spent on the code review process which also dictates when the release is supposed to take place. In order to understand the problem more closely, we need to look at how the various factors are interacting and affecting the release time.

To gain a deeper insight into the problem, we systematically performed a case study of chromium project data by following steps of an Analytics Pipeline. The results indicate that code review time is the major cause of delays of different releases. One of the main reasons supporting this results is that chromium is developed by a large community of developers who are geographically distributed and working in different time zones. This often leads to a waiting period of up to 24 hours for the code to be reviewed. Other plausible causes such as the code blocking time is found to be ineffective in causing delays. However, this cause could not be verified as ineffective with strong evidence due to lack of data.

# Goal, Question, Measures

As a Chromium project manager, I would like to know about the following for ensuring that the project is running smoothly.

**Question 1:**  Does fixing a bug introduce more bugs or reduces it?

**Motivation:** This question will answer the stability of the system.

**Outcome measure:** Frequency of bugs per File.

**Direct measures:**

a) File change frequency.

b) Number of bugs in each File.

**Confounds:**

a) Complexity of the code.

b) Number of lines of code (in each File).

**Hypothesis:** Changing a file frequently increase the number of bugs and make the system unstable.

**Question 2:**  Does increase in defect resolution time decrease the Defect age\* ?

**Defect age\* =** The time period (duration) for which the defect remains in the system. Basically, the difference in time when the defect is first reported and the time when it is resolved and released.

**Motivation:** This question will answer the quality and customer satisfaction of the system.

**Outcome measure:** Defect Age\*

**Direct measures:**

a) Number of defects.

b) Time to resolve defects.

**Confounds:**

a) Code review time.

b) Priority of defects.

We suspect that an increase in the defect resolution time will decrease the defect age and thereby increase quality and customer satisfaction.

**Hypothesis:** Increasing the defect resolution time will decrease defect age in the system.

**Question 3:** Does increase in code review time increases release time of a milestone?

**Outcome measure:** Release time gap of different versions.

**Direct measures:**

a) Time difference between the code review time and total bug fix time for each bug.

b) Time gap between releases of consecutive milestones.

**Confounds:**

a) Previously raised issues are blocking the code review process (bug blocking time).

b) Number of bugs in each milestone.

**Hypothesis:**

Longer code review time leads to increase in the release time of a project version.

# Raw Data

We used the following sources to extract raw data.

**Project Release Information**: Each chromium project version is associated with a date on which it is released. History of release dates of chromium project versions can be found on Wikipedia or directly from the chromium project page (in JSON format). We used the release history information found on chromium project page (https://www.chromium.org/developers/calendar) as raw data. Release information for version 36 to 43 (for each supported OS) is available from raw data.

**Code Review Information:** Code Review information for project bugs can be found on the chromium project page (https://codereview.chromium.org/). Each entry in the list is a link to a review web page associated with a unique ID. A review page contains a reference (bug id: XXXX) to bug ID (if the review is for a bug) on the web page text apart from other review related information. We downloaded and saved the web page corresponding to each review which is closed i.e. completed as raw data for extracting review information.

**Bug Information:** Bug information for chromium projects can be found on the chromium project page (https://code.google.com/p/chromium/issues/list). Each entry in the list of bugs is a link to the bug web page for a particular version of chromium release. For extracting the bug information, we first looked into each review, extracted the bug id from the review and then downloaded the bug web page corresponding to bug id. For e.g. https://code.google.com/p/chromium/issues/detail?id=478453 corresponds to the web page for bug with id 478453.

# Attribute Extraction

In the following text we describe the process to extract each attribute from Raw data.

**Table:** Release\_Info (Release Information from JSON)

**Attribute:** os (Target OS platform for the project version)

**Type**: String

**Scale:** Categorical

**Description:** This is the platform for the project version released. It is available from the JSON release history downloaded from chromium web page.

**Attribute:** channel (Channel indicating type of release)

**Type**: String (List of Values)

**Scale:** Categorical

**Description:** This is the type of the project version (e.g. Stable, Beta etc.). It is available from the JSON release history downloaded from chromium web page.

**Attribute:** major\_version (Project version number)

**Type**: Number

**Scale:** Categorical

**Description:** This is version number of the release. It is available from the JSON release history downloaded from chromium web page.

**Attribute:** release\_date (Release date of the project version)

**Type**: Date

**Scale:** Interval

**Description:** This is the release date of the project version. It is available from the JSON release history downloaded from chromium web page.

**Table:** Release\_Versions\_Time\_Gap (Derived Table)

**Attribute:** release\_time\_gap\_days (Release Time Gap)

**Type**: Amount of time in days between 2 consecutive releases (Major Release and Previous Release).

**Scale:** Ratio

**Description:** This is a derived attribute.Release date for each version corresponding to each OS is extracted from JSON and persisted into the DB (e.g. Release: 43, OS: Windows, Date: 12/3/2015, Channel: Stable). Pairs of releases are formed (in descending order for each OS and Channel) and time gap between release dates of values in the pair becomes time gap for the release with higher number (e.g. Pairs = {(43, 42), (42, 41), .... }. Difference between release dates of release 43 and 42 becomes the time gap for release 43).

**Attribute:** os (Target Operating System for each release)

**Type:** String

**Scale:** Categorical

**Description:** Target platform (OS) for each project version is available from the release history (JSON format) downloaded from chromium web page.

**Attribute:** major\_version (Release Version Number)

**Type:** Number

**Scale:** Categorical

**Description:** Version number for each release is available from the release history (JSON format) downloaded from chromium web page.

**Attribute:** previous\_major\_version (Previous Release Version Number)

**Type:** Number

**Scale:** Categorical

**Description:** Version number for each release is available from the release history (JSON format) downloaded from chromium web page.

**Attribute:** release\_date (Release date of a project version)

**Type:** Date

**Scale:** Interval

**Description:** Release date for each project version is available from the release history (JSON format) downloaded from chromium web page.

**Attribute:** previous\_release\_date (Release date of a previous project version)

**Type:** Date

**Scale:** Interval

**Description:** Release date for each project version is available from the release history (JSON format) downloaded from chromium web page.

**Table:** Bug\_Info

**Attribute:** issue\_id (Unique ID for identifying a Bug)

**Type**: String

**Scale:** Categorical

**Description:** Bug Id is available on each bug's web page. It identifies each bug uniquely. The bug ID is a number but without any value associated with it.

**Attribute:** priority (bug priority number)

**Type**: Number

**Scale:** Ordinal

**Description:** Priority of bug is available on each bug's web page. A priority defines the importance and urgency of the bug for resolution. Generally, a high priority bug (indicated by a lower number) must be resolved before a low priority bug (indicated by a higher number).

**Attribute:** milestone (project version number)

**Type**: Number

**Scale:** Categorical

**Description:** The project version in which the bug will be resolved and released. Milestone information is available on each bug's web page.

**Attribute:** status (Bug status)

**Type**: String (List of Values)

**Scale:** Categorical

**Description:** Status of each bug indicates its current state for e.g. Started, Fixed, Assigned etc. Status information is available on each bug's web page.

**Attribute:** os (Target Operating System for each Release)

**Type**: String (List of Values)

**Scale:** Categorical

**Description:** Target platform (os) for which the bug is fixed and will be released. OS information is available on each bug's web page.

**Attribute:** start\_date (date on which the bug is opened)

**Type**: Date

**Scale:** Interval

**Description:** Start date indicates when the bug is reported in the system. Start Date information is available on each bug's web page.

**Attribute:** end\_date (date on which the bug is closed)

**Type**: Date

**Scale:** Interval

**Description:** End date indicates when the bug is fixed and closed in the system. Start Date information is available on each bug's web page.

**Attribute:** blocked (Indicates if the bug is blocked)

**Type**: Boolean

**Scale:** Categorical

**Description:** Blocked indicates if the bug is blocked by some other bugs. If a bug is blocked it is not closed until the blocking issues are resolved first. Blocked information is available on each bug's web page.

**Table:** Review\_Info

**Attribute:** review\_id (Unique ID for each review)

**Type:** Number

**Scale:** Categorical

**Description:** This is the unique id for each review. Review id can be found on the review web page.

**Attribute:** bug\_id

**Type:** Number

**Scale:** Categorical

**Description:** This is the reference to the bug for which the review is conducted. Bug Id is found in the review web page text as BUG: XXXX. However, it is not necessary that a review is associated with a bug.

**Attribute:** start\_date

**Type:** Date

**Scale:** Interval

**Description:** The created/start date is represented in terms of interval time from the present date (for example it would take the form “1 day, 8 hours ago”) on the review web page. This text was parsed and the appropriate years, months, weeks, days, hours and minutes were collected and reduced from the current date to get the created/start date of the review. This data is used in the measurement of code review time.

**Attribute:** last\_modified\_date

**Type:** Date

**Scale:** Interval

**Description:** Similar to the start date, the last modified date is represented in terms of interval time from the present date (for example it would take the form “1 day, 8 hours ago”) on the review web page. This text was parsed and the appropriate years, months, weeks, days, hours and minutes were collected and reduced from the current date to get the last modified date of the review. This data is used in the measurement of code review time.

**Attribute:** bug\_blocked\_time

**Type:** Amount of time in days

**Scale:** Ratio

**Description:** The bug blocked time is the amount of time the bug is blocked by other bugs from being released (bugs are flagged as blocked; only those bugs were considered to have blocking time). The difference between the last modified time of the code review of that bug (the closed time of the code review, because there is no explicit closed date specified for a review other than the status update as closed) and the time the bug was closed is the time the bug was blocked from being released.

**Coupling Between Objects (CBO)**

**Type:** Number

**Scale:** Absolute

**Description:** First, we extracted all the system classes (excluding library classes) in the code base. Second, we followed the following steps to calculate CBO.

For each class in the set of system classes,

1. The classes which it is dependent on and the classes which depend on it are obtained and are stored in two separate sets.
2. The union set of the obtained two sets of classes are taken, resulting in a set of classes that the class is coupled with.
3. The intersection set between the obtained set of classes and the set of all classes within the codebase is performed to get only the classes the current class is coupled to within the code base excluding all other library or third party classes. The length of the obtained set of classes is the CBO value.

**Lack of Cohesion in Methods (LCOM)**

**Type:** Number

**Scale:** Absolute

**Description:** First, we extracted all the system classes (excluding library classes) in the code base. Second, we followed the following steps to calculate LCOM.

For each class in the code base,

1. Each attribute (member) in the class is obtained.
2. For each of the attribute, a array of set of methods is created which access the same attribute.
3. Next, the methods defined in the class are collected.
4. For each of the methods, an array of set of methods is created if the methods have call between them.
5. The array of sets from step 2 and step 4 are combined into a single array.
6. If all the sets of methods in the array are disjoint (intersection is null) then the length of the array is the LCOM value.
7. Otherwise, sets of methods which have a common member are combined (union set) until all the sets are disjoint sets. The step 7 is repeated to obtain the LCOM value.

# Link and Store

In this section we describe how chromium project data is linked to each other.

**Relationship : Release\_Info -> Release\_Versions\_Time\_Gap**

**Description:** Release\_Versions\_Time\_Gap is releated to Release\_Infovia attributes OS, Major\_Version, Previous\_Major\_Version, Release\_Date and Previous\_release\_Date. Each entry in Release\_Versions\_Time\_Gap is associated with a pair wise grouping of consecutive releases for each OS. Release\_Versions\_time\_Gap is a table derived from Release\_Info and used for calculating and storing the time gap (in days) between 2 consecutive releases.

**Relationship : Release\_Versions\_Time\_Gap -> Bug\_Info**

**Description:** Release\_Versions\_Time\_Gap is related to Bug\_Info via attributes Milestone and OS. Each bug in Bug\_Info is associated with a Milestone (project version or release) in which it is supposed to be fixed and released and the OS i.e. the corresponding platform of the project. By linking bugs with release version time gap info we can calculate the number of bugs in each release (or milestone) of the project for each platform.

**Relationship** : **Bug\_Info -> Review\_Info**

**Description:** Bug\_Info is related to Review\_Info via attributes Bug\_Id. Each review is associated with a bug id corresponding to the bug for which it is created. By linking reviews with bugs we can calculate the bug code review time and bug blocking time.

**Relationship** : **Release\_Versions\_Time\_Gap -> Bug\_Info -> Review\_Info**

**Description:** Release\_Versions\_Time\_Gap, Bug\_Info and Review\_Info are related by (OS and Milestone) and (Bug\_id) correspondingly. Each milestone (or release) of a Project for a particular platform has bugs on which code reviews have been performed. By linking all 3 tables together we can calculate the release time gap between release of major versions (or releases) of the project for each platform, total number of bugs in the release, code review time for the bugs in the release and total blocking time of bugs in the release.

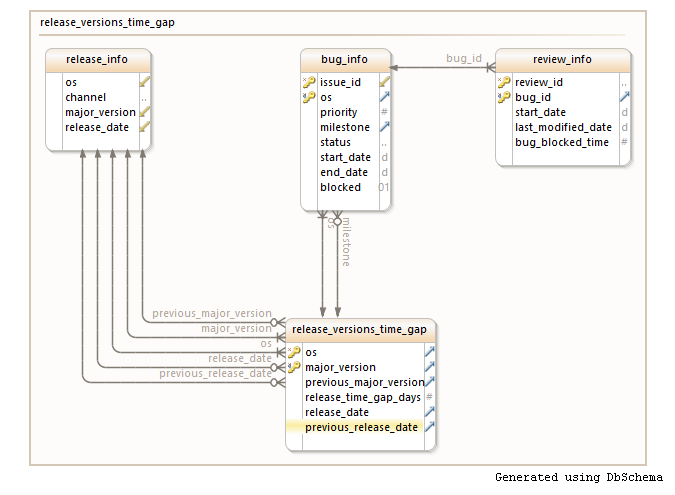


Figure 1: DB schema for Assignment 2.

**Table**: **Summary\_Info (Assignment 2)**

|  |  |
| --- | --- |
| Attributes | Scale |
| os | **Categorical** |
| release | **Categorical** |
| previous\_release | **Categorical** |
| release\_time\_gap | **Ratio** |
| number\_of\_bugs | **Absolute** |
| code\_review\_time | **Ratio** |
| total\_blocked\_time | **Ratio** |

**Table**: **LCOM\_CBO\_Metrics (Assignment 3)**

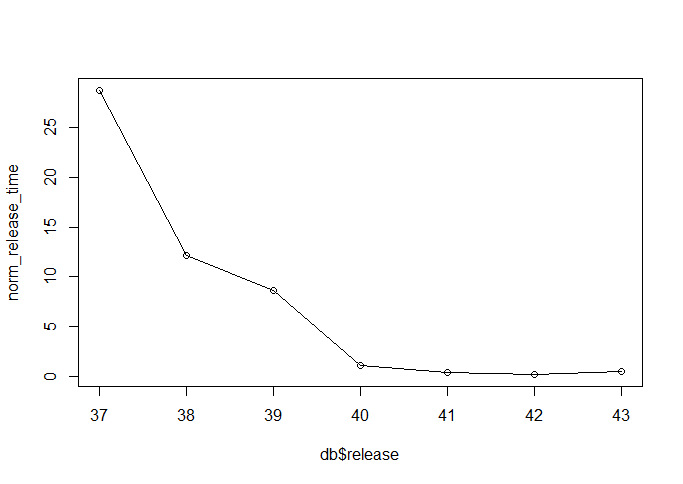
|  |  |
| --- | --- |
| Attributes | Scale |
| release | **Categorical** |
| class\_path | **Categorical** |
| cbo | **Absolute** |
| lcom | **Absolute** |

# Descriptive Statistics

Due to the nature of our data we found line chart to be more appropriate than a box plot to study the behaviour of data for Assignment 2. In this section we plot and describe the behaviour of important data for each release.

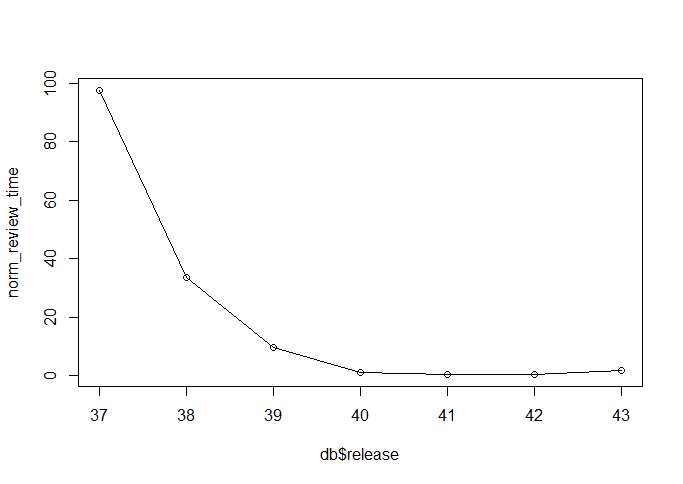
**Release Time Gap**

The following is a plot of release time gap for each release. The release time gap is normalized (by dividing it by the number of bugs) however, to make it comparable. From the graph it can be seen that release time is higher for older release (37, 38 and 39) and mostly constant for newer releases (40,41,42 and 43).

****

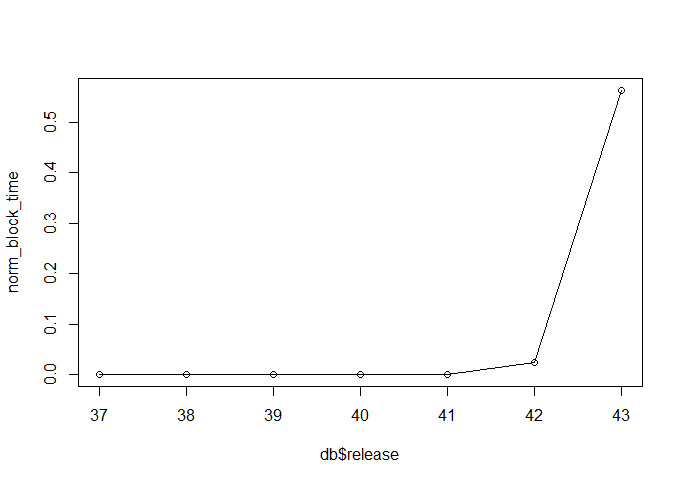
**Code Review Time**

The following is a plot of code review time for each release. The code review time is normalized (by dividing it by the number of bugs) however, to make it comparable. From the graph it can be seen that review time is higher for older release (37, 38 and 39) and mostly constant for newer releases (40,41,42 and 43).



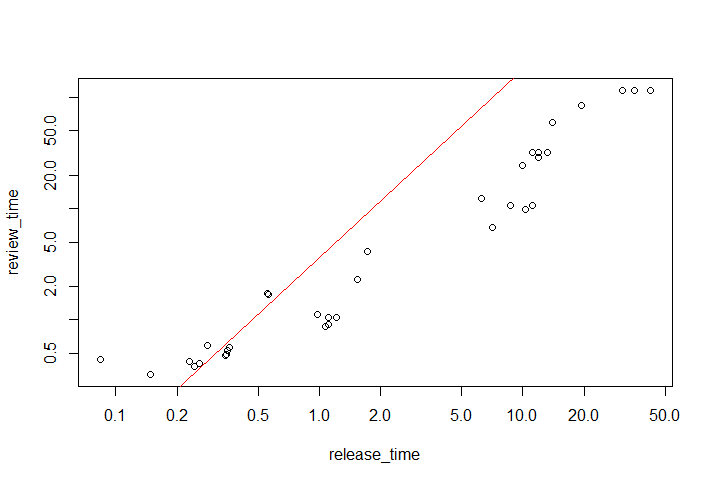
**Code Blocking Time**

The following is a plot of code blocking time for each release. The code blocking time is normalized (by dividing it by the number of bugs) however, to make it comparable. From the graph it can be seen that most of the releases (37, 38, 39, 40 and 41) have blocking time as zero and slightly higher blocking for release 42 and 43. The reason for not having blocking time for older releases could be insufficient data or the absence of the concept of 'blocking' issues.

****

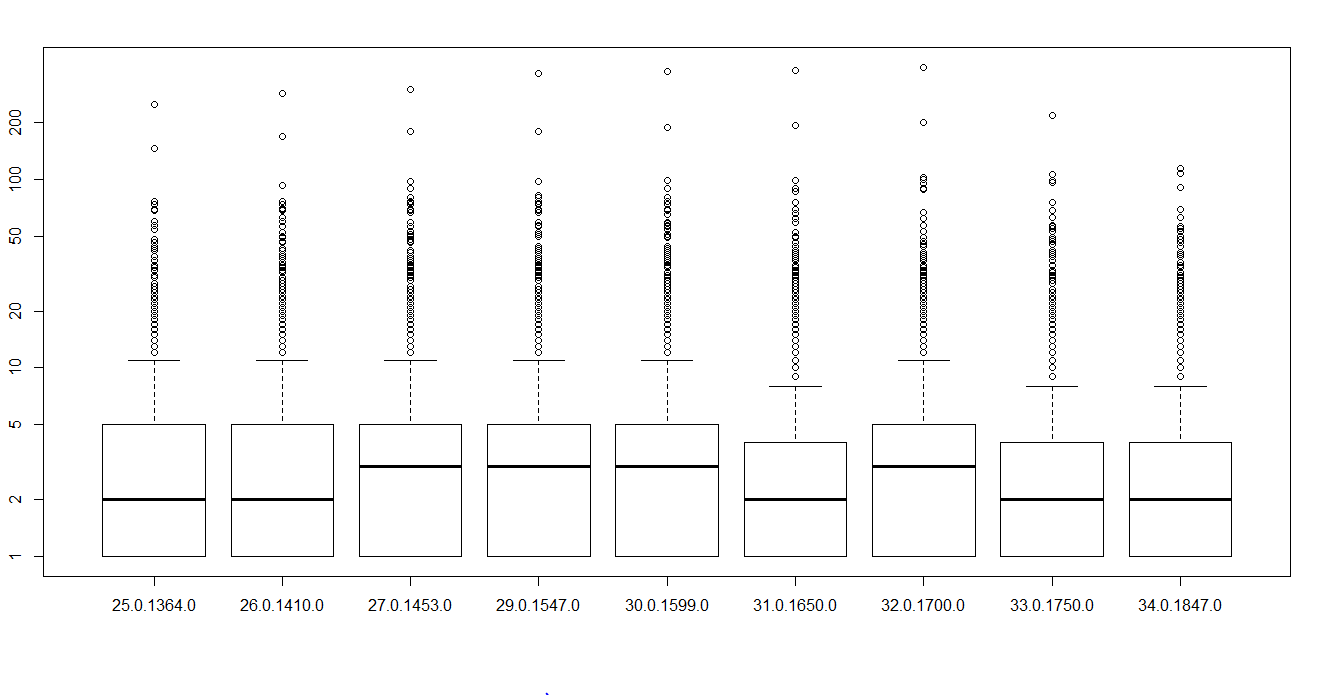
**Release Time Gap v/s Code Review Time**

The following is a scatter plot (log scales) of release time gap v/s code review time for each release (for each OS). The review time and release time are normalized (by dividing them by the number of bugs) for comparison. The red line is a linear regression line for fitting the maximum number of releases. From the graph It can be seen that there is a positive correlation (although not a very strong one) between review time and release time. The plot tends to suggest that as review time increases release time also increases.

****

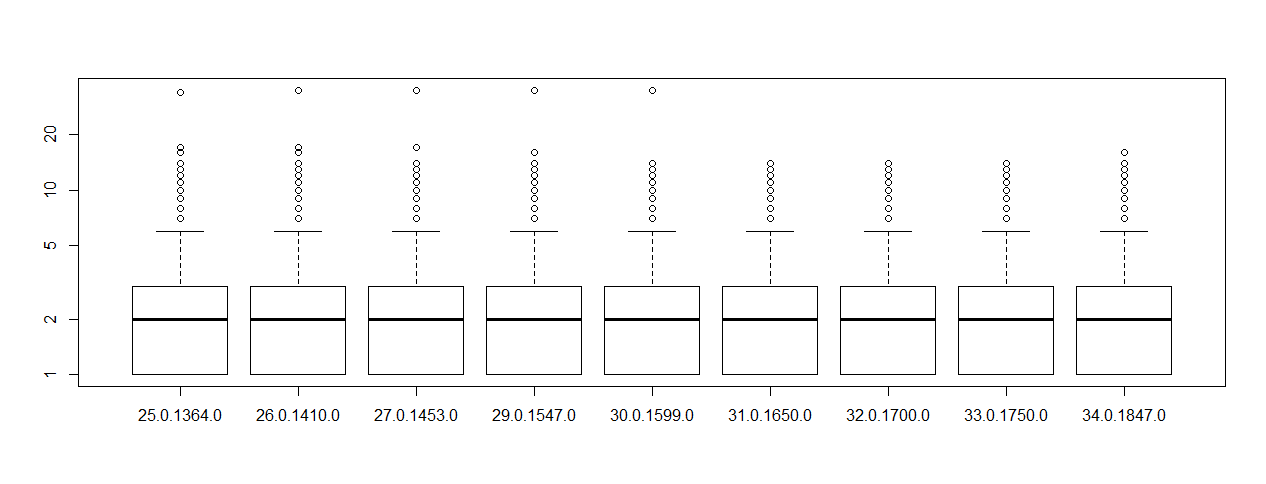
**CBO Metrics**

The following is a Box plot for CBO value for each file (for each release). The mean CBO value across the code base for each release is found to be similar but with slight variations. These variation may be due to the refactoring of the code that could have triggered it but when verifying with the metrics obtained from the Understand tool, we found the same variations. We could not find the exact reason for these variations.



**LCOM Metrics**

The following is a Box plot for LCOM value for each file (for each release). The mean LCOM value for each release is found to be almost constant. We suspect that chromium project development process follow some kind of best practices which prevents the quality of code from deteriorating.



# Statistical Model

We created a statistical model to find out the correlation between the release time gap and the time spent on code review and to check how the release time gap can be predicted with the time spent on code review; we also considered the other confounding factors such as the number of bugs and the blocking time in our model. Our unit of measure is a project release time gap. In order to normalize the release time, the code review time and the blocked time we divided these quantities by the number of bugs (confounding factor). In the model depicted below (Figure 2), the release time, the review time and blocked time are already normalized.

Model: Release Time ~ Review Time + Blocked Time.

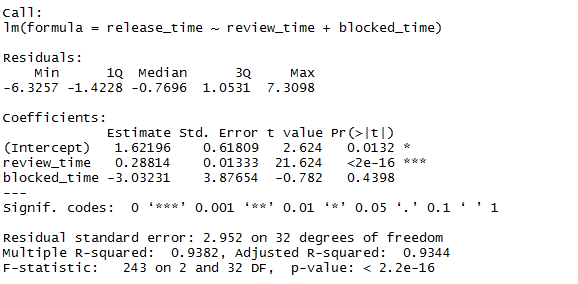


Figure 2: Liner Regression Model

From results of the statistical model above (Figure 2) we can see that the predictor code review time is statistically significant (since its p-value is very near to zero <2e-16) whereas the blocking time is not statistically significant (since its p-value is not near zero 0.4398). When plotting the linear model obtained in a scatter plot (see above - Release Time Gap v/s Code Review Time), we could observe that the model is not a perfect fit to the data but it approximately shows a positive correlation between the data. We can conclude from the results above that an increase in code review time does lead to an increase in release time gap. One of the main reasons for the results could be that developers and reviewers are working in different time zones leading to a delay in code reviews and thereby blocking the progress of the project.

# Threats to Validity

In the following text we describe various threats to validity of our case study.

**Conclusion Validity**

Our case study has conclusion validity since the correlation between review time and release time is found to be statistically significant (indicated by the p-value). However, the insignificance of code blocking time to the release time seems to be conclusion validity due to insufficient data.

**Construct Validity**

Our case study has construct validity since we have selected only the relevant and meaningful measures such as code time and blocking time which can affect release time. However, the main threat to construct validity in our case is the measurement of derived attributes such release time gap, code review time and blocking time. The measures of release time gap, code review time and blocking time are not very accurate since these attributes were derived from other atomic data. For e.g. bug blocking time is calculated as the difference between the end time of review and closed time of a bug. This rule may not be strictly followed for all the blocked bugs.

**Internal Validity**

This case study has internal threat to validity in the form of other relevant attributes that may affect the release time. For e.g. in our case study we have not considered the time spent on bugs for verification (testing) before release.

**External Validity**

This case study poses external threat to validity in the form difference in development processes. Chromium is a large open source project developed by community of geographically distributed developers working in different time zones. This may not be true for all the development projects some of which might not be open source or not developed by people working in different time zones.

# Recommendation and Actions

In our study we have tried to uncover the factors which are affecting the release time of different project versions. From the statistical model we see that code review time is positively correlated and is statistically significant. This suggests that increase in code review time is increasing the release time. One of the reasons that can be attributed to longer review time is the difference in time zones of reviewers and developers. This can lead up to a waiting period of 24 hours for the code to be reviewed. This waiting time can be reduced if developers could find reviewers who are working in their (developers) local time zones.

Another important factor which is found to be statistically insignificant is the code blocking time. It seems like code blocking time is not affecting the release time. However, this might be caused by insufficient data or the absence of the concept of 'blocking' code in older releases. Nevertheless, we are suggest having a priority assigned to each review so that the bugs which are blocking other bugs will be reviewed earlier and thereby reducing the blocking time to a certain extent.

Conclusion

To conclude, in this case study we successfully identified and extracted measures such as code review time and code blocking time and analysed their effect on release time. Our statistical model based on these measures suggests that increase in code review time does lead to an increase in release time. To reduce the time spent in code reviews, we propose that developers find reviewers who are available in the same time zone as they are working. Another important measure which is found to be statistically insignificant due to lack of sufficient data is the code blocking time. Irrespective of the results, our opinion is to reduce the blocking time by assigning a priority to reviews for bugs which are blocking other bugs. The findings in this study can be used as a reference to conduct a more detailed analysis of the measures affecting release time with more accurate data.