

*Software Measurement. SOEN 6611 2014/4 D*

*Dr. Peter Rigby*

**Chromium:**

**Collaboration and issue resolution**

**Class coupling and cohesion**

|  |  |
| --- | --- |
| **Team members information (in no particular order)** | |
| **Name** | **SID** |
| Ali Sangari | 6816304 |
| Hojabr Sattari | 6435807 |
| Bhaskar Baddam | 6761038 |

Contents

[Executive Summary 4](#_Toc417248311)

[Goal, Question, Measures 5](#_Toc417248312)

[Raw Data 8](#_Toc417248315)

[Attribute Extraction 10](#_Toc417248318)

[Link and Store 13](#_Toc417248320)

[Descriptive Statistics 16](#_Toc417248323)

[Statistical Model 22](#_Toc417248326)

[Threats to Validity 24](#_Toc417248329)

[Recommendation and Actions 25](#_Toc417248334)

[Conclusion 25](#_Toc417248335)

# Executive Summary

Addressing issues in a software system after it has been released, is an expensive process.

In this report we are looking at whether collaboration on reported issues increases productivity and lowers the cost associated with this process.

For this investigation we analyzed 60,000 reported issues. From these issues we looked at closed issues and the number of people involved in resolving them.

Another aspect of Chromium that we looked at is how the level of cohesiveness and coupling is maintained as new versions of the project are released. This indicates the level of maintainability of aging of the software.

# Goal, Question, Measures

## Assignment 2

Our initial goal in Assignment 2 was to study different aspects of Chromium project to make sure it is running smoothly.

We came up with the following list of questions for maintainability, reliability, productivity, compatibility, and quality.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Question** | **Metrics** | **Category** |
| 1 | Do external dependencies lower Chrome’s quality? | Number of issues closed with *ExternalDependency* status | Maintainability |
| 2 | How is Chrome aging? | Growth rate of *WontFixes* over time or version    Ratio of verified fixes vs. reported bugs over time or version. | Maintainability |
| 3 | How does rapid release impact the software quality? | Number of valid Issues per version | Reliability, Maintainability |
| 4 | How quickly are critical bugs (priority 0) fixed and verified? | Measuring “fixing time” (from an issue being reported until a fix is verified) of critical issues | Productivity |
| 5 | Does the number of CCed developers affect how quickly issues are resolved | Measuring avg. fixing time for issues with and without CC | Productivity |
| 6 | How important is for version number to be recorded in a ticket (logged issue) (what impact does it have on the issue status/ issue’s fate)? | Measuring avg. fix time for issues with and without version number. | Productivity |
| 7 | Based on logged issues, how well does Chrome support other platforms (operating systems)? | Grouping and counting issues per platform/OS. | Compatibility |
| 8 | What is the impact of issue attachment on the issue’s fate? | Percentage of fixed issues vs. reported issues (issues with attachments) | productivity |
| 9 | What is the defect density of chromium? | Defects per KLOC | Quality |

We then conducted an interview a Chromium committer at Google Montreal during a Google tech talk.

During this interview we learned the following,

* Priority setting for issues is mainly based on support or quality assurance engineer’s judgement.
* It is also the same person that initially forwards an issue to a specific person or group of people for further investigation and resolution.

These two pieces of information show that level of experience and good judgment of directly affects the fate of an issue.

From the above set of questions we decided to study the following question as it also helps the management team to know if the resources and budget is efficiently allocated in the issue resolution process.

|  |  |
| --- | --- |
| **Question 5.** | How does having CCed\*1 developers affect how quickly issues are resolved? |
| **Outcome** | Length of issue resolution time. |
| **Direct measure(s)** | Number of issues with CCed developers  Number of issues without any CCed developers  Delta between report date and close date |
| **Confounding factor(s)** | Assignment of issue to a senior member who’s familiar with the area of the issue would lower the importance of collaboration. |
| **Hypothesis** | Collaboration reduces issue resolution time. |

## Assignment 3

In Assignment 3 we looked at Chromium project from a different view point. This time we looked at CPP files in source code to know if coupling and cohesion of the project is improved or kept at the same level as new versions are released.

We used SciTools Understand API for Python for this purpose.

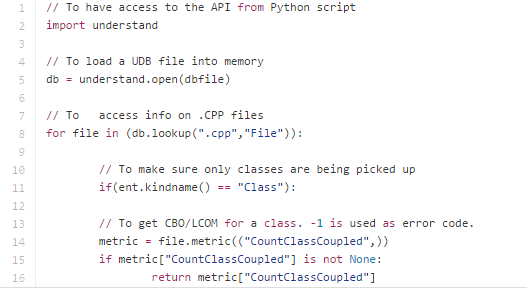


Figure 1. Shows basics of using Understand Python API.

# Raw Data

## Assignment 2

To start our data analysis for Assignment 2 we extracted data from 60,000 Chromium bug report text files that were provided to us. These bug reports included closed, pending and open issues for multiple versions of the project. They also included discussion threads on each issue.

The following information is logged in each ticket if data for it is available,

* Issue Number
* Title
* Description
* Status
* Owner
* Type
* Priority
* CC
* OS
* Reported By
* Reported On
* Closed Date
* Attachments
* Discussions
* Labels for further categorization such as Proj-Hera, Cr-UI-Browser-AppShortcuts

## Assignment 3

In assignment 3 we worked on Chromium source code from versions 25-34 and version 44 (chromium’s latest version). The task was to calculate/extract CBO (Coupling between objects) and LCOM (Lack of cohesion in methods) for all CPP class files.

To achieve this SciTools Understand and its Python API was used to first import each version of the project using the IDE and create a UDB file and then work with the UDB file using Python to extract CBO and LCOM.

The following is a list metrics available from Understand for Object Oriented C++,

* CountClassBase Number of immediate base classes. [aka IFANIN]
* CountClassCoupled Number of other classes coupled to. [aka CBO (coupling between object classes)]
* CountClassDerived Number of immediate subclasses. [aka NOC (number of children)]
* CountDeclClass Number of classes.
* CountDeclClassMethod Number of class methods.
* CountDeclClassVariable Number of class variables.
* CountDeclInstanceMethod Number of instance methods. [aka NIM]
* CountDeclInstanceVariable Number of instance variables. [aka NIV]
* CountDeclInstanceVariablePrivate Number of private instance variables.
* CountDeclInstanceVariableProtected Number of protected instance variables.
* CountDeclInstanceVariablePublic Number of public instance variables.
* CountDeclMethod Number of local methods.
* CountDeclMethodAll Number of methods, including inherited ones. [aka RFC (response for class)]
* CountDeclMethodFriend Number of local friend methods. [aka NFM]
* CountDeclMethodPrivate Number of local private methods. [aka NPM]
* CountDeclMethodProtected Number of local protected methods.
* CountDeclMethodPublic Number of local public methods. [aka NPRM]
* CountOutput Number of called subprograms plus global variables set. [aka FANOUT]
* MaxInheritanceTree Maximum depth of class in inheritance tree. [aka DIT]
* PercentLackOfCohesion Lack of Cohesion in Methods 100% minus the average cohesion for package entities. [aka LCOM, LOCM]

# Attribute Extraction

We extracted the following attributes from bug reports,

**CC**: we extract the number of people in CC for each issue to analyze if more people in CC results quicker issue resolution. We also included people in the discussion thread.

**Report date**: the data the issue is reported.

**Close date**: the data the issue is closed.

**Status**: this is the status of the issue which can be one of the following values (all of the highlighted items categorize as **Closed**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Invalid | Verified | Fixed | Assigned | Duplicate |
| WontFix | Available | IceBox | Untriaged | Started |
| Unconfirmed | ExternalDependency | Archived | FixUnreleased |  |

**Id**: the ID attributes is a unique value.

## Assignment 3

**LCOM** metric measures the correlation between the methods and the attributes of a class. High cohesion indicates good class subdivision. Lack of cohesion or low cohesion increases complexity. Classes with low cohesion could probably be subdivided into two or more subclasses with increased cohesion.

The LCOM version that we preferred to use is LCOM4 (Hitz & Montazeri, 1995). It is calculated as the number of disjoint components in the graph that represents:

* each method as a node
* sharing of at least one attribute between two methods as an edge
* presence of a method call between two methods as an edge

The cohesiveness of a class can be determined from the LCOM value as follows:

LCOM = 1 indicates a cohesive class, which is a "good" class.

LCOM >= 2 indicates a problem. The class should be split into smaller classes.

LCOM = 0 happens when there are no methods in a class. This is also a "bad" class.

However, SciTools Understand uses the following to define LCOM:

100% minus average cohesion for class data members. Calculates what percentage of class methods use a given class instance variable. To calculate, average percentages for all of that class’es instance variables and subtract from 100%. A lower percentage means higher cohesion between class data and methods. [**Research:** Chidamber & Kemerer – Lack of Cohesion in Methods (LCOM/LOCM)]

To calculate LCOM4 (Hitz & Montazeri, 1995), we need the following metrics,

1. Number of functions/methods in a class (given by Understand as "CountDeclFunction")
2. Number of method pairs in class with at least one instance variable that they commonly use or define in their body.
3. Number of method pairs in class that have at least one instance method that they commonly call in their body.

From the three metrics that are mentioned above, Understand only provides the first item. A solution would be to calculate the two metrics (items 2 and 3) by parsing the code using a home grown tool. In this assignment we use LCOM provided by Understand.

The following links show a list of available metrics form SciTools and a stackoverflow question as part of our research.

* List of metrics made available through Understand API: <https://scitools.com/support/metrics_list/>
* Stackoverflow question and reply: <http://stackoverflow.com/questions/29464392/how-to-calculate-cbo-and-lcom-for-a-class-using-python-and-scitools-understand-a>

**CBO** metric represents the number of classes coupled to a given class. This coupling can occur through method calls, field accesses, inheritance, return types, and exceptions. High CBO is undesirable, high coupling has been found to indicate fault-proneness.

We used CBO (Chidamber & Kemerer) in Understand API. It is calculated as the number of unique classes a class references excluding base classes and nested classes.

# Link and Store

## Assignment 2

At this stage, we store all the extracted attributes in the database.

One of the issues that we faced was with level detail and format of closeDate and reportDate. reportDate is a full date-time format to seconds, but closeDate is a string representing only Year and Month of when the issue was closed. To solve this problem, we converted both of these dates to year/month/day format and pushed closeDate to the end of the month to avoid situation where closeDate is before reportDate.

We also created a CCs table that links emails in CC and discussion thread to bug reports.

ccs [table]  
*(CCed email per issue)*

id

issueId

email

filteredIssues [table]  
*(closed issues)*

id

attachment

cc

closeDate

os

owner

priority

reportDate

status

type

closeDateF

reportDateF

timeDelta (in days)

rawIssues [table]  
*(all issues)*

id

attachment

cc

closeDate

os

owner

priority

reportDate

status

type

Summary [view]  
(CCed email per issue)

cc *(grouped by # CCed devs)*

avg(deltaTime)

Figure 2. Tables and relationships in assignment 2

## Assignment 3

In assignment 3 the initial extraction and storage is done by Understand. We then created CSV files for each version. Each record in csv files includes “version”, “class\_name”, “cbo”, and “lcom” that are extracted from understand database.

CSV files

containing class\_name, cbo, lcom

UDB file

Understand db containing all metrics

*\*.cpp*   
source files

Figure 3. Attribute extraction from source code to UDB using Understand and to CSV using Python (Understand API)

# Descriptive Statistics

## Assignment 2

As we can see from the summary, bug reports with grouped by 0, 1, 10,13,14,16, and 22 CCed developers are significant.

Call:

lm(formula = timeDelta ~ as.factor(cc))

Residuals:

Min 1Q Median 3Q Max

-607.4 -173.2 -149.2 -95.8 7438.7

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 215.516 20.371 10.580 < 2e-16 \*\*\*

as.factor(cc)1 -51.248 21.612 -2.371 0.01773 \*

as.factor(cc)2 -19.297 21.805 -0.885 0.37619

as.factor(cc)3 -34.190 22.149 -1.544 0.12269

as.factor(cc)4 -17.178 22.772 -0.754 0.45064

as.factor(cc)5 -6.669 23.421 -0.285 0.77586

as.factor(cc)6 -38.769 24.546 -1.579 0.11424

as.factor(cc)7 20.799 26.757 0.777 0.43698

as.factor(cc)8 -25.514 29.869 -0.854 0.39301

as.factor(cc)9 27.454 33.835 0.811 0.41715

as.factor(cc)10 92.903 40.815 2.276 0.02284 \*

as.factor(cc)11 63.658 45.326 1.404 0.16019

as.factor(cc)12 71.767 56.009 1.281 0.20008

as.factor(cc)13 159.137 63.897 2.491 0.01276 \*

as.factor(cc)14 132.369 77.226 1.714 0.08653 .

as.factor(cc)15 91.780 101.250 0.906 0.36470

as.factor(cc)16 365.242 116.321 3.140 0.00169 \*\*

as.factor(cc)17 -42.864 138.683 -0.309 0.75726

as.factor(cc)18 -89.071 156.397 -0.570 0.56901

as.factor(cc)19 -173.849 191.004 -0.910 0.36273

as.factor(cc)20 195.270 177.003 1.103 0.26995

as.factor(cc)21 209.818 191.004 1.098 0.27199

as.factor(cc)22 399.848 199.403 2.005 0.04495 \*

as.factor(cc)23 -23.266 329.572 -0.071 0.94372

as.factor(cc)24 -147.516 465.640 -0.317 0.75140

as.factor(cc)25 -152.516 380.375 -0.401 0.68845

as.factor(cc)26 -48.849 380.375 -0.128 0.89781

as.factor(cc)28 -158.516 658.200 -0.241 0.80969

as.factor(cc)29 -192.516 658.200 -0.292 0.76991

as.factor(cc)30 -64.516 658.200 -0.098 0.92192

as.factor(cc)32 -191.516 465.640 -0.411 0.68086

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

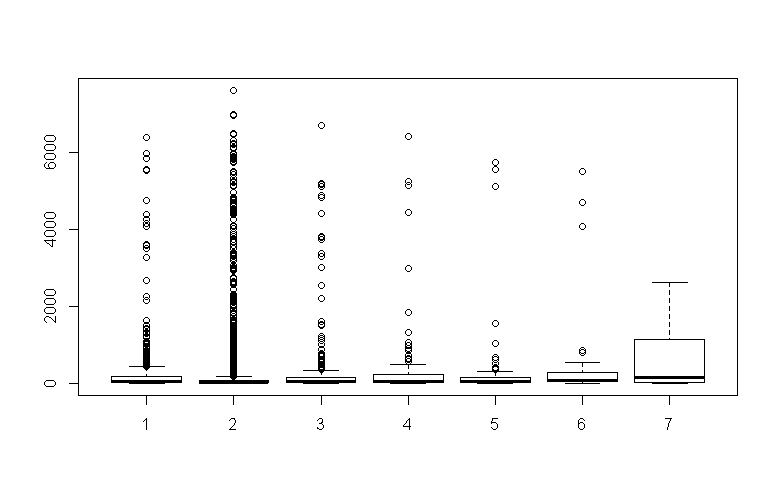
Residual standard error: 657.9 on 36009 degrees of freedom

Multiple R-squared: 0.002374, Adjusted R-squared: 0.001543

F-statistic: 2.856 on 30 and 36009 DF, p-value: 2.991e-07

|  |  |
| --- | --- |
| **Number of CCed developers** | **Avg of timeDelta** |
| 0 | 131.600736 |
| 1 | 106.4569166 |
| 2 | 129.1752522 |
| 3 | 109.08632 |
| 4 | 126.9983852 |
| 5 | 156.6293413 |
| 6 | 103.5320218 |
| 7 | 181.1924119 |
| 8 | 139.9071505 |
| 9 | 192.710311 |
| 10 | 243.744382 |
| 11 | 159.8494624 |
| 12 | 152.0958084 |
| 13 | 235.2926829 |
| 14 | 216.097561 |
| 15 | 209.1956522 |
| 16 | 339.3055556 |
| 17 | 165.3333333 |
| 18 | 119.6842105 |
| 20 | 410.7857143 |
| 21 | 285.3846154 |
| 22 | 446.5833333 |
| 23 | 192.25 |
| 24 | 68 |
| 25 | 63 |
| 26 | 166.6666667 |
| 28 | 57 |
| 29 | 23 |
| 30 | 151 |
| 32 | 24 |

All highlighted groups are plotted in the following boxplot diagram.



0

1

10

13

14

16

22

Figure 4. Boxplot for CC groups 0, 1, 10, 13, 14, 16, and 22.

Above diagram shows that majority of values for each group is close to the mean value of that group.

Next chart draws all of the CC groups and the average time they took to get closed and draw a trend line by paying specific attention to the significant groups listed above.

Figure 5. Number of CCed developers against issue resolution time.

Above diagram shows as the number of CCed developers increases, the time that is closing an requires increases.

## Assignment 3

Next diagram shows that coupling drastically drops as the application moves from version 29 to 30. This improvement in coupling between classes may have been the result of a major refactoring.

Figure 6. Average CBO for each release

25

26

27

28

29

30

31

32

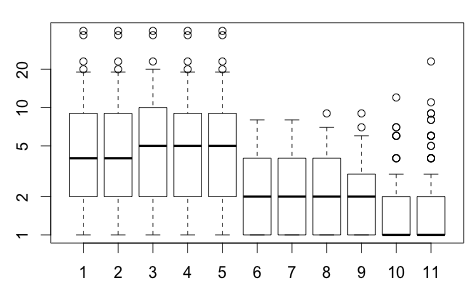
33

34

44

Below boxplot shows that

* Versions under 30 have CBO between 2 and 10.
* Versions 30 – 33 have lower coupling compared to their older versions.
* From version 34 – 44 CBO improves even further and remains consistent



**25**

**26**

**27**

**28**

**29**

**30**

**31**

**32**

**33**

**34**

**44**

Figure 7. Boxplot for CBO in each release

# Statistical Model

## Assignment 2

Call:

lm(formula = assignment2$timeDelta ~ as.factor(assignment2$cc))

Residuals:

Min 1Q Median 3Q Max

-607.4 -173.2 -149.2 -95.8 7438.7

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 215.516 20.371 10.580 < 2e-16

as.factor(assignment2$cc)1 -51.248 21.612 -2.371 0.01773

as.factor(assignment2$cc)2 -19.297 21.805 -0.885 0.37619

as.factor(assignment2$cc)3 -34.190 22.149 -1.544 0.12269

as.factor(assignment2$cc)4 -17.178 22.772 -0.754 0.45064

as.factor(assignment2$cc)5 -6.669 23.421 -0.285 0.77586

as.factor(assignment2$cc)6 -38.769 24.546 -1.579 0.11424

as.factor(assignment2$cc)7 20.799 26.757 0.777 0.43698

as.factor(assignment2$cc)8 -25.514 29.869 -0.854 0.39301

as.factor(assignment2$cc)9 27.454 33.835 0.811 0.41715

as.factor(assignment2$cc)10 92.903 40.815 2.276 0.02284

as.factor(assignment2$cc)11 63.658 45.326 1.404 0.16019

as.factor(assignment2$cc)12 71.767 56.009 1.281 0.20008

as.factor(assignment2$cc)13 159.137 63.897 2.491 0.01276

as.factor(assignment2$cc)14 132.369 77.226 1.714 0.08653

as.factor(assignment2$cc)15 91.780 101.250 0.906 0.36470

as.factor(assignment2$cc)16 365.242 116.321 3.140 0.00169

as.factor(assignment2$cc)17 -42.864 138.683 -0.309 0.75726

as.factor(assignment2$cc)18 -89.071 156.397 -0.570 0.56901

as.factor(assignment2$cc)19 -173.849 191.004 -0.910 0.36273

as.factor(assignment2$cc)20 195.270 177.003 1.103 0.26995

as.factor(assignment2$cc)21 209.818 191.004 1.098 0.27199

as.factor(assignment2$cc)22 399.848 199.403 2.005 0.04495

as.factor(assignment2$cc)23 -23.266 329.572 -0.071 0.94372

as.factor(assignment2$cc)24 -147.516 465.640 -0.317 0.75140

as.factor(assignment2$cc)25 -152.516 380.375 -0.401 0.68845

as.factor(assignment2$cc)26 -48.849 380.375 -0.128 0.89781

as.factor(assignment2$cc)28 -158.516 658.200 -0.241 0.80969

as.factor(assignment2$cc)29 -192.516 658.200 -0.292 0.76991

as.factor(assignment2$cc)30 -64.516 658.200 -0.098 0.92192

as.factor(assignment2$cc)32 -191.516 465.640 -0.411 0.68086

(Intercept) \*\*\*

as.factor(assignment2$cc)1 \*

as.factor(assignment2$cc)2

as.factor(assignment2$cc)3

as.factor(assignment2$cc)4

as.factor(assignment2$cc)5

as.factor(assignment2$cc)6

as.factor(assignment2$cc)7

as.factor(assignment2$cc)8

as.factor(assignment2$cc)9

as.factor(assignment2$cc)10 \*

as.factor(assignment2$cc)11

as.factor(assignment2$cc)12

as.factor(assignment2$cc)13 \*

as.factor(assignment2$cc)14 .

as.factor(assignment2$cc)15

as.factor(assignment2$cc)16 \*\*

as.factor(assignment2$cc)17

as.factor(assignment2$cc)18

as.factor(assignment2$cc)19

as.factor(assignment2$cc)20

as.factor(assignment2$cc)21

as.factor(assignment2$cc)22 \*

as.factor(assignment2$cc)23

as.factor(assignment2$cc)24

as.factor(assignment2$cc)25

as.factor(assignment2$cc)26

as.factor(assignment2$cc)28

as.factor(assignment2$cc)29

as.factor(assignment2$cc)30

as.factor(assignment2$cc)32

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 657.9 on 36009 degrees of freedom

Multiple R-squared: 0.002374, Adjusted R-squared: 0.001543

F-statistic: 2.856 on 30 and 36009 DF, p-value: 2.991e-07

As we can see in above model, records with 0, 1, 10, 13, 14, 16, and 22 CCed developers are marked significant.

## Assignment 3

Call:

lm(formula = cbo ~ as.factor(release) + lcom, data = vallLcomCbo)

Residuals:

Min 1Q Median 3Q Max

-12.5782 -1.9711 -0.3938 1.2321 26.6368

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 2.774513 0.535320 5.183 2.83e-07 \*\*\*

as.factor(release)26 -0.026606 0.710603 -0.037 0.970144

as.factor(release)27 0.613541 0.725993 0.845 0.398328

as.factor(release)28 0.637773 0.715086 0.892 0.372749

as.factor(release)29 0.619243 0.707225 0.876 0.381537

as.factor(release)30 -1.570400 1.056280 -1.487 0.137518

as.factor(release)31 -1.570400 1.056280 -1.487 0.137518

as.factor(release)32 -1.518297 0.979814 -1.550 0.121675

as.factor(release)33 -1.580797 0.979814 -1.613 0.107096

as.factor(release)34 -2.266555 0.708733 -3.198 0.001443 \*\*

as.factor(release)44 -2.685781 0.691009 -3.887 0.000111 \*\*\*

lcom 0.108037 0.006277 17.212 < 2e-16 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.673 on 732 degrees of freedom

Multiple R-squared: 0.3968, Adjusted R-squared: 0.3877

F-statistic: 43.77 on 11 and 732 DF, p-value: < 2.2e-16

# Threats to Validity

## Construct validity

Construct validity discusses whether the test is measuring what it intended to measure.

In assignment 2 we wanted to know if having more developers collaborating on issues would result in faster issue resolution. What we found was that number of people has negative effect on issue resolution.

In assignment 3 we wanted to know if coupling and cohesion of cpp files improve as new versions are released. Through Understand we noticed that in new versions of Chromium, classes in .cpp files have lower cbo values.

## Internal validity

Internal validity looks at the quality of the test. For example it wants to know if there are confounding measures present in the test. More than one confounding measure in a test is undesirable.

In assignment 2 we had developer experience as a confounding measure.

## External validity

How and whether an experiment would apply to other situation is external validity.

Both assignments discuss situations that are general to any software development project today.

## Reliability

A test must be reliable enough to give us predictable results every time it is run in all situations that it is intended for.

Tests in assignment 3 can be run on every cpp class objects and must return the how cohesive they are and how coupled they are to other classes.

Tests in assignment 2 can only be run on Chromium bug reports. This is because every project has a different structure and format in its bug reports.

# Recommendation and Actions

Number of people collaborating on an issue resolution costs more without producing higher productivity. Keep the number people collaborating on issue resolution at a minimal level.

Frequently refactor your code to make sure cohesiveness and coupling of your code remains at a good level.

# Conclusion

The number of CCed developers on a issue has direct relationship with the average issue resolution time. This could mean this type of collaboration is counterproductive.

Studying CBO metric in different versions of Chromium suggests that coupling between classes is decreasing and maintainability is increasing.