Global and Geographically Work Teams: Understanding the Bug Fixing Process and Bug-Prone Activity Patterns

Daniel Izquierdo Cortázar

dizquierdo@libresoft.es
GSyC/Libresoft, Universidad Rey Juan Carlos

February 3, 2012

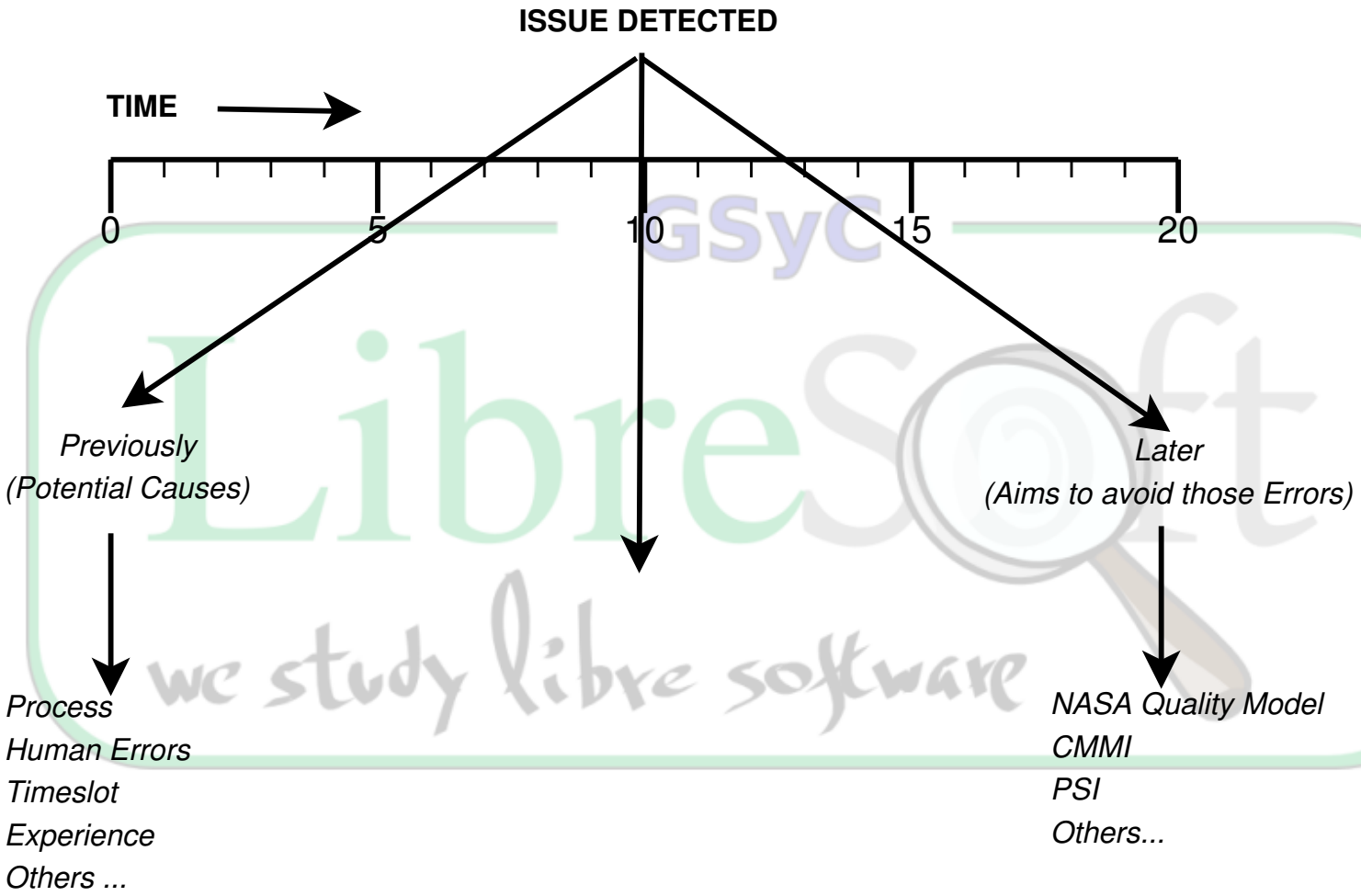
Table of contents

Index

Motivation

- Study in depth the bug life cycle and potential causes of "buggy" changes in the source code due to human related factors.

Motivation


ISSUE DETECTED
TIME
0
5
10
15
20
Previously
(Potential Causes)
Process
Human Errors
Timeslot
Experience
Others ...
Later
(Aims to avoid those Errors)
NASA Quality Model
CMMI
PSI
Others...

Useful for: Software Maintenance

- Software maintenance may be defined as changes done in the source code after delivering a product; and those changes can be classified as:
  1. Adaptative: adapt to new environment
  2. Perfective: improve performance or maintenance
  3. Corrective: correct discovered problems
  4. Preventive: prevention of potential issues
- These changes reach in some cases a **70% of the final cost** of a product

Useful for: Software Development Process Models

- This is a process to continually improve the development process (e.g. CMMI)
- Based on three steps:
  1. Analyze defects to trace the **root causes**
  2. Suggest preventive actions to eliminate the defect root causes
  3. Implement the preventive actions

Research Questions

- The whole approach is based on the analysis and understanding of the bug life cycle process.
- So far, studies with bugs were mostly focused on the analysis of data obtained from the BTS or focused on predicting future bugs based on the current state or evolution of the source code.

Research Questions

- However, this thesis has aimed to go a step further and analyse the point in time when a bug was *involuntary* introduced in the source code.
- Thus, there exist four main steps:
  - Involuntary introduction of a bug (SCM)
  - Discover of that issue in the source code and process of opening a bug report (BTS)
  - Fixing action, closing the bug report and changing the source code (BTS and SCM).

Research Questions: bug life cycle

- Thus, let us **study the bug life cycle** adding the action of involuntary introducing an issue in the source code (*bug seeding commit*).

Research Questions: causes of buggy changes

- Once this is understood, it is possible to study potential causes of bugs introduction.
- This thesis is focused on two of them:
  - **Causes due to experience of developers** (it is expected to find that the higher the expertise, the lower the ratio of buggy changes).
  - **Causes due to time issues** (deadlines, tiredness or other issues may provoke errors)

Research Questions: summarizing

- Bug life cycle, focusing on the bug seeding commit actions
- +
- Potential causes of bug introduction: **experience** and **time of the day**.

Index

Initial approach

- The whole approach is based on the analysis of the source code management systems
- Usually, they provide a tool to analyse the differences between two revisions
- And in addition, all of the development information is stored by them: author, date of commit, files and lines added, modified or removed

Initial approach: Mercurial and Mozilla

- Some considerations about the distributed SCM Mercurial:
  - The concept of committer does not exists, only authors do
  - The time of commit specifies the local time and timezone of each author and not a centralized server (such as CVS or SVN).
- Some considerations about the case study, Mozilla:
  - Policy defined for core projects: double check of changes (Mozilla Central, Mobile Browser, ...)
  - Part of the source code is based on XUL

Method

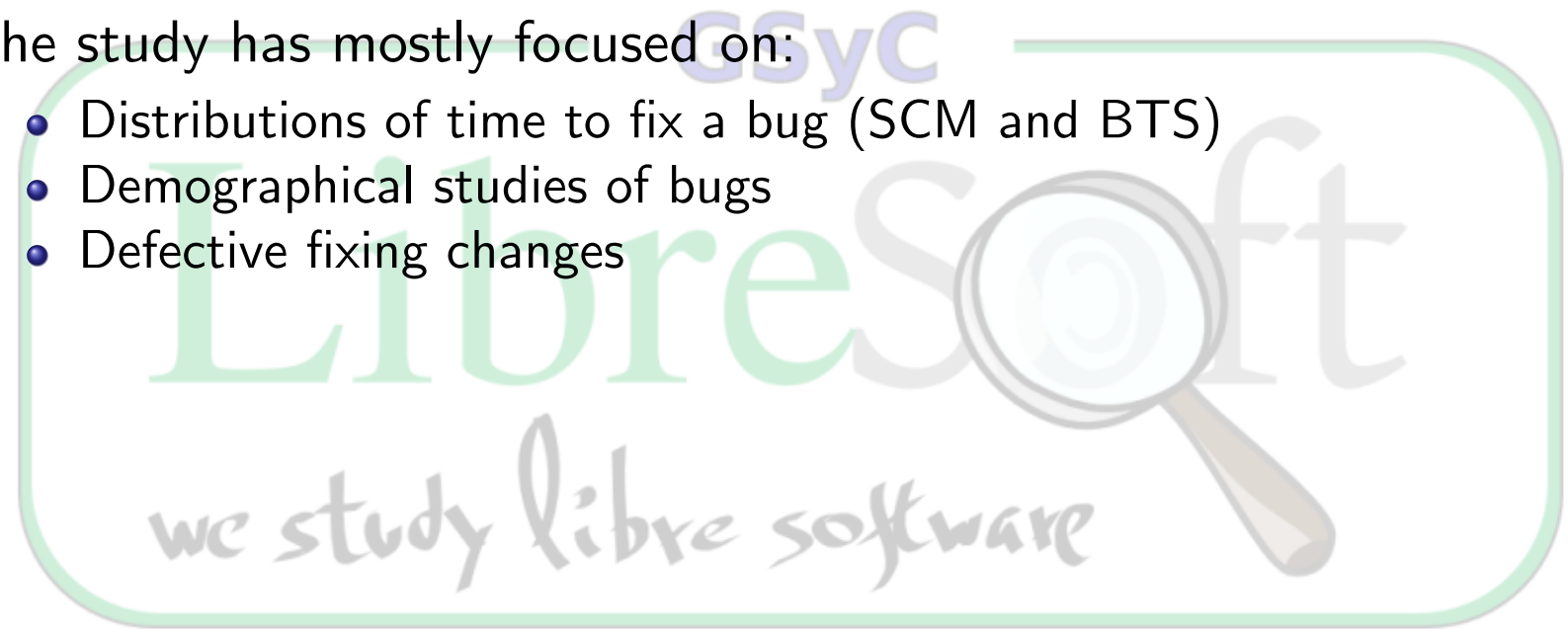
- Thus, how is the information retrieved?
- Three main steps:
  - Identification of fixing commits: by means of the log message
  - Identification of modified and removed lines in those fixing commits: by means of the diff changes
  - Identification of seeding commits, following the life of the lines modified or removed in a fixing commit (SZZ algorithm): by means of BlameMe
- For this purpose BlameMe was created:
  `http://git.libresoft.es/`

Method

- Thus, it is known when a bug was fixed
- and the commits where that bug was seeded
- Let us play now with the data

Index

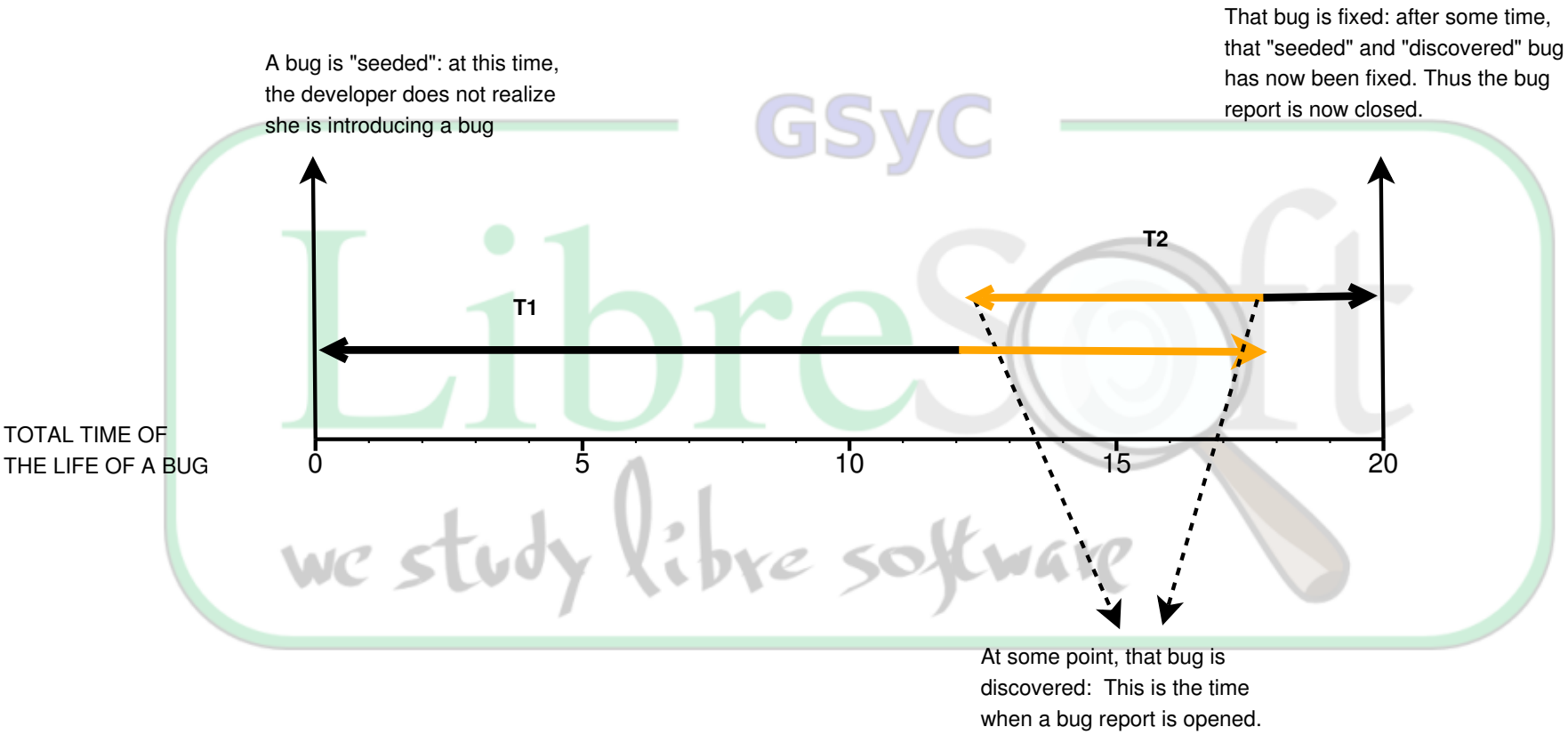
- The study has mostly focused on:
  - Distributions of time to fix a bug (SCM and BTS)
  - Demographical studies of bugs
  - Defective fixing changes



Bug life cycle: distributions and time to fix a bug

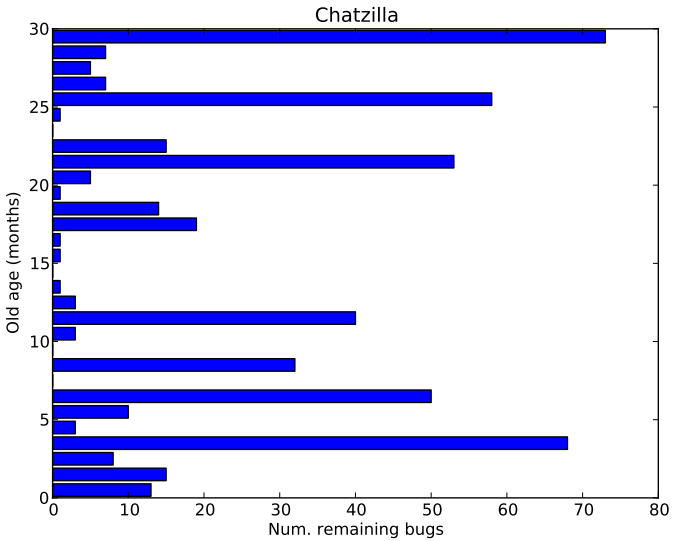
- Distributions are statistically different between the SCM and the BTS
- If a project is declared as "core", the time to fix a bug decreases by 10X.
- However,
  - In some cases, bug reports have appeared in time before the bug-seeding activity.
  - this indicates that the bug-seeding commit is not exactly the correct one, but at least this delimits a low barrier in time
  - In some cases this number reaches a 45% of the total commits studied depending on the community.

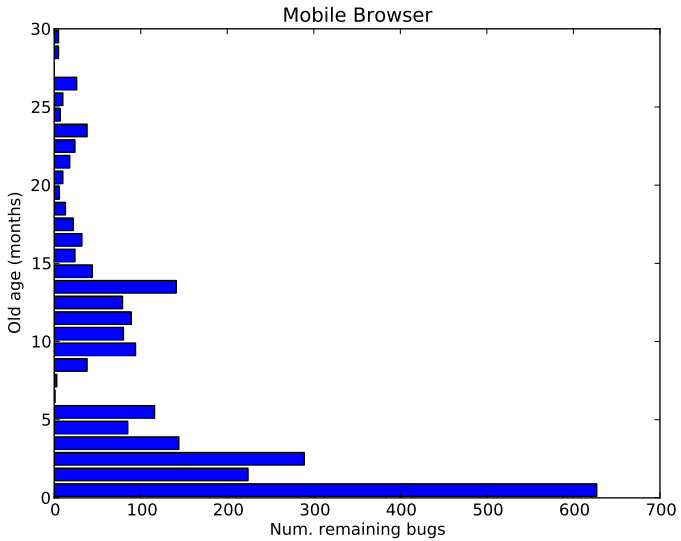
Bug life cycle: distributions and time to fix a bug



Bug life cycle: bugs demography

- Expected pyramid of population: old bugs tend to disappear
- However:
  - Projects defined as "core" show an expected pyramid
  - Projects defined as "non-core" show an unexpected pyramid





Bug life cycle: bugs demography

- And the following question: how is the ratio of decrease of bugs?:
  - In most of the cases, the bug finding ratio follow a lineal or exponential distribution
  - The best goodness of fit for the overall dataset is lineal

Bug life cycle: defective fixing changes

- These are defined as those fixing changes that later become an issue
- It has been detected up to a 87% of the changes become defective (Thunderbird)
- However, the most common value goes around a 55% of the fixing changes become "buggy" again

Causes: Experience and bug seeding ratio

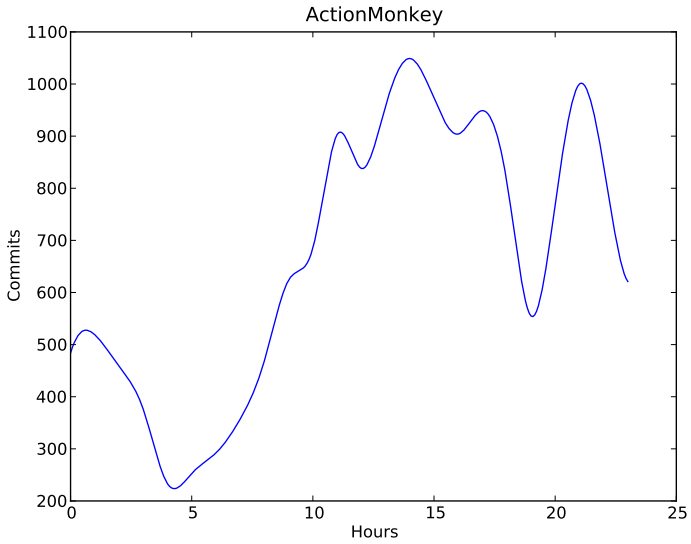
- Expertise is quantitatively measured as:
  - Number of commits
  - Number of fixing commits
  - Ownership of the source code (territoriality)
- In addition the concept of *bug seeding ratio* is introduced:
  - $BSR = \frac{Number\ of\ bug\ seeding\ commits}{Number\ of\ commits} * 100\%$

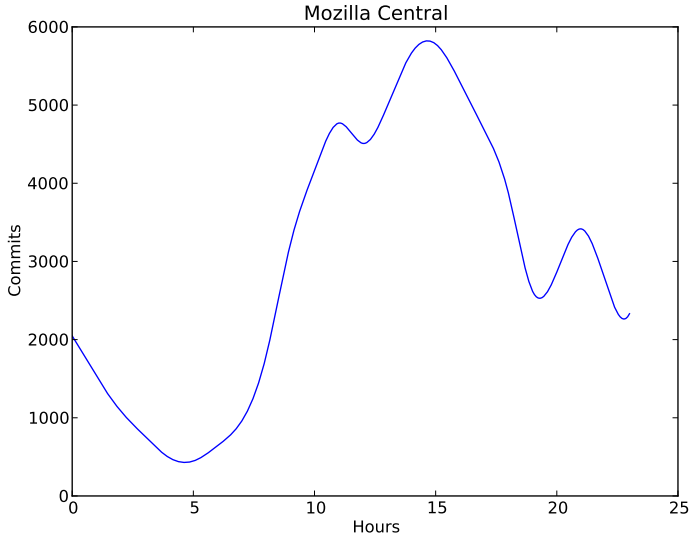
Causes: experience and bug seeding ratio

- Surpringsinly, no relationship was found between any type of experience and bug seeding ratio.
- Although, there is a high correlation between number of commits and number of fixing commits as a metric.

Causes: general activity in a 24 hours framework

- A double hump has been observed. The first one following a 9-18 timeframe of activity and a second one during evening-midnight timeframe.
- Distribution of activity among projects are different if compared, thus aggregated data is not valid





Causes: Time of the day

- Distribution of bug-seeding ratios are also different if divided into the two humps of activity
- Although the bug-seeding ratio does not change significantly after office hours, there is an increase in the bug-seeding ratio, pointing this as potentially more buggy

Causes: time of the day

- Out of the common timeframe of the day when a developer is more active, there is an increase in the bug-seeding ratio

Causes: experience and time of the day

- There is not an actual difference between these two group of developers

Index

Conclusions

- Although there are threats to validity that may bias the results, it is possible to study the *roots* of the errors what may help to improve the development and maintenance process
- Introducing the concept of bug seeding ratio helps to better understand the bug life cycle and how the fixing process is being carried out by developers

Conclusions

- In addition, specific actions could be taken to avoid the effect of human related factors:
  - Extra care when changing source code out of typical timeframes of activity
  - Peer review of changes after *office time*

Questions?

Thanks for your attendance!
Questions?
–

Daniel Izquierdo Cortázar
dizquierdo@libresoft.es
GSyC/LibreSoft - Universidad Rey Juan Carlos

Slides: `http://www.scribd.com/dicortazar/`