# BugMaps-Granger: A Tool for Causality Analysis between Source Code Metrics and Bugs

**César Couto[1,2], Pedro Pires[1], Marco Túlio Valente[1], Roberto S. Bigonha[1], Andre Hora[3], Nicolas Anquetil[3]**

[1]Department of Computer Science – UFMG – Brazil

[2]Department of Computing – CEFET-MG – Brazil

[3]RMoD Team – INRIA – France

{cesarfmc,ppires,mtov,bigonha}@dcc.ufmg.br,{firstName.lastName}@inria.fr

***Abstract.*** *Despite the increasing number of bug analysis tools for exploring bugs in software systems, there are no tools supporting the investigation of causality relationships between internal quality metrics and bugs. In this paper, we propose an extension of the BugMaps tool called BugMaps-Granger that allows the analysis of source code properties that caused bugs. For this purpose, we relied on Granger Causality Test to evaluate whether past changes to a given time series of source code metrics can be used to forecast changes in a time series of defects. Our tool extracts source code versions from version control platforms, generates source code metrics and defects time series, computes Granger, and provides interactive visualizations for causal analysis of bugs. We also provide a case study in order to evaluate the tool.*

## 1. Introduction

A number of tools for software analysis has been proposed recently to improve software quality [Nierstrasz et al. 2005, Hovemeyer and Pugh 2004, Wettel 2009]. Such tools use different types of information about software system structure and history. Basically, they can be used to analyze software evolution, manage the quality of the source code, compute metrics, analyze coding rules, etc. In general, such tools help software maintainers to understand large amounts of data coming from source code repositories.

Particularly, there is a growing interest in software analysis tools for exploring bugs in software systems [Hora et al. 2012, D'Ambros and Lanza 2010, Sliwerski et al. 2005]. Such tools help maintainers understand the distribution, the evolutionary behavior, the lifetime, and the stability of bugs. Basically, they work by retrieving history data from bug tracking and version control platforms, by mapping bugs to defects in source code modules, and by processing data to extract and reason about bugs.

Despite the increasing number of bug analysis tools, they typically do not provide mechanisms for assessing the existence of correlations between the internal quality of a software system and the occurrence of bugs. To the best of our knowledge, there are no bug analysis tools that investigate and highlight possible causes for the occurrence of bugs in source code modules of a system. More specifically, there are no tools designed to infer causal relations between source code properties (as measured by source code metrics) and defect occurrences in object-oriented classes.

In this paper, we propose and describe the BugMaps-Granger tool—an extension of the BugMaps tool [Hora et al. 2012]—that supports detection of causal relations between source code metrics and bugs. Basically, this tool provides mechanisms to retrieve software repositories data, to compute source code metrics, to generate source code metrics and defect time series, and to infer causal relations between source code properties and defects. Moreover, BugMaps-Granger provides visualizations for decision support. More specifically, to identify causal relations on the time series of source code metrics and defects, our tool relies on the Granger Causality Test [Granger 1981]. Such test evaluates whether past changes to a given time series of source code metrics can be used to forecast changes in a time series of defects. The proposed tool has the following features:

- The tool automatically extracts source code models of the target system from its version control platform in predefined time intervals.
- The tool generates time series of twelve source code metrics and time series with the number of defects in each class of the target system.
- The tool applies the Granger Test considering the metrics and defects time series to identify causal relations.
- The tool integrates models extracted from the source code with models that represent the number of bugs.
- The tool provides a set of interactive visualizations that supports software maintainers in answering questions such as: (a) What is the number of bugs in a module? (b) What are the modules usually involved in bug-fixing? (c) What are the source code properties that Granger-caused bugs in a given module? and (d) What is the lifetime of a bug?

## 2. BugMaps-Granger: Overview

The execution of the BugMaps-Granger tool is divided into two stages: preprocessing and visualization. The preprocessing stage is responsible for extracting source code models, creating time series, and applying the Granger Test to identify possible causal relations between source code metrics and bugs. In the visualization stage, the user can interact with the tool. In this stage, the user can localize the most defective classes of the target system and visualize the source code properties that Granger-caused bugs.

The system requirements to execute the tool are: (i) Java-based systems; (ii) identifiers and creation dates of bugs stored in a csv file; and (iii) URL of the version control platform (SVN or GIT). Figure 1 shows the architecture of the BugMaps-Granger, which includes the following modules:

**Model Extraction:** This module receives as input the URL associated to the version control platform of the target system and a time interval used in the analysis of the bugs. To extract the source code models, this module performs the following sub-procedures: (a) it extracts the source code versions from the version control platforms in intervals of bi-weeks; (b) it removes test classes. In order to remove test classes, this module removes the directories and subdirectories whose name starts with the words “Test” or “test”; and (c) it parses the source code versions and generates MSE files using the VerveineJ tool[1]. MSE is the default file format supported by the Moose platform[2] to persist source code models.

[1] http://www.moosetechnology.org/tools/verveinej

[2] http://www.moosetechnology.org.

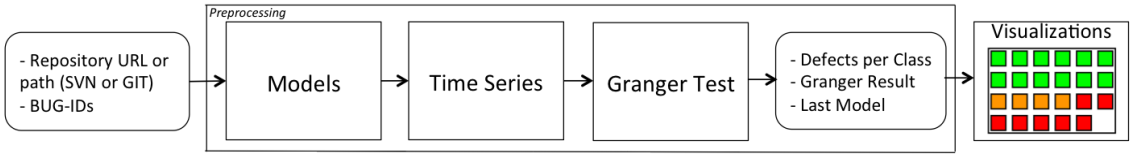



**Figure 1. BugMaps-Granger's architecture**

**Time Series Creation:** To create the time series of source code metrics, this module receives as input the models extracted by the previous module. For each class of each extracted model, the module relies again on the Moose platform to compute twelve source code metrics including six CK metrics (proposed by Chidamber e Kemerer [Chidamber and Kemerer 1994]) and five others, such as lines of code, fan-in, fan-out etc. To create the time series of defects for each class, this module receives as input a csv file containing the BUG-IDs and the bug creation dates collected from the bug tracking platforms of the target system. Basically, this module maps the bugs to their respective commits using the BUG-ID. Next, the files changed by such commits are used to identify the classes changed to fix the respective bugs (i.e., the defective classes). More details can be checked in [Couto et al. 2012].

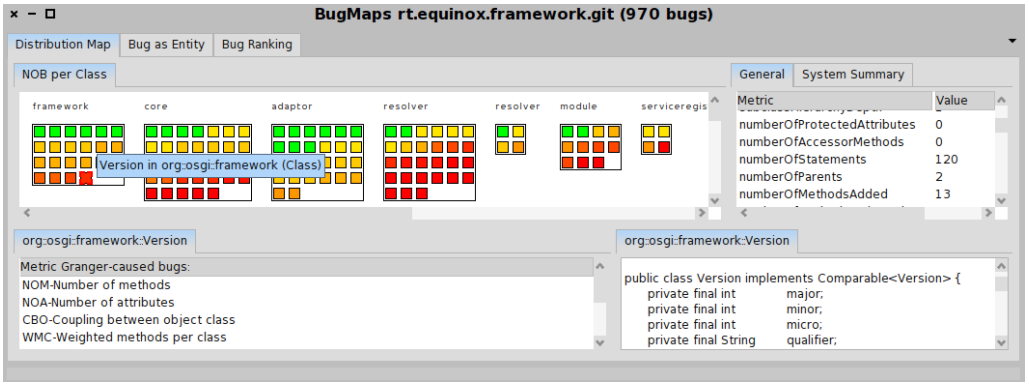
**Granger Test Module:** This module applies the Granger Causality Test considering the metrics and defects time series. More specifically, Granger is responsible for identifying causal relations on time series of source code metrics and defects. To apply the test, this module relies on the function *granger.test()* provided by the *msbvar* package from the R statistical system. More details about the Granger Test and the algorithm used to identify causal relations can also be checked in [Couto et al. 2012].

**Visualization Module:** This module receives as input a file containing the bugs mapped to their respective classes and the result of the Granger Test, a model extracted from the last source code version, and the source code itself of the system under analysis. From such informations, the module provides interactive visualization browsers. Two browsers are used for analysis, the first one deals with the classes, the number of bugs, and the Granger results of the system under analysis (called Granger browser) and the second one deals with the complexity of the bugs (called Bug as Entity browser). Such browsers are implemented in Pharo (a Smalltalk dialect), using visualization packages provided by the Moose Platform.

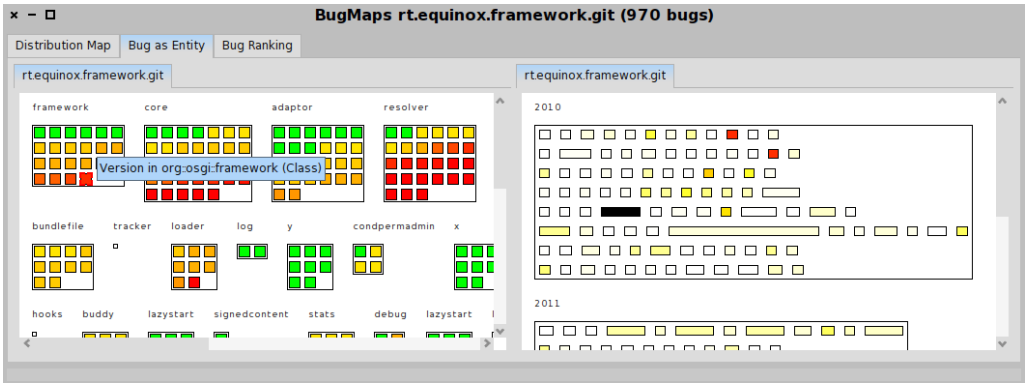
Figure 2(a) shows the Granger browser, which has four panes: visualization of classes and packages (top left), measures (top right), Granger results (bottom left), and source codes (bottom right)[3]. Metrics, source codes, and Granger results are updated according to the selected class in the classes and packages pane. Figure 2(b) shows the Bug as Entity browser which is composed by two panes: visualization of classes and packages (left pane) and bugs (right pane). When a defective class is selected, the bugs in this class are colored in black. In contrast, when a bug is selected, the classes changed to fix this bug also are colored in black.

The visualizations provided by BugMaps-Granger are based on Distribution Map, a generic technique to reason about the results of software analysis and to investigate how

[3] Since most of our visualizations make use of colors, we provide high-resolutions versions of these figures in a companion website (http://java.labsoft.dcc.ufmg.br/bugmaps).




(a) Granger browser




(b) Bug as Entity browser

**Figure 2. BugMaps-Granger's browsers**

a given phenomenon is distributed across a software system [Ducasse et al. 2006]. Using a Distribution Map three metrics can be displayed through the height, width and color of the objects in the map. In our maps, rectangles represent classes or bugs and containers represent packages.

## 3. Case Study: Eclipse Equinox Framework

In order to evaluate BugMaps-Granger, we provide an example of use involving data from the Eclipse Equinox Framework system collected during three years (2010-01-01 to 2012-12-28). First, the tool extracted 79 source code versions in intervals of bi-weeks, including 417 classes. Next, for each class, the tool created twelve time series of source code metrics (one for each metric) and one time series of defects. Finally, for each pair of time series (source code metrics and defects), the tool applied the Granger Test to identify causal relations. We analyzed the Equinox Framework system according to the proposed visualizations, which are showed in the Figures 3 and 4.

**Granger:** In this map, the objects are the classes of the target system. The color of a class represents the number of bugs detected through its history ranging from green to red (the closer to red, more bugs the class had in its history). By selecting a defective class, the bottom pane is updated showing the source code metrics that Granger-caused bugs in this class. Figure 3 provides an overview of the distribution of the bugs in the Eclipse Equinox Framework system. We can observe that the `resolver` package contains a significant number of classes with bugs. Moreover, for the class

`org.eclipse.osgi.internal.resolver.StateImpl`, we can observe that the source code metrics that Granger-caused bugs were CBO (Coupling between object class), WMC (Weighted methods per class), and RFC (Response for class).

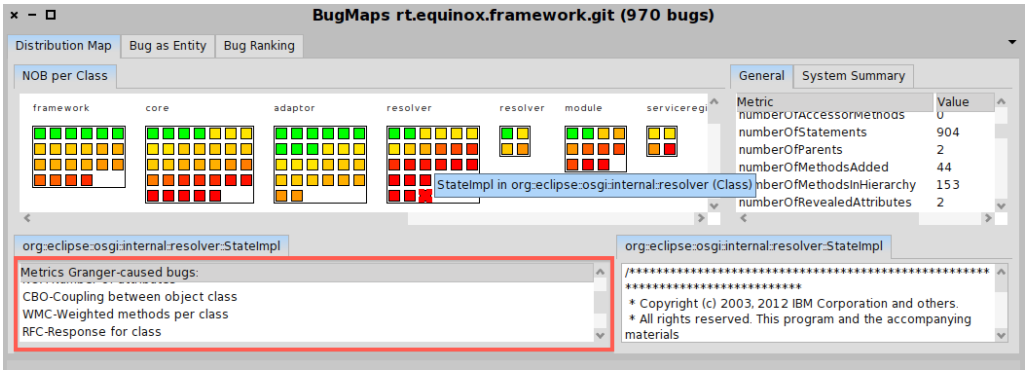



**Figure 3. Granger result per class**

**Bug as Entity:** This map represents bugs instead of classes. The color of a bug represents its lifetime, i.e., the number of days it remained opened. Blue denotes a bug that was still open at the end of the time period considered. In the analysis, white denotes a bug that was open for a short time. Simirlarly, yellow is a bug that was open up to 3 months, and red is a bug that was opened for more than 3 months. The width of a bug representation denotes its complexity, measured as the number of classes changed to fix the bug. Bugs are sorted according to the date they were created. Figure 4 shows the bugs of the Equinox Framework created in 2010. We can observe that all bugs from 2010 were fixed, only two bugs remained openfor more than 3 months (bugs going to red), and that complex bugs (long width) are dispersed in time. In a detailed analysis, we can note that the bug 324774 is quite complex. In this specific case, 191 classes were changed to fix this bug. According to this bug description: *"Update Framework source code for generics"*, we can conclude that this bug is related to a complex task that involved a major change in the system.

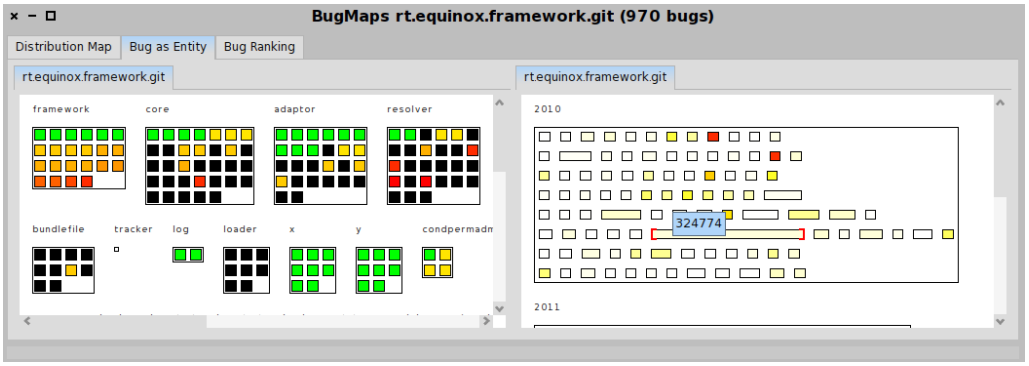



**Figure 4. Bug as entity**

## 4. Related Work

Churrasco is a web-based tool for collaborative software evolution analysis [D'Ambros and Lanza 2010]. The tool automatically extracts information from a variety of software repositories, including versioning systems and bug management systems. The ultimate goal is to provide an extensible tool that can be used to reason

about software evolution under different perspectives, including the behavior of bugs. In contrast, BugMaps-Granger provides information about source code properties (as measured by source code metrics) that caused bugs, at least according to Granger. Hatari [Sliwerski et al. 2005] is a tool that provides views to browse through the most risky locations and to analyze the risk history of a particular location in a system at the level of lines of code. On the other hand, BugMaps-Granger works at the level of classes and packages.

## 5. Conclusions

In this paper we proposed a tool to support the extraction and statistical and visual analysis of bugs stored in bug-tracking systems. The tool extracts time series of defects from such systems and allows the visualization of different bug measures including the source code properties that Granger-caused bugs. Its ultimate goal is to indicate the classes of the target system that are more subjected to bugs and the source code metrics that statistically can be used to antecipate the occurrence of bugs in such classes. The proposed tool is publicly available at: **http://java.labsoft.dcc.ufmg.br/bugmaps**.

**Acknowledgment:** This work was supported by FAPEMIG, CNPQ and INRIA.

## References

Chidamber, S. R. and Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493.

Couto, C., Silva, C., Valente, M. T., Bigonha, R., and Anquetil, N. (2012). Uncovering causal relationships between software metrics and bugs. In *European Conference on Software Maintenance and Reengineering (CSMR)*, pages 223–232.

D'Ambros, M. and Lanza, M. (2010). Distributed and collaborative software evolution analysis with churrasco. *Science of Computer Programming*, 75(4):276–287.

Ducasse, S., Girba, T., and Kuhn, A. (2006). Distribution Map. In *International Conference on Software Maintenance (ICSM)*, pages 203–212.

Granger, C. (1981). Some properties of time series data and their use in econometric model specification. *Journal of Econometrics*, 16(6):121–130.

Hora, A., Couto, C., Anquetil, N., Ducasse, S., Bhatti, M., Valente, M. T., and Martins, J. (2012). Bugmaps: A tool for the visual exploration and analysis of bugs. In *European Conference on Software Maintenance and Reengineering (CSMR Tool Demonstration)*.

Hovemeyer, D. and Pugh, W. (2004). Finding bugs is easy. *SIGPLAN Notices*, 39(12):92–106.

Nierstrasz, O., Ducasse, S., and Gĭrba, T. (2005). The story of Moose: an agile reengineering environment. In *European Software Engineering Conference (ESEC)*, pages 1–10.

Sliwerski, J., Zimmermann, T., and Zeller, A. (2005). Hatari: Raising risk awareness. In *European Software Engineering Conference (ESEC)*, pages 107–110.

Wettel, R. (2009). Visual exploration of large-scale evolving software. In *International Conference on Software Engineering (ICSE)*, pages 391–394.