BugCrawler: Visualizing Evolving Software Systems

Marco D'Ambros and Michele Lanza
Faculty of Informatics
University of Lugano, Switzerland

Abstract

Software evolution is aimed at analyzing and understanding the present state of a software system and at predicting its future development. This knowledge supports reverse engineering activities since it allows the analyst to infer causes of problems in the system and to detect components which need to be restructured. However, effectively using evolutionary information is challenging because it typically comes in large amounts, especially when several years of evolution are considered. Techniques are needed to break down the data quantity and complexity.

BugCrawler *is a language independent tool which supports software evolution and reverse engineering*[1]*. It is based on a combination of software metrics and interactive visualizations. BugCrawler integrates structural information computed from the source code with evolutionary information retrieved from CVS log files and Bugzilla problem reports. It has been validated on several large projects.*

1 Principles

BugCrawler supports the analysis of the evolution of software systems by showing them under different perspectives [1]. It provides visualizations targeted at answering reverse engineering questions such as:

- *Commit information*: Which are the parts of the system with the most intense development? Which are the stable/dead parts of the system? Which parts have grown/shrunk?
- *Author information*: How many developers worked on the entity? How was the effort distributed among them? Is there an "owner" of the entity?
- *Bugs*: Are there components affected by many bugs? Which are the bugs affecting many components?
- *Conceptual entities*: How has the entity evolved over time? When was it introduced in the system? When did it generate many bugs? When did it have intense development? Which phases did it go through?
- *Logical coupling*: Which artifacts are most coupled?

BugCrawler provides visualizations "in the large" and "in the small". Visualizations in the large are used to obtain an overview of the system in terms of modules, modules evolution over time and module dependencies. The user can then use visualizations in the small to study the internal structure of individual modules, going from directories to single commits and bugs. The views in the small cover evolutionary aspects like effort distribution among developers and over time, stability/instability of components, *etc.*

In [3] we present a list of visualizations supporting the reverse engineering of a system. We also discuss a methodological approach describing how to use, combine and interpret the views to answer the questions mentioned above.

2 BugCrawler at Work

Figure 1 shows the three main parts of the tool. The toolbar (annotated with "A") contains the most important actions such as creating/spawing the views, interacting with the view (cut/copy/paste figures, group/ungroup, change layout), designing new visualizations and connecting to the database containing the evolutionary data. The part annotated as "B" displays information about the entity under the pointer: The type, the name, the metrics mapping (*e.g.,* number of bugs mapped on the width) and the metrics value. The main surface contains the visualization and allows the user to interact with it by means on contextual menus.

In the figure we see BugCrawler at work: Window 1 shows author information for a directory hierarchy of Gimp. Each author is represented by a color, and the area of each colored rectangle is proportional to the work done by the corresponding author. Window 2 shows the contents, in terms of files, of the selected directory (annotated with an arrow). For further details on the views see [1, 3, 4].

[1] The tool is a major extension of CodeCrawler [5]. The main difference is that CodeCrawler supports reverse engineering by visualizing a single version of a system. BugCrawler supports software evolution by showing all historical information together with the last version of the system.

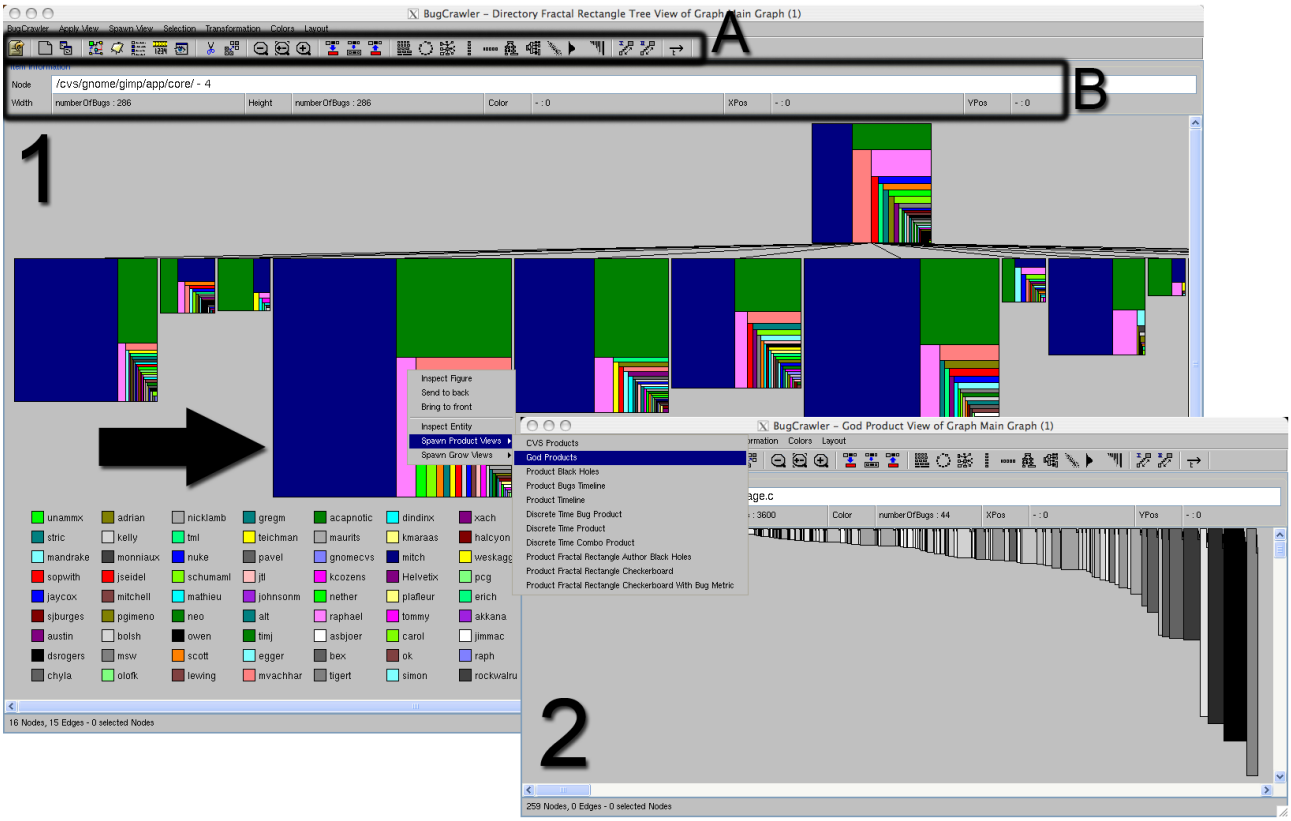

Figure 1. BugCrawler visualizing author information on a directory hierarchy of Gimp (window 1). Each directory is represented as a Fractal Figure [4]. Window 2 shows the contents (in terms of files) of the selected directory (annotated with an arrow).

The tool is interactive: It is possible to navigate between the views and to inspect every entity. When these entities are files the source code can be read on-the-fly. The tool allows the user to select a group of entities and to change their visual representation, *i.e.,* change the figures representing them. To decrease the level of granularity, new views can be applied on the contents of a visualized entity.

Since every software system has its own characteristics, a predefined set of views could not be the best solution for the analysis of all the systems. In BugCrawler we address this problem by adding to the predefined views the possibility to create user-defined visualizations. The user can (i) choose which entities visualize, (ii) which figures and layouts use and (iii) define the metric mappings.

We have evaluated the effectiveness of BugCrawler to support the understanding of software evolution by applying it on several large software systems, such as Mozilla [1–4], Gimp [3], Apache [2], Gcc [2].

References

[1] M. D'Ambros. Software archaeology - reconstructing the evolution of software systems. Master thesis, Politecnico di Milano, Apr. 2005.

[2] M. D'Ambros and M. Lanza. Software bugs and evolution: A visual approach to uncover their relationship. In *Proceedings of CSMR 2006 (10th IEEE European Conference on Software Maintenance and Reengineering)*, pages 227 – 236. IEEE Computer Society Press, 2006.

[3] M. D'Ambros and M. Lanza. Reconstructing the evolution of software systems. In *Submitted to CSMR 2007 (11th IEEE European Conference on Software Maintenance and Reengineering)*, 2007.

[4] M. D'Ambros, M. Lanza, and H. Gall. Fractal figures: Visualizing development effort for cvs entities. In *Proceedings of Vissoft 2005 (3rd IEEE International Workshop on Visualizing Software for Understanding)*, pages 46–51, 2005.

[5] M. Lanza. Codecrawler — lessons learned in building a software visualization tool. In *Proceedings of CSMR 2003 (7th IEEE European Conference on Software Maintenance and Reengineering)*, pages 409–418. IEEE Computer Society Press, 2003.