The GZoltar Project: A Graphical Debugger Interface

André Riboira and Rui Abreu

Faculty of Engineering, University of Porto
Rua Dr. Roberto Frias, s/n 4200-465 Porto, Portugal
andre.riboira@fe.up.pt, rui@computer.org
http://www.gzoltar.com/

Abstract. Software debugging is one of the most time-consuming and expensive tasks in software development. There are several tools that contribute to this process to become faster and more efficient, but are not integrated with each other, nor provide an intuitive interface. These tools can be integrated to create an IDE plug-in, which gathers the most important debugging information into one place. GZoltar is a new project to create that missing plug-in. The main goal of GZoltar project is to reduce debugging process time and costs.

Keywords: Debug, Spectrum-Based Fault Localization, Code Dependency Graphs, Project Hierarchy Trees.

1 Introduction

In software development, debugging (localization and correction of software faults) is one of the most expensive tasks [1,3]. Although existing automatic debugging tools are quite powerful, some developers tend to use basic manual debug functionalities that their Integrated Development Environment (IDE)'s offer. There are plenty of tools to help developers to find the faults of their software. Unfortunately, those tools tend to not be integrated with each other, and the developer does not have a place to get all the information he wants at the same time. Moreover, debugging tools traditionally provide an unattractive output and sometimes also rather confusing, especially with regard to large software projects. Code coverage tools, such as Zoltar [1], allows developers to know which lines of code were executed in a given test. Usually, those tools use source code lines highlight in different colors to show if a line was executed or not. Code dependencies graphs [2] allows the creation of a graph of the entire project, in which the nodes of the graph represent the different modules of the project and the links represent the dependencies between these modules. This information allows an overview of the project and makes it possible to analyze fault propagations between modules. With tree-mappings is possible to have a clear understanding of the different components of the project, and the way they are related hierarchically. This is useful because we can easily navigate through the different levels of detail on our software projects, and have a clearer picture

L. Bottaci and G. Fraser (Eds.): TAIC PART 2010, LNCS 6303, pp. 215–218, 2010.

of sub-modules of a given module. Furthermore, there are tools available that automatically calculate the failure probability of each software module. Lately there has been a clear interest in developing tools for automatic debugging. These tools are mainly based on Model-based software debugging (MBSD) or Spectrum-based fault localization (SFL) [3]. These tools are all very useful, but may work much better if they collaborate with each other. Integrating some of these functionalities would give to the developer a very powerful tool for all debugging processes. But although the results of these tools would be very useful, they should also be presented in a way that the developer can quickly assimilate all the information, and navigate through it intuitively. The integration of these features will have a better result if it is done in an IDE.

2 Zoltar: A Toolset for Automatic Fault Localization

On automatic debug field, SFL techniques are shown to have better performance than those of MBSD [3]. Zoltar is a tool that implements SFL [3] and can predict, with a high success rate, the localization of software faults. Zoltar hosts a range of spectrum-based fault localization techniques featuring Barinel [3]. The toolset provides the infrastructure to automatically instrument the source code of software programs to produce runtime data, which is subsequently analyzed to return a ranked list of diagnosis candidates [1]. Despite the usefulness of this tool, its output can be difficult to analyze because it is mainly textual. As output we obtain a listing of code blocks with the failure probability of each of these blocks. In a long project this list can become quite confusing and the navigation can be particularly difficult.

3 GZoltar: A Graphical Debugger Interface

This paper proposes a new project, that uses Zoltar's output, as well as a generated code dependency graph and a project hierarchy tree as input. The main goal of this project is to build a useful graph where the developer could not only better understand the organization of his project, but also the module dependencies and failure probability of each module. This rich information could then be used to ease the debugging process, therefore reducing the overall debugging time. The startup view of this GZoltar tool would be a tree with the software hierarchy, so that the developer could have a general view of all the project. User would be able to navigate through that tree, like zooming in and out in the different levels of detail. That navigation would give the developer a sense of depth, that would help him to more clearly understand the location of each module in the whole system. Please see Figure 1 for a prototype. All this modules would be colored differently to represent their failure probability. At all times, the developer would be able choose to see the dependency graph of a given module. This would help the developer to analyze the possibility of having errors propagated by the modules due to its dependency.

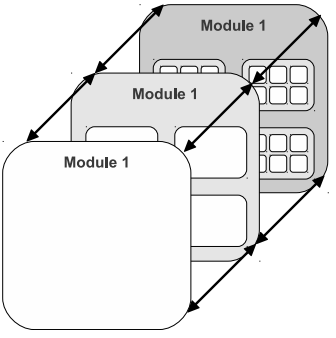

Fig. 1. Sense of depth in GZoltar Hierarchical View

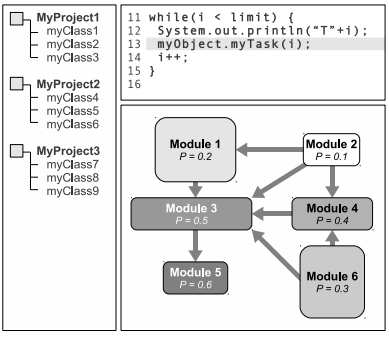

Fig. 2. GZoltar Code Dependency Graph View in IDE integration

Sometimes the origin of a fault is not in the most affected module itself, but is inherited of another module that it depends on. GZoltar tool could be integrated on a popular IDE like Eclipse [4] for developer comfort, that could reduce the learning curve of the tool and increase its use. A GZoltar plug-in would bring a useful tool to all Java coders. Such a tool could allow the programmer to quickly identify a faulty module in a global overview, and expand that module to see which sub-modules are problematic. Even if the failure is not implicit to that given module, he is able to analyze the module dependencies to try to find out where is the failure origin. Please see Figure 2 for a prototype. Surely the time and costs devoted to debugging would be reduced considerably with the use of this tool. We can use available open source tools, like EclEmma [5] to get the project code coverage to be used as Zoltar input, recode Zoltar in Java, to provide a better integration in Eclipse, and use a tool like PDE Incubator Dependency Visualization [6] to provide the code dependency graph, and a tool like Tree Views for Zest [7] to provide the project treemap. An OpenGL view was also taken into consideration, to provide a powerful navigation though all the project modules. This view can provide a 3D map of the application, where all the nodes features like its size, color, transparency and position would have a special meaning, like the number of code lines of a given module, its failure probability, the number of times it was executed or its detail level. GZoltar will be released as an open source tool, so anyone can contribute to it in the future. This also gives greater versatility to any organization that wants to implement GZoltar. The purpose of this tool is to provide a powerful and integrated view of a software project, with a good navigation system and the indication of the fault probability of each module, allowing the programmer to quickly find and correct software faults. Being a free open source tool that is able to free the programmer from consuming steps of finding software faults, we believe that it will certainly help to reduce the overall cost of debugging process.

4 Conclusions

Software debugging is the most time-consuming and expensive phase of the software development cycle [1,3]. There are several tools that contribute to the debugging process to become faster and more efficient, but unfortunately these tools are not integrated with each other, nor provide an intuitive interface for their use is the mass. These tools can be integrated to create a single tool, which allows the developer to obtain all the needed information in one place, preferably inside his IDE. From the output of Zoltar as base, and adding a code dependency graph and a hierarchy tree of the project, we can build a very useful tool, allowing a pleasant way to pass to the developer not only the structure of his project but also the dependencies of each module and their failure probability. Navigation through all of this information should also have a special attention. In long projects, visualization is particularly important because it should allow the developer to, at the same time, have an overview of the whole project, but also be able to achieve high levels of detail to identify exactly where the location of the faults of his project. This tool could be created as an Eclipse plug-in [4] to increase the developer convenience. A plug-in like this could greatly improve the debugging process, reducing time and cost of it.

References

1. Janssen, T., Abreu, R., van Gemund, A.J.C.: Zoltar: A toolset for automatic fault localization. In: International Conference on Automated Software Engineering, New Zealand (2009)
2. Balmas, F.: Displaying dependence graphs: a hierarchical approach. In: Workshop on Analysis, Slicing and Transformation, Germany (2001)
3. Abreu, R.: Spectrum-based Fault Localization in Embedded Software. PhD Thesis, Delft University of Technology, Netherlands (2009)
4. Eclipse.org, `http://www.eclipse.org/`
5. EclEmma - Java Code Coverage for Eclipse, `http://www.eclemma.org/`
6. PDE Incubator Dependency Visualization, `http://www.eclipse.org/pde/incubator/dependency-visualization/`
7. Tree Views for Zest, `http://wiki.eclipse.org/Tree_Views_for_Zest`