Proceedings of the 29th International Conference on Software Engineering (ICSE),
Minneapolis, USA, May 2007, pp: ???

STRADA: A Tool for Scenario-based Feature-to-Code Trace Detection and Analysis

Alexander Egyed
Teknowledge Corporation
4640 Admiralty Way
Marina Del Rey
CA 90292, USA
aegyed@teknowledge.com

Gernot Binder
Systems Engineering and Automation
Johannes Kepler University
A-4040 Linz, Austria
Gernot.Binder@students.jku.at

Paul Grünbacher
Christian Doppler Laboratory for Automated Software Engineering
Johannes Kepler University
A-4040 Linz, Austria
Paul.Gruenbacher@jku.at

Abstract

Software engineers frequently struggle with understanding the relationships between the source code of a system and its requirements or high-level features. These relationships are commonly referred to as trace links. The creation and maintenance of trace links is a largely manual, time-consuming, and error-prone process. This paper presents STRADA (Scenario-based TRAce Detection and Analysis) – a tool that helps software engineers explore traces links to source code through testing. While testing is predominantly done to ensure the correctness of a software system, STRADA demonstrates a vital secondary benefit: by executing source code during testing it can be linked to requirements and features, thus establishing traceability automatically.

1. Introduction

Understanding the relationships between the features of a software-intensive system (i.e., requirements) and its implementation (i.e., source code) is very difficult. Software traceability aims at defining such relationships. *Trace links* support engineers in understanding complex software systems and answer questions about completeness, conflict, coverage, or consistency. A significant body of work has been published about the usefulness of software traceability [5][6][1] and the need for traceability has expedited its way into numerous software engineering standards and initiatives, such as ISO 15504, MDA, or the CMMI. However, trace identification and evolution requires significant effort [6] and is thus highly challenging for both researchers and practitioners.

Tool support is essential for dealing with the scale and complexity of traceability in real-world systems. Here we present STRADA, a tool for Scenario-based TRAce Detection and Analysis. STRADA provides two major capabilities:

(1) Scenario-based Trace Capture: Given a set of features and knowledge on how to test those features, the tool silently observes what code is being accessed during testing. The tool then concludes that the code accessed during the testing of a feature must implement that feature – a trace dependency. However, the features of a software system are typically interwoven in the implementation (i.e., crosscutting concerns). This is evident in two, rather common, situations: (i) Test scenarios typically relate to more than one feature. As a result, there is an uncertainty about which section of the accessed code belongs to what feature. (ii) Test scenarios often access code that does not belong to one of its features. This is typically the case with code that is co-located (i.e., executed together) but otherwise independent. As a result, there is uncertainty as to who owns any given method.

(2) Trace Analysis: STRADA helps engineers to identify and resolve uncertainties. It does so automatically in cases where the captured trace links constrain one another in a way that logical consequences exclude certain feature/method ownerships [1]. It also provides a simple but powerful set of features to manually assist this process when the automated approach dead ends.

We evaluated STRADA on a half dozen industrial and open-source software systems – including the ArgoUML, GanttProject, Siemens Route Planner, and Video-on-Demand Player. Although not a commercial-grade product, STRADA is quite scalable and fairly reliable. Its capabilities are the result of thorough research on scenario-based trace analyses over the last couple of years [1][2][3][4].

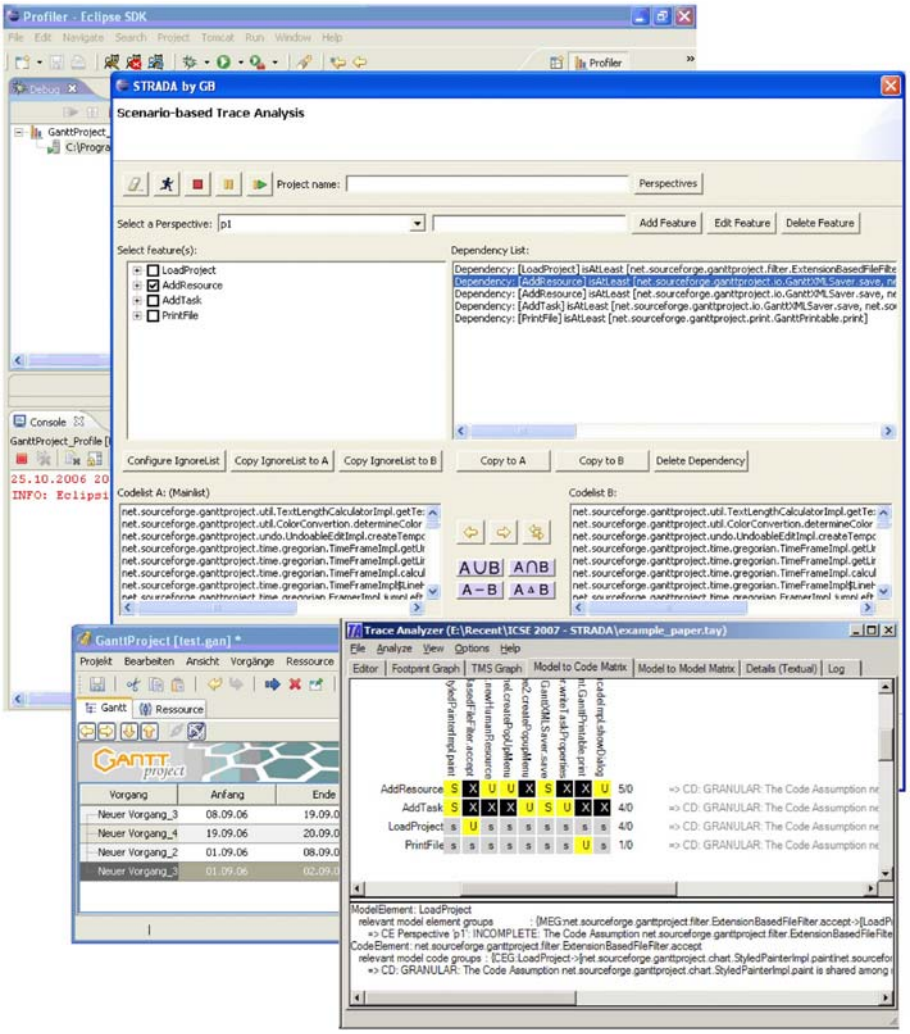

Figure 1. STRADA with Eclipse Profiler and Ganttproject

4. Tool and Experiences

Figure 1 depicts a screen snapshot of the tool which illustrates the use of STRADA while observing the execution of the GanttProject (lower left) through the help of the Eclipse Profiler (upper left). Both components of the STRADA tool have their own visualizations. The Trace Capture component (middle) is currently observing the execution of a test scenario for the Add Resource feature. The Trace Analysis component (bottom right) is visualizing the current knowledge on feature-to-code mapping in form of a trace matrix.

STRADA can be applied to any software system that can be observed during execution. It does not necessarily require source code, though source code allows point-inspections to clarify uncertainties. The use in distributed environments is limited to the ability of the profiler. When working with STRADA one has to be aware of the limitations of profiling: profilers are limited in that they just monitor the execution. They neither understand the code they observe nor the system's features. During testing, they do not know whether a steam of method invocations are logically connected or belong to separate requirements/feature that just happen to be executed at the same time.

The tool is currently restricted to Java-based systems because the Trace Capture component uses a Java profiler. However, the approach is not restricted to this language and would work if integrated with profilers for other languages.

References

[1] Egyed A.: A Scenario-Driven Approach to Trace Dependency Analysis. *IEEE Transactions on Software Engineering (TSE)* 29(2), 2003, 116-132.

[2] Egyed A., Grünbacher P.: Identifying Requirements Conflicts and Cooperation, *IEEE Software* 21(6), 2004.

[3] Egyed A., Grünbacher P., "Supporting Software Understanding with Automated Traceability", Int'l Journal of Software Engineering and Knowledge Engineering, Vol. 15, No. 5, World Scientific Publishing Comp., pp 783-810, 2005

[4] Egyed, A., Heindl, M., Biffl, S., and Grünbacher, P.: "Determining the Cost-Quality Trade-off for Automated Software Traceability," *Proc. 20th IEEE/ACM Int. Conf. on Automated Software Engineering*, Long Beach, CA, 2005.

[5] Gotel, O. C. Z. and Finkelstein, A. C. W.: "An Analysis of the Requirements Traceability Problem," *Proc. of the 1st Int. Conf. on Rqts Eng.*, 1994, pp.94-101.

[6] Ramesh, B., Stubbs, L. C. and Edwards, M. Lessons learned from implementing requirements traceability. Crosstalk, 8(4):11–15, 1995.

[7] Ramesh B., Jarke M.: Toward Reference Models of Requirements Traceability. IEEE TSE 27(1): 58-93 (2001)