Building Reverse Engineering Tools with Software Components: Ten Lessons Learned

Holger M. Kienle
University of Victoria
Victoria, Canada
hkienle@acm.org

Abstract

My dissertation explores a new approach to construct tools in the domain of reverse engineering. The approach uses already available software components as building blocks, combining and customizing them programmatically. This approach can be characterized as component-based tool-building. The goal of the dissertation is to advance the current state of component-based tool-building towards a discipline that is more predictable and formal. This is achieved with three research contributions: (1) an in-depth literature survey that identifies requirements for reverse engineering tools, (2) a number of tool case studies that utilize component-based tool-building, (3) and ten lessons learned for tool builders that have been distilled from these case studies.

1. Introduction

Typical research in the reverse engineering domain proposes techniques to improve the comprehension of software systems. In order to evaluate a proposed technique and to demonstrate its feasibility many researchers implement a tool.[1] Such a tool can range from a simple proof-of-concept prototype to a fully-fledged, industrial-strength application.

My dissertation [5] explores the current state of tool building in the reverse engineering domain with the aim to improve upon the current practice. This topic is worthwhile to address because even though tool building is a popular technique to validate research, it is neither simple nor cheap to accomplish. Nierstrasz et al., who have developed the Moose reengineering tool, observe that "crafting a tool requires engineering expertise and effort, which consumes valuable research resources" [12].

[1] An analysis of the software engineering literature of six leading research journals found that 17.1% of the publications employ proof-of-concept implementations as a research method, placing it second only after conceptual analysis with 43.5% [2].

Traditional approaches to tool building have resulted in tools that have a high degree of custom code and exhibit little reuse. This approach offers the most flexibility, but can be costly and can result in highly idiosyncratic tools that are difficult to use. To compensate for the drawbacks of building tools from scratch, researchers have started to reuse existing functionality, leading towards an approach that leverages software components—such as off-the-shelf components and integrated development environments—as building blocks. This approach can be characterized as *component-based tool-building*, as opposed to traditional tool-building

The use of components fundamentally changes the development of tools and has unique benefits and drawbacks. However, this often is not realized by tool builders. Furthermore, the use of components in tool building is currently ad hoc, because of a lack of known experiences and guidelines. As a result, the building of tools resembles a *craft* rather than *professional engineering* [15].

The goal of the dissertation is to advance the current state of component-based tool-building towards a more disciplined, predictable approach. To achieve this goal, the dissertation provides three major contributions:

requirements for reverse engineering: To gain a better understanding of a domain, one can look at its requirements. Unfortunately, there is no catalog that spells out the requirements for tools in the reverse engineering domain. To close this gap, I have conducted an in-depth literature survey that synthesizes five major tool requirements. These requirements can be used to assess and compare reverse engineering tools based on objectively derived criteria.

own tool-building case studies: The dissertation provides a detailed account of six reverse engineering tools that have been constructed with the component-based tool-building approach. They have been developed over a period of five years within the Adoption-Centric Reverse Engineering (ACSE) project at the Univer-

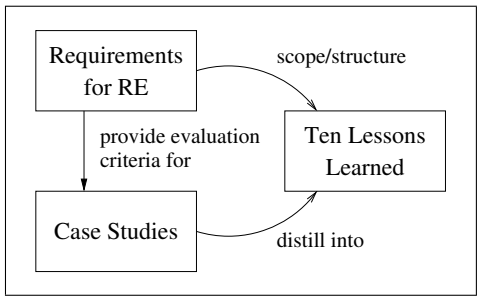

Figure 1. Contributions of the dissertation

sity of Victoria (`www.acse.cs.uvic.ca`). These case studies constitute a body of valuable tool-building knowledge. The constructed tools have been evaluated using the identified requirements for the reverse engineering domain.

ten lessons learned: The six case studies constitute a body of valuable tool-building experiences. However, these experiences represent somewhat "raw" knowledge. To guide researchers that apply the component-based tool-building approach, I have distilled the various experiences made by myself and others into ten lessons learned. The lessons cover the five major tool requirements.

Figure 1 summarizes the dependencies among the three contributions. The case studies are assessed using the tool requirements as evaluation criteria. The lessons learned are distilled from the case studies, treating them as a body of knowledge of component-based tool-building experiences. Furthermore, the lessons are scoped and structured by the requirements. In the following, each contribution is discussed in more detail.

2. Requirements for Reverse Engineering

Tool building in the reverse engineering domain has its own characteristics, which may differ from other domains. For this reason, it is worthwhile to identify and synthesize requirements that have been reported by researchers in the reverse engineering area.

The requirements have been identified based on a careful review of the relevant literature (i.e., relevant journals, conference proceedings, and theses). To my knowledge, this is the first attempt of a comprehensive requirements survey in this domain. The requirements have been identified following a process proposed by Kitchenham et al. called evidence-based software engineering (EBSE) [8].

The survey has identified the following five major requirements for reverse engineering tools:

scalability: The tool should be able to effectively and efficiently deal with a large amount of data. For example, a graph visualizer should be able to render thousands of nodes and arcs.

interoperability: A tool should be able to interact with other tools. In other words, tools that interoperate enable them "to pass information and control among themselves" [19]. This allows to (opportunistically) assemble a tool set that supports a particular reverse engineering task or process.

customizability: A tool should be customizable. This is important because reverse engineering activities are quite diverse and depend on many factors. As a result, reverse engineers have to continuously adapt their tools to meet changing needs.

usability: A tool should be usable. Usability results in more satisfied tool users, increases the reputation of the tool and its developers, and decreases costly redevelopment caused by user complaints.

adoptability: A tool should be adopted by a (industrial) user community. On of the reasons why tools are not adopted by industry is that "many of these tools do not support the right tasks" [16]. Conversely, an adopted tool shows that it is useful in the eyes of its users.

Besides these major requirements, researchers sporadically mention that tools should be interactive, exploratory and lightweight in nature; should have multi-language support; and should enable traceability of artifacts.

The identified requirements are useful to communicate assumptions about tool building in the reverse engineering domain. A requirement that has been reported (independently) by several sources is a good indication that reverse engineering tools should meet this requirement in order to be useful and fulfill the expectations of users.

Tool	Component	Implementors	Publications
REOffice	Excel, PowerPoint	Yang, Weber	[17] [18]
SVG editor	SVG	Kienle	[7]
REVisio	Microsoft Visio	Zhu, Chen	[20] [1]
RENotes	Lotus Notes	Ma	[11] [10]
REGoLive	Adobe GoLive	Gui	[4] [3]
WSAD extr.	IBM WSAD	Kienle	[6]

Table 1. Summary of case studies

3. Own Tool-Building Case Studies

The dissertation describes six case studies that realize different reverse engineering tools, covering a spectrum of reverse engineering functionalities, software components, and technologies. Table 1 provides a brief summary of the tools, characterizing the leveraged component, the primary implementors, and the published research results. In the following each tool is briefly described.

Lesson learned	Major Requirement	Case studies					
		(1)	(2)	(3)	(4)	(5)	(6)
1. Benchmark the scalability of visualizer functionality.	scalability	●	●	●			
2. Coarser-grained extractors do not cause scalability problems.	scalability	●	●	●	●	●	●
3. Presentation integration has to leverage a wiring standard.	interoperability				●	●	●
4. File-based data interoperability is very effective.	interoperability	●	●	●	●	●	
5. Components enable rapid prototyping of tool functionality.	customizability	●	●	●	●	●	●
6. Limited customizability of host components often causes problems.	customizability			●	●	●	
7. Customization samples reduce development effort and learning curve.	customizability	●	●	●	●	●	●
8. Components that are uniform and familiar benefit usability.	usability	●		●	●	●	●
9. Adoptability must address also nontechnical aspects.	adoptability	●	●	●	●	●	●
10. Adoptability is a concern during the whole life cycle.	adoptability	●	●	●	●	●	●

Case studies: (1) REOffice, (2) SVG graph editor, (3) REVisio, (4) RENotes, (5) REGoLive, (6) WSAD extractor

Table 2. Summary of lessons learned and supporting case studies

REOffice and REVisio: Both tools provide graph and metrics visualization of reverse engineering data similar to Rigi. REOffice is based on Excel and PowerPoint, whereas REVisio is based on Visio. Generally, Microsoft Office is a popular target for component-based tool-building. However, there are still not many reverse engineering tools that have customized Office.

SVG graph editor: The SVG graph editor leverages the SVG vector graphics format (standardized by the W3C) to embed interactive graphs into diverse documents (e.g., Web pages, PowerPoint presentations, and Word documents). The graph editor has been implemented with about 7,000 lines of JavaScript code.

RENotes: RENotes leverages Lotus Notes to provide software reengineering functionality such as artifact filtering and graph manipulation. Notes has been mostly customized in the context of industrial groupware and email solutions; to my knowledge, there are no research tools that have customized Notes to realize reverse engineering functionality. RENotes shows that Notes is sufficiently customizable to realize a rudimentary reverse engineering tool.

REGoLive: Similar to RENotes, REGoLive targets a COTS product for which little or no tool-building experience exists in the reverse engineering domain. Furthermore, whereas many IDEs have been customized to realize functionality for traditional reverse-engineering tasks, REGoLive is the first attempt to customize a Web authoring tool, Adobe GoLive, to provide functionality for the reverse engineering of Web sites.

WSAD extractor: The WSAD Web site extractor is based on IBM's Websphere Application Developer, which in turn is built on top of Eclipse. The extractor's schema is defined with the Eclipse Modeling Framework. To visualize the extracted facts, SHriMP's Eclipse plug-in is used. The extractor has been developed as a response to industrial need—the reengineering of the IBM CAS Cassis Web site [9], which has been developed as a J2EE Web application.

The above case studies have confirmed that (1) component-based tool-building has the potential to produce tools that can meet the tool requirements identified by the survey, and (2) component-based tool-building is a feasible approach that is worthwhile to pursue.

4. Ten Lessons Learned

While the individual case studies provide valuable tool-building experiences that other researchers can build upon, they are somewhat dependent on the tool's functionality, utilized components, etc. In contrast, the lessons learned distill the case studies into more general observations. The lessons learned provide valuable advice for researchers who want to follow a component-based tool-building approach. Table 2 summarizes the ten lessons learned and identifies the relevant case studies that provide support for them. Furthermore, each lessons can be assigned to one of the five major requirements (cf. Section 2).

For the CeBASE repository, Rus et al. classify lessons learned into hypothesis, observation, good practice, and bad practice [14]. Lesson 2 can be seen as good practice that is widely applied. The lessons about scalability, interoperability, customizability, and usability classify as observations that are supported by concrete experiences. The lesson about usability is a hypothesis that should be confirmed with user studies. The lessons about adoptability can be seen as hypothesis because they are not backed up with formal experiments and empirical data; they represent my beliefs after several years of tool-building and research in reverse engineering.

Because of space constraints, I only discuss one lesson learned in more detail; the full discussion is available in the dissertation [5, sec. 7.1].

Lesson Learned 5: Host components enable rapid prototyping of tool functionality by leveraging existing functionality and scripting languages.

Explanation: The ability to customize host components is a necessary prerequisite for component-based tool-building approach. Fortunately, many components support customizability via scripting or API programming. If a component offers both an API and scripting, often the latter is preferred. For example, tools that leverage Microsoft Office often rely on Visual Basic instead of using another COM-aware language such as C++. This is the case for REOffice and REVisio. The SVG graph editor, implemented in JavaScript, shows that scripting languages can be used to write applications with thousands of lines of code in a disciplined manner and using design patterns. Scripting of customizations eases experimentation and allows rapid prototyping of functionality. In a research environment, these benefits outweigh potential drawbacks of scripting such as lack of a strong type system, and worse maintainability. If the component is carefully selected, its existing functionality can cover a significant part of the tool functionality. As a result, significantly less code needs to be written and subsequently maintained. For example, Reiss reports that for Desert's editor, "FrameMaker provides many of the baseline features we needed" [13]. All of the case studies benefit from a significant amount of functionality that is provided by their host components. REOffice, REVisio, and the SVG graph editor mostly leverage the drawing capabilities of their leveraged components; RENotes leverages textual representations of relational data and the underlying repository. The WSAD extractor leverages various plug-ins to extract information and visualize it textually and graphically.

5. Conclusions

My dissertation strives to advance component-based tool-building beyond the craft stage. What is needed is a more predicable approach that provides processes, techniques, and guidance to tool builders. The dissertation's contributions—most notably the ten lessons learned—are a first step in the direction towards the professional engineering stage. However, to reach that stage more documented experiences and case studies are needed; more lessons learned have to be distilled; and more disciplined approaches for tool construction have to be proposed, tried out, and evaluated.

I hope that other researchers will pick up where this dissertation left off. Especially, it would be worthwhile to gain experiences with component-based tool-building in other domains besides reverse engineering. At the very least, this dissertation should have been able to establish that component-based tool-building is worthwhile to explore further.

References

[1] Y. Chen. Building a software metrics visualization tool using the Visio COTS product. Master's thesis, Department of Computer Science, University of Victoria, 2006.

[2] R. Glass, I. Vessey, and V. Ramesh. Research in software engineering: an analysis of the literature. *Information and Software Technology*, 44(8):491–506, June 2002.

[3] G. Gui. Extending a Web authoring tool for Web site reverse engineering. Master's thesis, Department of Computer Science, University of Victoria, 2005.

[4] G. Gui, H. M. Kienle, and H. A. Müller. REGoLive: Building a web site comprehension tool by extending GoLive. *7th IEEE International Symposium on Web Site Evolution (WSE'05)*, pages 46–53, Sept. 2005.

[5] H. M. Kienle. *Building Reverse Engineering Tools with Software Components*. PhD thesis, Department of Computer Science, University of Victoria, Nov. 2006. `https://dspace.library.uvic.ca:8443/dspace/handle/1828/115`.

[6] H. M. Kienle and H. A. Müller. A WSAD-based fact extractor for J2EE web projects. *Technical Report, University of Victoria*, June 2006.

[7] H. M. Kienle, A. Weber, and H. A. Müller. Leveraging SVG in the Rigi reverse engineering tool. *SVG Open / Carto.net Developers Conference*, July 2002.

[8] B. A. Kitchenham, T. Dyba, and M. Jorgensen. Evidence-based software engineering. *26th ACM/IEEE International Conference on Software Engineering (ICSE'04)*, pages 273–281, May 2004.

[9] P. Kolari, T. Finin, Y. Yesha, K. Lyons, J. Hawkins, and S. Perelgut. Policy management of enterprise systems: A requirements study. *7th International Workshop on Policies for Distributed Systems and Networks (POLICY'06)*, pages 231–234, June 2006.

[10] J. Ma. Building reverse engineering tools using Lotus Notes. Master's thesis, University of Victoria, Department of Computer Science, Oct. 2004.

[11] J. Ma, H. M. Kienle, P. Kaminski, A. Weber, and M. Litoiu. Customizing Lotus Notes to build software engineering tools. *Conference of the Centre for Advanced Studies on Collaborative Research (CASCON'03)*, pages 276–287, Oct. 2003.

[12] O. Nierstrasz, S. Ducasse, and T. Girba. The story of Moose: an agile reengineering environment. *10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE-13)*, pages 1–10, Sept. 2005.

[13] S. P. Reiss. The Desert environment. *ACM Transactions on Software Engineering and Methodology*, 8(4):297–342, Oct. 1999.

[14] I. Rus, M. Lindvall, C. Seaman, and V. Basili. Packaging and disseminating lessons learned from COTS-based software development. *27th NASA Goddard/IEEE Software Engineering Workshop (SEW-27'02)*, pages 131–138, Dec. 2002.

[15] M. Shaw. Prospects for an engineering discipline of software. *IEEE Software*, 7(6):15–24, Nov. 1990.

[16] M.-A. D. Storey. *A Cognitive Framework for Describing and Evaluating Software Exploration Tools*. PhD thesis, Simon Fraser University, Dec. 1998.

[17] A. Weber, F. Yang, and H. Müller. Office XP in scenarios. *CASCON 2001 Workshop on Adoption-Centric Tool Development (ACTD'01)*, Nov. 2001. `https://www-927.ibm.com/ibm/cas/toronto/publications/TR-74.182/37/anke-ACTD.pdf`.

[18] F. Yang. Using Excel and PowerPoint to build a reverse engineering tool. Master's thesis, Department of Computer Science, University of Victoria, 2003.

[19] M. V. Zelkowitz. Modeling software engineering environment capabilities. *Journal of Systems and Software*, 35(1):3–14, Oct. 1996.

[20] Q. Zhu, Y. Chen, P. Kaminski, A. Weber, H. Kienle, and H. A. Müller. Leveraging Visio for adoption-centric reverse engineering tools. *10th IEEE Working Conference on Reverse Engineering (WCRE'03)*, pages 270–274, Nov. 2003.