# Must Tool Building Remain a Craft? (Position Paper)

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

## Abstract

*The building of tools is an important part of many research efforts, especially in the reverse engineering domain. However, despite the fact that many researchers build tools, it is still done in an ad hoc fashion, resembling a craft rather than professional engineering.*

*In this paper, we give examples of why we believe that tool building is pursued in an ad hoc manner. We argue that in order to advance the current state of tool building it is necessary that researchers start publishing their tool building experiences. Furthermore, these experiences could be distilled into a tool building handbook that makes tool building more predictable and formal.*

## 1. Introduction

Tools and tool building often play an important role in applied computer science research. Tools are a fundamental part of software engineering research in general, and reverse engineering research in particular. The tangible results of research projects are often embodied in tools, for instance, as a reference or proof-of-concept implementation. In fact, 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% [7].

Even though tool building is a popular technique to validate research, it is neither simple nor cheap to accomplish. Tool building is costly, requiring significant resources. This is especially the case if the tool has to be robust enough to be used in (industrial) user studies. Sometimes a significant part of the resources of an entire research group are devoted to building, evaluating, and improving a tool.

Given the importance of tool building for researchers and the significant costs associated with it, it is surprising that tool building is still done in an ad hoc fashion. Specifically,

- researchers do not elicit requirements for their tools.
- researchers do not reflect on design-related issues such as their approach to tool-building, standard architectures, technologies, etc.
- researchers do not follow a process when building tools.

As a result, the building of tools resembles a *craft* rather than *professional engineering* [19].

In the following sections, we report on the current state of tool building in the reverse engineering domain with respect to requirements elicitation for tools, component-based tool building, and the use of processes for building tools. Our report is grounded in a review of the literature and our own experiences.

## 2. Requirements of Tools

When researchers are building a tool, they need to know about the functional and non-functional requirements that their tools should meet. Understanding of tool requirements is a necessary prerequisite to build tools that users find useful and usable. Furthermore, once tool requirements have been elicited, they can be used as a yardstick for tool assessment.

We have conducted an extensive literature review on requirements that researchers report for reverse engineering tools [11, sec. 3.1]. We have found that many researchers address the following non-functional requirements: scalability, interoperability, customizability, usability, and adoption of tools. 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.

A number of observations can be made based on the conducted survey:

**requirements elicitation is ad hoc:** Requirements are often discussed without citing related work, or mentioned without giving a detailed explanation or rationalization. Particularly, it is often not clear where requirements come from—have they been derived based on observation of users, questioning of users, experiences of the developers, personal intuition, etc.? Identified requirements are useful, regardless whether they originated in empirical research, or practical experience and intuition. However, researchers should nevertheless clearly state how the requirements originated.

**requirements are not scoped:** Researchers do not discuss the applicable scope of a stated requirement. For instance, is the requirement believed to apply to all software systems, to the domain of software or reverse engineering, to certain kinds of tools, or to one tool in particular?

**requirements are unstructured:** A requirement is often discussed in isolation, without addressing dependencies or trade-offs with other requirements.

**requirements are ill-defined:** Most of the requirements in the survey are discussed on an abstract level. Whereas such requirements are useful, they do not allow to measure objectively to what extent a tool fulfills a requirement, or to compare tools quantitatively. This is understandable for ill-defined requirements such as usability. However, other requirements such as scalability are easier to quantify. Researchers should take this opportunity to precisely state the metric and the experimental data that makes them believe that their tool is, say, scalable.

Even though it is by now well understood that a thorough requirements elicitation should be conducted before starting to implement a system, it appears that reverse engineering tools are often built without a clear understanding of either the general requirements of the reverse engineering domain, or the specific requirements of a particular kind of tool.

## 3. Design-Related Issues of Tool Building

There are many issues that researchers need to consider when designing a tool, for instance

**tool-building methodology:** For example, the tool could be build from scratch, or could reuse existing functionality via a component-based tool-building approach (see below).

**tool architecture:** There has to be a decision on the tool's high-level architecture. This decision is important because it has an impact on the realization of the non-functional requirements [6] [21]; conversely, the identified non-functional requirements of a tool often drive the decision to select a particular architecture or architectural style [2] [20].

**enabling technologies:** Different technologies have different benefits and drawbacks. When interconnecting tools, one can choose among many different (middleware) technologies, which have different characteristics in terms of portability, scalability, and vendor lock-in. For storage of information one can choose among exchange formats and relational/XML-based databases. And so on.

The above issues—as well as the tool requirements (cf. Section 2)—are influencing each other.

In our research, we have explored the idea of *component-based tool-building* [11, ch. 5]. The drawbacks of building tools from scratch are slowly causing a shift in the way tools are now built by researchers. More and more tools are built using components (such as off-the-shelf (OTS) components and integrated development environments (IDEs)). Components offer a different way of tool building, which replaces hand-crafted code with "pre-packaged" functionality. With components, the emphasis shifts from "tool-crafting" to "tool-assembling," meaning that existing, pre-packaged functionality in the form of components needs to be assembled and customized rather than written from scratch. Finding appropriate host components that already provide a baseline infrastructure, and being able to customize them via adding functionality on top of them is critical for succeeding in the development effort. Tool-assembling promises many benefits, but requires different skills and methods, compared to traditional tool-building.

As a first step to understand how researchers build tools, we have looked through relevant literature for appropriate experiences. We were especially interested in reports of component-based tool-building. Unfortunately, our literature research showed that few researchers bother to report their tool-building experiences. It is often difficult or impossible to understand how researchers have built a tool. The fact that a component has been leveraged is often touched on only in a single sentence. This lack of published experiences is frustrating. It also shows that tracking down evidence of component-based tool-building is tedious and difficult to accomplish. However, there are a few examples of researchers who have explored component-based tool-building in more depth and have published their experiences in some detail. These are the Visual Design Editor (based on Microsoft PowerPoint) [8] [1], Reiss's Desert software development environment (based on Adobe FrameMaker)

[18] [17], and the Galileo fault tree analysis tool (based on Microsoft Visio and Word) [5] [23].

| Tool | Component | Publications |
|---|---|---|
| REOffice | Excel, PowerPoint | [26] |
| SVG graph editor | SVG | [13] |
| REVisio | Microsoft Visio | [28] [3] |
| RENotes | Lotus Notes | [16] [15] |
| REGoLive | Adobe GoLive | [10] [9] |
| WSAD extractor | IBM WSAD | [12] |

**Table 1. Summary of own tool-building experiences**

To gain a better understanding of component-based tool-building, we have constructed a number of tool prototypes that use a variety of components (cf. Table 1). These tool prototypes can be seen as case studies that augment the existing body of knowledge with additional experiences [11, ch. 6]. In contrast to most published experiences of other researchers, our case studies provide a more detailed treatment of implementation issues and reflect on the impact of using a component-based tool-building approach. Furthermore, we have distilled our experiences into ten lessons learned [11, sec. 7.1].

To summarize the current state, almost all of the published tool-building examples lack a conscious exploration of their approach in the sense that there is little or no reflection on the employed tool-building approach and the effect that it has on the development effort and the tool users. Consequently, tool-building continues to be approached in an ad hoc manner, lacking general solutions.

## 4. Processes for Tool Building

All software development projects should use a well-defined process. Tool-building in an academic environment is no exception to this rule. A process should provide guidance on what work products should be produced when, how, and by whom. A well-defined process enables a repeatable and predictable way of building tools.

We have conducted a literature study to find out about desirable characteristics of processes for tool building [11, sec. 3.2]. Furthermore, we wanted to know what kind of processes are used or proposed (if any). Unfortunately, most researchers do not report at all about their tool development process. It seems that tool development in an academic research environment often is ad hoc and unstructured.[1] As a result, tools are developed without following a well-defined process. However, researchers have reported some experiences that allow to distill properties that an appropriate development process should probably possess.

Researchers have started to think about how reverse engineering tools should be built—but examples are scant. Lethbridge and Singer focus on requirements elicitation from future users of the system (feedback-based) [14]. Storey proposes an iterative cycle of tool design and tool evaluation (iterative) [22]. Yang advocates to use rapid prototyping (prototype-based) [27]. Unfortunately, most of these ideas are incomplete and lack necessary details. The literature survey has identified the following requirements for tool development processes: feedback-based, iterative, and prototype-based. Based on our experiences, we believe that processes should be lightweight and adaptive as well.

The identified requirements can, on the one hand, provide important background information and constraints for developing an appropriate tool-building method, and, on the other hand, serve as evaluation criteria to judge the efficacy of a proposed tool-building method. Based on the identified process requirements for the reverse engineering domain, we propose a process framework for tool development [11, sec. 7.2]. The process framework is tailorable so that individual, project-specific characteristics can be accommodated. The process framework follows the ideas of Vessey and Glass, who believe that approaches to systems development should be strong (i.e., domain specific) as opposed to weak (i.e., domain independent) [24].[2]

## 5. Conclusions

In this paper, we have presented evidence that tool building in academia is conducted in an ad hoc manner. This is apparent by the way in which requirements for tools are elicited (Section 2), by the lack of reflection on design-related issues such as the employed tool-building methodology (Section 3), and by the fact that researchers have neither proposed a tool building process nor follow established processes (Section 4).

As a first step to make tool building less ad hoc, researcher have to start to publish their experiences. Generally, researchers are good at evaluating their tools (i.e., the outcome), but poor at reflecting on how they accomplished the building of the tool (i.e., how the outcome was produced). Based on our knowledge of the literature, we have observed that software engineering researchers tend not to

[1] Tools are often constructed by students that have only a few years of programming experience. They typically work alone or in small teams with informal communication flow and without an explicit process. Furthermore, they often work not closely supervised and are evaluated based on their finished product, not on how they have constructed it.

[2] An example of a strong development approach is Extreme Researching (XR), which is a process specifically designed for applied research and development in a distributed environment that has to cope with changing human resources and rapid prototyping [4]. It is an adaptation of Extreme Programming and has been developed by Ericsson Applied Research Laboratories to support distributed telecommunications research.

report their tool-building experiences in scientific publications. Indeed, Wirth has observed in 1995 that "there is a lack of published case studies in software construction" [25]. One reason for the lack of published experiences might be the perception that they are not part of the publishable research results. We believe, however, that these experiences are valuable, could greatly benefit other researchers, and constitute a first step towards a better and more formal understanding of tool-building issues. A published body of experiences could, for example, lead to a better understanding of how different tool-building approaches affect the resulting tools' quality attributes and other properties.

Secondly, once there is a body of experience available, it can be used towards the distilling a *handbook*—or *bible*—for the domain of reverse engineering tools, covering issues such as:

- requirements elicitation
- standard architectures
- tool-building design patterns
- benchmarks (to evaluate and compare tools)
- guidelines for user studies
- discussion of benefits and drawbacks of technologies
- lessons learned (i.e., what works and what does not)
- catalogs of reusable components
- tool-building methodologies and processes
- etc.

This would enable researchers to build tools in a faster and more predictable manner. The resulting tools might be more usable and useful. Such an endeavor would require the concerted effort of a group of researchers. The writing of a dedicated handbook for tool building is certainly a significant effort. However, the benefits that could be reaped from it are equally significant.

## References


[1] R. M. Balzer and N. M. Goldman. A COTS based product line architecture for generating design editors. *IFIPS World Computer Conference 2000*, Aug. 2000.

[2] L. Bass, M. Klein, and F. Bachmann. Quality attribute design primitives and the attribute driven design method. In F. van der Linden, editor, *PFE-4 2001*, volume 2290 of *Lecture Notes in Computer Science*, pages 169–186. Springer-Verlag, 2002.

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

[4] O. Chirouze, D. Cleary, and G. G. Mitchell. A software methodology for applied research: eXtreme Researching. *Software—Practice and Experience*, 35(15):1441–1454, Dec. 2005.

[5] D. Coppit and K. J. Sullivan. Multiple mass-market applications as components. *22nd ACM/IEEE International Conference on Software Engineering (ICSE'00)*, pages 273–282, June 2000.

[6] E. Folmer and J. Bosch. Architecting for usability: a survey. *Journal of Systems and Software*, 70(1-2):61–78, Feb. 2004.

[7] 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.

[8] N. M. Goldman and R. M. Balzer. The ISI visual design editor generator. *IEEE Symposium on Visual Languages (VL'99)*, pages 20–27, Sept. 1999.

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

[10] 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.

[11] 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/handle/1828/115`.

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

[13] 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.

[14] T. C. Lethbridge and J. Singer. Strategies for studying maintenance. *2nd Workshop on Empirical Studies of Software Maintenance (WESS'96)*, pages 79–83, Nov. 1996.

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

[16] 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.

[17] S. P. Reiss. Program editing in a software development environment (draft). `http://www.cs.brown.edu/~spr/research/desert/fredpaper.pdf`, 1995.

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

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

[20] M. Shaw and P. Clements. A field guide to boxology: Preliminary classification of architectural styles for software systems. *27th IEEE International Computer Software and Applications Conference (COMPSAC'03)*, pages 6–14, Aug. 1997.

[21] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

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

[23] K. J. Sullivan and J. C. Knight. Experience assessing an architectural approach to large-scale systematic reuse. *18th ACM/IEEE International Conference on Software Engineering (ICSE'96)*, pages 220–229, May 1996.

[24] I. Vessey and R. Glass. Strong vs. weak approaches to systems development. *Communications of the ACM*, 41(4):99–102, Apr. 1998.

[25] N. Wirth. A plea for lean software. *IEEE Computer*, 28(2):64–68, Feb. 1995.

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

[27] H. Yang. The supporting environment for a reverse engineering system—the maintainer's assistant. *Conference on Software Maintenance (CMS'91)*, pages 13–22, Oct. 1991.

[28] 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.