Research on HCI Toolkits and Toolkits for HCI Research: A Comparison

Ulrich von Zadow, Raimund Dachselt
Interactive Media Lab Dresden
Technische Universität Dresden
Dresden, Germany
{firstname}.{lastname}@tu-dresden.de

ABSTRACT
In this position paper, we categorize toolkits in HCI research into two types. The first type, which we will call *Research Toolkits*, enable development of interfaces based on entirely new paradigms. In contrast, *Toolkits for Research* speed up development by encapsulating common code revealed during research, enabling faster iterations and research participation by more people. Put another way, Research Toolkits demonstrate research, while Toolkits for Research aid—and sometimes enable—research. This paper describes properties of the two toolkit types and examines criteria for evaluation in the light of these properties. Our discussion is based on publications on toolkit evaluation, on sample HCI toolkits, on industry works that cover toolkit design, and on our own experiences in writing toolkits.

ACM Classification Keywords
H.5.2. User Interfaces

Author Keywords
User Interface Systems Evaluation; Toolkits; Frameworks

INTRODUCTION
Toolkits are instrumental in enabling us to build user interfaces quickly, hiding complexity and codifying best practices. Toolkit research is therefore an important research subject, and historically, toolkit ideas that first appeared in research have seen remarkable use in industry. One example for an indisputable success in this area is the story of GUI interface builders, the graphical editors that allow us to place UI elements in dialogs. The first interface builders were developed in research projects (e.g., Buxton et al.'s MenuLay [2] and Xerox PARC's Trillium system [6]), before they evolved to the systems that are integrated into virtually every major UI toolkit today.

Like other tools, interface builders work because they save their users from reinventing the wheel again and again. Toolkits make building UIs easier and enable the construction of larger systems [10, 11]. If designed correctly, they channel creativity, making known-good paths more accessible and helping to focus research [4, 10, 11].

An examination of prior work suggests that two types of toolkits have evolved in the HCI community, for which we use the terms *Research Toolkits* and *Toolkits for Research*. Research Toolkits enable development of interfaces based on entirely new paradigms, such as proxemic interaction [9] or zoomable user interfaces [14]. The corresponding publications have new abstractions and concepts as major contributions and use concrete toolkits to demonstrate their usefulness. On the other hand, Toolkits for Research essentially encapsulate common concepts found during development of (often short-lived) research prototypes, thus enabling faster iterations and research participation by more people. The goal in this case is a practical, usable toolkit that makes it easier to conduct research in a certain area.

This position paper gives evidence for the existence of the two distinct toolkit types and compares them concerning goals, properties and criteria for evaluation. To evaluate and discuss current practices, we examine a sampling of toolkit publications as well as publications on toolkit evaluation. In addition, we look at works on toolkit best practices from an industry perspective as well as our own experience in writing and maintaining a moderately successful Post-WIMP UI toolkit—libavg[1]—over a course of 15 years. Together, this provides grounding for a discussion in which we examine criteria for the evaluation for both types of toolkits.

EXAMPLE HCI TOOLKIT PUBLICATIONS
To understand the current state of toolkit evaluation, we examined a sampling of toolkit publications with respect to the benefits claimed and the methods used to evaluate them. These were:

- GroupKit [12], a groupware toolkit,
- PyMT [5], a toolkit for touch-based user interfaces,
- the Proximity Toolkit [9], which enables building applications based on proxemics, and

[1] www.libavg.de

- the ZOIL Framework [14], a toolkit for zoomable user interfaces.

While these toolkits cover a wide variety of application cases and research subjects, the publications share remarkable similarities. All of them claim abstractions as major contribution: GroupKit abstracts away all network and connectivity issues, PyMT has persistent event objects, the Proximity Toolkit hides sensing hardware and delivers high-level proxemics data, and ZOIL's central abstraction is a zoomable canvas that contains the complete UI. In all cases, the source code is available under a permissive license.

Most the toolkits we looked at (GroupKit, Proximity Toolkit, ZOIL) are validated empirically using comparatively simple example applications often written by students at the respective research labs. Thus, they can argue that they are easy to use, since it is possible for students to use them. Conversely, they cannot empirically argue that they are useful for larger systems. The PyMT paper is an exception in that it additionally describes somewhat larger applications developed outside of the lab and deployed in public venues.

We can clearly categorize GroupKit and PyMT as Toolkits for Research, while the Proximity Toolkit and the ZOIL framework fit our definition of Research Toolkits. GroupKit and PyMT focus on practical usability (the GroupKit paper specifically states that it encapsulates common code revealed during research). Both also have a longer history of use before the actual publication and the authors made an effort to maintain them long after publication: GroupKit was maintained for ten years, while PyMT is still maintained, albeit under the name Kivy.

In contrast, the Proximity Toolkit and ZOIL enable development of interfaces based on entirely new paradigms, and they exist to prove that this is possible in general. There is a clear novelty to the abstractions they provide. The concrete toolkit is therefore less important than the theoretical contribution. Perhaps accordingly, both research toolkits in our sample were maintained for less than two years after publication.

WORK ON TOOLKIT EVALUATION

We can find criteria for toolkit design and evaluation in several HCI publications, the foremost of these being Olsen's 2007 paper on toolkit evaluation [11]. This paper is cited in the CHI reviewing guide and as such is the closest to a standard for toolkit evaluation that we have. Olsen enumerates a number of ways toolkits can demonstrate usefulness, which we paraphrase here:

- Demonstrate importance: The importance of a toolkit hinges on the number of potential users, on meaningful target tasks, and on the situations in which it can be used.
- Problem not previously solved: A toolkit can claim novelty, i.e., demonstrate that it is the first tool for the task.
- Generality: Importance increases if the toolkit can claim to support multiple user populations and/or target tasks.
- Reduce solution viscosity: Toolkits can claim to support faster iterations, e.g., by allowing rapid changes in designs.
- Empower new design participants: If a toolkit allows people to work on a solution that previously couldn't, e.g., by making hard problems tractable, this makes it useful.
- Power in combination: Allowing users to combine building blocks to create a larger solution quickly can make a toolkit useful.
- Scalability: Toolkits should demonstrate that they can be used to tackle large problems.

Olsen further argues for the publication of incomplete toolkits. His view is that missing features are inevitable in research toolkits for workload reasons, and further, that incompatibility with legacy code is to be expected and the "price of progress".

If we apply Olsen's criteria to the different toolkit types we identified, we find that most criteria apply to both types. One exception is novelty, which is essential for Research Toolkits but less easy to achieve when building a Toolkit for Research. Further, Toolkits for Research cannot have missing features or be unusable for compatibility reasons in major use cases, since their goal is practical usefulness.

Myers et al.'s paper on User Interface Software Tools [10], published in 2000, is at its heart a call for Post-WIMP UI toolkits, and much of the work is concerned with the transition from WIMP to the more varied world of today's UIs. However, it also contains a number of criteria for evaluating tools. Among these are the concepts of *threshold* (how difficult is it to learn system use) and *ceiling* (how much can be done using the tool). In addition, the authors argue that tools "influence the kinds of user interfaces that can be created" and can therefore be used to promote the use of known good concepts. Further, they make the point that building tools needs "significant experience with, and understanding of, the tasks they support".

In his paper "Toolkits and Interface Creativity" [4], Greenberg examines the role of toolkits in fostering programmer creativity. He argues that good tools are "a language that influences [programmers'] creative thoughts": "Simple ideas become simple for them to do, innovative concepts become possible, and designs will evolve as a consequence." The work is based on several groupware toolkits (including GroupKit) initially developed in-house to enable rapid iterations during research. From this experience, he derives a number of design guidelines for toolkit design:

- Base toolkits on "lessons learned from one-off system design".
- Make an effort to create good, clean APIs, since APIs "create the language that people will use to think about design".
- Embed toolkits within "well-known languages and programming paradigms".
- Disseminate tools: Make them available, well-documented, make it easy to "quickly get going".
- "Recognize toolkit creation as an academic contribution": "Currently, toolkit development is rarely rewarded in the major interface conferences, for toolkits are typically perceived as software that just package already known ideas."

Greenberg's focus is clearly on Toolkits for Research: He describes toolkits built to directly support in-house research and subsequently disseminated and published and argues for compatibility with existing systems. In contrast to Olsen, he does not particularly emphasize novelty. Further, he considers compatility to "well-known languages and programming paradigms" to be important, contradicting Olsen's view that incompatibility with legacy code is the "price of progress" and thus not an issue.

BEST PRACTICES IN INDUSTRY

In addition to the research publications above, we looked at a number of sources that describe toolkit design from an industry standpoint. These are a talk by J. Bloch (among others designer of the Java Collections Framework) on API design [1], a chapter on reuse in R. Glass' Book on Software Engineering [3], and a blog post by J. Atwood[2], founder of stackoverflow.com.

Finally, we base our arguments on our own experience in developing and maintaining a software framework, libavg[3]. This toolkit was originally written starting in 2003 to support developing software for museum exhibits, and essentially combines an efficient 2D scene graph with first-class support for touch input and easy scripting in Python. It is moderately successful in industry (use, e.g., by ART+COM AG[4], Archimedes Exhibitions GmbH[5], and Garamantis GmbH[6]) and has been used it to build several hundred exhibits. Since 2013, we have been using it extensively at the Interactive Media Lab Dresden, among others as technological basis for a number of publications (e.g., [7, 8, 13]).

A number of Olsen's criteria (among them easy iterations, new design participants, combinable building blocks and scalability) clearly apply to real-world toolkits as well. However, there are a number of additional aspects that make toolkits successful in practice.

First, industry publications consider the design of reusable components to be very hard and recommend trials in varying scenarios. Glass [3] refers to this as "Rules of Three": "(a) It is three times as difficult to build reusable components as single use components, and (b) a reusable component should be tried out in three different applications before it will be sufficiently general to accept into a reuse library", and Atwood[2] affirms: "We think we've built software that is a general purpose solution to some set of problems, but we are almost always wrong. We have the delusion of reuse". This is in contrast to toolkit publications that claim toolkit use only in the author's lab.

Second, toolkits often need to maintained for extended periods of time, and therefore, maintainability is important. In our experience with libavg, a significant amount of time is spent adapting the toolkit to the changing world around it: As examples, since libavg's inception in 2003, touch has become an important input method, GPUs have become immensely more powerful, and various technologies in use have become unmaintained or been superseded by more powerful, modern ones. Time spent maintaining software is overhead. It is therefore important that this requires minimal effort, and that makes appropriate internal abstractions and readable, well-documented code critical.

Third, API usability is important. Bloch [1] emphasizes the importance of designing an easy-to-use and powerful API for the first public release: "Public APIs, like diamonds, are forever. You have one chance to get it right so give it your best". He therefore promotes a user-centered approach to API design, structures "requirements as use-cases" and suggests the equivalent of paper-prototyping for APIs: "Code the use-cases against your API before you implement it" as well as expert reviews: "Show your design to as many people as you can".

DISCUSSION

Both the HCI toolkits we examined and the works on toolkit evaluation give evidence towards the existence of two clearly different toolkit types that need different criteria for evaluation. Olsen's criteria favor new abstractions and concepts as major contributions and therefore fit very well to Research Toolkits. A number of Olsen's criteria are also important in both cases: For instance, a large potential user population, the ability to combine building blocks to create larger solutions and the scalability to large problems are important in both cases.

However, several criteria do not fit in the case of Toolkits for Research: Since they are meant to be practically usable, compatibility with legacy code becomes important and missing features hinder acceptance. Further, since they are designed in response to concrete needs in prototype development, it may be harder for them to demonstrate novelty. Greenberg hints at this when he writes: "Currently, toolkit development is rarely rewarded in the major interface conferences, for toolkits are typically perceived as software that just package already known ideas"[4]. Still, Greenberg's publication as well as our own experiences in building and maintaining in-house toolkits suggest that they can play an important role in speeding up research and channeling creativity.

Should we be interested in this type of toolkit for our community, it may be beneficial to look at best practices in industry for additional criteria. In this case, examining toolkit maintainability and API usability (based on sound API design principles) may give us candidates. The PyMT paper also gives evidence that Toolkits for Research may in some cases be able to demonstrate scalability to larger problems empirically: Toolkit publications later in the toolkit's lifecycle might make it feasible to write larger applications and even demonstrate practical use by a non-captive audience, i.e., outside of the original research lab.

CONCLUSION

Based on a sample of toolkit publications as well as publications on toolkit evaluation, we have categorized toolkits in HCI research into two distinct types, which we have named Research Toolkits and Toolkits for Research. Further, we have

[2]https://blog.codinghorror.com/rule-of-three/
[3]https://www.libavg.de/
[4]http://artcom.de/
[5]https://www.archimedes-exhibitions.de/
[6]https://www.garamantis.com/

compared these types concerning development goals and properties and looked at works on toolkit best practices from an industry perspective. Based on this research as well as our own experiences in toolkit development, we have additionally discussed criteria for the evaluation of both types of toolkits.

ACKNOWLEDGMENTS

We wish to thank Ulrike Kister and rest of the IMLD for fruitful discussions on the subject of HCI toolkits.

REFERENCES

1. Joshua Bloch. 2006. How to Design a Good API and Why It Matters. In *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications (OOPSLA '06).* ACM, New York, NY, USA, 506–507. DOI: http://dx.doi.org/10.1145/1176617.1176622
2. W. Buxton, M. R. Lamb, D. Sherman, and K. C. Smith. 1983. Towards a Comprehensive User Interface Management System. *SIGGRAPH Comput. Graph.* 17, 3 (July 1983), 35–42. DOI: http://dx.doi.org/10.1145/964967.801130
3. Robert L. Glass. 2002. *Software Engineering: Facts and Fallacies*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
4. Saul Greenberg. 2007. Toolkits and interface creativity. *Multimedia Tools and Applications* 32, 2 (2007), 139–159. DOI:http://dx.doi.org/10.1007/s11042-006-0062-y
5. Thomas E. Hansen, Juan Pablo Hourcade, Mathieu Virbel, Sharath Patali, and Tiago Serra. 2009. PyMT: A post-WIMP Multi-touch User Interface Toolkit. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces (ITS '09).* ACM, New York, NY, USA, 17–24. DOI: http://dx.doi.org/10.1145/1731903.1731907
6. D. A. Henderson, Jr. 1986. The Trillium User Interface Design Environment. *SIGCHI Bull.* 17, 4 (April 1986), 221–227. DOI:http://dx.doi.org/10.1145/22339.22375
7. Ulrike Kister, Patrick Reipschläger, Fabrice Matulic, and Raimund Dachselt. 2015. BodyLenses: Embodied Magic Lenses and Personal Territories for Wall Displays. In *Proceedings of the 2015 International Conference on Interactive Tabletops & Surfaces (ITS '15).* ACM, New York, NY, USA, 117–126. DOI: http://dx.doi.org/10.1145/2817721.2817726
8. Ricardo Langner, Ulrich von Zadow, Tom Horak, Annett Mitschick, and Raimund Dachselt. 2016. Content Sharing Between Spatially-Aware Mobile Phones and Large Vertical Displays Supporting Collaborative Work. In *Collaboration Meets Interactive Spaces*, Craig Anslow, Pedro Campos, and Joaquim Jorge (Eds.). Springer International Publishing, 75–96. DOI: http://dx.doi.org/10.1007/978-3-319-45853-3_5
9. Nicolai Marquardt, Robert Diaz-Marino, Sebastian Boring, and Saul Greenberg. 2011. The proximity toolkit: prototyping proxemic interactions in ubiquitous computing ecologies. In *Proceedings of the 24th annual ACM symposium on User interface software and technology (UIST '11).* ACM, New York, NY, USA, 315–326. DOI: http://dx.doi.org/10.1145/2047196.2047238
10. Brad Myers, Scott E. Hudson, and Randy Pausch. 2000. Past, Present, and Future of User Interface Software Tools. *ACM Trans. Comput.-Hum. Interact.* 7, 1 (March 2000), 3–28. DOI:http://dx.doi.org/10.1145/344949.344959
11. Dan R. Olsen, Jr. Evaluating User Interface Systems Research. In *Proc. UIST '07*. ACM, 251–258. DOI: http://dx.doi.org/10.1145/1294211.1294256
12. Mark Roseman and Saul Greenberg. 1996. Building Real-time Groupware with GroupKit, a Groupware Toolkit. *ACM Trans. Comput.-Hum. Interact.* 3, 1 (March 1996), 66–106. DOI: http://dx.doi.org/10.1145/226159.226162
13. Ulrich von Zadow and Raimund Dachselt. 2017. GIAnT: Visualizing Group Interaction at Large Wall Displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '17).* ACM, New York, NY, USA. DOI: http://dx.doi.org/10.1145/3025453.3026006
14. Michael Zöllner, Hans-Christian Jetter, and Harald Reiterer. 2011. *ZOIL: A Design Paradigm and Software Framework for Post-WIMP Distributed User Interfaces*. Springer London, London, 87–94.