HCI.Tools

Strategies and Best Practices for Designing, Evaluating and Sharing Technical HCI Toolkits

CHI 2017 Workshop

Organizers:

Dr. Nicolai Marquardt – University College London, UK
Dr. Steven Houben – Lancaster University, UK
Prof. Michel Beaudouin-Lafon – Univ. Paris-Sud & CNRS / Inria, FR
Dr. Andrew Wilson – Microsoft Research, USA

Paper Reference:

Nicolai Marquardt, Steven Houben, Michel Beaudouin-Lafon, and Andrew D. Wilson. 2017. HCITools: Strategies and Best Practices for Designing, Evaluating and Sharing Technical HCI Toolkits. In Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '17). ACM, New York, NY, USA, 624-627.

DOI: https://doi.org/10.1145/3027063.3027073

Paper: https://dl.acm.org/citation.cfm?id=3027073

Human-centered Toolkit Design

Henrik Mucha
OWL University of Applied Sciences
Lemgo, Germany
henrik.mucha@hs-owl.de

Karsten Nebe
Rhine-Waal University of Applied Sciences
Kamp-Lintfort, Germany
karsten.nebe@hochschule-rhein-waal.de

ABSTRACT
Human-computer interaction (HCI) is a tool-intensive domain. The multitude of perspectives yields a significant diversity in terms of processes, methods, and tools. Toolkits can support practitioners in selecting and applying appropriate tools for specific tasks. However, in order to be used effectively, toolkits must be designed well. Given the heterogeneous perspectives within the HCI community, we propose to start by differentiating between methodical and technical toolkits. Further, we argue for embracing human-centered design methods (methodical toolkits) to systematically develop high-quality (technical) toolkits. Finally, we focus on challenges and opportunities by presenting examples from many years of working on methodical toolkits for design and usability engineering. Our intention is to share research experiences on methodical toolkits and juxtapose it with the technical toolkit expertise of the workshop participants. Thus, we hope to steer the discussion towards a holistic understanding that promotes toolkits as a research method for HCI and, ideally, develop a tool-chain that supports the systematic design of high-quality technical toolkits.

Author Keywords
human-computer interaction; design; human-centered design; usability engineering; toolkits.

ACM Classification Keywords
D.2.2 Design Tools and Techniques

INTRODUCTION
Human-computer interaction brings together people from various backgrounds equipped with their very own processes, methods, and tools. Frankly, this can be blessing and curse at the same time. But we embrace the opportunities which arise from multidisciplinary collaboration more than we fear the possibility of failure.

Our approach towards researching, designing and evaluating interactive systems can be described as tool-supported human-centered design (HCD). While the principles of HCD are thoroughly described by the ISO standard [3] we want to shed light upon our concept of human-centered tool-support for the design of interactive systems. Most generally speaking, one needs the right tool at the right time for the job one seeks to accomplish. Given the vast number of methods and tools [16] within our domain, toolkits can be regarded as facilitators.

What is a toolkit in general and what is it for us?
Toolkits are capable of bridging the gap between concept design and full implementation by facilitating rapid prototyping and the exploration of novel designs without in-depth technical knowledge [5,10,15]. However, to make our point, we propose a differentiation between technical and methodological toolkits.

Technical toolkits are platforms for rapid prototyping comprising hardware and software building blocks.

Methodical toolkits are collections of methods together with information on when and how to apply them.

As with many terms we are concerned with (e.g. *design*) the term toolkit leaves ample room for interpretation. Since we like to bring a new perspective to the discussion, this differentiation seems somewhat necessary. Our goal is to find ways to systematically produce high-quality toolkits. To this end, we firstly need to establish a common ground for the discussion. This, we achieve by agreeing upon what a toolkit is and what is not (or should be or should not be) and how we go about designing toolkits. Taking this thought further, we state that both are inter-connected. Technical toolkits are the tools used by the users to create a specific outcome while methodical toolkits define the overall process of methods, for which technical toolkits need to be developed. They share common goals and should not be considered in isolation: *Speeding up the process while being easy-to-use; mitigating engineering challenges; lowering the entry bar for engagement; allowing to easily experiment, build and evaluate; improving the quality of the solution; improving interdisciplinary skills; creating a common understanding about the relevance of HCD while at the same time supporting the process with tools for its implementation.*

Why are we interested in toolkits?

Löwgren [7] reasonably distinguishes between engineering design and creative design. The former applies wherever the problem is comprehensively described and the mission is to find one solution to the problem. The latter is described as a tight interplay between problem setting and problem solving where the design space is explored via many parallel ideas and concepts. With each of the authors coming from opposing ends of this spectrum, our collaboration pursues a synthesis of both approaches on a methodical level. Our prime vehicle of scientific and methodical exchange is our shared interest in toolkits. More precisely, we are interested in toolkits as a research method for HCI. Following the research through design approach, we can state that communication among the HCI research and practice community relies heavily on research artifacts [18]. Building theses artifacts requires technical tools as well as, and this is important to us, process and method knowledge - hence our distinction between technical and methodical toolkits. Ideally, toolkits are collections of tools developed or curated by experts to put (experts and) non-experts in places where they can more easily create research artifacts without having to tediously learn highly specific skills.

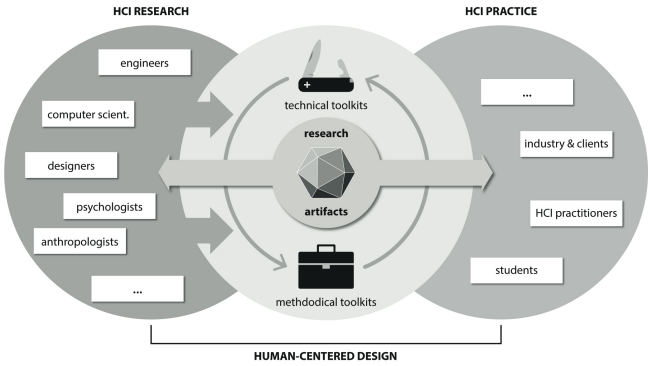

Figure 1 Tool-supported HCD

What is our contribution to the workshop topic?

We propose tool-supported HCD for the design of technical toolkits because ultimately, toolkits are interactive systems. Interactive systems must be designed with the user in focus. Therefore, toolkit designers should rely on established practices. In this regard, HCD appears to be the most suitable procedure to yield high-quality toolkits. Our intention is to share research experiences on methodical toolkits and juxtapose it with the technical toolkit expertise of the workshop participants. Thus, we hope to steer the discussion towards a holistic understanding that promotes toolkits as a research method for HCI. How this could unfold in practice is illustrated in Figure 1.

Why is our contribution relevant?

If we understand research artifacts as the primary entity of idea exchange in HCI, we can conclude that the ability to create such artifacts should not be exclusive to highly specialized experts. Having said this, the significance of toolkits as facilitators becomes apparent. However, effective facilitation requires solid toolkit design. We claim that the latter is most effectively achieved by applying appropriate human-centered methods to the development of technical toolkits as well as to the creation of any research artifact. Thus, actors of the HCI community may directly benefit from the work of colleagues with different backgrounds and skills by easily and correctly applying (or tailoring) their methods and tools. Toolkit design must consider the entire process of context of use, requirements, design and evaluation to produce high quality toolkits. In other words, there is a distinct need for thoroughly designed tool-chains which address the entire process. Following this proposal, we would enable the community to systematically and collaboratively produce high-quality research artifacts, which then serve to communicate ideas, attitudes and solutions to the practice community. Taking this thought even further, we could ultimately use these thoroughly designed toolkits, methodically and technically, in a participatory manner to actively create awareness for the HCI approach and support the establishment of HCD in e.g. organizational settings.

CHALLENGES AND OPPORTUNITIES

Challenge 1: Toolkits as a Research Method for HCI

Having engaged in toolkit design and evaluation ourselves we can state that building a path towards tool-supported research first and foremost assumes to overcome the lack of common language in terms of methods and tools among the HCI community. Often we find ourselves talking about the same activity or outcome but calling it by different names. This is certainly owed to historical disciplinary perspectives [13]. It is one of the reasons why we proposed the distinction between technical and methodical toolkits in the first place. By doing so we add complexity but we also clarify our standpoint. HCD offers a framework to systematically understand such different perspectives and should therefore be applied to toolkit design. It provides an opportunity to foster collaboration through understanding. Establishing such a common ground may result in great opportunities to push the envelope of the field. By emphasizing the interplay of process and artifacts can elevate our communication in terms of transparency, traceability and reproducibility with the goal of motivating and attracting e.g. students, business partners, and many more to adopt the human-centered mindset.

Challenge 2: Designing and Building Toolkits

Generally speaking, to design something of quality requires a clear and robust notion of the needs you want to satisfy and whom you are designing for. A standardized process for this is HCD (‘engineering design’ [7]). At the same time, HCD provides leeway for exploring design spaces in a more creative manner (‘creative design’ [7]). In a nutshell, it is a framework that offers guidance for design and development activities. However, it comes to live only through the people who apply it. This is also the reason why although standardized it is often applied incorrectly due to a

lack of knowledge or resources [13, 14]. Thoroughly designed toolkits could be a remedy. Hence, the challenge would be to find a way to systematically design high quality technical and methodical toolkits, which achieve the aforementioned goals. The opportunity in the context of this workshop could be to identify a tool-chain that may facilitate toolkit design.

Challenge 3: Methods for Evaluating Toolkits

Current attempts to evaluate toolkits comprise efforts to compare different toolkits with one another [17]. We would rather argue for a usability testing approach. Following this idea, when evaluating toolkits, we must focus on three aspects: *Quality of outcome for the toolkit's purpose (intended users, their tasks and intended outcome); but also, quality of outcome for the process, which may include more stakeholders (receiver of the outcome) than just the intended user; Chain of information. Input & output. To support the whole process of transformation of information in a holistic tool-driven approach.* During the workshop we want to discuss strategies to ensure and implement the testing perspective in toolkit design.

SUCCESSFUL TOOLKITS

In order to further elaborate on our proposed categorization, we present a number of toolkits for each category, technical, methodical and hybrid, that in our own experience worked well with clients and students alike.

Toolkit 1: Technical Toolkits

We present examples from our domain, which is predominantly concerned with software design and usability engineering. We focus on interface prototyping software that supports code-less prototyping of GUIs acknowledging recent developments towards a human-centered approach. These tools are quite often also used by non-experts (design or HCD). We chose two popular examples, *Axure* [2] and *Adobe XD* [1] to make our point. These products are relevant because both go beyond pixel design. *Adobe XD* and *Axure* have increasingly adopted a process approach, i.e they integrated features that support testing and collaboration as part of a coherent workflow. Both products do not only address the designer but whole teams. They inherently uphold usability and user experience practices. This can be considered a success in terms of human-centred tool design. However, they do not yet represent an entire tool-chain in the sense that we propose beforehand.

Toolkit 2: Methodical Toolkits

We present one example of a hands-on methodical toolkit which enhanced our educational work with students but did also contribute to designing our very own toolkit presented in the *Toolkit 3* part. *Sprint* [6] is a hands-on guide to the focused and effective application of the human-centered design approach in practice. It lays out how the *Google Ventures* team conducts design sprints as a consulting service for start-ups that struggle with developing their product. Essentially a design sprint is a five-day workshop, a variation of commonly known design thinking workshops. Each day is dedicated to accomplishing another goal in the development process. Each step encompasses different methods which have to be performed in order to move to the next phase. Sprints are literally compressed versions of the HCD process. What makes it so interesting is that it delivers quick and tangible results. Participants spend a given amount of time working intensively on a specific task. At the end of this focused period of time they can see or experience the artifacts they created, a pile of sticky notes, paper prototypes, etc. This quality conveys a feeling of efficiency and satisfaction for everyone involved and fulfils all requirements of a valuable experience. In absence of scientific evaluation, we can only tell from our experience that the sprint format works. Methods such as *Crazy Eights* bear great potential for motivating people who are usually reluctant to pick up pen and paper. The combination of story-telling and detailed tutorials makes it easy to apply and tailor design sprints.

Toolkit 3: UX Method Toolkit

The *UX Method Toolkit* is the result of Henrik's master thesis [9]. For the most part it is a methodical toolkit which employs digital and analogue means to support HCD projects. It comprises 16 HCD methods. As a whole they constitute an entire HCD process. Most methods are suitable to be conducted during UX workshops with users unfamiliar with the methodology. The methods are represented as physical trading cards, digital method pages, and a database entry. All representations are interlinked. These artifacts are shipped in a sturdy briefcase emphasizing the physical presence and contributing to the overall user experience. The Toolkit provides multiple tools to navigate the collection and theoretically the domain itself. First, a visual selection tool – the method map – assigns the methods to phases. Second, QR codes link analogue and digital content. Third, an interactive infographic visualizes appropriate method sequences. These tools facilitate the application of the methods by providing video tutorials, print-able templates, and method-related metrics. This toolkit seeks to combine methodical and technical elements. It is an examination of the interplay of different toolkits within HCI. It is relevant in terms of *lessons learned*: *It is hard to systematically evaluate toolkits with users; talking about methods can be difficult due to a lack of common language; one cannot draw a clear border between disciplines; one has to get the why and how-to across as efficiently as possible.*

ABOUT THE AUTHORS

Henrik Mucha currently works as a research associate at the Institute Industrial IT (inIT) in Lemgo, Germany where he is part of the HCI Lab. Henrik holds degrees in Industrial Design (Dipl.-Des., University Duisburg-Essen) and Usability Engineering (M.Sc., Rhine-Waal University of Applied Sciences). His current work is concerned with human-machine interactions in industrial contexts [8]. His interest in toolkits is e.g. expressed by his master thesis *UX*

Method Toolkit: User Experience Methods for Human-centered Design Workshops [9]. Generally, Henrik's research and starting doctoral thesis revolve around the question of how design methodology and concepts such as UX can be applied to the design of industrial human-machine interactions. **Karsten Nebe** is full time professor for Usability Engineering and Digital Fabrication (since 2011) at the Rhine-Waal University of Applied Sciences, Faculty of Communication and Environment in Kamp-Lintfort, Germany. He was working as Usability Engineer since 2002 and did his doctoral thesis in the field of integrating usability engineering and software engineering [13]. He is head of the degree program "Usability Engineering, M.Sc." and an active member of various DIN, ISO/IEC working groups related to HCD. Since 2014 he is the director of the FabLab Kamp-Lintfort. (Expert member (besides others) in ISO/TC 159/SC 4/WG 28 (Joint between ISO/IEC JTC 1/SC 7 and ISO/TC 159/SC 4) Common Industry Formats for Usability Reports, and ISO/TC 159/SC 4/WG 6 Human-centred design processes for interactive systems).

SUGGESTIONS FOR TOPICS

- *Discussion of the proposal to 'understand' toolkits as a way to perform HCD in a tool-supported way (methodical and technical)*
- *Define the framework for chain of (future) tools*
- *Report on current developments in ISO committees with regards to toolkits*

REFERENCES

1. Adobe XD. Retrieved February 2, 2017 from: http://www.adobe.com
2. Axure. Retrieved February 2, 2017 from: https://www.axure.com/
3. DIN EN ISO 9241. 2010. Ergonomics of human-system interaction, Part 210: Human-centred design for interactive systems.
4. Holger Fischer, Karsten Nebe, Florian Klompmaker. 2007. A Holistic Model for Integrating Usability Engineering and Software Engineering. In *Proceedings of the International Conference on Human Computer Interaction (HCII); Orlando, Florida, USA*
5. Steven Houben, Nicolas Marquardt. 2015. WATCHCONNECT: A Toolkit for Prototyping Smartwatch-Centric Cross-Device Applications. In *Proceedings of CHI'15, ACM Press Pages 1247-1256*
6. Jake Knapp. 2016. *Sprint. How to solve Big Problems and Test New Ideas in just Five Days.* Bantam Press
7. Jonas Löwgren. 1995. Applying Design Methodology to Software Development. In *Proceedings of DIS 1995 ACM Press (1995), 87-95.*
8. Henrik Mucha, Sebastian Büttner, Carsten Röcker. 2016. Application Areas for Human-Centered Assistive Systems. In *Human-Computer Interaction – Perspectives on Industry 4.0. Workshop at i-KNOW 2016 Graz, Austria, Oct 2016.*
9. Henrik Mucha. 2015. *UX Method Toolkit: User Experience Methods for Human-centred Design Workshops.* Master Thesis: Rhine-Waal University of Applied Sciences
10. Brad Myers, B. A., Scott E. Hudson, S. E., Randy Pausch, R. 2000. Past, Present, and Future of User Interface Software Tools. In *ACM Transactions on Computer-Human Interaction 7, 1, 3–28.*
11. Karsten Nebe, Snigdha Baloni. 2016. Agile Human-Centred Design: A Conformance Checklist. *In Proceedings of International Conference on Human Computer Interaction (HCII); Toronto, Canda*
12. Karsten Nebe, Volker Paelke. 2011. Key Requirements for Integrating Usability Engineering and Software Engineering. In *Proceedings of International Conference on Human Computer Interaction (HCII); Orlando, Florida, USA*
13. Karsten Nebe. 2009. *Integration von Usability Engineering und Software Engineering: Konformitäts- und Rahmenanforderungen zur Bewertung und Definition von Softwareentwicklungsprozessen.* Book (Ph.D. Thesis); Shaker Verlag 2009; 383228074X
14. Karsten Nebe, Dirk Zimmermann. 2007. Suitability of Software Engineering Models for the Production of Usable Software. In *Proceedings of the Engineering Interactive Systems - IFIP WG 13.2 1st Conference on Human Centred Software Engineering; Salamanca, Spain*
15. Jakob Nielsen. 1993. *Usability Engineering*. Morgan Kaufmann Publishers.
16. Neville A. Stanton, Paul M. Salmon, Laura A. Rafferty, Guy H. Walker, Chris Baber, Daniel P. Jenkins. 2005. *Human Factors Methods: A Practical Guide for Engineering and Design*. Ashgate Publishing Company, Burlington
17. Brian Tidball, Pieter J. Stappers, Ingrid Mulder. 2010. Models, Collections and Toolkits for Human Computer Interaction. *Paper presented at The 24th BCS Conference on Human Computer Interaction - HCI2010. HCI-Educators Workshop. Dundee, Scotland.*
18. John Zimmerman, Jodi Forlizzi, Shelley Evenson. 2007. Research Through Design as a Method for Interaction Design Research in HCI. In *Proceedings of CHI'07, ACM Press, Pages 493 – 502*

Code and Contribution in Interactive Systems Research

James Fogarty
Computer Science & Engineering
DUB Group | University of Washington
jfogarty@cs.washington.edu

ABSTRACT

The scale and complexity of interactive systems research often require care in distinguishing: (1) the code that implements a system, versus (2) the research contribution demonstrated or embodied in a system. This position paper for the CHI 2017 workshop on #HCI.Tools reflects on this contrast and some common forms of contribution in interactive systems research. We explore several forms of interactive systems contribution based in differentiating: (1) *what* a system accomplishes, versus (2) *how* it accomplishes that. We argue some interactive systems should be considered sketches that use code as a medium to explore their research contributions, while others embody their contributions more directly in their code. Finally, we argue the progress and impact of our field requires diverse forms of contribution across interactive systems.

INTRODUCTION

The scale and complexity of modern interactive systems is daunting along several dimensions. Weiser characterized important aspects of this in a trend from many-to-1 (i.e., many people sharing a single device), to 1-to-1 (i.e., each person with a dedicated device), to 1-to-many (i.e., each person having many devices), to many-to-many (i.e., many people connected through many devices) [14]. As technology enters later stages of this trend, researchers now explore interactive systems that span multiple devices, require massive volumes of data to enable seemingly simple interactions, or require entire social networks before key aspects of their design can surface. Such barriers to real-world deployment of interactive systems create important challenges for interactive systems research.

This reflection focuses primarily on the relationship between code and contribution. Interactive systems research generally contains both, but they are not always well-distinguished. Prior discussions include consideration of the limitations of usability testing [6], examination of common pitfalls in evaluating interactive systems [11], and discussion of technical HCI research as an activity of invention that contrasts with activities of discovery [8]. Additional discussion considers how these challenges manifest or can be magnified in social computing systems [1], with their corresponding need for a critical mass of participation [7]. Our reflection is intended to complement existing discussions without contradiction.

Presented at the CHI 2017 Workshop on #HCI.Tools: Strategies and Best Practices for Designing, Evaluating, and Sharing Technical HCI Toolkits.

This position paper first considers the case where code is closely linked to contribution. It then explores cases where the link is less direct. Consistent with the workshop's proposal to explore conceptual roles for toolkits in HCI research, we examine several forms of interactive systems contribution based in a differentiation of: (1) *what* a system accomplishes, versus (2) *how* it accomplishes that. We conclude with brief comments on our prior interactive systems research as a background for participation in the #HCI.Tools workshop.

WHEN CODE IS THE CONTRIBUTION

Some interactive systems research contributions are directly manifested in code. Although these are a minority, they are important for both: (1) their own research value and impact, and (2) the contrast they can provide for other styles of research. A well-known example is the $1 Recognizer, a template-based unistroke gesture recognizer implemented in approximately 100 lines of code [15]. The paper has been widely cited, both in applications that use the recognizer and in later extensions of the underlying recognition technique. A project website also hosts community implementations of the recognizer in multiple programming languages. The contribution and impact of this research thus directly results from solving a technical challenge in code that people can easily adopt and adapt in their applications and contexts.

Replication, Validation, and Extension

Discussions of replication within the CHI community often focus on experimental replication, which remains relevant in our current context. For example, the $1 Recognizer's project website includes data to replicate its performance experiments. But contributions associated with code also provide opportunity for stronger validation: each future use of that code in a new application, or in a context beyond the original research, validates the underlying research contribution. This validation is riskier and therefore stronger than simply re-executing the original data analysis or replicating the prior experiment.

Figures 1 and 2 illustrate this using a simple visual language we develop in figures throughout this paper. In Figure 1, we distill the contribution of the $1 Recognizer down to a circle. The circle is filled (i.e., purple) to indicate that contribution is novel. In contrast, we will use empty circles (i.e., white) to illustrate components of a system that are not themselves novel (e.g., replicate a prior result, otherwise already known). Figure 2 illustrates this in a research progression based on the $1 Recognizer. This progression begins with Protractor, a recognizer informed by techniques in the $1 Recognizer [9]. Protractor is then used in implementing Gesture Script, a novel tool for interactively authoring compound gestures [10].

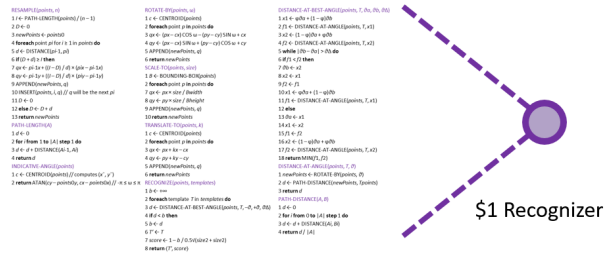

Figure 1: The $1 Recognizer's contribution closely corresponds to its code, allowing re-use of its solution to a technical problem.

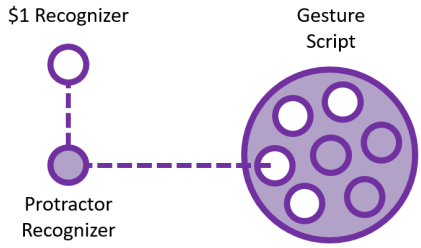

Figure 2: Protractor extends techniques introduced in the $1 Recognizer. It is therefore novel (i.e., filled), while also replicating and extending the $1 Recognizer (i.e., now shown as empty). Gesture Script then directly replicates Protractor as part of a larger system that also includes other components. Some of those are already known (i.e., are also empty), while some are novel (i.e., filled). The overall functionality presented by Gesture Script is also novel (i.e., the large circle is filled).

At each step in this progression, previously novel elements (i.e., filled circles) become already known (e.g., the circle illustrating Gesture Script's use of Protractor is now empty). Gesture Script is itself novel (i.e., the large circle is filled), uses several known techniques (i.e., empty inner circles), and requires several underlying innovations (i.e., filled inner circles). Many other progressions similarly trace back to the $1 Recognizer, and such illustrations can be considered at multiple scales. If we were to "zoom in" on Protractor relative to the $1 Recognizer, we would see they have many identical inner circles, but Protractor includes a novel closed-form similarity metric (i.e., a novel inner circle).

WHEN CODE IS NOT THE CONTRIBUTION

Most interactive systems contributions do not correspond so directly to their code. Identifying the contributions of a system can therefore be considered in terms of: (1) *what* a system accomplishes (i.e., the outer circle and its novelty), and (2) *how* a system accomplishes that (i.e., the inner circles and their novelty). We consider three important combinations.

Figure 3 illustrates the easiest case, for which interactive systems researchers often strive, and for which interactive systems reviewers are often most comfortable. The illustrated system accomplishes novel functionality (i.e., the outer circle, the *what*), enabled in part by novel techniques (i.e., the inner circles, the *how*). Such contributions can often be motivated either bottom-up (i.e., introducing novel inner techniques and then novel outer functionality they enable) or top-down

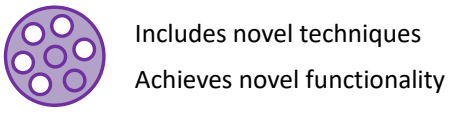

Figure 3: Interactive systems that include both novel functionality (i.e., the outer *what*) and novel techniques (i.e., the inner *how*) can often be motivated and validated in either contribution.

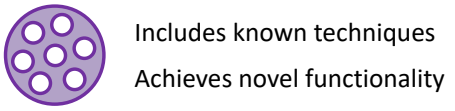

Figure 4: When underlying techniques are known (i.e., the inner *how*), the question is whether their combination in new functionality is a significant contribution (i.e., the outer *what*).

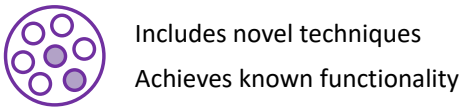

Figure 5: When applied in known overall functionality (i.e., the outer *what*), the question is whether implications of novel inner techniques are a significant contribution (i.e., the inner *how*).

(i.e., introducing novel outer functionality and then novel inner techniques required to achieve that functionality). At the scale of a paper, validation can often focus on either the outer circle (i.e., the *what*) or the inner (i.e., the *how*). Researchers therefore have flexibility in deciding what to highlight. Individual reviewers may prefer novelty with regard to either the *what* (i.e., the outer circle) or the *how* (i.e., the inner circles), but a system that makes novel contributions in both is more robust to such variation in reviewer taste.

Figure 4 illustrates a system that uses known techniques to accomplish novel functionality. The challenge for researchers and reviewers is deciding whether the outer circle (i.e., the *what*) is a significant contribution (i.e., versus the case where both outer and inner circles are already known). Reasonable people may disagree regarding significance of contribution in the outer circle, but it seems inappropriate to dismiss any contribution because "we already know" the inner circles. Such work is sometimes dismissed as "obvious", but if no prior work has previously combined these pieces into the resulting functionality, a more appropriate question is whether that novel functionality is a significant contribution.

Figure 5 conversely illustrates a system that applies novel techniques in the context of known overall functionality. The challenge is deciding whether improvements to the inner circles are a significant contribution, and it seems inappropriate to dismiss that potential because the outer circle "has been done". Such work is sometimes dismissed as "just engineering", but a more appropriate question is whether the implications of novel inner techniques make them a significant contribution. These can include advances in performance or robustness, but a researcher is often wise to show how advances impact functionality (i.e., to convert this into our first case by showing how inner contributions enable novelty in the outer circle).

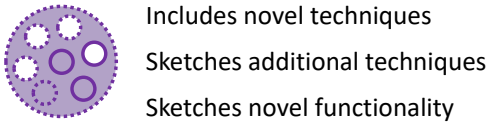

Figure 6: Many research systems are sketches supporting a research contribution. This example includes concrete and novel contributions (i.e., solid inner filled circles) as part of a larger system sketching novel functionality. Dashed elements need additional work before they are fully achieved, but the sketch allows critique to focus on the current contributions.

Scale and Sketching

Given the above forms of contribution, we now return to the problem of scale and complexity. If a researcher's intended contribution is the outer circle (i.e., the *what*), then elements of inner circles may be irrelevant. For example, consider a system that requires persistent storage, but has no interesting requirements of that storage. A decision to use a local file, a local database, or a cloud database will impact the system's code, but is irrelevant to its research contribution. Conversely, if the intended contribution is an inner circle (i.e., the *how*), details of an outer circle may be irrelevant. Overall, a researcher is generally not developing a product and will make choices that impact code according to whatever is most expedient without sacrificing the research contribution. Instead of criticizing this as "research code", or demanding unreasonable standards, we must remember the researcher pursues different goals than a product developer.

We believe many interactive systems developed in research should be considered sketches, as described by Buxton [2]. Sketches allow rapid exploration of many possibilities, with each sketch surfacing its key properties for critique. Sketches are also intentionally left ambiguous in many ways, with additional details to be defined if the idea is further pursued. This property expedites sketching because it allows proceeding without spending time or resources defining details. It also improves critique by remaining focused on important aspects.

Figure 6 extends prior examples to illustrate this, using dashed circles to show sketched elements. The system includes novel and concrete techniques, but other elements remain sketched. The system works well enough to demonstrate the proposed functionality and to validate the novel techniques that were developed. But fully achieving its proposed functionality still requires additional work implementing known techniques (i.e., dashed inner empty circles) and additional research addressing remaining challenges (i.e., dashed inner filled circles). Most research systems are sketches in this regard, emphasizing key contributions while leaving other aspects underdeveloped. Many demonstrations are also sketches, aiming to validate the contribution of an inner circle by sketching multiple outer circles that are potentially enabled.

Visions and Realizations

Even more than a sketch, a technology vision suggests a direction while leaving many unanswered questions in how such a vision will actually be achieved. Figure 7 illustrates this with larger holes in the vision. Realizing the vision thus

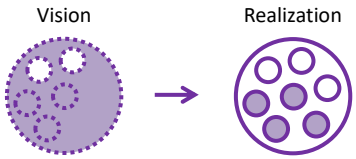

Figure 7: Research visions leave larger holes, in the form of more questions that need to be answered to realize the vision.

requires both: (1) research that addresses known challenges (i.e., circles that were sketched in the vision), and (2) research that defines and then addresses the larger holes in the vision.

Although the difference between a sketch and a vision is obvious at the extremes, the boundary between them is unclear and likely based in a judgment regarding the size of the holes. In visions with larger holes, it is increasingly likely the outer circle (i.e., the *what*, the vision) will be significantly changed in its realization (e.g., by implications of the specific inner circles developed in pursuing the vision), and therefore should often be considered a novel contribution. But even when a realization remains close to an original vision, there are often contributions in the inner circles needed to achieve that vision.

DISCUSSION

We have aimed to unpack common forms of contribution in interactive systems research, arguing those contributions are: (1) distinct from each other, and (2) often distinct from the code that is used as a medium to demonstrate or manifest a contribution. Our reflection is motivated by challenges we observe in researchers and reviewers conflating these aspects of work. If researchers believe their contributions are in one regard, while reviewers consider them in another, resulting mutual frustration generally undermines progress in our field.

In focusing on differentiating the *what* from the *how*, we have intentionally not engaged questions of validating either form of contribution. We have also not engaged questions of how much sketching is acceptable in a system, versus what aspects of a system must be more complete to be considered a contribution. Such questions seem better addressed in more specific contexts where they can be grounded in details of the work, but several points can be discussed more generally.

Irrelevant Detail and Irrelevant Replication

Some common pitfalls emerge when: (1) the difference between code and contribution is confused, or (2) notions of replicability in experimental contexts are misapplied in interactive systems. Consider the sketch from Figure 6, with a pair of novel techniques (i.e., solid filled inner circles). These techniques are intended as contributions and should be thoroughly reported so they can be understood and considered by reviewers. But desire for thoroughness sometimes leads reviewers to probe irrelevant details of a sketch. A known technique (i.e., a solid empty inner circle) has previously been validated. Use of a known technique is further validation, and it is likely inappropriate to expect the current work to explicitly revisit its validation. Similarly, the sketched inner circles should be considered only to the extent they impact the intended contribution. The choice of

how to implement these techniques, in the current sketch and in any future realizations, obviously impacts the code of such systems. But probing at unresolved details of these sketches, or attempting to evaluate such irrelevant details, is often obscuring the work's actual contribution.

Promoting Diverse Forms of Contribution

The long-term health and impact of our field requires all of the forms of contribution considered here. Visions can inspire other researchers to pursue a direction, allowing the field to explore and understand that space more quickly and effectively than waiting for the original researcher to "fill in the holes". Research systems that sketch relationships between *what* (i.e., their outer circles) and *how* (i.e., their inner circles) similarly allow the field to better explore and understand such relationships without them being hindered or obscured by other irrelevant details. But visions and sketching have limits. Achieving a full realization may reveal that prior sketches were incomplete or incorrect in important aspects of an idea. Full realizations also allow confident incorporation of prior work in new explorations, a contrast to stacking sketches that may eventually crumble under their own incompleteness. Full realizations thus enable both direct impact of the current research and future exploration of additional research.

From this perspective, it seems strange and unfortunate for our field to simultaneously lament: (1) a perception among researchers that innovation and novelty are limited by questions of validation that seem to work against exploring new directions, and (2) a perception among researchers that progress is limited by novelty fetishes that seem to work against building upon what is already known in pursuing deeper understanding and impact. We obviously need both, need authors and reviewers to be clear which is pursued, and need discussions of contribution and validation to be based in how specific work contributes to this balance.

PRIOR INTERACTIVE SYSTEMS RESEARCH

We look forward to workshop discussions of these and other perspectives on interactive systems. As background, our prior interactive systems research includes toolkits for sensor-based statistical models [5], exploration of tool challenges applying machine learning in everyday applications [12], techniques enabling graphical interfaces composed of mutually untrusted elements [13], a gesture authoring tool [10], and techniques and tools enabling pixel-based interpretation and runtime modification of graphical interfaces (e.g., [3,4,16]).

ACKNOWLEDGMENTS

This reflection began as a discussant talk in the HCIC 2011 workshop. We thank Sean Munson and Jacob O. Wobbrock for their persistent encouragement to prepare a written version of this reflection. This work was supported in part by the National Science Foundation under awards IIS-1053868 and SCH-1344613.

REFERENCES

1. Michael S. Bernstein, Mark S. Ackerman, Ed H. Chi, and Robert C. Miller. (2011). The Trouble with Social Computing Systems Research. *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems (CHI 2011)*, 389–398.
2. Bill Buxton. (2007). *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann.
3. Morgan Dixon and James Fogarty. (2010). Prefab: Implementing Advanced Behaviors Using Pixel-Based Reverse Engineering of Interface Structure. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2010)*, 1525–1534.
4. Morgan Dixon, Gierad Laput, and James Fogarty. (2014). Pixel-Based Methods for Widget State and Style in a Runtime Implementation of Sliding Widgets. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2014)*, 2231–2240.
5. James Fogarty and Scott E. Hudson. (2007). Toolkit Support for Developing and Deploying Sensor-Based Statistical Models of Human Situations. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2007)*, 135–144.
6. Saul Greenberg and Bill Buxton. (2008). Usability Evaluation Considered Harmful (Some of the Time). *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2008)*, 111–120.
7. Jonathan Grudin. (1988). Why CSCW Applications Fail: Problems in the Design and Evaluation of Organizational Interfaces. *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW 1988)*, 85–93.
8. Scott E. Hudson and Jennifer Mankoff. (2014). Concepts, Values, and Methods for Technical Human-Computer Interaction Research. In *Ways of Knowing in HCI*. Springer, 69–93.
9. Yang Li. (2010). Protractor: A Fast and Accurate Gesture Recognizer. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2010)*, 2169–2172.
10. Hao Lü, James Fogarty, and Yang Li. (2014). Gesture Script: Recognizing Gestures and their Structure Using Rendering Scripts and Interactively Trained Parts. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2014)*, 1685–1694.
11. Dan R. Olsen, Jr. (2007). Evaluating User Interface Systems Research. *ACM Symposium on User Interface Software and Technology (UIST 2007)*, 251–258.
12. Kayur Patel, Naomi Bancroft, Steven M. Drucker, James Fogarty, Andrew J. Ko, and James A. Landay. (2010). Gestalt: Integrated Support for Implementation and Analysis in Machine Learning. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2010)*, 37–46.
13. Franziska Roesner, James Fogarty, and Tadayoshi Kohno. (2012). User Interface Toolkit Mechanisms for Securing Interface Elements. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2012)*, 239–250.
14. Mark Weiser and John Seely Brown. (1996). The Coming Age of Calm Technology. Xerox PARC.
15. Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. (2007). Gestures Without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2007)*, 159–168.
16. Xiaoyi Zhang, Anne Ross, James Fogarty, Anat Caspi, and Jacob O. Wobbrock. (2017). Interaction Proxies for Runtime Repair and Enhancement of Mobile Application Accessibility. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI 2017)*, To Appear.

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.

Paper accepted, toolkit abandoned.

Roman Rädle & Clemens N. Klokmose
Department of Digital Design and Information Studies
Aarhus University, Aarhus, Denmark
roman.raedle@cc.au.dk, clemens@cavi.au.dk

ABSTRACT
Developing and maintaining HCI toolkits is a challenging task. In this position paper, we present three types of challenges that potentially turn toolkits into abandonware: organizational, institutional, and technological challenges. We derive the challenges from our own experiences in developing HCI toolkits and they have been consolidated based on the three sample toolkits Squidy, HuddleLamp, and Webstrates. We describe the overall goals of the toolkits and their application areas, report on their uses and the current state of development, and link them to the challenges. We conclude with questions for the HCITools workshop.

Author Keywords
Abandonware; toolkits; frameworks; libraries; enabling technology.

ACM Classification Keywords
H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous

INTRODUCTION
Third-party software like libraries, frameworks, and toolkits[1] often make the foundation of research prototypes used in HCI research. They enable researchers to explore new ways and ideas of computing systems to improve interaction between humans and computers. More general, in the fast-advancing field of computer science it is nearly impossible nowadays to have a complete understanding of inner-workings of computing hardware or software algorithms. A third-party software thus can provide a level of abstraction that allows researchers to build prototypes with complex computations – like 3d transformation, server/client or peer-to-peer communication, image- and video-processing, artificial intelligence, or machine learning – even with only basic programming skills.

[1]We will use toolkit, library, and framework synonymously in this paper as existing definitions lack clear distinction or contradict each other.

HCITools: Strategies and Best Practices for Designing, Evaluating and Sharing Technical HCI Toolkits. Workshop in conjunction with CHI '17.

For example, OpenCV is a framework that implements hundreds of computer vision algorithms in C/C++. It provides them to the community in a comparatively low-effort application programming interface (API) and for real-time computer vision processing. In the past, the OpenCV library was mainly for vision-based processing, hence the name *open source computer vision*. Nowadays, the library also implements neural networks for machine learning. Programmers can use the library without prior or extensive knowledge in computer vision and i.e. apply image processing algorithms to an input image such as canny edges. Beyond the C/C++ API, various open source libraries bind to OpenCV and make it accessible in other programming languages like Java or C# (e.g., Emgu CV).

OpenCV is a great example of a successful project that went from research to industry, and myriads of commercial, open source or research projects use it. A large community maintains it, continuously develops new algorithms, and publishes stable releases on a public website.

Large communities maintaining software, however, is very unusual for projects originating from research. This entails a risk, especially when toolkits are part of a research paper contribution. From our experience, it happens very often that building a project from source code or running it fails due to "out-of-date" third-party libraries, libraries that are not available for download anymore, or projects do not support the latest version of a runtime environment like the Common Language Runtime (CLR for .NET), Java Runtime Environment (JRE), or use deprecated Web standards. Such toolkits developed in research projects are deemed to end as abandonware where the authors stop working on it and reported bugs will not get fixed after the associated paper was accepted.

Despite this risk, HCI toolkits have many potentials and play an essential role in advancing HCI. They are often a great source of inspiration and help researchers to think outside the box. Duval argues that *"[...] innovation in general – is that [...] sometimes just showing somebody a concept is all that you have to do to start an evolutionary path. And once people get the idea of 'hey we can do that', then somebody does something, somebody does something better, that just keeps developing."* (Duval 2011 as cited in [1])

In retrospect, much of our research has been inspired by HCI toolkits. The Proximity Toolkit by Marquardt et al. [7] is an excellent example (the accompanying paper has around 70 citations at ACM DL). We have used it for a study to investigate the effect of body movements on users' spatial memory while navigating large virtual information spaces [9]. It al-

lowed as to rapidly test various interaction techniques during a focus group and leverage the group to discuss pros and cons of each interaction. The toolkit helped to choose an interaction technique appropriate to answer the research question.

In the remainder of this position paper, we will introduce three challenges that are potential causes for abandonware, report our experience in creating software toolkits for HCI, and conclude with questions that we would like to discuss with participants during the workshop.

CHALLENGES AND OPPORTUNITIES

Resonating with Duval's statement above, we believe that HCI toolkits are catalysts for innovation and spark new ideas. However, in this position paper, we also want to emphasize on three challenges that HCI toolkits face before they eventually become abandonware.

Organizational

Open source projects need a large community to maintain code, and ideally more than one responsible person (maintainers) who take the lead on the project and divide the workload equally among each other. They are responsible for accepting pull requests, continuously check the quality of the source code, make sure the project builds correctly, put new releases online, and define goals for future developments. An important and often criticized aspect of open source software is the lack of proper documentation or working examples. In the case of poor or no documentation, programmers who use the toolkit have to dig through the source code manually to decode and understand how it should be used.

Institutional

The reusability of research toolkits is often limited for scenarios presented in their accompanying research papers, which impedes the possibility of using them in new and meaningful ways. We believe there are a number of institutional factors that impede the continued development of toolkits. Firstly, time is a precious resource in academia and scientists who pursue an academic career are often confronted with the balancing act between community service by offering toolkits to be used freely and advancing the career by increasing publication count and boosting the h-index. Secondly, paying developers to maintain a toolkit or renting proper build infrastructure (e.g., continuous integration server) is costly and may be difficult to budget on research grants. Thirdly, continued work on an already published system or toolkit may be discouraged as it is considered incremental rather than novel work.

Technological

The technological challenges also often impede with the reuse of toolkits. For example, runtime environments deprecate or special hardware is required to run it but is not available for purchase anymore. Or infrastructure like code repositories disappear. Changes in the technological landscape can also impact the reuse of toolkits when technology emerges, and suddenly previously popular technology fades away. A few years ago TCL/TK was fashionable, but nowadays it is rarely used. This can likewise happen to current mainstream technology (e.g., HTML5 or Unity3d).

EXPERIENCE WITH HCI TOOLKIT DEVELOPMENT

We have worked on several research toolkits that reached various levels of maturity, which also led to a consolidation of challenges as mentioned earlier. We present three of our toolkits by describing their overall goal, report on their uses and the current state of development, and link them back to the challenges.

Squidy – Open Source but Concluded

Squidy [6][2] is an interaction library which eases the design of post-WIMP or "natural user interfaces" by unifying various device drivers, frameworks, and tracking toolkits in a common toolkit. It provides a visual design environment for visually connecting input devices to filters (e.g., a Kalman filter or a gesture recognizer) and map them to an output (e.g., Microsoft PowerPoint) (see Figure 1). Squidy offers diverse input modalities such as multi-touch input, pen interaction, speech recognition, laser pointer-, eye- and gesture-tracking. The visual user interface hides the complexity of the technical implementation from the user by providing a simple visual language based on high-level visual data flow programming combined with zoomable user interface concepts. Furthermore, Squidy offers a collection of ready-to-use devices, signal processing filters, and interaction techniques. The trade-off between functionality and simplicity of the user interface is especially addressed by utilizing the concept of semantic zooming which enables dynamic access to more advanced functionality on demand. Thus, developers, as well as interaction designers, are able to adjust the complexity of the Squidy user interface to their current need and knowledge.

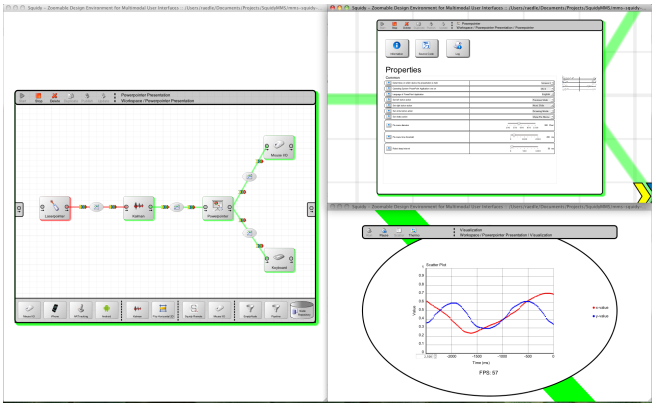

Figure 1. Squidy's user interface with a pipe-and-filter metaphor to visually connect nodes (filters and input and output devices) (left), a property view per node to change settings during runtime (top-right), and a debug view to inspect current data flow between nodes (bottom-right).

Squidy was used to design the interaction for the artistic installation Globorama [5][3]. It was deployed for a week during the Ideenpark 2008 "Zukunft Technik Entdecken" (Future Discover Technology) at the fair trade center in Stuttgart, which was sponsored by ThyssenKrupp. The installation was

[2] [6] has 54 citations on Google Scholar (accessed 02/17/2017)
[3] [5] has 54 citations on Google Scholar (accessed 02/17/2017)

exposed from May 17th to May 25th, and around 290.000 people were visiting the Ideenpark. It was used to allow a single user to control a world map application. This application was projected onto a 360-degree panoramic display where users could navigate to particular locations all over the world.

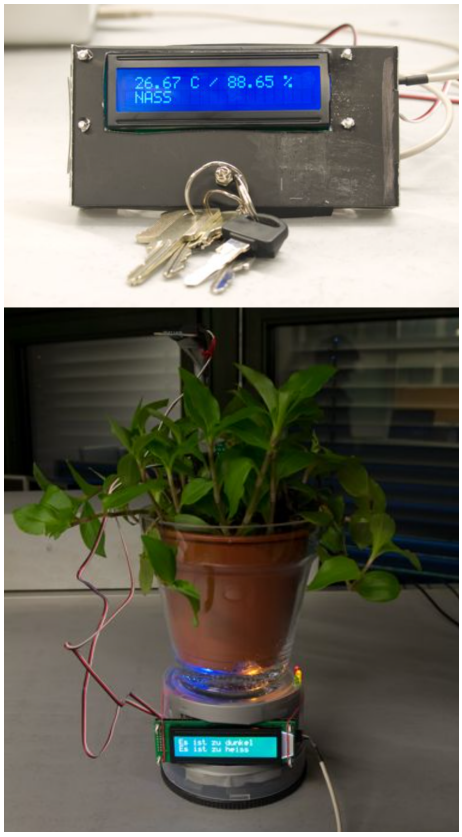

Figure 2. Everyday widgets: advising key holder (top), TakeCare flower (bottom). Interaction were designed with Squidy.

Squidy was also employed during several university classes and courses to allow non-programmers to design everyday widgets (see Figure 2). For example, an advising key holder (top) reports on weather and outside lighting conditions or a TakeCare flower pot (bottom) gives agency to a flower, which expresses feelings like "It is too dark" or "I'm hot."

A summative evaluation of Squidy showed the applicability also for programmers with little programming experience. It offers a low barrier to entry for beginners with its visual pipe-and-filter metaphor (low threshold [8]), but also enables experienced programmers to implement advanced interaction techniques (high ceiling [8]).

However, the project is no longer maintained. Keeping up-to-date with third-party libraries of supported input and output devices was tedious (technological). Also, build- and continuous integration servers had to be maintained (organizational). The project ended with the end of the research funding and authors had to move on with other research projects (institutional).

HuddleLamp – Open Source and Ongoing

Another example is HuddleLamp [10][4], which is a desk lamp with an integrated low-cost RGB-D camera that detects and identifies mobile displays (e.g., smartphones or tablets) on tables and tracks their positions and orientations with sub-centimeter precision. Users can add or remove off-the-shelf, web-enabled devices in an ad-hoc fashion without prior installation of custom hardware (e.g., radio modules, IR markers) or software. Because of HuddleLamp's web-based pairing, adding a new device is simply done by opening an URL on the device and putting it on the table so that it is visible to the camera.

HuddleLamp was demonstrated at ITS 2014 and has been used for research studies (e.g., [11]). Apart from the hybrid sensing pipeline presented in the research paper, HuddleLamp contributes a visual editor to rapidly change the pipeline and test and try out different settings for each processing step (see Figure 3).

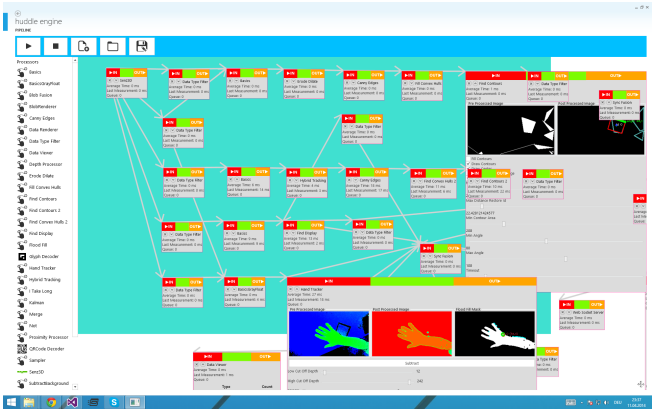

Figure 3. HuddleLamp's visual editor to change the processing pipeline and to rapidly test different processing filter settings.

This project is still ongoing and used in research projects, but the public source code is currently not maintained. Unfortunately, Creative does no long manufacture the supported Senz3D camera (technological), which requires implementing a new input node to allow tracking with an alternative camera (e.g., Microsoft Kinect v2).

Webstrates – Open Source and Highly Active

Webstrates [4][5] is the most recent toolkit and under active development. It has reached a state beyond prototypical use. Webstrates presents an alternative take on the future of web use and development. In Webstrates, changes to the Document Object Model (DOM) of webpages (called webstrates) are persisted across reloads and synchronized to other clients of the same webstrate. This includes changes to embedded JavaScript and CSS. Webstrates addresses the challenge that while web technologies have become increasingly powerful, technologies such as server-side scripting have turned the web into a sophisticated application platform rather than a user-driven hypermedia system [2] as envisioned by Tim-Berners Lee.

[4] [10] has 56 citations on Google Scholar (accessed 02/17/2017)
[5] [4] has 12 citations on Google Scholar (accessed 02/17/2017)

Webstrates was originally developed to prototype a reiteration of Kay and Goldberg's vision of interaction with computers as being interaction with personal dynamic media [3], but with an emphasis on shareability. We therefore referred to this reiteration as *shareable dynamic media* (SDM). The core principles of SDM are *"Malleability: users can appropriate their tools and documents in personal and idiosyncratic ways; Shareability: users can collaborate seamlessly on multiple types of data within a document, using their own personalized views and tools; and Distributability: tools and documents can move easily across different devices and platforms."* [4]

Since the original publication Webstrates has gone from being a proof-of-concept to a robust web framework used in multiple research projects internationally and by the paper authors for their daily activities (e.g., note taking, lectures, presentations, teaching material, and rapid prototyping).

To become more than a one-time affair, a full-time developer implements new features and maintains the quality of the code. The development is covered by research funding and costs approximately $70.000 per year. The paper authors also dedicate a significant portion of their time to advance Webstrates and to define future directions together with the full-time developer.

Potential Measure of HCI Toolkits

In contrast to Squidy and HuddleLamp, the public interest in Webstrates and its community of users is growing. At the time of writing, the GitHub repository has eight forks, and more than 50 people bookmarked it. Of course, GitHub forks and bookmarks should not be stressed too much as a reliable measure of the success of Webstrates. But unlike commercially sold hardware and software and their success measured by a company's revenues, it is difficult to quantify the success of an HCI toolkits. Measuring a toolkit's success could be based on a jury assessing it according to pre-defined metrics (e.g., generalizability to other application areas) or ranking it by how often it is used in research prototypes.

CONCLUSION

As argued in this position paper, we believe that the viability of HCI toolkits developed in research is largely constrained by the three presented challenges: organizational, institutional, and technological. However, we also believe in the power of HCI toolkits. They serve as factual manifestations of the advancement of socio-technical systems and help the HCI community (and industry) to crystallize a shared vision for HCI, and herewith we are emphasizing on Duval's statement at the beginning of this position paper.

In the workshop, we would like to share our experiences in building HCI toolkits and discuss the presented challenges with participants. We would further like to spark discussion around the legacy of HCI toolkits. How can we, as HCI research community, preserve the genealogy of HCI toolkits? Does it make sense to start an encyclopedia of HCI toolkits that answer questions like (i) What does a particular toolkit do?, (ii) What did the authors inspire to create it?, and (iii) How did it push the field forward?

REFERENCES

1. Barnet, B., and Moulthrop, S. *Memory Machines: The Evolution of Hypertext*. Anthem Press, 2013.
2. Bouvin, N. O., and Klokmose, C. N. Classical hypermedia virtues on the web with webstrates. In *Proceedings of the 27th ACM Conference on Hypertext and Social Media*, HT '16, ACM (New York, NY, USA, 2016), 207–212.
3. Kay, A., and Goldberg, A. Personal dynamic media. *Computer 10*, 3 (Mar. 1977), 31–41.
4. Klokmose, C. N., Eagan, J. R., Baader, S., Mackay, W., and Beaudouin-Lafon, M. Webstrates: Shareable dynamic media. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, UIST '15, ACM (New York, NY, USA, 2015), 280–290.
5. König, W., Rädle, R., and Reiterer, H. Interactive design of multimodal user interfaces. *Journal on Multimodal User Interfaces 3*, 3 (2010), 197–213.
6. König, W. A., Rädle, R., and Reiterer, H. Squidy: A zoomable design environment for natural user interfaces. In *CHI '09 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '09, ACM (New York, NY, USA, 2009), 4561–4566.
7. Marquardt, N., Diaz-Marino, R., Boring, S., and Greenberg, S. 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, 2011), 315–326.
8. Myers, B., Hudson, S. E., and Pausch, R. Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact. 7*, 1 (Mar. 2000), 3–28.
9. Rädle, R., Jetter, H.-C., Butscher, S., and Reiterer, H. The effect of egocentric body movements on users' navigation performance and spatial memory in zoomable user interfaces. In *Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces*, ITS '13, ACM (New York, NY, USA, 2013), 23–32.
10. Rädle, R., Jetter, H.-C., Marquardt, N., Reiterer, H., and Rogers, Y. Huddlelamp: Spatially-aware mobile displays for ad-hoc around-the-table collaboration. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*, ITS '14, ACM (New York, NY, USA, 2014), 45–54.
11. Rädle, R., Jetter, H.-C., Schreiner, M., Lu, Z., Reiterer, H., and Rogers, Y. Spatially-aware or spatially-agnostic?: Elicitation and evaluation of user-defined cross-device interactions. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, ACM (New York, NY, USA, 2015), 3913–3922.

Technology and Community in Toolkits for Musical Interface Design

Andrew P. McPherson
Centre for Digital Music
Queen Mary University of London, UK
a.mcpherson@qmul.ac.uk

Fabio Morreale
Centre for Digital Music
Queen Mary University of London, UK
f.morreale@qmul.ac.uk

ABSTRACT

This position paper discusses toolkits for creating digital musical instruments. Musical interaction imposes stringent technical requirements on interactive systems, including high spatial and temporal precision and low latency. Social and community factors also play an important role in musical interface toolkits, including design sharing and the ability of performers and composers to count on the longevity of an instrument. This paper presents three examples of musical interface toolkits, including our own Bela, an open-source embedded platform for ultra-low-latency audio and sensor processing. The paper also discusses how the requirements of specialist musical interface toolkits relate to more general HCI toolkits.

ACM Classification Keywords

H.5.5. Sound and Music Computing: Systems; H.5.1. Multimedia Information Systems: Evaluation/Methodology

Author Keywords

Toolkit, digital musical instrument, embodied interaction, maker community, latency, longevity, pluggable communities.

INTRODUCTION

Musical interaction presents a number of interesting opportunities and challenges for HCI. Many digital musical instruments (DMIs), like their acoustic counterparts, are useful case studies in embodied interaction: extended practice leads to the instrument becoming a transparent extension of the musician's body, where the operations of manipulating the instrument become automatic, allowing the musician to focus on higher-level musical actions [20]. Musical interaction also places stringent technical demands on digital systems, including spatial and temporal precision, high sensor and audio bandwidth, predictability and low latency [7].

Toolkits for creating DMIs have become increasingly common [18, 21, 19, 2, 23, 13, 5], with different projects aimed at a variety of musical contexts and technical skill levels. These toolkits share a desire to enable musicians who are not engineers to create their own high-quality instruments by solving common technical challenges and optimising for the qualities musicians find important.

DMI toolkits provide a common platform for instrument creators to share designs, which serves both research and artistic goals. It takes time for performers to acquire expertise on a new instrument, and encouraging composers to write music for a new instrument requires assurance that the instrument will remain in existence for the piece to continue to be played [15].

In the DMI community, published papers typically contain insufficient detail to fully replicate an instrument design, especially in regard to aesthetic choices and fine details of craftsmanship which are important to the performer experience but might not follow established scientific processes. Some online DMI repositories have been created[1] inspired by more general sharing platforms such as Instructables,[2] however this remains an outstanding challenge for the DMI community. DMI toolkits, by providing a common platform for designers, reduce the barriers to exchanging fully functioning designs.

This position paper explores the current state of musical interface toolkits and their relation to more general HCI toolkits.

[1]For example, Muzhack by Arve Knudsen: https://muzhack.com
[2]http://www.instructables.com

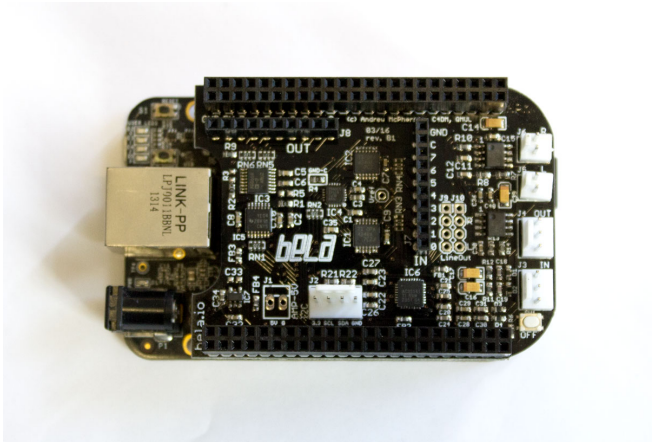

Figure 1. Bela, which consists of a custom hardware board ("cape") on a BeagleBone Black running specialised software.

CHI 2017, May 6-11, 2017, Denver, CO, USA.

ACM ISBN 978-1-4503-4655-9/17/05 ...$15.00.
http://dx.doi.org/10.1145/3025453.3026056

We discuss both technical and social challenges for toolkit design, including the central role of the user community in sustaining a viable toolkit. We then present three established toolkits, including our own Bela[3] (Figure 1), an open-source embedded hardware platform for ultra-low-latency audio and sensor processing [13]. We conclude with suggestions for toolkit designers and topics for discussion at the workshop.

THREE CHALLENGES FOR HCI TOOLKITS

The technical and social challenges we discuss in this section are particularly important for musical interface toolkits, but also apply to HCI toolkits more generally.

Latency

Action-to-sound latency (delay) is a critically important factor in DMI design. Wessel and Wright recommended that DMIs exhibit no more than 10ms latency with no more than 1ms jitter (variation in latency) [24]. An experiment with a digital percussion instrument confirmed these recommendations, showing that performers rated an instrument with consistent 10ms latency the same as one with under 1ms latency, but that 20ms and 10ms $\pm$ 3ms of jitter both rated significantly lower, even when performers did not identify an audible delay [7].

The effects of excess latency include making the instrument feel less responsive, reducing its perceived quality, and potentially disrupting the sensorimotor processes needed for accurate performance. Latency can also be used for deliberate effect: for example, in a multimodal smartphone interface, adding latency to the tactile feedback channel made virtual buttons feel heavier [8].

Surprisingly, 15 years after Wessel's recommendation of less than 10 ± 1ms latency, many commonly used tools for creating DMIs do not meet this standard [14] with jitter posing a particular problem. Achieving low and consistent latency also remains an issue for embodied interaction in other contexts.

Longevity

Many experimental DMIs are designed to be used for only a few performances, but some continue to be used for many years. Once a DMI is created, there is typically little incentive to upgrade its hardware and software, as any change to form or behaviour might disrupt its familiarity to the performer. When DMIs are built using laptops or mobile devices, however, the aim of long-term stability comes into conflict with the need for regular system updates.

Hardware toolkits based on embedded processors (e.g. [2, 13]) provide a potential solution by allowing the instrument to operate standalone without a computer. Ideally, the instrument can be maintained indefinitely on this dedicated hardware. In practice, keeping a DMI toolkit operational over many years remains a challenge. Toolkit design considerations include: high reliability, minimum external hardware or software dependencies, rapid setup time (especially when revisiting an instrument after a long period of disuse) and availability of spare parts. The last consideration points to the value of open-source hardware designs [17], or at least the use of commodity hardware where possible.

[3] http://bela.io

Another question for toolkit designers is whether they seek to support prototyping, extended use, or both. Few mass-market commercial products would be built with HCI toolkits, but at least in the DMI community, it is not uncommon for a toolkit to be used for both prototyping and subsequent production on a scale of dozens or even hundreds of instruments.

Community

The utility of a toolkit cannot be assessed solely by its technical specifications, nor even by the quality of its documentation. An active and cooperative user community also plays a major role in making a toolkit useful to new designers [4, 10]. The success of open-source platforms like Arduino and Processing owes as much to their vibrant online communities as to their engineering features. These communities contribute by publishing example code, providing technical support, creating hardware and software accessories, and helping the original designers maintain the core platforms.

In [17] we explore the process of creating a community around an open-source platform based on our experiences with Bela [13] (described below). We observed that the Bela community grew not only around the intrinsic features of Bela itself, but also through connecting to other established open-source tools. We describe this process as *pluggable communities*: growing a new community in discrete leaps by leveraging established communities around other tools.

THREE MUSICAL INSTRUMENT TOOLKITS

Many musical interface toolkits have been created. The three mentioned here are all open source, publicly available toolkits for creating standalone musical instruments, and all three are still in regular use.

Satellite CCRMA

Satellite CCRMA [2] is a platform for building musical instruments which eliminates the need for a computer. It consists of an ARM Linux distribution with several popular audio programming environments preinstalled, and it is accompanied with a set of example materials for creating instruments [1]. Originally created for the BeagleBoard[4] single-board computer, it has since been released for the popular Raspberry Pi. Instruments built with Satellite CCRMA frequently make use of an Arduino[5] microcontroller board for gathering sensor data, with the Raspberry Pi responsible for audio processing.

Satellite CCRMA is in regular use by DMI designers. Its website[6] links to performances of instruments built with it, and a wiki and online forum provide a means for the community to share knowledge. Its use of the widely available Raspberry Pi, with no dependence on custom hardware, means that the platform itself should be maintainable for years to come, and that software should be easily shared amongst different users. Leaving to the designer decisions on sensors and other external hardware provides significant flexibility, but with the tradeoff of placing responsibility on the designer to maintain and document their own hardware contributions.

[4] http://beagleboard.org
[5] http://arduino.cc
[6] https://ccrma.stanford.edu/~eberdahl/Satellite/

Hoxton OWL

The Hoxton OWL [23] is an open-source programmable audio effects pedal. More recently, it has also been released as a synth module in the popular Eurorack form factor. Like Satellite CCRMA, the OWL is designed for creating musical instruments and audio processing systems, and features a large example library and an active online community.[7]

In contrast to Satellite CCRMA, the OWL is a complete hardware unit based on a custom (though open source) design. Thus, where DMI creators using Satellite CCRMA would likely add their own sensors and other hardware, OWL creators will typically work with the existing controls and focus on software development. Though this reduces the variety of interactive systems that can be created, it makes design sharing especially straightforward. Since the OWL pedal is a self-contained device in a robust stage box, it is likely that any designs running on it can be maintained for many years, though the ability to edit code on the device will remain dependent on a working computer-based compiler toolchain.

Bela

Our lab has created Bela [13] (Figure 1), an embedded platform for ultra-low-latency audio and sensor processing. Bela is based on the BeagleBone Black[8] single-board computer with a custom expansion board ("cape") providing stereo audio I/O with onboard speaker amplifiers, 8 channels each of 16-bit analog I/O and 16 digital I/Os. It uses the Xenomai real-time Linux kernel extensions to process audio and sensor data at higher priority than anything else on the board, including the Linux kernel itself.

The signature feature of Bela is its extremely low latency, under 1ms round-trip for audio or down to 100μs for analog and digital data, with less than 25μs of jitter, outperforming other computer audio environments [14]. It also features an onboard, browser-based IDE with support for C/C++, PureData and SuperCollider programming languages, and an in-browser oscilloscope. Like Satellite CCRMA, Bela is designed for creating self-contained musical instruments, where the designer attaches their own sensor hardware and Bela handles all the computation that would normally be performed by a laptop and a microcontroller board like Arduino.

The platform that later became Bela was originally created for the D-Box, a musical instrument designed to be modified and hacked by the performer [25]. In a CHI 2017 paper [17], we describe the process of developing it from a single-function device to a maker community platform, gradually broadening its scope and improving usability. In April 2016, Bela successfully launched on Kickstarter with the support of over 500 backers. Hardware and open-source design plans are available for sale and download,[9], and we maintain a library of example projects and an online forum[10] for community support and idea exchange.

[7] https://hoxtonowl.com

[8] https://beagleboard.org/black

[9] https://shop.bela.io and http://bela.io/code, respectively

[10] http://forum.bela.io

OUR RELATED WORK

The Augmented Instruments Laboratory,[11] led by the first author, is a research team within the Centre for Digital Music at Queen Mary University of London. An augmented instrument is a traditional musical instrument whose capabilities have been technologically extended, maintaining the familiarity and cultural connotations of the original instrument while extending its capabilities.

In addition to Bela [13], described in the preceding section, our previous projects include several augmented instruments including the magnetic resonator piano [15], an electromagnetically-actuated acoustic grand piano and TouchKeys [12], a sensor kit adding multi-touch sensing to the surface of the piano keyboard. Our research also encompasses studies of performer-instrument interaction in solo [7] and group [16] settings and studies of audience perception of performance [3].

CONCLUSION: POSITION ON TOOLKITS

As creators of an open-source DMI toolkit, we are especially interested in the potential for toolkits to broaden access to interactive system design. HCI toolkits, including those for creating musical interfaces, contribute to and benefit from larger trends in maker culture [9]. Here we set out two specific arguments for possible discussion at the workshop.

Toolkits need a two-way dialogue with their communities

User-centred and participatory design methodologies have long histories in HCI, so to say that a toolkit should respond to the needs of its community borders on cliché. In fact, we would argue that there is a risk in being *too* reactive to perceived user requirements. It has long been observed that people use technology in unexpected ways, and this process of appropriation has influenced HCI design methods [6]. Similarly, the history of music is replete with examples of people playing instruments in unexpected ways [25]. Performers, upon encountering a new instrument, explore its creative opportunities and constraints [11], but they should not be expected to imagine hypothetical capabilities of instruments that do not yet exist [15].

We argue that HCI toolkits, while being sensitive to community needs, should also express the creative intentions of the toolkit designer. This way, the designer not only contributes new ideas back to the community, they also provide unique "signature features" that may improve the uptake of their tools [17].

No toolkit is aesthetically neutral

Every musical instrument encourages certain possibilities while discouraging others. Some constraints are obvious: the piano can only play 88 discrete notes, and does not allow the performer to shape them after they are struck. Others are less obvious: patterns of notes in piano music are typically those which fit the shape of the hand, which are different than the patterns likely to be convenient on a wind or string instrument. Similarly, even if two DMIs can control identical dimensions of the sound, their differing physical designs might encourage different choices of actions.

[11] http://www.eecs.qmul.ac.uk/~andrewm

Tuuri et al. [22] distinguish between *push effects* which force or guide the user to particular choices, versus *pull effects* which relates to the ease of conceiving how an action relates to an output. Music programming languages, supposedly able to create any sound, may nonetheless exhibit strong pull effects by making certain structures and actions easier than others. For that reason we might speculate that every computer music language has its own signature sound.

More broadly, we would argue that toolkit designers can and should embrace the aesthetic influence of their toolkits. A good toolkit might allow the creation of many different types of systems with widely varying aesthetics, but certain possibilities will always be more obvious than others. Rather than striving for an elusive neutrality, toolkit creators might do best to acknowledge their own personal outlook and the influence it is likely to have on designers and end users.

ACKNOWLEDGEMENTS

This work was funded by EPSRC under grant EP/N005112/1 (Design for Virtuosity: Modelling and Supporting Expertise in Digital Musical Interaction).

REFERENCES

1. Edgar Berdahl. 2014. How to Make Embedded Acoustic Instruments.. In *Proc. NIME*.
2. Edgar Berdahl and Wendy Ju. 2011. Satellite CCRMA: A Musical Interaction and Sound Synthesis Platform.. In *Proc. NIME*.
3. S Astrid Bin, Nick Bryan-Kinns, and Andrew McPherson. 2016. Skip the Pre-Concert Demo: How Technical Familiarity and Musical Style Affect Audience Response. In *Proc. NIME*.
4. Leah Buechley and Benjamin Mako Hill. 2010. LilyPad in the wild: how hardware's long tail is supporting new engineering and design communities. In *Proceedings of the 8th ACM Conference on Designing Interactive Systems*. ACM, 199–207.
5. Filipe Calegario, Marcelo M Wanderley, Stéphane Huot, Giordano Cabral, and Geber Ramalho. 2017. A Method and Toolkit for Digital Musical Instruments: Generating Ideas and Prototypes. *IEEE MultiMedia* 24, 1 (2017).
6. A. Dix. 2007. Designing for appropriation. In *Proc. British HCI Group Conf. on People and Computers*.
7. Robert H Jack, Tony Stockman, and Andrew McPherson. 2016. Effect of latency on performer interaction and subjective quality assessment of a digital musical instrument. In *Proceedings of the Audio Mostly 2016*.
8. Topi Kaaresoja and Stephen Brewster. 2010. Feedback is... late: measuring multimodal delays in mobile device touchscreen interaction. In *International Conference on Multimodal Interfaces*.
9. Stacey Kuznetsov and Eric Paulos. 2010. Rise of the expert amateur: DIY projects, communities, and cultures. In *Proc. NordiCHI*.
10. Silvia Lindtner, Garnet D Hertz, and Paul Dourish. 2014. Emerging sites of HCI innovation: hackerspaces, hardware startups & incubators. In *Proc. CHI*.
11. T. Magnusson. 2010. Designing Constraints: Composing and Performing with Digital Musical Systems. *Computer Music J.* 34 (2010), 62–73. Issue 4.
12. A. McPherson, A. Gierakowski, and A. Stark. 2013. The space between the notes: adding expressive pitch control to the piano keyboard. In *Proc. CHI*.
13. Andrew McPherson and Victor Zappi. 2015. An environment for submillisecond-latency audio and sensor processing on BeagleBone Black. In *Proc. AES 138th Conv.*
14. Andrew P McPherson, Robert H Jack, Giulio Moro, and others. 2016. Action-Sound Latency: Are Our Tools Fast Enough?. In *Proc. NIME*.
15. Andrew P McPherson and Youngmoo E Kim. 2012. The problem of the second performer: Building a community around an augmented piano. *Computer Music Journal* 36, 4 (2012), 10–27.
16. Fabio Morreale, Antonella De Angeli, Raul Masu, Paolo Rota, and Nicola Conci. 2014. Collaborative creativity: The music room. *Personal and Ubiquitous Computing* 18, 5 (2014).
17. Fabio Morreale, Giulio Moro, Alan Chamberlain, Steve Benford, and Andrew P. McPherson. 2017. Building a Maker Community Around an Open Hardware Platform. In *Proc. CHI*.
18. Axel Mulder. 1995. The I-Cube system: moving towards sensor technology for artists. In *Proc. of the Sixth Symposium on Electronic Arts (ISEA 95)*.
19. D. Newton and M. T. Marshall. 2011. Examining How Musicians Create Augmented Musical Instruments. In *Proc. NIME*.
20. Luc Nijs, Micheline Lesaffre, and Marc Leman. 2009. The musical instrument as a natural extension of the musician. In *Proc. Interdisciplinary Musicology*.
21. Dan Overholt. 2006. Musical interaction design with the Create USB interface. In *Proc. ICMC*.
22. Kai Tuuri, Jaana Parviainen, and Antti Pirhonen. 2017. Who Controls Who? Embodied Control Within Human–Technology Choreographies. *Interacting with Computers* (2017).
23. Thomas Webster, Guillaume LeNost, and Martin Klang. 2014. The OWL programmable stage effects pedal: Revising the concept of the on-stage computer for live music performance.. In *Proc. NIME*.
24. D. Wessel and M. Wright. 2002. Problems and Prospects for Intimate Musical Control of Computers. *Computer Music Journal* 26, 3 (2002), 11–22.
25. Victor Zappi and Andrew McPherson. 2014. Design and Use of a Hackable Digital Instrument. In *Proc. Live Interfaces*.

The Toolkit / Audience Challenge

David Ledo, Lora Oehlberg, Saul Greenberg
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada, T2N 1N4
{david.ledo, lora.oehlberg, saul.greenberg}@ucalgary.ca

ABSTRACT
A variety of HCI toolkits help designers and developers author particular styles of interactive systems. However, the design, use and evaluation of toolkits are fraught with many challenges. This paper focuses on a subset of challenges that arise from the fit between the toolkit and its intended audience. These challenges include the skill set of that audience, the resources they have, and how they learn. We illustrate these challenges via three toolkits: Phidgets, d.Tools, and the Proximity Toolkit.

Author Keywords
Toolkits; Prototyping Tools.

ACM Classification Keywords
H.5.2. Information interfaces and presentation (e.g., HCI): Miscellaneous; Prototyping.

INTRODUCTION
Toolkits are means of encapsulating design concepts to help a developer realize particular styles of interaction design without undue effort [2]. The developer chooses a toolkit to design within HCI genres and/or to exploit interaction techniques. Broad genres include GUIs, physical / tangible interfaces, and ubicomp. Interaction techniques are narrower, such as gesture recognition and input sensing. Toolkits range in how they can be accessed [3], and can include:

- ***traditional programming***, usually through coding via a functional or object-oriented API;
- ***coding support tools,*** such as SDKs with interface builders, widget sets, and physical building blocks (e.g. electronics);
- ***authoring tools that minimize coding*** by providing a higher-level means for authoring interactivity or for creating interactive behaviours (e.g. visual programming [5,7,15] or programming by demonstration [4,16]);
- ***high-level tools that supports debugging and understanding of the run-time system state,*** for example, Papier-Mache [6] and the Proximity Toolkit [11] provide visualizations that allows the end-developer to monitor, record, and even modify runtime information (e.g. sensor data, notifications, variables, etc.).

Yet there is another important way that toolkits vary: their intended audience. Understanding the toolkit's audience is critical, for it will influence how the toolkit will be used, what support tools should be offered, learnability, and even how the toolkit should be evaluated.

NOTABLE TOOLKITS
Our interests lie in toolkits that let the end-developer create ubicomp-style physical interfaces that: gather data from the real world (e.g. sensors); respond in software and physical objects (e.g. visualizations, motors); and support creating behaviours linking the two. We consider three toolkits within this genre that will act as running examples to discuss various toolkit/audience challenges: Phidgets [3], d.Tools [5] and the Proximity Toolkit [11].

Phidgets
Fitchett and Greenberg [3] introduced Phidgets in 2001. Phidgets comprise both hardware and software. Hardware includes USB-based circuit boards that provide different sensors and actuators. The software includes an API for interacting with each type of board. The API controls the board's components (e.g. rotating a servo motor to a particular angle) and delivers changes to sensor values as events. The software includes graphical widgets representing each board for developers to view and test the hardware counterpart.

Phidgets originated from frustrations its authors had in creating early tangible user interfaces. To build such interfaces, developers had to be knowledgeable in many areas, including circuitry, micro-programming, networking, etc. Acquiring that knowledge came at a high cost and time demand. Thus, Greenberg and Fitchett designed Phidgets with computer programmers as its audience in mind – people who do not necessarily understand electronics but are proficient in writing event-driven object-oriented software [3]. They designed Phidgets to mimic traditional UI widget programming, as it would then be easy for developers to integrate into their existing workflow.

Phidgets became a commercial product, one which is now widely recognized and used within the HCI community. Other researchers have since incorporated Phidgets into their own platforms [5,12].

d.Tools
d.Tools [5] is a high-level authoring tool, which (in part) incorporates Phidgets. A designer prototypes interactive behaviours by manipulating state-diagrams that move through different outputs based on sensor interpretations. d.Tools' audience is interaction designers – people without specialized engineering or programming knowledge who want to quickly iterate through the early designs of functional interactive objects [5]. d.Tools is widely cited in HCI. It was later

extended into Exemplar [4], which incorporated pattern recognition and programming by demonstration.

The Proximity Toolkit

The Proximity Toolkit [11] audience is highly specialized researchers investigating the design of *proxemic interactions*. Proxemic interaction imagines a world of devices and interaction behaviors that have fine-grained knowledge of nearby people and devices: how devices and people move into range, their precise distance from one another, their identity, and even their relative orientation. The toolkit encapsulates and abstracts sensor data (e.g. Vicon, Kinect), as relations between entities. Developers can focus on designing proxemic-aware applications rather than the setup and complex programming of tracking equipment and its data.

The Proximity Toolkit includes an event-driven API that informs the system of changes in proxemic values for different entities. Developers can monitor objects that move in the environment at runtime, either by showing numeric changes in variables of interest, or by interacting with a visualization showing all tracked entities and the proxemic relations between them. It also eases development by recording and storing tracked data, which can then be replayed as a simulation. The authors and their colleagues developed a large number of proxemic interaction techniques and applications [1,10] to investigate how proxemic interactions can be applied to other domains, such as advertising [17], and remote controls [8].

THE TOOLKIT/AUDIENCE CHALLENGES

Prototyping toolkits often refer to their end-developer in a range of ways: programmer, designer, developer, end-user, maker, researcher, etc. Regardless of how the expected end-developer is labelled, toolkits need to define and understand their target audience. Indeed, Olsen [14] argues for the importance of understanding situation, tasks, and user when creating a toolkit. Below are a few sample challenges that can help unpack attributes about the primary end-developer and how it relates to the toolkit.

Challenge 1. End-Developer Skills

Myers et. al. [13] argue that one aspect of evaluating a toolkit is its threshold and ceiling. *Threshold* refers to the developer effort to get started, while the *ceiling* defines how much can be done using the tools. Ideally, a toolkit would have a low threshold and high ceiling. Yet the notions of threshold and ceiling are actually relative to the skills of the end-developer.

Toolkits often extend existing programming languages, which affect the threshold for the end-developer. With Phidgets, originally built atop Visual Basic, the end-developer would have a very low ceiling only if they were proficient in Visual Basic and its interface builder. In contrast, an interaction designer with no programming background would find the threshold high, as they would have to learn to program before using Phidgets. The commercialized version of Phidgets mitigated this issue somewhat by making its API accessible to a broader audience skilled in different programming platforms: core languages (e.g., C#, Java), mobile (iOS, Android), scripting (Python), multimedia platforms (Flash), etc. d.Tools further reduce the threshold for non-programmers by providing an authoring environment that substituted programming with state-diagrams.

High ceilings also depend on the audience. Toolkits offer high ceilings through flexibility and expressiveness, but this only works when the end-developer has design skills in the area that the toolkit is trying to open. For example, the Proximity Toolkit offers a high ceiling for proxemic interaction development via: a myriad of proxemic variables; relationships between people, devices and objects; and flexible configuration of the physical sensing environment. It assumes its end-developers have knowledge in proxemic theory and how to apply it to interaction design. If the developer does not have that knowledge, the richness of the Proximity Toolkit can easily become a liability. This is especially true if the end-developer wants to take the *path of least resistance*, where the toolkit guides them to 'do the right thing, away from the wrong things' [13].

Challenge 2. End-Developer Resources

Toolkits may rely on commercial or DIY hardware. Sometimes the underlying technologies can be acquired with ease and at reasonable cost (e.g. Phidgets). However, other toolkits assume a larger infrastructure (e.g. the Proximity Toolkit requires a dedicated room and specialized hardware such as the Vicon motion tracking system). As the required resources and costs increase, the expected audience will narrow to only those very interested in the area.

Challenge 3: End-Developer Learning

Another consideration is how the end-developer will learn the toolkit.

First, end-developers need to learn what the toolkit offers over and above its base platform. For example, Phidgets and the Proximity Toolkit both offer an API to particular capabilities, and they must be learned, along with the patterns that best exploits that API. While D.Tools offers the state diagram approach, that too must be learned.

Second, end-developers also need to make sense of the overall data, automated processes, etc. as provided by the toolkit. For example, various ubicomp-oriented toolkits exploit sensor data, often delivered as low-level, frequently updated variables. Yet, learning what that sensor data means (especially if it is noisy) can be quite challenging, for that data must be related to real-world phenomena. This is partially why toolkits provide high-level tools that visualize and/or aggregate sensor data [6,11] or store information for further scrutiny [9,11]. To illustrate, the Proximity Toolkit shows a visualization of all objects in a scene, and how the proxemic variables (i.e., aggregated sensor values) track the proxemic relations between those objects. The end-developer can view the visualization to learn and understand the changes as they occur, which become references for creating the new system.

Third, research-oriented systems often assume knowledge of an underlying design paradigm. Phidgets and d.tools assume

some knowledge and experience in physical and tangible user interfaces. The Proximity Toolkit assumes some knowledge in Proxemics theory and proxemic interaction. However, the toolkits themselves do not offer easy ways to acquire that knowledge, except perhaps by referring to external resources such as publications.

Toolkits must be constructed with learnability in mind, which depends on the intended audience. They need to give the end-user an idea of what is possible, help make sense of (and debug) the data, and include resources to help new users understand the toolkit. Thus, the toolkit should offer a broad range of simple example systems, extensive documentations, repositories of examples, video tutorials, etc., as well as published papers of the design space supported by the toolkit.

SUGGESTED WORKSHOP TOPIC

Based on the above challenges, we propose the following topic for the workshop: who is the audience, and how does the toolkit design fit that audience? The prior challenges document only a few examples of concerns related to the end-developers. We expect workshop members will suggest other concerns, and elaborate on the ones mentioned here.

For example, toolkit evaluation is a great concern for many toolkit researchers, especially because submitted toolkit publications are usually accepted only if they are accompanied by a convincing evaluation of the toolkit. However, evaluation without the context of the intended audience is a somewhat pointless (and perhaps misleading) exercise. To illustrate, various toolkits are evaluated by illustrating how end-developers can quickly create prototypes within a short period of time. Yet, such an evaluation is meaningful only if the intended audience has the core skills behind the toolkit, is able to learn the toolkit quickly, and has the resources already on hand. In many toolkit papers, the audience doing the evaluation was prepared *a priori*. For example, Phidgets were evaluated by showing how undergraduate students created many Phidget prototypes quickly. Those students were already knowledgeable in Visual Basic (skills), were given a collection of Phidget hardware and cables ahead of time (resources), and were provided with lectures illustrating the tangible interface genre along with a step by step tutorial of how to use Phidgets (learnability). The Proximity Toolkit was evaluated by illustrating graduate student projects: those students were also prepared in a manner similar to the Phidgets study, and the specialized equipment required was already in place. d.Tools performed two evaluations with audiences knowledgeable in design, and were also prepared and supported. The first audience comprised participants with general design experience who were assigned particular tasks to do. The second were students in a masters-level HCI design course. In all the above cases, the context of the chosen audience approached a 'best case' for evaluation.

AUTHOR BACKGROUND AND POSITION

David Ledo is a PhD student at the University of Calgary working under supervision of Lora Oehlberg and Saul Greenberg. During his undergraduate, he took part in building toolkits (e.g. [9]), while during his masters he worked with the Proximity Toolkit, creating remote control applications [8]. Given his training and practice, he often creates libraries and wrappers for different tasks (e.g. visualization, networking). As part of teaching an undergraduate class in advanced HCI, David created a toolkit for his students to connect mobile devices and Phidgets to author new smart interactive objects. As part of his PhD topic, he works on creating prototyping tools for interaction designers to author smart interactive objects using mobile devices instead of electronic processors or components [7].

Lora Oehlberg is an Assistant Professor at the University of Calgary. Her research focuses on interactive tools and technologies that support creativity, innovation, and multi-disciplinary collaboration in domains such as interaction design, maker communities, and health care. Due to Lora's research background in product design theory and methodology, she cares about how prototyping tools and toolkits fit into real-world interaction design practice. She is interested in the intersection of interaction design and product design practice – when designers want to use prototyping tools to define the behavior and form of interactive physical objects.

Saul Greenberg is a Faculty Professor and Emeritus Professor in the Department of Computer Science at the University of Calgary. While he is a computer scientist by training, the work by Saul and his students typify the cross-discipline aspects of Human Computer Interaction, Computer Supported Cooperative Work, and Ubiquitous Computing. He and his crew are well known for their development of: toolkits for rapid prototyping of groupware and ubiquitous appliances; innovative and system designs based on observations of social phenomenon; articulation of design-oriented social science theories; and refinement of evaluation methods.

REFERENCES

1. Till Ballendat, Nicolai Marquardt, and Saul Greenberg. Proxemic interaction: designing for a proximity and orientation-aware environment. *Proc. ACM ITS 2010.*
2. Saul Greenberg. Toolkits and interface creativity. *Multimedia Tools and Applications,* 2007.
3. Saul Greenberg and Chester Fitchett. Phidgets: easy development of physical interfaces through physical widgets. *Proc. ACM UIST 2001.*
4. Björn Hartmann, Leith Abdulla, Manas Mittal, and Scott R. Klemmer. Authoring Sensor-based Interactions by Demonstration with Direct Manipulation and Pattern Recognition. Proc. ACM CHI 2007.
5. Björn Hartmann, Scott R. Klemmer, Michael Bernstein, Leith Abdulla, Brandon Burr, Avi Robinson-Mosher, and Jennifer Gee. Reflective Physical Prototyping Through Integrated Design, Test, and Analysis. *Proc. ACM UIST 2006.*
6. Scott R. Klemmer, Jack Li, James Lin, and James A. Landay. 2004. Papier-Mache: Toolkit Support for Tangible Input. *Proc. ACM CHI 2004.*

7. David Ledo, Fraser Anderson, Ryan Schmidt, Lora Oehlberg, Saul Greenberg, and Tovi Grossman. Pineal: Bringing Passive Objects to Life with Embedded Mobile Devices. *Proc. ACM CHI 2017*.
8. David Ledo, Saul Greenberg, Nicolai Marquardt, and Sebastian Boring. Proxemic-Aware Controls: Designing Remote Controls for Ubiquitous Computing Ecologies. *Proc. ACM MobileHCI 2015*.
9. David Ledo, Miguel Nacenta, Nicolai Marquardt, Sebastian Boring, and Saul Greenberg. The HapticTouch Toolkit: Enabling Exploration of Haptic Interactions. *Proc. ACM TEI 2012.*
10. Nicolai Marquardt, Till Ballendat, Sebastian Boring, Saul Greenberg, and Ken Hinckley. Gradual Engagement between Digital Devices as a Function of Proximity: From Awareness to Progressive Reveal to Information Transfer. *Proc. ACM ITS 2012*.
11. Nicolai Marquardt, Robert Diaz-Marino, Sebastian Boring, and Saul Greenberg. 2011. The proximity toolkit: prototyping proxemic interactions in ubiquitous computing ecologies. *Proc. ACM UIST 2011.*
12. Nicolai Marquardt and Saul Greenberg. Distributed Physical Interfaces with Shared Phidgets. *Proc. ACM TEI 2007.*
13. Brad Myers, Scott E. Hudson, and Randy Pausch. Past, Present, and Future of User Interface Software Tools. *ACM ToCHI* 7, 1: 3–28, 2000.
14. Dan Olsen. Evaluating user interface systems research. *Proc. ACM UIST 2007.*
15. Raf Ramakers, Kashyap Todi, and Kris Luyten. PaperPulse: An Integrated Approach for Embedding Electronics in Paper Designs. *Proc. ACM CHI 2015.*
16. Valkyrie Savage, Colin Chang, and Björn Hartmann. Sauron: Embedded Single-camera Sensing of Printed Physical User Interfaces. *Proc. ACM UIST 2013.*
17. Miaosen Wang, Sebastian Boring, and Saul Greenberg. Proxemic Peddler: A Public Advertising Display That Captures and Preserves the Attention of a Passerby. *Proc. ACM PerDis 2012.*

Malleable User Interface Toolkits for Malleable Cross-Surface Interaction

James R. Eagan
LTCI, Telecom ParisTech, Université Paris-Saclay
75013 Paris, France
james.eagan@telecom-paristech.fr

ABSTRACT
Existing user interface toolkits are based on a single user interacting with a single machine with a relatively fixed set of input devices. Today's interactive systems, however, can involve multiple users interactive with a heterogeneous set of input, computational, and output capabilities across a dynamic set of different devices. The abstractions that help programmers create interactive software for one kind of system do not necessarily scale to these new kinds of environments. New toolkits designed around these new kinds of environments, however, need to be able to bridge existing software and libraries or recreate them from scratch. In this position paper, we examine these new constraints and needs and look at three strategies for software toolkits that help to bridge existing toolkit models to these new interaction paradigms.

Author Keywords
user interface toolkits, cross-surface interaction, instrumental interaction, malleable software

INTRODUCTION
The design of current user interface toolkits has changed relatively little since the first graphical user interfaces first began to appear: users interact with interface controls, or widgets, using a pointing device and a keyboard. While these toolkits have seen minor improvements over time, such as handling pointing with a finger or performing gestures, the overall design approach has changed little.

A programmer creates an interface using a collection of widgets. They typically come from a standard set of pre-defined widgets, but programmers may propose their own set of supplemental widgets. They may modify the behavior of an existing widget in some small way, such as a custom list view that might show font names rendered in the relevant font. Or they may provide wholly new behaviors, such as a double-ended range slider or an interactive graph view.

Ultimately, however, the applications programmers create are heavily influenced by the set of widgets provided by the toolkit. These widgets, however, were designed around at a time when one user interacted with one machine, and applications consisted of a single process running on that machine.

These assumptions start to break down in the context of multi-surface or cross-device interactions, where a logical application—from the user's point of view—might involve processes running across multiple devices, such as a phone, tablet, tabletop, wall, or motion tracking system.

Moreover, adapting an existing application's interaction or functionality to handle unanticipated usage scenarios may be cumbersome. A user who wishes to add a new toolbar button for a frequently-used task, or a teacher who wishes to extend an email client's data detectors [5,6] to link course numbers to an intranet course management website, or a technical writer who wishes to reference BibTeX citations in email client or presentation tool would be hard-pressed to do so without modifying existing applications' source code.

Currently, developers of systems that permit such kinds of interaction in new environments or such malleable interfaces must create explicit support on an ad-hoc basis. We need to provide programmers the appropriate tools and abstractions on such concepts that make creating future applications feasible in the same way the current UI toolkits greatly reduced the barriers to making graphical user interfaces.

THREE CHALLENGES FOR TOOLKIT RESEARCH
We view three primary challenges for toolkit research in such future applications: handling the heterogeneity of future application contexts with multiple users, multiple machines, and multiple interaction modalities; making software more malleable to support users' own particular needs in their own situated contexts; and bridging between the current state-of-the-art and future development models.

Multiple users, machines, interaction modalities
Application developers can no longer make the assumption that there will be one user, interacting with a single machine using a single mouse and a single keyboard. Applications in multi-surface environments, for example, may run on one or multiple machines, and the number of machines present may evolve during a single interaction session.

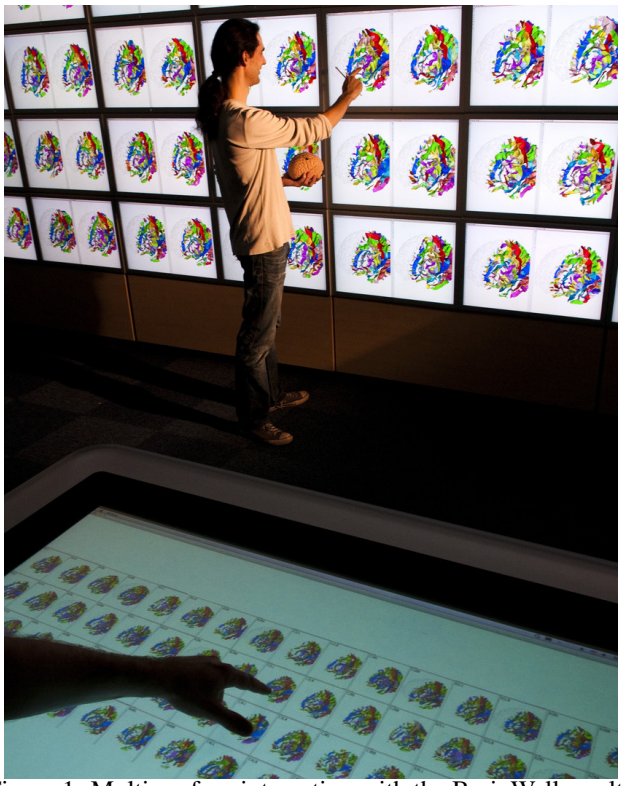

Figure 1: Multi-surface interaction with the BrainWall: multiple users interact with a table, wall-sized display, and physical interface props.

Figure 1 shows a real user scenario from our work with Substance Grise to create the BrainWall application [3]: A neuroscientist has the latest 3D brain scans of healthy and unhealthy brains on her smartphone. She enters the room and places her phone on an interactive tabletop to bring her data into the environment. Around the table, she and her colleagues arrange the brains to better facilitate comparison. To get a better view of the brains, she and her colleagues move in front of a wall-sized display that mirrors the display on the table, showing a high-resolution image of the brain scans. A colleague uses a wooden chopstick to point at a related structure on a plastic model of a brain, causing all of the brains displayed on the wall and table to re-orient their 3D views to that part of the brain. Another colleague takes out a tablet and selects one of the brains on the wall. On his tablet, he begins annotating the different structures and changing their colors, using the tablet as a personal workspace before sharing them with the group.

As this scenario shows, the BrainWall application actually consists of multiple applications running on multiple devices: a data provider on the neuroscientist's phone, an organizer on the tabletop, a viewer on the wall, a 3D tracker for the physical brain and pointer, and an annotator on the colleague's tablet.

Even without each of these individual components, the application would continue to function but without that component's capabilities. A scientist could continue to sort brains without the wall, or view the brains without the 3D tracker, etc. These devices may dynamically come and go, as when the colleague takes out a tablet to join the environment and annotate brains, or if the neuroscientist leaves the room to take a phone call.

Moreover, each of these different devices offers a different computational and interactional profile, with different memory, storage, processing, communication, input, or output characteristics. An application developer must manage the complexities of this heterogeneity and dynamicity manually.

To help with these challenges and to provide developers with a set of abstractions that simplify data sharing strategies and different functionalities between devices, discovery, and interaction, we created the Shared Substance prototype [3], described below. While this approach reduces the barrier to creating multi-surface applications, programmers must master a new "data-oriented programming" model and still must explicitly manage the specifics of data sharing strategies (such as via local replication or remote querying). Some of these details may require specific consideration from the programmer. For the rest, we need to develop a collection of appropriate abstractions much as undo managers have freed programmers from needing to explicitly support such capabilities.

Making software malleable

Current applications are designed for a particular context of use, with developers making a certain set of assumptions about how the software will be used. It is not possible, however, for designers to foresee and anticipate the myriad ways that a user may make use of the software.

Current software toolkits provide relatively little support for end-users to extend the capabilities of their software. Some systems do provide support for users to create macros to automate certain actions, as in Microsoft's Office suite or with AppleScript interfaces, but these are limited to the customization hooks that developers explicitly embed in their software and maintain as a separate set of functionalities. As such, developers must work explicitly to support these features and thus expose a larger surface area for potential bugs.

Some applications do provide support for plugins, but these interfaces are up to the individual application developer. Each application developer must create her own infrastructure for detecting loading, unloading, and sandboxing such plugins on an ad-hoc basis. As a result, each application, if it provides a plugin interface at all, offers a differing degree of access to program concepts and objects, and each modification creator must learn the specific intricacies of that particular program.

What is missing is explicit support in the toolkit to create generalizable application objects that programmers can re-use. In the 1980's and 90's, it was common for applications to provide their own "macro" capabilities, where users could automate their software using "macro" scripts. Each application provided its own set of capabilities, using its own specific macro scripting language. Today, Mac applications built with the standard Cocoa toolkit are automatically scriptable using AppleScript and support a standard set of universal objects and commands common to GUI applications: opening windows, selecting the frontmost document, clicking buttons, etc.

If application developers explicitly support it, they can add higher-level concepts as messages in an email program or todo items in a task manager. Nonetheless, application developers must explicitly provide such support, and the user is limited to the specific hooks provided by the developer.

Supporting such kinds of customization should be an automatic consequence of using standard toolkit elements and design patterns for interactive software. If a user wishes to, for example, overlay subtitles downloaded from the internet on a movie file downloaded from the iTunes store, it should be feasible for the user to be able to connect a subtitles loader to the video playback controller, even if the application developer did not anticipate such a feature.

Recreating the universe

We have created various toolkits that attempt to explore these concepts [2–4]. One of the challenges in creating new ways of building interactive software is the bootstrapping problem. If the toolkit is completely built from scratch, all applications in the environment need to be created from scratch. This approach offers great flexibility, but requires significant development effort and tends to lead to "toy" examples.

The Shared Substance [3] environment is an example of this kind of toolkit. Shared Substance is based on a data-oriented programming model similar to object-oriented programming in the sense that data associated with program concepts can be grouped together into objects and can have associated methods. In data-oriented programming, however, these methods are separable from the underlying data. Thus, an object running on a tabletop might offer a different set of functionalities than an object running on a smartphone, despite using the same underlying set of data. Data itself are organized into trees, providing a scene graph that can be shared across the different devices in a multi-surface environment.

Shared Substance programmers thus choose which subtrees to make available to other devices. The toolkit provides builtin discovery capabilities. When data are shared, or new devices become available, programs can either replicate the data by maintaining a cloned copy that must be kept in sync with its source, or they can mount the data, assuring that the original always maintains an authoritative copy of the data. Adding new functionality involves associating new Facets, or collections of methods, that can be attached to different parts of the data.

While this approach provides a set of transparent abstractions that frees the programmer from many of the challenges of multi-surface interactions, it does require programmers to think about and write software in a very different way. We found that the mental gymnastics of contorting one's brain into a new way of thinking hindered the development of software in this environment. Moreover, any new capabilities or applications, such as displaying a new kind of data on the wall, involved writing the code from scratch.

Bridging to legacy software: Scotty

Scotty [2] uses a different philosophy. Its goal is two-fold: to provide a test-bed for exploring instrumental interaction [1] and to provide a toolkit for the development of malleable applications. Rather than create a toolkit from scratch built around these concepts, we built Scotty as a meta-toolkit that grafted new capabilities into Cocoa. Thus, existing Cocoa applications can benefit from Scotty's new capabilities without modification of their source code.

Scotty thus is able to give arbitrary Cocoa applications the ability to load Scotty plugins that can be built using concepts of instrumental interaction. Scotty instrument plugins draw upon the Scotty toolkit to provide lenses into the underlying application's objects, views, and controllers. As such, creating the subtitles modification described above is "simply" a matter of identifying the playback window's playback controller to extract the current time. Scotty itself provides tools for helping a plugin developer to inspect and make sense of a host application's interface and core program objects. Adding this new functionality is thus a matter of attaching a transparent overlay window, loading a Python module that decodes subtitles, etc. In about 150 lines of Python code, a programmer can "teach" Quicktime Player to load subtitles from an external file, overlay them on the screen, and integrate them to the playback of the movie.

This approach has the advantage of quickly being able to take advantage of the full ecosystem of existing Mac applications. In theory, researchers are not bound by what they can develop. If an existing program offers the core functionality necessary, it should be straightforward to incorporate it into a research prototype.

The reality, of course, is different: design choices made by the original developers are invisible. Shoe-horning them into a new environment, with different core assumptions and core values, can involve considerable effort. In the case of instrumental interaction, for example, Mac applications just aren't designed with this style of interaction in mind. A developer may end up spending as much effort adapting an existing application to a new interaction paradigm as she would have implementing a proof of concept. Only in hindsight can she evaluate whether that effort is worth the benefits of having a real application running in a real environment versus a functional proof-of-concept research prototype.

Webstrates

In between these two approaches, we built Webstrates as a sort of putty to build shareable dynamic media on top of existing web technologies. Developers can take advantage of existing knowledge of HTML, JavaScript, and CSS to build webstrates in this new environment. The learning curve to at least be functional in this environment is relatively low. Moreover, existing web applications and libraries are readily available so long as they meet or can be made to meet certain core assumptions of Webstrates (notably that the DOM in a web browser is no longer ephemeral).

Webstrates combines several appealing properties for the exploration of new kinds of applications: it is compatible with a large selection of applications that meet the requirements above, developers can leverage their existing knowledge and experience, web environments present a relatively low barrier

to entry, and the webstrate canvas itself is a relatively open environment in which developers can experiment freely.

In contrast to a strict bridge such as Scotty, where application developers must fit within the constraints of the Cocoa development environment first, and then figure out how to express their new interactions within its concepts, Webstrates allow developers to focus first on their concepts, and then on the constraints of the web framework. Both approaches use a bridging approach to bootstrap the development environment, but Webstrates finds a more lightweight balance than does Scotty.

CONCLUSIONS

Modern toolkits need to break the core assumptions implicit in historical interface toolkits. No longer do we live in a world of a single user at a single keyboard and mouse, interacting on his or her own. Multiple users may interact with multiple devices each in dynamic environments with different interactive and computational properties. The interactive metaphors that worked in historic environments do not necessarily hold up in the face of these new additions. We thus need to build new toolkits to help programmers build interactive software that can handle these changing constraints.

We have presented a collection of three toolkits that we have built, deployed, and published. Through this experience, we have explored different styles of implementing new interaction models and new ways of modeling interactive software; and new ways of building off of the existing set of tools that have been created over the course of the WIMP and early post-WIMP era.

ACKNOWLEDGEMENTS

[Redacted for review.]

REFERENCES

1. Beaudouin-Lafon, M.: Instrumental interaction: an interaction model for designing post-wimp user interfaces. In: CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems. pp. 446–453. ACM, New York, NY, USA (2000)
2. Eagan, J.R., Beaudouin-Lafon, M., Mackay, W.E.: Cracking the cocoa nut: user interface programming at runtime. In: Proceedings of the 24th annual ACM symposium on User interface software and technology. pp. 225–234. UIST '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/2047196.2047226
3. Gjerlufsen, T., Klokmose, C.N., Eagan, J., Pillias, C., Beaudouin-Lafon, M.: Shared substance: developing flexible multi-surface applications. In: Proceedings of the 2011 annual conference on Human factors in computing systems. pp. 3383–3392. CHI '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/1978942.1979446
4. Klokmose, C., Eagan, J., Baader, S., Mackay, W., Beaudouin-Lafon, M.: Webstrates: Shareable Dynamic Media. In: ACM (ed.) ACM Symposium on User Interface Software and Technology (UIST). pp. 280–290. UIST '15 Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology, Charlotte, United States (Nov 2015), https://hal.archives-ouvertes.fr/hal-01242672
5. Nardi, B.A., Miller, J.R., Wright, D.J.: Collaborative, programmable intelligent agents. Communications of the ACM 41(3), 96–104 (1998)
6. Pandit, M.S., Kalbag, S.: The selection recognition agent: Instant access to relevant information and operations. In: Proceedings of the 2Nd International Conference on Intelligent User Interfaces. pp. 47–52. IUI '97, ACM, New York, NY, USA (1997), http://doi.acm.org/10.1145/238218.238285

Looking for the Right Toolkit at the Right Time

Fabio Paternò
CNR-ISTI, HIIS Laboratory
Via Moruzzi 1, 56124 Pisa, Italy
fabio.paterno@isti.cnr.it

ABSTRACT
I have been working for several years in the area of tools for supporting design, development, and evaluation of interactive systems. In this position paper following the suggested format, I report on my personal view on the workshop topics based on such experience.

Author Keywords
End user development, cross-device user interfaces, model-based design.

ACM Classification Keywords
H.5.2 [Information Interfaces and Presentation]: User Interfaces - Interaction styles.

THREE CHALLENGES/OPPORTUINITIES

End User Development

The design and development of flexible software able to match the many possible users' needs is still a difficult challenge. It is almost impossible to identify all the requirements at design time and, in addition, such requirements are not static since user needs are likely to change/evolve over time, and designers have to consider the wide variability of the possible contexts of use. Indeed, the explosion of mobile technologies has made it possible for people to access their applications from a variety of contexts of use that differ in terms of available devices, things, and services, and that require specific actions when various types of events occur. Thus, it is not possible to guarantee a complete fit between the initially designed system and actual needs at any given time. As a result, it is important to design interactive software through methods and tools capable of dynamically and quickly responding to new requirements without spending vast amounts of resources, and which are able to consider that boundaries between design-time and run-time have become more and more blurred. Achieving this can increase the impact of software development companies since their applications can more easily penetrate many markets thanks to their ability to be customized directly by domain experts.

End-User Development (EUD) [1] approaches can help to solve such issues by enabling the possibility of customizing software applications by domain experts that have not experience in programming. Users' backgrounds can vary from management, engineering, construction, education, research, health, insurance, sales, administration or other areas. On the one hand, such users share a common requirement for software to support their common tasks, which may vary rapidly, and some of them cannot even be anticipated at design time, but discovered only during actual use. On the other hand, current slow software development cycles and the lack of domain knowledge on the part of software developers are limitations to addressing the requirements of the different users. Thus, EUD can reduce time and costs needed for customizations and increase their quality by avoiding potential misunderstandings between final users and developers.

Cross-device User Interfaces

The increasing availability of various types of devices in our daily life is often a missed opportunity since current applications are limited in supporting seamless task performance across them. Users often perceive device fragmentation around them rather than an ecosystem of devices that supports their activities. In order to address such issues a number of frameworks, platforms, and authoring environments have been proposed, mainly in research environment. The goal is to facilitate design and development of multi-device user interfaces. Responsive design is not enough to address such issues since its basic assumption is that the user at a given time interacts with only one device, such device may vary over time, and thus it provides the possibility of easily modifying the presentation mainly according to the screen size. In cross-device design such assumption no longer holds, and the goal is to allow developers to create applications that allow users to interact with many devices at the same time, and the various parts of the user interfaces distributed across the different devices can keep their state synchronized.We can distinguish various types of multi-device user interfaces depending on the features that they support: migratory user interfaces are able to dynamically migrate from one device to another in order to follow users' movements while preserving their state; distributed user interfaces allow users to interact with an application through multiple devices at the same time; cross-device user interfaces are distributed user interfaces, with the additional capability to synchronise their state, so that the interactions through some element in one device update the state of the corresponding elements (if any) in another device. Such categories are not mutually exclusive, so for example it is possible to have user interfaces that are both migratory and cross-device.

Model-based User Interfaces

Model-based approaches [2] have been proposed to manage the increasing complexity derived from user interfaces in multi-device environments. They have also been considered in W3C for standardization in order to ease their industrial adoption. The main idea is to provide a small general conceptual vocabulary to support user interface developers in the design process. The resulting logical descriptions can then be transferred into a variety of implementation languages with the support of automatic transformations, thereby hiding the complexity deriving from the heterogeneous languages and devices, and sparing developers the need to learn all the details of such implementation languages. Thus, they can be useful to obtain more accessible applications as well since they make more explicit the semantics and the role of the various user interface elements, which is also important for access through assistive technologies.

One important contribution in the area of intelligent user interfaces was Supple [3], which provided a tool able to consider aspects related to the user and the device at hand for generating a personalized version of the interactive application. However, the combined explosion of mobile technologies and Internet of Things has posed new challenges to address since there is a variety of contextual aspects to consider when generating interactive applications. More recently, a generative approach with semantic interaction descriptions for obtaining user interfaces for smart things was proposed [4] but again it assumed some previous knowledge of the smart things to address, and thus lacked the ability to support customization for dynamic contexts of use.

Other authors have more focused on how the use of model-based UI development approaches can improve the experience of users interacting with context-dependent, multi-platform applications. Recent contributions [5] presented a UI development framework for ambient applications integrated with a user modelling system, in order to provide usability predictions during early development stages.

One issue in model-based techniques is that because heuristics are often involved in the generation process, the connection between specification and final result can be problematic to understand and control. Programmers must also learn a new language for specifying the models, which raises the threshold of their use. However, the importance of such approach is demonstrated by its adoption, to some extent, by a widely used languages as HTML 5. Indeed, one of the main HTML 5 features is the introduction of more semantics tags, which better express the purpose of the possible user interface elements. On the other hand, HTML 5 is still limited with respect to the potentialities of model-based approaches since it mainly considers only graphical user interfaces, while languages such as MARIA [6] can be used also for generating multimodal user interfaces [7] as well.

THREE SUCCESSFUL TOOLKITS

The Context Toolkit

The Context Toolkit [8] aimed at facilitating the development and deployment of context-aware applications. The Context Toolkit was among the earliest supports for developing context-enabled applications by providing a library to facilitate integration with sensors. It was one of the first attempts to support developers in creating applications able to consider not only the events that occur on the screen but also those that are generated in the surrounding environment.

The authors defined context as environmental information that is part of an application's operating environment and that can be sensed by the application. It consisted of context widgets and a distributed infrastructure that hosted the widgets. Context widgets are software components that provide applications with access to context information while hiding the details of context sensing. In the same way GUI widgets insulate applications from some presentation concerns, context widgets insulate applications from context acquisition concerns. To summarize, the main features of the Context Toolkit were: encapsulation of sensors; access to context data through a network API; abstraction of context data through interpreters; sharing of context data through a distributed infrastructure; storage of context data, including history; basic access control for privacy protection.

It initially considered a limited set of events and led to meld the context awareness code with the application. More recently, the Context Toolkit has been augmented with support to facilitate development and debugging of context-dependent applications [9]. Programming abstractions, called Situations, expose an API supporting both developers and designers to provide application adaptivity without coping with low-level code.

Scratch

Scratch [10] is a visual programming language developed by the MIT Media Lab, which targets students, scholars, and teachers parents to easily create animations, games, and similar applications. Thus, it represents a useful tool for a range of educational and entertainment purposes., and it can be a way for introducing to the more advanced world of computer programming. Scratch represents an example of tool for end user development environment based on the jigsaw puzzle metaphor, in particular in creating interactive applications with multimedia content. In this metaphor, each software components is seen as a piece of a puzzle and the shapes of the various pieces provide the cognitive hints needed to understand the possible compositions. As such, non-expert users can easily associate each puzzle piece with the component it represents. While this metaphor supports

more complex configurations than the ones supported by the pipeline metaphor, one disadvantage is that it has constrained capability for limited expressiveness. Indeed, the pieces of the puzzle have a limited number of interfaces (i.e. sides), thereby restricting the set of possible programming expressions. AppInventor [11] has then exploited such metaphor to support the development of functionalities triggered by events on an app user interface. While in Scratch and AppInventor the puzzles pieces are used to represent low-level programming constructs, a different approach has been proposed in Puzzle [12], in which it has been adopted to support development of Internet of Things applications on smartphones:, and the elements are associated with high-level functionalities that can also control various actuators. Thus, Puzzle has been designed to facilitate the composition of various pieces through a touch interface for a screen with limited size. Each possible puzzle piece represents a high-level functionality that can be composed, and its shape and colours indicate the number of inputs and outputs and the information type of information that they can communicate. Thus, the tool provides a usable solution but is limited to the composition of functionalities for which a puzzle piece has been provided.

Supple

One important contribution in the area of intelligent user interfaces was Supple [3], which provided a tool able to consider aspects related to the user and the device at hand for generating a personalized version of the interactive application.

Supple uses decision-theoretic optimization to automatically generate user interfaces adapted to a person's abilities, devices, preferences, and tasks. In particular, SUPPLE can generate user interfaces for people with motor and vision impairments and the results of our laboratory experiments show that these automatically generated, ability-based user interfaces significantly improve speed, accuracy and satisfaction of users with motor impairments compared to manufacturers' default interfaces. It takes a functional specification of the interface, the device-specific constraints, a typical usage trace, and a cost function. The cost function is based on user preferences and expected speed of operation. SUPPLE's optimization algorithm finds the user interface, which minimizes the cost function while also satisfying all device constraints.

The SUPPLE authors then focused on how to exploit SUPPLE in order to support disabled users, for example, by automatically generating user interfaces for a user with impaired dexterity based on a model of her actual motor abilities. More generally, we can consider adaptation useful for both permanent and temporary disabilities. An example of temporary disability is when the user has to move fast and interact with a graphical mobile device. Thus, the user's visual attention cannot be completely allocated to the interaction.

BRIEF OVERVIEW OF MY PAST WORK

The goal of my research work has been to bring human values such as usability and accessibility in the design and development of interactive technologies.

At the beginning, my main research area was model-based design of interactive applications. I developed the ConcurTaskTrees notation for specifying task models and also designed an associated environment (CTTE) to support the development and analysis of task models specified through this notation, which has been widely used in various industries and universities in various parts of the world (with 26000 downloads). The tool has been applied in several application domains including ERP, interactive safety-critical systems (for example in air traffic control[1]), and the language has been the input for a W3C standard in the area (http://www.w3.org/TR/task-models/). I also worked on the design of the MultiModal TERESA and MARIA languages and the associated tools, whose main purpose is to support designers of multi-device, multi-modal interactive applications starting with user interface logical descriptions.

A considerable amount of work has also been dedicated to mobile guides. I designed the first museum mobile guide in Italy for the Marble Museum. Then, various solutions for this type of guide have been further designed, some in collaboration with local museums, which exploit various technologies for location awareness, collaboration and multimodal interaction.

From this type of experience I have broaden my interests to ubiquitous interactive systems. In particular, to address issues related to multi-device environments by proposing original solutions for migratory and cross-device user interfaces, which allow seamless access through a variety of devices ranging from wearable to large public displays, and dynamic allocation of interactive components across them. I also edited and wrote part of a book on Migratory Interactive Applications for Ubiquitous Environments published by Springer Verlag.

In parallel, another research topic in which I have played a pioneering role is end-user development, in which area I coordinated a European Network of Excellence (EUD-net). I also co-edited (together with Henry Lieberman from MIT, and Volker Wulf from University of Siegen) one of the best-known books on End User Development (widely cited), and carried out various research studies in the area. In this area I have actively worked at the design of various authoring environments and tools, such as Puzzle for intuitively editing interactive applications from a smartphone, or a mashup tool for creating new Web applications by composing existing components using the

[1] https://www.eurocontrol.int/ehp/?q=node/1617

familiar copy-paste interaction across them. More recently, I have focused on how to personalize context-dependent applications through trigger-action rules [13].

SUGGESTIONS FOR TOPICS FOR DISCUSSION

I see room for discussion at the workshop from different perspectives. One perspective is to consider the main technological trends and how they impact on the design of user interface toolkits. For example, we are in the Internet of Things time, which means more and more surrounded by various types of sensors, objects, devices and services that can be associated dynamically to meet user's expectations. Are current toolkits able to sufficiently support the design and development of applications for such environments ?

Another perspective is to analyse trends and solutions in this area through the threshold/ceiling criteria (the "threshold" is how difficult it is to learn how to use the toolkit, and the "ceiling" is how much can be done using the toolkit) [14]. Various toolkits have been criticized because they have high threshold and low ceiling, which means they require considerable effort for learning how to use them, and then in the end they support limited functionalities. What approaches should be considered to invert this situation and obtain low threshold / high ceiling solutions ?

One further perspective is to focus on the main attributes that user interface toolkits should have, and thus they can be useful to design an evaluation framework for toolkits. Such attributes are usually important both for the toolkits and the applications that they allow developers to obtain. Example of attributes that seem particularly relevant are: coverage, consistency, interoperability, usability, customizability, extensibility, and scalability.

REFERENCES

1. H. Liebermann, F. Paternò, V. Wulf. "End-User Development", Springer, Dordrecht, 2006
2. J.Fonseca (Ed.) (2010) Model-Based UI XG Final Report, W3C Incubator Group Report 2010.
3. Krzysztof Gajos and Daniel S. Weld. SUPPLE: automatically generating user interfaces. In IUI '04: Proceedings of the 9th international conference on Intelligent user interface, pages 93-100, New York, NY, USA, 2004. ACM Press.
4. Simon Mayer, Andreas Tschofen, Anind K. Dey, and Friedemann Mattern: User interfaces for smart things--A generative approach with semantic interaction descriptions, ACM Transactions on Computer-Human Interaction (TOCHI) 21 (2), 12, 2014.
5. Marc Halbrügge, Michael Quade, Klaus-Peter Engelbrecht, Sebastian Möller, Sahin Albayrak, Predicting user error for ambient systems by integrating model-based UI development and cognitive modelling Proceedings Ubicomp'16, pp.1028-1039, ACM Press
6. F. Paternò, C. Santoro, L.D. Spano, "MARIA: A Universal Language for Service-Oriented Applications in Ubiquitous Environment", ACM Transactions on Computer-Human Interaction, Vol.16, N.4, November 2009, pp.19:1-19:30, ACM Press,
7. M Manca, F Paternò, C Santoro, LD Spano. Generation of multi-device adaptive multimodal web applications. International Conference on Mobile Web and Information Systems, 218-232
8. D. Salber, A. K. Dey, and G. D. Abowd: The Context Toolkit: Aiding the Development of Context-Enabled Applications. CHI 1999: 434-441
9. A. K. Dey, and A. Newberger: Support for context-aware intelligibility and control. CHI 2009: 859-868
10. M. Resnick, J. Maloney, A. Monroy-Hernandez et al., "Scratch: programming for all," Communications of the ACM, vol. 52, no.11, pp. 60–67, 2009.
11. App Inventor MIT, 2012, http://appinventor.mit.edu/.
12. Danado J., Paternò F., Puzzle: Puzzle: A Mobile Application Development Environment using a Jigsaw Metaphor, Journal of Visual Languages and Computing, 25(4), pp.297-315, 2014.
13. Giuseppe Ghiani, Marco Manca, Fabio Paternò, Carmen Santoro: Personalization of Context-dependent Applications through Trigger-Action Rules. ACM Transactions on Computer-Human Interaction (ACM TOCHI), 2017.
14. B. A. Myers, S. E. Hudson and R. Pausch: Past, Present and Future of User Interface Software Tools, ACM Transactions on Computer Human Interaction. March, 2000. 7(1). pp. 3-28.

Decomposing Interactive Systems

Philip Tchernavskij
LRI, Univ. Paris-Sud, CNRS,
Inria, Université Paris-Saclay
F-91400 Orsay, France
philip.tchernavskij@lri.fr

ABSTRACT
I argue that systems-oriented HCI should explore software engineering principles and architectures that emphasize user interaction over designer control. Many researchers have argued that user-empowering interaction should decouple tools from the objects they act on. Implementing this decoupling requires actively subverting the traditional architectures of interactive systems, including the encapsulation of interactive systems into closed applications, and the overly coupled event-driven programming model. I present a sketch of an architecture where *interaction instruments* are a first-class object to address these issues.

ACM Classification Keywords
H.5.2. Information Interfaces and Presentation (e.g. HCI): User-centered design

Author Keywords
Toolkits; Interaction paradigms; Software architecture;

INTRODUCTION

Interactive systems, which nowadays are primarily desktop, mobile, and web applications, are notoriously inflexible: they encapsulate a fixed user interface to manipulate a predefined type of data, with little user control over the configuration and capabilities of the software. Beaudouin-Lafon argues that *"the only way to significantly improve user interfaces is to shift the research focus from designing interfaces to designing interaction."* [2] He outlines several challenges for moving towards novel interactive systems in HCI research, among them developing novel *interaction architectures*, which support interaction at the tool and middleware level: *"Interactive systems are by definition open: they interact with the user (or users) and often with other programs. They must therefore adapt to various contexts of use, both on the users side and on the computer side. [. . .] I believe it is critical that we define interaction architectures that give more control to end users, that are more resistant to changes in the environment, and that scale well. I call these three properties reinterpretability, resilience and scalability."* [2]

Consider a user writing a document, who decides she wants to add a figure. She may have several applications with sophisticated illustration tools, but none of them allows her to just draw the figure directly on the "paper" of the document. If she is writing a math report, she can write formulae in her word processor, but she cannot ask it to evaluate them. By contrast, when interacting with the physical world, people spontaneously extend their capability to manipulate particular objects by adding tools, and use tools and objects in ways that they were not necessarily designed for. Can we achieve such flexibility in software systems? Can we decompose interactive systems into components such that users can compose them in ways that correspond to their idiosyncratic needs?

Allowing users to actively de-compose and re-compose systems is a way of letting them do more with less. Indeed, this would let users: replace basic tools that exist in many variations across a system with the one they prefer, e.g., they can choose their preferred way of picking and applying colors rather than the one imposed by each application they use; select and combine parts of different systems coming from different vendors to support their particular workflow; and adapt tools to contexts they were not designed for, e.g., use a statistical graphing tool to create drawings.

In software architecture, flexibility is the quality of being able to change a system by adding, rather than modifying parts [6, p. 35]. Gjerlufsen et al. distinguish between flexibility at *design-time* and *runtime* [10]. Whereas design-time flexibility is advantageous to engineers who will need to reuse and extend a system architecture, runtime flexibility can allow users to extend the capabilities of a system in use. Therefore, to create reinterpretable systems we should develop toolkits that shift flexibility towards users rather than towards designers and developers.

In this paper, I critique asepcts of common interaction architectures, and sketch a critical alternative. I argue that the application model of software and event-driven programming create static systems where user-facing flexibility is exceptional. I describe an architecture based on *interaction instruments* that users can freely appropriate and combine according to their needs.

CRITIQUING INTERACTIVE SOFTWARE

Today, most of our interactions with the digital world are mediated by *applications* (*apps* for short). Apps make for static, closed systems, where there is typically one user, one device, a prescribed set of tools, and one or more digital artifacts, such as a document. Apps are isolated from the environment in which they are used. Their internals are encapsulated by a strict interface for input and output. To work on a document stored in a file, an app has to load the file and create an internal representation that it can change. This encapsulation strictly limits how apps can be combined. Apps can be sequenced, i.e. a file output by one app can be loaded by another (if the formats are compatible), but they cannot concurrently work with the same file. On the output side, apps each have their own window, so their content cannot be mixed or exchanged, except through copy-paste — which duplicates, rather than shares, content. Some apps share content through a remote database, but then bear the burden of maintaining consistency between the database and their internal state. This is more akin to a distributed app than an open environment.

As apps couple what you can do (commands) with what you can do it to (content), they implement *procedures* rather than tools and materials. Procedures are idealized descriptions of how work is done. In real life, the boundaries between different types of work are porous, and people constantly add tools, materials, and collaborators to expand their capabilities. Apps are too rigid and monolithic for flexible interactive systems.

Gat argues that apps accumulate complexity *because* they are closed systems [9]. Since each app defines all the available tools in its particular domain, vendors end up competing on having the most features. According to Gat, the app model inevitably leads to large systems that function poorly. In practice, since it is impossible to meet the needs of every member of some particular community of practice, apps ends up being designed from a one-size-fits-all approach.

At the programming language level, the most common model for defining interactions is *event-driven programming*. Event-driven programming chains together statement of the form "When this input event happens on this graphical object, do that". In other words, event-driven interactions are programmed by creating design-time bindings between concrete user actions and concrete commands. This programming model creates strong coupling between tools and their targets by binding objects and input methods to particular interactions in program code. It also creates interactions that are opaque to users, because they have no knowledge of which bindings are in effect at any point in time.

Several programming models have been developed as alternatives to event-driven programming, such as functional reactive programming [8, 7] and hierarchical state machines [4]. These models attempt to improve on the limitations of event-driven programming for maintainable and flexible code, but do not address the user-facing flexibility of tools.

The app model is the result of common architectures and software engineering principles that have good properties for engineers and developers, *but not for users*. From an engineering perspective, the app model is very reasonable: Encapsulation means that each application can be developed with the assumption that it exists in a vacuum. It creates less opportunities for users to cause errors, and allows designers to keep a lot of control over how their software is used.

The limitations of the app model cannot be addressed at the level of individual systems. Rather, we should investigate and demonstrate alternative programming models and architectures that embed qualities such as reinterpretability, resilience, and scalability.

EXAMPLES OF RELEVANT SYSTEMS AND TOOLKITS

There are many cases of research advocating for and investigating user-facing flexibility. For example, Meyrowitz [14] critiqued the monolithic aspect of hypertext systems at the time, and Gat [9] argued for abolishing the division between users and programmers. The early Butttons system [13] demonstrated how to create and exchange small interactive components; Wulf et al. [17] created an architecture supporting the notion of "casual programmer"; Newman et al. [15] describe a system that encourages opportunistic uses of resources discovered in a ubicomp environment; Shared Substance [10] provides a flexible environment for multi-surface interaction.

I emphasize the next two examples as they are particularly relevant to the approach descibed in the next section: Olsen gives an analysis of how the Unix operating system model, where "everything is a file", allows users to compose and extend (terminal-based) tools flexibly: *"In the UNIX environment all commands are expected to read ASCII text from standard input and write ASCII to standard output. By unifying everything around ASCII text it was possible to build a wide range of pluggable tools that would store, extract, search, edit, and transform text. Because programs output readable text, users could readily see how some other program could manipulate such output for purposes not considered by the creators. This recognition of potential new uses in information, coupled with standard tools for text manipulation, is very powerful."* [16] This principle of designing systems from small composable parts is echoed in both functional and object-oriented programming, but is rarely present in graphical user interfaces.

Webstrates [12] is an example of an interaction architecture that shifts flexibility to users. It is a web server that supports real-time sharing of a large class of HTML documents: any change made to the Document Object Model (DOM) of a page loaded in a web browser is sent to the server, stored, and broadcast to the other browsers that have loaded that page. *Webstrates* turns the web into a medium where sharing and transclusion, the ability to include one document within another, are basic properties of the document model. Klokmose et al. show that this shareable medium can serve as a building block to create multi-user, multi-device systems that are extensible and reconfigurable at run-time.

Webstrates demonstrates that a toolkit can use existing infrastructure to create a novel model of interactive software. In the same vein, I argue that an interaction architecture where reinterpretable tools are first-class objects can be created economically by relying on existing platforms, e.g., the web.

A SKETCH OF REINTERPRETABLE TOOLS

The *instrumental interaction* model describes *instruments* as the primary means of interacting with the digital world: *"An interaction instrument is a mediator or two-way transducer between the user and domain objects. The user acts on the instrument, which transforms the user's actions into commands affecting relevant target domain objects. Instruments have reactions enabling users to control their actions on the instrument, and provide feedback as the command is carried out on target objects"* [1].

Interaction instruments are a good starting point for reinterpretable tools, because they are explicit objects, conceptually independent from apps, as opposed to the rules used by event-driven programming to define the behavior of each domain object when clicked on, dragged, etc. The physical tool metaphor also has the advantage that it is clear to users which instruments are available and active at any given moment.

Instrumental interaction is a descriptive and generative model that has been applied to design systems with novel interfaces, such as a bimanual colored Petri-nets editor [3] or digital curation on a tabletop [5]. In a *toolkit* for interaction instruments, several questions for the entities and processes around instruments occur:

- How is an instrument described?
- How are instruments decoupled from both the devices used to manipulate them and the target objects they operate on?
- What are the user actions and commands that instruments transduce?

Describing instruments

Instrumental interaction covers both physical and logical devices. The latter consist of a graphical representation to show their state and represent feedback, input channels to receive user actions, and a logic for mapping actions on the instruments to operations on the instrument's target. There are multiple abstractions that could implement this logical component, such hierarchical state machines (HSM's) [4, 11]. HSM's can be described as simple, isolated systems, which can be composed to create more complex behaviors. Importantly, instrument are defined independently of their concrete input mechanisms and output targets.

The instrument chain

Instruments can be chained, e.g., a pen instrument can be operated with a mouse or with a stylus. An action performed through a chain of instruments might look like this: "Alice clicks and drags the mouse to move the cursor instrument, which operates the paintbrush instrument, which leaves a red trail on the canvas."

The instrument chain, from physical action to final result, is continuously established, broken and re-created through use. In real life, we grasp tools and assemble them, e.g., combine a pen and a ruler to draw a straight line. In software, we implement the grasping metaphor with simple actions, e.g., clicking to select. A toolkit for instrumental interaction should have a richer set of elementary gestures and simple rules to combine instruments into a chain. In particular, a low-level collision-detection routine would determine when objects overlap to establish (and break) the instrument chain: The dragging instrument would activate the paintbrush when clicked on top of it, and the paintbrush would determine that it is over the canvas and lay ink on it.

User actions as signals

Once the chain of instruments is established, they can exchange actions and reactions. In event-driven programming, events travel from manipulated objects to observers. This is inappropriate for instruments, because instruments should not limit what type of event can be applied to them. Functional reactive programming extends the notion of events to signals — time-varying streams of values. This seems more appropriate for instruments: interaction is represented as a signal travelling through the instrument chain.

If an instrument cannot distinguish between operating another instrument or manipulating a domain object, this implies that instrument input and output are isomorphic. At the end of the instrument chain, the result of an action is some combination of reading and writing to the target object's state. We therefore model instrument operation in terms of mutating state: Instruments send each other *operations*, which can be stated as sequences of insertions, deletions, and updates. Returning to the previous example, the dragging instrument changes the position of the paintbrush instrument, and the paintbrush adds brushstrokes to the canvas. Operations do not need to be interpreted, as opposed to events. This means that the set of possible operations is open, and can be extended by new instruments.

With support for instruments at the operating system level, interaction scenarios such as those described in the introduction become feasible. Under this architecture, applications could be replaced by packages of instruments that fit a coherent domain, e.g., word processing or illustration. Users are empowered to reuse, adapt, and combine instruments as they see fit: One user may reuse a text cursor that supports his most used editing commands for writing e-mails, filling out forms, and taking notes; another may adapt a pen instrument from a drawing suite to handwrite annotations when reviewing homework; and a third may combine a word processing suite and a math evaluation instrument to write math reports. The ability to chain instruments further supports combining and extending instrument behaviors, e.g., a pen instrument can be used for freehand drawing, but chaining the same instrument with a dragging instrument constrained by a geometric shape turns it into a pen for drawing shapes.

CONCLUSION

Architectures are not neutral in implementing interaction. The app model of software couples tools and the objects they manipulate at design-time. At a lower level, this coupling is reflected in the coupling between input and output in, e.g., event-driven programming. This interaction architecture translates into static systems that limit the ability of users to de-compose and re-compose interactive systems. Shifting flexibility towards users is an outstanding challenge for systems-oriented HCI.

Interactive systems should leverage our natural ability to reuse, adapt, and combine tools. The tool metaphor of instrumental interaction is a good starting point for treating interactions as first-class objects and making interactive systems more flexible. By providing architectural support for interaction instruments, I hope to demonstrate the power of this approach. The ability to compose instruments should not only benefit end-users, but would also motivate software producers to create systems that are small and composable, rather than monolithic and isolated [9].

ACKNOWLEDGMENTS

Thanks to Michel Beaudouin-Lafon for editing this paper. This work was partially funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme grant n° 695464 ONE: Unified Principles of Interaction.

REFERENCES

1. Michel Beaudouin-Lafon. 2000. Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces. (2000), 446–453. DOI: http://dx.doi.org/10.1145/332040.332473
2. Michel Beaudouin-Lafon. 2004. Designing Interaction, Not Interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '04)*. ACM, New York, NY, USA, 15–22. DOI: http://dx.doi.org/10.1145/989863.989865
3. Michel Beaudouin-Lafon and Henry Michael Lassen. 2000. The Architecture and Implementation of CPN2000, a post-WIMP Graphical Application. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology (UIST '00)*. ACM, New York, NY, USA, 181–190. DOI: http://dx.doi.org/10.1145/354401.354761
4. Renaud Blanch and Michel Beaudouin-Lafon. 2006. Programming Rich Interactions Using the Hierarchical State Machine Toolkit. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '06)*. ACM, New York, NY, USA, 51–58. DOI: http://dx.doi.org/10.1145/1133265.1133275
5. Frederik Brudy, Steven Houben, Nicolai Marquardt, and Yvonne Rogers. 2016. CurationSpace: Cross-Device Content Curation Using Instrumental Interaction. In *Proceedings of the 2016 ACM on Interactive Surfaces and Spaces (ISS '16)*. ACM, New York, NY, USA, 159–168. DOI:http://dx.doi.org/10.1145/2992154.2992175
6. Henrik Bærbak Christensen. 2010. *Flexible, reliable software: Using patterns and agile development*. Taylor and Francis(Chapman and Hall/CRC).
7. Evan Czaplicki and Stephen Chong. 2013. Asynchronous Functional Reactive Programming for GUIs. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '13)*. ACM, New York, NY, USA, 411–422. DOI: http://dx.doi.org/10.1145/2491956.2462161
8. Conal Elliott and Paul Hudak. 1997. Functional Reactive Animation. In *Proceedings of the Second ACM SIGPLAN International Conference on Functional Programming (ICFP '97)*. ACM, New York, NY, USA, 263–273. DOI: http://dx.doi.org/10.1145/258948.258973
9. Erann Gat. 2001. Programming Considered Harmful. *OOPSLA 2001 Feyerabend Workshop* (2001).
10. Tony Gjerlufsen, Clemens N. Klokmose, James Eagan, Clément Pillias, and Michel Beaudouin-Lafon. 2011. Shared Substance: Developing Flexible Multi-Surface Applications. (2011), 3383–3392. DOI: http://dx.doi.org/10.1145/1978942.1979446
11. Clemens N. Klokmose and Michel Beaudouin-Lafon. 2009. VIGO: Instrumental Interaction in Multi-Surface Environments. Association for Computing Machinery (ACM). DOI:http://dx.doi.org/10.1145/1518701.1518833
12. Clemens N. Klokmose, James R. Eagan, Siemen Baader, Wendy E. Mackay, and Michel Beaudouin-Lafon. 2015. Webstrates: Shareable Dynamic Media. (2015), 280–290. DOI:http://dx.doi.org/10.1145/2807442.2807446
13. Allan MacLean, Kathleen Carter, Lennart Lövstrand, and Thomas Moran. 1990. User-tailorable Systems: Pressing the Issues with Buttons. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '90)*. ACM, New York, NY, USA, 175–182. DOI: http://dx.doi.org/10.1145/97243.97271
14. Norman K. Meyrowitz. 1989. The Missing Link: Why We're All Doing Hypertext Wrong. In *The society of text: hypertext, hypermedia, and the social construction of information*. MIT Press, 107–114.
15. Mark W. Newman, Jana Z. Sedivy, Christine M. Neuwirth, W. Keith Edwards, Jason I. Hong, Shahram Izadi, Karen Marcelo, and Trevor F. Smith. 2002. Designing for Serendipity: Supporting End-user Configuration of Ubiquitous Computing Environments. In *Proceedings of the 4th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques (DIS '02)*. ACM, New York, NY, USA, 147–156. DOI:http://dx.doi.org/10.1145/778712.778736
16. Dan R. Olsen, Jr. 1999. Interacting in Chaos. *interactions* 6, 5 (Sept. 1999), 42–54. DOI: http://dx.doi.org/10.1145/312683.312720
17. Volker Wulf, Volkmar Pipek, and Markus Won. 2008. Component-based tailorability: Enabling highly flexible software applications. *International Journal of Human-Computer Studies* 66, 1 (2008), 1–22.

The Evaluation of a Physiological Data Visualization Toolkit for UX Practitioners: Challenges and Opportunities

Vanessa Georges
HEC Montréal
Montréal, Canada
vanessa.georges@hec.ca

François Courtemanche
HEC Montréal
Montréal, Canada
francois.courtemanche@hec.ca

Sylvain Sénécal
HEC Montréal
Montréal, Canada
sylvain.senecal@hec.ca

Pierre-Majorique Léger
HEC Montréal
Montréal, Canada
pml@hec.ca

Lennart Nacke
University of Waterloo
Waterloo, Canada
lennart.nacke@acm.org

Marc Fredette
HEC Montréal
Montréal, Canada
marc.fredette@hec.ca

ABSTRACT
The objective of this paper is to outline the challenges we have encountered during the development and evaluation of our Physiological Data Visualization (UX Heatmap) toolkit and discuss the opportunities that emerged from our experience. The main goal of the Physiological Data Visualization toolkit is to allow simpler and richer interpretation of physiological signals for UI evaluation, in order to reduce the barriers associated with the use of physiological measures in the fields of user experience design and research. Following a user test with 11 UX experts from the industry, we were able to better understand how and in which contexts they would use the proposed UX Heatmap toolkit in their practice.

Author Keywords
Interface design; heatmaps; physiological computing; affective computing, toolkits.

ACM Classification Keywords
H.2.1Design; Experimentation; HCI; Human Factors; Measurement; User interfaces. I.3.6 Computer graphics: Methodology and Techniques.

INTRODUCTION
Measuring the emotional state of users during the interaction is essential to the design of richer user experience. Users' emotional and cognitive states can be inferred using physiological signals such as electrodermal activity, heart rate, eye tracking, and facial expressions [1-2]. These measures can provide important temporal information to UX experts as to what the user is experiencing throughout the interaction without retrospective or social desirability bias [3]. However, these measures are still difficult to contextualize and interpret as they are not specifically associated with user behavior or interaction states. Physiological signals also require a certain degree of interpretation, as outputs need to be processed in order to transition from raw data to useful actionable insights. The objective of the proposed UX Heatmap toolkit is to address these issues and help UX experts incorporate physiological data in their user tests.

TOOLKIT OVERVIEW
Traditional gaze heatmaps are used in eyetracking as intuitive representations of aggregated gaze data [4]. Their main use is to help researchers and UX experts answer the question: "Where in the interface do people tend to look?" [5]. In the proposed UX Heatmap toolkit, the users' gaze now serves as a mean of mapping physiological signals onto the user interface. The resulting heatmaps represent the physiological signals' distribution over the interface, and can help answer the following question: "Where in the interface do people tend to emotionally or cognitively react more strongly?"

Toolkits aim at facilitating and fostering the adoption of emerging and complex technologies to new and non-expert populations [6]. Our toolkit aims to tackle the following challenges regarding the use of physiological measures in the prototyping and evaluation of user interfaces by UX experts: (1) data synchronization of multiple signals from various apparatus (e.g., visual attention using eyetracking and arousal using electrodermal activity), (2) valid physiological inferences (e.g., taking into account different signal latencies), and (3) the interpretation and contextualization of data using multiple types of visualizations. We thus designed a visualization toolkit that contextualizes physiological and behavioral signals to facilitate their use, see figure 1 [7].

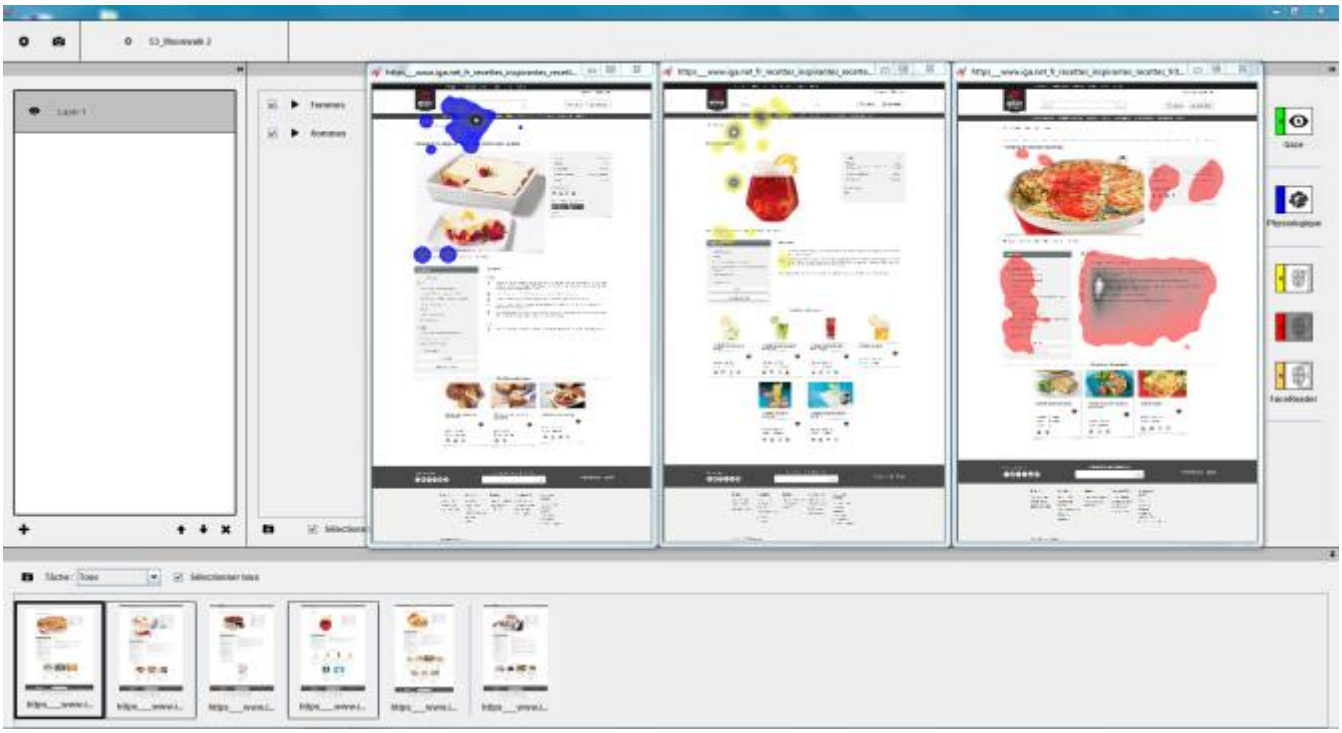

Figure 1. Screenshot of the UX Heatmap Toolkit Software Interface.

The Motives behind the Toolkit

Our recent work with industry has lead us to question a major discrepancy between industry and academic practices: while physiological measures are increasingly used in research, the adoption of these methods as UX evaluation tools remain uncommon in industry. From our experience interacting with the industry, there is a growing demand for more quantitative user research to provide data-driven recommendations based on physiological data such as eye tracking and emotional reactions. We therefore wanted to understand what can be done to facilitate their adoption in the industry.

Furthermore, physiological measures, in combination with traditional methods, can help practitioners to better evaluate the emotional dimension of user experience, which focuses on emotional responses triggered by the system interaction [8-10], as they each provide complementary information on how users feel about a system, game, or web interface [11]. While traditional evaluation methods can offer episodic data, i.e., before or after the interaction, physiological measures can provide moment-to-moment information [12]. For example, the addition of physiological measures can help practitioners identify the cognitive and emotional reactions users experienced using an interface, while a post-task interview can help delve further, after having identified these emotions. Therefore, this toolkit was built for UX experts, whose needs differ from those of academic researchers. While the former's needs are to analyze and adequately communicate findings to improve user experience, the latter's interest resides in the validation and understanding of phenomena, based on hypotheses [13].

PREVIOUS WORKS

Although there exists a series of HCI evaluations toolkits, the majority do not include physiological measures. Most usability toolkits simply guide the user toward the right tool to use for his or her context. However, certain toolkits do integrate physiological measures, tackling one aspect of the problem at hand. Most researchers have concentrated their efforts on finding ways to measure physiological signals and interaction states synchronously. For example, Kivikangas et al. [14] have developed a triangulation system to interpret physiological data from video game events. Dufresne et al. [15] have proposed an integrated approach to eyetracking-based task recognition as well as physiological measures in the context of user experience research. Other researchers have also developed tools that allow users' to manually assign subjective emotional ratings on visual interfaces [16] or to visualize emotional reactions in terms of GUI widgets [17]. While these research streams have produced interesting results, they are not easily transferable to new contexts of use, as they are based on internal information from the interactive system (e.g., video game logs, application events, or areas of interest). So, we looked to package and streamline this process and provide an integrated solution for UX experts.

THE EVALUATION OF THE TOOLKIT

To evaluate the proposed UX Heatmap toolkit, we conducted a study with eleven UX practitioners and consultants. Each interview lasted about one hour and a half, following a variation on the think aloud protocol, cooperative evaluation [18]. As such, participants were

asked to talk through what they were doing; the interviewer taking on a more active role by asking questions along the way (e.g., 'Why?' 'What do you think would happen?').

UX experts were also asked to complete a user testing evaluation report using the toolkit. The PowerPoint report included a study summary, a research scenario and qualitative data. Participants were first briefed on the task at hand (i.e., complete a UX report for an online grocer using the UX Heatmap toolkit), before going through the partially completed report with the interviewer, to put them into context and get a sense of what was required of them. Participants had to complete two PowerPoint slides. The physiological signal data set used for the evaluation was collected in a previous study [19]. Practitioners were asked to: (1) generate and select data visualizations to include in their report using the toolkit, (2) interpret the results, and (3) provide recommendations to the client. The remainder of the time was used to discuss of the advantages and disadvantages physiological measures as a UX evaluation method, as well as the toolkit itself. This evaluation task was added to help UX experts integrate the information on physiological measures quickly and effectively, and to give them a concrete opportunity to use the toolkit in order to envision themselves using it in their own practice.

WHAT WE LEARNED

The goal behind the creation of the Physiological Data Visualization toolkit was to bring physiological measures to UX practitioners. Below are some of the findings we uncovered during the evaluation process of this toolkit.

Having a clear understanding of: **(1) the constraints and limitations of potential user's,** as this may limit the extent to which these users will consider the toolkits as a viable addition to their practice. The way in which researchers process and use information is quite different than the way practitioners do. For example, the automatization of certain functionalities (i.e. participant and layer creation), which we saw as superfluous, or nice-to-haves, were seen as crucial by practitioners in order to accelerate the interpretation of the visualizations generated with our toolkit. For researchers, this represent a minimal gain; as for practitioners faced with short development cycles, this is seen as saving, both time and money, two important attributes when choosing new methodologies.

(2) potential users' context of use, as this may limit the extent to which novice practitioners and researchers will consider these technologies as a viable addition to their projects. When asked about their intent to reuse the tool, ten out of eleven practitioners interviewed stated that they would use the toolkit in their practice. When inquired further, six of them declared that their use of the tool would depend on the projects, using it only in the interventions where emotions are an important component or if clients specifically requested them to use physiological signals. This could translate into a steep and ever present learning curve, as practitioners must re-learn how to use the toolkit and materials associated with the data collection of physiological signals at each use. As a result, it could be challenging for practitioners to master the toolkit if only used occasionally. Therefore, a barrier to entry exist for professionals unable to justify the financial investment due to sparse usage of such tools (i.e., eyetrackers, sensors).

Additional Challenges related to physiological measure usage by UX experts: **reliance on external sources and tools can become an issue.** For non-expert users, learning how to correctly take ownership of a new technology using a toolkit is already an endeavor; the addition to peripheral techniques and methods creates a much higher learning curve than they may have initially anticipated. Although the Physiological Data Visualization toolkit makes physiological measures more accessible to UX practitioners by addressing the interpretation of signals, there remains a need to better educate professional on some of the more technical aspects of physiological measurements. Data collection, experimental setup, and data extraction still have to be overseen by UX experts, representing an important time and financial constraints.

CONCLUSION

The objective of this paper was to highlight the challenges we have encountered during the development and evaluation of our Physiological Data Visualization toolkit and discuss the opportunities that emerged from our experience. Designing, building and sharing toolkits are a great way to bring new technologies to the larger HCI community. Understanding the end user is primordial to the development of a toolkit that will truly reach its goal and audience, which we uncover through the evaluation of out toolkit, which makes discussions on the methods and metrics that can be used to evaluate toolkits all the more interesting. By pursuing our working with potential users during the design and development phases of the toolkit, our objective is to continue to reduce the barriers associated with the use of physiological measures in the fields of user experience design and research.

ACKNOWLEDGMENTS

Authors want to thank Brendan Scully for manuscript revision and the UX practitioners who participated to this study This work was supported by the CRSNG *(Conseil de recherches en sciences naturelles et en génie du Canada).*

REFERENCES

1. Zhihong, Z., Pantic, M., Roisman, G.I. and Huang, T.S. 2009. A Survey of Affect Recognition Methods: Audio, Visual, and Spontaneous Expressions. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31, 1: 39-58.
2. Calvo, R.A. and D'Mello, S. 2010. Affect Detection: An Interdisciplinary Review of Models, Methods, and Their Applications. *Affective Computing, IEEE Transactions on* 1, 1: 18-37.

3. King, M.F. and Bruner, G.C. 2000. Social desirability bias: A neglected aspect of validity testing. *Psychology and Marketing* 17, 2: 79-103.
4. Nielsen, J. and Pernice, K. *Eyetracking Web Usability*. New Riders, Berkeley, California, 2012.
5. Wooding, D.S. Fixation maps: quantifying eye-movement traces *Proceedings of the 2002 symposium on Eye tracking research & applications*, ACM, New Orleans, Louisiana, 2002, 31-36.
6. Cartwright, William, et al. "Geospatial information visualization user interface issues." *Cartography and Geographic Information Science* 28.1 (2001): 45-60.
7. Georges, Vanessa, et al. "UX heatmaps: mapping user experience on visual interfaces." *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 2016.
8. Février, F., N. Gauducheau, E. Jamet, G. Rouxel, and P. Salembier, *La prise en compte des affects dans le domaine des interactions humain-machine.* Le travail humain, 2011. **74**: p. 183-201.
9. Mahlke, S. and M. Minge, *Consideration of Multiple Components of Emotions in Human-Technology Interaction*, in *Affect and Emotion in Human-Computer Interaction*, C. Peter and R. Beale, Editors. 2008, Springer Berlin Heidelberg. p. 51-62.
10. Hassenzahl, M., *The Interplay of Beauty, Goodness, and Usability in Interactive Products.* Human–Computer Interaction, 2004. **19**(4): p. 319-349.
11. Roto, Virpi, Marianna Obrist, and Kaisa Väänänen-Vainio-Mattila. "User experience evaluation methods in academic and industrial contexts." *Proceedings of the Workshop UXEM*. Vol. 9. 2009.
12. Zhihong, Z., Pantic, M., Roisman, G.I. and Huang, T.S. 2009. A Survey of Affect Recognition Methods: Audio, Visual, and Spontaneous Expressions. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31, 1: 39-58.
13. Roto, V., Vermeeren, A.P.O.S., Väänänen-Vainio-Mattila, K., Law, E. and Obrist, M., Course notes: User Experience Evaluation Methods - Which Method to Choose? in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, (Paris, France, 2013), ACM.
14. Kivikangas, M., Nacke, L. and Ravaja, N. 2011. Developing a triangulation system for digital game events, observational video, and psychophysiological data to study emotional responses to a virtual character. *Entertainment Computing* 2, 1: 11-16.
15. Dufresne, A., Courtemanche, F., PromTep, S. and Sénécal, S., Physiological Measures, Eye Tracking and Task Analysis to Track User Reactions in User Generated Content. in *Measuring Behavior*, (Eindhoven, The Netherlands, 2010), 218-222.
16. Huisman, G., Hout, M.v., Dijk, E.v., Geest, T.v.d. and Heylen, D. LEMtool: measuring emotions in visual interfaces *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, Paris, France, 2013, 351-360.
17. Cernea, D., Weber, C., Ebert, A. and Kerren, A., Emotion scents: a method of representing user emotions on gui widgets. in *IS&T/SPIE Electronic Imaging*, (2013), International Society for Optics and Photonics, 86540F-86540F-86514.
18. Dix, Alan. *Human-computer interaction*. Springer US, 2009.
19. Desrocher, C., et al. "The influence of product type, mathematical complexity, and visual attention on the attitude toward the website: The case of online grocery shopping." *Fourteenth Pre-ICIS SIG-HCI Workshop, Fort Worth, TX*. 2015.

Playing the Tricky Game of Toolkits Research

Michael Nebeling
University of Michigan School of Information
nebeling@umich.edu

ABSTRACT
In this paper, I reflect on my experience from the past several years conducting toolkit driven multi-device interaction research that appeared in CHI and EICS. I discuss lessons learned and share my perspective on the larger field of user interface engineering, including what I think the main challenges and opportunities are with toolkits research and good examples of it. I hope that sharing my perspective is useful for the new generation of researchers interested in, and potentially struggling with, doing engineering research in HCI.

Author Keywords
technical HCI; systems research; user interface toolkits.

BEFORE I GET STARTED

Toolkits and system-driven research is one of the most challenging, but perhaps also the most interesting, kinds of research we have in HCI. It is challenging for many reasons: *results wise*—because it takes a lot of time and effort to create a system that can be studied to answer the research questions behind it, *process wise*—because every system is different and there are too many technical, design and evaluation challenges that cannot all be addressed at once and therefore need to be well balanced, and *publication wise*—because the resulting artifact is likely to come close to other systems and it is neither an easy task for authors to articulate the differences nor for reviewers to judge whether these are significant.

I learned this the hard way as a PhD student interested in designing systems and tools that solve real-world problems. I started out with publication attempts in web engineering and HCI conferences, and was pushed between the two worlds as neither wanted to accept my work. For web engineering, I did too much on interfaces, and for HCI it was too much engineering. I also think part of the problem was lack of a clear research method. I did not think of system building as a research method at the time; in fact, I was warned about it and, depending on who I talk to, sometimes still struggle to explain that even though my research involves a lot of engineering, it is still research. When I was introduced to Alan Hevner's design science research, I thought that is what I was doing. However, I still think that even with specific engineering committees at CHI and whole conferences such as EICS, there is still a lot of confusion about the science part, and we still have a hard time acknowledging systems research.

My PhD thesis [17] essentially developed around a set of tools [21, 24, 25] that I designed to investigate new methods and techniques to create more flexible and adaptive interfaces. I was interested in this topic because of the ongoing proliferation of new computing devices, with many new touch devices coming out in all kinds of form factors since the iPhone started the trend in 2007. This was a risky PhD topic for all the three reasons stated above; in particular, there was already a long history of research into context-aware and adaptive interfaces. However, existing research struggled to meet the needs of practitioners and industry as the proposed solutions did not always seem useful and practical. This introduced me to two additional tools research challenges.

First, some of the tools that I created (e.g., jQMultiTouch [24] and W3Touch [25]) were much more simple and practical than a lot of the existing user interface research which was based on more generic notions of context awareness and complex model-based approaches. While my work seemed more closely aligned with practitioner needs, it also seemed less generalizable and, to some, probably even less "research-y."

Second, some of the techniques that I created ended up being similar to what is now called "responsive web design." While I would argue that my PhD thesis pioneered many of the concepts, or at least developed them in parallel, it is difficult to hold up this claim because of articles in popular science[1] that appeared before a Master's thesis [31] was published in [22].

After my PhD thesis, I started to work on cross-device interfaces, which seemed like a natural follow-on and nice extension of my prior work on context-adaptive interfaces because techniques had so far been limited to adapting interfaces to one device at a time. As part of this research, I created a family of *XD* tools that addressed all kinds of issues around the design, development, and testing of cross-device interfaces. For example, I created XDStudio [23], a new GUI builder for visually designing distributed user interfaces for multi-device environments such as meeting rooms or classrooms, investigating simulated and cross-device authoring strategies. After XDStudio, I created tools like XDKinect [26] to enable rapid prototyping of cross-device interfaces using Kinect as an intermediator, XDSession [20] to provide new tools for developing and testing cross-device interfaces based on useful

#HCI.Tools 2017: Strategies and Best Practices for Designing, Evaluating and Sharing Technical HCI Toolkits

[1] https://alistapart.com/article/responsive-web-design

abstractions in a multi-device data session concepts, and XDBrowser to enable end-users by making the concepts of distributed interfaces so far limited to toolkits directly available in web browsers. In a first paper [19], I used XDBrowser to study what kinds of cross-device interfaces end-users would want to have given an existing single-device interface. This study led to a first set of cross-device patterns. In a second paper [18], I used XDBrowser to study how single-device interfaces can be semi-automatically transformed into cross-device interfaces based on the patterns.

OPPORTUNITIES FOR TOOLKITS RESEARCH

While most of the paper talks about challenges, let us start by highlighting a few of the opportunities for toolkits research.

First, by doing research on novel kinds of user interfaces and toolkits to support the creation of them we as researchers have an important say in what the next generation of user interfaces might be. For example, a lot of the research on multi-touch was only made possible through new technologies such as the DiamondTouch table [7] and toolkits such as DiamondSpin [30]. They formed the basis of a wide range of studies on multi-touch interaction and collaborative tabletop interfaces for many years and still continue to play a role even today.

Second, toolkits are important to push two primary aspects of research: concepts and applications.

When examining a toolkit, I look for interesting new concepts that make existing techniques significantly easier and/or faster. In practice, jQuery and Bootstrap are two of the most disruptive JavaScript and HTML/CSS toolkits we have in the web development domain. The elegance, expressiveness, and power of both found such wide adoption among developers and also researchers (jQMultiTouch [24], Weave [2]) that some of the concepts made it to the HTML5 and CSS3 standards. In research, web automation and manipulation toolkits like Chickenfoot [1] and CoScripter [14] had similar impact due to their concepts being based on rendered web pages and sloppy keywords rather than proper references to interface elements in code. A lot of the research on end-user scripting and programming by demonstration was pushed by these toolkits with Highlight [27] being an example that builds on CoScripter to enable the desktop-to-mobile adaptation based on end-user demonstration of desired interactions.

The other major question I ask about a toolkit is what kinds of new applications it enables. For example, when looking at cross-device toolkits such as Panelrama [32], Weave [2], and WatchConnect [8], one thing to notice is the increased effort to support cross-device interfaces around smartwatches. In WatchConnect, this effort does not stop with toolkit support in software. Rather, it also provides hardware support for developers to create new kinds of smartwatch sensors.

CHALLENGES FOR TOOLKITS RESEARCH

Now let us turn to some of the challenges for toolkits research. These range from practical, to technical, to methodical challenges. Another challenge is the writing of a toolkits paper itself. Despite some good pointers and recommendations from senior researchers in the field [28, 9], my own experience both as an author and as a reviewer for CHI and EICS for many years shows that there is still little agreement among researchers on what makes good systems research.

Staying Ahead of the Game

Toolkits research should always attempt to stay ahead of the game. I have seen many "good" papers rejected because they either did not significantly push the concepts, the applications, or both parts. It seems harder to "sell" a toolkit that tackles an old problem, even though it might do it very well, than a toolkit that tackles a new problem, even though it might just be scratching the surface. So one way to alleviate shortcomings in toolkit design can be targeting cutting-edge interaction technologies. For example, I would say that the earlier generation of multi-touch toolkits did not innovate with concepts, but it enabled new applications. In the later generation (e.g., Proton [12]), this shifted towards new concepts that essentially enabled very similar applications, but did so in much more innovative ways. This was quite similar with multimodal and multi-device toolkits. After crowdsourcing, it is currently 3D printing and fabrication that receive a lot of interest in systems research. Note that many of these technologies were not novel at the time; rather, we speak of the multi-touch and 3D printing *re*volution. Interestingly, although IoT definitely received a big push in industry, in toolkits research this was not so much the case. The researchers that I know worked on IoT toolkits (e.g., fabryq [16], Bluewave [6]) were given a hard time making the unique challenges clear given that a lot of the problems seemed to have already been addressed by prior multi-device research. Given the proliferation of new VR/AR consumer devices, it will be interesting to see whether there will be another wave of VR/AR toolkits, perhaps focused on wearable devices, after the success of projection-based toolkits such as RoomAlive [10]. In any case, support for blending the physical and the digital design worlds will become more important in the future. Again, WatchConnect [8] is a good example here as it supports both software and hardware interface prototyping in one toolkit.

Balancing Toolkit Practicality and Generalizability

This goes back to what I said earlier about practical vs. generic solutions. The literature on model-driven user interface research is full of comprehensive approaches based on complex models and multi-level abstractions. For example, MARIA [29] is a versatile and powerful model-based framework that was created based on many years of research. Yet, the process required to define interfaces and the kinds of interfaces that can be generated in the end often seem neither practical nor complex. My stance on this is that less is more. It is okay if a particular proposal does not provide full-fledged support as long as the design rationale is sound and limitations are clearly articulated. I find an elegant solution for a well-scoped interface problem is more likely to generate concrete results and hence gain traction as long as it improves, rather than trying to replace, existing workflows. As an inspiring example, I would like to mention the case of Adobe Lightroom here [11], where studies with professional and serious amateur photographers provided unique insights into their existing patchwork processes and how to best provide a solution that integrates well with Adobe Photoshop.

Designing for the Next Generation of Designers

Another common pitfall with toolkits research is not clearly identifying the users. This was not so much a problem some years ago when the distinction between users and developers was clearer, but given that users nowadays often are both consumers and producers thanks to enabling tools, the line becomes fuzzier. In the research on end-user programming, the term "end-user" was commonly used to refer to non-technical users as opposed to developers with programming skill. It can help to put the research into the appropriate context by citing relevant research in that domain (e.g., from [4]), but it is better to make it explicit by clearly stating the assumed skill of target users and ideally include studies that help identify the needs of those users. This is something that I think was quite well done in Snap-to-It [5]. It goes without saying that the expectation will be that it is also those kinds of users that will be recruited for testing a toolkit as part of the evaluation. For many of my cross-device systems, I had to explain and justify why I studied with participants that only had experience creating mobile and responsive interfaces rather than "real" cross-device developers. It took some effort to convince reviewers that this generation of developers does not exist yet, as the solutions so far are often still research prototypes and it will take some time before they mature and are picked up.

Dealing with the Proliferation of New Toolkits

In some of the recently booming areas such as multi-touch or cross-device interfaces, a large number of toolkits were created and documented in the literature. In particular, in the cross-device domain, many of them almost seem to have been developed in parallel, without actually citing or building on top of each other. I remember presenting in the CHI 2014 session on multi-device interfaces and all of us were surprised to see that we worked on toolkits pushing similar ideas and developing many of the same features. I was surprised to see the sheer number of cross-device toolkits that came out in 2014 and 2015. I would say that in some areas the "market" is saturated and any new attempt to publish a toolkit may just be turned down as "yet another toolkit." This does not mean that there is no more need for new systems research, but it will become increasingly difficult for a new toolkit to be significantly different in the concepts that it proposes or applications that it enables. Exceptions include the Weave and WatchConnect toolkits mentioned earlier, where jQuery-like device selection techniques, storyboard generation from cross-device code [3], and support for hardware prototyping of smartwatch interfaces added significant research value.

Releasing Toolkits to Facilitate Toolkits Research

Last, I wanted to raise the issue that most toolkits research I know ended with the release of the toolkit for download. This is necessary but not sufficient. It is necessary to enable others to try out toolkits and do comparative evaluations, which is often asked by reviewers despite the fact that many previous systems are not actually available and only "exist" in research papers. It is not sufficient, however, because in most cases this is where the toolkits research actually begins. To truly understand the capabilities and value proposition of a toolkit, it is important to study how it is used by others than the toolkit authors and a few study participants. The true value of Chickenfoot and CoScripter was revealed when others started to adopt the ideas and build on top of those tools. Unfortunately, in the multi-device domain, there are no such leading examples. Perhaps tools like Webstrates [13] and XDBrowser [19] could grow into that role as they recently enabled some workshop activities at CHI and EICS. But we need more hands-on and discussion-heavy workshops like XDUI, Cross-Surface and now #HCI.Tools to foster discussion around those issues.

WORKSHOP CONTRIBUTION AND ACTIVITIES

In summary, I am excited about the #HCI.Tools workshop to be held at CHI 2017. I would like to contribute my experience and knowledge in the form of discussion or presentation of selected research prototypes from my *XD* tools research.

Moreover, I would like to propose three types of activities that I think would benefit #HCI.Tools workshop participants.

First, I have previously organized mock program committees reviewing systems papers and run reading groups analyzing examples of "good" toolkits papers. This would require some prep work of participants, but could be limited to one paper, e.g., WatchConnect [8] and a framework such as Olsen's [28].

Second, I think it will be interesting to look at how toolkits research has evolved over time both in terms of design and evaluation. An activity that extracts best practices and guidelines from a larger corpus of papers could be based on group work reviewing selected genres of toolkit papers, e.g., on multi-touch or cross-device interfaces, or time, e.g., before and after Olsen's framework appeared at UIST 2007.

Third, let us discuss new trends in toolkits research such as blending the digital and the physical, and again the impact on both design and evaluation. Interesting examples can be found in the research on cross-device and proxemic interaction, including WatchConnect and Proximity Toolkit [15].

ABOUT THE AUTHOR

Michael Nebeling is an Assistant Professor at the University of Michigan School of Information. He investigates new methods, tools and technologies that enable users to interact with information in more natural and powerful ways, and also make it easier for designers to create more usable and effective user interfaces. As part of his research, he has created many systems to support the design and evaluation of rich, context-aware and adaptive, cross-device, multi-touch and multi-modal gesture and speech interfaces. He is committed to promoting engineering research within the HCI community. He has been an Associate Chair for the CHI Technology, Systems and Engineering subcommittee for CHI 2014-2016. He was EICS 2015 Papers co-chair and EICS 2014 Late-Breaking Results co-chair. He has been a member of the steering committee and Senior PC for EICS since 2016.

REFERENCES

1. Bolin, M., Webber, M., Rha, P., Wilson, T., and Miller, R. C. Automation and Customization of Rendered Web Pages. In *Proc. UIST* (2005).

2. Chi, P. P., and Li, Y. Weave: Scripting Cross-Device Wearable Interaction. In *Proc. CHI* (2015).
3. Chi, P. P., Li, Y., and Hartmann, B. Enhancing cross-device interaction scripting with interactive illustrations. In *Proc. CHI* (2016).
4. Cypher, A., Dontcheva, M., Lau, T., and Nichols, J. *No Code Required: Giving Users Tools to Transform the Web*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010.
5. de Freitas, A. A., Nebeling, M., Chen, X. A., Yang, J., Ranithangam, A. S. K. K., and Dey, A. K. Snap-to-it: A user-inspired platform for opportunistic device interactions. In *Proc. CHI* (2016).
6. de Freitas, A. A., Nebeling, M., Ranithangam, A. S. K. K., Yang, J., and Dey, A. K. Bluewave: Enabling Opportunistic Context Sharing via Bluetooth Device Names. In *Proc. EICS* (2016).
7. Dietz, P. H., and Leigh, D. Diamondtouch: a multi-user touch technology. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology, UIST 2001, Disney's BoardWalk Inn Resort, Walt Disney World, Orlando, Florida, USA, November 11-14, 2001* (2001).
8. Houben, S., and Marquardt, N. Watchconnect: A toolkit for prototyping smartwatch-centric cross-device applications. In *Proc. CHI* (2015).
9. Hudson, S. E., and Mankoff, J. *Concepts, Values, and Methods for Technical Human–Computer Interaction Research*. Springer New York, New York, NY, 2014, 69–93.
10. Jones, B. R., Sodhi, R., Murdock, M., Mehra, R., Benko, H., Wilson, A., Ofek, E., MacIntyre, B., Raghuvanshi, N., and Shapira, L. Roomalive: magical experiences enabled by scalable, adaptive projector-camera units. In *Proc. UIST* (2014).
11. Kim, G. K. Early Research Strategies in Context: Adobe Photoshop Lightroom. In *Proc. CHI EA* (2007).
12. Kin, K., Hartmann, B., DeRose, T., and Agrawala, M. Proton: multitouch gestures as regular expressions. In *Proc. CHI* (2012).
13. Klokmose, C. N., Eagan, J. R., Baader, S., Mackay, W. E., and Beaudouin-Lafon, M. *Webstrates*: Shareable dynamic media. In *Proc. UIST* (2015).
14. Leshed, G., Haber, E. M., Matthews, T., and Lau, T. A. CoScripter: Automating & Sharing How-To Knowledge in the Enterprise. In *Proc. CHI* (2008).
15. Marquardt, N., Diaz-Marino, R., Boring, S., and Greenberg, S. The Proximity Toolkit: Prototyping Proxemic Interactions in Ubiquitous Computing Ecologies. In *Proc. UIST* (2011).
16. McGrath, W., Etemadi, M., Roy, S., and Hartmann, B. fabryq: using phones as gateways to prototype internet of things applications using web scripting. In *Proc. EICS* (2015).
17. Nebeling, M. *Lightweight Informed Adaptation: Methods and Tools for Responsive Design and Development of Very Flexible, Highly Adaptive Web Interfaces*. PhD thesis, ETH Zurich, 2012.
18. Nebeling, M. XDBrowser 2.0: Semi-Automatic Generation of Cross-Device Interfaces. In *Proc. CHI* (2017).
19. Nebeling, M., and Dey, A. K. XDBrowser: User-Defined Cross-Device Web Page Designs. In *Proc. CHI* (2016).
20. Nebeling, M., Husmann, M., Zimmerli, C., Valente, G., and Norrie, M. C. XDSession: Integrated Development and Testing of Cross-Device Applications. In *Proc. EICS* (2015).
21. Nebeling, M., Matulic, F., and Norrie, M. C. Metrics for the Evaluation of News Site Content Layout in Large-Screen Contexts. In *Proc. CHI* (2011).
22. Nebeling, M., Matulic, F., Streit, L., and Norrie, M. C. Adaptive Layout Template for Effective Web Content Presentation in Large-Screen Contexts. In *Proc. DocEng* (2011).
23. Nebeling, M., Mintsi, T., Husmann, M., and Norrie, M. C. Interactive Development of Cross-Device User Interfaces. In *Proc. CHI* (2014).
24. Nebeling, M., and Norrie, M. C. jQMultiTouch: Lightweight Toolkit and Development Framework for Multi-touch/Multi-device Web Interfaces. In *Proc. EICS* (2012).
25. Nebeling, M., Speicher, M., and Norrie, M. C. W3Touch: Metrics-based Web Page Adaptation for Touch. In *Proc. CHI* (2013).
26. Nebeling, M., Teunissen, E., and Norrie, M. H. M. C. XDKinect: Development Framework for Cross-Device Interaction using Kinect. In *Proc. EICS* (2014).
27. Nichols, J., Hua, Z., and Barton, J. Highlight: A System for Creating and Deploying Mobile Web Applications. In *Proc. UIST* (2008).
28. Olsen Jr., D. R. Evaluating User Interface Systems Research. In *Proc. UIST* (2007).
29. Paternò, F., Santoro, C., and Spano, L. MARIA: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environments. *TOCHI 16*, 4 (2009).
30. Shen, C., Vernier, F., Forlines, C., and Ringel, M. Diamondspin: An extensible toolkit for around-the-table interaction. In *Proc. CHI* (2004).
31. Streit, L. Investigating Web Site Adaptation to Large Screens. Master's thesis, ETH Zurich, DOI: 10.3929/ethz-a-006250434, 2010.
32. Yang, J., and Wigdor, D. Panelrama: Enabling Easy Specification of Cross-Device Web Applications. In *Proc. CHI* (2014).