Look without Feel –A Basal Gap in the Multi-Touch Prototyping Process

Georg Freitag, Michael Wegner, Michael Tränkner, Markus Wacker

Informatik/Mathematik, Hochschule für Technik und Wirtschaft - Dresden

Abstract

Prototyping a user interface is an important workflow step to establish the *look* & *feel* of an application in early development. We discuss a model for this process and show that, currently, it is heavily skewed toward the *look* aspect. This could prove to be a problem when designing highly interactive natural user interfaces, which put a stronger emphasis on the *feel* of an application. In order to thoroughly analyze this gap we compare eight current prototyping tools, by using a multi-touch application scenario. From this evaluation we derive requirements for a tool more suited towards multi-touch prototyping.

1 Motivation

Each application comes with its own *look* and *feel,* carefully implemented by its designers. The *look* describes the appearance of an application, which is not necessarily restricted to properties like layout or color but can also include visual aspects of interactions. The *feel* of an application is described by its interactive behavior. This contains actions the user can take, response to input, and interaction cues like feedback and feed-forward. Nowadays, with increasing interactivity of applications there is a noticeable shift in relevance towards the *feel* aspect. This finding is particularly true for so called *natural user interfaces,* which use dynamic gestures and full body involvement instead of simple point&click interactions.

Unfortunately, prototyping currently is dominated by the *look* from the very first moment. Sketching a user interface is more *look* than *feel*. Creating a wireframe or a paper-prototype focuses more on the *look* than the *feel,* since it represents the abstract user interface of a future application without any interactive parts. Later prototypes include “interaction” as series of linked pictures or require the scripting of simple behaviors. Both neglect the importance of designing interactions during the complete process of prototyping, which is especially relevant for multi-touch applications. Here, user interface elements are highly interactive, can influence the behavior of other elements, and demand a high level of activity from the user.

In this paper, we will show that the latest and most common prototyping tools put too much focus on the *look* and create prototypes with little to no *feel*. To verify this we created a prototyping scenario, which was used to rate the tools. Based on these findings and factoring in the strength of each tool, we derive necessary characteristics of prototyping tools that put a stronger emphasis on the *feel* aspect. The outline of this paper is as follows: First of all, we describe the process of prototyping, its advantages, forms, and phases in chapter 2. In the next chapter we discuss related work, describe our test multi-touch scenario, and explain our list of basic and extended criteria. In chapter 4 we evaluate the eight presented prototyping tools and discuss the results for the individual criteria as well as our overall observations. We conclude with the definition of requirements for an ideal multi-touch prototyping tool and give a short outlook of what such a tool could look like.

2 Prototyping

All software development models organize the development process into different phases, like collection of requirements, implementation, or testing. The more recent of these models (e.g. Extreme Programming or Unified Process) embed these steps into an iterative process in which experience from previous passes is directly incorporated into the next one (Anderson et al. 2010, p. 254). This approach can also be adapted to the development of an application’s design and user interface. In this case, prototyping is not about presenting a first solution to a design problem, but to support the analysis of that problem (Adenauer & Petruschat 2012, 17). Nielsen (Nielsen 2001) places special emphasis on this early phase because, at this stage, ignorance can lead to substantial danger. Development could miss the central problem or only create a local (very specific) instead of a global maximum (Dix 2007, 220ff.). As one solution to this problem Nielsen proposes parallel design, in which multiple designers tackle a problem at the same time. Every one of them works alone and only in the subsequent step, the individual results are unified into a concept that can be further refined.

We concentrate on the development process for multi-touch applications and adopt the above approach for our own model, which consists of four phases (cf. Figure 1): We start with the orientation, the analysis of the problem and inception of a solution. The next phase is the concentration, where ideas are collected and iteratively refined. The third phase is the implementation of the proposed solutions, which finally leads to the last step, the optimization of specific aspects of a prototype. Each of these phases can be iterative, consisting of multiple cycles or may seamlessly shift into the next phase. Every cycle consists of the following steps: analysis, interaction design, user interface design, prototyping and evaluation. These steps are interconnected and strongly influence each other. The duration of a cycle generally gets longer with every repetition as all steps become more and more complex.

When looking at prototyping, we differentiate between two aspects, namely the process itself and the resulting product. Optimizing the process as described in (Adenauer & Petruschat 2012, 17ff.) is particularly important in the first two phases of our model. Since here the main objective is problem analysis and looking for solutions, the process has to be streamlined for collaboration so that ideas and skills of the entire team can be utilized effectively (Schrage 2000, 28). The product of the process or artifact (Adenauer & Petruschat 2012, 27) has priority during the implementation phase, when the *look* and the *feel* of an (MT-) application are being defined. When building these prototypes the economic principle of prototyping should be respected (Lim et al. 2008), meaning that a representative result should be reached with as little effort/cost as possible.

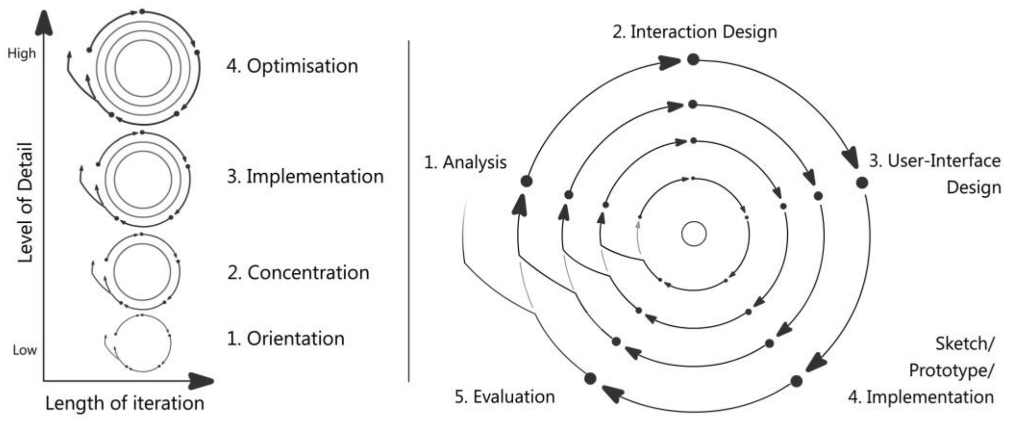


Figure 1- Left: The four steps of the development process; Right: The five phases of the iteration cycle that occur in each step.

From these thoughts, we derive the following requirements in particular for multi-touch prototyping tools: They should promote communication between team members and help document the development process. Furthermore, owing to the iterative nature of the process, these tools should enable to switch between different versions of a prototype. Since early iterations are very fast, creation of initial prototypes needs to be simple, without cumbersome complexities that may only be important in later iterations. Additionally it should be possible to concentrate on core elements of the scenario (Anderson et al. 2010, p. 243), which in the case of multi-touch applications means to focus on the *feel* & *look* instead of the other way around.

3 Evaluation Criteria

Before creating our own list of evaluation criteria we studied the approach of similar works. Grigoreanu et al. (Grigoreanu et al. 2009) analyzed needs and wants of interaction designers, concluding their work with 20 requirements of an interaction design tool as diverse as flow, usability or testing. It should be noted that flow and *feel* were ranked as the most important needs, confirming that interactivity is highly relevant in modern user interface design. Wu et al. (Wu & Graham 2002) use their model of different work styles when comparing and rating different tools and techniques typically used in the first steps of a software project. Their collaboration style emphasizes teamwork while their artifact style is more concerned with early states of the process and reusability of first concepts. Campos & Nunes (Campos & Nunes 2007) work with three categories in their tool comparison: notation – the concreteness of an artifact, collaboration - the way teams cooperate, and tool-usage – the usage of a tool itself. The compared tools were arranged in a usefulness-usability diagram to highlight the best of each category. Finally, Bergh et al. (Bergh et al. 2011) used the "GRIP Framework" to compare twelve prototyping tools. They draw special attention to aspects of prototyping that are very rudimentarily or not at all implemented in current tools, like the integration of sketches and components into the prototype or the ability to reuse parts of a prototype later on. All these papers helped us select our own categories and get an idea of the relevant aspects of prototyping tools.

3.1 List of Criteria

In the next step we assembled our own list of criteria, which we evolved from previous surveys (Campos & Nunes 2007, Wu & Graham 2002, www2). We refined the list by selecting prominent properties that were present in all the other surveys or by condensing multiple criteria into one for simplicity. We also introduced new criteria, if they seemed especially relevant to the scenario (e.g. gestures). As introduced in chapter 2, where we distinguished between the process of prototyping and the prototype itself as the product of this process, we classify the criteria into two dimensions – process and artifact.

3.1.1 Process dimension

Usability – Usage of the prototyping tool should be intuitive and direct, meaning it should be possible to work with the tool solely by using previous knowledge and to directly manipulate objects and values without having to navigate through complex menu systems or deal with different states of the user interface.

Collaboration – The ideal tool enables team members to cooperate in the same place at the same time (for example in a whiteboard scenario). Still, any feature that promotes working together, like a comment function or sharing screens, increases the collaboration score.

Ubiquity – This requirement goes beyond the precondition of multi-operating system support. Members of a team may work everywhere, every time, alone, or in groups. The aim is to let users choose their own work style instead of restricting them.

Versioning – The comparison of the different versions of prototypes is a central aspect of the prototyping process. To facilitate this process a prototyping tool should have a versioning system which easily supports the matching of *look* & *feel* over different iterations.

3.1.2 Artifact dimension

Gestures – The tool should include gesture support (e.g. pinch to zoom) as one of its main elements. Without gestures only the creation of restricted and static prototypes is possible, which are not sufficient to convey the *feel* of an application.

Interdependence – To simulate the *feel* of an application, the implementation of gestures alone is not enough. The visual elements of a user interface must be connectable to the gestures and be able to react to input by switching to other screens or states or by being directly manipulated themselves (e.g. rotation or zoom).

Behavior – It should be possible to define animations for screen transitions or single visual elements and to trigger these animations or other events using custom conditions.

GUI-Elements – Components or GUI-Elements are the basic modules of a user interface. Prototyping Tools should give the user the control to add, remove, or manipulate them. Furthermore, GUI-Element's appearance should be changeable.

Code- and structure-generation – Ideally, it should be possible to reuse parts of the prototype for the implementation, by exporting code or structure information as outlines.

3.2 Application Scenario

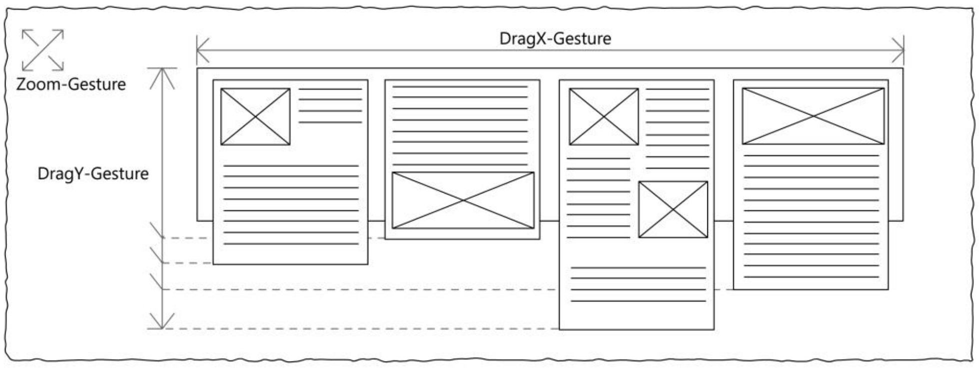


Figure 2 – Sketch of the digital newspaper scenario we used for evaluation

To rate the chosen prototyping tools (cf. chapter 4) consistently, a simple scenario of a digital newspaper application was used. The scenario is structured into two screens. The first screen contains the main news bar, which allows horizontal scrolling through a list of news blocks, each with a set of media elements (cf. Figure 2). The content of these news blocks can be scrolled vertically and a single block can be focused by using pinch to zoom. This screen also contains an additional scrollbar at the very bottom and an expandable menu with three entries: 'Info', 'Options', and 'Logout'. Tapping 'Options' takes the user to the second screen, which shows a list with all subscribed news feeds. By tapping an entry the user can subscribe or unsubscribe. After finishing the selection, users navigate back to the home screen by a separate button. This simple scenario nonetheless contains all the elements for testing the artifact criteria (especially gestures, interdependence and behavior). The process criteria were evaluated while building the scenario with the different tools as well as through further study of these tools' documentation.

4 Evaluation

Before we could start testing, using the scenario we just described, we had to choose the prototyping tools we wanted to evaluate. Initially, we assembled a list of 40 test candidates, which included not only specific prototyping software, but also tools that can be extended to serve this purpose. These included special template libraries for PowerPoint, Keynote or Visio as well as authoring environments like Blend or Flash, which might be used to create quick mockups. All 40 candidates underwent a quick pretest to gain a first glimpse into their abilities, before we selected eight tools for extensive evaluation. Our aim was to get a good representation of available prototyping tools, so we tried to take as many criteria as possible into account. We chose both very complex and very simple tools. Some of our choices are more suited for low-fidelity prototyping while others are more suited for high fidelity prototyping. Another important factor was the working environment of the tools, so we chose two classical Windows applications, two mobile applications (running on a third generation iPad) and four web-based solutions. It should be stressed that we did not pick tools based on the evaluation criteria (e.g. do they support gestures or not), since we did not want to influence the outcome in one way or the other. The tools we finally chose were *Axure* (www.axure.com), *Balsamiq* (www.balsamiq.com), *Blueprint lite* (available at itunes.apple.com), *FluidUI* (www.fluidui.com), *HotGloo* (www.hotgloo.com), *JustInMind* (www.justinmind.com), *Mockups.me* (mockups.me) and *Proto.io* (proto.io). Every tool was used to recreate our scenario and analyzed with respect to the established criteria. We conducted the study with the help of a group of colleagues, by testing the tools and discussing the results amongst ourselves. We will now summarize the outcome of our study for every criterion separately, comparing the performance of the different tools.

Usability – As would be expected, usability was mainly connected to the complexity of the tool. Simple tools that are designed for quickly creating throw-away prototypes (cf. Dix 2007, 241ff) were easy to handle. *Balsamiq*, *HotGloo*, *Mockups.me*, *FluidUI* and *Blueprint lite* fall into this category. *Proto.io*, *Axure* and *JustInMind* allow for the creation of more complex prototypes with richer interactions and more depth. Naturally this means a longer learning period before a user can effectively work with these tools.

Collaboration – Collaboration support varies heavily among the tools. *FluidUI* offers no collaboration features whatsoever and *Blueprint lite* merely allows to share the created files with other users of the app. *Axure* only facilitates cooperation through a versioning system – apart from notes when committing changes there is no possibility for communication. All the other tools allow inviting cooperators to review and comment on prototypes. Some even allow assigning roles like designer or evaluator or offer functions for testing prototypes with larger groups. *HotGloo*, *Mockups.me* and *JustInMind* allowed for actual parallel cooperation of multiple users on a single prototype at the same time, albeit with each user working from his own device.

Ubiquity – Only *Balsamiq* and *Mockups.me* are developed as HTML 5 applications, running on each device with the same *look* and *feel*. Additionally both tools are also available as offline apps, meaning they do not require an internet connection. All the other tools have the disadvantage of being either specialized for specific platforms (*Axure*, *JustInMind*, *Blueprint lite*) or being restricted to exclusive online work (*FluidUI*, *HotGloo*, *Proto.io*).

Versioning – *FluidUI* and *Blueprint lite* provide no possibilities for creating different versions of a prototype. *Proto.io*, *HotGloo* and *Mockups.me* automatically save their projects in intervals. However, since versions are only distinguishable by their timestamps, it is very tiresome to find a particular file. *Axure* actually provides a sophisticated versioning system, with capabilities to check-out and commit different parts of a prototype. *Balsamiq* and *JustInMind* have more user-friendly versioning systems. Both automatically store different states of a prototype just as the other tools. But through preview images and lists of changes, working with revisions is made much easier.

Gestures – Some tools offer no support for gestures at all (*Balsamiq*, *HotGloo*, *Mockups.me*, *Axure*), making them unsuitable for prototyping touch applications. The other tools have small libraries for basic touch gestures. All four (*FluidUI*, *Blueprint lite*, *JustInMind*, *Proto.io*) support tap, tap hold and swipes with only *JustInMind* additionally providing rotate and pinch gestures. It must be stressed, however, that these gestures are only recognized as singular events and therefore cannot be used to simulate continuous input like scrolling through a text. Only *Proto.io* and *Axure* offer solutions to this problem, by having the ability to make containers scrollable in *Proto.io* and by using mouse move events in *Axure*.

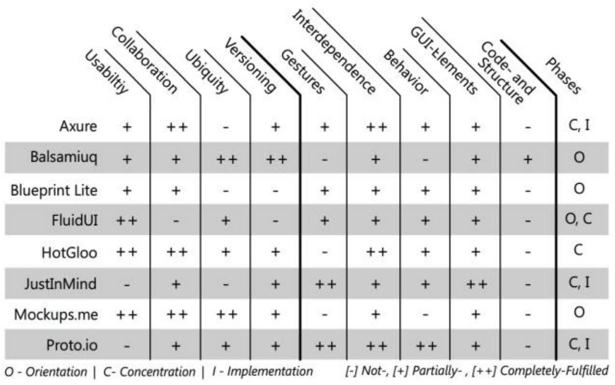
Interdependence – Half of the tools (*Balsamiq*, *FluidUI*, *Blueprint lite*, *Mockups.me*) only allow to statically link different screens of a prototype. The other tools (*Proto.io*, *Axure*, *HotGloo*, *JustInMind*) offer more possibilities for defining relationships between elements, some even enabling the user to string together multiple of these relations into chains simulating very complex interactions.

Behavior – *Balsamiq* and *Mockups.me* offer neither the ability to animate elements nor to define conditions for interactions or events. *FluidUI* and *Blueprint lite* at least offer transitional animations between the different screens of a prototype. Complex animations like moving or fading elements are supported by both *Proto.io* and *Axure*. *Axure* as well as *HotGloo* and *JustInMind* also allow for the definition of conditions that trigger animations or transitions.

GUI-Elements – Visual fidelity of elements differs between very basic wireframes (*Balsamiq*, *HotGloo*) and detailed designs emulating the styles of specific platforms (*Blueprint lite*). Most tools (*FluidUI*, *Proto.io*, *Axure*, *Mockups.me*, *JustInMind*) have extensive libraries with designs for multiple systems, letting the user decide the level of detail needed for their prototype. Customization capabilities are very different across the tools. Most allow for basic actions like changing background colors or size, while *JustInMind* enables reskinning of elements by using a custom placeholder image.

Code- and Structure-Generation – No tool was able to create code directly from the prototype. However, *Balsamiq* can export structure information in xml format for later use. Additionally the *Balsamiq* file format is supported by many other prototyping tools, further enhancing reusability. Some tools (*FluidUI*, *Proto.io*, *Axure*, *JustInMind*, *Mockups.me*) support the generation of html-Files, which might be reusable by experienced users.

Of the eight tools only *Axure* and *Proto.io* were able to completely recreate our scenario as a working prototype. All the others fell short, mostly because they were unable to provide the necessary gestures or animations required for our highly interactive requirements. Still, not even the tools fit for the scenario could convince in every aspect as our in depth analysis of the test will show. In an extended overview, we summarized the result of our tool evaluation in Figure 3 (left). We rated each criterion on a scale from zero to two: Not- (-), partially- (+), and completely-fulfilled (++). Based on this system, each prototyping tool got a rating score for each dimension. We also recommend a suitable development phase from our model of the prototyping process (cf. Figure 1, left) for every tool.



	Usabiltiy	Collaboration	Ubiquity	Versioning	Gestures	Interdependence	Behavior	GUI-Elements	Code- and Structure	Phases
Axure	+	++	-	+	+	++	+	+	-	C, I
Balsamiuq	+	+	++	++	-	+	-	+	+	O
Blueprint Lite	+	+	-	-	+	+	+	+	-	O
FluidUI	++	-	+	-	+	+	+	+	-	O, C
HotGloo	++	++	+	+	-	++	+	+	-	C
JustInMind	-	+	-	+	++	+	+	++	-	C, I
Mockups.me	++	++	++	+	-	+	-	+	-	O
Proto.io	-	+	+	+	++	++	++	+	-	C, I

O - Orientation | C- Concentration | I - Implementation *[-] Not-, [+] Partially- , [++] Completely-Fulfilled*

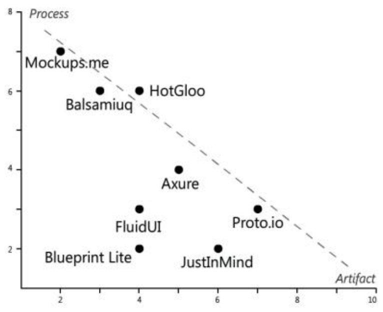


Figure 3– Left: Evaluation overview; Right: Artifact-Process diagram

Referring to the classification of tools in a usefulness-usability diagram in (Campos & Nunes 2007), we used the rated scores as coordinates for a process and artifact diagram (Figure 3, right). Here, the x-axis maps the artifact score to a maximum of 10 points, while the y-axis maps the score of the prototyping process up to 8 points. The tested tools are allocated on a straight line from the middle up to the left corner. It is apparent that the actual tools are heavily focused on some specific aspect of the prototyping process. Tools like *Balsamiq*, *HotGloo*, or *Mockups.me*, which are located in this upper corner focus more on the process itself than on the artifact. All other tools are more oriented towards the prototype and its functionality, especially *Proto.io* and *JustInMind*. However, all tools are ranked more or less in the middle or in the upper left corner of the diagram. No tool reaches a top result for the prototype or a high rating for both dimensions combined.

5 Result and conclusion

First and foremost we learned from our evaluation that in order to build good mock-ups of multi-touch applications a prototyping tool has to primarily fulfill the requirements of the artifact dimension. The process dimension itself is important in general but there are no fundamental differences between prototyping applications with classical user interfaces and prototyping natural user interface applications.

The primary aspects that contribute to the *feel* of a prototype are gestures and interdependence. However, both these aspects get complex quickly, providing a challenging task for anyone wishing to design a suitable prototyping tool. There are a high number of different gestures that need to be supported. The same is true for the various ways in which elements and gestures can be connected in modern UIs creating a maze of interdependencies. This contributes to the problem of having to find a suitable compromise between an easily usable tool and one which allows creating sufficiently complex results.

Gesture formalization languages like *GeForMT* (Kammer et al. 2010) could help increase possible gestures in such a prototyping tool, by allowing to combine gestures from a predefined set of basic movements like tapping, holding or drawing certain shapes. Add to this, the ability to define properties like finger count, length, duration or direction of a gesture and your gesture library should be extensive enough to cover most use cases. To satisfy the need for interactivity, the user interface elements have to be able to react in various ways to these gestures. Therefore, it has to be possible to freely map touch input to element properties as diverse as size, position, rotation or visibility. Defining the nature of the mapping is equally important, i.e. if input is used to continuously manipulate an object or if only complete gestures lead to state changes. In case an interaction is too complex to assemble, it should at least be possible to mimic it by using animations.

The next challenge is to fit the vast number of possible interactions we just described into a clean and usable interface and ensure that this interface does not become cluttered when a complex prototype is configured. This means finding the right balance between concentrating information on a single screen page and distributing this information between multiple pages. Being able to compartmentalize elements and interactions into groups and templates might also help to keep complexity in check.

After having identified these requirements our next goal is to integrate them into our *Liquid* framework (Freitag et al., 2011), in order to enhance its multi-touch prototyping capabilities. Due to its focus on the usage of interdependence and animation (*feel*), the use of data driven programming aspects (flow), and a simple UI-Editor (*look*), *Liquid* already performs well for the top three requirements that visual and interaction designers are looking for (Grigoreanu et al.2009). Furthermore, our tool focuses on interactivity in low-fidelity prototyping, which is an important step to bridge the gap that was shown in figure 3 (right). To further enhance the prototyping experience with *Liquid* we want to upgrade its versioning capabilities based on our own findings and the requirements of user interface designers described elsewhere (Carter & Hundhausen 2010). This means researching possibilities like displaying and editing multiple versions of a prototype at once, but also techniques for visualizing the evolution of a prototype or quickly evaluating different versions. After our expansion of *Liquid* we intend to test it thoroughly against state of the art prototyping tools to validate our results.

Reference List:

Adenauer J. & Petruschat, J. (2012). *Prototype!*. Berlin: Form+Zweck, 2012

Anderson, J., McRee, J. & Wilson, R. (2010). *Effective UI*. Sebastopol: O'Reilly (1.ed). 2010

Bergh, J., Sahni, D., Haesen, M., Luyten, K. & Coninx, K. (2011). *GRIP : Get better Results from Interactive Prototypes Expertise Centre for Digital Media*. EICS '11, New York: ACM, 143-148

Campos, P., & Nunes, N. (2007). *Towards useful and usable interaction design tools: CanonSketch. Interacting with Computers*. New York: Elsevier Science Inc., Interaction with Computers, Volume 19, 597-613.

Carter, A. & Hundhausen, C. (2010). *How is User Interface Prototyping Really Done in Practice? A Survey of User Interface Designers*. 2010, Washington: IEEE, VLHCC, 207–211.

Dix, A. (2007) *Human-comuter interaction*. London: Pearson (3.ed). 2007

Freitag, G., Kammer, D., Tränkner, M, Wacker, M., & Groh, R. (2011). *Liquid: Library for Interactive User Interface Development*. Mensch & Computer '11, Oldenbourg: Oldenbourg Verlag, 202-210.

Grigoreanu, V., Fernandez, R., Inkpen, K., & Robertson, G. (2009). *What designers want: Needs of interactive application designers*. 2009, Washington: IEEE, Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 139–146.

Kammer, D., Wojdziak, J., Keck, M., Groh, R., & Taranko, S. (2010). *Towards a formalization of multi-touch gestures*. ITS'10, New York: ACM, 49-58

Lim, Y.-K., Stolterman, E., & Tenenberg, J. (2008). *The anatomy of prototypes*. TOCHI '08, New York: ACM, , 7:1-7:27.

Nielsen, J. (2001). *Usability Engineering*. San Diego: Morgan Kaufmann,, 2001

Schrage, M. (2000). *How the World's Best Companies Simulate to Innovate*. Havard: Business School Press, 2000.

Wu, J., & Graham, T. C. N. (2002). *Modeling Style of Work as an Aid to the Design and Evaluation of Interactive Systems*. In Christophe Kolsk & J. Vanderdonckt (Eds.), CADUI '02, 217–228.

www1. http://www.uie.com/articles/four_phases_prototyping version: 03/2013

www2. http://www.uie.com/articles/prototyping_tools version: 03/2013

Contact information

Georg Freitag (HTW Dresden), Email: Freitag@htw-dresden.de