Designing Interaction Concepts, Managing Customer Expectation and Mastering Agile Development in Rich Application Product Development

Marcela Esteves and Vladimir Andrade

Siemens Corporate Research
755 College Road East, Princeton, NJ 08540, USA
{marcela.esteves,vladimir.andrade.ext}@siemens.com

Abstract. The emergence of rich application implementation frameworks, such as WPF and Silverlight, promoted a new collaboration paradigm between developers and designers where ownership of the user interface code is transferred to the user experience team. The implications of this new paradigm for the user centered design process impact its technical, collaborative, and business dimensions. The traditional design prototype can now demonstrate most of the desired user experience and could be directly integrated with the back-end code, significantly reducing the design revision costs. Creating the rich prototypes demand enhanced technical skills from visual designers, who become a member of both the design and implementation teams. The implementation tools provided by the rich application frameworks aim to simplify the prototype creation task for the designer, but can potentially lead customers to expect a lower effort associated with the user centered design process.

Keywords: user centered design process, WPF, Silverlight, visual design, interaction design, rich application.

1 Introduction

User Centered Design (UCD) describes a multidisciplinary user interface[1] development process that positions the user's goals, workflows, and requirements as the main drivers of an application's navigation, interaction, functionality, and behavior. UCD promotes early user involvement in requirements definition and user interface evaluations, which should be performed in multiple iterations until a final solution is reached. The UCD process has been advocated and practiced by multiple members of the user experience community with successful measurable results for the past decade and is supported by an array of established methodologies [1,2,3]. The international standard ISO 13407 [4], "Human Centered Design Process for Interactive Systems", identifies four activities that are part of a best practice process: specify the context of use, specify user and organizational requirements, produce design solutions, and evaluate designs against user requirements.

[1] We focus the scope of "user interface" on software interfaces only.

The context of usage analysis involves understanding the users' profiles, their goals, tasks, workflows, decision making patterns, and environments. This knowledge is consolidated in a user behavior model, which allows the user experience team members to build a solid framework that will later enable the definition of a user interface concept - including information architecture, navigation, and interaction - aligned with the users' mental model. Contextual inquiry, focus groups, personas/user profiles are examples of methodologies used in context of usage analysis.

Requirements define the functional scope of an application across multiple versions, focusing on what is needed by different stakeholders ("problem space") but not on how those needs should be addressed in the user interface ("solution space"). While the context of usage and requirements specifications are sequential activities in the process, it is not uncommon within the industry for the requirements to be defined prior to or in parallel to context of usage analysis. In these cases, the user behavior modeling provides usage context for the requirements and helps define the focus of each application version based on the users' priorities. Use Cases have proved to be a useful tool to connect user requirements with user interface concept: Use Cases contain a detailed task description, and tasks are connected to user needs which are then documented as requirements.

The design solution and design evaluation steps of the UCD process are performed sequentially over multiple iterations where early user feedback guides the user interface towards a useful, usable, and enjoyable application. Design solutions start as high level user interface concept alternatives (e.g., documented as wireframes) focusing on basic information architecture, navigation, and interaction models, and mature over multiple iterations to become the final complete solution that addresses all requirements (e.g., final design screens covering all steps for task completion). User feedback can be gathered through different validation techniques using low or high fidelity prototypes depending on the maturity level of the user interface solution.

Among UCD's strengths, its ability to provide a user interface solution that is validated by the user is of special interest for a fruitful collaboration between user experience and software development teams: when applied early in the product life-cycle, UCD process has the power to inform the implementation team and significantly decrease development effort [4]. Over the past years UCD has been combined with frequently used software development models, the V model and Agile development.

The V model process organizes the deliverables that are part of software development along product definition and product testing steps that correspond to specific validation and integration levels. Requirements, Functional, and Design Specs are sequentially created as part of product definition and serve as a roadmap for the Unit, Sub-system, and System tests that take place prior to commercialization. A smooth integration of the UCD and the V model relies on 1. the context of usage and requirements definition activities taking place prior to the Requirement Specification milestone; and on 2. the design solution and evaluations to be completed for high level user interface concept and mostly complete for detailed requirement screen design before the Design Specs milestone, when implementation starts.

Agile development [5] is a feature centric process where requirements are grouped in smaller sets that describe a user story. The implementation team aims to deliver those sets in quick iterations ("sprints") that allow the customer to validate working software early and multiple times during the development process. Research on the topic of combining UCD and Agile development [6, 7, 8] has indicated that the most successful approach is to allow sufficient time for UCD's context of usage analysis, requirements definition, and high level design solution activities to take place prior to implementation. In this approach the high level user concept design is previously defined and screen design refinements needed to address specific user story points are included in the sprint planning activity of Agile development.

During the aforementioned software development models, the prototypes that allow early validations with users are created using technology and tools that require lower implementation effort and can be confidently utilized by resources outside of the development team. Examples include Microsoft's Power Point, Axure RP, and Adobe Flash platform. The "work in progress" nature of the iterative process prevents the prototypes from being integrated into the application's code base, since early in the UCD process the user interface concept suffers dramatic changes with each iteration[2]. As a result, once the application's navigation, interaction, and behavior are stabilized, the prototype becomes a reference to the development team, who then has to re-implement the application user experience in a different code platform.

However, a new paradigm has emerged where the difference between prototype and application code is blurred, with important implications for the UCD process as discussed below.

2 Rich Applications and the Emergence of a New Software Development Paradigm

A Rich Internet Application (RIA) is an application that normally runs on a web browser, while providing similar functionality to traditional desktop software without needing to be installed into the user's computer. These are developed either using cross-browser and cross-platform technologies, such as HTML, JavaScript and CSS or using application frameworks and tools developed by third party companies, like Microsoft's Expression Blend, Adobe's Flash /Flex technologies and Sun's Java.

These technologies, when compared to regular, static web pages, enable a richer user experience by giving the software creators control over interactions, with custom and more engaging controls that improve overall user experience, system and network performance, with localized data refreshing that reduce server requests, content reloads and data transfers, while also providing opportunities to build a strong visual language.

With the advent of Web 2.0, social media and user-generated content, RIAs can be found everywhere: webmail clients, content dashboards /start pages, blogs, social networks, video sharing websites -- they are an integral part of today's internet culture. And, with the rising popularity of connected mobile devices, they are available virtually everywhere with little to no functionality loss in their mobile counterparts. Due to the broader appeal of the Internet, RIAs began setting the standard in terms of

[2] While Agile development allows for code validation over multiple iterations/sprints, it doesn't accommodate significant ongoing changes to the user interface concept due to its impact on the software architecture. Therefore, the user validations that focus on the basic user interface concept are performed prior to implementation using prototypes.

visual design, its supporting elements and how the user interacts with the software. This, in turn, increased the user's expectations towards traditional desktop applications. They expected lighter, more usable, better-looking and more interactive software, similar to what they could easily find in their web-browser.

While RIAs were wildly popular, they also facilitated the creation of a new development paradigm. While the standard model, in which a designer sends design files with specifications to the development team, is effective, it is also very time consuming: for every design file sent, rounds of validations and corrections are required to achieve acceptable implementation results. Because of the shorter turnaround times typically expected when developing for the Internet, it became important to develop ways of shortening the amount of time and effort spent on user interface implementation and validations.

Within the new paradigm, the designer is also a part of the implementation team. He is the one responsible for turning his own design files into usable front-end code. This is made possible by tools, such as Microsoft Expression Blend and Adobe Flash Professional, which facilitate the generation of said code by providing a graphical user interface that is somewhat similar to the tools traditionally used by designers.

While the front-end code, once integrated with the back-end, is an integral part of the project, it is also a completely independent entity works and run on its own as a high fidelity prototype, with static content and functionality. With regards to the UCD process, working on a high-fidelity prototype from the beginning of the project could be very time consuming. Therefore, low-effort prototyping for early user validations might be better suited for the early iterations in the project. Then, once the visual language is defined, assisted by design tools such as Adobe Photoshop and Illustrator, the designer can proceed to create a high fidelity prototype with the final design specifications. This, however, is a two-step process, as design tools are better suited for design tasks as discussed below.

Following this process enables great savings in effort and time, since the development team is freed from the task of translating design specifications into front-end, thus focusing more on the actual back-end coding. This also reduces the amount of rounds needed to refine the front-end implementation since this task is now in the hands of the designer. Now, the developers integrate, instead of implementing the front-end.

3 UCD Process and the New Software Development Paradigm

The new software development paradigm allows a more efficient collaboration between designers and developers, leading to an optimized implementation process with lower costs and decreased time to market. This optimization can only take place to its full extent if the user experience team, with special regards to visual designers, embraces the ownership not only of the user interface definition but also of its implementation. To successfully bridge UCD and the new development paradigm, the visual designer needs to acquire additional technical expertise, maintain a clear direction with regards to the application's visual language, and understand the development team's needs from an implementation perspective. Furthermore, the user experience team should educate and manage customers' expectations regarding the impact of the user new interface code activity on the standard UCD process.

3.1 Visual Designers' Skill Set

Visual designers define the application's visual language through a consistent graphic representation of all user interface elements that promotes the navigation and interaction models. In performing this systematic and creative task, visual designers are concerned with hierarchy, relationship, and weight of visual widgets to harmonically combine them in a high fidelity screen design. Designers typically use dedicated tools that provide extensive functionality to create and manipulate widgets, with Adobe Illustrator and Adobe Photoshop being among the most prominent examples.

By contrast, implementing a visual design requires a completely different mental model from defining visual design. The high fidelity screens provided by the visual designers are decomposed in connected, coded controls associated with pre-defined behaviors and states. While visual design definition is a conceptual task, visual design implementation is an execution task. In the new paradigm, the visual design implementation is owned by the user experience team, specifically the visual designer, which allows the front-end and back-end coding activities to take place in parallel and ensures the application's user interface matches the design specs.

While some rich application platforms offer dedicated tools that are designer friendly, such as Microsoft Expression Blend, those tools are not organized around a conceptual workflow but rather around an execution workflow. For instance, on design tools graphical shapes can be combined to create the visual properties of a UI element such as a menu, without the need to consider what type of control the menu may be when the screen design is implemented. Meanwhile implementation tools offer controls that have to be nested into a larger screen structure, so there has to be an a priori choice on the type of control to be used (e.g., a menu could be a grid or a stack panel) so its graphic properties can then be manipulated. Furthermore, despite the availability of implementation tools, full control over interaction behavior typically is only accessible through code writing. Therefore, the benefits brought by the new paradigm depend on the visual designer's ability to learn the implementation mental model and related technical skills.

3.2 Collaboration

The new development paradigm promotes intense collaboration between designers and developers during the implementation process. Whereas in traditional software development models designers hand over specifications to the development team and later provide input on design corrections needed on the implemented code, in the new paradigm designers deliver front-end code and are part of the development team. As designers and developers create a partnership over the same code case, the quality of the application's user interface code dramatically improves since designers make sure that the front-end implementation is fully compliant with the user experience specifications they created.

The collaboration process can be greatly improved if a developer in an "integrator" role is available to support the visual designer during implementation. The integrator not only connects the front-end code delivered by the designer with the back-end code, but also supports the designer with implementing complex and/or non-standard user interface behaviors that require heavy code writing.

3.3 Visual and Interaction Design Bias

While not a direct consequence of the new development paradigm, it is worth noting that visual designers need to be attentive to avoid bias towards the look & feel (e.g., gradients, 3D effects) of standard controls provided by the implementation tools such as Expression Blend.

The history of software development shows that visual languages established by large software vendors such as Microsoft and Apple for a given operating system release tend to influence the look & feel of applications developed during the release's life cycle. Implementation tools reflect the current look & feel and as such will likely be updated when a new operating system is launched. Because consumer oriented applications are typically enhanced through frequent releases, they can more easily adapt to changes in the industry's look & feel status quo. However, visual designers working on business to business industries should consider that the typically long life cycle of these industries' applications requires their visual language to persist through look & feel trends to avoid being outdated and alienating users.

3.4 Customer Expectation

The implementation tools available in some of the rich application platforms substantially decrease the effort needed for a visual designer to create functional front-end code. Customers of user experience teams can be led to believe that the simplification provided by the designer friendly implementation tools will cause the design solution activity of the UCD process to require less effort since elements of the user experience are promptly available. In that case the user experience team should aim to continuously educate customers about the scope of the design solution activity which entails developing a user interface concept that defines navigation, interaction, and behavior in contrast with the widget level control provided by the implementation tools. Developing applications using rich platforms and their implementation tools provide for appealing widget behavior but doesn't guarantee a useful, usable, and enjoyable application.

4 Examples of Rich Application Product Definition in Agile Development

Two previous and current product development projects of rich applications developed in Microsoft's WPF and Silverlight provided a body of experience in successful execution of UCD process in the new paradigm within the agile development model.

The first project concerned a platform development in WPF that took place between 2008 and 2009, included approximately 2,500 market requirements for the first product release, and involved 10 development teams distributed in 5 locations over 3 continents. The UCD process was followed from project start and the basic user interface concept containing navigation, interaction, and behavior definition was completed within the first 8 months of the project. During that period the platform's visual language was also defined. Each application provided by the platform was divided into a module that was implemented during one or more sprints. While the basic user interface concept and visual language was defined, the design solution of each module had to be refined to account for interaction model to support detailed requirements and the visual design of specific UI elements. Those two activities were performed for each module as part of the project's sprint schedule.

Figure 1 shows how the interaction and visual design refinement for each module was included in the sprint schedule. The interaction and visual design refinement tasks for the first module to be implemented ("Module A") and the interaction refinement for the second module ("Module B") took place before sprint 1. During sprint 1 and subsequent sprints, front-end code, interaction refinement, and visual design refinement happened in parallel for different modules, to ensure that when the sprint for the implementation of a given module started, this module's interaction and visual design were finalized and approved.

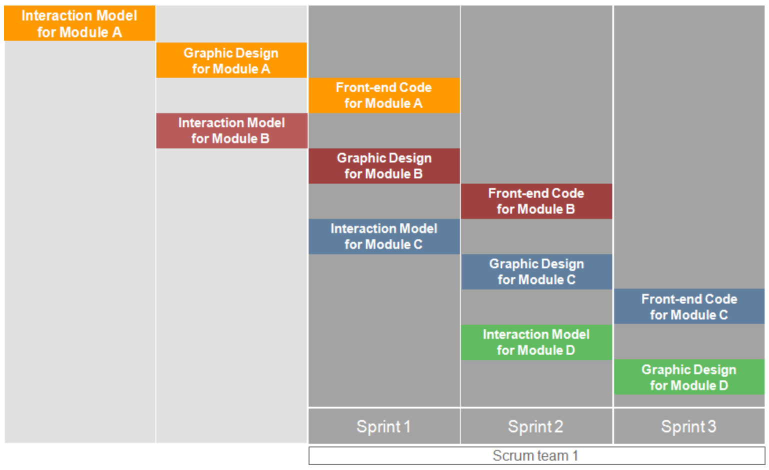
Interaction Model for Module A
Graphic Design for Module A
Front-end Code for Module A
Interaction Model for Module B
Graphic Design for Module B
Front-end Code for Module B
Interaction Model for Module C
Graphic Design for Module C
Front-end Code for Module C
Interaction Model for Module D
Graphic Design for Module D
Sprint 1
Sprint 2
Sprint 3
Scrum team 1

Fig. 1. Sprint Schedule accounting for UCD process activities and front-end code delivery by user experience team

During that period the Expression Blend tool provided by the WPF platform was in its infancy (e.g., during the project duration the visual designers worked with Blend's beta version and release candidates). Given the high complexity of the software architecture, the fact that both user experience and development teams had to learn the new WPF platform, and the state of the Blend tool, the collaboration paradigm chosen was as follows: during the sprint the development team would deliver to the user experience team the module's functional code, and the visual designer's main role was to style it in the Blend tool/code with continuous support from the scrum team's integrator. In this project the visual designer didn't create the front-end code from scratch.

The second project is a product development based on an existing platform that is currently coded in WinForms. This project is developed in Silverlight, started on May

2010 with first product release by end of 2011, and involves 5 development teams distributed across 2 continents. The UCD process was also followed from project start with the design solution activity completed prior to sprint implementation. In the course of 6 months the user interface concept, interaction refinements to address all requirements, and visual design definition were completed in an iterative process involving user validations. The product is also divided in modules which are developed over one or more sprints.

The visual designer's role within development is to create the front-end code from scratch for each module and to create a Resources Dictionary (library of UI components' code supported by the xaml programming language that is behind the WPF and Silverlight platform) for the entire application. Because this project's software architecture complexity is more easily managed and because the Silverlight platform and Blend tool have greatly improved over the past couple of years, the visual designer is able to create a module's front-end code in Blend and deliver it to the development team who then integrates the module's front-end and back-end. In addition the visual designer also saves all UI components (called "resources" within the Blend tool, primarily composed of styles, templates, and brushes) into the Resource Library which is used to manage the look & feel of all modules.

Because of the aforementioned difference between visual design definition being a conceptual task in contrast with visual design implementation being an execution task, the visual designer first created the module's screen designs in Adobe Photoshop (which provides better control over visual properties than Adobe Illustrator) and then recreated the module in Blend to generate the front-end code. In this process the visual designer is supported by a senior developer who troubleshoots resolves front-end code behavior needed for given modules.

5 Conclusion

The new software development paradigm is a major step forward towards resolving the previously discussed inefficiencies caused by the hand-over of the user experience that happens between designers and developers in the traditional collaboration paradigm. The main challenge for user experience practitioners, with special regards to visual designers, is learning a new, complex skill set that typically lies beyond the interest of visual design domain experts.

Furthermore, most of the rich application platforms such as Adobe Flash learned by students during their academic interaction and visual design courses are not typically associated with product development. This creates a situation where visual designers hired by companied that build desktop applications are typically not familiar with rich application platforms that are beneficial for product development, such as WPF and Silverlight. This causes the ramp-up phase and learning curve of visual designers to be time and effort consuming when they are first exposed to those platforms.

The currently available designer friendly development tools are widget centric and tend to require that visual designers first approach the front-end implementation and then define the visual design. However, the definition process often involves experimentation with different approaches to the UI elements that may require revisions to the interaction concept and to the types of controls used.

Ultimately, the design definition and implementation processes are different in nature and will likely continue to be performed as sequential steps, in the same way that an architecture plan is defined prior to construction. The ideal situation for visual designers would be a tool that allows the screen designs that emerge from the conceptual design definition task to be easily converted into front-end code. This approach would eliminate the need from visual designers to either perform the design task in a development tool or to recreate the design into a software development tool.

References

1. Beyer, H., Holtzblatt, K.: Contextual Design: Defining Customer-Centered Systems. Morgan Kaufmann, San Francisco (1997)
2. Cooper, A., Reimann, R., Cronin, D.: About Face 3: The Essentials of Interaction Design. Wiley, Hoboken (2007)
3. Rubin, J., Chisnell, D., Spool, J.: Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests. Wiley, Hoboken (2008)
4. International Organization for Standardization, http://www.iso.org
5. Martin, R.: Agile Software Development, Principles, Patterns, and Practices. Prentice Hall, Upper Saddle River (2002)
6. Fox, D., Sillito, J., Maurer, F.: Agile Methods and User-Centered Design: How These Two Methodologies Are Being Successfully Integrated in Industry. In: Agile 2008 Conference (2008)
7. Labib, C., Hasanein, E., Hegazy, O.: Early Development of Graphical User Interface (GUI) in Agile Methodologies. J. Comp. Met. Sci. Eng. 9, 239–249 (2009)
8. Patton, J.: Designing Requirements: Incorporating User-Centered Design into an Agile SW Development Process. In: XP/Agile Universe 2002. Springer, Heidelberg (2002)