敏捷你的用户界面开发

作者 Dave Churchville译者 胡键 发布于 2007年5月14日 上午3时9分

社区 Agile 主题 质量交付 客户及需求

Two important disciplines in the area of software development have emerged over the last decade - Agile software development and user-centered design (UCD, also referred to as interaction design). These approaches are lauded by supporters as critical to end user and customer satisfaction with software but are seemingly at odds on some key points.

Agile software development processes have many variants, but the common themes include close communication with customer, rapid iterations, and a focus on delivering value as quickly as possible. Agile processes frown on long up-front design and analysis phases, preferring instead to mix analysis, design, development, and even delivery throughout the project.

Interaction design/UCD are a set of practices that are designed to uncover user goals in detail in order to create experiences that efficiently, effectively, and enjoyably guide the user through software to meet these goals. UCD practitioners spend as much time as is needed up front researching user behavior and goals, and developing interaction designs to support them.

This article will explore how these disciplines can peacefully co-exist. By considering ways to integrate UCD techniques into an Agile project lifecycle, development teams can get the benefits of both approaches. Understanding how best to satisfy user goals while simultaneously focusing on delivering frequent and continuous business value is a combination worth trying.

Background

背景

Complex user interface development (referred to as UI development in this article) presents some unique challenges for an Agile development effort. Unlike projects where most of the functionality is behind the scenes in the form of automated business rules or payroll calculations, UI development projects often cannot afford to make drastic changes due to issues with user training and productivity loss, especially for applications that are used constantly (such as a call center application, email reader, or data entry system).

复杂用户界面开发（本文用作UI开发），为敏捷开发提出了某些独特的挑战。与那些功能大部分是自动运行的业务规则或工资计算的项目不同，因为由用户培训或生产力缺失引

起的各种问题，UI开发项目往往不能承受剧烈的变动，尤其是那些经常被使用的应用程序（如呼叫中心、E-mail阅读器或数据登记系统等）。

Many Agile projects rely on the ability to "just fix it in the next iteration" as a hedge against making the wrong decision. The idea is to fail quickly while it is still relatively inexpensive. But end users of UI development projects often need substantial training to use the systems, and are thus much less tolerant of dramatic changes in the user experience.

许多敏捷项目通过“下次迭代再修改”的方法防止做出错误的决定。它的思想是在代价相对低廉时快速地失败。但是，UI开发项目的最终用户需要经常进行关于系统使用的充分培训，因此就用户体验来说，不能容忍巨大的改变。

User experience disciplines such as interaction design promote a more up-front design process that researches and captures user goals to attempt to optimize software workflow before development begins. This is somewhat in conflict with the Agile approach - this upfront analysis and design can often take several months for a non-trivial project, and user goals may shift as the project unfolds.

用户体验原则，如交互式设计，鼓励更多的预先设计过程来研究和捕捉用户目标，试图在开发之前最优化软件工作流。这有点与敏捷方法相冲突——对于重要的（non-trivial）项目预先分析和设计往往需要花费几个月的时间，而且随着项目的开展用户的目标可能会转变。

But with restrictions on the degree of UI changes, and the high risk of delivering an inappropriate user interface in an early iteration, what's an Agile developer to do?

但是面对用户界面改变程度的限制，以及早期迭代交付不合适用户界面的高风险，敏捷开发者打算做什么呢？

An Agile Approach to UI Design

UI 设计的敏捷方法

An approach to this problem requires a quick check of Agile foundations. Many of the practices of Agile development are geared towards reducing risk and lowering the cost of change. In the case of UI development, the cost of incremental change and deployment is higher than we'd like because of concerns about end-user retraining and loss of productivity. We also run the risk of delivering the wrong user interface initially, but being limited in how much we can change it.

这个问题的解决办法需要快速地检查一下敏捷基础。很多敏捷开发的实践是朝减小风险和降低变更成本的方向去调整。在UI开发时，因为考虑到最终用户的再培训和生产率的降低，增量改变和部署的成本比我们期望的高。我们同样也会冒初始交付错误用户界面的风险，

但是这受限于我们可以在多大程度上改变它。

With this in mind, an approach that lowers the initial risk of delivering a UI that is "way off" will give us a good start in the right direction. If we get the UI right initially, then the types of changes we make in subsequent iterations won't be nearly as traumatic for our users, and will have lower deployment and training costs. By understanding the most important and common user goals and making the initial interface support these goals effectively in terms of efficient navigation, appropriate feedback, and a minimum of distraction, we stand the best chance of getting the design right. User goals tend to be more stable over the life of a project, unlike more detailed requirements.

注意到这些，降低初始交付“离题”用户界面风险的方法，将会在正确的方向上给我们开个好头。如果我们一开始就得到正确的用户界面，那么在随后迭代中我们所做的改变，对用户几乎不会造成伤害，兼具更低的部署和培训成本。通过理解最重要和公共的用户目标，完成支持这些目标的初始界面并根据它有效地导航、适当地反馈、最小化偏差，我们将有获得正确设计的最佳机会。与更详细的需求不同，随着项目的进展，用户目标会倾向于更加稳定。

To stay within the spirit of Agile, though, we ideally need this approach to allow us to get rapid feedback from our customers, and to quickly iterate our user interface to a point where we have confidence that our initial development effort will deliver what our customers need.

保持敏捷的精神，理想上来说，我们需要这个方法让我们可以快速地获得用户反馈，快速地迭代用户界面，指出在哪处我们可以放心。这样，我们的初始开发成果将交付用户所需的东西。

Agile Interaction Design

敏捷交互设计

Interaction design is a discipline that looks to create optimal user experiences by studying the domain and environment of the target users of the software. Traditionally, this is done with an up-front research and analysis of user roles and goals, with an eye towards identifying the primary users of the system, what they need to do, and what secondary factors come into play (such as repetitive use, sensitivity to errors, etc.)

交互设计是通过研究软件目标用户的领域和环境，关注于创造最优用户体验的方法。传统上这是由预先研究和分析用户角色和目标完成的，通过辨识系统的主要用户，他们需要什么，接下来考虑次要因素（如重复使用，对错误的敏感性等等）。

A more agile spin of interaction design would be to do "just enough" study of user goals to make a good first guess at what might work, then test it with real users. While not as thorough, this would allow quick turnaround and validation early in the process, instead of a longer up-front analysis.

更敏捷的交互设计迭代自旋将会是：先对用户目标进行“刚好足够”的研究，然后猜测软件将如何工作，最后用真实的用户去测试它。尽管不彻底，它允许快速地转变和在过程早期进行验证，而不是更长的预先分析。

In short, there is significant value in doing at least some up front work in order to get a better understanding of user goals, as long as it doesn't prevent development from starting as soon as possible. Depending on the makeup and philosophy of the team, this may take one of the following forms:

简而言之，只要不阻止开发尽可能早的开始，为更好的理解用户目标而进行一些起码的预先设计还是有显著价值的。根据团队的组成和哲学，可以采用以下形式之一：

- UCD work is performed an iteration ahead of development work. Designers work closely with users and developers to do "just enough" analysis to provide solid guidance. Feedback from end users and customers will play an even more critical role than in projects where most UCD work is done up front.
- 在开发工作之前，UCD 工作被执行一个迭代。设计人员与用户和开发人员一起紧密工作，进行“刚刚足够”的分析，以提供扎实的引导。在项目中，来自最终用户和用户的反馈将扮演比绝大多数预先 UCD 工作更重要的角色。
- UCD work is performed in conjunction with development. Similar to above, but designers work in parallel with developers, perhaps starting with an initial user research effort at the beginning of the project. Designers constantly inform the team of customer feedback and changes to the design. This actually happens frequently on teams with no formal UCD staff. A potential tradeoff is that new discoveries of ways to streamline software workflow are likely to be found in mid-iteration, and may require rework of major elements of the system, slowing down overall team velocity.
- UCD 工作与开发结合进行。与上类似，但是设计者与开发者并行工作，可能以项目开始时的初始用户研究成果作为开始。设计者不断地通知团队用户反馈，并改变设计。 这实际经常发生在没有正式 UCD 人员的团队中。一个潜在的折中是，在迭代的中期发现新的流水线软件工作流方式，可能要求返工系统的主要元素，降低团队整体速度。

Personas and Goals

角色和目标

A quick technique for discovering user goals is to use "personas" - specific user profiles of real people who might need to use the system, what their tasks are, and even their emotional outlook. For example, for an email system, we might have a persona that looks like this:

一种发现用户目标的快速技术是使用“角色”——即可能使用系统的真实人群的大体概

要，他们的任务是什么，甚至他们的感情观。例如，就某个邮件系统来说，我们可能有一个这样的角色：

- John is a busy executive at a Fortune 500 company. He is not very comfortable with computers, but uses email as his primary application. John gets about 100 emails a day, and usually sends less than 10. He does not enjoy using software, and prefers to call his associates in person.
- John 是一位就职于某财富 500 强公司的忙碌执行官。他不十分擅长电脑，但使用电子邮件作为他的主要应用程序。John 每天大约收到 100 封邮件，通常发送不到 10 封。他不喜欢使用软件，更愿意亲自给他的同事打电话。

A full persona description might go into even more detail, but the idea is to use not a "stereotype" of a user, but a specific instance of a user with certain goals and traits that is representative of your expected end user. Depending on the type of system, you might need several personas to represent the different kinds of users.

一个完整的角色描述可能会更详细，但是它是为了使用而非为了给用户“定型”，角色是一个有某种目标和特征的用户的特定实例，作为你所期望的最终用户的代表。根据系统类型，你可能需要几个角色代表不同种类的用户。

Once you've defined personas, you can start to analyze the kinds of goals that they have. A goal is really more about outcomes that they want to occur, not tasks that they perform. An example of goals versus tasks:

一旦你定义了角色，你就可以开始分析角色的目标种类。目标实际更接近于他们期望出现的产出，而不是他们要执行的任务。目标与任务比较的例子：

- Goal: I want to stay up to date with my colleagues and keep track of past conversations
- Task: I need to check my Email Inbox and view past conversations by Contact.
- 目标：我想与我的同事通宵达旦并能追踪过去的谈话。
- 任务：我需要检查我的收件箱，通过联络查看过去的谈话。

Eventually you'll need to come up with specific tasks that help your user reach her goals based on her preferences and needs. By starting with goals, though, you can refer to them as you consider different technical possibilities for achieving them. Just because a user performs certain tasks today doesn't mean that those are the easiest or most effective way of meeting their goals.

最终，你需要提出明确的任务来帮助你的用户，以他的偏好和需要，达到他的目标。通过由目标开始，你可以将它们作为参考，考虑不同的技术可行性。仅仅因为今天用户执行某种任务，并不意味着那些是达到他们目标最简单或最有效的方式。

For further reading on personas, see About Face 2.0 - The Essentials of Interaction Design by Alan Cooper and Robert Reimann.

关于角色的更详细说明，可以参考Alan Cooper和Robert Reimann 的著作About

Face 2.0 - 交互设计精髓。

Paper Prototyping

书面原型

One of the tools we can use for quick exploration of interface ideas is the use of paper prototyping. This is a technique of creating drawn or printed mockups of a user interface and testing them with users. The testing is done by having a facilitator act as the "computer", and having her add other pieces of paper demonstrating the functionality in response to a user "clicking" on different areas of the paper.

我们可以用来快速探索界面想法的工具之一是书面原型法。它创建一组画出或打印出来的伪用户界面，然后与用户一起验证它们。测试通过由一个协调人来扮演“电脑”，当用户“点击”纸上的不同区域时，作为回应，会加入其它纸上图示来示范功能。

An entire book has been devoted to this subject - Paper Prototyping by Carol Snyder, and is an excellent reference for the techniques and tradeoffs of this approach.

有一本整本专门讨论这个主题的书籍——书面原型法，作者是Carol Snyder，它是这项技术及其权衡利弊的优秀参考书籍。

The primary advantages of paper prototypes are that:

书面原型的主要优点：

- They are easy to create - anyone with paper and pencil can do it
- They often uncover interaction issues that more formal prototypes do not, since users are less distracted by look and feel issues
- They are clearly throw-away: you can't put a piece of paper "into production", regardless of how much pressure is applied.
- 它们很容易创建——任何人只要纸和铅笔就可以做。
- 它们经常能发现更加正式的原型所不能发现的交互问题，因为用户可以更少的被观感问题分心。
- 它们显然是要被抛弃的：你不能把一页纸作为产品，不管压力多大。

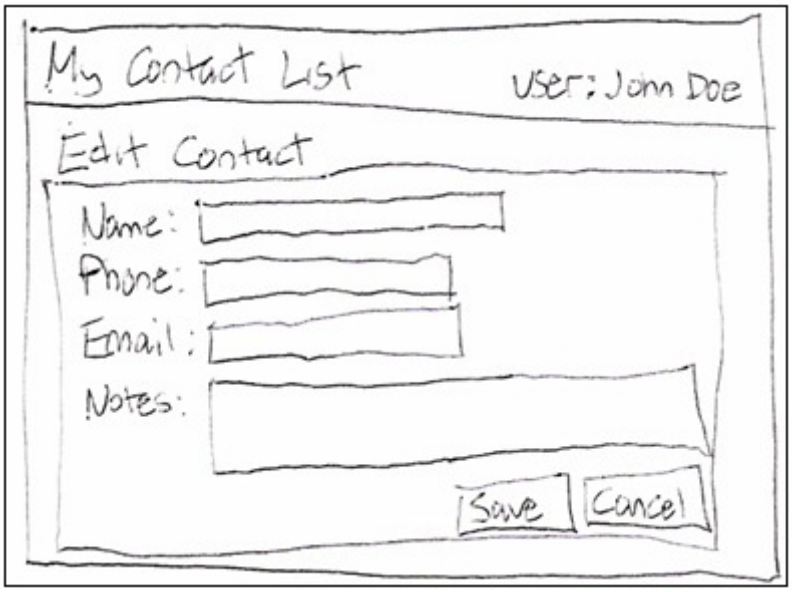

图 1 为使用而画的书面原型

Other Prototyping Techniques

其他原型技术

As beneficial as paper prototyping can be for quick feedback, sometimes it isn't practical. For example, if you have remote customers, facilitating a paper prototyping session can be difficult if not impossible.

尽管书面原型法的好处是可被用于快速反馈，但有时它不太实用。如，如果你有远程客户，书面原型会话即使不是不可能，那也会变得困难。

One alternative technique is to take digital photos or scans of paper or whiteboard drawings, and to link these into interactive mockups using an HTML editor, presentation software, or specialized tools. This approach has the advantage of still using paper to model concepts, but allows remote customers to be sent copies, or review them interactively online with a conference call or webinar.

一种可替换的技术是使用HTML编辑器、表现层软件或专用工具，将数码照片、扫描纸张、白板画链接这些进入一个交互式的模型。这种方法仍然具有使用纸张建模的优点，但允许发送拷贝给远程用户，或使用电话会议或webinar在线交互式地审查它们。

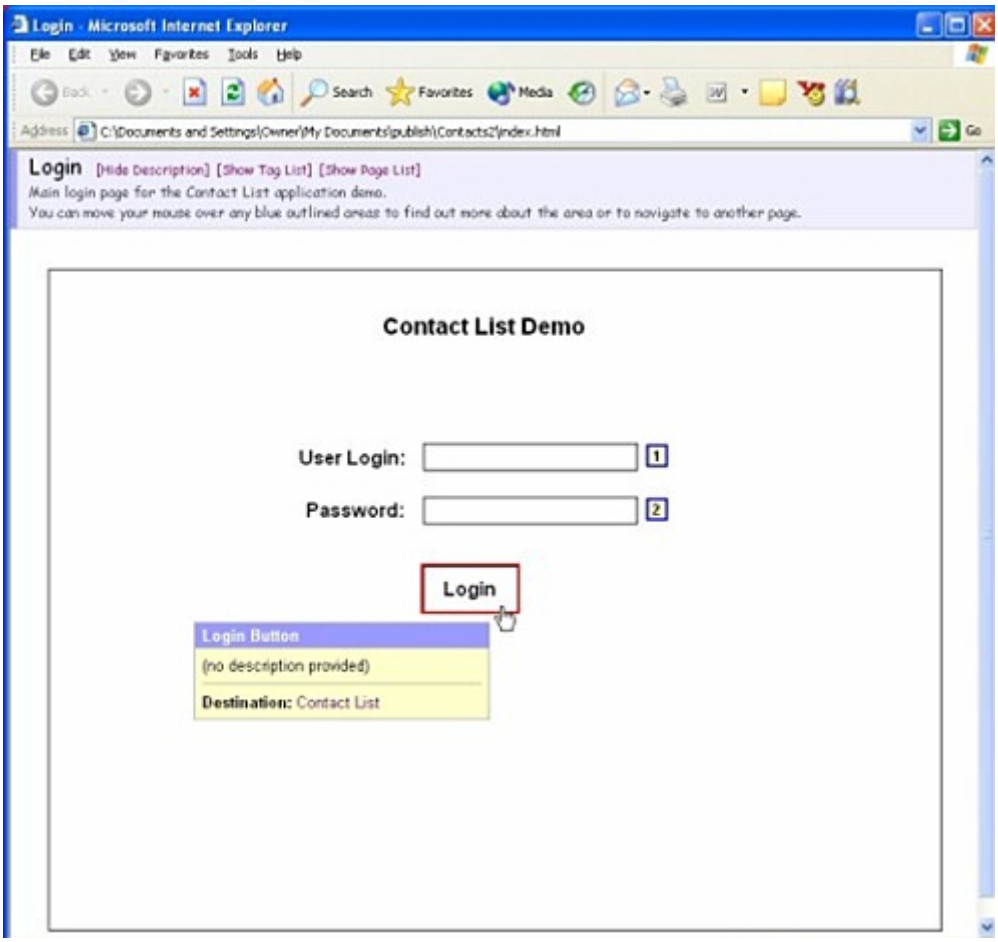

图 2 一个简单的 html 原型

Another option is to create a bare-bones HTML page for each screen of your prototype that just has text descriptions of the purpose of the screen, with links for each action needed on that screen. Web designers call this process wireframing.

另一个选择是为每个原型屏幕创建主干HTML页面，只有描述它们目的的文字，以及屏幕上需要动作的链接。Web设计者称这个方法是“接线框”。

As a last resort, you could create a prototype in a tool like Visual Basic or your favorite development language IDE. Many of these products have rapid prototyping features for creating dialogs and screens without programming. The main drawback, of course, is that these prototypes can look "finished" or very close to the target application technology. Because of the realistic look, end users may tend to focus on the details of the implementation instead of the workflow ("Shouldn't that button be left-aligned? It looks pretty ugly").

作为最后一招，你还可以使用如VB或你喜爱开发语言的IDE创建一个原型。很多这样的产品都具有快速原型特性，创建对话框和屏幕而无需任何编程。当然，主要缺点是这些原型看上去就像是“成品”或与目标应用程序技术非常接近。正因为看上去真实，最终用户可能倾向于讨论实现细节，而非工作流（比如“这个按钮难道不能是左对齐的吗？它看上去真的很难看”）。

"Spike" Implementations

“探究”实现

For certain projects, you may not be able to communicate a sophisticated interface technique or widget by simply drawing it. In these cases, it can be beneficial to create a "spike" implementation. A "spike" is a narrow slice of functionality that goes end to end, like driving a spike into the ground.

就某些项目而言，你可能没法通过简单地作图来交流复杂的界面或组件。此时，创建一个“探究”实现会很有用。“探究”是功能的狭长切片，端到端的，类似将长钉敲入地面。

For example, if you're building a new kind of calendar widget or fancy animated toolbar, it might be necessary to spend a few days creating something close to the real thing so that you can demonstrate the subtle behavior to get feedback.

例如，如果你在构建新类型的日历组件或梦幻动画工具条，那么非常有必要花费几天创建一些与真实事物相近的事物，这样你可以示范精巧的行为以获得反馈。

Summary

总结

In general, Agile techniques work well for both back-end systems and non-trivial user interfaces. However, there are many useful techniques from other disciplines that can add significant value. Specifically, techniques from interaction design can help with designing and testing user interface ideas before investing time in coding, and certainly before those systems are put into production, where interface changes may be constrained by training and productivity concerns:

通常，敏捷技术在后台系统和非琐碎用户界面时工作非常好。但是还是有很多来自其它方法的有用技术可以带来显著的价值。尤其是来自交互设计的技术，在为编码而投入时间之前可以帮助设计和测试用户界面想法。这当然也是在这些系统产品化之前，产品化后界面变化可能受培训和生产力因素制约

- By using **personas and goals**, developers can get an early feel for what kinds of tradeoffs might be necessary when designing the software.
- 通过使用角色和目标，开发者可以得到早期的感觉，在设计软件时可以进行权衡。
- **Prototyping techniques** ranging from paper prototyping to interactive electronic prototypes can provide a means of rapidly iterating and testing user interfaces at a low cost.
- 原型技术，由书面原型到交互式电子原型可以提供快速迭代和低成本测试用户界面

的方法。

- Finally, **spike solutions** can be helpful when the user interface needs of a project are more subtle or complex than simple prototypes can communicate. By implementing a small slice of functionality that demonstrates a key user interface concept, a spike can ensure clear communication with customers.
- 最后，探究解决方案在项目用户界面需要比简单原型进行更精细或复杂的交流时非常有用。通过实现功能的小切片来示范关键用户界面概念，一个探究可以确保与用户清晰的交流。

Combining user experience design techniques with an Agile development process can create a powerful synergy that results in better software that end users love, while still delivering the consistent business value for which Agile is known.

结合用户体验设计技术与敏捷开发过程可以创建强大的优势互补，这将产生更好的受最终用户喜爱的软件，同时交付业已为人称道的敏捷商业价值。

查看英文原文：Agile User Interface Development

About the Author

关于作者

Dave Churchville has over 15 years of experience in software development and management ranging from Fortune 500 companies to tiny startups. He has worked with agile methods since 2000, and is the founder of ExtremePlanner Software which develops tools that help distributed Agile software teams communicate and collaborate more effectively.

Dave Churchville拥有15年软件开发和管理经验，包括从财富500强公司到小的项目。他从2000年开始使用敏捷方法进行工作，是ExtremePlanner 软件公司的创始人。该公司开发帮助分布式敏捷软件团队更有效地进行沟通和协作的开发工具。

Dave writes about Agile development topics at

http://www.extremeplanner.com/blog.

Dave在http://www.extremeplanner.com/blog撰写关于敏捷开发的主题。