Editor: Charles Petrie • petrie@stanford.edu

From Palaces to Yurts

Why Requirements Engineering Needs Design Thinking

Christophe Vetterli, Walter Brenner, Falk Uebernickel, and Charles Petrie • *University of St.Gallen*

The German saying "von Palästen zu Zelten" compares different systems to different levels of flexibility and agility — that is, "from palaces to yurts." Requirements engineering systems are geared for developing information system palaces and aren't what's needed for today's world of rapidly changing, app-enabled products. These Web and mobile apps are small, require rapid development, must closely fit customer needs, and change often. Requirements engineering for these would greatly benefit from *design thinking* — that is, a human-centered, rapid-prototyping method for innovative design.

All house construction requires a solid basement, a supporting infrastructure that provides efficiency in maintenance, and some adjustable elements that will be continuously updated for the house's lifetime. Large, complex houses provide more comfortable living space, but more groundwork is needed if any changes are necessary. IT systems are similar. To meet today's challenges with small, easily changed systems that are more function than infrastructure, we need more that are like yurts rather than palaces.

Evolving Apps

In the first phase of Internet application development for products and services, such applications used the Web to provide a front end to simple functions, such as looking up stock quotes or current weather. In the second phase, the Web acted as a front end to large, integrated back-end systems. These systems require the typical requirements engineering approach — long, careful study and development. However, the new generation of apps is loosely bound to back-end systems, if any, and employs algorithms that can easily run on mobile devices as well as the Web.

One example is the Azumio Stress Tester, which uses a sophisticated algorithm to measure variations in pulse to determine stress or conditioning (see https://play.google.com/store/apps/details?id=com.azumio.android.stresscheck&hl=en). The PeakFinder uses GPS and compass data to determine a person's position and his or her relation to mountains (www.peakfinder.org). Such apps also connect to back-end systems on the Web, but only loosely, and they can run without a connection.

Although large back-end systems will continue to be needed, an emerging trend is that of app-enabled products. Increasingly, many products, both software and tangible, are released and accompanied by Web or mobile apps that add value. Even taxis benefit from today's apps, which we can use to look for parking spaces or share cars. One example originates from Nobel Biocare, a dental solutions company: OsseoCare Pro is a tablet-based app that lets a dentist control his or her drill motor and work with the patient to plan and set up the treatment sequence prior to surgery; it also enables multiple user log-ins for sharing treatment data between different clinical partners (see www.nobelbiocare.com/en/campaigns/osseocare/default.aspx). Even these complex apps are small and run on small mobile devices independently of larger systems such as databases that might be sporadically reachable on the Web for updating and sharing.

We can expect apps to become more integrated with future products. Imagine, for example, drones that make small deliveries, homing in on the smartphone requesting them. These apps would be small programs, often updated via cellular marketplaces, that provide limited functionality and connect to larger systems on the Web asynchronously, perhaps connecting to other users' similar apps while adding value to mobile devices and tangible products. The apps might exchange data over various channels and push data as well as make connections. These apps won't be at all like the big back-end systems that current requirements engineering supports.

Requirements Engineering Approaches

The primary measure of an information system's success is the degree to which it meets its original purpose. We can define requirements engineering as the process of initially discovering and defining that purpose.[1] As Pamela Zave states,

"Requirements engineering is the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behaviour, and to their evolution over time and across software families."[2]

Thus, we can view requirements engineering as inherently difficult. Betty Chen and Joanne Atlee state that requirement analysts start with ill-defined, and often conflicting, ideas.[3] By simplifying this problem space, we can constrain the environmental conditions in which the system or applications should operate. The requirements engineering procedure is more iterative and involves many more players with different backgrounds than other software engineering activities. Besides this complexity, requirements engineering needs more extensive analyses of options and must call for more complex verifications of more diverse components, such as technological, human, legal, and cultural. In the app context, which changes rapidly, the challenge will be to redefine this process.

We've observed many global companies educating their developers to devote all their efforts toward those aspects of software development that are intended to last for eternity, such as achieving the highest possible security capability and being available 24/7. Such back-end systems are based on big data models, have a long-lasting life cycle, and assume that users are technical. The goal is to develop a system as complete as possible and integrate all possible functions to kill two (or more) birds with one stone. The result is something like a palace, built on a strong foundation with a large fixed infrastructure where everything works together and would be difficult to change.

The neighborhood has changed, however, and the concrete and crane that were used to build palaces are no longer needed to build the mobile and agile community of apps that are more like yurts. It isn't that some large back-end systems aren't needed or that they won't connect to apps, but rather that app development isn't supported by the requirements engineering process used to develop these large systems.

Look at your own experience in downloading an app from any smartphone app marketplace. It installs within minutes, its focus serves exactly what you were looking for, and, if not, you download another one. Moreover, you can set the app for automatic updating, and probably will, given that many apps are updated frequently. App users require speed, frequent change, convenience, and limited functionality. The game has changed, and the rules are different. Apps are small, stand alone with few intertwining functionalities, and run quickly on small computers. These changing demands are critical for business. If companies don't catch up with the new app environment, their back-end software house will be a lonely palace standing somewhere hundreds of miles away from the next palace with hardly any connection to users.

Today, users expect a wide selection of apps that they can integrate into their daily lives and behavior. Developing such apps requires flexibility, agility, and strong customer orientation. Companies now face the challenge of producing app-enabled products — such as OsseoCare Pro or PeakFinder — that have a few integrated functions that are highly relevant to the user's life.

The problem the software engineering community has been trying to solve from its beginning is how to go from the problem space (user requirements) to the solution space (design and implementation) with a methodological guidance. Requirements engineering processes usually include following steps — elicitation, analysis and negotiation, specification, and validation — as a standard way to solve this problem.

The IS community has already recognized that for a changing world and fast development — which apps take to an extreme — this approach isn't sufficient, resulting in so-called *agile development* approaches. These alternative processes certainly have advantages, but they tend to throw out the baby with the bathwater, especially for apps that need to connect to back-end systems.

Agile development tends to focus on code traceability rather than the documentation characteristics of large system development. It involves the customer in interactive prototypes throughout the development process, whereas requirements engineering tends to drop customer involvement after initial elicitation. Agile development is driven by customer descriptions of what they require, but captures these from a functional requirements perspective only. Even with a strong customer orientation and good developers, the distinction between functional and nonfunctional requirements is difficult to catch and needs other perspectives. For apps that connect to back-end systems, combining the two approaches is especially crucial because the same developers working on the palace of software comprising the company's operations are often the ones assigned to develop the mobile app yurts. So, the question is how to improve requirements engineering to incorporate agile development's useful features in a way that supports app development, perhaps in concert with large system development. We need a method that combines the best of both approaches.[4]

Design Thinking

Design thinking provides a methodology for eliciting customer needs, rather than requirements, and producing a series of fast and simple prototypes that eventually converge on innovative solutions. Researchers at Stanford University have been studying, testing, and modifying this methodology in product design for the past 40 years. The methodology has been abstracted and has spread to other universities, such as Aalto, Potsdam, and St. Gallen. It's been incorporated into practices at large companies such as Deutsche Bank, Proctor and Gamble, and SAP. Design thinking is consistent with the initial elicitation practices of requirements engineering and the rapid prototyping and customer involvement of agile development methods. It offers a consistent methodology for doing both as well as documentation, consistent with requirements engineering, and team management, a focus of agile development.

Design thinking emphasizes the human perspective. We apply this human-centered innovation method to ill-defined problems within a real-world context, which is characteristic of apps for mobile phone users. Creating desirability for potential customers drives design thinking activities and captures potential customers' needs. Unlike requirements engineering, design thinking aims to fail early in order to succeed sooner. This learning process doesn't focus on searching for requirements specifications even in terms of agile methods. Rather, it involves quickly learning from early errors how best to articulate and solve human needs.

Starting with quick, low-resolution prototypes helps design teams diverge within the design space to avoid settling on solutions that might only be local maxima in the solution space and might not actually meet human needs. Design thinking moves from such intensive learning phases toward higher-resolution prototypes that converge on novel solutions.

Such prototypes help concretize different ideas without simplifying the environment, while focusing on specific and important needs within the design space. Although agile development and requirements engineering use prototypes as well, these mainly help converge and eliminate technical inconsistencies as fast as possible early in the process. Although it involves the customer throughout the process, agile development has no methodology for eliciting needs that might be other than the stated requirements and, again, tends to focus on code consistency and traceability. Design thinking offers an additional elicitation methodology.

Design thinking is also about changing the involved parties' mindsets — that is, keeping ambiguity high during the projects' early stages until developers are certain of identified needs and desires. Thus people are needed that can handle this ambiguity and have empathy for their potential customers. This requires an environment that supports a collaborative, engaging working style with customers as part of the team. At Stanford, this team makeup aspect is already being employed in design thinking research, with a working environment often characterized by substantial collaborative space, including discussion-enabling areas as well as quickly reachable prototyping space. One recent result from Stanford indicates that teams function better without a designated leader and with certain personality types and particular documentation styles over others.

We can illustrate industrial experiences with customer-centric design thinking via two successful examples. First, one credit-card company from Switzerland solved a customer relationship management problem by using design thinking to produce a novel tablet app for its customers. Second, a major automobile manufacturer wanted to rethink mobility and used design thinking to develop a tablet app that helps move customers from one location to another with different forms of transportation. Neither requirements engineering nor agile development were well-suited to these tasks, although some form of each was naturally used to develop the software. It was the holistic approach that was successful.

Apps are a major new type of software component, especially as the Internet of Things becomes the app-enabled world. Companies that wish to play in this world must establish the right environment for their workforce. Merging design thinking with requirements engineering and agile development will let them consider the strongly diverging human-oriented working mode as well as the more technically driven perspectives of the other two methodologies. The HPI in Potsdam is already researching combining design and engineering by injecting design thinking into requirements engineering, and at St. Gallen, we're beginning a major initiative in this area, focusing on app development and management. We're aware that we have an intensive and exhausting journey ahead, and we invite others to join with us in this exciting exploration.

References

1. B. Nuseibeh and S. Easterbrook, "Requirements Engineering: A Roadmap," *Proc. Conf. Future of Software Eng.* (ICSE 00), ACM, 2000, pp. 35–46.
2. P. Zave, "Feature Interactions and Formal Specifications in Telecommunications," *Computer*, vol. 26, no. 8, 1993, pp. 20–29.
3. B.H.C. Chen and J.C. Atlee, "Research Directions in Requirements Engineering," *Proc. Future Software Eng.* (FOSE 07), IEEE CS, 2007, pp. 285–303.
4. F. Paetsch, A. Eberlein, and F. Maurer, "Requirements Engineering and Agile Software Development," *Proc. 12th IEEE Int'l Workshop Enabling Technologies: Infrastructure for Collaborative Enterprises* (WETICE 03), IEEE, 2003, pp. 308–313.

Christophe Vetterli is a research associate at the Institute of Information Management at the University of St. Gallen (HSG). His PhD research is in embedding design thinking into the corporate IS environment. Vetterli has an MA in business innovation from HSG. Contact him at christophe.vetterli@unisg.ch.

Walter Brenner is a professor of information management and the managing director of the Institute of Information Management at the University of St. Gallen (HSG). His research focuses on the intersection of business and information technology at the executive level. Brenner has published more than 25 books. Contact him at walter.brenner@unisg.ch.

Falk Uebernickel is an assistant professor at the Institute of Information Management at the University of St. Gallen (HSG). His research focuses on business innovation and information management. Uebernickel has a PhD in business administration from HSG. Contact him at falk.uebernickel@unisg.ch.

Charles Petrie was a guest professor at Karlsruhe University, Germany, in 2012, and at the University of St. Gallen, Switzerland, where he will be continuing in 2013. He retired as a senior research scientist from the Stanford University Computer Science Department. His research topics are concurrent engineering, enterprise management, and collective work. Petrie has a PhD in computer science from the University of Texas at Austin. He was a founding member of technical staff at the MCC AI Lab, the founding editor in chief of *IEEE Internet Computing*, and the founding chair of the Semantic Web Services Challenge. He teaches and coaches in innovation via design thinking. Contact him at petrie@stanford.edu.