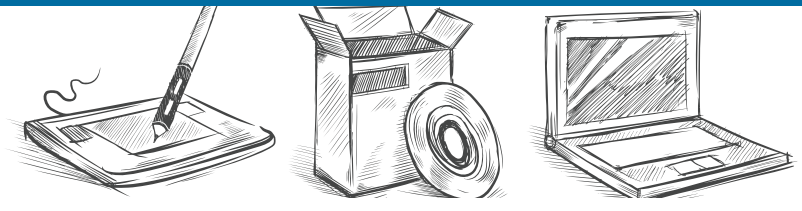
Software Sketchifying: Bringing Innovation into Software Development

Željko Obrenović, Software Improvement Group

// Software sketchifying is a software development activity that stimulates spending more time generating and considering alternative ideas before making a decision to proceed with engineering. It's supported by Sketchlet, a flexible tool that empowers both engineers and nonengineers to work with emerging technologies and explore their possibilities. //



HENRY FORD'S ASSEMBLY-LINE production of the Model T inspired changes in the automotive industry, and the software industry has made numerous attempts to apply similar ideas (for example, see the chapter, "Will the Real Henry Ford of Software Please Stand Up" in Robert L. Glass's book).[1] While the assembly-line philosophy is well known, Ford's approach to innovation and the process that preceded the Model T's production is less so. Between 1892 and the formation of the Ford Motor Company in 1903, while working mostly for the Edison Illuminating Company, Ford built about 25 cars. In the five years after the company's formation, he built and sold eight models—Models A, B, C, F, K, N, R, and S— before settling on the Model T. He tested prototypes labeled with the 11 missing letters. Ford summed up this experience this way:[2]

> I do not believe in starting to make until I have discovered the best possible thing. This, of course, does not mean that a product should never be changed, but I think that it will be found more economical in the end not even to try to produce an article until you have fully satisfied yourself that utility, design, and material are the best. If your researches do not give you that confidence, then keep right on searching until you find confidence.... I spent twelve years before I had a Model T that suited me. We did not attempt to go into real production until we had a real product.

Today's automotive industry has changed significantly since Ford's initial success, but some of his philosophy behind innovation still remains. For example, Toyota's "nemawashi" principle states that decisions should be implemented rapidly but made slowly, by consensus, and after considering all options.[3] Bill Buxton, who studied innovation in the automotive industry, noted that a new car's design phase starts with a broad exploration that culminates in the construction of a full-size clay model and costs over a quarter of a million dollars.[4] Only after bringing the new concept to a high level of fidelity in terms of its form, business plan, and engineering plan does a proj-

0740-7459/13/$31.00 © 2013 IEEE

ect get a "green light." After that, it typically takes a year of engineering before the project can go into production.

Inspired by general ideas about how the automotive industry brings innovation into manufacturing, I developed software sketchifying as an activity to stimulate and support software stakeholders to spend more time generating and considering alternative ideas before making a decision to proceed with engineering. My view on software sketchifying combines general ideas of sketching[4] and creativity support tools[5] with several existing software engineering approaches. To support and explore this view, I developed Sketchlet (http://sketchlet.sourceforge.net), a flexible, Java-based tool that empowers engineers and nonengineers to work with emerging software and hardware technologies, explore their possibilities, and create working examples—called *sketchlets*—that incorporate these emerging technologies.

Product Innovation and Software Engineering

Contrary to the automotive industry, the software industry has a rich history of engineering wrong products. Ill-defined system requirements and poor communication with users remain top factors that influence software project failures.[6] Frederick Brooks also noted that the hardest single part of building a software system is deciding precisely what to build.[7] He proposed rapid system prototyping and iterative requirements specification as a way to solve this problem. Many existing software engineering methodologies, including the Rational Unified Process, Extreme Programming, and other agile software development frameworks follow iterative and incremental approaches.

However, these approaches have limitations when it comes to true innovation. Although prototyping can let us cheaply represent and test our ideas, and iterative and incremental development can help further refine our ideas based on frequent user feedback, neither approach directly supports the generation of new product ideas, nor do they encourage the consideration of alternatives.

Buxton went further in his critique of the innovation capacity of iterative, incremental software development, seeing no comparison between software product design and the development of new automobiles.[4] He argued that innovative software projects need at least a distinct design phase followed by a clear green-light process before proceeding to product engineering. He saw design and engineering as different activities that employ different processes and for which people suited to one are typically not suited for the other.

Software Sketchifying

I built on Buxton's suggestion by introducing software sketchifying into software product development as a complement to prototyping and engineering. The sidebar presents a sketchifying example scenario of how it might work in developing software systems for an automobile.

> Neither prototyping nor incremental development directly support the generation of new product ideas.

Software Sketchifying Approach

One key characteristic of this approach is postponing the main development activity for the benefit of free exploration, following a main principle of creativity: to generate a good idea, you must generate multiple ideas and then dispose of the bad ones.[1,4] Another key characteristic is stimulating early involvement of nonengineers. Such users often have expertise that's important for understanding customers and their needs. More specifically, the example scenario in the sidebar illustrates several points about software sketchifying:

- The designer's main activity is exploration, learning about a problem and potential solutions and answering a question about what to build.
- Such explorative activity is heuristic, creative, and based on trial and error, rather than incremental and iterative. The designer generates several ideas, most of which will be rejected. However, this process yields important lessons and stimulates generation of novel ideas. These lessons and ideas are the activity's main outcome.
- The exploration activity is not accidental, but disciplined and systematic.
- The exploration is holistic, enabling designers to reflect on relations among user issues, software and hardware possibilities, and the overall dynamics of human-computer interaction. The ideas in the example scenario are influenced not only by software but also by human factors and problems related to car mechanics and equipment.
- The exploration enables early user

A SKETCHIFYING SCENARIO

Consider an example scenario with Mirko, an interaction designer at a company that builds software for new generations of cars with advanced sensing and display technologies. Mirko has recently joined the company to explore ideas for software applications that exploit novel opportunities, such as using data from a car radar, GPS sensors, and links to Web services.

Mirko's first task is to explore two applications: a system for warning about the proximity of other cars and a system for presenting news in idle situations, such as waiting for a traffic light. Mirko isn't a programmer, nor is he familiar with all the technical possibilities of modern cars, but he uses a design environment through which he can access and explore software services and components related to his task without serious programming.

To understand what's possible, Mirko first talks with several of his company's engineers. They advise him to start by using a car simulator, which provides a realistic but safe environment to learn about new automotive technologies. One engineer also writes a small adapter that connects the car simulator logger to Mirko's design tool. This adaptation gives Mirko immediate access through a simple spreadsheet-like interface to the simulator data—such as the car's speed and its distance from the car in front of it.

Mirko starts a design environment on his laptop and connects it to the simulator. After becoming familiar with the simulator's possibilities, he turns to his laptop to create a few *sketchlets*, which are simple interactive pieces of software.

PROXIMITY WARNING SYSTEM

To explore the options for implementing a proximity warning system, Mirko first considers three presentation modes: graphical, audio, and haptic (vibration). For graphical presentation, he uses an editor in his design environment and creates several simple drawings. Then he opens the properties panel and connects the variables from the car simulator to the graphical properties of drawn regions. For example, he creates a sketchlet in which an image's transparency dynamically changes as a function of the distance from the car in front of the driver. He then experiments with other graphical properties, such as image size, position, or orientation. He returns to the simulator and tries each alternative. He also invites a few colleagues to try out and comment on his ideas.

After exploring graphical options, he proceeds to create audio sketchlets. He first tries a MIDI-generator service and connects data coming from the sensor to MIDI note parameters, such as pitch or tone duration. He also experiments with a text-to-speech service, generating speech based on the conditions derived from car data. Finally, he explores using an MP3 player with prerecorded sounds. He then goes back to the simulator and tries these alternatives.

Mirko also wants to try a vibration modality to present navigation information, which the simulator doesn't support. He decides to use a simple trick, starting an application on his mobile phone that lets his design environment control the phone's resources, including its vibrator. Using gaffer tape, he fixes the mobile phone to the steering wheel and creates several sketchlets that map the distance from the car in front of him to vibration patterns. Marko knows it's not a very elegant solution, but it lets him explore basic opportunities of this modality with available resources and little work.

NEWS PRESENTATION

Mirko also plays with some other options related to the application for presenting news. He starts a Google news service in his design environment and creates a simple page that presents an HTML output of the news service. He then creates a condition for the page's visibility so that the news appears as an overlay on part of the windshield, but only when the car's speed is zero and the automobile is not in gear. He also experiments with speech services that let a user set a news search query by speech.

After finishing his work in the lab, Mirko decides to collect some real-world experiences and try some of his more promising sketchlets in a real car. With help from engineers who are working on testing cars, Mirko gets an extension of his design environment that uses a Bluetooth connection to a test car's onboard diagnostic (OBD) system. With this addition, Mirko creates a simple setting using his smartphone as a presentation device, positioned under a windshield. He connects the smartphone to his laptop, which uses a simple remote desktop client to capture a part of a screen from his laptop. On the laptop, Mirko is running the sketchlets that he created in the lab and that are now connected to the car's OBD system. He asks a colleague to drive the car while he observes a situation and videorecords a whole session for later analysis.

During the process, Mirko constantly interacts with other stakeholders, regularly presents his findings, and lets clients and colleagues try out some of his sketchlets. In this way, Mirko is helping develop new products by providing realistic and tested ideas before and outside the main development activity.

involvement through simple but functional pieces of software in the form of sketchlets.
- Working with real systems, such as the car simulator, car diagnostic system, and Web services, lets a designer learn about the possibilities and limitations of software technologies and create conceptual proposals that are more realistic.

Designers generally aren't engineers who can program and extend their design environments. However, they're part of a broader community of people who can help them learn and extend the exploration space on an ad hoc basis. Sketchifying supports this interaction without taking too much time, thereby empowering nonengineers to explore emerging technologies and to test their ideas without additional help from developers.

Software Sketchifying Tools

To support and explore this approach, Sketchlet combines elements from traditional sketching, software hacking, opportunistic software development, and end-user development. Sketchlet builds on the results of the Sketchify project (http://sketchify.sourceforge.net), which explored possibilities to improve early design stages and education of interaction designers.[8]

Sketchlet has two main roles:

- to enable designers to create a number of simple pieces of software—sketchlets—as a way to quickly and cheaply explore software and hardware technologies and their potential applications, and
- to support involvement of software engineers in short, ad hoc sessions that give designers realistic pieces of technologies that might be useful for design exploration.

Sketchlet lets designers interact directly with software and hardware technologies through a simple, intuitive user interface. To simplify the integration with these technologies, Sketchlet combines ideas from opportunistic software development with techniques used by hacking and mashup communities.[9,10] A full description of Sketchlet is out of scope for this article. Two appendices containing more detail about how Sketchlet implements the sidebar's example scenario are available online at http://doi.ieeecomputersociety.org/10.1109/MS.2012.71. I also encourage readers to download and try the tool for themselves.

Sketchlet combines elements from traditional sketching, software hacking, and opportunistic software development.

Initial Sketchlet Applications and Results

I've developed and applied the ideas about software sketchifying in three projects that featured collaboration among software engineers, interaction designers, and researchers. In these projects, interaction designers and researchers were primarily responsible for creating and evaluating novel conceptual proposals and ideas:

- *Connect & Drive project* (www.tue.nl/en/university/departments/industrial-design/research/research-programs/user-centered-engineering/projects/explorations-in-interactions/connect-drive). Several researchers used Sketchlet to explore options for developing software systems for cooperative adaptive cruise control systems in cars, based on Wi-Fi communication between vehicles and road infrastructure.
- *Persuasive Technology, Allocation of Control, and Social Values project* (http://hti.ieis.tue.nl/node/3344). Sketchlet played a similar role as it did in the Connect & Drive project, helping researchers investigate software products for developing persuasive technologies that encourage people to hand over control to intelligent automation of cars.
- *Repar project* (*Re*solving the *Par*adox; www.repar-project.com). Sketchlet was one of the flexible prototyping tools in user-centered design processes, allowing designers to create and evaluate (ill-defined) product concepts early in the development.

Although Sketchlet is still in early development, the approach and tool showed several positive effects in these projects. First, it broadened the opportunities to constructively involve nonengineers, including interaction designers, psychologists, and students. Our tools empowered nonengineers to easily explore relevant technologies and to independently create and test their ideas. The companies involved benefited from their nonengineering expertise and knowledge early in the design process.

Sketchlet also promoted different collaboration between engineers and nonengineer designers. Prior to using Sketchlet, most of the companies followed the approach of making designers responsible for creating a conceptual proposal, which they gave to developers for implementation with

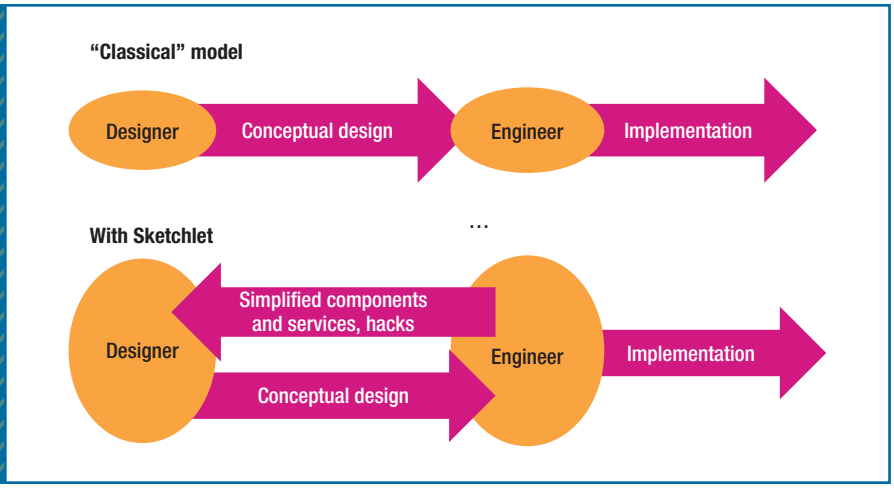


FIGURE 1. Comparing the classical design-engineering interaction with sketchifying. With sketchifying, supported by tools like Sketchlet, the interaction between designers and engineers can work in two ways, allowing engineers to give designers early access to simplified versions of software components and services that the engineers might use later in the implementation.

little interaction, except to clarify their designs. With Sketchlet, the interaction between designers and engineers could work in two ways, with engineers giving designers simplified versions of software components and services—early in the design process—that the engineers might use later in the implementation (see Figure 1).

The connected services, although simplified, resemble real components, and sketchlets expressed in terms of these services come closer to the implementation platform that the engineers will use. This change addressed one problem that many companies experience when designers and engineers need to work together—namely, the engineers perceive designers' ideas as unrealistic, too distant from available technology, and not precise enough to be useful. Through the exploration of these services, designers can develop more realistic expectations about the possibilities and limitations of technologies, and incorporate this understanding into design proposals.

Lastly, Sketchlet influenced the mindset of companies toward more and broader explorations early in the software design. Sketchlet helped illustrate the potential of such exploration and inspire the companies to think how other tools could be used in a similar explorative way.

Sketchifying Benefits and Relation to Other Approaches

Software sketchifying can help better define product requirements so that the subsequent engineering process has a clear focus and goal. It promotes direct exploration of emerging technologies and creation of working examples of simple pieces of software with these technologies as a way to identify potential problems and provoke reactions from users as early as possible. The tool shows the effects of design decisions on user experience and supports user testing before actual development starts.

Exploring the possibilities and limitations of technologies early in the design helps identify a number of problems or user issues before investing in a significant development effort. Discovering such problems later in the process could require changes and additional effort. Early discovery is particularly important in projects using emerging technologies, which have many unknowns—including how well users will accept them.

Promoting the constructive involvement of nonengineers in the design process opens the door to help from experts in fields such as human psychology, which in turn reduces the burden on developers. Moreover, as Glass noted,[1] users who understand the application problem to be solved are often more likely to produce innovation than computer technologists, who understand only the computing problem to be solved. The sketchifying approach requires occasional involvement of developers, but it aims to incorporate them in short ad hoc sessions, and the intent is to empower nonengineers to explore further without developers' help. Once the developer adapts some technology for Sketchlet, nonengineers can work with this technology through a simple end-user interface that does not require technical expertise or programming knowledge.

Relation to Prototyping and Engineering

Software sketchifying complements existing prototyping and engineering approaches by its focus on free exploration and a trial-and-error approach versus a more iterative, incremental approach of prototyping and engineering (see Figure 2).

Sketchifying supports users in constructing a novel idea and enables nonengineers to actively contribute. This brings software design closer to the practice of other engineering disciplines, in which the design phase precedes the main engineering activity, and designers (usually nonengineers) are encouraged to freely explore ideas before consolidating a few of them for

further development. For instance, it's not unusual for an industrial designer to generate 30 or more sketches a day in the early stages of design, each possibly exploring a different concept.[4]

Software sketchifying precedes prototyping, which tests, compares, and further develops aspects of selected ideas. With a prototype in place, the development can take an evolutionary approach. Prototyping should assess whether selected ideas are feasible and should help decide whether to proceed with engineering. Prototyping aims at making an idea more detailed and concrete, rather than coming up with radically new ideas. Engineering turns the winning idea into a robust and usable product.

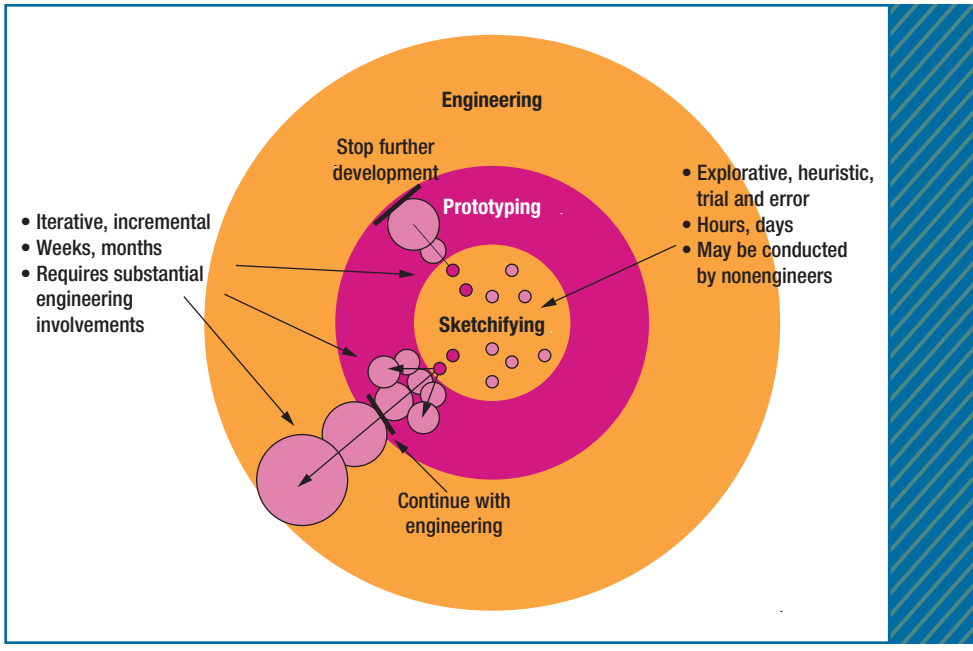


FIGURE 2. An idealized representation of relationships among sketchifying, prototyping, and engineering. Sketchifying supports users in constructing a novel idea. It precedes prototyping, which tests, compares, and further develops aspects of selected ideas. Engineering turns the winning idea into a robust and usable product.

Relation to Other Software Tools

In principle, tools other than Sketchlet could implement the sketchifying idea. However, many current tools can't fully support it because they're not optimized for free exploration and involvement of nonengineers. For example, we could use standard programming languages, such as Java, C#, C++, or programming tools oriented toward interaction design such as Flash and Processing to implement our example scenario. However, programming requires significant expertise, time, and effort—an investment that's simply too high for the intended purpose of generating new ideas and exploring possibilities.

Existing low-fidelity prototyping environments provide ways to quickly create prototypes with inputs taken from external services or sensors.[11,12] These environments might be excellent choices for exploring interactions in various domains. The problems I'm addressing cross these domains and require a variety of sensory inputs and links to diverse software services as well as additional components specific to the companies I'm working with. In addition, such tools often require too much precise specification, partly because they're primarily developed for advanced prototyping rather than for free and broad exploration.

Electronic sketching systems are another promising direction for design tools, enabling designers to create interactive systems with ease using intuitive and natural pen gestures.[13] From the viewpoint of my example scenario, these systems have the drawback of being specialized for specific domains and used successfully only in inherently graphical domains that have a stable and well-known set of primitives, such as 2D and 3D graphics or websites.

Another alternative is to use simple freehand drawings and techniques such as screen prototyping. Such techniques can help in exploring a solution's graphical elements. However, they can describe overall system interactions, such as sensing device inputs and user response dynamics, only in very abstract terms. Moreover, paper sketching doesn't let users explore the possibilities and limitations of emerging technologies. Direct exploration of such technologies yields more concrete ideas about how to best employ them.

Sketchlet borrows ideas from existing solutions, while trying to overcome some of their limitations. I also see it as a complement to existing tools, rather than a replacement. On several occasions, designers have used Sketchlet in conjunction with other tools. For example, some of our users employed Max MSP for signal processing and audio effects and Sketchlet for connections to sensor devices and visualization.

My initial experiences with applying software sketchifying are encouraging. However, an important limitation of this approach is that it requires significant changes of current development culture

ABOUT THE AUTHOR

ŽELJKO OBRENOVIĆ is a technical consultant at Software Improvement Group, Amsterdam. He did the work reported here while working as an assistant professor in Eindhoven University of Technology's Department of Industrial Design. His professional interests include, software engineering, design of interactive systems, end-user development, rapid prototyping, creativity support tools, and universal accessibility. Obrenović received a PhD in computer sciences from the University of Belgrade. Contact him at obren@acm.org.

in its emphasis on postponing the start of development to benefit free exploration, more active involvement of nonengineers and end users, and new forms of interaction between engineers and nonengineers prior to the main development activity. Such changes, in my experience, aren't easy to achieve, but without them, the sketchifying tools are less effective and tend to be used in a limited way.

In future work, I plan to develop a more general approach toward building software services and components so that each service could have two sets of APIs: one engineering API with full functionality, and one sketchifying API that would represent a simplified, limited sample of the full functionality. I also plan to address collaboration because the current implementation primarily supports individual use and is of limited value in collaborative design sessions.

References

1. R.L. Glass, *Software Creativity 2.0*, developer.* Books, 2006.
2. J. Grudin, "Travel Back in Time: Design Methods of Two Billionaire Industrialists," *ACM Interactions*, vol. 15, no. 3, 2008, pp. 30–33.
3. J. Liker, *The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer*, McGraw-Hill, 2004.
4. B. Buxton, *Sketching User Experiences: Getting the Design Right and the Right Design*, Morgan Kaufmann, 2007.
5. B. Shneiderman, "Creativity Support Tools: Accelerating Discovery and Innovation," *Comm. ACM*, vol. 50, no. 12, 2007, pp. 20–32.
6. R.N. Charette, "Why Software Fails," *IEEE Spectrum*, vol. 42, no. 9, 2005, pp. 42–49.
7. F. Brooks, "No Silver Bullet—Essence and Accidents of Software Engineering," *Computer*, vol. 20, no. 4, 1987, pp. 10–19.
8. Ž. Obrenović and J.B. Martens, "Sketching Interactive Systems with Sketchify," *ACM Trans. Computer-Human Interaction*, vol. 18, no. 1, 2011, article 4.
9. B. Hartmann, S. Doorley, and S.R. Klemmer, "Hacking, Mashing, Gluing: Understanding Opportunistic Design," *IEEE Pervasive Computing*, vol. 7, no. 3, 2009, pp. 46–54.
10. Ž. Obrenović, D. Gaševic, and A. Eliëns, "Stimulating Creativity through Opportunistic Software Development," *IEEE Software*, vol. 25, no. 6, 2008, pp. 64–70.
11. M. Rettig, "Prototyping for Tiny Fingers," *Comm. ACM*, vol. 37, no. 4, 1994, pp. 21–27.
12. Y.K. Lim, E. Stolterman, and J. Tenenberg, "The Anatomy of Prototypes: Prototypes as Filters, Prototypes as Manifestations of Design Ideas," *ACM Trans. Computer-Human Interaction*, vol. 15, no. 2, 2008, article 7.
13. J.A. Landay and B.A. Myers, "Sketching Interfaces: Toward More Human Interface Design," *Computer*, vol. 34, no. 3, 2001, pp. 56–64.