

Improving Mobile Application Development

Ray Bareiss and Todd Sedano

Carnegie Mellon University

Silicon Valley

{ray.bareiss, todd.sedano}@sv.cmu.edu

Many mobile app developers act more like end-user programmers – professionals in disciplines other than software development who create computer programs as pragmatic tools to solve work-related problems – than like professional software engineers. They want the quick gratification of creating a program rather than what they view as the drudgery of following engineering processes. They focus on the construction phase of software development, slighting other phases such as requirements, design, testing, and debugging.

Ko, et al, have drawn a clear contrast [1]:

Software Engineering Activity	Professional SE	End-user SE
Requirements	<i>explicit</i>	<i>implicit</i>
Specifications	<i>explicit</i>	<i>implicit</i>
Reuse	<i>planned</i>	<i>unplanned</i>
Testing and Verification	<i>cautious</i>	<i>overconfident</i>
Debugging	<i>systematic</i>	<i>opportunistic</i>

The end-user software engineering research community is actively working to improve the quality of end-user-produced software [1,2,3]. End-user software engineering is defined as systematic and disciplined activities performed by end-user programmers that address software quality concerns [1]. We can draw upon the lessons of end-user software engineering research to devise a lightweight set of tools, techniques, and processes that mobile app developers may recognize as being worth additional overhead because of improvements in software quality, ideally leading to higher app store ratings and greater customer adoption. (Most mobile app developers are keenly aware of their ratings, and any techniques that demonstrably increase ratings will be easy to “sell” to the community.)

Research in end-user software engineering has primarily addressed requirements, design, reuse, verification, and testing.

In reality, the requirements and design processes are tightly intertwined and are, perhaps best treated as one. This combined requirements and design process maybe the easiest one to sell to app developers: In the abstract, design is sexy and better designed apps that better meet the user’s needs are likely to have higher adoption rates and, thus, revenues. A lightweight, user-centered requirements and design process might comprise Rapid Contextual Inquiry [4], persona- and scenario-based design [5], adherence to platform guidelines [6,7], early prototyping, and

lightweight user testing [8]. (Note that user testing is an important way to discover requirements in addition to providing usability feedback, per se.)

Reuse can comprise both reuse of patterns and of code. Pattern reuse is somewhat encouraged at the user interface level by platform design guidelines which are, at least, loosely enforced by the two major app stores [6,7]; in particular, Apple tightly controls their API to encourage developers to use the phone in a certain way. Code reuse for the construction of human and, especially, sensor interfaces is a relatively easy sell given the complexity of coding such interfaces. The complexity of cloud back-ends, which are increasingly important to nontrivial apps, may also encourage reuse of both architectures and code. The key to promoting reuse may simply be making developers more aware of available patterns and code.

Promotion of a rigorous testing process is more problematic because testing is arguably the least glamorous aspect of software development. Much research on end-user software engineering has focused on this area. The most promising approach seems to be the incorporation of visual indicators into development environments to encourage software testing, e.g., WYSIWYT (what you see is what you test) [9]; ironically, common development platforms do not make automated testing easy. Research also suggests the promise of a Surprise-Explain-Reward approach [10] to rouse a developer's curiosity and then concern, thus promoting engagement in a systematic testing process such as statement and branch coverage testing.

Perhaps the most problematic goal is to convince app developers to engage in some degree of formal verification of their code. Despite its long-time popularity in software engineering curricula, the use of verification techniques by professional software engineers remains practically nonexistent, so the prospect of getting app developers to adopt verification seems remote indeed.

Over the long-term, there is another approach to improving software quality. The software craftsmanship movement appeals to developers' desire to be recognized as great programmers by their peers [11]. Developers are encouraged to practice areas in which they are weak in order to become experts in their field [12]. Some programmers convert their practices into a codified kata so that other developers can more easily learn the same lessons. While katas are prevalent for most programming languages, there is a dearth of specific mobile katas. (At the time of writing, we could only find one.) Developers also pair program with other developers for the express purpose of improving their craft, and in fact, some will even travel around the world on a "journeyman tour" [13].

This paper has proposed a set of techniques, tools, and processes, drawn in large part from end-user software engineering research, that the authors believe to compose a reasonable approach to improving the quality of mobile apps. The remaining questions are "Are developers willing to learn these techniques?" and "Are they willing to use them consistently?" One way to explore this question is to

offer a course in mobile application development that embodies these techniques (assuming that appropriate tools are in place as noted earlier in the paper). A project-based educational approach using the pedagogy pioneered by Carnegie Mellon University - Silicon Valley seems most appropriate for the targeted students [14]. Such a course could be offered to matriculating and nonmatriculating students at our campus. However, it might reach a wider audience at a community college. Finally, the best approach of all might be to use principled mobile app development to interest young people in careers in software development via a “summer camp” format. We hope to return in the near future to report on the development of such a course.

References

- [1] Ko, A. J., Abraham R., Beckwith L., Blackwell A., Burnett M.M., Erwig M., Scaffidi C., Lawrence J., Lieberman H., Myers B.A., Rosson M.B., Rothermel G., Shaw M. and Wiedenbeck S., “The State of the Art in End-User Software Engineering,” ACM Computing Surveys, to appear.
- [2] Burnett, M., “What is end-user software engineering and why does it matter?” In End User Development. Proceedings of the 2nd International Symposium, IS-EUD 2009, Wulf, V. and Burnett, M. (Eds.) Siegen, Germany, March 2-4, 2009. LNCS Volume 5435/2009. Dordrecht: Springer, pp. 15-28, 2009.
- [3] Dwivedi, V., “End-User Software Engineering” (PowerPoint presentation)
<http://www.cs.cmu.edu/~bam/uicourse/2011hasd/lecture27-End%20User%20Software%20Engineering.pptx>
- [4] Holtzblatt, K., Burns Wendell, J., and Wood, S., Rapid Contextual Design, San Francisco: Morgan Kaufmann, 2005.
- [5] Goodwin, K. and Cooper, A., Designing for the Digital Age: How to Create Human-Centered Products and Services, San Francisco: Wiley, 2009.
- [6] Anonymous, Apple iOS Human Interface Guidelines,
<http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/mobilehig/Introduction/Introduction.html>
- [7] Anonymous, Android User Interface Guidelines,
http://developer.android.com/guide/practices/ui_guidelines/index.html
- [8] Krug, S., Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability Problems, Berkeley: New Riders, 2010. (Also see <http://sensible.com>, especially the video of an example usability test.)

- [9] Rothermel G., Burnett M., Li L., Dupis C. and Shertov A., "A Methodology for testing spreadsheets," ACM Transactions on Software Engineering Methodologies, 10(1), 110-147, 2001.
- [10] Wilson, A., Burnett, M., Beckwith, L., Granatir, O., Casburn, L., Cook, C., Durham, M., Rothermel, G.: Harnessing Curiosity to Increase Correctness in End-User Programming. In: ACM Conference on Human Factors in Computing Systems. ACM, New York, 2003.
- [11] Anonymous, Manifesto for Software Craftsmanship,
<http://manifesto.softwarecraftsmanship.org>
- [12] Ericsson, K., Prietula, M., Cokely, E.: "The Making of an Expert." Harvard Business Review. 2007.
- [13] Hoover, Dave. "Dave Hoover's Journeyman Tour"
<http://nuts.redsquirrel.com/post/1181144648/dave-hoovers-journeyman-tour>
- [14] Bareiss, R. and Sedano, T. , Developing Software Engineering Leaders, Proceedings of the First International Symposium on Tangible Software Engineering Education (STANS-09 in Tokyo, Japan), October 2009.