

# “Don’t Tick Off the User”

## ...and other industry perspectives on mobile application design

Michael Pearce, Craig Janssen and Nitya Narasimhan, Motorola Labs

**Abstract**—The rich capabilities and personal context history available on mobile devices makes them ideal targets for enabling a wide range of applications. However, while researchers tend to develop interesting and useful applications for the mobile environment, very few of these applications successfully navigate the road to commercial adoption. For pervasive computing to become a reality, we need more innovative architectures and applications to be deployed in commercially-available devices – not as post-sale extensions used only by the average geek, but as built-in features targeting the average user. In this paper, we share some lessons learned from years of industry experience in architecting and developing applications for mobile devices. Our goal is not only to offer simple guidelines to help other researchers, but also to motivate a wider collection of such design rules for consideration by the pervasive computing community.

**Index Terms**— application design rules, mobile computing, commercialization, pervasive computing, human interaction

### I. INTRODUCTION

MOBILE phones are gaining in importance to application developers and to end users. More than a billion phones shipped last year<sup>1</sup> (~1 device for every 6 people worldwide) making it arguably the most ubiquitous device on the planet. The size of this market, combined with the rich computing and context capabilities of the mobile device, makes it attractive to application developers and researchers. However, while we see numerous innovative and useful applications being developed for this platform, very few actually make it to product. There are many reasons behind such failures, primarily:

- *Costs* – associated with product hardening.
- *Commercial appeal* – to a target user demographic
- *Viability* – in terms of market readiness and usability

Application design directly impacts (or is impacted by) every one of these factors. Because pervasive computing relies on the *widespread* deployment of innovative applications and architectures, it is imperative that we (as researchers) also pay attention to design factors that drive the commercialization of such technologies. In this paper, we attempt to articulate a few simple design guidelines that are based on our collective experiences over several years in developing pervasive applications and architectures for the mobile phone platform. Our *goal* is to help researchers avoid the more common pitfalls

encountered in taking research prototypes to product on this platform; our *hope* is to stimulate a broader discussion on the subject and drive additional efforts to collect “do’s and don’ts” for the next generation of pervasive computing researchers.

### II. A FEW SIMPLE RULES (TO START WITH)

We start with a few basic observations. First, mobile phones are used throughout the world by people of diverse cultures, demographics and interests – one size does *not* fit all. Second, mobile phones are resource-constrained – while their storage and processing capabilities are increasing steadily, so too are the demands of their applications and users. Third, mobile phones are primarily meant to be communication devices, not computational ones – increasing device convergence often blurs the distinction between a smart *phone* and a telephony-capable *computer*, leading to bad design decisions. Finally, the network is *not* free. While “all-you-can-eat” data models exist, in reality, requiring all subscribers to sign up for such plans (in order to use specific applications) is not beneficial either to the user or to the carrier. With these factors in mind, we define the following simple rules for application design:

1. **Don’t tick off the user.** Our cardinal rule! Many of the rules that follow can in some sense be traced back to this overriding goal. It’s much easier to frustrate users than it is to make them happy. So, before introducing a feature that’s *intended* to make users happier, make sure that the chances it will frustrate a significant number of users is minimal. A user study can be used to provide evidence one way or the other – *before* the product is in the field. For example, most users often learn complex navigational patterns over time and perform them without conscious effort. Reorganizing menus based on inferences made about popular choices can not only confuse users – it may frustrate them enough to return the product and sour them on the entire brand. These users are expensive to win back.
2. **Keep the interface simple.** Mobile pervasive computing devices simply don’t have the display real-estate to afford the kind of clutter desktop computing systems can. With every extra click necessary to discover a feature, half of the user population gives up. Trade off between navigation breadth (display clutter) and navigation depth (click count) in deciding the number of features to support.
3. **Don’t expect users to configure anything (ever).** Most users simply won’t take the time to change default

<sup>1</sup>“Cell phone shipments top 1 billion for the first time”, USA Today online, published Jan 26, 2007. [http://www.usatoday.com/tech/wireless/phones/2007-01-26-cellphone-shipments\\_x.htm?csp=15](http://www.usatoday.com/tech/wireless/phones/2007-01-26-cellphone-shipments_x.htm?csp=15)

configuration settings. So, those defaults had better work for most users. If these defaults won't work for everyone, a simple startup wizard can be used to establish the initial configuration. But, resist the temptation to ask more of the user here than is absolutely necessary – it may seem like a good idea to ask users for their font and color preferences, but the more questions asked at this point, the fewer the number of users who will continue to stick with the dialog and eventually end up using the feature.

4. **Most alerts and interruptions will be ignored.** Users have learned to pay attention to certain classes of interruptions. On a desktop, incoming email and instant messages get their attention. Similarly, on mobile phones, users pay attention to incoming phone calls and text messages. However, if the interruption isn't coming from another *person*, it is much less likely to be read and acted on. The user simply wants to get on with what they're doing. The bigger danger here lies in creating a system that interrupts the user frequently with trivial information. Such systems simply train the user to ignore *all* alerts, thereby completely nullifying the point of having alerts in the first place.
5. **“Easy” is hard.** Pervasive computing systems by their very nature are not like desktop computers – they are not used continuously for long periods of time to accomplish complex tasks. Furthermore, mobile pervasive systems have limited user-interface capabilities. For these types of systems, it is critical that the tasks they enable are as easily accomplished as possible. This necessitates investing significant time and effort on ease-of-use. We simply can't expect users to be willing to puzzle out some complex aspect of the system. If you've ever tried to pair-bond two Bluetooth devices, you know what we're getting at here. Making a complex task look easy can be a significant investment, but one that usually pays off.
6. **Reduce feature bloat.** Each additional feature incurs cost in development, testing and technical support for achieving quality experiences. Many applications will follow the 80-20 rule (80% of users will use only 20% of features). Make a clear distinction between the core functionality (e.g., critical 20%) and other “bells and whistles” that offer potential enhancements to the user experience. Any group of creative engineers will be capable of dreaming up a huge pile of “neat” things a system could do. It is much harder to take a critical look at the resulting pile and throw out most of it.
7. **Polling is evil.** In a mobile environment – where processors are less powerful, wireless communication is expensive and batteries are limited – polling should not be the first choice in design patterns. Of course, there are some data sources that must be polled. In that case, the best you can do is to wrap the source in an observable module so that a single polling process suffices. Obviously, selection of the polling frequency is critical in reducing the impact of polling overhead while still maintaining sufficient responsiveness. For example, Bluetooth supports low-power operating

modes that essentially reduce the polling frequency. Supporting different Bluetooth modes is not difficult and allowing applications to switch between multiple operating regimes does not add significant complexity.

8. **Be bit-efficient.** This goes hand-in-hand with rule #7, but there's more to it than avoiding polling. Spending the time to design lightweight protocols with smart caching and batching strategies – rather than dropping in simple, always-on, chatty connections – will pay off in the final product. Networking interfaces not only consume resources (battery and, unfortunately, money) but potentially interfere with peer devices' communications (particularly in wireless ad hoc networks). Given the relatively smaller ecosystem of services resident on the phone, there is enough likelihood of data reuse to make such caching effective. This requires an extra investment in designing, implementing, and validating more complex protocols.
9. **Resist the temptation to over-design.** Evaluate solutions that can provide best-effort experiences with limited knowledge before developing more comprehensive but ultimately resource-hungry alternatives. For example, simple data representations (name-value pairs) and relationships may suffice in lieu of richer ontologies when crafting simple context-aware applications. Some day, more complexity may be necessary. Wait until then to introduce it.
10. **Never interfere with the primary function of a device.** On mobile phones in particular, voice is king. Always anticipate your application being pre-empted by a voice call, and potentially resumed when that call is completed. Ensure that your application's resource consumption (e.g., battery) doesn't degrade voice capability significantly; a poor communication experience can easily overshadow the utility of the application held responsible.
11. **Respect boundaries.** Maintain user privacy – especially when targeting cultures that may be more sensitive to privacy than others. Sometimes, it's enough to make sure the user understands what information they're actually sharing, with whom they are sharing it, and what it will be used for.

### III. DISCUSSION

Note that traditional rules like “Keep it simple, stupid” don't appear in this list. While KISS is a good principle in many cases – it is obviously our primary guiding principle when it comes to user interaction – keeping *everything* simple isn't always possible, and is occasionally a bad idea (see rule #5).

These rules are not exhaustive, universally applicable or even orthogonal. However, we believe that motivating a widespread agreement and adoption of such rules is critical to driving the transition from research to product. Finally, though we crafted these rules based on our experience in the mobile domain, we hope to motivate similar discussions for infrastructure and hybrid domains – leading to a collection of rules that truly encompass all aspects of pervasive computing.