Usability-improving mobile application development patterns

Article · July 2010

DOI: 10.1145/2328909.2328923

2 authors:

Bettina Biel
University of Leipzig
14 PUBLICATIONS **85** CITATIONS
SEE PROFILE

Volker Gruhn
University of Duisburg-Essen
470 PUBLICATIONS **3,404** CITATIONS
SEE PROFILE

Usability-Improving Mobile Application Development Patterns

Bettina Biel, Volker Gruhn
University of Duisburg-Essen
paluno, The Ruhr Institute for Software Technology
Essen, Germany
[bettina.biel,volker.gruhn]@paluno.uni-due.de

ABSTRACT
The fast evolution of mobile devices and their complex functionalities require an improvement of the design process applied: Using patterns allows to document and use up to date and already proven solutions. This paper introduces two usability-improving mobile application development patterns for software designers of mobile applications that run on mobile devices with no or infrequent access to remote logic or data storage. We present a client-side solution for the design of usable applications for heterogenous devices and a mobile application usability test suite.

Categories and Subject Descriptors
H.5.5 [**Information Interfaces and Presentation (e.g. HCI)**]: User Interfaces – *style guides, theory and methods; training, help, and documentation; user-centred design;* D.2.2. [Software Engineering]: Design Tools and Techniques – *user interfaces*

General Terms
Usability, Mobile Applications

Keywords
Patterns, design methodologies, interdisciplinary design

1. INTRODUCTION
Mobile applications and their design gain more and more importance. Existing mobile devices are enhanced with more computing power, connectivity and interaction capability. The fast evolution of mobile devices and tools, their heterogeneity and their complex functionalities make it very difficult to design for a short time to market. This requires an improvement of the design process applied for designing new and enhancing existing mobile applications. In order to comply with this evolving requirement we apply the approach of using patterns that hold already proven and up to date solutions for developing and designing mobile applications.

Usability is a quality attribute of the usage of an application. It describes from the users' point of view, how effectively, efficiently, safely a user can accomplish tasks with good utility that are easy to learn and to memorize (usability goals), and how pleased one feels about the whole user experience [1]. Usability can be improved by skillful design of the user interface, and, as importantly, by careful design of user-system interactions. For these, we propose usability-improving mobile application development patterns to be used by software designers in the area of mobile applications.

There are several pattern collections available that discuss graphical design; our collection shall also focus on issues beyond the graphical layout of the user interface. Such usability patterns were coarse-grainedly designed by Bass et al. [2]. Another research project, "Mobile Design Patterns and Architectures" [3], lists software development patterns with reported use in the mobile environment. Several well-known patterns [4] and some specific patterns regarding interaction design were found to be useful in the mobile domain.

The patterns presented are derived from a review of literature about best practices and online resources for developers in order to support the software design of an Android application by an interdisciplinary team of usability engineers, software architects and programmers.

The following map gives an overview of relations between patterns and their hierarchy (boxes of this papers' patterns are grey, others just indicate these patterns' broader context).

The first pattern describes how efforts to design a usable application can be optimized.

- CLIENT-SIDE MULTI-SCREEN SUPPORT**

  How can developers design their applications for correct display on many devices efficiently?

  Design for standard screen sizes and ratio and use a small number of predefined media classes that are scaled to the need of the specific device.

The resulting application has to be tested on all devices it was designed for.

- MOBILE APPLICATION USABILITY TEST SUITE **

  What tests are necessary and how can they be coordinated efficiently in order to save error correction post launch?

  Test and review from the beginning and involve users and their data even for automatic tests.

Our patterns follow a standard format of patterns, supplemented by a section called "usability consequence", a category for describing the effects on the usability goals and on user experience. Table 1 shows the pattern format.

This paper presents work-in-progress and introduces two patterns for the development of mobile applications running on mobile devices without accessing remote logic or data storage.

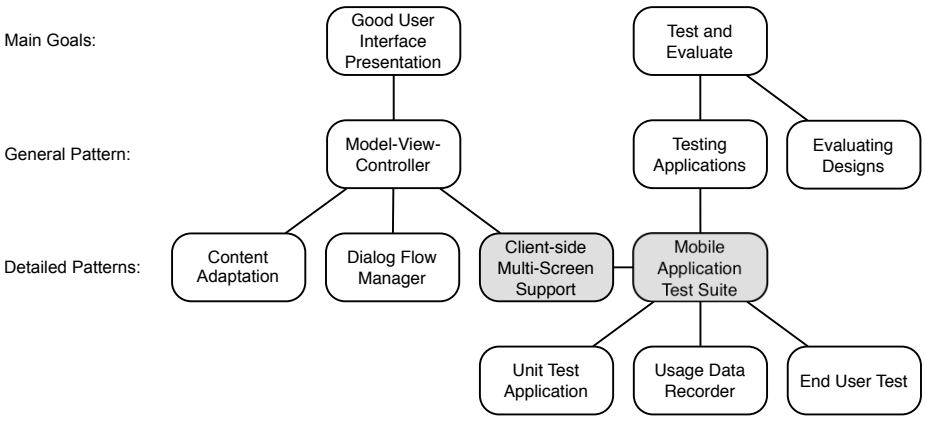

Figure 1. Relations between patterns and their hierarchy

Table 1. Pattern Format

Part	Description
Trust factor	Newly identified pattern: no asterisk; Relevant pattern: *; Pattern proven to describe a real approach: **; Regularly used and evaluated pattern: *** [5].
Name	Short name based on the solution.
Categories	User- or system-initiated interaction.
Context	Technical context regarding communication, distribution of logic and data storage.
Scenario	Situations in which the pattern is useful and why.
Problem	Problem which the pattern addresses.
Forces	Forces with an effect on the solution.
Solution	Abstract description of the proposed solution.
Related Patterns	Alternative or extending patterns.
Known uses	Example implementations.
Conse-quences	The solutions' effects on the forces.
Usability Conse-quences	Effects on usability goals: Effectiveness, efficiency, safety, utility, learnability, memorability; Effects on user experience goals such as fun, satisfaction, aesthetics [1]
Literature	References.

2. PATTERNS

2.1 Client-side Multi-Screen Support **

Interaction Category/ies: User interface presentation, device independence

Context: A mobile application that runs on a mobile device with local data storage and logic with no or infrequent access to a server.

There are multiple mobile devices and, due to short innovation cycles, new mobile devices are launched every day.

Scenario: A user wants to access the system using his own specific type of device, and he expects the application just downloaded to run on his device. It runs, but the content is not rendered properly. Another user downloads the mobile application but it does not start at all: the mobile platform on his mobile device filtered out the application because it could not display it correctly – his device's screen size is smaller than expected by the application developers.

Problem: Mobile platforms run on different devices with different screen sizes and resolutions. In order to support all user interfaces, and thus usability, application developers have to optimize usage of their applications to be displayed on multiple screens.

How can developers design their applications for correct display on many devices efficiently?

Forces:

- The heterogeneity of devices and the short innovation cycles in general make designing for all devices a time-consuming task.
- If variants for each device are provided, the application needs too much hard disk storage.
- Mobile platforms manage most of adaptation of an application to the current screen, but for a precise control it is necessary to create screen-specific resources.
- Some scaling mechanisms for resources as images can be CPU-expensive.
- Height and length of a UI-element (e.g., a button) are usually defined through pixel. A screen's density is defined as pixel per inch, so that a UI-element is displayed bigger on a screen with low density than on a screen with high density.

- Application developers have to carefully consider the possibilities of design, content and functionality within the small screen area. Screen layout could seem cluttered at a smaller device and offer to much unused space on a bigger device.

Solution: Design for standard screen sizes and ratio and use a small number of predefined media classes that are scaled to the need of the specific device.

Scaling mechanisms provided by the platform can display applications properly on most devices, especially when the screen is the same size or larger. But it may need minor adjustments before they display on smaller screens, because of the reduced screenarea there may be tradeoffs in design, content, and function. The application should therefore be designed for a standard screen of intermediate size and medium density. For example, Table 2 shows the range of typical screen sizes for mobile devices covered by HVGA (320 x 480) as standard screen.

Table 2. Range of screens supported by Android [DA2010]

	Low density (120dpi)	Medium density (160dpi)	High density (240dpi)
Small screen	QVGA (240x320), 2.6"-3.0" diagonal		
Normal screen	WQVGA (240x400), 3.2"-3.5" diagonal FWQVGA (240x432), 3.5"-3.8" diagonal	HVGA (320x480), 3.0"-3.5" diagonal	WVGA (480x800), 3.3"-4.0" diagonal FWVGA (480x854), 3.5"-4.0" diagona
Large screen		WVGA (480x800), 4.8"-5.5" diagonal FWVGA (480x854), 5.0"-5.8" diagonal	

The layout defined should be flexible, using relative units and positions, instead of absolute pixels. Also, layout containers should not demand fixed positions, and they should position their contents flexible.

For the exact control of the application's design, specific resources should be prepared and provided. Using media variants for classes of devices (high, medium, small density classes) also provides a compromise between memory and screen optimization. Instead of using a default directory, directories named according to the density provide specific resources for classes of devices.

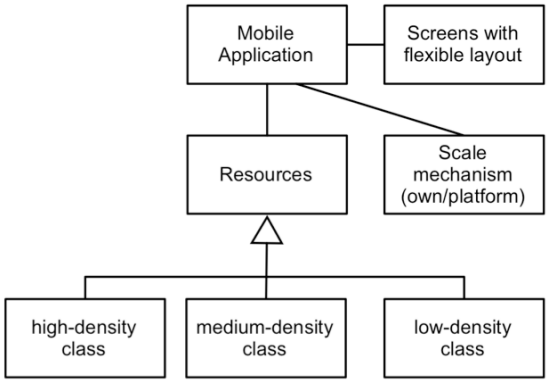

Figure 2. Components of the solution

Consequences:

[+] Programming effort is minimized when few variants are used.

[+] Regarding disc storage, providing only few variants for classes of devices needs less space than providing variants for all devices.

[+] If groups of screen-specific resources are designed, media is only scaled a little, which is nearly invisible to the user.

[+] Flexible layout improves results because then positions can be determined directly for the device.

[+/-] Some scaling mechanisms for resources as images can be CPU-expensive: If the mobile application creates a resource internally, e.g., a bitmap, and draws something on it, it creates bitmaps at the moment when it is drawn ("draw time"). Scaling at draw time is more CPU-expensive, but it uses less memory.

[+/-] Using the relative layout for a standard screen and scaling, the application will be layed out nicely for different devices. But if the screen sizes vary a lot, it makes sense to think about different versions of the application with different functionalities available.

[-] Design efforts are higher than providing just one variant, because several versions of media (images, video) have to be designed.

[-] Programming effort is higher than providing just one variant, but usability is improved.

Usability consequences:

[+] Utility: The application will be displayed correctly when screen sizes and density are considered during the design.

[+] User experience: the application must be displayed correctly for a good user experience. As users expect this, there is no additional benefit for them. A correct presentation is a basic usability expectation that must be met. If it is absent, only then users will notice, and they will not like it.

Example: The Android platform supports the developer in two ways: it selects the variants automatically based on name conventions, and it provides a scale mechanism. Developers could also make use of their own scale mechanism that adapts resources to match the display's density.

Related Patterns: The parent pattern is MODEL VIEW CONTROLLER which also refers to alternative solutions for client/server systems. Alternatives are the DIALOG FLOW MANAGER [6] and SERVER-SIDE CONTENT ADAPTATION [7] for server-side multi-channel access and the INTERMEDIATE CONTENT ADAPTATION [7] if a proxy is used. The application has to be tested, even during development. The patterns of the MOBILE APPLICATION TEST SUITE show how.

Known uses: Android applications best practice [8]

2.2 Mobile Application Usability Test Suite

Interaction Category/ies: Test and evaluate

Context: A mobile application that runs on a mobile device with local data storage and logic with infrequent access to a server.

There are multiple mobile devices and, due to short innovation cycles, new mobile devices are launched every day.

Scenario: An expert programmer developed the system but had to minimize testing due to time limits. The software delivered was not tested with end users, and crashes often.

Problem: What tests are necessary and how can they be coordinated efficiently in order to save error correction post launch?

Forces:

- Programmers must be aware that what they have not tested they cannot expect to work.
- Testing has to start as early as possible although it is time-consuming.
- Developers cannot foresee all ways that users interact with their application. They designed it to perform tasks in a certain way, but maybe there are other ways not thought of, unknown uses, user slips and mistakes. The automatic test should regard such user behavior.
- Users have to be involved at every design step nevertheless. But it would be time-consuming to test the application on every device with a large number of end-users.
- Involving users can become costly and time-consuming if too many users are consulted. Yet it is not very easy to find the right number of tests.

Solution: Test and review from the beginning and involve users and their data even for automatic tests.

Different emulators should be used as a first test environment: All supported screens (sizes, densities, platform API) should be defined and used for the tests. An emulator for the test should run at a similar physical size of the targeted mobile devices, i.e., the emulator running on a laptop with a density of 96 dpi should scale the emulator display according to the mobile device.

Of course, it is also necessary to conduct a walkthrough by yourself in order to test whether screens are presented correctly.

Involve typical users of your application to test it regularly, e.g., users of a certain age span, at certain locations, with certain interests and background knowledge. The pattern END USER TEST describes how to conduct a simple user analysis, a test with end-users and what can be learned from the data. A good limit for end user testing is to stop when adding more users leads to no new feedback.

Use a test program that sends random UI-events to the application and monitors its behavior. The application should be able to monitor and report application faults, unhandled exceptions and crashes. For example, a UNIT TEST APPLICATION is able to test the application during development and afterwards automatically.

In order to combine the two previous approaches, prepare the UNIT TEST APPLICATION to be used with real user data. The application can send streams of user events recorded earlier using the USAGE DATA RECORDER and known patterns of user slips/mistakes, interaction events or system errors.

Consequences:

[+] Using a Unit Test Application, testing starts during development.

[+] The combination of tests covers several aspects of testing.

[+] Users are involved often and regularly.

[-] Testing is always more time-consuming than not testing.

[-] End user testing is not removed completely, so it still produces considerable effort.

[-] User tests cannot last that long because they might lack concentration after a while. Thus it is not possible to test every task or workflow. Still, the test has to be designed in a way that the most use cases are covered.

[-] Of course, not every bug or usage problem can be found by testing, in process of time, unexpected flaws will appear. Don't be surprised, always count on the unexpected and provide resources for ongoing improvement.

Usability Consequences:

[+] Utility: The more tests, the less unexpected crashes.

Related patterns: UNIT TEST APPLICATION shows how to test the application designed by using the pattern CLIENT-SIDE MULTI-SCREEN SUPPORT during development and afterwards automatically. Real data can be obtained using the USAGE DATA RECORDER and used to test the system again with the UNIT TEST APPLICATION. The pattern END USER TEST describes how to conduct a test with end-users and what can be learned from the data.

Known uses: Android applications best practice [8]. And the UI/Application Exerciser Monkey can be used for stress tests on a device or on the emulator [8].

3. CONCLUSION

We presented two patterns for usability-improving mobile applications development for software engineers about multi-screen support and testing mobile applications. In order to provide more mobile development best practices, our work-in-progress comprises the identification of more patterns and their combination into a pattern language.

4. ACKNOWLEDGMENTS

The authors would like to thank Yishay Mor, Andreas Fießer and the workshop participants very much for their reviews and comments that helped to improve the paper.

5. REFERENCES

[1] Preece, J. and Y. Rogers and H. Sharp: Interaction Design: Beyond Human-Computer Interaction, Wiley 2nd edition, ISBN 978-0470018668.

[2] Bass, L., John, B.E., Kates, J. 2001. *Achieving Usability Through Software Architecture*. Technical Report CMU/SEI-2001-TR-005.

[3] MODPA – Mobile Design Patterns and Architecture, http://www.titu.jyu.fi/modpa/index.htm

[4] Gamma, E., R. Helm, R. Johnsson, J. Vlissides: Design Patterns: Elements of Reusable Object Oriented Software, ISBN: 0-201-63361-2

[5] Biel, B., Grill, T., and Gruhn, V.. 2008. *Patterns of Trust.* Proc. of Mobile MultiMedia 2008, 391-396, ISBN 978-1-60558-269-6

[6] Biel, B. and Gruhn, V.. 2006. *Dialog Flow Manager*. Proc. of EuroPLoP 2006.

[7] Biel, B. and Gruhn, V. 2007. *Content Adaptation.* Proc. of VikingPLoP 2007.

[8] Android Developer Guide, available online, http://developer.android.com/