


Developing Software
for Mobile Devices:
How to Do That Best

Institut für
Computertechnik
ICT
Institute of
Computer Technology

Moderator:
Hermann Kaindl
TU Wien
Institute of Computer Technology

Panelists

- **Roberto Meli** DPO Srl, Italy
- **Rodion Podorozhny** Texas State University, USA
- **Andreas Kurtz** BMW AG / University of Augsburg, Germany
- **Andreas Ibing** TU München, Germany
- **Petre Dini** Concordia University, Canada / China Space Agency Center, China



Institute of Computer Technology

Introduction

Mobile devices vs. traditional computers:

- Different and possibly adaptive mobile user interfaces (UIs)
- Context-aware/context-sensitive mobile applications
- Ubiquitous interactions, e.g., with wearables

Challenges beyond that of traditional software development



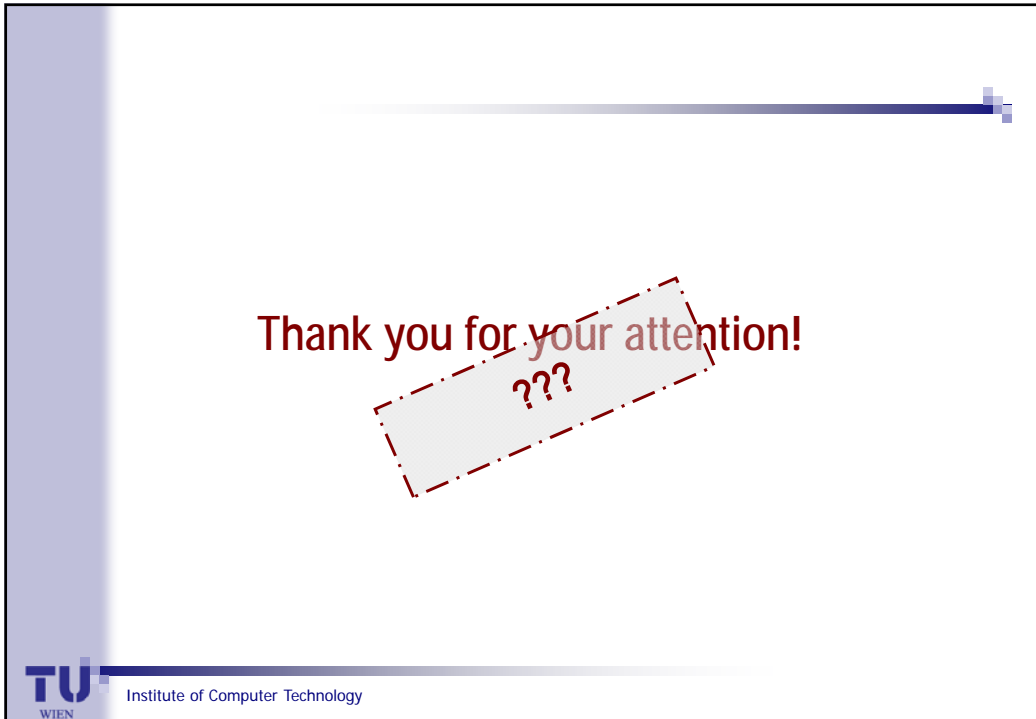
Institute of Computer Technology

Tailored UIs for Smartphones and Tablet Computers

- Relative screen sizes
- Automated generation of Graphical User Interfaces (GUIs)
- Automated tailoring through optimization techniques:
 - A demo flight booking GUI
<http://ucp.ict.tuwien.ac.at/UI/FlightBooking>
 - An accommodation booking GUI
<http://ucp.ict.tuwien.ac.at/UI/accomodationBooking>
- Trade-off between size and mobility



Institute of Computer Technology



Thank you for your attention!

???

TU
WIEN Institute of Computer Technology

APP DEVELOPMENT & MEASUREMENT: ALLIES OR ENEMIES?

Dr. Roberto Meli



Context - 1

Mobile application development and maintenance are often characterized by:

- ❑ small project, sizes and short schedules,
- ❑ a volatile scope,
- ❑ the use of diverse technologies,
- ❑ user interface and user experience relevance
- ❑ multimedia integration
- ❑ geographical information integration
- ❑ social remote and local interaction

Context - 2

These elements require an organizational approach based on:

- ❑ time responsiveness
- ❑ agile or evolutionary processes
- ❑ small and very integrated teams
- ❑ strong user (representative) involvement
- ❑ interdisciplinary skills
- ❑ supportive architectures and tools

None of these elements are “against”
measurement !



Measurement: useful or not ?

- ❑ Corporate context
- ❑ Project oriented development
- ❑ Prioritized and variable resource allocation
- ❑ Internal User driven
- ❑ Project productivity assessment needs
- ❑ Cost control emphasis
- ❑ **Tender / Contract Management**
- ❑ Personal/Team context
- ❑ Service oriented development
- ❑ Fixed resource allocation
- ❑ Market User driven
- ❑ Business Unit productivity assessment needs
- ❑ Time to market emphasis
- ❑ **Informal internal contracts**



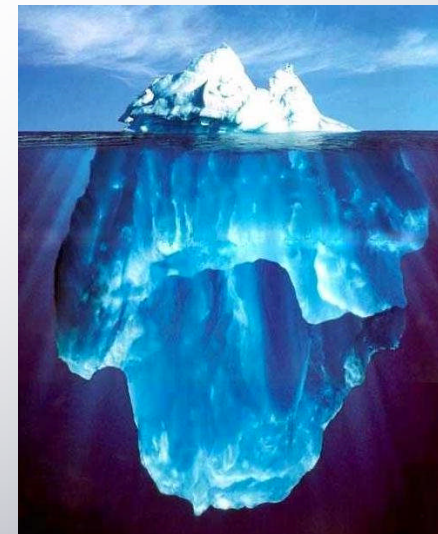
Production teams sometime feel like that...





Estimations ?

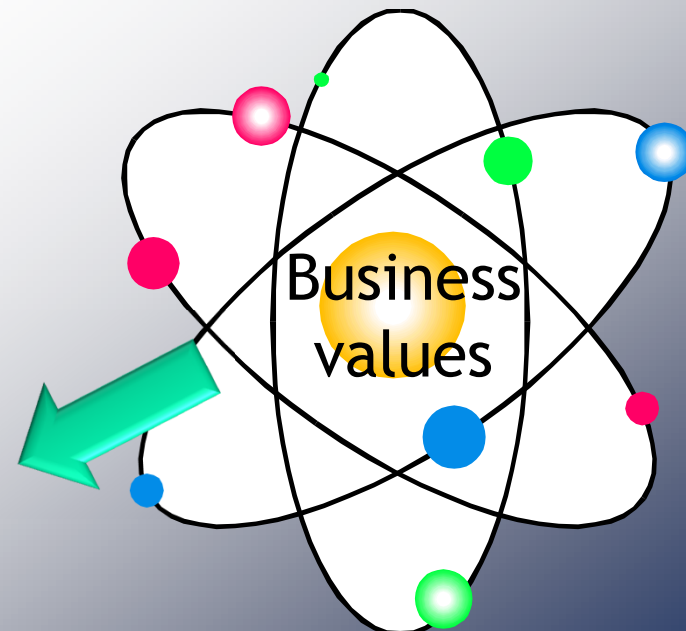
When we consider App development effort, duration and staff estimation, apparently, there is no spread adoption of formal methods. Expert judgment seems to be the most adopted strategy. Unfortunately, the quality of these estimates is dependent on the quality of the estimators and many times it is impossible to compare different situations and to share expertise among different teams.



Measurement's role

- ❑ To help knowing (explicit not implicit)
- ❑ To help estimating resources
- ❑ To help communicating
- ❑ To help demonstrating added value
- ❑ To help benchmarking

- To realize useful systems!
- To do it efficiently !



Which Measurement and Models ?

- ❑ Any adopted measurement model should be:
 - ❑ light
 - ❑ quick
 - ❑ simple
 - ❑ used by developers
 - ❑ complete
 - ❑ standard
 - ❑ product oriented
 - ❑ easy to learn



Integrated
Software
Cost
Model





WWW.IARIA.ORG

PANEL SOFTENG

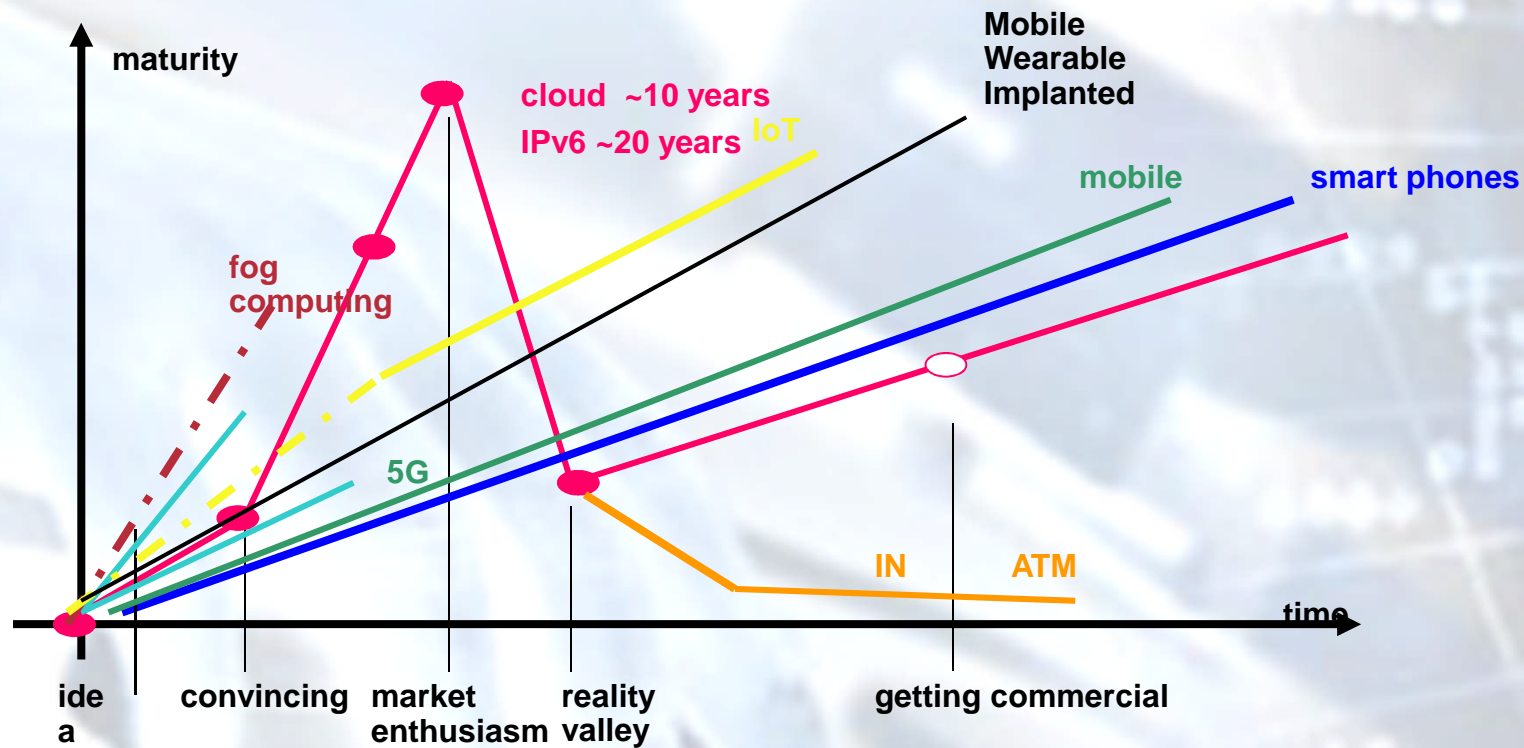
Designing Software for Mobile Devices: How to Do That Best

Petre Dini, Concordia University, Canada | China Space Agency Center, China

From Requirements to Software

- **Centralized systems | hardware vs. software**
- **Distributed systems | hardware vs. software**
- **Real-time systems | embedded software**
- **Mobile systems | systems on the chip**
- **Wearable systems | systems on the chip**
- **Implantable systems | systems on the chip**
- **Body systems | cyberman**

Technology/Maturity Lifecycle



Requirements → Systems → Testing and Validation

Mobile/Wearable/Implantable → Human Behavior/ Body Features

Specifics

- **Standardization and methodologies**

 - Screen

 - API

- **The finance sector is helping, e.g., the introduction of**

 - Apple Pay along with the Apple Watch are current solutions;

 - Even more, payment-capable bracelets are offered by CaixaBank and Barclays.

- **Special considerations**

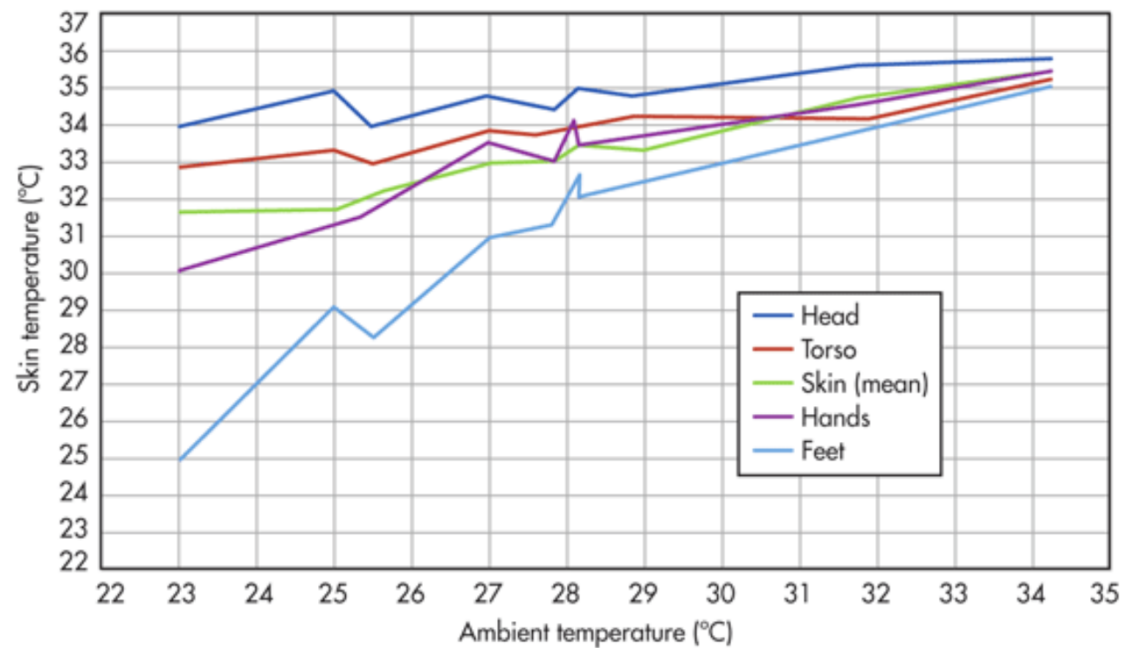
 - Thermal considerations

 - Material and environment

 - Testing

Thermal considerations

- A specific aspect is that wearable devices introduce some unique thermal design challenges that should be considered for devices, Apps and the entire system. This is not only referring to operability, but also to a required comfort level for humans. This design challenge is mainly for processor intensive applications and units with complex displays.



Testing

Battery Life

- Energy and battery-based operation raises special maintenance issues and a real challenge for both mobile (and wearable, ...)

App developers and testers

- These need also suitable testing criteria tuned to the new features of devices and Apps.

Testing for-Real

- As wearable devices are quite specific, simply substituting them with emulators is not suitable; as the discipline is evolving in a rapid pace, trusting the results of such emulator is doubtful. Still, there are a few wearables on the market, e.g., Tizen, Android, etc.

Materials-oriented Testing

- Due to metal migration concerns, biased testing is increasingly important to validate sensitivity in moist environments and to validate the risk of tin whiskers [8].

Testing body-wearable systems

- There is a large variety of wearable devices and Apps, from fitness bands (which are essentially data collectors) to portable heads-up display; additionally, complex interactions occur between the touch display, cameras, and fast data communication with mobile platforms

Cyberman



Challenges

- Different mobile devices need different user interfaces. With regard to screen size, automated GUI generation with automated tailoring may become an option.
- What is specific on designing and testing wearable devices and Apps is that user experience is more relevant than in traditional approaches.
- “It is a challenge to develop and test very specific features; e.g., “smart watches have very small screens and almost no buttons, making the use of space, navigation and user interaction incredibly important””

Thanks

Thanks



WWW.IARIA.ORG

SOFTENG panel

Testing and Runtime Validation

Andreas Ibing
TU München
Chair for IT Security

International Conference on Advances & Trends in Software Engineering
24.02.2016

Software Bugs: Common Software Weaknesses

- Common Vulnerabilities and Exposures (CVE)
 - over 75000 entries (US National Vulnerability Database)
- Common Weakness Enumeration (CWE)
 - bug type classification, CVE entries mapped to CWE entries
 - 33 views, 244 categories, 719 weaknesses, 1004 entries
 - example: 'out-of-bounds write' (CWE-787)
- Common Weakness Scoring System (CWSS)
 - risk metric to prioritize software weaknesses
 - 16 factors in formula
 - e.g. 'acquired privilege' or 'likelihood of exploit'
- current program analysis tools adhere in their reports to this taxonomy

Software Testing, Coverage and Instrumentation

- Test suite: specify input / output vectors
 - detect bugs with unexpected output (including OS output)
 - limited execution coverage
 - control flow (branch coverage; modified condition / decision coverage)
 - data flow coverage; input coverage; thread interleaving coverage
- Improve bug detection with instrumentation (add code for dynamic checks)
 - source instrumentation (e.g. CCured)
 - binary instrumentation
 - dynamic (Pin, Valgrind)
 - examples: Memcheck, Helgrind
 - static / compiler instrumentation
 - examples: AddressSanitizer, ThreadSanitizer

Symbolic Methods

- increase input coverage to arbitrary input
 - symbolic variables and logic backend for path-sensitive bug detection (SMT solver)
 - independent of test suite execution coverage
- automated test case generation
- symbolic execution
 - automatically explore satisfiable program paths
 - allow false negative detections
 - program path prioritization / pruning
 - concretization (partially symbolic)
 - examples: SAGE, KLEE
- abstract interpretation
 - automated generalization; overapproximation of bug-free program paths
 - allow false positive detections
 - examples: Polyspace, Astree

Runtime Validation

- if check algorithm has 'negligible' overhead, can be continuously applied at runtime
- checks at different levels
 - **instrumentation** for runtime checks; examples:
 - stack canaries
 - object size checking
 - checks in **managed** runtime; examples:
 - Java
 - 'Goldilocks', Java virtual machine with race detection
 - **CPU acceleration** (extra hardware); examples:
 - Intel Memory Protection Extensions (MPX)
 - control flow signature checks in smartcard CPUs
 - 'RADISH' CPU proposal for race detection

Arguable Trends ?

- Trend I: symbolic methods gaining importance for testing

- Trend II: from instrumentation for testing to CPU integration for runtime checks
 - Moore's law still working
 - Server/PC innovations diffuse to embedded world
 - (like virtualization, trusted computing...)

References

- R. Martin, S. Barnum, and S. Christey: "Being explicit about security weaknesses", *CrossTalk The Journal of Defense Software Engineering*, 20:4–8, 3 2007.
- G. Necula, J. Condit, M. Harpen, S. McPeak, and W. Weimer: "CCured: Type-Safe Retrofitting of Legacy Software", *ACM Trans. Programming Languages and Systems*, 27(3):477–526, 2005.
- C. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. Reddi, and K. Hazelwood. "Pin: Building customized program analysis tools with dynamic instrumentation", *Conf. Programming Language Design and Implementation, PLDI 2005*, pages 190–200
- N. Nethercote and J. Seward. *Valgrind: "A framework for heavyweight dynamic binary instrumentation"*, *Int. Conf. Programming Language Design and Implementation*, 2007.
- J. Seward and N. Nethercote: "Using Valgrind to detect undefined value errors with bit-precision", *USENIX Annual Technical Conference*, 2005.
- A. Muehlenfeld and F. Wotawa: "Fault detection in multi-threaded C++ server applications", *Int. Workshop Multithreading in Hardware and Software*, 2006
- K. Serebryany, D. Bruening, A. Potapenko, and D. Vyukov: "AddressSanitizer: A fast address sanity checker", *USENIX Annual Technical Conference*, pages 28–28, 2012
- K. Serebryany and T. Iskhodzhanov: "ThreadSanitizer: data race detection in practice", *Workshop on Binary Instrumentation and Applications*, pages 62–71, 2009.
- P. Godefroid, M. Levin, and D. Molnar: "Automated whitebox fuzz testing", *Network and Distributed System Security Symp. (NDSS)*, pages 151–166, 2008.
- C. Cadar, D. Dunbar, and Dawson Engler: "KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs", *USENIX Symp. Operating Systems Design and Implementation (OSDI)*, pages 209–224, 2008.
- D. Kästner, S. Wilhelm, S. Nenova, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, X. Rival: "Astrée: Proving the Absence of Runtime Errors. *Embedded Real Time Software and Systems Congress*", *ERTS 2010*.
- T. Elmas, S. Qadeer, and S. Tasiran: "Goldilocks: a race-aware Jave runtime", *Communications of the ACM*, 53(11):85–92, 2010
- R. Ramakesavan, D. Zimmerman, P. Singaravelu, G. Kuan, B. Vajda, S. Gibbons, and G. Beeraka: "Intel Memory Protection Extensions Enabling Guide, Rev. 0.89", Intel Corporation
- J. Devietti, B. Wood, K. Strauss, L. Ceze, D. Grossman, S. Qadeer: "RADISH: Always-on sound and complete race detection in software and hardware", *Int. Symp. Computer Architecture 2012*