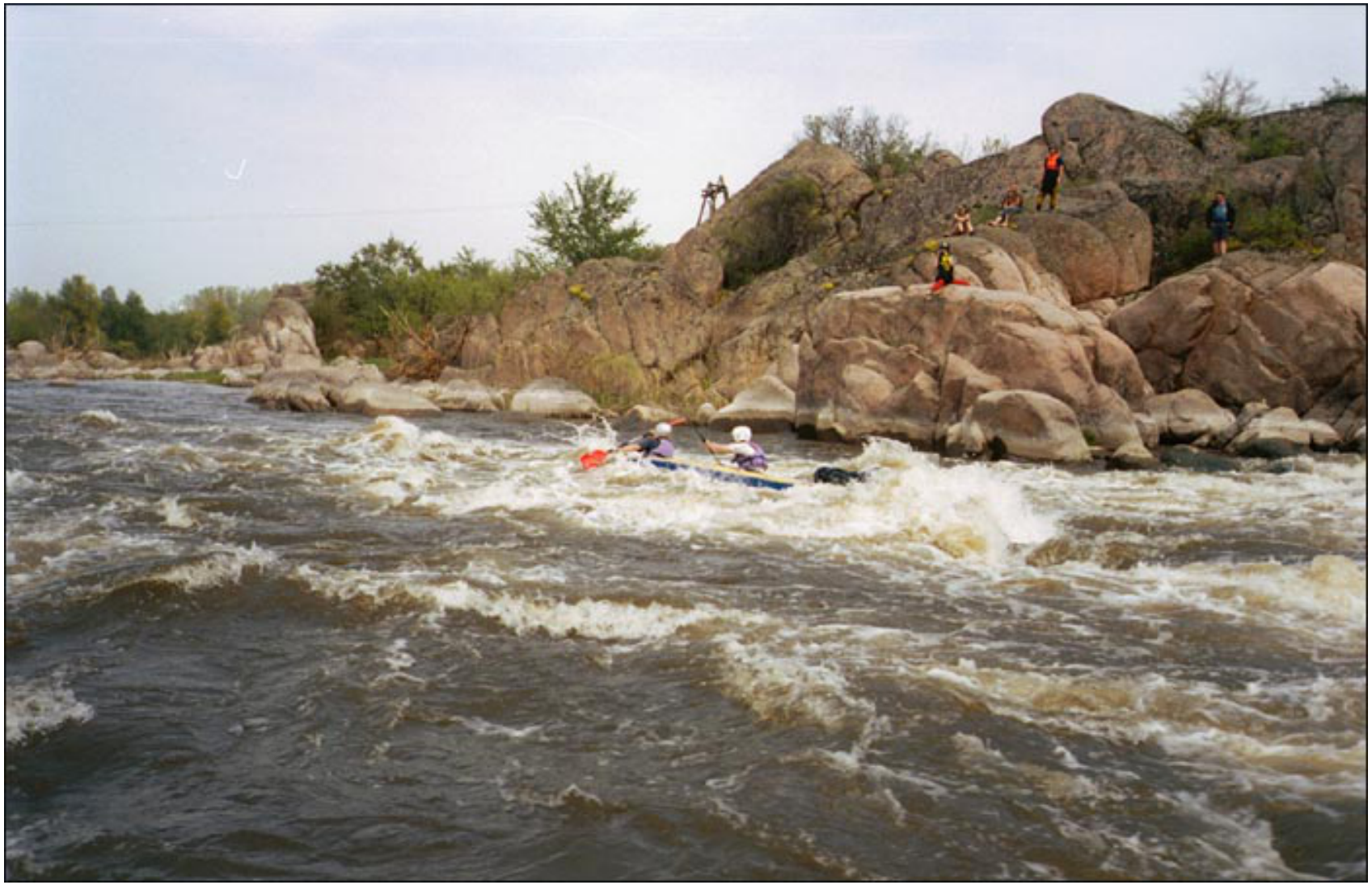
Cross-platform software development in practice. Object-Oriented approach.

Vitaly Repin
Maemo Devices, Nokia

Maemo

March 25, 2010

It's life...



Outline

Abstract

- The basic foundations of your software are changing all the time. The real challenge: *The only constant is change.*
- How to be prepared for the changes? Is it possible to predict the future? Yes, to some extent...
- Don't be afraid! There are technologies around which will help you. But *there is no silver bullet.* Use your brain and common sense to win!
- Qt and C++ are your friends. But use them wisely. *THERE IS NO SILVER BULLET.*

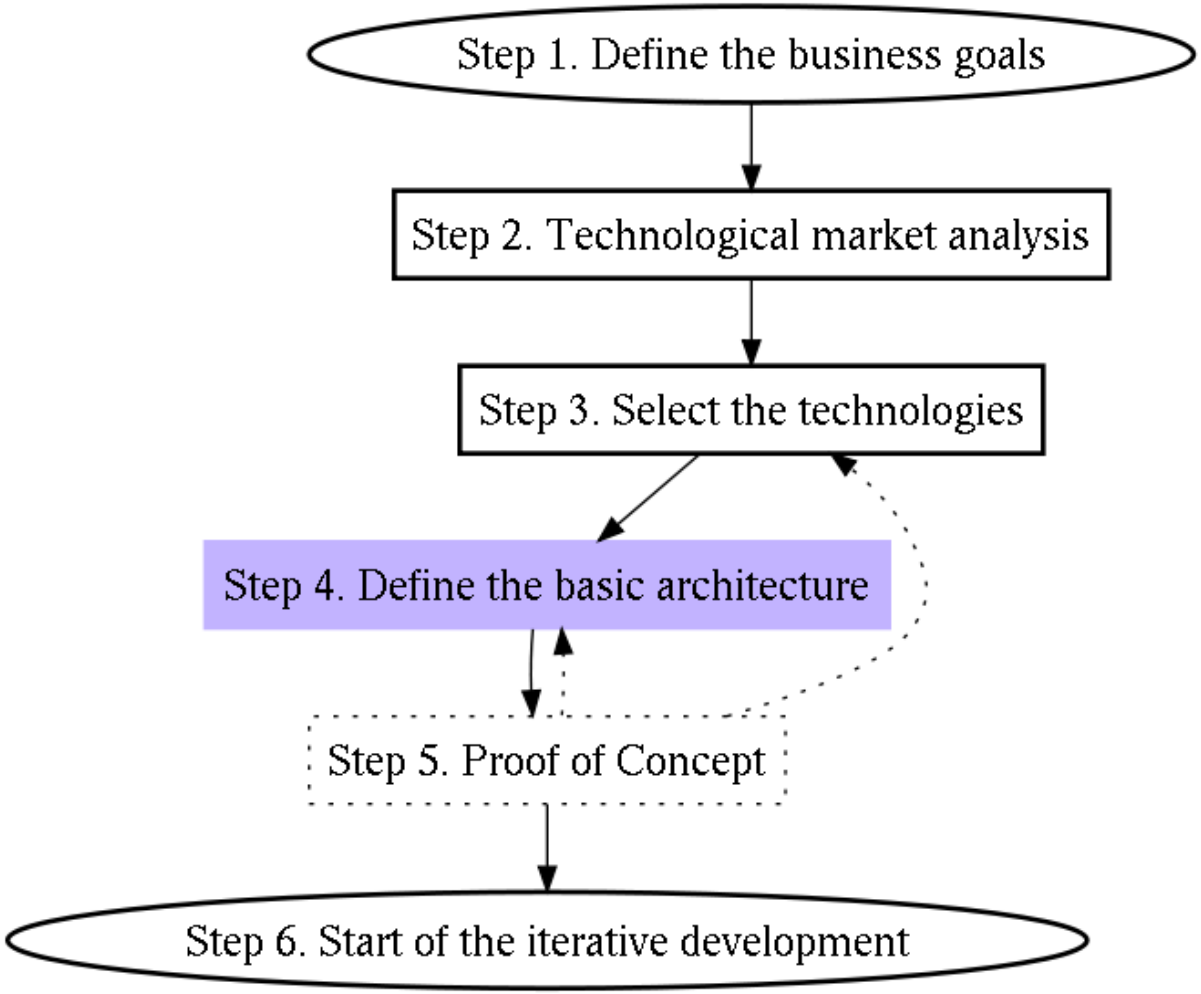
What are they doing there?

Why cross-platform?

- The basic foundations of your software are changing all the time. The real challenge: *The only constant is change.* *Yes, Diogenes Laertius is very actual in the XXI century!*
- € *Money talks:* it is all about the cost of application maintenance and development of the new features

OK. Let's hope these boys know where they are...

The typical steps



Too far view can be misleading. But in the same time the distance is needed to see the “big” picture

Technological landscape analysis

This phase is extremely important (and extremely difficult and risky) if your aim is to make an application which is prepared for the FUTURE. And this is the usual story in the modern word...

- What technologies are used?
- What APIs are used?
- What is going to be changed in the forecastable future? E.g., UI tends to change more often then backends. Typically. But everything depends on the project!
- Try to be Cassandra for the destiny of the technologies and APIs your application uses. What is their future? Use all the available information - insider's knowledge, expert's opinions, keep an eye on the researching projects. You shall be aware of the university activities in order to be prepared for the technology switch.

Try to forecast. To predict the future.

Cross-platform enablers

- Divide the (future) application into the packages.
- Put the portability requirements towards every package. E.g., “package A shall be portable to every system which has gcc and glibc”, “package B shall be portable to every system which has Qt core libraries”, “package C will be reimplemented for every system. APIs shall be carefuly designed”.
- Keep in mind - stricter portability requirements mean more efforts to implement specific component.

The risk — to make the architecture more complex than needed. To reinvent glibc.

Step 4. Define the basic architecture

- Isolate the dependencies through internal APIs (set of classes)
- Document in the paper the most critical architectural decisions (including portability requirements towards each component)
- Agree on the basic architecture with your team and other experts you can reach. This is important step as it allows to find teh design problems in the very early phase. Even before the development starts! Never sit in the "architectural ivory tower"!
- € If needed, make a prototype. It shall be FAST prototyping. It can not take "1 man month". But it depends on the project as always!

Step 6. Start of the (iterative) application development

- Start the development based on the architecture agreed with the team
- **Architecture is not a dogma.** It could be changed if the development shows that it does not suit the business needs
- Regulary check that your application is really portable. Just build it for 2 platforms at least. And run the unit tests.

Don't be afraid! Just do!

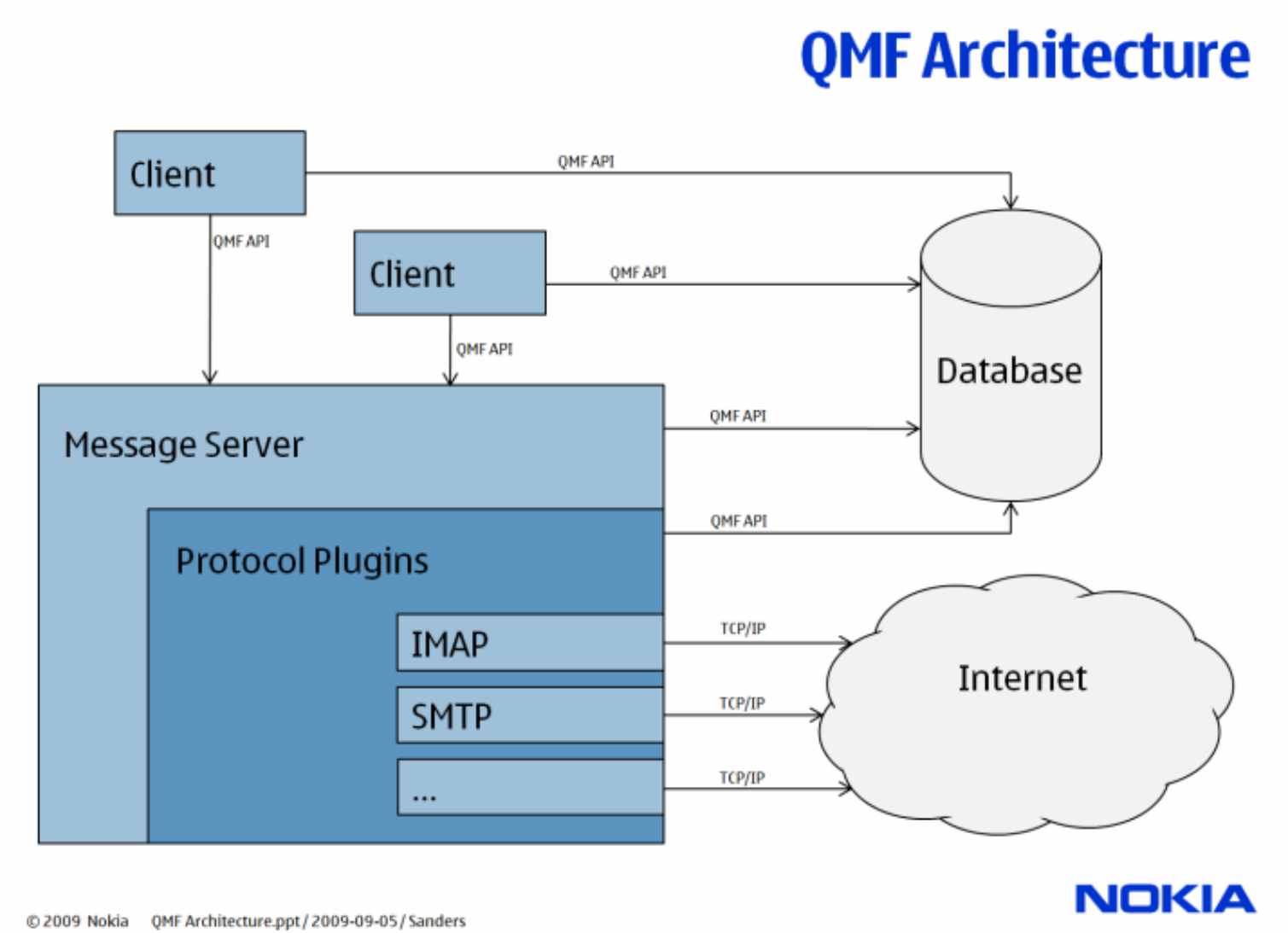
Qt Messaging framework. Introduction.

- Cross-platform (Linux, Mac™, Maemo™, MS Windows™) C++ middleware to build email clients, and more generally software that interacts with messaging servers.
- Based on Qt
- Platform-dependent stuff is isolated in the corresponding classes which can have different implementation for different platforms.
- Extensible by protocol and content storage plugins. E.g., plugin to access your messages box in social network, can be easily developed
- Open-sourced: http://qt.nokia.com/doc/status/qdoc-output/public-messagingframework/html/

Technologies

- APIs: networking (including SSL), SQL (sqlite), file access, synchronization primitives. Use through Qt classes whenever it is possible.
- Documentation system: qdoc3 (as in Qt)
- Portability: all the major desktop platforms. Designed with being cross-platform in mind. Shall be easily ported to future computer platforms.

Define the basic architecture



Define the basic architecture

- Run-time executable: messageserver
- Mechanism to support 3-party protocols: plugins loadable into the messageserver in runtime
- Clear separation between UI and backend
- Use the C++ PIMPL idiom to encapsulate the platform-dependent stuff. Make porting efforts easier for the maintainers
- Use the Model/view paradigm to separate data and representation

Iterative application development

- The architecture was constantly changing with the project evolution but the major architecture ideas are still valid
- New capabilities were added to the plugins APIs during the QMF evolution
- It was possible to port QMF to desktop and mobile platforms with different software architectures.

Remember: THE ONLY CONSTANT IS CHANGE!

Thanks a lot for your attention! Any questions?

vitaly.repin@nokia.com