Marco Piccolino

Smart UI development with QML and Qt Quick: lessons learned & best practices

http://marcopiccolino.eu/dev

In the last episodes...

QtDay 2016 - **Design Systems & Pattern Libraries**

- Design system: improve UI consistency
- Pattern library (aka style guide): living UI components doc

QtDay 2017 & QtWS 2017 - **Clean Architecture & BDD for Qt apps**

- Behaviour-Driven Development: runnable specifications
- Clean Architecture: testable and maintainable business logic

Today

QML + UI development tips & tricks

Focus on QML UIs (& Qt Quick).

- Commonly encountered code structure issues
- Solution set 1: Responsive & modular UIs
- Solution set 2: UI meets Business Logic
- Practical project structure (Demo time)

Commonly encountered code structure issues

The Spaghetti UI

I'm just a poor developer, don't ask me how it works!

Signal & S... ignals

With nested components, signal propagation can get out of hand

Component boundaries and APIs

Component APIs? In QML? Never thought about those!

I would use `qmlscene`...

...but C++ is in the way

Solution set 1: Responsive & modular UIs

Not all UI components are created equal

Hierarchy and layers in the UI

You know a component by the company it keeps

Style guides revisited

Additive methods

aka mobile first

Solution set 2: UI meets Business Logic

One layer to rule them all

The ControllerView pattern

Clean Architecture for Qt apps

The client becomes a remote client

Practical project structure (demo time)

Let QMake help with project complexity

Use `.pri`'s and sub-projects

Namespace UI components, package them into modules

Give a coherent file structure to help Resource System and Creator

Create a living styleguide along with the client app

See how your UI components behave in a different context

Commonly encountered code structure issues

The Spaghetti UI/1

Problem

I am not able to describe the structure of my UI

Indicators

- big fat `main.qml`
- sloppy `id`'s (only assigned for property bindings)
- can't name (or describe) the components that make up the UI

The Spaghetti UI/2

Problem

I am not able to describe the structure of my UI

Indicators (cont'd)

- UI refactoring feels "dangerous"
- behaviour of UI "components" in a slightly different context is unknown
- interspersed business logic entry points (or even implementations)

Signal & S... ignals

Problem

When creating multiple QML types, need many "bridge" signals

Indicators

- signals with the same name in components at different levels
- affected code feels redundant & harder to maintain

Component boundaries and APIs

Problem

Can't list properties, signals & functions that my components expose to client code

Indicators

- refactoring is a problem
- never gave a thought about the programming interface of my components
- rely too much on stock types (QtQuick, Controls)
- add properties to items in a liberal manner (e.g. to store values)

I would use `qmlscene`, but I can't

Problem

I would use `qmlscene` to prototype, but I can't because of my code structure

Indicators

- get errors about unknown object instances from C++ when launching `qmlscene`
- get errors about missing QML modules when launching `qmlscene`
- slower dev cycle, possibly harder refactoring

Solution set 1

Responsive & modular UIs

Not all UI components are created equal

If a design system is already in place, need to follow it. Otherwise, you need one.

- What are the building blocks of the design language?
  Colors, typography, iconography, etc.
- What are the generic and recurring UI components?
  Buttons, fields, list items, etc.
- What are the specific components that support Business Entities and Business Rules?
  Lists of entity instances, etc.
- What are the components that "glue together" other components into views?
  Page templates, etc.

You know a component by the company it keeps

- Need to know expected behaviour of a UI component in advance
- Before embedding in final view's context, think about it
  - in isolation (`qmlscene`)
  - in a style guide (a simple QtQuick application),
    preferably placed with a different positioning model than in app
    (layouts vs. anchors vs. positioners vs. x/y)

Additive methods

- If cross-device and cross-form-factor is likely, responsiveness becomes a necessity
- better adding than subtracting
- modularity (defining components as mentioned above) is key to success
- Style guides provide responsiveness analysis almost for free


Solution set 1 demo:
style guide (pattern library)

Solution set 2

UI meets Business Logic

One layer to rule them all

Make sure there is only one (max 2) application layer that acts as a contact point between UI and Business logic.

- signals help you avoid coupling and have a well-defined information flow
- property aliases make targeting components easier
- if you know Flux/React, it's the Controller View pattern
- applies to Business Logic but also to routing (navigation between views)

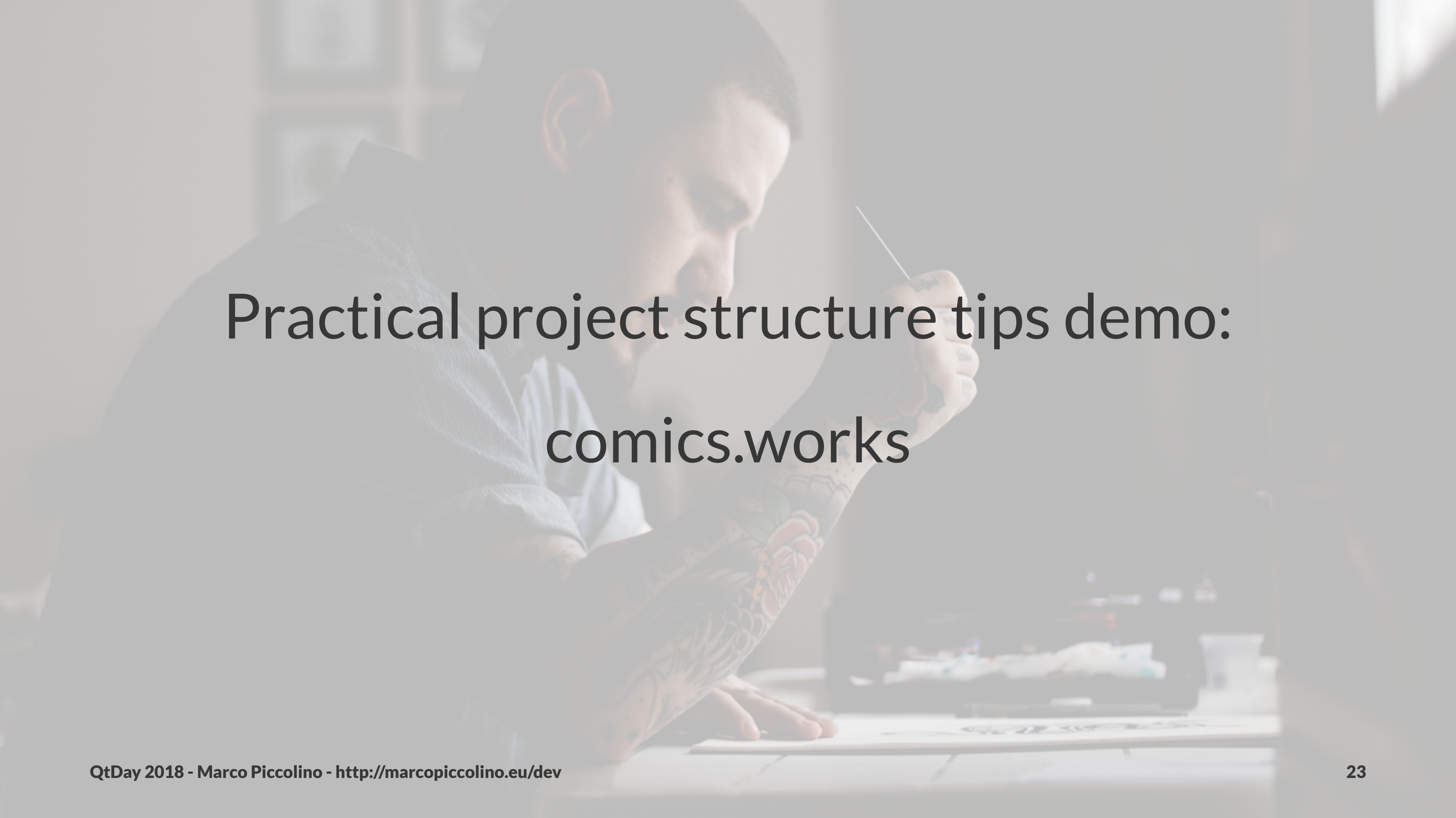
Clean Architecture for Qt apps

Clean Architecture (uncle Bob Martin) helps organising application layers. For UI-Business Logic interaction, it encourages message passing (see Daniele's presentation tomorrow).

- Messages contain all information that is needed by the UI
- Messages contain simple value types, not references/pointers
- Feed Qt's smart models with this data
- To future-proof your app, view-aware models should better stay in a presentation layer below the UI (tip: Cutehack's `JsonListModel`)

Solution set 2 demo:

UI / Business Logic contact points



Practical project structure tips demo:

comics.works

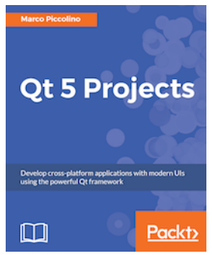
Resources

Personal page

http://marcopiccolino.eu/dev/

Qt 5 Projects book

http://marcopiccolino.eu/dev/qt5projects.html



JSON Model for QML (by Cutehacks)

github.com/Cutehacks/gel

Hopefully one or more blog posts soon...

Thank you!