Functional Programming Invades Architecture

George Fairbanks
SATURN 2017
3 May 2017
Yesterday:
Functional Programming is PITS, i.e., “just inside modules”

Today:
FP is also PITL

Why FP now?

Problems grew bigger
● FP good for parallelism and concurrency
● Systems run on many computers

Coping strategy for complexity
● As systems get bigger but our brains stay the same size we invent tech to cope
● Statelessness, immutability, and pure functions make it easier to reason about how a system behaves

We have more money than sense
● 100 cloud machines is cheaper than one junior developer
● So why not:
  ○ Continuous Integration with every source code commit?
  ○ Server farm running different versions of your code?
  ○ Burn RAM with immutable data structures?
Current perspective on old ideas

- FP increasingly popular in PITS
  - Lambdas and streams in Java, JS, Python, Ruby, ...
  - Renewed interest in pure FP languages

- We can intersect two domains
  - Architecture and FP
  - Perspective on existing patterns

Group exercise
Cube Composer
Get ready: RxMarbles

We won’t use this until later, but let’s get it now.

Laptop: [www.rxmarbles.com](http://www.rxmarbles.com)


Ooh, shiny toy! But please wait for it...
Cube Composer game

Goals
● Get a feel for functional transformation
● Feel awkward, like CS 101
● Sense that you will get better at this
● Sense of “rightness” from a different reasoning style

http://david-peter.de/cube-composer/
Boring slides about an interesting topic

Big Ideas in FP
Function composition

- Build programs by combining small functions
  
  $g(f(x))$ or $f(x) \rightarrow g(x)$

- Seen in pipelines, event-based systems, machine learning systems, reactive
  
  `ls | grep "foo" | uniq | sort`

Note: We’re just covering the FP ideas that seem relevant to architecture
Pure functions, no side effects

- Calling function with same params always yields same answer

- So: Reasoning about the outcome is easier

```
curl http://localhost/numberAfter/5 → [always 6]
curl http://localhost/customer/5/v1 → [always v1 of customer]
```

vs

```
curl http://localhost/customer/5 → [may be different]
```
Statelessness and Immutability

**Statelessness**
- If there’s no state:
  - Easy to reason about
  - All instances are equivalent

**Immutability**
- If you have state, but it never changes:
  - Easy to reason about
  - Concurrent access is safe
Idempotence

- Idempotence: get the same answer regardless of how many times you do it
  
  $\text{resizeTo100px(image)} \text{ vs } \text{shrinkByHalf(image)}$

- Often hard to guarantee something is done exactly once

- Eg: Is the task stalled or failed?
  - Go ahead and schedule it again without fear that you’ll get a duplicate result
Declarative, not imperative

- Functional programming
  - Define what something *is*
    - ... or how it relates to other things
- Versus
  - Series of operations that yield desired state
- Definition, not procedure
- Example: how much paint do I need?
  - `while(!done) { fence.paint(); }`
  - Vs function parameterized by length and width
More boring slides about an interesting topic

Reactive Programming
Reactive programming

- Operations on time-series event streams
- Reactive = Observer Pattern + onError and onCompletion
- Inband vs out of band signaling

Streams and operators

- Event streams

- Rich set of stream operators
  - Transform streams into streams

- Example: Detect double clicks
  - Input: user click stream
  - Transform: group nearby clicks
  - Transform: count group size
  - Transform: drop size-1 groups
  - Output: double click stream

Uni-directional User Interfaces

Model-View-Controller and siblings
- Mutable state
- Problems with async calls to server

Reactive, Uni-directional UI pattern
- One-way stream transformation
- Human action → Transform → HTML

Group exercise
RxMarbles
Filtering

- first, last

- filter

- take, takeUntil

Diagram source: Andre Staltz, rxmarbles.com
Transforming

- **merge**

- **map**

Diagram source: Andre Staltz, rxmarbles.com
Calling a server

- Stream 1: User clicks
- Stream 2: RPC requests
- Stream 3: RPC responses
- Stream 4: UI data

map(click, rpcRequest)

<send RPC request>

map(rpcResponse, html)
Group exercise

- Situation: Your UI needs an authentication token

- Challenge 1:
  - Build a stateless UI (ie no field holding the token)
  - Can you use a (transformed) stream of events?

- Challenge 2:
  - Tokens that expire and must be refreshed
Even more boring slides about an interesting topic

Functional ideas in architecture
Client-side

- Reactive patterns and frameworks in UI
  - React, Elm (and The Elm Architecture), CycleJS, Flux, Redux

- (So-called) “serverless”
  - App composes domain-neutral remote infra services


Server-side

- Stateless middle tier servers, pure functions (AWS lambda)
- Immutable state
  - Resource versioning (not mutation), eg with REST
- (So-called) “serverless” pure functions
  - On-demand deployment, no stable-identity “business logic” servers
- Reactive services, event queues
  - Services: Transform input stream to output stream
  - Events/Messages: Routed to services
Persistence

- Append-only datastores
- Event sourcing
- Command Query Response Segregation (CQRS)
Batch sequential & pipeline

- Big data processing (Hadoop, Spark)
- Map-Reduce
- Tensorflow and other machine learning graph-based functional transformation
DevOps infrastructure

- Version control everything (append-only)
  - Including the scripts that test and deploy code

- So-called “immutable” and “idempotent” infrastructure
  - Never modify a running service
  - Redeploy to change config (no mutation)
Group exercise

Architecture using FP
Design a library: non-FP

- Problem: Library (browse, checkout, checkin)

- Design 1 (do this together):
  - Styles used: Client-Server, 3-tier, Repository (Relational DB)

- Outputs:
  - Runtime view
  - Message sequence diagram
  - DB tables
Design a Library: Functional Ideas

- Design this in groups of 3-5

- Try using some of:
  - Reactive client
  - Reactive server
  - Stateless server functions
  - Event sourcing or append-only database

- Hints
  - Can you define what is on the screen or DB as “All the <things> such that…”
  - Are there places (eg tiers) you can eliminate state?
The end of the boring slides

Conclusion
Conclusion

- FP not just PITS
- Many FP ideas in PITL
  - Function composition
  - Pure functions
  - Statelessness / Immutability
  - Idempotence
  - Declarativeness
- Architecture advantages, especially for large distributed systems

- Big changes in UI / client
  - (mutable) MVC → Uni-directional
  - Object graphs → stream transforms

- Wanted: FP tactics catalog
  - Why use Event Queues?
  - Or Append-only datastores?
  - Or stateless servers?
Interesting links

http://www.eugenkiss.com/b/overview-of-reactive-gui-programming/


