Who is @thoughtworks
Who is @patkua

#architect  #author
#developer  #facilitator
#leader     #lifelong-learner
#coach      #speaker

tiny.cc/twtl  tiny.cc/retros
Evolutionary Architecture

Rebecca Parsons
(ThoughtWorks CTO)

Patrick Kua
(Technical Principal)

Neal Ford
(Meme Wrangler)

Photos by Martin Fowler: http://martinfowler.com/albums/ThoughtWorkers/
If you haven’t done so, please introduce yourself the people around you - you will be working in pairs

• Name
• Role/Title
• Company/Institution
• Background
What is one thing you want to get out of this talk?

COLLECT THESE ON FLIPCHARTS
EVOLUTION
CHANGE

... is inevitable
Technical

Programming languages
Libraries
Frameworks
Tools
Operating environments
Technical constraints
Domain

Revenue models
Base technology adoption
Competitors
Customer needs
Markets
Products
CHANGE

... is inevitable
If CHANGE... is inevitable then
CASE STUDY
Customer case study
WHAT IF...

We architected a system specifically for change?
DEFINITION

Our current working version
An evolutionary architecture supports continual and incremental change as a first principle along multiple dimensions.
But our architecture already supports change!
or does it?
Example Architectural Patterns

Big Ball of Mud

Layered Architecture

Microkernel

Microservices
How do each of these architectures support change (Technical + Domain)

- **Big Ball of Mud**
- **Layered Architecture**
- **Microkernel**
- **Microservices**
Big ball of mud

coupling connections

classes

DIMENSIONS: 0
## Layered architecture

<table>
<thead>
<tr>
<th>Layer</th>
<th>Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Presentation</td>
<td>COMPONENT COMPONENT COMPONENT</td>
</tr>
<tr>
<td>Business</td>
<td>COMPONENT COMPONENT COMPONENT</td>
</tr>
<tr>
<td>Persistence</td>
<td>COMPONENT COMPONENT COMPONENT</td>
</tr>
<tr>
<td>Database</td>
<td></td>
</tr>
</tbody>
</table>

**Dimensions:** 1
Layered architecture

request

PRESENTATION
COMPONENT
COMPONENT
COMPONENT

BUSINESS
COMPONENT
COMPONENT
COMPONENT

PERSISTENCE
COMPONENT
COMPONENT
COMPONENT

DATABASE

DIMENSIONS: 1
Layered architecture

- Presentation
- Business
- Service
- Persistence
- Database

Dimensions: 1
Layered architecture
Microkernel

CORE SYSTEM

PLUGIN PLUGIN PLUGIN PLUGIN

DIMENSIONS: 1
Microkernel

Maven

eclipse

Mozilla Firefox
Layered architecture

PRESENTATION

BUSINESS

PERSISTENCE

DATABASE

DOMAIN DIMENSIONS: 0
DEFINITION

An evolutionary architecture supports continual and incremental change as a first principle along multiple dimensions.
Technical Domain

- Does not dictate schedule
- Supports fast feedback
- Appropriate coupling
- Iterative

Matches business capabilities
- Enables experimentation
- Decentralised governance
- Fitness function
Fitness function
Fitness function

**IMPORTANT**
- Low response time
- Mobile responsive
- Internationalisation & Localisation

**UNIMPORTANT**
- Large # of users
- Availability
- Heavy legal compliance
- Monitoring
- Strong audit trail

NFRs
CFRs
Quality Attributes
Fitness function

<table>
<thead>
<tr>
<th>IMPORTANT</th>
<th>UNIMPORTANT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large # of users</td>
<td>Strong audit trail</td>
</tr>
<tr>
<td>Low response time</td>
<td>Internationalisation &amp; Localisation</td>
</tr>
<tr>
<td>Availability</td>
<td>Heavy legal compliance</td>
</tr>
<tr>
<td>Mobile responsive</td>
<td>Monitoring</td>
</tr>
</tbody>
</table>
Fitness function

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Tests</th>
</tr>
</thead>
</table>

There are known knowns
There are known knowns
There are known unknowns
But there are also unknown unknowns

- Donald Rumsfeld
Generations

6 months
3 months
1 month
daily?
Generations = Cycle time

Time taken to get a single change into production repeatedly reliably
Generations = Cycle time
Conway’s Law

“Organisations which design systems ... are constrained to produce designs which are copies of the communication structures of these organisations”

- Melvin Conway, 1968

en.wikipedia.org/wiki/Conway%27s_law
Conway’s Law Side Effect

UI Specialists

Middleware Specialists

DBAs
Monolith’s vs Microservices
user interface

server-side

DBA
Inverse conway manoeuvre

Orders

Catalog

Shipping
Inverse conway manoeuvre

cross-functional teams...

...organised around business capabilities

Because Conway’s Law!
DDD’s “bounded context”

...physically realised
Products, not projects

projects:

products: 

amazon.com's "You build it, you run it"
Last responsible moment

- complexity
- time
Last responsible moment
Last responsible moment
Last responsible moment

Ports and Adapters

Domain

Adapters
Last responsible moment
Last responsible moment

this is not an excuse to abstract all the things!
Sense and probe
Last responsible moment

Architectural Spikes
Bring the pain forward
Bring the pain forward

continuous integration

deployment pipelines

database migrations/
refactoring

automation
Principle driven architecture

over
PUTTING IT INTO PRACTICE
Architecture is abstract until operationalised

nealford.com/memeagora/2015/03/30/architecture_is_abstract_until_operationalized.html
Evolving your architecture

Architect → Develop → Release
Evolving your architecture

Architect

Develop

Release
Evolving your architecture

Architect

Reflect

Develop

Release
Evolving your architecture

Cycle time = constraint
ENABLING CHANGE

Foster architectural thinking
Foster architectural thinking with ARCHITECTURAL BRIEFINGS
ARCHITECTURAL BRIEFINGS
Design decision
Tool
Implementation
ARCHITECTURAL BRIEFINGS
ARCHITECTURAL BRIEFINGS
ARCHITECTURAL BRIEFINGS
ARCHITECTURAL BRIEFINGS
ARCHITECTURAL BRIEFINGS
Everyone becomes an Architect
Think like a town planner
<table>
<thead>
<tr>
<th>Development practices that help</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuous Delivery</td>
<td>Cross Functional Teams</td>
</tr>
<tr>
<td>Early identification of fitness functions</td>
<td>Architectural briefings</td>
</tr>
<tr>
<td>Spikes</td>
<td></td>
</tr>
<tr>
<td>Review fitness functions</td>
<td>Tracer bullet deployments</td>
</tr>
<tr>
<td>Feature Toggles</td>
<td>Branch by abstraction</td>
</tr>
</tbody>
</table>
CHOOSING STYLES
Build and/or Buy
Build and/or Buy

Functionality

Custom code
Libraries
Frameworks
COTS or Software Products

Ability to change
Strategic Commodity

Need for rapid change

Value generating

Experimental

Support

Commodity

Low

High
Things that prevent change

Coupling

Cohesion

Slow feedback cycles
Cohesion

Functional
Sequential
Informational
Procedural
Temporal
Logical
Coincidental
TRAPS
TRAPS: EXUBERANT COUPLING
TRAPS: PRODUCT CUSTOMISATION
TRAPS: INTEGRATION AT THE DB LAYER
TRAPS: INTEGRATION AT THE DB LAYER
TRAPS: SOA VIA THE ESB ROUTE

Hidden Coupling

Smart Endpoints
Dumb Pipes
DEFINITION

An evolutionary architecture supports continual and incremental change as a first principle along multiple dimensions.
TO CONSIDER

Architectural choices
Decision making process + thinking
Organisational and cultural elements
THANK YOU

@patkua