Eoin Woods &
Nick Rozanski

ECSA 2010, Copenhagen, August 2010

UNIFYING SOFTWARE ARCHITECTURE WITH ITS IMPLEMENTATION

CONTENT

- Characterising the Problem
  - motivation for the challenge
- Existing Research and Practice
- Connecting Architecture and Implementation
  - existing approaches
  - possible avenues for investigation
- The Benefits for Research and Practice
- Summary

CHARACTERISING THE PROBLEM

LOSING THE DESIGN IN THE CODE

- Good designers talk about structures that aren't part of most programming languages
  - layers, adapters, proxies, components, brokers, …
- Yet the code is all that exists at runtime
  - or indeed at all in many cases
- But if we've only got code where does all that design information go?

PROBLEMS WITH CODE AS DESIGN

- Only language structures are present
  - language structures are fine grained
- Fine grained nature is very detailed
  - difficult to discern structure from the details
- Design structures don’t exist
  - need to discern design structure from the code
  - a lot of mapping and assumption required
  - relies on well structured and named code

STRENGTHS OF CODE AS DESIGN

- Interpretation by machine
  - searching
  - checking
  - querying
- Certainty with respect to runtime
  - “the code doesn’t lie”
- Integration of different abstraction levels
  - see where a line fits in the overall design
- People actually write code!

USING CODE AS OUR DESIGN NOTATION

- Need to extend our notion of “code”
- Extend languages to allow design elements
  - Sapir-Whorf – extend language, extend thinking
- One approach is to create new languages
  - e.g. aspect based languages, Arch Java, ...
- Alternatively use extension mechanisms
  - comments, annotations, packages, aspects, ...
  - supplementary notations adding design information

EXISTING RESEARCH AND PRACTICE

THE PRACTITIONER APPROACH

- Naming and conventions
  - classes, packages, files, namespaces, …
  - build systems and binary modules
- Component models
  - Spring, OSGi, EJB3, SCA, …
- Metadata
  - .NET attributes / Java annotations / Doxygen comments
- Commenting
  - structured or informal commenting conventions
- AOP
  - separate aspects of the design into different code modules

THE RESEARCH APPROACH

Disclaimer: I'm not a researcher

- Model driven development
  - generate the code from the design
  - something of a niche, some practical concerns, high commitment
  - doesn't necessarily address the problem
- Recovery of architecture from code
  - inevitably struggles because the information is missing, not hidden
- Architectural description languages
  - usually single dimensional, little direct link to implementation
  - little industrial acceptance (rarely address industrial concerns)
- Programming language extensions
  - approaches like Arch Java show potential
- Design decision capture
  - promising work, but solving a different problem

TOWARDS A CONNECTION BETWEEN ARCHITECTURE AND IMPLEMENTATION

CURRENT SITUATION

- Currently another research / practice disconnect
  - research and practice diverging rather than converging
- Yet there are signs of the need for solutions
  - industrial module systems (Spring, OSGi, …)
  - dependency capture and analysis tools
    - annotations, dependency tools (Structure 101, Lattix, …)
  - design visualisation and comprehension tools
    - Panopticode, Code City, Citylyzer, X-Ray, DrArch, …
  - rules based open source utilities
    - Architecture Rules, Pattern Test, Japan, …
- Great opportunity for research/practice collaboration

LIKELY REQUIREMENTS FOR SUCCESS

- Accessible, well-defined, embeddable notation
- Describe what designers design
- Multi-technology applicability
- Extensible
- Focus on functional core of systems
- Tools to make it easy to write and use
- Modular and layered
- Machine processable
- Freely available ideally open source
  - without patents or difficult licenses

POTENTIAL AVENUES FOR INVESTIGATION

- Extending programming languages
  - e.g. Arch Java and similar
- Augmenting with additional design descriptions
  - e.g. a Groovy DSL to describe system structures
  - annotations to indicate roles and relationships
- Aspects
  - can they be used beyond logging and security?
- Visualising more than metrics

EXTENDING LANGUAGES

```
public component class PricingEngine {
   protected owned MarketDataSource marketData = ... ;
   protected owned StressTester tester = ... ;
   protected owned PricingPolicyStore policyStore = ... ;

   connect pattern MarketData.priceService, StressTester.setPrices ;

   public GraphicsPipeline() {
       connect(marketData.priceService, tester.setPrices);
   }

   public BigDecimal addSecurityToPricingSet(String ticker) {
      marketData.addSubscription(ticker) ;
      this.addSecurityOfInterest(ticker)
   }
   ...
}
```

This example is based on Arch Java syntax
http://archjava.fluid.cs.cmu.edu

ADDING METADATA

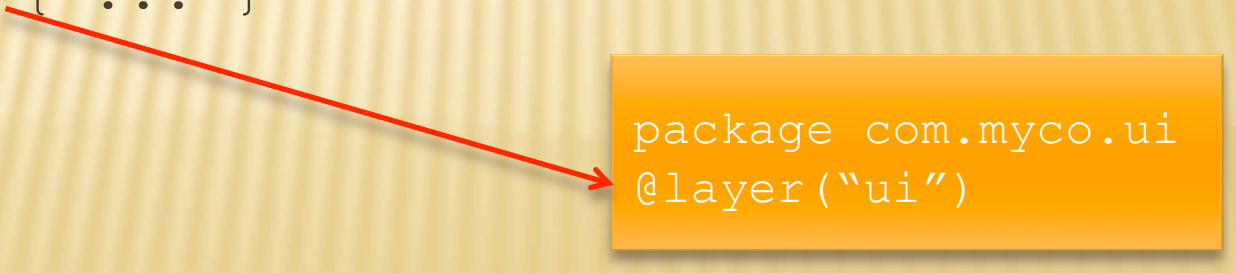
- For example, possible Java annotations
  - @FunctionalElement / @FunctionalInterface
  - @Layer
  - @Tier
  - @PatternElement<T extends Pattern>(T.Role r)
  - @Requires / @DependsOn
- This is only part of the story though
  - need to tie pieces together across technologies & tiers
  - probably need an overall description (e.g. via DSL)
  - overall description would reference annotated elements

EXTERNAL DESCRIPTION LANGUAGES

- Create ADLs that live in and reference the code
  - part of the system source code, not separate design artefact
  - “compile” the ADL as part of the build
- Create languages in common technologies
  - e.g. XML / YAML or Groovy / Scala / F#
- Languages to tie the pieces of the system together
  - overall structure, reference code elements to define the system elements in the ADL description
- Vocabulary taken from common design elements
  - layer, component, message transport, tier, client, server, database
- Allow them to be part of the running system
  - e.g. expose ADL elements as MBeans in the JVM

EXTERNAL DESCRIPTION LANGUAGES

```
system "fixed_income_support" {
    tiers {
        client, server, cache, persistence
    }
    client dependsOn server
    server dependsOn cache, persistence
    ...
}
tier "client" {
   layers { ui, framework, comm, cache}
   layer "ui" { ... }
   ...
}
```



DECLARATIVE RULE SETS

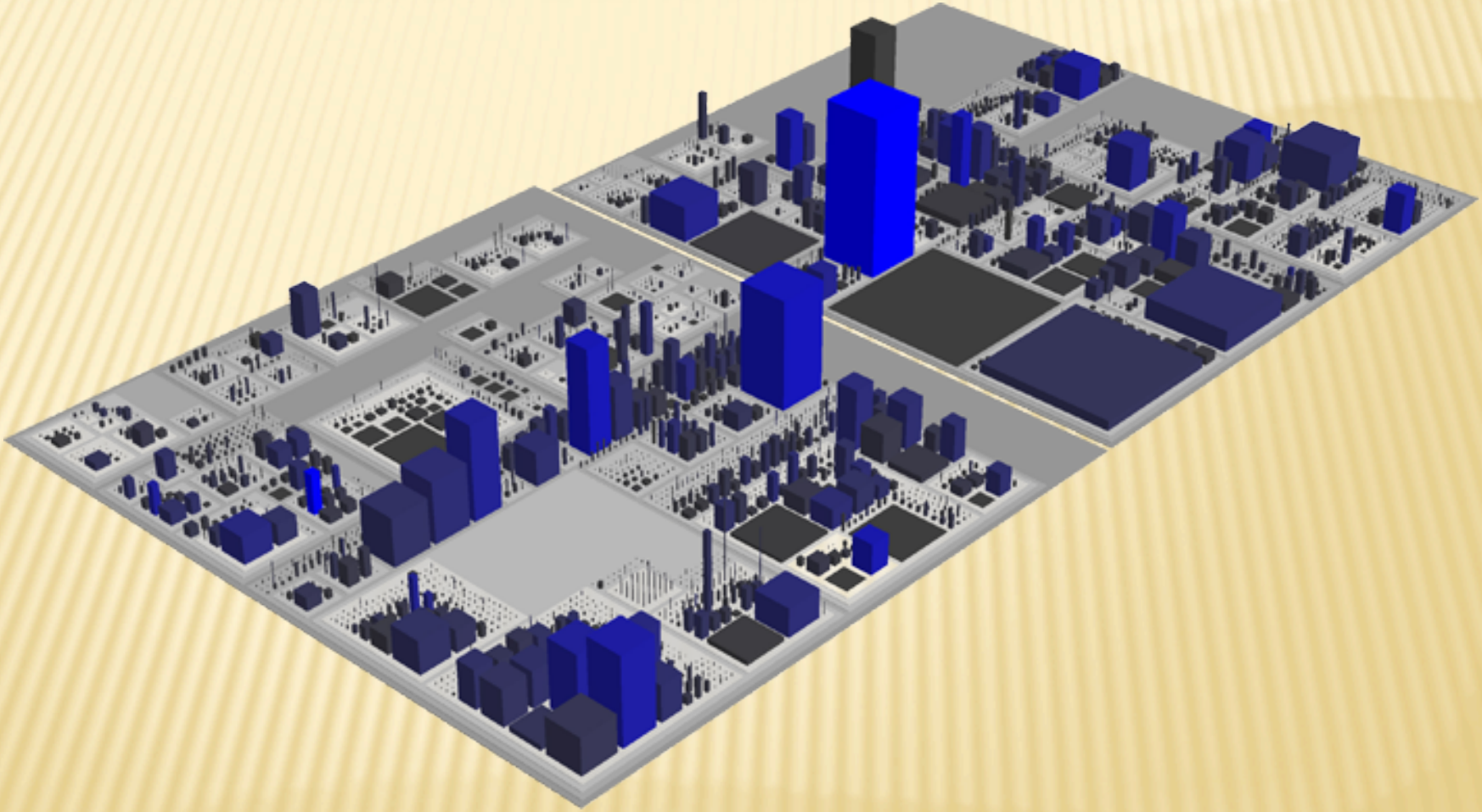
```
<architecture>
    <configuration> ... </configuration>

    <rules>
        <rule id="web-layer-separation">
          <comment>web and dao mix like oil and water</comment>
          <packages>
            <package>com.company.app.web</package>
            <package>com.company.app.web..*</package>
          </packages>
          <violations>
            <violation>com.company.app.dao</violation>
            <violation>com.company.app.dao..*</violation>
          </violations>
        </rule>

        <rule id="dao">
            ...
       </rule>
    </rules>
</architecture>
```

This example is taken from the documentation of Architecture Rules
http://www.architecturerules.org/

VISUALISATION BEYOND PIE CHARTS



This is Code City primarily showing metrics rather than structure
http://www.inf.usi.ch/phd/wettel/codecity.html

BENEFITS FOR RESEARCH & PRACTICE

THE POTENTIAL BENEFITS

- Provide software designers with:
  - accurate design information
  - a way of communicating design information in code
  - easier ways of recovering design from code
- Provide researchers with:
  - a platform to experiment on (like UML or Java)
  - provide a common technology for industrial collaboration
- Allow the creation of:
  - design recovery, analysis, verification, transformation tools
  - diagnostic and impact analysis tools for production environments
- Wide use would lead to:
  - domain specific specialisations
  - criticism and the creation of v2!

SUMMARY

SUMMARY

- A lot of design information gets lost in the code
- Researchers & practitioners see this differently
  - research – a reason for MDA, ADLs & recovery tools
  - practice – rules checkers & analysis tools
- I personally think there's a better way
  - stop the information getting lost, augment the code
- This isn't easy but could really change things
  - tangible benefits for research *and* practice

Eoin Woods
eoin.woods@artechra.com
www.eoinwoods.info

Nick Rozanski
nick.rozanski@artechra.com
www.rozanski.org.uk