(Experimental) Software Architecture Matters

Klaus Marius Hansen
Professor, PhD
Department of Computer Science (DIKU)
Outline

Klaus Marius Hansen

Software architecture matters
Observation and intervention matter(s)
Experiments matter(s)

Outlook and summary
Klaus Marius Hansen
Klaus Marius Hansen

Education
- MSc, Computer Science (minor in Mathematics), Aarhus University, 2000
- PhD, Computer Science, Aarhus University, 2002
  - "Experimental Object-Oriented Modelling"

Employment
- Software industry, 2000-2002, 2007-
- Aarhus University, 2002-2008, 2010
- University of Iceland, 2008-2010
- University of Copenhagen, 2010-

Research
- Software architecture, dependable systems, experimental approaches
- A large element of collaboration with (software) industry
  - Last count: more than 60% of papers written in connection to projects with industry
- A large focus on working software
  - Last count: 13 (more or less working) research prototypes
Shaping Experiences

1997-1998: The Dragon Project (Christensen et al., 1998)

- Prototyping a Global Customer Service System with a large container transportation provider
- Iterative, incremental, experimental development
- Ethnography, participatory design, and object-orientation
- System structure/software architecture

“Multi-perspective Application Development”...
Shaping Experiences

1999-2002: The Knight Project (Damm et al., 2000)

- Developing gesture-based diagramming environments (e.g., for Unified Modeling Language (UML) modeling)
- Iterative, incremental, experimental development
- Ethnography, participatory design, and object-orientation
- System structure/software architecture

And further

- EPCiR, LIWAS, OOSafety, eu-DOMAIN, PalCom, KILO, Hydra, SA@Work, ...
Shaping conclusions

1. Software architecture matters

2. Observation (and intervention) matter

3. Experiments matter

In the following, we will look closer at these in turn from a software architecture perspective
Software Architecture
Software Architecture?

*The software architecture of a computing system is the structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them* (Bass et al., 2003)

*A software system’s architecture is the set of principal design decisions made about the system* (Taylor et al., 2009)
Software Architecture – Electronic Patient Records

Functional structure
- Medication, booking, requisition, notes, ...

Module structure
- EPJClient, EPJIntegrationServer, Administration, Modeling, Integration, ... packages

Deployment structure
- Clients, Borland Application Server, Oracle Database, TIBCO Rendezvous, LABKA, ...

Allocation structure
- Medication team, booking team, server team, integration team, ...

Relationships
- Usage, imports, allocation, ...

Externally visible properties
- Services, interfaces, qualities, ...
Software Architecture – Electronic Patient Records

Principal decisions

- Multiple EPR systems
- Data models

Data models are complex, situated, change, ...

- Study of Danish EHRs (Christensen and Hansen, 2002)
- Amager Hospital, Ribe Amt, Aarhus Amt

How to architect for this?

- “Nucleus” (Systematic)
  - Create a meta-model of data, allow users to change at runtime
- “Nora” (Acure/IBM)
  - Create simple architecture so that it is less expensive to do re-development
Does software architecture matter?

Answer 1:

• There are many jobs/much press/... in/about/... software architecture (and infrastructure/enterprise architecture/service-oriented architecture)...

Copenhagen, 2011-01-13
Slide 12
Does software architecture matter?

Answer 2:

- There is much academic interest in software architecture and many interesting results

2002: Dynamic architectures
2008: Architectural competence
2004: Design decisions
1998: ATAM
1995: 4+1 Architectural Description
1994: Foundations for the study of software architecture
1992: Formalizing architectural connectors
1995: 4+1 Architectural Description
1992: Foundations for the study of software architecture
1994: Formalizing architectural connectors
1995: 4+1 Architectural Description

Google Scholar publication count

Year

Does software architecture matter?

Answer 3 (Booch, 2007):

• Software architecture constrains or facilitates the achievement of
  • Functional requirements
  • Non-functional requirements
  • Business goals
• Focus on software architecture supports
  • Simplification
  • Reuse
  • Acquisition
  • Continuous evolution
  • Risk mitigation
  • Systems-of-systems
Does software architecture matter?

Answer 4 (Hansen et al., 2010, in progress):

- Does “good” software architecture correlate with “good” software products? Can we show this quantitatively?
- “Good”?
  - E.g., software architecture metrics such as level of coupling
  - E.g., software product metrics such as #defects/line of code

Further questions

- Does good architecting lead to good architecture?
- Is any aspect of architecture predictive of quality?
- Do the results generalize to other types of systems?
- ...
An empirical study of software architecture quality

We studied a large number of software systems
- Used open source due to availability of data (specifically SourceForge)
- Used one programming language due to uniformity (specifically Java)
- Defined metrics to be able to be automatically computable

Analysis based on a gross list of the 21,904 most highly ranked Java projects on SourceForge (2009-03-17)

<table>
<thead>
<tr>
<th>Relevance filtering (&quot;All projects&quot;)</th>
<th>Classification filtering (&quot;Mature projects&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of bugs reported</td>
<td>≥ 1</td>
</tr>
<tr>
<td>Download rate (downloads per day)</td>
<td>≥ 2</td>
</tr>
<tr>
<td>Number of developers</td>
<td>≥ 2</td>
</tr>
<tr>
<td>Development status</td>
<td>4, 5, 6</td>
</tr>
<tr>
<td>Project age (days)</td>
<td>≥ 180</td>
</tr>
<tr>
<td>SLOC$^*$</td>
<td>≥ 2000</td>
</tr>
<tr>
<td>Total number of projects</td>
<td>1,141</td>
</tr>
</tbody>
</table>

($^*$SLOC = physical source lines of code which is the total number of non-comment lines in the code)
Method

We calculated architecture and product metrics for all projects.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Full Name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACP</td>
<td>Average Classes per Package</td>
<td>The total number of classes divided by the total number of packages</td>
</tr>
<tr>
<td>AND</td>
<td>Average Normalized Distance</td>
<td>A measure of how abstract (ratio of abstract classes/interfaces to concrete classes) and unstable (ratio of outgoing dependencies to all dependencies) packages are on average</td>
</tr>
<tr>
<td>CEX</td>
<td>Coupling Excess</td>
<td>A measure to which degree the coupling of packages to other packages exceeds our coupling model</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Full Name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODR</td>
<td>Open Defect Ratio</td>
<td>The ratio of open defects to the total number of defects</td>
</tr>
<tr>
<td>ROU</td>
<td>Rate Of Usage</td>
<td>The number of downloads per month the project has existed</td>
</tr>
<tr>
<td>AMC</td>
<td>Average Methods per Class</td>
<td>The total number of methods divided by the total number of classes</td>
</tr>
<tr>
<td>ACD</td>
<td>Average Complexity Density</td>
<td>The sum of cyclomatic complexities for all methods divided by the number methods</td>
</tr>
</tbody>
</table>
Results

We performed pairwise, linear regression for all 12 pairs of (architecture, product) metrics with the architecture metric as an independent variable

- E.g., Rate Of Usage vs. Average Normalized Distance

All pairs gave a model significantly different from a constant model (using a standard t-test)

In general relationships are stronger for mature projects
Observations and Interventions
User involvement

In general software development
- User involvement is often critical
  - E.g., Standish (1995): user involvement most critical factor in success and lack of user involvement most critical factor in failure
- Practically, users know what they need
- Morally, users have to live with consequences

Cooperative development/participatory design
- Users are an indispensable resource
- Users are involved through observation and intervention

In software architecture?
- Software architecture design
  - “Users” = software developers?
- Software architecture research
  - “Users” = software architects?
Can we use the same approaches as for software development?
Developers as users

Distributed software development involves teams in different geographical locations

- How do you maintain architectural integrity? Ensure buy-in in an architecture?...

“Traveling Architects” (Corry et al., 2006)
- One or more architects visit teams – elicits architecture

Examples
- The Palpable Computing (PalCom) project
- Global software development
- Site visits in the Dragon Project
Software architects as users

Software architects are users of architecture description languages (e.g., UML), design techniques (e.g., ADD, evaluation techniques (e.g., ATAM), ...

What do good software architects actually do?
- Use established techniques → high quality?
- Do not use established techniques → low quality?
- Use other techniques → high quality?

We extensively studied software architects in four different organizations
- Ethnography-inspired observations
- Analysis and design of practice
- Intervention with new practices
Why observations?
Observations of software architects
Analyses of architects concerns

Qualitative analysis

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Realisation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mgt/Eval</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interaction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend

- Strong focus
- Some focus
- Weak focus
- Little or no focus

Quantitative analysis

C3/A3

- a: analysis 40%
- 30%
- f: interaction 20%
- 10%
- e: management 0%
- d: evaluation

- b: design
- c: realization

Figure 1 Percentage of time spent on types of concerns

Figure 2 Concern grid for company 3 and architect J3

Figure 4 Concern grid for company 3 and architect J3

Copenhagen, 2011-01-13
Slide 25
Generalized observations of software architects

Architects are information hubs/gateways
  • Information from developers and business meets at architects

People quality is as important as structure quality
  • As much time and attention is paid on interaction as on “architecture-related” activities

Architecture as distributed cognition
  • The “real” architecture does not exist solely either in code, documents, or in the mind of developers/architects

Architecture is in the small (and in the large)
  • A default random allocation in a database table column may mean a non-performing system
Experiments
Experiments?

experiment
noun  |  ik'sperəmənt |
a scientific procedure undertaken to make a discovery, test a hypothesis, or demonstrate a known fact: laboratory experiments on guinea pigs | I have tested this by experiment.  
- a course of action tentatively adopted without being sure of the eventual outcome: the previous experiment in liberal democracy had ended in disaster.

1) Experiments as demonstration

2) Experiments as exploration
Architectural prototyping

An architectural prototype consists of a set of executables created to investigate architectural qualities related to concerns raised by stakeholders of a system under development. Architectural prototyping is the process of designing, building, and evaluating architectural prototypes (Bardram et al., 2004)
Architectural prototyping example - telemedicine

Concerns
- Does RMI/REST/SOAP perform adequately?
- How much effort needs to be invested into integration?
- Will our architecture scale?

Qualities
- Performance
- Modifiability/buildability
- Scalability

Prototyping
- Non-functional prototype of data transfer using RMI, REST, SOAP; measure round-trip time
- Exploratory integration of an iPhone client with Columna; note experience of integrating
Architectural prototyping example - telemedicine

Architectural prototypes
(1) are constructed for exploration and learning of the architectural design space
(2) address issues regarding architectural quality attributes in the target system
(3) do not provide functionality per se
(4) typically address architectural risks
(5) address the problem of knowledge transfer and architectural conformance
Architectural prototyping – safety-critical system

Design and development of frequency inverters for use in safety-critical systems
- IEC 61508, Safety Integrity Level 3

Models of functionality and architecture
- Safety properties modeled using state-charts and Coloured Petri Nets
- Software architecture designed in UML

Built architectural prototypes (Hansen and Wells, 2006)
- Execution of prototype traced using AspectJ
- Program execution events mapped to discrete-state model execution events
Architectural prototyping in practice

Case studies of architectural prototypes in four companies

Survey study of architectural prototyping

- 20 developers/architects, mean "age" 10.0 years

Architectural prototyping is used frequently

Architectural prototyping is construed of as important
Outlook
Outlook – Architectural emulation

To which extent is it possible to emulate software architectures?

• How do you achieve fidelity of emulations?
• How can code, emulation, and simulation co-exist?
• Which qualities can be analyzed by emulation?
Outlook – Architectural reconstruction

Architectural design is often not present in software systems
- Layering is, e.g., compiled away
- Design decisions often do not translate to code
... neither is it documented outside the system...

Can we support architectural reconstruction in experimental/agile development in a non-intrusive way? (Christensen and Hansen, in progress)

```java
@Pattern(name = "Strategy", role = "Context", id = "PriceCalculation")
public class PayStation {
    ...
}

@Pattern(name = "Strategy", role = "Strategy", id = "PriceCalculation")
public interface RateStrategy {
    ...
}

@Pattern(name = "Strategy", role = "ConcreteStrategy", id = "PriceCalculation")
public class LinearRateStrategy
    implements RateStrategy {
    ...
}
```
Outlook – Supporting experimentation

Accidental complexity in software system development has been drastically reduced in the last decades

- Due to product-related advances, compare, e.g., Parnas (1972) and Hansen (2010)
- Due to process-related advances, e.g., (Systematic, 2008)

One consequence is that the principal design decisions of yesterday are not the principal design decisions of today

How can we exploit this in software architecture design?
Summary
Summary

( XKCD, 2011)
Summary

Software architecture is concerned with principal design decisions for complex software systems. It is crucial technically and organizationally in complex system development.

Experimental techniques can play a major role in software architecture design:

- Observations – (field) studies of software architects and software architectures
- Interventions – introducing new techniques and artifacts
- Experiments – using partial software systems in architecture design

1 postdoctoral position and 3 PhD positions

- Postdoctoral Research Position within Information Visualization on Large Displays
- PhD scholarship within Interaction Techniques for Wall-Sized Displays
- PhD scholarship within Computer-Supported Cooperative Work
- PhD scholarship within Software Architecture/Software Ecosystems