About the Software Improvement Group (SIG)

- Factual and actionable management advice through source code analysis and structured methods
- Assessment of a variety of technologies by using technology-independent methods

Providing management with insight into IT projects and systems by means of facts and measurements

Active contributor to scientific research in software engineering
SIG and Huawei are performing a R&D cooperation project in the area of software architecture analysis.

- Architecture of large software landscapes
- Dynamic architecture measurement
- Identification of architecture hotspots
About us

Dennis Bijlsma

- Senior consultant at SIG
- Evaluated 200+ software projects
- Leading R&D projects in software architecture analysis

dr. Haiyun Xu

- Chief Security Officer at SIG
- PhD in biometric security and privacy
- Leading R&D projects in software architecture analysis
Part 1

Industry trends for software landscapes
Current software trends are about handling increased expectations

- Greater flexibility
- Faster to introduce new functionality
- Higher scalability
- Lower maintenance costs
Trend: the amount of software is increasing

Number of people working in the Dutch IT sector

- 2009: 262K
- 2011: 267K
- 2013: 329K
- 2015: 365K
Trend: software code/deployment architectures are getting larger

Average number of components per application

Source: SIG benchmark
Trend: software is getting more interconnected

Typical software landscape (anno 2006)

Customer-facing website

| Database |
| Back-office application |

Typical software landscape (anno 2018)

Customer-facing website

| Android app |
| iOS app |

| Service layer |
| Legacy system |

| Affiliate website |
| Payment provider |
A typical example of communication lines within a software landscape
Trend: need for more flexibility necessitates a more flexible release process

Teams using DevOps

Teams using continuous delivery

Source: SIG benchmark
Part 2

The problem with big landscapes
The relation between volume and software quality

Source: SIG benchmark
The relation between volume and defect density

Source: Code Complete, S. McConnell
The need for evolving large architectures
The need for evolving large architectures
Larger systems and landscapes – a dilemma

Disadvantage of large systems
- Require large team to maintain
- Tend to have more defects
- Less scalable

Trend for even larger software landscapes
- More functionalities
- More complicated features
- Large connected systems
Larger systems and landscapes – a dilemma

Disadvantage of large systems
- Require large team to maintain
- Tend to have more defects
- Less scalable

Advanced architecture to cope with the dilemma
- Decompose large system into smaller services that enables team autonomy
- Define clear interfaces between services to enable well communicated services

Trend for even bigger software systems
- More functionalities
- More complicated features
- Large connected systems
A decomposed software landscape does not automatically produce benefits

Loosely coupled services

Service A

Service B

Services without clear business domain

Service A

Service B
Part 3

How to use measurements to improve software landscapes
Why do we need measurements?

“You can’t control what you can’t measure, you can’t improve what you can’t measure.”

-- Tom DeMarco, Controlling software project management
The goal: using measurements for continuous architecture improvement

- **Measurement**
  - Determine the current state of the architecture

- **Analysis**
  - Interpretation of measurement results
  - Identify improvement areas

- **Improvement**
  - Implement the proposed changes
  - Re-measure to verify whether the changes have the desired consequences
The goal/question/metric approach

Identify goals
Derive questions
Specify metrics

How to use measurements: the goal/question/metric approach

Identify goals -> Derive questions -> Specify metrics

**Example**

> Become a sheep

> What do sheeps usually eat?
> Where do sheeps typically stay?
> What for?

> Consumed kilos of grass
> Time spent living on farmland
> Produced kilos of wool
Be careful when trying to influence people’s behavior using measurements

“Not everything that counts can be counted.”
-- Albert Einstein
-- William Bruce Cameron

“Not everything that can be counted counts.”
-- Carlo Ancelotti, football coach
Pitfalls of measurement-based evaluation

People will change their behavior if you introduce measurements to monitor them – and not always in a good way

> It is impossible to quantify a person’s overall contribution into a numerical measurement

> Also, people will over-focus on the part of their work that is measurable and used to judge them
Evaluation criteria for choosing the right type of measurement

**Purpose**
- The measurement helps you to achieve one of the project’s goals

**Practicality**
- The measurement is easy to understand
- The measurement can be be performed quickly and frequently

**Actionability**
- The measurement makes it clear *when* action should be taken
- It is clear *how* to take action to address the issue

**Incentive**
- People have a reason to care about the measurement’s results
- People’s behavior is influenced in the desired way
Different types of measurements

- Compliance measurements
- Violation-based measurements
- Vanity measurements
- Risk-based measurements
- Pattern conformance measurements
Compliance measurements: example

Safety-critical software, such as in trains, must pass a safety certification process before it can be used.
Compliance measurements

Compliance measurements are usually performed in a certification or approval process

> Consist of a number of checks that must be passed

> Approval process can take months and results in written report

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Practicality</th>
<th>Actionability</th>
<th>Incentive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Violation-based measurements: example

Many of those violations are probably important, but the sheer number means it is hard to prioritize.

> Also hard to tell whether this is good or bad, the absolute number is hard to interpret.
Violation-based measurements

- Purpose
- Practicality
- Actionability
- Incentive

Violations
Threshold
Vanity measurements: example

Of course 10 billion lines of code is a huge number... but the number in itself is meaningless

> Maybe we just analyzed one script a billion times (note: we didn’t 😃)
Vanity measurements

“Vanity metrics: good for feeling awesome, bad for action.”

-- Tim Ferris

“You measure things so you can make decisions based on the measurements, but vanity metrics don’t contribute to that process.”

-- Richard Bayston
Vanity measurements are not always bad!

Vanity measurements can be a great tool for motivating people and reminding them to stay focused
Risk-based measurements: example

Unit test coverage over time

- Unrealistically high for most projects
- OK for most projects
- Danger zone

Week 20  Week 21  Week 22  Week 23  Week 24  Week 25  Week 26

0%  20%  40%  60%  80%  100%
Risk-based measurements

- Very high risk
- High risk
- Moderate risk

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Practicality</th>
<th>Actionability</th>
<th>Incentive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pattern conformance measurements: example

Even though there are infinite possibilities and combinations for the overall solution, it tends to be comprised of many small standardized patterns.
Pattern conformance measurements

Patterns are similar to violations and risk-based measurements, but are more actionable because they provide guidance:

- Clear instructions on what (not) to do
- More convincing due to being based on industry consensus

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Practicality</th>
<th>Actionability</th>
<th>Incentive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pattern (industry best practice)
### Pros and cons of different types of measurements

<table>
<thead>
<tr>
<th>Type</th>
<th>Purpose</th>
<th>Practicality</th>
<th>Actionability</th>
<th>Incentive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compliance measurements</td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Red" /></td>
<td><img src="#" alt="Red" /></td>
<td><img src="#" alt="Green" /></td>
</tr>
<tr>
<td>Violation-based measurements</td>
<td><img src="#" alt="Orange" /></td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Orange" /></td>
<td><img src="#" alt="Orange" /></td>
</tr>
<tr>
<td>Vanity measurements</td>
<td><img src="#" alt="Orange" /></td>
<td><img src="#" alt="Red" /></td>
<td><img src="#" alt="Red" /></td>
<td><img src="#" alt="Green" /></td>
</tr>
<tr>
<td>Risk-based measurements</td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Orange" /></td>
<td><img src="#" alt="Orange" /></td>
</tr>
<tr>
<td>Pattern conformance measurements</td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Orange" /></td>
</tr>
</tbody>
</table>
Part 4

Dynamic architecture: what’s *really* happening
Measuring static versus dynamic architecture
Dynamic architecture is about behavior in practice

- Designed Architecture
- Implemented Architecture
- Dynamic Architecture
So why don’t we just measure the dynamic architecture?

Because it’s difficult

> Requires a running version of the system
> Long(er) feedback loop
> Accuracy depends heavily on the data gathered during the testing period
> Hard to relate findings back to the developer perspective
Measuring static versus dynamic architecture

**Static architecture**

- Matches the developer perspective
- More about structure
- Easy to measure

**Dynamic architecture**

- Matches the perspective of the system in use
- More about behavior
- Difficult to measure
Part 5

Putting it all together: improving your landscape architecture
Architecture measurement requires input from both static and dynamic perspectives

- Extract source code information
- Static architecture data
- Extract runtime information
- Runtime data
- Connect
- Dynamic architecture information
Dynamic architecture model: from input to output

Lower layer is the architecture analysis foundation; upper layers enhance analysis techniques step by step by analyzing extra available information.

**Examples of input data**
- Code instrumentation information; records of runtime data
- Output of DevOps tools
- Build scripts, configuration files, deployment scripts
- Source code

**Measurement model**
- Code Instrumentation
- Dynamic data
- Environment data
- Static architecture analysis

**Expected output**
- Dynamic analysis like runtime coupling
- Mapping between operational & architecture perspectives
- Improved architecture analysis with dynamic information
- Static architecture
Mapping between the static and dynamic perspectives

**DevOps tools (dynamic perspective)**

<table>
<thead>
<tr>
<th>Current load</th>
<th>Requests/Minute current</th>
<th>Requests last 24 hours</th>
<th>Runtime MCycles last hour</th>
<th>Average latency last hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>/notes</td>
<td>0.4</td>
<td>11</td>
<td>4,559</td>
<td>5,251 ms</td>
</tr>
<tr>
<td>/folders</td>
<td>0.4</td>
<td>11</td>
<td>239</td>
<td>189 ms</td>
</tr>
<tr>
<td>/folders-sync</td>
<td>0.4</td>
<td>10</td>
<td>206</td>
<td>233 ms</td>
</tr>
<tr>
<td>/notes-sync</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

**Code perspective**

```java
@Rest(method = Method.GET, path = "/notes", authorized = "user")
public HttpResponse listNotes(RestRequest request) {
    Account userAccount = getUserAccount(request);
    List<Note> notes = dataStore.findNotes(userAccount);
    return toResponse(notes);
}
```

**Static architecture perspective**
Using a combination of static and dynamic architecture patterns

**Code architecture level**
- Code structure
- Code dependencies

**Deployment architecture level**
- Deployment architecture
- Interface dependencies
- Database dependencies

**Runtime architecture level**
- Performance, scalability, reliability
Example code-level architecture pattern: no cyclic dependencies

Component A

Component B

Component C

Component D

Component E

Component F

Component G

Component H
Example code-level architecture pattern: excessive code sharing

Component A
Component B
Component C

Shared platform

Component A
Component B
Component C

Shared platform
Example deployment-level architecture pattern: no shared databases
Example deployment-level architecture pattern: service without cohesion
Example runtime-level architecture pattern: overly chatty service communication

Service A
- getCustomerInfo

Service B

Service A
- getCustomerName
- getCustomerAddress
- getCustomerContactDetails

Service B
Combined visualizations for static and dynamic architecture

**Time period:**
13-12-2017 0:00 to 14-12-2017 0:00

**Selected transaction:**
Cancel subscription

- **Browser**
  - Website
    - Back-end system A
      - Database A
    - Back-end system B
      - Database B
    - Back-end system C

- Time period: 0.4 seconds
- Time period: 1.1 seconds
- Time period: 2.9 seconds

Page 56 of 60
Prioritizing architecture issues

Addressing *all* architecture issues is not viable; the key challenge is identifying the improvement areas that would bring the most benefit.
Epilogue

Building a maintainable architecture
Measurements as a feedback loop for architects and development teams

Measure current state

Prioritize findings

Identify improvement areas

Implement improvements
What did we discuss today?

- **Disadvantage of large systems**
  - Require large teams to maintain
  - Tend to have more defects
  - Less scalable

- **Trend for even bigger software systems**
  - More functionalities
  - More complicated features
  - Large connected systems

- **Advanced architecture to cope with the dilemma**
  - Decompose large system into smaller services that enable team autonomy
  - Define clear interfaces between services to enable well communicated services

- **Extract source code information**
  - Source code
  - Related source code information

- **Extract runtime information**
  - Extract runtime call graph
  - Connect

- **Compliance measurements**
  - Violation-based measurements
  - Vanity measurements
  - Risk-based measurements
  - Pattern conformance measurements

- **Advanced architecture**
  - Prioritized dynamic architecture advice