Research Statement

Kevin Steppe
School of Information Systems, Singapore Management University
Tel: (65) 6828-0905; Email: kevinsteppe@smu.edu.sg
24-1-2014

Background

It is widely understood and agreed that the only way for humans to create large complex systems – buildings, airplanes, companies, and software to run them – is through decomposition into modules. These modules must be linked to perform the overall goal, which creates coupling and dependencies. Designers have long been aware that minimizing these dependencies makes systems easier to understand, change, and build. I intend to study these dependencies and inter-module couplings in order to aid designers in constructing better systems through an understanding of the impact of those dependencies. Specifically, a designer will want to know:

- What is the impact of changing a particular module?
- What are the effects of replacing a module with another “compatible” module? When can this be done dynamically?
- Does a particular technology, such as messaging, actually produce less coupling and reduce inter-module dependence?
- Which dependencies will have the most impact on future modifications?
- Is ‘this’ composition of modules more easily changed than ‘that’ one?

Current research

I have created a graph model for dependencies which allows designers to answer the above questions. The graph model does not aim to provide new solutions to any particular design problem. Rather, it provides a ‘language’ and statements about structures expressed in that language which allow analysis of a wide range of designs.

The graph model views systems as being composed of behaviors – functions within the system – the modules which implement behaviors, and interfaces to those modules. These elements – the nodes in the graph – are related by different kinds of dependencies. The model makes statements about the modifiability of portions of the system based on the decomposition and dependency relations. Given these modifiability statements we can compare to alternative designs for the same system.
One way to illustrate the usefulness of this technique compared to others, is to take a sampling of patterns from Design Patterns[17] and Larman[22] as examples of designs which are recognized by practitioners as useful for giving modifiability or reusability in certain contexts. After applying several analysis techniques to the before-refactoring and after-refactoring versions of the software, we ask: does this analysis indicate that the refactoring should be helpful?

<table>
<thead>
<tr>
<th>Pattern:</th>
<th>Dependency Graph Model</th>
<th>Parnas’s “Uses”</th>
<th>Count CBO (calls)</th>
<th>Assumptions - Jackson</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Post-pattern improved?</td>
<td>Post-pattern improved?</td>
<td>Post-pattern improved?</td>
<td>Post-pattern improved?</td>
</tr>
<tr>
<td>Factory</td>
<td>Yes</td>
<td>Ambiguous</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Adapter</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Template</td>
<td>Yes</td>
<td>Ambiguous</td>
<td>Ambiguous (even)</td>
<td>No</td>
</tr>
<tr>
<td>Information Expert (Larman p283)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Decorator</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>State</td>
<td>Yes</td>
<td>Ambiguous</td>
<td>No, but better cohesion</td>
<td>unclear</td>
</tr>
<tr>
<td>Strategy</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Ambiguous</td>
</tr>
<tr>
<td>Observer</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

We can immediately see that most techniques either frequently contradict the pattern community’s insight or are frequently ambiguous or both. My dependency model however, correctly identifies all these patterns are useful – without the need to memorize so many constructs.

I have also used the model to construct dependency graphs for the software managing the Genting casino floors. This system has nearly one million lines of code organized into several thousand modules. I was able to automate construction of dependency graphs and demonstrate that the conservative assumption of transitive ripple effects is not a realistic assumption. We also see that the dependency graph is a better way to predict the number of ripples than coupling metrics.
I have also made a study of novice designers. I have shown that the dependency model is usable by university students and leads to a good analysis of sample designs.

**Future research**

The workshop materials for teaching to novices can be improved. I expect much better outcomes for the students after another iteration or two on the materials.

In future work I plan to team with professionals to test the impact of the dependency theory on design decision-making.

In addition to the Software design research, I am also interested in pedagogy generally. Along with other faculty at SIS, we plan to explore what factors in university applicants best predict programming course grades. We are also planning to investigate the potential of combining automated programming assignments with peer mentoring.

**Selected Publications and Outputs**


"Flexible Grouping and Multiple Centers for Preserving Simplicity and Flexibility in Animation Sketches", by Richard C. DAVIS, Kevin STEPPE, Mengyuan GUAN, Jing Ting KHOO, Rui ZHANG, Quee Boon KOH, 09/2013, *The 11th Asia Pacific Conference on Computer Human Interaction (APCHI 2013)*, Bangalore, India