software studio

concepts & models

Daniel Jackson

For now, all rights reserved. Daniel Jackson, 2012.
things we love & hate
ratings

from http://amplicate.com
how to add a signature in adobe acrobat
-- open document in adobe acrobat
-- Tools->Advanced Editing->Touchup Object Tool
-- right click at desired point | Place Image... then select jpg

how to add date
-- Tools->Typewriter
why we love & hate
Easy Email Newsletters

MailChimp helps you design email newsletters, share them on social networks, integrate with services you already use, and track your results. It's like your own personal publishing platform.

2 million people use MailChimp. Join them today.

Sign Up Free

Need convincing?
skin deep?

It looks like you're writing a letter.
Would you like help?
- Get help with writing the letter
- Just type the letter without help
- Don't show me this tip again

Clippy
2003-2008
RIP

- Lightroom Application Shell
- Camera Raw
- Library Toolkit
- Net I/O
- Etc.
- Store Provider
- Substrate

**User's model**

**Conceptual model**

**Interface**

**Code**
putting it another way...

“make your abstractions robust”

but which abstractions?
<table>
<thead>
<tr>
<th>weak concepts</th>
<th>strong concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>hard to use</td>
<td>intuitive, predictable</td>
</tr>
<tr>
<td>a mess to maintain</td>
<td>decoupling &amp; localization</td>
</tr>
<tr>
<td>unreliable &amp; buggy</td>
<td>more dependable</td>
</tr>
</tbody>
</table>
what we’re doing

thinking & sketching
simulating features

normal design practice
copying good ideas

evaluating products
user feedback

discarding failed designs
“refactoring”

To design something really well, you have to get it. You have to really grok what it’s all about. It takes a passionate commitment to really thoroughly understand something, chew it up, not just quickly swallow it. Most people don’t take the time to do that.
what we’re not doing

being explicit

focusing: what are the concepts?
relating: how are they related to each other?
analyzing: what properties do they have?
models of problems & solutions
The first scheme represents the unselfconscious situation described in Chapter 4. Here the process which shapes the form is a complex two-directional interaction between the context C1 and the form F1, in the world itself. The human being is only present as an agent in this process. He reacts to misfits by changing them; but is unlikely to impose any “designed” conception on the form.

The second scheme represents the selfconscious situation described in Chapter 5. Here the design process is remote from the ensemble itself; form is shaped not by interaction between the actual context’s demands and the actual inadequacies of the form, but by a conceptual interaction between the conceptual picture of the context which the designer has learned and invented, on the one hand, and ideas and diagrams and drawings which stand for forms, on the other. This interaction contains both the probing in which the designer searches the problem for its major “issues,” and the development of forms which satisfy them; but its exact nature is unclear. In present design practice, this critical step, during which the problem is prepared and translated into design, always depends on some kind of intuition. Though design is by nature imaginative and intuitive, and we could easily trust it if the designer’s intuition were reliable, as it is it inspires very little confidence.

In the unselfconscious process there is no possibility of mis-construing the situation: nobody makes a picture of the context, so the picture cannot be wrong. But the selfconscious designer works entirely from the picture in his mind, and this picture is almost always wrong.

The way to improve this is to make a further abstract picture of our first picture of the problem, which eradicates
co-evolution

problem space

solution space
other approaches
UML
co-evolution in UML
co-evolution in UML

heavy documentation
complex notations
tool support deferred
Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.
co-evolution in agile
co-evolution in agile

baby out with bathwater today’s orthodoxy?