thoughts on software process

Daniel Jackson
process orderings
local vs global process

- **Global ordering of phases**
- **Local ordering of phases**
risk-driven development

Risk = \text{Prob}\text{(failure)} \times \text{Cost}\text{(failure)}

a strategy
\begin{itemize}
  \item list failures & determine their risks
  \item devise a strategy to reduce highest risks
\end{itemize}

sample failures: how would you mitigate?
\begin{itemize}
  \item performance is unacceptable
  \item product is unusable because its too complex
  \item customer changes mind about what product does
  \item developer solves the wrong problem
  \item product fails in catastrophic way
  \item competitor beats you into marketplace
  \item product has reputation for bugs
  \item development runs out of time and money
  \item developers rely on platform that turns out bad
\end{itemize}
doing design
small design upfront

Agilistas deride “Big Design Upfront” (BDUF)

what about Small Design Upfront?
› what isn’t worth designing?
› can you recover from a bad design?
› what’s the cost of design?

SDUF strategies
› precise but lightweight notations
› separate concerns & focus on risks
› avoid implementation bias
be like a beaver!

small nibbles, big outcome
intuitive vs data-driven design

When a company is filled with engineers, it turns to engineering to solve problems. Reduce each decision to a simple logic problem. Remove all subjectivity and just look at the data. Data in your favor? OK, launch it. Data shows negative effects? Back to the drawing board. And that data eventually becomes a crutch for every decision, paralyzing the company and preventing it from making any daring design decisions. Doug Bowman
Spectrum of Design

Intuition-Driven

- Make best-guesses
- Rely on previous experience
- Study what others are doing
- Use best practices, principles & patterns
- Aesthetics are integral
- Rely on our gut
- Creative, visionary
- Inherently risky

Data-Driven

- Every design choice is tested
- Takes others experience with a grain of salt
- Design is a logic problem
- Rely on data for decision-making
- Aesthetics are secondary
- Never trust your gut
- Cold, calculating
- Risk-averse

Doug’s words:

- instinctive, subjective, daring

Assumed:

- deliberate, objective, safe

from Joshua Porter, bokardo.com
radical design
Sudoku

My plan, subject as always to change, is to code something up in that way that I have, to see what happens. Right now, I'm planning to implement a fairly naive strategy, and a tree-trimming one that I think should solve all problems, albeit perhaps too slowly, and then leave it open to my readers to propose new squares and new heuristic algorithms.

I'm re-ripping my entire CD collection, so I have to sit here anyway. Might as well code something.

The Game
I'm not going to talk much here about the game. There's a square of cells, with side length of \( n \)-squared, for order \( n = 1, 2, 3, 4 \), etc. You fill in the squares with the integers from 1 to \( n \)-squared, subject to the rule that the same integer cannot appear more than once in the same row, same column, or same \( n \)-size subsquare as the cell you're filling in. The game begins with some squares "given". Reportedly games come in a range of difficulty. Since I've never played the game, I don't know what makes them more or less difficult. Maybe I'll find out.

Why is This Interesting?

Frankly, I don't know, since I don't play the game. I think that during this exercise we might hit some interesting notions about solving computing problems we couldn't solve by hand, and addressing problems about which we know very little. If nothing else, it may be amusing watching me drown.

Begin With a Test

I'm going to do this in Ruby. My plan is to start with 9 by 9 squares, just because I can, in spite of the fact that I can see already, having thought about it, how to use order to compute a bunch of the items. I'll keep it specific just by way of tempting the YAGNI gods.

My Ruby style uses a project.rb file to map all the files in the app, and various .rb files to contain the tests and classes. My base setup looks like this:

```ruby
project.rb
require 'test/unit'
```
still going after five long blog posts...

Peter Norvig solves in one:

Solving Every Sudoku Puzzle
by Peter Norvig

In this essay I tackle the problem of solving every Sudoku puzzle. It turns out to be quite easy (about one page of code for the main idea and two pages for embellishments) using two ideas: constraint propagation and search.

see http://norvig.com/sudoku.html
lessons?

risk
› Ron Jeffries focuses on class design
› but real risk is algorithmic?

Norvig’s advantage
› he knows AI: applies standard solution

Walter Vincenti’s dichotomy
› normal design: tweaking parameters
› radical design: never done this before
co-evolution
co-evolution

problem space

solution space
UML
co-evolution in UML
co-evolution in UML

heavy documentation
complex notations
tool support deferred
the cost of complex tools

# pages in definition

- Algol-68 1975
- SA/SD 1979
- CLU 1981
- JSD 1983
- C++ 1985
- OMT 1991
- Syntropy 1994
- Fusion 1994
- Java 1996
- UML 1999

- Algol-60 1960
- Pascal 1974
- C 1978
- SML 1990
- Scheme 1986
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?
big questions
The first was never to accept anything for true which I did not clearly know to be such; that is to say, carefully to **avoid precipitancy** and prejudice, and to comprise nothing more in my judgment than what was presented to my mind so clearly and distinctly as to exclude all ground of doubt.

The second, **to divide each of the difficulties** under examination into as many parts as possible, and as might be necessary for its adequate solution.

The third, to conduct my thoughts in such order that, by **commencing with objects the simplest and easiest to know**, I might **ascend by little and little**, and, as it were, step by step, to the knowledge of the more complex; assigning in thought a certain order even to those objects which in their own nature do not stand in a relation of antecedence and sequence.

And the last, in every case to make enumerations so complete, and **reviews so general**, that I might be assured that **nothing was omitted**.
One of the planning documents for software research revealed --in a parenthetical remark only-- an unchallenged tacit assumption by referring to "the tradeoff between cost and quality". Now in all sorts of mechanical engineering it may make sense to talk about "the tradeoff between cost and quality", [but] in software development this is absolute nonsense, because poor quality is the major contributor to the soaring costs of software development.
Fred Brooks, *Mythical Man Month*

The management question, therefore, is not *whether* to build a pilot system and throw it away. You *will* do that. The only question is whether to plan in advance to build a throwaway, or to promise to deliver the throwaway to customers. Seen this way, the answer is much clearer. Delivering that throwaway to customers buys time, but it does so only at the cost of agony for the user, distraction for the builders while they do the redesign, and a bad reputation for the product that the best redesign will find hard to live down.

Hence *plan to throw one away; you will, anyhow.*