RELATIONAL LOGIC FOR SOFTWARE DESIGN

Daniel Jackson · Lipari Summer School · July 18-22, 2005

MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

what these lectures are about

how to model & analyze software abstractions

focus on engineering, not research
- language, idioms, examples
- not semantics, types, algorithms

what I assume you know
- basic set theory & logic

schedule
- Tuesday: Logic & language
- Wednesday: State, operations & traces
- Thursday: Events & environment

my story

formative experiences

programmer at Logica UK (1984-86)

› worked on systems for London Underground

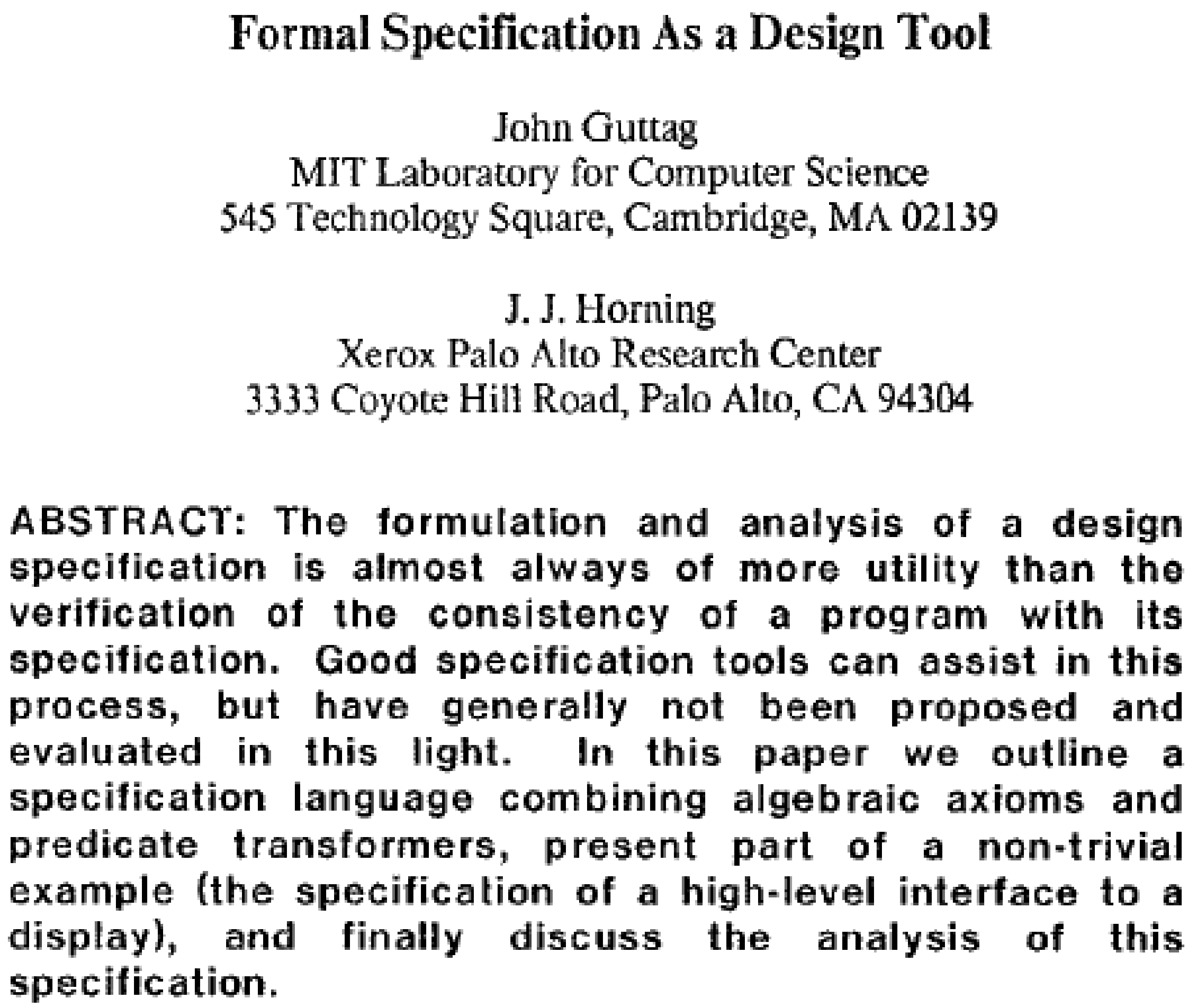
› became passionate advocate of formal methods

› joined Sesame: internal advocacy group

formal methods (esp. JSP/D and VDM)

› a solution to the software crisis!

software blueprints?



Formal Specification As a Design Tool

John Guttag
MIT Laboratory for Computer Science
545 Technology Square, Cambridge, MA 02139

J. J. Horning
Xerox Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304

ABSTRACT: The formulation and analysis of a design specification is almost always of more utility than the verification of the consistency of a program with its specification. Good specification tools can assist in this process, but have generally not been proposed and evaluated in this light. In this paper we outline a specification language combining algebraic axioms and predicate transformers, present part of a non-trivial example (the specification of a high-level interface to a display), and finally discuss the analysis of this specification.

what I learned in grad school

Larch approach wasn't yet practical
› algebraic specs hard to read and write
› automation elusive

so, for my doctorate, focused on checking code
› but hoped to come back to world of design

when I finished in 1992
› Ken McMillan's SMV: automation
› Spivey et al's Z: elegant, clear expression

bridging the atlantic?

Oxford, home of Z

Pittsburgh, home of SMV

the alloy project, 1994-2005

Nitpick [1995]

› a relational subset of Z (Tarski's RC: binary relations, no ∀∃)

› analysis: enumeration of relations + symmetry

Alloy 1.0 [1999]

› language: object modelling (set-valued 'navigation' exprs, ∀∃)

› analysis: WalkSAT, then Davis-Putnam

Alloy 2.0 [2001]

› language: relational logic (arbitrary arity, scalar ∀∃)

› analysis: Chaff, Berkmin

Alloy 3.0 [2004]

› added castless subtypes & overloading

perspective

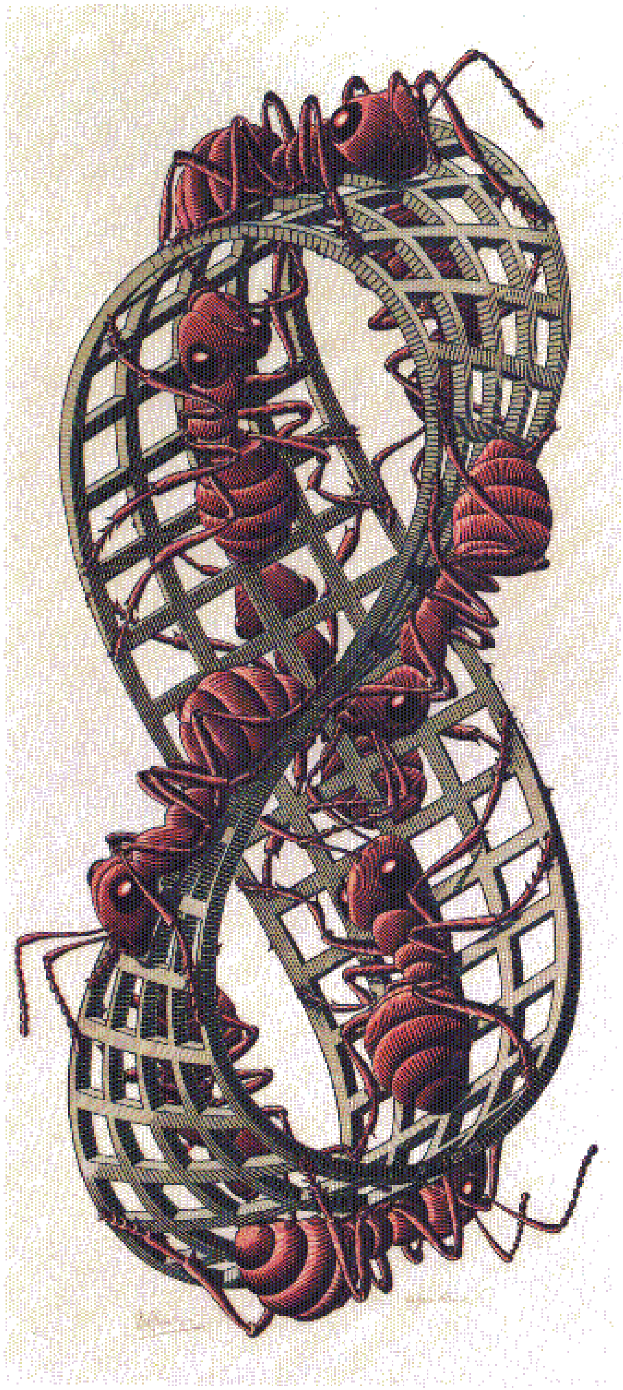
model checking for software?

is software like hardware?
› simple spec, complex realization
› a few, subtle & catastrophic bugs
› cost to fix dwarfs cost of detection

so analyze designs to find subtle bugs

software is very different
› complex specs, simple realization
› failure from complexity, not bugs
› cost to fix is usually low

so analyze designs to explore abstractions



software abstractions

software is built on **abstractions**

pick good ones, and you get
- clean interfaces (they're simple and fit well)
- a more useable system (they're the user's concepts)
- maintainable code (they match the problem)

pick bad ones, and you get
- a mess that gets worse over time
- the only refactoring that works is starting over
- special cases, hard to use

why struggle with abstractions?

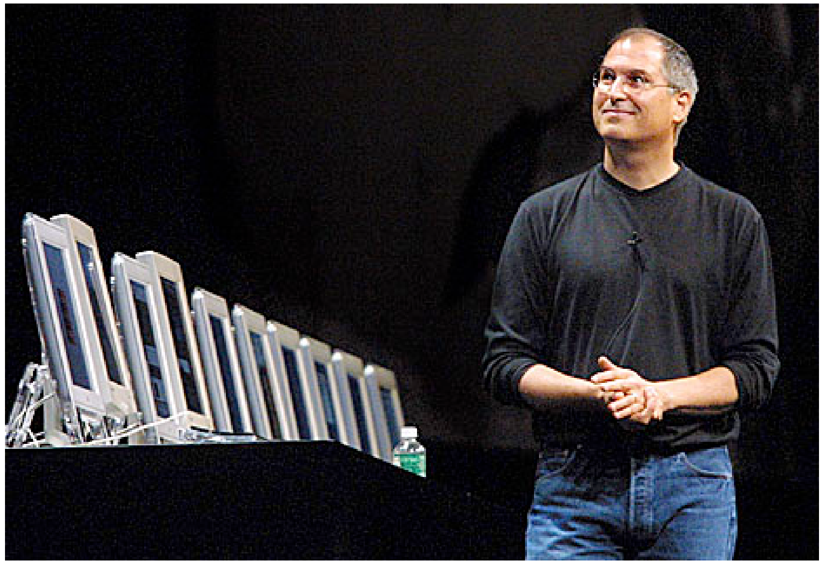
I conclude there are two ways of constructing a software design: One way is to make it so simple there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies.

-- Tony Hoare [Turing Award Lecture, 1980]

why explore details?

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.

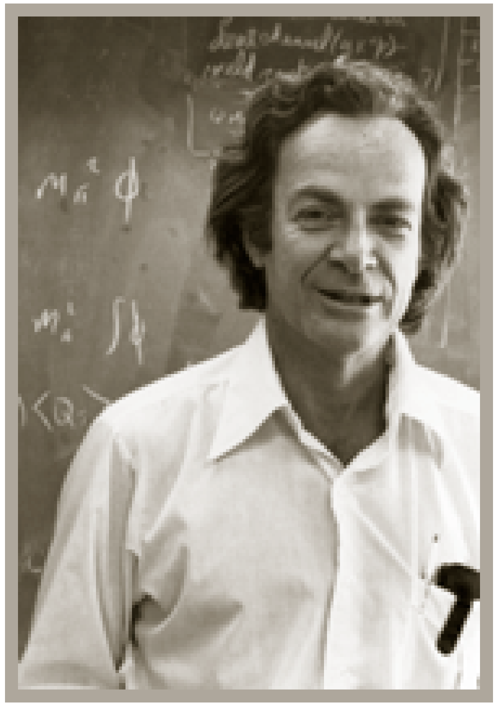
-- Steve Jobs [Wired Interview, Issue 4.02, Feb 1996]



why analyze?

The first principle is that you must not fool yourself, and you are the easiest person to fool

-- Richard P. Feynman



alloy demo

what we didn't do

incrementality
› didn't write a long model and then analyze it

low burden
› no test cases, lemmas or tactics

concrete feedback
› no false alarms, easy to diagnose

key ideas

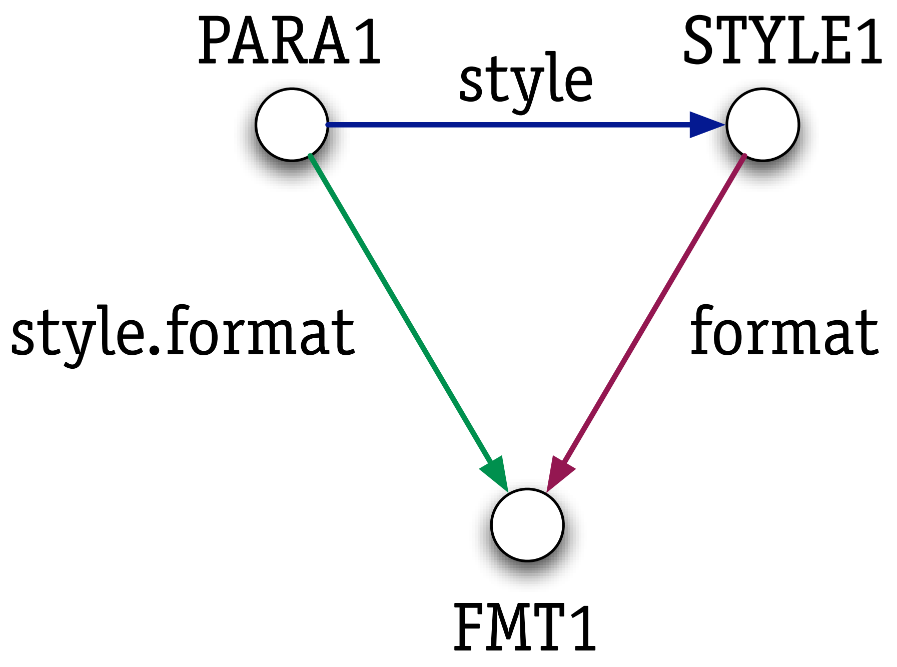
#1: everything's a relation

Alloy uses relations for

› all datatypes -- even sets, scalars and tuples
› structures in space *and* time

key operator is **dot join**

› for taking components of a structure
› for indexing into a collection
› for resolving indirection

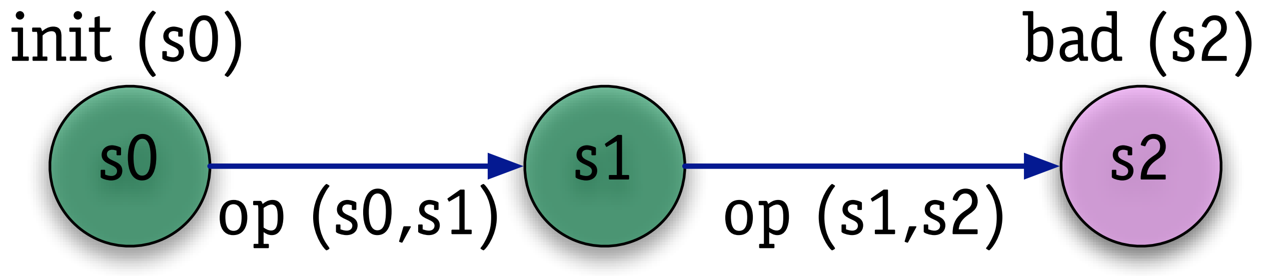


#2: pure logic

no special syntax or semantics for state machines

use constraints for describing

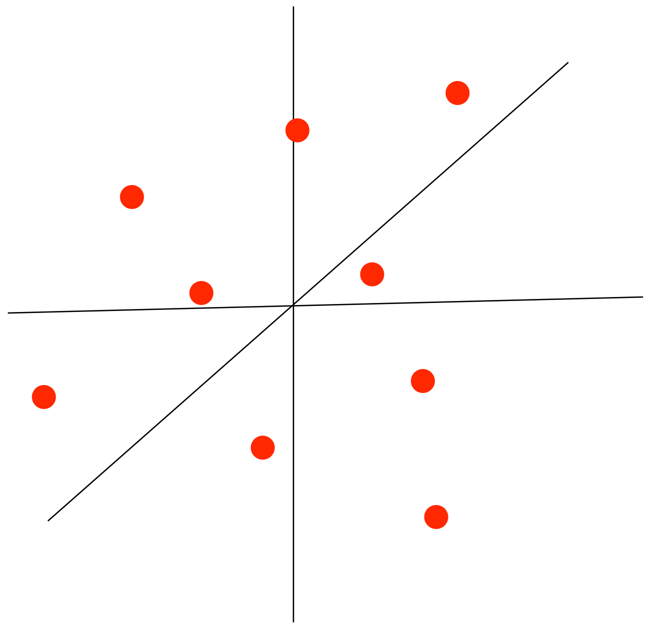
› subtypes & classification
› declarations & multiplicity
› invariants, operations & traces
› and for assertions, including temporal
› equivalence under refactoring



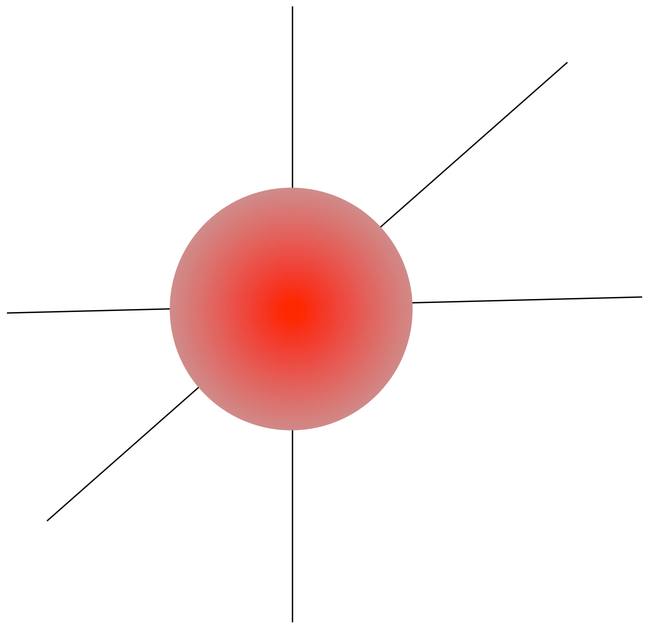
#3: counterexamples & scope

observations about analyzing designs

- most assertions are wrong
- most flaws have small counterexamples



testing:
a few cases of arbitrary size



scope-complete:
all cases within small scope

#4: analysis by SAT

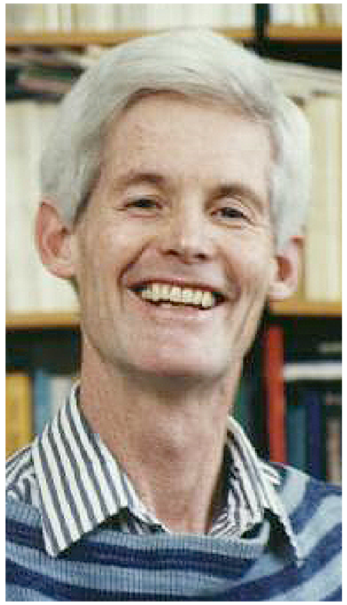
SAT, the quintessential hard problem (Cook, 1971)

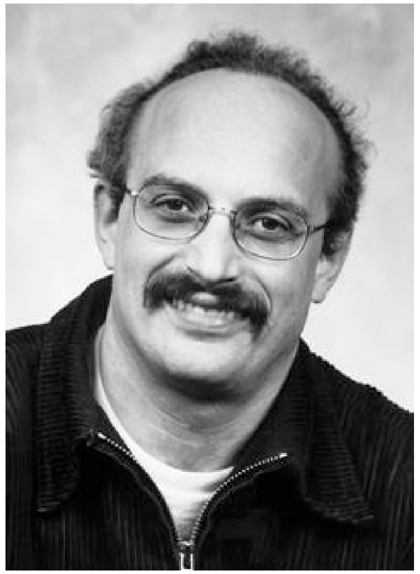
› SAT is hard, so reduce SAT to your problem

SAT, the universal constraint solver (Kautz, Selman et al 1990's)

› SAT is easy, so reduce your problem to SAT

› solvers: Chaff (Malik), Berkmin (Goldberg), many others


Stephen Cook


Eugene Goldberg

experience with alloy

alloy case studies at MIT

many small case studies
› intentional naming [Balakrishnan+]
› Chord peer-to-peer lookup [Kaashoek+]
› Unison file sync [Pierce+]
› distributed key management
› beam scheduling for proton therapy

typically
› 100-1000 lines of Alloy
› analysis in 10 secs - 1 hour
› 3-20 person-days of work

sample alloy applications

in industry

› animating requirements (Venkatesh, Tata)
› military simulation (Hashii, Northtrop Grumman)
› role-based access control (Zao, BBN)
› generating network configurations (Narain, Telcordia)

in research

› exploring design of switching systems (Zave, AT&T)
› checking semantic web ontologies (Jin Song Dong)
› reference model for ODP (Naumenko)
› checking refinements (Bolton, Oxford)
› security features (Pincus, MSR)

alloy in education

courses using Alloy at Michigan State (Laura Dillon), Imperial College (Michael Huth), National University of Singapore (Jin Song Dong), University of Iowa (Cesare Tinelli), Queen's University (Juergen Dingel), University of Waterloo (Joanne Atlee), Worcester Polytechnic (Kathi Fisler), University of Wisconsin (Somesh Jha), University of California at Irvine (David Rosenblum), Kansas State University (John Hatcliff and Matt Dwyer), University of Southern California (Nenad Medvidovic), Georgia Tech (Colin Potts), Politecnico di Milano (Carlo Ghezzi), Rochester Institute of Technology (Michael Lutz), University of Auckland (John Hamer, Jing Sun), Stevens Institute (David Naumann), USC (David Wilczynski)

your homework for today

homework: handshaking

Paul Halmos's handshaking problem

- Alice and Bob invite four couples for dinner. When they arrive, they shake hands. Nobody shakes their own or spouse's hand. After some handshaking, Alice asks how many hands each person has shaken. All the answers are different. How many hands has Bob shaken?

solution

```
module examples/handshake/handshake
sig Person {spouse: Person, shaken: set Person}
fact {
   no (iden + spouse) & shaken
   shaken = ~shaken
   spouse = ~spouse
   no iden & spouse
   }

pred solve () {
   some Alice: Person |
       no disj a,b: Person - Alice | #a.shaken = #b.shaken
   }

run solve for exactly 10 Person
```

your task

find the solution by running Alloy

look at the different visualizations

see what happens if you delete some facts

extra slides

why not model with code?

nothing, so long as

› you're prepared to start again
› you like reading other people's code
› you like writing test cases

kent beck on design notation

Another strength of design with pictures is speed. In the time it would take you to code one design, you can compare and contrast three designs using pictures. The trouble with pictures, however, is that they can't give you concrete feedback. The XP strategy is that anyone can design with pictures all they want, but as soon as a question is raised that can be answered with code, the designers must turn to code for the answer.
The pictures aren't saved.

Kent Beck
Extreme Programming Explained, 1999