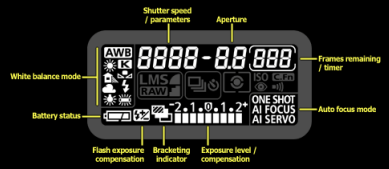
CONCEPTUAL MODELS & MODERN SOFTWARE CONSTRUCTION

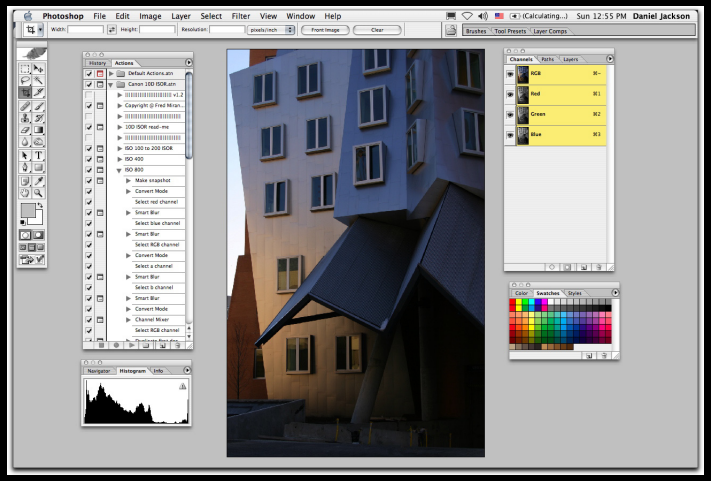
Daniel Jackson · Computer Science & Artificial Intelligence Lab · MIT

what makes software different?

› ISO speed
› shutter speed
› aperture
› focus
› depth of field

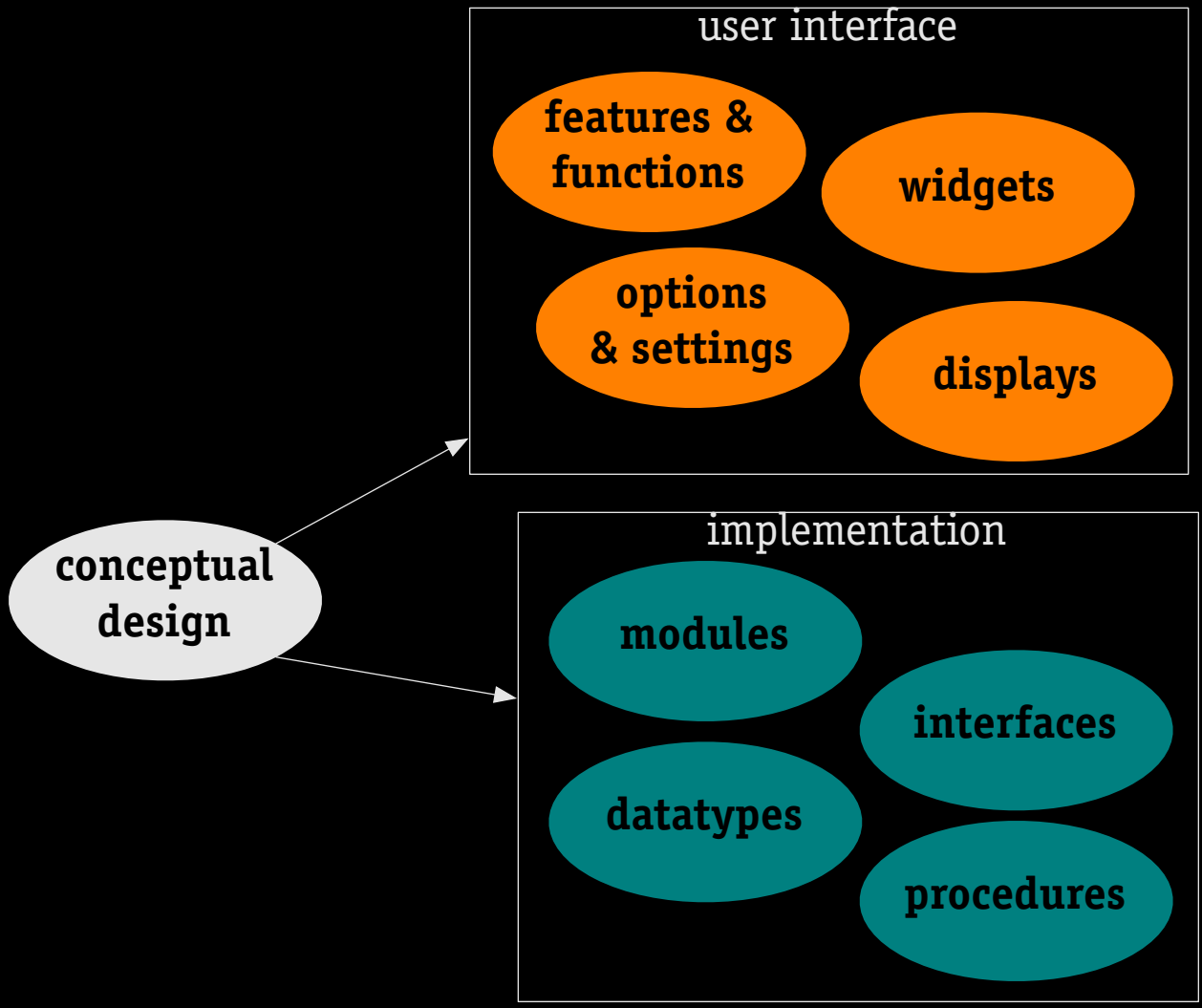


› ISO speed
› shutter speed
› exposure
› aperture
› depth of field
› white balance (4pp)
› autofocus
› metering mode
› drive mode
› bulb mode
› image playback
› image protection
› direct printing
› date and time



› selection (21pp)
› layers (43pp)
› channels & masks
› shapes, paths & pens
› filters & sharpening
› color spaces
› history, actions
› painting & brushes

impact of conceptual design



ease of use
flexibility
robustness

clean interfaces
decoupling
extensibility
dependability

fred brooks on conceptual integrity

I will contend that conceptual integrity is the most important consideration in system design. It is better to have a system omit certain anomalous features and improvements, but to reflect one set of design ideas, than to have one that contains many good but independent and uncoordinated ideas. *1975*

I am more convinced than ever. Conceptual integrity is central to product quality. *1995*

example: formatting text

Preface

Introduction

In every large software system, there is a small model trying to get out. It's the model that you'd get if you cleared away all the clutter – all the irrelevant details, unused features, performance hacks and workarounds. It captures the essence of the system – what it's about, its key concepts and how they fit together.

Rationale

If the designers had written it down, the maintainers wouldn't need to struggle through the source code, putting the model together like a jigsaw puzzle. The testers would know where weak spots in the implementation are likely to be, and wouldn't have to fumble in the dark. The users wouldn't have to learn the system function by function, because the authors of the user manual would have seen more clearly what all the functions had in common.

Preface

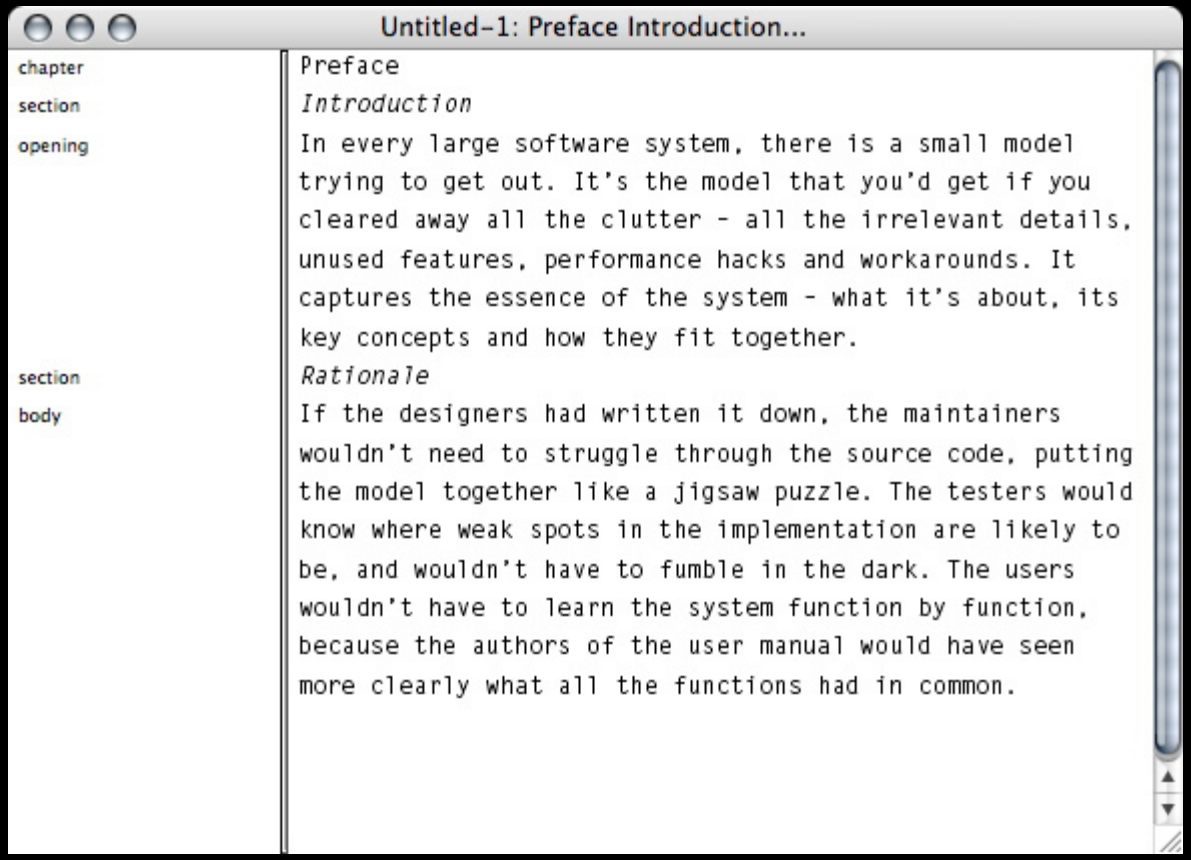
Introduction

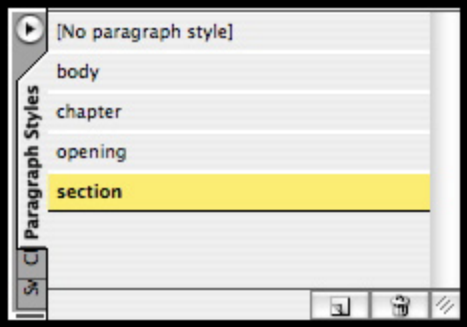
In every large software system, there is a small model trying to get out. It's the model that you'd get if you cleared away all the clutter – all the irrelevant details, unused features, performance hacks and workarounds. It captures the essence of the system – what it's about, its key concepts and how they fit together.

Rationale

If the designers had written it down, the maintainers wouldn't need to struggle through the source code, putting the model together like a jigsaw puzzle. The testers would know where weak spots in the implementation are likely to be, and wouldn't have to fumble in the dark. The users wouldn't have to learn the system function by function, because the authors of the user manual would have seen more clearly what all the functions had in common.

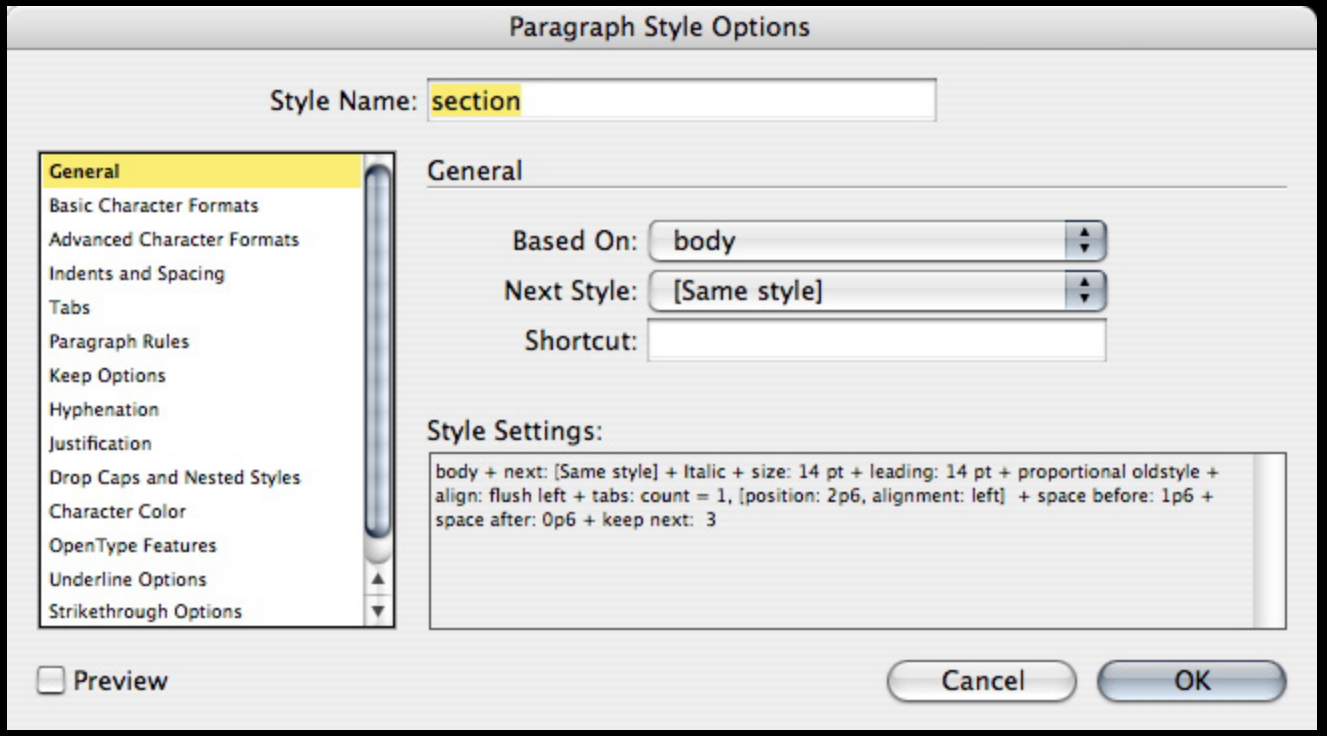
a document marked up with styles





first use of style sheets in a text formatter:
Bravo, Xerox PARC (1973-79)
Butler Lampson and Charles Simonyi

a style sheet dialog



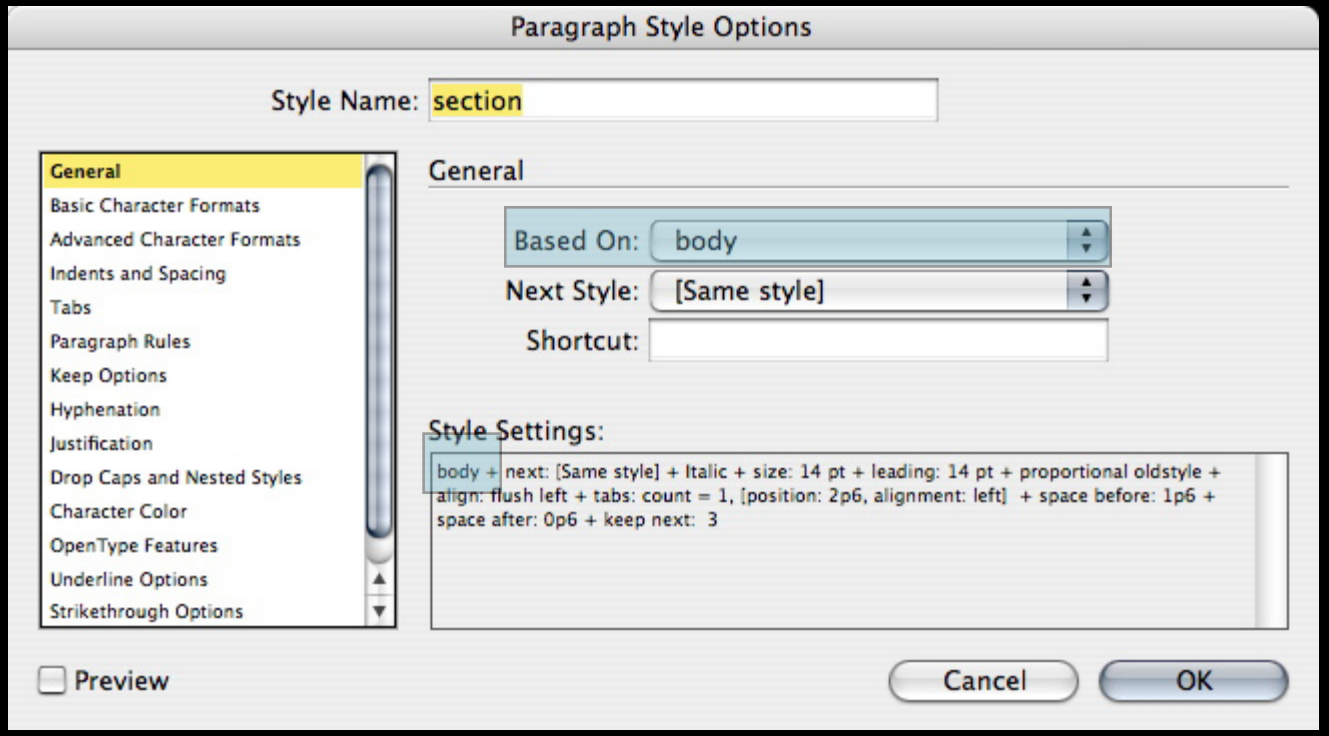
There is no problem in computer science that cannot be solved by an extra level of indirection. *--David Wheeler*

style hierarchy

for consistent formatting

› styles arranged in hierarchy, and inherit properties

› change to parent affects child automatically



from microsoft.public.office.word

Certain styles (Heading 1 to 3, Normal...) must exist in the document structure, so they always appear. Word can't let you delete them because the document binary structure would implode if it did.

I have been writing my thesis paper and it has grown to be about 100 pages. The problem that I am having is with the styles preferences... The problem I am having is if I choose and highlight maybe two words to bold, the entire document becomes bold.

I'm having problems with Word 2002 automatically creating new styles. For example, I have a style called "Figures", but for some reason Word has created another style called "Figures Char Char3 Char Char Char Char Char1".

First, make sure you have updated Word 2002 with at least SP-1. That solved a lot of the erroneous Char Char Char problems. Second, it's worth understanding how these 'char' styles are created. In Word 2002, you can have a paragraph in, say, Body Text. If you select *part* of that paragraph and apply a different paragraph style (say, Style 2), then Word creates a kind of hybrid, called Style 2 Char. It's part-paragraph style and part-character style. This only happens if, when you applied the style, some characters were selected, but not the whole paragraph including the paragraph mark.

You can get very bizarre behavior when importing one style that is based on another style without importing the based-on style. This won't be a problem when you update all styles based on a newly attached template. Even if you import the whole set, you will get anomolies unless you import the set multiple times. I import (copy) three times.

I'll look some more, but it seems that the "RedefineStyle" command is buggy in Word2002/2003. Redefining a style shouldn't touch manual formatting. But it seems that "RedefineStyle" removes all manual formatting from all paragraphs formatted in that style. It sure didn't work like this up to Word2000, and whoever thought it a good idea to change this must have ample access to psychedelic chemicals.

conceptual integrity: how?

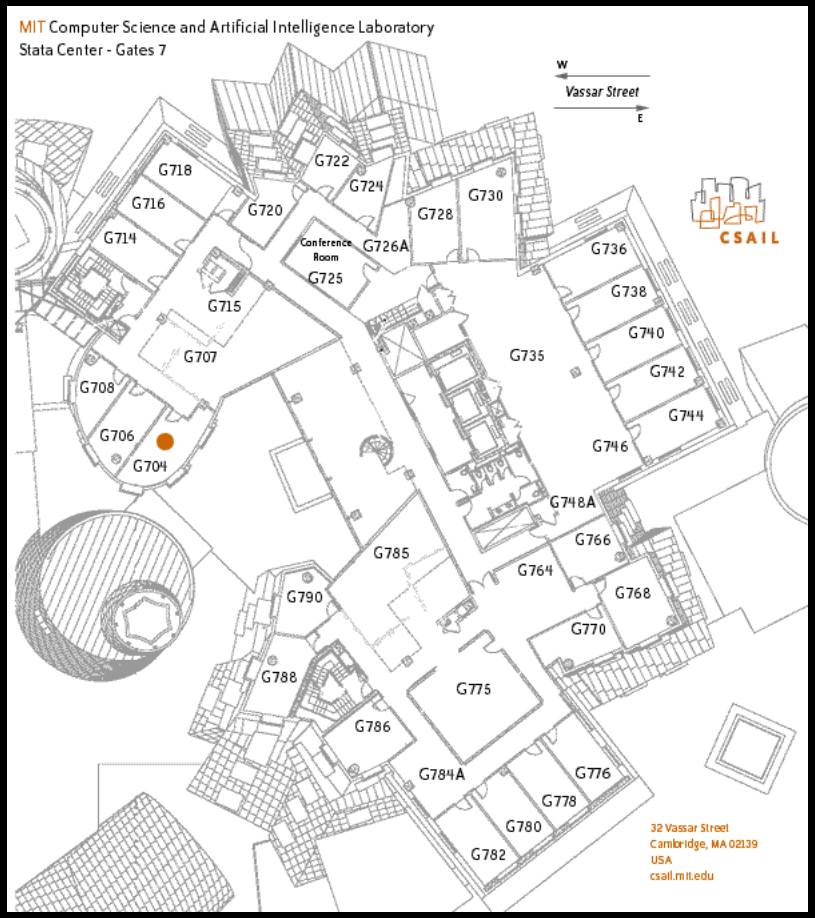
Fred Brooks
› one system architect
› plan to throw one away

extreme programming
› system metaphor

why not focus on concepts?
› express, analyze, explore

traditional engineering models

traditional engineering models



software models: desiderata

focus
› risk-driven, so partial
› separation of concerns
› tiny compared to code

language
› graphical & textual
› small & uniform
› flexible & expressive

analyzability
› deep semantics, not just syntax
› simulation & property checking

alloy: a new approach to modelling

language and analysis designed hand-in-hand
› maximize expressiveness without losing tractability

everything's a constraint
› state invariants, operations, properties

a kernel relational logic
› simple & uniform
› no programming language notions
› no harder to grasp than SQL, eg

no hard-wired idioms
› good for many kinds of model
› can tailor to your own style

what drives modelling in alloy

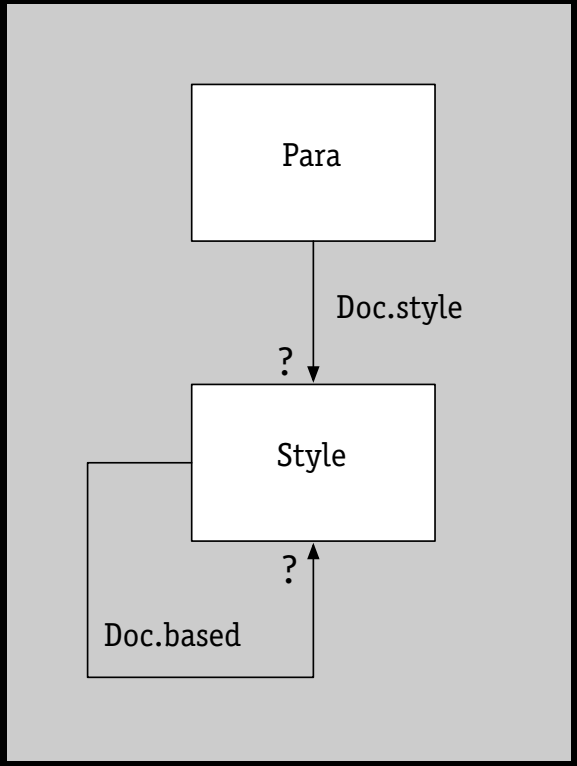
show me
› a state meeting these constraints …
› a state breaking these constraints …
› an execution of this operation …
› an execution of this operation that breaks this constraint …

Look Ma, no test cases!

incremental & partial

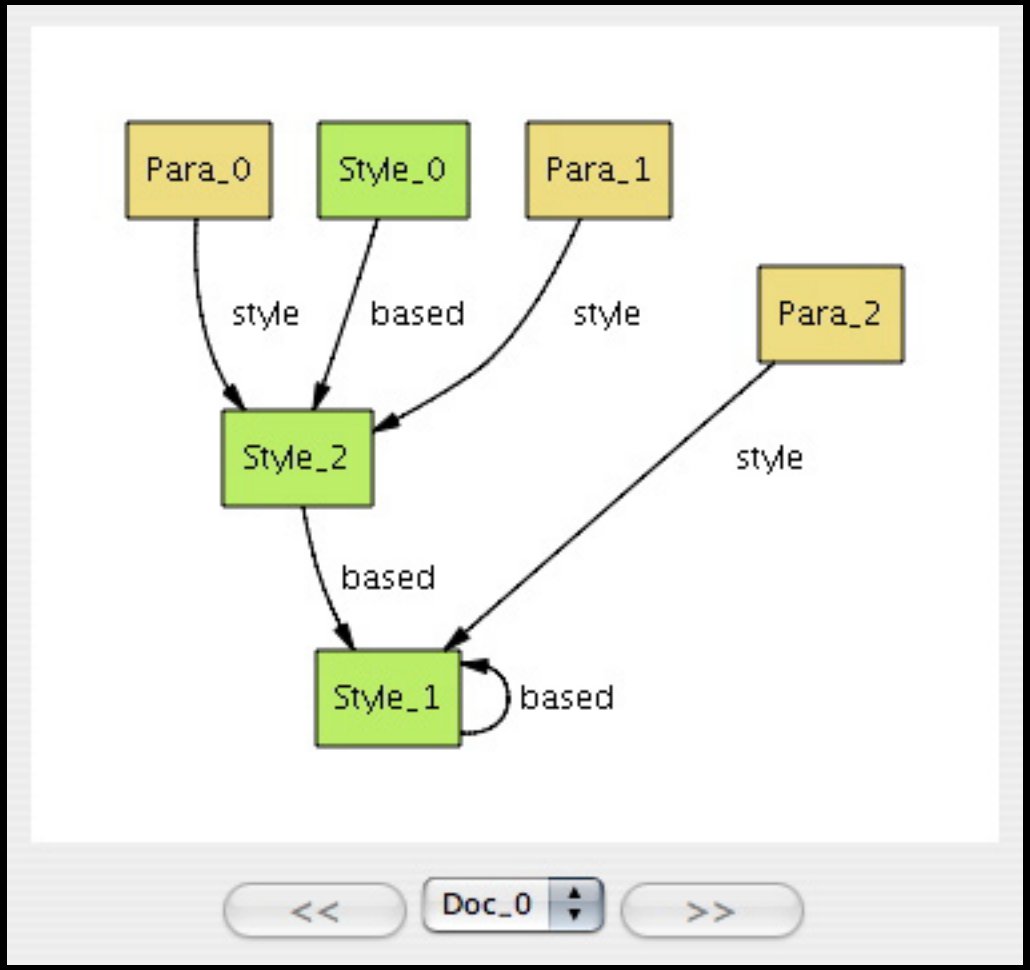
styles & paragraphs

```
module styles_1

sig Para, Style {}

sig Doc {
  based: Style ->? Style,
  style: Para ->? Style
  }
```



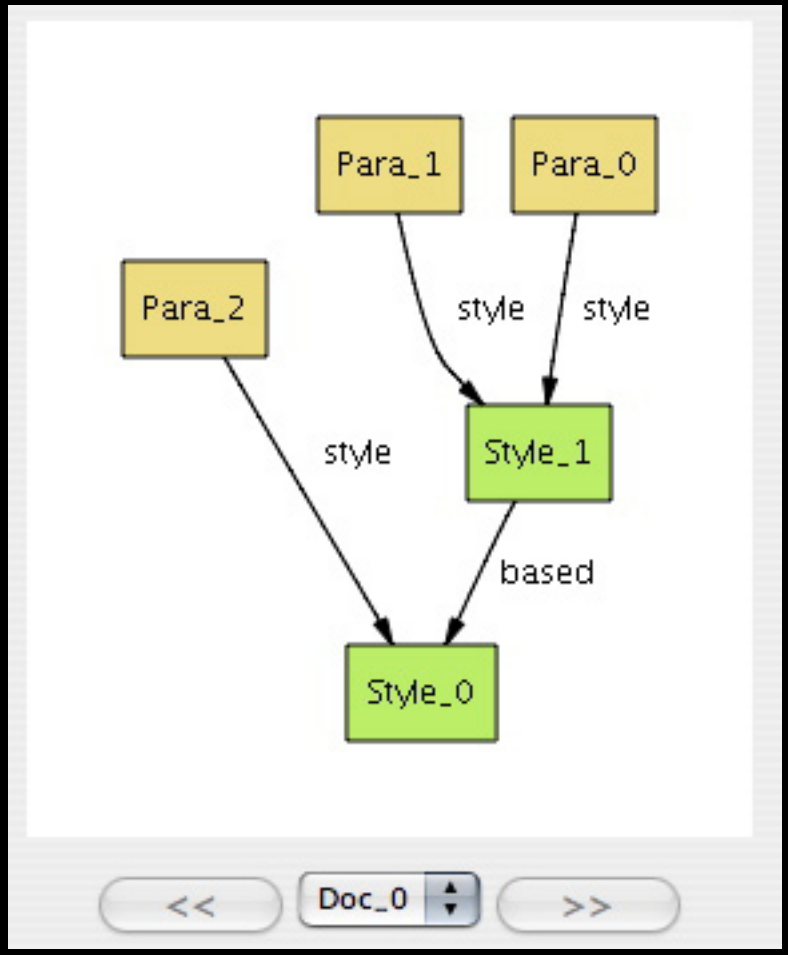
styles & paragraphs

```
module styles_1

sig Para, Style {}

sig Doc {
  based: Style ->? Style,
  style: Para ->? Style
  }

pred show () {}
run show for 3 but 1 Doc
```

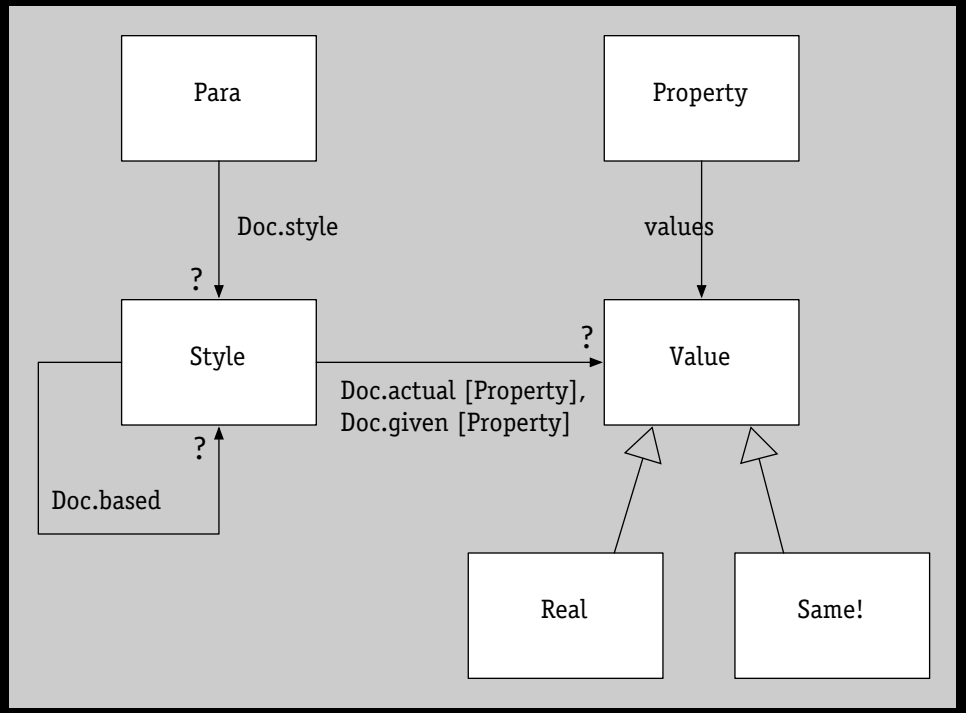


styles & paragraphs

```
module styles_1

sig Para, Style {}

sig Doc {
  based: Style ->? Style,
  style: Para ->? Style
  }

pred WellFormed (d: Doc) {
  acyclic (d.based)
  }
run WellFormed for 3 but 1 Doc
```



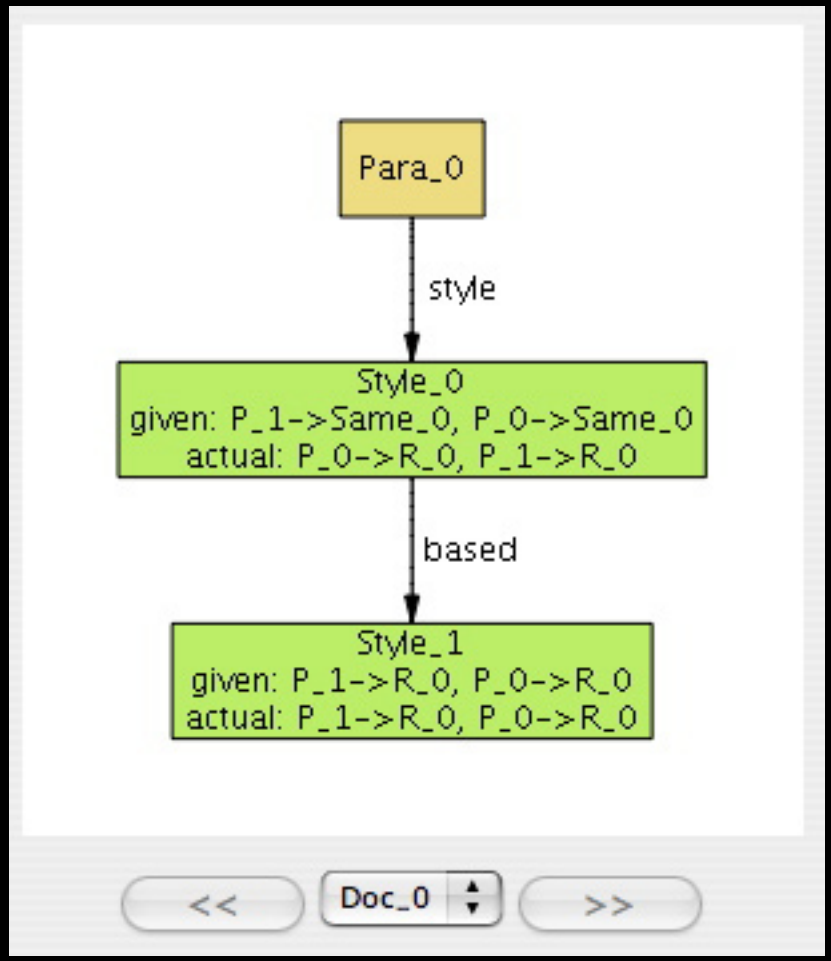
adding value




```
module styles_1

sig Para, Style {}
sig Property {}
abstract sig Value {}
static sig Same extends Value {}
sig Real extends Value {}

sig Doc {
  based: Style ->? Style,
  style: Para ->? Style,
  actual, given: (Style -> Property) ->? Value
  } {
  all s: Style, p: Property {
    some s.given[p]
    s.actual [p] = if s.given [p] in Same then s.based.actual [p] else s.given [p] }
  }
```

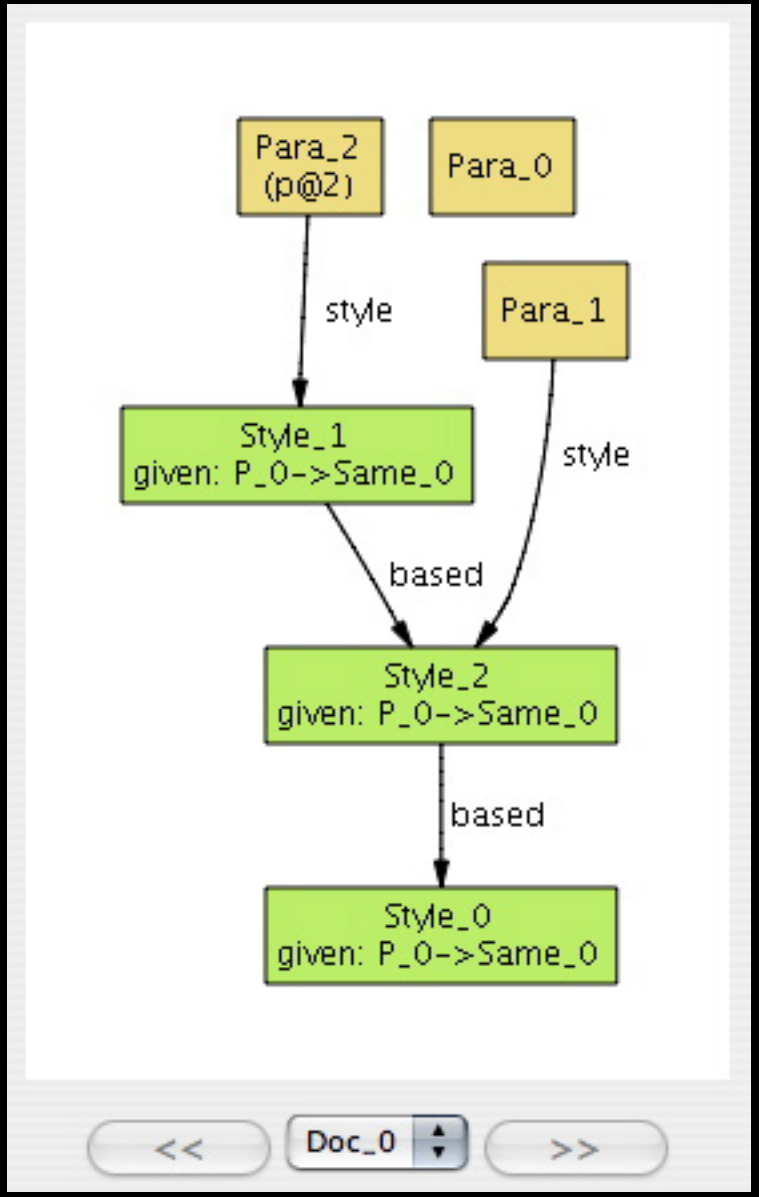
adding value

```
module styles_1

sig Para, Style, Property {}
abstract sig Value {}
static sig Same extends Value {}
sig Real extends Value {}

sig Doc {
  based: Style ->? Style,
  style: Para ->? Style,
  actual, given: (Style -> Property) ->? Value
  } {
  all s: Style, p: Property { some s.given[p]
    s.actual [p] = if s.given [p] in Same
    then s.based.actual [p] else s.given [p]}}

pred WellFormed (d: Doc) {acyclic (d.based)}

run WellFormed for 3
    but 1 Doc, exactly 2 Property
```



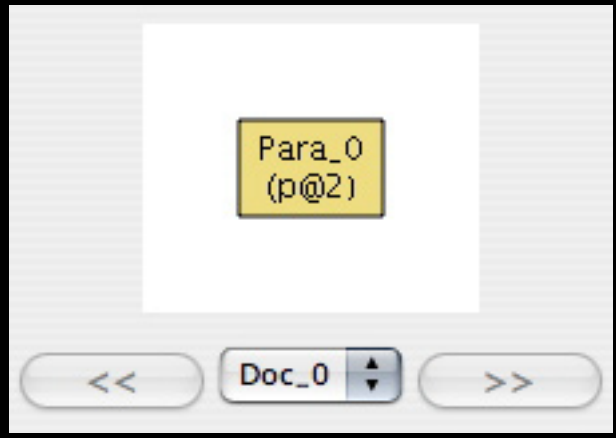
are paragraphs formatted?

```
module styles_1

sig Para, Style, Property {}
abstract sig Value {}
static sig Same extends Value {}
sig Real extends Value {}
sig Doc {
  based: Style ->? Style,
  style: Para ->? Style,
  actual, given: (Style -> Property) ->? Value
  } {...}
pred WellFormed (d: Doc) {acyclic (d.based)}

pred AllParasFormatted (d: Doc) {
  all x: Property, p: Para |
    some d.actual[d.style[p]][x] & Real
  }

assert DocOK {
  all d: Doc | WellFormed (d) => AllParasFormatted (d)}
check DocOK for 3 but 1 Doc
```



attempting a fix

```
module styles_1
sig Para, Style, Property {}
abstract sig Value {}
static sig Same extends Value {}
sig Real extends Value {}
sig Doc {
  based: Style ->? Style,
  style: Para ->? Style,
  actual, given: (Style -> Property) ->? Value
  } {...}

pred WellFormed (d: Doc) {
  acyclic (d.based)
  all s: Style |
    no s.(d.based) => s.(d.given) [Property] in Real
  }
pred AllParasFormatted (d: Doc) {
  all x: Property, p: Para | some d.actual[d.style[p]][x] & Real }

assert DocOK {
  all d: Doc | WellFormed (d) => AllParasFormatted (d)}
check DocOK for 3 but 1 Doc
```

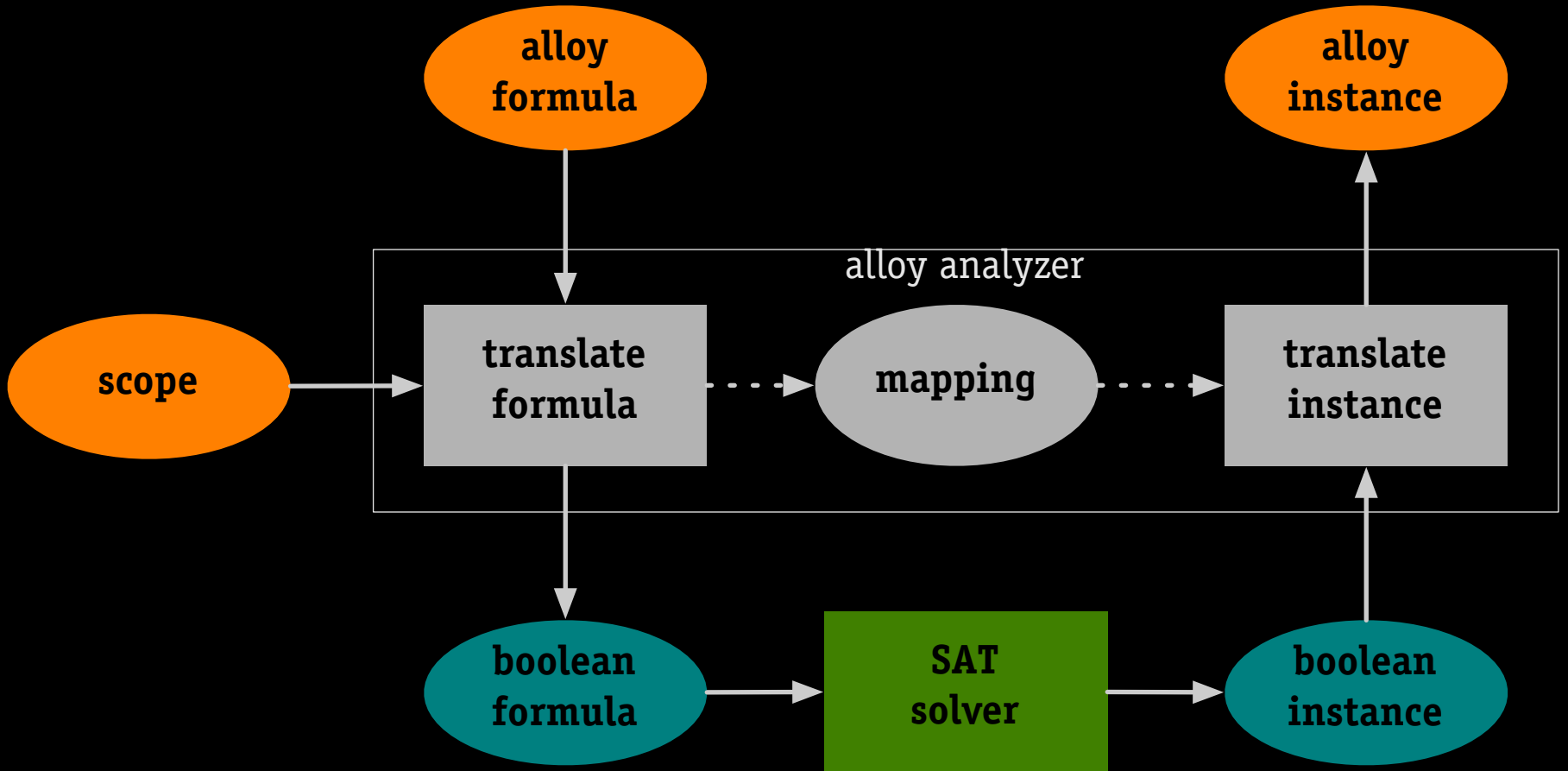


closing comments on style model

where to go from here?
› modelling paragraph overrides
› specify and analyze deleteStyle operation

typical modelling experience?
› yes: small model, incrementality, quick feedback, hard questions
› no: usually more clumsy, not liner

how alloy works



why alloy works

small scope hypothesis
› most interesting cases can be shown with small numbers of objects
› dense analysis works better than random testing

random testing
› check some large cases picked randomly

dense analysis
› check every case up to a certain size

example
› 3 styles, 3 paras, 3 properties, 3 values, 2 documents
› 2^90 (10^30) values for relations
› can analyze in seconds

experience with alloy: case studies

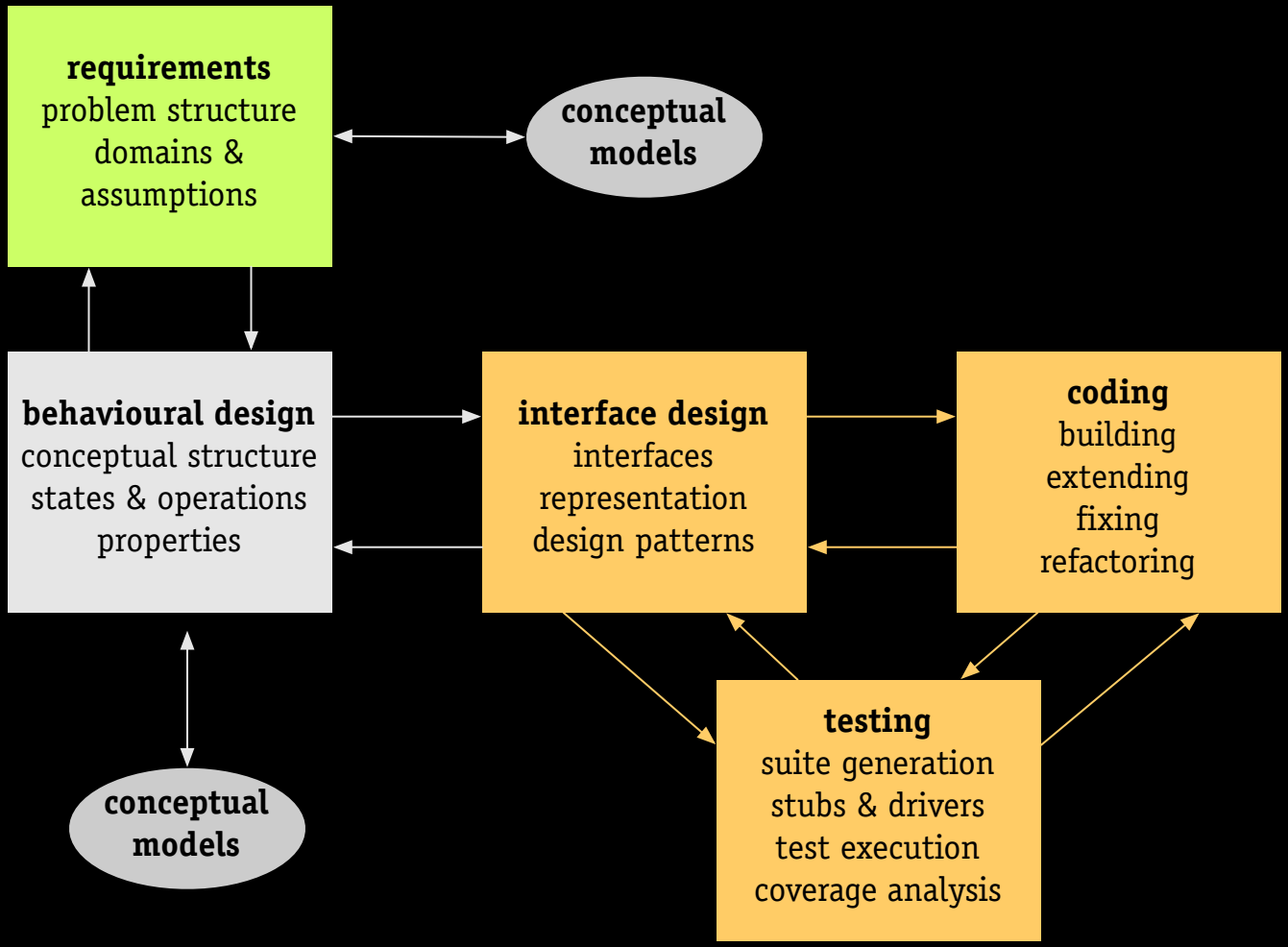
- firewire configuration protocol
- unison file sychronizer
- IMPP presence protocol for instant messaging
- query interface in COM
- key distribution for multicast
- intentional naming
- Chord distributed hash table
- role-based access control
- web ontologies
- military simulation
- telephone switch feature configuration
- proton beam scheduling

experience with alloy: education

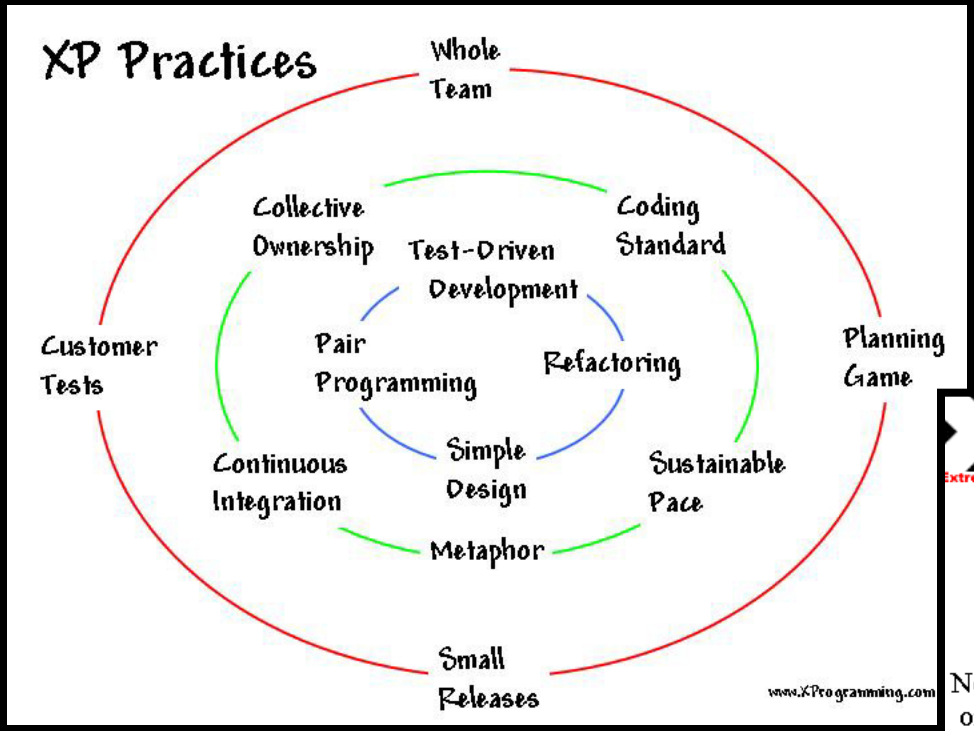
typical learning experience
› a few years of programming
› minimal background in discrete math
› writing small alloy models in a week
› modelling with confidence in a month

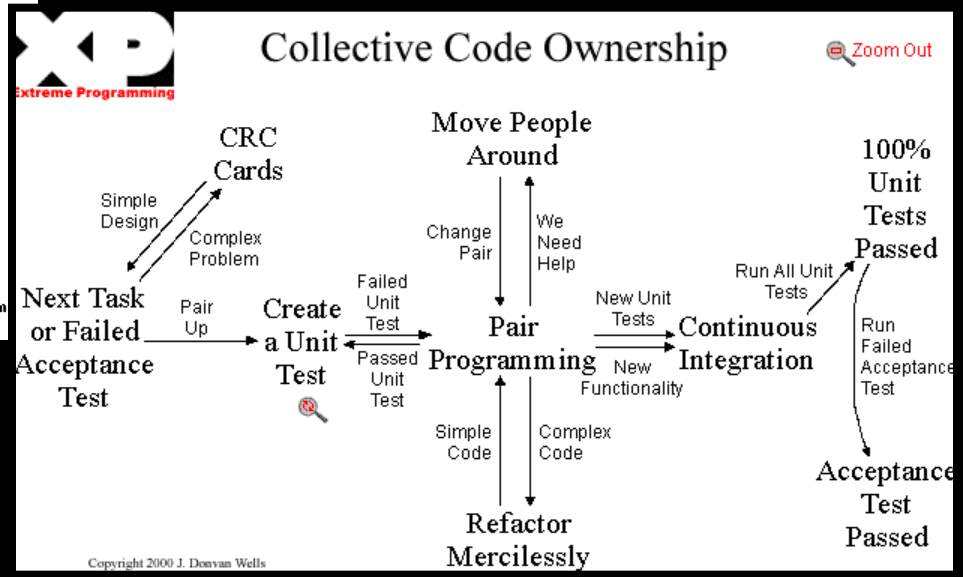
course usage
› taught in >12 universities worldwide
› mostly in masters of software engineering degrees

conceptual modelling in context



extreme programming





from extremeprogramming.org, xprogramming.com

extreme programming and design models

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 (2000)*

why XP needs design models

can't always refactor your way to design quality
› interface design can evolve (on small projects)
› conceptual design changes are painful

coding isn't the right way to answer design questions
› tedious, irrelevant details to complete
› test cases are hard work to write, give poor coverage
› code is not so amenable to Alloy-like analysis

how to get going

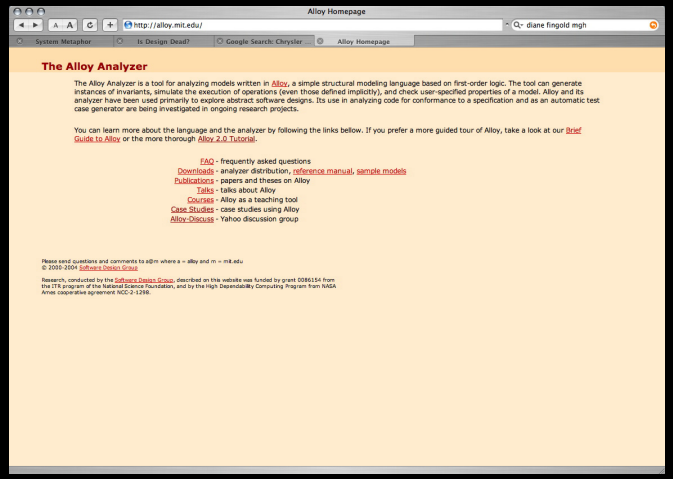
pick the right problem
› small enough to be manageable
› hard enough to be interesting
› pressing enough to be worth the trouble
› not so pressing you can't experiment

try alloy
› simulate some small object models
› customize diagrams to give intuitive appearance

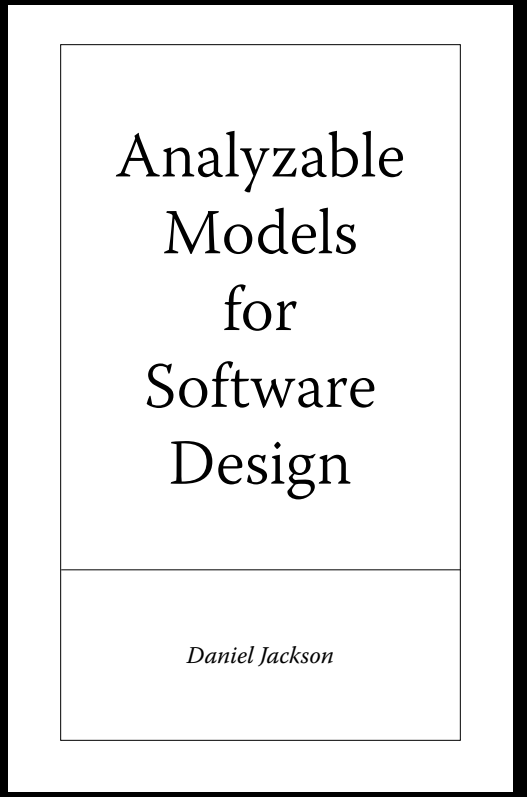
contact us if you need help
› dnj@mit.edu
› alloy@mit.edu, alloy-discuss@yahoogroups.com

for more information

website
› case studies
› courses
› tutorial
› downloads



good books
› Martin Fowler, *Analysis Patterns*
› William Kent, *Data and Reality*
› Michael Jackson, *Software Requirements and Specifications*



upcoming book
› Fall 04