DESIGN VISUALIZATION: SMOKE AND MIRRORS

A rethinking of software architecture, its place in software design, and how we approach it

Going to bat for wisdom, and

Putting visualization back in the tool belt

by:
Ruth Malan
Bredemeyer Consulting

October 14, 2015
Introduction

- Introduction
- Theme of talk
- Outline of talk
These are the slides and notes from my Design Visualization: Smoke and Mirrors talk at the Software Architect Conference in London in October, 2015.

This is the part where I tell you "This flight is going to Indiana by way of South Africa. If you’re on the wrong flight, this would be a good time to attend one of the awesome talks down the hall."

My name is Ruth Malan -- I don’t know how to pronounce it either, so any way you say it is fine.

Well, as long as you’re staying, let me tell you that Dana Bredemeyer says “There is a silver bullet in software engineering, and it is relationships of goodwill and a commitment to objectivity.” There’s a catch though. Those relationships take effort and attention to build and to sustain and grow.

So here I am, relying on you to extend a little goodwill on credit to me, and as we proceed, I hope I’ll earn a little more, as we talk about some ways we can work on objectivity.

I’ll try to keep the notes close to what I did or intended to say...
Imagine you are looking at M.C. Escher’s Day and Night.

The M.C. Escher Company holds and protects their copyrights to M.C. Escher’s work quite tenaciously. Not being able to reproduce Day and Night for you here, presents a little opportunity to demonstrate a point at the heart of this talk.

Do you see it in your mind’s eye? It is the one where there is a town in the light, and a mirror image in the dark of night. A river runs by, in the light, and the night. There are fields that become abstractions that become duck (or geese), flying <gestures a mesh with interwoven fingers of both hands> from the light, others in mirror image out of the night.

When you called it to mind, perhaps at first you were thinking of another Escher, or perhaps it struck you differently and you remembered different features.

And that’s the point, really. We don’t know if what I call to mind, and see in my mind’s eye is quite what you see, even if the original was the same image. Our software systems are so large, we can never see them in their entirety. Our mental models are central. But idiosyncratic and fallible.

It is worth noticing how different our mental models (recollection of “fact” and making of meaning) can be, even when we are referring to a picture held in memory. Our software systems are so large, we can never see them in their entirety. Our mental models are central. But idiosyncratic and fallible.
That is where we are headed -- to talk about design as intention, and reflection of design as realized, and how we express and envision our designs. But first I would like to revisit how we think of software architecture, and the role visual expression plays in design.
Software Architecture

- Intention and Reflection
- Technical Wisdom
- Structurally and Strategically Significant
- Minimalist
- Intentional and Emergent
Intention and Reflection.
Or Wizardry.

I playfully subtitled the talk “Smoke and Mirrors” -- for it can seem like we practice magic and sleights of hand when we don’t expressly communicate our design intention, or reflect on the design as realized, in order to iterate on and improve our understanding of what we have, and intend.
The Devil’s Dictionary on Intention and Reflection

Here we have the definitions of intention and reflection from Ambrose Bierce’s *Devil’s Dictionary*. If you’re not familiar with the Devil’s Dictionary, it gives definitions a cynical side-eye, helping us understand something better by looking at the (negative) space left by contact of the concept with reality, and what that defines. Intention, then, is what we get, despite our intention -- an involuntary act. It is a big wink at our bias and foible and self delusions and intellectual aggrandizements.

As for reflection: “An action of the mind whereby we obtain a clearer view of our relation to the things of yesterday and are able to avoid the perils that we shall not again encounter.” That so well captures the weltanschauung of our day and field -- we forget, of course, that it was Heraclitis who said “Change is the only constant.”

Now, when it comes to definitions I’m on the same page as Richard Feynman when he said: ‘We can’t define anything precisely. If we attempt to, we get into that paralysis of thought that comes to philosophers… one saying to the other: “you don’t know what you are talking about!”’. The second one says: “what do you mean by talking? What do you mean by you? What do you mean by know?”

Undeterred, we’re going to take a look at the definition of software architecture.
This is the definition from the touchstone of our era, wikipedia.

When I see a definition, especially in a context like this, I tend to see it as words words words... words words. So I helped a little to lead attention from software architecture to system.

What do we mean by system?
System

Dana Meadows characterized a system as "an interconnected set of elements that is coherently organized in a way that achieves something."

System: What?

“A system is an interconnected set of elements that is coherently organized in a way that achieves something”
-- Donella Meadows
Software Architecture

Returning to Paul Clements and colleagues definition on wikipedia, we note "high level structures" -- comprising elements and relations, along with properties.

High level?
Grady Booch memorably observed "All architecture is design, but not all design is architecture. Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change."

-- Grady Booch, blog post, March 2, 2006
Decisions!

We know what to do with decisions! We'll name them, describe them, identify what problem the decision addresses and the forces we're weighing to harmonize, resolve and balance. We'll keep track of assumptions we're making, explicitly identifying what in the context we presume to be stable and what to watch.

We'll connect the dots to the business rationale and technical goals we're trying to achieve, or the hard-(l)earned scars of experience we're seeking to avoid this time round. We'll outline alternatives we considered but ruled out, so we don't have to revisit those arguments again and again. And we'll note implications so we -- our teams and those we collaborate with -- can be prepared. Then we'll consider what further requirements are implied by, or derive from this decision we're making and documenting. And identify related decisions we need to make next...

Decisions are central, and it is a great template, but you can just hear the captain in the cockpit yelling "pull up, pull up" -- we'll run into a veritable forest of decision trees if we speed too far too fast down that runway just now.
Architecture: Again?

"All architecture is design but not all design is architecture. Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change."

-- Grady Booch, blog post, March 2, 2006

Which decisions?

Which decisions? Design decisions.

Design!
Design?

"Everyone designs who devises courses of action aimed at changing existing situations into preferred ones."

-- Herbert Simon

Herbert Simon, according to Jabe Bloom, is "ground zero in design discourse." And this is quintessential Herb: "Everyone designs who devises courses of action aimed at changing existing situations into preferred ones."
Design?

"The engineer, and more generally the designer, is concerned with how things ought to be - how they ought to be in order to attain goals, and to function."

-- Herbert Simon

More Herb on Design

A variation on the theme: "The engineer, and more generally the designer, is concerned with how things ought to be - how they ought to be in order to attain goals, and to function."
Design?

We design to get more what we* want.

* For some value of we(tbd)

And that is just one of those so precious moments when the world is full of simple wonder again -- we design to get more what we want.
"All architecture is design but not all design is architecture. Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change."

-- @Grady_Booch, blog post, March 2, 2006

\textit{Which decisions?}

Returning to the eminent Mr. Booch: "Architecture represents the significant design decisions"

Significant!
Architecturally Significant?

“What decisions does the software architect make?”
Architecturally significant ones.
“What is architecturally significant?”
The architect decides!

Architecturally Significant?

“What decisions does the software architect make?”
The architecturally significant ones.

“What is architecturally significant?”
The architect decides!

The architect decides. That sounds like a tautology, but it really is the crux of the matter.
Is this the part where we get to talk about *Technical Wisdom*?! 

[insert obligatory Yoda image]
Is this the part where we get to talk about *Technical Wisdom*?!  

No, this is the part where we get to say “Awww”  

Just kidding.
Architecturally Significant?

"the architecture of a system is the naming of the most significant design decisions that shape a system, where we measure significant by cost of change and by impact upon use."

– @Grady_Booch

Technical wisdom factors. Factors in and factors out. What is shapingly crucial?
What decisions are architecturally significant?

Ok, so if architecture is a set of decisions, but not all decisions, and we’re asking which belong in the architecture, we’re looking for those that shape, that

• give form to the system
• set direction
• constrain
• bring integrity and consistency

And those that have the highest cost of change. Decisions which would incur substantial

• resources and time
• operational downtime
• deferral of value
• emotions and politics
to change or revert and rework, are architecturally significant.

Also, if a decision meaningfully reduces the cost of making changes, enabling our business to be more fleet, adapting and extending its services or product set as the market shifts, it is architecturally significant.
Architecturally Significant?

Structurally significant
- Organizing structure
- Architecturally significant mechanisms
- Structural integrity and sustainability

Strategically significant
- Game shapers and game changers

Architecture decisions are those that are:
- Make or break!
- Game shapers and game changers!

Strategically and Structurally Significant

Those are important insights, but I would like to add: architecture decisions are structurally significant. They deal with the organizing structure of the system and the design of architecturally significant mechanisms to yield desired system outcomes, including system properties, while addressing inherent challenges. Structurally significant. You know, make or break.

And strategically significant. "Software is eating the world" (Marc Andreessen). Software enables. And more, across industries, software is increasingly a source, if not the source, of differentiation. It is game shaping and game changing.
Architecture

Architecture enables and constrains

Minimalist

— Dana Bredemeyer

Creating Ground Under the Feet

Software systems are in place -- for 3 months, 3 years, 10 years. 20. Because they enable something our business depends on. But, increasingly, and more as systems age and the architecture erodes under the weight of accommodations and agglomerations, these systems constrain the business, impede agility and adaptability and responsiveness to an ever shifting context.

And architecture, the critical decisions that hold the system up and tie it down, in turn enables and constrains the code (and those who write it).

Architecture decisions create the context for further decisions, reducing -- cleaving -- the decision space. This is good. It reduces the overload of overwhelming ambiguity and uncertainty, creating "ground under the feet" (Dana Bredemeyer) that we can move forward on. Critical decisions take time to make attentively and can be fraught with downstream consequences if made inattentively and without foresight. And this is bad, if it takes decisions away, reduces empowerment or degrees of freedom and motivation, where it matters. So we seek to keep our architecture decisions, in Dana Bredemeyer's urging, to a minimal set.

Less is More with Minimalist Architecture
To Do Big Things, When We Need To

To reach a helpful notion of what, then, we consider architecturally significant, Dana points to Daniel Day-Lew--er(r) Abraham Lincoln:

"The legitimate object of government is to do for a community of people whatever they need to have done, but cannot do at all, or cannot so well do, for themselves – in their separate, and individual capacities."

– Abraham Lincoln
Architecture decisions are those that impact system outcomes. That is, outcomes that can’t be ascribed to locally-scoped parts of the system. Intentional architecture decisions are those, from a standpoint of experience, we deem must be made from a system perspective, to get more what we want from the system. More than we would get, if we left the decision to be made at a more narrow scope with only local information about the forces that impinge upon, and outcomes that are contingent on, the decision.

The Arc of Architecture

That is, architecture decisions are those that need to be made across boundaries - the system boundary, and boundaries within the system, in order to achieve desired system outcomes -- to meet system goals with more the system properties we want. Properties like usability or performance that the user cares about. Properties like resilience and scalability that the ops team cares about. Properties like understandability and adaptability that the dev team cares about (or will, as these properties are compromised). Properties that emerge from interactions and relations among elements, rather than localized concerns.

Architects need to take a broader perspective -- across the system, across stakeholders, and a longer time horizon. But we do just enough, to achieve concert and coherence, structural and design integrity, to get more the outcomes we seek.
Some part intention,
Some part emergence

Of course, despite our best intentions, some implicit decisions will prove to be architecturally significant.

The point is, do we want to leave matters of system integrity and strategic import to accident, or do we want to bring what we can to bear, to get more what we want?
Visual Design

Sketching in Engineering: Leonardo da Vinci
Visualization in Other Fields
What’s Missing from Code
Why We Model

A Change in Perspective
Mechanisms
Wisdom and Fallibility
Extraordinary Moment Principle
Coincidence?

When I googled to find an image of Grady Booch for the previous slide, this was one of the results. It's on an IBM site related to a conference, and next to the image of Da Vinci and Booch there is the playful caption “separated at birth?”

The relationship, at least in my view, has to do visual design, for Grady Booch is one of the fathers of visual design in software. While Leonardo da Vinci is one of the fathers of visual design in engineering.

As we turn our attention to how we design systems to get more what we want, we’re going to look at a great engineer-designer for inspiration.
Serendipity

Anyway, I thought the image was a neat serendipity because when I was collaborating with Grady Booch on software visualization several years ago, Grady introduced me to this book: *Engineering and the Mind’s Eye*. Which in turn introduced me to Da Vinci as engineer.

Of course, I was familiar with Da Vinci’s work as artist -- even privileged to see his cartoon (as a full-size preparatory study for a painting is known) of the *Virgin and Child* (with St Anne and St John the Baptist) in the National Gallery. I’d seen his sketches of imaginative flying machines and was aware of his notebooks.

“Engineering and the Mind’s Eye” comes highly recommended.
Pushing boundaries of science and engineering, not just art

But I hadn't explored the extent, manner and contribution of his work in engineering.

Now this Renaissance man would be inspiring in any event, but additionally so in this context, for he used sketches to investigate, to find out, to create visual "demonstrations" that teach.

He puzzled things out, to astonishing effect, foreshadowing Copernicus by 40 years when he declared "IL SOLE NO SI MUOVE" ("The sun does not move.") ), adding "The earth is not in the center of the circle of the sun, nor in the center of the universe." Further, 200 years before Newton, he wrote "Every weight tends to fall towards the center by the shortest possible way." He was prescient in other fields too. 400 years before Darwin, he placed man in the same broad category as apes. [Gimpao Ni Ei Suuh]
Knowing How to See

Indeed, he developed and practiced a rather modern form of cognition—studying what was known from masters current and past (his library was extensive), but extending that study by "knowing how to see" (saper vedere) with scientific inquiry augmented and abetted by technology—that of pen/pencil.

He and other engineers of the day would study each other's works.
Copying (a good thing)

And they copied designs from each other's notebooks (Engineering and the Mind's Eye).
Inventions — By Thinking!

Among the inventions credited with passing into general practical use are:

- the strut bridge,
- the automated bobbin winder,
- the rolling mill,
- the machine for testing the tensile strength of wire and
- the lens-grinding machine.

Leonardo's notebooks capture and extend understanding of phenomenon of nature and machine, and include numerous inventions, some built in his time, others later -- such as the lens grinding machine pictured.

Sketching to understand, to invent -- and convince!

These inventions, we might note, were constructed as sketch-prototypes so compellingly drawn as to both persuade feasibility in many cases, and to inform construction. (The Archimedes steam cannon is a fun story).

Some of his designs were used in his day, while others were ahead of their time.
Sketching

• To observe (more attentively)
• To study, think, reason, to puzzle things out
• To record
  • to think longer, harder
  • to show, to teach
• To invent, to combine, to make (new) connections
• To test ideas
  • thought experiments
• To persuade

Leonardo’s notebooks stand testimony to sketching as a means to

• observe more closely
• study, think, reason
• record, not just to persist, but to extend the range of cognition -- one’s own, and also to communicate with and teach others
• invent, by making novel connections
• test ideas on paper
• persuade

Several of Agassiz's students tell similar stories, one recounting that he'd picked up a pencil to draw the fish, to sketch details and Agassiz had remarked "That is right, a pencil is one of the best eyes." But those seeming endless days of looking -- seriously observing, studying, patiently noticing, formulating conjectures, reaching for understanding, was a lesson students pointed back to as career-shaping. As they came to see more, they'd get more from Agassiz, but the training was in learning to see. To see, to understand the fish. Its structure, its symmetries and patterns, the relation of structure to function.
Centuries before, Leonardo da Vinci had studied human anatomy with the same emphasis on “knowing how to see” -- especially with the aid of a pencil or pen.

This is from the Encyclopedia Britannica, which I reproduce here [but didn’t include in the talk] because the language is telling:

"Leonardo combined anatomical with physiological research. From observing the static structure of the body, Leonardo proceeded to study the role of individual parts of the body in mechanical activity. This led him finally to the study of the internal organs; among them he probed most deeply into the brain, heart, and lungs as the "motors" of the senses and of life."

And from Martin Clayton’s book on Da Vinci’s anatomy work:

"First of all, as a sculptor, engineer, architect, he had an intuitive understanding of form — when he dissected a body, he could understand in a very fluid way how the different parts of the body fit together, worked together. And then, having made that understanding, as a supreme draftsman, he was able to record his observations and discoveries in drawings of such lucidity, he’s able to get across the form, the structure to the viewer in a way which had never been done before and, in many cases, has never been surpassed since."

The point that I want to draw out here, is that of sketching not only to see structure, but the relation of structure to function. The emphasis on understanding mechanisms by considering which parts work in concert to achieve some function or capability, and how they do so.

Seeing to understand, to ask new questions. Looking for what’s typical or expected and what’s distinct and surprising; noticing contradictions that unseat our assumptions, so we see from a new perspective.
Seeing inside

Leonardo’s studies of the brain were on corpses. Today we can study the brain structure and function while the person is alive.
Visualization in Medicine Today

Brain Scans: peer inside your head – while you’re alive and thinking!

[This fMRI scan shows the areas of the brain active when you tell a lie.]

Visualization to understand, to diagnose, to...

We can use software to see ourselves thinking about software!

Peering inside the head, while the person is thinking! Gaining a better understanding of what parts of the brain are active, when we are doing tasks of different kinds.
We use software to see into

We're learning more, through visualization, of such deep and dynamic structures as the brain's pathways.

We can see structures in new ways, through behaviors they enable.
Advantage Seeing

Now. No talk on visual design would be complete without the obligatory reference to the amount of the brain devoted to visual processing.

It’s not my field and I don’t know what is considered the definitive reference, but one source mentioned 30 percent and another 50 percent of the cortex is devoted to processing visual information -- the discrepancies being due, as I understand it, to visual processing being used in conjunction with other systems, such as the motor system, so it is hard to separate out.

At any rate, we are geared to take in a lot of the information we do, through our visual system. Advantage seeing over hearing, touch, smell, taste. Though smell plays a strong enough role to be the referent metaphor when code goes bad, and we seek to identify and remove code smells.
Of course, before we concede victory to those whoop-whooping at the advantage this confers on visual design expression, we might note that code is visual.
Code is Design; We Design in Code

In an essay titled What is Software Design (published in the C++ Journal in 1992), Jack Reeves argues that source code is software design.

I agree. Indeed, I’d argue code is the dominant medium of expression of software designs.

We design in the medium of code (including tests) -- that is, we reason about, we shape, we express how we are creating system capabilities, assigning system responsibilities and responding to forces, in the medium of code.

Code isn’t just a language for computers, but for design thinking and expression.
We Design in Code

Let’s think about that, for a moment. A few years ago, this data point struck me - the six million lines of code for an airplane, is equivalent to a 3-story high pile of books!

“Typical GM car in 2010: 100M. Yes, really. 100” — Dan Dvorak, JPL *

* http://www.slideshare.net/NASAPMC/dvorakdan
That's a lot of truth!

Sam Guckenheimer said "The code is the truth" and Grady Booch added "but not the whole truth."

Three stories, and it's not even the whole truth?!
We Design ...in Code?!

That's a lot of truth! It is hard to wrap our arms, or heads, around quite so much truth.

Dana Bredemeyer puts it differently: "Code is fact. Truth is emergent." From interactions within the code, and of the software with its context of operation and use. That's even more to wrap our heads around!

Individually, there's a lot to try to comprehend and shape, to intellectually gain traction on, the design -- so we get more what we want. But software systems are complex, and more than one mind can manage, so we're not just communicating the design to the compiler, but with others on the team, now and in the future.
What’s Missing?

Okay, so I was being a little dramatic there. The software for an airplane isn’t one humongous monolith. Various systems contribute to the aforementioned volume of code. No-one has to understand all that truth, in its entirety. But that helps make my point. There are dimensions of design where code is not the medium that best supports reasoning and expression.

The code doesn’t generally express, in any ready way, a "big picture" view of the system as a whole, and we have to build up a mental model of the elements and relationships. Zooming out further, the code doesn’t express the system in the context of other systems (the use context, and the deployment context).

From the standpoint of human understanding, code is lossy. We have to mentally simulate behavior, conceptualizing interactions. Further, the following is generally missing from the source (though, especially at narrow/local scope, may be partially inferred from tests and assertions or comments):

- assumptions (where? under what conditions? ...)
- rationale (why? connecting the dots to intended outcomes/value and challenges)
- theory of operation (how? explaining how key mechanisms are intended to deliver outcomes with an emphasis on qualities or cross-cutting concerns which by nature are non-local and emergent, identifying challenges faced and how they’re addressed)
- alternatives considered, but ruled out (what we’re not doing and why not)

Recall: Our software systems are so large, we can never see them in their entirety. Our mental models are central. But idiosyncratic and fallible.
Jack Reeves’ point that source code is software design was, alas, taken to mean “Jack says forget design and just start coding.” So much so, that Jack had to write a follow-up article and later letter to the editor, explaining his position and remaking the case that “When I am coding, I am designing.” Writing code (and tests, and doing debugging) is not just implicitly doing design, it is explicitly doing design. It is a medium for expressing and externalizing our design thinking (making things more the way we want them to be), and interacting with it, probing and testing it, improving it. But it is not the only means of expressing our software design. Code is the expression we must end up with, so we should do all we can in code. Still, Jack reminds us of a point he made in his first article:

“Today, I would phrase it differently. I would say we need good architectures (top level design), good abstractions (class design), and good implementations (low level design).” – Jack Reeves

We Design at Different Scopes

Jack Reeves’ point that source code is software design was, alas, taken to mean “Jack says forget design and just start coding.” So much so, that Jack had to write a follow-up article and later letter to the editor, explaining his position and remaking the case that “When I am coding, I am designing.” Writing code (and tests, and doing debugging) is not just implicitly doing design, it is explicitly doing design. It is a medium for expressing and externalizing our design thinking (making things more the way we want them to be), and interacting with it, probing and testing it, improving it. But it is not the only means of expressing our software design. Code is the expression we must end up with, so we should do all we can in code. Still, Jack reminds us of a point he made in his first article:

“In software engineering, we desperately need good design at all levels. In particular, we need good top level design. The better the early design, the easier detailed design will be. Designers should use anything that helps. Structure charts, Booch diagrams, state tables, PDL, etc.—if it helps, then use it.”
Why We Model

- To abstract
- To reason
- To show
- To test
- To document
- To transform

To address wicked problems:
- To deal with complexity
  -- buffer overflow!
- To work together
  -- shared thought-space
- To communicate
  -- explain, defend, preserve

"Language shapes the way we think and determines what we can think about" — Benjamin Whorf

memory, so we can hold more information and we can see the relationships, the connections, and we can reason about causalities and consequences.

We draw informal sketches or more formal, precise models, to collaborate much more effectively with ourselves--with our moment ago, our yesterday, and our next month self. We draw to see, and record our thinking, and to expand and enhance our own thinking. To test and challenge and improve our designs.

And we use sketches and other models to collaborate with others. Now, and over time and distance. By collaborating on pictures, on design models, we create a shared thought-space. Now we have more minds actively engaged in coming up with alternatives, testing and challenging and improving the models--while they are just sketches, and the thought experiments and reasoned arguments are quick to play out and the biggest cost of change is the cost of letting go, the cost to egos.
Abstractions

“As programmers we deal with abstractions all the time and we have to invent them in order to solve our problems”
Michael Feathers

“Programming is a process of designing a DSL for your own application.” – Dave Thomas

Turtles ...all.the.way.down

Modeling abstracts away complexity inessential to the reasoning under the lens of the moment, enabling us to bring complex systems within the grasp of our bounded cognitive capacity. [You could apply caveats to my statement, but you’d be adding complexity.] As programmers, we’re using abstraction, and abstractions, rather fluidly. We build our systems out of richer and richer, more and more powerful, abstractions.

A children’s rhyme comes to mind:

Big fleas have little fleas,
Upon their backs to bite ’em,
And little fleas have lesser fleas,
and so, ad infinitum.

-- Augustus De Morgan.

But adapted to:

Big things are made of smaller things,
With interactions to unite ’em,
And smaller things have smaller things,
and so, ad infinitum.

[That’s mainly a play on Jonathan Swift, but I expect you’re hearing echoes of Herbert Simon’s “The Architecture of Complexity” too. And the Eames “Powers of Ten”.]
Crisp and Resilient Abstractions

You’re no doubt familiar with the well-trodden conference joke:

“There are two hard problems in computer science: caching, naming, and off-by-one errors.” -- source? (Tantek Çelik?)

Well, “places to put things” is the dual of the “naming things” bugbear in coding, no? What we are striving to do is to craft “crisp and resilient abstractions.”

Granularity is a judgment call, and cohesion of responsibility can be a bit of a slippery notion. Sure, we have heuristics, but assumptions are a kind of dependency, if you like, on some external condition. I stray. You see? Cohesion. Slippery slope.

A Conjecture and a Knock-Down Argument were taking a hike...

Conjecture: SRP applies at the level of abstraction of the abstraction

Knock-down argument: at larger grains, abstractions may play different roles (so have more than a single responsibility cluster)

Anyway... Crisp speaks to being non-leaky, non-tangly, having an internal cohesion or relatedness, with clear boundaries. While resilient speaks to adaptable under change without flaking out.
Crisp and Resilient Abstractions

You’re no doubt familiar with the well-trod conference joke:

“There are two hard problems in computer science: caching, naming, and off-by-one errors.” -- source? (Tantek Çelik?)

Well, “places to put things” is the dual of the "naming things" bugbear in coding, no? What we are striving to do is to craft those “crisp and resilient abstractions.”

Granularity is a judgment call, and cohesion of responsibility can be a bit of a slippery notion. Sure, we have heuristics, but assumptions are a kind of dependency, if you like, on some external condition. I stray. You see? Cohesion. Slippery slope.

A Conjecture and a Knock-Down Argument were taking a hike...

Conjecture: SRP applies at the level of abstraction of the abstraction

Knock-down argument: at larger grains, abstractions may play different roles (so have more than a single responsibility cluster)

Anyway... Crisp speaks to being non-leaky, non-tangly, having an internal cohesion or relatedness, with clear boundaries. While resilient speaks to adaptable under change without flaking out.
Complex systems can't be approached as merely “parts flying in formation, trying to be an airplane” (Wim Roelandts)
What’s next?

Visual Design

Sketching in Engineering: Leonardo da Vinci
Visualization in Other Fields
What’s Missing from Code
Why We Model
A Change in Perspective
Mechanisms
Wisdom and Fallibility
Extraordinary Moment Principle
More Part II: Visual Design

Coming soon

Thanks for your encouragement
– it makes a difference!
And then?

### Intention and Reflection

<table>
<thead>
<tr>
<th>Intention: how do we envisage the system?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling approaches</td>
</tr>
<tr>
<td>Intention, Code, Reflection: Round-trip?</td>
</tr>
<tr>
<td>Revisiting intention: design across boundaries</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reflection: seeing the system (as built)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software visualization landscape</td>
</tr>
<tr>
<td>What’s it good for?</td>
</tr>
<tr>
<td>Closing thoughts</td>
</tr>
</tbody>
</table>
Thanks

Thanks to Dana Bredemeyer for endless generosity in sharing ideas and insight, as well as encouragement and support.

Thanks also to Duarte for the SlideDocs idea, and the template that I started with. See http://www.duarte.com/slidedocs/

About me:

Ruth Malan
Bredemeyer Consulting
Web: bredemeyer.com

Twitter: @RuthMalan
Web: ruthmalan.com
Journal: ruthmalan.com/Journal.htm