Why Don’t Developers Draw Diagrams?
Grady Booch*
IBM Fellow
gbooch@us.ibm.com

Why Don’t Developers Draw Diagrams?

* With inspiration from Ruth Malan @ http://www.ruthmalan.com/
Developers (for the most part) don’t draw diagrams because they (the diagrams, that is) (rarely) offer any fundamental value that advances their (the developers, that is) essential work
What Is A System?

- A system is a construct or collection of different elements that together produce results not obtainable by the elements alone. The elements, or parts, can include people, hardware, software, facilities, policies, and documents; that is, all things required to produce systems-level results. The results include system level qualities, properties, characteristics, functions, behavior and performance. The value added by the system as a whole, beyond that contributed independently by the parts, is primarily created by the relationship among the parts; that is, how they are interconnected.
Narrowing Our Scope

- I largely care about the creation, development, deployment, evolution, operation, and support of software-intensive systems.

- A software-intensive system
  - Is about software and hardware and the squishy bits (aka wetware aka humans).
  - Is also about a system of systems.

- As for the artifacts of a software-intensive system
  - The most important artifact is the raw, running naked code that runs on hardware and interacts with humans the real world.
  - All other artifacts are secondary, but nonetheless they are still critical, for they help the enterprise deliver the right system at the right time to the right stakeholders with the right balance of cost and value.
What role do/should diagrams have in creating, developing, deploying, evolving, operating, and supporting, software-intensive systems?
Modeling The Cosmos

- We have the facilities of all of our senses – seeing, hearing, touching, tasting, smelling – whereby we can visualize the world and interact with the things in it.
- There is a large class of problems that we increasingly face for which our basic senses fail us:
  - How do we visualize the social connections among a large group of people?
  - How do we picture the way a virus – either a physical one or one that is manifest only in ones and zeros – works?
  - How do we reason about the millions of parts that constitute a large software-intensive system?
- How does one visualize ultra-complex structures that have no physical manifestation?
A History of Diagrams

- Maps
- Geometric diagrams
- Cartesian diagrams
- Diagrams for physics and chemistry
- Topological diagrams
- Tree diagrams
- Venn diagrams
- ...

The Visualization Zoo

- Time series
  - Index charts
  - Stacked graphs
  - Small multiples
  - Horizon graphs

- Statistical distributions
  - Stem and leaf plots
  - Q-Q plots
  - Scatter plot matrix
  - Parallel coordinates

- Maps
  - Flow maps
  - Chloropleth maps
  - Graduated symbols maps
  - Cartograms

- Hierarchies
  - Node-link diagrams
  - Adjacency diagrams
  - Enclosure diagrams

- Network
  - Force-directed layouts
  - Arc diagrams
  - Matrix views

Graphic from Ruth Malan
Another Visualization Zoo

# A Periodic Table of Visualization Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Visualization</td>
<td>Visual representations of quantitative data in schematic form (either with or without axes)</td>
</tr>
<tr>
<td>Strategy Visualization</td>
<td>The systematic use of communication through visual representations in the analysis, development, formulation, communication, and implementation of strategies in organizations</td>
</tr>
<tr>
<td>Information Visualization</td>
<td>This use of innovative visual representations of data to simplify complex information. It is a method to make data more accessible and understandable.</td>
</tr>
<tr>
<td>Metaphor Visualization</td>
<td>Visual Metaphors assign information graphically to organize and structure information. They also convey an insight about the represented information through the key characteristics of the metaphor that is employed.</td>
</tr>
<tr>
<td>Concept Visualization</td>
<td>Methods to elaborate (visualize) qualitative concepts, ideas, plans, and analyses</td>
</tr>
<tr>
<td>Compound Visualization</td>
<td>The complementary use of different graphic representation formats in one single scheme or frame</td>
</tr>
</tbody>
</table>

Note: Depending on your location and connection speed, it can take some time to load a pop-up picture.

© Ralph Langner & Martin J. Epppler, www.visual-literacy.org

http://www.visual-literacy.org/periodic_table/periodic_table.html
We’ve (largely) mastered the technique
But there’s been an emphasis on easy eye candy
And so we need to return to the basics of why we model and seek visualizations that are actionable.
What Pain Do You Feel?

- How do we attend to new requirements without being saddled by our legacy (but at the same time not compromising that legacy)?
- How do we integrate new technology into our existing code base?
- How do we integrate our existing systems to extract greater value from the whole?
- How do we increase our agility in response to the market while simultaneously improving efficiency and quality yet also reducing costs?
- How do we attend to assets introduced through acquisition?
- How do we use software to improve market efficiency through the creation of dominant product lines?
- How do we attend to a continuously refreshed stakeholder community, a globally and temporally distributed development team, and inevitable leakage/loss of institutional memory?
- While doing all this, how do we continue to innovate?
Points Of Friction

- Cost of start up and on-going working space organization
- Inefficient work product collaboration
- Maintaining effective group communication, including knowledge and experience, project status, and project memory
- Time starvation across multiple tasks
- Stakeholder negotiation
- Stuff that doesn’t work
- High semantic density artifacts
The Limits Of Computing

- The laws of physics
- The laws of software
- The challenge of algorithms
- The difficulty of distribution
- The problems of design
- The importance of organization
- The impact of economics
- The influence of politics
- The limits of human imagination
Forces
An Observation

- While these points of pain are legion, a common thread that weaves through them is that of architecture
  - Every software-intensive system has one
  - Most are accidental, a few are intentional
  - A considerable amount of architectural knowledge lies in tribal memory
- The presence of a reasonably well understood, syndicated, and governed architecture has a positive impact upon each of these points of pain
There Are Some Known Knowns

- All architecture is design; not all design is architecture. A system’s architecture is defined by its significant design decisions (where “significant” is measured by the cost of change).
- Most architectures are accidental; some are intentional.
- Every software-intensive system has an architecture, forged from the hundreds of thousands of small decisions made every day.
- The code is the truth, but not the whole truth: most architectural information is preserved in tribal memory.
Architecture Metamodel
Architecture Metamodel
Architecture Metamodel
Representing Software Architecture

Logical View
- End-user Functionality

Process View
- System integrators
  - Performance
  - Scalability
  - Throughput

Use Case View

Deployment View
- System engineering
  - System topology
  - Communication
  - Provisioning

Implementation View
- Programmers
  - Configuration management

Conceptual

Physical
Games
Pathfinder
Adobe Photoshop
The Well-Structured Architecture

- All well-structured systems are full of patterns
- All well-structured systems embody
  - Crisp abstractions
  - A clear separation of concerns
  - A balanced distribution of responsibilities
Fundamentals

- Development takes place at two levels: architecture and implementation.
  - Both are ongoing, and they interact with each other strongly. New implementations suggest architectural changes. Architectural changes usually require radical changes to the implementation.
The Organizational Architecture
Focus Over Time

Discovery | Invention | Implementation

Focus
The Software-intensive System Lifecycle

- **Inception**
  - Understand what to build
- **Elaboration**
  - Understand how to build it
- **Construction**
  - Build the product
- **Transition**
  - Deliver and adapt the solution
Software Archeology

- The recovery of essential details about an existing system sufficient to reason about, fix, adapt, modify, harvest, and use that system itself or its parts.
Nine Things To Do With Old Software

- Give it away
- Ignore it
- Put it on life support
- Rewrite it
- Harvest from it
- Wrap it up
- Transform it
- Preserve it
Complexity

- Classical
  - Computational complexity
  - Kolmogorov complexity
- From my experience
  - Software mass
  - The enumeration of things per view
  - The enumeration of connections per view
  - The presence of patterns per view
  - The number of possible states
Triggers Of Complexity

- Significant interactions
- High number of parts and degrees of freedom
- Nonlinearity
- Broken symmetry
- Nonholonomic constraints
  - Localized transient anarchy
  - Time-triggered vs state machines
Fundamental Challenges Of Discrete Systems

- Presence of essential complexity [Brooks]
- Non-continuous behavior of discrete systems
- Combinatorial explosion of states
- Corruption from unexpected external events
- Lack of mathematical tools and intellectual capacity to model the behavior of large discrete systems
The Texture Of Software-Intensive Systems

- Bits at the bottom
- Decomposition by
  - Verbs (algorithmic)
  - Nouns (object-oriented)
- Most interesting software-intensive systems are manifest as fragments of algorithmic code swimming in a primordial soup of objects
- The code is the truth, but not the whole truth
  - Architecture as a collection of significant design decisions
  - Patterns as themes
  - Cross-cutting concerns as traces
  - Rationale as the backstory
  - Tribal memory as the human story
Observations

- Hierarchy is an illusion
- There is no top
- Characterization as an input-output mapping is naïve
- Multiple simultaneously interlocking views are necessary
From Complexity To Simplicity

- Simplicity can only be found by adding energy
  - That energy is best applied in a process of continuous architectural refactoring
  - The power of patterns
- Patterns help you manage software complexity [Buschmann]
- While we refactor code for many reasons, the following motivations are among the most common: make it easier to add new code; improve the design of existing code; gain a better understanding of code; make coding less annoying. [Kerievsky]
Why Don’t Developers Draw Diagrams?

- Our primary product is raw, running, naked code, not diagrams
- Diagrams and code have an uneasy relationship that quickly drifts into oblivion and usually ends in tears
Some Basic Principles

Edward Tuftee
- He abhors eye candy, and values meaning over presentation
- “The minimum we should hope for with any display technology is that it should do no harm”
- Any representation should not obscure, bias, or obfuscate reality
- He warned against gratuitous representations “without realizing the cost to the content and the audience in the process”
- Representations must simplify, not contribute to, complexity
- “The point is that analytical designs are not to be decided on their convenience to the user or necessarily their readability or what physiologists or decorators think about them; rather, design architectures should be decided on how the architecture assists analytical thinking about evidence.”

Scott McCloud
- Amplification through simplification

Carl Sagan
- “The brain has its own language for testing the structure and consistency of the world”
What Modeling Is

- Abstraction of reality
- Abstractions are not reality
What Modeling Should Be

- Abstraction with freedom but without ambiguity
- Abstraction with focus
- Artifacts at a moment in time
- Artifacts across time and space
- Artifacts for many stakeholders
- Artifacts made manifest
Why We Model

- To abstract
- To reason about
- To document
- To transform
To Abstract

- Amplification through simplification
  - Abstractions are not reality
  - Abstractions are intentionally incomplete
To Reason About

- Compare, synthesize, analyze, generate abstractions
  - The brain has its own language for testing the structure and consistency of the world.
- Different models attend to the needs of
  - Different stakeholders with
  - Different concerns and thus
  - Different viewpoints

http://www.youtube.com/watch?v=zSgiXGELjbc
To Document

- The code is the truth, but not the whole truth
  - Architecture as a collection of significant design decisions
  - Patterns as the themes
  - Cross-cutting concerns as the traces
  - Rationale as the back story
  - Tribal memory as the human story
To Transform

- Abstractions made manifest as the executable system itself
  - Some transformations are tedious
  - Some transformations are noisy
  - Some transformations are lossy
  - Some transformations are somewhat reversible
Yet Another Visualization Zoo

- Static vs dynamic
- Generative vs reflective
- Dashboard vs predictive
A Visualization Grand Challenge

- How can I visualize any of the points of friction?
- How can I model any one of the triggers of complexity?
- How can I visualize multiple interlocking architectural views?
- How can I highlight/discover the patterns in a system?
- What can I learn from visualizing the views of a system’s architecture overlaid with the architecture of the organization?
- How might I use an iPad as an architect’s workbench?
Give me a white board and a place to stand and I can model the world