No Silver Bullet

- *Essence and Accidents of SE*

CS580 Advanced Software Engineering
http://cs580.yusun.io
October 1, 2014

Yu Sun, Ph.D.
http://yusun.io
yusun@csupomona.edu

CAL POLY POMONA

No Silver Bullets

- Essence and Accident in Software Engineering
  - IEEE Computer, April 1987

- Frederick Brooks
  - Project manager of OS/360
  - Mythical Man Month
  - Brooks' Law
    - "Adding manpower to a late software project makes it later"
  - UNC (graphics researcher)
  - 2000 Turing Award winner

Paper Title

- Software is like a werewolf – it looks normal until the moon comes out and it turns into a monster
  - Missed deadlines
  - Blown budgets
  - Flawed products

- We want the silver bullet to kill the monster
  - "To make software costs drop as rapidly as computer hardware costs"

Silver Bullets

- "But, as we look to the horizon of a decade hence, we see no silver bullet. There is no single development, either in technology or management technique, which by itself promises even one order of magnitude improvement in productivity, in reliability, in simplicity."

- "Not only are there no silver bullets in view, the very nature of software makes it unlikely there will be any."

- No magical cure for software crisis…

In a Nutshell

- "I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation."

- "This essence is abstract in that the conceptual construct is the same under many different representations."

Essence and Accident

- All software construction involves
  - Essential tasks – the fashioning of the complex conceptual structures that compose the abstract software entity (i.e., the problem space)
  - Accidental tasks – the representation of the abstract entities in programming languages and the mapping of these onto machine languages within space and time constraints (i.e., the solution space)
- Distinction originally due to Aristotle

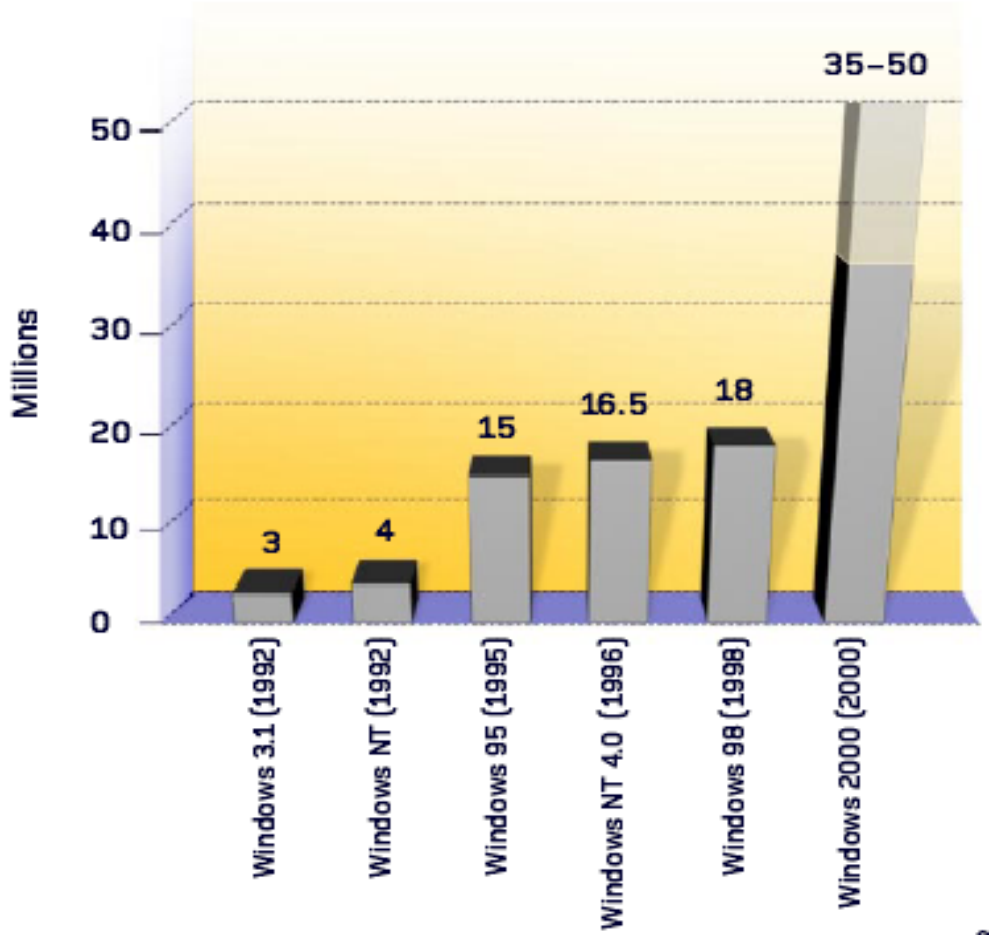
No Silver Bullet – Fred Brooks

- Like physical hardware limits, there are problems with software that won't be solved easily
- Inherent difficulties of software production
  - Complexity
  - Conformity
  - Changeability
  - Invisibility

Complexity

- Software is complex, even when done right
  - Many components with many kinds of interactions
  - Combinatorially many states
    - "Software systems have orders-of-magnitude more states than computers do"
- Mathematics and physical sciences
  - Complex phenomena simplified through models
  - Some complexities can be ignored
- In software, complexity is an essential property
  - Complexity cannot be abstracted

Explosion of Software Size

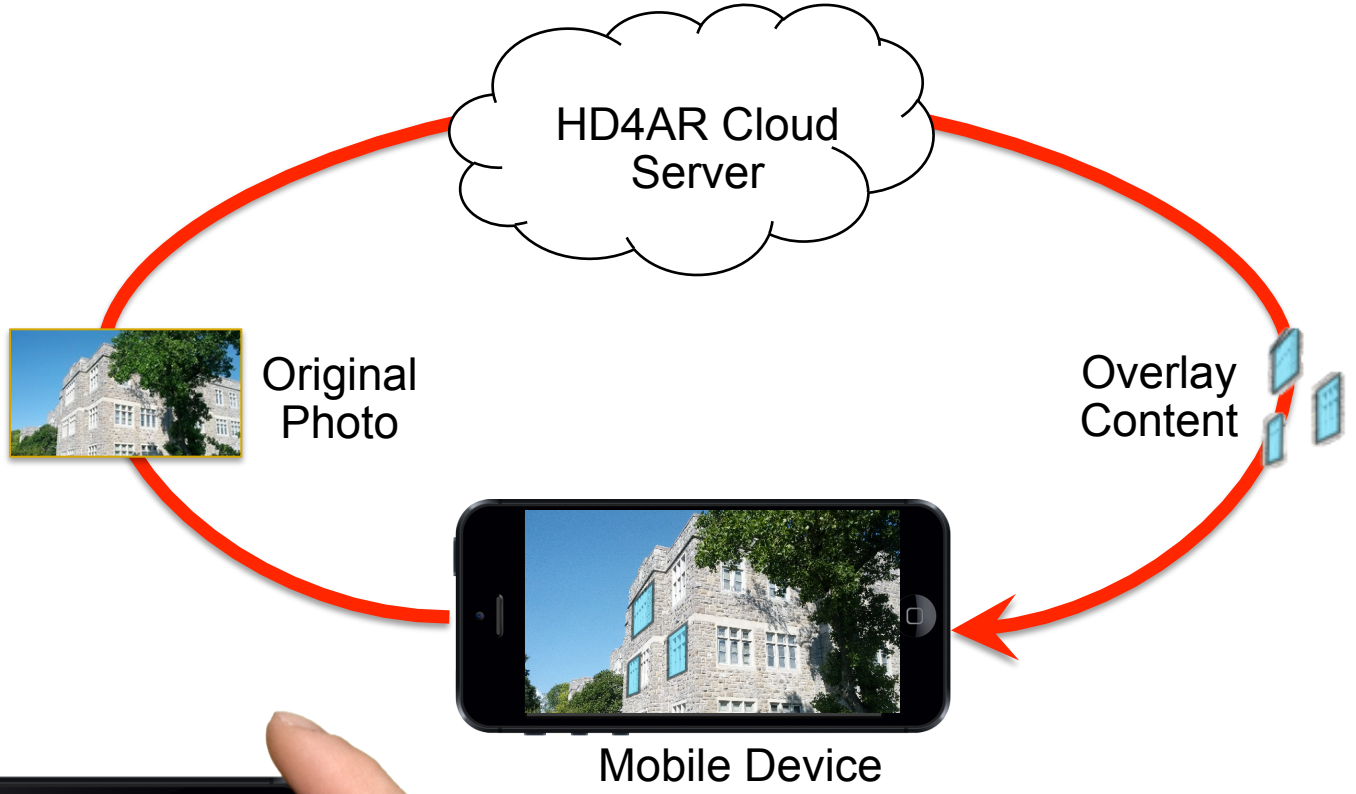


Source of image on right: Bruce Schneier
http://www.counterpane.com/presentation1.pdf

Complexity

- Complexity of software grows non-linearly in size
- Results
  - Difficulty of communication
  - Difficult to understand whole product (all possible states)
    - Unreliability
    - Turnover of personnel “a disaster”
  - Hard to manage
    - Difficult to extend new functions without creating side effects
    - Security risks due to “unvisualized states”

Cloud-based Computer Vision Architecture



Overview of the Algorithms

Photos Captured → Feature Extraction → Feature Matching → Track Creation → Structure from Motion

~15+ photos are taken of a physical object, such as a building, engine, etc.

Overview of the Algorithms

Photos Captured → **Feature Extraction** → Feature Matching → Track Creation → Structure from Motion

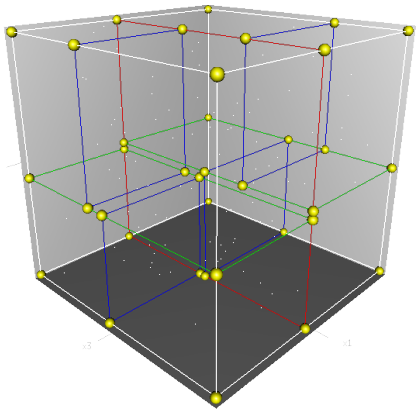
Image features (e.g. prominent points in the image) are extracted and represented using descriptors

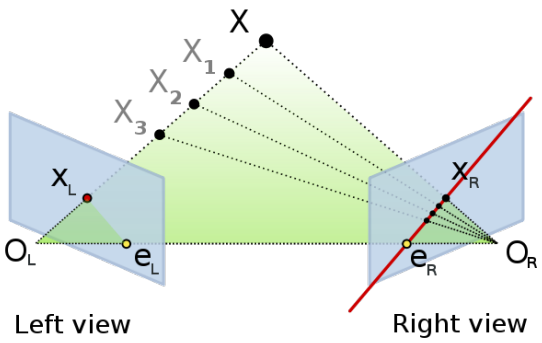
- Example Feature Detectors/Descriptors
  - SIFT
  - SURF
  - FREAK

Overview of the Algorithms

Photos Captured → Feature Extraction → **Feature Matching** → Track Creation → Structure from Motion

K-D Tree and Fast Approximate Nearest Neighbors (FANN) used to find initial feature correspondences

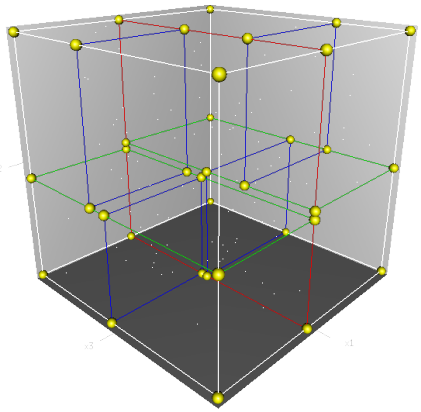


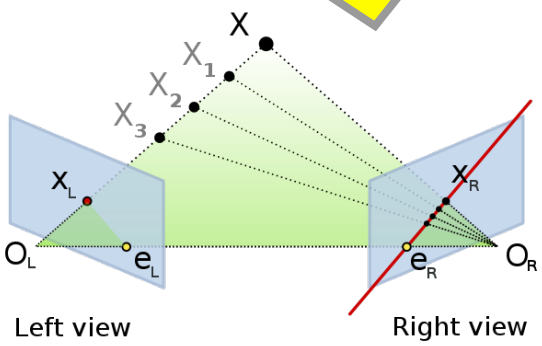


Overview of the Algorithms

Photos Captured → Feature Extraction → **Feature Matching** → Track Creation → Structure from Motion

RANSAC algorithm and 8-point method is used to estimate a fundamental matrix between image pairs and points more than σ pixels from an epipolar line are eliminated

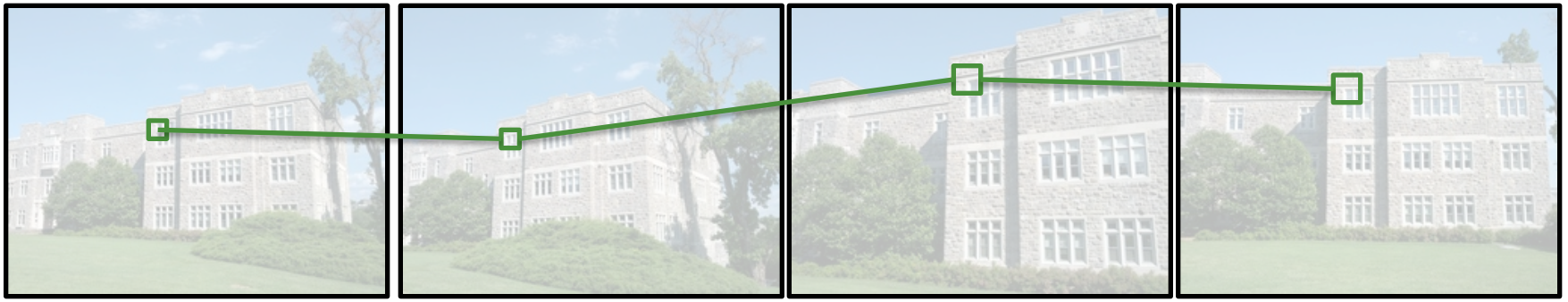




Overview of the Algorithms

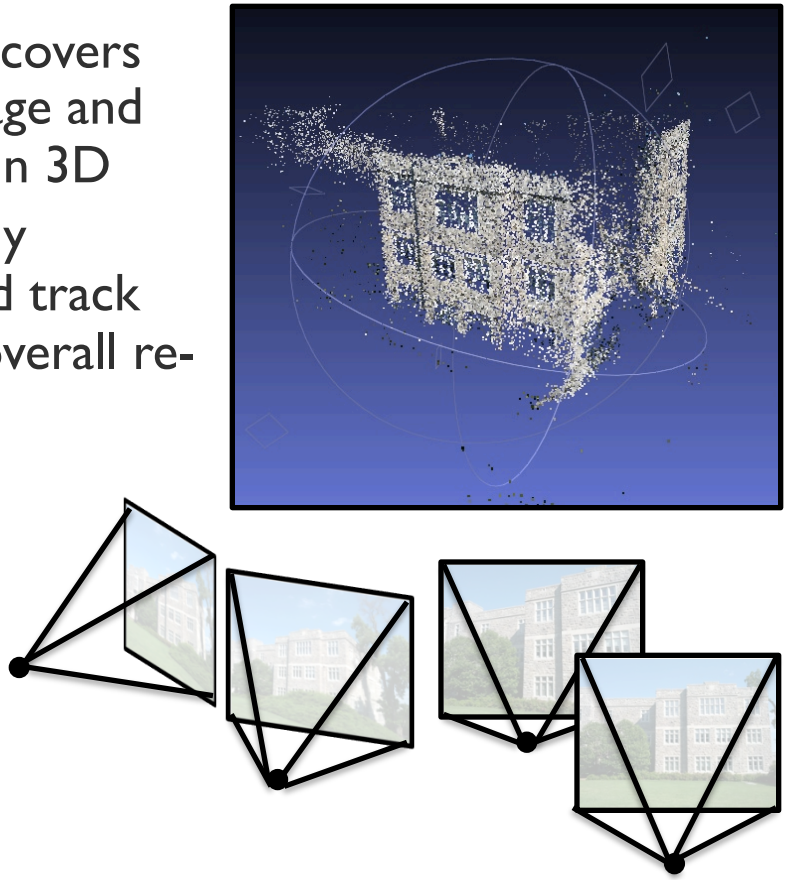
Feature tracks are created that track features across multiple images and a track cleaning operation is performed

- Track Cleaning: the Euclidean distance between matching points is compared to the minimum Euclidean distance between any pair in the track and outliers are removed

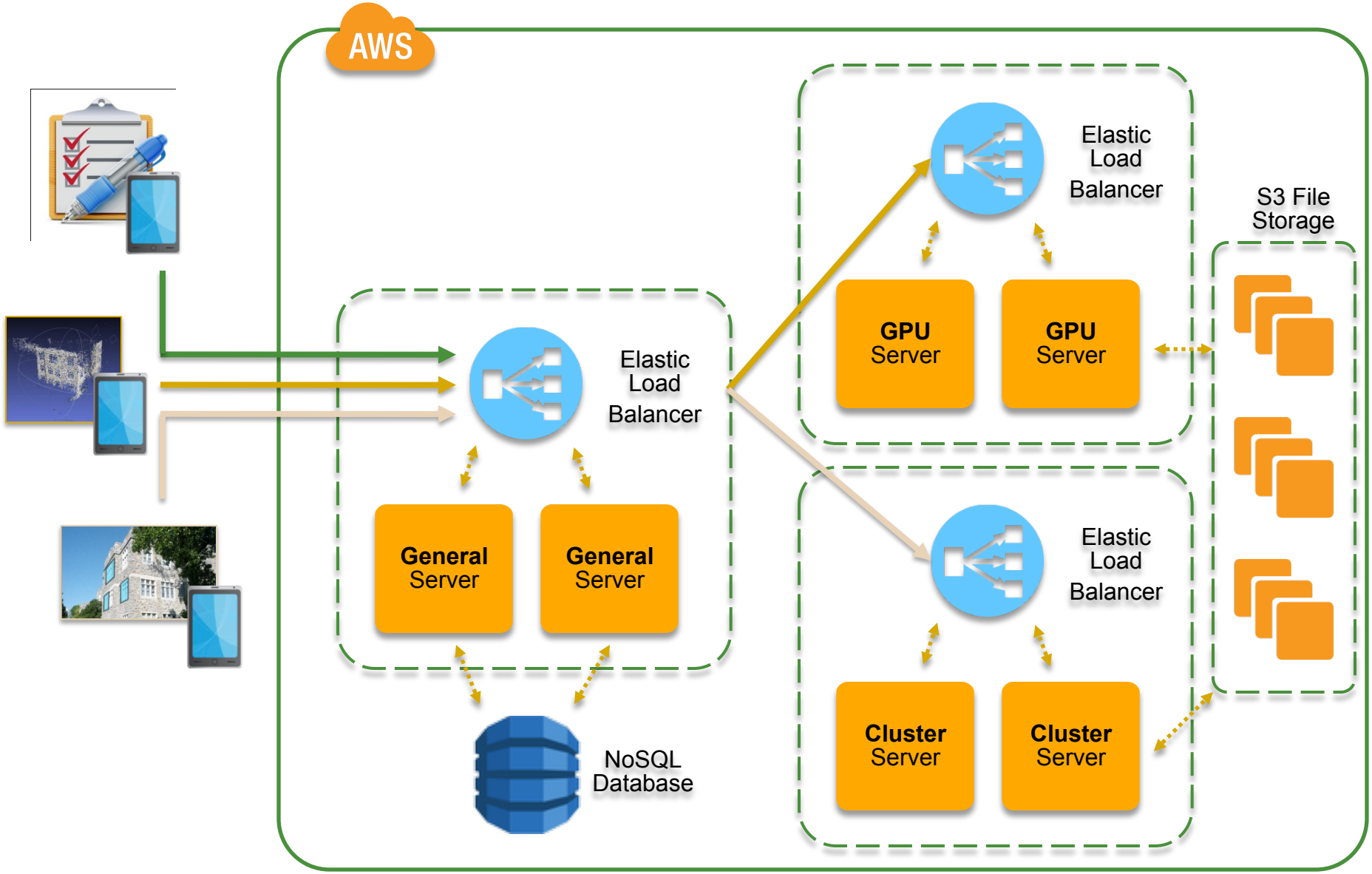


Overview of the Algorithms

- Structure from Motion iteratively recovers the camera parameters for each image and then triangulates the feature tracks in 3D
- Bundle Adjustment is used to globally optimize the camera parameters and track locations in order to minimize the overall re-projection error



Cloud-based Computer Vision Architecture



Conformity: Software is Everywhere

- 98% of all microprocessors control devices other than desktop computers
  - Automobiles, airplanes, televisions, copiers, razors…
- These devices also need software and often require strong technical skills to develop

15-20Kb

1-1.5Mb

>10Mb embedded software

Conformity

- Software must conform to fit someone else's prefab technology
  - New kid on the block!
    - Rules were already made
  - Perceived as more flexible than hardware or humans
  - Software has to be made to agree to common interfaces, protocols, and standards
    - Adds to complexity
  - Wrapping and unwrapping, data marshalling, etc.
- Result
  - Bewildering diversity of requirements

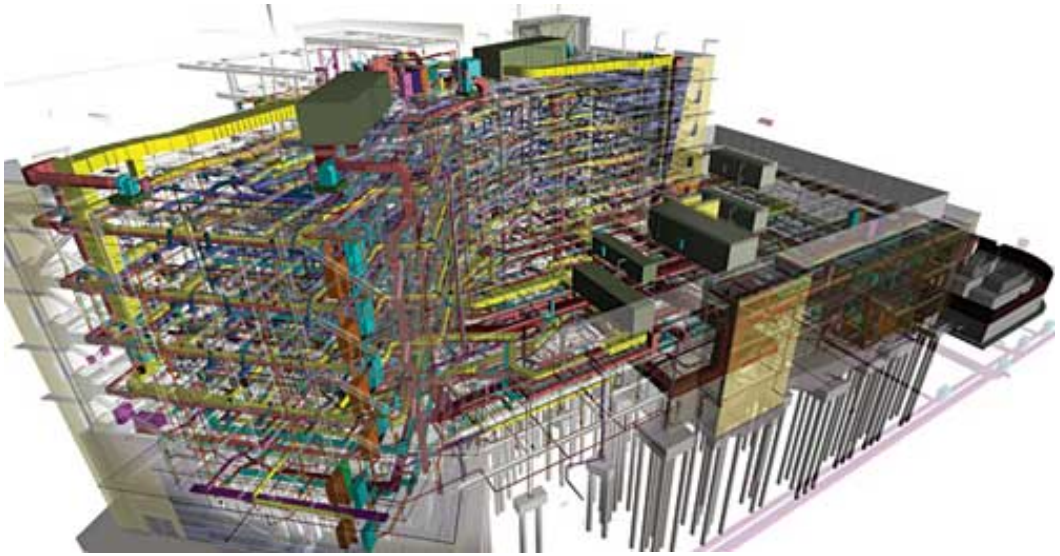
Changeability

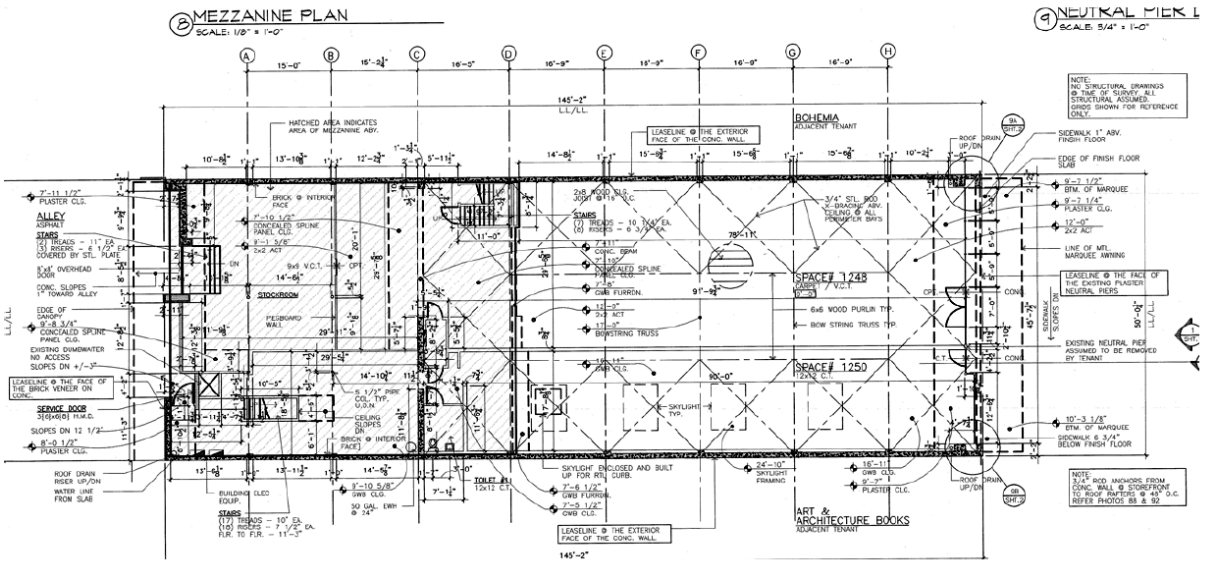
- "… the software product is embedded in a cultural matrix of applications, users, laws, and machine vehicles. These all change continually, and their changes inexorably force change upon the software product."
- We change software because we can!
  - "pure thought-stuff, infinitely malleable"
  - Easy to update existing systems in the field (in theory)
    - e.g., Windows update tool
- To be successful is to be changed!
  - Intuit and tax software
- Pressure to change
  - Useful software will encourage new requests
  - Long lifetime (~15 yrs) vs. hardware (~4 yrs)

Invisibility

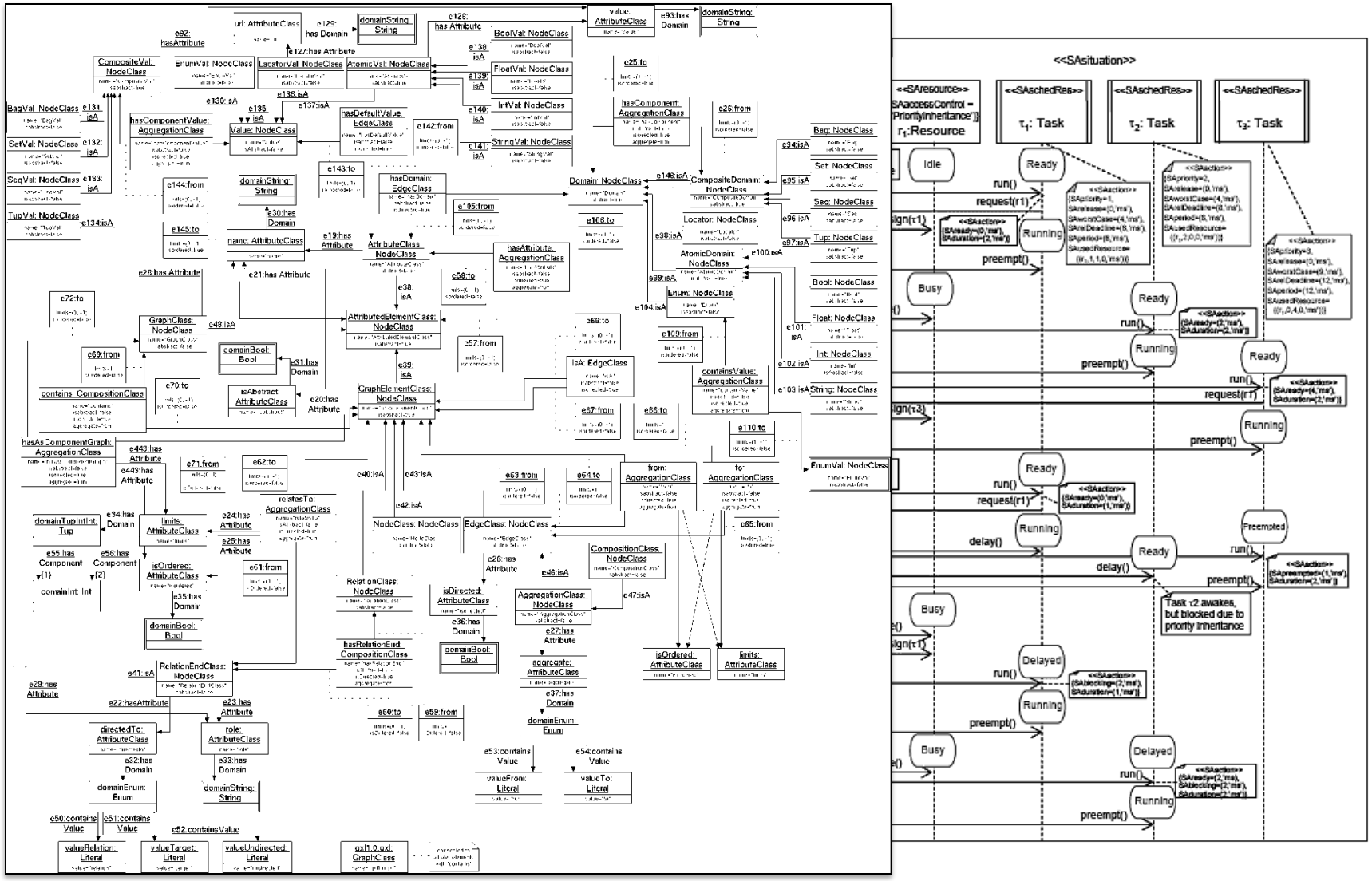
- "The reality of software is not inherently embedded in space."
- Can you touch a Java class?
- Software is invisible & unvisualizable
  - Different from physical and mechanical parts
  - No way to represent a complete product
- We can use various techniques to help us visualize aspects of software (control flow, data dependencies, or UML)
  - But that's NOT quite the same thing as, say, the blueprints of a building (no physical frame of reference in software)

Visibility - Floorplan





Invisibility - UML



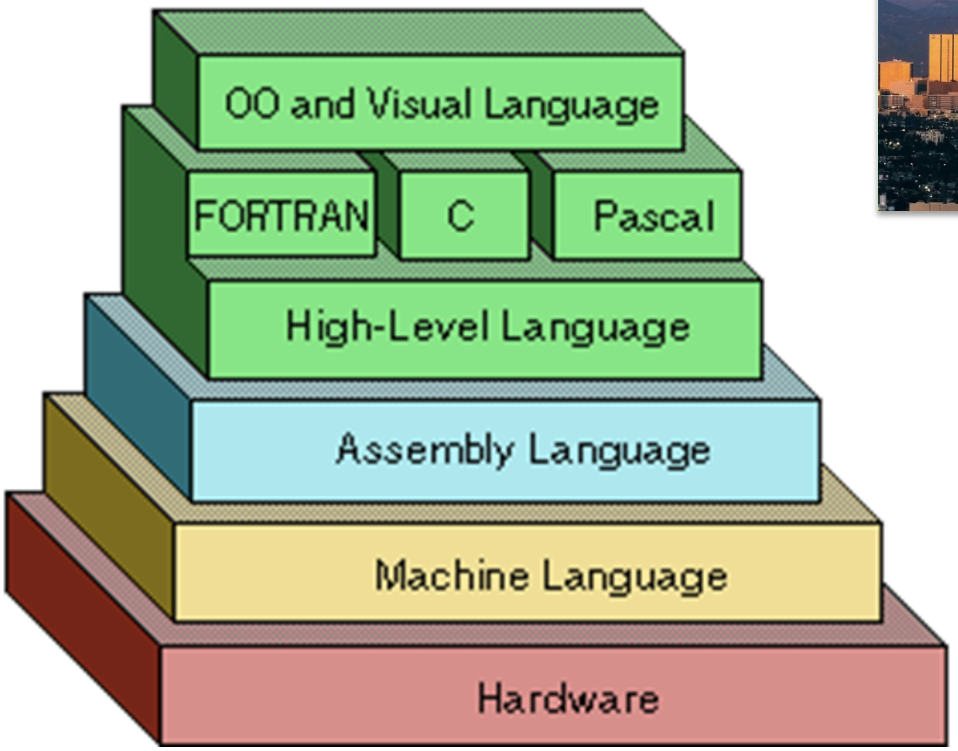
Date of Paper

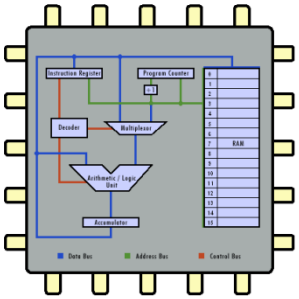
- Remember, this paper was written about 30 years ago…

Attacks on Accidental Problems

- HLLs
  - Frees us from byte-level thinking (accidental complexity)
  - Takes us up to "generic problem space"
    - e.g., C, sizeof(int), register allocation vs. pure objects and garbage collection
  - What we need beyond this is common abstractions in problem space …
    - i.e., domain modeling: telephony, avionics, flight reservation
  - … and in solution space, too
    - e.g., frameworks, libraries, components

Level of Abstraction





Attacks on Accidental Problems

- IDEs (SDEs)
  - This was novel then in 1987!
  - Even a smartly tweaked vim/emacs was a big step forward
  - In the old days, correct and reasonably efficient compiling was a notable feat
  - Current: Eclipse

What About These Silver Bullets?

- HLLs and OOP
  - Replace "Ada" by "Java"
    - "Philosophy of moduralization, of abstract data types, of hierarchical structuring"
  - Good training/use will make for better systems
    - Modern software-design techniques
  - Won't "solve" the SE problem, but the abstraction aspects of OO live in problem space, too
    - This has been a HUGE win in attacking essential complexity

What About These Silver Bullets?

- AI and expert systems
  - Mostly not much impact related to software development
  - Sometimes they help, e.g., test oracles
  - Certainly, AI techniques are useful in many application domains, but not as a silver bullet to solve the SE problem
    - WWW, faster processors, cheap memory have made some AI techniques quite useful, much more so now than then

What About These Silver Bullets?

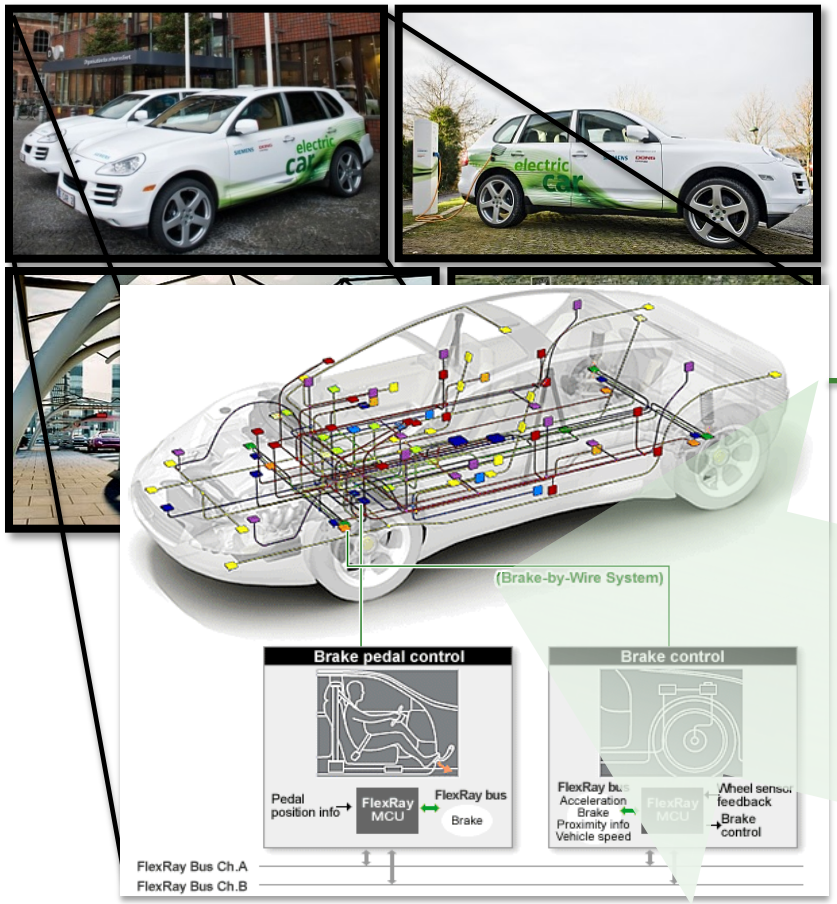
- "Automatic" programming
  - i.e., state the parameters, turn the crank, hey presto a software system!
  - We can do this now in some cases, works quite well
    - [It didn't work very well at the time of writing]
    - Some systems generate part of their source code
      - Generative programming; model-based approaches; software factories
  - Will never work completely in the general case, but is certainly useful when the domain of focus is narrow
    - Really, all this does is bump up the abstraction level by one
    - Great potential in domain-specific environments
      - Parsers: yacc, lex
      - Model-Driven Engineering

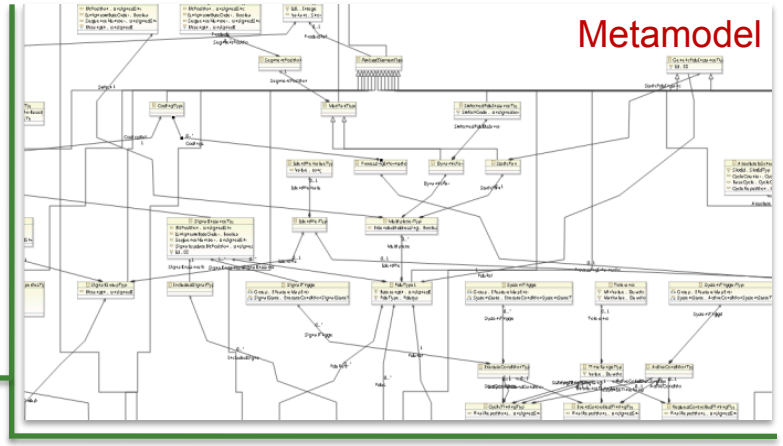
What About These Silver Bullets?

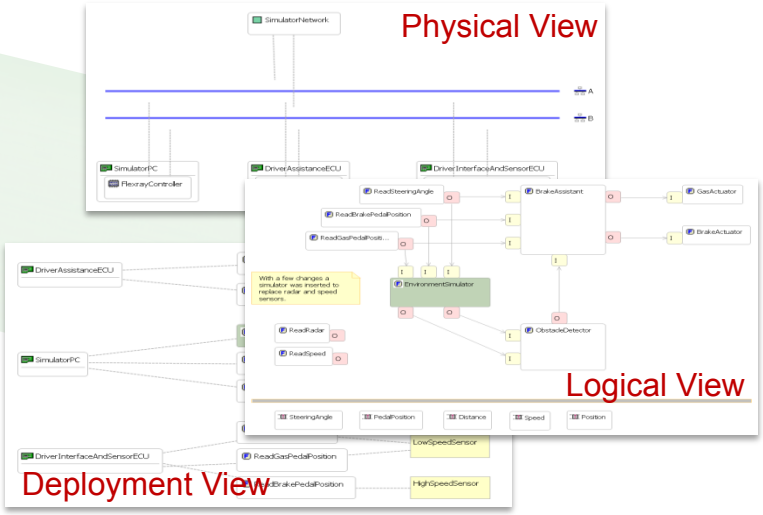
- Graphical programming
  - e.g., flow charts, and more recently Unified Modeling Languages (UML) and Specification and Description Language (SDL)
  - Always tempting to try, as certain aspects of software systems do seem inherently "topological"
    - In general, software is ethereal, unvisualizable (unlike hardware) with truly bizarre dependencies
  - Have you ever looked at a complicated SDL state-machine diagram? A class diagram with lots of complex interdependencies? A use-case scenario diagram with lots of variation?
    - Improvement: Domain-specific modeling languages

Model-Driven Engineering

SIEMENS





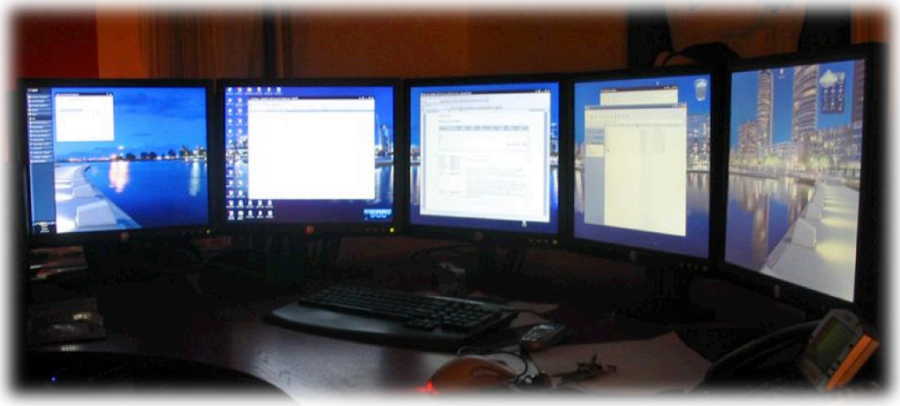


What About These Silver Bullets?

- Formal program verification
  - (Dijkstra: formal derivation)
  - Does program P implement specification S?
  - Undecidable in the general case, can be done by hand until scale overwhelms
  - In practice, has been done with some great successes, but only when investment seems to be worthwhile
    - Tremendously expensive; requires expert logicians!
  - The hard part, as Brooks rightly points out, is making sure you have the right specification S!
- Verification and Validation
  - Building the product right; building the right product

What About These Silver Bullets?

- Better tools and faster computers
  - Always good. Usually solution-space though
  - Technological Peter Principle:
    - “The selection of a candidate for a position is based on the candidate's performance in his or her current role rather than on abilities relevant to the intended role.”
    - In a hierarchically structured administration, people tend to be promoted up to their "level of incompetence"
    - Cheap disks, fast connections? iTunes and YouTube!



Promising Attacks on Conceptual Essence

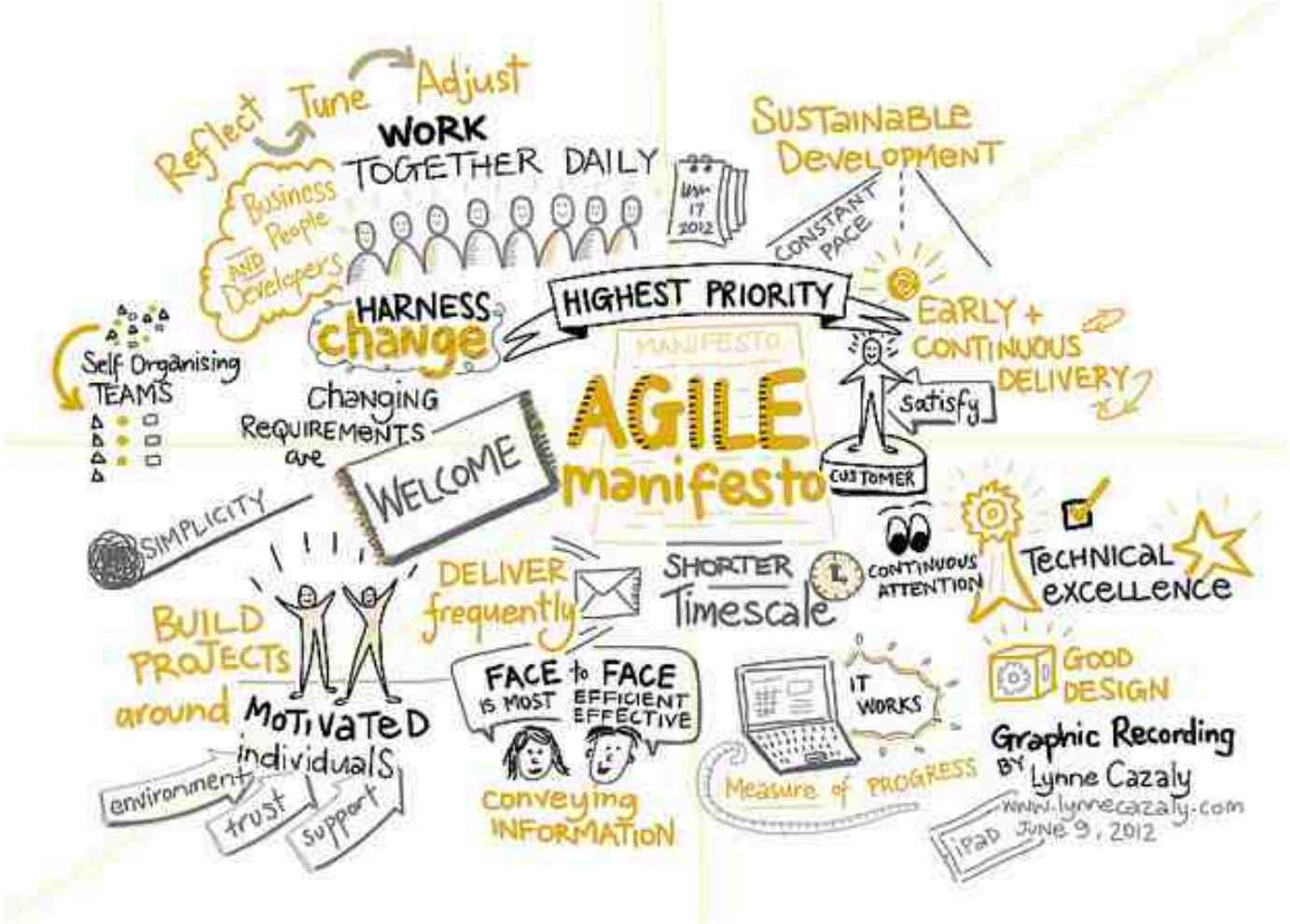
- Buy, don't build

Promising Attacks on Conceptual Essence

- Rapid prototyping and refinement
  - Take requirements, build mock up, show to user, analyze feedback, repeat
  - Early feedback means less chance for requirements errors (which are the most expensive), fast turnaround in problem space to narrow misunderstandings and educate customer
  - Requirements are the essence of a software system
    - Focusing on getting the requirements right is a direct attack on essential problems

Promising Attacks on Conceptual Essence

- Agile Development



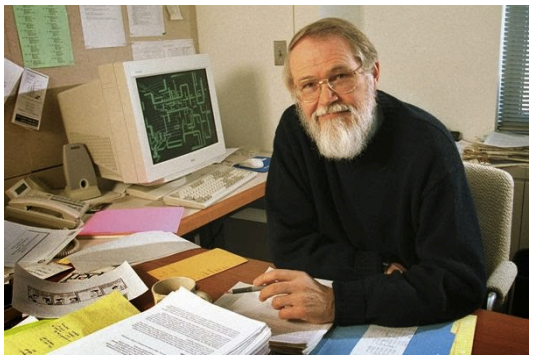
Promising Attacks on Conceptual Essence

- Virtuoso designers
  - Find good people and keep them









The top 10 greatest programmers in the world of all time
http://www.thecrazyprogrammer.com/2014/02/the-top-10-greatest-programmers-in-the-world-of-all-time.html

Obtaining the Increase

- Some people interpreted Brooks as saying that the essence could never be attacked
- That's not his point; he said that no single technique could produce an order of magnitude increase by itself
- He argued that several techniques in tandem could achieve that goal, but that requires industry-wide enforcement and discipline