The Big Idea

Software Architecture
Lecture 1

The Origins

- Software Engineers have always employed software architectures
  - Very often without realizing it!
- Address issues identified by researchers and practitioners
  - Essential software engineering difficulties
  - Unique characteristics of programming-in-the-large
  - Need for software reuse
- Many ideas originated in other (non-computing) domains

Software Engineering Difficulties

- Software engineers deal with unique set of problems
  - Young field with tremendous expectations
  - Building of vastly complex, but intangible systems
  - Software is not useful on its own e.g., unlike a car, thus
  - It must conform to changes in other engineering areas
- Some problems can be eliminated
  - These are Brooks' "accidental difficulties"
- Other problems can be lessened, but not eliminated
  - These are Brooks' "essential difficulties"

Essential Difficulties

- Only partial solutions exist for them, if any
- Cannot be abstracted away
  - Complexity
  - Conformity
  - Changeability
  - Intangibility

Complexity

- No two software parts are alike
  - If they are, they are abstracted away into one
- Complexity grows non-linearly with size
  - E.g., it is impossible to enumerate all states of program
  - Except perhaps "toy" programs

Conformity

- Software is required to conform to its
  - Operating environment
  - Hardware
- Often "last kid on block"
- Perceived as most conformable

Changeability

- Change originates with
  - New applications, users, machines, standards, laws
  - Hardware problems
- Software is viewed as infinitely malleable

Intangibility

- Software is not embedded in space
  - Often no constraining physical laws
- No obvious representation
  - E.g., familiar geometric shapes

Promising Attacks On Complexity (In 1987)

- Buy vs. Build
- Requirements refinement & rapid prototyping
  - Hardest part is deciding what to build (or buy?)
  - Must show product to customer to get complete spec.
  - Need for iterative feedback

Promising Attacks On Complexity (cont'd)

- Incremental/Evolutionary/Spiral Development
  - Grow systems, don't build them
  - Good for morale
  - Easy backtracking
  - Early prototypes
- Great designers
  - Good design can be taught; great design cannot
  - Nurture great designers

Analogy: Architecture of Buildings

- We all live in them
- (We think) We know how they are built
  - Requirements
  - Design (blueprints)
  - Construction
  - Use
- This is similar (though not identical) to how we build software

Some Obvious Parallels

- Satisfaction of customers' needs
- Specialization of labor
- Multiple perspectives of the final product
- Intermediate points where plans and progress are reviewed

Deeper Parallels

- Architecture is different from, but linked with the product/structure
- Properties of structures are induced by the design of the architecture
- The architect has a distinctive role and character

Deeper Parallels (cont'd)

- Process is not as important as architecture
  - Design and resulting qualities are at the forefront
  - Process is a means, not an end
- Architecture has matured over time into a discipline
  - Architectural styles as sets of constraints
  - Styles also as wide range of solutions, techniques and palettes of compatible materials, colors, and sizes

Limitations of the Analogy…

- We know a lot about buildings, much less about software
- The nature of software is different.
- Software is much more malleable than physical materials
- Software deployment has no counterpart in building architecture
- Software is a machine; a building is not

The Architect

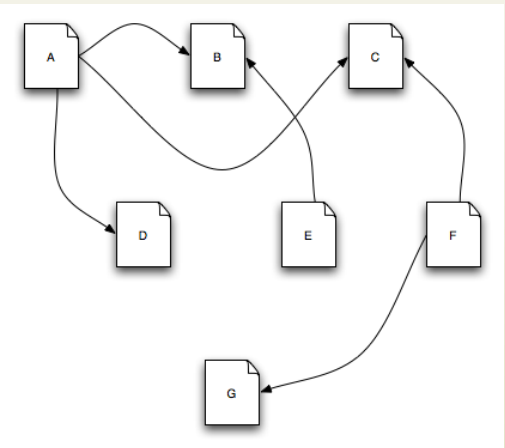
- A software architect makes high-level design choices.
  - This might include tools, software coding standards, or platforms to be used
- A distinctive role and character in a project
- Very broad training
- Amasses and leverages extensive experience
- A keen sense of aesthetics
- Deep understanding of the domain
  - Properties of structures, materials, and environments
  - Needs of customers

Power of Architecture

- Giving preeminence to architecture offers the potential for:
  - **Intellectual control**
    - The establishment and maintenance of documentation.
    - Program complexity could cause us to lose "intellectual control" due to the limited nature of our minds
  - **Conceptual integrity** (anywhere you look in your system, you can tell that the design is part of the same overall design)
  - **Effective basis for knowledge reuse**
  - **Realizing experience, designs, and code**
  - **Effective project communication**
  - **Management of a set of variant systems**
- Limited-term focus on architecture will not yield significant benefits!

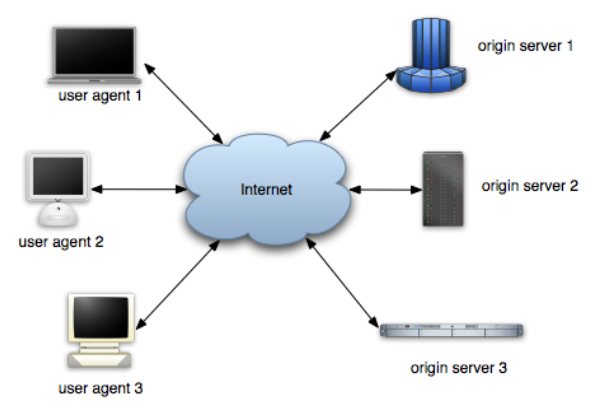
Architecture in Action: WWW

- This is the Web



Architecture in Action: WWW

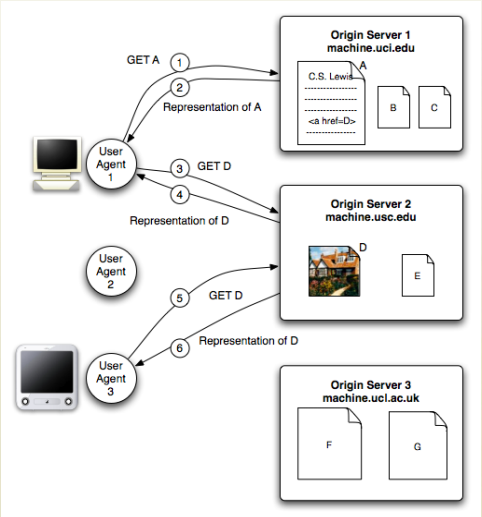
- So is this



Software Architecture: Foundations, Theory, and Practice

Architecture in Action: WWW

- And this



WWW's Architecture

- Architecture of the Web is wholly separate from the code
- There is no single piece of code that implements the architecture.
- There are multiple pieces of code that implement the various components of the architecture.
  - E.g., different Web browsers

WWW's Architecture (cont'd)

- Stylistic constraints of the Web's architectural style are not apparent in the code
- One of the world's most successful applications is only understood adequately from an architectural vantage point.

Structure Reuse

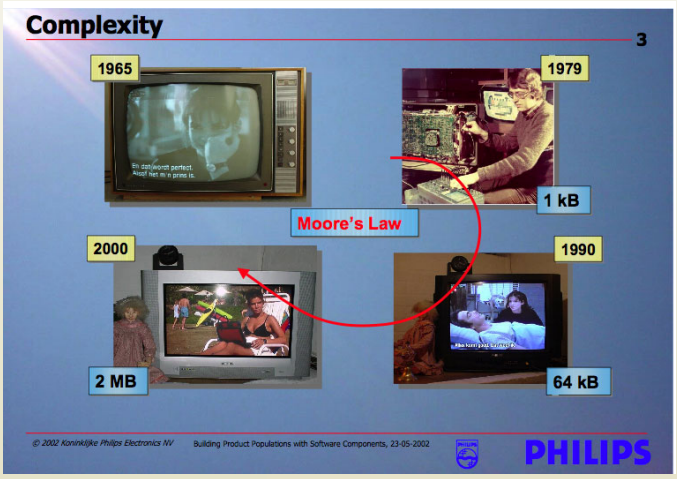
- Motivating example
  - A consumer is interested in a 35-inch HDTV with a built-in DVD player for the North American market.

    Such a device might contain upwards of a million lines of embedded software.

    This particular television/DVD player will be very similar to a 35-inch HDTV without the DVD player, and also to a 35-inch HDTV with a built-in DVD player for the European market, where the TV must be able to handle PAL or SECAM encoded broadcasts, rather than North America's NTSC format.

    These closely related televisions will similarly each have a million or more lines of code embedded within them.

Growing Sophistication of Consumer Devices



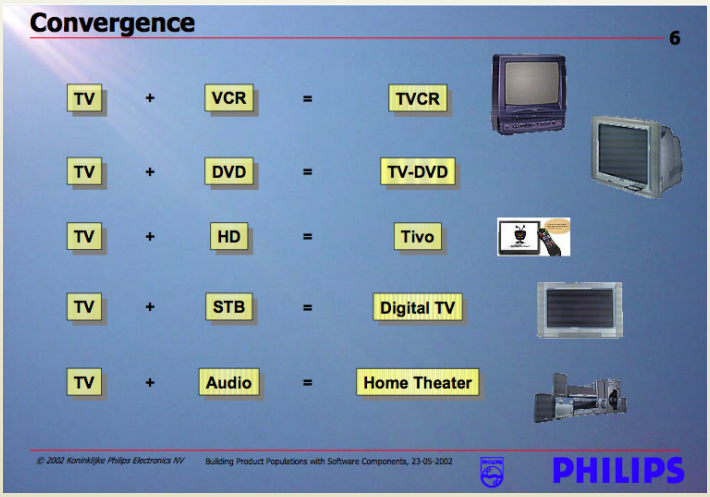
The Necessity and Benefit of Structure Reuse

- Building each of these TVs from scratch would likely put Philips out of business
- Reusing structure, behaviors, and component implementations is increasingly important to successful business practice
  - It simplifies the software development task
  - It reduces the development time and cost
  - it improves the overall system reliability
- Recognizing and exploiting commonality and variability across products

Reuse as the Big Win

- Architecture: reuse of
  - Ideas
  - Knowledge
  - Patterns
  - engineering guidance
  - Well-worn experience
- Product families: reuse of
  - Structure
  - Behaviors
  - Implementations
  - Test suites…

Added Benefit – Product Populations



Summary

- Software is complex
- So are buildings
  - And other engineering artifacts
  - Building architectures are an attractive source of analogy
- Software engineers can learn from other domains
- They also need to develop—and have developed—a rich body of their own architectural knowledge and experience