The role of software

- Our civilization runs on software
  - Innovation to capture new value
  - Improving productivity of resources deployed

- The privilege and responsibility of the software professional
How much software exists in the world?

- **SLOC is a measure of labor** *(not of value)*
  - Old code never dies *(you have to kill it)*
  - Some code is DOA

- **Some assumptions**
  - 1 SLOC = 1 semicolon
  - Number of software professionals worldwide
  - % of software professionals who cut code
  - SLOC/developer/year
  - $100US/SLOC
Number of software professional worldwide

Number of IT professionals worldwide

\[ y = -128.47x^3 + 12800x^2 - 59294x + 146623 \]
% of software professionals who cut code

\[ y = -0.0075x + 0.7575 \]
SLOC/developer/year

New or modified source lines of code per year per developer

\[ y = -0.0328x^3 + 4.8392x^2 - 67.596x + 1062.8 \]
New or modified SLOC/year and cumulative

New or modified source lines of code per year per developer & cumulative

- New or modified source lines of code per year
- Cumulative source lines of code
Dimensions of software complexity

Higher technical complexity
- Embedded, real-time, distributed, fault-tolerant
- Custom, unprecedented, architecture reengineering
- High performance

Lower technical complexity
- Mostly 4GL, or component-based
- Application reengineering
- Interactive performance

Higher management complexity
- Large scale
- Contractual
- Many stakeholders
- “Projects”

Lower management complexity
- Small scale
- Informal
- Single stakeholder
- “Products”

An average software project
- 5-10 people
- 3-9 month duration
- 3-5 external interfaces
- Some unknowns & risks

Sample projects:
- Embedded Automotive Software
- Commercial Compiler
- CASE Tool
- Defense Weapon System
- National Air Traffic Control System
- Telecom Switch
- Large-Scale Organization/Entity Simulation
- Enterprise IS (Family of IS Applications)
- Defense MIS System
- Small Scientific Simulation
- IS Application Distributed Objects (Order Entry)
- IS Application GUI/RDB (Order Entry)
- Business Spreadsheet
- Defense MIS System
- Telecommunication Switch
Creating the illusion of simplicity
The entire history of software engineering
Is one of rising levels of abstraction

Languages: Assembly -> Fortran/COBOL -> Simula -> C++ -> Java
Platforms: Naked HW -> BIOS -> OS -> Middleware -> Domain-specific
Processes: Waterfall -> Spiral -> Iterative -> Agile
Architecture: Procedural -> Object Oriented -> Service Oriented
Tools: Early tools -> CLE -> IDE -> XDE -> CDE
Enablement: Individual -> Workgroup -> Organization
Architecting a dog house

Can be built by one person
Requires
  Minimal modeling
  Simple process
  Simple tools
Architecting a house

Built most efficiently and timely by a team
Requires
  Modeling
  Well-defined process
  Power tools
Architecting a high rise
Early architecture

Progress
- Limited knowledge of theory
Contemporary architecture

**Progress**
- Advances in materials
- Advances in analysis

**Scale**
- 5 times the span of the Pantheon
- 3 times the height of Cheops
Modeling a house
Movements in civil architecture

- Bronze age/Egyptian (Imhotep)
- Grecian/Roman (Vitruvius)
- Byzantine/Romanesque
- Gothic
- Mannerism (Michelangelo, Palladio)
- Baroque
- Engineering/Rational/National/Romantic
- Art nouveau
- Modern movement (Wright, LeCorbusier, Geary, Libeskind)

Progress
- Imitation of previous efforts
- Learning from failure
- Integration of other forces
- Experimentation
Kinds of civil architecture

- **Community**
  - houses, flats and apartments, gardens, education, hospitals, religion

- **Commerce**
  - shops and stores, restaurants, hotels, office buildings, banks, airports

- **Industry**
  - industrial buildings, laboratories, farm buildings

- **Leisure**
  - sport, theaters and cinemas, museums

*Neufert, Architect’s Data*
Mechanisms in mechanical engineering

- Screws
- Keys
- Rivets
- Bearings
- Pins, axles, shafts
- Couplings
- Ropes, belts, and chains
- Friction wheels
- Toothed wheels
- Flywheels
- Levers and connecting rods
- Click wheels and gears
- Ratchets
- Brakes
- Pipes
- Valves
- Springs
- Cranks and rods
- Cams
- Pulleys
- Engaging gears
Forces in civil architecture

Compression

Tension

Kinds of loads
- Dead loads
- Live loads
- Dynamic loads

Avoiding failure
- Safety factors
- Redundancy
- Equilibrium

Any time you depart from established practice, make ten times the effort, ten times the investigation. Especially on a very large project.

- LeMessuier
Shearing layers of change

- Site
- Skin
- Structure
- Services
- Space plan
- Stuff

Brand, *How Buildings Learn*
Everything has an architecture
The economics of software

Performance = (Complexity)^{(Process)} * (Team) * (Tools)

- **Performance** = Effort or time
- **Complexity** = Volume of human-generated code
- **Process** = Methods, notations, maturity
- **Team** = Skill set, experience, motivation
- **Tools** = Software process automation

Boehm, *Software Engineering Economics*
The software development paradox

- **Building quality software that matters is fundamentally hard**
  - All interesting software embodies essential complexity

- **Software-intensive systems**
  - Can amplify human intelligence, but they cannot replace human judgment
  - Fuse, coordinate, classify, and analyze information, but they cannot create knowledge

- **From vision to execution**
  - Not everything we want to build can be built
  - Not everything we want to build should be built
The limits of technology

- 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
Software development has been, is, and remains hard
Forces In Software

Functionality

Cost/Schedule

Compatibility

Performance

Reliability/Availability

Capacity

Security

Scalability

Fail safe/Fault tolerance

Technology churn

Resilience
Points of friction

- Start up
- Work product collaboration
- Communication
- Time starvation
- Stakeholder cooperation
- Stuff that doesn’t work
Misconceptions about architecture

- Architecture is just paper
- Architecture and design are the same things
- Architecture and infrastructure are the same things
- `<my favorite technology>` is the architecture
- A good architecture is the work of a single architect
- Architecture is simply structure
- Architecture can be represented in a single blueprint
- System architecture precedes software architecture
- Architecture cannot be measured or validated
- Architecture is a science
- Architecture is an art
Architecture is just paper  

- A system’s architecture ultimately resides in executable code
- A system’s architecture may be visualized in models
- Every system has an architecture; some architectures are made manifest and visible, many others are not
Architecture is design

- All architecture is design, but not all design is architecture
- Architecture focuses on *significant* design decisions, decisions that are both structurally and behaviorally important as well as those that have a lasting impact on the performance, reliability, cost, and resilience of the system
- Architecture involves the how and the why, not just the what
Architecture is infrastructure

- Infrastructure is an integral and important part of architecture, but there is more to architecture than just infrastructure.
- Significant patterns will manifest themselves at many different levels and dimensions of a system.
- Too narrow a view of architecture will lead to a very pretty infrastructure, but the wrong infrastructure for the problem at hand.
A given technology only serves to implement some dimension of an architecture

- The network is the architecture
- The database is the architecture
- The transaction server is the architecture
- J2EE is the architecture

Architecture is more than just a list of products

Technology shapes an architecture, but a resilient architecture should never be bound to all of the technologies that form it
Architecture is the work of a single architect

- Conceptual integrity is essential, but the complexity of most interesting systems leads development to be a team sport
- Fred Brooks (1975), but then Fred Brooks (1995)
- The architecture team is
  - Not a committee
  - Not a problem clearinghouse
  - Not an ivory tower
- The architecture team
  - Needs a clear leader
  - Requires a mix of specialties
  - Manifests itself at many levels in the system

Coplien & Harris, Organizational Patterns
Architecture is structure

- Architecture does involve structure, decomposition, and interfaces
- Architecture also involves behavior
- A system’s architecture is always projected to a given context
Architecture is flat

- Architecture is flat only in trivial systems
- Multiple stakeholders with multiple concerns lead to multiple views with multiple blueprints
- Using a single blueprint to represent all or most of system’s architecture leads to a semantic muddle
- The 4+1 view model has proven to be both necessary and sufficient for most interesting systems

Kruchten, “The 4+1 Model View”
System architecture comes first

- Software has a longer life than hardware
- Complex systems require well-informed hardware/software tradeoffs, which cannot be made in a strict sequence
- Forcing a hardware-first process typically leaves to stove pipe systems
Architecture can’t be measured

- The very purpose of a blueprint is to provide a tangible artifact that can be used to visualize, specify, construct, document - and reason about - a system

- A system’s architecture can be used to
  - Mitigate technical risks through the release of a continuous stream of executables
  - Improve learning and understanding and communicate important decisions
  - Accelerate testing and attack integration risks
  - Set expectations
  - Break in the development environment and the team
Architecture is a science ❌

- There exists only a modest body of knowledge about software architecture
- Scientific and analytical methods are lacking; those that do exist are hard to apply
- There is no perfect design; architecture involves the management of extreme ambiguity and contradiction
- Experience counts: the best architects are grown, not born

Petroski, Small Things Considered
Architecture is an art

- Even the best architects copy solutions that have proven themselves in practice, adapt them to the current context, improve upon their weaknesses, and then assemble them in novel ways with very modest incremental improvements.
- The “artsy” part of software architecture is minimal.
- An architectural process can be established with intentional artifacts, clear activities, and well-defined.
Architecture defined

- **Architecture n (1563)**
  - The art or science of building or constructing edifices of any kind for human use
  - The action or process of building
  - Architectural work; structure, building
  - The special method of 'style' in accordance with which the details of the structure and ornamentation of a building are arranged
  - Construction or structure generally
  - The conceptual structure and overall logical organization of a computer or computer-based system from the point of view of its use or design; a particular realization of this

*Oxford English Dictionary, 2nd ed.*
Physical systems

- **Mature physical systems have stable architectures**
  - Aircraft, cars, and ships
  - Bridges and buildings

- **Such architectures have grown over long periods of time**
  - Trial-and-error
  - Reuse and refinement of proven solutions
  - Quantitative evaluation with analytical methods

- **Mature domains are dominated by engineering efforts**
  - Analytical engineering methods
  - New materials
  - New manufacturing processes
Software-intensive system

- A system in which software is the dominant, essential, and indispensable element
  - E-commerce system
  - IT (business) system
  - Telephone switch
  - Flight control system
  - Real-time control system (e.g. industrial robot)
  - Sophisticated weapons system
  - Software development tools
  - System software (e.g. operating systems or compilers)
Architecting software is different

- No equivalent laws of physics
- Transparency
- Complexity
  - Combinatorial explosion of state space
  - Non-continuous behavior
  - Systemic issues
- Requirement and technology churn
- Low replication and distribution costs
Architecture defined

- **Software architecture is what software architects do**
Architecture defined

- Perry and Wolf, 1992
  - A set of architectural (or design) elements that have a particular form

- Boehm et al., 1995
  - A software system architecture comprises
    - A collection of software and system components, connections, and constraints
    - A collection of system stakeholders' need statements
    - A rationale which demonstrates that the components, connections, and constraints define a system that, if implemented, would satisfy the collection of system stakeholders' need statements

- Clements et al., 1997
  - The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them

http://www.sei.edu/architecture/definitions
Common elements

- Architecture defines major **components**
- Architecture defines component **relationships** (structures) and **interactions**
- Architecture omits content information about components that does not pertain to their interactions
- Behavior of components is a part of architecture insofar as it can be discerned from the point of view of another component
- Every system has an architecture (even a system composed of one component)
- Architecture defines the **rationale** behind the components and the structure
- Architecture definitions do not define what a component is
- Architecture is not a single structure -- no single structure is the architecture
Architecture defined

- Architecture establishes the context for design and implementation

Architectural decisions are the most fundamental decisions; changing them will have significant ripple effects.
Architecture defined

- **IEEE 1471-2000**
  - Software architecture is the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution

- **Software architecture encompasses the set of significant decisions about the organization of a software system**
  - Selection of the structural elements and their interfaces by which a system is composed
  - Behavior as specified in collaborations among those elements
  - Composition of these structural and behavioral elements into larger subsystems
  - Architectural style that guides this organization

Booch, Kruchten, Reitman, Bittner, and Shaw
Architecture defined

- **Software architecture also involves**
  - Functionality
  - Usability
  - Resilience
  - Performance
  - Reuse
  - Comprehensibility
  - Economic and technology constraints and tradeoffs
  - Aesthetic concerns
Architectural style defined

- **Style** is the classification of a system's architecture according to those with similar patterns.

- A pattern is a common solution to a common problem; patterns may be classified as idioms, mechanisms, or frameworks.
Model, views, concerns, and stakeholders

- A model is a simplification of reality, created in order to better understand the system being created; a semantically closed abstraction of a system.

- A view is a representation of a whole system from the perspective of a related set of concerns.

- A concern is those interests which pertain to the system's development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders.

- A stakeholder is an individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system.
Stakeholders and views

- Architecture is many things to many different stakeholders
  - End user
  - Customer
  - Sys admin
  - Project manager
  - System engineer
  - Developer
  - Architect
  - Maintainer
  - Tester
  - Other systems

- Multiple realities, multiple views and multiple blueprints exist
Representing software architecture

Logical View
- End-user
  - Functionality

Implementation View
- Programmers
  - Configuration management

Use Case View

Process View
- System integrators
  - Performance
  - Scalability
  - Throughput

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

Conceptual
Physical

Clements, et al, *Documenting Software Architectures*
Adapting views

- Not all systems require all views
  - Single process (ignore process view)
  - Small program (ignore implementation view)
  - Single processor (ignore deployment view)

- Some systems require additional views
  - Data view
  - Security view
  - Other aspects
Cross functional mechanisms

- Some structures and behaviors crosscut components
  - Security
  - Concurrency
  - Caching
  - Persistence

- Such elements usually appear as small code fragments sprinkled throughout a system

- Such elements are hard to localize using traditional approaches
 Logical view

- The view of a system’s architecture that encompasses the vocabulary of the problem and solution space, the collaborations that realize the system’s use cases, the subsystems that provide the central layering and decomposition of the system, and the interfaces that are exposed by those subsystems and the system as a whole.

- Focuses on
  - Functionality
  - Key Abstractions
  - Mechanisms
  - Separation of concerns and distribution of responsibilities
Process view

- The view of a system’s architecture that encompasses the threads and processes that form the system’s concurrency and synchronization mechanisms

- Focuses on
  - Performance
  - Scalability
  - Throughput
Implementation view

- The view of a system's architecture that encompasses the components used to assemble and release the physical system.

- Focuses on
  - Configuration management
Deployment view

- The view of a system’s architecture that encompasses the nodes that form the system’s hardware topology on which the system executes

- Focuses on
  - Distribution
  - Communication
  - Provisioning
Use case view

- The view of a system’s architecture that encompasses the use cases that describe the behavior of the system as seen by its end users and other external stakeholders.
Relations among views

Logical view

Implementation view

Process view

Deployment view

α

β
Architecture metamodel

QuickTime™ and a TIFF (Uncompressed) decompressor are needed to see this picture.
Architecture metamodel

QuickTime™ and a TIFF (Uncompressed) decompressor are needed to see this picture.
Architecture metamodel

QuickTime™ and a TIFF (Uncompressed) decompressor are needed to see this picture.
Fundamentals never go out of style
Fundamentals

- Crisp abstractions
- Clear separation of concerns
- Balanced distribution of responsibilities
- Simplicity
Sources of architecture

Theft → Method → Intuition

Classical system

Theft → Method → Intuition

Unprecedented system
The best architectures are full of patterns
Patterns

- A pattern is a common solution to a common problem
- A pattern codifies specific knowledge collected from experience in a domain
- A pattern resolves forces in context
- All well-structured systems are full of patterns
  - Idioms
  - Mechanisms
  - Frameworks

http://www.hillside.net
Mechanisms

- Mechanisms (design patterns) are the soul of an architecture
- Gang of Four patterns
  - Creational patterns
  - Structural patterns
  - Behavioral patterns

Gamma, et al Design Patterns
The soul of an architecture is found in its mechanisms that cut across the components of the system, thus yielding its essential structures and behaviors.
Frameworks

- **Frameworks (architectural patterns) provide an extensible template for applications within a domain**

- **Shaw/Garlan patterns**
  - Dataflow systems
    - Batch sequential
    - Pipes and filters
  - Call-and-return systems
    - Main program and subroutine
    - OO systems
    - Hierarchical layers
  - Independent components
    - Communicating processes
    - Event systems
  - Virtual machines
    - Interpreters
    - Rule-based systems
  - Data-centered systems
    - Databases
    - Hypertext systems
    - Blackboards

Shaw et al, *Software Architecture*
Elements of a pattern

- Name
- Problem
- Forces
- Solution
- Strategies
- Benefits and drawbacks
- Related patterns
The simplest architectures are best
Beauty

- Elegance is not an approach to finding a solution to a problem, it is the label we stick on the optimum solution.
- Elegance is doing the most with the least.
- Elegance means simplicity and less new code.
- An elegant solution solves the whole problem." [Fisher, p. 37]
Architect defined

- An architect is the person, team, or organization responsible for a system’s architecture
- The life of a software architect is a long and sometimes painful succession of suboptimal decisions made partly in the dark
- Not just a top level designer
  - Need to ensure feasibility; has skin in the game
- Not the project manager
  - But “joined at the hip”
- Not a technology expert
  - Purpose of the system and “fit”,
- Not a lone wolf
  - Communicator and leader

Kruchten
Responsibilities of the architect

- Define and validate the system’s architecture
- Maintain the conceptual integrity of the system
- Assess and attack technical risks to the system
- Propose the order and contents of the continuous stream of executable releases
- Facilitate communication among team members and resolve conflict
- Mentor team members
- Along with the project manager, serve as the public persona of the project
Focus over time

Discovery  Invention  Implementation

Focus
Process best practices

- Attack major risks early and continuously or else they will attack you
- Ensure that you deliver value to your customer
- Have a maniacal focus on working software
- Accommodate change early in the project
- Baseline an executable architecture early on
- Build your system with components
- Work closely together as one team
- Make quality a way of life, not an afterthought
The development lifecycle

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preliminary</td>
<td>Architect.</td>
<td>Architect.</td>
<td>Transition</td>
</tr>
<tr>
<td>Devel.</td>
<td>Devel.</td>
<td>Devel.</td>
<td>Iteration</td>
</tr>
<tr>
<td>Iteration</td>
<td>Iteration</td>
<td>Iteration</td>
<td></td>
</tr>
</tbody>
</table>

- **Inception**
  - Understand what to build

- **Elaboration**
  - Understand how to build it

- **Construction**
  - Build the product

- **Transition**
  - Deliver and adapt the solution
Iterative and incremental development

Modern Project Profile

Waterfall Project Profile

Project Schedule

Development Progress (% Coded)
Architecture first

Risk resolution  Controlled risk management

Risk

Time

Waterfall

Iterative

Risk

© 2004 IBM Corporation
An architecture must grow and adapt or die
Every architecture has a half-life; sometimes you have to break the foundation
Software archeology defined

- 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.
The process of archeology

- Study source code
- Reverse engineering
- Probing and other instrumentation
- Review existing documents
- Interview tribal leaders
The process of archeology

- Most design information lives in tribal memory
- Typically there exists very high level architectural views and very low level design views, but little in between
- Reconstructing deployment and implementation views is easy
- Reconstructing the use case view is possible
- Reconstructing the logical and process views is hard
- Harvesting patterns is harder still
Preservation of classic software

- No comprehensive and intentional activity has yet been undertaken to preserve the industry’s seminal software artifacts

- There are a number of reasons to act now
  - Many of the authors of such systems are still alive
  - Many others may have the source code or design documents for these systems collecting dust in their offices or garages
  - Time is our enemy

http://www.computerhistory.org
Goals

- Preserving such artifacts for future generations is more than a valuable historical curiosity
  - The understanding and codification of architectural patterns
  - The evolution of software architecture and how they were products of their time
  - A statement of prior art relevant to the issues of proving and disproving software patents

- Preserving such artifacts provides raw materials for future generations of archeologists, historians, and developers who can learn from the past regarding
  - What worked and what didn't
  - What was brilliant and what was an utter failure
The big questions

- What systems do we preserve?
- What artifacts should we collect?
- How do we make these artifacts available?
- How can we create a sustainable program?
Handbook of software architecture

- No architectural reference exists for software-intensive systems

- Goals of the handbook
  - Codify the architecture of a large collection of interesting software-intensive systems
  - Study these architectural patterns in the context of the engineering forces that shaped them
  - Satisfy my curiosity

http://www.booch.com/architecture
Systems under study

- **Artificial Intelligence**
  - Systems that simulate or augment human cognition, locomotion, or other organic processes

- **Commercial and Non-Profit**
  - Systems that are fundamental to the operation of a business enterprise or that provide some essential end customer value

- **Communications**
  - Systems that provide the infrastructure for transferring and managing data, for connecting users of that data, or for presenting data at the edge of an infrastructure

- **Content Authoring**
  - Systems that are used to create or edit textual or multimedia artifacts
Systems under study

- **Development**
  - Systems that are used to develop other systems

- **Devices**
  - Systems that interact with the physical world to provide some individual point service

- **Entertainment and Sports**
  - Systems that manage public events or that provide a large group entertainment experience

- **Financial**
  - Systems that provide the infrastructure for transferring and managing money and other securities
Systems under study

- **Games**
  - Systems that provide an entertainment experience for individuals or groups

- **Industrial**
  - Systems that control physical processes

- **Legal**
  - Systems that support the legal industry

- **Medical**
  - Systems that diagnose or heal or that contribute to medical research

- **Military**
  - Systems for consultation, communications, command, control, and intelligence (C4I) as well as offensive and defensive weapons
Systems under study

- **Operating Systems**
  - Systems that sit just above hardware to provide basic software services

- **Platforms**
  - Systems that sit just above operating systems to provide advanced services

- **Scientific**
  - Systems that are used for scientific research and applications

- **Transportation**
  - Systems that control ground, air, or water vehicles

- **Utilities**
  - Systems that interact with other software to provide some individual point service
Summary

- Every interesting software-intensive system has an architecture
- The most successful software architectures are intentional, manifest, and visible - and beautiful
Thank you!