Modern Trends through an Architecture Lens

Linda Northrop
Society’s Dependence on Software
Modern Technology Trends

- Cloud Computing
- Mobile Computing
- AI Machine Learning
- Autonomous Systems
- Cyber-Physical-Social Ecosystems
Software Development Trends

- Agile approaches
- DevOps
- Scripting languages
- Dashboards
- Application frameworks
- Distributed development environments
- Open source libraries
- Containers
- Microservices
- NoSQL
Mired in Legacy Systems
Today’s Software Workforce
Different Abilities Have Diverse Needs

https://www.wpsbc.org/
Software Engineering Challenges
The Intersection and Architecture

At the intersections there are difficult tradeoffs to be made - in structure, technology, process, and cost.

Architecture is the enabler for tradeoff analyses.
Software Architecture

- High-level system design providing system-level structural abstractions and quality attributes, which help in managing complexity
- Makes engineering tradeoffs explicit
Quality Attributes

Quality attributes

- properties of work products or goods by which stakeholders judge their quality
- stem from business and mission goals.
- need to be characterized in a system-specific way

Quality attributes include
- Performance
- Availability
- Interoperability
- Modifiability
- Usability
- Security
- Etc.
Central Role of Architecture

Diagram:
- BUSINESS AND MISSION GOALS
- ARCHITECTURE
- SYSTEM

Steps:
1. DESIGN
2. IMPLEMENT
3. SATISFY
4. CONFORM

Flow:
- From BUSINESS AND MISSION GOALS to ARCHITECTURE
- From ARCHITECTURE to SYSTEM
- From SYSTEM to BUSINESS AND MISSION GOALS
Architectural Advancements Over the Years

Architectural patterns and tactics
Component-based approaches
Company-specific product lines
Model-based approaches
Aspect-oriented approaches
Frameworks and platforms
Standard interfaces
Standards
SOA
Microservices

Persisting Themes
- Modularity
- Commonality vs Variability
What Now?

Is structure still important?
Are old architectural principles still relevant?
What are the architectural drivers in today’s systems?
What kind of new architectural styles are needed to be able to ensure intended behavior?
What kinds of design paradigms will help us ensure the safety, security, and reliability of systems with artificial intelligence and autonomy?
Should such systems be tested with different approaches?
Can the design of these systems be engineered to ensure their testability?
How can the design of software assist in ensuring its intended ethical use?
Cloud Computing Models and Essential Characteristics

- **Service Models**
  - Software as a Service (SaaS)
  - Platform as a Service (PaaS)
  - Infrastructure as a Service (IaaS)

- **Deployment Models**
  - Public
  - Private
  - Hybrid
  - Community

- **Essential Characteristics**
  - Measured Service
  - Resource Pooling
  - On-Demand Self Service
  - Broad Network Access
  - Rapid Elasticity

Source: National Institute of Standards and Technology (NIST), 2011
Architecture Perspective

Shared responsibility

Two potentially different sets of business goals and quality attributes
• SLA is an architectural decision
• Portability tradeoffs
• Tempo differences
• Runtime tradeoffs

Architectural tradeoffs involve cost

Testing challenges that require architectural support
• Controllability
• Observability
Special Cases

Migration to the cloud
• Challenges in interoperability, latency, legal, platform or language constraints, security, skill set, compliance, and portability
• Restructure to take advantage of cloud performance

Function as a Service (FaaS)
• Serverless architecture style
• Cloud-aligned architectures
Mobile Computing

Today’s UI is increasingly mobile.
Related Architecture Trends: Edge and Fog Computing

Edge and Fog Computing push cloud resources to the edge of the network

Cyber-foraging is the process of discovering Edge and Fog resources for computation offload and data staging

- Reduced network traffic
- Reduced latency
- Improved user experience

Architecture challenges

- Data and computation allocation to the right nodes at the right time
- Resource discovery
- Security and privacy

Involves
- Data analytics
- Infrastructure

Analytics is typically a massive data reduction exercise – “data to decisions.”

Computation infrastructure necessary to ensure the analytics are
- fast
- scalable
- secure
- easy to use
Data is a business asset.
Big Data Software Architecture

Big Data Characteristics
- Volume
- Velocity
- Variety
- Value
- Veracity

Big Data Architecture
- Messaging and Storage
- Analytics
- Big Data Sources
- Consumption and Utilization

Architectural Requirements
- (Hyper)-Scalability
- Resilience
- Distribution
- Observability
- Adaptability
- Security
Artificial Intelligence (AI)

AI is creating a revolution in system capability.

• Data analytics
• Cooperative autonomous systems
• UX/collaboration modalities
• Cyber autonomy and counter-autonomy
• Bug repair, self-healing, and self-adaptive systems

There are also reasonable fears.

“A Vision for Software Development,” Andrew Moore, Carnegie Mellon University, Jan 6, 2018
"A Vision for Software Development," Andrew Moore, Carnegie Mellon University, Jan 6, 2018
Machine Learning (ML) Systems Today

Machine learning: learning to predict by extrapolating from data
- Can provide rapid response to dramatically changing contexts
- Algorithms are readily accessible
- Effectiveness is highly variable across different domains
- Overall, today still a cottage industry
ML, Software Engineering, and Architecture

1. Correctness will not be possible.
   • ML has an experimental mindset.
2. Holistic testing is impossible
   • uncertainty and error will be part of the output
3. Deductive reasoning from the code and metadata is not and will not be effective.
4. Data and its attributes must be first class.
5. Divide and conquer doesn’t work.
6. Quality attribute focus
   • reliability
   • observability
Autonomous Systems
Autonomous Systems, Software Engineering, and Architecture

ML issues plus structural support for
• Human/Machine collaboration
• Safety
• Timing
• Security
• Adaptation
• Edge case handling
Cyber-Physical-Social Ecosystems
Software Engineering and Architecture

Design of and at all levels
• Governance
• Standards
• Certification

Platforms that admit heterogeneity and provide
• Interoperability
• Scalability
• Extensibility
• Timing
• Security
• Monitorability
• Resilience/self-adaptation
Incremental Development and Architecture

Architecture design can be done incrementally. There is a difference between being agile and doing agile.

Agility is enabled by architecture – not stifled by it. Architecture needs to be versatile, easy to evolve, and easy to modify, while resilient enough not to degrade after a few changes.

Architecture has a role to play in supporting agile at scale.
Integrated Agile/Architecture Practices

Successful project teams find architecture practices to be a key enabler for agility.


DevOps

Focus is on

- culture and teaming
- process and practices
  - value stream mapping
  - continuous delivery practices
  - *Lean* thinking
- tooling, automation, and measurement
  - tooling to automate manual, repetitive tasks
  - static analysis
  - automation for monitoring architectural health
  - performance dashboards
DevOps and Architecture

Architect the system for deployability.
Architect the tool chain.
- Integrate security into DevOps.
Architect the IaC.
Implement the architecture you design.
- Write custom checks for implementation conformance.
- Automate tests for quality attributes.
- Collect data to monitor health of the architecture.

Design decisions that involve deployment-related limitations can blindside teams.
## Deployability Tactics

<table>
<thead>
<tr>
<th>Availability</th>
<th>Modifiability</th>
<th>Performance</th>
<th>Testability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor</td>
<td>Encapsulate</td>
<td>Increase Resources</td>
<td>Sandbox</td>
</tr>
<tr>
<td>Exception Detection</td>
<td>Defer Binding</td>
<td>Increase Currency</td>
<td>Specialized Interfaces</td>
</tr>
<tr>
<td>Exception Handling</td>
<td>Abstract Common Services</td>
<td>Schedule Resources</td>
<td>Record/Playback</td>
</tr>
<tr>
<td>Voting</td>
<td></td>
<td>Reduce Overhead</td>
<td></td>
</tr>
<tr>
<td>Rollback</td>
<td></td>
<td>Maintain Multiple Copies of Computations</td>
<td></td>
</tr>
<tr>
<td>Active Redundancy</td>
<td></td>
<td>Maintain Multiple Copies of Data</td>
<td></td>
</tr>
<tr>
<td>Reconfiguration</td>
<td></td>
<td>Limit Event Response</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Prioritize Events</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage Sampling Rate</td>
<td></td>
</tr>
</tbody>
</table>

Frameworks, Libraries, Containers, etc.

Reuse abounds.
Rapid construction through assembly.
Architecture is implicit.
Undesirable behavior can occur.
Debilitating technical debt can occur.
Technical Debt*

Technical debt* is a collection of design or implementation choices that are expedient in the short term, but that can make future changes more costly or impossible.


### Exists in

- Code
- Build scripts
- Data model
- Automated test suites
- Structural decisions
Results from over 1800 developers from two large industry and one government software development organization list architecture as the most costly technical debt.

All systems have technical debt. The impact depends on how you manage it.
Security concerns are paramount.

It’s not just about security, but functioning as intended and only as intended.

Supply chains, open source, frameworks, outsourcing introduce unknowns.

Tool chains that generate code, configuration files, etc. introduce unknowns.

Autonomy, machine learning, and connected physical systems introduce unknowns.

Humans in the system introduce unknowns.

Consequences include operational failures, security and privacy compromises, reputational impact, etc.
So Where Are We?

There are tradeoffs, tension, and needs for educated decisions and measurement. Architecture is still the enabler for tradeoff analyses and evidence, but there is a changed architectural workforce and new architectural needs.
Today’s Software Architecture Workforce

Differs widely by organization and domain
Reveals a democratization of the architecture
Has a spectrum of experience
Tension between wisdom of the crowd and experience
And yet…

• How do quality attributes get distributed across team(s)?
• Technical debt is accruing due to lack of architectural thinking.
• More design horse power (not less) needed for complex systems and specialized domains.
• More talent (not less) needed, some from other disciplines.
Net Sum Architectural Needs

Tradeoffs, decisions, structure persists.
Security needs are heightened.
Different quality attributes at the fore.
New focus on
- Evolution
- Runtime
- Data
- ML
- Automation
Evolution and Runtime

Evolution
  • Explicitly design for continuous evolvability and adaptability in order to deal with uncertainty and not incur prohibitive technical debt
  • Decisions will reflect changing principles, policies, and algorithms

Runtime
  • Architecture needs to be seen at runtime
  • Observability: mechanisms to support continuous monitoring
  • Recovery, auto-scaling, managed roll-out
  • Dynamic adaptation to support environmental changes and tradeoff priorities
  • Configuration changes at runtime without performance hits
  • Human-in-the-system models
  • Situational awareness and explanation
Data and ML

Data

• Data and its attributes must be first class citizens
• Relax current design heuristics; e.g., how to decouple components and data
• Software analysis tools will need to reason about data

ML

• Certainty will give way to probability
• Ability to articulate the tradeoffs in ML
• Criteria for whether ML is a good solution for a given problem
• Architecture patterns that allow post mortem of ML systems
Automation

Tools to support design and architecture

- At design time for discovery, envisioning, and collaboration
- At run time for observation and environmental monitoring
- To embed design alternatives with code as part of the build system
- To detect and manage technical debt
- To move from explicit decisions to principles with guide rails
  - guide rails that are manifest in the code
  - “smart” frameworks; architecture hoisting

ML to collaborate with designers and to understand the impact of design decisions
Conclusion

Structural decisions continue to be made. Tradeoffs continue to be made. Software architecture importance persists. But…

- The focus must fit today’s environment and needs.
- Architects need to embrace uncertainty.
- New tooling is essential.
Contact Information

Linda Northrop
SEI Fellow
Telephone: 1+ 412-268-7638
Email: lmn@sei.cmu.edu
@LindaNorthrop
Website: http://www.sei.cmu.edu/architecture

U.S. Mail:
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA  15213-3890