Trends and New Directions in Software Architecture

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA  15213

Linda Northrop
Chief Scientist, Software Solutions Division
SEI Fellow
Copyright 2015 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

This material has been approved for public release and unlimited distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

ATAM® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

Team Software Process℠ and TSP℠ are service marks of Carnegie Mellon University.

DM-0001699
Software Architecture

- The quality and longevity of a software-reliant system is largely determined by its architecture.
- Recent US studies identify architectural issues as a systemic cause of software problems in government systems (OSD, NASA, NDIA, National Research Council).

Architecture is of enduring importance because it is the right abstraction for performing ongoing analyses throughout a system’s lifetime.
Software Architecture Thinking

- 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

IMPLEMENT AND EVOLVE

DESIGN

BUSINESS AND MISSION GOALS

ARCHITECTURE

IMPLEMENT

SYSTEM

SATISFY

CONFORM

SATISFY
Our View: Architecture-Centric Engineering

- explicitly focus on quality attributes
- directly link to business and mission goals
- explicitly involve system stakeholders
- be grounded in state-of-the-art quality attribute models and reasoning frameworks
Advancements Over the Years

- Architectural patterns
- Component-based approaches
- Company specific product lines
- Model-based approaches
- Frameworks and platforms
- Standard interfaces
What HAS Changed?

- Increased connectivity
- Scale and complexity
  - decentralization and distribution
  - “big data”
  - increased operational tempo
  - inter-reliant ecosystems
  - vulnerability
  - collective action
- Disruptive and emerging technologies
Software Development Trends

- Application frameworks
- Open source
- Cloud strategies
- NoSQL
- Machine Learning
- MDD
- Incremental approaches
- Dashboards
- Distributed development environments
- DevOps
Technical Challenges
At the intersections there are difficult tradeoffs to be made in structure, process, time, and cost.

Architecture is the enabler for tradeoff analyses.
Architecture and Accelerated Capability

How much architecture design is enough?

Can architecture design be done incrementally?

There is a difference between being agile and doing agile.

Agility is enabled by architecture – not stifled by it.

Managing technical debt is key.
Managing Technical Debt*

A design or construction approach that's expedient in the short term but that creates a technical context that increases complexity and cost in the long term.

Some examples include:

- continuing to build on a foundation of poor quality legacy code
- prototype that turns into production code
- increasing use of "bad patches," which increases number of related systems that must be changed in parallel

Technical Debt Impact

From: Jim Highsmith
2010
“invisible results of past decisions about software that negatively affect its future…deferred investment opportunities or poorly managed risks”

Making Hard Choices About Technical Debt

In the quest to become market leader, players race to release a quality product to the marketplace.

The Hard Choices game is a simulation of the software development cycle meant to communicate the concepts of uncertainty, risk, options, and technical debt.

Do you take the time to gather more tools or do you take a shortcut?
Our Current Research

What code and design indicators that correlate well with project measures allow us to manage technical debt?

1. time technical debt is incurred
2. time technical debt is recognized
3. time to plan and re-architect
4. time until debt is actually paid-off
5. continuous monitoring
Bolsa Mexicana de Valores (BMV) operates the Mexican Financial Markets on behalf of the Mexican government.

Bursatec is the technology arm of the BMV.

BMV desired a new stock trading engine to drive the market.

BMV performed a build vs. buy analysis and determined that Bursatec would replace their three existing trading engines with one in-house developed system.

Bursatec committed to deliver a trading engine in 8-10 quarters.

- High performing
- Reliable and of high quality
- Scalable
Approach

Team Software Process (TSP) and Architecture-Centric Engineering
Effort in Percent over Cycles – 1

Cycle 1 – 14 Weeks

Reqs: Requirements
HLD/Arch: High level Design / Architecture
DLD: Detailed Design (UML)
Code: Coding (no detailed design)
Test: Testing
Effort in Percent over Cycles – 2

Cycle 2 – 10 Weeks

- Reqs: Requirements
- HLD/Arch: High level Design / Architecture
- DLD: Detailed Design (UML)
- Code: Coding (no detailed design)
- Test: Testing
Effort in Percent over Cycles – 3

Cycle 3 – 18 Weeks

Reqs: Requirements
HLD/Arch: High level Design / Architecture
DLD: Detailed Design (UML)
Code: Coding (no detailed design)
Test: Testing
• The fourth cycle of three weeks was used to rethink garbage collection handling and cleaning up.

• No effort data was collected during that cycle.
Effort in Percent over Cycles – 5

Cycle 5 - 25 Weeks

- Reqs: Requirements
- HLD/Arch: High level Design / Architecture
- DLD: Detailed Design (UML)
- Code: Coding (no detailed design)
- Test: Testing
Effort in Percent over Cycles – 6

Cycle 6

- Reqts: Requirements
- HLD/Arch: High level Design / Architecture
- DLD: Detailed Design (UML)
- Code: Coding (no detailed design)
- Test: Testing
# Results

<table>
<thead>
<tr>
<th>Results</th>
<th>Target</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency</td>
<td>1ms</td>
<td>0.1ms</td>
</tr>
<tr>
<td>Throughput (transactions per second)</td>
<td>1,000</td>
<td>200,000</td>
</tr>
<tr>
<td>Schedule (months)</td>
<td>18</td>
<td>17</td>
</tr>
<tr>
<td>Quality (defects/KLOC found during validation testing)</td>
<td>0.25</td>
<td>0.1</td>
</tr>
</tbody>
</table>
Deployment Challenges

The **DevOps** movement continues what Agile started.

---

**Features Complete**
Not Released

**Tried to Deploy,**
errors cause rollback

---

**ACCELERATING**
**CAPABILITY**
DevOps: State of the Practice

Focus is on

- culture and teaming
- process and practices
  - value stream mapping
  - continuous delivery practices
  - *Lean* thinking
- tooling, automation, and measurement
  - tooling to automate repetitive tasks
  - static analysis
  - automation for monitoring architectural health
  - performance dashboards
Design decisions that involve deployment-related limitations can blindside teams.
DevOps Tips

- Don’t let designing for deployability be an afterthought.
- Use measurable deployability quality attributes.
- Consider architectural tactics that promote modifiability, testability, and operational resilience.
- Use architectural abstractions to reason about deployability implications of design options and tradeoffs.
- Establish monitoring mechanisms.
Architecture and Scale

- Cloud strategies
- Cloud strategies for mobility
- Big data

“Scale Changes Everything”
Two Perspectives of Software Architecture in Cloud Computing

Two potentially different sets of business goals and quality attributes
Cloud Computing and Architecting

- SLAs cannot prevent failures.
- In cloud environments,
  - cloud consumers have to design and architect systems to account for lack of full control over important quality attributes.
  - cloud providers have to design and architect infrastructures and systems that provide the most efficient way to manage resources and keep promises made in SLAs.
Architecture Trends: Cyber-Foraging

- Edge Computing
- Using external resource-rich surrogates to augment the capabilities of resource-limited devices
  - code/computation offload
  - data staging
- Industry is starting to build on this concept to improve mobile user experience and decrease network traffic.
- Our research: cloudlet-based cyber-foraging
  - brings the cloud closer to the user
Two very distinct but related technological thrusts
- 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
Building scalable, assured big data systems is hard.

Building scalable, assured big data systems is expensive.
Big Data Survey

Top Requirements of Big Data Solutions

55% of Big Data Projects are not completed.

When it comes to Big Data projects, the most significant challenge is:

- 58% Inaccurate Scope
- 80% Finding Talent
- 76% Finding the Right Tools
- 73% Understanding Education

#1 Ease of Management
#2 Ability to Scale

http://visual.ly/cios-big-data
System costs must grow more slowly than system capacity.

Approaches
- scalable software architectures
- scalable software technologies
- scalable execution platforms

Scalability reduces as implementation complexity grows.

NoSQL models are not created equal.

You can’t manage what you don’t monitor.
Our Current Research

- Lightweight Evaluation and Architecture Prototyping for Big Data (LEAP4BD)
- QuABase: A Knowledge Base for Big Data System Design
  - semantics-based knowledge model
    - general model of software architecture knowledge
    - populated with specific big data architecture knowledge
  - dynamic, generated, and queryable content
  - knowledge visualization
Architectural Models

- capture architecture in a form amenable to analysis, which contributes to assurance
- range from informal (e.g., visio diagrams) to formal (e.g., with precisely defined execution semantics)
- In safety critical systems formality is warranted.
High Fault Leakage Drives Major Increase in Rework Cost

Aircraft industry has reached limits of affordability due to exponential growth in SW size and complexity.

- 70%, 3.5% 1x
- 10%, 50.5% 20x
- 20%, 16% 5x

80% late error discovery at high rework cost

Where faults are introduced
Where faults are found

- The estimated nominal cost for fault removal
- Where faults are introduced
- Where faults are found

70% Requirements & system interaction errors

Post-unit test software rework cost: 50% of total system cost and growing

Software as % of total system cost
1997: 45% → 2010: 66% → 2024: 88%

Total System Cost
Boeing 777 $12B
Boeing 787 $24B

Acceptance Test
System Test
Integration Test
Unit Test
Code Development
Component Software Design
Software Architectural Design
System Design
Requirements Engineering

Sources:
SAE Architecture Analysis & Design Language (AADL) Standard Suite (AS-5506 Series)

- Core AADL language standard (V2.1-Sep 2012, V1-Nov 2004)
  - Strongly typed language with well-defined semantics
  - Textual and graphical notation
  - Standardized XMI interchange format

**Standardized AADL Extensions**
- Error Model language for safety, reliability, security analysis
- ARINC653 extension for partitioned architectures
- Behavior Specification Language for modes and interaction behavior
- Data Modeling extension for interfacing with data models (UML, ASN.1, …)
Architecture-Centric Quality Attribute Analyses

Single Annotated Architecture Model Addresses Impact Across Operational Quality Attributes

Safety Reliability
- MTBF
- FMEA
- Hazard Analysis

Data Quality
- Data precision/accuracy
- Temporal correctness
- Confidence

Architecture Model

Auto-generated analytical models

Real-time Performance
- Execution time/deadline
- Deadlock/starvation
- Latency

Security
- Intrusion
- Integrity
- Confidentiality

Resource Consumption
- Bandwidth
- CPU time
- Power consumption
Conclusion

- Software architecture principles and their importance persist.
- Change brings new challenges.
- Software architecture practices and research are key to meeting these challenges.
- Much remains to be done.
This Is the Work of Many

At the SEI
- Felix Bachmann
- Stephany Bellomo
- Peter Feiler
- Ian Gorton
- James Ivers
- Rick Kazman
- John Klein
- Mark Klein
- Grace Lewis
- Ipek Ozkaya
- Rod Nord
- and many more…
More Information

Related Blogs:
http://blog.sei.cmu.edu/archives.cfm/category/architecture

Technical Debt:
http://www.sei.cmu.edu/architecture/research/architectural_debt/architectural_debt_library.cfm

Agile Architecting:
https://www.sei.cmu.edu/architecture/research/agile-architecting/index.cfm

Cloudlets:
http://www.sei.cmu.edu/mobilecomputing/research/tactical-cloudlets/index.cfm

AADL and Model-Based Engineering:
http://www.sei.cmu.edu/architecture/research/model-based-engineering/index.cfm
Contact Information

Linda Northrop
SEI Fellow
Chief Scientist
Software Solutions Division
Telephone: 412-268-7638
Email: lmn@sei.cmu.edu
Website: http://www.sei.cmu.edu/architecture

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

SEI Fax: 412-268-5758