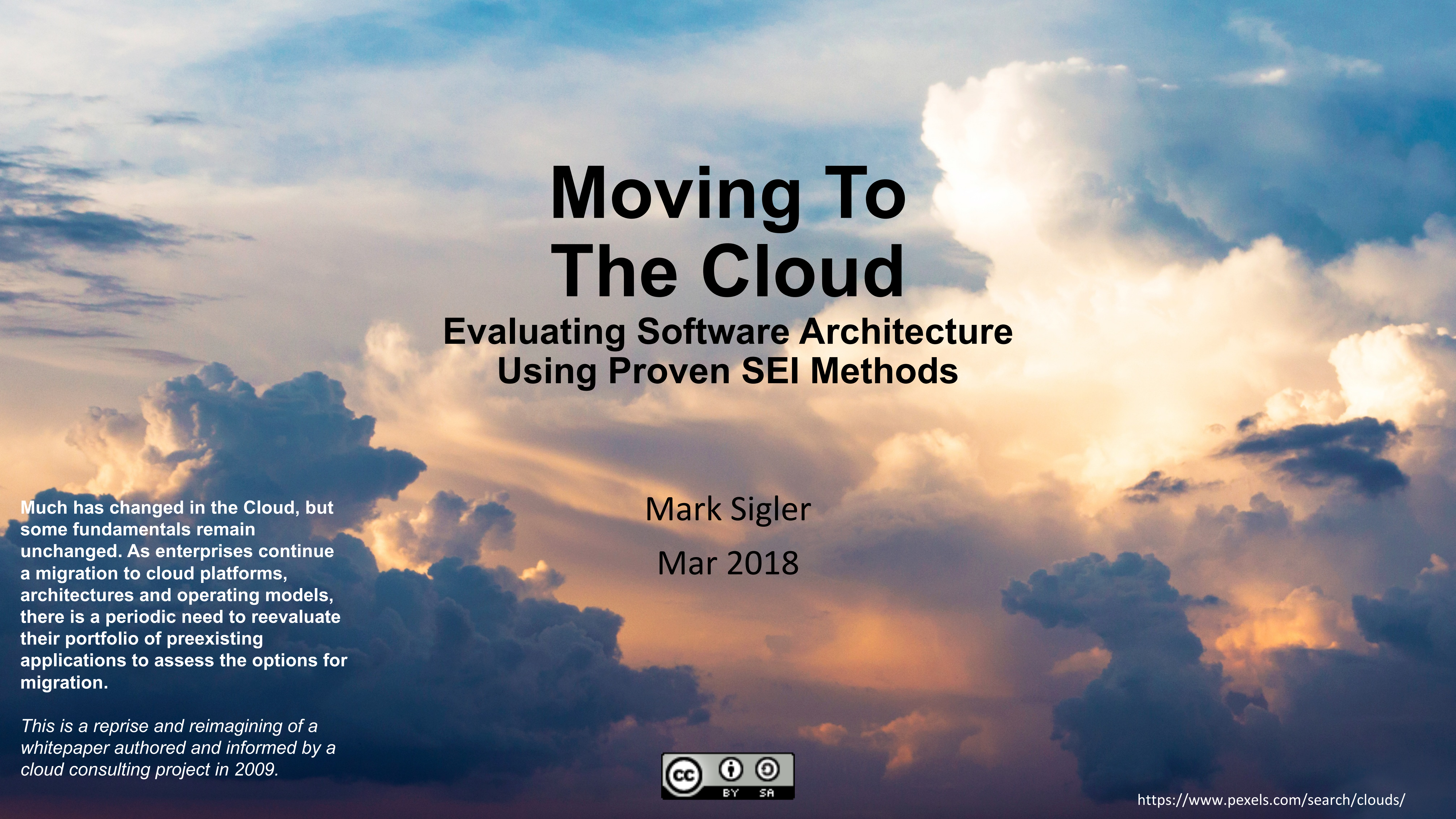

# Moving To The Cloud

## Evaluating Software Architecture Using Proven SEI Methods

Mark Sigler

Mar 2018

**Much has changed in the Cloud, but some fundamentals remain unchanged. As enterprises continue a migration to cloud platforms, architectures and operating models, there is a periodic need to reevaluate their portfolio of preexisting applications to assess the options for migration.**

*This is a reprise and reimagining of a whitepaper authored and informed by a cloud consulting project in 2009.*

https://www.pexels.com/search/clouds/

# EVALUATING A SOFTWARE ARCHITECTURE BALANCING RISK AND REWARD

- How to optimize for time to market and economics?
- How to reduce risk and improve capabilities?

"As in life, if you marry your architecture in haste, you and your stakeholders will repent in leisure."[1]
—Barry Boehm

[1] "Spiral Development: Experience, Principles and Refinement", Barry Boehm, Feb 2000

## AGENDA

- What is an Architecture?
- How to Validate a Software Architecture?
- Why Evaluate an Architecture?
- When Can an Architecture Be Evaluated?
- Who's Involved?
- What Are the Outputs of an Architecture Evaluation?
- What Are the Benefits and Costs?
- Conclusions

## WHAT IS AN ARCHITECTURE?

- Foundation for any software system.
- Basis for all system Abilities:
  - Usability
  - Maintainability
  - Reliability
  - Scalability
  - Extensibility
  - Securability
  - Portability
    - *Not necessarily in that order of priority*

Refer to Appendix for additional details

## HOW TO VALIDATE A SOFTWARE ARCHITECTURE?

- Three proven methods, each and all developed at the

Software Engineering Institute of Carnegie Mellon University:

  - ATAM: Architecture Tradeoff Analysis Method
  - SAAM: Software Architecture Analysis Method
  - ARID: Active Reviews for Intermediate Designs

# ARCHITECTURE TRADEOFF ANALYSIS METHOD (ATAM)

- Structured technique for understanding the tradeoffs inherent in the architectures of software-intensive systems.
- Principled way to evaluate a software architecture's fitness with respect to multiple competing quality attributes.
- Spiral model of design: one of postulating candidate architectures followed by analysis and risk mitigation, leading to refined architectures.

## SOFTWARE ARCHITECTURE ANALYSIS METHOD (SAAM)

- Predict the quality of a system before it has been developed.
- Quality of the architecture is validated by analyzing the impact of predefined scenarios on architectural components.
- Addresses concerns at the architecture design level which inherently crosscut multiple architectural components.

## ACTIVE REVIEWS FOR INTERMEDIATE DESIGNS (ARID)

- Method for reviewing preliminary software designs (such as for a component or a subsystem) for suitability in its intended usage context and environment.
- Results in a high-fidelity design review coupled with high-quality familiarization with the design.

## WHY EVALUATE AN ARCHITECTURE?

- The cost to fix an error found during requirements or early design phases is orders of magnitudes less to correct than error found in testing.
- Architecture determines the structure of the project: schedules and budgets, performance goals, team structure, documentation organization, and testing and maintenance activities.

## WHEN CAN AN ARCHITECTURE BE EVALUATED?

- Architecture evaluation occurs when the architecture has been specified but before implementation has begun.
- Lean and Agile methods suggest incremental experimentation, reevaluation and correction during implementation.
- Plan, Experiment, Adjust, Validate

# WHEN CAN AN ARCHITECTURE BE EVALUATED?

- Two useful variations:
  - Early - at any stage in the architecture creation process to examine those architectural decisions already made and choose among architectural options.
  - Late - takes place when the architecture is nailed down and the implementation is complete. Mainly used when architecture is inherited from legacy system.
- Migrating legacy applications to the cloud inherently is Late, but subject to further evaluation.

## WHO'S INVOLVED?

- Evaluation team - People who will conduct the evaluation and perform the analysis.
    - Responsible and some perhaps Consulted (RACI)
- Stakeholders - Stakeholders are people who have a vested interest in the architecture and the system.
    - Consulted and possibly Accountable (RACI)

## WHAT ARE THE OUTPUTS OF AN ARCHITECTURE EVALUATION?

- Prioritized Statement of Quality Attribute Requirements.
  - Having a prioritized statement of the quality attributes serves as an excellent documentation record to accompany any architecture and guide it through its evolution.

## WHAT ARE THE OUTPUTS OF AN ARCHITECTURE EVALUATION?

- Mapping of Approaches to Quality Attributes.
  - Produces a mapping that shows how the architectural approaches achieve (or fail to achieve) the desired quality attributes.

## WHAT ARE THE OUTPUTS OF AN ARCHITECTURE EVALUATION?

- Risks and Non-risks.
    - Risks are potentially problematic architectural decisions.
    - Non-risks are good decisions that rely on assumptions that are frequently implicit in the architecture.

## WHAT ARE THE BENEFITS AND COSTS?

- Forces an Articulation of Specific Quality Goals.
- Results in the Prioritization of Conflicting Goals.
- Puts Stakeholders in the Same Room.
- Improves the Quality of Architectural Documentation.
- Uncovers Opportunities for Cross-Project Reuse.

## CONCLUSION

- The average architecture evaluation adds no more than a few days to the project schedule. Spike sprint.
- Architecture created in haste will precipitate disaster: performance goals not met, Security goals falling, customer dissatisfaction, system that is too hard to change, and schedules and budgets through the roof. Operating model will disintegrate.

# Appendix

Background Information

# Abilities

Definitions for a Foundation

## ABILITIES

1) Usability

- Software usability can be described as how effectively end users can use, learn, or control the system. Some questions to ask yourself to determine usability might be:
  - Is there a UI metaphor that I am using to help users adapt? (for example, the ‘desktop’ is a metaphor)
  - Are the most common operations streamlined to be performed quickly?
  - Can new users quickly adapt to the software without help? (is it intuitive?)
  - Do validation and error messages make sense?
  - Is the visual design atheistically pleasing?

Paraphrased from http://codesqueeze.com/the-7-software-ilities-you-need-to-know/

## ABILITIES

2) Maintainability

- The relevant definition of maintainability implies how brittle the code is to change. As a result, I tie the terms flexibility and testability into the overall maintainability of a project.
  - Does the entire team understand the code base or does knowledge islands exist?
  - Is the code throughly regression tested?
  - Can modifications to the project be done in a timely manner?
  - Will upgrades of major and minor releases including patches be seamless, not impacting availability or functionality of the users?

Paraphrased from http://codesqueeze.com/the-7-software-ilities-you-need-to-know/

## ABILITIES

3) Scalability

- Scalability is the ability for your program to gracefully meet the demand of stress caused by increased usage. In short, ensuring your program doesn’t slow or bust when pounded by more users than you originally anticipated.
  - What is your current peak load that you can handle?
  - How many database records can create until critical operations slow down?
  - Is the primary scaling strategy to “scale up” or to “scale out” — that is, to upgrade the nodes in a fixed topology, or to add nodes?

Paraphrased from http://codesqueeze.com/the-7-software-ilities-you-need-to-know/

# ABILITIES

4) Reliability

- How long the system is up and running and the Mean Time Between Failure (MTBF) is known as the availability of a program.
  - How long does the system need to run without failure?
  - What is the acceptable length of time for the system to be down?
  - Can down times be scheduled?

Paraphrased from http://codesqueeze.com/the-7-software-ilities-you-need-to-know/

## ABILITIES

5) Extensibility

- Are there points in the system where changes can be made with (or without) program changes?
  - Can the database schema flex to accommodate change?
  - Does the system allow Inversion of Control (IoC)?
  - Can end users extend the system (scripts, user defined fields, etc)?
  - Can 3rd party developers leverage your system?

Paraphrased from http://codesqueeze.com/the-7-software-ilities-you-need-to-know/

## ABILITIES

6) Securability

- Measure of system's ability to resist unauthorized attempts at usage or behavior modification, while still providing service to legitimate users.
  - Does the system need user or role based security?
  - Does code access security need to occur?
  - What operations need to be secured?
  - How will users be administered?

Paraphrased from http://codesqueeze.com/the-7-software-ilities-you-need-to-know/

## ABILITIES

7) Portability

- Portability is the ability for your application to run on numerous platforms. This is can include actual application hosting, viewing, or data portability.
  - Can the data be migrated to other systems?
  - For web applications, which browsers does your web app support?
  - Which operating systems does your program run on?
  - Can the system be migrated to other cloud platforms?

Paraphrased from http://codesqueeze.com/the-7-software-ilities-you-need-to-know/

# References

Recommended Reading

## RECOMMENDED READING

- [Experiences in Migrations of Legacy Systems]
- [Methodology for the Cost Benefit Analysis of a Large Scale ...]
- [Automated Provisioning of Cloud and Cloudlet Applications]

https://resources.sei.cmu.edu/