Risk-Centric Model of Software Architecture

George Fairbanks

16 November 2009

Rhino Research
Software Architecture Consulting and Training

Full paper: http://rhinoresearch.com/content/risk-centric-model-software-architecture

Overview

- Software architecture techniques are helpful
- We cannot afford to apply them all

- Important: **How much should we do?**
  - Old: Yardstick model
  - Old: Full description model

- This talk describes a new model: Risk-Centric Model
  1. Identify and prioritize risks
  2. Apply relevant architecture activities
  3. Re-evaluate

- It is not: Big Design Up Front
- It is not: A full software development process

- It is: Compatible with agile & other processes

Talk outline

- **Architecture essentials**
- When do we design the architecture?
- How much architecture should we do?
- Risk-centric model
- Risks
- Techniques
- Processes & related work
- Conclusion

What is software architecture?

> The **software architecture** of a program or computing system is its structure or structures, which comprise software elements, the externally visible properties of those elements, and the relationships among them. [Bass, Clements, Kazman 2003]

- In loose language:
  - It's the macroscopic organization of the system
- **Must keep these ideas separate:**
  - The job title/role "architect"
  - The process of architecting/designing (also: when)
  - The engineering artifact called the architecture
- Every system has an architecture
  - Identify it by looking back (avoids tangling with process & roles)
  - E.g., "Aha, I see it is a 3-tier architecture"

Every system has an architecture

Viewtype	Contents
Module	Modules, Dependencies, Layers
Runtime	Components, Connectors, Ports
Allocation	Servers, Communication channels

- **Three primary viewtypes**: Module, Runtime, Allocation
  - Many views within a viewtype

- **Architectural styles**
  - Big ball of mud
  - Client-server
  - Pipe-and-filter
  - Map-reduce
  - N-tier
  - …

Architecture as skeleton

- An animal's skeleton:
  - Provides its overall structure
  - Influences what it can do

- Examples
  - Birds fly better because of wings and light bones
  - Kangaroos jump because of leg structure
  - Convergent evolution: Bat skeletons have wings

- **Tradeoffs**
  - 4 legs faster vs. 2 legs easier to use tools

- Software skeleton examples
  - 3-tier: localize changes, concurrent transactions
  - Cooperating processes: isolate faults
  - Peer-to-peer: no central point of failure

Architecture influences quality attributes

- **Function: Carrying apples to market**
  - Option 1: Use humans
  - Option 2: Use horses
- Both options work
  - Yet options differ in their quality attributes
- **Quality attributes**
  - A.k.a. extra-functional requirements, the "ities"
  - E.g., latency, modifiability, usability, testability
- Architecture influences the system's quality attributes
  - E.g., pipe and filter is reconfigurable
  - E.g., map-reduce is scalable

Architecture orthogonal to functionality

- Architecture orthogonal to functionality
  - Can mix-and-match architecture and functionality
  - Can (mostly) build any system with any architecture
- Shift in thinking
  - Old: good or bad architectures
  - New: suitable or unsuitable to supporting the functions
- Architecture can help functionality
  - E.g., use horses to **transport** apples
- Architecture can hurt functionality
  - E.g., use horses to **stack** apples

Talk outline

- Architecture essentials
- **When do we design the architecture?**
- How much architecture should we do?
- Risk-centric model
- Risks
- Techniques
- Processes & related work
- Conclusion

Q: When do we design the architecture?

- **Recall these three distinct ideas:**
  - The job title/role "architect"
  - The process of architecting/designing (also: when)
  - The engineering artifact called the architecture
- At some point, the architecture (the artifact) exists
- Question: **When did we decide on it?** (our process)
- To understand, worth considering two strawmen caricatures:
  - Creation-centric developer
  - Jaded and careful developer

Strawmen caricatures of developers

Creation-centric developers

- Revel in our human ability to create
- Their thoughts are on how to create the next bit of functionality

Jaded and careful developers

- Worry that our creations often fail
- Their thoughts are on how to prevent the next failure or ensure success

Each of us is a mixture of these two strawmen
- Parts of us delight at what lines of code can create
- Parts of us worry how our creations will fail

Evolutionary design vs. planned design

Evolutionary design

- Fundamental beliefs
  - Hard to correctly decide early in project
  - Refactoring reduces cost of change
- So: Defer decisions when possible

Planned design

- Fundamental beliefs
  - Possible to paint yourself into a corner
  - Cost of change is high
- So: Decide now to reduce costs

Shared ground
- Cost of change is never zero
- Every project has some evolutionary, some planned design

Engineering failures

> The concept of failure is central to the design process, and it is by thinking in terms of obviating failure that successful designs are achieved.... Although often an implicit and tacit part of the methodology of design, failure considerations and proactive failure analysis are essential for achieving success. And it is precisely when such considerations and analyses are incorrect or incomplete that design errors are introduced and actual failures occur. [Henry Petroski, *Design Paradigms,* 1994]

Required

- Considering failures
- Analyzing options
- Designing a solution

You can choose

- When design happens
- Which analyses
- Formality / precision
- Depth

Talk outline

- Architecture essentials
- When do we design the architecture?
- **How much architecture should we do?**
- Risk-centric model
- Risks
- Techniques
- Processes & related work
- Conclusion

How much architecture?

- Situation:
  - Lots of architecture techniques
  - All cost something
  - Cannot afford to do them all
- Problem
  - **Which techniques to use** and **when to stop?**

- Answer 1: **Yardstick**
  - E.g., spend 10% of your time designing
  - Not so helpful on Wednesday
- Answer 2: **Documentation package** (i.e., full design)
  - Expensive
  - Overkill for most projects
- Answer 3: **Ad hoc** compromise

Desired: A principled way to decide how much is enough

Inspiration: Dad vs. mailbox

- My Dad
  - Mechanical engineer
  - Capable of analyzing stresses and strains
- The problem
  - Install new mailbox
- His solution
  - Dig hole
  - Fill with concrete
  - Stick in post
- **Q: Why no mechanical engineering analyses?**
- A: Risk
  - He just wasn't worried enough

Insight #1: Prioritize using risks

- At any given moment, you have worries and non-worries
  - Worry: Will the server scale up?
  - Worry: Will bad guys steal customer data?
  - Response time will be easy to achieve
  - We have plenty of RAM
- Cast these worries as engineering risks
  - Focus on highest priority risks
- Good news: **prioritizing risks is easy for developers**
  - They can tell you what they are most worried about
- Bad news: you might get blindsided
  - But you cannot plan for the unexpected

Insight #2: Techniques mitigate risks

- Many architecture techniques exist
  - Protocol analysis
  - Component and connector modeling
  - Queuing theory
  - Schedulability analysis
  - Threat modeling
- Techniques are not interchangeable
  - E.g., cannot use threat modeling on latency risks
- So, must **match risks with techniques**
  - I.e., mapping from risks → techniques

Talk outline

- Architecture essentials
- When do we design the architecture?
- How much architecture should we do?
- **Risk-centric model**
- Risks
- Techniques
- Processes & related work
- Conclusion

Risk-centric architecture

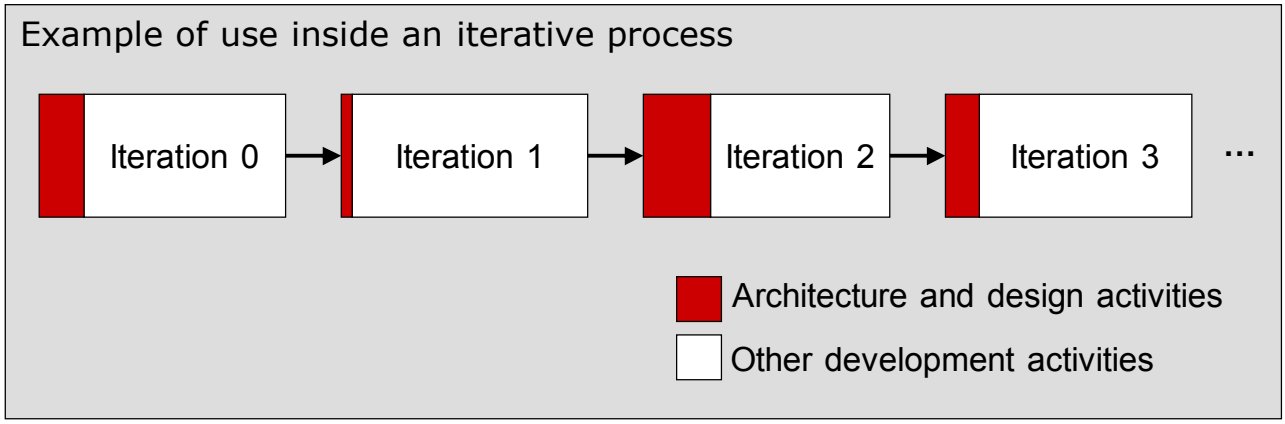
- **The risk-centric model:**
  1. Identify and prioritize risks
  2. Apply relevant architecture activities
  3. Re-evaluate

- **Promotion of risk to prominence**
  - Today, developers think about risks
    …but they think about lots of other things too
  - [Babar 09] describes team so functionality focused that quality attribute concerns deferred until development ceased and product was in maintenance mode.

- **Must balance**
  - Wasting time on low-impact techniques
  - Ignoring project-threatening risks

Risk-centric model applies to design

- Many models organize the software development lifecycle:
  - Spiral, waterfall, iterative, RUP, XP
- **Risk-centric model applies only during design**



- Q: When were (perceived) risks the highest?

Risk-centric model is uncommon

- I.e., You are probably not doing this today
- Some test questions:
- **Are risks primary?**
  - Are your risks written down?
  - Any developer can list the features
  - …but list of risks seems to be made up on the spot
- **Are techniques decided per-project?**
  - Most teams have standard set of techniques
    - Often defined by process or template
  - Do all projects have the same risks?
    - E.g., customer-facing and infrastructure

Talk outline

- Architecture essentials
- When do we design the architecture?
- How much architecture should we do?
- Risk-centric model
- **Risks**
- Techniques
- Processes & related work
- Conclusion

Definition of risk

- Simple definition

  Risk = probability of failure x impact of failure

- But, uncertainty is inherent
  - What is the probability & impact of failure?
  - Resort to educated guesses

- Revised definition

  Risk = *perceived* probability of failure x *perceived* impact of failure

- Consequences of uncertainty
  - Can perceive risks even if none exist
  - Can fail to perceive risks
  - Risk prioritization is hard and subjective

Describing risks

- Schemas for describing risks
  - Condition-Transition-Consequence [Gulch94]
  - Failure scenarios (testable)
  - Categorical (e.g., "security risk")
- **Examples**
  - "[Cr|H]ackers exist, so sensitive data may be revealed leading to reputation and financial loss."
  - "The chosen web framework may prevent us from meeting transactions-per-second requirements"

Engineering risk vs. management risk

Software engineering risks

- "I'm afraid that the server will not scale to 100 users"
- "Parsing of the response messages may not be robust"
- "It's working now but if we touch anything it may fall apart."

Project management risks

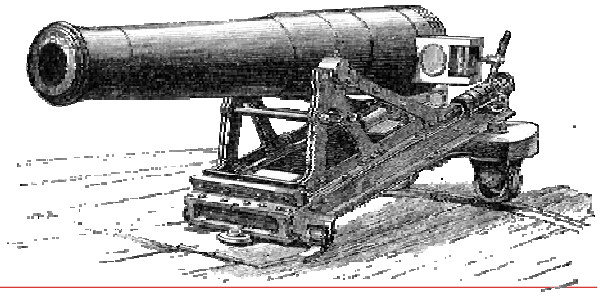
- "Lead developer hit by bus"
- "Customer needs not understood"
- "Senior VP hates our manager"
- "Competitor is first to market"

Mismatches

- Use PERT chart to eliminate buffer overruns
- Use caching to prioritize features

Canonical risks in domains

- **Information Technology (IT)**
  - Complex, poorly understood problem domain
  - Unsure we're solving the real problem
  - May choose wrong COTS software
  - Integration with existing, poorly understood software
  - Domain knowledge scattered across people
  - Modifiability
- **Systems**
  - Composition risk—will it go together?
  - Performance, reliability, size, security
  - Concurrency
- **Web**
  - Developer productivity
  - Security
  - Scalability



Talk outline

- Architecture essentials
- When do we design the architecture?
- How much architecture should we do?
- Risk-centric model
- Risks
- **Techniques**
- Processes & related work
- Conclusion

Techniques in other fields

- In mechanical engineering
  - Stress calculations, breaking point tests
- In electrical engineering
  - Circuit analysis
- In aerospace engineering
  - Thermal analysis, electrical analysis, mechanical analysis
- Prototyping: everywhere

Recall insight #2: Map techniques to risks

- This talk: How much architecture? When to stop?
- Complimentary idea: **How to perform like virtuoso?**
- Virtuosos solve problems intuitively
  - Often they can jump directly to the solution
  - Mysterious
- **Aha!: Encode virtuoso judgment**
  - Make tacit knowledge explicit
  - Accelerate learning
  - Can discuss suitability
- **If risk X, do technique Y**
  - Thinking process
  - Tabularized data

Risk → technique table

Risk	Technique
Problem domain poorly understood	• Create a Domain model (Problem frames, info model, behavior model including use cases)
Others must understand our design	• Create a context diagram • Create a use case model • Low fidelity UI prototyping (e.g., cards)
We don't know the interfaces between our components / teams	• Create Module and Component & Connector view of system • Co-locate teams • Peer-to-peer communication not mediated by managers
Problem is too big to be solved by one person or one team	• Partition into components • Encapsulate components • Design for extension. Ensure style is clear (e.g., J2EE, or WinAmp plugin API) • Document architecture (all 3 viewtypes) and invariants clearly and communicate it to teams
…	

- Just an illustrative example
- Goal: general handbooks; company-specific best practices

Talk outline

- Architecture essentials
- When do we design the architecture?
- How much architecture should we do?
- Risk-centric model
- Risks
- Techniques
- **Processes & related work**
- Conclusion

Related work

- Attribute Driven Design (ADD)
  - Tabularizes mapping from quality attributes → patterns
  - E.g., availability → ping/echo pattern
  - Bass, Clements, Kazman, *Software Architecture in Practice,* 2003
- Global Analysis
  - Identify "Factors", choose matching "Strategies"
  - Bridges engineering & management
  - Hoffmeister, Soni, Nord, *Applied Software Architecture,* 2000
- Spiral model
  - Specialization of iterative model that prioritizes risk
  - Boehm, *A Spiral Model of Software Development and Enhancement,* 1988
- Balancing risk and agility
  - Processes: use risk to choose process rigor
  - Boehm and Turner, *Balancing Agility and Discipline,* 2003

Talk outline

- Architecture essentials
- When do we design the architecture?
- How much architecture should we do?
- Risk-centric model
- Risks
- Techniques
- Processes & related work
- **Conclusion**

How much architecture?

- **How much architecture is enough?**
  - Time spent designing vs. time spent building
  - Ideal: objective, quantitative decision
- Old answers: Yardsticks & full design
- Risk-Centric Model answer
  - Do architecture until risks subside
  - But: Risks never eliminated, subjective risk estimates
- **So, does risk help?**
  - Yes: Shared vocabulary ("risks") with management
  - Yes: Refined evaluation ability
- Subjective risk estimate better than:
  - Yardstick (e.g., spend 20% of your time designing)
  - Full design

Takeaway ideas

- **3 distinct ideas**:
  - The job title/role "architect"
  - The process of architecting/designing
  - The engineering artifact called the architecture
- Every project is combination of planned & evolutionary decisions
  - You should calibrate your project's balance using risks
- Risk is a unifying concept
  - Managers understand it
  - Engineers understand it
- **Architecture is compatible with agile**
  - Agile will do more evolutionary design
  - Use risk to mix in architecture design

Conclusion

- Software architecture techniques are helpful
- We cannot afford to apply them all

- Important: **How much should we do?**
  - Old: Yardstick model
  - Old: Full description model

- This talk describes a new model: Risk-Centric Model
  1. Identify and prioritize risks
  2. Apply relevant architecture activities
  3. Re-evaluate

- It is not: Big Design Up Front
- It is not: A full software development process

- It is: Compatible with agile and other processes