SWE 621

FALL 2021

DESIGN AS RISK MINIMIZATION

IN CLASS EXERCISE

- What were the most important risks you faced in a recent software project?

LOGISTICS

- HW1 now out

WHAT IS A RISK?

WHAT IS RISK?

- risk = perceived prob of failure * perceived impact

RISK IN SOFTWARE ENGINEERING

- Software architecture about making important decisions
  - Decisions made with goals in mind.
- Does decision achieve intended goal?
  - If not, will implement system following decision, find out system does not achieve goal.
  - May then need to throw away system, change decision, build new system based on decision.
  - This is a disruptive and high impact change.

EXAMPLE: RACKSPACE ARCHITECTURE V1

- Rackspace email server
  - Has log files which record what happened, helping respond to customer queries about problems
- V1
  - Each service on each email server writes to a separate log file.
  - To answer customer inquiry, execute grep query.
- Challenges
  - As system gained users, overhead of running searches on email servers became noticeable.
  - Required engineer, rather than support tech, to perform search

EXAMPLE: RACKSPACE ARCHITECTURE V2

- Every few minutes, log data sent to central server and indexed in relational database
  - Support techs could query log data through web-based interface
- Challenges
  - Hundreds of servers constantly generating log data --> took long to run queries, load data
  - Searches became slow; could only keep 3 days of logs
  - Wildcard searches prohibited because of extra load on server
  - Server experienced random failures, was not redundant

EXAMPLE: RACKSPACE ARCHITECTURE V3

- Save log data into distributed file system (Hadoop)
  - Indexing and storage distributed across servers
  - All data redundantly stored
- Indexed 140 GB of log data / day
- Web-based search engine for support techs to get query results in seconds
- Engineers could write new types of queries, exposed to support techs through API

RISKS: RACKSPACE EXAMPLE

- Hardware failure leads to data loss
- Too much data to store on one server, must be discarded or stored differently
- Queries execute slowly
- Support techs frequently require new types of queries
- Others?

SOME TYPES OF RISKS: FAILING TO ACHIEVE QUALITY ATTRIBUTE

- Performance: how fast is the system
- Reliability: how likely is the system to be available
- Scalability: how well does adding more computing resources translate to better performance
- Maintainability: how hard is system to change
- Extensibility: in what ways can new components be added without changing existing components
- Configurability: how easily can the system behavior be changed by end-users
- Portability: in what environments can the system be used
- Testability: how easy is it to write tests of the system's behavior

TYPICAL RISKS VARY BY DOMAIN: INFORMATION TECH PROJECT

- Complex, poorly understood problem
- Unsure solving the real problem
- May choose wrong frameworks / libraries / technologies
- Integration with existing, poorly understood software
- Domain knowledge scattered across people
- Modifiability

TYPICAL RISKS VARY BY DOMAIN: SYSTEMS

- Performance
- Reliability
- Size
- Security
- Concurrency
- Composition

TYPICAL RISKS VARY BY DOMAIN: WEB

- Security
- Framework / library / technology choice
- Application scalability
- Developer productivity / abstractions

RISK-DRIVEN ANALYSIS

- Identify and prioritize risks
- Select and apply a set of techniques
- Evaluate risk reduction

WHY DO RISK ANALYSIS?

- Don't want project to fail.
  - Identify and address potential reasons for failure
- Not only about implementing features, but ensuring that way they are implemented meets goals
  - Not meeting these goals is a risk

WHY DO RISK ANALYSIS: IDENTIFYING DECISIONS

- Design isn't simply selecting from alternatives, but realizing their existence
  - May identify decisions you did not think of as decisions.
- Example: Rackspace
  - In v1 data stored as log files.
  - This was how the data was already being logged. No "decision" to choose how to implement something: it's already implemented.
  - But by keeping this choice, constrained how the rest of the system was built.
  - Looking for risks might have uncovered this as a decision.

IDENTIFYING ARCHITECTURAL DECISIONS

- Quality attribute --> technique used to achieve quality attribute --> DECISION!
- Whenever you have an important quality attribute to achieve, you have at least one architectural decision, probably several
  - Ask what alternative approaches might achieve this
  - Why did you choose the one you did

WHAT'S AN ARCHITECTURAL RISK?

- Where does architecture end and design begin?
- Architectural decisions are those that require substantial development effort to change
- It's why it's risky: if not right, would be high impact to change it
- cost of change, architectural, high-level design, low level design, implementation

GROUP ACTIVITY: TERMINOLOGY REVIEW

- Build design space for Rackspace
- Label figure with following terms
  - design dimension
  - design alternative
  - design decision
  - architectural driver
  - risk
  - design rationale

GETTING AROUND A DESIGN SPACE

- Focusing on risks may identify design decisions, create additional design dimensions and alternatives
- As understanding of problem deepens and relationships between different decisions become clearer, may lead to clearer way to restate what the real choices are
- Many levels of abstraction to talk about decisions
- Some decisions may be closely tied to other decisions (we'll come back to this later in the course)

TECHNIQUES TO REDUCE RISKS

RACKSPACE

- Rackspace risks:
  - Hardware failure leads to data loss
  - Too much data to store on one server, must be discarded or stored differently
  - Queries execute slowly
  - Support techs frequently require new types of queries

- What might developers have done to reduce these risks up front before implementing the whole system?

TECHNIQUES TO REDUCE RISKS

- Analytical: build a model, analyze its properties
  - Model how a system that follows decisions would work
  - Simulate its behavior in various situations
- Empirical: build a prototype, test it
  - Don't need to invest in building out every feature
  - Focus on high risk part, build it, check if it works as required

PROTOTYPING

- Risks
  - We expect system to achieve this property. Will it in practice?
- Prototyping
  - Build the simplest possible example system.
    - Might simulate some with dummy data, simulate component failure
  - Measure: does it achieve property?

DOMAIN MODELLING

- Risks
  - Complex, poorly understood problem
  - Unsure solving the real problem
- Domain modeling (next lecture)
  - What entities are there?
  - How are they related?
  - How do these behave in different scenarios?

PERFORMANCE MODELLING

- Risks
  - Performance: does my system meet performance goal?s
  - Scalability: does adding more servers to my system increase its throughput?
  - Robustness:
    - what happens when a server crashes?
    - can the order of requests sometimes cause bad output data?
- Systems modeling
  - Take a distributed systems class (CS 675)

MAINTAINABILITY

- Risks
  - Maintainability: how much developer time does it take to add a new feature?
- Design abstractions (future lectures!)
  - Find core abstractions that enable common new features to be built easily on top of existing code
- Design for change (future lectures!)
  - Identify decisions likely to change, hide them in modules

HOW MUCH EFFORT TO SPEND REDUCING RISKS?

DISCUSSION: HOW MUCH EFFORT TO SPEND REDUCING RISKS?

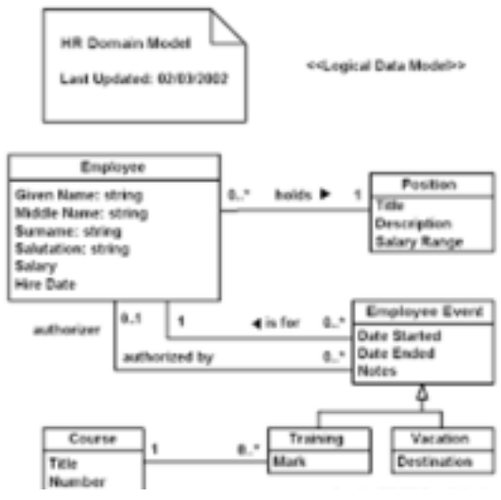
- Nearly infinite number of risks
  - Do you use techniques to reduce all of them?
- When do you stop designing your system and start implementing it?

BIG DESIGN UP FRONT

"Many times, thinking things out in advance saved us serious development headaches later on. ... [on making a particular specification change] ... Making this change in the spec took an hour or two. If we had made this change in code, it would have added weeks to the schedule. I can't tell you how strongly I believe in Big Design Up Front, which the proponents of Extreme Programming consider anathema. I have consistently saved time and made better products by using BDUF and I'm proud to use it, no matter what the XP fanatics claim. They're just wrong on this point and I can't be any clearer than that."

-- Joel Spolsky, co-creator of Stack Overflow

- Rather than commit time implementing a potentially risky design, use modeling techniques to reduce risks before implementing
- Can build models of many aspects of your system (big design) before you implement it (up front)
- Rather than waste time implementing three versions of Rackspace, figure out the right architectural decisions up front



RISKS OF BIG DESIGN UP FRONT

- May make false assumptions
  - e.g., component x will be able to perform y
  - may not be possible in practice
- May over design
  - Complex design that is time consuming and challenging to build
- May not identify the right risks to focus on
- Requirements may be nebulous or change

AGILE SOFTWARE DEVELOPMENT

- Just implement it
- Don't worry about up front modeling
- Rather than making every important architectural decision up front, architectural decisions are made, and changed, constantly based on latest feature being added
- Code should be updated to be consistent with decisions as they change

http://agilemanifesto.org/

RISK-DRIVEN SOFTWARE ARCHITECTURE

- No right answer, depends on context
  - Stop doing upfront modeling and architecture when risks can be better addressed by building something.
- Identify risks. For each risk, consider
  - Is it better to tackle empirically or analytically? Which will take more time? What assumptions will each approach require?
- May combine
  - Build small prototypes for some risks (empirical), models for other risks (analytical)

REVERSE ENGINEERING

- In HW1, you will be reverse engineering architectural decisions and drivers.

- What are some techniques for identifying architectural decisions?

READ THE DESIGN DOCUMENTS

- Wiki pages, guides for new developers, guide for contributors
  - But make sure you're looking at documents for how to edit the code rather than an intro to the API for reusing the code
- Look for decisions, not just statements about how the system works.
  - If the document talks about how something works, does it mention why? Can you infer why?

READ THE ISSUES ON ISSUE TRACKER

- Many modern projects have public issue tracker where defects are reported and fixes proposed and discussed
- Individual bug fixes unlikely to reveal architectural decisions.
- May see architectural decisions indirectly in discussions of why bugs were fixed in a specific way. When multiple alternatives were discussed or an initial fix rejected, what considerations drove these choices?

READ RELEASE NOTES

- Descriptions of new features and changes when new version of software released
- Most interesting: discussion of larger changes
  - e.g., In this version, we changed how data is stored by introducing a cache in each individual server node. This enables lower latency in many typical real world requests. Introducing a cache required making several important changes to the logic for when and how data is sent between server nodes.

READ THE CODE

- Ground truth about how system works
- Challenge: understanding rationale **why** code was written as it was written
  - May see patterns in code.
    - Can you understand why these patterns are there? What decision led to this, and why was it made?
- Given an architectural driver discussed elsewhere, may be able to identify how it is achieved by reading code
  - e.g., Important that all requests be serviced in less than 800 ms to keep users happy.

GETTING HELP

- If you're still struggling with how to get started, drop by instructor or TA office hours.
  - We won't be experts in your systems, but can offer advice on reverse engineering strategies.

IN CLASS ACTIVITY

DESIGN ACTIVITY: COGNITIVE CHAT-BOT USER ASSISTANCE

- The CEO of your company just decided that your company needs to embrace the future and offer users a way to interact with your system through an AI-based chat system.
- Your team was just assigned to build this, but doesn't yet know anything about AI-based chat systems.
- Deliverable: List a set of risks. For each risk
  - Describe the risk. What might fail?
  - Estimate perceived prob of failure (low / med / high) and perceived impact (low / med / high).
  - Describe a technique and plan for reducing this risk.

DESIGN ACTIVITY: STEP 2

- Swap groups.
- Compare the risks you found.
- What assumptions led to the differences in risks? How did your groups differ in what you believed to be hard or easy?

DESIGN ACTIVITY: DISCUSSION

- What did you learn about the practice of design from this activity?

RISKS