Large Scale Software Engineering

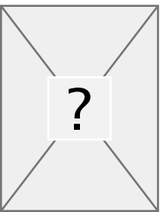
Week 04

Mike Helmick
University of Cincinnati
CS6028
Spring 2014

Chapter 6, 7

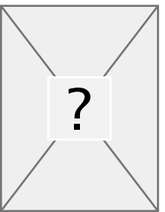
- Moving on to chapter 6, 7

- Also, part II of the book: Architecture Modeling



Models

- Our book has talked a lot about “models”
  - Hasn’t touched on what they actually are
  - How to build them



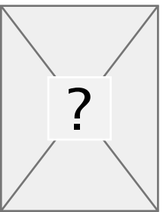
On Calculus…

- In chapter 6 the author shares a story of his father not being able to help with his high school calculus homework

- I feel the same way about calculus

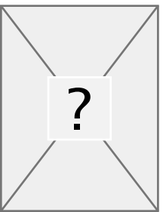
  - It helped me learn to solve problems with abstraction and modeling

  - It isn't something that I have really used on the job



Engineers Use Models

- When complex problems arise
  - Engineers map that problem onto an abstract model
  - Solve the problem within the model
  - Translate the model back into a solution
- Software engineers walk the line between abstract models and real world solutions
  - This is one of the things I like about this field

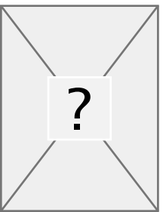


Abstraction

- I define Computer Science as “the study of abstraction”

Scale and Abstraction

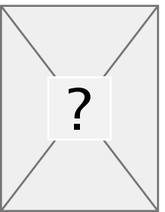
- As the number of classes in a system grow
    - We use more and more abstraction to keep things under control
- The abstractions morph into the one needed to support larger scale or the added complexity



Learning

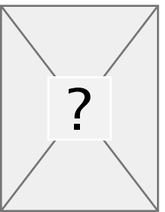
- Abstractions can make it more efficient to learn a new system

- Sketching a model of the system on the whiteboard is often the initial introduction for new developers on a project



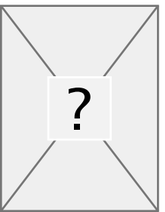
Benefit of Abstractions

- Learning to model software as a set of abstractions helps us to reason about and solve problems
- We learn patterns
- We make leaps and gain the ability see common solutions



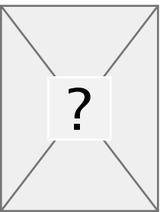
System Qualities

- The story that starts section 6.3 is quite interesting
  - Using framework X, inability to make framework X run quickly
  - Switched to framework Y, improved interfaces, improved extensibility, and demonstrated higher throughput
  - But why?
    - hierarchical storage under framework X
    - flat storage under framework Y
    - 20 DB queries for framework X vs 1 for framework Y

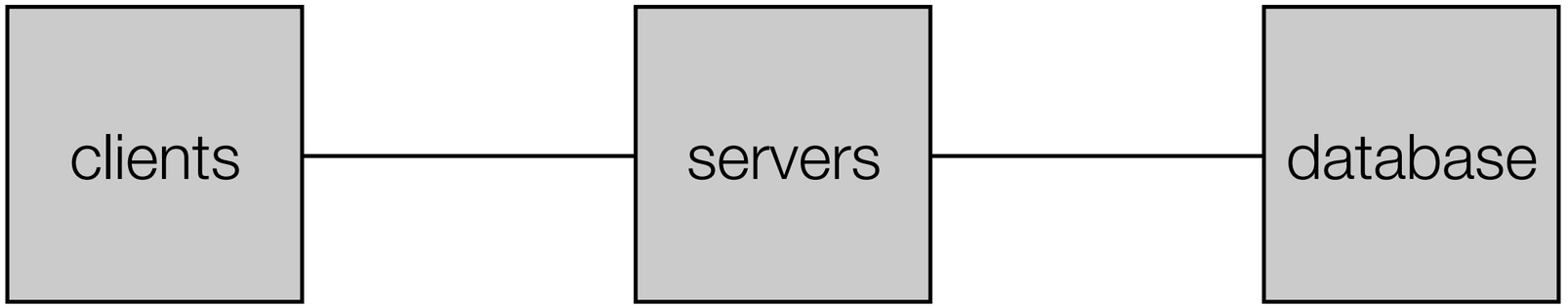


System qualities

- For reasoning about system qualities, we should have a model that works equally well for technologies X and Y
- Complete enough to analysis details that are known



Example

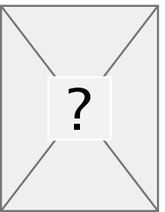


Really simplified model

Putting some basic facts in, can help us make decisions

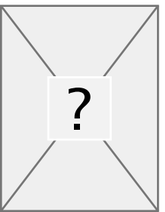
10 ms transport time, 10ms server processing

25 ms database processing time per query



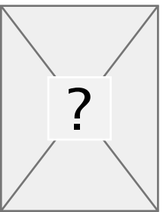
Identifying the issue

- If you are using something like technology X that makes multiple database calls on a single quarry
  - Even this model might not be sufficient to identify the details
  - But the necessary components are there



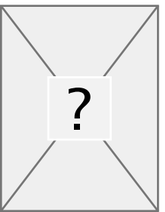
Models Omit Details

- Some details aren't necessary at a all
    - Too many details, and it will be impossible to reason about your model
- Omitting to many details will make it difficult to reason about your model



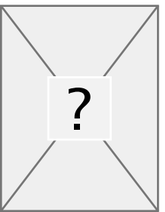
Modeling Skill Levels

- Reading models
- Writing models
- Amplifying reasoning with models



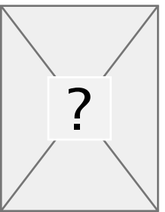
Levels

- Reading a model is the most common
  - You don’t need to be able to build a model of the system to be able read one
- Writing a model and amplifying reasoning
  - The model allows one to design systems more complex than if everything had to be done in memory
  - The model allows you to identify issues and make corrections, etc



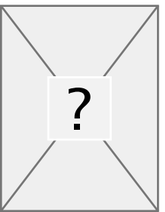
Different Models

- Different kinds of models have their places
  - a model for predicting latencies will not aid in security (threat modeling)
- Know what question you are trying to answer with the model



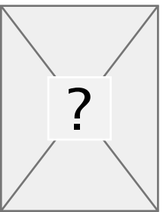
Engineers Use Models

- Engineers use models to help solve complex problems
- Models help you organize what you know and reason about it
- From models, we can shift back to real world solutions



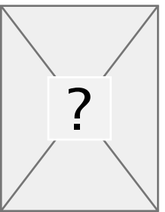
Chapter 7

- Conceptual Model of Software Architecture



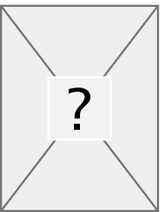
Models vs Process

- Using models is not the same as choosing your software process



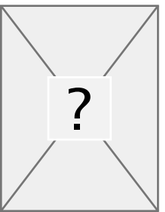
Common misunderstandings

- Every project should document its architecture
- Architecture documents should be comprehensive
- Design should always precede coding
    - All of these are false



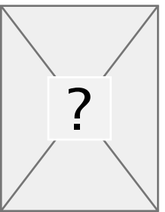
Architecture Models

- Using software architecture models help you to see what's coming next
- To be able to make broader decisions



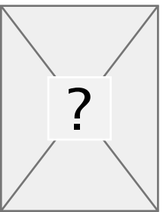
Models Accelerate Progress

- Conceptual models accelerate progress
- You may not remember the exact details
    - But the conceptual models will accelerate your abilities in the future



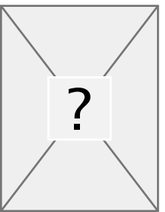
Domain Model

- Described the truths about the domain
    - Financial system: Attributes of accounts, laws
    - Largely not under your control



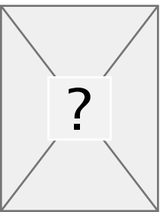
Design Model

- The domain model doesn’t define your system
- The design model does
  - The design model is not complete
  - Component model / boundary model



Code Model

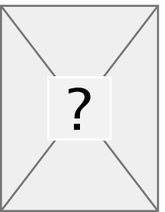
- Low level - may actually be the implementation



Designation

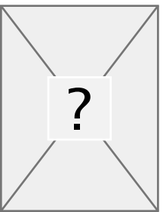
- Designation relationship between models “enables you to say similar things in different models should correspond”

- Using designation, you can see how the truths in the domain model are represented in the design model



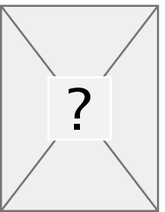
Refinement

- Relationship between low-detail and high-detail model of the same thing



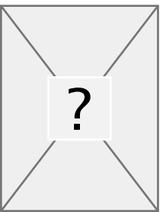
Views

- A view, or projection
  - Shows a subset of a model's details
  - A model may, and probably will, have several different views



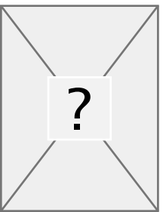
View consistency

- Each view shows a single perspective of the model it is part of
- These live in isolation
  - Yet, they are intertwined
  - Keeping these models up to date is a difficult task



Master Model

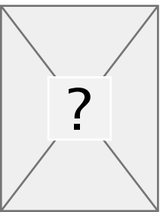
- Everything in the domain model, design model, and code model
- Taken to the logical conclusion
  - There could be a programming environment that keeps all of this in sync
  - *This would be overkill*



UNIVERSITY OF Cincinnati

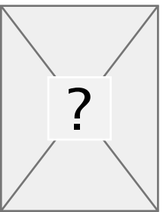
The Domain Model

Chapter 8



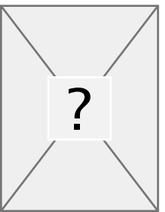
Domain Model

- “Enduring truths about a domain”
- This is the world within which your system operates
- It will embody constraints
- Can help you gain insights that will be helpful when moving down to the next level



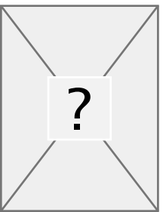
Usefulness of Domain Model

- Depends on the complexity of the domain



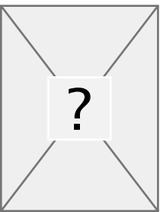
Objections to using a domain model

- You already know your domain
- The domain is too simple to bother modeling it
- The domain is irrelevant to your architecture choices
- It is someone else's job to do requirements
- The base way to learn the domain is incrementally, as you write code
- Domain modeling is an open-ended analysis paralysis activity



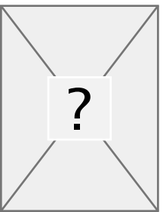
You already know your domain

- This depends on the domain
- There is a very good chance that the business stakeholders know more about the domain than you
- This is why requirements are important



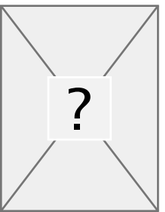
The domain is too simple

- In the context of large scale software
- There is likely not a domain so simple that is also worth scaling.
  - but Twitter is simple…
  - No, the domain is actually quite complex, and would probably need some domain modeling to reason about it



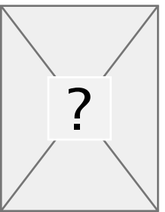
Domain is irrelevant to your architecture

- Yes domain influences architecture
- At a minimum, there will be certain quality attributes that may not be achievable in certain architecture
- However
  - It may be true that the domain does not dictate the architecture



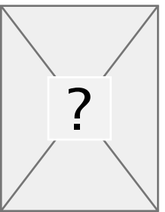
Someone else's job to do requirements

- It is dangerous to not have engineering involved in requirements
- Otherwise, you could end up with requirements that no amount of software architecture knowledge can help with



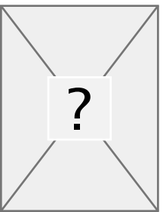
Learn domain incrementally

- Sure, you can’t learn the entire domain up front
- But, you can make an effort to learn what is most important (i.e. what carries the most risk if you get it wrong



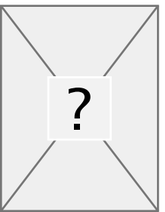
Domain modeling is analysis paralysis

- Do just enough
- That advice in itself is not completely sufficient
  - You're looking to capture major components
  - User interactions with the system
  - Interactions *between* users



All the things…

- Describing the basic entities and their interactions / relationships is the essential information of the domain model
  - You want this to be more than just in one person's head
    - Good to write it down
  - Invariants
    - The domain model should capture the limits of your system (as they can help influence architecture).
    - i.e. our financial system supports no more than 10 accounts per person, with a limit of 100 transactions per day, and we will keep 10 years of transaction history.
      - From this, I can make reasonable design choices



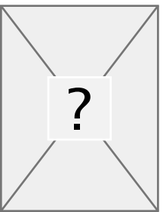
UNIVERSITY OF Cincinnati

The Design Model

Chapter 9

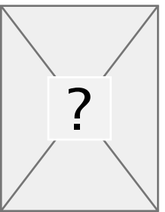
Design Model

- Or just “the design”
  - This is where you have control
- Can’t be complete or comprehensive
  - It would take forever to build, and be difficult to use



Tools

- Views
  - A project of a model that reveals selected details
- Encapsulation
  - Separation of the interface model from the *internals model*
- Nesting
  - Allows for several layers of boundary and internal models

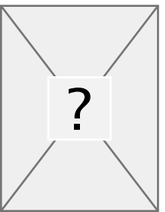


Practical?

- Is this level of modeling practical?
    - Probably not. I can’t envision a fast moving, large scale system that would have time to support this type of modeling and still be successful.

Boundary Model

- What outsiders can see of the system
- Interface of the components of your system
  - Says nothing about the implementation of those components



Internals Model

- A refinement of the boundary model
- This is the one that most software engineers care about
  - It describes the design of the components that are actually being built