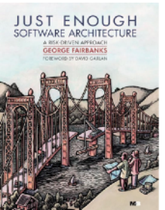
Large Scale Software Engineering

Week 03

Mike Helmick
University of Cincinnati
CS6028
Spring 2014

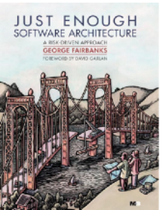
Architecture Modeling

- On to chapters 4 and 5 this week. Chapter 6 next week



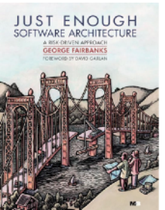
Chapter 4

- An example architecture for a Home Media Player
- Using a the risk-driven approach to software architecture
- Asks the question “What are my risks?” over and over again



Home Media Player

- Computer that plays media: music, videos
- Normal computer with audio and video output, connected to TV, maybe an A/V receiver
- Plays media from local disk and from the internet
- Simultaneous picture / music playback
- Simultaneous video playback + view information about the view
- Ability for third party extensions to be built



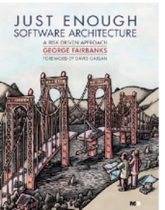
Chapter Flow

- A team that has already built a prototype of the home media player, and these issues have been found

1. Team communication: New developers have been added are a remote site. The existing team worries that new developers might not understand the design and architecture.

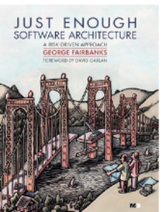
2. Integration of COTS components: Currently only runs on a single platform. Team asked to integrate some COTS components. This always brings along risks.

3. Metadata consistency: Worried that internal metadata representation will become incompatible with that of media found on the internet.



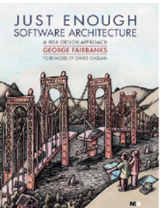
Team Communication

- Background
  - Small team, co-located, working long hours
  - All of the developers worked on design and know the architecture
    - This tells us that it is a really small team
    - Design isn't written down (known only by the team)
  - Existing team is worried about integrating new team members and a rapid push to turn a prototype into a launched product
  - They recall Brooks' advice about adding developers to a project (The Mythical Man-Month)



Addressing Team Communication Risk

- Reduce the risk by communicating the design to the new developers
- 3 primary models: domain, design, and code models
- 3 primary architectural view types: the module, runtime, and allocation views
- They start with the least expensive techniques and select more expensive techniques until they believe the risk has subsided

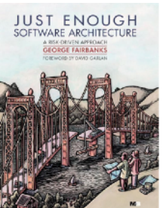


Alternative Approach

- What is not mentioned in the book is team structure
- In the prototype, it seems that all developers treated as equal, and everyone working on any part of the system
  - And - that the new developers are being asked to function the same way
- What is not considered in the text
  - Organizing the team in a different way, forming sub-teams, forming areas of specialization

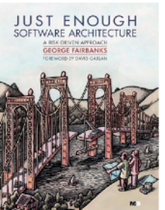
Team Communication: Read the source code

- Easy, no cost for existing team, nothing needs to be created
- New team members can study
- Design decisions are *not* captured in the code
- Purely reading code will likely be an ineffective use of time for new developers
  - Lengthens ramp up time, reduces their productivity
- The team decides that additional techniques are necessary



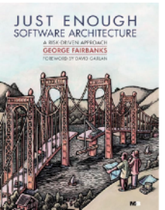
Module model

- Module model identifies the high level components and their dependencies
- The labels in the module model largely identify the logical structure of the application
  - However, it doesn't match they layout of the source code on the filesystem
- The team feels that the team communication risk hasn't been completely mitigated
  - However, the module model was probably very easy to create
- More needs to be done to convey design decisions / architecture decisions



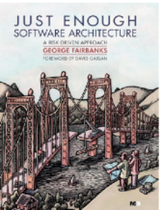
Quality Attributes and Design Decisions

- The team prioritized quality attributes based on experience with existing products in the area
    - UI responsiveness, smooth playback, etc…



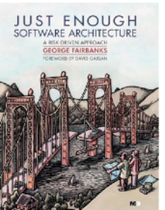
Tradeoffs

- Two Tradeoffs
  - Portability and smooth playback
    - Portability is achieved by hardware abstraction layers
    - Platform specific APIs can lead to smoother playback
  - Playback efficiency and modifiability
    - Chose an architecture that allowed the plug-in capability for new codecs and video sources
    - Instead of highly tweaking and specializing codecs
- These tradeoffs were never written down, but influenced everything the team did



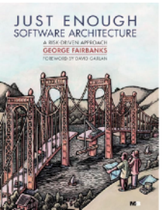
Architecture Drivers

- Latency
  - When the user presses a button, the command should happen within 50ms
  - When 50ms cannot be met, the system should provide feedback (buffering…)
- Reference video should play smoothly, form local disk, on reference hardware
  - *Note: this is not an architecture decision, it is an architecture driver.*



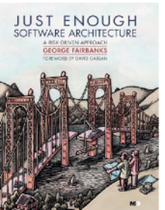
Design Decisions

- **Process isolation**: Each top level component will run as a separate process
- **Shared memory**: Playback component communications with media buffer through shared memory
- **Buffering**: To ensure all playback, all data sources are buffered in RAM
- **Metadata Repository**: All content has metadata stored in the metadata repository
- **Private Metadata Repository**: Only the media player core component can write to the metadata store
    - Risk addressed, plugins corrupting metadata store



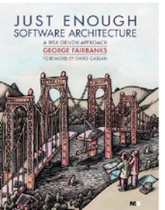
Runtime Models

- How the components interact at runtime
- Given as general guidance as to where new functionality should fit



Integration of COTS components

- The team is asked to expand the software to additional platforms
  - Cross Platform AV component for playback
  - Asked to use a specific product for rendering, “NextGenVideo” in this case
    - Product is better performing, but less robust (by reputation)



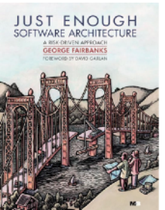
COTS integration failure risks

- **Integration** - Will the new components fit into the architecture? The team doesn't have knowledge of the new opponents, and don't know in advance if this will work

- **Reliability** - Since one of the components has a reputation for crashing, isolation is necessary. The source code is not available.

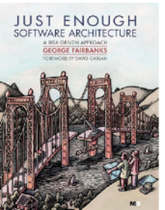
- **On-Screen Display** - The old video component handled drawing the UI and the playback. The new one just does playback

- **Latency** - This is a core architecture driver, and the primary components that can affect this are changing.



Integrating the COTS

- Involves lots of research, reading documentation
- Almost like building another prototype
- Lots of time spent on detecting crashes, and restarting the playback component

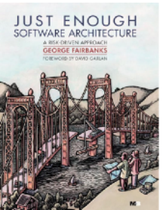


Success?

- The description of the integration of these new components makes it sounds like it went really well
- We should be suspicious of this, especially in the context of the team growing at the same time

Metadata consistency

- Good design decision
    - Enables consistency
    - Enables fast search (something not really mentioned)
- Constructed a domain model to test usability by plug-in developers
- In researching alternatives, they made some changes to the data model
- **Importance**: Any API that you release to the public instantly becomes hard to change.



The Mythical Man-Month

- Software projects go wrong when you consider calendar time. Always has been, maybe always will?

- Why?

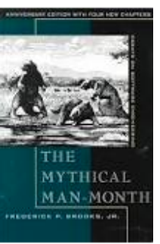
    1. Estimating techniques are poorly developed (true in 1975, still has some truth today)

    2. Estimating techniques confuse effort with progress. Hides the assumptions that people and months are interchangeable

    3. Because estimates are uncertain, managers lack “courteous stubbornness.” *Good cooking takes time. If you are made to wait, it is to serve you better, and to please you.*

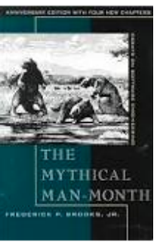
    4. Schedule progress is poorly monitored

    5. When schedule slippage is recognized, an easy response is to add engineers. Brooks likens this to “dousing a fire with gasoline”



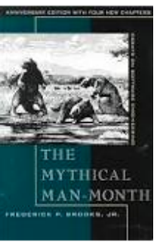
Optimism

- “All programmers are optimists”
- We’ve all said things like
  - “This time it will surely run”
  - “I just found the last bug”
- This leads to a false assumption that underlies all software estimates
  - **Everything will go well**
  - So, we give our estimates in terms of how long something *should* take



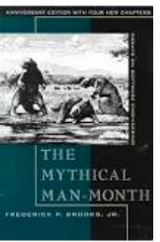
Why are programmers optimists?

- This does seem to be largely true.
    - There are exceptions. There are pessimistic programmers out there.
- This may be due to the fact that this is a creative endeavor
    - We work in a completely open and capable medium
    - We think anything is possible (possibly rightly so)
    - But - our ideas are faulty, this leads to bugs, and makes the optimism unjustified



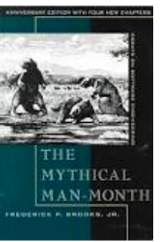
Going Well…

- The probability that any single task will go well isn't too bad
- Projects are made up of a huge number of tasks
  - The probability that all the tasks will go well gets worse as you add tasks



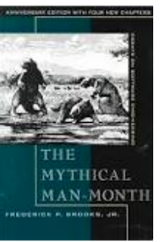
The Person-Month

- Cost is a function of the number of people and time (months)
- Progress is *not* a function of the same inputs
- The person-month as a unit for measuring the size of a job is incorrect, and dangerous for measuring success



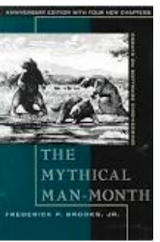
Interchangeability

- People and months are interchangeable only when there is no communication between the people
- Any amount of communication will slow people down
- There are tasks / jobs in the world where this is the case
  - Programming is not one of them
- Some tasks cannot be partitioned



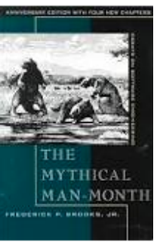
-or-

- Nine women cannot have a baby in one month
- Some tasks simply can’t be partitioned



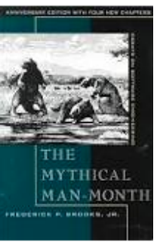
Tasks that can be partitioned

- Automatically incur communication overhead
- The communication time must be factored into the amount of work done
- A 1 month task that can be perfectly partitioned
  - Will take a single person, 1 month
  - Will take two people > 1/2 a month, but probably < 1 month



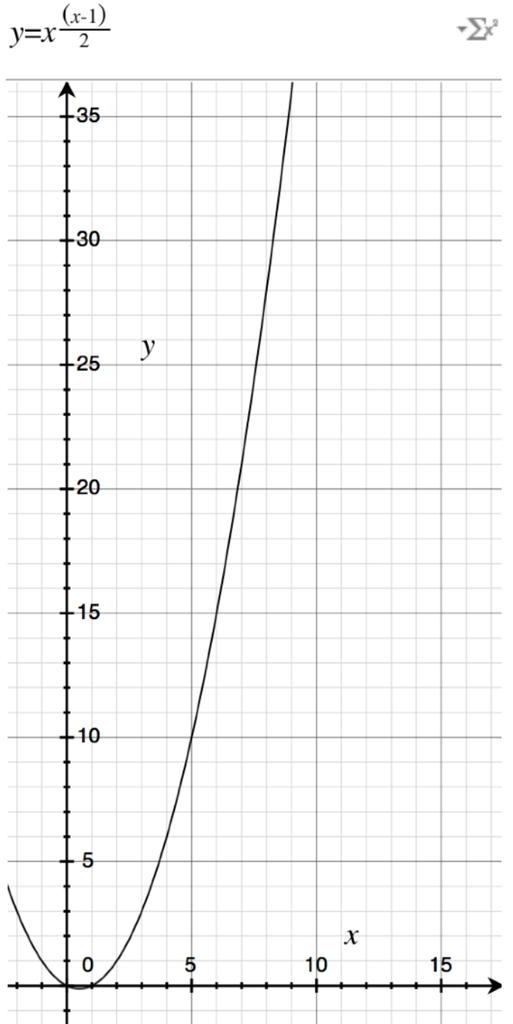
Communication

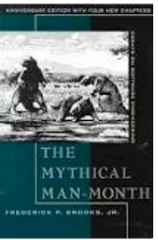
- Training (assuming new people are added to the project to increase the number of people)
- Intercommunication
  - Formula given: $\frac{n(n-1)}{2}$
  - This is the pairwise communication effort on a project
  - Additional team meetings, add overhead in excess of this



Communication

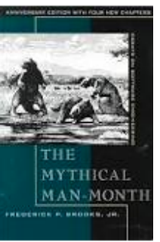
- Training (assuming new people are added to the project to increase the number of people)
- Intercommunication
  - Formula given: $\frac{n(n-1)}{2}$
  - This is the pairwise communication effort on a project
  - Additional team meetings, add overhead in excess of this





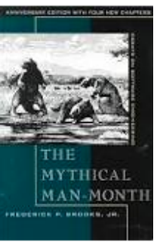
Testing

- Very difficult to estimate test time
  - Ideally this is zero, right? No bugs
  - In reality, this is always non-zero, and usually much more time consuming than anticipated



Estimating

- A schedule can only be that, a scheduled completion date
  - It cannot set the actual completion date
  - However, scheduling to meet a customer's requested date is common
- Proposed solutions
  - Develop and publish productivity figures (this hasn't really happened)
  - Managers should defend their estimates



Architecture Modeling - Another Example

- A more large scale example (since that is the theme of this class)
- It is perfectly reasonable that a small team could build site like twitter very quickly
    - 2 people could probably do this in a weekend

Prototype to Product

- We can envision the same type of scenario as the home media player
    - except, maybe this was launched, live on the internet
    - Initially just friends and family join, but then they get additional people to sign up
- Now, the team is under pressure to scale

Issues to Address

- Growing the team / team communication
- Scaling number of users
- Scaling number of updates / tweets

Growing the Team

- Not much different from what is described in Chapter 4
- This would probably be a very similar situation
  - Not much documented
  - Existing small team knows the whole design / architecture, but it is in their heads only

Tradeoffs

- The last to concerns, scaling the number off users and scaling the throughput of the system are related, but are distinct

- Tradeoffs we might consider

  - **Consistency of updates**: We might design a system where all updates are globally sequenced so that if user A tweets before user B, users C and D will always see the updates in that order. An alternative would be that if user A tweets before user B, user C and D could see those tweets in a different order. *Let's assume the latter.*

  - **Controlled growth vs open signups**: Making a choice in either of these will impact priorities in the short term. Choosing open signups makes scaling a less immediate concern, but

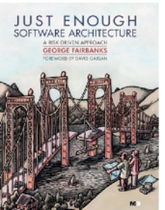
  - **Tight social graph vs large number of followers**: Related to consistency of updates. Should we design a system that allows for limited number of followers or large numbers of followers. The largest number of followers on twitter is > 49,700,000.

Architecture Drivers

1. Tweets should be visible by all followers within 15 seconds of a tweet being saved.
    - This may seem like a long time, but see number 2
2. Users should be able to have extremely large numbers of followers, 10s of millions of followers.
    - This decision will drive other things, like the tradeoff on tweet ordering, and the delivery time (# 1)
3. Signups will initially be invite only at first, and then proceed to open signups when the team feels confident it can be handled.
    - This is a risk mitigation. Allowing open signups on an untested architecture can lead to disiaster.

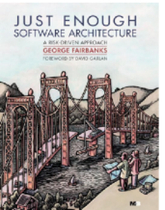
Chapter 5

- Last chapter of the first part of the book
- This means that we will get to more interesting items soon



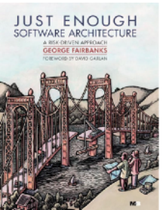
Understand your Architecture

- The author compares experienced software architects with coaches of sports team
- They are able to see things at a higher level than rookie player
  - More importantly, they are able to see this links between things / chain reactions, etc
- This is the thing that no amount of textbook knowledge will provide
  - Experience matters in software architecture
  - Seeing how systems are built, evolve, fail, recover, etc… will



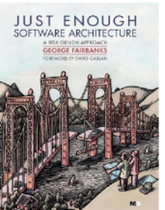
Distribute Architecture Skills

- However, a successful sports team will be one where it is more than just the coach that understands the game at a higher level
- Your software project will be more effective if it is more than just one or two high level people that understand the project at an architecture level
- If a large portion of the team has the necessary architecture skills, and you employ a risk-driven approach, it seems to follow that overall, the project will move forward more quickly



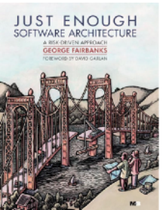
Make rational architecture choices

- Every decision you make will involve a tradeoff (even if you don't explore what that tradeoff is)
  - You should make rational choices
  - i.e. your tradeoffs should align with mitigating your risks and ensuring that your quality attributes are achieved
- <x> is a priority, so we chose design <y>, and accepted downside <z>.



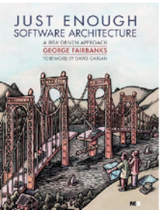
Who makes irrational architecture choices?

- People make mistakes
- If not everything is known, a decision may seem rational until more information appears
  - This is where agile processes, combined with the risk-driven approach can help
- Documenting the decision making process, particularly on large decisions, and help avoid team conflict as well



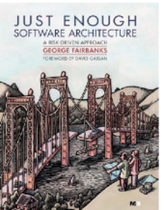
Remaining Challenges

- **Estimating Risks** - risk can help you model, decide how much design to do, decide when to start implementing
  - Risk identification
    - Can be difficult to do
    - unforeseen risks
    - No excuse for missing risks that you've seen on previous projects (or especially, earlier in this project)
  - Risk prioritization
    - Identified risks must be ranked against other risks (priorities)
    - Guess to high, you miss something important. Guess too low, you miss what may be the most important



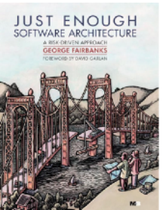
Evaluating Alternatives

- It can be easy to build a simple model of an alternative architecture
    - And I think many would argue that you should do just that
- Details will be missed
    - Some details are only apparent once implementation has begun



Reusing Models

- This is the power of design patterns (we're getting there)
- Models omit details
  - This is the reason they're difficult to use for evaluating alternatives
  - Just like they are difficult to reuse



Issues…

- Management / business stakeholders probably aren't interested in the details of your architecture
- However, some management decisions will impact your architecture
    - Following form the examples in the book - how the support team is organized can impact your thinking here