SWE 621

FALL 2018

COURSE OVERVIEW, DESIGN AS CHOICE

IN CLASS EXERCISE

- As you come in and take a seat

- What is software architecture?
- What is software design?

- Write down a description of what these terms mean to you

WHAT IS SOFTWARE ARCHITECTURE?

WHAT IS SOFTWARE DESIGN?

WHY STUDY SOFTWARE ARCHITECTURE AND DESIGN?

WHY DOES SOFTWARE HAVE TO BE REWRITTEN?

- Team spent 1 year building v1, decided to throw it away and build v2.
  - What happened?
- What **risks** cause software to need to be rewritten to meet its requirements?
- What activities can help **reduce** these risks?
- How much, or how little, design should you do **before** building your system?

HOW CAN YOU UNDERSTAND YOUR SOFTWARE?

- Your codebase is 900,000 lines of code.
  - You haven't read every one of the 900,000 lines (and your teammate might have changed them anyway).
  - You need to add a new feature. How can you build on top of what's already there? How do you understand how to do this?
- How can you find **abstractions** that let you focus on the right level of detail and reduce the amount of work?
- How can you use knowledge from the problem **domain** to help understand your system?

HOW CAN YOU DESCRIBE HOW YOUR SYSTEM SHOULD WORK?

- I'm implementing a new feature. That will add a dependence on this other part of the system.
  - Is that ok? What are the consequences of introducing this dependency here?
- How can we describe the system elements, relationships, and constraints on these relationships?
- What are the consequences of following these constraints, or violating these constraints?

WHAT CAN I BORROW?

- I'm trying to make my backend system scale better.
  - Others have solved similar design problems before.

- How can design solutions be described, generalized, and shared?

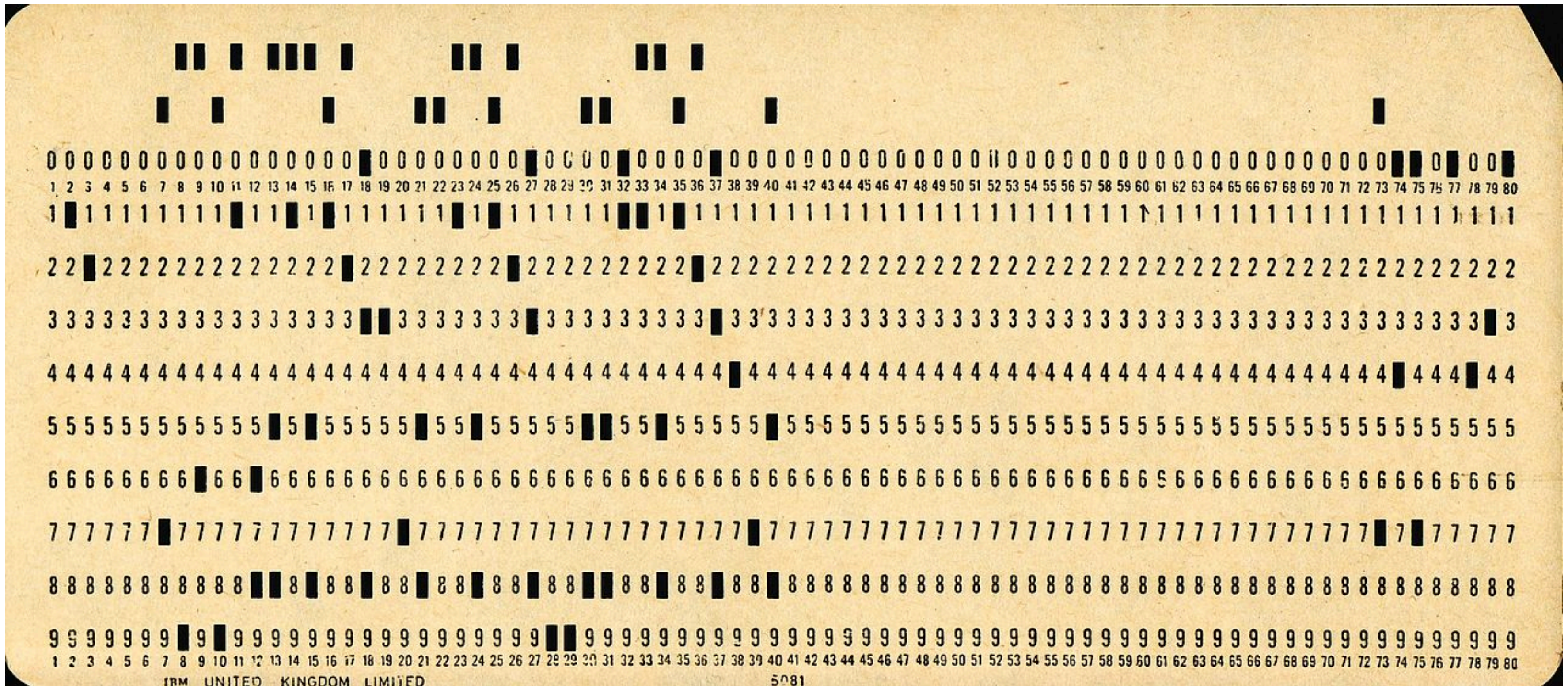
HOW DO YOU FOLLOW A DESIGN?

- What happens when developers make a decision that violates the existing design?
  - Might be intentional choice. Or accidental because of unawareness.
- What techniques and approaches can be used to reduce design violations?

HOW CAN I MAKE IT EASIER FOR OTHERS TO REUSE MY CODE?

- Code often packaged into libraries and frameworks, which have application programming interfaces (APIs)
- How can I design the API so it's easy to learn and less likely to be used incorrectly?
- What happens when I need to change an interface that others depend on?

HISTORY OF SOFTWARE DESIGN AND ARCHITECTURE

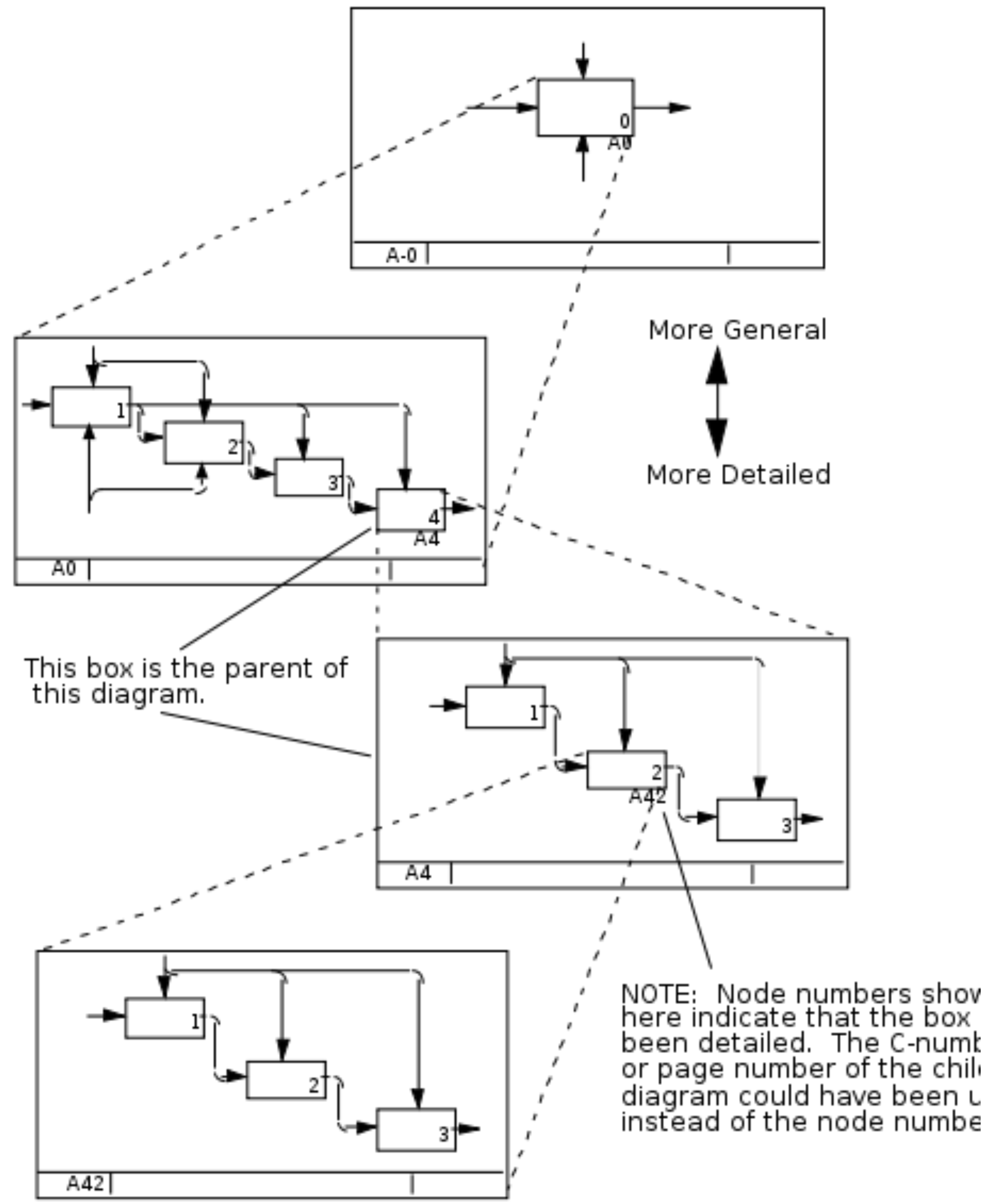


By Pete Birkinshaw from Manchester, UK - Used Punchcard, CC BY 2.0, https://commons.wikimedia.org/w/index.php?curid=49758093

- How do you make changes to this code?
- How do you organize this code?

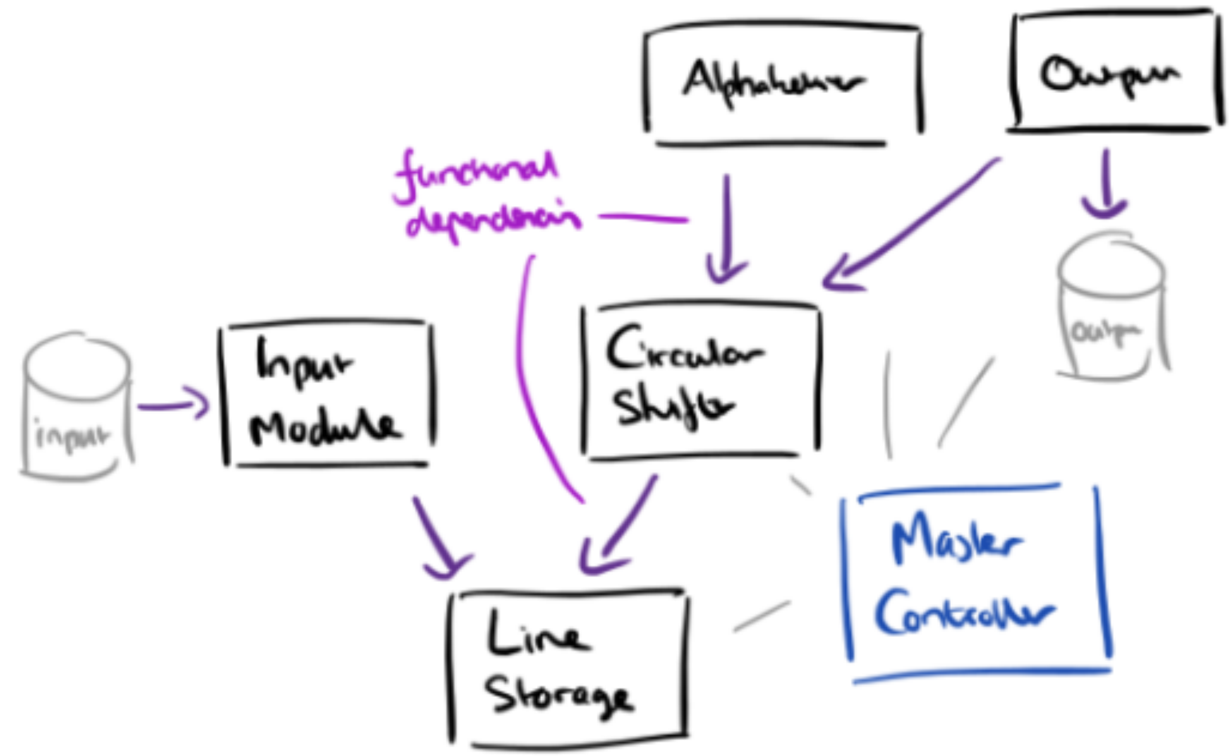
STRUCTURED DESIGN (1970S)

- Can use representations beyond software to describe structure of design
- Make choices about structure in order to make software easier to modify and maintain



INFORMATION HIDING (1970S)

- Software contains **design decisions** which may change
- Code made more maintainable by **hiding** design decisions in module, enabling change to decision without change **rippling** outward and causing changes to dependencies



SOFTWARE ARCHITECTURE (1990S)

- Structural constraints on elements and element relationships can be codified as architectural styles
- Any system following an architectural has specific properties inherent to the architectural style

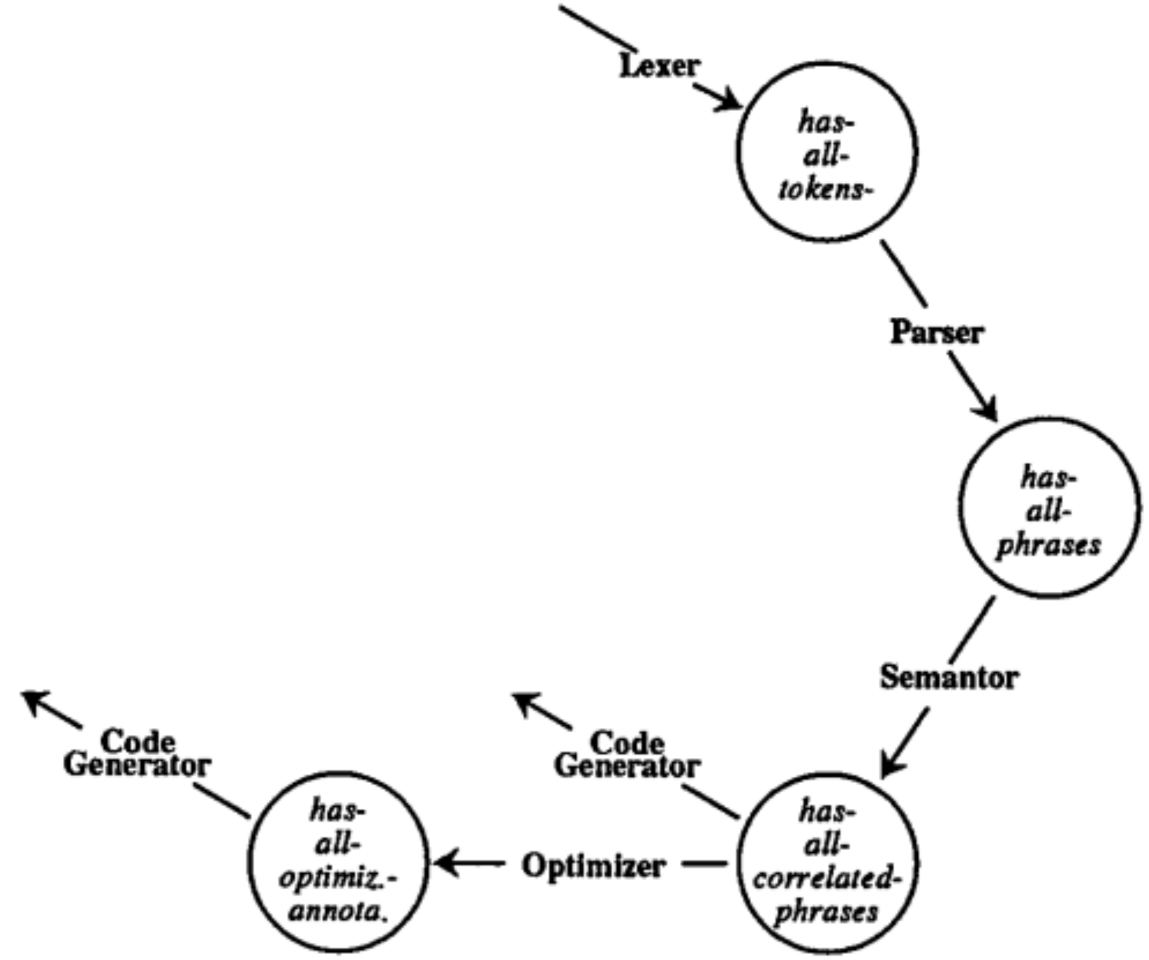
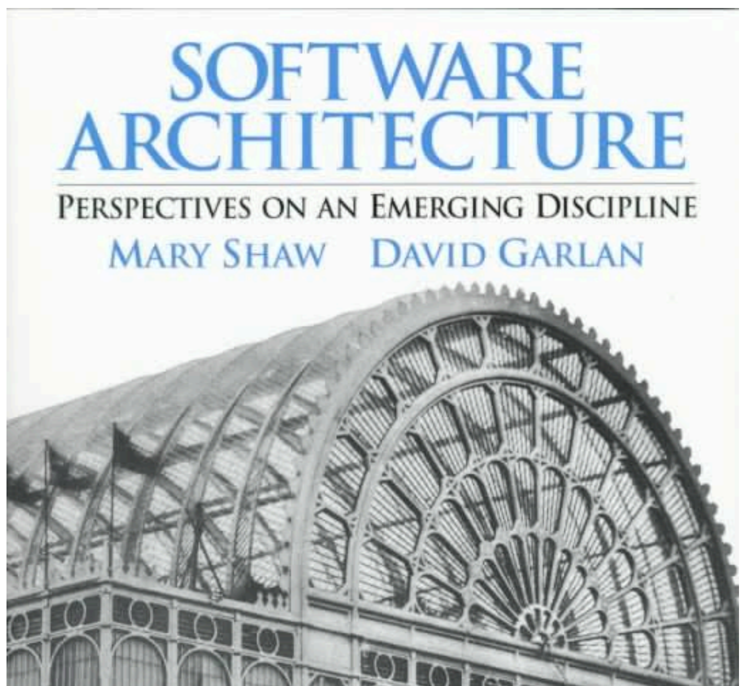


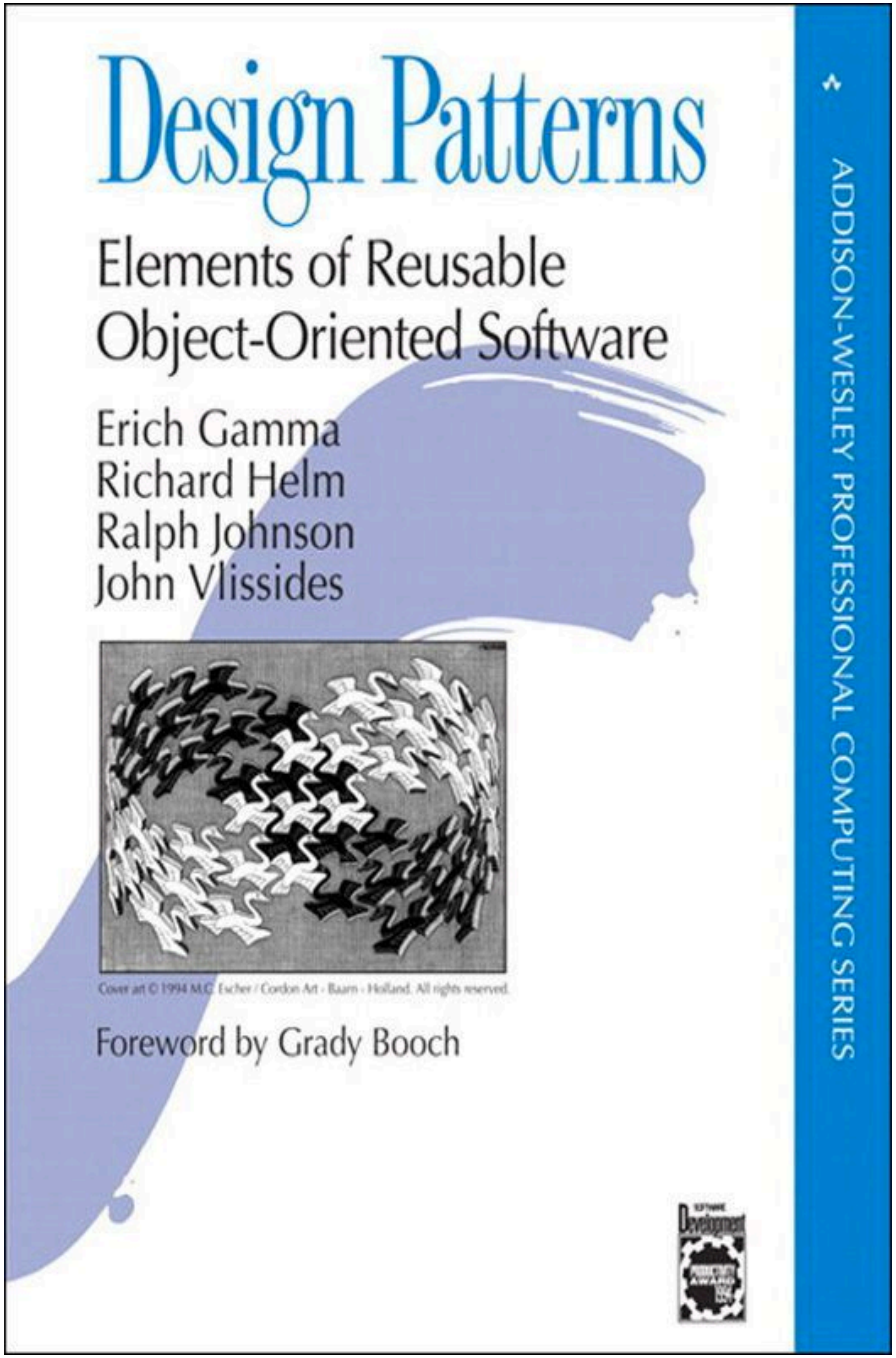
Figure 3: Data View of Sequential Compiler Architecture.

Perry and Wolf. (1992). Foundations for the study of software architecture. FSE.



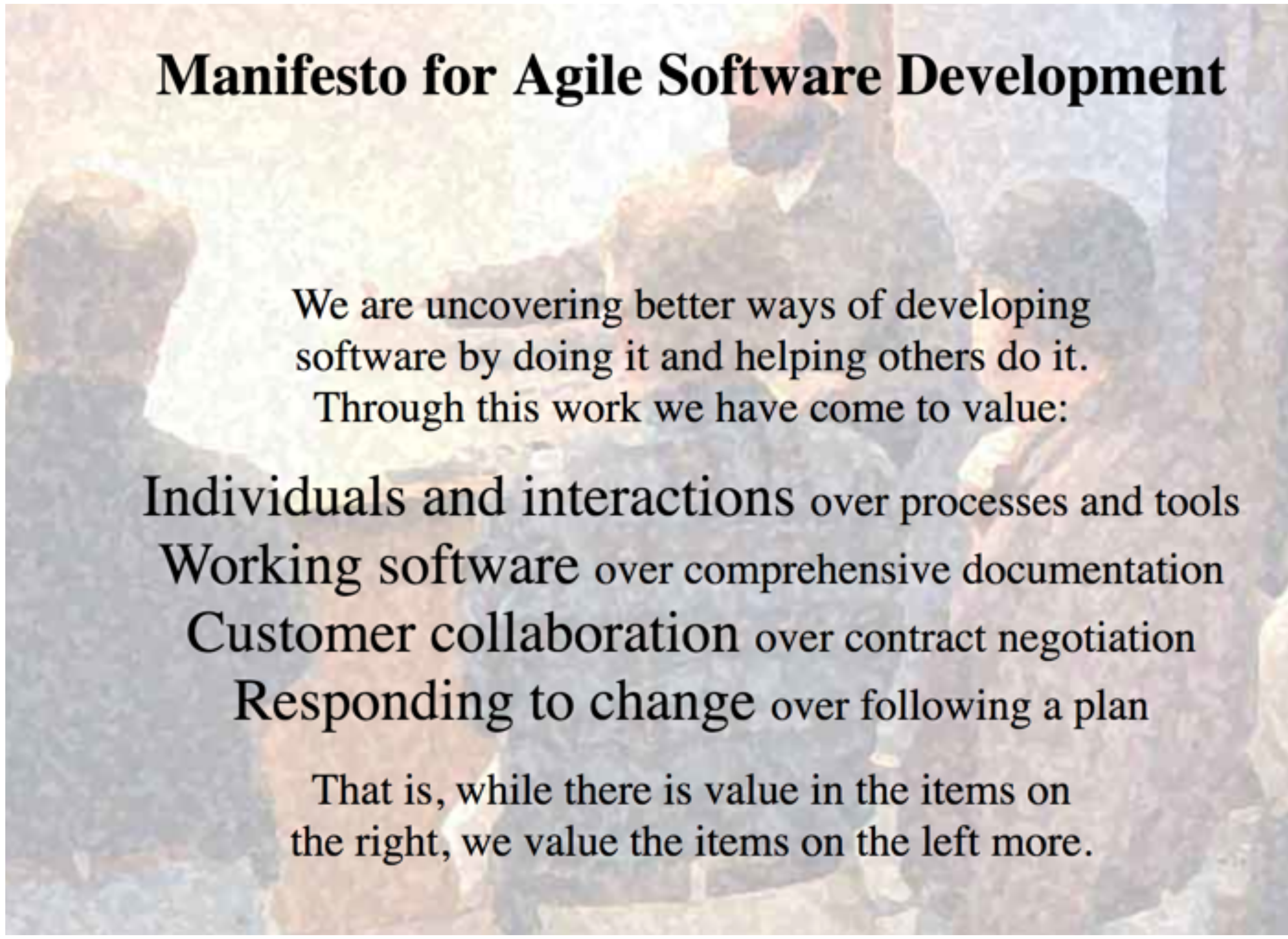
DESIGN PATTERNS (1990S)

- Reusable solution to a problem in a context
- Rather than solving problems from scratch, experts borrow existing solutions to common design problems.
- Giving them names allows them to be recognized and taught



AGILE SOFTWARE DEVELOPMENT (2000S)

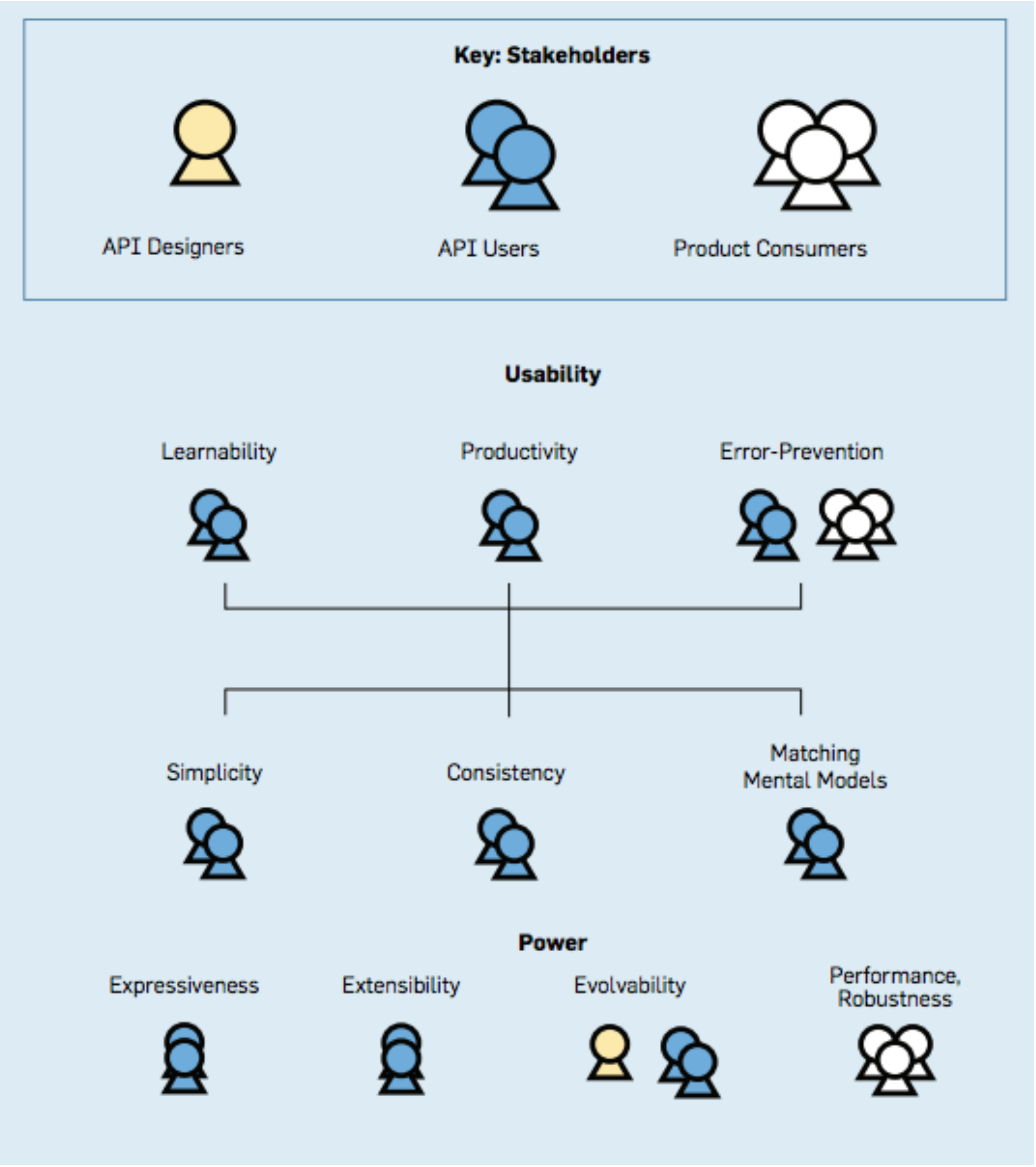
- Architecture built upfront can sometimes be mismatched to goals, particularly when new information is revealed later
- Update architecture and design to respond to change and better knowledge



http://agilemanifesto.org/

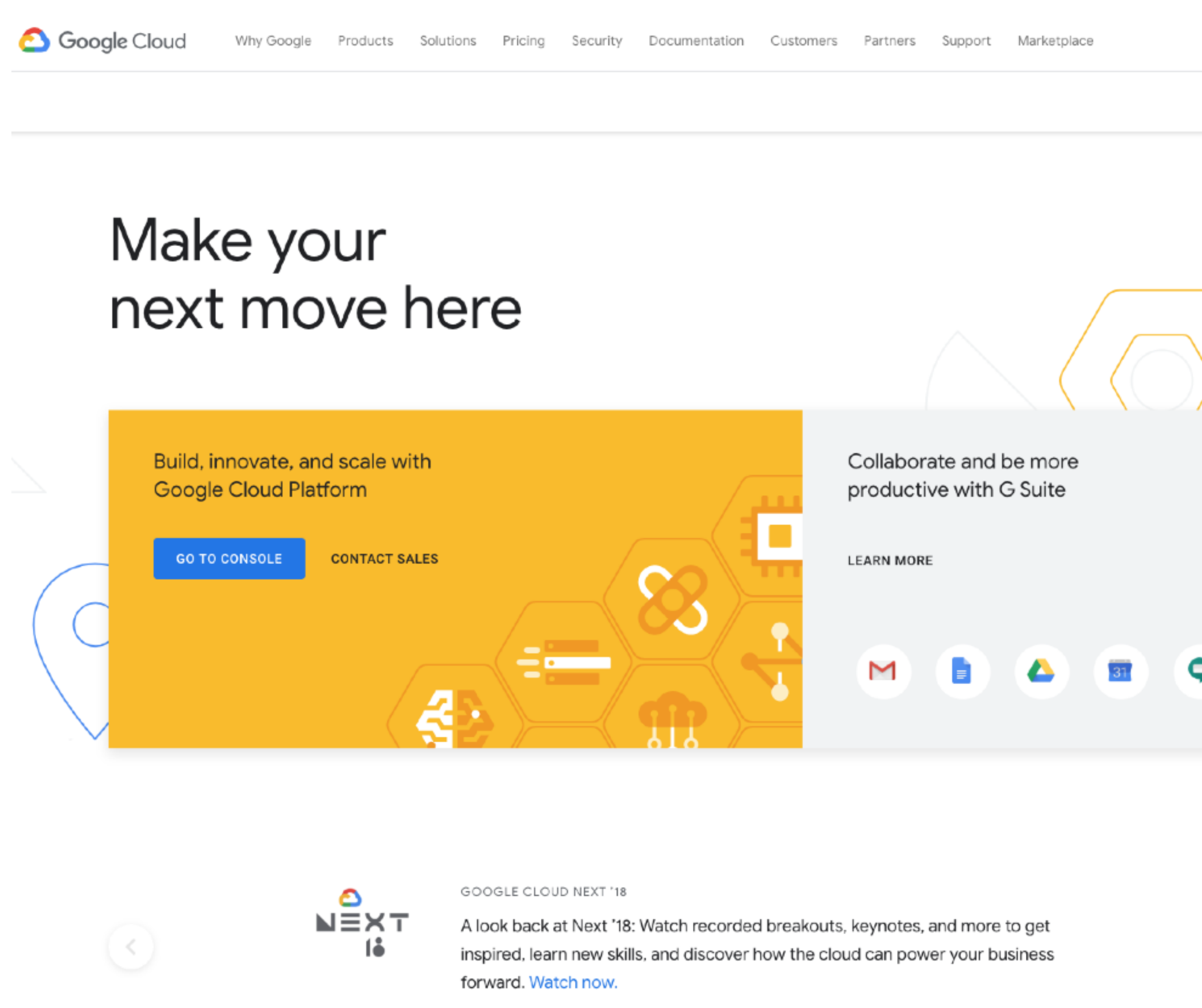
APIS (2000S)

- Web increased availability and number of libraries and frameworks, often as free open source projects
- How do you learn how to use these?
- What can you do to make your API easier to learn and use?



Myers & Stylos, [Improving API Usability](), CACM 59 (6), 2016

SOFTWARE ECOSYSTEMS (2010S)



- Businesses expose web services
  - Market for for software and services
  - APIs create value for organizations
- Systems of systems, where no single owner controls the design of the system from end to end
- Work more distributed, through crowdsourcing, hackathaons, bug bounties
- How do you change an interface, when widely used?

THIS COURSE

- Comprehensive introduction to software architecture and design, including methods, processes, and notations
- Will draw on
  - **Guidelines** and examples from experienced developers
  - **Principles** from software engineering research
  - **Empirical** results and theories from studies of software developers

EXPECTED LEARNING OUTCOMES

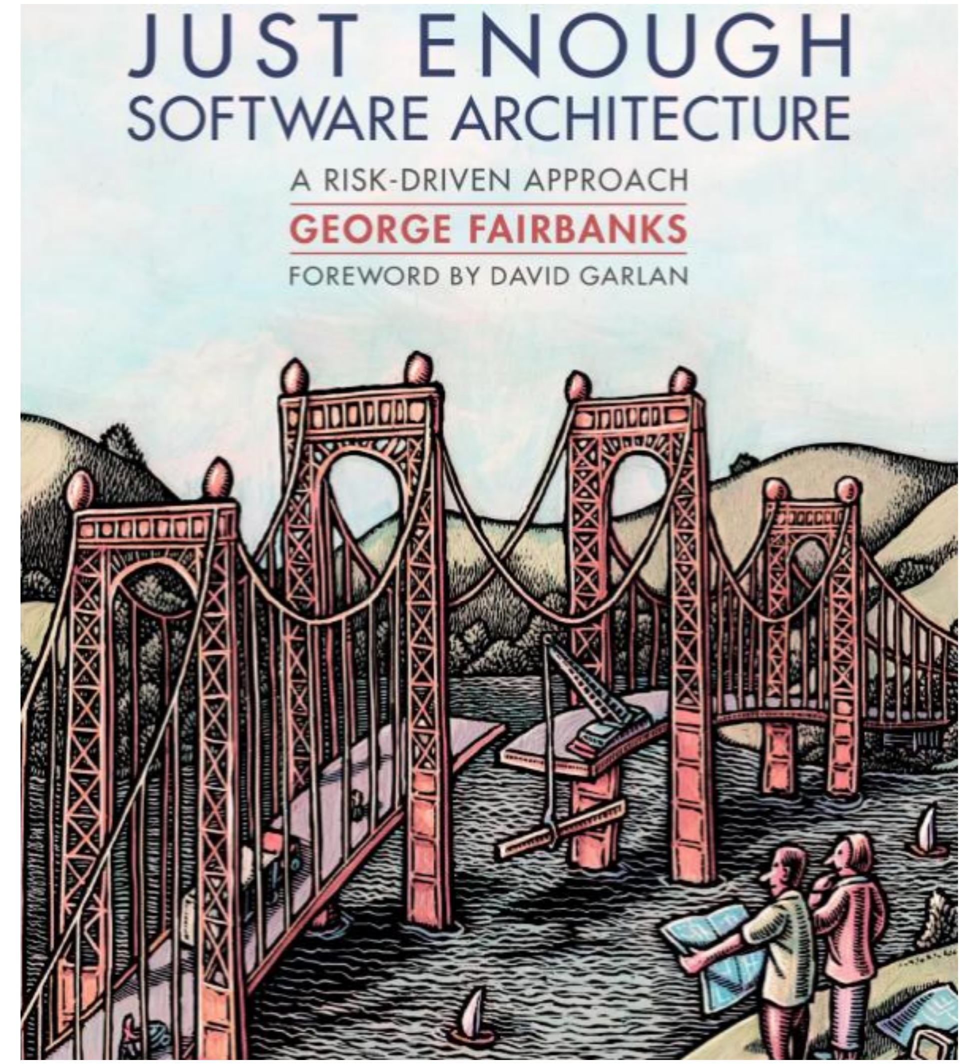
- Characterize a design space, including identifying risks and key architectural decisions
- Design abstractions that express a system or domain model
- Use notations to describe elements and relations in an architecture
- Reverse engineer design decisions, architectural styles, design patterns, design rules, and programming styles from existing systems
- Create and evaluate designs for modifiability and reuse

RESOURCES

- Course website - Syllabus, Schedule
- Piazza - Announcements, Assignments, Discussion, Questions
- Blackboard - grades

READINGS

- Weekly readings from the course textbook and from classic software engineering papers
- Should read the assigned readings before each class. Class may involve discussion of readings
- Papers available online
  - Access the link from a device connected through a Mason IP address, which you can get through Mason VPN if off campus



IN-CLASS ACTIVITIES

- Each class will include an extended in-class activity, often in groups
- Practice architecture and design on small problems
- Will generate a small **deliverable**, often in form of diagram, sketch, or list
- Graded
  - **Satisfactory**: put forth a good effort in accomplishing the activity's goals (10/10)
  - **Needs improvement**: substantially misunderstood the activity or did not make meaningful progress (5/10)
  - **Not present**: did not submit deliverable from activity (0/10)
- To accommodate planned or unplanned absences, three lowest scores (including absences) dropped

HWS (A.K.A. "PROJECT")

- Practice reverse engineering the architecture and design of real systems
- Gain experience with working with architecture and design in context of real world problems
- Reverse engineer the design and architecture of three competing open source applications that each offer alternative solutions to the same underlying problem
- 6 assignments
- In class presentation at end of semester

HWS: GROUPS

- Strongly encouraged (but not required) to form a project group of two or three.
- Many assignments require substantial work and have been designed with group in mind
- Encourage you to work together and discuss your assignments over Skype, Google Hangouts, IM, Slack, etc
- May choose to work in a group of one, but responsible for the same work

LATE HW ASSIGNMENTS

- Can submit up to 24 hours late, lose 10%
- **HW submissions more than 24 hours late will receive a 0**

HW0

- Due next Tues before class
- Form a group of 1, 2, or 3
- Select 3 systems to reverse engineer

EXAMS

- Midterm exam and comprehensive final
- Includes both in class lectures and material from assigned readings
- Mix of multiple choice, short response, short essay

GRADES

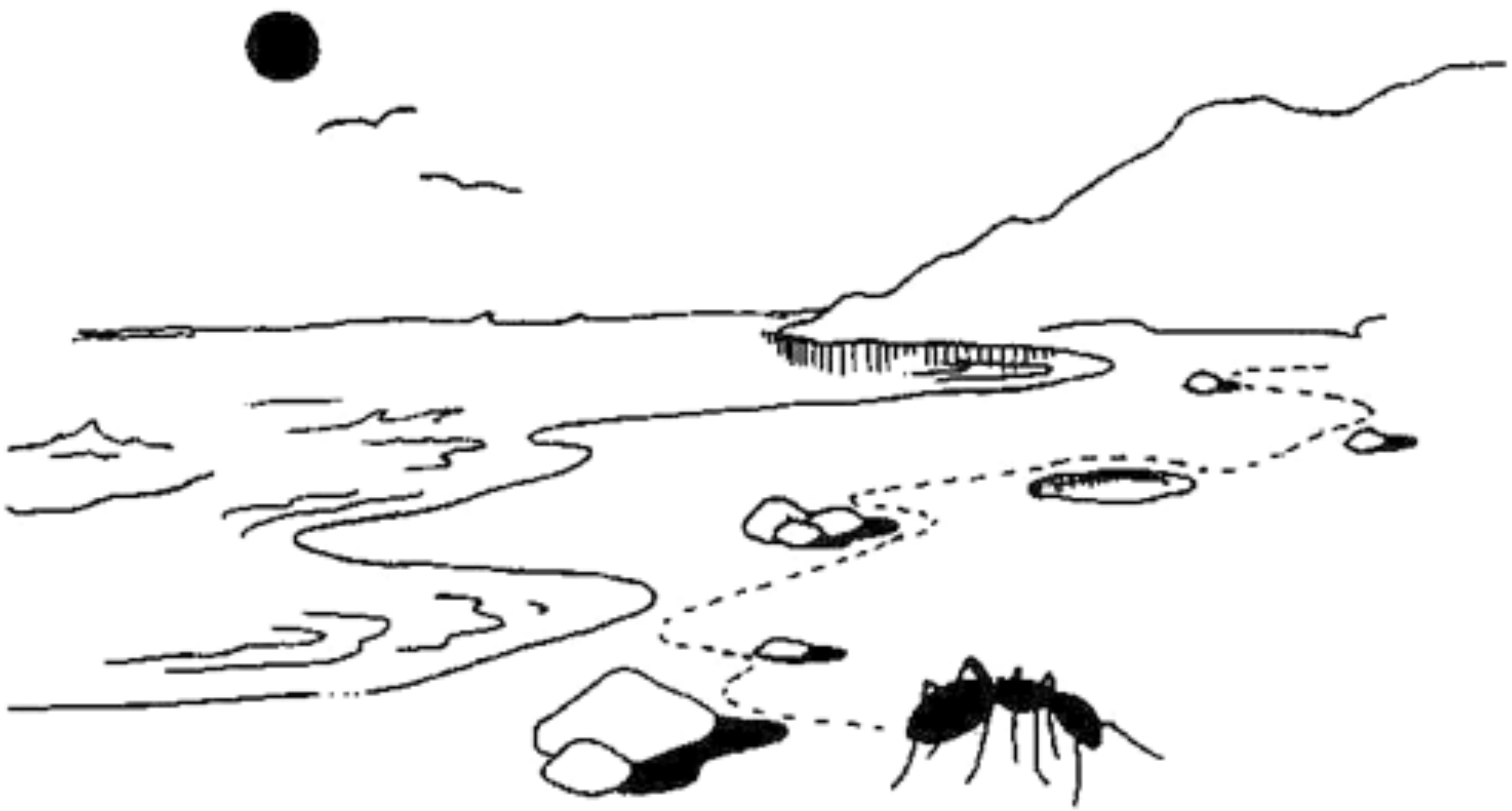
- In-Class Activities: 10%
- HWs and project presentation: 40%
- Mid-term exam: 20%
- Final exam: 30%

WHAT CHOICES DO YOU MAKE WHEN DESIGNING?

DESIGN AS CHOICE

- Should you prioritize initial loading time or responsiveness?
- Should you choose a layered architecture, a pipes and filter architecture, or something else?
- Should you implement this feature in the server, the client, or both?
- Should you implement this feature in class A or class B?

ANT ON A BEACH



Herb Simon. The sciences of the Artificial.

SYSTEMS ADAPT TO TASK ENVIRONMENT

- Can describe the literal path the ant took.
- Or could describe the underlying environment that lead to the ant's behavior.
  - Task environment: What is the ant being asked to do. What can it do.
  - Goals / objectives: How is the performance of the ant assessed?
  - Solutions: What is the final realized behavior, given these objectives?
- Describing system in this way leads to deeper understanding of how system works.

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

COMPARISON

- All offer the same functionality
- But differ in their **quality attributes**
- Ease of modifiability
  - V1 and V2 supported ad hoc queries in seconds by writing a new grep expression or changing SQL query
  - V3 required a new program to be written to build a new query type
- Scalability --> V3 more scalable
- Liveness of results --> V1 always got latest results, V3 short delay

SYSTEMS DESIGNED WITH GOALS IN MIND

- Lists of requirements and features systems should include
- List of quality attributes by which to compare alternative designs, both of which offer the same features

EXAMPLES OF QUALITY ATTRIBUTES ("ILLITIES")

- 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

IN CLASS ACTIVITY

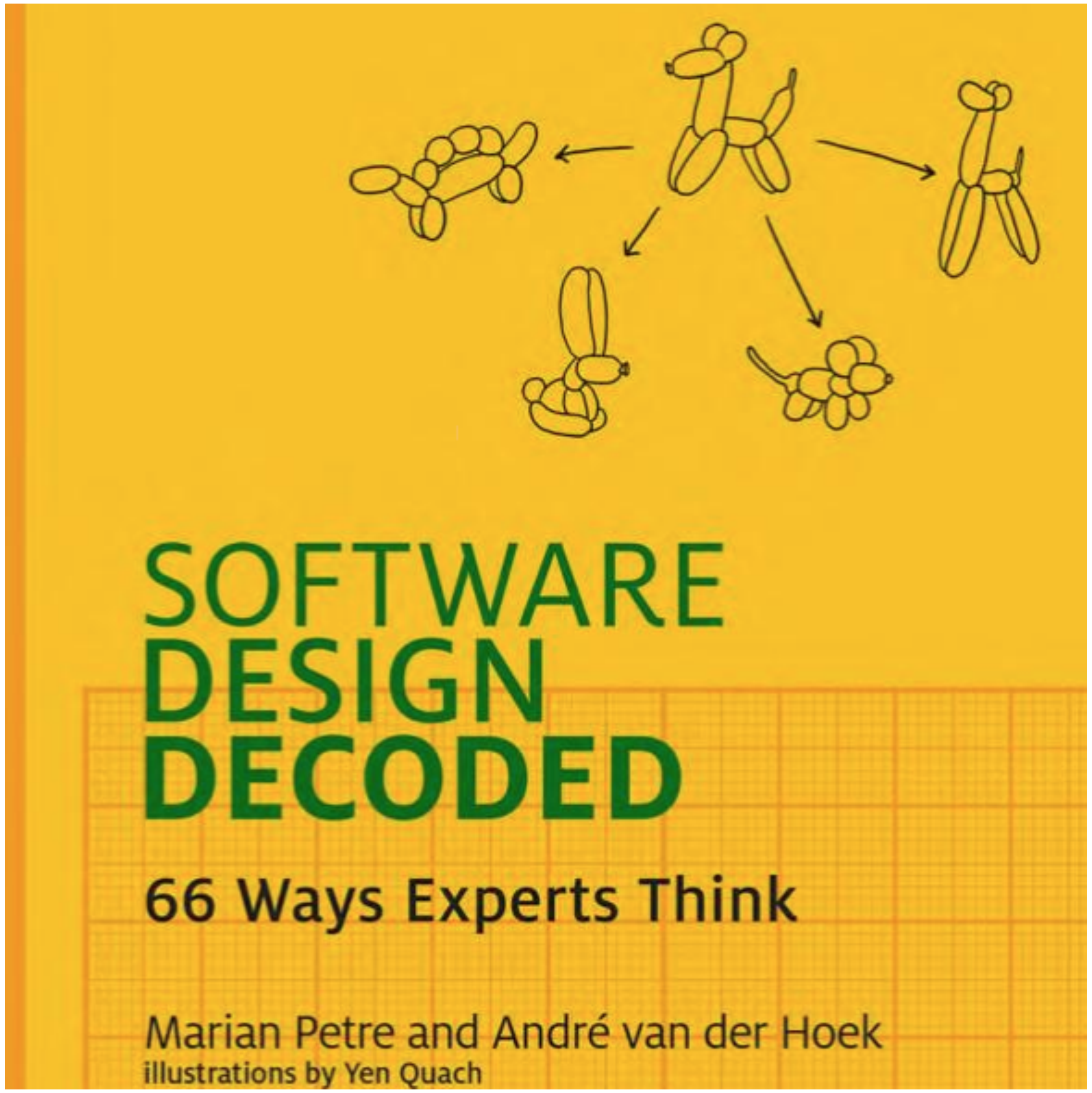
- Find a partner. Discuss with partner examples of projects where the system has been rebuilt.
- What motivated this?
- What changed? How much effort was it to change?

DESIGN SPACE

- What are the **dimensions** along which a design could vary?
  - What are mutually exclusive design decisions?
- Goal: justify selection of design choices with regards to goals
- Sometimes just built something, and not clear what other ways might have been possible.
  - Requires brainstorming **alternatives.**
- Sometimes not clear if choice was correct
  - Requires considering which alternative best satisfies goals

TECHNIQUES FOR MAKING BETTER DECISIONS

- Researchers have observed expert designers and described how they work to build **guidelines** for making better design decisions.



EXPERTS WORK WITH UNCERTAINTY

- Experts keep options open
  - Know that decisions may be **revised** and defer decisions that do not need to be made yet
- Experts see error as opportunity
  - Understanding what happened reveals insights about the problem, such as **assumptions**, misconceptions, misalignments
- Experts make tradeoffs
  - Experts collect as much **information** as possible and consider how decision trades off with goals
- Experts adjust to the degree of uncertainty present
  - Routine problems have less uncertainty and decisions made earlier; original problems require more exploration, invention, and backtracking

EXPERTS ARE NOT AFRAID

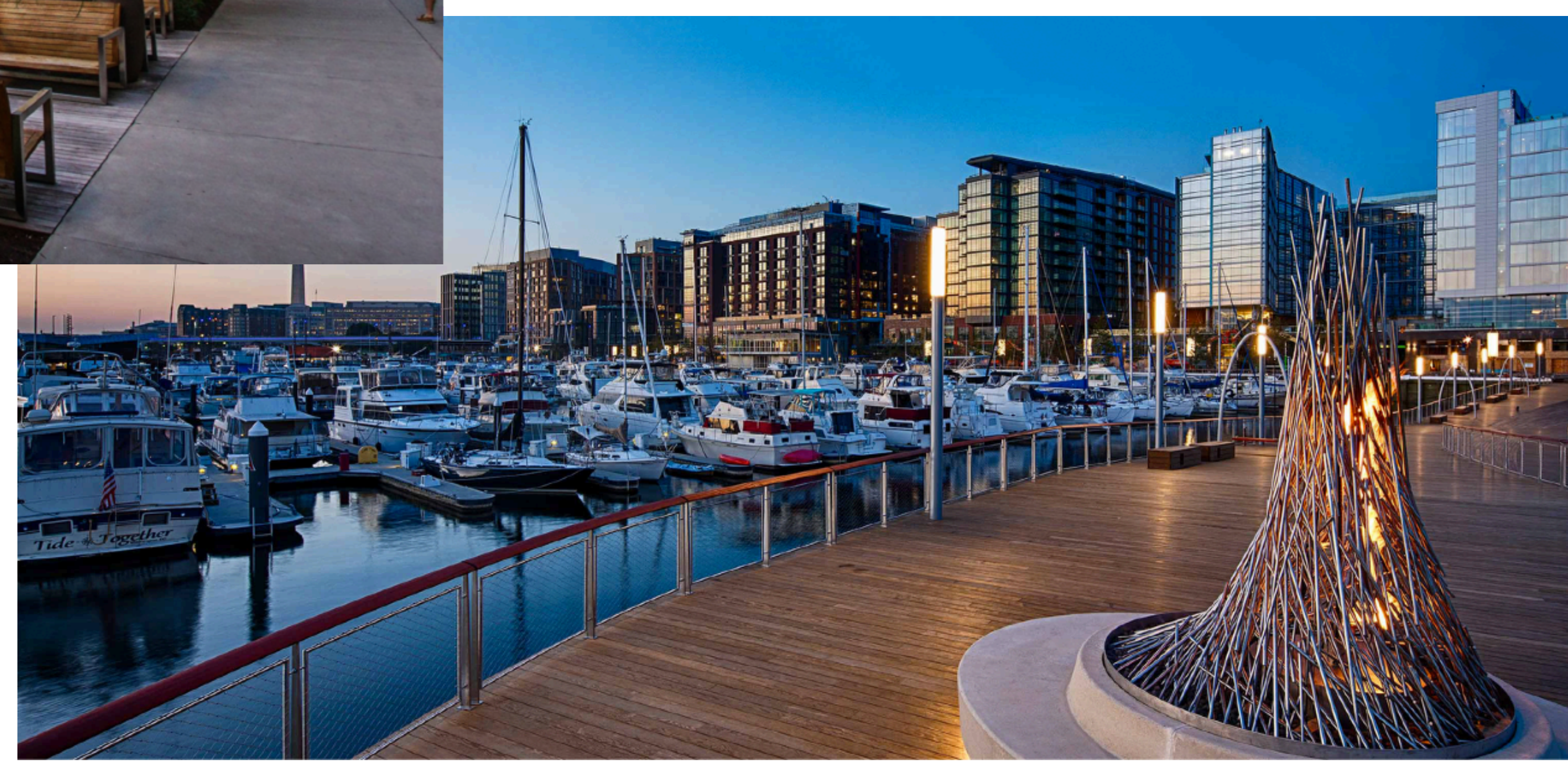
- Experts focus on the essence
  - Identify a **core** set of considerations first, nailing the core before focusing effort on peripheral decisions
- Experts address knowledge deficiencies
  - Look for **gaps** in understanding of problem, try to address these deficiencies. Explicitly identify assumptions and try to test when possible.
- Experts try the opposite
  - When they are stuck, they might try the **opposite**, generating new ideas for alternatives.

EXPERTS TEST

- Experts are skeptical
  - When others are content, experts remain **skeptical** that the current leading solution is good or even good enough.
- Experts simulate continually
  - Experts imagine how a design will work, **simulating** aspects of the envisioned system and how it will support variety of scenarios steps by step
- Experts alert to evidence that challenges theory
  - Open to **unexpected** information that does not conform with current understanding and which suggests problems lurking beneath the surface.

IN CLASS ACTIVITY

DESIGN ACTIVITY: MIXED USE DEVELOPMENT



DESIGN ACTIVITY

- Will explore **design space** for mixed-used development for Amazon's future HQ2
- Problem: Design a site plan for how individual spaces within site will be **arranged**
    - Which spaces will be **adjacent** on same street, hallway, floor, building, etc.
        - e.g., Will there be a single food court for high end dining and fast food? Or will each dining establish be by itself?
    - Includes spaces of several types: office space, hotel space, apartments, condos, retail (clothing, etc.), high end dining, fast food
    - Can assume for each type, will be several instances (e.g., at least 3 fast food restaurants)
- Example design goals: minimize pedestrian travel time, retail discoverability

DESIGN ACTIVITY: LOGISTICS

- Form groups of 2 or 3
- Will occur in several steps. Each step will have a deliverable in the form of a separate piece of paper.
- Each deliverable should have names of group members, step number,
- Will hand in deliverable at end of class
- Graded based on following design activity steps, not the quality of the final site design
- Have fun!

DESIGN ACTIVITY: STEP 1

- Deliverable: design a solution
  - Should list at least 4 key decisions and a short explanation (e.g., a sentence) justifying each one
  - Might (optionally) include a sketch of what site plan would look like

DESIGN ACTIVITY: STEP 2

- Swap designs with a 2nd group
- Use a **new** piece of paper
- At the top of the paper, put your names and the names of the design you are critiquing
- Write a critique of the design decisions made
- Deliverable: for each decision
    - What assumptions does it make?
    - Do you think these assumptions are reasonable? If not, why not? What additional analysis or evidence might help test these assumptions?

DESIGN ACTIVITY: STEP 3

- Return the design back to group, along with your critique
- Using the critique you just received, iterate on your design
  - May change or keep original decision
  - May add new decisions or delete decisions you want to leave open
- Deliverable
  - New list of decisions
  - Update rationale and assumptions for decisions based on critique

DESIGN ACTIVITY: STEP 4

- Join together with another group (not one you have exchanged materials)
- Deliverable: build a design space
  - List dimensions that you considered that characterize design solutions
  - List alternatives for each dimension
  - Briefly summarize tradeoffs between alternatives

DESIGN ACTIVITY: DISCUSSION

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