

ECE444: Software Engineering

Introduction to Software Architecture

Shurui Zhou

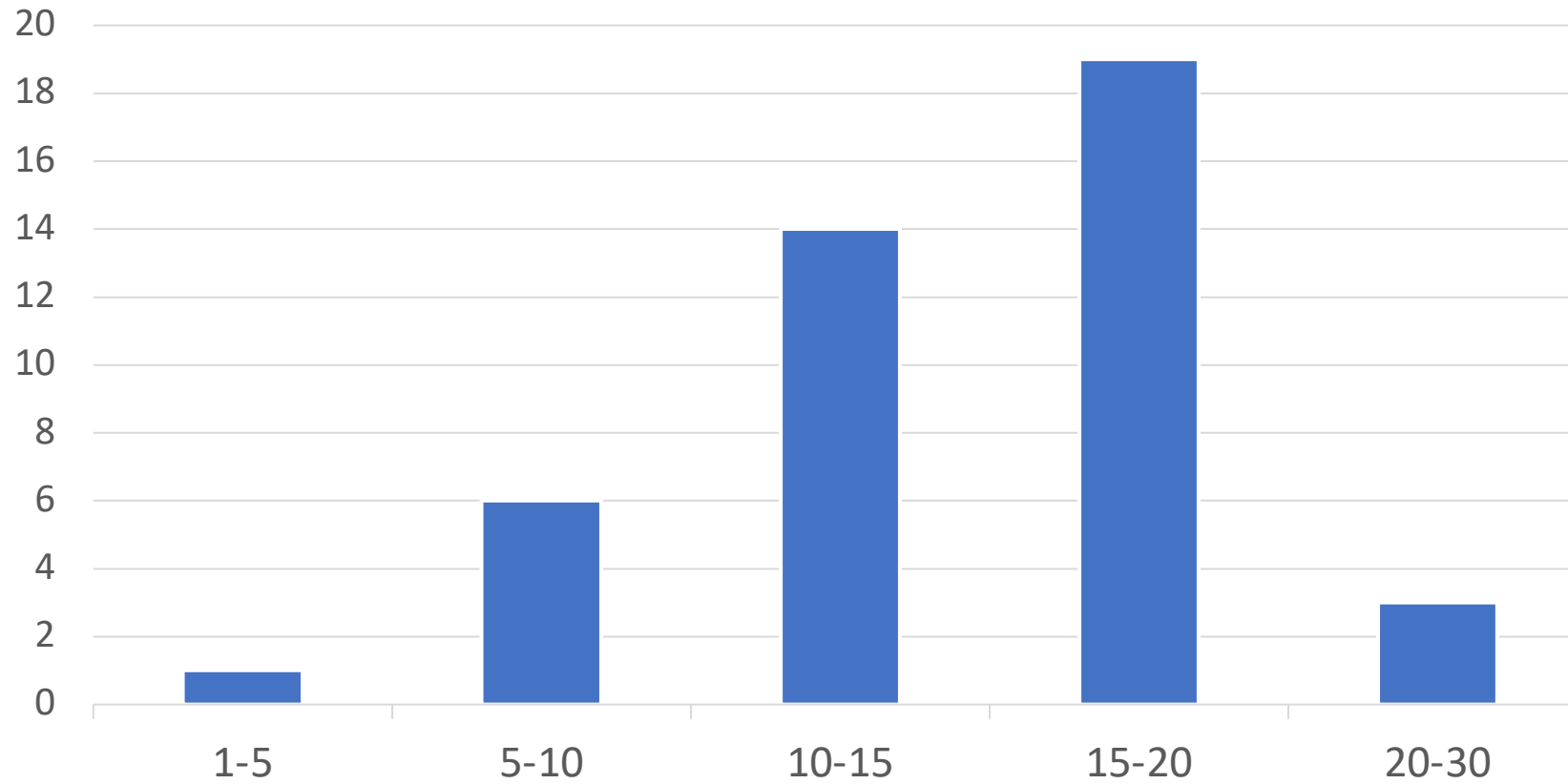


The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

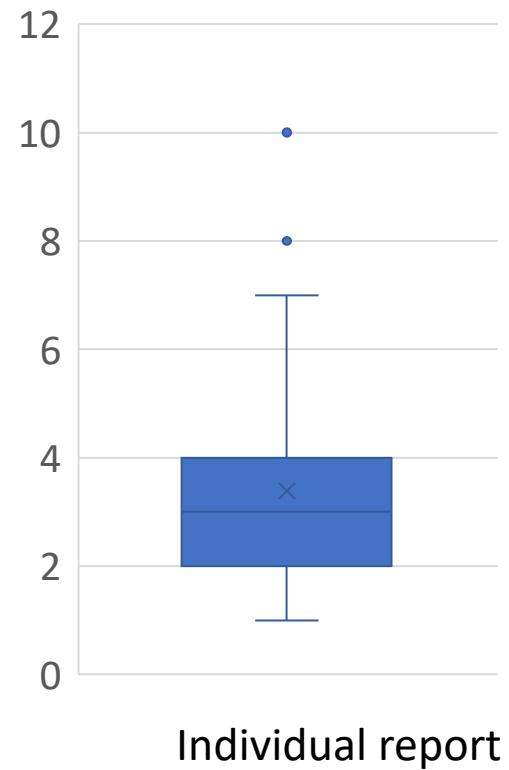
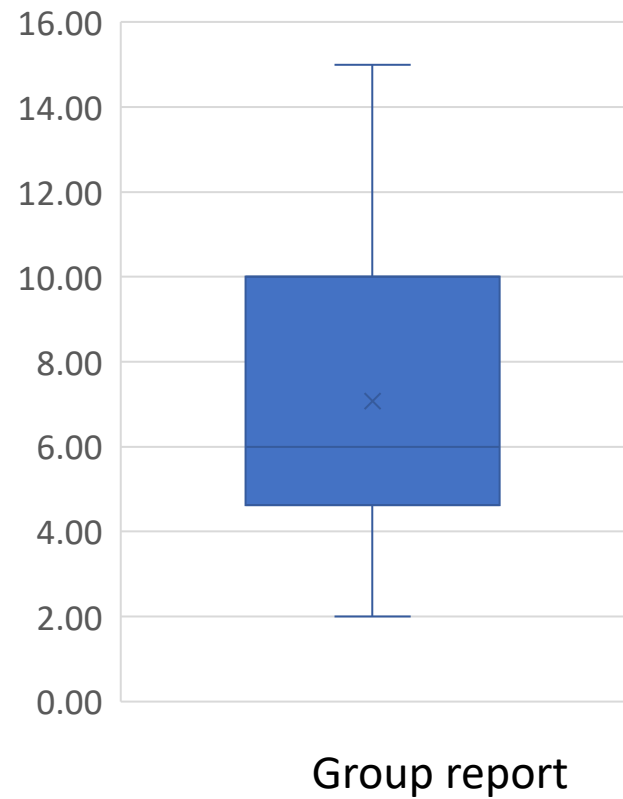
Administrivia

- Milestone 3 peer review Due Friday (10/8) 11:59pm
- Milestone 2 grade release by Sunday
- About workload survey (44 response/70 students)

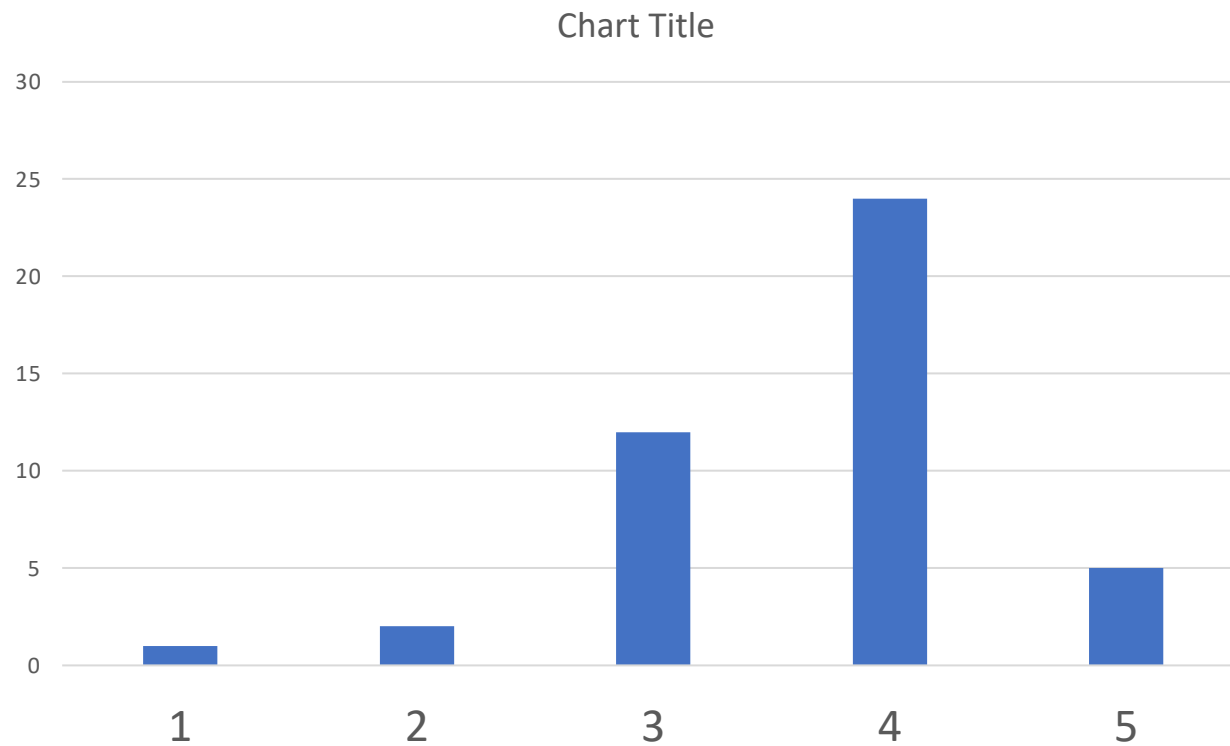
How much time have you spent on ECE444
for the past 3 weeks (hours/week)?



- How much time have you spent on group meetings? (hours/week)
- 2-4h
- Hours on Milestone 2

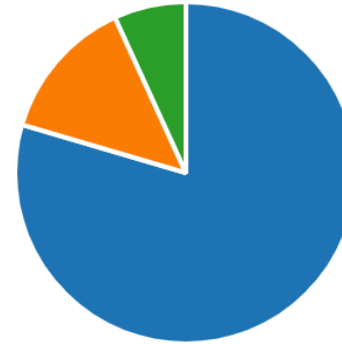


- I have understood most of the content of ECE444 so far
(1 -- Strong Disagree, 5 -- Strongly Agree)



15. I am ok with increasing the portion of Lab tasks in the Composition of Final Marks and remove the rest paper reading assignments

[More Details](#)



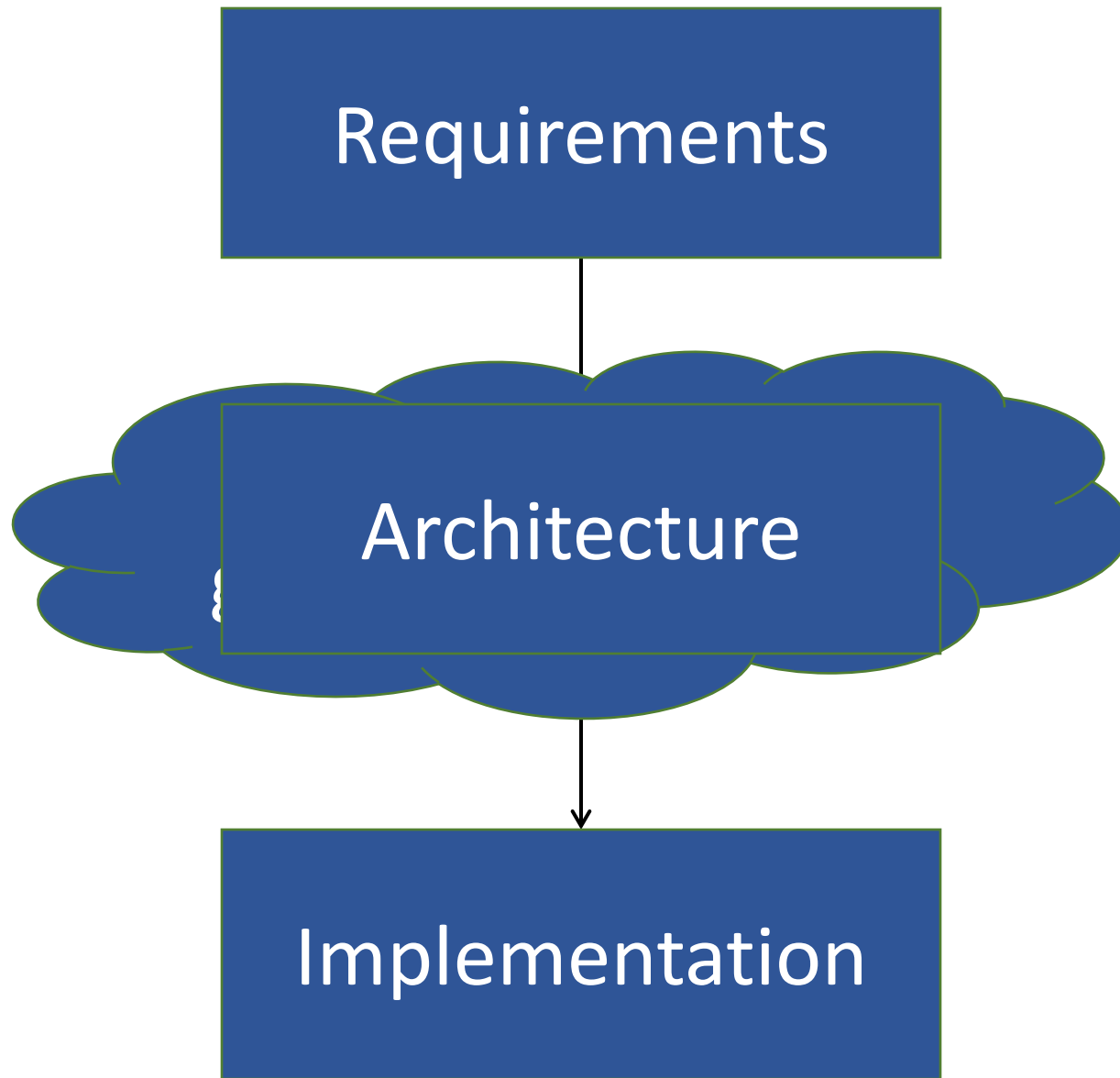
16. I am ok with decreasing the number of Lab tasks in later weeks to give us more time to work on the project

[More Details](#)



Learning Goals

- Understand the abstraction level of architectural reasoning
- Approach software architecture with quality attributes in mind
- Distinguish software architecture from (object-oriented) software design
- Use notation and views to describe the architecture suitable to the purpose

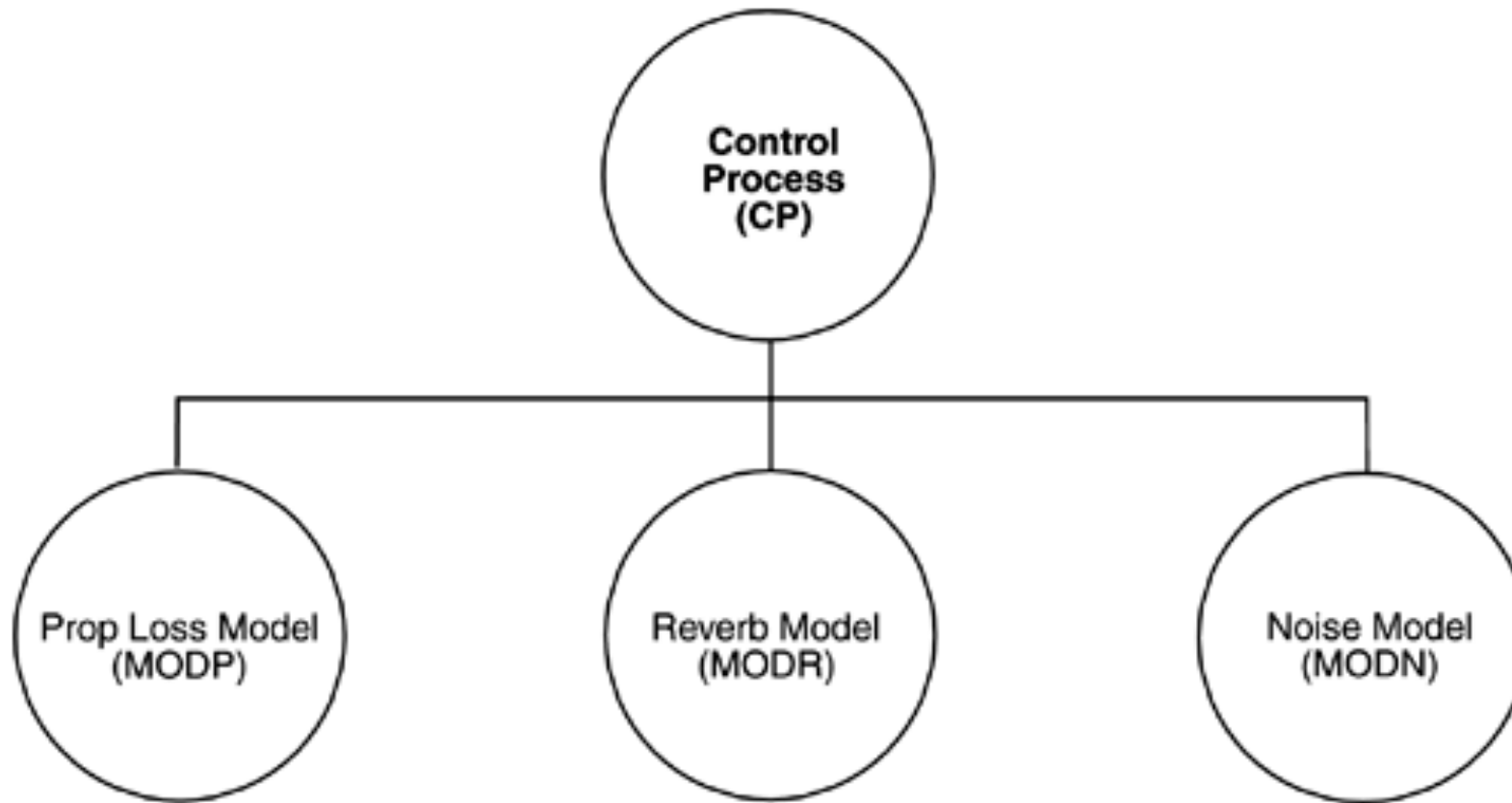


Quality Requirements, now what?

- "should be highly available"
- "should answer quickly, accuracy is less relevant"
- "needs to be extensible"
- "should efficiently use hardware resources"

Software Architecture

Typical, but uninformative, presentation of a software architecture



From Bass et al. Software Architecture in Practice, 2nd ed.

Software Architecture

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

[Bass et al. 2003]

Software Architecture

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

[Bass et al. 2003]

Software Architecture

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

[Bass et al. 2003]

Software Architecture

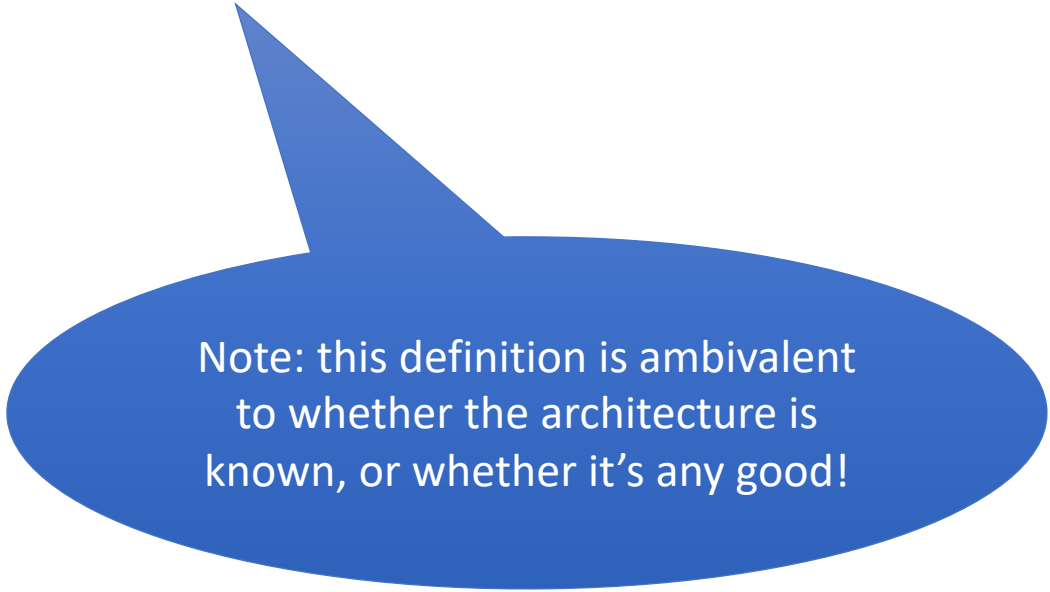
The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

[Bass et al. 2003]

Software Architecture

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

[Bass et al. 2003]



Note: this definition is ambivalent to whether the architecture is known, or whether it's any good!

Why is software architecture important?

1. inhibit or enable a system's driving quality attributes.
2. to reason about and manage change as the system evolves.
3. enables early prediction of a system's qualities.
4. enhances communication among stakeholders.
5. a carrier of the earliest and hence most fundamental, hardest-to-change design decisions.
6. defines a set of constraints on subsequent implementation.
7. Influencing the organizational structure
8. provide the basis for evolutionary prototyping.
9. the key artifact that allows the architect and project manager to reason about cost and schedule.
10. can be created as a transferable, reusable model that forms the heart of a product line.
11. Architecture-based development focuses attention on the assembly of components.
12. architecture channels the creativity of developers, reducing design and system complexity.
13. can be foundation for training a new team member.

Why is software architecture important?

1. inhibit or enable a system's driving quality attributes.
2. to reason about and manage change as the system evolves.
3. enables early prediction of a system's qualities.

Beyond functional correctness

- Quality matters, eg.,
 - Performance
 - Availability
 - Modifiability, portability
 - Scalability
 - Security
 - Testability
 - Usability
 - Cost to build, cost to operate

Why is software architecture important?

1. inhibit or enable a system's driving quality attributes.
2. to reason about and manage change as the system evolves.
3. enables early prediction of a system's qualities.
4. enhances communication among stakeholders.

Why is software architecture important?

1. inhibit or enable a system's driving quality attributes.
2. to reason about and manage change as the system evolves.
3. enables early prediction of a system's qualities.
4. enhances communication among stakeholders.
5. a carrier of the earliest and hence most fundamental, hardest-to-change design decisions.
6. defines a set of constraints on subsequent implementation.
7. Influencing the organizational structure
8. provide the basis for evolutionary prototyping.

Why is software architecture important?

1. inhibit or enable a system's driving **quality** attributes.
2. to **reason** about and manage change as the system evolves.
3. enables early prediction of a system's **qualities**.
4. enhances **communication** among stakeholders.
5. a carrier of the earliest and hence most fundamental, hardest-to-change design decisions.
6. defines a set of **constraints** on subsequent implementation.
7. Influencing the **organizational** structure
8. provide the basis for **evolutionary prototyping**.
9. the key artifact that allows the architect and project manager **to reason about cost and schedule**.
10. can be created as a **transferable, reusable model** that forms the heart of a product line.
11. Architecture-based development focuses attention on the **assembly** of components.
12. architecture channels the creativity of developers, **reducing design and system complexity**.
13. can be foundation for **training** a new team member.

[Bass et al. 2013]

Case Study: Architecture and Quality at Twitter

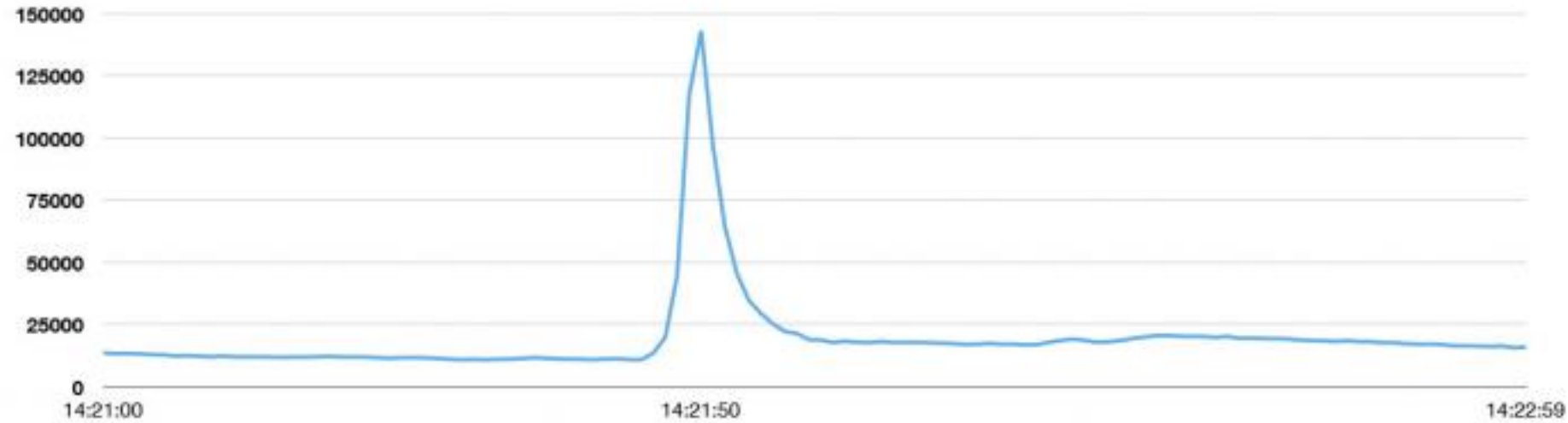
Balse!



Toei Company

A scene from *Castle in the Sky*, the classic 1986 film by Hayao Miyazaki.





On Dec 9, the television screening in Japan of Hayao Miyazaki's "Castle in the Sky" led to 25,088 Tweets per second – a new Twitter record.

— Twitter Comms (@twittercomms) **December 14, 2011**

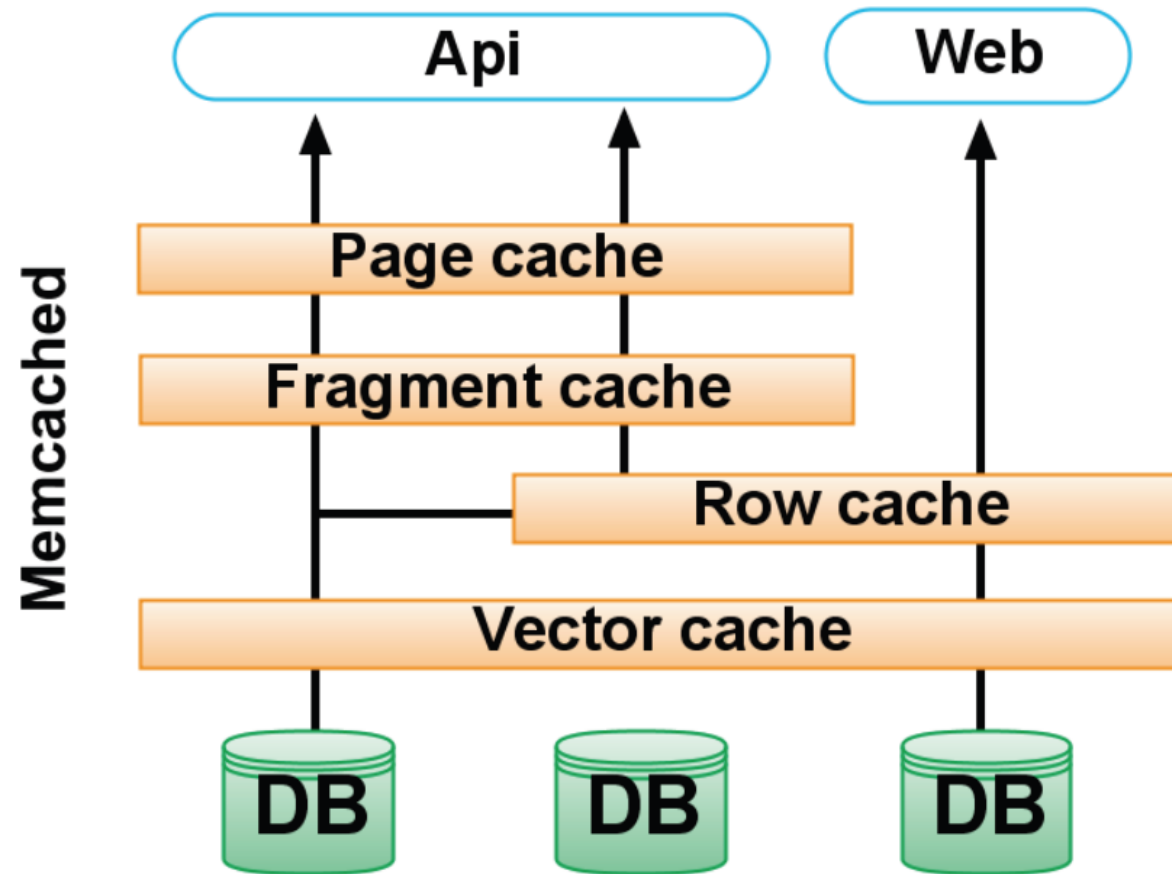


"After that experience, we determined we **needed to step back**. We then determined we needed to **re-architect** the site to support the continued growth of Twitter and to keep it running smoothly."

Inspecting the State of Engineering

- Running one of the world's largest Ruby on Rails installations
- 200 engineers
- Monolithic: managing raw database, memcache, rendering the site, and presenting the public APIs in one codebase

Caching



Inspecting the State of Engineering

- Running one of the world's largest Ruby on Rails installations
- 200 engineers
- Monolithic: managing raw database, memcache, rendering the site, and presenting the public APIs in one codebase
- Increasingly difficult to understand system; organizationally challenging to manage and parallelize engineering teams
- Reached the limit of throughput on our storage systems (MySQL); read and write hot spots throughout our databases
- Throwing machines at the problem; low throughput per machine (CPU + RAM limit, network not saturated)
- Optimization corner: trading off code readability vs performance

Twitter's Quality Requirements/Redesign goals

- Improve median latency; lower outliers *performance*
- Reduce number of machines 10x
- Isolate failures *reliability*
- "We wanted cleaner boundaries with “related” logic being in one place"
 - encapsulation and modularity at the systems level (rather than at the class, module, or package level) *maintainability*
- Quicker release of new features
 - "run small and empowered engineering teams that could make local decisions and ship user-facing changes, independent of other teams" *modifiability*

JVM vs Ruby VM

- Rails servers capable of 200-300 requests / sec / host
- Experience with Scala on the JVM; level of trust
- Rewrite for JVM allowed 10-20k requests / sec / host

Programming Model

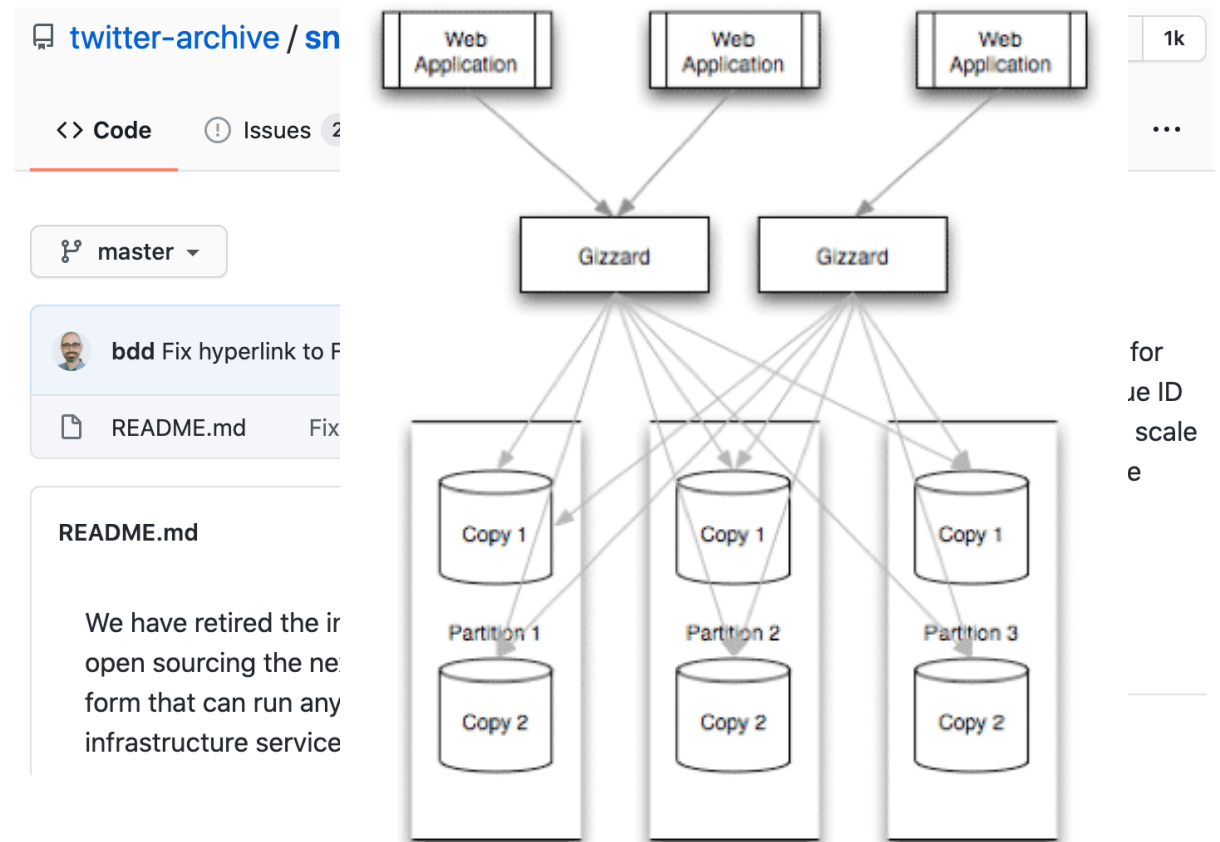
- Ruby model: Concurrency at process level; request queued to be handled by one process
- Twitter response aggregated from several services – additive response times
- *"As we started to decompose the system into services, each team took slightly different approaches. For example, the failure semantics from clients to services didn't interact well: we had no consistent back-pressure mechanism for servers to signal back to clients and we experienced "thundering herds" from clients aggressively retrying latent services."*
- **Goal:** Single and uniform way of thinking about concurrency
 - Implemented in a library for RPC (Finagle), connection pooling, failover strategies and load balancing

Independent Systems

- From monolithic system to multiple services
 - Agree on RPC interfaces, develop system internals independently
 - Self-contained teams
- Monolithic vs Service-oriented
 - *In our monolithic world, we either needed experts who understood the entire codebase or clear owners at the module or class level. Sadly, the codebase was getting too large to have global experts and, in practice, having clear owners at the module or class level wasn't working. Our codebase was becoming harder to maintain, and teams constantly spent time going on "archeology digs" to understand certain functionality. Or we'd organize "whale hunting expeditions" to try to understand large scale failures that occurred. At the end of the day, we'd spend more time on this than on shipping features, which we weren't happy with."*

Storage

- Single-master MySQL database bottleneck despite more modular code
- Move to distributed database (Glizzard on MySQL) with "roughly sortable" ids
- Stability over features – using older MySQL version



Data-Driven Decisions

- Many small independent services, number growing
- Own dynamic analysis tool on top of RPC framework
- Framework to configure large numbers of machines
 - Including facility to expose feature to parts of users only

Key Insights: Twitter Case Study

- Architectural decisions affect entire systems, not only individual modules
- Abstract, different abstractions for different scenarios
- Reason about quality attributes early
- Make architectural decisions explicit

Question: *Did the original architect make poor decisions?*