

# 17-423/723: Designing Large-scale Software Systems

## Quality Attributes & Trade-offs

Jan 27, 2024

# Learning Goals

- Describe different types of quality attributes (QAs)
- Determine QAs that are relevant for a system
- Determine metrics for measuring QAs
- Specify quality attribute requirements using scenarios
- Identify trade-offs among different QAs and compare design options with respect to those trade-offs

# Course Roadmap

- Foundational concepts & techniques for design
  - Domain & design modeling, **quality attributes & trade-offs**, generating design alternatives, design review, design processes
- Designing for quality attributes
  - Design for change, testability, interoperability, reuse, scalability, robustness, security, usability, AI, ethics

# Today's Questions

- What are quality attributes (QAs), and why should I care?
- How do I determine what QAs are relevant for my system?
- How do I measure and specify QAs?
- What are trade-offs among different QAs, and how do I make the right trade-offs?

# Quality Attributes (QAs)

- **Measurable** and **testable** properties of a product that are used to indicate how well it functions
- **Examples**
  - Reliability
  - Availability
  - Performance
  - Scalability
  - Robustness
  - Safety
  - Security
  - Extensibility
  - Maintainability
  - Usability, and many others...

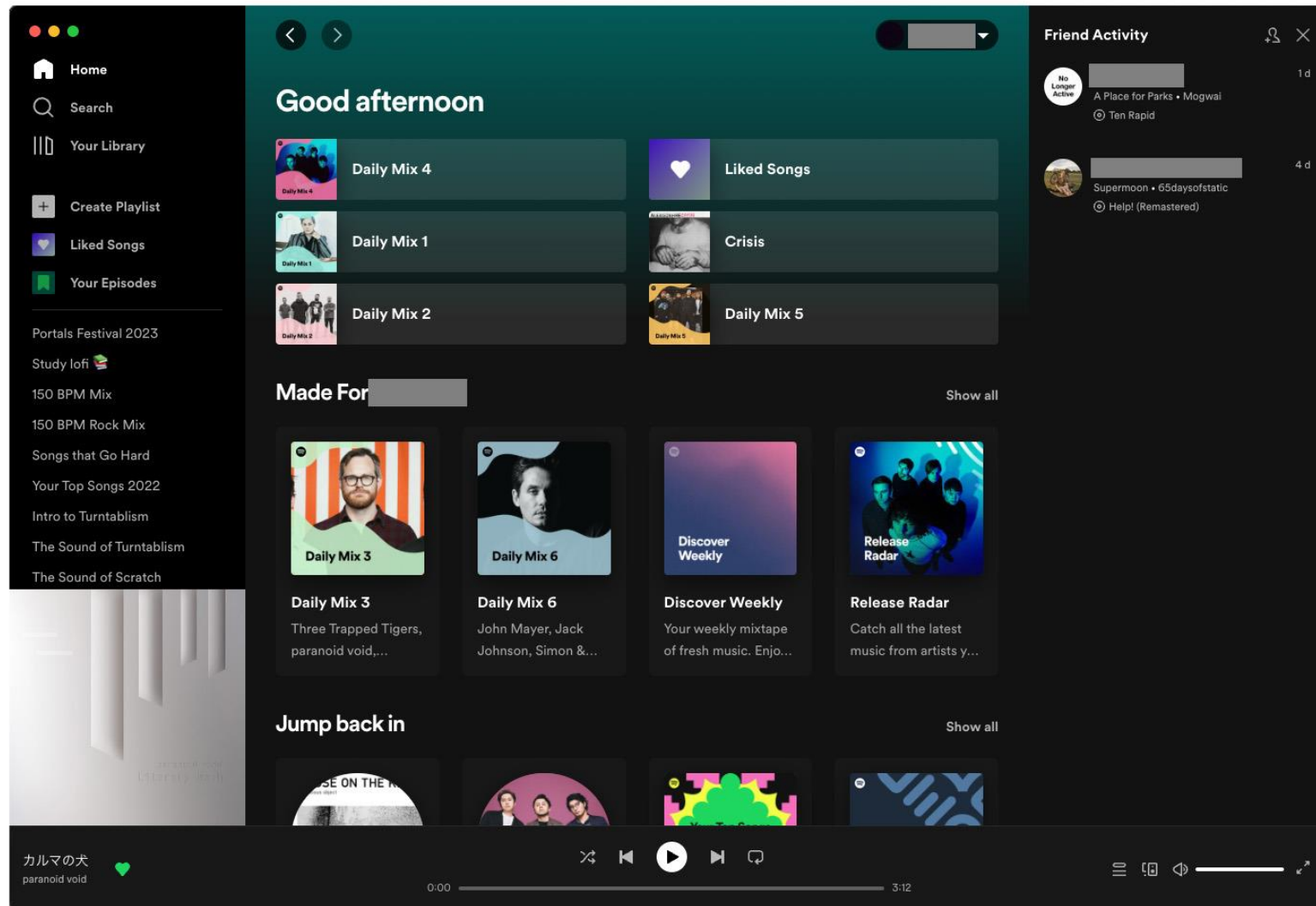
# Where do QAs come from?

- Stakeholder needs & incentives!
- **Guiding question:** Who are the most important stakeholders, and what qualities of my product do they care most about?
  - **Stakeholders:** End users, customers, investors, government regulators, integrators, developers, etc.,

# Stakeholders? Relevant QAs?



# Stakeholders? Relevant QAs?



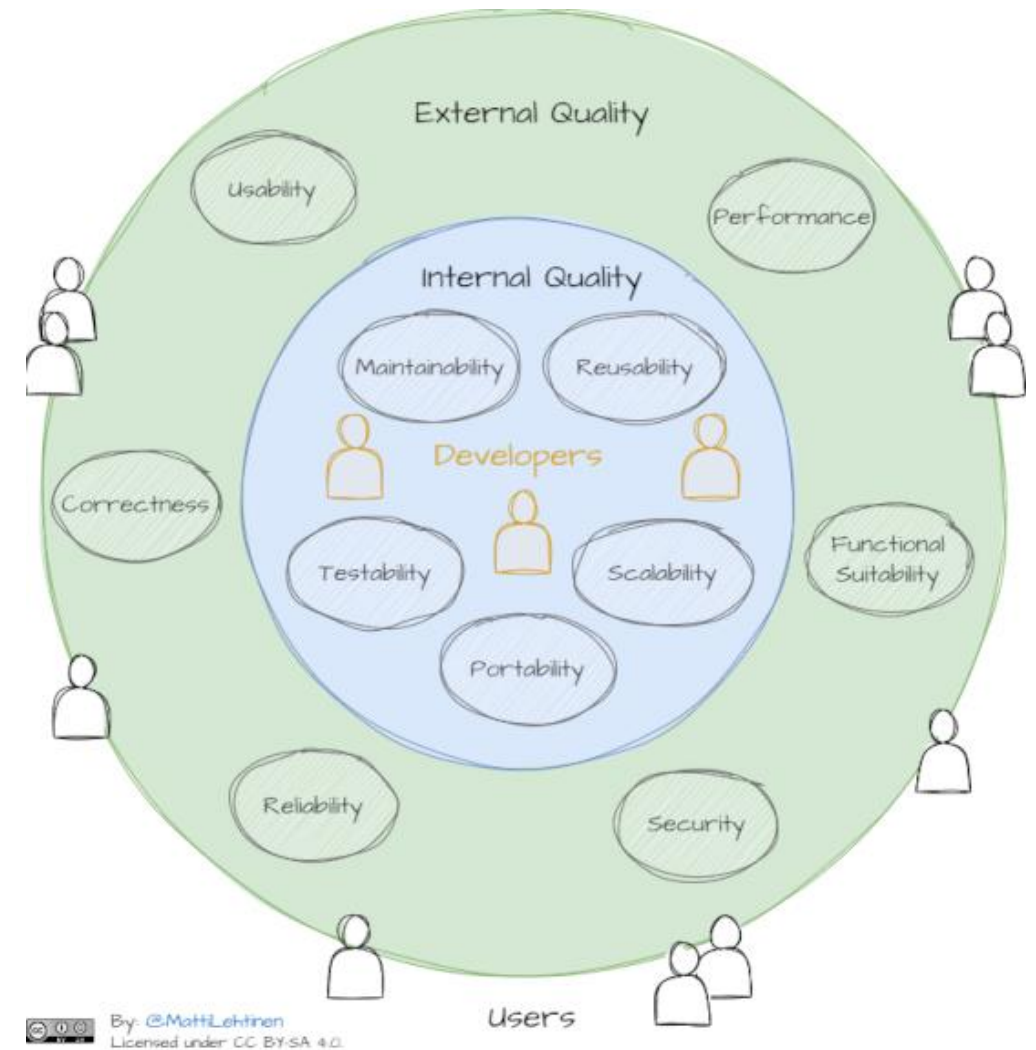


# Stakeholders? Relevant QAs?



# External vs. Internal Quality Attributes

- **External QAs:** Qualities that are visible to the stakeholders
- Common tendency is to focus on external QAs only
- But **internal QAs** also matter! When neglected:
  - Increase in developer effort
  - Increase in development costs & time
  - Decrease in code quality, which also likely leads to decrease in external QAs
  - Remember: **Developers are also stakeholders of the software that you create!**



# QAs are “Load-Bearing Walls”

- QAs are very hard to “add in later”
- Early design decisions strongly impact the qualities of a system
- QAs are often cross-cutting concerns and spread throughout a system, not localized in one part
- Improving a QA typically involves significant changes or even redesign of the system



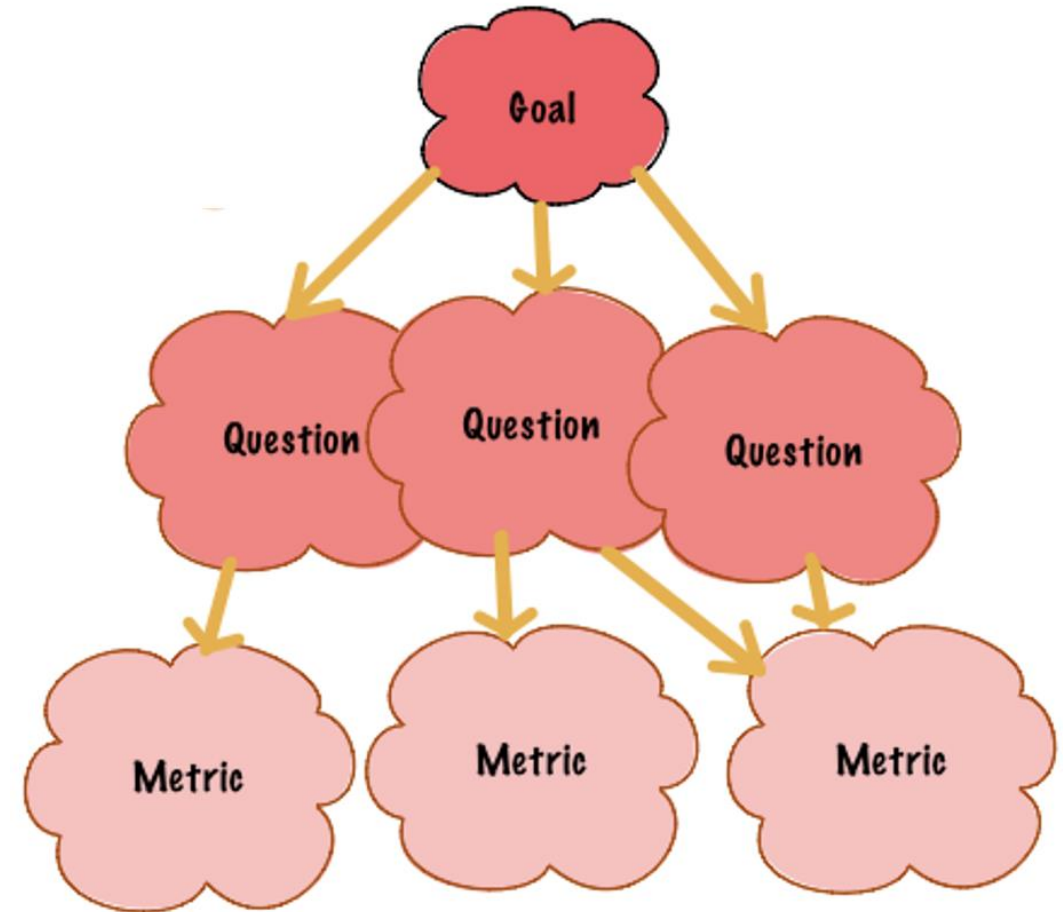
How do I measure quality attributes?

# Measuring Quality Attributes

- To test and improve desired qualities of a software product, we must be able to **measure** them
- Some QAs seem less measurable than others (security, usability vs. performance, reliability)
- Even for a single QA, different metrics make sense for different applications
  - e.g., “Performance” has different meanings for different apps
- How do we come up with suitable metrics for a QA?

# Goal-Question-Metric (GQM)

- A method for identifying metrics for software quality
- **Goal**: A high-level goal for evaluating the quality of a software artifact
- **Questions** for characterizing the artifact with respect to the quality
- **Metrics** for answering the above questions



*The goal question metric approach.* Basili, Caldiera & Rombach (1994).

# GQM Example

**GOAL:**

Evaluate effectiveness of programming tool 'X'

**QUESTIONS:**

Who is using  
tool X?

What is coder  
productivity?

What is code  
quality?

**METRICS:**

Proportion  
of coders

Experience of  
coders:  
- years coding  
- month using X

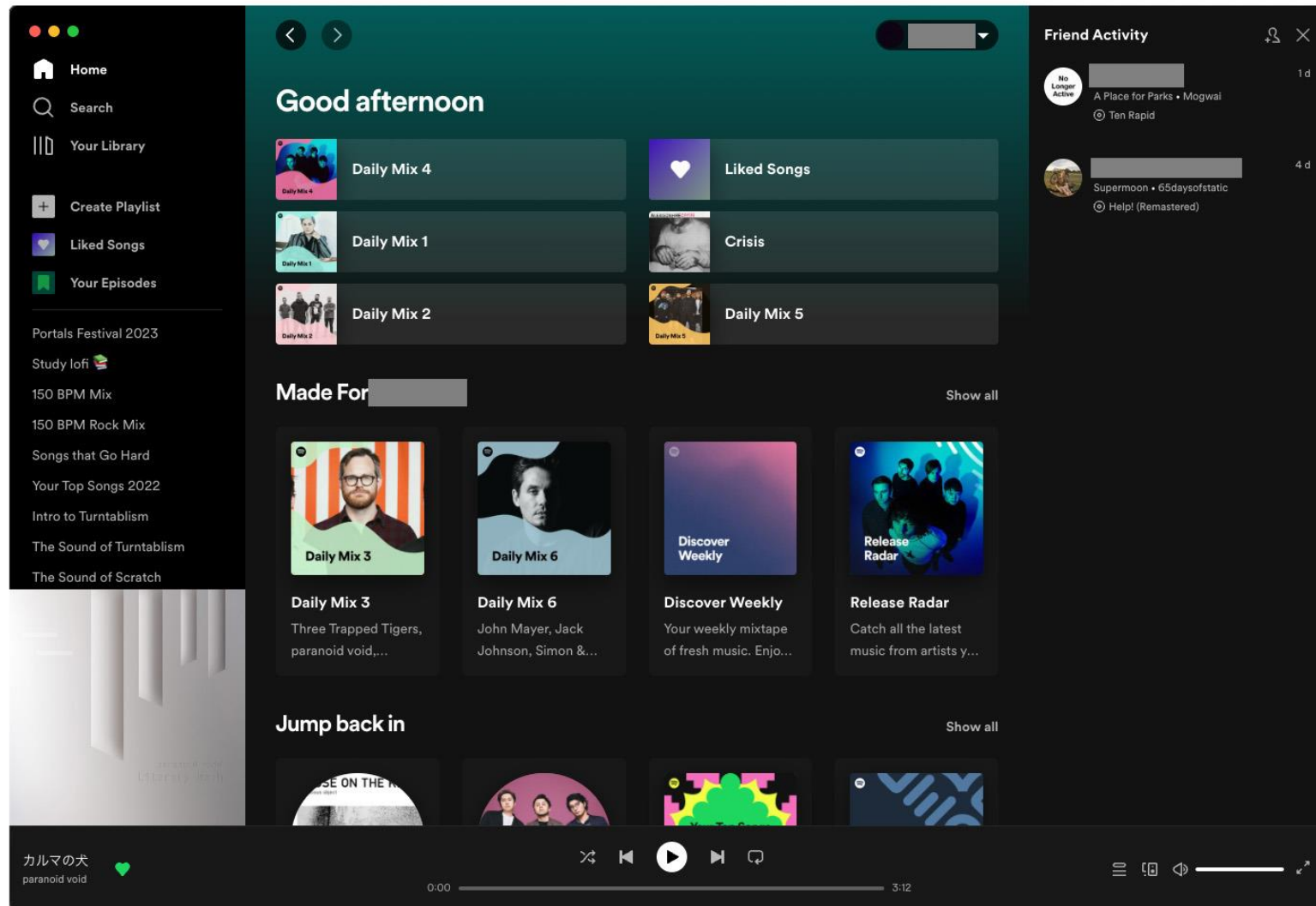
Code size  
- LOC  
- Function points  
- ...

Effort

Defects ...



# Today's Case Study: Spotify





# GQM Example: Performance

**Goal**

Evaluate the performance of the  
Spotify mobile app

**Questions**

??

??

??

**Metrics**

??

??

??

# GQM Example: Performance

## Goal

Evaluate the performance of the Spotify mobile app

## Questions

How fast does the app load a request song?

How long does a search query take?

How much mobile resources does it consume?

## Metrics

Average song load time (ms)

Average query response time (ms)

Cache usage (GB)

Memory usage (MB)

# GQM Example: User Satisfaction

**Goal**

Evaluate the user satisfaction of  
the Spotify mobile app

**Questions**

??

??

**Metrics**

??

??

# GQM Example: User Satisfaction

**Goal**

Evaluate the user satisfaction of the Spotify mobile app

**Questions**

How much do the users use the app?

How satisfied are the users with the app?

**Metrics**

Number of concurrent users

Average time spent on app (min)

Average app ratings (1-5)

% users returning to app

# QA Metrics: Tips and Caveats

- Choose metrics that are observable & testable
  - “Likelihood of a security attack: Impossible to observe in general
  - “Development time”: Too difficult to estimate accurately, even for repeated projects
- Reuse existing metrics where possible; don’t invent your own
  - We will cover these in the following lectures
- Metrics are often *proxies* for the underlying concept being measured, and can sometimes be misleading
  - High “User rating” or “number of user accounts” does not necessarily mean “usable”
  - “Lines of code” does not necessarily indicate “programmer productivity”

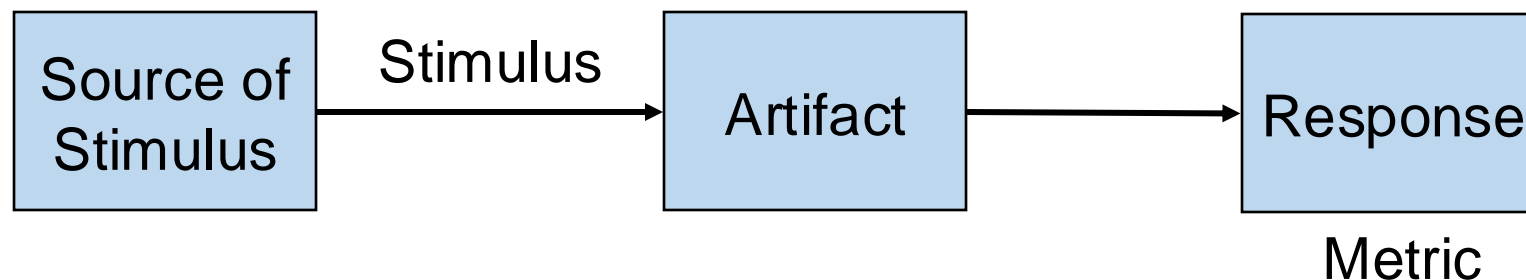
How do I specify quality attributes requirements?

# Specifying QA Requirements

- A **QA requirement** describes the level of QA that the system is expected to achieve
- Metrics alone are often not enough for specifying QA requirements precisely
- For a specific QA & a metric, the system may be required to achieve different levels of quality depending on the **context**
  - **Context:** Specific environmental conditions, system state, user inputs, or use cases
- **Scenarios** are one way to describe different contexts

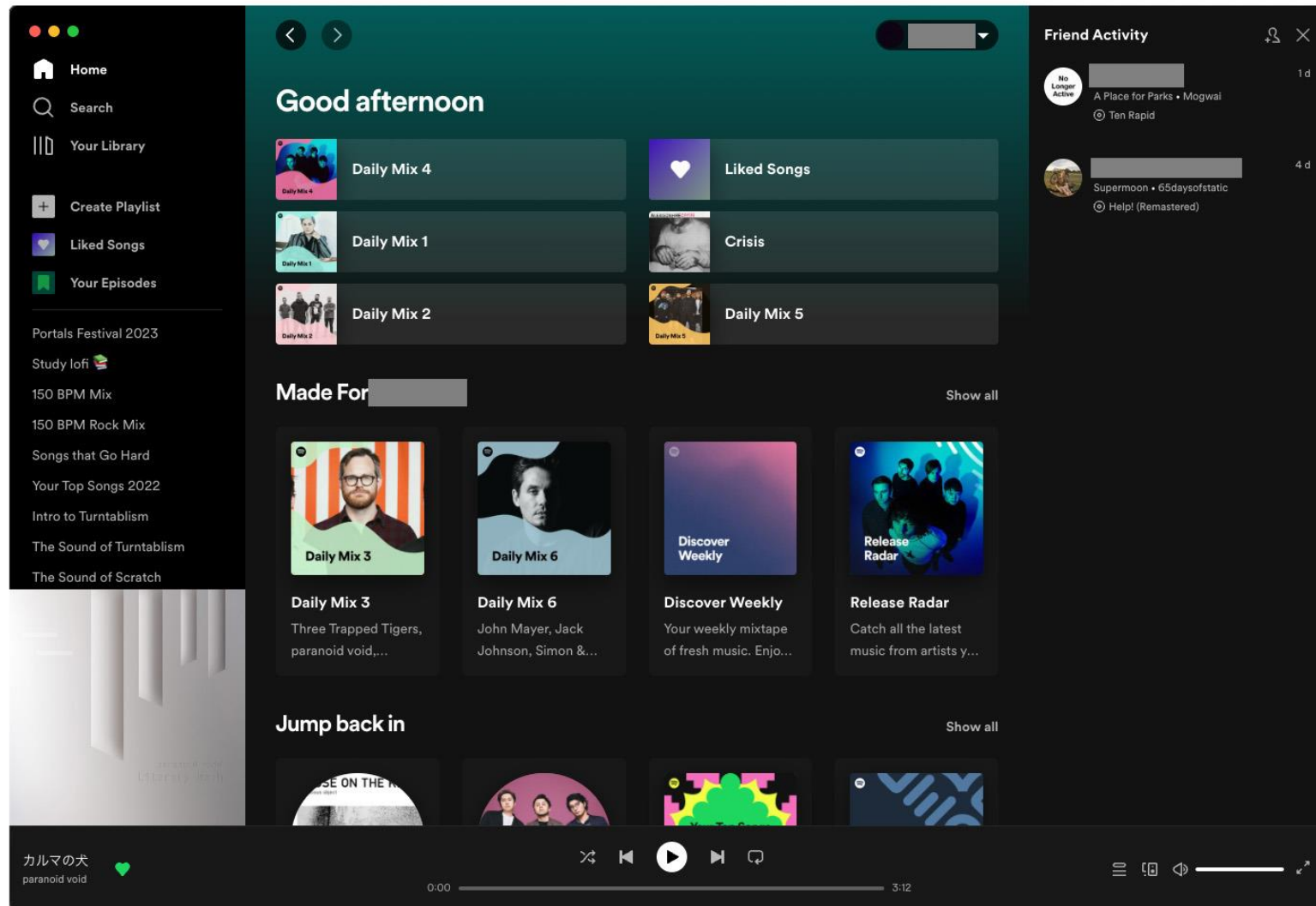
# Specifying QAs with Scenarios

- What does a **QA scenario** consist of?
  - **Artifact**: Software artifact (an app, a module, an API function...)
  - **Stimulus**: An input event or condition that causes the artifact to produce a response
  - **Source of stimulus**: The entity that generates the stimulus (e.g., user, another application, the API client...)
  - **Response**: What artifact does, given the stimulus, with a **metric** to define what a successful response is (e.g., time)





# QA Scenarios: Examples



# QA Scenarios: Examples

- *When the user requests to play a song, the app must load and play the song within the next 500 ms.*
  - **Artifact:** Spotify app
  - **Source of stimulus:** The app user
  - **Stimulus:** Request to play a song
  - **Response:** Load & play the song on the user's phone
  - **Metric:** Song latency (500 ms)

# QA Scenarios: Good or bad examples?

- **(Performance)** *When the user requests to play a song, the app must load and play the song within the next 500 ms.*
- **(Performance)** *When the user requests to play one of the current top 100 songs, the app must load and play the song within next 1000 ms.*
- **(Availability)** *If the user's phone loses the Internet connection, the app must continue to play the current song.*
- **(Scalability)** *The system must be able to handle 200 million active users at the same time.*
- **(Usability)** *The next released version of the app must maintain or increase the user satisfaction rating on the Google Play store.*

# QA Scenarios: Tips and Caveats

- In general, there are too many system scenarios to enumerate
- Focus on scenarios that represent the most common use cases
- For certain qualities like robustness, security, and reliability, also consider edge cases
  - Unexpected/malicious user inputs, server failures as stimulus
  - In later lectures, we will discuss methods for coming up with some of these scenarios
- Even if the stimulus/source/response may seem obvious, be explicit about them
  - There's always a risk of ambiguity/misinterpretation (e.g., what does the “user” mean?)

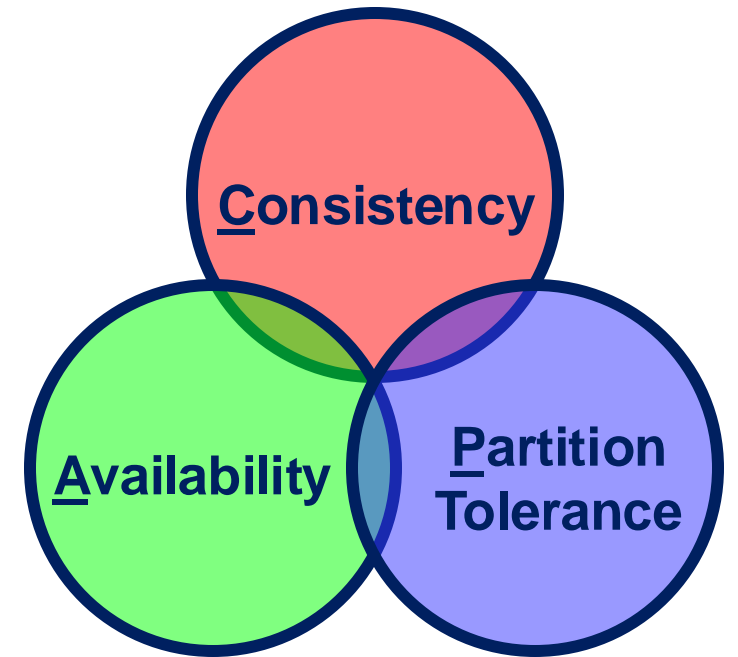
How do I make trade-offs among QAs?

# QA Trade-offs

- In a typical problem domain, there are multiple desirable QAs
- In most cases, it is difficult to design & implement a software solution that achieves all of these QAs
  - Constraints and assumptions imposed by the problem space
  - Conflicts among different stakeholders' needs
  - Limited resources and time for development
  - Inherent conflicts among certain types of QAs
- Often, **trade-offs** need to be made among QAs, to obtain certain qualities while sacrificing others

# QA Trade-off Example: Distributed Systems

- **Consistency:** Clients always read the latest data
- **Availability:** Client requests always result in a response
- **Partition tolerance:** System continues to operate despite network failures
- **CAP theorem:** Choose two out of three
  - e.g., if a network failure occurs (and system tolerates it), choose consistency or availability
  - (Generally agreed to hold in practice, although not without some controversies)



“Towards robust distributed systems”, Eric Brewer (2000)

# QA Trade-offs: Other Examples

- **Security vs. usability**

- Two-factor authentications is more secure but harder to use
- Remembering a long password is harder, but also more secure

- **Security vs. performance**

- Encrypting and decrypting data slows down the system while making it more secure

- **Performance vs. reliability**

- TCP (slow but reliable) vs. UDP (fast but unreliable)
- Use of redundancy (backup servers) increase reliability, but also introduces performance overhead (must keep the servers consistent)



# QA Synergies: Examples

- Not all QA interactions are negative! QAs can also amplify each other under certain scenarios
- **Performance & usability**
  - Faster response times make it easier to use interactive systems
- **Performance & security**
  - Faster intrusion detection can keep the system more secure
- **Performance & reliability**
  - Components with message queues lose fewer messages if they process messages faster
  - Highly reliable connections do not require many retries, resulting in faster average case delivery

# Trade-off Analysis: Example

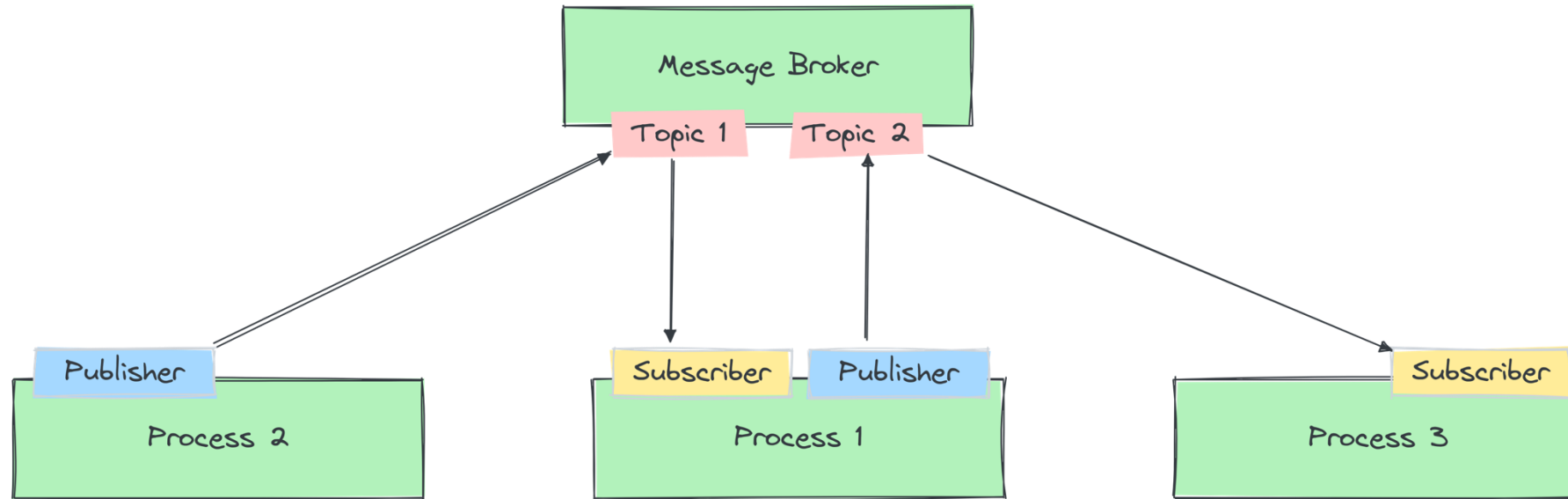
- Consider two models of message communication between processes: **Point-to-point** and **publish-subscribe**

# Point-to-Point Communication



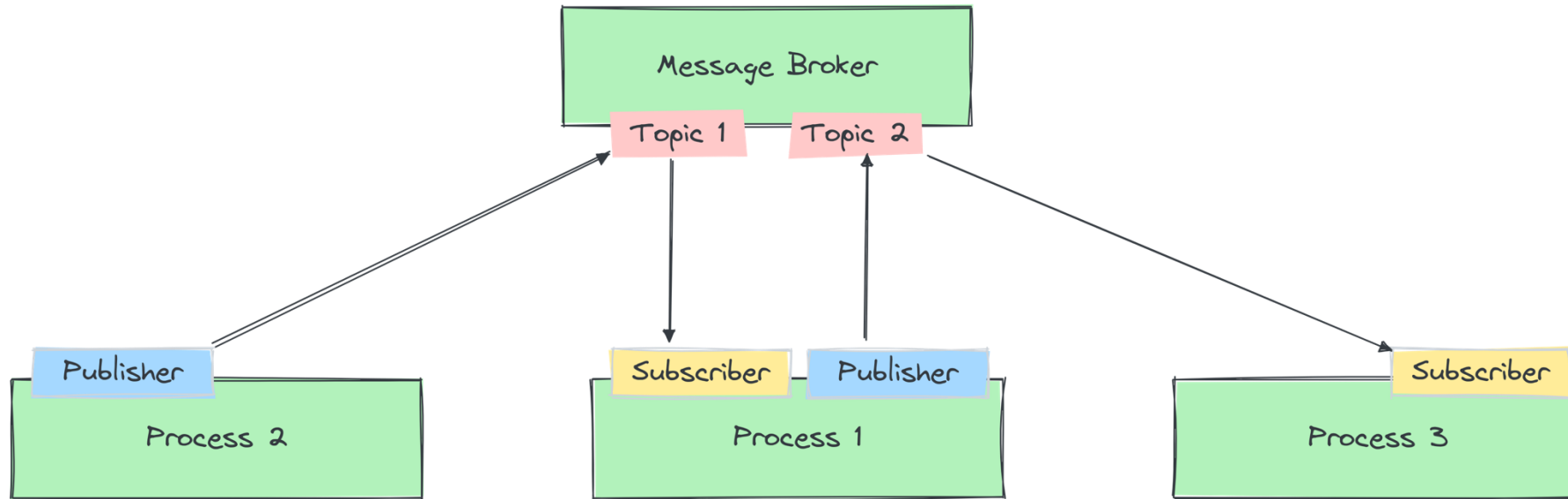
- A process directly communicates to another process
- e.g., “Client-server” or “request-response” model

# Publish-Subscribe Communication



- Each process publishes or subscribes to a topic
- When a publisher sends a message, every subscriber receives it through a message broker

# Publish-Subscribe Communication



- Each message reaches multiple subscribers, not just one
- Decouples publishers from subscribers; can add new publishers/subscribers without affecting each other
- Topics can be dynamically added at run-time

# Discussion: Pub-Sub vs. Point-to-Point

- Compare Pub-Sub and Point-to-Point models with respect to the following quality attributes:
  - **Performance**: How quickly are messages delivered from a sender to a receiver?
  - **Scalability**: How many additional concurrent messages can the system handle?
  - **Extensibility**: How much effort does it involve adding new types of messages?
  - **Robustness**: How well does the system handle component failures or unexpected external events?

# Trade-off Analysis with Decision Matrix

- **Decision matrix:** Summarizes trade-offs among design options with respect to different quality attributes

| QAs                  | Option: Point-to-Point  | Option: Publish-Subscribe   |
|----------------------|---|---|
| <b>Performance</b>   | Direct messaging; stronger guarantee on delivery time           | Message delivery time affected by number of subscribers                             |
| <b>Scalability</b>   | Limited support for large-scale messaging                       | Support sending messages to an arbitrary number of subscribers                      |
| <b>Extensibility</b> | Adding new message types involve changes to sender and receiver | Can dynamically add topics, publishers, and subscribers without changing the others |
| <b>Robustness</b>    | A failure in the receiver disrupts the senders                  | A failure in the broken disrupts all publishers & subscribers                       |

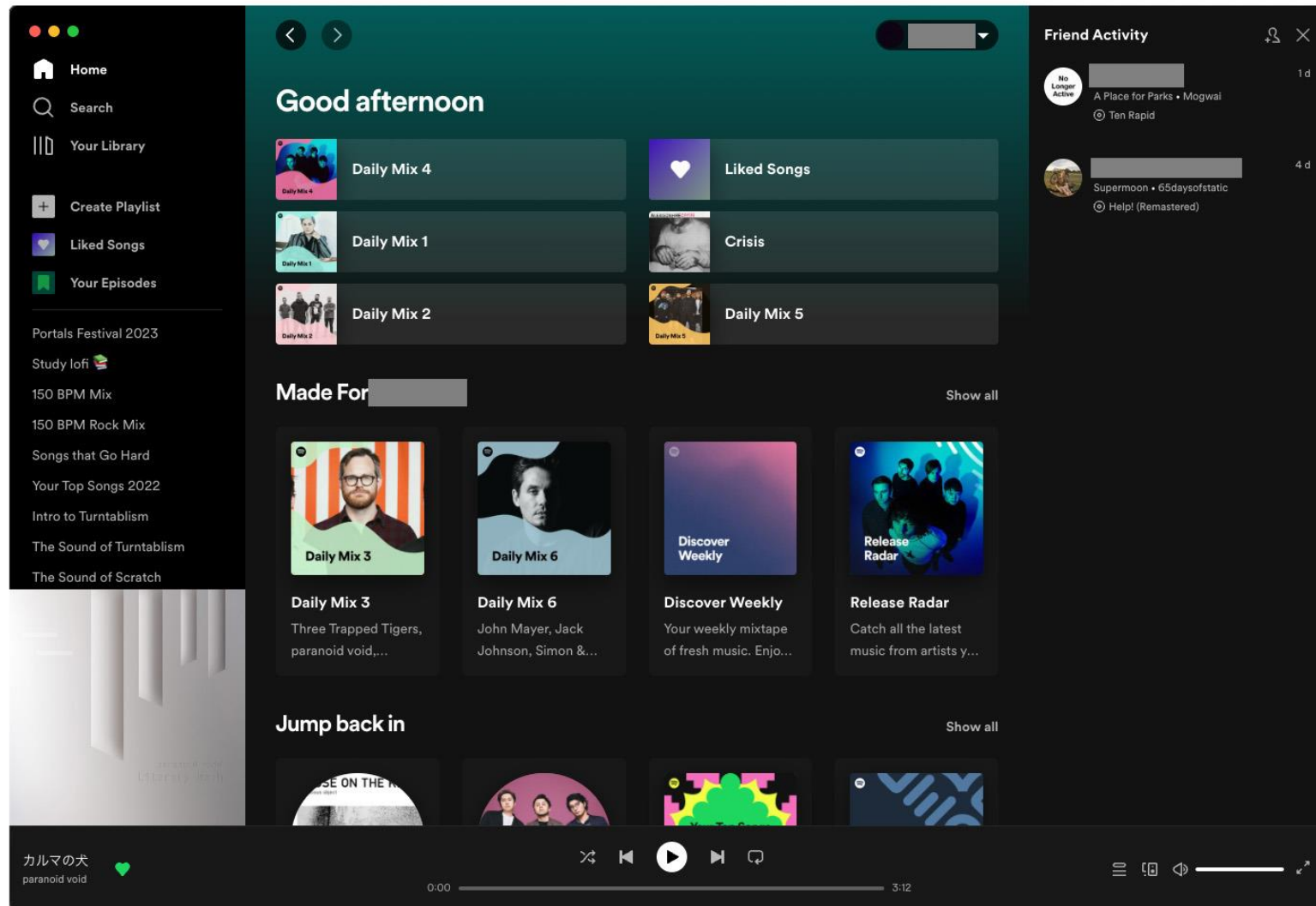
# Making Trade-off Decisions

- Which of the options to select? Depends on the **context!**
- Identify QAs that are relevant to a specific use case scenario

| QAs                  | Option: Point-to-Point  | Option: Publish-Subscribe   |
|----------------------|---|---|
| <b>Performance</b>   | Direct messaging; stronger guarantee on delivery time           | Message delivery time affected by number of subscribers                             |
| <b>Scalability</b>   | Limited support for large-scale messaging                       | Support sending messages to an arbitrary number of subscribers                      |
| <b>Extensibility</b> | Adding new message types involve changes to sender and receiver | Can dynamically add topics, publishers, and subscribers without changing the others |
| <b>Robustness</b>    | A failure in the receiver disrupts the senders                  | A failure in the broken disrupts all publishers & subscribers                       |



# Back to Spotify



# Making Trade-off Decisions

- **Q. In Spotify, that are scenarios where the point-to-point is more suitable? Publish-subscribe model?**

| QAs                  | Option: Point-to-Point  | Option: Publish-Subscribe   |
|----------------------|---|---|
| <b>Performance</b>   | Direct messaging; stronger guarantee on delivery time           | Message delivery time affected by number of subscribers                             |
| <b>Scalability</b>   | Limited support for large-scale messaging                       | Support sending messages to an arbitrary number of subscribers                      |
| <b>Extensibility</b> | Adding new message types involve changes to sender and receiver | Can dynamically add topics, publishers, and subscribers without changing the others |
| <b>Robustness</b>    | A failure in the receiver disrupts the senders                  | A failure in the broken disrupts all publishers & subscribers                       |

# Costs in Trade-off Analysis

- Every design decision regarding a QA has some development costs associated with it
  - Achieving security involves adding encryption, storing secrets in databases, hiring a security expert/tester, etc.,
  - Achieving scalability involves purchasing more servers, implementing distributed protocols to keep the servers consistent...
- In this class, we will (mostly) ignore the issues related to costs
- In practice, costs should be considered as an additional dimension in the decision matrix along with other QAs



# Quality Attributes: Takeaways

- Functionality is not the only concern of software design
- Quality attributes measure the “goodness” of a design along a certain dimension
- Quality attributes should be measurable
- Quality attribute requirements should be specified using a scenario that describes a particular system context
- Quality attributes are very hard to “add in later” and must be considered early in the design process
- Achieving all QAs may often be impossible, and thus trade-offs among them must be made

# Summary

- Exit ticket!