API DESIGN

OBJECTIVES

- Make it easy for consumers to do the right thing
- Make it possible for the provider to keep evolving

FIRST PRINCIPLES

- Design API from the perspective of the caller
- Offer what the caller wants to achieve
- Avoid:
  - Internal state names
  - Coded fields
  - Composite fields
  - Conversations
- Be suspicious of Booleans in arguments

BREAKING APIs

BREAKING APIS

API BREAKS WHEN

- Rejects a request it would have accepted before.
- Returns less than it would have before

BREAKING APIS

API BREAKS WHEN

- Rejects a request it would have accepted before.
- Returns less than it would have before

NOT BROKEN

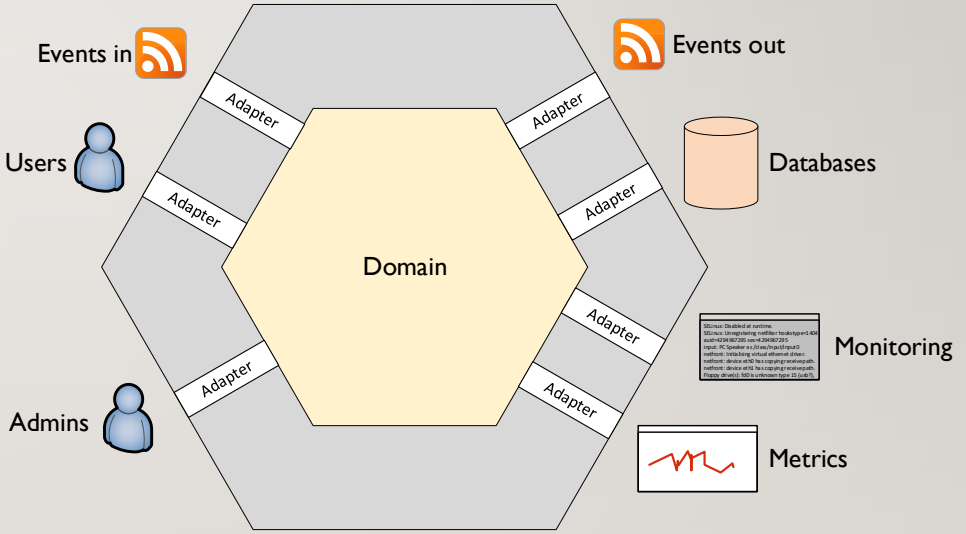
- Accepts more data in a request
- Accepts additional kinds of requests
- Rejects old requests on a new interface
- Returns more than it returned before

AVOID BREAKAGE

- Prefer adding a new interface
- Prefer adding new request types that are more restrictive
- This is easy **unless** you're use annotation based mapping on your domain classes.

USE THE HEXAGONS

- The API is **not** sticking a wire protocol on your domain entities.
- Each is an adapter from "API land" to "domain land"
- Avoid breakage by adding new adapters or changing the "outside" end of an adapter.



USE CONSUMER DRIVEN CONTRACT TESTS

- Test only the behavior needed by one consumer
- Offer those tests to the interface provider

OPEN VS CLOSED WORLD

OPEN VS CLOSED WORLD

CLOSED WORLD

- We know all the members of a set
- We can fully partition the space of values

OPEN WORLD

- We know some of the members of a set
- We may discover that some members are actually the same thing
- We may discover that our partitioning is incomplete
- We may discover some members occupy more than one partition

RELATE BACK TO IDENTIFIERS

CLOSED WORLD IDENTIFIERS

123123123

ab2798

STATE_UNDOCKED

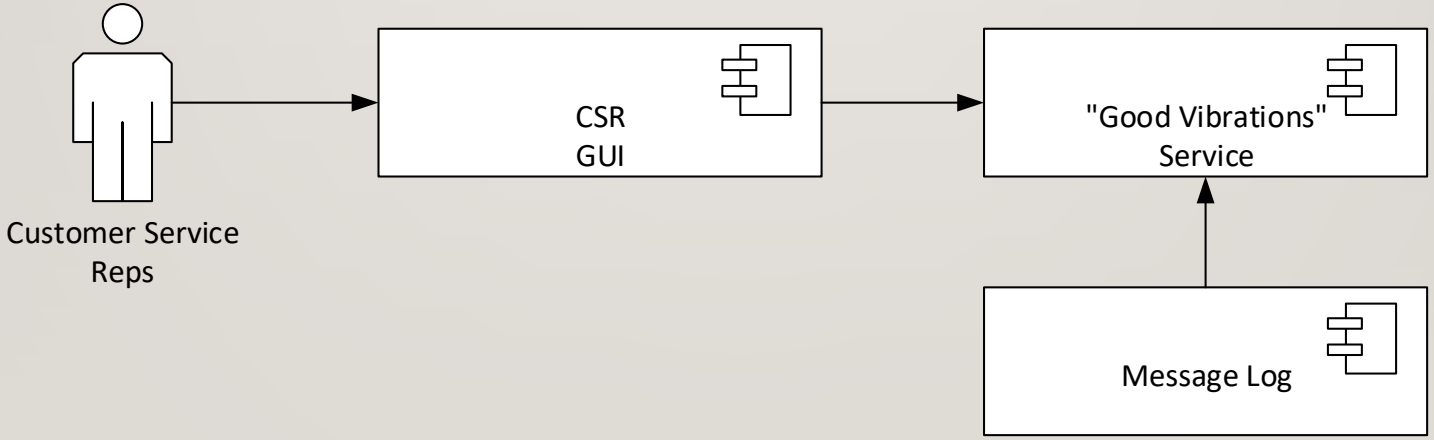
OPEN WORLD IDENTIFIERS

doi:10.1007/s10111-011-0211-6

https://example.com/people/laura

ACTIVITY: DESIGN API

- Customer service GUI needs to see all subscribers, subscriptions, vendors, payments, emails, email attempts, previous calls, … pretty much everything.
- We could couple the CSR app to every other system, but that seems brittle.
- Instead, we're going to harness the power of events.



EVENTS ARE NORMALIZED

Vendor	Customer	Event type	Event detail	Timestamp

Can view record of events for a customer when a customer calls.
Can view record of events for a vendor when a vendor calls.
Can view events from systems that just got deployed today. No changes necessary to the CSR GUI.

Your homework:

1. Define an HTTP interface to the "Good Vibrations" service for the CSR GUI to use.
2. Define queries it can accept and the response formats it will generate.
3. Think about what needs the "open world" treatment and what can use "closed world".
4. Design from the caller's perspective. What does the CSR GUI need?