Exploring your Business Domain with Event Storming and Clean Architecture

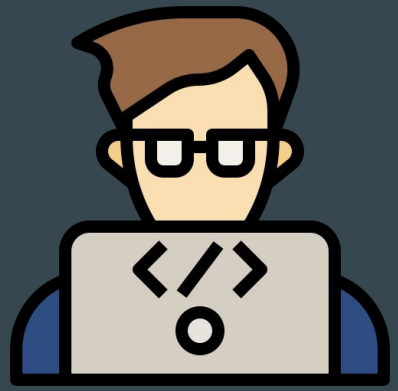
•••

applied on a fictional business

Motivation

AGILE DEVELOPMENT

DOMAIN-DD



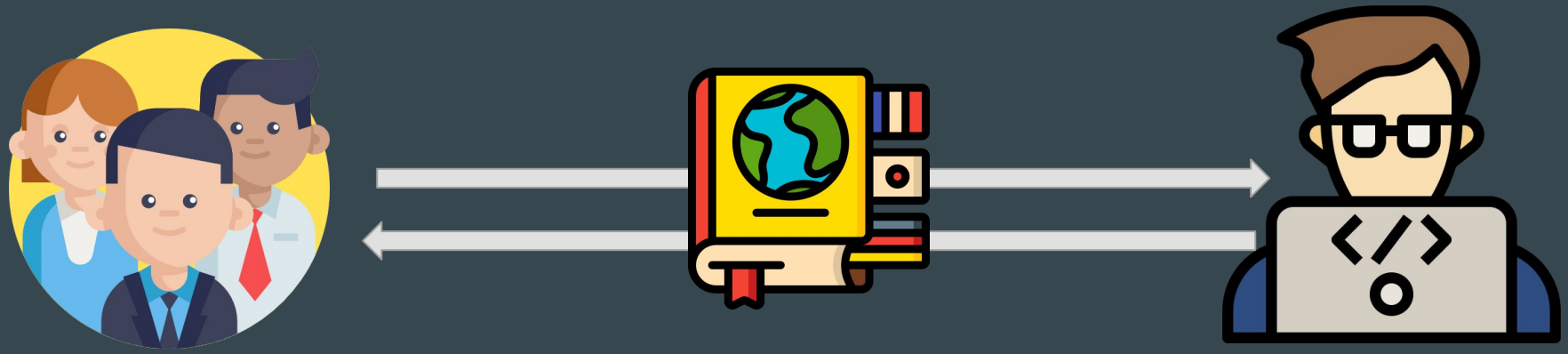
OOP

USER STORIES

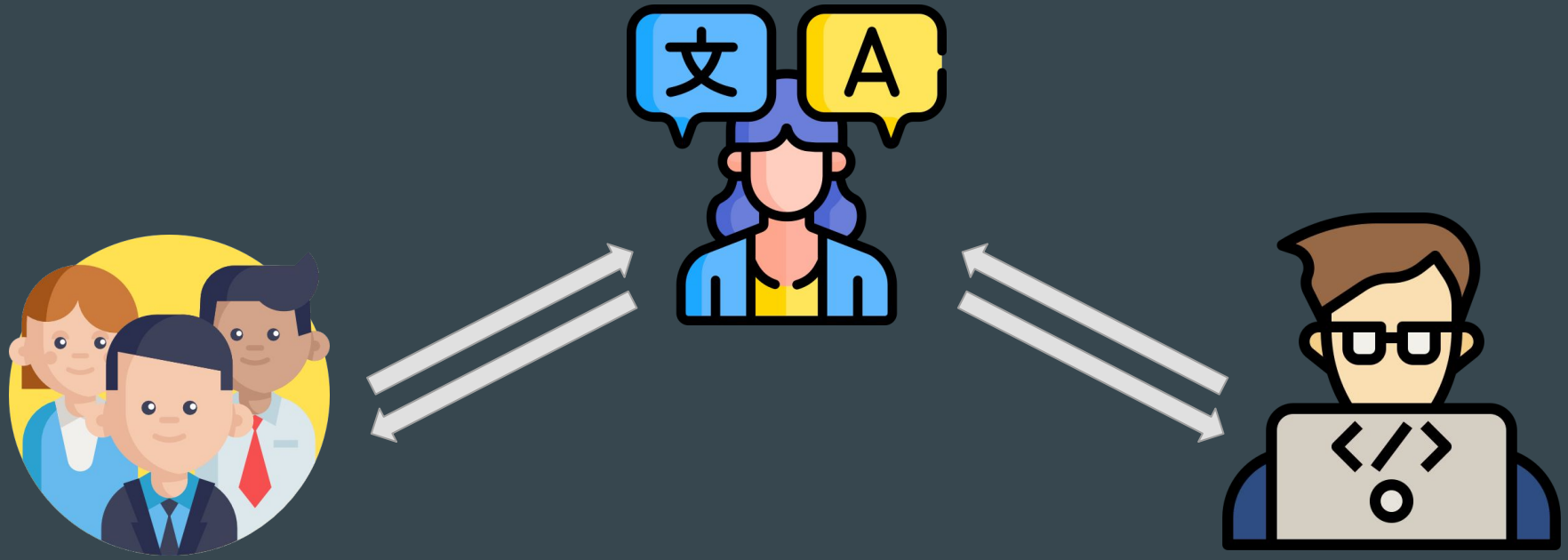
USE CASES

ACCEPTANCE TESTS

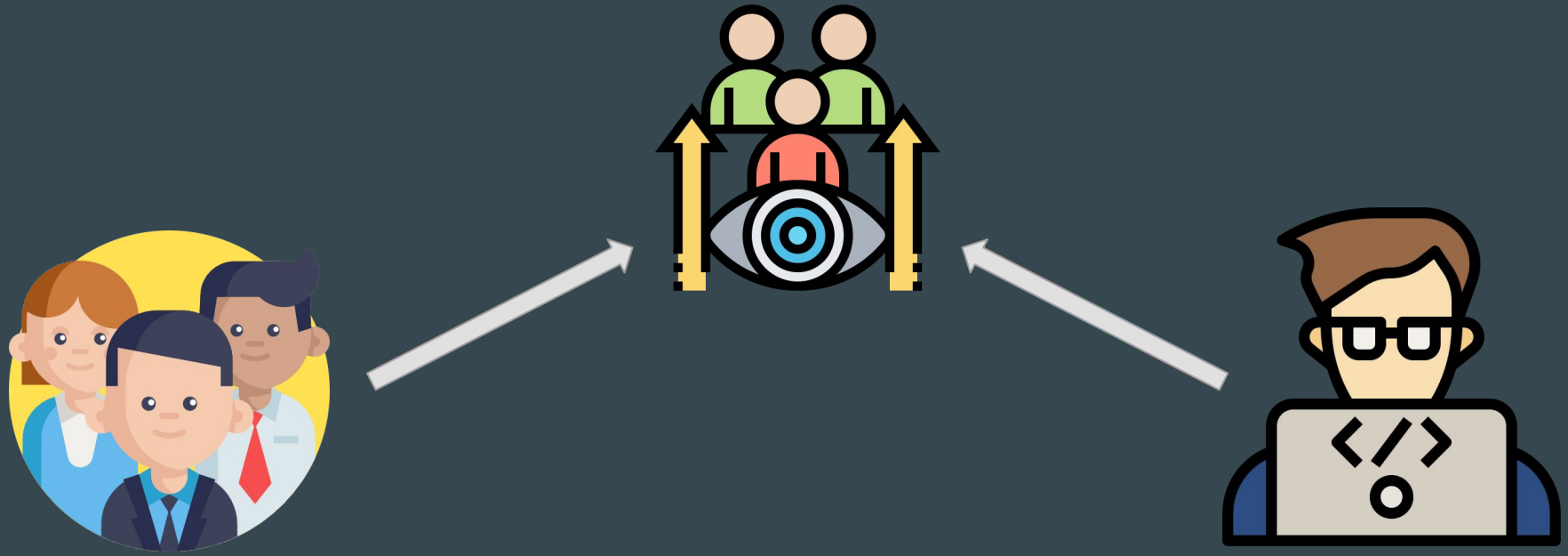
Business & devs speak different languages



Translators (BA, PO, ...) as “Middle-Person Smell”

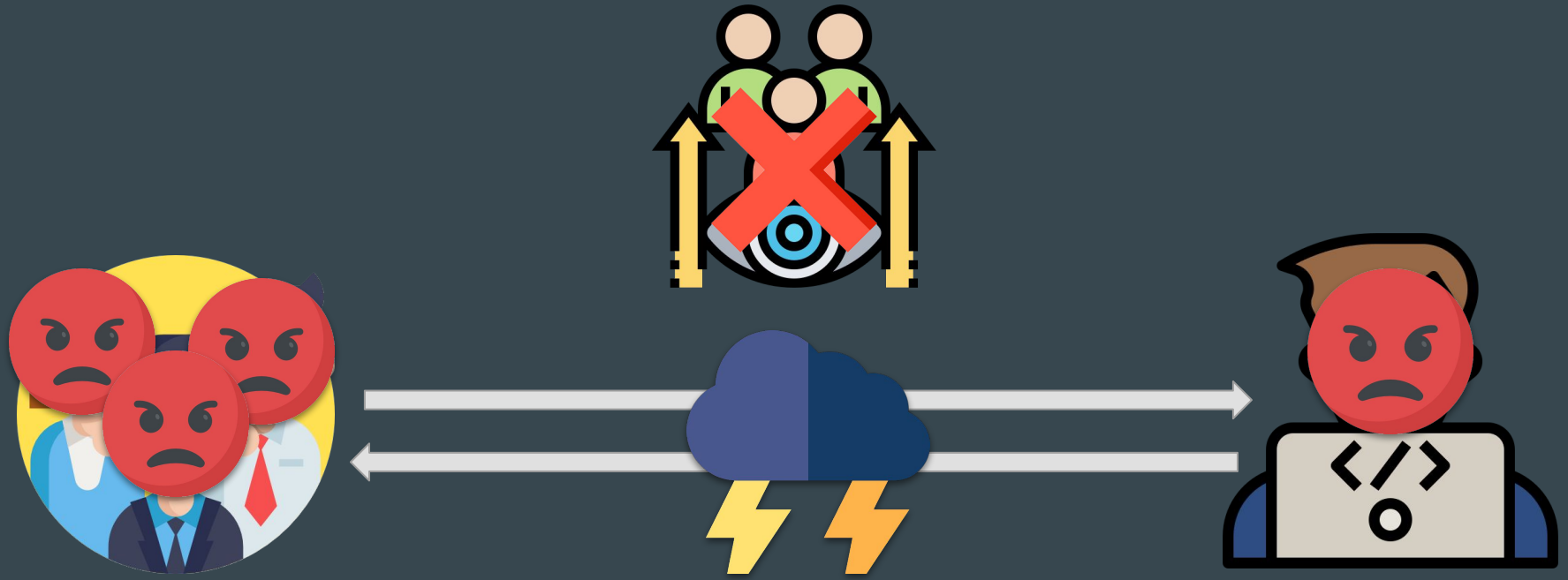


Business & Devs don’t share the same vision

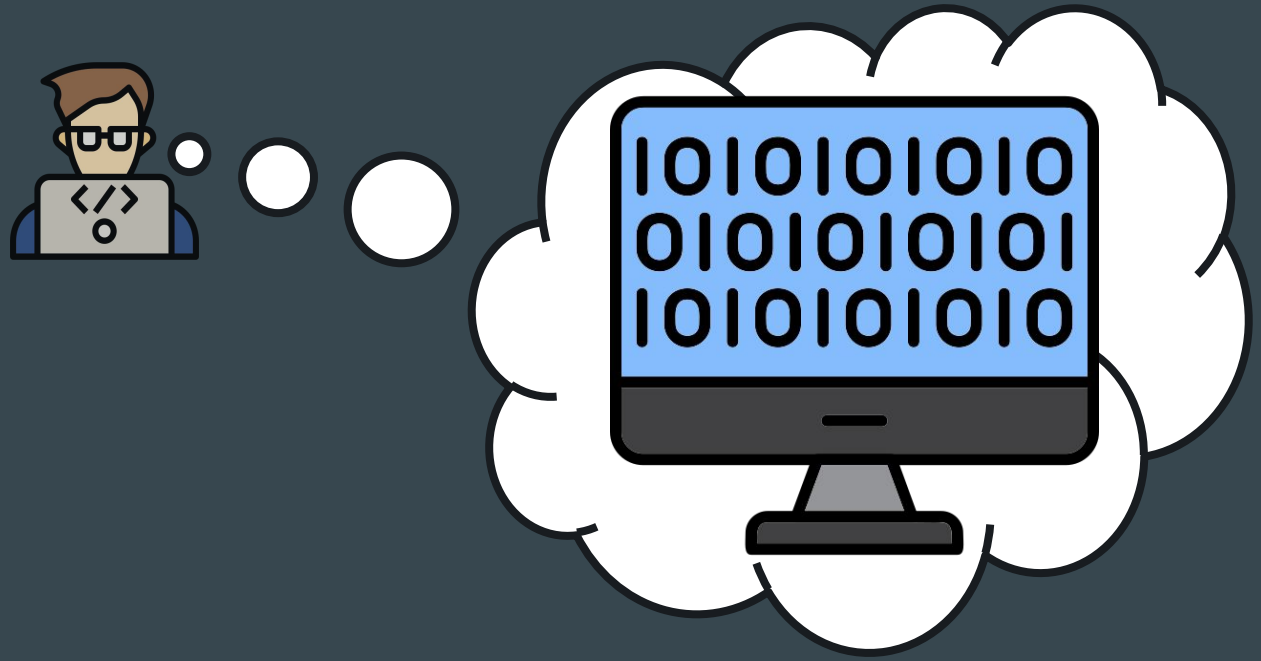


Fear leads to Command and Control

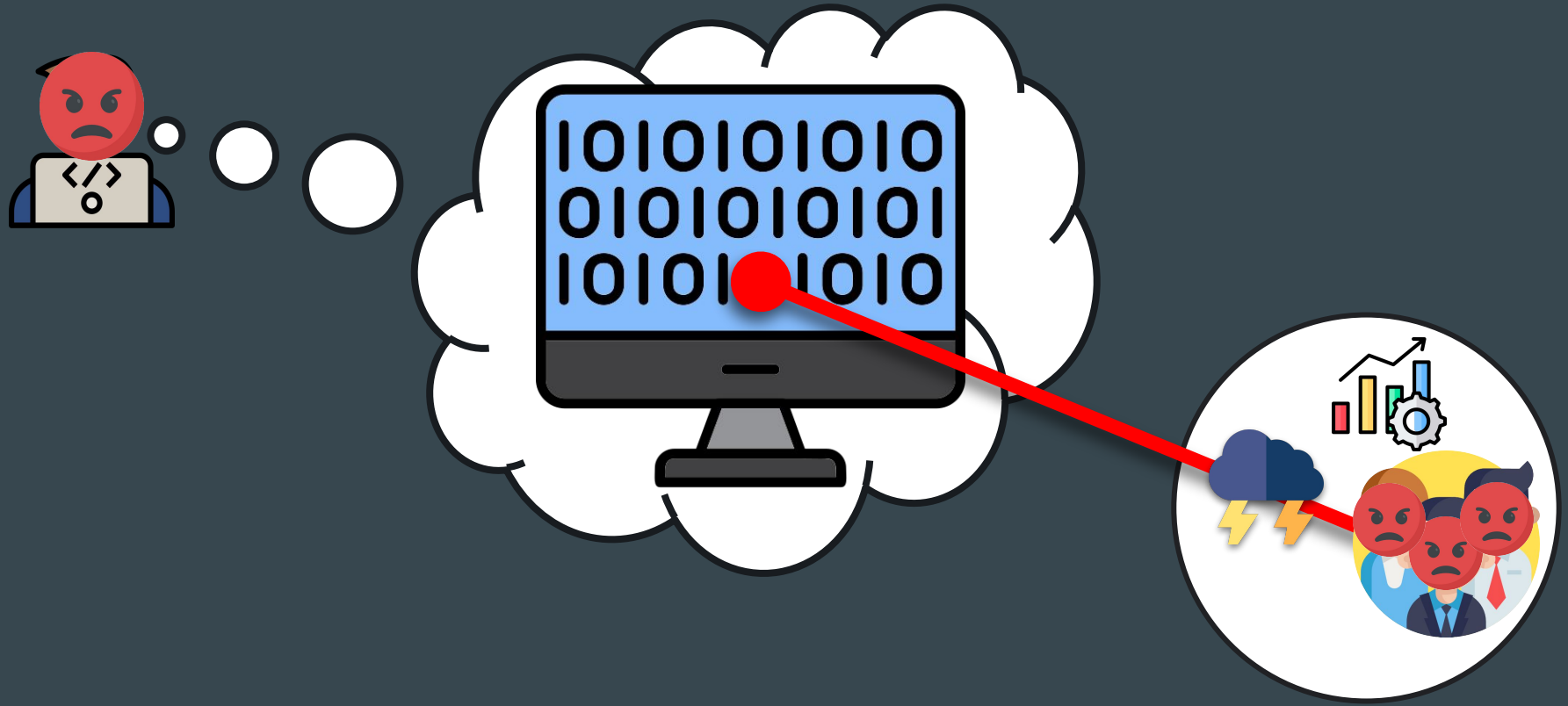
Overall distrust, dissatisfaction, risk of project failure



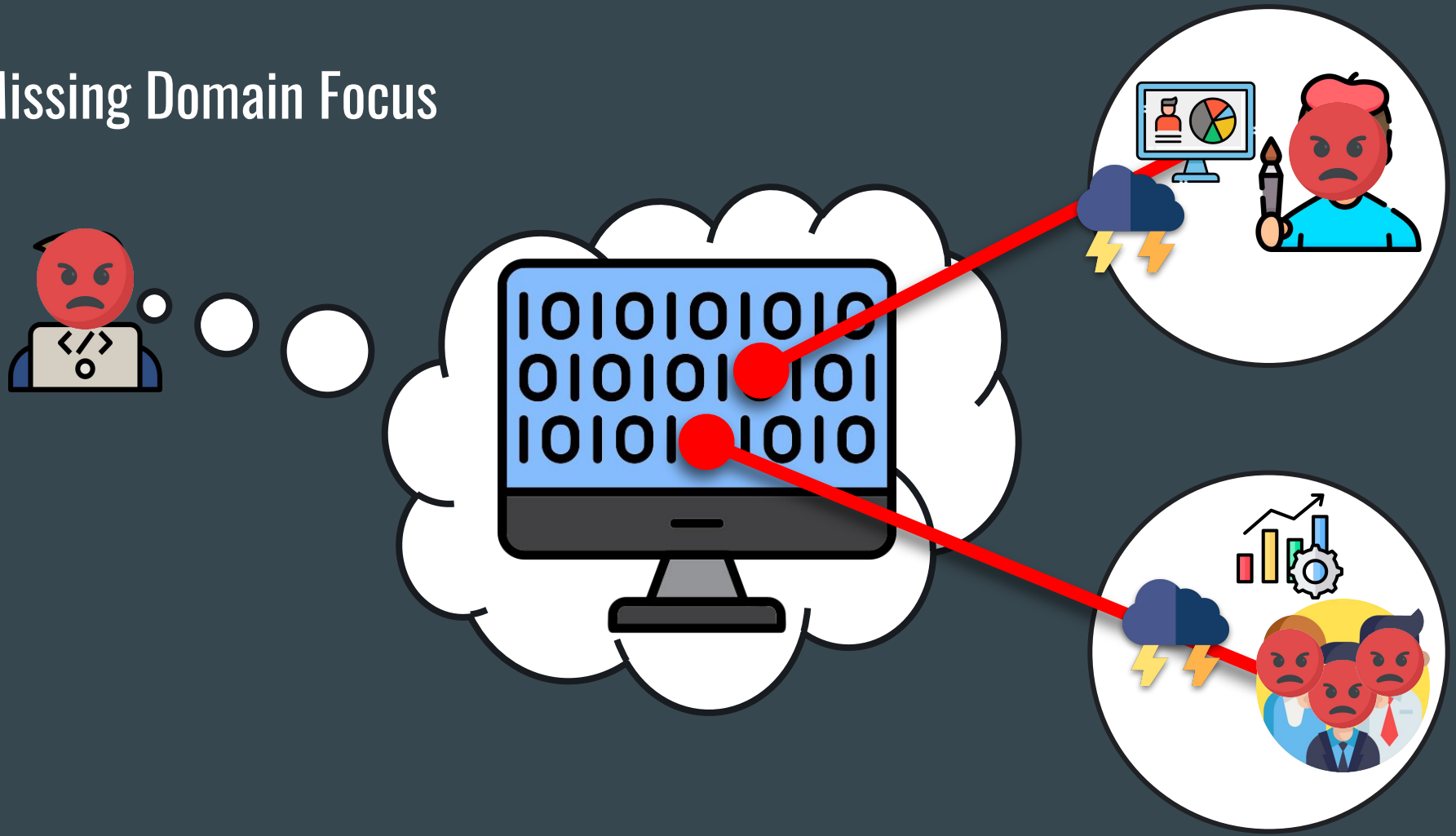
Missing Domain Focus



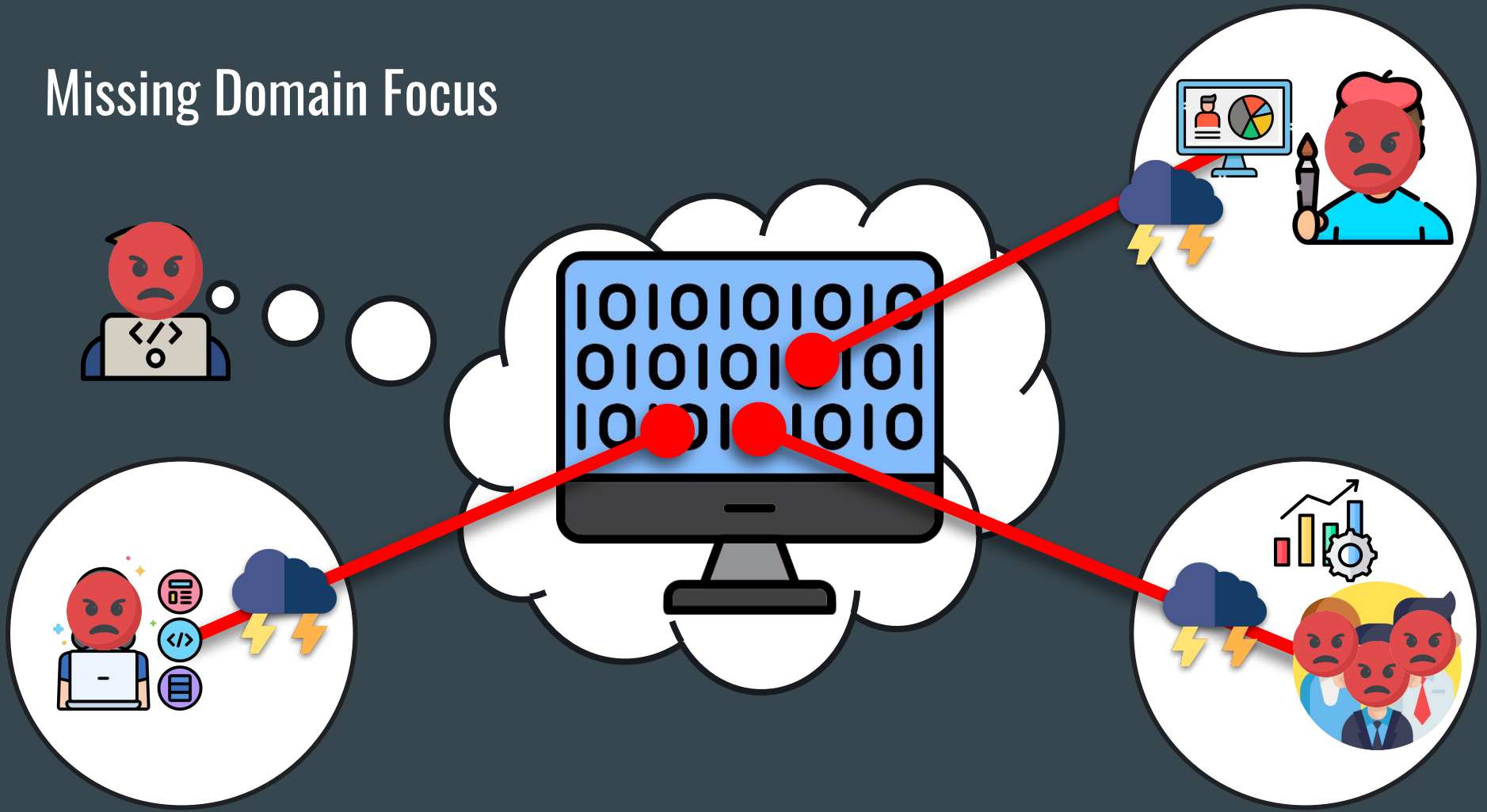
Missing Domain Focus



Missing Domain Focus

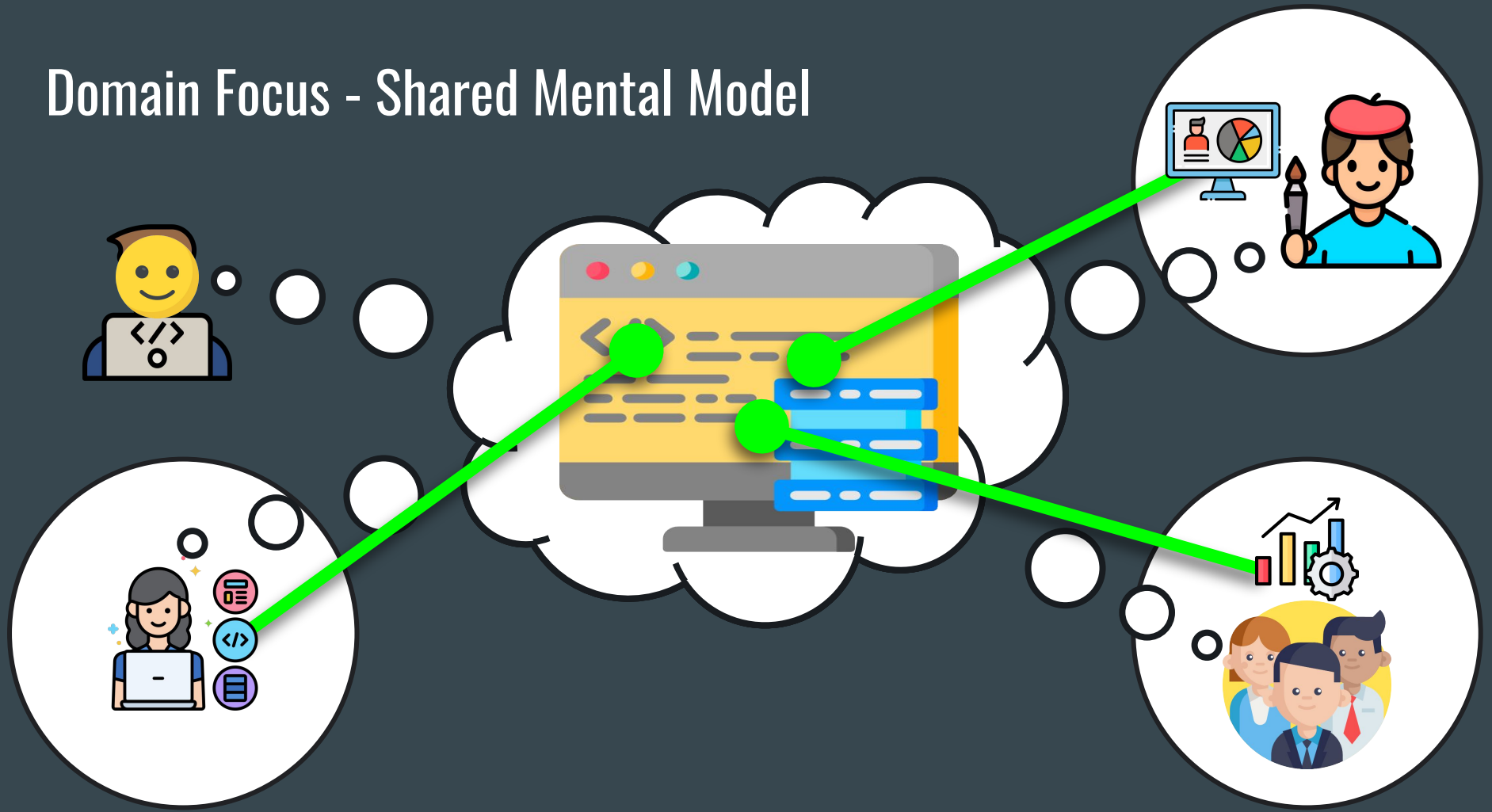


Missing Domain Focus

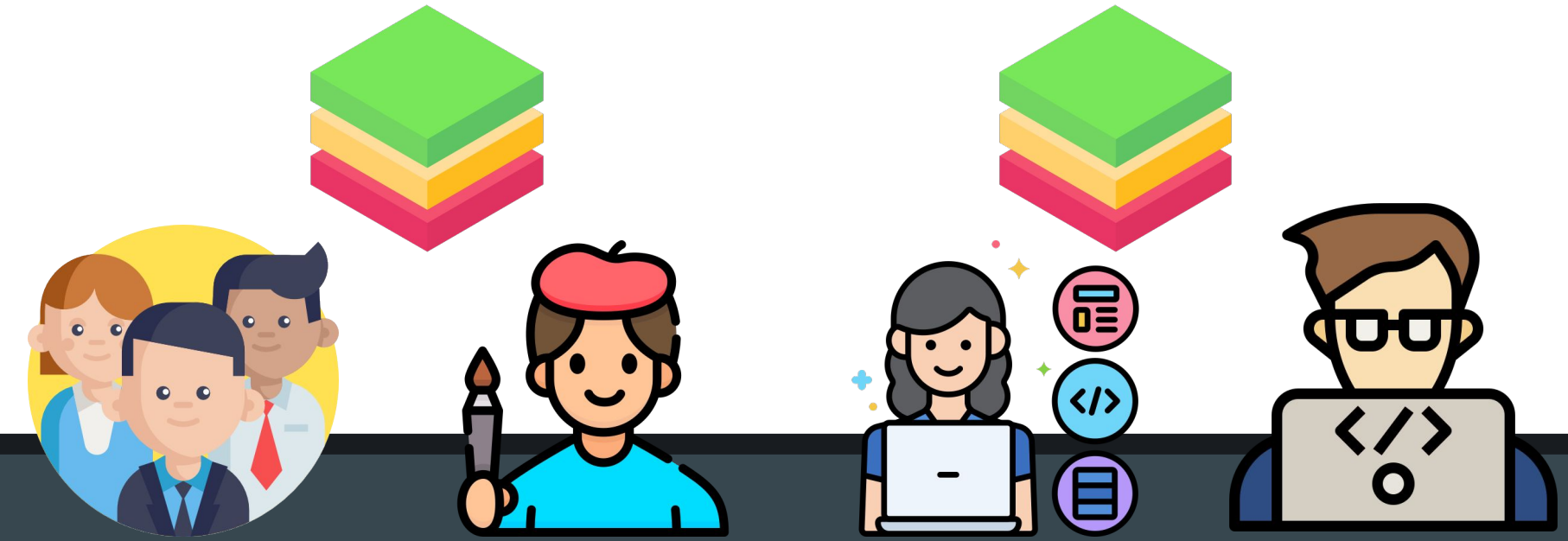


Missing Domain Focus

Domain Focus - Shared Mental Model

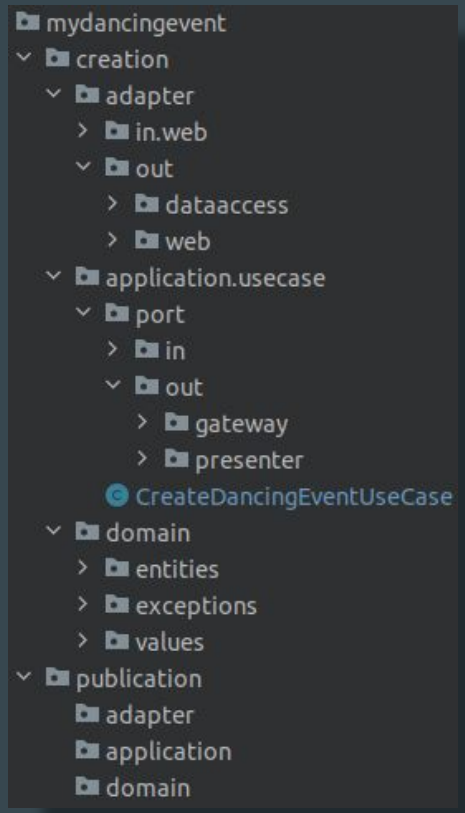


But how to get to a shared mental model?



Enjoy the show

Clean Architecture - Screaming Architecture



Clean Architecture - Testing Use Cases

```
class CreateDancingEventUseCaseShould {
    @Test
    void createADancingEvent() {...}
    @Test
    void limitTheNumberOfUnpublishedDancingEventsTo5AtATime() {...}
    @Test
    void requireAnEventOrganiserToExistInOrderToAddANewDancingEvent() {...}
```

Clean Architecture - Use Case

```java
public class CreateDancingEventUseCase implements CreateDancingEvent {
    private final UpdateUnpublishedDancingEvents updateUnpublishedDancingEvents;
    private final FetchUnpublishedDancingEvents fetchUnpublishedDancingEvents;

    @Override
    public void executeWith(CreateDancingEventInput input,
                            PresentCreateDancingEventSuccess presentSuccess,
                            PresentCreateDancingEventFailure presentFailure) {
        try {
            UnpublishedDancingEvents unpublishedDancingEvents = fetchUnpublishedDancingEvents.ofEventOrganiser(input.eventOrganiserId());

            DancingEvent dancingEvent = DancingEvent.createWith(input.title(), input.description(), input.dateOfEvent());

            unpublishedDancingEvents.add(dancingEvent);

            updateUnpublishedDancingEvents.withDancingEvents(unpublishedDancingEvents);

            CreateDancingEventOutput output = CreateDancingEventOutput.from(dancingEvent);

            presentSuccess.ofDancingEventCreation(output);
        } catch (Exception e) {
            presentFailure.ofDancingEventCreation(e);
        }
    }
```

Clean Architecture - Business Logic

```java
public class UnpublishedDancingEvents {
    private static final int MAX_NUMBER_OF_UNPUBLISHED_DANCING_EVENTS = 5;
    private final List<DancingEvent> unpublishedDancingEvents;

    public void add(DancingEvent dancingEvent) throws NumberOfUnpublishedDancingEventsExceededException {
        if (numberOfUnpublishedDancingEventsExceeded()) {
            throw new NumberOfUnpublishedDancingEventsExceededException();
        }
        this.unpublishedDancingEvents.add(dancingEvent);
    }

    private boolean numberOfUnpublishedDancingEventsExceeded() {
        return count() >= MAX_NUMBER_OF_UNPUBLISHED_DANCING_EVENTS;
    }

    public int count() { return unpublishedDancingEvents.size(); }
```

Summary

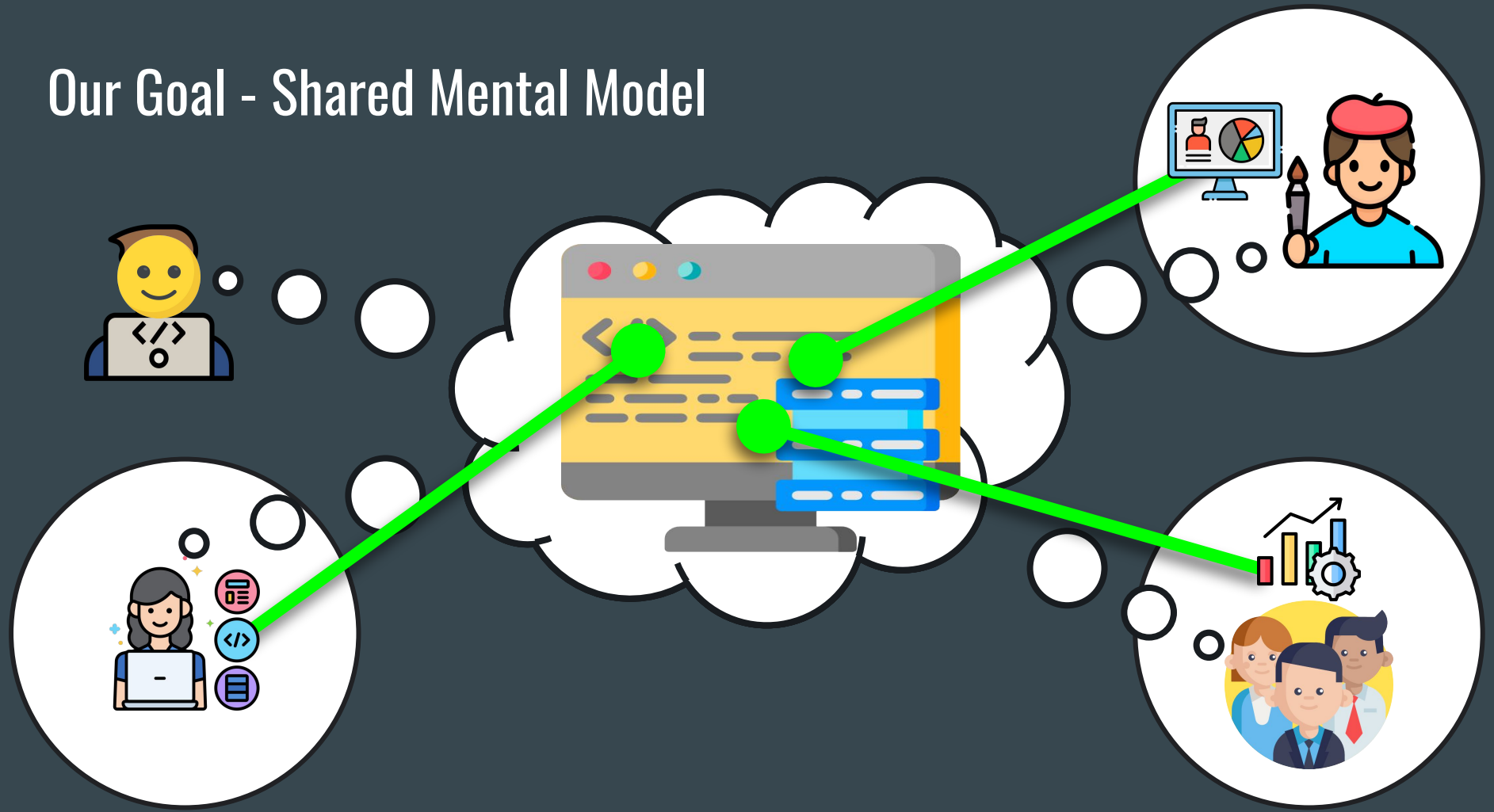
- Developing complex software is hard!
  - All stakeholders involved have different mental models and different languages
  - Software built by software engineers does not necessarily represent the outside world appropriately

- Focusing on the (business) **domain** at hand (i.e., the real world), at least subjectively, seems to provide a more productive way to develop software as it limits the mental mapping between all stakeholders and between outside world and internals of the software

- **Event Storming** is a more recent, more strategic DDD tool that can be used to explore a problem domain, find possible contexts, define ubiquitous languages, while at the same time break up company and people boundaries

- **Clean Architecture** is a more recent, more tactical tool that improves on DDD tactical patterns through its focused separation of concerns into concepts of the outside world (use cases w/ core business domain, presentation, gateways) that further minimises the mental mapping needed between code and people

Our Goal - Shared Mental Model



Sources

- Introducing EventStorming (Alberto Brandolini)
- Scoping and Organizing .NET Microservices Using Event Storming (Pluralsight)
- Domain-Driven Design (Eric Evans)
- Implementing Domain-Driven Design (Vaughn Vernon)
- Domain-Driven Design Distilled (Vaughn Vernon)
- Learning Domain-Driven Design (Vlad Khononov)
- Patterns, Principles, and Practices of Domain-Driven Design (Millett & Tune)
- Domain-Modelling made Functional (Scott Wlaschin)
- Clean Architecture, Clean Craftsmanship (Bob C. Martin)
- Get your hands dirty on Clean Architecture (Tom Hombergs)
- beyond-agility.com/die-hirnpsychologischen-und-verhaltenspsychologischen-hintergrunde-was-guten-von-schlechten-code-unterscheidet-german-post (Rick Janda)
- domaincentric.net
- threedots.tech/post/software-dark-ages
- Graphics: flaticon.com