GOPHER TOOLBOX

TECHNIQUES AND TOOLS EVERY SENIOR GO DEVELOPER SHOULD UNDERSTAND

ROBERT LASZCZAK

PRINCIPAL ENGINEER /id

CO-FOUNDER OF three dots labs

GOPHER TOOLBOX

TECHNIQUES AND TOOLS EVERY SENIOR GO DEVELOPER SHOULD UNDERSTAND

janusz
@janusz

Don’t try to use the pneumatic hammers in work. You’ll be frustrated, and there’s no need.

I see many videos with thousands of views, showing people how to use them. They are doing more harm than good.

#craft

9:57 AM · Dec 14, 2022

29 36 308 33

Don’t try to apply the same old DDD and OOP patterns to Go. You’ll be frustrated, and there’s no need.

I see many repositories with thousands of stars, showing people how to implement those in Go. They are doing more harm than good.

#golang

9:57 AM · Dec 14, 2022

29 36 308 33

GENERALISATION IS THE ROOT OF MOST EVIL IN PROGRAMMING

@levelsio 
@levelsio

PhotoAI.com is now almost 14,000 lines of raw PHP mixed with inline HTML, CSS in <style> and raw JS in <script> tags

I did not use TS, flexbox or frameworks except jQuery

A lot of $.ajax() and float:left though

It has 1,872 paying customers making $61,808 per month

Jon Sustar @jonsustar · Jul 3, 2023

Is... Is that a single file?

2 2 109

@levelsio @levelsio · Jul 3, 2023

Yes

8 144

DISCLAIMER:

DON'T TAKE ANYTHING AS GRANTED

ORM

ORM

SOLVED PROBLEM

- MAPPING DB RELATIONS TO IN-MEMORY OBJECTS AND VICE VERSA

ORM

CHALLENGES

- HARDER TO UNDERSTAND HOW CODE TRANSLATES TO QUERIES
- EASY TO INTRODUCE PERFORMANCE BOTTLENECKS

THEY ARE GOOD AND BAD ORMS

```
// Model specify the model you would like to run db operations
//
// // update all users's name to `hello`
// db.Model(&User{}).Update("name", “hello")
//
//  // if user's primary key is non-blank,
//  // will use it as condition, then will only update that user's name
//  // to `hello`
//
// db.Model(&user).Update("name", "hello")
func (db *DB) Model(value interface{}) (tx *DB) {
```

SQLBOILER

```
// Find a pilot and update his name
user, _ := models.FindUser(ctx, db, 1)
user.Name = "hello"
rowsUpdated, err := user.Update(ctx, db, boil.Infer())
```

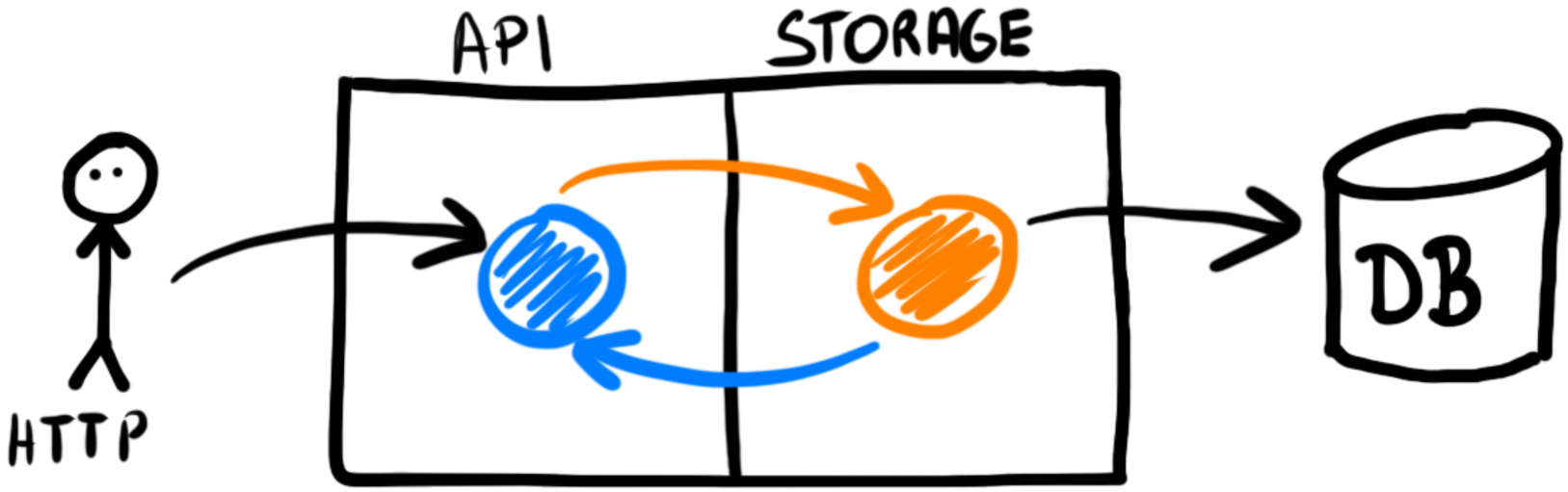
IT'S EASY TO RE-IMPLEMENT ORM AND END UP WITH SAME CHALLENGES

```
func (s *SqlThreadStore) GetThreadsForUser(userId, teamId string, opts model.GetUserThre
  pageSize := uint64(30)
  if opts.PageSize != 0 {
    pageSize = opts.PageSize
  }

  unreadRepliesQuery := sq.
    Select("COUNT(Posts.Id)").
    From("Posts").
    Where(sq.Expr("Posts.RootId = ThreadMemberships.PostId")).
    Where(sq.Expr("Posts.CreateAt > ThreadMemberships.LastViewed"))

  if !opts.Deleted {
    unreadRepliesQuery = unreadRepliesQuery.Where(sq.Eq{"Posts.DeleteAt": 0})
  }

  query := s.threadsAndPostsSelectQuery.
    Column(postSliceCoalesceQuery()).
    Columns(
      "ThreadMemberships.LastViewed as LastViewedAt",
      "ThreadMemberships.UnreadMentions as UnreadMentions",
    ).
    Column(sq.Alias(unreadRepliesQuery, "UnreadReplies")).
    Join("Posts ON Posts.Id = Threads.PostId").
    Join("ThreadMemberships ON ThreadMemberships.PostId = Threads.PostId")
```

RAW QUERIES ❤️

```go
func (r *SalesReadModel) CalculateSales(ctx context.Context) (int64, error) {
  var value int64
  err := queries.Raw(`
    WITH RECURSIVE sales_running_total AS (
      SELECT
      sale_date,
      SUM(price * quantity) AS daily_revenue,
      SUM(price * quantity) AS running_total
      FROM
      sales
      WHERE
      sale_date = (SELECT MIN(sale_date) FROM sales)
      GROUP BY
      sale_date
    )
    SELECT
      sale_date,
      daily_revenue,
      running_total,
      (running_total / (SELECT SUM(price * quantity) FROM sales)) AS running_total_percentage
    FROM
      sales_running_total
    ORDER BY
      sale_date;
    `,
    oid.ToUUID(),
  ).QueryRowContext(ctx, r.db).Scan(&value)
  if err != nil {
    return 0, err
  }
```

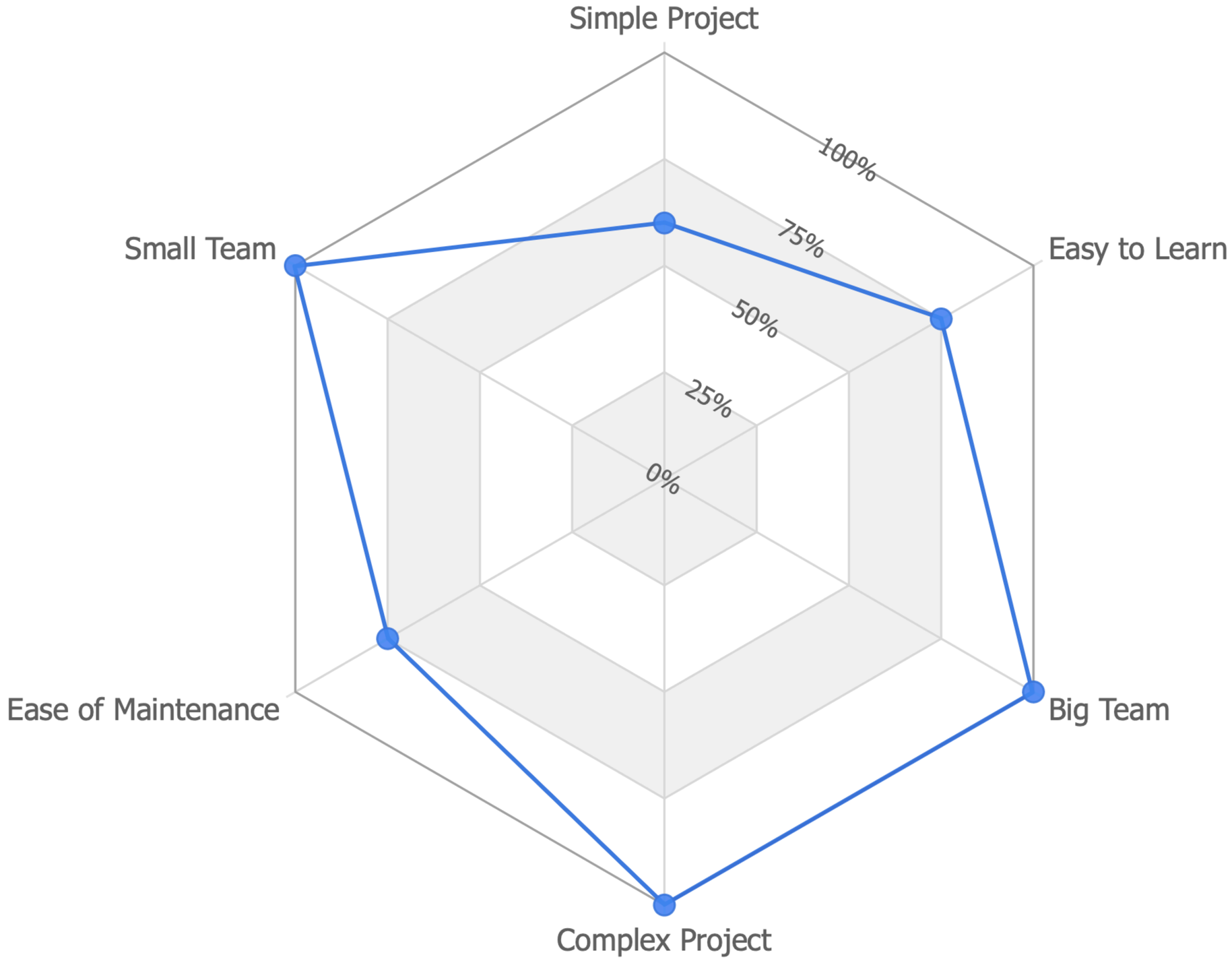
CLEAN ARCHITECTURE TO DE-COUPLE




```go
type UserResponse struct {
  ID          int                `json:"id"`
  FirstName   string             `json:"first_name"`
  LastName    string             `json:"last_name"`
  DisplayName string             `json:"display_name"`
  Emails      []EmailResponse    `json:"emails"`
}
```

```go
type UserDBModel struct {
  ID           int              `gorm:"column:id;pri
  FirstName    string           `gorm:"column:first_
  LastName     string           `gorm:"column:last_n
  Emails       []EmailDBModel   `gorm:"foreignKey:Us
  PasswordHash string           `gorm:"column:passwo
  LastIP       string           `gorm:"column:last_i
  CreatedAt    *time.Time       `gorm:"column:create
  UpdatedAt    *time.Time       `gorm:"column:update
}
```

CONSIDER NOSQL
OR
EVENT-SOURCING


Simple Project
Easy to Learn
Big Team
Complex Project
Ease of Maintenance
Small Team
100%
75%
50%
25%
0%

MOCKING

MOCKING

SOLVED PROBLEM

- SPEED UP TESTS FOR LOGIC THAT DEPENDS ON SLOW I/O (FILE SYSTEM, EXTERNAL SERVICES, DB ETC.)
- HELPS TO INTEGRATE WITH UNFINISHED FUNCTIONALITY

MOCKING

CHALLENGES

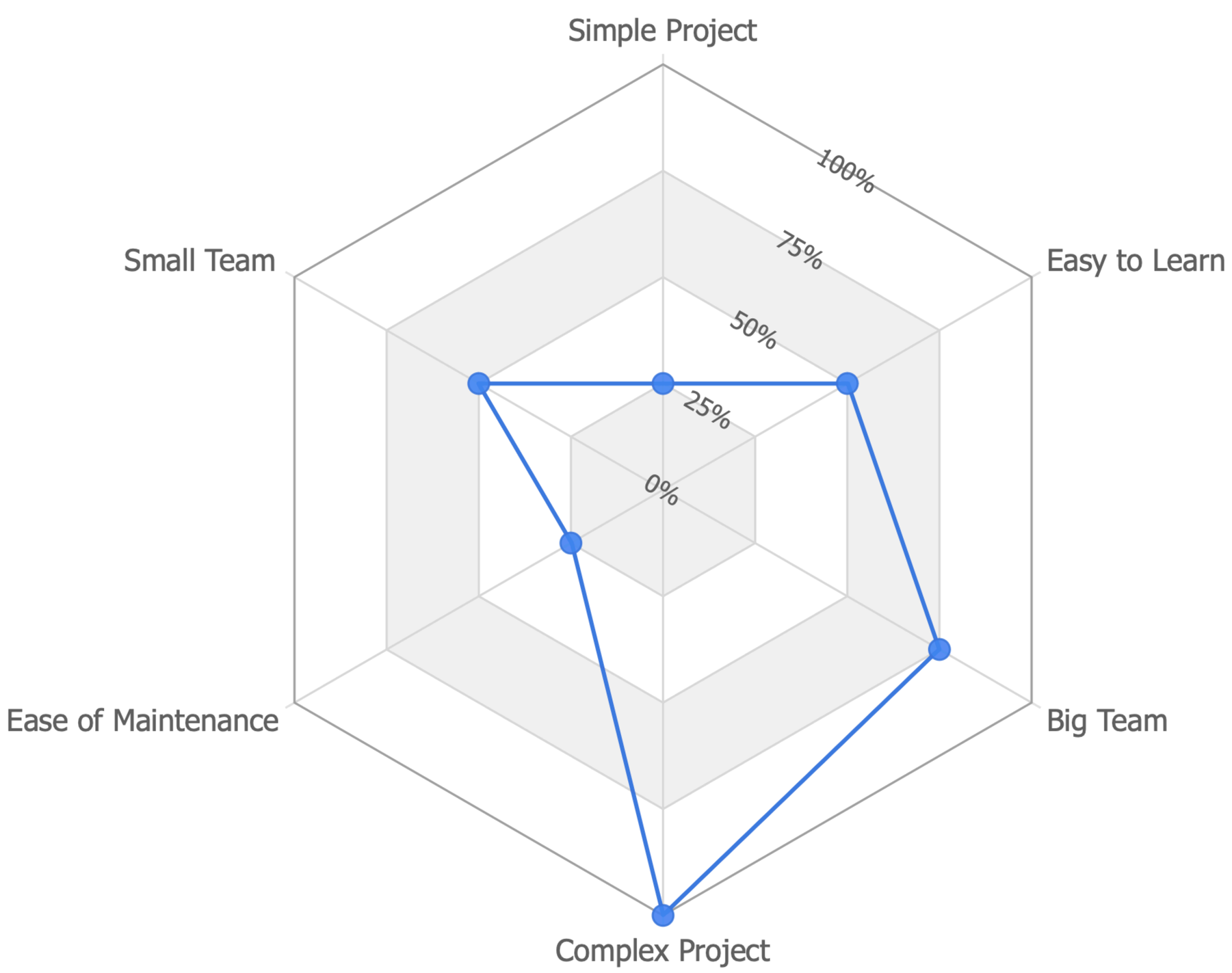
- KEEPING MOCKS UP-TO DATE
- MOCK BEHAVIOUR CAN BE WRONG
- INJECTING DEPENDENCIES

AVOID PREMATURE MOCKING

```
rentalsRepo.EXPECT().GetRentalByID(gomock.Any(), testRental.GetID()).Return(testRental, nil)

customersRepo.EXPECT().CustomerByExternalID(gomock.Any(), gomock.Not(nil), gomock.Not(nil)).Return(cus
rentalManager.EXPECT().GetRentalDurationSeconds(gomock.Any(), gomock.Not(nil)).Return(int64(10000), ni
insuranceManager.EXPECT().GetInsuranceOptions(gomock.Any(), gomock.Not(nil), gomock.Not(nil)).Return(
carsRepo.EXPECT().ListCarsForCustomer(gomock.Any(), gomock.Not(nil), gomock.Not(nil)).Return(customerC
rentalManager.EXPECT().GetCarCategoryClaimName(gomock.Any(), gomock.Not(nil)).Return(carCategoryClaimN
paymentProcessor.EXPECT().ProcessPayment(gomock.Any(), gomock.Not(nil), gomock.Not(nil)).Return(paymen
notificationService.EXPECT().SendConfirmationEmail(gomock.Any(), gomock.Not(nil), gomock.Not(nil)).Ret

rentalAgreement, err := bookingManager.CheckRental(context.Background(), testReqMetadata, testRental.C
require.NotNil(t, rentalAgreement)
require.NoError(t, err)
```

CONSIDER WRITING BY HAND

```go
type BalanceUpdate struct {
  UserID       string
  AmountChange int
}

type BalanceRepositoryMock struct {
  BalanceUpdates []BalanceUpdate
  balanceUpdatesLock sync.Mutex
}

func (u *BalanceRepositoryMock) UpdateTrainingBalance(
   ctx context.Context,
   userID string,
   amountChange int,
) error {
  u.balanceUpdatesLock.Lock()
  defer u.balanceUpdatesLock.Unlock()

  u.BalanceUpdates = append(u.BalanceUpdates, BalanceUpdate{userID, amountChange})
  return nil
}
```


Simple Project
Easy to Learn
Big Team
Complex Project
Ease of Maintenance
Small Team
100%
75%
50%
25%
0%


APPLYING THE HIGH SCALABILITY TIPS FROM GOOGLE AND NETFLIX SOFTWARE ARCHITECTURES TO YOUR TRIVIAL PROJECT.


SURVEY

MICROSERVICES

MICROSERVICES

SOLVED PROBLEM

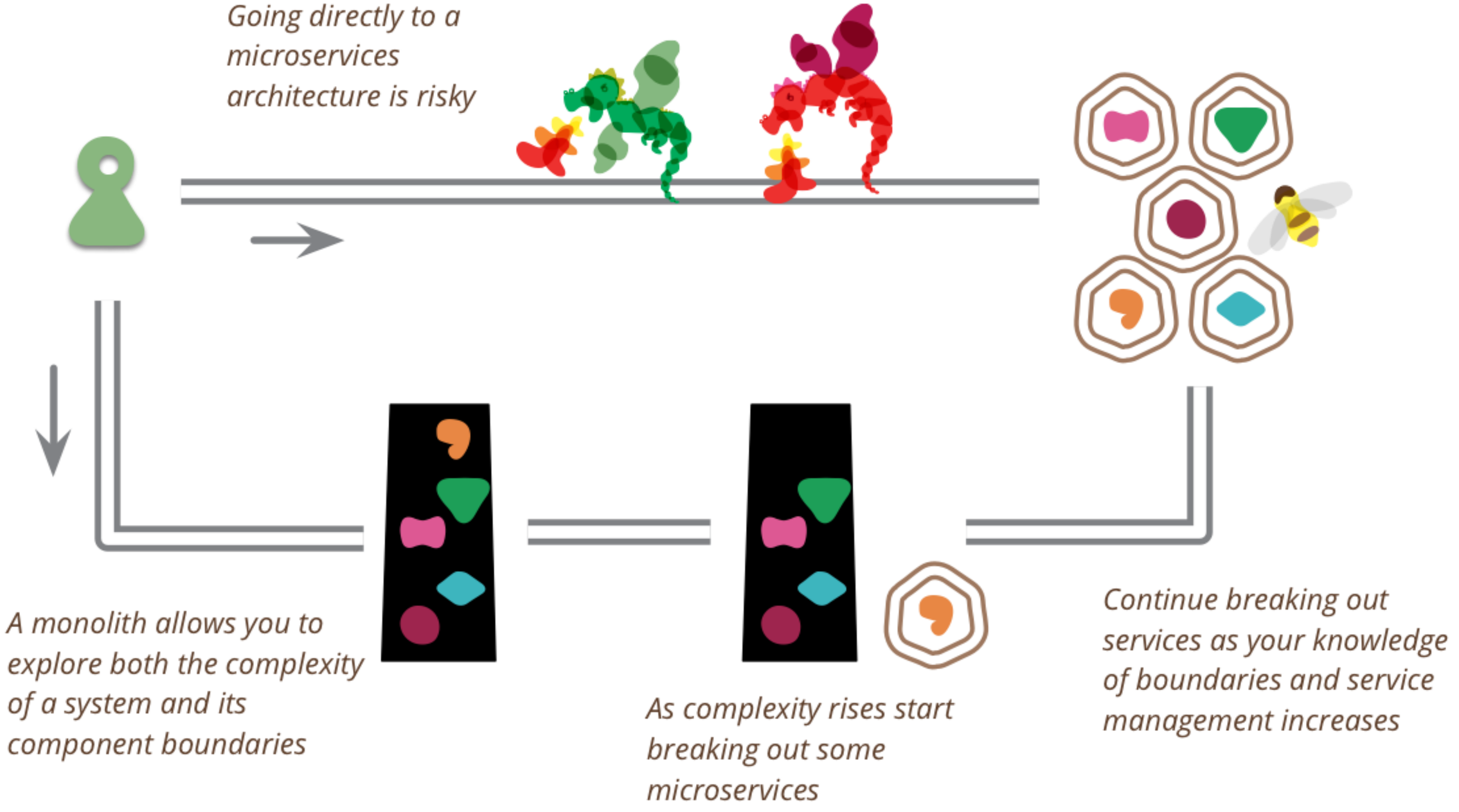
- IMPROVE TEAM'S AUTONOMY
- IMPROVED RESILIENCY*
- EXTREME SCALABILITY

MICROSERVICES

CHALLENGES

- DEBUGGING
- TESTING
- RUNNING LOCALLY
- PROPER SERVICES SEPARATION
- MORE COMPLEX INFRASTRUCTURE
- HARDER TO EXCHANGE DATA

CONSIDER “MONOLITH FIRST” APPROACH



https://martinfowler.com/bliki/MonolithFirst.html

MODULAR MONOLITH

MODULAR MONOLITH

SOLVED PROBLEM

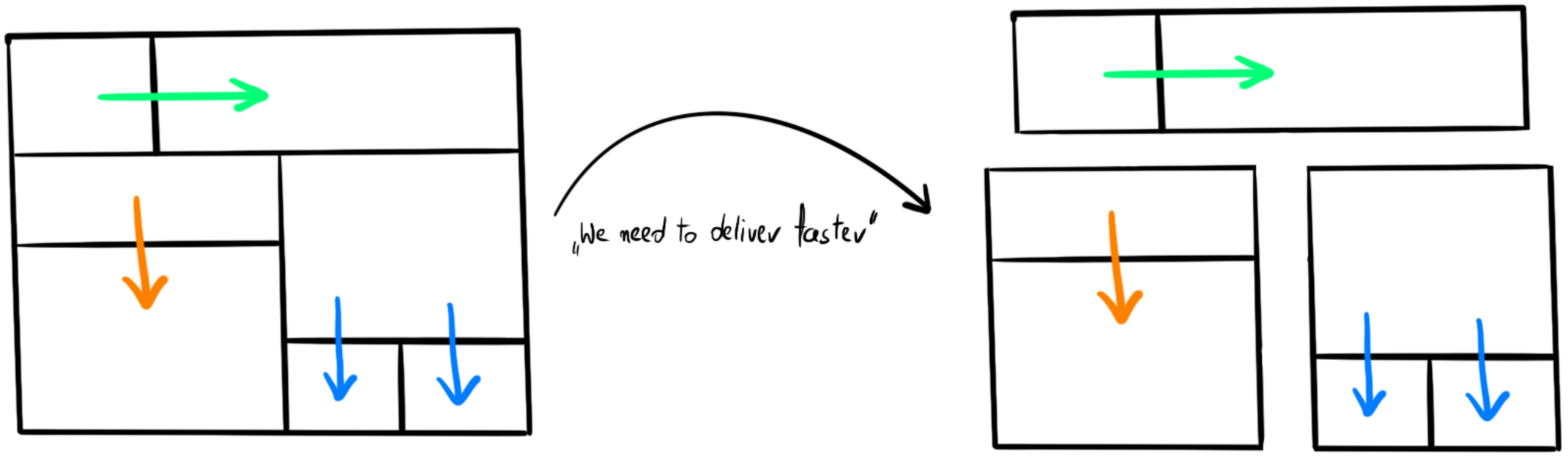
- EASIER TO DEBUG
- EASIER TO TEST THE ENTIRE SYSTEM
- EASIER (POSSIBLE) TO RUN THE APPLICATION LOCALLY
- EASIER TO CHANGE THE MODULE BOUNDARIES

MODULAR MONOLITH

CHALLENGES

- CAN’T DEPLOY INDIVIDUAL PARTS OF SYSTEM
- IF THE CI IS BROKEN, IT DOESN'T WORK FOR EVERYONE

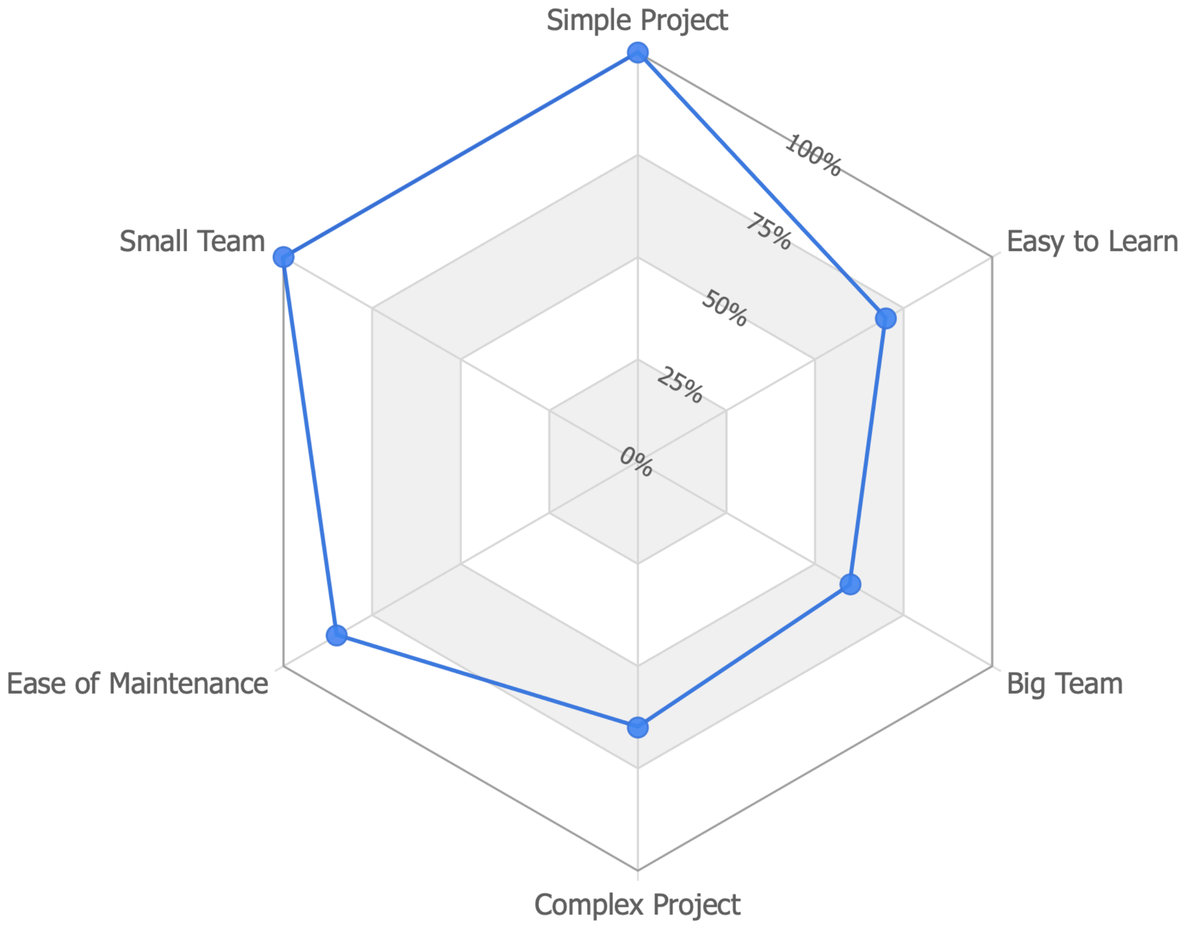
TRY TO MODULARISE PROPERLY FROM DAY 1

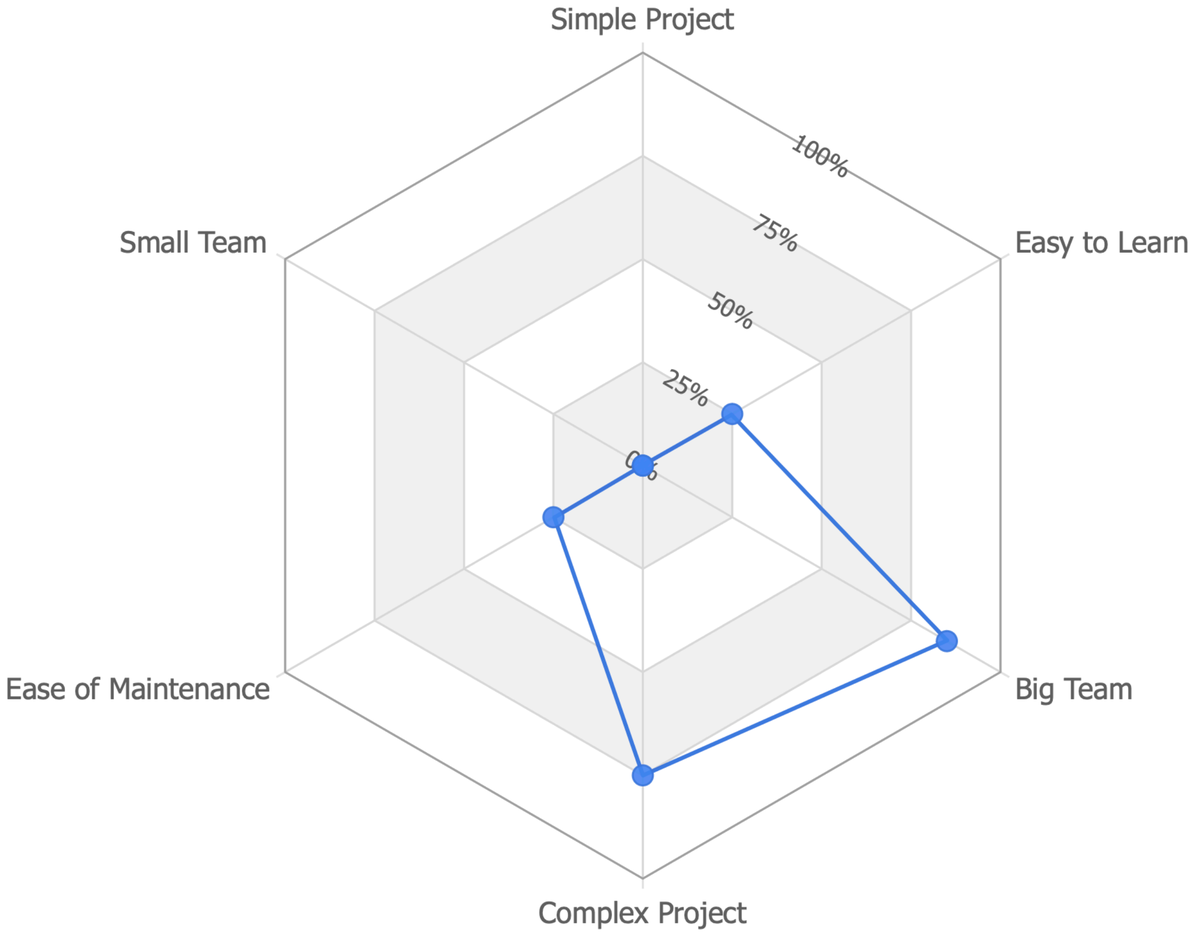


CLEAN ARCHITECTURE USEFUL FOR MODULARIZATION

EASIER TO DO REFACTORING AT SCALE

MONOLITH VS MICROSERVICES





HYPE CYCLES

THE AHA STACK

Auto

Thin client

Ship plain HTML to the browser, not a thick JavaScript app that runs inside the browser.

Hypermedia

Send HTML over the wire, not JSON.

Sprinkles of JavaScript

Add client-side interactivity with JavaScript.

HTML partials

Send HTML partials for UI updates.

Simple yet awesome backend

Astro wins our heart

Lightweight frontend

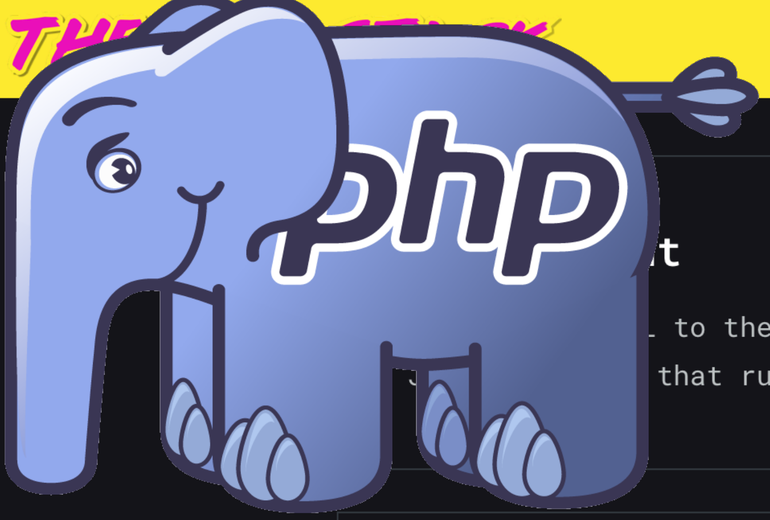
htmx and Alpine.js are the perfect match

Declarative

Use as little JavaScript as possible, add functionality through a declarative approach.

Minimalist

Use as few dependencies as possible, to make your app solid as a rock.



Auto

t

to the browser, not a thick
that runs inside the browser.

Hypermedia

Send HTML over the wire, not JSON.

Sprinkles of JavaScript

Add client-side interactivity with JavaScript.

HTML partials

Send HTML partials for UI updates.

Simple yet awesome backend

Astro wins our heart

Lightweight frontend

htmx and Alpine.js are the perfect match

Declarative

Use as little JavaScript as possible, add functionality through a declarative approach.

Minimalist

Use as few dependencies as possible, to make your app solid as a rock.

2014 - We must adopt #microservices to solve all problems with monoliths

2016 - We must adopt #docker to solve all problems with microservices

2018 - We must adopt #kubernetes to solve all problems with docker

roblaszczak

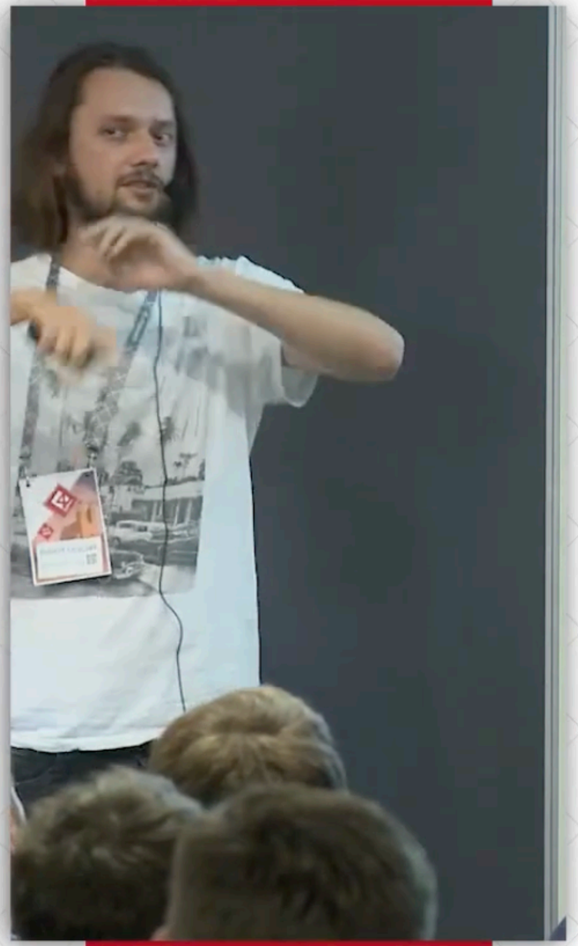
Robert Laszczak

Pragmatyczny Hype-driven development

~~2020 - We must adopt #serverless to solve all problems with kubernetes?~~

2020 - We must adopt #monolith to solve all problems with kubernetes?

roblaszczak



Robert Laszczak

Pragmatyczny Hype-driven development

4Developers Kraków 2018: Pragmatyczny Hype-driven development, Robert Laszczak

EVENT-DRIVEN ARCHITECTURE

EVENT-DRIVEN ARCHITECTURE

SOLVED PROBLEM

- EXTREME SCALABILITY
- HANDLING SLOW BACKGROUND PROCESSES
- IMPROVED RESILIENCY
- SOLVES MICROSERVICES' DATA SHARING CHALLENGES
- INCREASED TEAM'S AUTONOMY

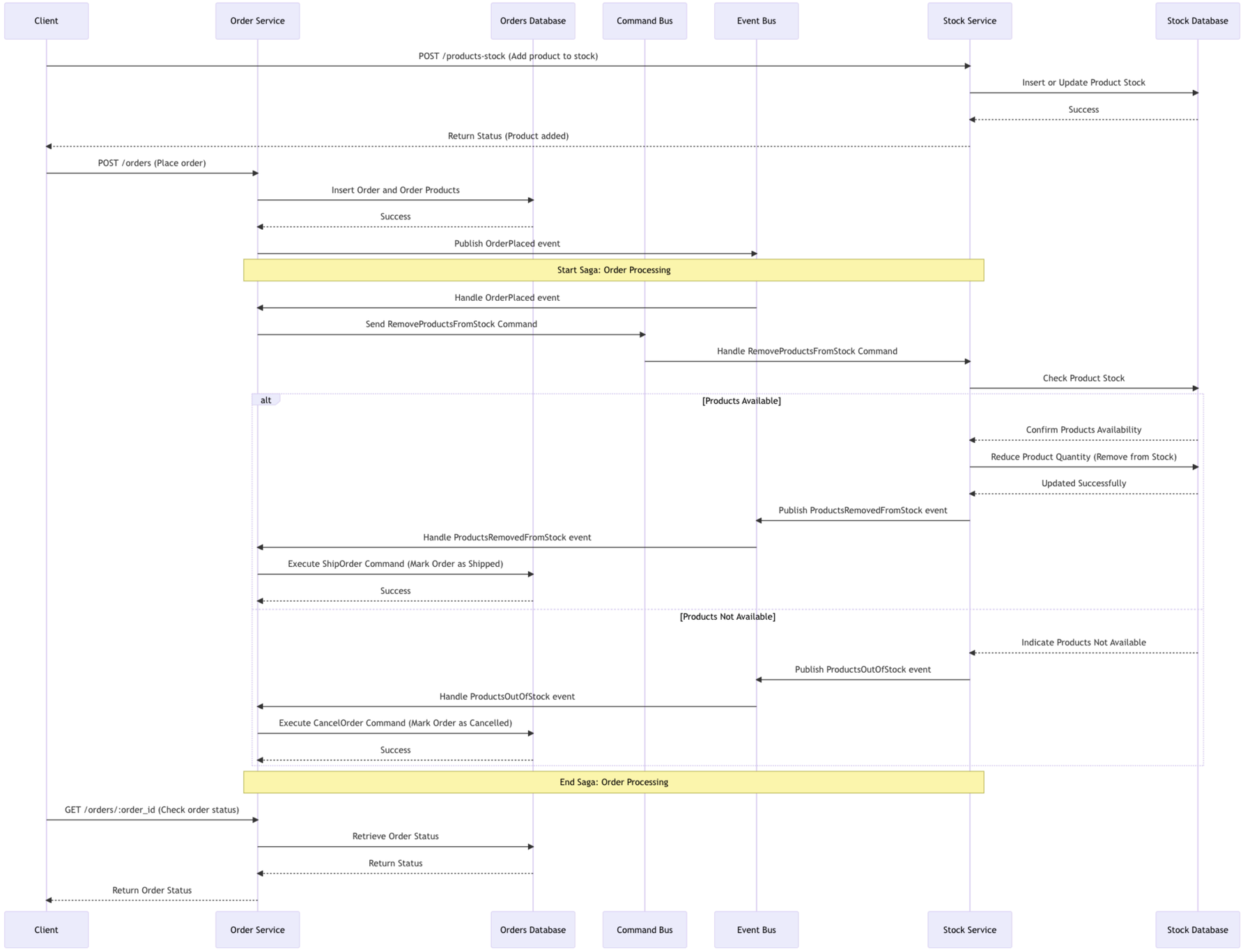
EVENT-DRIVEN ARCHITECTURE

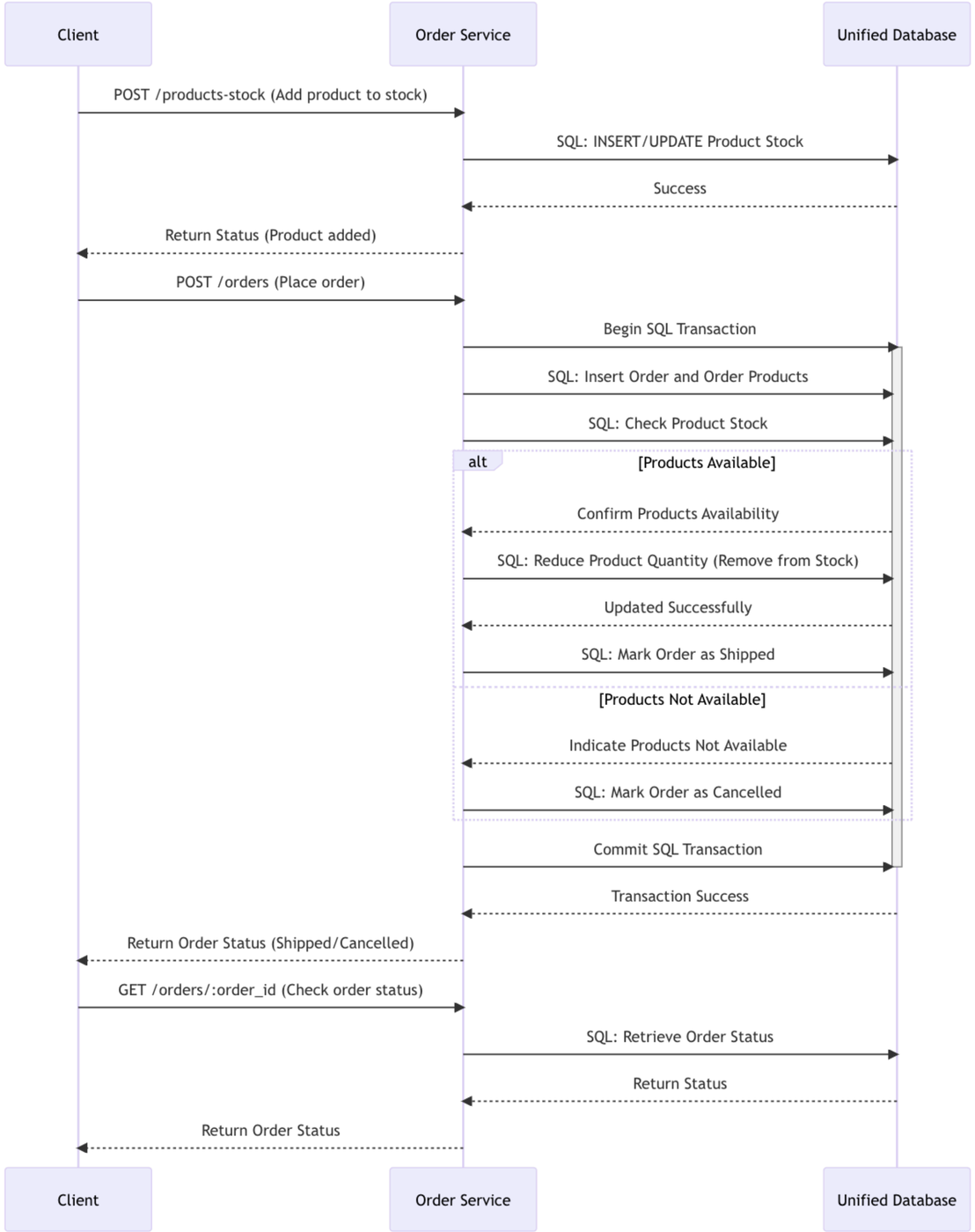
CHALLENGES

- DECOUPLING
- DEBUGGING
- TESTING
- EVENTUAL-CONSISTENCY
- ERROR HANDLING PATTERNS
- EVENT ORDERING
- AT-LEAST ONCE DELIVERY

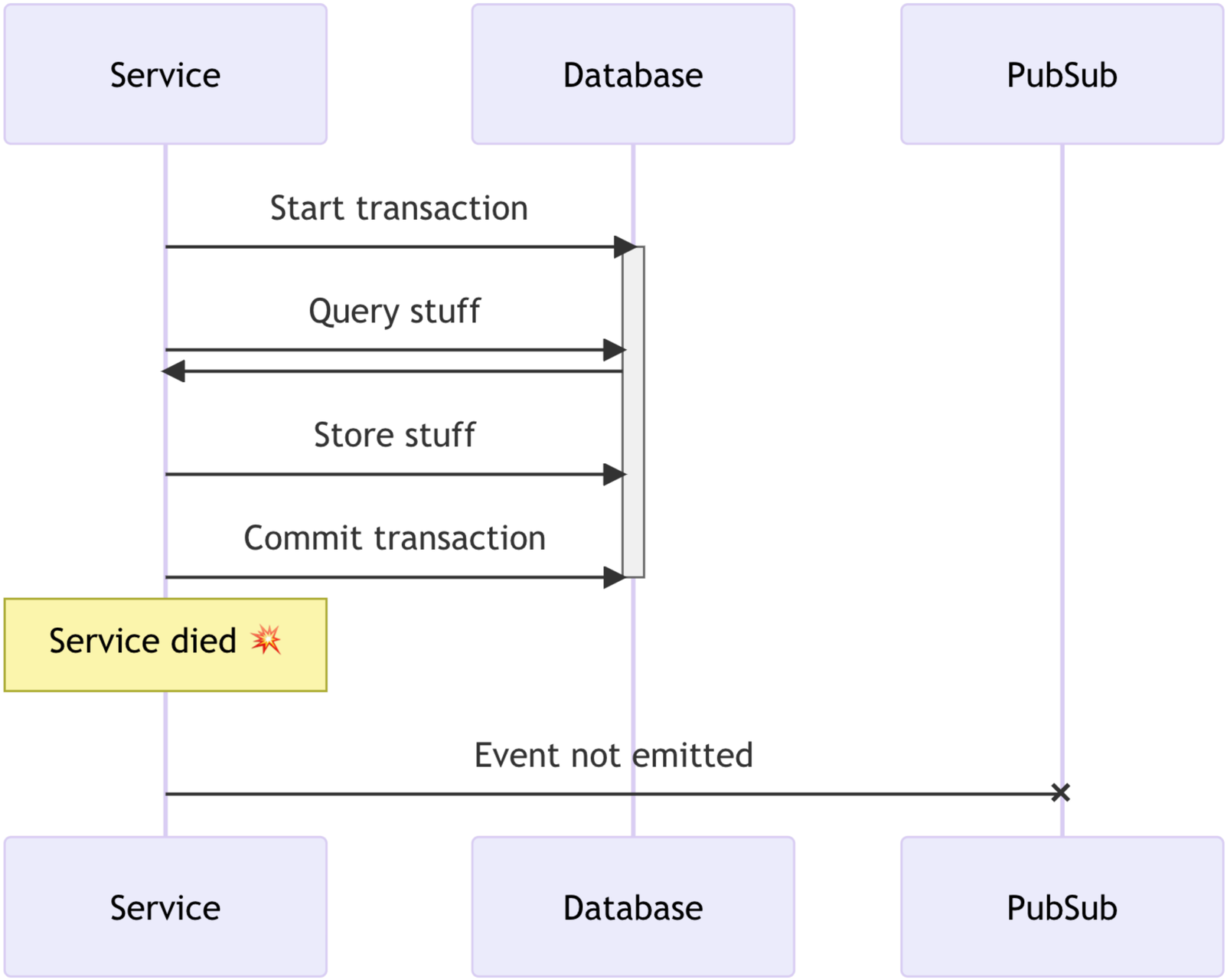
DON’T RE-INVENT THE WHEEL

BEFORE USING SAGA ASK YOURSELF "DOES THIS TRANSACTION NEED TO BE DISTRIBUTED?"

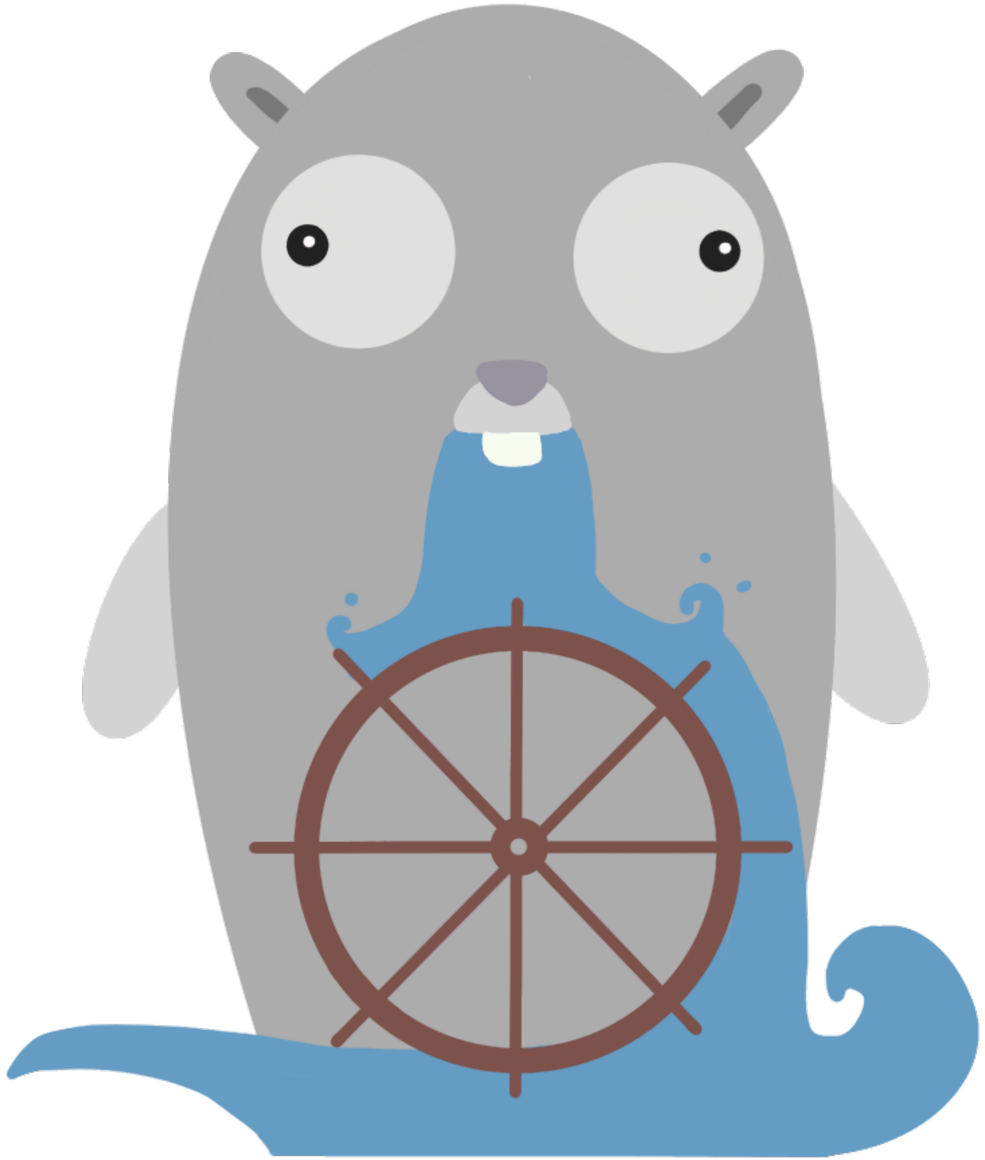

Client
Order Service
Orders Database
Command Bus
Event Bus
Stock Service
Stock Database
POST /products-stock (Add product to stock)
Insert or Update Product Stock
Success
Return Status (Product added)
POST /orders (Place order)
Insert Order and Order Products
Success
Publish OrderPlaced event
Start Saga: Order Processing
Handle OrderPlaced event
Send RemoveProductsFromStock Command
Handle RemoveProductsFromStock Command
Check Product Stock
alt
[Products Available]
Confirm Products Availability
Reduce Product Quantity (Remove from Stock)
Updated Successfully
Publish ProductsRemovedFromStock event
Handle ProductsRemovedFromStock event
Execute ShipOrder Command (Mark Order as Shipped)
Success
[Products Not Available]
Indicate Products Not Available
Publish ProductsOutOfStock event
Handle ProductsOutOfStock event
Execute CancelOrder Command (Mark Order as Cancelled)
Success
End Saga: Order Processing
GET /orders/:order_id (Check order status)
Retrieve Order Status
Return Status
Return Order Status
Client
Order Service
Orders Database
Command Bus
Event Bus
Stock Service
Stock Database


Client
Order Service
Unified Database
POST /products-stock (Add product to stock)
SQL: INSERT/UPDATE Product Stock
Success
Return Status (Product added)
POST /orders (Place order)
Begin SQL Transaction
SQL: Insert Order and Order Products
SQL: Check Product Stock
alt
[Products Available]
Confirm Products Availability
SQL: Reduce Product Quantity (Remove from Stock)
Updated Successfully
SQL: Mark Order as Shipped
[Products Not Available]
Indicate Products Not Available
SQL: Mark Order as Cancelled
Commit SQL Transaction
Transaction Success
Return Order Status (Shipped/Cancelled)
GET /orders/:order_id (Check order status)
SQL: Retrieve Order Status
Return Status
Return Order Status
Client
Order Service
Unified Database

REMEMBER ABOUT THE OUTBOX PATTERN

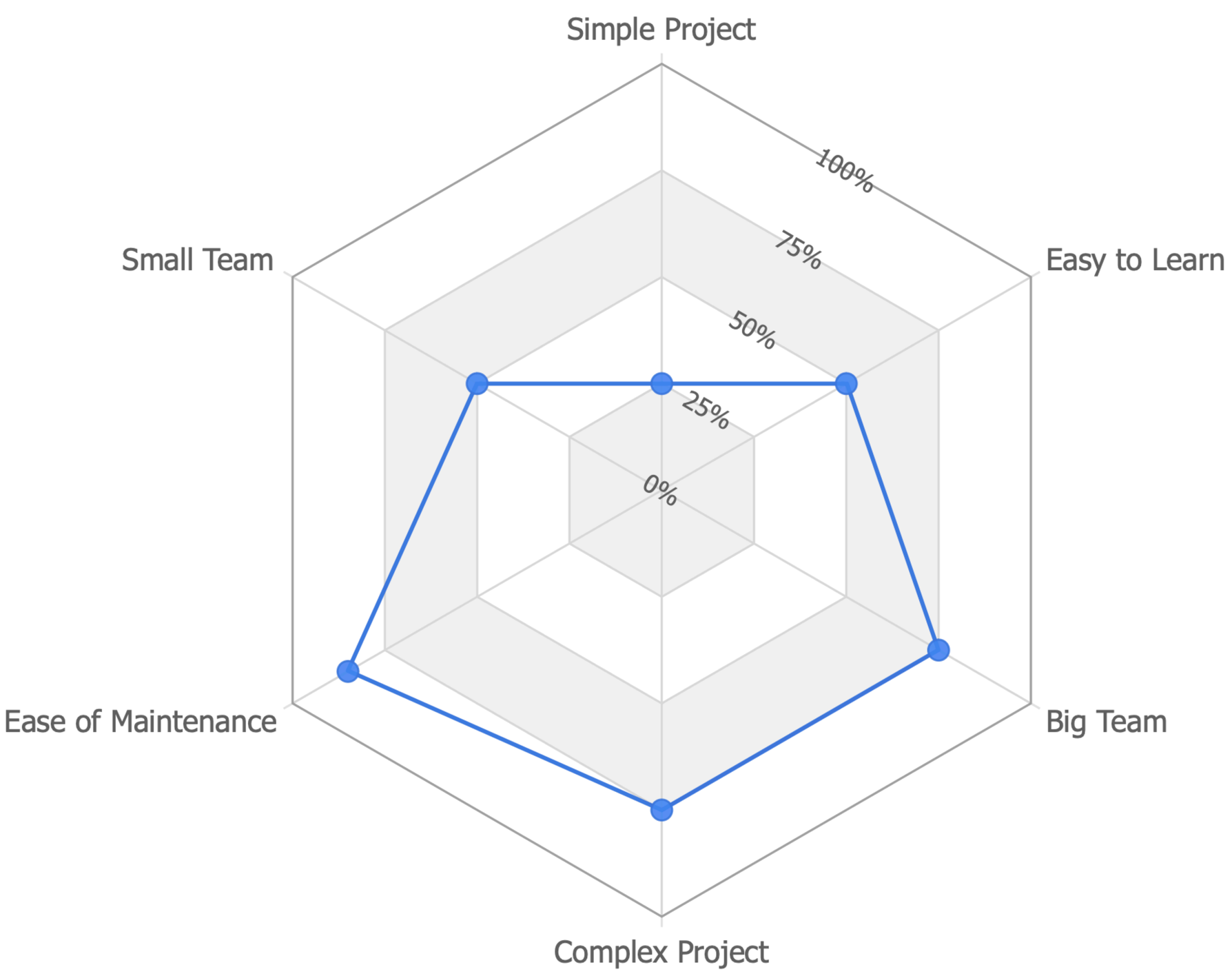

Service
Database
PubSub
Start transaction
Query stuff
Store stuff
Commit transaction
Service died
Event not emitted
Service
Database
PubSub

https://watermill.io/



https://threedots.tech/event-driven/

Available till 26 April


Simple Project
Easy to Learn
Big Team
Complex Project
Ease of Maintenance
Small Team
100%
75%
50%
25%
0%

1. GOOD PRACTICES DON’T EXIST

2. UNDERSTAND THE TOOL BEFORE USING IT

3. BE PREPARED TO BE WRONG

4. GENERALISATION IS EVIL

Gopher Toolbox: presentation materials

Thanks for attending my talk! Here you can find the materials that may be useful for you.

- 💌 Our newsletter (13k+ subscribers)
- 🖨️ Why re-using some struct for database, domain model and API is a bad idea
- ✍️ How we approach writing mocks by hand
- 📦 How you can apply modular monolith in Go
- 📩 Our Go Event-Driven online training
- 💦 Watermill - our library for building event-driven applications
- 🗂️ What ORMs we recommend
- 🫧 How to use Clean Architecture
- 📕 Our free e-book showing how to apply Clean Architecture in Go
- 🔀 Trunk Based Development

THANKS!

https://threedots.tech/toolbox/