



RETHINKING DDD IN GO

FROM MYTHS TO REDUCED PROJECT COMPLEXITY

ROBERT LASZCZAK

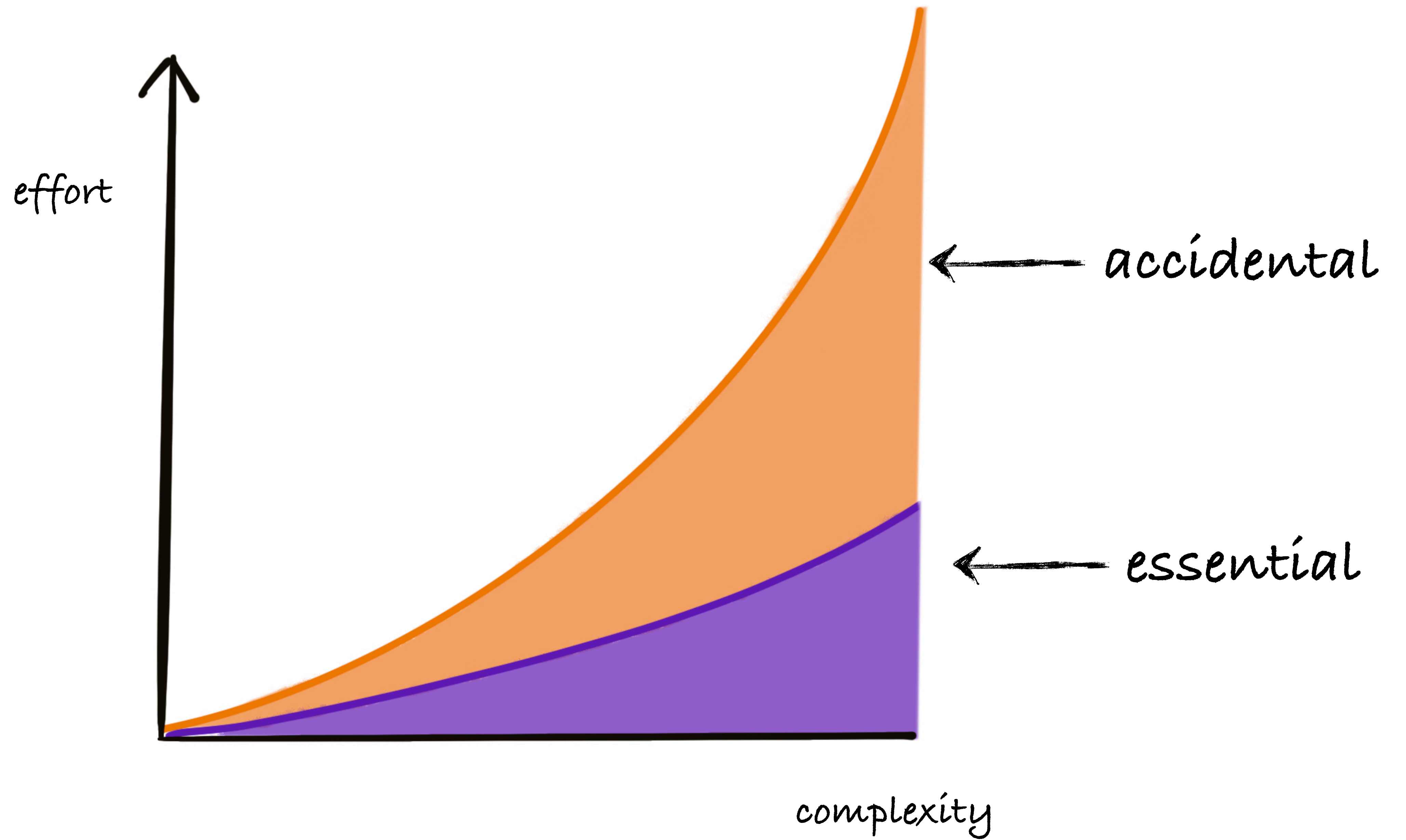
PRINCIPAL ENGINEER [/id](#)

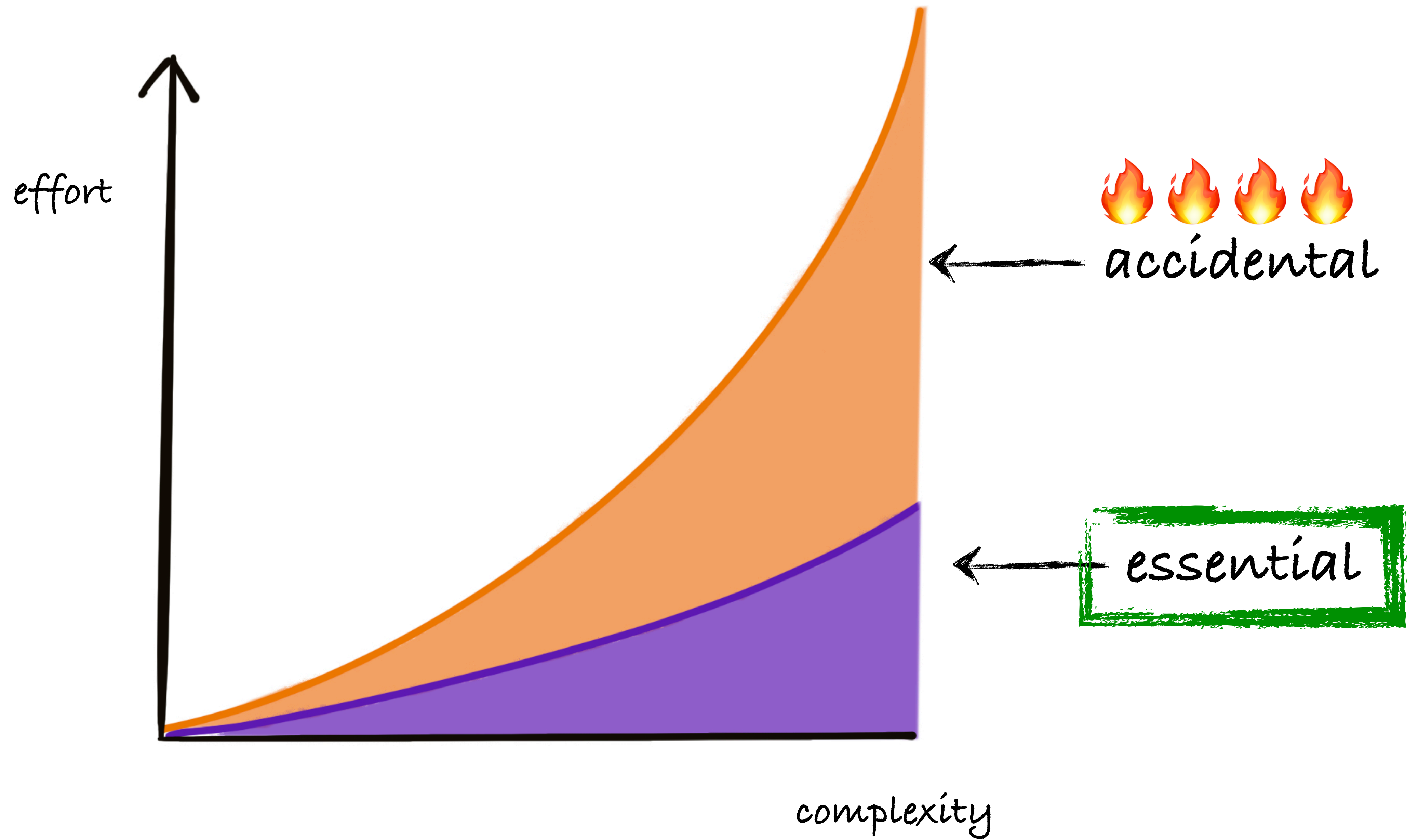
CO-FOUNDER OF [three dots labs](#)



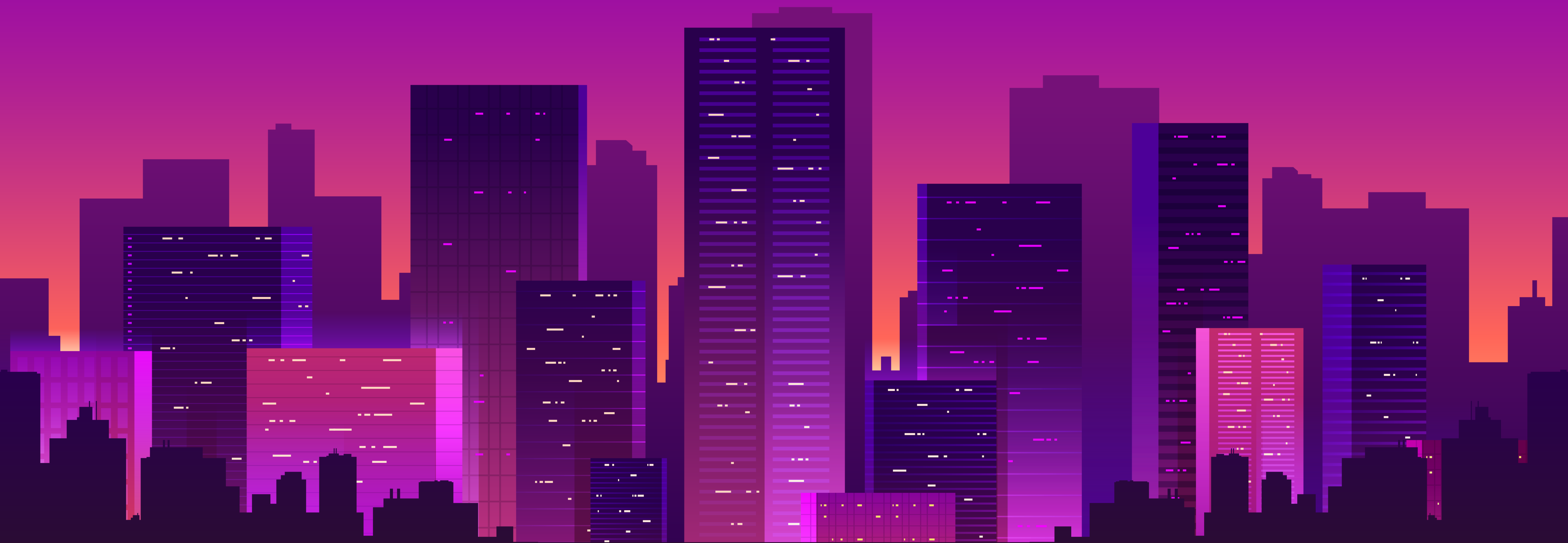
**HOW TO DEVELOP
SOFTWARE MORE
EFFICIENTLY?**

COMPLEXITY





BILLING MATE



"ONE YEAR AGO, IMPLEMENTING SUCH A FEATURE TOOK A WEEK; NOW IT'S TAKING TWO MONTHS; WHAT'S HAPPENING?"

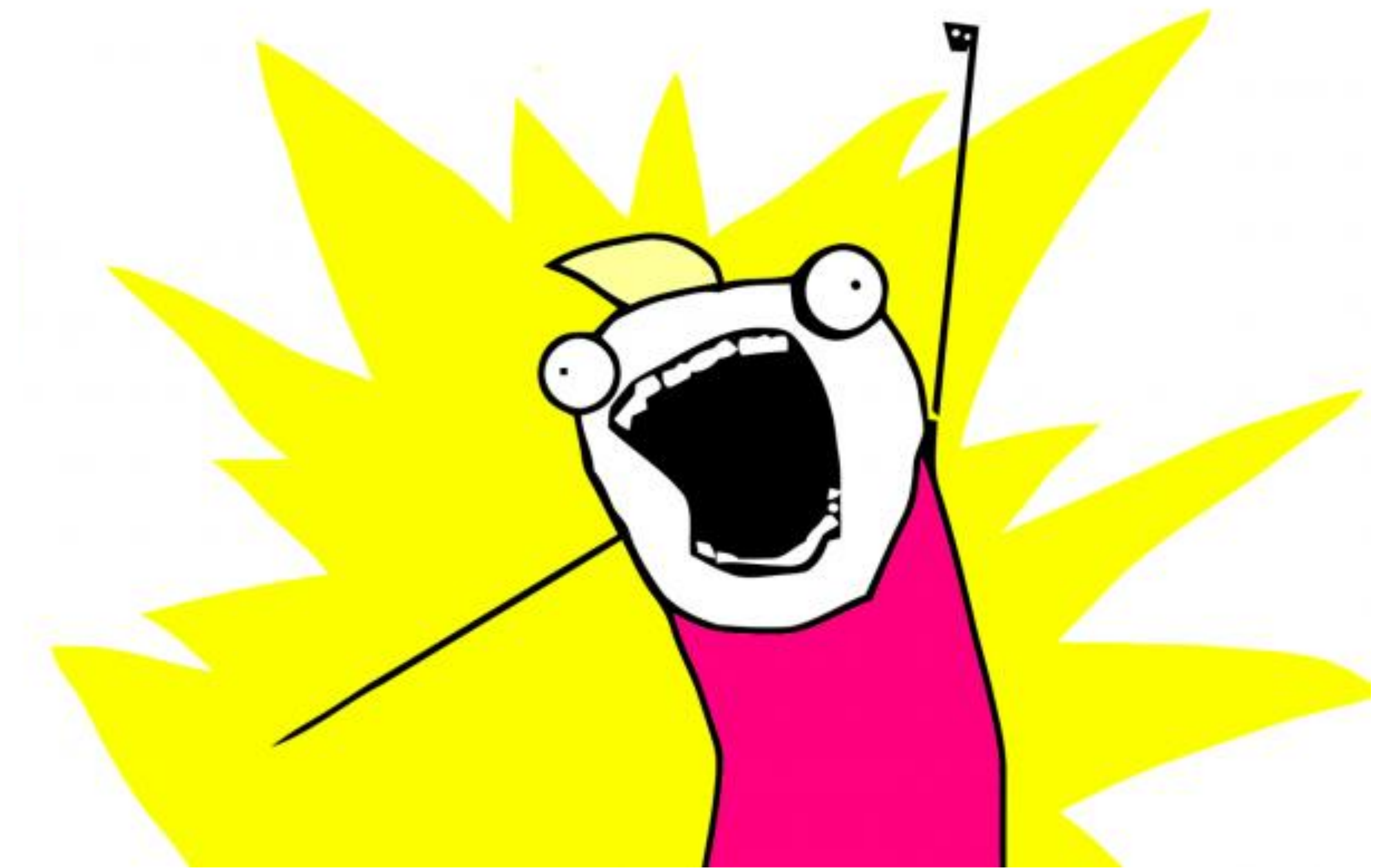
BILLING MATE MANAGEMENT



INVOICE INVADER



***"LET'S HIRE MORE
PEOPLE!"***



**MIGRATION TO MICROSERVICES
AND KUBERNETES**

**BECAUSE OUR
MONOLITH
APPLICATIONS
- WE NEED TO SPLIT**

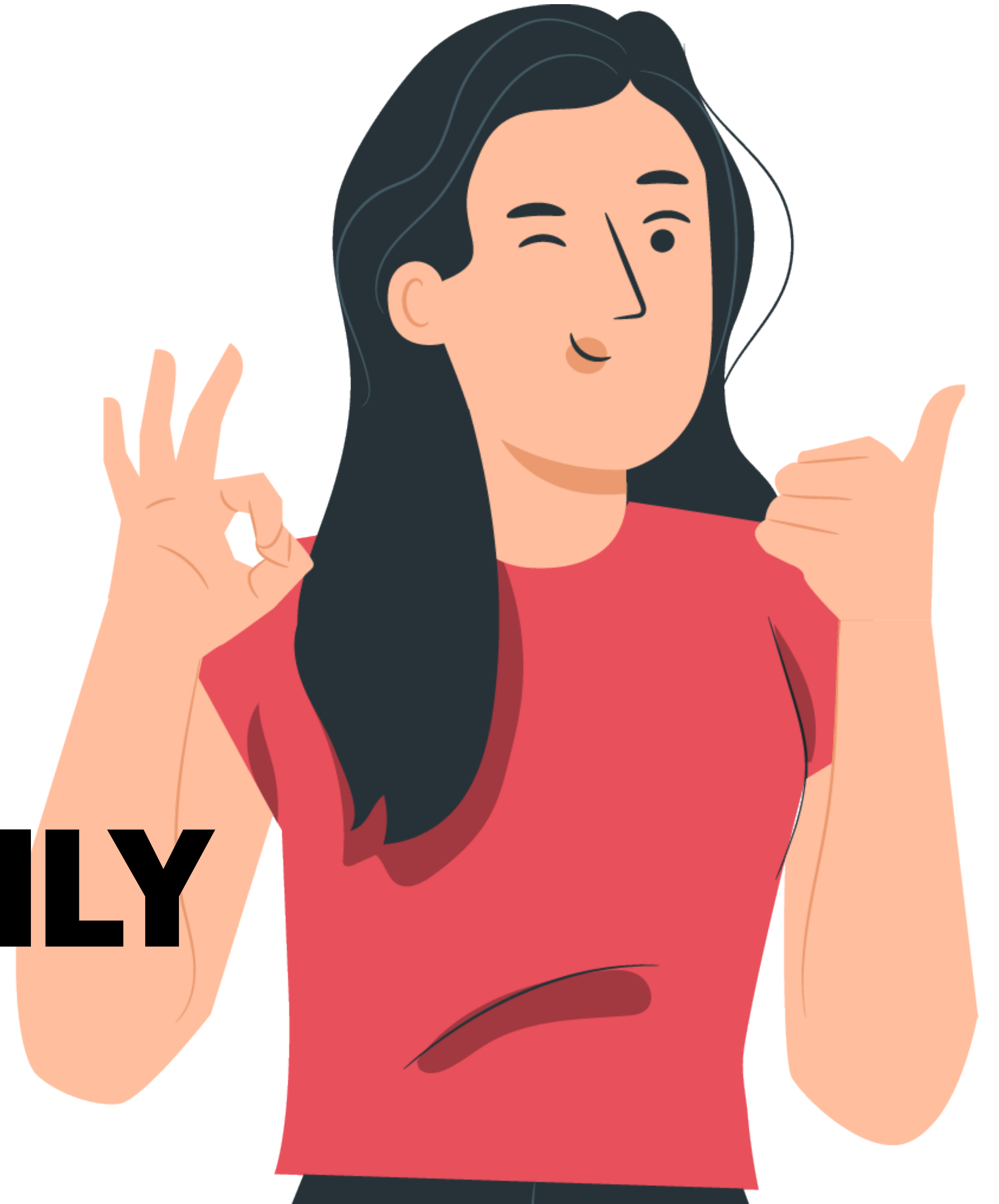




***"IT NEEDS TO BE LIKE
THAT, I'VE SEEN IT IN
EVERY COMPANY"***

BILLING MATE EMPLOYEE

LET'S MEET EMILY



INVOICE

1/4/2024

Issue date 29.04.2024
Date of service 29.04.2024
Due date 6.05.2024

From:

Robert Laszczak
ul. Grzegórzecka 14
31-823 Kraków

Billed to:

Three Dots Labs
Pawia 8/9
31-812 Kraków

Position.	Description	Quantity	Unit cost	Tax rate [%]	Tax amount	Total
1	Software development services	1	\$90,000.00	23%	\$20,700.00	\$110,700.00
2	Consulting services	1	\$8,000.00	23%	\$1,840.00	\$9,840.00
			Total	23%	\$22,540.00	\$120,540.00

Description	Quantity	Unit cost	Tax rate [%]	Tax amount	Total
Software development services	1	\$90,000.00	23%	\$20,700.00	\$110,700.00
Consulting services	1	\$8,000.00	23%	\$1,840.00	\$9,840.00
		Total	23%	\$22,540.00	\$120,540.00

CREDIT NOTE

CN1/4/2024

Issue date 29.04.2024
Date of service 29.04.2024
Due date 6.05.2024

From:

Robert Laszczak
ul. Grzegórzecka 14
31-823 Kraków

Billed to:

Three Dots Labs
Pawia 8/9
31-812 Kraków

Position.	Description	Quantity	Unit cost	Tax rate [%]	Tax amount	Total
1	Software development services	1	-\$81,000.00	23%	-\$18,630.00	-\$99,630.00
			Total	23%	-\$18,630.00	-\$99,630.00

3 TEAMS ENGAGED

7 MICROSERVICES CHANGED

3 MONTHS OF WORK

RESULT: 2 PDFS

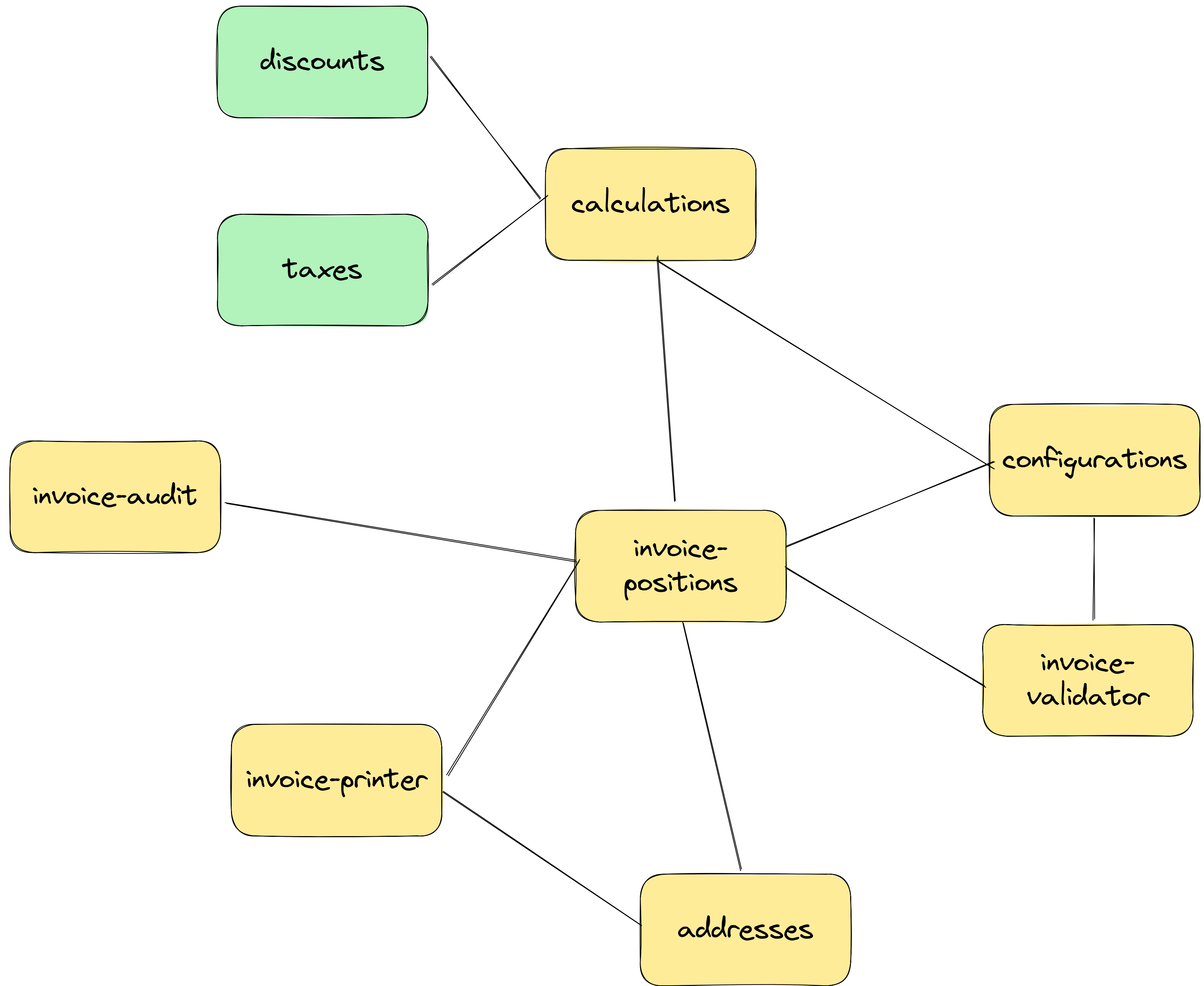


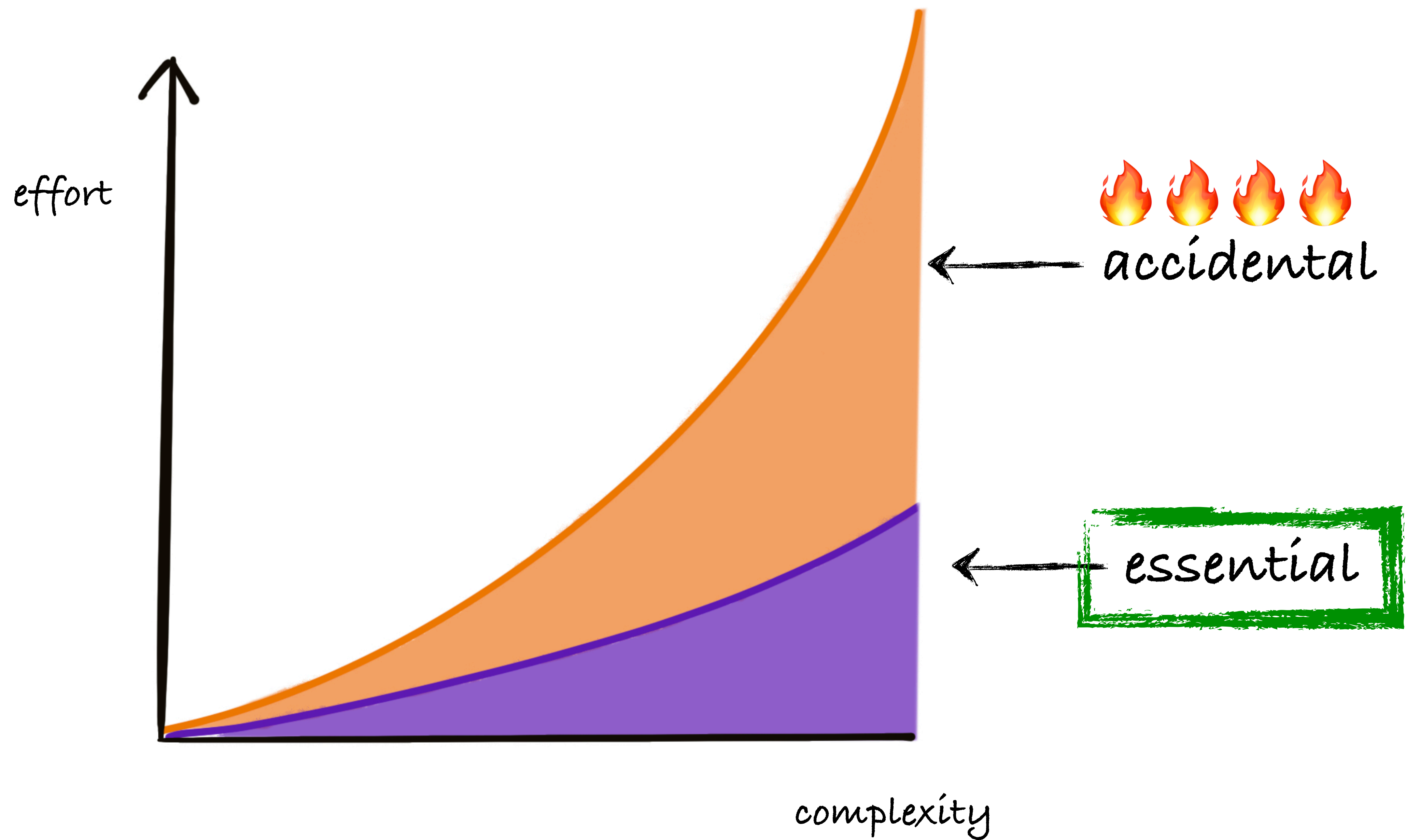
WHAT WENT WRONG?

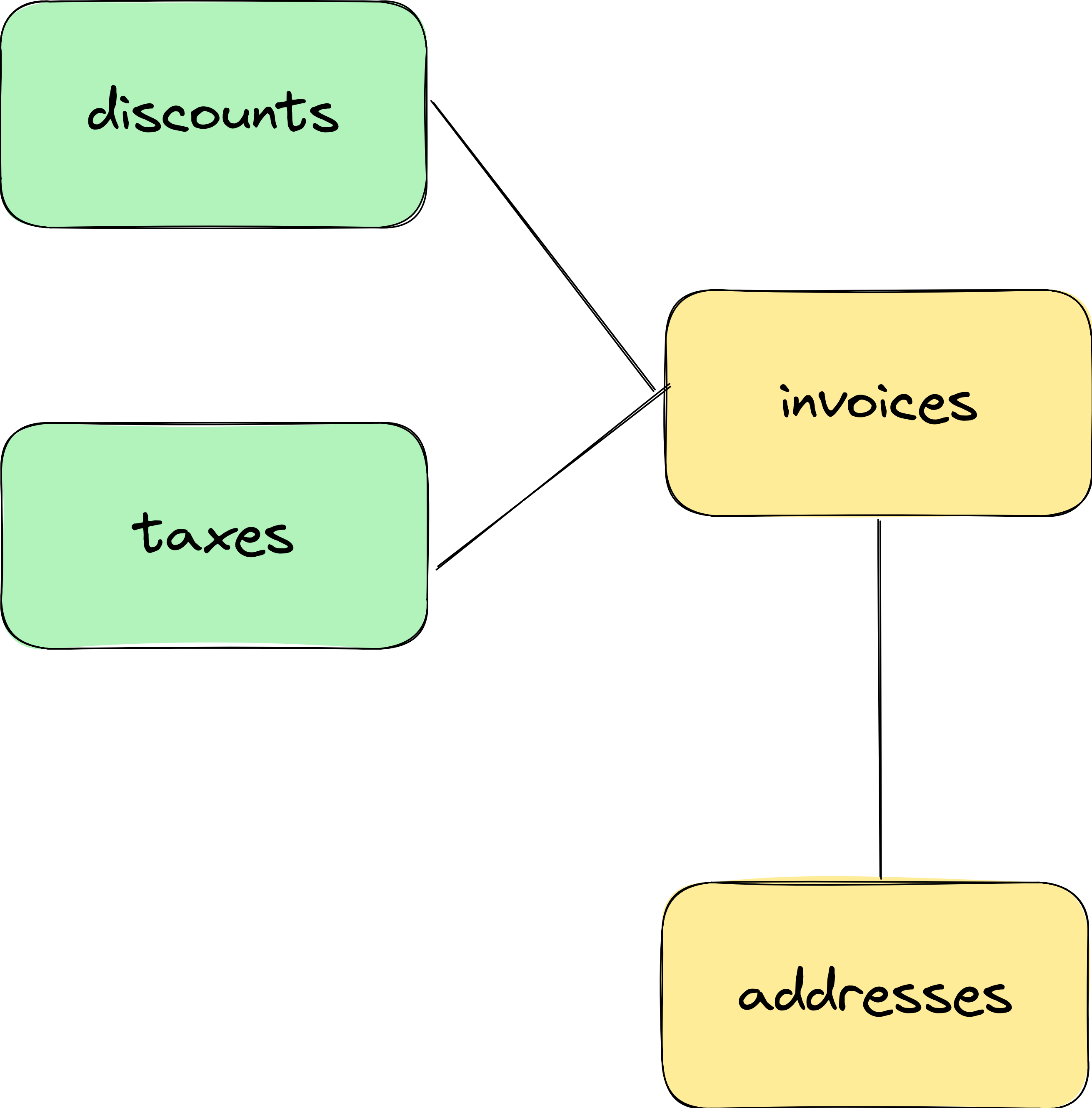
✅ **DEVELOPING NEW FEATURES - ESSENTIAL COMPLEXITY**

✅ **HIRING NEW PEOPLE - WE NEED TO HIRE PEOPLE TO SCALE**

🤔 **MICROSERVICES**







DOMAIN-DRIVEN

DESIGN

(DDD)

**“DDD IS A SET OF TECHNIQUES
THAT HELPS BUILD COMPLEX
SYSTEMS THAT ARE
MAINTAINABLE IN THE LONG
TERM”**

ME

#1

**DDD IS NOT THE RIGHT
TOOL FOR SIMPLE
PROJECTS**

#2

**DDD IS NOT THE BEST FIT
FOR PROOF OF CONCEPT
OR THROWAWAY
PROJECTS**

#3

DDD HELPS

HOLISTICALLY

3 PATTERNS

THAT YOU CAN IMPLEMENT

IN YOUR PROJECT

TOMORROW

#1

**ALWAYS KEEP A VALID
STATE IN THE MEMORY**

```
type Invoice struct {  
    Number      string  
    Positions   []InvoicePosition  
    TotalAmount int  
}
```

```
type InvoicePosition struct {  
    Product  string  
    Quantity int  
    Value    int  
    TaxRate  int  
}
```

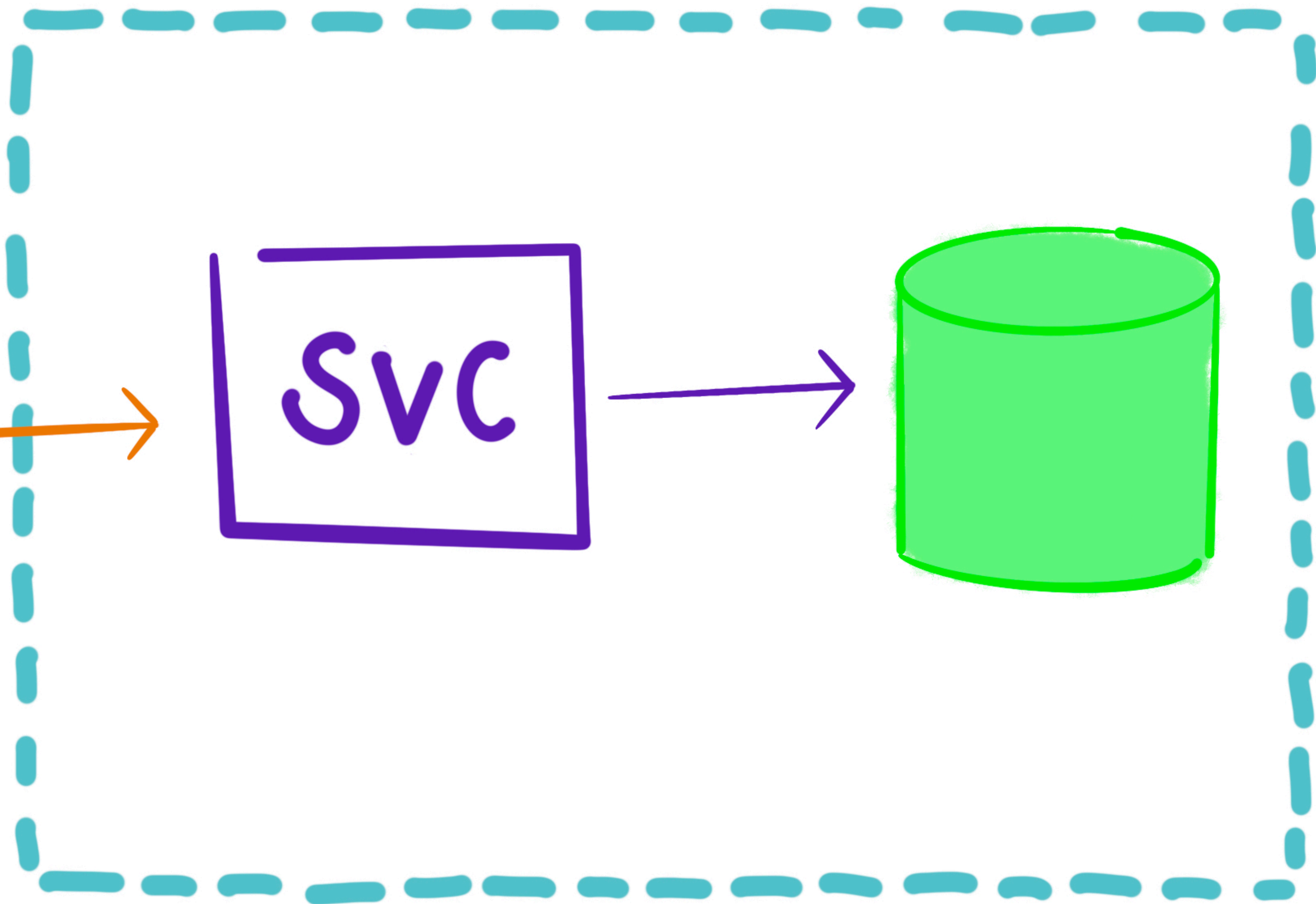
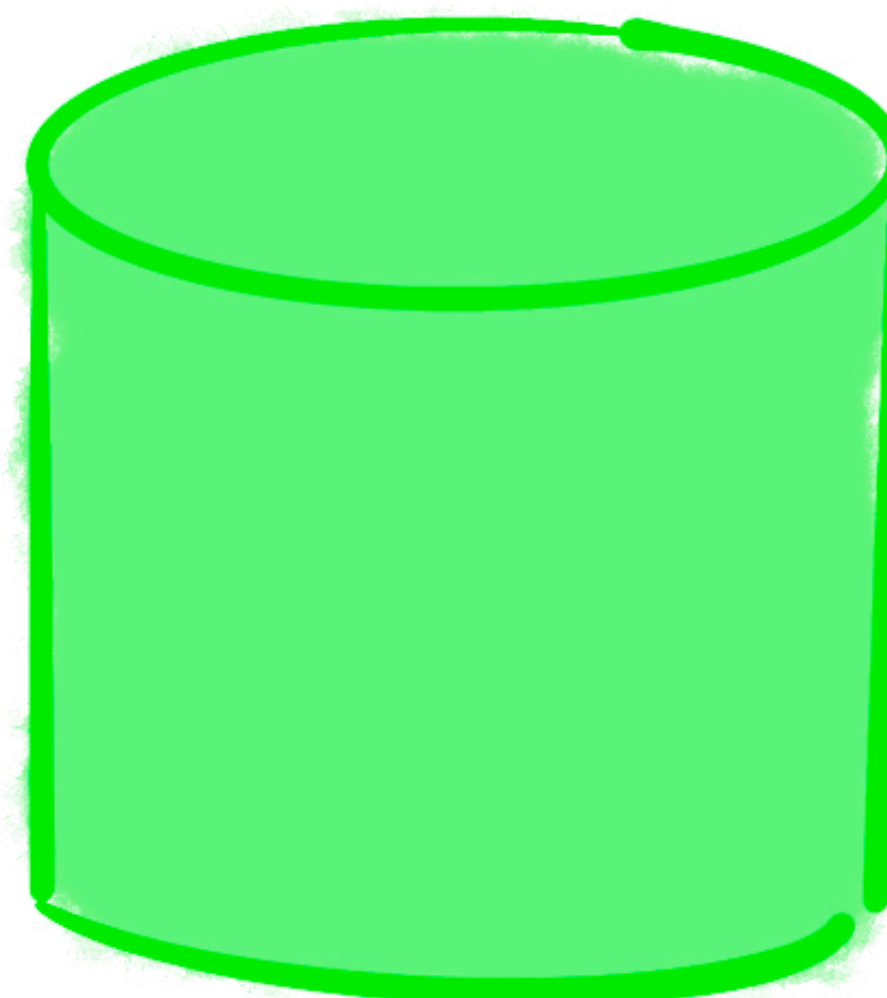
```
type Invoice struct {  
    Number      string  
    Positions   []InvoicePosition  
    TotalAmount int  
}
```

```
type InvoicePosition struct {  
    Product  string  
    Quantity int  
    Value    int  
    TaxRate  int  
}
```

```
inv.Positions = append(  
    inv.Positions,  
    InvoicePosition{  
        Product:  "Invalid product",  
        Quantity: -2,  
        Value:    -100,  
    },  
)
```




SVC





est. 1965

ENCAPSULATION

```
package invoice
```

```
type Invoice struct {
```

```
    number string
```

```
    positions []InvoicePosition
```

```
}
```

```
func NewInvoice(number string, positions []InvoicePosition) (*Invoice, error) {  
    if number == "" {  
        return nil, fmt.Errorf("number cannot be empty")  
    }  
    if len(positions) == 0 {  
        return nil, fmt.Errorf("positions cannot be empty")  
    }  
    for _, position := range positions {  
        if position.IsZero() {  
            return nil, fmt.Errorf("position cannot be empty")  
        }  
    }  
    return &Invoice{  
        number:    number,  
        positions: positions,  
    }, nil  
}
```

```
func NewInvoicePosition(
    product string, quantity int, value int, taxRate int) (InvoicePosition, error) {
    if product == "" {
        return InvoicePosition{}, fmt.Errorf("product cannot be empty")
    }
    if quantity <= 0 {
        return InvoicePosition{}, fmt.Errorf("quantity must be greater than 0")
    }
    if value <= 0 {
        return InvoicePosition{}, fmt.Errorf("value must be greater than 0")
    }
    if taxRate < 0 {
        return InvoicePosition{}, fmt.Errorf("taxRate must be greater than or equal to 0")
    }
    return InvoicePosition{
        product:  product,
        quantity: quantity,
        value:    value,
        taxRate:  taxRate,
    }, nil
}
```

```
func (i *Invoice) AddPosition(position InvoicePosition) error {  
    if position.IsZero() {  
        return fmt.Errorf("position cannot be empty")  
    }  
  
    i.positions = append(i.positions, position)  
    return nil  
}
```

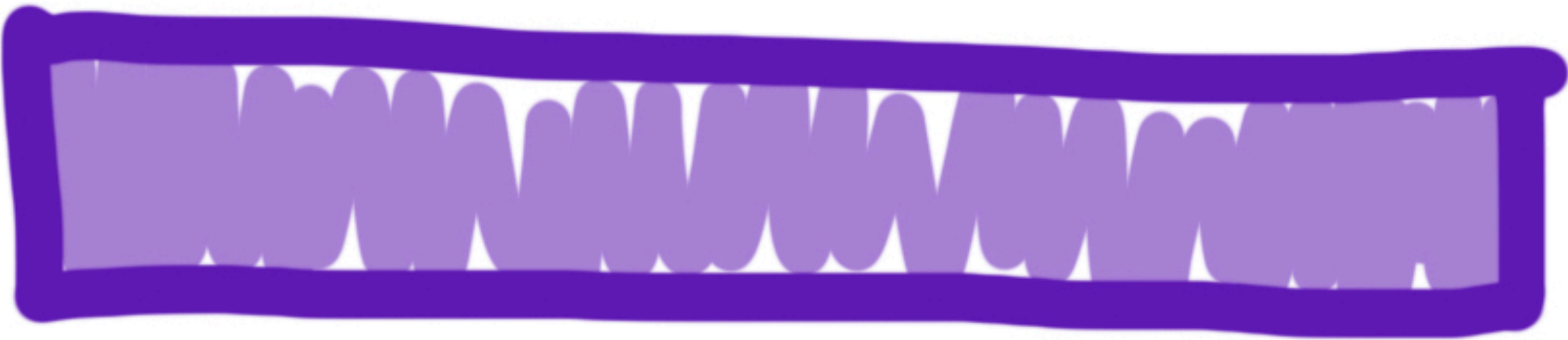
HOW IT HELPS?

#2

**KEEP THE DOMAIN PACKAGE
DATABASE AGNOSTIC**

database

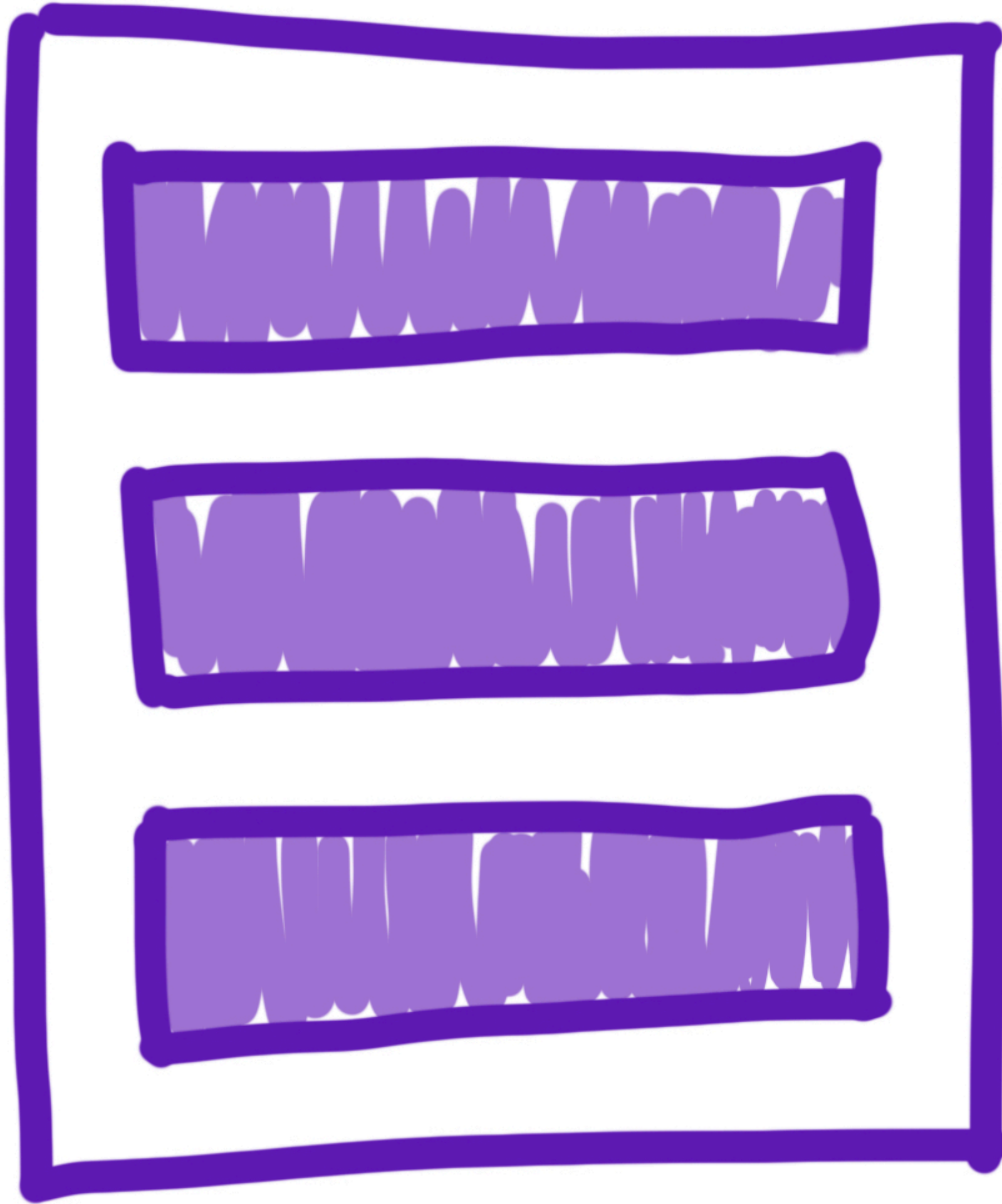
logic



domain

logic

domain layer



adapters layer



```
package invoice
```

```
type Repository interface {
```

```
    CreateInvoiceDraft(ctx context.Context, invoice Invoice) error
```

```
    UpdateInvoiceDraft(
```

```
        ctx context.Context,
```

```
        invoiceNumber InvoiceNumber,
```

```
        updateFn func(h *Invoice) (*Invoice, error),
```

```
    ) error
```

```
}
```

```
type InvoiceRepositoryStub struct {
    Invoices map[InvoiceNumber]Invoice
    lock sync.Mutex
}

func (s *InvoiceRepositoryStub) CreateInvoiceDraft(ctx context.Context, invoice Invoice) error {
    s.lock.Lock()
    defer s.lock.Unlock()

    if _, ok := s.Invoices[InvoiceNumber(invoice.Number())]; ok {
        return fmt.Errorf("invoice already exists")
    }

    s.Invoices[InvoiceNumber(invoice.Number())] = invoice
    return nil
}
```

#3

REFLECT YOUR BUSINESS

LOGIC LITERALLY

```
type Invoice struct {  
    number InvoiceNumber  
  
    positions []InvoicePosition  
  
    buyer Company  
    seller Company  
  
    issueDate      time.Time  
    dateOfService time.Time  
    dueDate        time.Time  
  
    isIssued bool  
}
```

```
type Invoice struct {  
    number InvoiceNumber  
  
    positions []InvoicePosition  
  
    buyer Company  
    seller Company  
  
    issueDate      time.Time  
    dateOfService time.Time  
    dueDate        time.Time  
  
    isIssued bool  
}
```


***CREDIT NOTE IS A
SPECIAL TYPE OF
INVOICE. INVOICE
POSITIONS VALUE CAN
BE POSITIVE AND
NEGATIVE. BUYER AND
SELLER NEED TO BE
THE SAME AS THE
REFERENCED INVOICE***

CREDIT NOTE IS A SPECIAL TYPE OF INVOICE. INVOICE POSITIONS VALUE CAN BE POSITIVE AND NEGATIVE. BUYER AND SELLER NEED TO BE THE SAME AS THE REFERENCED INVOICE

```
func NewCreditNote(  
    referencedInvoice *Invoice,  
    positions []CreditNotePosition,  
    dueDate time.Time,  
) (*CreditNote, error) {  
    // ...  
    return &CreditNote{  
        number: "CN" + refere  
        referenceInvoiceNumber: referencedInvoi  
        positions: positions,  
        buyer: referencedInvoice.Buyer(),  
        seller: referencedInvoice.Seller(),  
        issueDate: time.Now(),
```

**CAN I NOW ADD DDD
TO MY CV?**

**DOMAIN-DRIVEN
DESIGN IN GO
MYTHS**

**1. GO IS A SIMPLE LANGUAGE, YOU SHOULD
NOT USE ANY COMPLEX PATTERNS LIKE DDD**

2. DDD SHOULD BE COMPLEX AND

HARD

3. GO IS NOT A GOOD LANGUAGE FOR

BUSINESS LOGIC AND DDD

GENERALISATION

1. GO IS A SIMPLE LANGUAGE,

YOU SHOULD NOT USE ANY

~~COMPLEX PATTERNS LIKE DDD~~

ONLY FOR MORE COMPLEX

PROJECTS

**2. DDD MAKES CODE MORE
COMPLEX AND HARDER TO
READ IF YOU WILL USE IT FOR
SIMPLE DOMAINS**

~~3. GO IS NOT A GOOD
LANGUAGE FOR BUSINESS
LOGIC AND DDD~~

MYTH BUSTED

 **Tactical DDD**

Strategic DDD

modularisation

how to split microservices

architecture

DON'T NEED DDDD YET?



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

- [🖥 Slides](#)
- [👨 Wild Workouts - fully functional DDD Go project](#)
- [📖 Go With The Domain - our e-book showing how to use DDD in Go](#)
- [📦 The Repository pattern: a painless way to simplify your Go service logic](#)
- [👍 Introduction to DDD Lite](#)



<https://tdl.is/gc24/>

ThreeDotsLabs / **wild-workouts-go-ddd-example**

<> Code Issues 30 Pull requests 2 Discussions

Go DDD example application. Complete project to show how to apply DDD, Clean Architecture, and CQRS by practical refactoring.

threedots.tech

MIT license

4.9k stars 456 forks 89 watching 5 Branches 12 Tags

Activity Custom properties

Public repository



<https://tdl.is/gc24/>

THANKS!

<https://tdl.is/gc24/>

