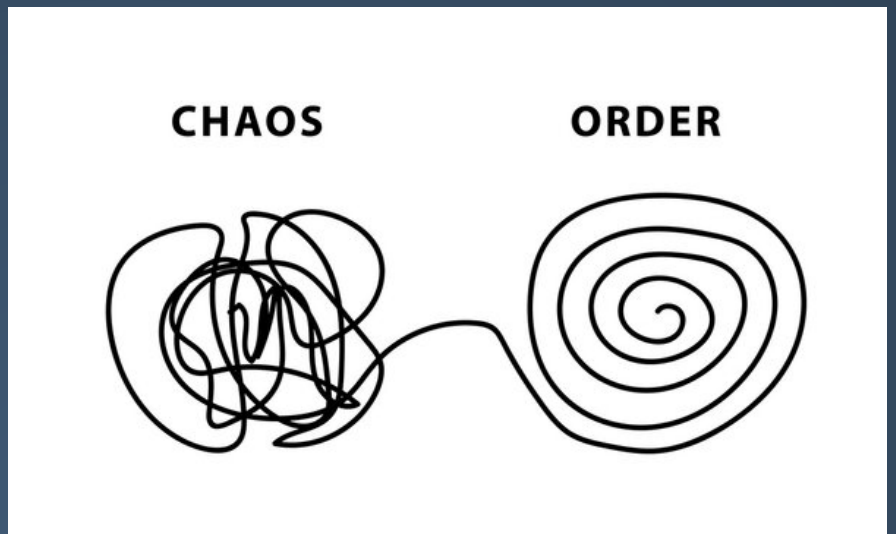
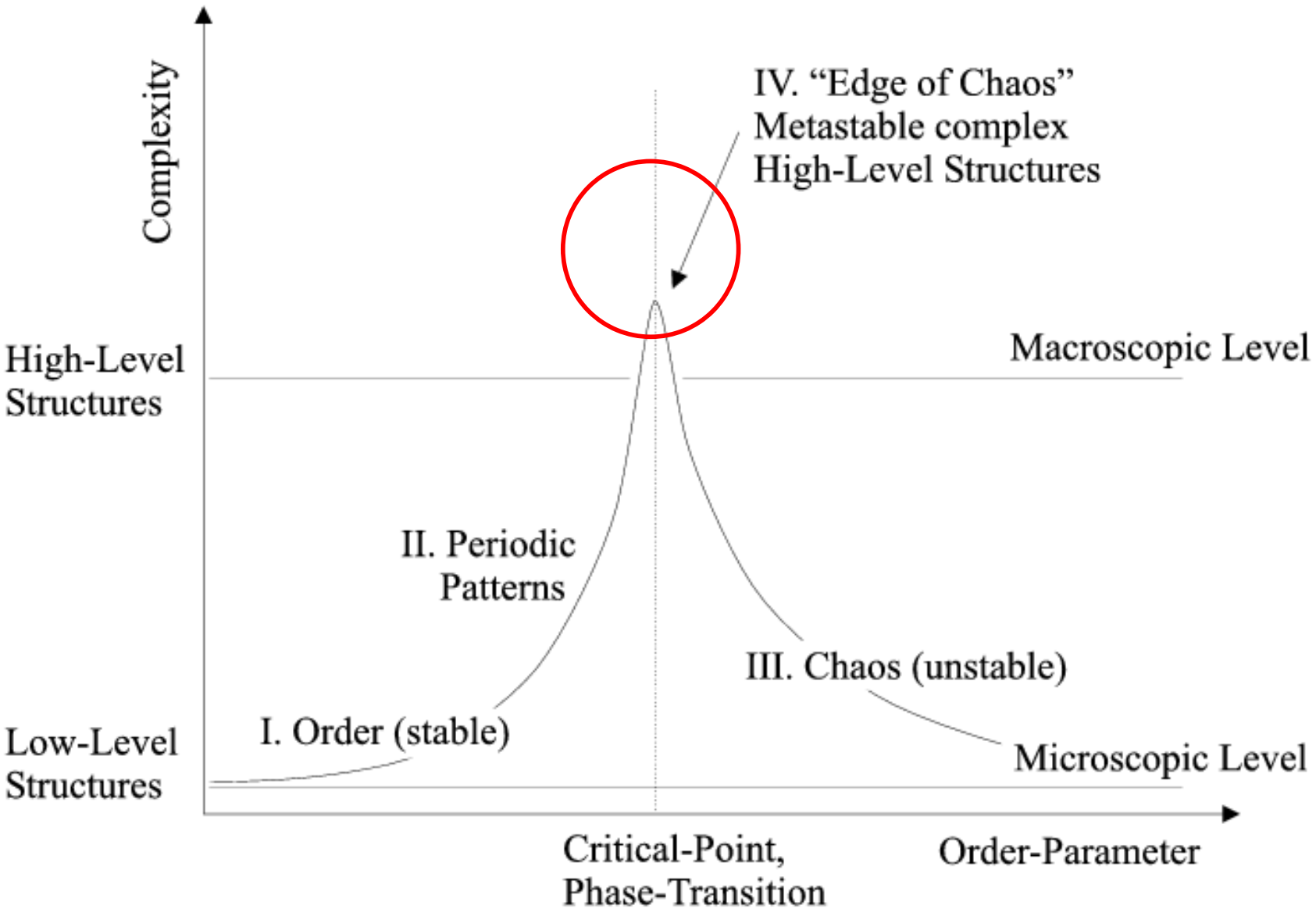


Software Development at the Edge of Chaos

Kamil Sebastian Mijacz

Edge of Chaos



about me

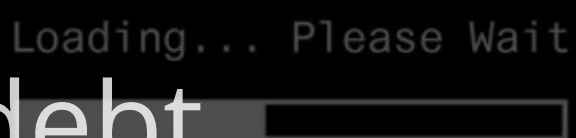
- kamil.mijacz@gmail.com
- Software Architect British Council
- ** lat w technologii .NET, web, js
- "były" uczestnik Białostockiej Grupy <placeholder>
- JavaScript Developer PB: DevOps aka GIT
- znam Karola
- prywatnie żyje


At the edge of chaos,
unexpected outcomes occur.
The risk to survival is severe.
Michael Crichton
quotefancy

Complex software problem

Software Architecture support

Tech debt



Fight tech debt

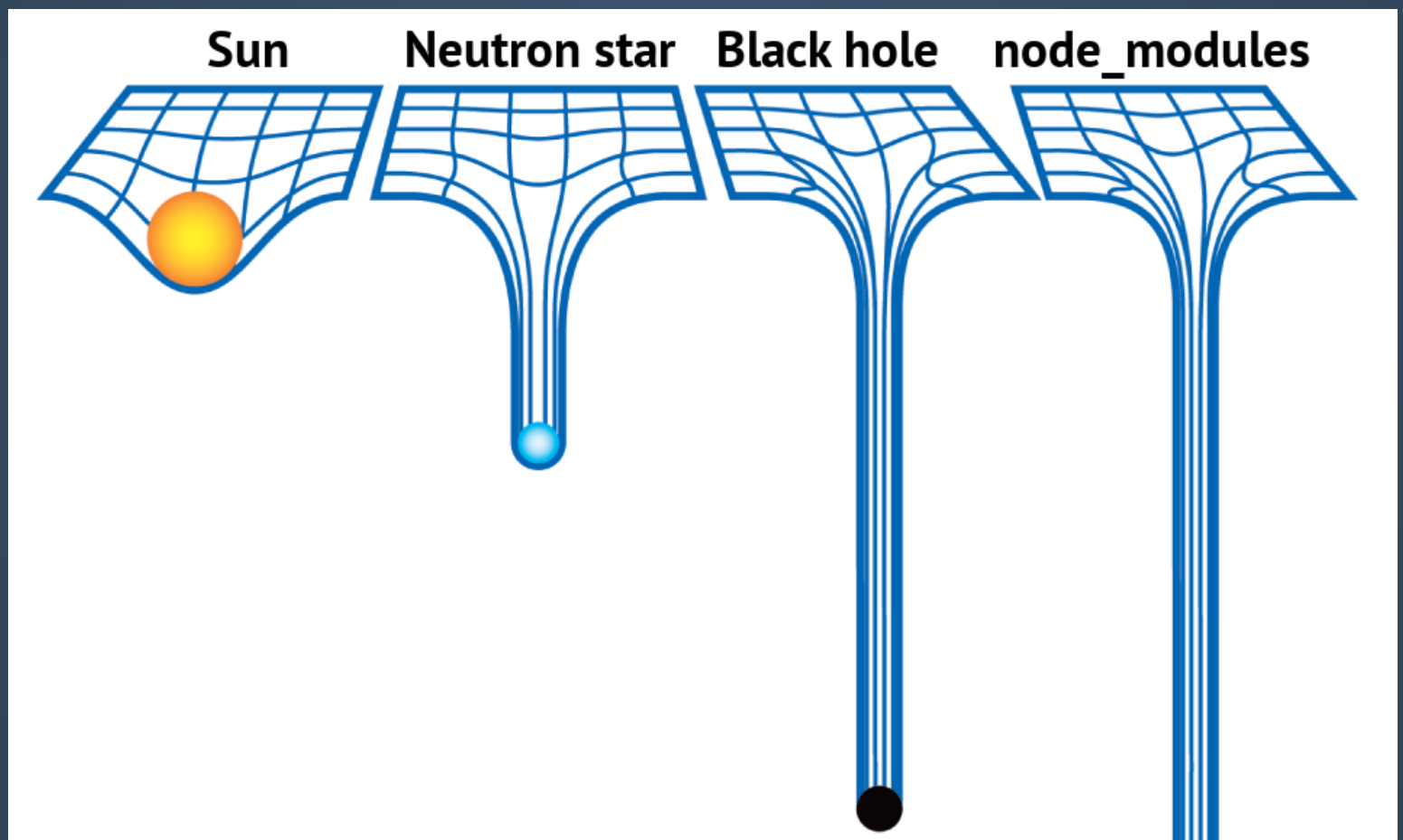
... with dependency/code analysis

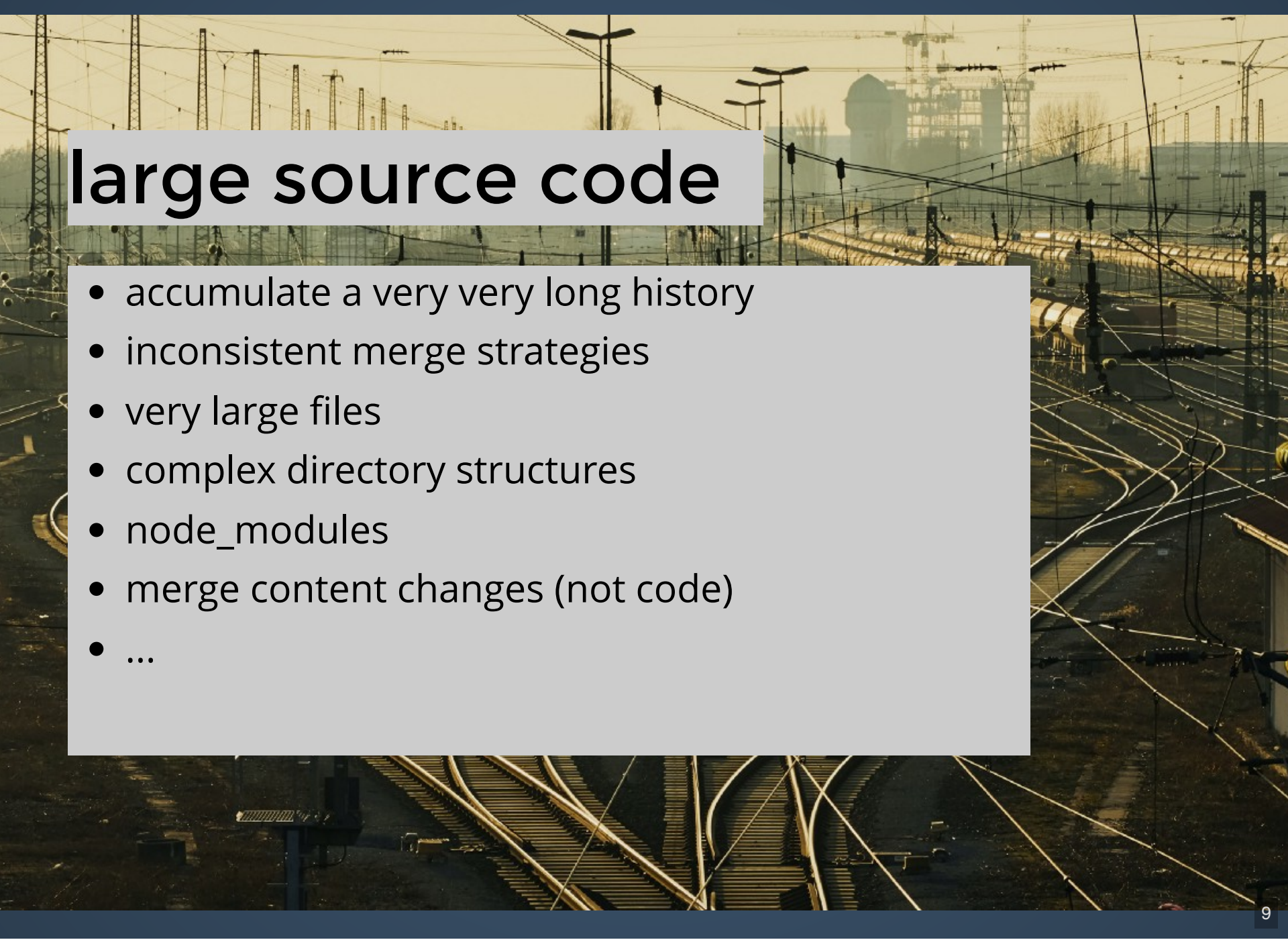

Complex Software

Complex software problem

1. software is complex by def
2. interrelated requirements to combine
3. teams need to find an effective way to work ...
4. with customers

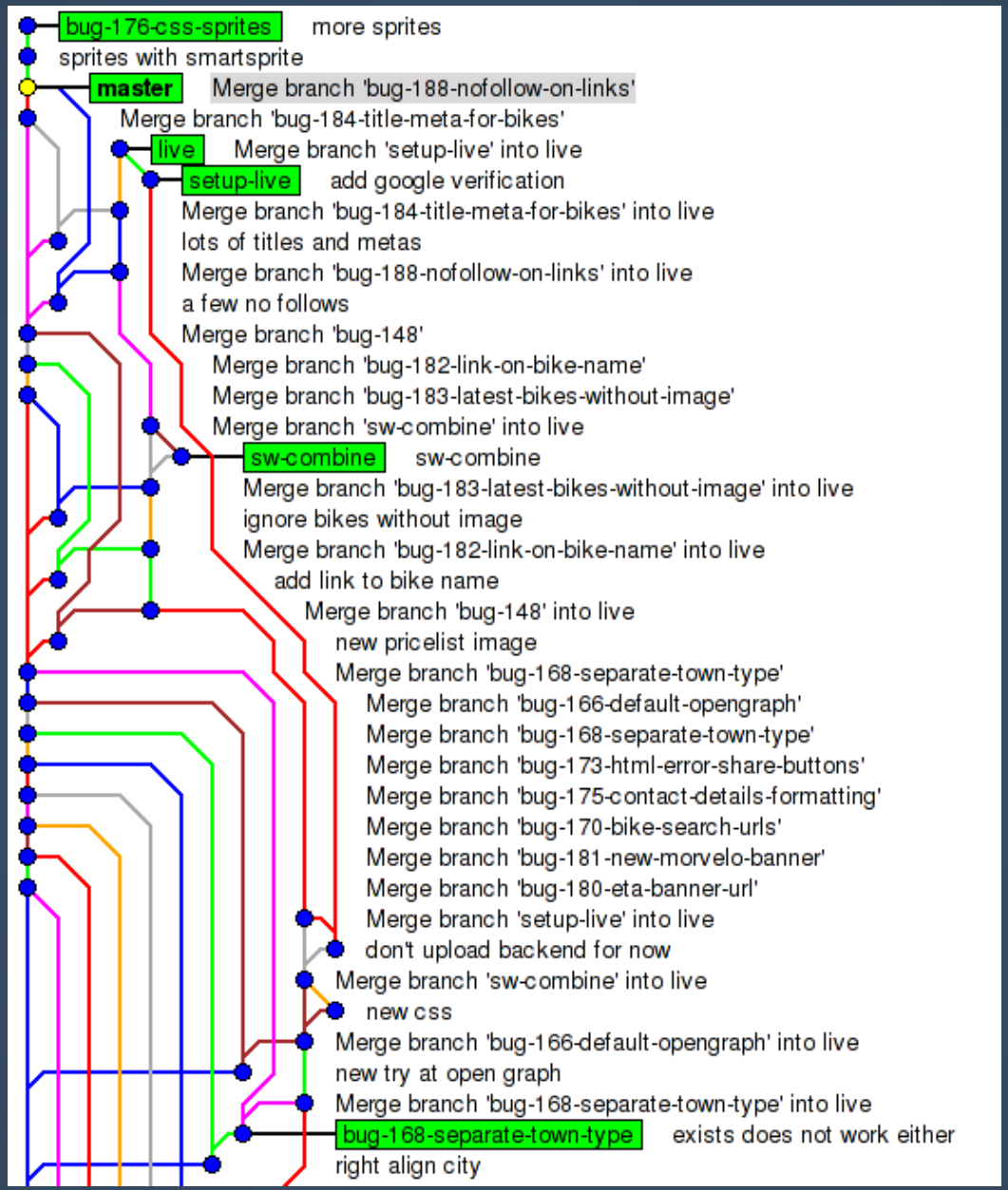
node modules





large source code

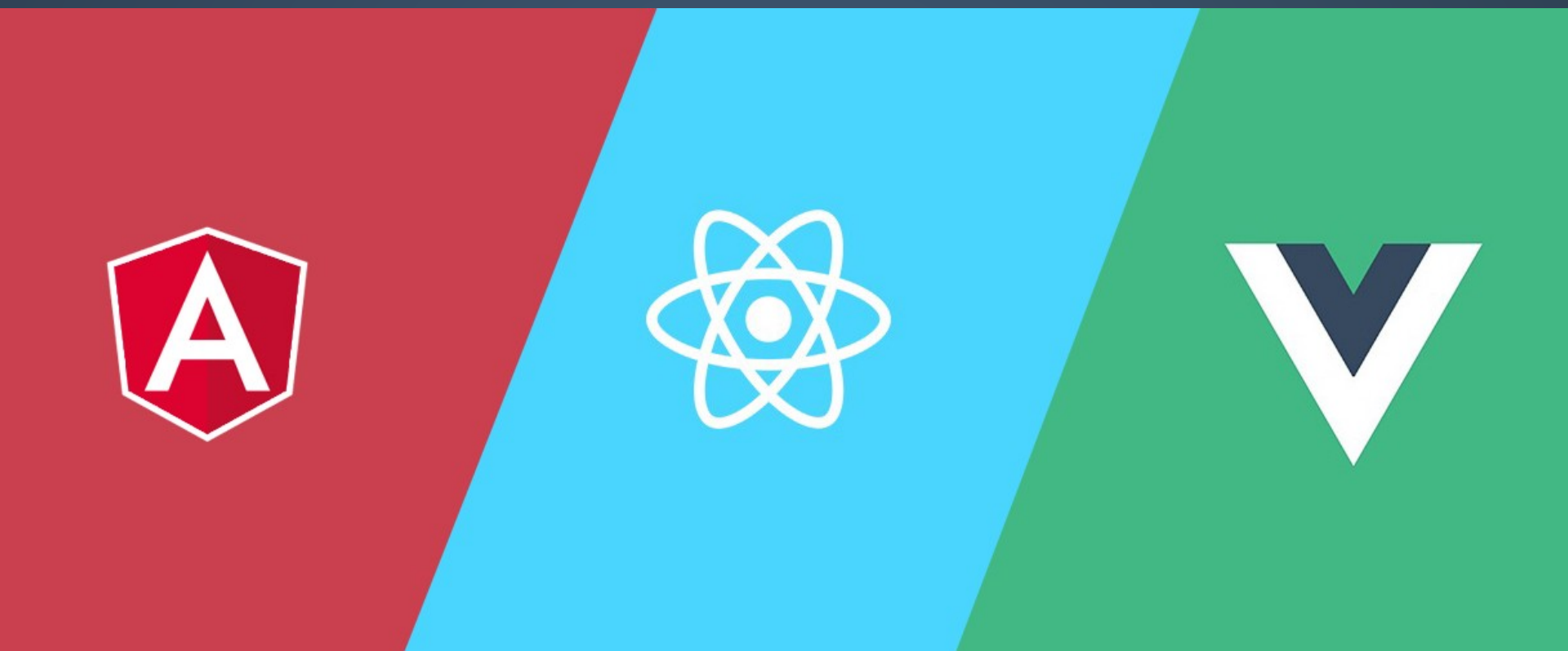
- accumulate a very very long history
- inconsistent merge strategies
- very large files
- complex directory structures
- node_modules
- merge content changes (not code)
- ...


bug-176-css-sprites
more sprites
sprites with smartsprite
master
Merge branch 'bug-188-nofollow-on-links'
Merge branch 'bug-184-title-meta-for-bikes'
live
Merge branch 'setup-live' into live
setup-live
add google verification
Merge branch 'bug-184-title-meta-for-bikes' into live
lots of titles and metas
Merge branch 'bug-188-nofollow-on-links' into live
a few no follows
Merge branch 'bug-148'
Merge branch 'bug-182-link-on-bike-name'
Merge branch 'bug-183-latest-bikes-without-image'
Merge branch 'sw-combine' into live
sw-combine
sw-combine
Merge branch 'bug-183-latest-bikes-without-image' into live
ignore bikes without image
Merge branch 'bug-182-link-on-bike-name' into live
add link to bike name
Merge branch 'bug-148' into live
new pricelist image
Merge branch 'bug-168-separate-town-type'
Merge branch 'bug-166-default-opengraph'
Merge branch 'bug-168-separate-town-type'
Merge branch 'bug-173-html-error-share-buttons'
Merge branch 'bug-175-contact-details-formatting'
Merge branch 'bug-170-bike-search-urls'
Merge branch 'bug-181-new-morvelo-banner'
Merge branch 'bug-180-eta-banner-url'
Merge branch 'setup-live' into live
don't upload backend for now
Merge branch 'sw-combine' into live
new css
Merge branch 'bug-166-default-opengraph' into live
new try at open graph
Merge branch 'bug-168-separate-town-type' into live
bug-168-separate-town-type
exists does not work either
right align city

framework "bend the knee" dependency

1. complex code
2. rules
3. "one size fits all" myth
4. a lot of abstractions
5. compatibility

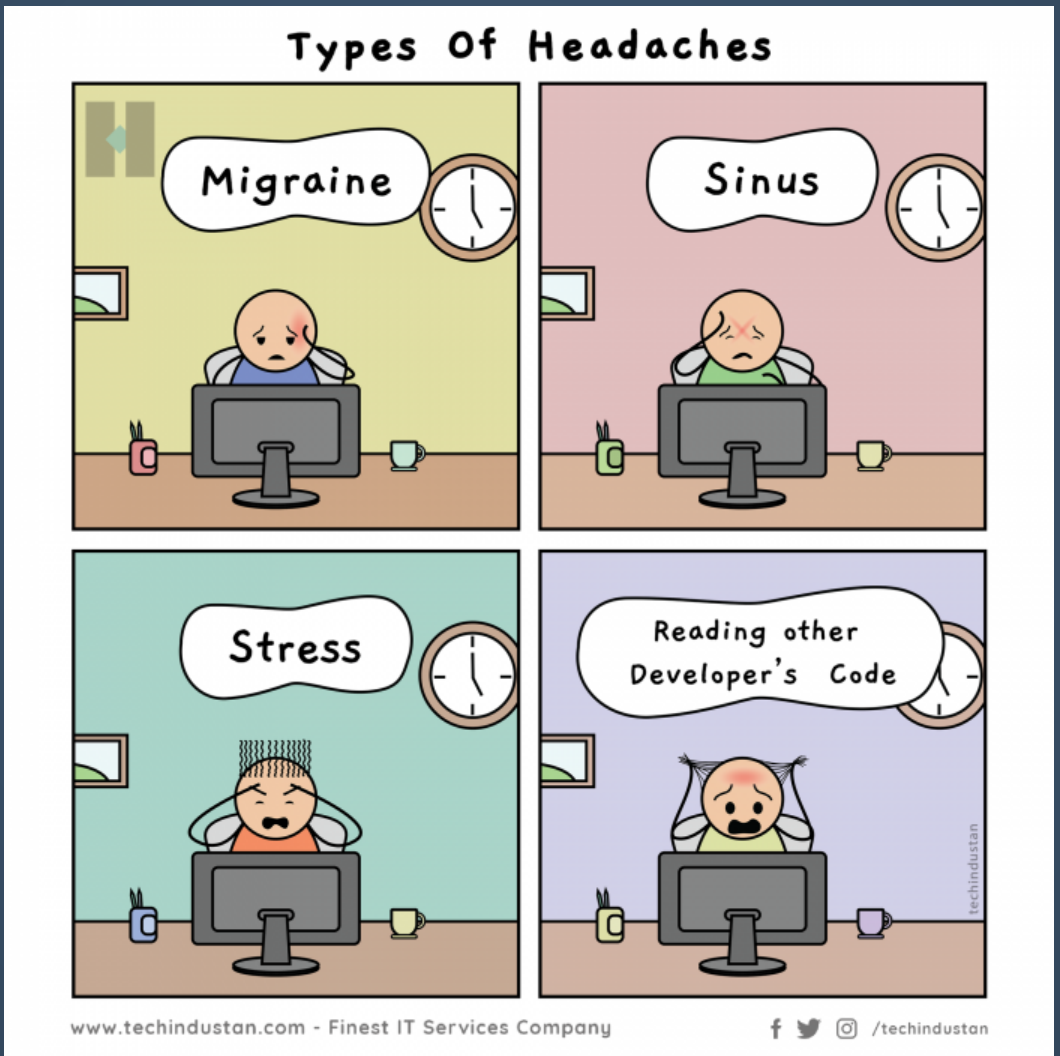
which one?




JAVASCRIPT
HD
HISTORY.COM
imgflip.com

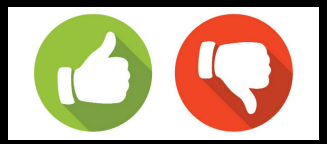
technological
information overload

Problem


Types Of Headaches
Migraine
Sinus
Stress
Reading other Developer's Code
techindustan
www.techindustan.com - Finest IT Services Company
/techindustan

```
/**
 * Invoke this method to explicitly process change detection and its side-effects.
 *
 * In development mode, `tick()` also performs a second change detection cycle to ensure that no
 * further changes are detected. If additional changes are picked up during this second cycle,
 * bindings in the app have side-effects that                    ingle change detection
 * pass.
 * In this case, Angular throws an error, sinc                   can only have one change
 * detection pass during which all change detection must complete.
 */
tick(): void {
  NG_DEV_MODE && this.warnIfDestroyed();
  if (this._runningTick) {
    const errorMessage = (typeof ngDevMode === 'undefined' || ngDevMode) ?
        'ApplicationRef.tick is called recursively' :
        '';
    throw new RuntimeError(RuntimeErrorCode.RECURSIVE_APPLICATION_REF_TICK, errorMessage);
  }

  try {
    this._runningTick = true;
    for (let view of this._views) {
      view.detectChanges();
    }
    if (typeof ngDevMode === 'undefined' || ngDevMode) {
      for (let view of this._views) {
        view.checkNoChanges();
      }
    }
  } catch (e) {
    // Attention: Don't rethrow as it could cancel subscriptions to Observables!
    this._zone.runOutsideAngular(() => this._exceptionHandler.handleError(e));
  } finally {
    this._runningTick = false;
  }
}
```

```ts
export enum OrderStatus {
  PENDING = 'PENDING', ACCEPTED = 'ACCEPTED', REJECTED = 'REJECTED'
}

export interface IOrderProps {
  productId: string
  status: OrderStatus
}

export class Order {

  constructor(private readonly props: IOrderProps, private readonly _id: string = Order.genId())

  get id(): string {
    return this._id
  }

  get productId(): string {
    return this.props.productId
  }

  get status(): OrderStatus {
    return this.props.status
  }

  public accept() {
    if (this.props.status !== OrderStatus.PENDING) {
      throw new Error('could not reject processed order')
    }
    this.props.status = OrderStatus.ACCEPTED
  }

  public reject() {
    if (this.props.status !== OrderStatus.PENDING) {
      throw new Error('could not reject processed order')
    }
    this.props.status = OrderStatus.REJECTED
```

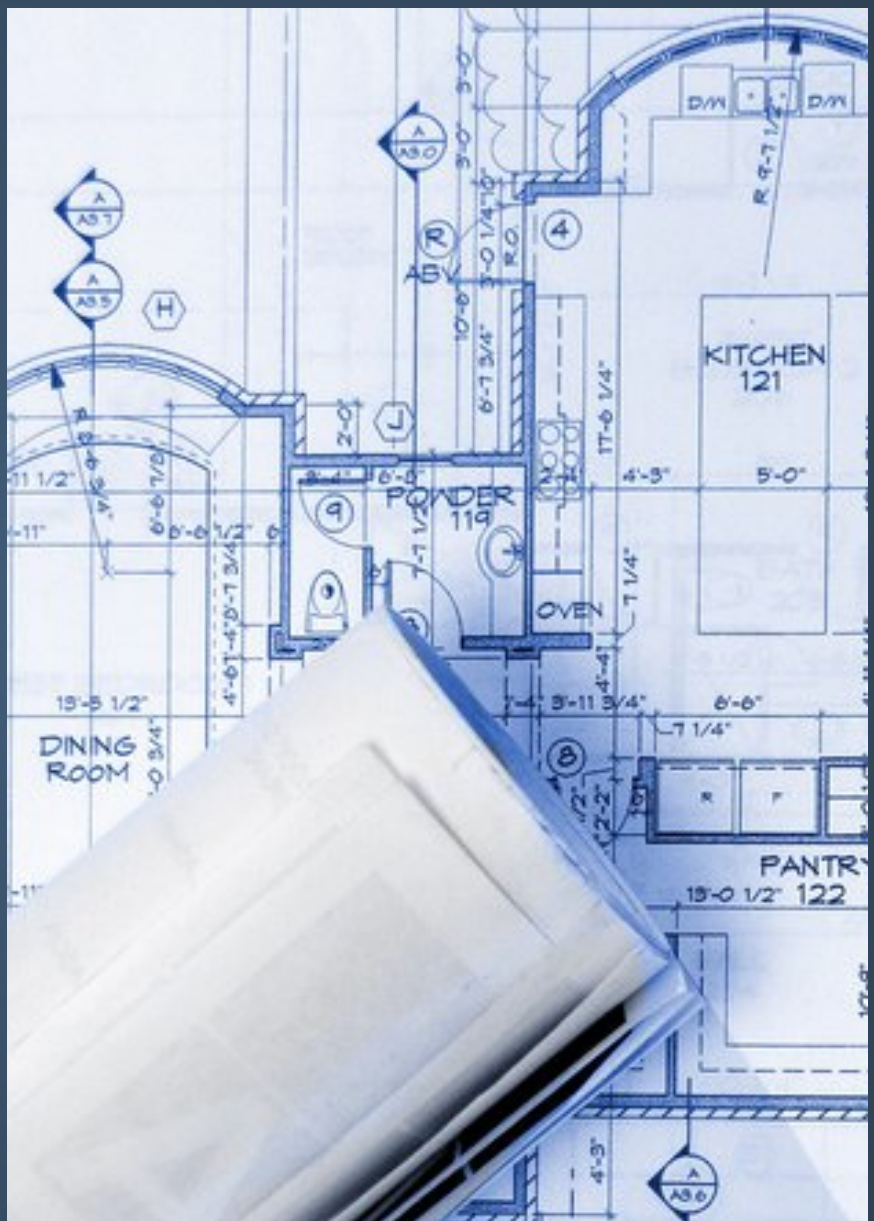


which one is better
to read?

Software Architecture

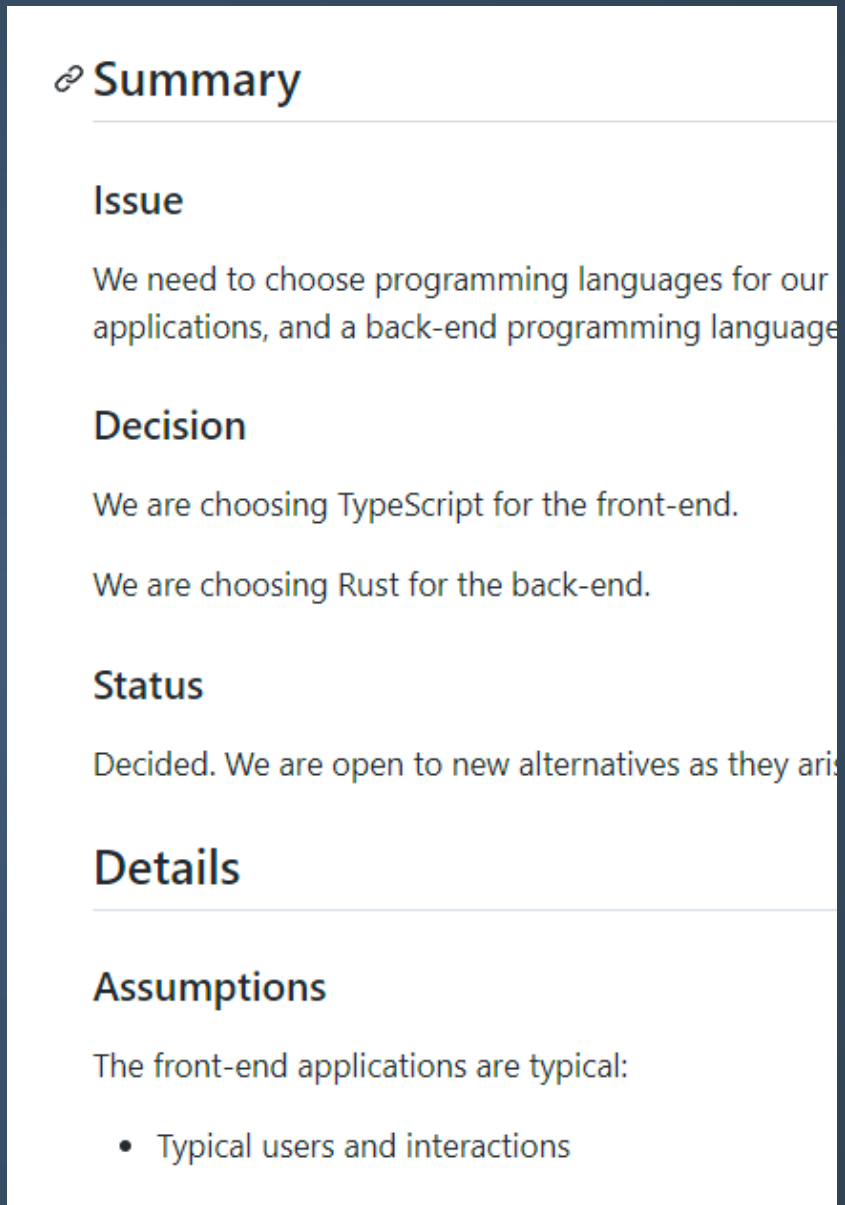
Software Architecture

... is not about structure



Software Architecture

… is about capturing decisions



Summary

Issue

We need to choose programming languages for our applications, and a back-end programming language

Decision

We are choosing TypeScript for the front-end.

We are choosing Rust for the back-end.

Status

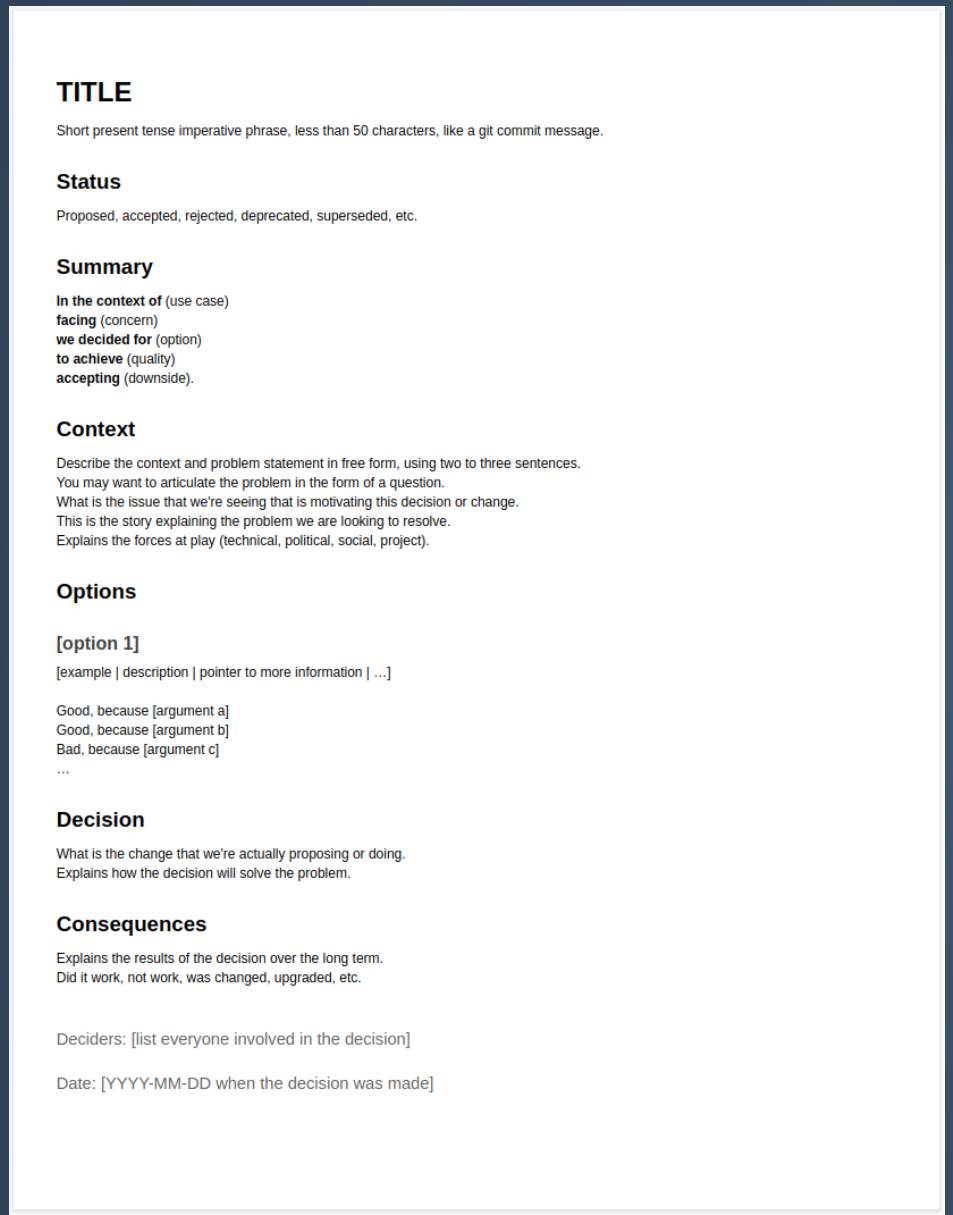
Decided. We are open to new alternatives as they aris

Details

Assumptions

The front-end applications are typical:

- Typical users and interactions



TITLE

Short present tense imperative phrase, less than 50 characters, like a git commit message.

Status

Proposed, accepted, rejected, deprecated, superseded, etc.

Summary

In the context of (use case)
facing (concern)
we decided for (option)
to achieve (quality)
accepting (downside).

Context

Describe the context and problem statement in free form, using two to three sentences.
You may want to articulate the problem in the form of a question.
What is the issue that we're seeing that is motivating this decision or change.
This is the story explaining the problem we are looking to resolve.
Explains the forces at play (technical, political, social, project).

Options

[option 1]

[example | description | pointer to more information | ...]

Good, because [argument a]
Good, because [argument b]
Bad, because [argument c]
...

Decision

What is the change that we're actually proposing or doing.
Explains how the decision will solve the problem.

Consequences

Explains the results of the decision over the long term.
Did it work, not work, was changed, upgraded, etc.

Deciders: [list everyone involved in the decision]

Date: [YYYY-MM-DD when the decision was made]

ADR

Architecture Decision Record

>>

Architecture Decision Log

...

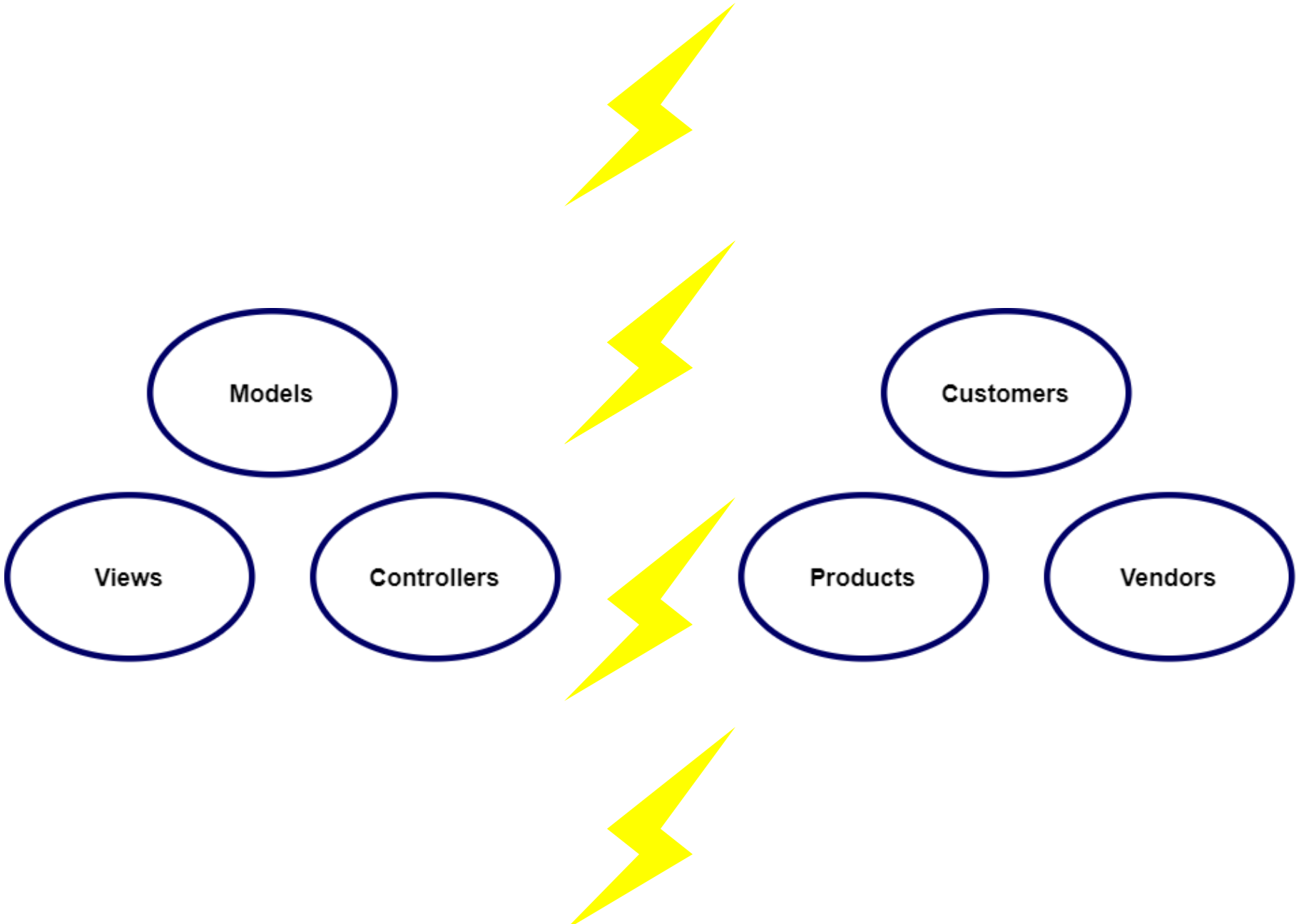
Architecturally-Significant Requirement

...

Architecture Knowledge Management

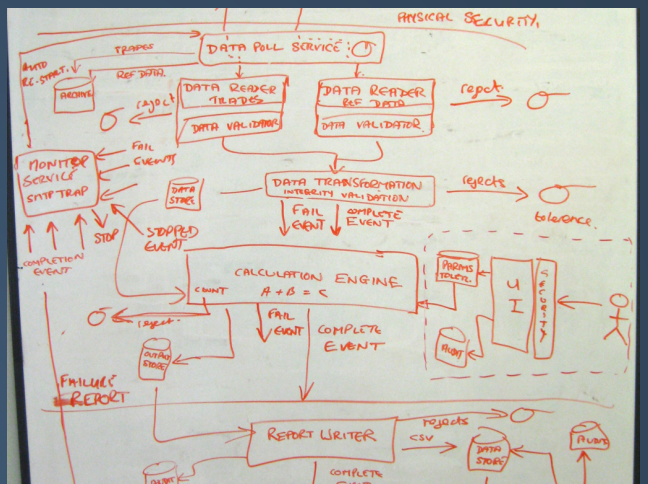
Software
Architecture

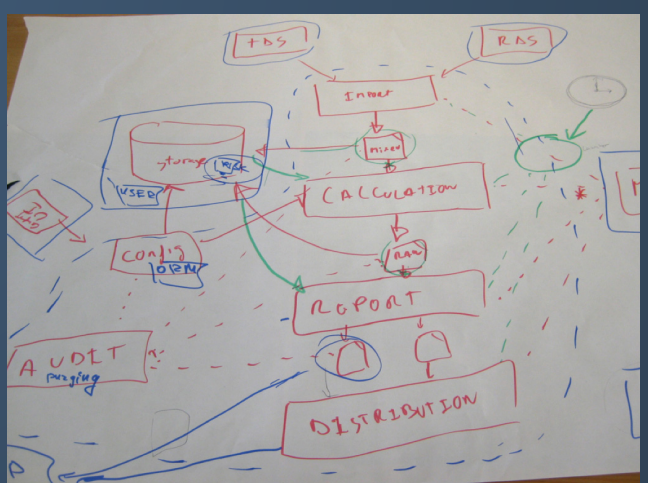
... should
be
screaming


Models
Views
Controllers
Customers
Products
Vendors

Software
Architecture

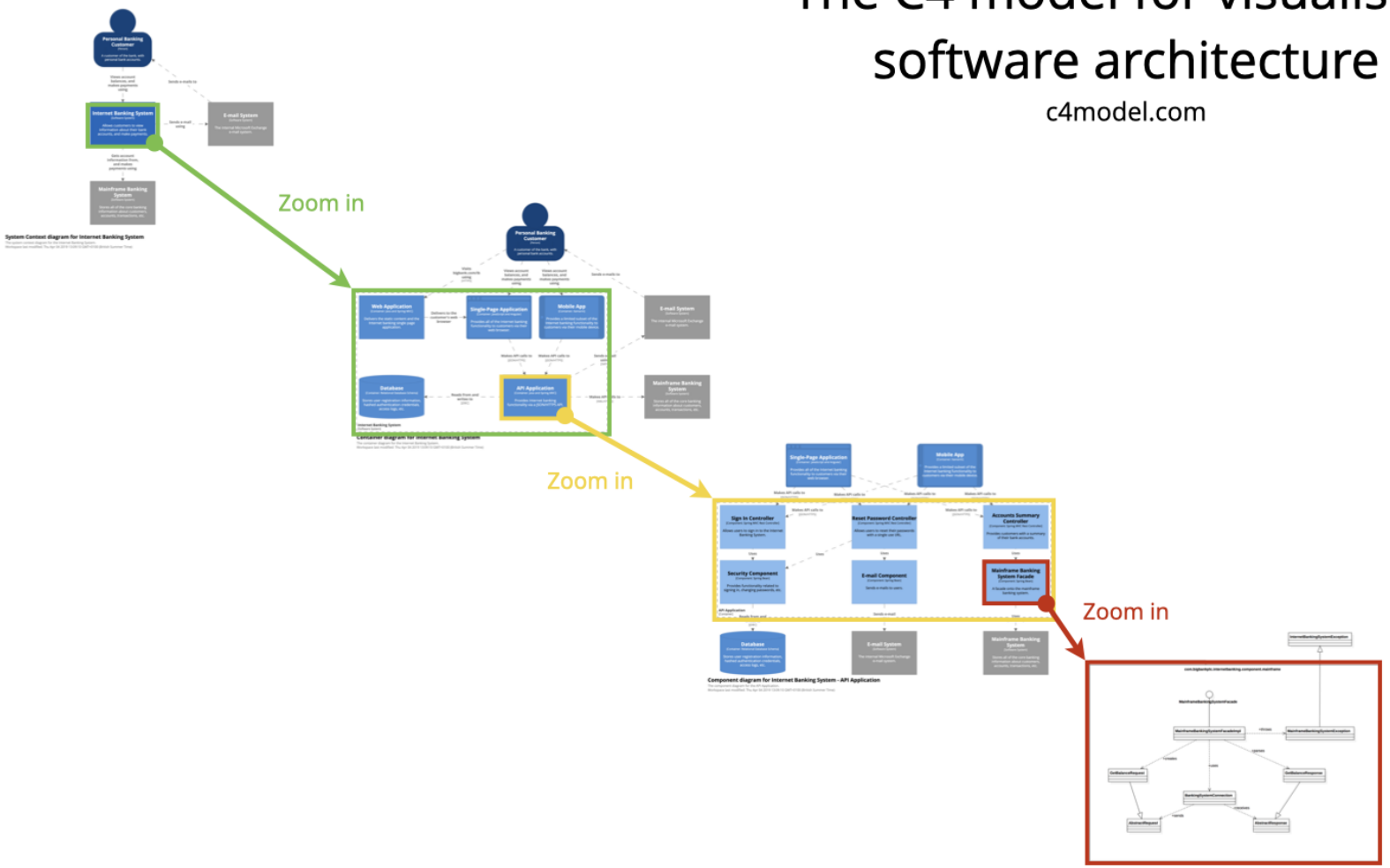
... should be documented





The C4 model for visualising software architecture

c4model.com

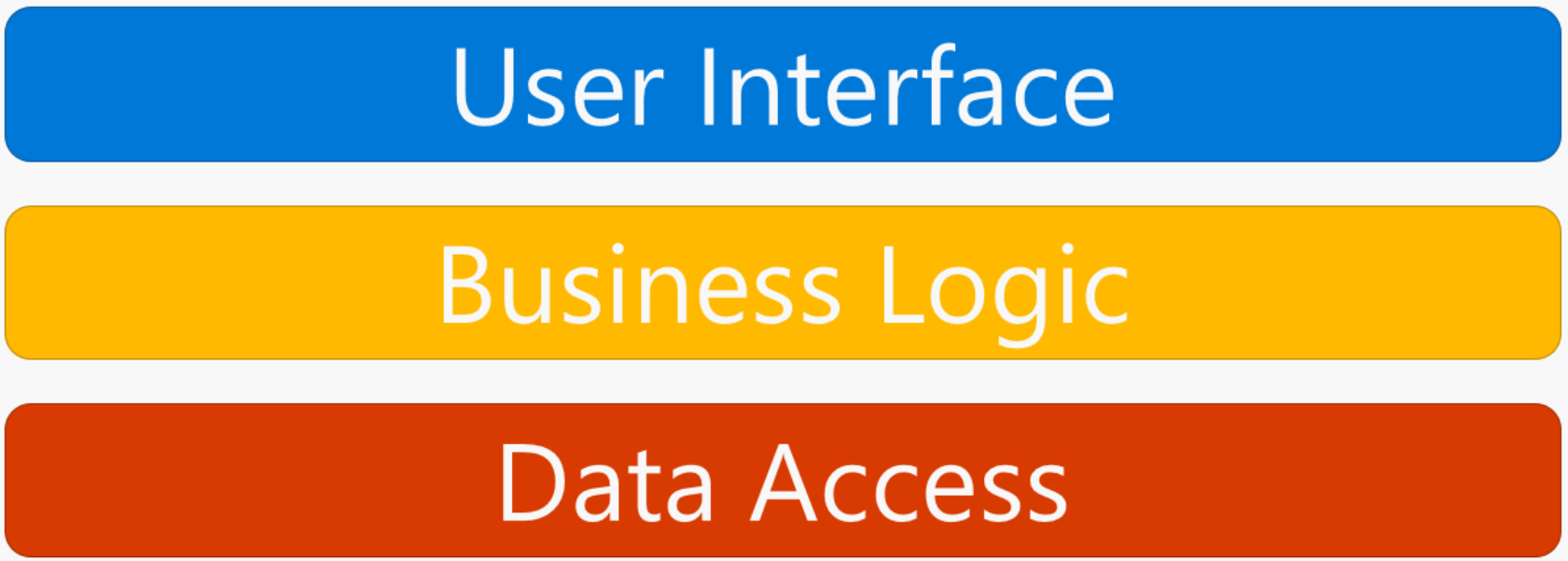


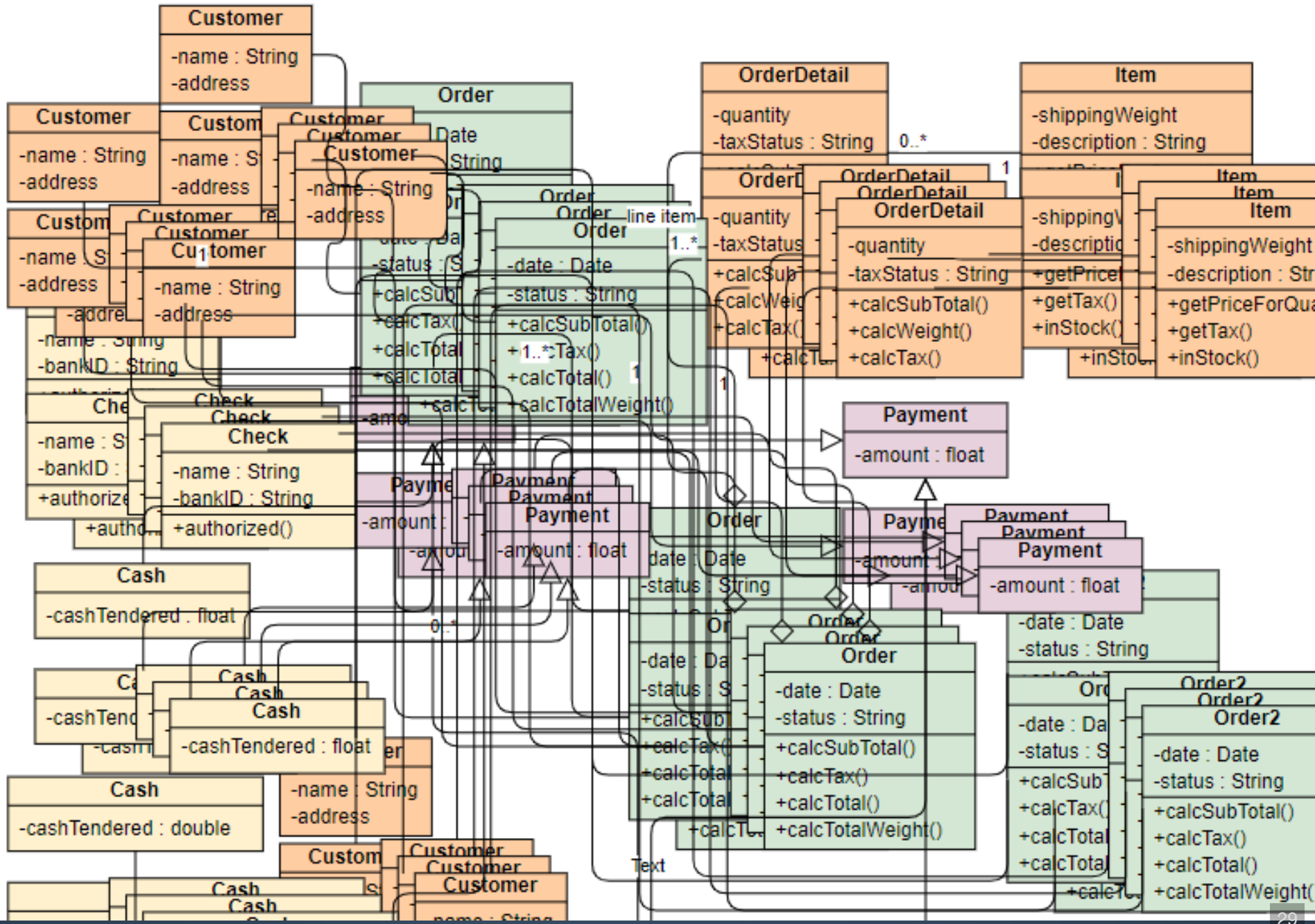
Level 1	Level 2	Level 3	Level 4
Context	Containers	Components	Code

Software
Architecture

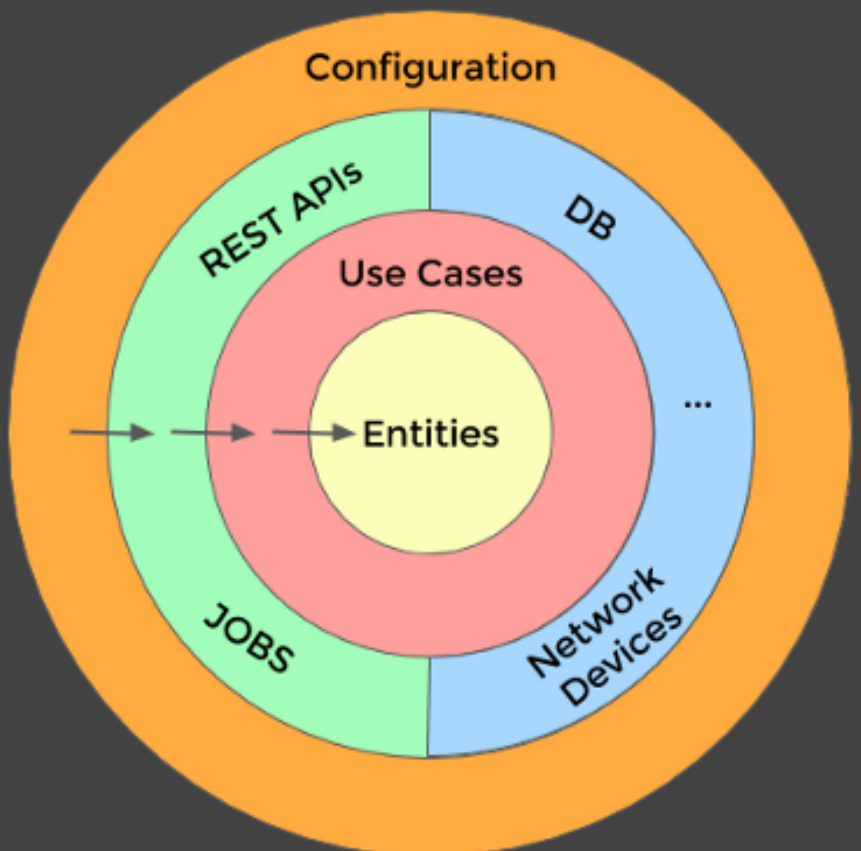
... should be clean

Application Layers




Customer
-name : String
-address
Order
-date : Date
-status : String
+calcSubTotal()
+calcTax()
+calcTotal()
+calcTotalWeight()
OrderDetail
-quantity
-taxStatus : String
+calcSubTotal()
+calcWeight()
+calcTax()
Item
-shippingWeight
-description : String
+getPriceForQuantity()
+getTax()
+inStock()
line item
0..*
1..*
Payment
-amount : float
Check
-name : String
-bankID : String
+authorized()
Cash
-cashTendered : float
-cashTendered : double
Order2
-date : Date
-status : String
+calcSubTotal()
+calcTax()
+calcTotal()
+calcTotalWeight()
Text

A.k.a (unclebob's style)



\+ Core: business logic

Entrypoints: access to the application

Data Providers: retrieve and store information

Configuration: wires everything together

** arrows represent dependency*


DEBT

Technical debt
is metaphor

Technical debt

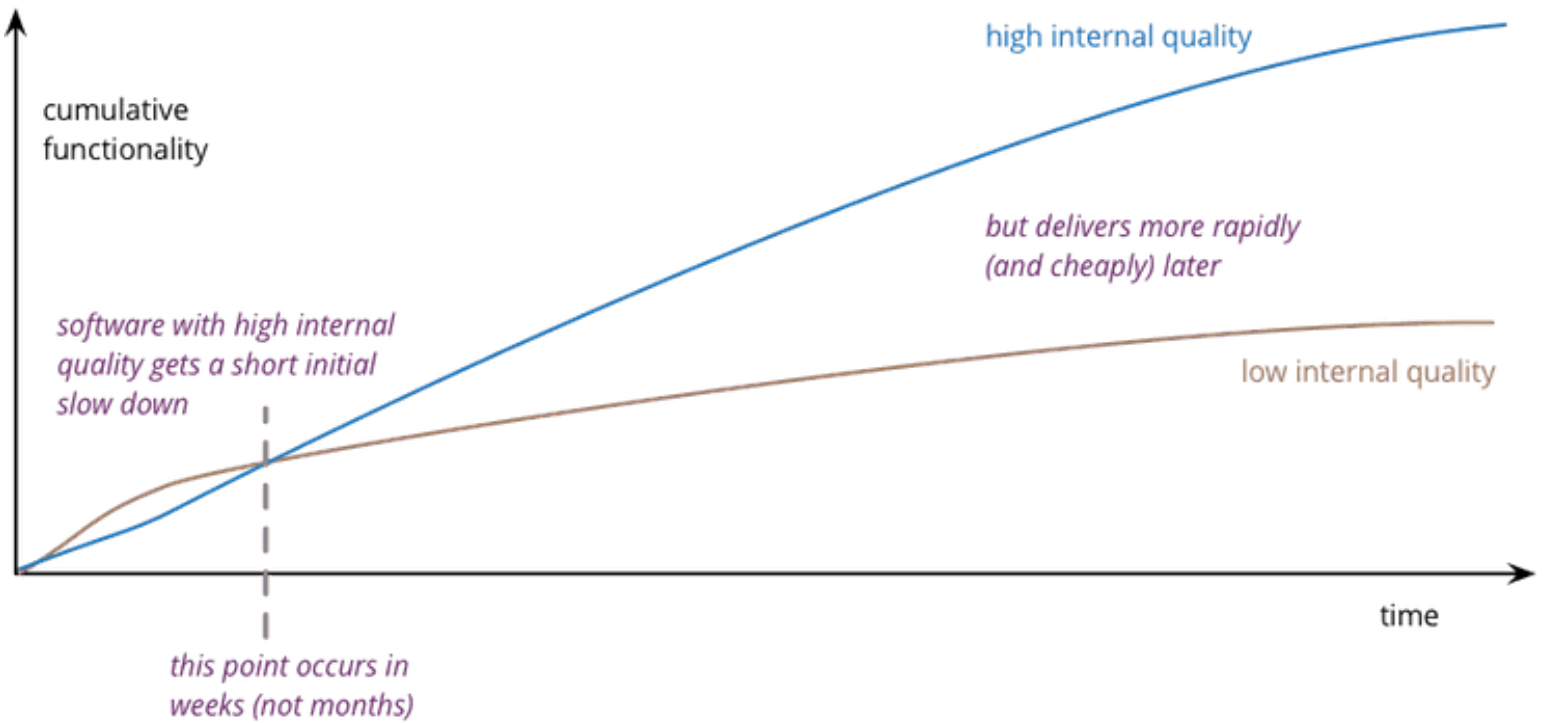
is making things easier and faster loosing quality

Technical debt

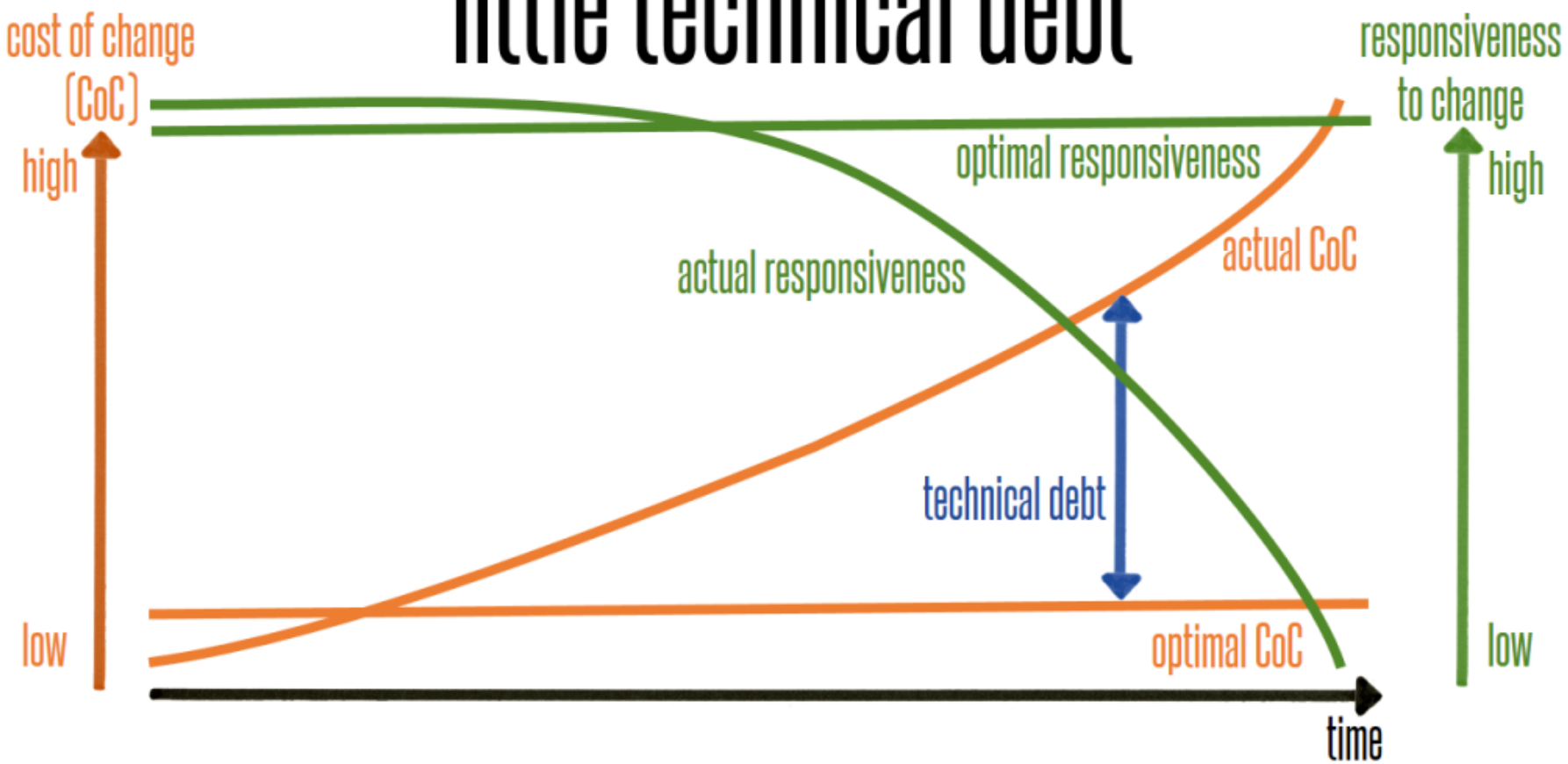
is complicating

maintenance

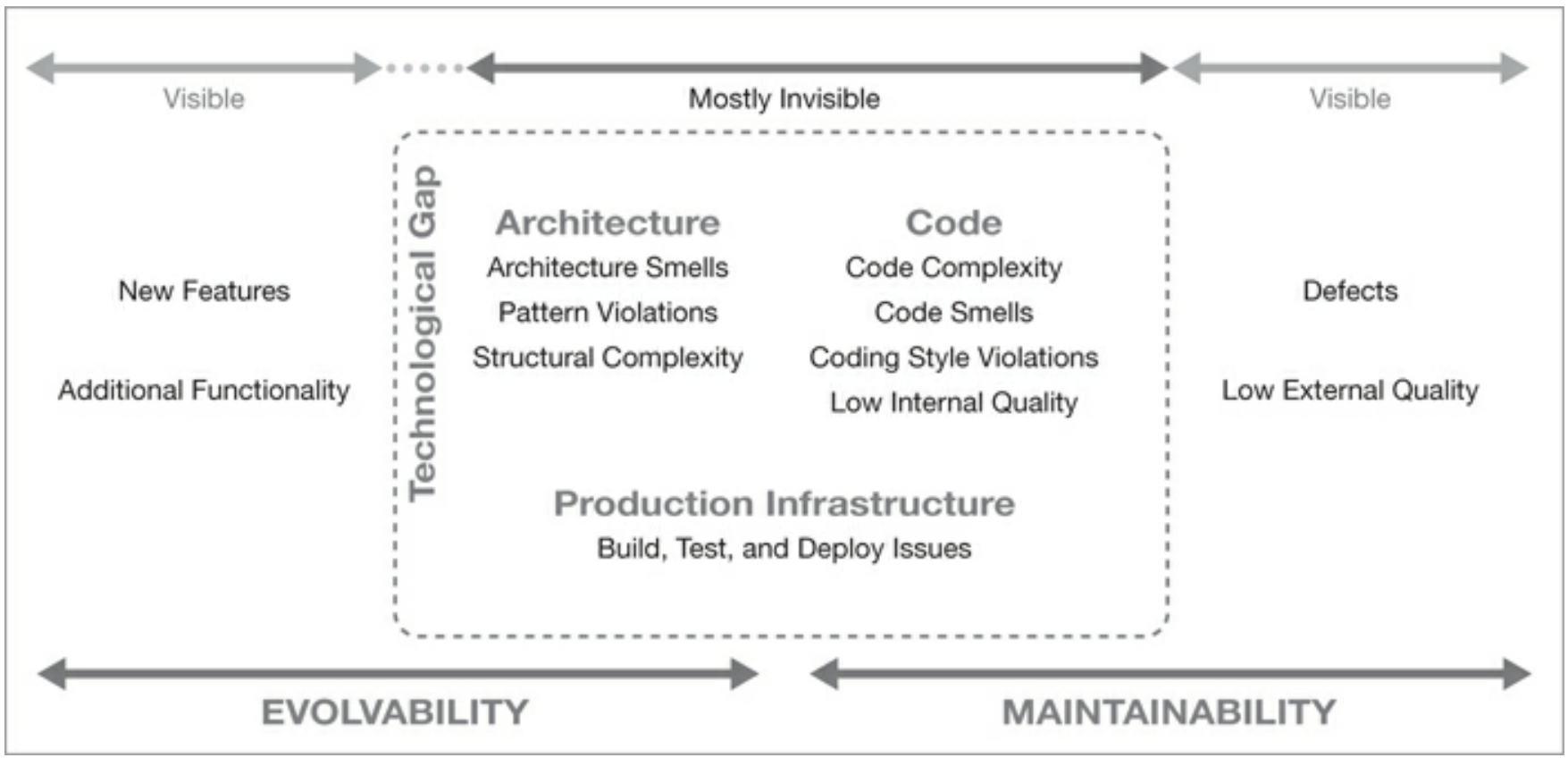
Quality investment



little technical debt



Technical debt landscape



Technical debt
what to do?

Track/Analyse	**Easy**
Manage	**Medium**
Payoff (aka. Refactor)	**Hard**

Summary

- Software is Complex
- Software Architecture can help with handling complexity
- Software Architecture is a process (continuous)
- Tech debt can mess with software
- We need to track it and manage it

https://github.com/Banndzior

kamil.mijacz@gmail.com

Thank you!

kamil.mijacz@gmail.com