BC/NAV TECH DAYS 2022

mibuso.com

10 YEAR ANNIVERSARY

www.bctechdays.com

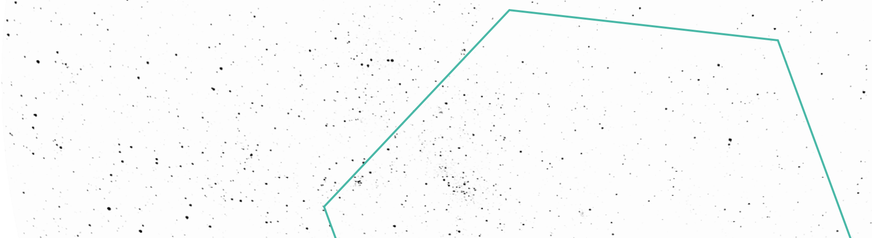
Multi-Application Architecture

Kamil Sáček, Jeremy Vyska
NAVERTICA a.s., Spare Brained Ideas

Multi-Application Architecture

What does it mean „Architecture“?

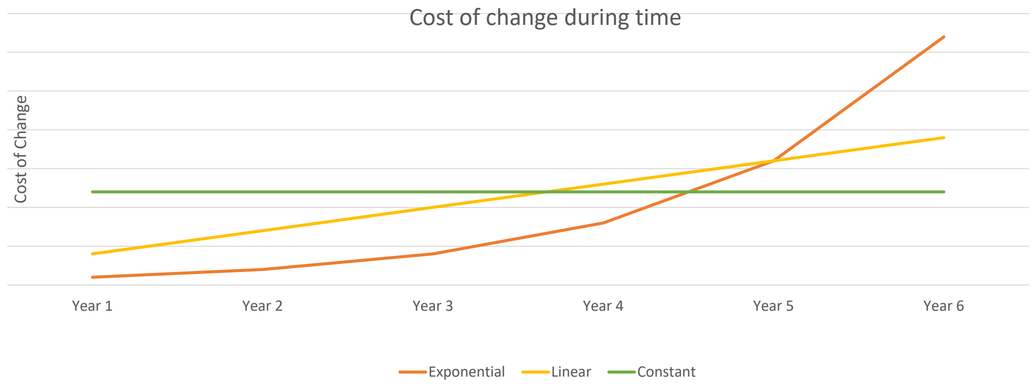
Few quotes…

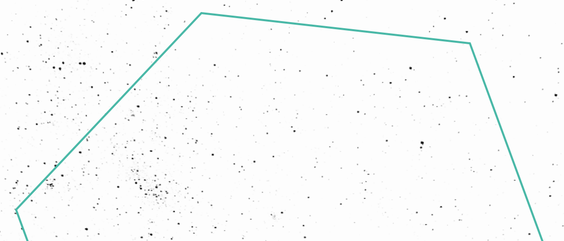


Multi-Application Architecture

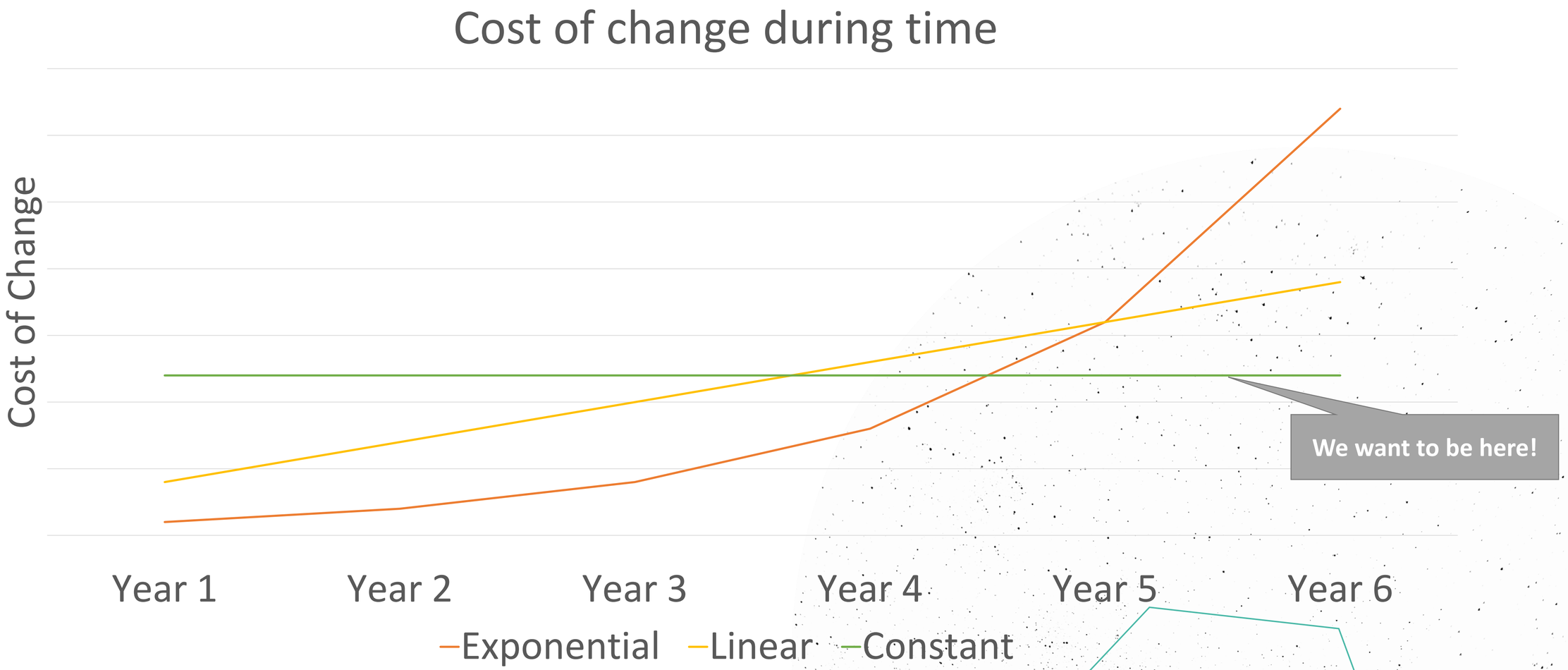
„Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change.“

Grandy Booch





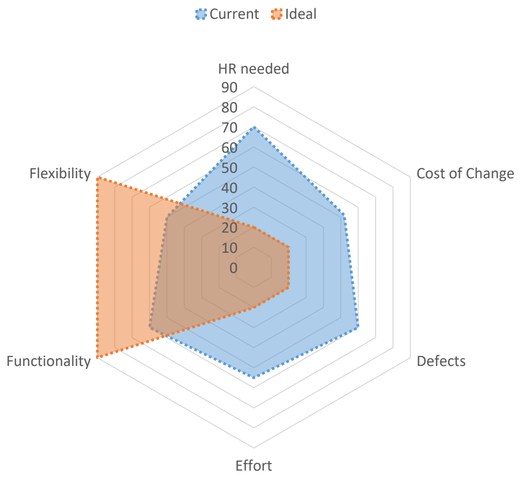
Multi-Application Architecture

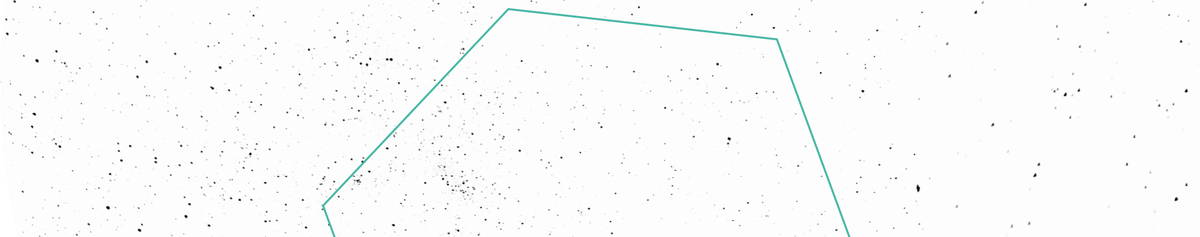


Multi-Application Architecture

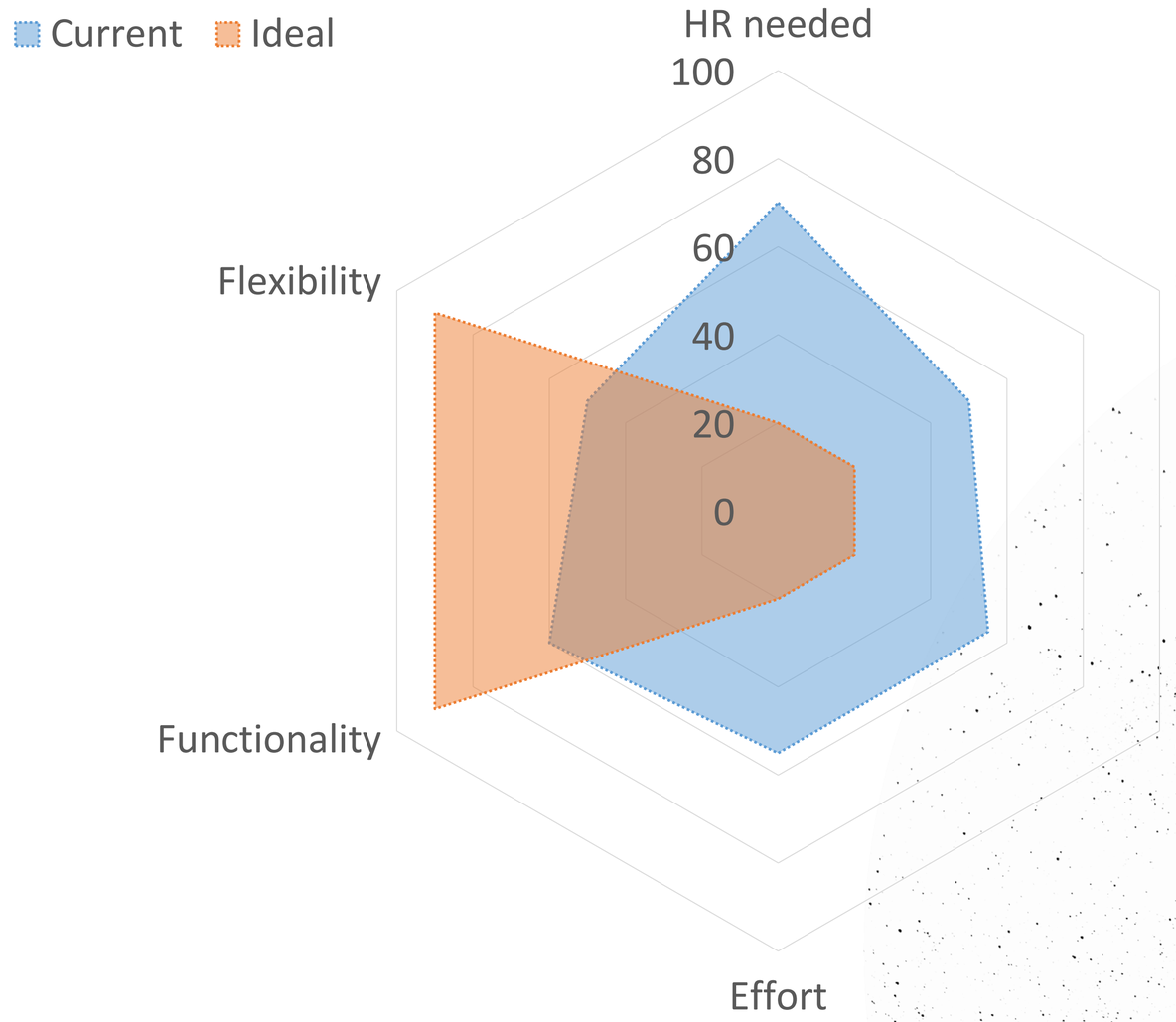
„When software is done right, it requires a fraction of the human resources to create and maintain. Changes are simple and rapid. Defects are few and far between. Effort is minimized, and functionality and flexibility are maximized.“

Clean Architecture, Robert C. Martin (Uncle Bob)





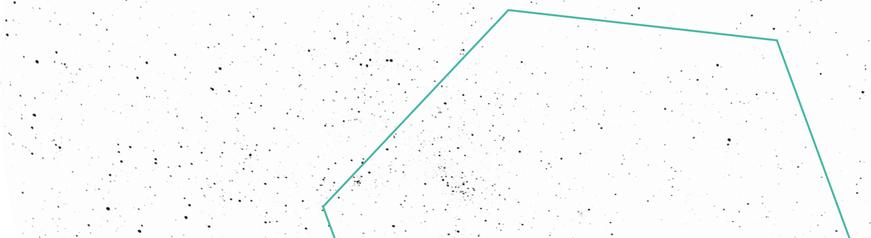
Multi-Application Architecture



Multi-Application Architecture

„The goal of software architecture is to minimize the human resources required to build and maintain the required system“

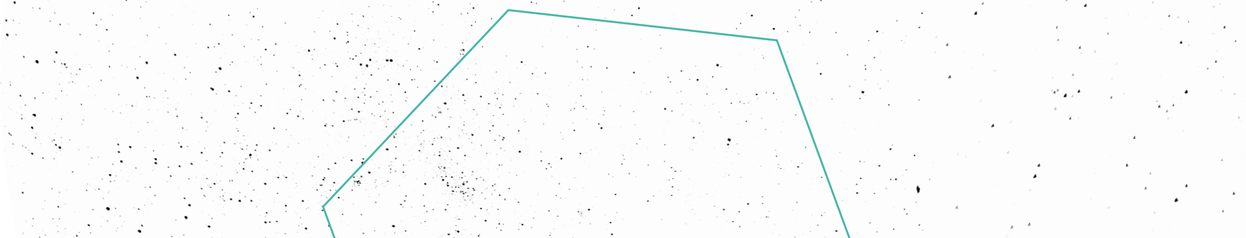
Robert C. Martin (Uncle Bob)



Multi-Application Architecture

„The only way to go fast, is to go well.“

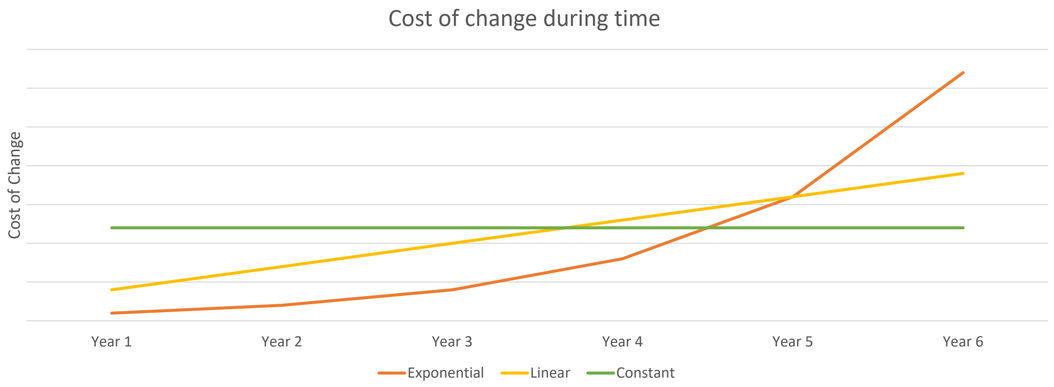
Robert C. Martin (Uncle Bob)



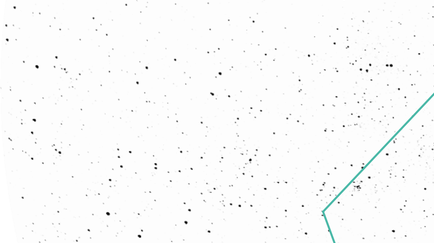
Multi-Application Architecture

BC/NAV TECH DAYS 2022
mibuso.com

„If you think good architecture is expensive, try bad architecture.“



Brian Foote and Joseph Yoder



Multi-Application Architecture

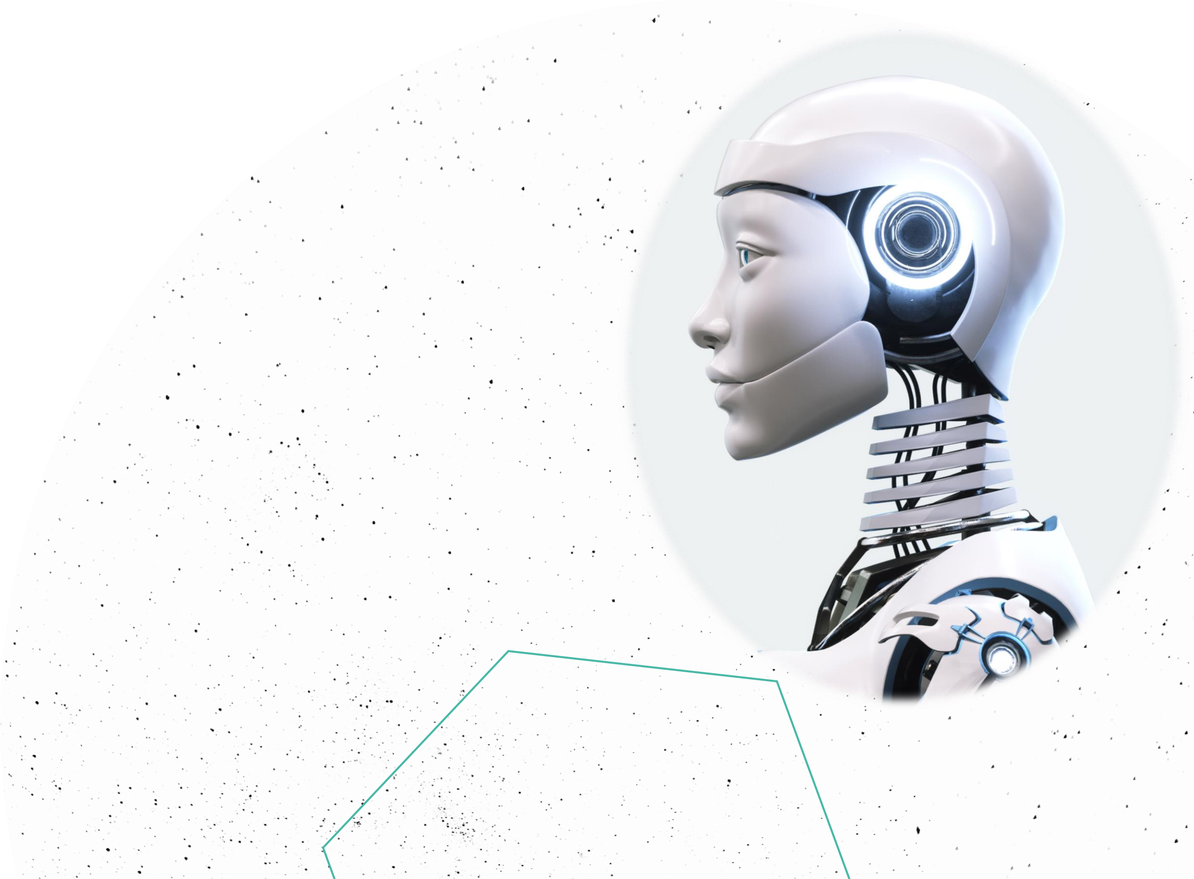
„Making messes and clean them is always slower than staying clean“

Look for “Bad habits of AL Developers” session!

Multi-Application Architecture

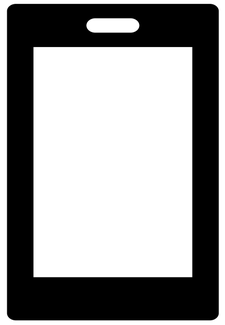
BC/NAV TECH DAYS 2022
mibuso.com

- And do not forget:
  - We are creating **Soft**ware:
    - **Soft**-ware – soft – easily change the behavior of machines (of **Hard**ware).



Multi-Application Architecture

- And do not forget:
  - We are creating **Soft**ware:
    - **Soft**-ware – soft – easily change the behavior of machines (of **Hard**ware).



Prepare your phones for a poll!

Multi-Application Architecture

- What is better?
  - Software which perfectly works but is impossible to change
  - Software which doesn’t work but is easy to change?

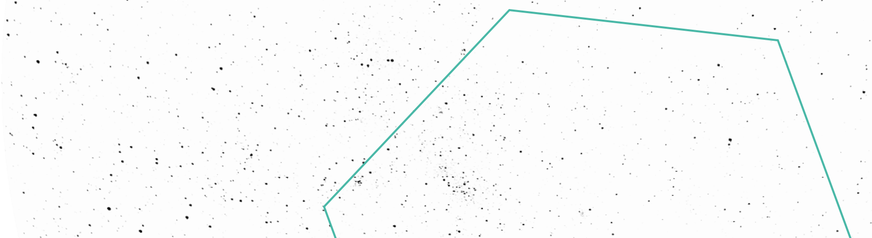
Multi-Application Architecture

- Software which doesn't work but is easy to change is BETTER!

- WHY?
  - You can change it easily to work correctly…

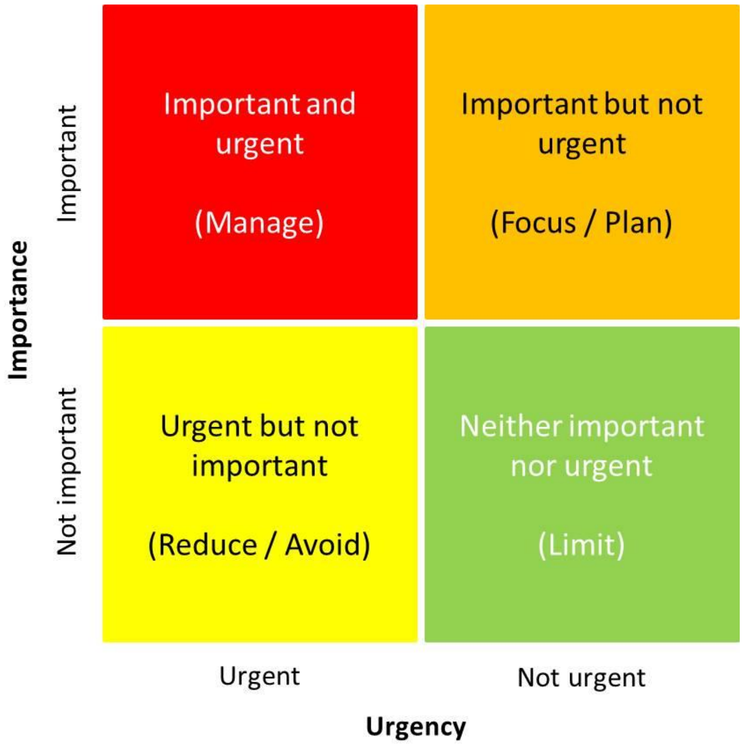
#CostOfChangeMatters

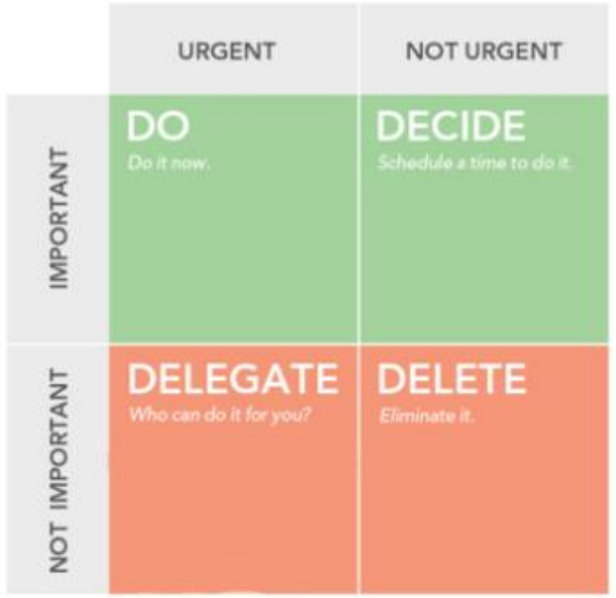
#MSDyn365BCDesignMatters

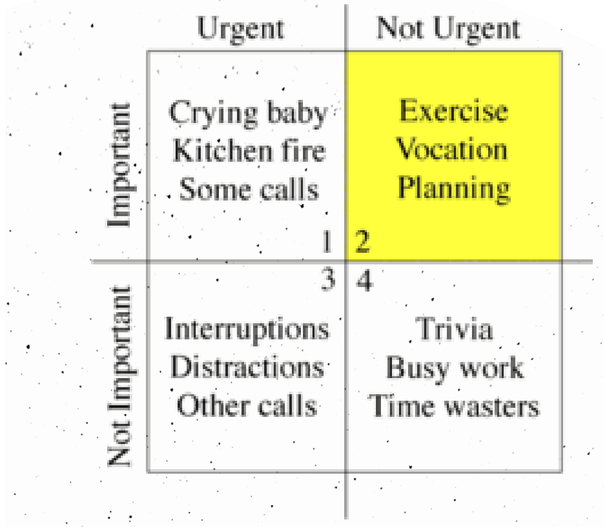


Intermezzo - Eisenhower Matrix

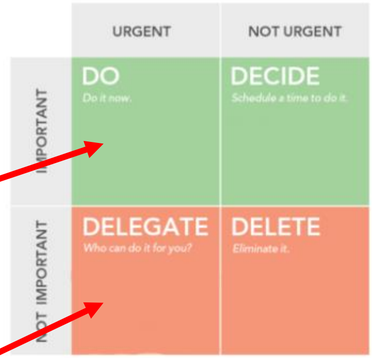
- Important vs Urgent







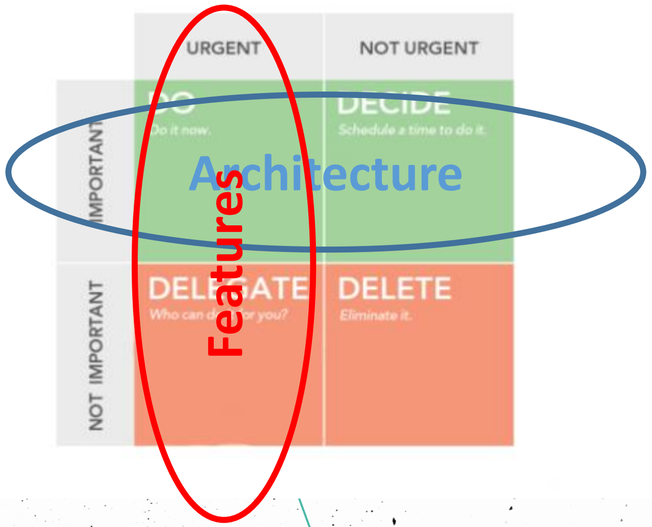
Important vs Urgent



- Urgent – YES! Call the Fire Department!
- Important –
- YES if you are in danger => RUN/fire extinguisher etc.!
- NO if you are not in danger => you have delegated it to Fire Department

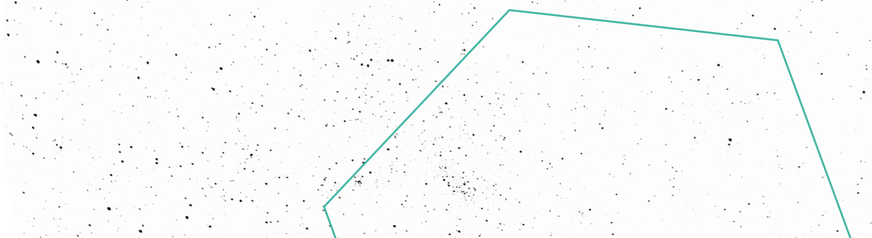
Architecture vs Features

- Feature (urgent) vs Architecture (important)
  - Managers and customers wants Features
  - We tend to prioritize **URGENT** but **NOT IMPORTANT** things
  - **Developers must fight for Architecture** (“managers do not understand architecture – that’s why they hired developers/consultants”)
  - **Development team is responsible** to assert the importance of architecture over the urgency of features



#CostOfChangeMatters

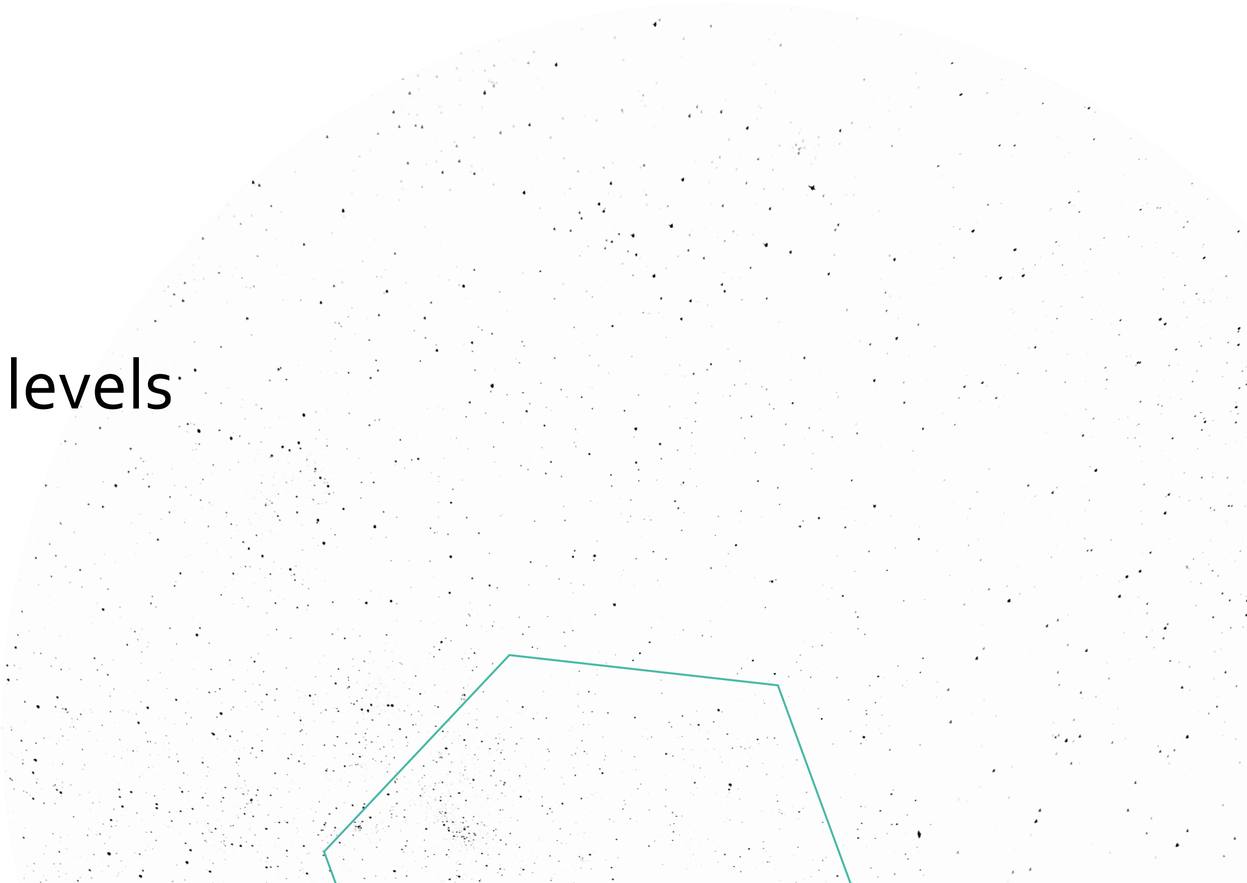
#MSDyn365BCDesignMatters



Architecture level

- Levels
  - Source code/functions
  - Objects/Classes
  - Source files
  - Modules/Apps
  - Components/Solutions

- Same principles could be applied on all levels
- We will focus on Modules/Apps



A story to help us connect to the ideas…

BC/NAV TECH DAYS 2022
mibuso.com

AmuseYou

AmuseYou Inc. is a small company. They have a variety of bounce castles, slides, and other amusements that people can book them by the day.

The AmuseYou team has been growing through some strategic mergers and now has facilities for

- Laser Tag Arenas
- Go-Kart Tracks
- Escape Rooms
- Paintball Fields

For the next year or two, they want to use the existing Booking system, but they know they want to potentially leave their options open to change that in the future.

AmuseYou Solution Today

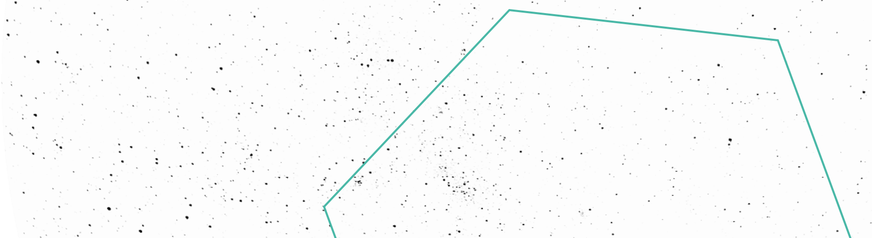
AmuseYou – Bounce, Bookings, Everything

Business Central

Design principles

- **S.O.L.I.D.**
  - **S**ingle Responsibility Principle (SRP)
  - **O**pen-Closed Principle (OCP)
  - **L**iskov Substitution Principle (LSP)
  - **I**nterface Segregation Principle (ISP)
  - **D**ependency Inversion Principle (DIP)

Based on book “Clean Architecture”, Robert C. Martin (Uncle Bob)



Single Responsibility Principle (SRP)

Single Responsibility Principle (SRP)

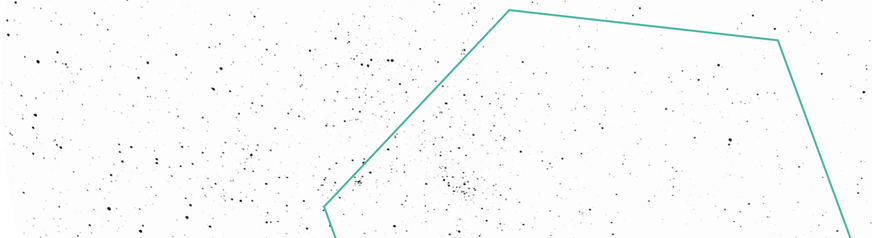
BC/NAV TECH DAYS 2022
mibuso.com

- A module should be responsible to one, and only one, **reason to change**
- A module should be responsible to one, and only one, **user or stakeholder**
- A module should be responsible to one, and only one, **actor**

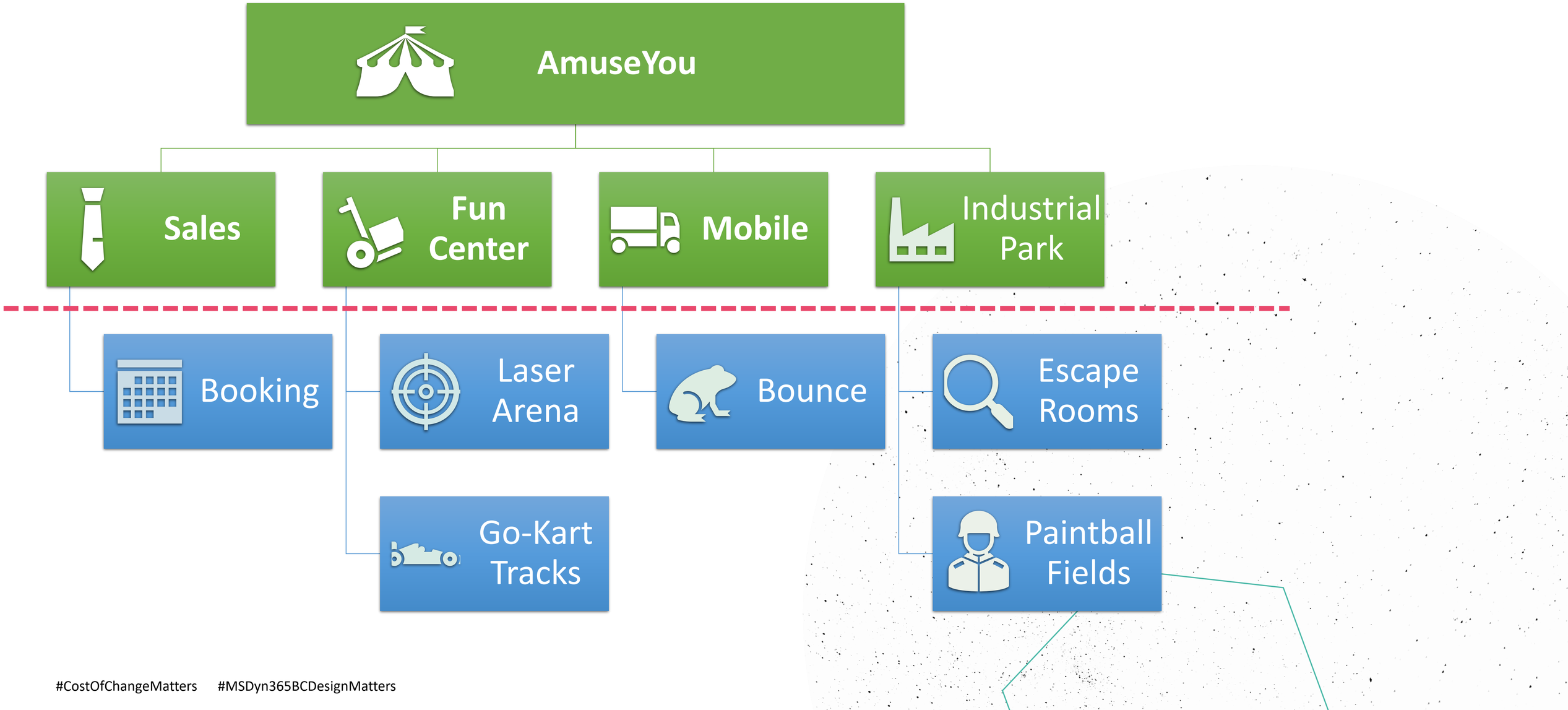
- What is module?
  - Unit of deployment (have own version, could be deployed)
    - Extension in BC

#CostOfChangeMatters

#MSDyn365BCDesignMatters



Single Responsibility Principle (SRP)



AmuseYou – A Better Plan (right?) – v1.0

Hint: Remember that they wanted to remove Bookings?

Single Responsibility Principle (SRP)

BC/NAV TECH DAYS 2022
mibuso.com

- Can be applied to any part of the code, not only modules/apps
  - Standard example of breaking this rule:
    - **Reservation table**
    - Monolithic applications

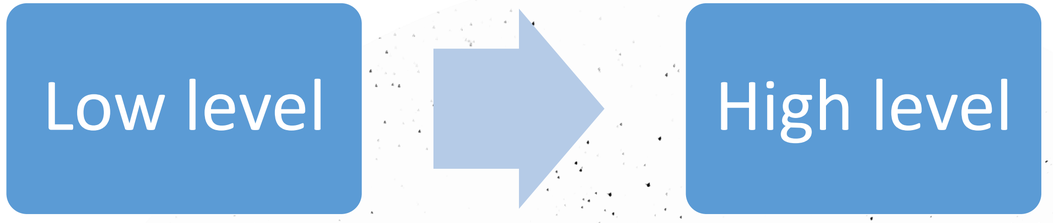
- What can help?
  - Ask “What is the responsibility of the app/object/function/component?”
  - Add Actor/User/Stakeholder to the User Story
    - If one US changes another US but for different Actor/User/Stakeholder, solve the conflict (create new app or process, discuss with the original actor/user etc.)
  - Do not create multi-purpose objects/tables/Apps

Open-Closed Principle

Open-Closed Principle

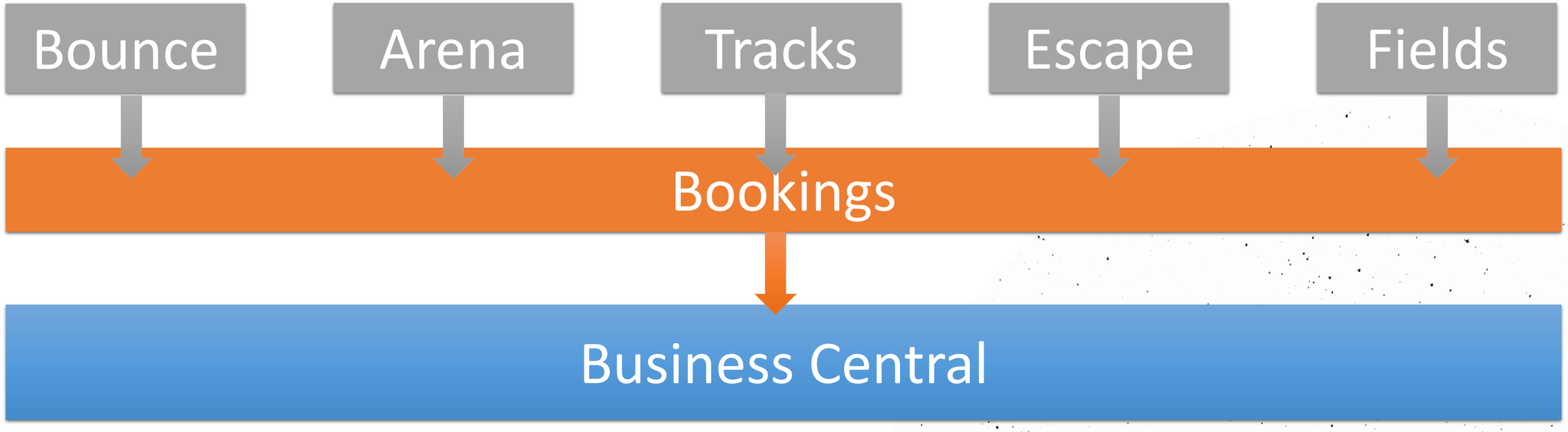
- *"Software artifacts should be open for extension but closed for modification."*

- Changing by adding, not changing existing
  - Basic feature of the platform – USE IT!
  - Waldo's "Generic Method" pattern
  - Eventing
  - Interfaces

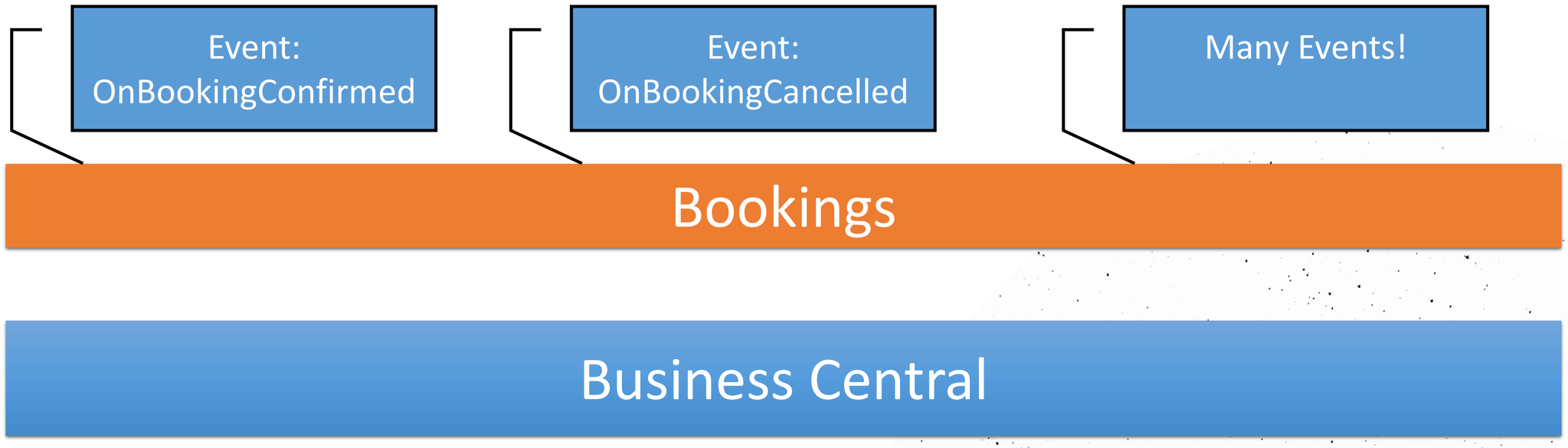


- Separate functionality based on **how, why and when** it changes and organize that separated functionality into a hierarchy of components.
  - High level policies (central concern, business rules…) vs peripheral concerns (specific implementation, low-level access…)
  - High level component is protected from changes in low level components

Open-Closed Principle Example

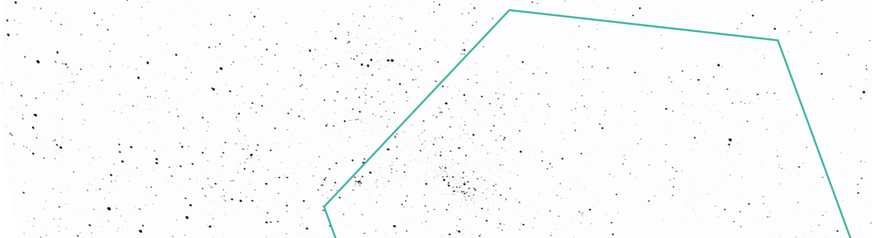


Open-Closed Principle Example



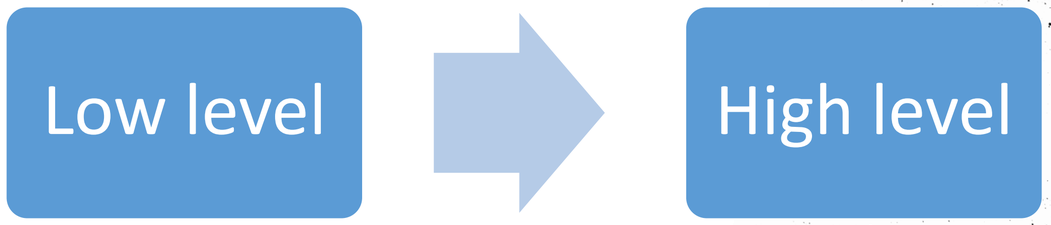
#CostOfChangeMatters

#MSDyn365BCDesignMatters



Open-Closed Principle

- Use the known patterns to open the app for dependencies
- Do not change the existing code if the kind of the change is different than the original code (different concern, responsibility, stability etc.)
- Check if dependency is in correct directions (Low to High)
  - Specific to generic
  - Low level to High level
  - Not important to important/critical
  - Not stable (often changed) to stable (not so often changed)

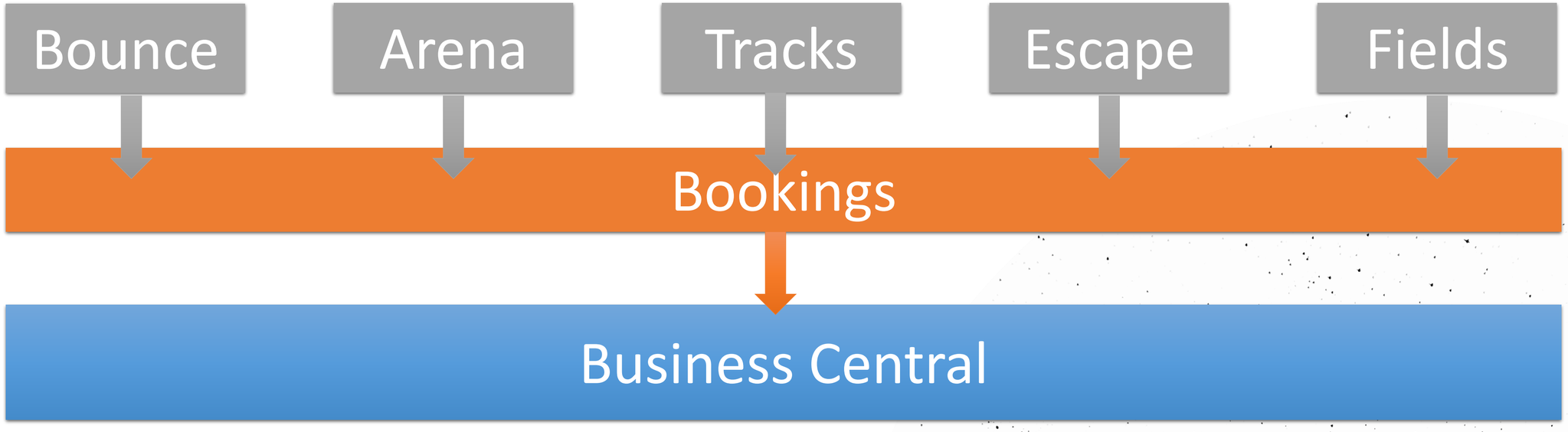


Liskov Substitution Principle (LSP)

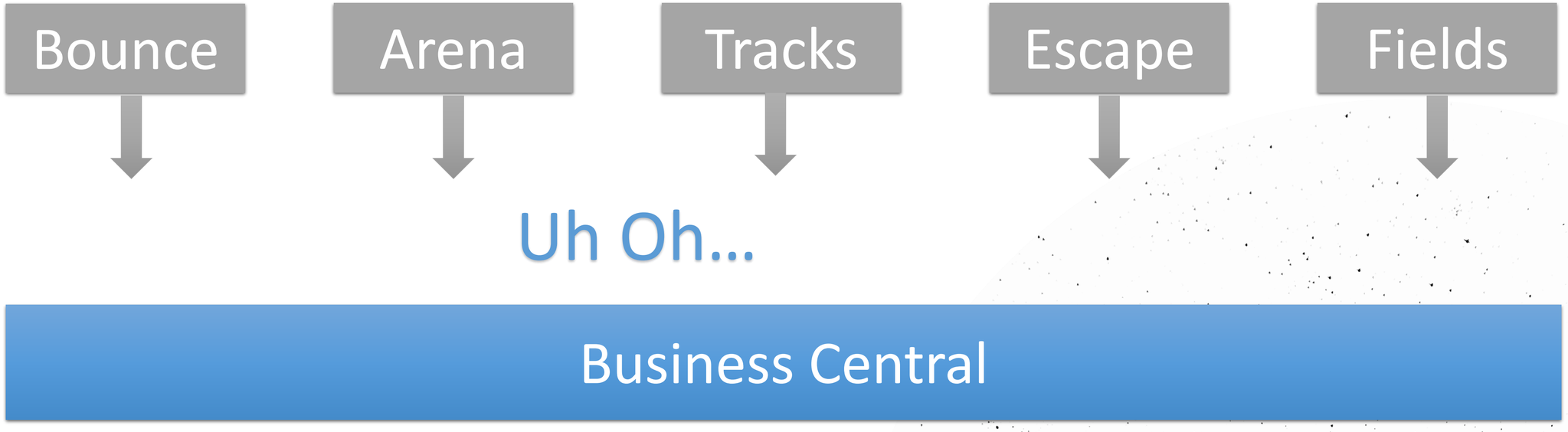
Liskov Substitution Principle (LSP)

- Introduced by Barbara Liskov in 1988
- You can replace one part with something else without need to change the depending part without change of behavior
- Originally about super-classes and sub-classes, but could be applied even for AL
- Solution: Interfaces, Events
- Violation of substitutability cause pollution with a significant amount of extra mechanisms (if-then-else/case switch for different cases)

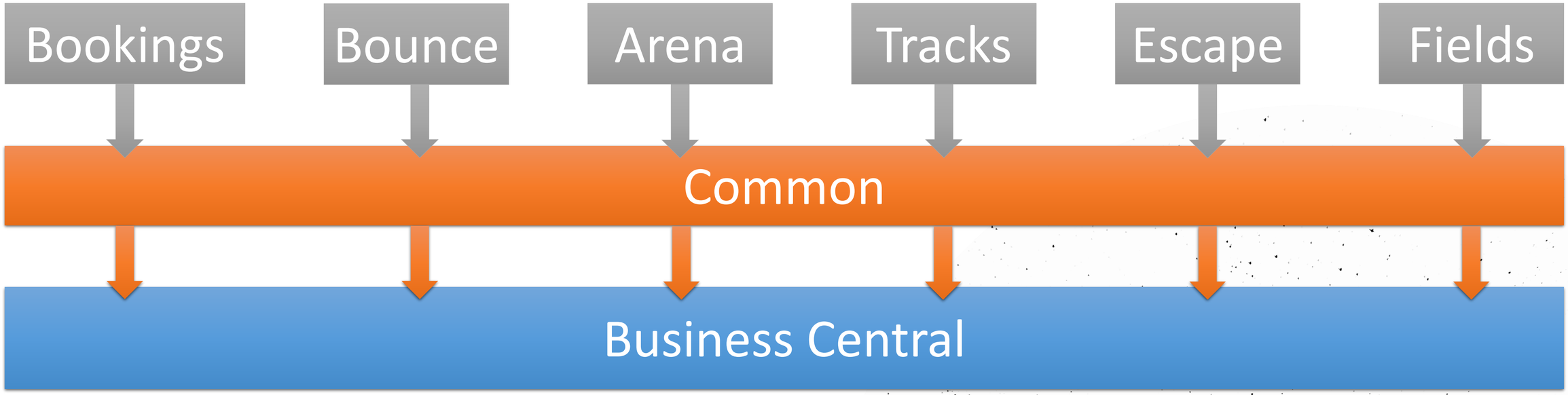
Liskov Substitution Principle (LSP)



Liskov Substitution Principle (LSP)

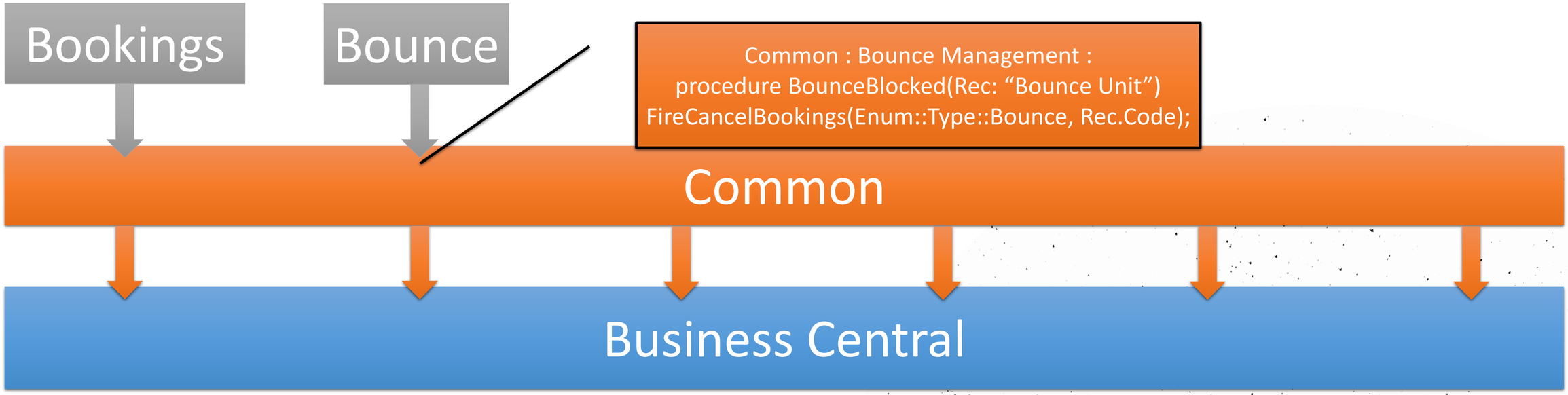


AmuseYou – An even better plan (?) – v2.0

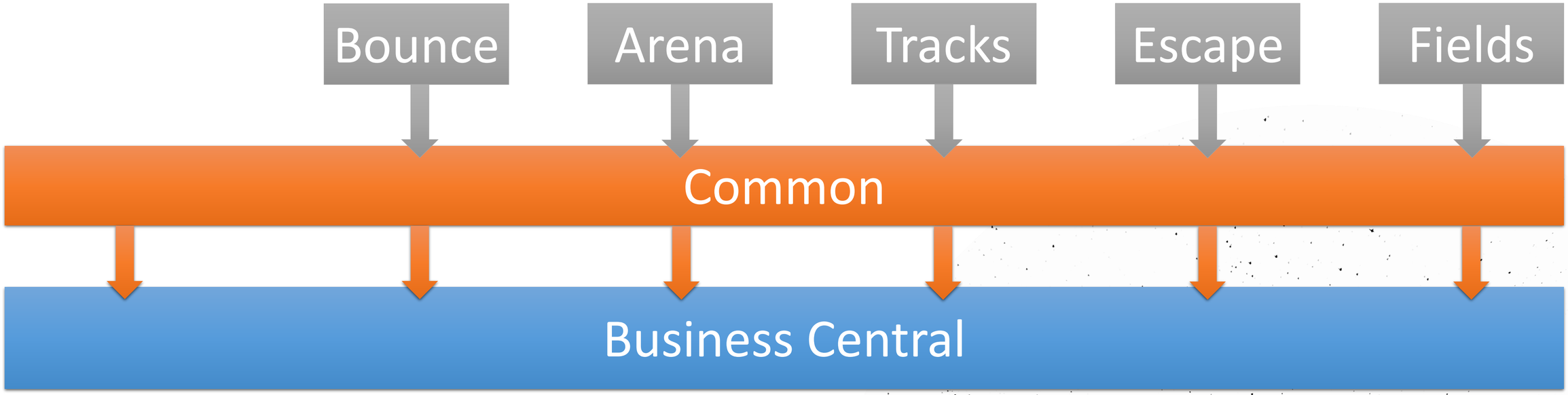


Hint: They've just moved the problem

AmuseYou – An even better plan (?) – v2.0

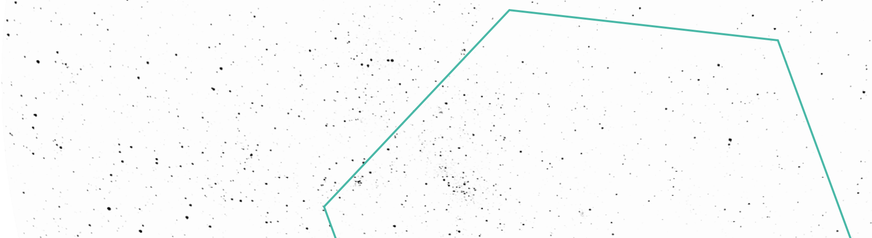


AmuseYou – An even better plan (?) – v2.0



#CostOfChangeMatters

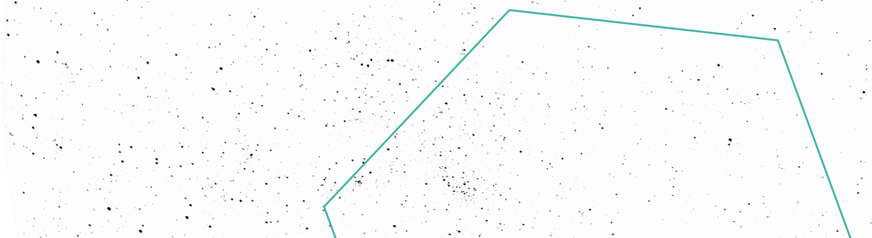
#MSDyn365BCDesignMatters



Liskov Substitution Principle (LSP)

BC/NAV TECH DAYS 2022
mibuso.com

- If you tend to put new If-then-else or Case to make something different on some Enum/option, consider using Interface
- Use interface, whenever it is possible that implementation of the specific process/function could be different for different cases
- Do not be depending on specific implementation, rather prepare interface which will be generic enough and then implement it in specific way – develop the main process for "**Unknown**"
- If something should be interchangeable:
  - Inputs cannot be stricter (original -1 000..1 000, new -100..100 etc.)
  - Output cannot be wider (original -1..1, new -100..100 etc.)

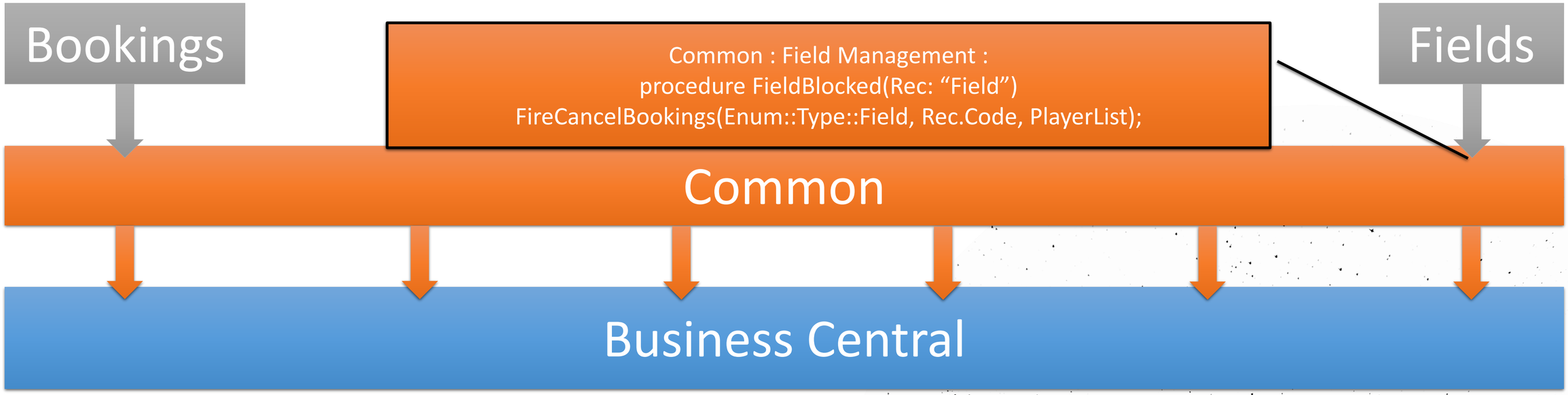


Interface Segregation Principle (ISP)

Interface Segregation Principle (ISP)

- “Clients should not be forced to depend upon interfaces that they do not use.”
- We do not want to pollute application or interface with not needed dependencies

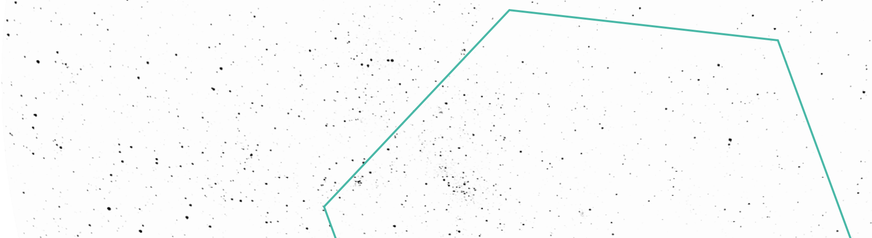
AmuseYou – An even better plan (?) – v2.0



Common : Bounce Management :
procedure BounceBlocked(Rec: “Bounce Unit”)
FireCancelBookings(Enum::Type::Bounce, Rec.Code, EmptyList?!);

#CostOfChangeMatters

#MSDyn365BCDesignMatters

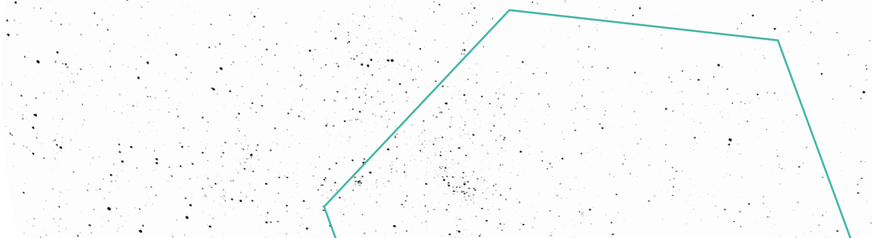


Interface Segregation Principle (ISP)

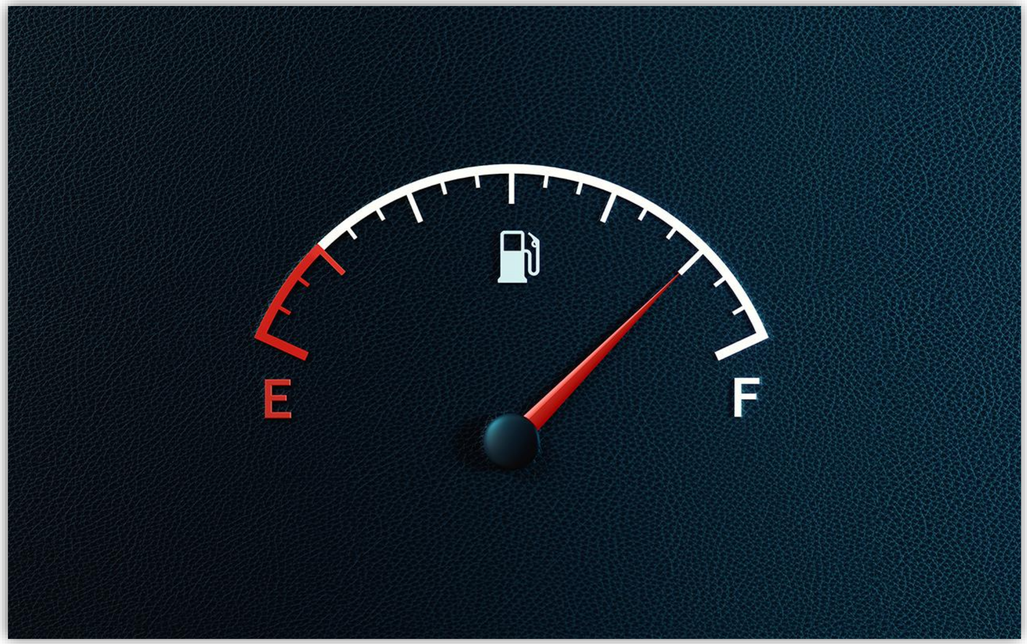
BC/NAV TECH DAYS 2022
mibuso.com

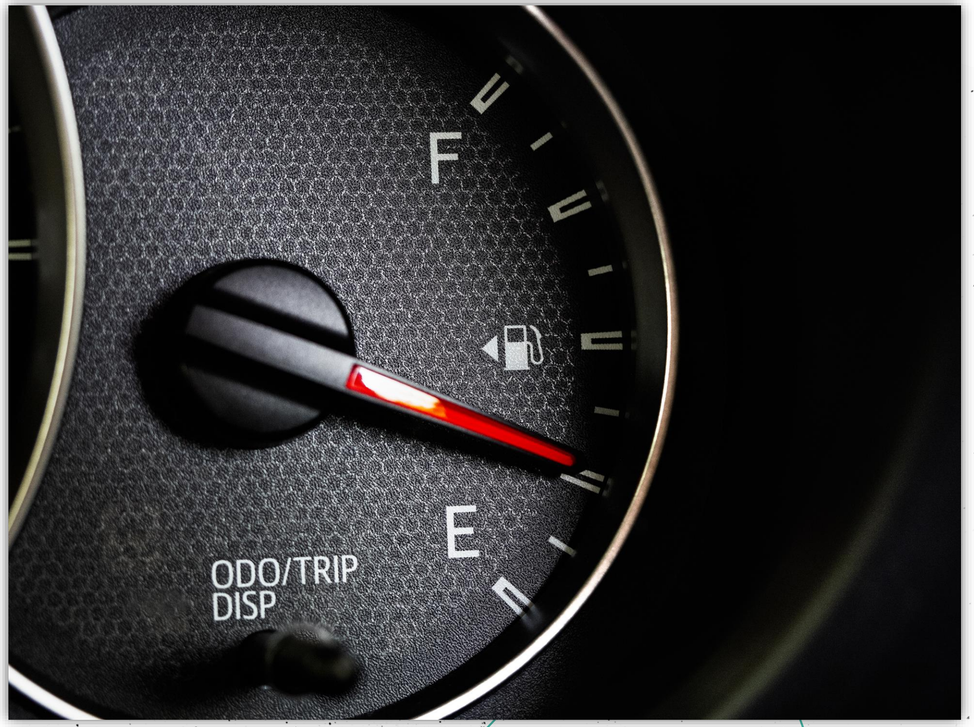
- If applications are loosely coupled, do not make them dependent (A can exists without B)
- Use dependency only if the functionality is tightly coupled (A have no meaning without B)
- If only part of the app is tightly coupled, maybe it is better to split the app.

- Keep interface minimal – do not pollute it with unnecessary things
- For specific things create new interface



Dependency Inversion Principle (DIP)



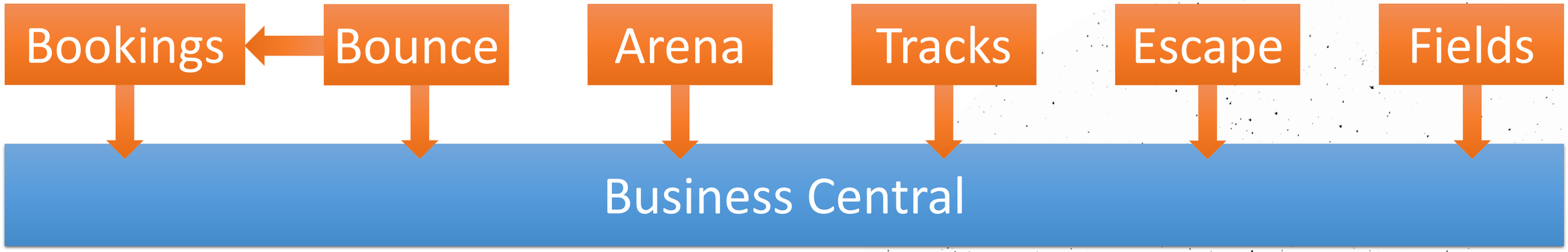


Dependency Inversion Principle (DIP)

- Normal is to have dependency in same directions as flow of control (A calls B, A depends on B)
  - Direct function calls etc.
- DIP is switching the direction of the dependency – dependency is in opposite direction than flow of control (A "calls" B, B depends on A)
- DIP is tool helping to implement previous principles

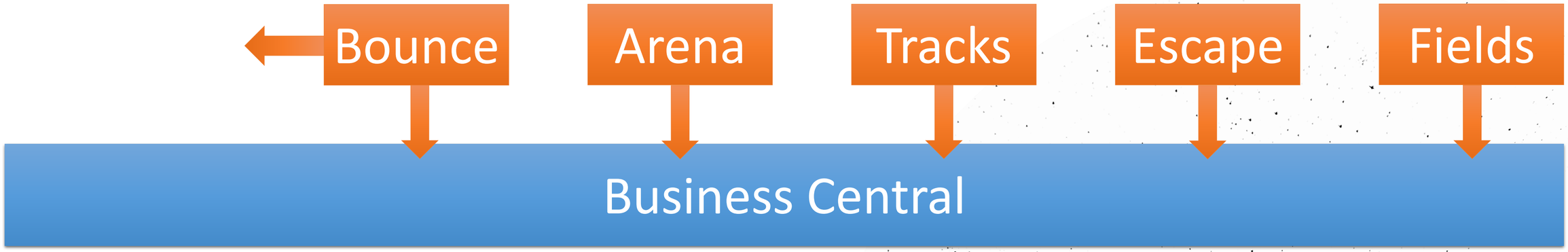
- Solution in AL
  - Events
  - Interfaces

AmuseYou – An ever betterer plan (?!) v3.0

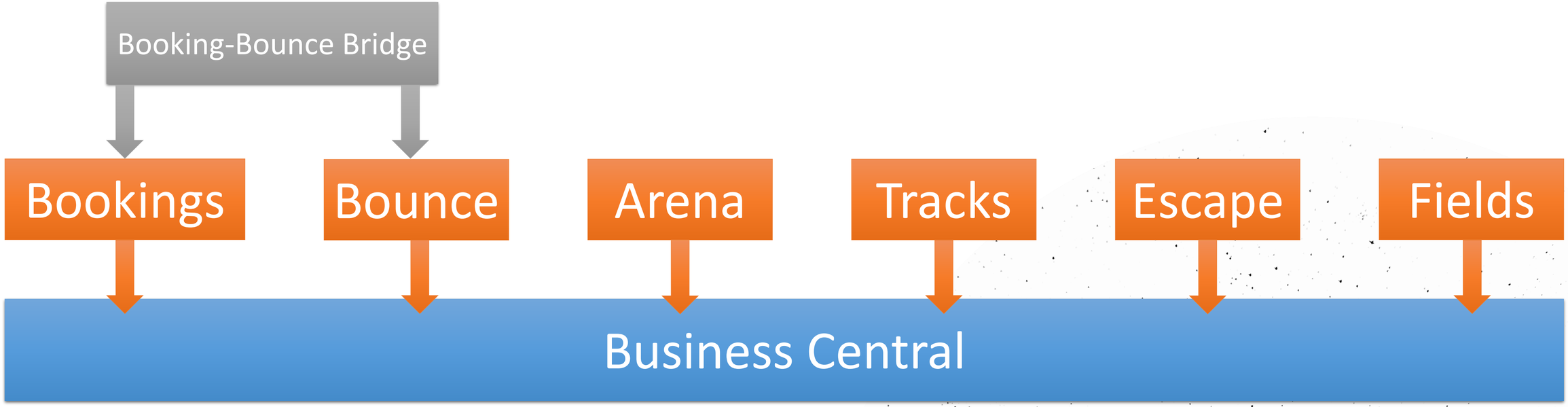


Hint: Still fragile, many dependencies between things

AmuseYou – An ever betterer plan (?!) v3.0



AmuseYou – Final Final Really Plan (?!) v4.0



Dependency Inversion Principle (DIP)

BC/NAV TECH DAYS 2022
mibuso.com

Tools you may already know/use (if not, you should!):

AYAmusementType.Enum.al

Shared/Common > src > AYAmusementType.Enum.al > ...

```
enum 74281 "AY Amusement Type" implements "AY IBookableAmusement"
{
    Extensible = true;
    DefaultImplementation = "AY IBookableAmusement" = "AY Graceful Failover";
    UnknownValueImplementation = "AY IBookableAmusement" = "AY Bookable Missing";

    value(0; " ")
    {
        Caption = ' ';
    }
}
```

AYIBookableAmusement.Interface.al

Shared/Common > src > AYIBookableAmusement.Interface.al > ...

```
interface "AY IBookableAmusement"
{
    procedure SetPriceFromBookedUnit(BookedUnitNo: Code[20]): Decimal;

    procedure ValidateBookedUnit(BookedUnitNo: Code[20]): Boolean;

    procedure GetBookingChargeAccount(BookedUnitNo: Code[20
}
```

Bookings

AYBKBUBookableAmusement.Codeunit.al | AYBKBUBookable.EnumExt.al

Process/BK-Bounce > src > AYBKBUBookable.EnumExt.al > EnumExtension 74410 "AYBKBU Bookable"

```
enumextension 74410 "AYBKBU Bookable" extends "AY Amusement Type"
{
    value(74410; Bounce)
    {
        Caption = 'Bounce Unit';
        Implementation = "AY IBookableAmusement" = "AYBKBU BookableAmusement";
    }
}
```

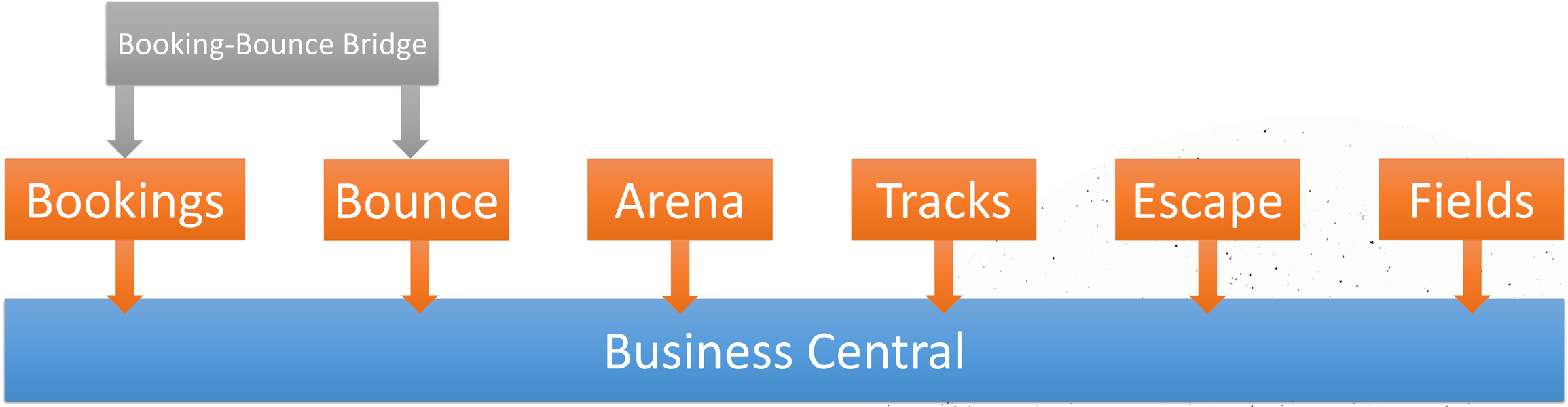
AYBKBUBookableAmusement.Codeunit.al

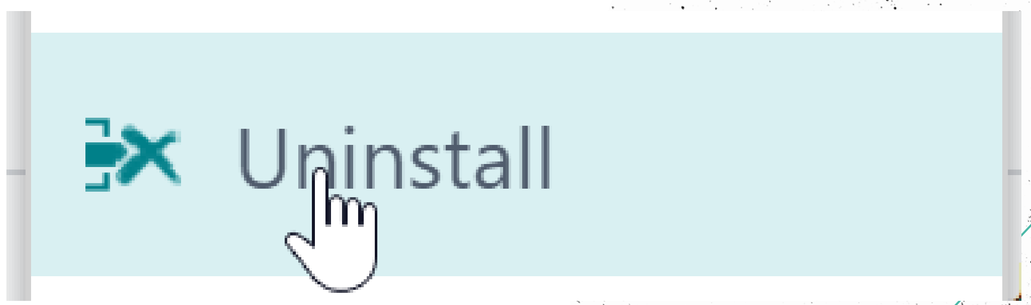
Process/BK-Bounce > src > Interface > AYBKBUBookableAmusement.Codeunit.al > Codeunit 74410 "AYBKBU E

```
1 reference | Run Tests | Debug Tests
codeunit 74410 "AYBKBU BookableAmusement" implements "AY IBookableAmusement"
{
    0 references | 0% Coverage
    procedure SetPriceFromBookedUnit(BookedUnitNo: Code[20]) DailyPrice: Decimal
    var ...
    begin ...
    end;

    0 references | 0% Coverage
    procedure ValidateBookedUnit(BookedUnitNo: Code[20]) Valid: Boolean
    var ...
    begin ...
    end;

    0 references | 0% Coverage
    procedure GetBookingChargeAccount(Booke
    var ...
    begin ...
    end;
```

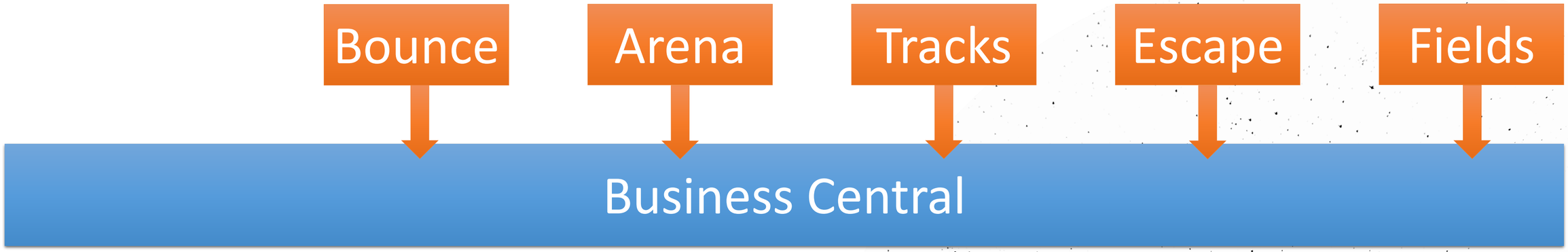
Booking-Bounce Bridge

AmuseYou – Final Final Really Plan (?!) v4.0



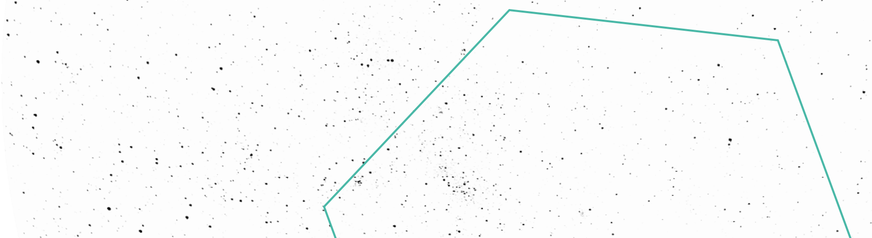


AmuseYou – Final Final Really Plan (?!) v4.0



#CostOfChangeMatters

#MSDyn365BCDesignMatters



Dependency Inversion Principle (DIP)

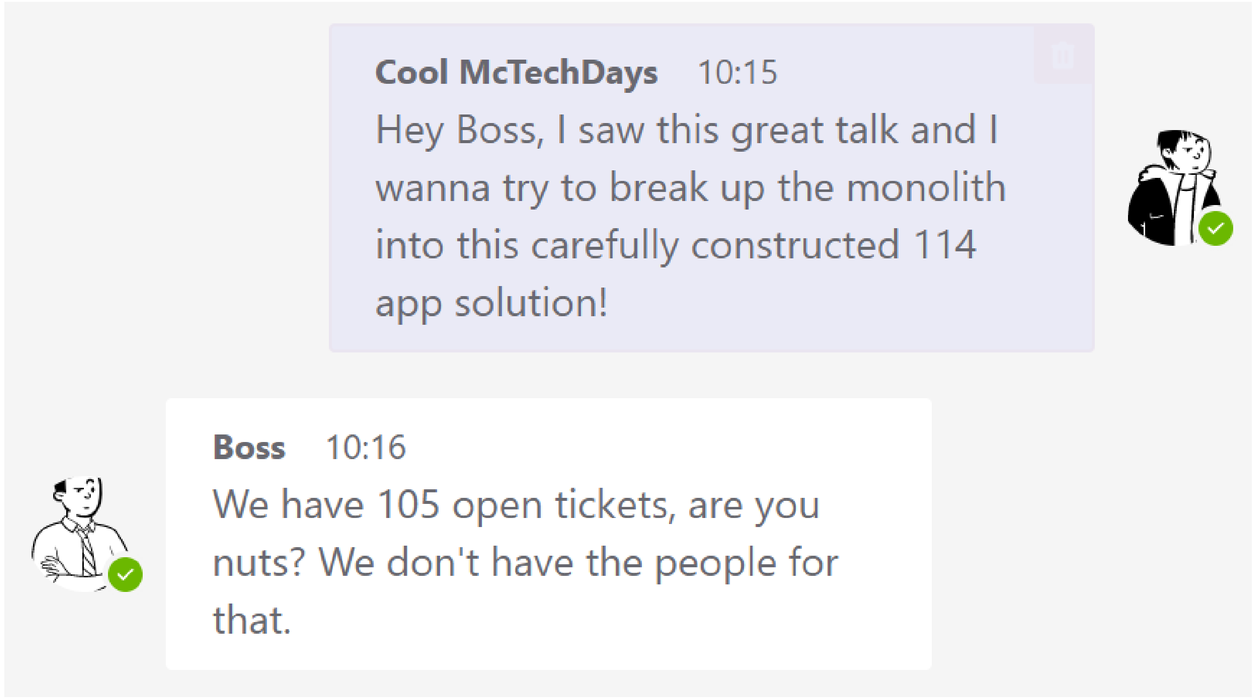
- If you find out that dependency should goes opposite, use Events or Interface to invert it
  - Specific depends on generic
  - Low-level depends on high-level
  - Unstable depends on stable

- If we want to protect A from changes in B, B must depend on A

- (Remember the Open-Close Principle)
- We are protecting modules containing “policies” or „rules“

Don’t be soft, be SOLID!

When Monday Comes



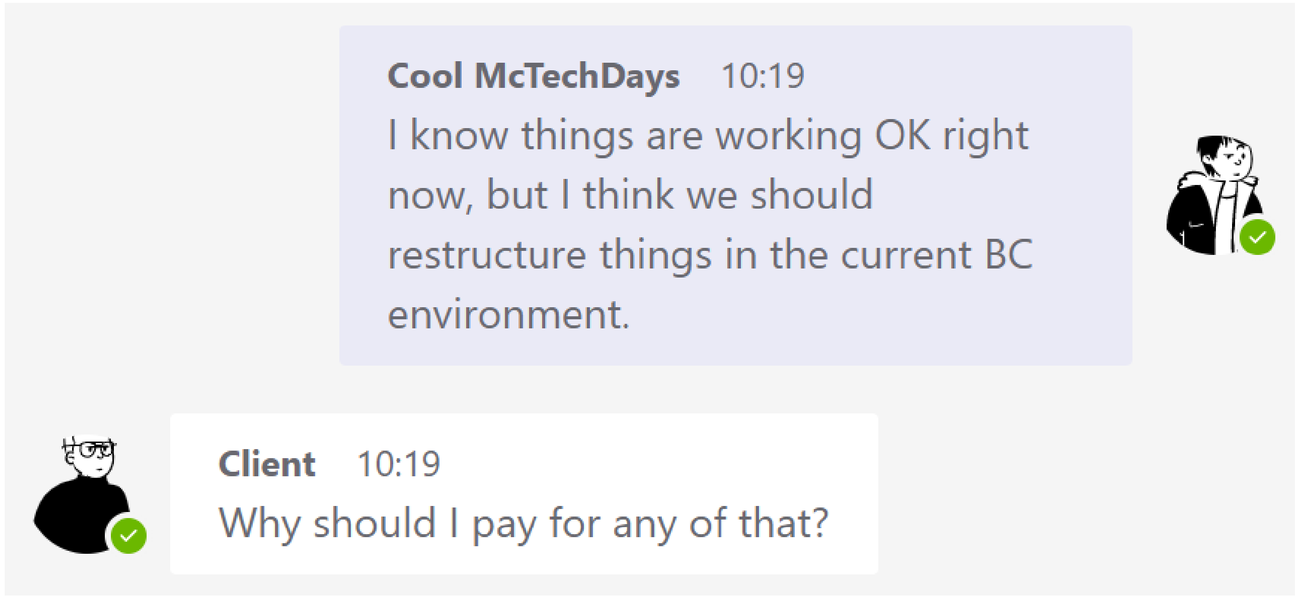
Selling SOLID

- Start small
- Work from ‘outer’ towards inner

Benefits: Cost/HR/Sales

- Small changes are easier to delegate and safer
- Re-useable!
- Cross-ISV Partnerships?

When Monday Comes

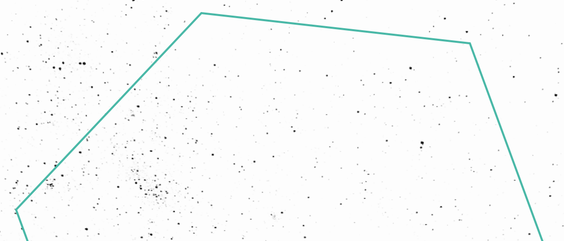


Selling SOLID

- You will pay it, just like testing
- Investing now saves bigger

Benefits: Cost/Flexibility

- Easier to do big changes
- Less time on defects



When Monday Comes

Cool McTechDays 10:22
I know that converting the 15 year old NAV client to BC was rough, but I saw a great TechDays talk about how we could move forward, and reduce the technical debt.

Señor DeVeloppah 10:23
I don't have the time for that. I have 3 different P1's with things not shipping! Some yahoo changed the printer .NET and now things won't post! This BC stuff is nuts, NAV was better.

Selling SOLID

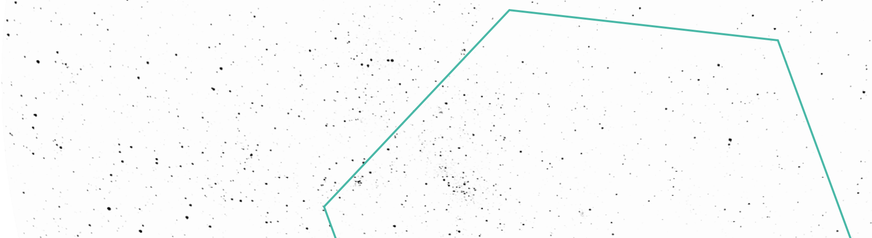
- Segregation keeps you from having to fix the “fixes”
- Hand off parts

Benefits: Work

- Easier to break up projects
- Smaller can be faster
- Defect Hunting is *much* faster

#CostOfChangeMatters

#MSDyn365BCDesignMatters

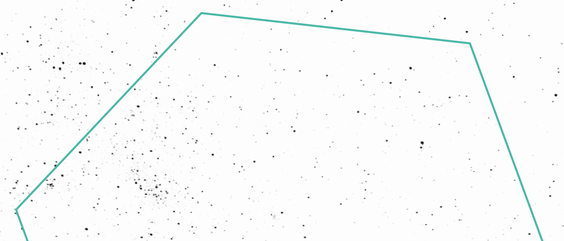


Remember the Principles

BC/NAV TECH DAYS 2022
mibuso.com

SOLID

- **S**ingle Responsibility Principle (SRP)
- **O**pen-Closed Principle (OCP)
- **L**iskov Substitution Principle (LSP)
- **I**nterface Segregation Principle (ISP)
- **D**ependency Inversion Principle (DIP)



Any Questions?

SOLID

- **S**ingle Responsibility Principle (SRP)
- **O**pen-Closed Principle (OCP)
- **L**iskov Substitution Principle (LSP)
- **I**nterface Segregation Principle (ISP)
- **D**ependency Inversion Principle (DIP)

BC/NAV TECH DAYS 2022
mibuso.com

Thank You!

Any Questions?

BC/NAV TECH DAYS 2022
mibuso.com

Thank You!