whogloo

# Ports, Adapters and ABL

A better shape for OpenEdge systems

Julian Lyndon-Smith
Archmage of Engineering, whogloo ltd

# Agenda

01

## What is Hexagonal Architecture ?

Brief introduction to the theory. Adapters, Ports, Domains, Applications

02

## ABL

How to implement Hexagonal Architecture in the ABL.

Obligatory AI ahead.

03

## Look and see

Show me the code !

# 01

# What is Hexagonal Architecture ?

“Allow an application to equally be driven by users, programs, automated test or batch scripts, and to be developed and tested in isolation from its eventual run-time devices and databases.”

- Alistair Cockburn

# Hexagonal architecture

Also known as the Ports and Adapters pattern

## Most apps are built in layers

- UI layer
- Business logic layer
- Database layer

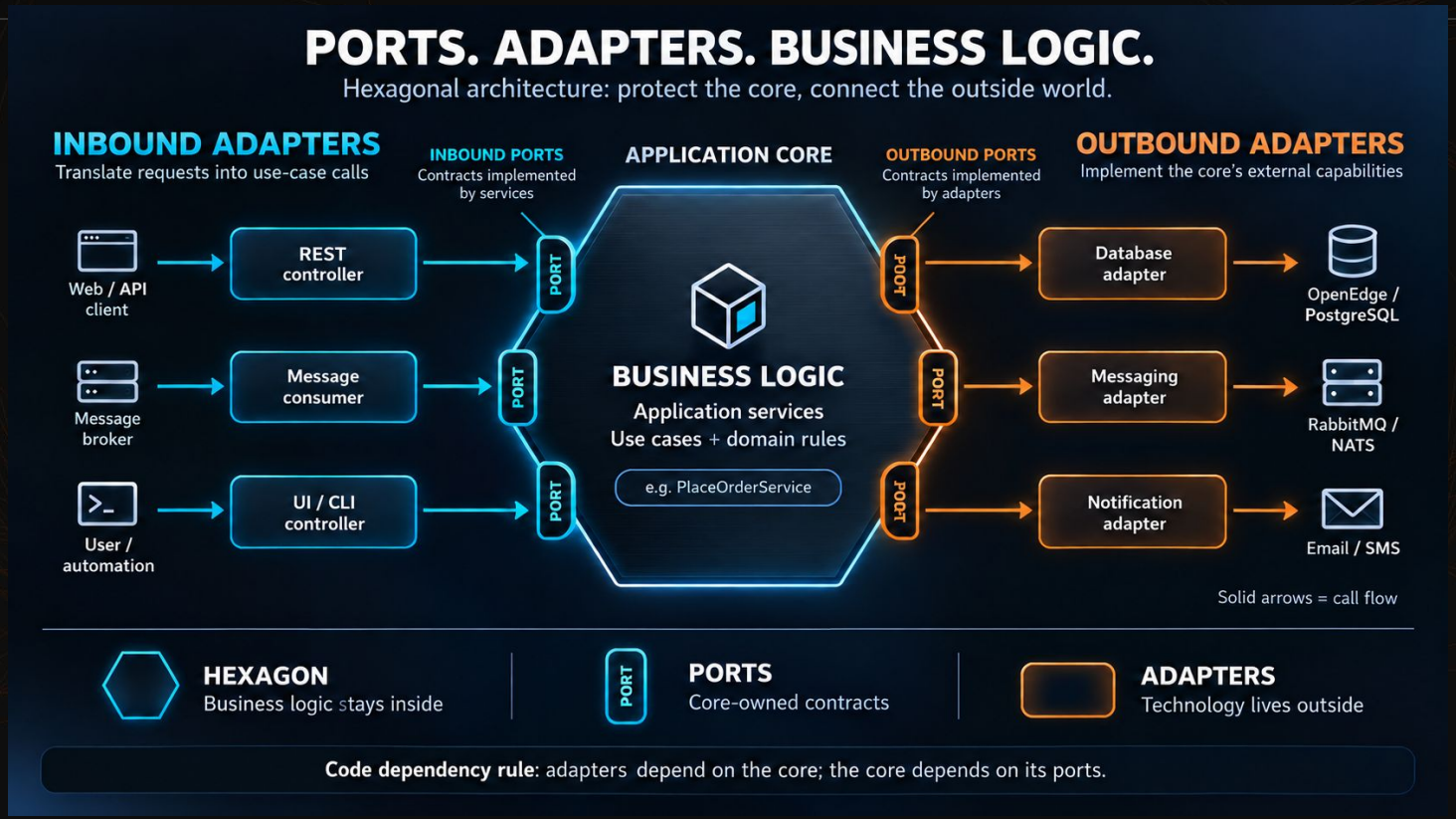
In a typical layered app, business logic quietly depends on the database. For instance, a function that calculates a customer's discount directly queries the customer table.

## Hexagonal flips this

Instead of your core logic depending on infrastructure, infrastructure depends on your core logic — through clearly defined boundaries.

Everything revolves around two concepts - ports and adapters.

# Hexagonal architecture



# Hexagonal architecture

So what are ports and adapters ?

## Ports

A port is a contract at the boundary of the application.

Inbound ports define what the application can do.

Outbound ports define what the application needs others to do.

It is irrelevant for the application what technical details are behind these ports.

## Adapters

These provide connection to the outside world. They translate the signals of the outside world to a form understood by the application.

Adapters only communicate with the application through the ports.

# The Business logic

What does the Business logic provide and need

## Inbound Port

“I provide this”

IOrderService
SaveOrder()
GetOrder()

Application implements this port

## Outbound Port

“I need this”

IOrderRepository
Save()
FindById()

Application executes/calls this port

# Adapters

There are two “different” types of adapters

## Primary / driving / inbound

Driving adapter asks the application to do something.

Normally drawn on the left of the hexagon.

## Secondary / driven / outbound

Driven adapter is asked by the application to do something.

Normally drawn on the right of the hexagon.

# Simple example

Let’s make this concrete with a basic order-processing example

## Inbound

IOrderService

- SaveOrder()
- GetOrder()

## Application

OrderService

- validate
- Save
- notify

## Outbound

IOrderRepository

IOrderNotifier

# Repository Adapter

```
/*------------------------------------------------------------------------
  Created by spirit-abl 1.2.0 (templates 1.2.0).
  Artifact: mod-c946af0feb/oe  Ownership: developer
  This file is yours: spirit-abl creates it once and never overwrites it.
------------------------------------------------------------------------*/
BLOCK-LEVEL ON ERROR UNDO, THROW.

/* OpenEdge repository for Order. Override the protected hooks of the
   generated base (MapFromBuffer, MapToBuffer, BeforeSave, AfterSave,
   BuildQuery) for database-specific mapping or query policy.
   Do not put business rules here — they belong in com.acme.sales.application.OrderService. */
CLASS com.acme.sales.adapters.outbound.openedge.OpenEdgeOrderRepository
  INHERITS com.acme.sales.adapters.outbound.openedge.generated.OpenEdgeOrderRepositoryBase:

  CONSTRUCTOR PUBLIC OpenEdgeOrderRepository():
    SUPER().
  END CONSTRUCTOR.

END CLASS.
```

# Notification Adapter

```
/*------------------------------------------------------------------------
  Created by spirit-abl 1.2.0 (templates 1.2.0).
  Artifact: mod-c946af0feb/email  Ownership: developer
  This file is yours: spirit-abl creates it once and never overwrites it.
------------------------------------------------------------------------*/
BLOCK-LEVEL ON ERROR UNDO, THROW.

/* Email notifier for Order events. Override the protected hooks of the
   generated base (FormatSubject, FormatBody, RecipientsFor, ShouldSend) to
   shape or route messages. Do not put business rules here — deciding WHEN an
   event happens belongs in com.acme.sales.application.OrderService. */
CLASS com.acme.sales.adapters.outbound.email.EmailOrderNotifier
  INHERITS com.acme.sales.adapters.outbound.email.generated.EmailOrderNotifierBase:

  CONSTRUCTOR PUBLIC EmailOrderNotifier():
    SUPER().
  END CONSTRUCTOR.

END CLASS.
```

# Application (Business Logic)

```
/*------------------------------------------------------------------------
  Created by spirit-abl 1.2.0 (templates 1.2.0).
  Artifact: mod-c946af0feb/service  Ownership: developer
  This file is yours: spirit-abl creates it once and never overwrites it.
------------------------------------------------------------------------*/
BLOCK-LEVEL ON ERROR UNDO, THROW.

/* Order application service — implement business rules here.
   Regeneration never touches this file. Use the injected Repository
   (IOrderRepository); never instantiate adapters here.
   Domain events go out through the Publish<Event>() helpers of the base
   (IOrderNotifier); call them only after the change succeeded. */
CLASS com.acme.sales.application.OrderService
  INHERITS com.acme.sales.application.generated.OrderServiceBase:

  CONSTRUCTOR PUBLIC OrderService():
    SUPER().
  END CONSTRUCTOR.

  METHOD PUBLIC OVERRIDE VOID SaveOrder(
    INPUT poOrder AS com.acme.sales.domain.Order):
    THIS-OBJECT:EnsureReady().
    IF poOrder = ? THEN
      UNDO, THROW NEW spirit.hex.error.ValidationError("Order required").
    poOrder:Validate().
    /* Add application-specific business rules here. */
    Repository:Save(poOrder).
    THIS-OBJECT:PublishOrderSaved(poOrder).
  END METHOD.

END CLASS.
```

# Wait. that’s not all the code.

Well, yes. We cheated. Claude did the hard bits

## Ai is great at generating

AI can follow patterns, obey corporate rules

Especially with a MCP server

## Boilerplate is cheap

Generated code takes most of the legwork out.

Concentrate on the business logic.

## Use overrides for custom code

Custom classes can have callbacks and overrides, allowing for customisation if needed

# Inbound Service Port

```
/*------------------------------------------------------------------------
  GENERATED by spirit-abl 1.2.0 (templates 1.2.0) — DO NOT EDIT.
  Artifact: mod-c946af0feb/inPort  Ownership: generated
  Regenerated deterministically; put custom code in the developer-owned
  subclass or extension point instead.
------------------------------------------------------------------------*/
/* Inbound (driving) port: the application capabilities of Order.
   Implemented by com.acme.sales.application.OrderService; called by inbound adapters. */
INTERFACE com.acme.sales.ports.inbound.IOrderService:

  /* Returns the Order or ? when it does not exist. */
  METHOD PUBLIC com.acme.sales.domain.Order GetOrder(
    INPUT piOrderId AS INTEGER).

  /* Validates and stores (insert or update) a Order. */
  METHOD PUBLIC VOID SaveOrder(
    INPUT poOrder AS com.acme.sales.domain.Order).

  /* Opens a row cursor; the caller MUST Close() it (use FINALLY). */
  METHOD PUBLIC com.acme.sales.ports.outbound.IOrderCursor ListOrders(
    INPUT poCriteria AS com.acme.sales.ports.outbound.OrderCriteria).

END INTERFACE.
```

# Notification Port

Reposito

```
/*------------------------------------------------------------------------
  GENERATED by spirit-abl 1.2.0 (templates 1.2.0) — DO NOT EDIT.
  Artifact: mod-c946af0feb/notifierPort  Ownership: generated
  Regenerated deterministically; put custom code in the developer-owned
  subclass or extension point instead.
------------------------------------------------------------------------*/
/* Outbound (driven) port: domain-event notifications published by the
   Order application after a mutating use case succeeds. Implemented by
   outbound adapters (email, RabbitMQ); exposes no transport types.

   Events:
     OrderSaved (order.saved) after SaveOrder

   Behavioural contract shared by every adapter:
   - Publish(?) raises spirit.hex.error.AdapterError INVALID_INPUT.
   - A transport failure raises AdapterError DB_UNAVAILABLE (endpoint
     unreachable) or NATIVE_ERROR (endpoint rejected the message); the
     application decides whether that fails the use case
     (see OnNotificationFailed in com.acme.sales.application.generated.OrderServiceBase).
   - Delivery is at-most-once and NOT transactional: an enclosing ABL
     transaction that later rolls back cannot recall a published event
     (docs/transactions.md). */
INTERFACE com.acme.sales.ports.outbound.IOrderNotifier:

  METHOD PUBLIC VOID Publish(
    INPUT poEvent AS com.acme.sales.ports.outbound.OrderEvent).

END INTERFACE.
```

# Repository Adapter (generated)

```
/* OpenEdge outbound adapter for Order over sports.Order.
   Extension points (override in com.acme.sales.adapters.outbound.openedge.OpenEdgeOr
   BeforeSave, AfterSave, BuildQuery, MapOrderLineFromBuffer, MapOrderLineToBuffer.
   BeforeSave/AfterSave run inside the transaction; throwing undoes it. */
CLASS com.acme.sales.adapters.outbound.openedge.generated.OpenEdgeOrderRepositoryBase
  INHERITS spirit.hex.runtime.BaseAdapter
  IMPLEMENTS com.acme.sales.ports.outbound.IOrderRepository,
             spirit.hex.runtime.IRowMapper,
             spirit.hex.contracts.ITransactionalResource:

  CONSTRUCTOR PROTECTED OpenEdgeOrderRepositoryBase():
    SUPER().
    Lifetime = spirit.hex.runtime.AdapterLifetime:PerSession.
  END CONSTRUCTOR.

  METHOD PUBLIC CHARACTER TransactionEngine():
    RETURN "openedge:sports".
  END METHOD.

  METHOD PUBLIC LOGICAL ParticipatesInAblTransaction():
    RETURN TRUE.
  END METHOD.

  METHOD PUBLIC com.acme.sales.domain.Order FindById(
      INPUT piOrderId AS INTEGER):
    DEFINE BUFFER bOrder FOR sports.Order.
    DEFINE VARIABLE oResult AS com.acme.sales.domain.Order NO-UNDO.
```

# Application (Business Logic) generated

```
/*------------------------------------------------------------------------      You,
  GENERATED by spirit-abl 1.2.0 (templates 1.2.0) — DO NOT EDIT.
  Artifact: mod-c946af0feb/serviceBase  Ownership: generated
  Regenerated deterministically; put custom code in the developer-owned
  subclass or extension point instead.
------------------------------------------------------------------------*/
BLOCK-LEVEL ON ERROR UNDO, THROW.

/* Application service base for Order. Implements the inbound port and
   invokes the outbound port. Simple declared reads are implemented here;
   mutating and custom use cases are ABSTRACT: their business rules belong in
   the developer-owned com.acme.sales.application.OrderService — this base never gues
   Dependencies are injected fluently and enforced by Seal() (spec 4.1). */
CLASS com.acme.sales.application.generated.OrderServiceBase ABSTRACT
  INHERITS spirit.hex.runtime.BaseService
  IMPLEMENTS com.acme.sales.ports.inbound.IOrderService:

  DEFINE PROTECTED PROPERTY Repository AS com.acme.sales.ports.outbound.IOrderReposit
    GET. PRIVATE SET.

  DEFINE PROTECTED PROPERTY Notifier AS com.acme.sales.ports.outbound.IOrderNotifier
    GET. PRIVATE SET.

  CONSTRUCTOR PROTECTED OrderServiceBase():
    SUPER().
  END CONSTRUCTOR.

  /* Required dependency. Assigns only; performs no I/O. */
  METHOD PUBLIC com.acme.sales.application.generated.OrderServiceBase WithRepository(
      INPUT poRepository AS com.acme.sales.ports.outbound.IOrderRepository):
    THIS-OBJECT:EnsureConfiguring().
    THIS-OBJECT:RequireArgument(poRepository, "IOrderRepository").
    THIS-OBJECT:CheckReinjection(Repository, "IOrderRepository").
```

# Generated vs Custom

Why do we need both types ?

## Generator owns this

CRUD, query plumbing, mapping, lifecycle, hooks

All the “plumbing” that every Business Logical needs

## Custom owns this

Custom mapping, query policy, overrides, special cases

Should be as small as possible

# The ABL problem

This should resonate immediately with you ...

## ABL Strengths

```
FOR EACH Customer:
    /* business rule */
    /* database access */
    /* UI behaviour */
    /* integration */
END.
```

## ABL makes database access wonderfully easy.

Which is exactly why it becomes wonderfully easy to couple everything to the database.

# Where does FIND / FOR EACH go

Or, where do I put my db access ?

## ABL Business Logic

```
    You, 6 minutes ago • Uncommitted cha
oOrder = Repository:FindById(piOrderId).

IF oOrder:Total > 10000 THEN
  oOrder:RequireApproval().

Repository:Save(oOrder).
```

## Openedge Adapter

```
FIND Order NO-LOCK
  WHERE Order.OrderId = piOrderId NO-ERROR.
```

ABL isn’t the problem.

# Hexagonal architecture just gives them a home.



# ABL is row-oriented

We all know this. ABL is not Typescript, or Go, or Java

## Instead of this

```
/* giant array */
repository.GetAllCustomers()
```

## Do this

```
oCursor = Repository:List(criteria).
DO WHILE oCursor:Next():
  oCustomer = oCursor:Current().
  ...
END.

FINALLY:
    oCursor:Close().
END FINALLY.
```

# Small ports beat God interfaces

Bigger is not always better

## Bad

IOpenEdgeDataService

- FindCustomer
- SaveCustomer
- FindOrder
- ...

## Good

ICustomerRepository

IOrderRepository

IOrderNotifier

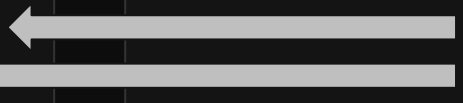
IExchangeRateProvider

# Injection without a DI framework

where ABL classes become a strength.

## ABL

```
oService = NEW OrderService()
  :WithRepository(oRepository)
  :WithNotifier(oNotifier)
  :Seal().
```



## Service only knows the interface

# Fail fast

where ABL classes become a strength.

## ABL

```
oService = NEW OrderService()
  :WithRepository(oRepository)
  :WithNotifier(oNotifier)
  :Seal().
```

## Validate configuration

Seal() verifies all required ports exist and prevents dependencies changing after startup.

# Wire it all up

Reposi

```
CLASS com.acme.sales.bootstrap.generated.OrderWiringBase ABSTRACT:
  /* Notifier kind used by Build(): email | rabbitmq | none.
     Read from SPIRIT_ORDER_NOTIFIER, when unset, "none" (the port is optional). */
  METHOD PUBLIC CHARACTER DefaultNotifierKind():
    RETURN THIS-OBJECT:Setting("SPIRIT_ORDER_NOTIFIER", "none").
  END METHOD.

  METHOD PUBLIC com.acme.sales.ports.inbound.IOrderService Build():
    RETURN THIS-OBJECT:BuildService(THIS-OBJECT:CreateRepository(THIS-OBJECT:DefaultAdapterKind()),
                                    THIS-OBJECT:CreateNotifier(THIS-OBJECT:DefaultNotifierKind())).
  END METHOD.

  METHOD PUBLIC com.acme.sales.ports.inbound.IOrderService BuildService(
      INPUT poRepository AS com.acme.sales.ports.outbound.IOrderRepository):
    RETURN THIS-OBJECT:BuildService(poRepository, ?).
  END METHOD.

  /* poNotifier may be ?: no events are published. */
  METHOD PUBLIC com.acme.sales.ports.inbound.IOrderService BuildService(
      INPUT poRepository AS com.acme.sales.ports.outbound.IOrderRepository,
      INPUT poNotifier   AS com.acme.sales.ports.outbound.IOrderNotifier):
    DEFINE VARIABLE oService AS com.acme.sales.application.OrderService NO-UNDO.
    oService = NEW com.acme.sales.application.OrderService().
    oService:WithRepository(poRepository).
    IF VALID-OBJECT(poNotifier) THEN
      oService:WithNotifier(poNotifier).
    THIS-OBJECT:ConfigureService(oService).
    oService:Seal().
    RETURN oService.
  END METHOD.
```

# There are multiple notification Adapters

Both implement `com.acme.sales.ports.outbound.IOrderNotifier`

## Email

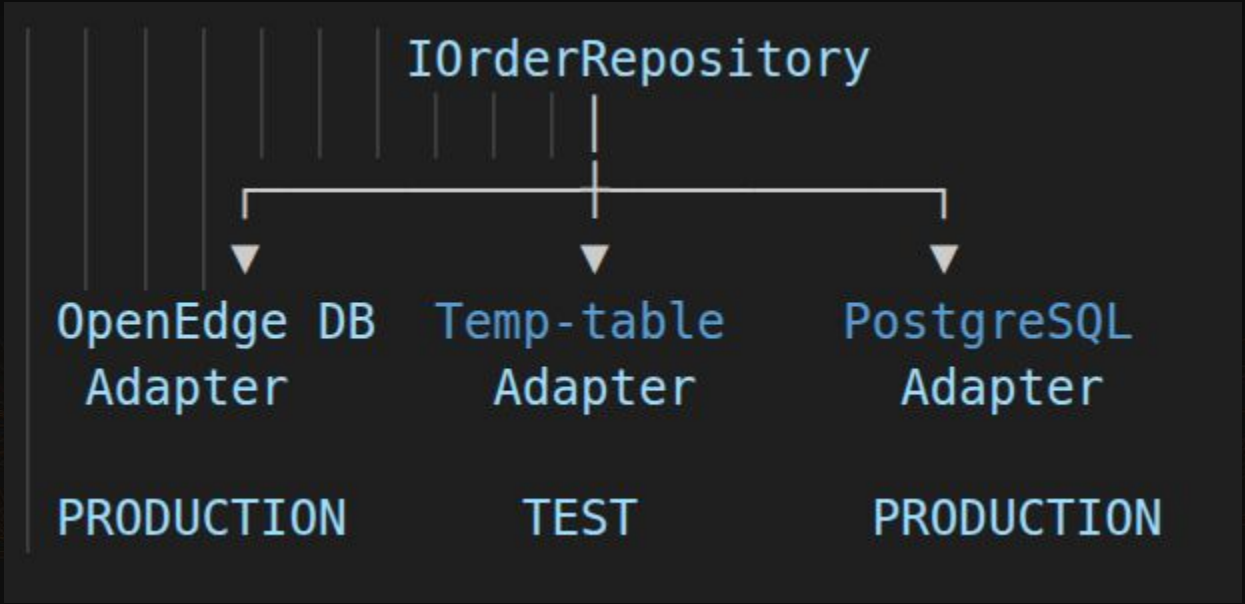
```
Lifetime: session; one SMTP connection per event, nothing held between calls. */
CLASS com.acme.sales.adapters.outbound.email.generated.EmailOrderNotifierBase ABSTRACT
  INHERITS spirit.hex.runtime.BaseAdapter
  IMPLEMENTS com.acme.sales.ports.outbound.IOrderNotifier:
```

## RabbitMQ

```
Lifetime: session; one HTTP request per event, nothing held between calls. */
CLASS com.acme.sales.adapters.outbound.rabbitmq.generated.RabbitMqOrderNotifierBase ABSTRACT
  INHERITS spirit.hex.runtime.BaseAdapter
  IMPLEMENTS com.acme.sales.ports.outbound.IOrderNotifier:
```

# There are multiple notification Adapters

OrderService doesn't know which one it got.



# Test it

```
/*------------------------------------------------------------------
  Created by spirit-abl 1.2.0 (templates 1.2.0).
  Artifact: mod-c946af0feb/serviceTest  Ownership: developer
  This file is yours: spirit-abl creates it once and never overwrites it.
------------------------------------------------------------------*/
BLOCK-LEVEL ON ERROR UNDO, THROW.
USING OpenEdge.Core.Assert.

  @Test.
  METHOD PUBLIC VOID SaveOrderPersists():
    DEFINE VARIABLE o AS com.acme.sales.domain.Order NO-UNDO.
    o = contracts.OrderRepositoryContract:Sample(4).
    oService:SaveOrder(o).
    contracts.OrderRepositoryContract:AssertSame(o, oRepository:FindById(o:OrderId)).
  END METHOD.

  METHOD PUBLIC VOID Setup():
    DEFINE VARIABLE oWiring AS com.acme.sales.bootstrap.OrderWiring NO-UNDO.
    oWiring = NEW com.acme.sales.bootstrap.OrderWiring().
    oRepository = oWiring:CreateFixtureRepository().
    oService = oWiring:BuildService(oRepository).
  END METHOD.
```

UNASHAMED PLUG!

# ABL NEEDS A WINGMAN TOMORROW

Come to the presentation for more on integrating ABL with other technology!

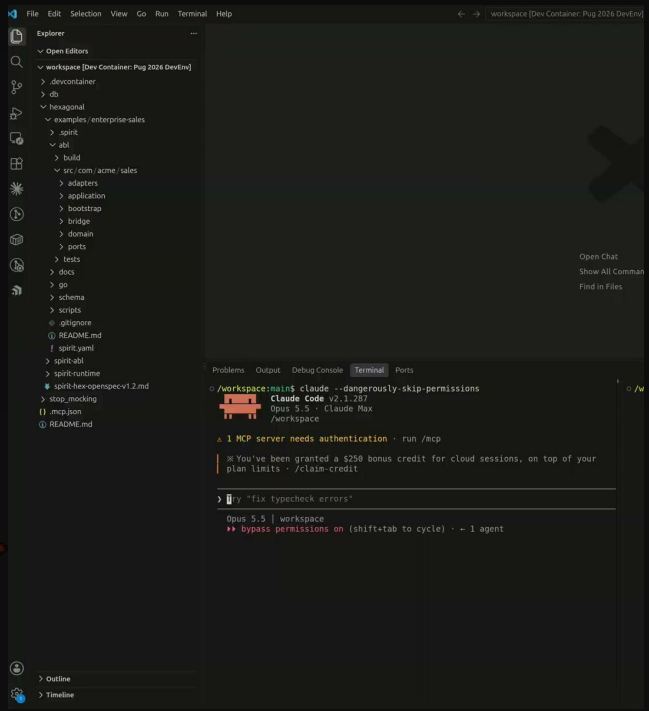
---

## LET ABL COOK. BRING BACKUP.



# Why use a framework and AI

Simple. Removes all the boring work `com.acme.sales.ports.outbound.IOrderNotifier`



# Don't ask AI to invent your architecture

AI decides what you want. The generator decides how it is built.

### "Claude, make me a Customer repository"

❌ Different structure every time
❌ Invented conventions
❌ Maybe valid ABL
❌ Maybe not

### Generate Customer

Spirit Hex MCP
↓
schema + rules + templates
↓
deterministic generator

### Consistency

Every new hexagon module has the same shape

Cheap on tokens

# Benefits of Hexagonal Architecture

Why adopt the hex ?

## Easier Testing

Swap real adapters for fake ones

Test business logic in isolation, with no database or network calls needed.

## Flexible Tech

Replace a database, message queue, or third-party service without touching core logic.

## Clearer Boundaries

Know what counts as a “business rule” and what counts as “plumbing” - makes the codebase easier to reason about

# Benefits of Hexagonal Architecture (cont.)

Why adopt the hex ?

## Multiple entry points

The same core logic can be driven by a REST API, a GraphQL API, a CLI tool, or a scheduled job, all at once, with zero duplication

## Maintainability

Business rules are not coupled to infrastructure.

They tend to survive framework upgrades, database migrations

## Infrastructure

Replace or upgrade the infrastructure code without having to touch the business logic.

Add new adapters for new tech.

# Challenges and Trade-offs

hexagonal architecture isn't free, so be honest about the costs before adopting it

## More upfront structure

Small CRUD apps with little business logic often don't need this level of separation, so it can feel like overkill.

## More code

Every port needs at least one adapter..

Generated code requires at least one base per adapter

## Developer learning curve

Understanding where to put code is initially challenging.

Generated code makes life so much easier

# Companies that use Hexagonal Architecture

Real world, real code

## Netflix

“Business logic should not depend on whether we expose a REST or a GraphQL API, and it should not depend on where we get data from”

## Qonto

“To keep each service’s domain fully separated from those of external systems, we chose to employ hexagonal architecture using Go”

## Cabify

“Hexagonal architecture provided the structure to model this domain cleanly.”

# Companies that use Hexagonal Architecture

Links to articles

Netflix, Qonto and Cabify give you particularly clear case studies.. Each explains the business problem the architecture helped solve.

Netflix : https://netflixtechblog.com/ready-for-changes-with-hexagonal-architecture-b315ec967749

Qonto: https://medium.com/qonto-way/transactions-in-go-hexagonal-architecture-f12c7a817a61

Cabify: https://tech.cabify.com/blog/engineering/where-is-my-domain

# When should you use Hexagonal Architecture ?

hexagonal architecture tends to pay off most when:

## Non-trivial business logic

Worth protecting

## Future-proofing

If you plan on changing databases, frameworks, apis etc

## Strong Testing

Automated test coverage

Standardized formats

# Hexagonal doesn't mean six sides

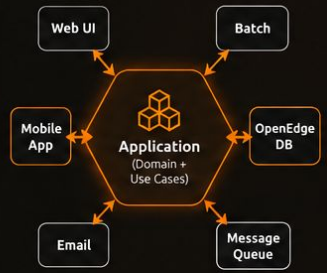
The hexagon is a drawing convention. What really matters is the boundary.

## 2 Ports



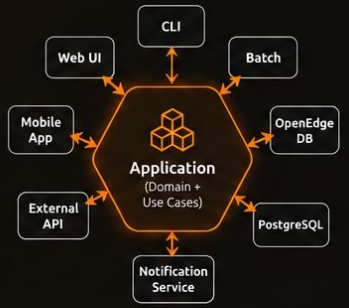

A simple application with one inbound and one outbound port.

## 6 Ports




The classic hexagon with multiple adapters on each side.

## 8 Ports




More ports, same principle. The shape is just a picture.

## 27 Ports




You can have 2, 6, 8, 27... however many your application needs.

### The key idea

It's not about six sides. It's about having a clear boundary between your application and the outside world, with explicit ports and adapters.

- The application has an **inside** and an **outside**.
- Communication crosses the boundary through **ports**.
- **Adapters** translate to the outside world.

**Same principle. Any shape. Infinite possibilities.**

# A better shape for OpenEdge systems

Clean architecture. Real-world ABL. Lasting options.

## The big idea

Keep your business logic at the centre, with clear boundaries to the outside world. Ports define the contracts. Adapters connect the technology.
The shape can be anything — it's the boundaries that matter.

## 1 Structure

Hexagonal Architecture




- Ports define the contracts
- Adapters connect technology
- Application is independent
- Works with any number of ports

## 2 ABL in practice

Natural ABL, better structure

```
/* Application */
oOrder = Repository:FindById(id).
IF oOrder:Total > 10000 THEN
   oOrder:RequireApproval().

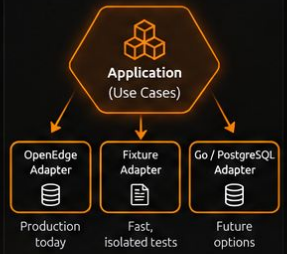
Repository:Save(oOrder).

/* Adapter */
FIND Order WHERE OrderId = id
   NO-LOCK NO-ERROR.
/* ABL stays ABL */
```

- FOR EACH in adapters
- Row-oriented processing
- Clean separation of concerns
- Familiar, maintainable ABL

## 3 Multiple adapters

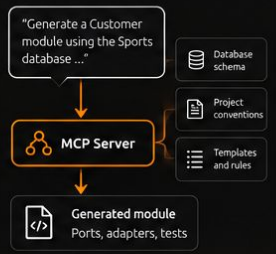
Same logic. Different world.




- Swap implementations
- Test without a database
- Add new technologies later
- Application code doesn't change

## 4 AI-assisted generation

Boilerplate is no longer the problem




- AI decides what you want
- Generator builds it consistently
- Follows your rules and standards
- You focus on the business logic

## 5 Real results

Maintainable systems with options

- **Flexible** — New adapters, new technologies
- **Testable** — Fast, isolated, reliable tests
- **Maintainable** — Clear structure, easier change
- **Future-ready** — Integrate with Go, PostgreSQL and modern services

- Protects your investment
- Supports gradual modernisation
- Works for real-world teams
- Built for the long term




**Same principle.**
**Any shape.**
**Infinite possibilities.**

# Questions ?