











Architecture & Process Diagram

Demo-G6 Refactoring, Regression Tests & Independency

1. Refactoring Process


```
graph LR
    A[" INNAN  
Monolitisk Kod"] -->|Refactoring| B[" EFTER  
3-Tier Arkitektur"]


    A --> A1[" Routes & Logic blandad  
 Data queries överallt  
 Tight coupling  
 Svårt att testa"]


    B --> B1[" Presentaton separat  
 Business logic i mitten  
 Data isolerad  
 Lätt att testa"]

    style A fill:#ffcccc
    style B fill:#ccffcc
    style A1 fill:#fff0f0
    style B1 fill:#f0fff0
```

2. Independency Layers

```
graph TB
    subgraph Presentation[" PRESENTATION LAYER  
(Routes & Templates)"]
        P1["Flask Routes  
HTTP Handlers  
Template Rendering"]
    end

    subgraph Business[" BUSINESS LAYER  
(Service)"]
        B1["Email Validation  
Duplicate Detection  
Data Normalisering"]
    end

    subgraph Data[" DATA LAYER  
(Repository)"]
    end
```

```

    D1["Database Operations
    CRUD Logic
    Query Abstraction"]
    end

    P1 -->|Uses| B1
    B1 -->|Uses| D1

    P_Benefit["✅ FÖRDELAR:
    • Kan ändra utan att påverka Business
    • Byt från Flask till Django
    • Lätt att maska för testning"]
    B_Benefit["✅ FÖRDELAR:
    • Kan testas utan Database
    • Kan testas utan HTTP
    • Återanvändbar från flera platser"]
    D_Benefit["✅ FÖRDELAR:
    • Byt från SQLite till PostgreSQL
    • Lätt att maska för testning
    • Centrliserad data-åtkomst"]

    style Presentation fill:#e3f2fd
    style Business fill:#f3e5f5
    style Data fill:#e8f5e9
    style P_Benefit fill:#e3f2fd,stroke:#1976d2,stroke-width:2px,text-align:left
    style B_Benefit fill:#f3e5f5,stroke:#7b1fa2,stroke-width:2px,text-align:left
    style D_Benefit fill:#e8f5e9,stroke:#388e3c,stroke-width:2px,text-align:left

```

3. Repository Pattern (Data Independency)

```

graph TB
    subgraph OldWay["❌ INNAN - Tight Coupling"]
        Service1["Service"]
        Service1 -->|Direct SQL| DB1["Database"]
    end

    subgraph NewWay["✅ EFTER - Repository Pattern"]
        Service2["Service"]
        Service2 -->|Interface| Repo["Repository"]
        Repo -->|SQL Implementation| DB2["Database"]
    end

    Benefits["REPOSITORY PATTERN FÖRDELAR:
    ✅ Service vet inte om Database detaljer
    ✅ Kan byta Database Implementation
    ✅ Kan testa Service med Mock Repository
    ✅ En plats för alla Data Queries"]

```

```

style OldWay fill:#ffebee
style NewWay fill:#e8f5e9
style Service1 fill:#ffcdd2
style DB1 fill:#ffcdd2
style Service2 fill:#c8e6c9
style Repo fill:#81c784
style DB2 fill:#c8e6c9
style Benefits fill:#fff9c4,stroke:#f57f17,stroke-width:2px

```

4. Dependency Injection (Loose Coupling)

```

graph LR
    subgraph Before["❌ TIGHT COUPLING"]
        S1["Service"]
        S1 -->|Creates| R1["Repository"]
        R1 -->|Creates| DB1["Database"]
        Note1["❌ Service är beroende  
av Repository implementering"]
    end

    subgraph After["✅ LOOSE COUPLING"]
        S2["Service"]
        S2 -->|Receives| R2["Repository Interface"]
        R2 -->|Can be any  
implementation| DB2["SQLite OR  
PostgreSQL OR  
Mock"]
        Note2["✅ Service är oberoende  
av Repository implementering"]
    end

    style Before fill:#ffebee
    style After fill:#e8f5e9
    style Note1 fill:#ffcdd2,text-align:center
    style Note2 fill:#c8e6c9,text-align:center
    style S2 fill:#a5d6a7,stroke:#2e7d32,stroke-width:2px
    style R2 fill:#81c784,stroke:#2e7d32,stroke-width:2px
    style DB2 fill:#c8e6c9,stroke:#2e7d32,stroke-width:2px

```

5. Regression Testing Cycle

```

graph TD
    A["📝 Skriv Test  
för nuvarande  
funktionalitet"] -->|Test fallerar först| B["🔴 RED  
Test misslyckas"]

```

```

    B -->|Implementera feature| C["🟢 GREEN
Test passar"]

    C -->|Refactor utan
att ändra beteende| D["🟡 REFACTOR
Förbättra kod"]

    D -->|Kör gamla tester| E["✅ Regression Tests
Säkerställer att
ingen funktionalitet
brast"]

    E -->|Confidence för
framtida ändringar| F["🚀 Safety Net
Kan ändra med
säkerhet"]

    Benefits1["TEST FÖRDELAR:
✅ Fångar bugs tidigt
✅ Dokumenterar beteende
✅ Ökar kodkvalitet
✅ Minskar human errors"]

    style A fill:#fff3e0
    style B fill:#ffcdd2
    style C fill:#c8e6c9
    style D fill:#bbdefb
    style E fill:#c8e6c9
    style F fill:#e1bee7
    style Benefits1 fill:#fff9c4,stroke:#f57f17,stroke-width:2px

```

6. Feature Independence

```

graph TB
    subgraph Features["🎯 INDEPENDENT FEATURES"]
        Joke["😄 Joke System"]
        nextJoke()
        Music Player
        Cloud Animations["☁️"]
        Subscribe["📧 Subscribe System"]
        Form Handling
        Validation
        Database Storage["🗄️"]
        Hero["🌟 Hero Section"]
        Typography
        Gradient
        Layout["📐"]
    end

```

Benefits["**FEATURE INDEPENDENCE FÖRDELAR:**

- ✓ Kan ta bort en feature utan att bryta andra
- ✓ Kan testa varje feature separat
- ✓ Kan vidareutveckla features oberoende
- ✓ Minimalt risk för side-effects"]

style Joke fill:#ffe0b2

style Subscribe fill:#c8e6c9

style Hero fill:#bbdefb

style Benefits fill:#fff9c4,stroke:#f57f17,stroke-width:2px

Joke -->|No Dependency| Subscribe

Subscribe -->|No Dependency| Hero

Hero -->|No Dependency| Joke

7. Complete Architecture Overview

```
graph TB
    subgraph Client["🖥️ CLIENT (Browser)"]
        HTML["HTML Templates"]
        CSS["CSS Styling"]
        JS["JavaScript Interactivity"]
    end

    subgraph Presentation["🌈 PRESENTATION LAYER"]
        Routes["Flask Routes @bp.route('/subscribe')"]
        Templates["Jinja2 Templates thank_you.html"]
        Static["Static Assets CSS, JS, Images"]
    end

    subgraph Business["🧠 BUSINESS LAYER"]
        Service["SubscriptionService"]
        Service -- validate_email()
        Service -- normalize_email()
        Service -- subscribe()
    end

    subgraph Data["💾 DATA LAYER"]
        Repository["SubscriberRepository"]
        Repository -- create()
        Repository -- find_by_email()
        Repository -- exists()
        Model["Subscriber Model"]
        Model -- id, email, name
        Model -- subscribed_at
    end
```

```

end

subgraph Database["🗄️ DATABASE"]
  SQLite["SQLite
news_flash.db"]
end

Client -->|HTTP Request| Routes
Routes -->|Render| Templates
Templates -->|Return HTML| Client
Client -->|CSS & JS| Static

Routes -->|Call| Service
Service -->|Call| Repository
Repository -->|Map to| Model
Model -->|SQL Queries| SQLite

PresentationBenefit["PRESENTATION:
✅ HTTP Handling
✅ Template Rendering
✅ Asset Management"]
BusinessBenefit["BUSINESS:
✅ Validering
✅ Normalisering
✅ Rules & Logic"]
DataBenefit["DATA:
✅ Database Abstraction
✅ Model Mapping
✅ Query Execution"]

style Client fill:#e0f2f1
style Presentation fill:#e3f2fd
style Business fill:#f3e5f5
style Data fill:#e8f5e9
style Database fill:#fff3e0

style PresentationBenefit fill:#e3f2fd,stroke:#1976d2,stroke-width:2px
style BusinessBenefit fill:#f3e5f5,stroke:#7b1fa2,stroke-width:2px
style DataBenefit fill:#e8f5e9,stroke:#388e3c,stroke-width:2px

```

8. Testing Strategy

```

graph TB
  subgraph UnitTests["🧪 UNIT TESTS
(Isoleraad testing)"]
    UT1["Test: validate_email()"]
    UT2["Test: normalize_email()"]
    UT3["Test: exists()"]
  end
end

```

```

    subgraph IntegrationTests["🔗 INTEGRATION TESTS
(Lager tillsammans)"]
        IT1["Test: subscribe() workflow"]
        IT2["Test: Database persistence"]
        IT3["Test: Duplicate detection"]
    end

    subgraph RegressionTests["🔄 REGRESSION TESTS
(Säkerställ ingenting brast)"]
        RT1["Gamla features fortsätter
att fungera"]
        RT2["Ingen oväntad side-effects"]
        RT3["Bakåtkompatibilitet"]
    end

    UnitTests -->|All Pass| Integration["✅ Integration Phase"]
    IntegrationTests -->|All Pass| Regression["✅ Regression Phase"]
    RegressionTests -->|All Pass| Deploy["🚀 Safe to Deploy"]

    UnitBenefit["UNIT TEST FÖRDELAR:
✅ Snabba att köra
✅ Lätt att debugga
✅ Hög test coverage möjlig"]

    IntegrationBenefit["INTEGRATION TEST FÖRDELAR:
✅ Testar verkligt workflow
✅ Fångar lager-problem
✅ Närmast produktion"]

    RegressionBenefit["REGRESSION TEST FÖRDELAR:
✅ Säkerhet för refactoring
✅ Förhindrar bugs
✅ Dokumenterar behavior"]

    style UnitTests fill:#fff3e0
    style IntegrationTests fill:#f1f8e9
    style RegressionTests fill:#fce4ec
    style Deploy fill:#c8e6c9,stroke:#2e7d32,stroke-width:3px

    style UnitBenefit fill:#fff3e0,stroke:#e65100,stroke-width:2px
    style IntegrationBenefit fill:#f1f8e9,stroke:#558b2f,stroke-width:2px
    style RegressionBenefit fill:#fce4ec,stroke:#c2185b,stroke-width:2px

```

9. Change Confidence Matrix

```

graph LR
    subgraph WithoutTests["❌ UTAN REGRESSION TESTS"]
        Risk["🔴 HIGH RISK"]
        Cannot1["Kan inte refactor
med säkerhet"]
    end

```

```

        Cannot2["Rädd för att ändra
gamma1 kod"]
        Cannot3["Bugs blir överraskningar"]
    end

    subgraph WithTests["✅ MED REGRESSION TESTS"]
        Safe["🟢 LOW RISK"]
        Can1["Kan refactor
med säkerhet"]
        Can2["Testar innan deploy"]
        Can3["Bugs fångas tidigt"]
    end

    style WithoutTests fill:#ffebee
    style WithTests fill:#e8f5e9
    style Risk fill:#ffcdd2,stroke:#c62828,stroke-width:2px
    style Safe fill:#c8e6c9,stroke:#2e7d32,stroke-width:2px

```

10. Development Workflow

```

graph LR
    A["1 REFACTOR  
Förbättra struktur"] -->|Creates| B["🌟 Clean Code  
with Clear Layers"]
    B -->|Enables| C["2 INDEPENDENCY  
Loose Coupling"]
    C -->|Allows| D["🔒 Isolated Testing  
Mock & Stub"]
    D -->|Requires| E["3 REGRESSION TESTS  
Safety Net"]
    E -->|Provides| F["🚀 Confidence  
Safe Changes"]
    F -->|Leads to| G["📈 QUALITY  
Maintainable Code"]

    style A fill:#fff3e0
    style C fill:#f3e5f5
    style E fill:#fce4ec
    style B fill:#fff9c4
    style D fill:#e8f5e9
    style F fill:#c8e6c9
    style G fill:#a5d6a7,stroke:#2e7d32,stroke-width:3px

```

Summary Table

Concept	Problem	Solution	Benefit
REFACTORING	Kod växer, blir svårt att underhålla	3-Tier Arkitektur, Separation of Concerns	Ren, organiserad kod
INDEPENDENCY	Ändringar påverkar allt	Loose Coupling, Dependency Injection	Säker, modulär design
REGRESSION TESTS	Räkna inte på manuell testning	Automatiserade tests för varje feature	Säkerhet för framtida ändringar

Key Takeaway

REFACTORING → INDEPENDENCY → REGRESSION TESTS = QUALITY CODE

- ✓ Kod som är lätt att förstå
- ✓ Kod som är lätt att testa
- ✓ Kod som är lätt att ändra
- ✓ Kod som är säker att deploya