Dependency Injection as a Programming Paradigm

CAB402 Assignment 2

Eric Joseph Kizhakkebhagathu
N11190728@qut.edu.au

Table of Contents

No table of contents entries found.

Dependency Injection as a Programming Paradigm

Introduction

Dependency Injection (DI) is an architectural design pattern that changes how programmers think about how to organise programs and how to assign tasks. Although most statically-typed languages offer DI through frameworks or libraries, it is not a built-in language feature or simple syntax shortcut. Rather, DI is a high-level design approach or a “programming paradigm” that teams can adopt to build more modular, loosely coupled systems.

DI is a technique in which objects get the dependencies they need from external sources (“an injector”) instead of creating those dependencies internally. (Spring, 2025). This means that a class no longer manages the construction of its collaborators. Instead, another component, usually an IoC (Inversion of Control) container or factory assembles the required objects and supplies them to the class. By removing such hard-coded dependencies, DI makes an application’s code easier to maintain and test (Microsoft, 2025).

Appendix

Appendix 1 : DI Introduction Code

```
// 1. Define an abstraction
public interface IMessageService
{
    void Send(string message);
}

// 2. Provide a concrete implementation
public class EmailService : IMessageService
{
    public void Send(string message)
    {
        Console.WriteLine($"[Email] {message}");
    }
}

// 3. Consume the service via constructor injection
public class NotificationManager
{
    private readonly IMessageService _messageService;

    // DI happens here: we ask for IMessageService rather than new one
ourselves
    public NotificationManager(IMessageService messageService)
    {
        _messageService = messageService;
    }

    public void Notify(string message)
    {
        _messageService.Send(message);
    }
}

// 4. Composition root: wire up dependencies and run
class Program
{
    static void Main()
    {
        // Manually create the dependency
        IMessageService emailSvc = new EmailService();

        // Inject it into the consumer
        var notifier = new NotificationManager(emailSvc);

        notifier.Notify("Dependency Injection in action!");
    }
}
```