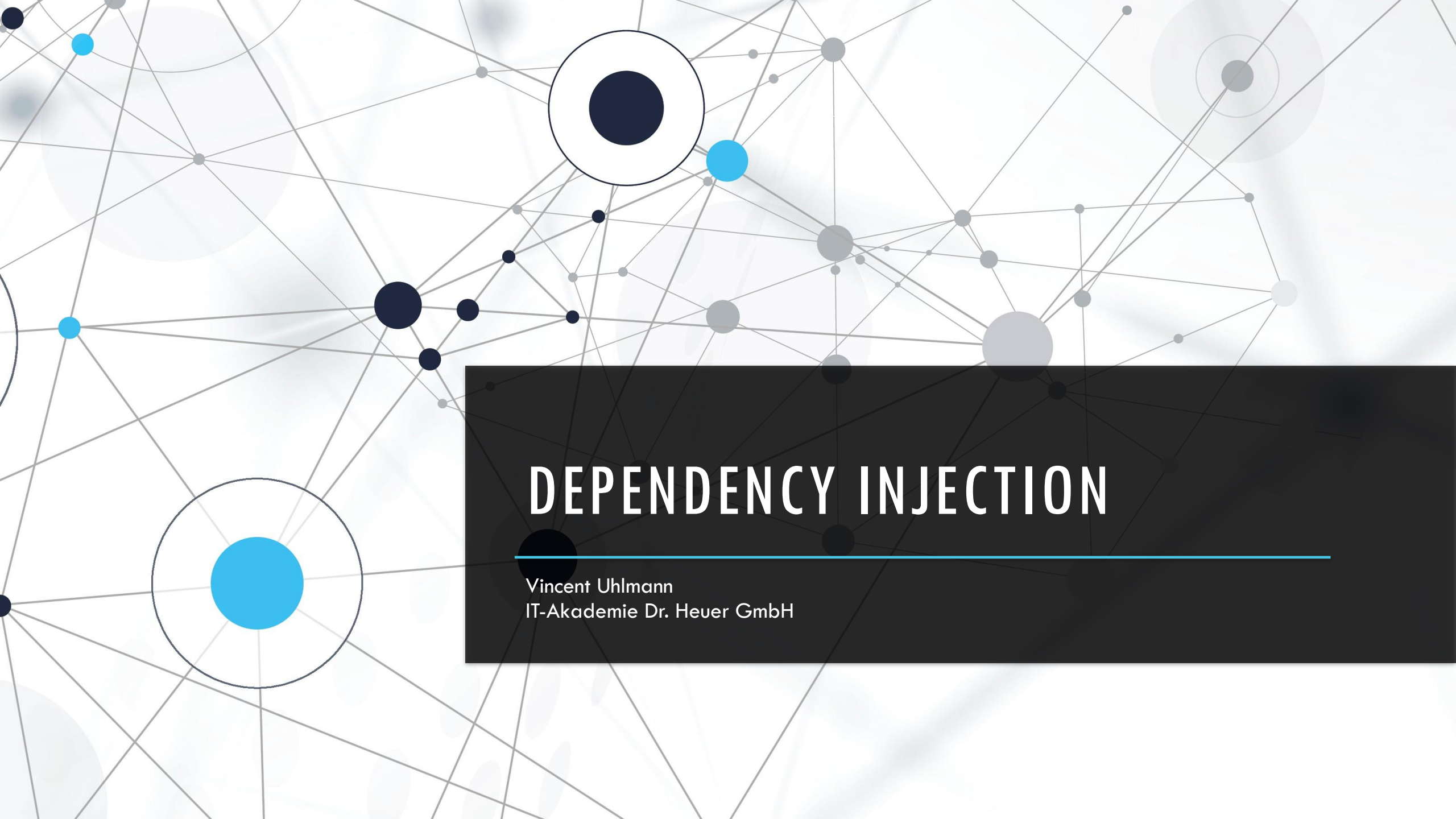

DEPENDENCY INJECTION

Vincent Uhlmann
IT-Akademie Dr. Heuer GmbH

DEPENDENCY INJECTION

- Ein Designmuster zur Verwaltung von Abhängigkeiten zwischen Objekten
- Reduziert die Kopplung und erhöht die Modularität des Codes
- Erleichtert das Management und die Erweiterbarkeit von Anwendungen

- Vereinfachung des Codes durch Trennung von Erstellung und Nutzung von Objekten
- Verbesserung der Testbarkeit durch Isolierung von Komponenten
- Flexibilität bei der Konfiguration und Erweiterung von Anwendungen

DEPENDENCY INJECTION

```
public class Client
{
    private IService _service;

    public Client(IService service)
    {
        _service = service;
    }
}
```

SERVICE COLLECTION

- **Was ist Service Collection?**
- Eine Klasse in .NET, die als Container für DI dient
- Verwaltet die Registrierung von Diensten und deren Lebensdauer

- **Verwendung von Service Collection**
- Registrierung von Diensten (Singleton, Scoped, Transient)
- Injektion von Abhängigkeiten über Konstruktoren oder Eigenschaften

SERVICE COLLECTION

- NuGet Packet: Microsoft.Extensions.DependencyInjection

```
ServiceCollection sv = new ServiceCollection();
sv.AddSingleton<IDataService, DataService>();

var provider = sv.BuildServiceProvider();

IDataService dataService = provider.GetRequiredService<IDataService>();
```

DEPENDENCY INJECTION

- **Vorteile von Interfaces in DI**
- Definieren klarer und konsistenter Vertragsbedingungen für Dienste
- Erleichtern das Mocking und Testen von Komponenten
- Unterstützen das Prinzip der Programmierung zur Schnittstelle, nicht zur Implementierung

- **Testbarkeit durch Interfaces**
- Mock-Objekte können leicht erstellt werden, um das Verhalten in Tests zu simulieren
- Ermöglicht Unit-Tests, die unabhängig von externen Ressourcen sind

DEPENDENCY INJECTION

- DataRepository kann in Tests durch MockupDataRepository ersetzt werden, um das tatsächliche Datenhandling zu simulieren

```
public interface IDataRepository
{
    IEnumerable<Data> GetAllData();
}

public class MockupDataRepository : IDataRepository
{
    public IEnumerable<Data> GetAllData()
    {
        // Rückgabe von Testdaten
        return new List<Data>();
    }
}
```

ILOGGER<T>

- ILogger ist ein Interface, das für die Protokollierung von Ereignissen verwendet wird
- Es bietet standardisierte Methoden zur Erstellung von Logs in Anwendungen
- Durch die Konfiguration von Log Levels, kann die Relevanz von Logs gesteuert werden
- Es sind verschiedene Implementationen vorhanden, die über NuGet Packete installiert werden können

ILOGGER<T> METHODEN

- **LogTrace**: Enthält ausführliche Meldungen und möglicherweise sensible Daten. Sollte niemals in einer Produktionsumgebung verwendet werden
- **LogDebug:** Zum Debuggen und für die Entwicklung
- **LogInformation:** Für den allgemeinen Ablauf der App, z.B. die Namen der aufgerufenen Methoden
- **LogWarning:** Für unerwartete Ereignisse, die allerdings nicht bewirken das die App abstützt oder nicht weiter verwendet werden kann
- **LogError:** Für Exceptions
- **LogCritical:** Für Fehler, die sofortige Aufmerksamkeit erfordern wie z.B. Datenverlust oder Speichermangel
- …

ILOGGER<T> IMPLEMENTATIONEN

- **ConsoleLogger:** Loggt Nachrichten in die Konsole
- **DebugLogger:** Loggt Nachrichten nur im Debug-Modus in Visual Studio oder anderen IDEs
- **EventLogger:** Loggt Nachrichten in das Windows-Ereignisprotokoll
- **FileLogger:** Loggt Nachrichten in eine Datei
- …

ILOGGER<T>

```
using ILoggerFactory factory = LoggerFactory.Create(builder => builder.AddConsole());
ILogger<Program> logger = factory.CreateLogger<Program>();
logger.LogInformation("Main");
logger.LogError("Invalid id exception");
logger.LogCritical("KRITISCHER FEHLER");
```

```
info: LoggerTest.Program[0]
      Main
fail: LoggerTest.Program[0]
      Invalid id exception
crit: LoggerTest.Program[0]
      KRITISCHER FEHLER
```

ILOGGER<T> & DI

```
#if DEBUG
    builder.Logging.AddConsole();

    // Optional
    builder.Logging.SetMinimumLevel(LogLevel.Debug);
#endif

    return builder.Build();
}
```

ILOGGER<T> & DI

```
public class SettingsViewModel
{
    private readonly ILogger<SettingsViewModel> _logger;

    public SettingsViewModel(ILogger<SettingsViewModel> logger)
    {
        _logger = logger;
        _logger.LogInformation("Created SettingsViewModel");
    }
}
```