MANTECH Publications

Journal of Android iOS Development and Testing
Volume 9, Issue 1, January-April 2024

Dependency Injection Using Dagger and Hilt

Pooja Sharma[1], Rohit Kumar[2]
Student[1], Lecturer[2]
Department of Information Technology
Aryavart Institute of Technology and Management
Email id: *poojasharma456@yahoo.com*

ABSTRACT

Dependency Injection (DI) has become a fundamental design pattern in modern Android application development due to its ability to improve modularity, testability, scalability, and maintainability of software systems. As Android applications grow in size and complexity, managing object creation and dependencies manually often leads to tightly coupled code, increased boilerplate, and difficulties in testing and maintenance. Dagger, a compile-time dependency injection framework developed by Google, addresses these challenges by providing a powerful and efficient DI solution. However, Dagger's steep learning curve and verbose configuration can be overwhelming for developers, especially beginners. To simplify this process, Google introduced Hilt, a higher-level dependency injection library built on top of Dagger, specifically designed for Android. This paper provides a comprehensive study of dependency injection concepts, the architecture and working principles of Dagger and Hilt, their components, scopes, lifecycle integration, advantages, limitations, and real-world use cases. A comparative analysis is also presented to highlight the evolution from manual DI to Dagger and further to Hilt. The paper aims to serve as an academic and practical reference for students, researchers, and Android developers seeking to understand and implement modern dependency injection techniques effectively.

KEYWORDS: *Dependency Injection, Android Development, Dagger, Hilt, Inversion of Control, Software Architecture*

INTRODUCTION

Modern Android applications are expected to be scalable, maintainable, and easy to test while supporting rapid feature development. Traditional approaches to object creation, where classes are responsible for instantiating their own dependencies, often result in tightly coupled systems. Such systems are difficult to modify, extend, or test independently. Dependency Injection (DI) is a design pattern that addresses these issues by externalizing the responsibility of providing dependencies to a dedicated framework or container.

In the Android ecosystem, dependency injection has evolved significantly over the years. Initially, developers relied on manual dependency injection using constructors or service locators. While effective for small applications, these approaches became difficult to manage as projects grew. To overcome these challenges, frameworks such as Dagger were introduced, offering compile-time dependency injection with strong performance guarantees. However, the complexity of Dagger configuration led to the development of Hilt, which provides a simplified API and better integration with Android lifecycle components.

This paper explores dependency injection in detail, focusing on Dagger and Hilt as industry-standard solutions for Android development.

FUNDAMENTALS OF DEPENDENCY INJECTION

Concept of Dependency Injection

Dependency Injection (DI) is a software design pattern that aims to reduce tight coupling between components by externalizing the responsibility of object creation and dependency management. In traditional programming approaches, a class often creates its own dependent objects internally using constructors or factory methods. This approach tightly binds the class to specific implementations, making the system rigid, difficult to test, and hard to extend.

Dependency Injection follows the principle of **Inversion of Control (IoC)**, where control over object creation is inverted from the class itself to an external framework or container. Instead of a class deciding *how* and *which* dependencies to create, these dependencies are provided to it from an external source. As a result, classes focus solely on their core responsibilities, improving separation of concerns.

In Android development, where applications consist of multiple interdependent components such as activities, fragments, view models, repositories, and services, DI plays a crucial role in managing complexity and ensuring maintainable architecture.

Inversion of Control and Dependency Management

Inversion of Control is a broader architectural principle, while Dependency Injection is one of its concrete implementations. IoC ensures that the flow of control is managed by a container or framework rather than by application-specific code. Dependency Injection implements IoC by supplying required objects at runtime or compile time instead of letting classes instantiate them directly.

Effective dependency management ensures that:

- Components remain loosely coupled
- Changes in one module do not heavily impact others
- Implementations can be replaced without modifying dependent classes

TYPES OF DEPENDENCY INJECTION

Constructor Injection

Constructor Injection is the most recommended and widely used form of dependency injection. In this approach, dependencies are passed to a class through its constructor at the time of object creation. This makes dependencies explicit and ensures that an object is always created in a valid state.

Advantages:

- Promotes immutability
- Improves readability and maintainability
- Simplifies unit testing

Field Injection

Field Injection involves injecting dependencies directly into class variables, usually using annotations. This approach is common in Android components such as activities and fragments where constructor modification is restricted.

Limitations:

- Dependencies are hidden and less explicit
- Makes unit testing more difficult

- Can lead to null-related issues if not handled properly

Method Injection

Method Injection supplies dependencies through setter methods or dedicated injection methods. This approach is useful when dependencies are optional or changeable during the lifecycle of an object.

Use cases:

- Runtime configuration changes
- Plugin-based architectures

MANUAL DEPENDENCY INJECTION VS FRAMEWORK-BASED DI

Manual Dependency Injection

In manual DI, developers create and pass dependencies explicitly without using a framework. While this approach offers full control and simplicity for small projects, it becomes unmanageable as application size grows.

Challenges of Manual DI:

- Increased boilerplate code
- Difficult dependency graph maintenance
- High risk of errors

Framework-Based Dependency Injection

Framework-based DI solutions like Dagger and Hilt automate dependency creation and injection. These frameworks maintain a dependency graph and ensure correct object provisioning based on defined rules and scopes.

Benefits:

- Compile-time safety
- Reduced boilerplate
- Consistent dependency lifecycle management

BENEFITS OF DEPENDENCY INJECTION IN ANDROID DEVELOPMENT

- **Improved Modularity:** Components can be developed and maintained independently.
- **Enhanced Testability:** Dependencies can be easily replaced with mock or fake implementations during testing.

- **Scalability:** DI frameworks efficiently manage growing dependency graphs in large applications.
- **Maintainability:** Changes in implementations do not require changes in dependent classes.
- **Reusability:** Well-defined components can be reused across different modules.

ROLE OF DEPENDENCY INJECTION IN CLEAN ARCHITECTURE

Dependency Injection plays a foundational role in implementing clean architecture principles. It ensures clear separation between presentation, domain, and data layers by decoupling interfaces from their implementations. This allows Android applications to evolve over time without major architectural refactoring.

By enforcing abstraction and controlled dependency flow, DI frameworks help developers build robust, flexible, and future-ready Android applications.

OVERVIEW OF DAGGER

Introduction to Dagger

Dagger is a fully static, compile-time dependency injection framework for Java and Kotlin, widely used in Android development. Unlike reflection-based DI frameworks, Dagger generates code at compile time, ensuring high performance and early error detection.

Core Components of Dagger

@INJECT

The `@Inject` annotation is used to request dependencies. It can be applied to constructors, fields, or methods.

@MODULE AND @PROVIDES

Modules define how to provide dependencies that cannot be constructor-injected. The `@Provides` annotation specifies the logic for creating objects.

@COMPONENT

Components act as bridges between modules and injection targets. They tell Dagger where dependencies are needed.

DAGGER SCOPES

Scopes define the lifecycle of dependencies. Common scopes include:

- `@Singleton`
- Custom scopes such as `@ActivityScope`

DAGGER ARCHITECTURE

Dagger builds a dependency graph at compile time. If any dependency is missing or incorrectly defined, compilation fails, ensuring safety and correctness.

LIMITATIONS OF DAGGER

Despite its strengths, Dagger has several limitations:

- Steep learning curve
- Extensive boilerplate code
- Complex setup for Android lifecycle components

These challenges motivated the development of Hilt.

INTRODUCTION TO HILT

WHAT IS HILT?

Hilt is a dependency injection library built on top of Dagger that simplifies DI implementation in Android applications. It provides predefined components, automatic lifecycle management, and reduced boilerplate.

DESIGN GOALS OF HILT

- Simplify Dagger usage
- Standardize DI across Android apps
- Seamless integration with Android components

HILT ARCHITECTURE AND COMPONENTS

ANDROID-ALIGNED COMPONENTS

Hilt defines a set of standard components that align with Android lifecycles:

- ApplicationComponent
- ActivityComponent

- FragmentComponent
- ViewModelComponent
- ServiceComponent

@HILTANDROIDAPP

This annotation triggers Hilt's code generation and is applied to the Application class.

@ANDROIDENTRYPOINT

Used to enable field injection in Android components such as activities and fragments.

@HILTVIEWMODEL

Simplifies dependency injection in ViewModels.

LIMITATIONS OF DAGGER

Despite its strengths, Dagger has several limitations:

- Steep learning curve
- Extensive boilerplate code
- Complex setup for Android lifecycle components

These challenges motivated the development of Hilt.

COMPARISON BETWEEN DAGGER AND HILT

Feature	Dagger	Hilt
Setup Complexity	High	Low
Boilerplate Code	Extensive	Minimal
Lifecycle Awareness	Manual	Automatic
Android Integration	Limited	Native

USE CASES AND PRACTICAL APPLICATIONS

ENTERPRISE ANDROID APPLICATIONS

Large-scale applications benefit from DI frameworks to manage complex dependency graphs.

TESTING AND MOCKING

Dependency injection simplifies unit testing by allowing easy replacement of real dependencies with mock objects.

MODULAR ARCHITECTURES

DI supports clean architecture and modular design by decoupling layers such as UI, domain,

and data.

PERFORMANCE CONSIDERATIONS

Both Dagger and Hilt use compile-time code generation, ensuring minimal runtime overhead. Hilt does not introduce significant performance penalties compared to Dagger.

CHALLENGES AND BEST PRACTICES

COMMON CHALLENGES

- Misconfigured scopes
- Over-injection
- Tight coupling through incorrect module design

BEST PRACTICES

- Prefer constructor injection
- Use appropriate scopes
- Keep modules small and focused

FUTURE TRENDS IN DEPENDENCY INJECTION

With the rise of Kotlin-first development and Jetpack Compose, DI frameworks are evolving to support new paradigms. Hilt continues to be actively developed to meet modern Android requirements.

CONCLUSION

Dependency Injection is a critical architectural pattern in modern Android development. Dagger provides a powerful, efficient, and compile-time safe DI solution, while Hilt builds upon Dagger to offer a more developer-friendly and Android-centric approach. By reducing boilerplate and handling lifecycle complexities automatically, Hilt lowers the barrier to adopting dependency injection without sacrificing performance or flexibility. Understanding both Dagger and Hilt enables developers to design scalable, maintainable, and testable Android applications. As Android development continues to evolve, dependency injection using Dagger and Hilt will remain a cornerstone of robust application architecture.

REFERENCES

1. Google Developers. (2023). *Dependency Injection in Android*. https://developer.android.com

2. Fowler, M. (2004). *Inversion of Control Containers and the Dependency Injection Pattern*. https://martinfowler.com
3. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley
4. Android Developers. (2024). *Hilt for Android*. https://developer.android.com/training/dependency-injection/hilt