SpecificationKit Project Architecture (Clean Domain-Agnostic)

This document outlines the domain-agnostic architecture for a Swift library implementing the Specification Pattern using macros, property wrappers, and context providers.

🧱 Hierarchical Architecture Overview

1. Core Layer – Foundational abstractions

Subsystem	Purpose
`Specification`	Protocol with `isSatisfiedBy` for T
`AnySpecification<T>`	Type-erased wrapper for any specification
`SpecificationOperators`	`.and()`, `.or()`, `.not()` and `&&` overloads
`ContextProviding`	Generic protocol for context suppliers

---

2. Composables Layer – Concrete reusable specifications

File / Module	Purpose
`TimeSinceEventSpec`	Checks if a minimum duration has passed
`MaxCountSpec`	Checks if a counter is below threshold
`CooldownIntervalSpec`	Ensures enough time has passed since last event
`PredicateSpec<T>`	Accepts closure for arbitrary logic

---

3. Macro Layer *(optional)*

Component	Purpose
`@specs(...)`	Generates composite spec from multiple inner specs
`@AutoContext`	(future) derives context from environment automatically

---

4. Property Wrapper Layer

Wrapper	Purpose
`@Satisfies`	Applies specification with auto context provider
`@Satisfies(context:)`	Manual context override variant

---

5. Specification Definitions Layer

Structure	Purpose
`CompositeSpec<T>`	Example of combining multiple specifications

`AutoContextSpecification`	Protocol for specs that encapsulate their context provider

6. Context Layer – Context generation and configuration

Component	Purpose
`EvaluationContext`	Holds data needed to evaluate a specification
`DefaultContextProvider`	Supplies context from runtime state or injected environment
`MockContextProvider`	Used in unit tests

7. Application Integration Layer

Example usage in an app (SwiftUI / UIKit / CLI):

```swift
@Satisfies(using: CompositeSpec.self)
var result: Bool
```

8. Tests Layer

Tests	What they cover
`*SpecTests`	Unit tests for individual specs
`CompositeSpecTests`	Logical operator correctness
`SatisfiesWrapperTests`	Property wrapper behavior
`MockProviderTests`	Provider injection and override behavior

📦 Suggested SwiftPM Folder Structure

```
Sources/
├── SpecificationKit/
│   ├── Core/
│   │   ├── Specification.swift
│   │   ├── AnySpecification.swift
│   │   ├── SpecificationOperators.swift
│   │   ├── ContextProviding.swift
│   ├── Specs/
│   │   ├── TimeSinceEventSpec.swift
│   │   ├── MaxCountSpec.swift
│   │   ├── CooldownIntervalSpec.swift
│   │   ├── PredicateSpec.swift
│   ├── Providers/
│   │   ├── EvaluationContext.swift
│   │   ├── DefaultContextProvider.swift
│   ├── Wrappers/
│   │   ├── Satisfies.swift
│   ├── Definitions/
│   │   ├── CompositeSpec.swift
│   └── Macros/ (optional)
│       ├── SpecsMacro.swift
│       ├── MacroPlugin.swift

Tests/
├── SpecificationKitTests/
```

```
│   ├── TimeSinceEventSpecTests.swift
│   ├── CompositeSpecTests.swift
│   ├── SatisfiesWrapperTests.swift
│   ├── ...
```

✅ Summary

- **Core** — reusable logic and protocols
- **Specs** — standalone reusable specification components
- **Definitions** — composed specs for example domains
- **Wrappers** — integration with Swift property system
- **Providers** — contextual runtime environment logic
- **Macros** — optional Swift macro extensions
- **Tests** — full test coverage for specs, wrappers, and logic