Architectural Decision Enforcement

From Architectural Decisions to Automated Conformance Checks

Graduate

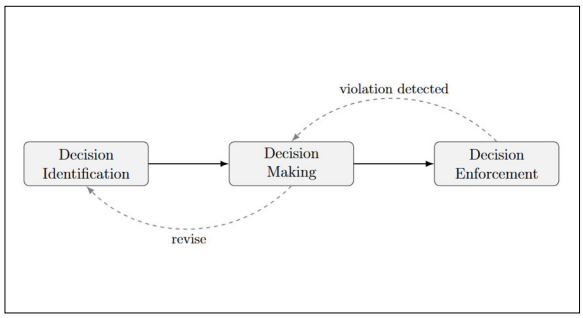
Raphael Schellander

Problem: Architectural decisions play a central role in software architecture and are often documented using Architectural Decision Records (ADRs). Managing such decisions involves identifying architecturally significant choices, documenting them as ADRs, and enforcing them as the codebase evolves. In practice, enforcement is often the weakest phase and is usually left to costly and inconsistent code reviews. Existing architecture conformance tools can automate checks on code, but they are disconnected from the ADRs that motivated their rules. This gap can lead to architectural erosion over time.

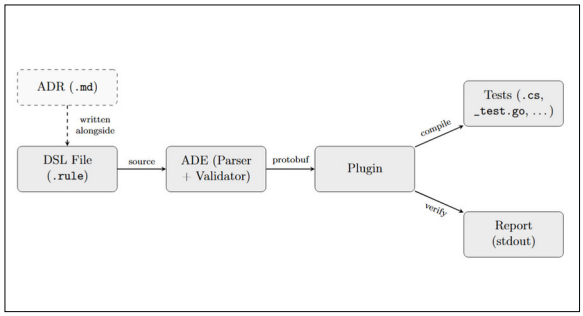
Objective: This thesis proposes to bridge the gap between decision documentation and automated conformance checking with the Architectural Decision Enforcement (ADE) tool. ADE introduces a tool-agnostic Domain-Specific Language (DSL) in which rule files, located alongside ADRs, express enforceable constraints without knowledge of specific testing frameworks. To decouple rule specification from execution, we developed a compiler architecture consisting of a parser, semantic validation, and an intermediate representation based on Protocol Buffers. This allows the same rule file to be used with multiple enforcement frameworks. We further define a plugin protocol that enables independently developed plugins to target different language ecosystems.

Result: The proposed approach supports architects in enforcing architectural decisions throughout software evolution. We implemented a command-line tool with reference plugins for Go, .NET, and file system assertions. Applying ADE to an open-source .NET project containing 17 ADRs showed that 9 decisions could be automatically enforced, resulting in 44 generated checks. As part of our action research, an architect applied ADE to a multi-language project and developed an independent Java plugin. This demonstrated that the plugin protocol can be applied to new ecosystems without changes to the ADE core. The validation activities indicate that a tool-agnostic DSL combined with a plugin-based execution model can effectively connect ADRs with automated conformance checks, minimizing architectural erosion.

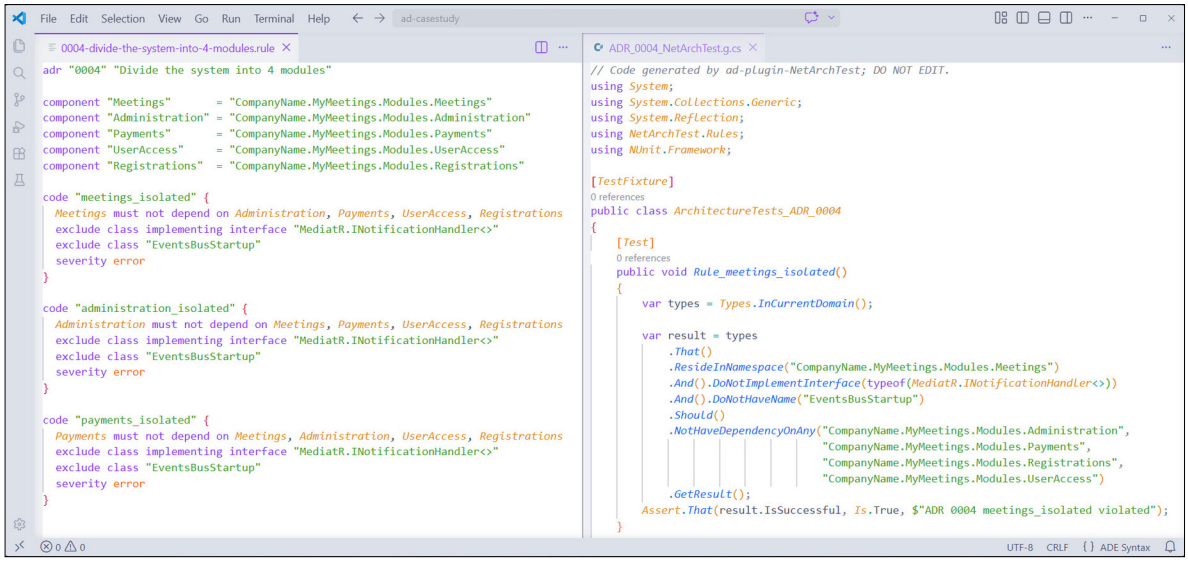
The three-phase architectural decision lifecycle.
Own presentment



Enforcement pipeline: from ADR to conformance output.
Own presentment



Example rules from case study (left) and a C# test of the first rule generated by the NetArchTest plugin (right).
Own presentment



```
adr "0004" "Divide the system into 4 modules"

component "Meetings"       = "CompanyName.MyMeetings.Modules.Meetings"
component "Administration" = "CompanyName.MyMeetings.Modules.Administration"
component "Payments"       = "CompanyName.MyMeetings.Modules.Payments"
component "UserAccess"     = "CompanyName.MyMeetings.Modules.UserAccess"
component "Registrations"  = "CompanyName.MyMeetings.Modules.Registrations"

code "meetings_isolated" {
  Meetings must not depend on Administration, Payments, UserAccess, Registrations
  exclude class implementing interface "MediatR.INotificationHandler<>"
  exclude class "EventsBusStartup"
  severity error
}

code "administration_isolated" {
  Administration must not depend on Meetings, Payments, UserAccess, Registrations
  exclude class implementing interface "MediatR.INotificationHandler<>"
  exclude class "EventsBusStartup"
  severity error
}

code "payments_isolated" {
  Payments must not depend on Meetings, Administration, UserAccess, Registrations
  exclude class implementing interface "MediatR.INotificationHandler<>"
  exclude class "EventsBusStartup"
  severity error
}
```

```
// Code generated by ad-plugin-NetArchTest; DO NOT EDIT.
using System;
using System.Collections.Generic;
using System.Reflection;
using NetArchTest.Rules;
using NUnit.Framework;

[TestFixture]
0 references
public class ArchitectureTests_ADR_0004
{
    [Test]
    0 references
    public void Rule_meetings_isolated()
    {
        var types = Types.InCurrentDomain();

        var result = types
            .That()
            .ResideInNamespace("CompanyName.MyMeetings.Modules.Meetings")
            .And().DoNotImplementInterface(typeof(MediatR.INotificationHandler<>))
            .And().DoNotHaveName("EventsBusStartup")
            .Should()
            .NotHaveDependencyOnAny("CompanyName.MyMeetings.Modules.Administration",
                                    "CompanyName.MyMeetings.Modules.Payments",
                                    "CompanyName.MyMeetings.Modules.Registrations",
                                    "CompanyName.MyMeetings.Modules.UserAccess")
            .GetResult();
        Assert.That(result.IsSuccessful, Is.True, $"ADR 0004 meetings_isolated violated");
    }
```


Advisor
Stefan Kapferer

Co-Examiner
Dr. Gerald Reif, Innovation Process Technology

Subject Area
Computer Science