Resenha – *Hotspot Patterns: The Formal Definition and Automatic Detection of Architecture Smells*

Kaio Souza Oliveira Mayer

O artigo de Ran Mo, Yuanfang Cai, Rick Kazman e Lu Xiao discute a detecção de problemas recorrentes em arquiteturas de software, chamados *hotspot patterns*. Esses padrões representam pontos críticos que aumentam o custo de manutenção e a propensão a erros em sistemas complexos. Os autores propõem cinco tipos principais: Unstable Interface, Implicit Cross-module Dependency, Unhealthy Inheritance Hierarchy, Cross-Module Cycle e Cross-Package Cycle.

A pesquisa se baseia na teoria de *Design Rules* de Baldwin e Clark, que considera que sistemas bem modularizados devem ter regras de projeto estáveis e módulos verdadeiramente independentes. No entanto, na prática, muitos sistemas violam esses princípios, gerando dependências ocultas, interfaces instáveis e ciclos prejudiciais que tornam o software mais difícil de evoluir.

Os autores também desenvolveram uma ferramenta capaz de detectar automaticamente esses hotspots a partir de informações estruturais do código e do histórico de evolução. Essa abordagem foi validada em nove projetos open source (como Hadoop, Cassandra e HBase) e em um projeto comercial, mostrando que arquivos envolvidos em hotspots apresentam taxas significativamente maiores de bugs e mudanças. Um dado interessante é que quanto mais padrões um arquivo acumula, mais propenso a falhas e retrabalho ele se torna.

No estudo qualitativo, arquitetos de um projeto industrial confirmaram que os hotspots identificados correspondiam de fato aos principais pontos de dor de manutenção, e que a ferramenta fornece orientações úteis para planejar refatorações. Entre os problemas encontrados estavam interfaces instáveis que haviam se tornado *God Interfaces* e dependências implícitas entre módulos que deveriam ser independentes.

Esse artigo mostra como a Engenharia de Software vai além de apenas escrever código limpo: é necessário entender a arquitetura e os impactos das decisões de design no longo prazo. Achei muito interessante a relação direta entre hotspots e esforço de manutenção, porque conecta teoria a problemas reais que enfrentamos em projetos. Para nós, estudantes, a ideia de usar análise histórica junto com a estrutura do código abre caminho para pensar em ferramentas que ajudem a prever e corrigir problemas antes que eles se tornem dívidas técnicas.