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

O artigo "Hotspot Patterns: The Formal Definition and Automatic Detection of Architecture Smells", de Ran Mo, Yuanfang Cai, Rick Kazman e Lu Xiao, propõe e valida empiricamente um conjunto de "padrões de hotspot" (hotspot patterns), que são problemas arquitetônicos recorrentes em sistemas complexos e que acarretam altos custos de manutenção. O trabalho introduz dois novos padrões de hotspot: `Unstable Interface` (Interface Instável) e `Implicit Cross-module Dependency` (Dependência Implícita entre Módulos).

Introdução e Motivação

Pesquisas anteriores no campo da previsão e localização de defeitos focaram em identificar problemas no código-fonte usando métricas estruturais e histórico de evolução. No entanto, os autores observaram que os arquivos mais propensos a erros são tipicamente altamente conectados arquitetonicamente e podem ser capturados por poucos `Design Rule Spaces` (DRSpaces). Cada DRSpace propenso a erros frequentemente apresenta múltiplos problemas arquitetônicos.

Após examinar centenas de DRSpaces em diversos projetos de código aberto e comerciais, os autores identificaram que existem poucos tipos distintos de problemas arquitetônicos que se repetem. Esses problemas, embora associados a alta propensão a erros e/ou mudanças, não são caracterizados por noções existentes como *code smells* ou *anti-patterns*, e, portanto, não são detectáveis automaticamente por ferramentas atuais. Isso levanta a questão de como gerentes de projeto ou arquitetos podem priorizar a manutenção e refatoração do código.

Padrões de Hotspot Propostos

Baseando-se na teoria das regras de design de Baldwin e Clark [1] e em princípios básicos de design de software, os autores resumem os problemas arquitetônicos recorrentes em cinco padrões de hotspot, que são frequentemente as causas raiz de altos custos de manutenção:

1. **Unstable Interface (Interface Instável):** Definido pela premissa de que interfaces importantes (regras de design) devem ser estáveis [1].
2. **Implicit Cross-module Dependency (Dependência Implícita entre Módulos):** Baseado no conceito de módulos verdadeiramente independentes, conforme descrito pela teoria das regras de design [1, 26], e revelado por um agrupamento hierárquico de regras de design [30]. Este padrão visa revelar dependências ocultas que conectam módulos que parecem ser mutuamente independentes [29].
3. **Unhealthy Inheritance Hierarchy (Hierarquia de Herança Não Saudável):** Detecta estruturas arquitetônicas que violam a teoria das regras de design ou os princípios de substituição de Liskov.
4. **Cross-Module Cycle (Ciclo entre Módulos):** Baseado na racionalidade de formar uma estrutura hierárquica adequada entre os módulos.
5. **Cross-Package Cycle (Ciclo entre Pacotes):** Similar ao anterior, mas definido no nível de pacotes.

Os padrões `Unstable Interface`, `Implicit Cross-module Dependency` e `Unhealthy Inheritance Hierarchy` não foram formalmente definidos antes e não são detectáveis por ferramentas existentes. Os autores definem "módulos" para `Implicit Cross-module Dependency` e `Cross-module Dependencies` como grupos de arquivos mutuamente independentes revelados por uma hierarquia de regras de design (DRH) [30].

DRSpace e Detecção de Hotspots

O conceito fundamental por trás dos padrões de hotspot é o `Design Rule Space` (DRSpace). Em vez de ver a arquitetura de software como um simples conjunto de componentes e relações, os autores a consideram estruturada por regras de design e módulos independentes, seguindo a teoria de Baldwin e Clark [1]. Regras de design

refletem as decisões arquitetônicas mais importantes que desacoplam o restante do sistema em módulos independentes, manifestando-se geralmente como interfaces ou classes abstratas.

Um DRSpace contém um conjunto de arquivos e um conjunto selecionado de relações (herança, agregação, dependência). Esses arquivos são agrupados em uma DRH [3, 30, 4] para manifestar a existência de regras de design e módulos independentes. A visualização de um DRSpace é feita usando uma `Design Structure Matrix` (DSM), que pode ser anotada para iluminar relações estruturais e evolutivas entre os arquivos (co-mudanças).

Os autores desenvolveram uma ferramenta, o `Hotspot Detector`, que detecta automaticamente instâncias desses padrões de hotspot. A ferramenta utiliza como entrada um arquivo DSM com dependências estruturais (SDSM), um arquivo DSM com informações de acoplamento evolutivo (HDSM) e um arquivo de agrupamento que contém a estrutura de pacotes ou o agrupamento DRH.

Avaliação Quantitativa e Qualitativa

O estudo avaliou os padrões usando nove projetos de código aberto Apache e um projeto comercial. A análise quantitativa investigou:

1. Se os hotspots realmente capturam problemas arquitetônicos que geram custos de manutenção caros em termos de propensão a erros e mudanças.
2. Se um arquivo envolvido em um número maior de padrões de hotspot é mais propenso a erros/mudanças.
3. Quais tipos de problemas arquitetônicos causam mais propensão a erros e mudanças.

Os resultados mostraram que os arquivos envolvidos nesses padrões têm taxas de bugs e mudanças significativamente mais altas. A propensão a erros e mudanças aumenta drasticamente com o número de padrões de hotspot em que um arquivo está envolvido. O padrão `Unstable Interface` teve a contribuição mais significativa para a propensão a erros e mudanças.

Na avaliação qualitativa, um estudo de caso industrial confirmou que o `Hotspot Detector` descobriu a maioria dos problemas arquitetônicos que causavam dificuldades de manutenção. Os arquitetos e desenvolvedores foram capazes de

identificar dependências ocultas e interfaces que haviam se tornado "interfaces Deus", necessitando de refatoração.

Conclusão

O artigo conclui que os padrões de hotspot são ubíquos e contribuem significativamente para a propensão a erros e mudanças, e, consequentemente, para o esforço de manutenção. O `Unstable Interface` e o `Cross-Module Cycle` são os que mais contribuem para esses problemas. A abordagem proposta é eficaz em ajudar desenvolvedores a encontrar problemas estruturais que afetam o esforço de manutenção e a guiar a refatoração.

Referências

[1] C. M. Baldwin and K. B. Clark. *Design Rules: The Power of Modularity*. MIT Press, 2000. [3] Y. Cai, R. Mo, and R. Kazman. Design rule hierarchy: A new approach to software architecture modeling. In *Proceedings of the 2012 34th International Conference on Software Engineering (ICSE)*, pages 1120–1123. IEEE Press, 2012. [4] Y. Cai, R. Mo, and R. Kazman. Design rule hierarchy: A new approach to software architecture modeling. In *Proceedings of the 2012 34th International Conference on Software Engineering (ICSE)*, pages 1120–1123. IEEE Press, 2012. [26] K. B. Clark and C. M. Baldwin. *Productivity and Technology in the Automobile Industry*. Brookings Institution Press, 1991. [29] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, October 1992. [30] R. Mo, Y. Cai, and R. Kazman. Design rule hierarchy: A new approach to software architecture modeling. In *Proceedings of the 2012 34th International Conference on Software Engineering (ICSE)*, pages 1120–1123. IEEE Press, 2012. [31] R. Mo, Y. Cai, and R. Kazman. Design rule space: A new approach to software architecture modeling. In *Proceedings of the 2012 34th International Conference on Software Engineering (ICSE)*, pages 1120–1123. IEEE Press, 2012.