Toward a Catalogue of Architectural Bad Smells

Joshua Garcia, Daniel Popescu, George Edwards, and Nenad Medvidovic
University of Southern California, Los Angeles, CA, USA
{joshuaga,dpopescu,gedwards,neno}@usc.edu

Abstract. An architectural bad smell is a commonly (although not always intentionally) used set of architectural design decisions that negatively impacts system lifecycle properties, such as understandability, testability, extensibility, and reusability. In our previous short paper, we introduced the notion of architectural bad smells and outlined a few common smells. In this paper, we significantly expand upon that work. In particular, we describe in detail four representative architectural smells that emerged from reverse-engineering and re-engineering two large industrial systems and from our search through case studies in research literature. For each of the four architectural smells, we provide illustrative examples and demonstrate the smell’s impact on system lifecycle properties. Our experiences indicate the need to identify and catalog architectural smells so that software architects can discover and eliminate them from system designs.

1 Introduction

As the cost of developing software increases, so does the incentive to evolve and adapt existing systems to meet new requirements, rather than building entirely new systems. Today, it is not uncommon for a software application family to be maintained and upgraded over a span of five years, ten years, or longer. However, in order to successfully modify a legacy application to support new functionality, run on new platforms, or integrate with new systems, evolution must be carefully managed and executed. Frequently, it is necessary to refactor, or restructure the design of a system, so that new requirements can be supported in an efficient and reliable manner.

The most commonly used way to determine how to refactor is to identify code bad smells [2, 1]. Code smells are implementation structures that negatively affect system lifecycle properties, such as understandability, testability, extensibility, and reusability; that is, code smells ultimately result in maintainability problems. Common examples of code smells include very long parameter lists and duplicated code (i.e., clones). Code smells are defined in terms of implementation-level constructs, such as methods, classes, parameters, and statements. Consequently, refactoring methods to correct code smells also operate at the implementation level (e.g., moving a method from one class to another, adding a new class, or altering the class inheritance hierarchy).

While detection and correction of code smells is one way to improve system maintainability, some maintainability issues originate from poor use of software
architecture-level abstractions — components, connectors, styles, and so on — rather than implementation constructs. In our previous work [8], we introduced the notion of architectural bad smells and identified four representative smells. Architectural bad smells are combinations of architectural constructs that induce reductions in system maintainability. Architectural smells are analogous to code smells because they both represent common “solutions” that are not necessarily faulty or errant, but still negatively impact software quality. In this paper, we expand upon the four smells identified in our previous work by describing them in detail and illustrating their occurrence in case studies from research literature and our own architectural recovery [4] [5] and industrial maintenance efforts.

The remainder of this paper is organized as follows. Section 2 explains the characteristics and significance of architectural smells. Section 3 summarizes research efforts in related topics. Section 4 introduces two long-term software maintenance efforts on industrial systems and case studies from research literature that we use to illustrate our four representative architectural smells. Section 5 describes our four architectural smells in detail, and illustrates the impact of each smell through concrete examples drawn from the systems mentioned in Section 4. Finally, Section 6 provides closing discussion and insights.

2 Definition

In this section, we define what constitutes an architectural smell and discuss the important properties of architectural smells.

We define a software system’s architecture as “the set of principal design decisions governing a system” [6]. The system stakeholders determine which aspects are deemed to be “principal.” In practice, this usually includes (but is not limited to) how the system is organized into subsystems and components, how functionality is allocated to components, and how components interact with each other and their execution environment.

The term architectural smell was originally used in [7]. The authors of [7] define an architectural smell as a bad smell, an indication of an underlying problem, that occurs at a higher level of a system’s granularity than a code smell. However, we found that this definition of architectural smell does not recognize that both code and architectural smells specifically affect lifecycle qualities, not just any system quality. Therefore, we define architectural smells as a commonly used architectural decision that negatively impacts system lifecycle qualities. Architectural smells may be caused by applying a design solution in an inappropriate context, mixing combinations of design abstractions that have undesirable emergent behaviors, or applying design abstractions at the wrong level of granularity. Architectural smells must affect lifecycle properties, such as understandability, testability, extensibility, and reusability, but they may also have harmful side effects on other quality properties like performance and reliability. Architectural smells are remedied by altering the internal structure of the system and the behaviors of internal system elements without changing the external behavior of the system. Besides defining architectural smells explicitly in terms of lifecycle