In this paper, it has been shown that how code smells could be mapped to maintainability factors and how it is possible to achieve comprehensive and accurate evaluations on the effect of each code smell and also to find those factors that are not reflected by code smells.

Prior to this research, others have shown the relation between individual code smells and different maintenance factors such as change size and defects. In this research, lacking of a good solution for general assessment, their motivation is to (1) extract important maintenance factors from project maintainer’s point of view; (2) find which code smell can assess those factors.

Their approach follows an industrial case study where six software engineer were hired to implement a number of changes over same set of systems. During 14 working days, they compare developer’s perception on the maintainability of systems and relate it back to code smells. Based on daily interviews and one wrap up interview, the analysis performed on cross-case synthesis technique for conducting perception comparison across sources.

They found thirteen maintainability factors by comparing expert assessment and developers’ impression. Nine of the factors confirm earlier findings and four new factors emerged from interviews. They associate each factor to the corresponding code smell. Some factors were mentioned more than by other developers; whereas some factors such as *standard naming conventions* and *comments* did not play an important role for developers. Conversely some developers unanimously mentioned factors such as *appropriate technical platform* and *simplicity* which might be an indication to give more attention for eliminating further problems.

They found that code smells can provide insight on different maintainability factors which can be improved using refactoring. On the other hand alternative approaches can be used to evaluate factor that are not reflected by code smells. By understanding the capability and limitations of different evaluation software maintainability, it’s possible to achieve better maintainability. According to previous research, they agree with statements that there is a need for combining different approaches in order to achieve more complete and accurate evaluation of maintainability.

This paper is taking advantage of recent advances and tools for identification of bad smells to explore the presence and evolution of bad smells by analyzing history of a project. They investigate whether the number of problems increase with the passage of software generations, whether problems vanish by time or only by user intervention or whether problems exist from the beginning of a project or occur in the course of evolution.

The number of studies have focused on the detections of changes and refactoring that have been applied in history of software projects acknowledging that historical data are valuable during maintenance. In contrast to previous works that mainly focused on identification of refactorings, they investigate the reasons causing the appearance of problems and removal during software evolution and also to find urgency of each type of bad smells.

In this work they investigate repository of two open source systems to reach the goal of paper. They employed JDeodorant which allows the identification of three non-trivial code smells, namely *Long Method*, *Feature Envy* and *State Checking*. They avoided looking at refactoring opportunities calling for refactorings with simple mechanics such as *Rename Method* and *Encapsulate Field*.

Functionality is enhanced in every new version of a software system and since open source software does not have systematic preventive maintenance, it’s reasonable to expect total number of design smells will increase with time. The number of *Long Method* smells is considerably larger indicating long and complex methods are more common than the other two symptoms. During evolution number of problems increases, although the rate of increase is lower for *Feature Envy* and *State Checking* smells.

In order to find urgency of a certain refactoring they employ past versions of the code smells with respect to the assumption the code fragments which have been subject to maintenance tasks in the past, are likely to undergo changes in the future versions. They coined the term “Active Smell” to refer to a problem that has been subject of maintenance at least once during its history. *Long Method*, *Feature Envy* and *State Checking* were three active code smells ordered by their prevalence. It means Long Methods code smell appear to be the most worrying one.