**2.1. Definition**

Deriving appropriate threshold values is a challenging open problem that has attracted the attention and effort of several researchers [20], [21], [22]. As a matter of fact, the approaches based on source code analysis suffer from high false positive rates [23] (i.e., they flag a large number of source code elements as problematic, while they are not perceived as such by the developers), because they rely only on the structure of the source code to detect code smells without taking into account the developers’ feedback, the project domain, and the context in which the code smells are detected.

However, relying solely on the developers’ comments to recover technical debt is not adequate, because developers might be unaware of the presence of some code smells in their project, or might not be very familiar with good design and coding practices (i.e., inadvertent debt). As a result, the detection of technical debt through source code comments can be only used as a complementary approach to existing code smell detectors based on source code analysis.

Despite the advantages of recovering technical debt from source code comments, the research in self-admitted technical debt, thus far, heavily relies on the manual inspection of code comments. The current-state-of-the art approach [14] uses 62 comment patterns (i.e., words and phrases) derived after the manual examination of more than 100K comments.

**2.2. Classification of Self-Admitted Technical Debt**

[Maldonado]

**Code debt**. “Problems found in the source code that can affect negatively the legibility of the code making it more difficult to be maintained”.

* Lower internal quality (unjustified issues): low readability, misuse of programming constructs, unnecessary code complexity…
  + A method that reads input string, writes split string and reads split string again;
  + A method that throws a wrong exception;
  + Multiple nested if-else blocks that may be simplified.
* Workaround (justified low quality code): compromise between code quality and specific software requirements.
  + Necessary “ugly” code for a specific unavailable feature, until some better function is developed;
  + Hacks for backward compatibility.

**Design debt.** “Debt that can be discovered by analysing the source code by identifying the use of practices which violated the principles of good OO-design”.

* Code smells: violation of OO design
  + Feature envy: a method that should be moved to a different class;
  + Code clones: duplicated block of code that may be reduced;
  + Lexical bad smells: poor lexicon that can lead to poor comprehensibility or increase software fault proneness;
  + Long method: method containing too many lines of code.
* Design patterns: need for introducing a design pattern

**Documentation debt:** “Problems found in software documentation that can identified by looking for missing, inadequate or incomplete documentation”.

* Inconsistent comments: misleading information, already addressed technical debts, comments still pointing to an issue that has been already closed and classified as “won’t fix.
* Licensing

**Defect debt:** “Known defects that should be fixed, but due to competing priorities and limited resources have to be deferred to a later time”.

* Defects: known issues to be solved, temporary patches implemented while waiting for an official fix.
* Low external quality: problems that might result in a code defect.
  + Missing thrown exception;
  + Missing input parameter control.

**Test debt.** “Issues which can affect the quality of testing activities”.

* Impossibility to reproduce bug behaviour;
* Failing assert statement to be checked;
* Low quality code in test suites.

**Requirement debt**. “Tradeoffs made with respect to what requirements the development team needs to implement or how to implement them”.

* Functional: existing features that need to be improved or new features to be implemented.
  + Comments reporting implementations that go against the requirement specification;
  + Comments referring to missing features;
  + Comments reporting doubts about the implementation of requirements.
* Non functional: performance issues.

**2.3. Previous Work**

[Russo & Bavota]