

1. Associate SATD with Bug-fix before SATD-fix
2. Associate SATD-fix with bug-fix. Find time window between SATD and SATD-Fix (count number of commits between)
3. If time window between SATD and SATD-fix is very large (| SATD ﹣ SATD-fix | >> 0) compute average of commit number (excluding outliers)
4. Associate SATD block to SATD-fix block

RQ: By fixing a SATD, does the code quality increases? ( # BugFix[satd] < # BugFix[satd-fix] )

RQ: Increase set size of SATD comment

RQ: Study evolution of a block of code including a SATD comment

* Analysis of 35 TODO comments from the Apache JMeter project repository [how were these selected exactly?] https://github.com/apache/jmeter/
* Built bash script to automate git command: git log -S’//TODO comment’ [—oneline] —file-path to retrieve the sha of the commits where the TODO was introduced and removed (not necessarily fixed)
* Realised that running the git log using just a portion of the comment (eg, without //TODO), some introduction or fixing shas are different: in some cases, the comment is also modified across different versions -> study of the evolution of the comment
  + Built script to compare shas using the whole comment vs a portion of the comment
  + Study of some interesting cases (4 out of 35) where the git log command returned more than 2 commits (i.e. not only the SATD introductory commit and the removing commit). For example, SATD #87 is introduced, then the method it refers to is marked as deprecated, and finally removed

- built java function that, from file path and comment text, returns the method containing the comment in the file (cases when the comment is inside a block or is external and refers to the block below)

**Java tool to parse code block from SATD comment**

We built a tool in Java that takes as input a SATD comment in the form “//TODO satd comment example” and the code of the class where the comment was found, and returns the code block that the comment refers to.

By analysing the set of SATD comments, we identified three recurring patterns defined as follows:

* Case A: the comment is contained in a code block and concerns one or more lines of code within a method (22 cases out of 35 comments).

[comment ID # 69]

public void clear() {

...

sequenceNumber=0; //TODO is this the right thing to do?

}

* Case B: the comment is outside a block refers to a method that is below the comment itself, recognizable from a method declaration (8 cases out of 35).

[comment ID #82]

//TODO - does not appear to be called directly

public static Vector getControllers(Properties properties) {

…

}

* Case C: the comment is again outside a block and refers to a single statement rather than a block of code, which usually lies immediately below the comment itself, or in some cases above the comment (4 cases out of 35). The statement may be a variable instantiation and is distinguishable from case B since it ends with a semicolon and does not have curly braces at the end of the line or in the line below.

[comment ID #100]

// TODO should the engine be static?

private static final JexlEngine jexl = new JexlEngine();

[comment ID #75]

/\*\*

\* Clear the TestElement of all data.

\*/

public void clear();

// TODO - yet another ambiguous name - does it need changing?

The tool addresses these three cases as follows:

* Case A: it searches for the SATD comment within the Java class. Then a parser analyses the code backwards until a method declaration is found, in the form (public|private|protected) [static] [final] return\_type method\_name(parameters) { ...

At this point, the parser moves forwards and keeps track of the curly braces until the count is greater or equal than 1 and the count of open braces is equal to the count of closed braces. The result is the code block that contains the comment.

* Case B: starting from the SATD comment line, the parser finds the first method declaration below (it may not be in the directly following line, if we deal with a multi-line comment). Then the parser proceeds like case A, by counting open and closed braces, and returns the complete block once the counters have the same value.
* Case C: starting from case B, if the parser finds a semicolon at the end of the end of the method declaration, it stops and returns it, as it is in fact an autonomous statement that does not precede a code block.   
  This technique will not work in the few cases where the comment is below the statement, as comment #75 shows. In these cases, however, developers clarified which statement the comment refers to by inserting separation blank lines between the statement itself and the surrounding ones. Therefore, the parser recognises these empty lines and is able to return the correct statement.

**Identification of bug reports related to SATD comments**

1. From the diff file of a Bug Report commit between SATD and SATD-fix, search for lines of code changed (starting with + or -) that are included in a SATD method block;
2. Within the diff file, search for calls of the SATD method related to lines of code that was changed in the BR commit.
3. Within the SATD-method, search for other method calls (also in different classes) that have changed lines in the BR commit.
4. Search for external variables, instantiated outside the SATD-method and user inside it, that have been changed in the BR commit.