Faculdade de Engenharia da Universidade Do Porto

Mestrado Integrado em Engenharia Informática e Computação

Engenharia de Software

T3 – Software Development in Practice - Jest

*Alunos:*

João Gama Amaral

João Nuno Ferreira

Pedro Galvão

 de Novembro, 2018

**Issue Documentation**

**Issue #5730**: Support dynamically detecting changes when *.gitignore* is updated

A certain repository has a *.gitignore* file in it. This type of file specifies intentionally untracked files that Git should ignore. Each line in a *.gitignore* identifies a pattern that must be disregarded.

If a *\*.js* is selected, then all changes in JavaScript (*.js*) files will not be taken into account. Besides this, Jest tests are coded in TypeScript files (*.ts*). The problem consists in the fact that when editing *.ts*, although Jest notices the file changes, it does not run the tests. But, when removing *\*.js* from the *.gitignore* the Jest runs those same tests.

**Requirements**

To solve this issue, you will need some basic knowledge in how the *.gitignore* files are supposed to work.

The case described on the issue documentation, implies that the users trying to fix the problem install and use either the suggested *Yarn* and *NPM* or any other dependency managers (for example, *IED*, *PNPM* or *NPMD*) that Jest can work with. These programs that are responsible for controlling the packages will be used to the user be able test.

**Example**

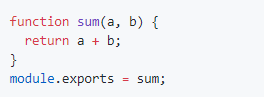
As said before, Jest is a robust framework, so it will be used along with *Yarn*/*NPM* for **package** and **dependency** management.

To test you must:

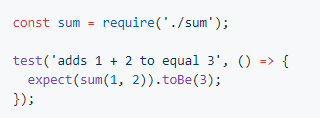
1. Install the *NPM*:

C:\Users\User\Desktop\download.png

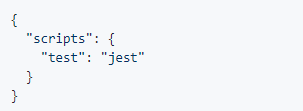
1. Create a *sum.js* file, with the following code:



1. Create a *sum.test.js* file, with the following code:



1. Modify the *package.json* file, changing the code inside the *“scripts”* curly brackets to *“"test": "jest"”*:



1. Create a file named *.gitignore*, that will be automatically hidden, with:



After the whole preparation, the user must keep running the *“tsc –watch”* and run *“jest –watch”*. Edit the *sum.test.js* and observe if the Jest notices the file change and if it runs any tests. Finally, the user should try removing the *“\*.js”*, and repeat the previous process. The results must be the same as the ones obtained by this GitHub user:

<https://user-images.githubusercontent.com/1370842/45393212-dec1e000-b665-11e8-988f-b8e7e1fea459.gif>

**Source Code Files**

Taking into account that the only files that are affected by this problem are the TypeScript tests, we can narrow them down to:

* *jest/e2e/coverage-remapping/\_\_tests\_\_/covered-test.ts*, which requires the file *jest/e2e/coverage-remapping/covered.ts*
* *jest/e2e/typescript-coverage/\_\_tests\_\_/covered-test.ts*, which requires the file *jest/e2e/ typescript-coverage /covered.ts*
* *jest/examples/typescript/\_\_tests\_\_/sum-test.ts*, which requires the files *jest/examples/typescript/sum.js* and *jest/examples/typescript/sum.ts*
* *jest/examples/typescript/\_\_tests\_\_/sub-test.ts*, which requires the same files in the previous point.

Finally, the *.gitignore* files are the ones that affect the *test.ts*, so we may also mention:

* *jest/.gitignore*
* *jest/e2e/coverage-report/.gitignore*
* *jest/examples/react-native/.gitignore*

The last two files may be ignored during the solving of this issue, since the solution for both of them must be exactly the same as the one in the first *.gitignore*, which is more general.

**System architecture Design of the fix**

**Issue Documentation**

**Issue #6830**: iterables and **toHaveBeenCalledWith** result in **RangeError**: Maximum call stack size exceeded.

If toHaveBeenCalledWith is used with an iterable, **RangeError: Maximum call stack size exceeded** will be thrown.

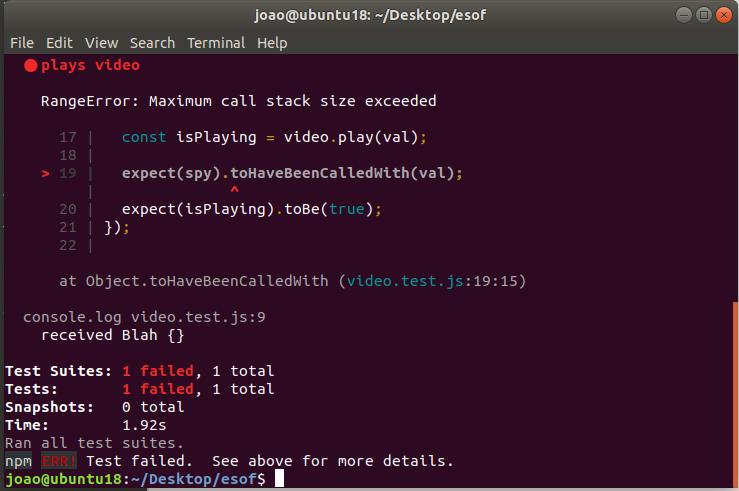
**Requirements**

To solve this issue, you will need at least some basic JavaScript knowledge.

Version in package.json: “jest”: “^23.5.0”,.

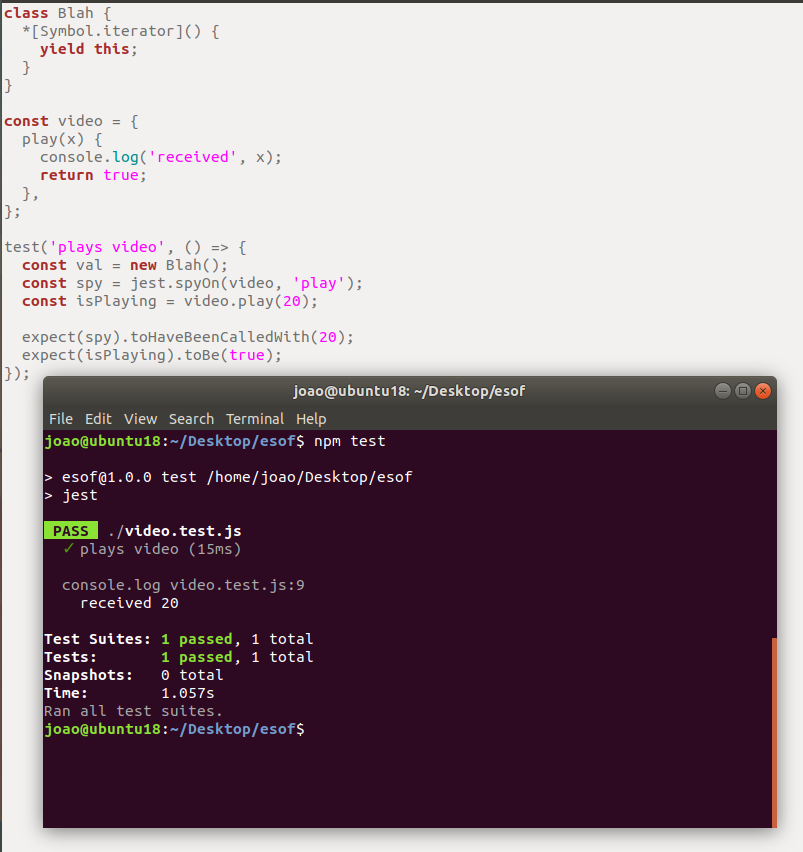
**To Reproduce**

Steps to reproduce the behavior, run: Resulting in:



The expected behavior is for the test to pass. To check it, we changed the values called in const isPlaying = video.play(val); and expect(spy).toHaveBeenCalledWith(val); The result was fine and the tests passed, so the problem has to be with the \*[Symbol.iterator]() somehow generating something too big, therefore, calling Maximum call stack size exceeded.

Symbol.iterator creates a costum iterator for Blah object. But written this way, the iterator never ends, so it is like a loop without an end-condition. **Yield.this** defines the value to return from the generator function via the iterator protocol, returning the optional value passed to the generator’s next() method.



With this information we can conclude that the problem must be directly related with toHaveBeenCalledWith. It might not be prepared to receive \*[Symbol.iterator](), assuming its an infinite value.

**Source Code Files**

Here are the files that are directly related to the issue:

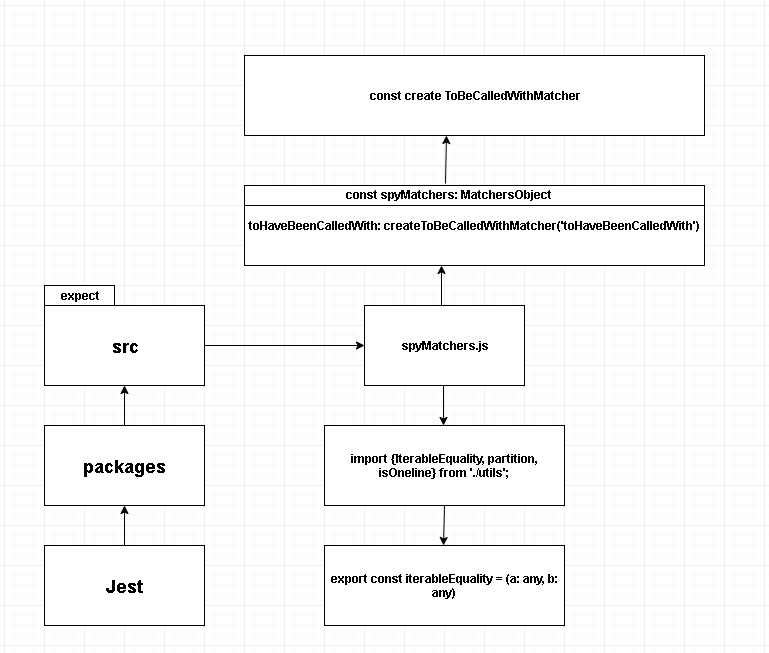
* jest/packages/expect/src/spyMatchers.js
* jest/packages/expect/src/utils.js

**System architecture**

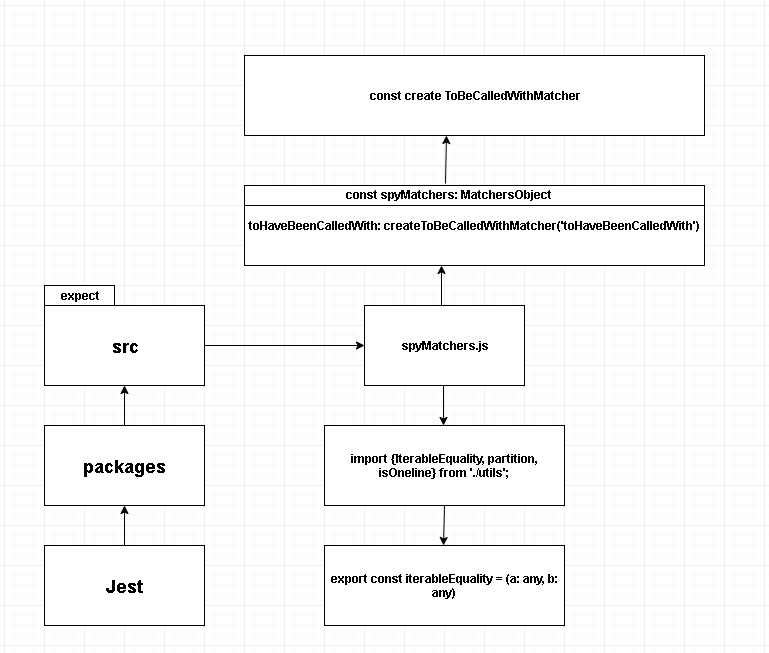
After searching on the repository, we could see that **toHaveBeenCalledWith** was declared by: **createToBeCalledWithMatcher**(‘.toHaveBeenCalledWith’). Even before this in the same file we can see: import {iterableEquality, partition, isOneline} from ‘./utils’;

On createToBeCalledWithMatcher there is a call to equals(call, expected, [iterableEquality]) which leads us to think that the issue must be corrected here or on the ‘./utils’.

**UML Diagram:**



**Design of the fix:**



In order to fix the issue there is no need to add classes, files or const to the repository, instead we will change directly the code of **const create ToBeCalledWithMatcher** or **export const iterableEquality = (a: any, b: any).**

The values might recurse forever here:

**jest/packages/expect/src/utils.js**

