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

15 de Novembro, 2018

Índice:

Issue #6830

* Documentation…………………………………………………………………...3
* Requirements…………………………………………………………………….3
* Source Code Files………………………………………………………………..4
* System Architecture……………………………………………………………..4
* Design of the fix…………………………………………………………………6

Issue #5730

* Documentation…………………………………………………………………...7
* Requirements…………………………………………………………………….7
* Source Code Files………………………………………………………………..8
* System Architecture……………………………………………………………..9
* Design of the fix…………………………………………………………………9

**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 use case -> user story -> feature description (se necessario…unico com codigo)**

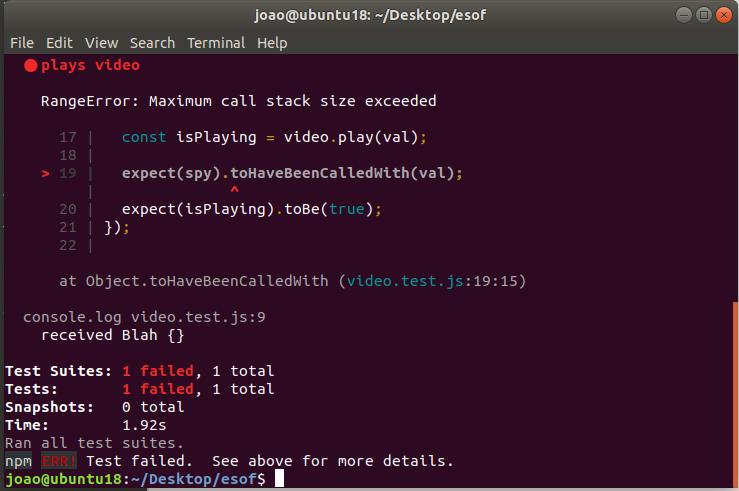
Jest is a desktop application that is used by Facebook to test all JavaScript code including React applications.

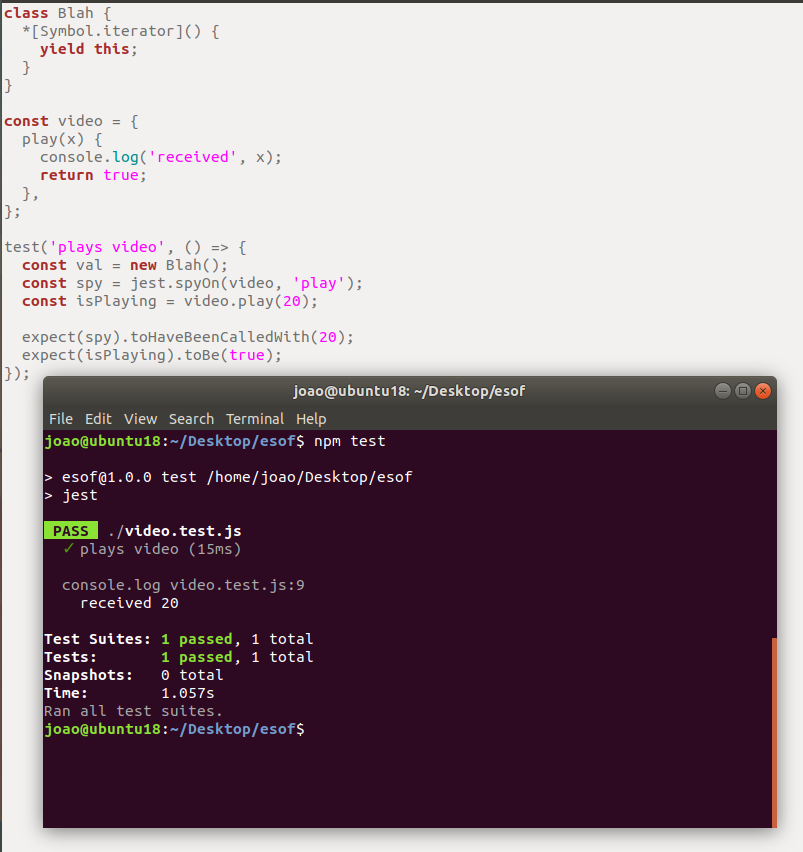
Version in package.json: “jest”: “^23.5.0”,.

**Use case:**

The function toHaveBeenCalledWith is supposed to verify if a given funtcion, specified by the user, was called with certain arguments, also specified by the user. However, when this function is used with an iterable that is too big or an iterator without end, the test fails and the error “Maximum call stack size exceded” is thrown. The expected behavior is for the test to pass. We can check it by declaring a variable with the specific iterator and then call functions defined in Jest to check it. Here is an example to reproduce the behavior:

First, a class that uses the iterator (\*[Symbol.iterator]()) is created. Then a function to be tested and the test itself are created. It can be checked that on the test function it is created a variable using the class Blah, this guarantees that the specific iterator is used. In the end the function toHaveBeenCalledWith is used, using as an argument the variable that was first declared. This should show successful tests but instead the tests fail.

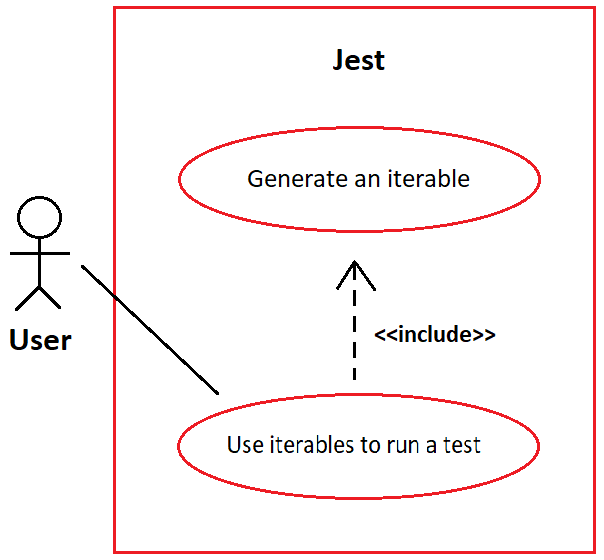
Results:

Somehow the class used is generating something too big, therefore, calling Maximum call stack size exceeded. In the reproduction of the error it is called a **console.log** to verify the value passed to the Jest test functions, and here it is printed “received Blah {}”, which is not an actual value.

In order to verify that the iterator was the actual cause of the issue, the value called on the test functions was changed to an integer. As expected, the tests succeeded. This leads to the conclusion that Jest does not have the proper code to receive and deal with this specific iterator.

**Explanation of the iterator:**

\*[Symbol.iterator] () creates a custom 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.

**Use case diagram:**

**User story:**

As a programmer for Jest I want an alternative for an iterable that is so big that it ends up calling the Maximum Stack Size Error, so that the test is able to pass.

**Source Code Files**

Files that are directly related to the issue:

1. jest/packages/expect/src/spyMatchers.js
2. jest/packages/expect/src/utils.js

The file number 1 is responsible for creating a “**spyMatcher**” that will test the function with the value that was passed to “toHaveBeenCalledWith”.

The file number 2 is used in the first file. The first file imports an “iterableEquality” from the second file. This “**iterableEquality**” is responsible for handling the values passed to the test functions like “**toHaveBeenCalledWith**”.

Files that are indirectly related to the issue:

1. jest/packages/expect/src/jasmineUtils.js

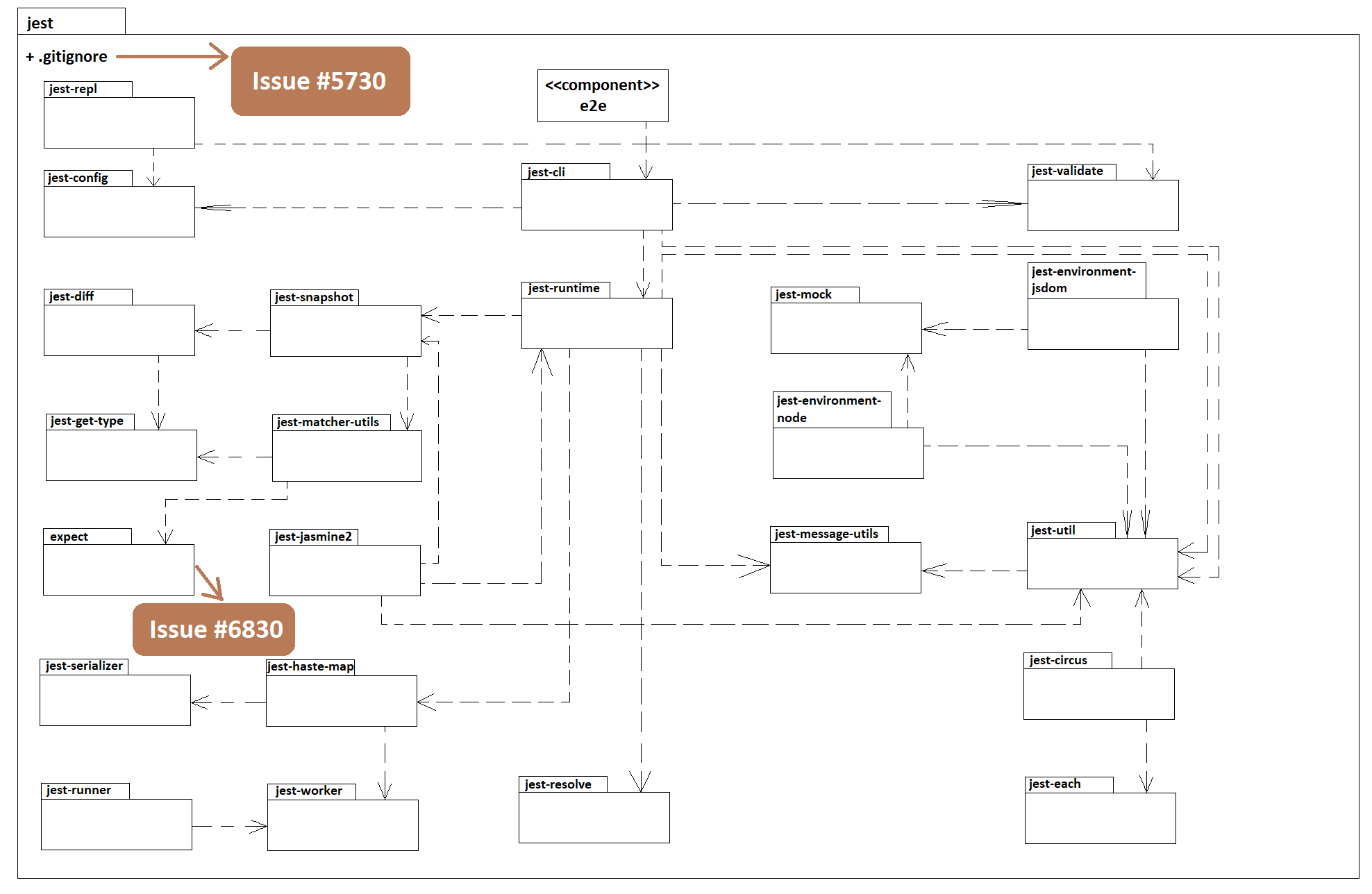
The file number 1 is indirectly related to the issue, that is because it contains a function “equals” that is used in “jest/packages/except/src/utils.js” to help handling the iterable.

**System architecture**

After a lot of research in this repository we were able to construct the system architecture to help localize the issues we are working on.

This issue in particular is related to the component marked as red **“expect”**.

**Package Diagram:**

****

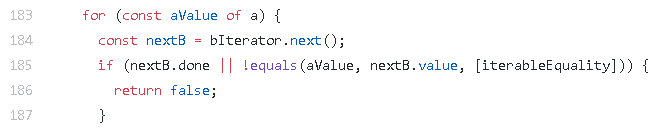
**Design of the fix:**

In order to fix the issue it is not necessary to add any aditional files or even classes.

We studied how the iterable was being handled by the function toHaveBeenCalledWith. As already explained, this function is supposed to verify if a given funtcion was called with certain arguments. To do it, the function uses the values returned by the spy object. From this object the arguments of each call of the function are identified and then each list of arguments is compared to the arguments specified by the user. If any of the calls used these same arguments, the test passes.

Each object to be compared is an iterable which contains the values of the arguments. The problem lays on the fact that the function that jest uses to compare those values, iterableEquality, is unable to handle iterables like the one described previously.

IterableEquality compares an iterable to another one by using a number of conditions, and if those conditions are insufficient to determine if the iterables are equal, the values returned by each respective iterator are compared one by one sequentially using the function equals, which calls, in this case, iterableEquality. Then the values of the iterable recurse forever in the following excerpt:

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

This recursion is useful for example when the arguments of the function include iterable objects, so we can’t simply eliminate it.

To fix it, it is necessary to change the cycle adding more conditions to stop it. In other words, the code of the defined function needs to be changed so that it can handle the iterables without creating an infinite loop.

(FALTA INSERIR UM UML NESTA SEÇÃO)

**Fix Source Code**

We decided to limit the cicle described above by adding a variable to count how many times the program passed by the instruction in the line 184. As we verified, a local variable is unable to count it because the infinite loop was not being caused by the “for” loop but by the call to iterableEquality inside this loop, so the local variable would be set to 0 again every time the function was called. Then we decided to use a global variable, initialized outside the function and being incremented just before the “if” condition shown in the previous section (in line 185 of jest/packages/expect/src/utils.js). We also added another condition before this one to test if the counter has reached certain limit, and if it reaches this limit the iterables are compared with “===” and the function returns the result.

This condition makes it possible to compare the iterables when they are too big or generate an infinite sequence of values, because we limit the number of comparisons.

Finally, we added instructions to set the counter variable to 0 before returning the result of the function. The objective of this is to assure that the counter will start from 0 if iterableEquality is called again later.

(COLOCAR IMAGENS DO CÓDIGO AQUI)

(COLOCAR TESTES E RESULTADOS AQUI)

**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**

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**

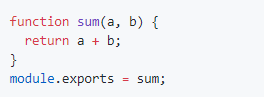
As said before, Jest is a robust framework, so it will be used along with *Yarn*/*NPM* for **package** and **dependency** management.

To test you must:

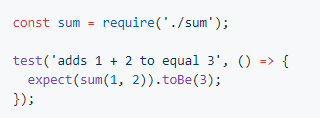
1. Install the *NPM*:

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

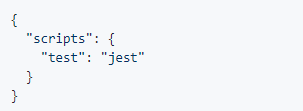
1. Create a *sum.js* file, with the following code:



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"”*:



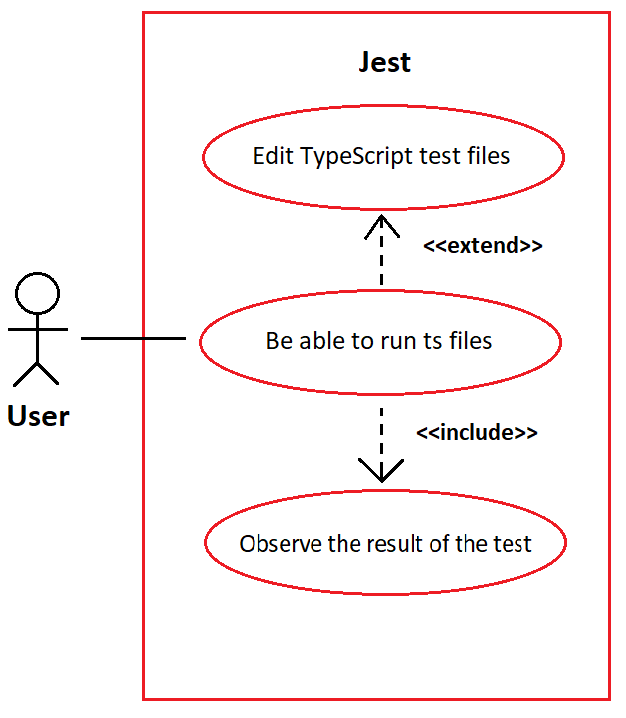
1. Create a file named *.gitignore*, that will be automatically hidden, with:



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>

**Use case:**

****

**User story:**

As a programmer for Jest I want to be able to test the TypeScript test files, so that Jest is able to run those same files in which some tests to the code are made.

**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:

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

Both files number 1 and 4 are responsible for testing whether the result of a difference between two numbers is right or not in TypeScript files.

The second file is similar to the previous one, but this time tests a sum, also in a TypeScript file.

Again, the third test is related to the test of sums with the difference that it tests in both JavaScript and TypeScript.

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

1. *jest/.gitignore*

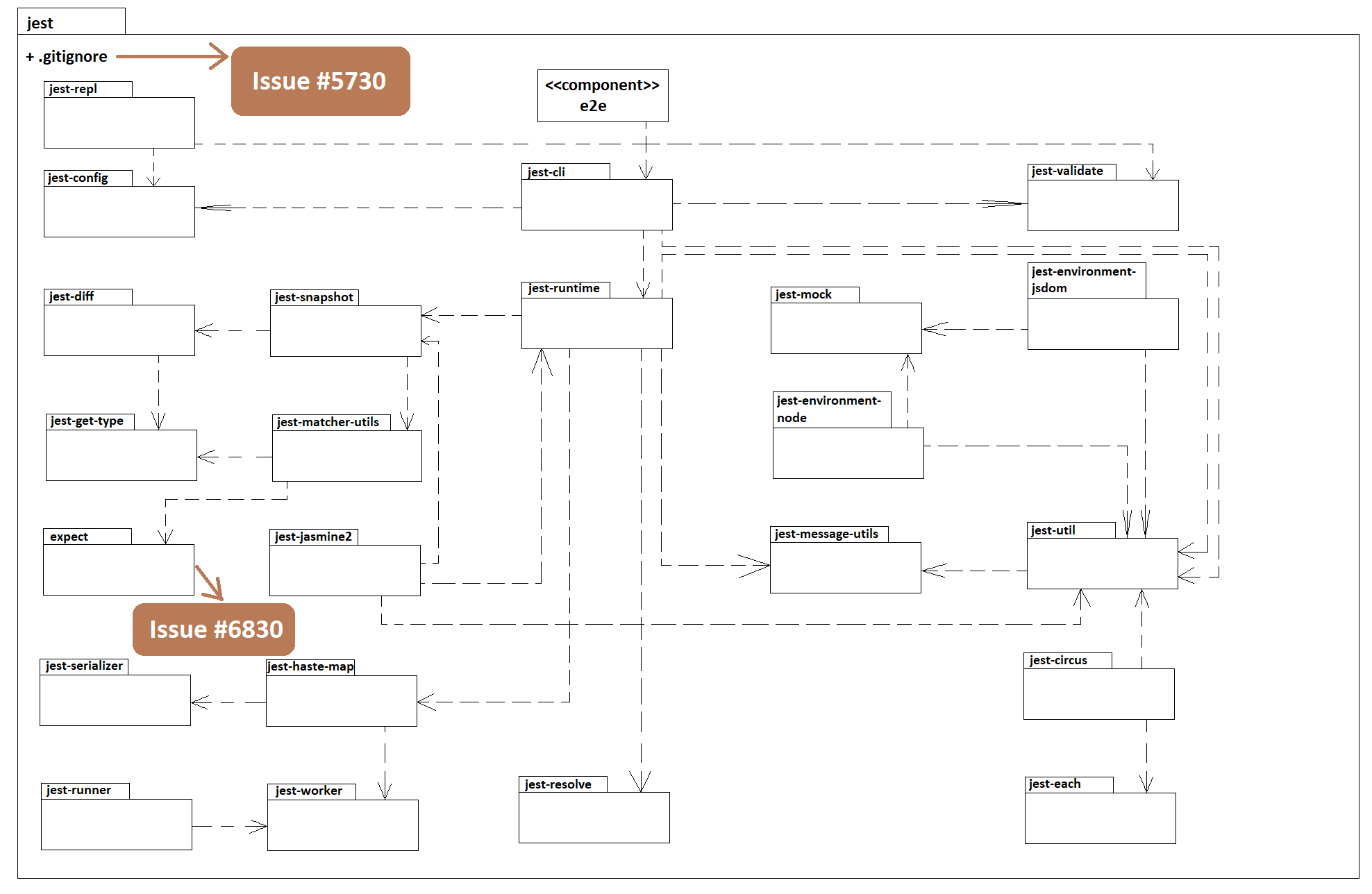
* *jest/e2e/coverage-report/.gitignore*
* *jest/examples/react-native/.gitignore*

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**

The system architecture is exactly the same as the previous issue, but this time this issue is related to the component marked as yellow.

**Package Diagram:**

****

**Design of the fix**

Just like the previous issue, we believe that the solution for this problem does not involve the creation of extra files or classes. In order to solve this particular issue, we must only change the *.gitignore* file. So the UML should remain the same as the one showed in the System Architecture point.

We have thought of changing this *.gitignore* in a way that keeps allowing Jest to ignore the changes in *.js* files, but not ignore the *.ts* or the *.test.ts* files.