Figure x

Figure x shows the approach variation. On the left side, we have the technique who considers only one level above the impacted class and on the right side we have the strategy, which considers several levels above the change. It walks up through the superior hierarchy of classes to reach the closest level to the interface as possible. The former is expected to be easier to evaluate, since it does not have to generate test for structurally complex objects, however, we do not know in which level the system camouflages a negative change such as a conditional structure makes a piece of code unreachable.

The second approach face the complex object problem aforementioned, but overcome the latter one.

Even though the second approach, face the complex object problem aforementioned we prefer to adopt it because

Cite that we can integrate new testing tools for our toolkit like

Cite and briefly describe it!

Shekoosh, Korat, etc …

Study these tools and explain how we could use to solve the problem.

However for a while, we are interested to statically evaluate this approach to take

New decision on the future ….

Pre-processor based system

*IC* outperforms *EIC* when the latter has to expose failures in structurally complex objects and *EIC* surpass the former in dead codes and masked-change situations.

Another challenge is the creation of complex objects to use as arguments in the method calls. Let’s suppose a class that implements an abstract data type or even a class that have so many object attributes and several dependencies. It is hard to exercise all of this objects and find a fault.

Just write!! Just write!!Just write!!Just write!! Just write!! Just write!!Just write!!

Just focus on writing!!!

First, explain the figure and each strategy

Then, advantages and drawbacks.