In practice, unit testing is hard and expensive to perform effectively because developer need to write driver/harness code to simulate the environment of isolated component. The paper DART: Directed Automated Random Testing proposed automatic random test generation method to reduce the labor cost of writing unit test and eventually improve software quality. DART uses static source-code parsing to automatically extract the interface of a program with its environment, then generate a test driver for the interface to perform random testing, finally use dynamic analysis to analyze program behaviors and automatically generate new test input to direct systematically the execution along alternative program paths. The author evaluate DART with three benchmark AC-controller, Needham-Schroeder Protocol and oSIP to compare direct search vs random search, measure effectiveness and scalability. As a result, DART is more effective than simple random search, effectively find assertion violation in Needham-Schroeder Protocol and scale well on large application oSIP with 30,000 lines of C code describing about 600 externally visible functions.

The main contribution of the paper are two points: 1. combine automatic random testing with automatic interface extraction, 2. use dynamic test generation to drive the program along alternative conditional branches to improve the effectiveness of finding errors. DART is able to dynamically learn the execution of the program in directed search. DART starts with a random input and a DART instrumental program will calculate an input vector during the execution. The input vector will force the next execution through a new path by having value that represents the solution of symbolic constrains which are gathered from predicated in branch during the execution. This dynamic input generation is much more effective than random testing since it attempts to force the program to run through all execution paths while the simple random testing are unlikely to discover all execution paths.

Though DART can generate input to crash one component but in reality the bugs that DART found were not necessary to be the vulnerability or bugs in the whole system. In evaluation, DART is tested against oSIP. The author claims that some oSIP functions do not have the check for NULL pointers so they will crash when it try to de-reference NULL pointers. However, in practice, the higher-level application that uses oSIP might have check for NULL pointers and never pass NULL pointers to oSIP API. Therefore, the bugs that DART found could be false positive when taking the whole system or software into account. This is also the limitation of unit testing, the implicit assumption for input of one specific component might exist in external component. By using DART in this situation will give numerous errors which might not worth taking a lot efforts to go through they and verify it is a bug that will threat the whole system. Furthermore, developers are often utilize the side-effects of certain functions which also might be reported as potential problems by DART.

A future work of DART can be automatically apply input constrains which are gathered from external component, developer documents, defined by the developer that could improve the accuracy for large scale software. By taking into account of external component which are calling the function to be tested, the potential problem that reported by DART could be more valuable for developer.