

SmartTC – A tool for optimizing and auto-generating test cases

Marzia Zaman, Alka Srivastava, Nita Goel
200-210 Colonnade Road
Ottawa, Ontario, CANADA, K2E 7L5

Email: marzia@cistel.com, asrivastava@cistel.com, nbansal@cistel.com

Kalai Kalaiichelvan
EION Inc.
945 Wellington Street
Ottawa, Ontario, Canada K1Y 2X5
Email: kalai@eion.com

Soumen Maity, Amiya Nayak
School of Information Technology and Engineering (SITE)
University of Ottawa, 800 King Edward Avenue
Ottawa, Ontario, CANADA, K1N 6N5
Email: maity@yahoo.co.in, nayak@uottawa.ca

Abstract. *Software testing is often performed in ad-hoc manner. A tester used his or her “gut feelings” to develop test-cases and execute them. Although this seems to be the most practical and effective method of testing, it is difficult to maintain the test-cases and/or automate them. Also, it is often impossible to test a software system exhaustively as the numbers test cases can grow astronomically even for a small number of input parameters and their possible values. This paper describes how SmartTC tool can be used to optimize the test cases in a systematic and effective way and how to generate the test cases automatically.*

1. Introduction

Software testing is an expensive, tedious and time-consuming activity in the entire software development life cycle. Studies [1] have shown that testing can take as much as 20% to 30% of total development budget of a software project. It can also span considerably to the overall length of the software development life-cycle. However, it is of critical importance to perform testing and detect and fix as many defects as possible before shipping the software to the customers. Improving test process can considerably reduce the testing cost.

Although there are many test tools available for developing test harness for automatically executing test cases, there are only few tools available in the market for optimizing and auto-generating the test cases [1,2]. These tools use the concept of design combinatorial theory to generate test cases such that the test set contains all possible pair-wise or higher order interactions. However, these tools are either quite expensive and/or difficult to use without enough consulting support. That again is quite discouraging especially for small to medium size company to use this type of tools. The other limitation of such tools lies in the fact that the optimization algorithm is inherently very complex and often uses many iterations to cover all the possible interactions. The speed of such tools in generating test cases is therefore an issue.

It has been reported that most of the field faults are occurred by interaction of one or two input fields [3]. SmartTC tool provides the pair-wise combination as the core algorithm for optimizing test cases. The tool uses computationally-efficient algorithm [4,5] to generate all possible pair-wise combinations of input values. The tool also includes features to include or exclude specific combinations that a tester may be interested in. This feature is extremely useful because while the pair-wise algorithm optimizes the test cases, the additional

include/exclude combinations make the test set more effective. Details on this feature will be discussed in later section.

The rest of paper is organized as follows: Section 2 presents an overview of the SmartTC tool covering the design objectives and main features of the tool. Section 3 demonstrates with examples the generation of test cases using SmartTC. Section 4 describes how this tool can be integrated in an end to end testing process. Section 5 provides a comparison study of SmartTC tool with a commercially available tool. Finally Section 6 discusses future work and the conclusion.

2. Features of SmartTC Tool

This section briefly describes the major features and the design objectives of the SmartTC tool. SmartTC is designed in object oriented fashion using Java as the programming language. The following attributes were considered while designing SmartTC:

Ease of use: This tool is intended to be used by software testers and in some cases by the designers. Usually both designers and testers deal with a number of tools in the software design, implementation and testing phases. Since this particular tool will introduce a new methodology in testing, the tool must be simple to use to attract user to get used to this methodology first.

Iterative/Incremental Design: The essential requirements are introduced in the initial release and once the user is comfortable using the tool and sees the benefits of using such a technique, more requirements will be captured and will be added incrementally in the tool. The architecture of the tool allows the incremental feature addition.

Input/Output: A simple text format for input file is chosen. The format also includes the predicate language to be used to define the “exclude” or “include” constraints. This gives the lots of flexibility to the user as well as scalability to the SmartTC designer. Outputs can be exported in different formats so that excel or other editor can be used to view and edit the output if necessary. The tool itself has its own simple editor to create the input file.

Portability: Java is chosen for implementing the SmartTC tool because the tool must be easily portable to different platforms as we foresee this tool being used by different design and test groups with various different platforms.

Performance: As mentioned earlier, the core pair-wise algorithm used in SmartTC is inherently computationally quite efficient. However, special attention is given to make sure the implementation is also done in an optimum fashion considering the performance of the test case generation.

Scalability: The tool must be able to handle multiple parameters of multiple values. The number of parameters and values chosen such that it can be used for any practical software system.

Cost-effective: The tool must be cost-effective.

The main features of the tool include ability to auto-generate test cases using default combination as well as the pair-wise combinations. The tool also allows user to input constraints on the top of the auto-generated combinations. The user of SmartTC can define the constraints in a SmartTC notation as will be discussed in details in next section to include or exclude specific combinations of interest.

SmartTC is designed with an aim to introduce the test generation and test optimization as an essential step in entire testing process. This step will force a tester to model the test inputs from the requirement specification in a formal syntax and generate the test cases in a systematic, efficient and effective manner. Developing a test plan would be much simpler if used this formal technique for generating test cases. The tester can gradually add the exclude/include constraints as the he/she becomes more familiar with the requirements and system under test.

3. Test Generation Using SmartTC

The step by step process for using SmartTC for generating test cases is illustrated in this section. Creating the input file for SmartTC requires understanding of the system under test and the requirements. The process is referred to as the data modeling [6,7]. This is a crucial step in using the test optimization and generation tool. The process of creating the input data model is described in details in the following section.

3.1. Input Data Model

This step also includes determining different possible scenarios for the system under test. Examples of such parameters are various configuration parameters, internal and external events, user inputs – basically parameters that will have impact on the system’s behaviour. Choosing interesting values for these parameters is also an extremely important step. For example, for an integer

parameter, one cannot possibly choose all or any randomly selected integer numbers. Depending on the system requirements, one must know what are the boundary values and/or other intermediate values that might be of interest. The concept of category partitioning can be used for data modeling [8]. For example, for testing an application which checks whether a given date is valid or not, one may wish to test the application with 29th February of a non-leap year. In this case, “non leap-year”, “February” and “29” are some of the interesting values for the input parameters year, month and date, respectively. This step although requires a thorough knowledge of the system, does not require each test case to be hand-crafted. The knowledge can be placed in a known notation from which test cases can be generated automatically using the tool. A sample data model is shown below in SmartTC syntax:

```
Day:15,29,30,31
Month:Jan,Feb,
Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec
Year:1999,2004
```

3.2. Constraints

As mentioned earlier, it is necessary to apply constraints on the test cases generated using pairwise combinations to include and/or exclude some specific combinations. This is important for several reasons, e.g.

- (i) some combinations may be already covered by other test cases,
- (ii) some combinations may need to be temporarily removed because it may block execution of other test cases,
- (iii) some combinations may not make sense at all as the design do not permit them,
- (iv) some specific 3-way or higher order interactions are necessary to include explicitly as they may not necessarily be covered in pair-wise test set.

For example, in previous example there is no point testing with day 29 for all months except February, because these numbers are as good as any other number (e.g. 15). The same argument is true for 30 for the months which end with 31 days. Also, note that in order to test the fact that 29th February is valid in the leap year whereas it is not valid in a non-leap year, one must enforce that specific test case. That can be done using the “include” constructs of SmartTC tool. Below, we present some constraints in SmartTC notation that are to be used to remove some redundant test cases and include some important test cases.

Rules:

```
EXCLUDE Month != Feb AND Day == 29
INCLUDE Month == Feb AND Year == 2004
AND Day == 29
INCLUDE Month == Feb AND Year == 1999
AND Day == 29
```

3.3. Pair Wise Test Generation

After the input file is created using SmartTC editor as shown in the screenshot below, the pair-wise test cases can be generated.

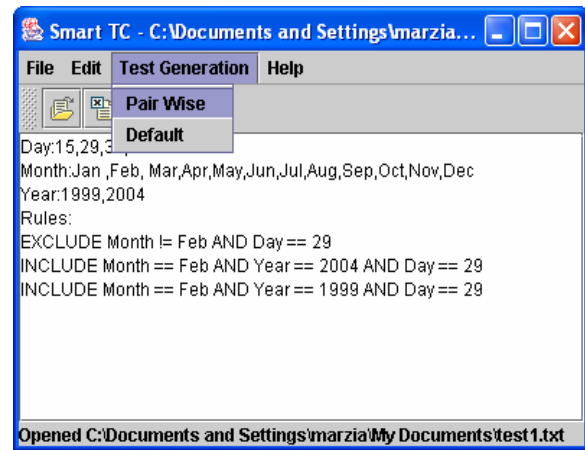


Figure 1: A sample model file in SmartTC

The test cases can be saved in different formats and that can be used later by the test harness for test automation or for manual testing. The number of generated test cases are 50 whereas the total number of exhaustive test cases are 96 – almost a 50% reduction. Note that, the gain in terms of number of test cases increases as the number of parameters increases. The following table shows the percentage gain in using the pair-wise algorithm when compared to the number of exhaustive test cases.

Table 1 Reduction in test set size using SmartTC’s pair-wise combinations

Number of parameters/values	All combinations	Pair wise combinations	% Reduction
4 parameters, 3 values each	81	11	86.41
6 parameters, 5 values each	15,625	25	99.84
8 parameters, 7 values each	5,764,801	49	99.99

3.4. Default Test Generation

Sometimes, it is important to test certain functionalities which do not involve interaction between different input parameter values. In the previous example, one may want to test the application using invalid day, invalid month or invalid year. In such cases, interaction between the parameters are not important and the testing can be done without doing pair-wise and/or any higher order interaction. To be honest, in this case, it is better to test each invalid value separately as testing them in combination may hide the real problem. In order to generate such combinations, “default generation” feature can be used. An example input model is shown below.

Month:Jan,Febb,13
Day:01,32,99,1,-1,First
Year:2004,20005,04,-04,TwoThousandFour

where, the first value in each line represents a valid value for that parameter. The first test case has all valid or default values. The idea is to generate the subsequent test cases including one invalid/non-default value at a time. The “default test generation” result is shown in the following table.

Table 2: Default Test Case Generation using SmartTC

Test Case	Month	Day	Year
1	Jan	1	2004
2	Febb	1	2004
3	13	1	2004
4	Jan	32	2004
5	Jan	99	2004
6	Jan	1	2004
7	Jan	-1	2004
8	Jan	First	2004
9	Jan	1	20005
10	Jan	1	4
11	Jan	1	-4
12	Jan	1	TwoThousandFour

4. Integrating SmartTC in Overall Testing Process

Test generation using SmartTC can be used at any level of testing i.e. unit, integration or system level testing. The tool can be integrated in any existing testing process. If test automation is desired, the SmartTC generated output can be fed to the test harness. The entire testing process which includes the optimal test set generation is summarized in the following figure. In the first step, the

data model is developed using the requirement specification or design documents depending on the level of testing. This may also involve discussion with designers to understand the test requirements. Once the data model is created, that can be fed to the SmartTC test generation tool to generate the optimal set of test cases. However, it is worth-mentioning that in order to make the test set more effective, domain knowledge is often desired. Pair-wise technique must be augmented with domain knowledge to generate more effective test set as shown in Figure 2. The output of the pair-wise technique can be used as input to the test harness. Specific customization may be required depending upon the test environment of specific project. Test automation or manual testing can be conducted using the test set generated by the pair-wise technique and domain knowledge.

Code coverage tool can be also used to measure the effectiveness of a test set. The goal is to exercise as much code as possible with minimum tests. The code-coverage results show what percentage of code has been exercised using the test set. If the code-coverage result is not satisfactory, one may need to refine the data model and repeat the entire process. It is always recommended to iterate the model few times [6,7] to get more effective test set.

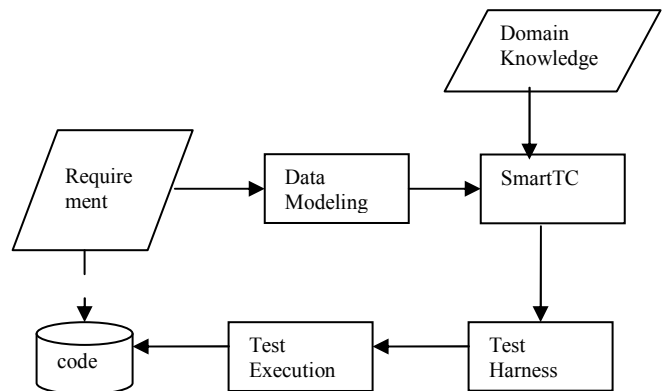


Figure 2: Testing process

Below illustrates an example where SmartTC can be used to generate test cases for installation testing. Imagine a web-based software system that works on different operating systems and on different web browsers. Other requirements involve installing the software for different packaging to enable/disable certain features, different modes of operation and different licensing models. The following table summarizes the different parameters and their possible values. The values are separated by commas under each parameter name.

Table 3: Input parameters and possible values

Platform	Web Browser	Feature	License	Operation Mode
HP, WIN	Internet Explorer, Netscape	Basic, Advanced	Fixed, Float	Local Remote

As can be seen from Table 3, there are $2^5 = 32$ possible combinations. Using SmartTC, the test set can be reduced to a size of 6. The output of SmartTC is shown in the following table.

Table 4: SmartTC generated output

	Platform	Web Browser	Feature	License	Op. Mode
1	HP	IE	Basic	Fixed	Remote
2	HP	Netscape	Advanced	Float	Local
3	WIN	IE	Advanced	Float	Local
4	WIN	Netscape	Basic	Float	Remote
5	WIN	Netscape	Advanced	Fixed	Remote
6	HP	IE	Basic	Fixed	Local

Assume there exist two different problems in the software which may be due to some defects in the code and/or packaging problem, (i) the system cannot be installed on an HP machine if remote mode of operation is chosen by the user and (ii) the system enables all features on Windows operating system even when the installation is done for “Basic” feature. Table 3 shows the output results. As can be seen, there are only 6 tests and out of that, tests 1 and 4 will be able to detect the above mentioned problems.

5. A Comparison Study

We conducted some preliminary comparison in terms of number of test cases (pair-wise combinations) generated by the tool and time required to generate the combinations with one of the commercially off the shelf tools named “ProTest” [2].

ProTest is a tool similar to SmartTC with a capability of generating pair-wise combinations. The tool also provides feature to include some simple type of constraints. The

pair-wise algorithm used in ProTest is iterative and the combinations are optimized in multiple iterations. The user can choose to stop at any point in time. For our comparison study, we have attempted to produce similar number of test cases as generated by SmartTC with minimum iterations.

Table 5 shows some sample results from our comparison study. Both tools were run on a Intel, Pentium III, 797 MHz, 192 RAM, Windows XP system on identical input.

As can be seen from Table 5, both the performance and the optimization in terms of number of test cases generated are better in SmartTC than that of ProTest. The performance of SmartTC is quite satisfactory. The speed is achieved because there is no iteration involved in generating the pair-wise combinations.

Table 5: Comparison Result

Input	Approximate Time (in sec)		Number of test-cases	
	Smart-TC	ProTest	Smart-TC	ProTest
6 parameters, 5 values each	1	22	25	36
24 parameters, 5 values each	2	500 ¹	49	65
6 parameters, 10 values each	1	108 ¹	120	129
24 parameters, 10 values each	4	2000 ²	219	238

¹ with 1000 iterations
² with 500 iterations

6. Future Work and Conclusions

SmartTC provides user a simple predicate language that can be used to define the constraints. Extensions are required to include more complicated constraints. Also, the constraint processing part can be further optimized. Although, currently only pair-wise combinations are included in the tool as the core algorithm for optimizing the test cases, efficient algorithms which would cover higher order interactions are also being investigated. SmartTC is currently being evaluated by some major software companies. Comparison of this tool with several other existing tools as well as considering the performance in presence of constraint is also underway.

The SmartTC tool can be a major motivation for using a systematic process while generating test cases. We have used the tool internally in various projects. The tool allows testers to focus on identifying test cases and

provides a formal mechanism to optimize and auto-generate them. This formal methodology can also save time and effort in developing test plan.

References

- [1] D.M. Cohen, S.R. Dalal, A. Kajla, and G.C. Patton, "The automatic efficient test generator", *Proc. IEEE Int. Symposium Software Reliability Engineering*, 1994.
- [2] <http://www.sigmazone.com/protest.htm>
- [3] D.M. Cohen, S.R. Dalal, M.L. Fredman, and G.C. Patton, "The Combinatorial Design Approach to Automatic Test Generation", *IEEE Software*, vol. 13, no. 5, 1996.
- [4] P. Agarwal, M. Zaman, A. Saxena, "Strategy for Generation of Pair-wise Test Patterns", Cistel Technical Report TR-CT-25, 2003.
- [5] S. Maity, A. Nayak, M. Zaman, A. Srivastava, and N. Bansal, "An Improved Test Generation Algorithm for Pair-wise Testing", *International Symposium on Software Reliability Engineering (ISSRE 2003)*, Colorado November 17-20, 2003.
- [6] K. Burr, W. Young, "Combinatorial test techniques: Table-based, test generation, and code coverage", In Proceedings of the International Conference on Software Testing, Analysis, and Review (STAR'98 West), Oct. 1998.
- [7] S.R. Dalal, A. Jain., N. Karunanithi, J.M. Leaton, C.M. Lott, G.C. Patton, B.M. Horowitz, "Model Based Testing in Practice", *Proc. 21st International Conference on Software Engineering*, pp. 285-294, 1999.
- [8] M. Zaman, A. Srivastava, N. Goel and V. Sehdev, "An Automated Test Generation and System Testing Strategy", Cistel Technical Report TR-CT-26, 2003.
- [9] T.J. Ostrand and M.J. Balcer, "The category-partition method for specifying and generating method for functional tests", *Communications of the ACM*, 31(6):676-686, June 1988.