CHAPTER - 7
Conclusions

In a hyper-competitive environment with constrained resources, the CDM (Combinatorial Design Method) provides an innovative approach for doing high-value testing with reduced costs and schedule. CDM provides good coverage of the various data paths in a small, commercial satellite ground system, with a limited number of test cases. The method is flexible, and does not require training in statistics to use or understand.

The RATG system uses new combinatorial design algorithms to generate test sets that efficiently cover the pair-wise or n-way combinations of a system's test parameters. Examples of such parameters are a system's configuration parameters, the parameters that define its environment, its inputs and internal events.

The basic RATG test requirement is that every pair-wise or n-way combination of parameter values is covered. In general, the number of tests required by the RATG method grows at the break points and remains same even the number of parameters are added until another break point commences. For example, checking all pair-wise combinations of 10 fields with 3 values each requires only 15 tests out of a potential one million test combinations ($3^{10}$).

Testers can use the RATG constructs to focus testing. The RATG constructs for relations, constraints and hierarchy allow testers to express knowledge about the system under test. The RATG test cases are far from random. In several experiments with code coverage, the RATG test sets gave significantly better coverage than randomly generated tests.
The RATG system is used in a variety of applications for unit, system, and interoperability testing. It has generated both high-level test plans and detailed test cases. Testers can base the RATG input on detailed development requirements or on a system's high-level functional requirements, such as its user manual. The experience with this new approach indicates that it is widely applicable and generates efficient test sets of good quality.

We believe our modeling and test-generation approach, satisfies the goal of usability by testers. In our experience, testers found the activity of specifying a software component's inputs to be natural and straightforward. By using the RATG software system, testers required minimal training to write their first data model and generate test tuples with pair wise combinations [GOPA 06]. As noted above, these tuples offer immediate value when used as test data sets (inputs for hand-crafted tests). Of course a significantly greater investment, mostly in software and script development, is required to develop the infrastructure such as an oracle that will allow the tests to be run wholly automatically.

Our Thesis work reports about systems that generate, document, execute, and evaluate thousands of test cases.

**Model of the test data is fundamental.** The model is comparable with an executable specification; like a specification, model development requires considerable domain expertise. For example, permissible data values and complex constraints among data values must be discovered and represented. Although a model-based test-generation system will require far more effort to develop than the model, development of the model should be allocated a significant portion of the up-front effort.
Model-based testing is a development project. The development, application, and ongoing maintenance of a test automation system requires expertise from software developers and professional testers. This mix of skill sets is difficult to find in either a development or a testing organization.

The document reports on experiments measuring the code coverage of RATG tests. We considered two different application domains, screen validation software and simple UNIX commands.

The RATG pair wise tests gave block coverage percentages of 85% and above for both application domains. The average coverage for the UNIX commands was 97%. Increasing the RATG coverage requirements from pair wise testing to triple testing did not appear to be worthwhile. The triple RATG tests sets gave the same coverage as the pair wise test sets but required substantially more test cases.

We tried several different RATG input models for the UNIX sort command. We found that lowering the height of the model greatly reduced the number of test cases, without sacrificing code coverage. Radical reductions in height can reduce the number of tests while maintaining a high level of code coverage.

We also experimented with two other test strategies, random testing and default testing. Default testing was equivalent to pair wise for screen testing (92% block coverage for both), but did poorly for testing the UNIX commands (e.g. 86% verses 95% for sort). Random testing did poorly for screen testing (67% verses 92%).
The RATG test sets found many discrepancies between the code and the requirements. These included "missing code" faults usually not detectable by code coverage alone. The people participating in the study reported that the RATG paradigm would cause developers, multi-unit testers, and system testers to view their work from a new perspective and help them to spot additional defects.

**Future Work**

Further research is focused to obtain a measure of goodness for the CDM strategy. The following issues are identified for prospective research in this area.

- When exhaustive testing is not performed additional risk that is incurred may be estimated.
- The level of certainty that the CDM strategy provides for discovering latent bugs may be researched into.
- The possibility of generated test cases covering the entire scope of test domain may be found.
- The conditions under which the CDM strategy is not suitable may be identified for robust design of experiments (like Taguchi techniques) [TAGU 86]. It would also be useful to develop case studies that compare the quantity and severity of Customer reported problems (post-delivery) for systems using CDM with systems using exhaustive or quasi-exhaustive testing. Our work has used CDM for the ground system described earlier, and has employed a CDM-like strategy (based on orthogonal arrays/Taguchi techniques) on two other ground systems. However, no valid, historical comparison data could be derived for these ground

99
systems. A hyper-competitive business market will not easily allow such comparisons.

Now we present a few directions for further enhancements of our work. The challenges that have to be encountered and benefits derived when model based testing is applied to different phases of testing may be a good research topic. The applicability of the same modeling language for different phases of testing could be another interesting topic.

Our future work will also explore combining behavioral models (covering possible paths) with input models (covering triple wise combinations of inputs) to reach new heights in test effectiveness and input domain coverage.