Abstract – Reactor Protection System (RPS) of High Temperature Gas-Cooled Reactor - Pebble bed Module (HTR-PM) is the first digital RPS designed and to be operated in the Nuclear Power Plant (NPP) of China, and its development process has received a lot of concerns around the world. As a 1E-level safety system, the RPS has to be designed and developed following a series of nuclear laws and technical disciplines including software verification and validation (software V&V). Software V&V process demonstrates whether all stages during the software development are performed correctly, completely, accurately, and consistently, and the results of each stage are testable. Software testing is one of the most significant and time-consuming efforts during software V&V.

In this paper, we give a comprehensive introduction to the software unit testing during the development of RPS in HTR-PM. We first introduce the objective of the testing for our project in the aspects of static testing, black-box testing, and white-box testing. Then the testing techniques, including static testing and dynamic testing, are explained, and the testing strategy we employed is also introduced. We then introduce the principles of three kinds of coverage criteria we used including statement coverage, branch coverage, and the modified condition/decision coverage. As a 1E-level safety software, testing coverage needs to be up to 100% mandated. Then we talk the details of safety software testing during software development in HTR-PM, including the organization, methods and tools, testing stages, and testing report. The test result and experiences are shared and finally we draw a conclusion for the unit testing process. The introduction of this paper can contribute to improve the process of unit testing and software development for other digital instrumentation and control systems in NPPs.

Keywords –HTR-PM, digital reactor protection system, software unit testing, software verification and validation
one of the most rigorous requirements is the software verification and validation (software V&V). The software V&V process demonstrates whether all stages during the software development are performed correctly, completely, accurately, and consistently, and the results of each stage are testable [3]. Software testing is one of the most significant and time-consuming effort in software V&V.

Structure and design principle of RPS software in HTR-PM is different for conventional NPPs [4]. Most digital RPS available in NPPs base on a safety control-system platform, and the RPS software is configured based on this platform according to the software specification. While the RPS of HTR-PM is a custom-made system and applies only to HTR-PM and its following projects. Therefore the software of RPS in HTR-PM is specifically developed and there is no need for platform software, function database, configuration software, etc., which decreases the amount of codes and simplifies the unit testing process greatly.

In this paper, we will give an overview to the unit testing process of RPS software in HTR-PM. We first introduce the objective of unit testing in Section II. The testing techniques including static testing and dynamic testing are given in Section III. Principles of testing coverage are also introduced in this section. In Section IV, the testing practice, such as testing organization, methods and tools, testing stages, and testing reports are proposed. The results and experiences we found during this process are summarized in Section V. Finally we draw a conclusions in Section VI.

II. TESTING OBJECTIVES

According to the plan of software V&V, the unit testing is accomplished during the software implementation process. Fig. 1 illustrates the software development lifecycle of the RPS in HTR-PM, and the software unit testing corresponds to the step “V5”. So the testing objective is to verify whether the output of the software implementation stage, i.e. the C codes, has reflected the expectation of the software design, and the process from software design to software implementation is correct, accurate, and complete. This objective can be achieved by the black-box testing of the dynamic testing.

To ensure that the software achieves high quality which is needed by related standards and regulation guidelines, the coding process has to be implemented with high dependability. There are four characteristics for the coding process: reliability, robustness, traceability, and maintainability. We proposed detailed criterions for the software coding, so the objective for the static testing is that the code shall satisfy the coding criterions proposed by the software V&V group.

The third objective is the testing coverage should be up to 100% to ensure the testing completeness and this is the objective for the white-box testing. Details of coverage will be introduced in Section III.

III. TESTING TECHNIQUES

Software unit testing has been developed to be a mature engineering technique since 1950s. In this section, we give a brief introduction to the unit testing techniques we employed during the software V&V process.

III.A. Static Testing

Static testing is a software testing method that involves examination of the program's code but does not require the program to be executed [5]. The static testing can be employed to achieve the second testing objective, i.e. to verify whether the code satisfies the coding criterion.

Static testing may be conducted manually or through the use of various software testing tools. The specific types of static software testing include code analysis, inspection, code review, and walkthrough. We employed code analysis and walkthrough during the software testing process.

Code analysis is employed to fulfill the objective for coding criterion verification. Code analysis is to analyze the code against the chosen coding standard, or the coding criterion in our project to indicate the bugs or unsatisfied parts. Code analysis can prevent bugs and identify coding issues extremely early in the development or unit testing cycle, passing better code into the latter stage of the development or software V&V process.

Walkthrough is a form of software peer review “in which a designer or programmer leads members of the development team and other interested parties through a software product, and the participants ask questions and make comments about possible anomalies, violation of development standards, and other problems” [6]. In other words, walkthrough can be regarded as an execution of the code by testers manually. It can be employed when the code cannot be dealt with the automatic testing tools.

III.B. Dynamic Testing - Black-box

Dynamic testing refers to the testing in which the software must be compiled and run actually. It examines the physical response of the system. It involves working with the software, giving input values and checking if the output is as expected by executing specific test cases which can be done with the help of automatic testing tools. The dynamic
testing can be classified into black-box testing and white-box testing according to whether the code under testing is known.

Black-box testing treats the software as a “black box”, examining functionality without any knowledge of internal structure and implementation. The testers are only aware of what the software is supposed to do, rather than how it does it. The input of black-box testing is the software requirement or specification that the development group uses to code the software.

The basic principle of black-box testing can be expressed in a number of different ways [7]:

- Test against the specification
- Use test coverage criteria based on the specification
- Develop test cases derived from the specification
- “Exercise” the specification

Black-box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, state transition tables, decision table testing, fuzz testing, model-based testing, use case testing, exploratory testing, specification-based testing, etc.

It should be noted that it is difficult to automatically measure the degree of coverage of the specification that black-box testing has achieved. Correct implementation of black-box tests therefore relies heavily on the quality of the tester’s work.

III.C. Dynamic Testing - White-box

White-box testing is the testing that takes into account the internal mechanism of a system or component. The name “white box” appropriately indicate that the tester has full visibility of the internal workings of the software product, specifically, the logic and the structure of the code.

In contrast with the black-box testing, the basic principle of white-box testing can be expressed in a number of different ways:

- Test against the implementation
- Use test coverage criteria based on the implementation
- Develop test cases derived from the implementation
- “Exercise” the implementation
- “Exercise” the specification

We mainly use black-box and white-box methods during the unit testing for the RPS software in HTR-PM, while the white-box testing can also be applied at the integration and system level of the software testing process and the application it is used for integration and system testing turns more frequent today. It can test paths within a unit, paths between units during integration, and between subsystems during a system–level test. This method of testing can uncover many errors or problems, and it has the potential to miss unimplemented parts of the specification or missing requirements.

III.D. Code Coverage of White-box Testing

Code coverage is a metric used to describe the completeness to which the source code of a program is tested by a particular test suite, which can be used to identify the test effect of white-box testing. Software with high code coverage illustrates that it has been thoroughly tested and has a low probability of failure.

Many different metrics can be used to calculate code coverage, such as statement coverage, branch coverage, condition coverage, modified condition/decision coverage (MC/DC), etc.

Statement coverage identifies the percentage of statement in the software that has been executed. Branch coverage, also called decision coverage, identifies the percentage of branch of all control structures (such as in “if” statement) that has been executed. Condition coverage identifies whether each Boolean sub-expression has been evaluated both to true and false. Modified condition/decision coverage requires that both decision and condition coverage been satisfied, and each condition should affect the decision outcome independently.

III.E. Testing Strategy

We follow the testing strategy proposed by G.J. Myers in book [8] during testing process, which is listed as followed:

1. If the specification contains combinations of input conditions, start with cause-effect graphing.
2. In any event, use boundary-value analysis. Remember that this is an analysis of input and output boundaries. The boundary-value analysis yields a set of supplemental test conditions, but, as noted in the section on cause-effect graphing, many or all of these can be incorporated into the cause-effect tests.
3. Identify the valid and invalid equivalence classes for the input and output, and supplement the test cases identified above if necessary.
4. Use the error-guessing technique to add additional test cases.
5. Examine the program’s logic with regard to the set of test cases. Use the decision-coverage, condition-coverage, decision/condition-coverage, or multiple-condition-coverage criterion (the last being the most complete). If the coverage criterion has not been met by the test cases identified in the prior four steps, and if meeting the criterion is not impossible (i.e., certain combinations of conditions may be impossible to create because of the nature of the program), add sufficient test cases to cause the criterion to be satisfied.
IV. TESTING PRACTICE

We will introduce the testing practice during the software unit testing of RPS in HTR-PM. This section mainly focuses on the testing organization, testing methods and tools, testing stages, and the testing report.

IV.A. Organization

The design, development (coding), and V&V of software are implemented by different groups in different organizations. The software design of RPS in HTR-PM is taken charge of by the design group in the Institute of Nuclear and New Energy Technology of Tsinghua University (INET). The safety software of RPS is developed independently by the development group in China Techenergy Co., LTD (CTEC). The software V&V group in INET is responsible for the software V&V work, which is organizationally independent with the development group in technical, management, and financial aspects.

The testing team in the software V&V group of INET is composed of 6 persons. There is one team leader and 5 members in this team. The responsibility of testing team leader and testing members are listed in Table 1.

<table>
<thead>
<tr>
<th>Role</th>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testing team leader</td>
<td>• Manage and guide members to work</td>
</tr>
<tr>
<td></td>
<td>• Oversee implementation of the testing plan; control the progress of the testing work</td>
</tr>
<tr>
<td></td>
<td>• Coordinate with development group and other team</td>
</tr>
<tr>
<td></td>
<td>• Responsible for reporting to the project management organization of related work</td>
</tr>
<tr>
<td>Testing member</td>
<td>• Perform testing tasks according to the plan</td>
</tr>
<tr>
<td></td>
<td>• Responsible for the preparation of test plans and test cases</td>
</tr>
<tr>
<td></td>
<td>• Responsible for unit testing</td>
</tr>
<tr>
<td></td>
<td>• Generate various stages of mission reports and other reports</td>
</tr>
<tr>
<td></td>
<td>• Report to the testing team leader of the related work</td>
</tr>
</tbody>
</table>

Table 1: Responsibility of the roles in testing team.

IV.B. Methods and Tools

We employed two kinds of automatically testing tools for the static testing and dynamic testing correspondingly.

We used an analysis tool of C code, QA C, for the static testing of PRS software. QA C can analyze whether the C code satisfies with the criterion. The testing criterion was made based on the coding criterion. Some coding rule sets and coding standards, including MISRA C, were also taken as a reference for the criterion. QA C can indicate the faults of the software intuitively and give a detailed analysis report, which is very convenient for the documentation process.

We adopted VectorCAST as the automatic unit-testing tool for black-box and white-box testing. VectorCAST can construct complete test-harness for the unit under test and build stub and driver code automatically. After execution of test cases, it provide clear testing coverage result, which is convenient for the white-box testing. It should be noted that one of the most time-consuming effort for dynamic testing is report arrangement. VectorCAST offers a series of reports which makes the documentation to be much easier.

The static testing is based on the coding criterion of RPS in HTR-PM and realized by the testing tool QA C; we used the strategy in Section III.E for dynamic testing when designing the testing case and the test was implemented by the testing tool VectorCAST. Although the tools we employed were powerful for static and dynamic testing, there were still some cases that the codes cannot be tested with simulated environment. For these cases, we employed the walkthrough method to review and analyze the code by the tester.

Testing coverage criterion we employed was statement coverage, decision coverage, and MC/DC. All coverages were expected to be up to 100% to insure the completeness and the reliability of testing.

IV.C. Testing stages

The safety software of RPS are composed of two parts: service software and application software. Service software refers to the codes that can be shared by different devices, such as codes for signal I/O and communication. Application software refers to software specifically for a single device, such as software for signal processing device and software for logic gating device.

The unit testing of RPS software in HTR-PM was divided into two stages: testing for service software and testing for application software. Stages for either type of software were similar. The static testing was first carried out. Then the problems detected against the coding criterion were reported by the testing team of V&V group to the development group for bug correction. After debugging and testing by development group, the software edition was updated and transmitted to the V&V group for regression. If there is no faults for this edition, the software was then passed on to the dynamic testing.
The dynamic testing was carried out strictly according to the strategies in Section III.E proposed by G.J. Myers. The software specification was used for the black-box testing to generate test cases. If the statement coverage, decision coverage, and MC/DC were not up to 100%, more test cases were supplemented through the assistant of testing tool. The faults found in this stage also needed to be reported to the development group and after a higher edition of software was released, a regression had to be implemented by the V&V group to verify whether the faults had been removed and no new faults were led in during this process.

There had always been some codes that cannot be tested by the dynamic testing tools, such as the value assignment of parameter with real physical address. We employed the walkthrough method for these kinds of situations.

IV.D. Testing report

The faults found during static and dynamic testing were classified into four severities: critical, general, minor, and slight. The definition for each severity level are listed in Table 2. Each fault detected during the static and dynamic testing was classified according to these definitions to remind the development group for correction.

<table>
<thead>
<tr>
<th>Severity</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>critical</td>
<td>• seriously affect the realization of the basic functions of the system requirements or function cannot be achieved</td>
</tr>
<tr>
<td>general</td>
<td>• partly affect the system requirements or the achievement of basic functions</td>
</tr>
<tr>
<td>minor</td>
<td>• slightly affect the system requirements or the achievement of basic function</td>
</tr>
<tr>
<td>slight</td>
<td>• do not affect the basic system requirements or the achievement of basic functions, but bring inconvenience to the operator, or make the system unprofessional, affect the overall impression of the system</td>
</tr>
</tbody>
</table>

Table 2: Definitions of four fault severities.

We made three types of testing reports according to the unit testing stages for every C file during dynamic testing: test case design report, testing execution report, and testing result report.

The test case design report records the structure information and detailed data of test case. The structure information includes the unit (i.e. the C file) name of software, the name of function, the name of all test cases for this unit, and the creation and execution date of test case. The test case data includes the input and the expected output of test case, and the user code that the tester adds.

The testing execution report includes the structure information, the configuration, and the execution result of the test case for each step. The test result report includes the structure, the overall result, the aggregated coverage, the MC/DC condition tables, and the metrics. The overall result are composed of the pass ratio of test cases and the expected values, the coverage of statement, MC/DC, and MC/DC pairs. The aggregated coverage shows intuitively three kinds of coverages we employed. And the MC/DC condition tables list the MC/DC possibilities for every control structure. Finally the metric shows the complexity and the detailed coverage data of every function.

We also proposed a standard format for the abnormal report. The abnormal report is composed of three parts. The first part is filled in by the V&V group and mainly includes the object of V&V, the position of the problem, the detailed description, and the severity of the problem. The second part is filled in by the development group and is composed of the analysis of cause, the processing scheme, and the analysis for the influence of the problem. Finally the V&V group verifies the processing result and makes a decision whether the problem should be closed.

V. TESTING RESULT

We spent one year on the unit testing of RPS software in HTR-PM. The stages included two rounds of static testing and dynamic testing. There are totally around 300 faults found in dynamic testing and all have been corrected during regression testing.

The testing experiences are listed as followed:

1. Quality of software specification plays a key role for the black-box testing. If the significant parameters, for example, the set-point of comparison or the value of time delay, are not declared clearly in the specification, the black-box testing cannot be advanced smoothly.

2. Another problem for software specification is that spurious codes instead of the flow chart are employed in the specification. As the spurious codes have similar contents with the real codes, it’s difficult to identify the validity of codes. A detailed flow chart is an important guarantee for the testing quality.

3. The complexity of C codes should be limited in order to insure the testability of dynamic testing. As the coverage is a criterion for the testing completeness, it’s rather difficult to
achieve the coverage object if the C code is developed with high complexity.

4 Another issue should be noted is the cooperation between the testing group and the development group. The testing group should be regarded as the helper for the quality improvement of the software rather than a fastidious troublemaker. There should be frequent communications between these two groups in order to avoid unnecessary quarrel.

VI. CONCLUSION

Safety has turned to be one of the most significant issues for the NPPs. The RPS of HTR-PM is the first digital RPS designed and to be operated in the NPP of China. Rigorous software V&V process was taken to ensure the reliability and safety of the software, and software unit testing is an important stage during this process.

In this paper we summarize the unit testing process of the RPS software in HTR-PM, including the testing objective, testing technology, testing practice, and testing result. We also propose the experiences and advices for the unit testing.

Software testing plays an important role in improving the software quality and should receive special attention. The introduction of this paper can contribute to improve the process of unit testing and software development for other digital instrumentation and control systems in NPPs.

ACKNOWLEDGMENT

This work was supported by the National Science and Technology Major Project of China under project ZX06901.

REFERENCES

Fig. 1: Life cycle model for the development and software V&V process of RPS in HTR-PM