Systematic Verification of Operational Flight Program through Reverse Engineering

Dong-Ah Lee, Jong-Hoon Lee, Junbeom Yoo, and Doo-Hyun Kim

College of Information and Communication, KONKUK UNIVERSITY
New Millennium Hall, 1 Hwayang-dong, Gwangjin-gu, Seoul, 143-701, Korea
{ldalove,kirdess,jbyoo,doohyun}@konkuk.ac.kr

Abstract. Software reverse engineering is an engineering process analyzing a system for specific purposes such as identifying interrelationship between system components or reorganizing the system structure. The HELISCOPE project aims to develop an unmanned helicopter and its on-flight embedded computing system for navigation and real-time transmission of motion video using wireless communication schemes. The OFP (Operational Flight Program) in HELISCOPE project keeps only informal and non-standardized documents and has made us difficult to analyze and test it thoroughly. This paper introduces a verification plan through reverse engineering to get over the difficulties, and we share an experimentation about a small portion of the plan to the HELISCOPE OFP.

Keywords: Operational Flight Program, Verification, Reverse Engineering, Testing.

1 Introduction

HELISCOPE [1] project aims to develop on-flight computing system, embedded S/W, and related services for unmanned helicopter that shall be used for disaster response or recovery using real-time transmission of the motion video through wireless communication scheme. OFP (Operational Flight Program) of the HELISCOPE project [2] is a control program which provides real-time controls with various sensors and actuators equipped in the helicopter.

The OFP as a safety-critical and mission-critical system should be sufficiently verified through application of various validation and verification techniques. For instance, formal verification technique [3] plays an important role in demonstrating safety and correctness of the system. Our previous work we used two formal verification techniques to verify process communications and timing constraints of the OFP [4][5]. Testing is also one of widely used technique to verify structure or functionality of software. For applying testing techniques to a target system, well-formed specifications such as SRS (Software Requirement Specification) or SDD (Software Design Description) are mandatory. The OFP, however, didn't have sufficient specifications or documentations to apply test techniques. It had only a few documents such as informal specifications and non-standardized documents.

T.-h. Kim et al. (Eds.): ASEA/DRBC/EL 2011, CCIS 257, pp. 285–291, 2011.

We decided to do software reverse engineering against the OFP in order to develop formal specification and structure information, which are the prerequisite for software testing. Reverse engineering is a process of analyzing a target system for a specific subject to identify the system components and their interrelationships, and create representations of the system in another form or at a higher level of abstraction [6]. There are two subareas that are widely referred to: redocumentation, and design recovery. The redocumentation is the creation or revision of a semantically equivalent representation within in the same relative abstraction level. On the other hand, the design recovery adds domain knowledge, external information and deduction or fuzzy reasoning to the observations of the subject system.

Results of redocumentation, represented by data flows, data structures, or control flows, make us possible to understand a whole structure and flow of a target system. The information will be useful to perform structural testing [7]. For example, understanding data structures and flows helps us rise test case adequacy like coverage. We, therefore, decided to use structural testing technique against the OFP with the results of redocumentation.

This paper introduces our plan to verify the OFP through reverse engineering systematically, and we share an experimentation about a small portion of the plan. The remainder of the paper is organized as follows: Section 2 introduces background information on the target system, OFP in HELISCOPE project, and reverse engineering briefly. Section 3 shows the plan from the reverse engineering to the test of the OFP and Section 4 covers the experimentation. Finally we conclude the paper and sum up some unsolved problems in Section 5.

2 Background

2.1 Operational Flight Program

OFP is developed as a subpart of the HELISCOPE project and it is based on the well-known TMO scheme [8]. The OFP support the unmanned helicopter navigation that is done by commands on flight mode from GCS (Ground Control System). It operates servo motors using collected data from various sensors such as GPS (Global Positioning System), navigation, CGS, and SWM (Helicopter Servo Actuator Switching Module).

There are six threads, four readers, one controller, and one monitor, and they run simultaneously. Fig. 1 shows an overview of the OFP working with servo motor and sensors. The monitor thread catches a data packet from sensors and operates one of reader threads which is supposed to collect the data. If the monitor thread operates one of reader threads, then the reader thread reads the data and saves the data in ODS (Object Data Storage). The controller thread, otherwise, computes collected data to control the servo motors. The reader threads and controller thread share ODS (Object Data Store) to forward data collected from sensors. To avoid simultaneous use of the OSD by them, mutual exclusion algorithm is used.

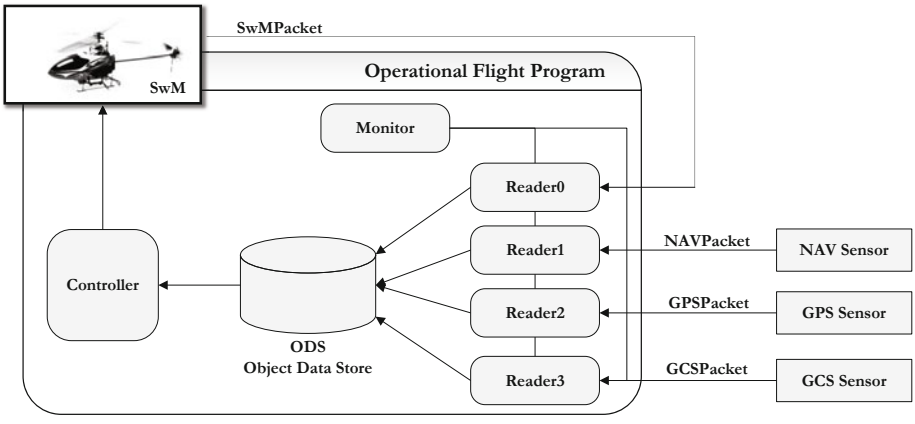

Fig. 1. An overview of Operational Flight Program with servo motor and sensors

2.2 Reverse Engineering

Origin of reverse engineering is in the analysis of hardware — where the practice of deciphering designs from finished products is commonplace. Reverse engineering of software is also the practice of analyzing a software system, either in whole or in part, to extract design and implementation information [9]. It can be performed any level of abstraction or at any stage of the life cycle. There are no changes or modifications about target system during performing reverse engineering. This performance only crates documents or abstract information about the target system.

3 A Testing Plan for the HELISCOPE OFP

The testing plan for the HELISCOPE OFP consists of two parts: *reverse engineering* and *testing*. Fig 2. describes the overall plan. Analysis on the source codes and informal/unstructured documents is the first step of our reverse engineering. It will progressively produce 4 different documents: data descriptions, structure charts [10][11], data flow diagrams, and control flow diagrams. Data descriptions are derived from definitions and uses of variables. The structure charts are also derived from the data descriptions and functions defining the relationship of data. The data flow and control flow diagrams are finally recovered from the structure charts. The more we perform the redocumentation, the more extractive the source code and informal documents become. All documents derived in this part become a source of activities in the testing part.

There are two activities in verification part. Test cases generation generates new test cases, while referring related documents produced from the reverse engineering (redocumentation) process. The other one is test execution with test cases including new test cases generated in test cases generation activity. We estimate that test case coverage (e.g. such as statement, branch, or MC/DC

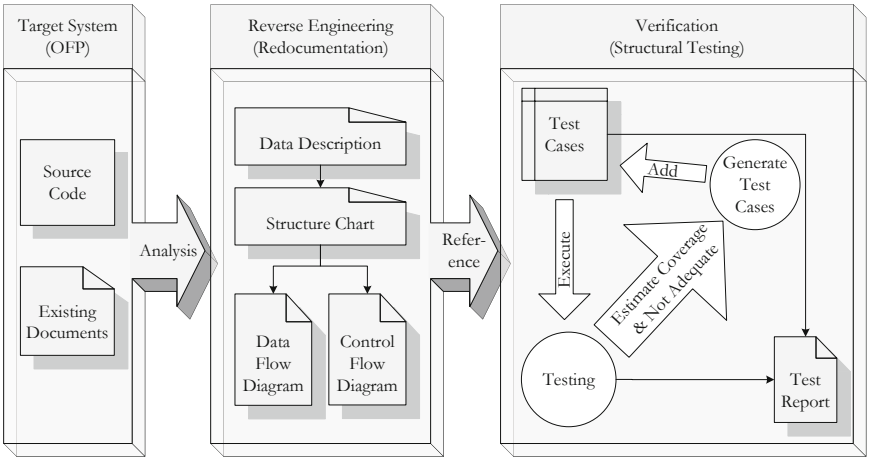

Fig. 2. The OFP verification plan through reverse engineering

(Modified Condition & Decision Condition) coverages) [12] is adequate. It means lack of test cases to cover the whole system that one of coverage which we decided to set a test criterion is not adequate sufficiently. We, thus, should generate new test cases until the coverage is adequate. Analysis of DFD or CFD makes flows the system, so we can discover uncovered area to generate new test cases. Test report include the results of the two activities.

4 Experimentation

In this section, we share our experiment with respect to the verification described in Section 3. We performed reverse engineering with source code and documents of OFP, and executed structural testing to its source code. To recover data description, first of all, we analyzed functions and its relationship using Doxygen [14] which is a documentation system. Next, we manually drew an outline of the structure chart, referring the generated relationship of function calls and the control flows and data flows are added on the structure chart. Fig. 3[1] shows the result of recovering structure chart used notations defined by Yourdon [13].

DFD (Data Flow Diagram) supports that we generate test cases, so we recovered DFD referring to the structure chart. Outlines of the DFD are derived from the structure chart, and detail data flows are referred from actual source code. We should derive the DFD starts from higher level which is level 0 expressing a outline of data flow roughly, because we only could refer the recovered structure of the target system. The deeper level of DFD is the more detailed analysis is progressed. Fig. 4[2] shows the DFD from level 0 to level 2. The whole of the

[1] Assumed names of all modules are substituted for original ones in source code for security reasons.

[2] Assumed names of all processes and flow labels under level 1 are substituted for original ones of source code for security reasons.

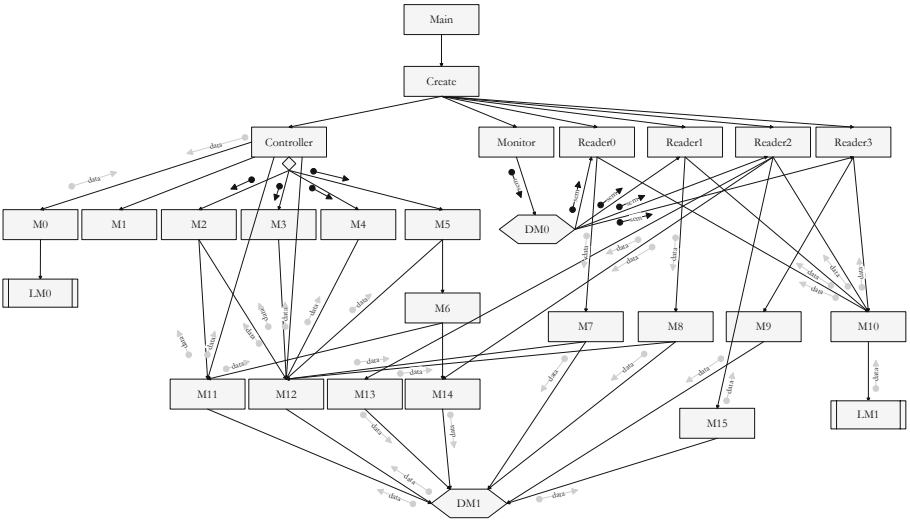

Fig. 3. Recovered structure chart of the Operational Flight Program

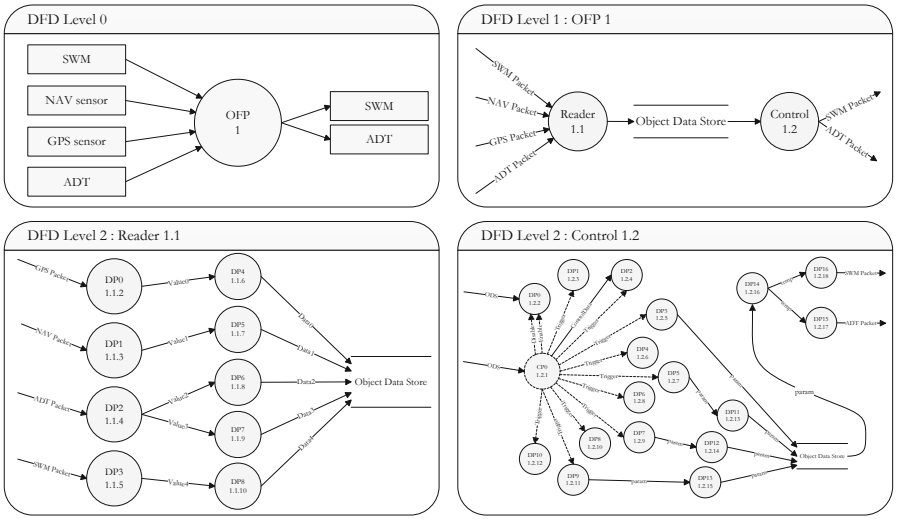

Fig. 4. Recovered data flow diagram of the Operational Flight Program

DFD consists of 5 levels and the last level, level 4, is made up of state transition diagram.

We focused on modules which affect control of unmanned helicopter without communicating sensors through serial ports. Testing environment within a PC, therefore, is sufficient without a embedded system environment in use. We set QNX Software Development Platform 6.5.0[15] as an operating system in virtual

environment. The QNX is one of RTOS (Real-Time Operating System) which the OFP use. To check statement coverage, we used a test coverage program named gcov [16].

We, first of all, selected data referring related documents which are data descriptions and DFD. Next we generated initial test cases about the data and executed the test. Estimation of statement coverage was not adequate at first. We, therefore, executed test case generation and testing activities over 10 times, and could get adequate test cases coverage. Table 1 shows the result of the testing. Unfortunately, some of target modules don't have 100 % statement coverage, because they include a few unused codes or codes to access serial ports. Those statements, however, is not our consideration which we set before start the test, so we could make decision that the test cases are adequate.

Table 1. Result of structural testing

Module Name	Number of Test Cases	Statement Coverage
Module 0	11	99.47 %
Module 1	1	100.00 %
Module 2	17	100.00 %
Module 3	4	100.00 %
Module 4	3	93.33 %
Module 5	4	86.26 %

5 Conclusion

This paper introduced a systematic verification plan and parts of practical use of OFP in HELISCOPE project through reverse engineering. We identified that results of performing reverse engineering, derived from source code and informal documents, are useful information to analyze structure and execute structural testing about the target system. Our experimentation did not cover widely used coverages such as branch or MC/DC, so we plan to perform testing with different coverage criterias.

Functional testing is good to verify functionality, and it is available through design recovery technique mentioned above. The technique, however, needs very close collaboration with developers of the target system, because the design includes additional domain knowledge, external information, etc. We also plan the collaboration with the developer of the OFP, and expect that those additional verification techniques make the OFP more reliable.

Acknowledgments. This research was supported by the MKE(The Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency) (NIPA-2011-C1090-1131-0003)

References

1. Kim, D.H., Nodir, K., Chang, C.H., Kim, J.G.: HELISCOPE Project: Research Goal and Survey on Related Technologies. In: The Proceeding of 12th IEEE International Symposium on Object /Component / Service-Oriented Real-Time Distributed Computing (ISORC), Tokyo, pp. 112–118 (2009)
2. Kim, S.-G., et al.: Design and Implementation of an Operational Flight Program for an Unmanned Helicopter FCC Based on the TMO Scheme. In: Lee, S., Narasimhan, P. (eds.) SEUS 2009. LNCS, vol. 5860, pp. 1–11. Springer, Heidelberg (2009)
3. Berard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., Schnoebelen, P.: Systems and Software Verification: Model-Checking Techniques and Tools. Springer, Heidelberg (2001)
4. Lee, D.-A., Yoo, J., Kim, D.: Formal Verification of Process Communications in Operational Flight Program for a Small-Scale Unmanned Helicopter. In: The 6th International Conference on Intelligent Unmanned Systems (ICIUS 2010), Bali, Indonesia, pp. 91–96 (2010)
5. Lee, D.-A., Sung, S., Yoo, J., Kim, D.-H.: Formal Modeling and Verification of Operational Flight Program in a Small-Scale Unmanned Helicopter. Journal of Aerospace Engineering (accepted, 2011)
6. Chikofsky, E.J., Cross, J.H.: II: Reverse engineering and design recovery: a taxonomy. IEEE Software 7(1), 13–17 (1990)
7. Pezze, M., Young, M.: Software testing and analysis: process, principles, and techniques. Wiley (2008)
8. Kim, K.H., Kopetz, H.: A Real-Time Object Model RTO.k and an Experimental Investigation of Its Potentials. In: 18th IEEE Computer Software & Applications Conference, Los Alamitos, pp. 392–402 (1994)
9. Stavroulakis, P., Stamp, M.: Handbook of Information and Communication Security. Springer, Heidelberg (2010)
10. Martin, J., McClure, C.: Diagramming Techniques for Analysts and Programmers. Prentice-Hall, Englewood Cliffs (1985)
11. Yourdon, E.: Constantine, Structured Design. Prentice-Hall, Englewood (1979)
12. Zhu, H., Hall, P., May, J.: Software Unit Test Coverage and Adequacy. ACM Computing Surveys 29, 366–427 (1997)
13. Yourdon, E.: Modern structured analysis. Yourdon Press (1989)
14. Doxygen, `http://www.stack.nl/~dimitri/doxygen/index.html`
15. QNX Software Systems, `http://www.qnx.com`
16. gcov—a Test Coverage Program, `http://gcc.gnu.org/onlinedocs/gcc/Gcov.html`