# A Method to Diagnose Self-weaknesses for Software Development Organizations

Chi-Lu Yang[1, 2], Yeim-Kuan Chang[1], Chih-Ping Chu[1]
[1] Department of Computer Science and Information Engineering, National Cheng Kung University
[2] Networks and Multimedia Institute, Institute for Information Industry
[1,2]Tainan, Taiwan R.O.C.
stwin@iii.org.tw, ykchang@mail.ncku.edu.tw, chucp@csie.ncku.edu.tw

## Abstract

A root cause is a source of a defect such that if it is removed, the defect is subsequently decreased or removed. By analyzing the root causes of defects of our software projects, we will be able to determine the weaknesses of our software development teams. We could thus decide on how much effort to be invested on specific actions to improve the weaknesses of the teams. In this paper, we first described how defects were objectively collected during project development. Second, the root causes were defined and categorized into six groups. Then we focused on analyzing defects to find out their root causes. Based on statistical results, the weaknesses of the project teams were determined. The results showed that the disturbing defects in our projects were mainly injected in the design phase, especially in the detail design phase. Moreover, we should invest considerable effort on enhancing our detail design skills, such as designing components, algorithms and interfaces, and so on. Some corrective actions and prevention proposals would correspondingly be acted upon and planned, respectively. Overall, we believe that our experiences and methods are worthy of sharing.

Keywords: cause analysis and resolution, root causes, defect detection, defect prevention, test cases generation

## 1. Introduction

Defects in software could have wide and varying effects with several levels of inconvenience to users. Some defects only result in a slight effect on the functionality. Thus, they may be undetectable for a long period of time. On the other hand, more serious defects may cause the system to crash or freeze. We should try our best to stop these defects from reaching to our customers. Defects might be discovered in different situations during project development [1], such as (1) Reports of project control with corrective actions for problems. (2) Defective reports from the customers. (3) Defective reports from end users. (4) Defects found in peer reviews. (5) Defects found in testing. (6) Process capability problems, etc. If defects are discovered, they will be fixed with specific actions at an extra cost.

A root cause is a source of a defect such that if it is removed, the defect is decreased or removed. By analyzing the root causes of defects in our projects, we will be able to determine the weaknesses of our software development team. This is useful for similar projects in the future. Based on this analysis, we could iteratively decide to invest how much effort is required on improving the weaknesses of the teams in the next project generation. To prevent defects through enhancing weaknesses of project teams is an ideal method for software development organizations.

## 2. Related works

A defect, synonymous with fault, is a deviation between the specification and the implementation. A defect implies a problem discovered after the product has been released to customers or end-users (or to another phase in the software life cycle) [2]. Consequently, defects reasonably make software people look bad. As software engineers, we hope to detect and prevent as many defects as possible before the customer and/or end-users encounter them.

A defect would naturally amplify in the next phase during the software life cycle, resulting in higher project cost. To detect or prevent defects injected into a product is an important task during software development. Meng [3] proposed a general framework to prevent defects. This framework consists of organization structure, defect definition, defect prevention process, and quality culture establishment. C.-P. Chang [4] proposed an action-based approach to prevent defect in software processes. Action-Based Defect Prevention (ABDP) approach applies the classification and Feature Subset Selection (FSS) technologies to project data during execution. Van Moll [5] discussed the importance of life cycle modeling for defect detection and prevention, proven methods that can be used in an efficient way. Marek [6] reported a retrospective root cause defect analysis study of the defect Modification Requests (MR) discovered. Through classifying root causes and defects, he tried to detect the defects at the earliest possible point during software development. Khaled [7] inquired the causal analysis of changes for a large system. The findings from Khaled served as input for process improvement within organizations.

The most significant challenge for causal analysis is to identify the causes of defects among large amounts of defect records where the cause-effect diagram and control chart can be utilized to support the analysis process [8]. The defect tendency is difficult to investigate when the root cause analysis schema is complicated and the sample size of the defect is small [9]. To solve this problem, the historic data on multiple releases of products can be utilized to discover the defect patterns, and be used to predict the possible defects. To decrease the effort involved in causal analysis, defect distribution

can be applied to show the metrics of defects and classify them in terms of their causes [10].

For defect prevention and root causes analysis, we would propose a defect analyzing method to diagnose the weaknesses of the project teams. In this paper, we would first describe how defects were objectively collected in distinct situations during project development. Second, root causes are defined and categorized by inducting our experiences. Then we will focus on analyzing defects to determine their root causes. When we could be aware of how and when defects are injected into our projects, we would be able to prevent them as early as possible. Since weaknesses of the team could be identified, corrective actions and prevention proposals could then be enacted and planned, respectively.

## 3. Project Information and Test policy

The objectivity for the detection and correction of defects is greatly related to project organization. Testers and developers should have equal relationships in the project position. Both of them report to the project leader in their own channels. Accordingly, the leader coordinates or arbitrates their arguments when they have issues. Testers will be able to detect bugs in an objective manner in this hierarchy. Furthermore, they will perform document review and collect information on defects through testing based on testing plan in phases. Project Organization is showed in figure 1. Other stakeholders' definitions in the project are described as follows.

### 3.1 Objectively Defects Acquirement

The project leader is responsible for the project. He should plan the project and monitor its progress. His group members cooperate with each others to accomplish the project. There is a special Project Analytic Group (PAG) between the members of the project and the stakeholders of customers. PAG is led by the project leader and is made up by sub-leaders of developers and testers. PAG is organized to draw out the requirements and business models from customers. It is also responsible for high level design, which is also called architecture design in the project life cycle.

Customers provide requirements and special needs to the project. In our project, the customers also work with us to plan innovative business models. The requirement analysis is considered as one of the most important techniques in software engineering. In real projects, certain defects are incurred in this phase. We spent about 30% of the project time in analyzing requirements. Another special role in the project is the Project Quality Assurance. His mission is to make sure that the project processes are correctly performed.

The testers' main missions are to plan and perform the test plan. In the test plan, there are test strategies, test items, test cases and simulate environment, etc. After the test plan has been confirmed, the testers in different phases will perform the unit test, integration test, system test, and B-test to collect test results. One special task with testers in the project is that their representative should join the requirement and design document review. Through this early stage of participation, they would be able to understand the key requirements.

In this project, there are three developing sub-groups, which are named homebox, bio-server, and gateway. Developers in the sub-groups will design detailed functions of modules and components, which are initially peer reviewed. In addition, the developers are responsible for developing and integrating modules and components. Fixing bugs is one of their most significant tasks.

The data analyzer is responsible for analyzing and implementing system data. When there are data demands existing in the systems, the data analyzer will design table schemas in database, system configuration by XML or data flowchart between modules, etc.

The project life cycle is divided into phases, which are system analysis (SA), system design (SD), coding, testing, demo, deployment, and so on. There are different missions performed by stakeholders during the project life cycle. These missions are defined in table 1. In different phases, stakeholders perform specific missions for the project. These are accordingly shown in table 2.

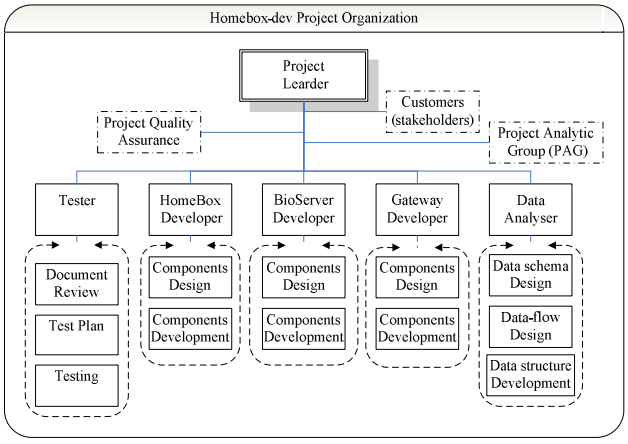


Fig. 1. Project Organization of Homebox-devices

Table 1. Specific Missions in a Project

| Missions | Descriptions |
|---|---|
| MM | To Manage and coordinate work items and members of the project |
| DR | To Review Documents, including requirement, design, and testing documents |
| PR | To Plan the Requirement document and analyze requirement |
| PD | To Plan the Design document, high level design, and detail design |
| PT | To Plan the Testing document, test strategies, test plan, and test cases |
| AF | To Analyze the data scheme and to design the data Flowchart. (high level data analysis in design document) |
| AI | To Analyze and Implement details of the data scheme. (detail data analysis in design document) |
| RD | To Research and Develop algorithms for functions |
| Dbug | To fix bugs which are detected in all phases |
| A-test | To test functions of software and system in the Lab |
| B-test | To test the system in real environment |

Table 2. Roles vs. Specific Missions

| Phases / Roles | SA | SD | Coding | Integration & Testing | Demo / Deploy-ment |
|---|---|---|---|---|---|
| Leader | PR | DR | DR | MM | MM |
| Customer | DR | -- | -- | --- | B-test |
| Tester | DR | DR | PT | A-test | B-test |
| Homebox Developer | DR | PD | RD | Dbug | Dbug |
| Bio-server Developer | DR | PD | RD | Dbug | Dbug |
| Gateway Developer | DR | PD | RD | Dbug | Dbug |
| Data Analyzer | DR | AF | AI | Dbug | Dbug |

Defects might be injected during development and would be discovered in later phases. Such defects would be hidden in documents or systems. The document defects could be detected by peer review, regular review or milestone review. Engineers with more experience would be able to detect more precisely based from the documents. However, system defects would be detected through code review, test in lab, training course to customers or bugs report from end-users, and so on. The test policy in our projects is described in the following sub-section.

### 3.2 Test Case Generation and Execution

Our healthcare projects are iteratively developed in the past three years. The life cycle of the projects are divided into phases, which are customers' needs and business modeling, requirement analysis, system design, implement and integration, and deployment. As the phases progress, defects are not only discovered by reviewing documents but also detected by executing test cases, which are generated based on documents in the early phases. The life cycle progress and test cases generation are showed in figure 2.

In figure 2, there are three defined types of test cases: unit test case, integrated scenario test case, and system test case. The objective of unit test cases is to verify the correctness of unit functions, which were built from design documents. On the other hand, the objective of integrated scenario test cases is to verify requirements or special goals, which were combined from several relative unit functions. Lastly, the objective of system test cases is to verify the full executing system. Unit test cases are generated at the design phase, while integrated scenario test cases are created by requirement documents. System test cases are used to validate original customers' needs.

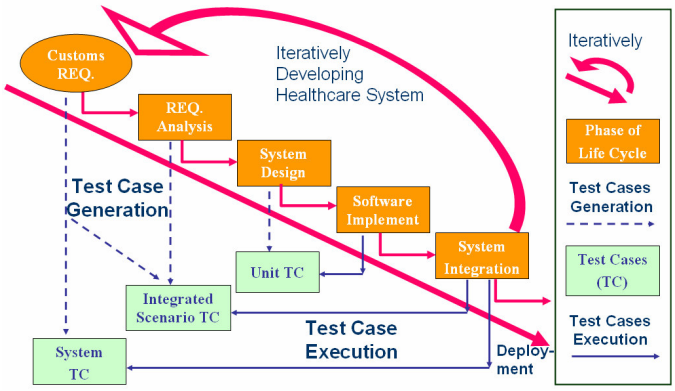


Fig. 2. Generating & Executing Test Cases to uncover defects

Every test case includes the following columns: test case ID, testing item, test step (each step contains pre-defined input data/frame, pre-defined output (pass/fail criteria), practice output, etc.), environment requirement (software and hardware), testing results, causes generally analysis, special needs of testing process, pre-test case ID, tester signature, tester/developer manager signature, remark, and so on.

When defects however are detected in later phases such as the deployment phase, even if it is discovered by engineers or customers, it would still require more cost to trace the root causes (when and how) of the defects. Therefore, the project cost would extremely increase.

## 4. Cause Analysis and Resolution

### 4.1 Root Causes Definition

The existing defects or problems in the project result from certain root causes during project development. After the defects or problems are injected, they would be detected or discovered in later phases. Engineers would spend considerable effort to detect and fix them. The later the defects are detected, the more project cost will be incurred. In order to decrease the project cost and prevent future defects, we should try to address when and how defects are injected into the projects. Our major objective is to prevent and fix defects as early as possible. We will define twenty-six frequent engineering defective types by inducing our defects' root causes. The descriptions of root causes are shown in table 3.

### 4.2 Grouping Root Causes

The root causes of defects are aggregated through the characteristics along the phases. They are categorized into six groups, such as business model, requirement, design, test, deployment, and hardware. Each root cause exactly belongs to one group. These groups are shown in the figure 3.

Table 3. Root Causes Types

| ID_RC | Root Causes Types | The Descriptions of Root Causes |
|---|---|---|
| B1 | Missing Address in Business | In early phases, requirements or constraints which customers didn't mention, and the analyzers also missed to address them. |
| B2 | Changing Functional Spec. | After the engineers implemented functions, customers suggest changing functional requirements or business models. |
| B3 | Changing Non-Functional Spec. | After the engineers implemented functions, customer suggests changing non-functional requirements or business models. |

| | | |
|---|---|---|
| R1 | Missing Functional Spec. | In the analysis phase, engineers missed functions which customers had mentioned or implied in early phases. |
| R2 | Faulty Functional Spec. | In the analysis phase, engineers planned wrong functions which customers had mentioned or implied in early phases. |
| R3 | Faulty Interface Spec. | In the analysis phase, engineers planned the wrong interface which customers had mentioned or implied in early phases. Interface types are categorized into users interface and systems interface. |
| R4 | Ambiguous Non-functional Spec. | Non-functional specs were lost in analysis phase. For example, performance constraint, response time, transmission rate, and services capacity. |
| D1 | Missing Design Spec. | In the design phase, engineers missed designs of functional details which had been recorded or implied in early phases. |
| D2 | Faulty Design Spec. | In the design phase, engineers designed wrong functional details which had been recorded or implied in early phases. |
| D3 | Missing Exception Handler | In the design phase, engineers missed exception handlers of functional details which are either necessary or implied in early phases. It is necessary to check the user's input format. |
| D4 | Faulty Data Schema Design | In the design phase, analyzers designed inapplicable data schema or structure. Data models do not match with customers' requests. |
| D5 | Faulty Data-flow Design | In the design phase, analyzers designed inapplicable data-flow for the schema. Data models do not match with customers' requests. |
| D6 | Faulty Algorithm Design | In the design phase, researchers designed inapplicable algorithms for the functions mentioned in earlier phases. Algorithms don't achieve customers' requests or requirement. |
| IM1 | Erroneous Implementation | In the implement phase, programmers make erroneous components or modules recommended in earlier phases. |
| T1 | Undetected Unit Test | Un-detected Unit Test |
| T2 | Undetected Integration Test | Un-detected Integration Test |
| T3 | Undetected System Test | Un-detected System Test |
| P1 | Real Network Mistake | In the real network, the product works abnormally as a result of other network devices. |
| P2 | Wrong Version Control | Defects arose from someone releasing wrong software version. |
| P3 | Wrong Configuration Setting | Defects arose from someone operating to setup the wrong configurations in the homebox or bio-server. |
| P4 | Improper User Operation | Defects occurred as a result of users operating the devices improperly. |
| H1 | Server Hardware Failure | Bio-server's exceptions result in system's failures. |
| H2 | Peripherals Failure | Peripherals' exceptions result in system's failures. |
| H3 | OS Failure | Operation System's exceptions result in system failures. |
| H4 | Homebox Hardware Failure | Defects arose from homebox hardware |
| O1 | Other Undefined Types | Undefined defective types. Certain defects which did not originate from engineering development processes. |

## 5. Experiment Results and Discussion

By reviewing documents and testing systems, defects were formally collected during the years 2005 and 2007. Along the product life cycle, certain defects were detected in the lab, while some were reported by customers. There were 256 defects which were formally recorded as shown in figure 4. The phase-distribution of the discovered defects is showed in figure 5. Here, we can observe that defects are almost normally distributed in all phases.

Subsequently, defects were also traced backwards to determine their root causes. Since all specifications are recorded in documents, we could then trace their developing progress to determine their root causes. The distribution of root causes of all defects is shown in figure 6. Here, the situation was generally surprising. Previously, we misapprehended that the disturbing defects came mainly from the requirement analysis. However, the 43% in figure 6 have shown that the disturbing defects were mainly injected in the design phase, especially in the detail design phase. This was rather interesting considering that we frequently hear from engineers that bugs usually come from careless requirement analysis. However, more bugs came indeed from thoughtless design. Although we always have to pay more attention on our testing skills for verification, we must, moreover, spend substantial efforts in enhancing our detail design skills, such as designing components, algorithms, interfaces, and so on.

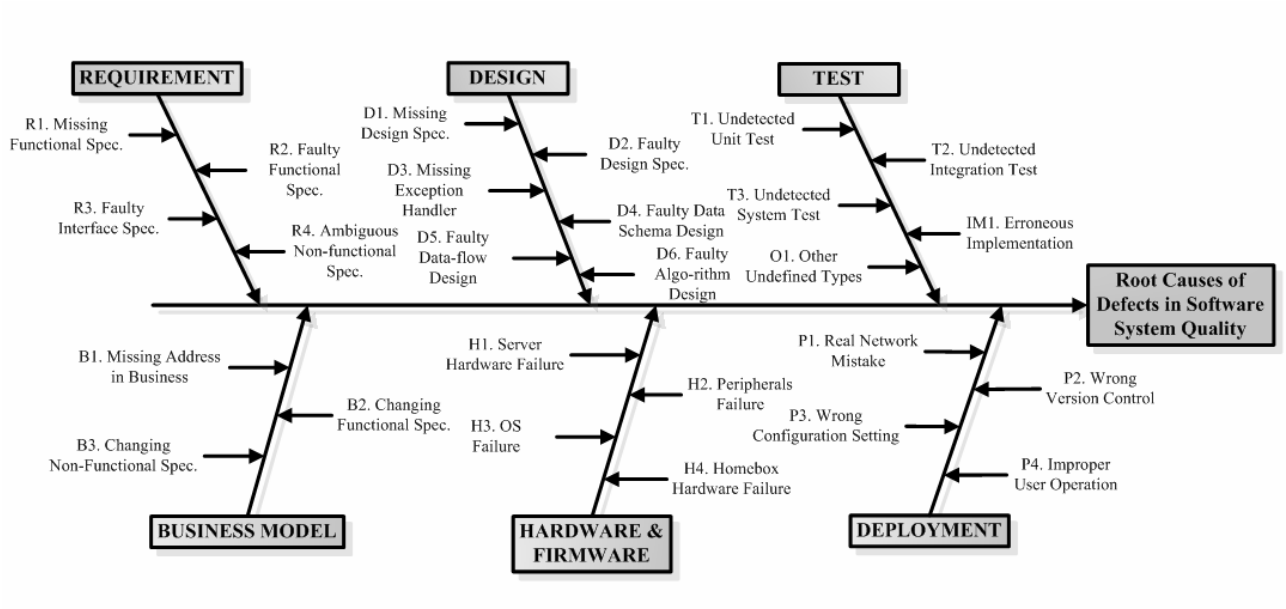


Fig. 3. Cause-and-effect (fishbone) diagrams

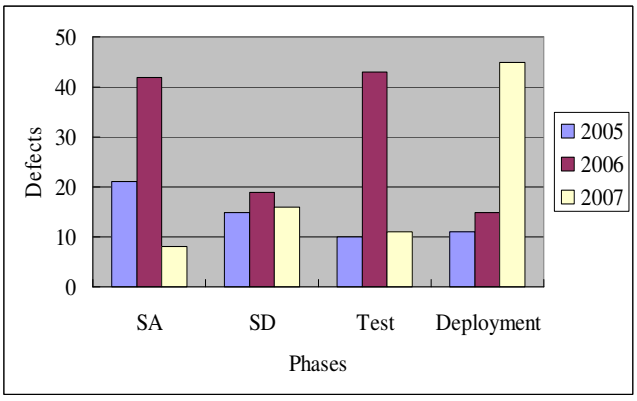


Fig. 4. Defects Collection by phases

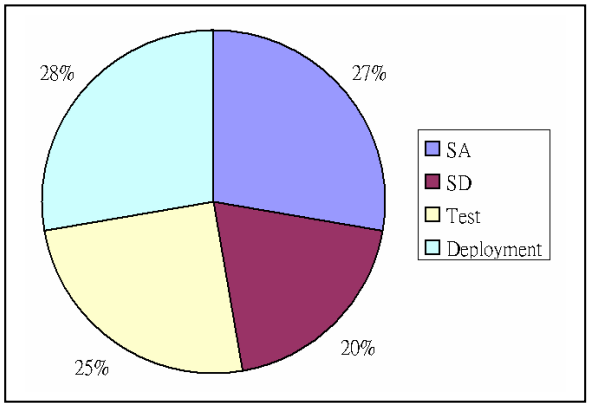


Fig. 5. The Phase-Distribution of the Discovered Defects

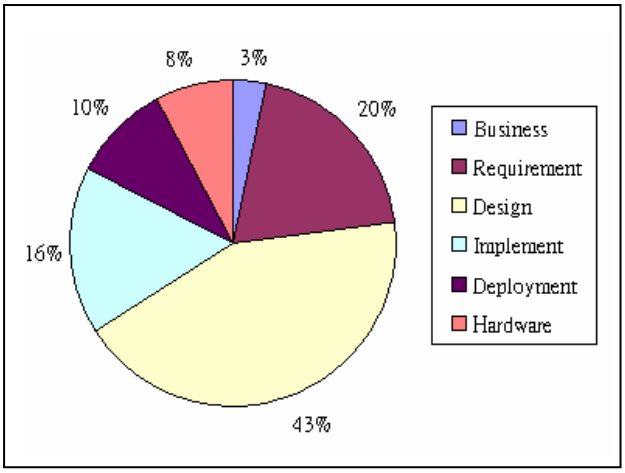


Fig. 6. The Distribution of Root Causes

In Figure 7 below are some interesting findings of our study. Figure 7 (a) shows that increasing efforts spent by engineers in SA and SD would lead to less defects detected in the test and deployment. Figure 7 (b) shows that as the efforts spent by engineers in SA increases, there would be lesser defects detected in the deployment. Consequently, less effort spent by engineers in SD would lead to more defects detected in the test. In figure 7 (c), when engineers spend less effort in SA, SD and the test, much more defects would be detected in the deployment phase.

## 6. Conclusion

Defects in software could have wide and varying effects with several levels of inconvenience to users. If defects are discovered, they will be fixed through specific actions with extra costs. In order to bring this cost down, preventing defects by enhancing the weaknesses of project teams is an ideal method for software development organizations. For defect prevention, we proposed a defects analysis method to diagnose the weaknesses of the project teams. We described how the defects were objectively collected during project development. The root causes were defined and categorized into six groups. Afterwards, we focused on analyzing the defects to determine their root causes. Our study has pinpointed how and when defects are injected into our projects. The weaknesses of the teams could be identified, so corrective actions and prevention proposals could then be enacted and planned, respectively. Statistical results have shown that the

disturbing defects in our projects were mainly injected in the design phase, especially in the detail design phase. Moreover, considerable effort should be invested on enhancing our detail design skills, such as designing components, algorithms and interfaces, and so on.

## Acknowledgements

This research was supported by the Applied Information Services Development and Integration project of the Institute for Information Industry (III) and sponsored by MOEA, Taiwan R.O.C.

## References

[1] CMMI Product Team, Capability Maturity Model Integration V. 1.2, Carnegie Mellon University: SEI, Pittsburgh, USA, Aug. 2006.

[2] Pressman, Roger S., Software Engineering: A Practitioner's Approach, 6th Ed., New York: McGraw-Hill, 2005.

[3] Meng Li., He X. and Ashok S., "Defect Prevention: A General Framework and Its Application," in the Proceedings of Sixth International Conference on Quality Software (QSIC'06), Beijing, China, Oct. 2006.

[4] C.-P. Chang and C.-P. Chu, "Defect prevention in software process: An action-based approach," The Journal of Systems and Software (80), p.559-570, 2007.

[5] J.H. Van Moll, J.C. Jacobs, B. Freimut and J.J.M. Trienekens, "The Importance of Life Cycle Modeling to Defect Detection and Prevention," In the Proceedings of the 10th International Workshop on Software Technology and Engineering Practice, Washington, DC, USA, 2002

[6] Marek L., Dewayne E. P. and Eieter S., "A Case Study in Root Cause Defect Analysis," In the Proceedings of the 22nd international conference on Software engineering, p.428-437, Limerick, Ireland, 2000.

[7] Khaled E.E., Drik H. and Nazim H. M., "Causal Analysis of the Requirements Change Process for a Large System," In the Proceedings of the International Conference on Software Maintenance, Washington, DC, USA, 1997

[8] Card, D.N., "Learning from our mistakes with defect causal analysis," IEEE Software 15 (1), p56-63, 1998.

[9] Leszak, M., Perry, D.E., and Stoll, D., "Classification and evaluation of defects in a project retrospective," The Journal of System and Software (61), p173-187, 2002.

[10] Pooley, R., Senior, D., and Christie, D., "Collecting and analyzing web-based project metrics," IEEE Software 19 (1), p52-58, 2002.

[11] Chi-Lu Yang and Richard Weng, "A Method for Verifying Usability and Performance of a Multi-user Healthcare Embedded System," in the Proceedings of the Thirteenth International Conference on Distributed Multimedia Systems (DMS'07), p75-80, San Francisco, USA, Sep. 2007.

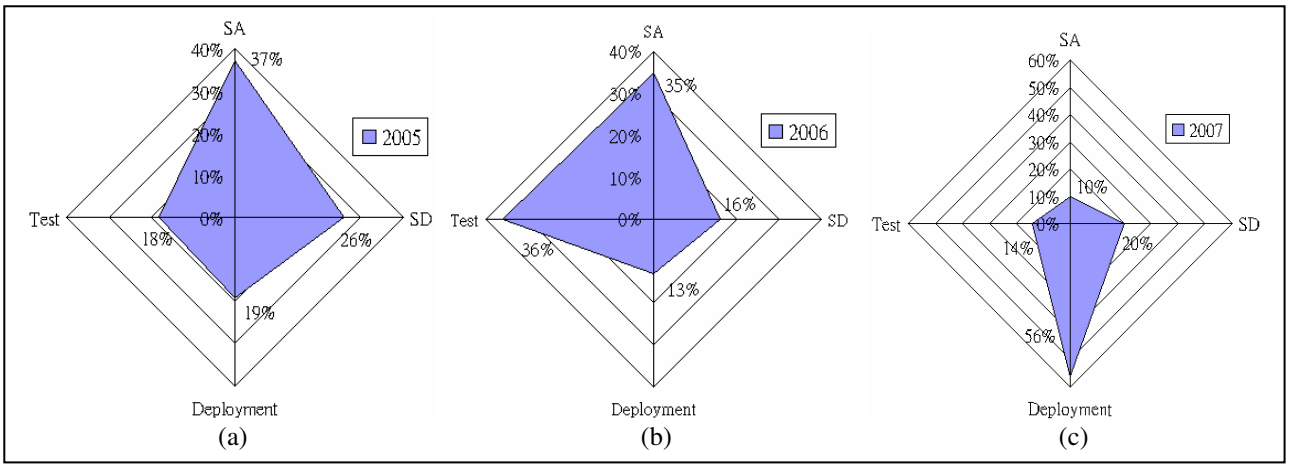


Fig. 7. Variance of Defects in Phases