STATIC TECHNIQUES

Static test techniques provide a powerful way to improve the quality and productivity of software development. This chapter describes static test techniques, including reviews, and provides an overview of how they are conducted. The fundamental objective of static testing is to improve the quality of software work products by assisting engineers to recognize and fix their own defects early in the software development process. While static testing techniques will not solve all the problems, they are enormously **effective**. Static techniques can improve both quality and productivity by impressive factors. Static testing is not magic and it should not be considered a replacement for dynamic testing, but all software organizations should consider using reviews in all major aspects of their work including requirements, design, implementation, testing, and maintenance.

“Static techniques”

MUHAMAD IQBAL SAKTI
11453101976

PROGRAM STUDI S1 SISTEM INFORMASI
FAKULTAS SAINS DAN TEKNOLOGI
UIN SUSKA RIAU
2017

http://sif.uin-suska.ac.id/
http://fst.uin-suska.ac.id/
http://www.uin-suska.ac.id/

STATIC TECHNIQUES

REVIEWS AND THE TEST PROCESS

Recognize software work products that can be examined by different static techniques. (K1)

Describe the importance and value of considering static techniques for the assessment of software work products. (K2)

Explain the difference between static and dynamic techniques. (K2)

In Chapter 1, several **testing** terms were presented. Also testing itself was defined. The latter definition is repeated here as a means for explaining the two major types of testing.

The definition of testing outlines objectives that relate to evaluation, revealing defects and quality. As indicated in the definition two approaches can be used to achieve these objectives, **static testing** and **dynamic testing.**

STATIC TECHNIQUES

REVIEW PROCE

Recall the phases, roles and responsibilities of a typical formal review. (K1)

Explain the differences between different types of review: informal review, technical review, walkthrough and inspection. (K2)

Explain the factors for successful performance of reviews. (K2)

Reviews vary from very **informal** to **formal** (i.e. well structured and regulated). Although inspection is perhaps the most documented and formal review technique, it is certainly not the only one. The formality of a review process is related to factors such as the maturity of the development process, any legal or regulatory requirements or the need for an audit trail. In practice the informal review is perhaps the most common type of review. Informal reviews are applied at various times during the early stages in the life cycle of a document. A two-person team can conduct an informal review, as the author can ask a colleague to review a document or code. In later stages these reviews often involve more people and a meeting. This normally involves peers of the author, who try to find defects in the document under review and discuss these defects in a review meeting. The goal is to help the author and to improve the quality of the document. Informal reviews come in various shapes and forms, but all have one characteristic in common – they are not documented.

STATIC TECHNIQUES

Phases of a formal review

In contrast to informal reviews, formal reviews follow a formal process. A typical formal review process consists of six main steps:

- Planning
- Kick-off
- Preparation
- Review meeting
- Rework
- Follow-up.

STATIC TECHNIQUES

Roles and responsibilities

The participants in any type of formal review should have adequate knowledge of the review process. The best, and most efficient, review situation occurs when the participants gain some kind of advantage for their own work during reviewing. In the case of an inspection or technical review, participants should have been properly trained as both types of review have proven to be far less successful without trained participants. This indeed is perceived as being a critical success factor.

The best formal reviews come from well-organized teams, guided by trained moderators (or review leaders). Within a review team, four types of participants can be distinguished: moderator, author, scribe and reviewer. In addition management needs to play a role in the review process.

STATIC TECHNIQUES

Types of review

A single document may be the subject of more than one review. If more than one type of review is used, the order may vary. For example, an informal review may be carried out before a technical review, or an inspection may be carried out on a requirements specification before a walkthrough with customers. It is apparent that none of the following types of review is the 'winner', but the different types serve different purposes at different stages in the life cycle of a document.

The main review types, their main characteristics and common objectives are described below.

STATIC TECHNIQUES

Success factors for reviews

Implementing (formal) reviews is not easy as there is no one way to success and there are numerous ways to fail. The next list contains a number of critical success factors that improve the chances of success when implementing reviews. It aims to answer the question, 'How do you start (formal) reviews?'.

Find a 'champion'

A champion is needed, one who will lead the process on a project or organizational level. They need expertise, enthusiasm and a practical mindset in order to guide moderators and participants. The authority of this champion should be clear to the entire organization. Management support is also essential for success. They should, amongst other things, incorporate adequate time for review activities in project schedules.

Pick things that really count

Select the documents for review that are most important in a project. Reviewing highly critical, upstream documents like requirements and architecture will most certainly show the benefits of the review process to the project. These invested review hours will have a clear and high **return on investment**. In addition make sure each review has a clear objective and the correct type of review is selected that matches the **defined objective**. Don't try and review everything by inspection; fit the review to the **risk** associated with the document. Some documents may only warrant an informal review and others will repay using inspection. Of course it is also of utmost importance that the right people are involved.

STATIC TECHNIQUES

STATIC ANALYSIS BY TOOLS

Describe the objective of static analysis and compare it to dynamic testing. (K2)

Recall typical defects identified by static analysis and compare them to reviews and dynamic testing. (K1)

List typical benefits of static analysts. (K1)

List typical code and design defects that may be identified by static analysis tools. (Kl)

There is much to be done examining software work products without actually running the system. For example, we saw in the previous section that we can carefully review requirements, designs, code, test plans and more, to find defects and fix them before we deliver a product to a customer. In this section, we focus on a different kind of static testing, where we carefully examine requirements, designs and code, usually with automated assistance to ferret out additional defects before the code is actually run. Thus, what is called **static analysis** is just another form of testing.

Static analysis is an examination of requirements, design and code that differs from more traditional dynamic testing in a number of important ways:

Static analysis is performed on requirements, design or code without actually executing the software artifact being examined.

Static analysis is ideally performed before the types of formal review discussed in Section 3.2.

Static analysis is unrelated to dynamic properties of the requirements, design and code, such as test coverage.

The goal of static analysis is to find defects, whether or not they may cause failures. As with reviews, static analysis finds defects rather than failures.

STATIC TECHNIQUES

Coding standards

Checking for adherence to coding standards is certainly the most well-known of all features. The first action to be taken is to define or adopt a coding standard. Usually a coding standard consists of a set of programming rules (e.g. 'Always check boundaries on an array when copying to that array'), naming conventions (e.g. 'Classes should start with capital C) and layout specifications (e.g. 'Indent 4 spaces'). It is recommended that existing standards are adopted. The main advantage of this is that it saves a lot of effort. An extra reason for adopting this approach is that if you take a well-known coding standard there will probably be checking tools available that support this standard. It can even be put the other way around: purchase a static code analyzer and declare (a selection of) the rules in it as your coding standard. Without such tools, the enforcement of a coding standard in an organization is likely to fail. There are three main causes for this: the number of rules in a coding standard is usually so large that nobody can remember them all; some context-sensitive rules that demand reviews of several files are very hard to check by human beings; and if people spend time checking coding standards in reviews, that will distract them from other defects they might otherwise find, making the review process less effective.

STATIC TECHNIQUES

Code metrics

As stated, when performing static code analysis, usually information is calculated about structural attributes of the code, such as comment frequency, depth of nesting, cyclomatic number and number of lines of code. This information can be computed not only as the design and code are being created but also as changes are made to a system, to see if the design or code is becoming bigger, more complex and more difficult to understand and maintain. The measurements also help us to decide among several design alternatives, especially when redesigning portions of existing code.

Experienced programmers know that 20% of the code will cause 80% of the problems, and complexity analysis helps to find that all-important 20%, which relate back to the principle on defect clustering as explained in Chapter 1. **Complexity** metrics identify high risk, complex areas.

The **cyclomatic complexity** metric is based on the number of decisions in a program. It is important to testers because it provides an indication of the amount of testing (including reviews) necessary to practically avoid defects. In other words, areas of code identified as more complex are candidates for reviews and additional dynamic tests. While there are many ways to calculate cyclomatic complexity, the easiest way is to sum the number of binary decision statements (e.g. if, while, for, etc.) and add 1 to it. A more formal definition regarding the calculation rules is provided in the glossary.

Below is a simple program as an example:

IF A = 354 THEN IF B > C THEN A = B ELSE A = C ENDIF

ENDIF

Print A

The **control flow** generated from the program would look like Figure 3.2.

The **cyclomatic complexity** metric is based on the number of decisions in a program. It is important to testers because it provides an indication of the amount of testing (including reviews) necessary to practically avoid defects. In other words, areas of code identified as more complex are candidates for reviews and additional dynamic tests. While there are many ways to calculate cyclomatic complexity, the easiest way is to sum the number of binary decision statements (e.g. if, while, for, etc.) and add 1 to it. A more formal definition regarding the calculation rules is provided in the glossary.

Below is a simple program as an example:

IF A = 354 THEN IF B > C THEN A = B ELSE A = C ENDIF

ENDIF

Print A

The **control flow** generated from the program would look like Figure 3.2.