A Preliminary Evaluation of Software Inspection Tools, with the DESMET Method

Henrik Hedberg and Jouni Lappalainen
Department of Information Processing Science
University of Oulu, Finland
henrik.hedberg@oulu.fi, jouni.lappalainen@oulu.fi

Abstract

Software inspections are established means of ensuring quality in software engineering. The traditional paper-based inspections are considered too laborious for widespread usage, and thus tool support for inspection has evolved. In parallel, generic document-processing tools such as Microsoft Word and Adobe Acrobat have developed support for features that can be utilized in inspections. This study aims to evaluate these two categories of tools with respect to their abilities to support software inspections. For this evaluation, the DESMET method will be employed, and a feature analysis of the tools will be conducted. The results of the evaluation show that the generic document producing tools provide many features applicable to inspections that even the dedicated tools lack. On the other hand, some features, such as support for metrics-based process improvement, are more effectively applied in the tools especially designed to support inspections.

1. Introduction

Software inspection is a well-known, acknowledged, and simple but disciplined method for locating defects in artefacts produced in any phase of software development. The term was first used by Michael Fagan in 1976 [2]. He defined it as a design or code inspection process with paper-based material and forms as well as face-to-face meetings.

Traditional software inspections are often considered too laborious and time-consuming to arrange. In addition to strict individual work and rigorous defect and metrics logging, the process includes physical meetings, where the main focus is on collecting all the data from participants. Finding a time and a place for collective inspection meetings is often impossible. One way to overcome these problems is to use tools to support collaboration, and to omit meetings whenever possible (see, for example [15, 9, 3, and 10]).

Tool support for software inspection evolved in the 1990s [7]. During this evolution, the principles of distributed and asynchronous inspections were outlined, and tested in many experiments. However, earlier studies [13, 14] have found that there is no tool that meets all the requirements of a software inspection tool. In addition, many of the tools are now obsolete or were developed only to demonstrate and experiment with very specific aspects of the transition from paper-based inspections to computer-mediated group work. New tools have arisen recently, but there is no study that evaluates those.

In addition to purpose-built tools, generic document processing tools may be used to support software inspections. For example, Microsoft Word has been evaluated [14], and Adobe Acrobat has been experimented with [5], but although both tools are widely used in practice when producing documents, the studies clearly implied that these tools were quite far from an ideal software inspection support system. Like the purpose-built inspection tools, these applications are under continuous development and constantly gain more and more features supporting group work.

In this paper, a preliminary model for evaluation based on the DESMET method [11] is built and applied, to systematically evaluate tools that can be used to support software inspections. The aim is to gather information on which to base a decision about suitability in a particular setting. By collecting experience from previous tools, experiments and evaluations, the requirements for the inspection tools can be specified and formulated into evaluation criteria.

The focus is on comparison of off-the-shelf tools and dedicated software inspection tools. Four tools were selected for the evaluation. The underlying assumption is that generic applications may now be exploited in software inspection if used creatively. The main advantage of these applications is that they support the
very same document types actually used, in most cases, in industry. The natural representatives of generic off-the-shelf document processing applications are the newest versions of Adobe Acrobat and Microsoft Word together with other Office applications supporting group work. IBIS (Internet Based Inspection System) [12] and XATI (XML Annotation Tool for Inspection), both developed recently, represent tools that have been specially designed to support inspections. These tools have an academic background, and have been tested in various experiments, and thus should encapsulate the long traditions of inspection tools. An additional factor used in selecting these representatives was their availability for installation, and the possibility of studying the tool thoroughly.

This paper is organized as follows. Chapter 2 describes the DESMET method, and Chapter 3 specifies the evaluation criteria. In Chapter 4 the feature analysis is conducted, and this is followed by discussion in Chapter 5. Chapter 6 concludes the study.

2. Evaluation method

This study aims to evaluate and compare a set of tools. To achieve this goal, an evaluation is conducted utilizing the DESMET method, developed by B.A. Kitchenham in conjunction with the U.K. DESMET project [11]. The generic nature of the DESMET evaluation method requires its user first to identify a suitable type of evaluation. This selection from nine different evaluation types depends on the parameters for the given evaluation context, which must be identified in the selection process. This section is concerned with the description of the generic DESMET method and the selection of a particular evaluation type conducted in this study, as well as with the more detailed description of the selected evaluation type.

2.1. DESMET

In a DESMET evaluation, a software tool or a software development method used by an organization is under scrutiny. For the purposes of this study, the method can be viewed only as a tool evaluation method [11].

DESMET evaluation is context-dependent and comparative. Therefore it is not used to find the absolute ranking of tools, but rather to gather information on which to base a decision about suitability in a particular setting. The context for this study is industry-driven, but it is academically conducted. The tools are to be used in a generic software-producing organization.

The first part of a DESMET evaluation includes the selection of an appropriate evaluation type. The evaluations identified with the DESMET method can be categorized according to the aspects of the tools that are to be studied. The effects of using the tool on the work of an organization are usually more easily measured with quantitative evaluations, whereas the appropriateness of a tool for a certain context can be better determined by qualitative means, usually concentrating on the features that the tools provide. Another way to classify the evaluations is according to their organization. Both quantitative and qualitative evaluations can be arranged as formal experiments, case studies or surveys. Qualitative evaluations can also be conducted as feature analysis screening, as described in Section 2.2. In addition to the aforementioned qualitative and quantitative evaluations, two hybrid methods have also been identified.

Due to the context given in this study, a qualitative method for evaluating tools was adopted. Feature analysis was selected to be conducted with a screening approach, and the goal for the evaluation was set as comparing different types of tools and providing information for further research. Of the other evaluation types, quantitative methods were deemed not as suitable for this study, since no single tool was selected as a focus. Qualitative surveys and experiments were also considered, but left for later research.

2.2. Feature analysis

Feature analysis is concerned with the degree of support that a tool provides for a given set of features. For each feature, the degree can be expressed either as a Boolean-type value indicating the presence or absence of support, or as a value on an ordinal scale, indicating the degree of support.

A set of main processes that a feature analysis consists of has been identified in [11]. The selection of candidate tools was made based on previous research, the generality and availability of the tools, and the coverage of different types of tools. The candidate tools are listed above. In the following section, a list of features to be evaluated for each tool is presented, along with feature priorities and a scoring scheme for the feature set. The reliability of the results of the evaluation was found to be something that was hard to assess due to the constraints imposed by the evaluation’s organization and execution. Therefore, a further, more detailed evaluation needs to be done before a single tool can be identified to be recommended for common inspection use. The results of the evaluation are reported in this paper, and a combination of graphical diagrams and textual descriptions of results is used.
3. Evaluation criteria

The evaluation criteria are derived from the functional requirements of a software inspection tool. Earlier evaluations [13, 14] have specified requirements, which we extend with the aspects of Virtual Software Inspection [6], namely flexibility and interoperability, as well as experience of software inspection tools. In addition, the achievements of the previous generations of tools [7] are also recognized. The criteria are divided into five categories: artefact, defect, and process management, as well as process improvement support and quality aspects, as summarized in Table 1.

The influence of the ISO 9126 Software engineering product quality standard [8] is especially evident in quality aspects which are based on the Quality in Use model specified in the standard. Internal and external quality models are also studied to complete the domain-specific requirements. Not all specific quality criteria are used, since they are not all applicable. For example, installability is an important aspect, but efficiency is not taken into account since in this context it is enough that the tool is usable, and thus the requirement is represented in another criterion. Integration is not seen as a distinct but crosswise requirement, and it appears in functional categories.

3.1. Artefact management

The target of software inspection is an artefact, such as a requirements document, a design document, or source code. Plain text is the simplest form of documentation, and is sufficient for code inspections, but design documents can also contain, for example, figures and tables. Structured document models can be used to document software artefacts, and to provide more specific metrics based on the meanings of the document particles, not only the rough positions of the defects. In addition, since there are many documents related to the actual artefact under inspection, the software inspection tool should also provide support for hyperlinks to help inspectors in their task.

Artefact management should integrate with other software development tools to ensure the effortless administration of the tool and the inspection. The tool should be aware of how the artefacts have been produced, where their revisions are located, and how they can be accessed. If the material has to be copied into a tool repository, there is a risk that the wrong version may be inspected or that two copies of the artefact will begin to evolve concurrently.

3.2. Defect management

The purpose of the software inspection is to locate potential defects in the artefact. Thus, defect management must be addressed carefully. The tool must support some kind of annotation, and the more structured the format is, the better. The way the tool stores and attaches defect data may vary. The tool may operate with a copy or some kind of image of the original artefact, and thus the exact location is lost. Ideally, the results should be linked to the original document in order to enable seamless analysis and to ensure that all the corrections will be made. The defect entries written down by the participants during the inspection should also contain information about the severity of errors, their classification, and references to checklist items.

Optionally, the tool may support discussion of the findings (commenting) so that inspectors may indicate that they found the same defect or state their agreement or disagreement. Furthermore, lists of findings help inspectors to become familiar with the artefact, and the
author to make corrections after the inspection. Ideally, the collected defect entries should be exported to the change management system.

3.3. Process management

Software inspection is a rigorous process. However, the classic Fagan inspection is not the only process model used in industry. Rather than a fixed process model, inspection tools should provide capabilities for customizing the process for an individual organization or project. When asynchronous working is used in a distributed environment, synchronous meetings are still sometimes needed. The workflow of the tool has to be flexible enough to support different types of inspection while still being rigorous. In addition, by assigning different roles to the inspectors or by exploiting, for example, perspective-based reading [1], the process can be made more effective.

The tool should inform inspectors about the state of the process. There should not be any need to copy or generate new workflow data for the tool, as the data should already be written into the project plan in the form of tasks and resources (people and time). If the tool can utilize the project management data, administration tasks will be minimized and the inspection leader will be able to focus on the inspection itself. In addition, information also travels in the other direction, because the project plan has to be updated as the inspection tasks are completed. Inspections usually produce new action points (for reworking), which it would be convenient to read directly from the inspection tool database and insert automatically into the task list in the project management tool.

3.4. Process improvement support

Software inspections are not only arranged to correct errors in documents, but to improve the software engineering process too. In addition, with the aid of detailed analysis of the metrics collected during the inspection, the inspection process itself can be made more effective. Gilb and Graham [4] list 42 metrics that should be collected during the inspection, and 14 variables that should be calculated. An inspection tool should unquestionably support these metrics and automate the collection, storage and analysis of the necessary data. For example, work effort estimations can be provided based on historical data. To encourage improvements in the process, it must be possible to calculate the derived metrics automatically, and the set of metrics must be flexible enough to focus on the most important aspects of a given situation.

In addition to built-in metrics, raw data with timestamps may be available for further analysis. Manual transfers are in any case laborious and unnecessary, as well as being error-prone. At the same time, they can cause the metrics to be buried somewhere in the tool database, because the time taken to handle the data is not thought valuable enough. Thus, the inspection tool should be integrated into generic repositories to enable the building of an organization-wide knowledge base.

3.5. Quality aspects

In addition to the functional requirements, there are quality aspects that the tool should fulfil. Safety is a critical aspect, and, at the very least, information security covering the artefacts and the defects detected must be addressed. The tool may exploit an external access control mechanism, or provide its own. However, integration with general authentication services is a good alternative. Installability is also a vital attribute. Some tools may be installed only partially in specific situations, or require modifications or extensive configuration to work correctly. Ideally, installation should be easy and straightforward.

Productivity and effectiveness are ultimate attributes describing if the tool is able and is practical to be used to support software inspections. Firstly, the tool must be suitable for the inspections, meaning that at least the focal task can be carried out with the tool. Secondly, the cost-effectiveness of the tool must be at least as good as in traditional software inspections.

4. Evaluation results

Here, the evaluation criteria are used to evaluate the four selected tools so that a comparison of off-the-shelf applications and dedicated software inspection tools is possible. For each tool, attempts are made to identify the feature specified by a given criterion using scenarios, and scores are assigned based on the success of this process. Each criterion has a specified acceptance threshold, and if a feature in a higher priority class fails to meet it, any lower-priority feature will not improve the final result for that category. For example, as part of the process management area, attempts were made to attach different roles to participants. If a tool supported some kind of predefined roles, it passed the acceptance threshold. Sophisticated tools supported various checklists, perspectives or scenarios, and it was possible to configure the contents of roles as needed. The more dynamic the role support, the more points the tool got. In Figure 1, the support profile for the combined set of tools is presented.
as a line graph, with bars providing the levels of support for each category of evaluation criteria.

4.1. Microsoft Word

The wide distribution of the Microsoft Office suite (http://office.microsoft.com/) has made the use of Word popular in generic document creation and editing. Its support for group working has developed over recent versions, and the latest version (2003) supports inspections with several features such as commenting, tasking, and online meetings. It should be noted that due to the nature of Microsoft’s products, many features that are relevant to inspection support are enabled in Word by utilizing other products, such as SharePoint Services, NetMeeting or Outlook. Therefore it is difficult to assess the inspection support of just Word, rather than the entire suite of tools provided with Windows (XP or 2000), Office (2003) and Windows Server 2003.

The feature scores for Word indicate that in most cases the support for inspections is on an adequate level. In the artefact management and quality aspects, all the features received scores above the acceptance threshold. In defect management and process management, all mandatory features (if any) and all but one mid-priority features met their acceptance levels. In the case of defect management, the lack of exportable summary reports was found to be a feature in need of improvement. With process management, the process model that Word imposes on inspections was found to be too predefined and not sufficiently customizable.

In light of the feature set used in the evaluation, the main problems in using Word with inspections would be in the area of process improvement support. The basic set of metrics required for process improvement support (time, size and defects) is only partially collected, which is deemed unsatisfactory. Lack of calculation for the metrics derived or statistical effort estimations from historical data follows from the inadequate collection of basic metrics. The only feature that achieves a sufficient level of support (though not necessarily as good as it could) is the support for timestamps in data.

4.2. Adobe Acrobat

Adobe Acrobat (http://www.adobe.com/products/acrobat/) is a famous document-producing tool for the Portable Document Format (PDF). All documents can be converted to PDF, and version 7 of the tool contains many useful features advertised as supporting peer reviews. Harjumaa [5] found that communication between team members was tedious and the lack of structure for defect information forced one to calculate metrics manually, but since then its development has paid special attention to collaboration and document sharing.

The features of Acrobat are quite similar to those of Word. Artefact management passed the acceptance level clearly, although the document under inspection is not the original one, but a copy or some kind of image of it. All important tasks could be performed with Acrobat as cost-effectively as in a traditional inspection, resulting in an adequate level in the quality aspects category. What is said about defect and process management in Word applies to Acrobat too. The tool is not designed to support software inspections, so the concepts are totally different and the tool does not directly provide a structure for defect data or workflow. Sophisticated process improvement based on metrics is not possible since only a minimal set of data is stored, and exporting the data for continued processing is not easy. However, new features make it possible to share documents and annotations effortlessly using, for example, web servers, network file systems, or email. This is a substantial improvement made after the experiment conducted by Harjumaa [5].

4.3. IBIS

IBIS [12] is specifically designed to support geographically dispersed inspection teams. It avoids synchronous activities and coordination problems by using its own process model that is slightly modified from that of traditional paper-based software inspection. The
functionality of the tool is implemented as dynamic web pages, and data is stored as XML documents.

The performance of IBIS shows that specialized features have their uses in software inspection. The tool stands out positively with the process management and process improvement support, but is quite self-contained and integration with other software engineering tools is not included. Inspector data and timetables have to be manually transferred from project plans, but the general format for the resulting data enables its analysis if needed. The biggest disadvantage is that the tool does not connect defects to the artefact; this work is left to the inspectors, who may write down the page and strict location on the paper-like form. Because attaching is suggested as a mandatory feature, IBIS gets very low scores in this category. Quality aspects are rated as high as for other tools.

4.4. XATI

XATI (http://www.tol.oulu.fi/i3/xati) is a new tool designed particularly with virtual software inspections in mind. Emphasis during development has been placed on interoperability and flexibility, as well as ease of use and deployment. XATI provides a web-based user interface and support for structured documents. It interacts with other software development tools and repositories through a single interconnectivity point, by fetching inspection data and providing inspection results.

Due to its ability to interoperate with other software development tools, XATI scores quite highly in the evaluation. Process management- and improvement-related inspector data, schedules and metrics can be derived effortlessly from project plans and exported for further analysis. However, it must be noted that the tool itself does not provide, for example, any kind of workflow support, which lowered its scores in the tool profile. This gives a huge amount of flexibility to the organization, but at the same time presupposes, or actually demands, that the software development environment contains the required features already. It seems that some of the external functionality should be a more integral part of the tool. XATI provides the most closely-integrated defect management, but on the other hand, practically forces one to use structured XML or at least HTML documents. The artefact management and quality aspects are at the same level as in other tools.

5. Discussion

The comparison between dedicated inspection tools and off-the-shelf document processing tools turned out to be very interesting. It seems that document processing tools are challenging the traditional inspection tools in many areas. Earlier studies [14, 5] found their suitability quite low, but since then the group work support of the applications has evolved. Both Adobe and Microsoft even talk about peer reviews in their advertisement material. The strength of document processing tools is their daily usage for other tasks, and thus their extensive support of different document formats. The inspection tools of web-generation emphasize the benefits of hypertext and structured documents, but still very few organizations exploit these features. Before these advantages can be capitalized on, organizational change must happen, so the evolution of the software inspection tools and process improvement based on metrics they provide is closely linked to the process which produces artefacts for inspection.

The main limitations of the off-the-shelf document producing tools are in the area of metrics collection and process improvement support. The manufacturers have identified the need to collect and manage comments from colleagues, but there is no support for the collection of even very basic metrics. However, document producing tools provide interfaces for extensions, which can potentially be used to implement missing functionality. For example, XATI uses the Mozilla/Firefox browser as a platform to display structured documents, on top of which the tool itself is built. It is worth investigating whether the document producing tools are as capable as current browsers of mutating into dedicated inspection tools.

The DESMET method was found to be useful, and aided the evaluation. It was found, however, that the method allows one too easily to focus on studying a tool from an integrated point of view, even if not directly encouraging this viewpoint. This is especially problematic in the design of the scoring scheme, where a lack of direct support for a feature is considered less desirable than an implementation of the solution in the evaluated tool. In some cases it could be more beneficial for the user of the tool to get the support for a feature from another tool, and thus interoperability between the tools should be of importance.

The feature set used in this evaluation had influences from (among other sources) ISO/IEC standard 9126, especially in quality aspects. It was found to apply less than ideally to feature analysis, since its focus is on measurements and quality metrics, as opposed to the functionalities and features that should be present in a given type of tool. Some of the metrics may, however, be applicable in developing the scoring scales for the features. The suitability of the standard in types of DESMET evaluations other than feature analysis should also be researched.
6. Conclusions

The preliminary evaluation model for a software inspection tool has been presented. It consists of criteria divided into five categories: artefact, defect, and process management, as well as process improvement support and quality aspects. The DESMET method used was found useful and aided the evaluation. In this paper, the focus was on the comparison of off-the-shelf document producing tools to specialized software inspection tools, which led to the selection of four tools: two from each category.

The tools fully achieve the acceptance levels in the artefact management and quality areas, meaning that inspections can be carried out with the aid of every one of them. The generic off-the-shelf document processing applications, Microsoft Word and Adobe Acrobat, had limitations in process improvement support, since they do not provide adequate support for metrics, which is, on the other hand, catered for in the dedicated software inspection tools, IBIS and XATI. Process management has been taken furthest in IBIS, which follows its slightly modified traditional software inspection process strictly, but does not provide the ability to adapt it to special needs. XATI left much of the process management to the other tools utilizing the interoperability framework. The generic applications required a good deal of creativity, and knowledge of software inspections, to be used according to an inspection process.

There is still a lot to do towards practical defect management. Generic applications provided good support for the most-used document formats, but there is still no way to define the structure for defect data. IBIS can display many documents in a browser window, but there is no connection between defect entries and the actual document. All locations have to be deduced and written down manually, which dramatically reduced the scores of this otherwise excellent tool. XATI follows the structure of the inspected artefact, but the requirement of using XML documents may limit its employment.

It is important to evaluate software inspection tools, since it seems there is still much to be done in that field. Furthermore, the potential of generic document producing applications must be taken seriously, since as this study points out, they provide many applicable features that even dedicated software inspection tools lack. In future, the evaluation framework should be made even more specific and used against a wider variety of tools. These results already give the developers and users of software inspection tools a lot to think about.

References


