UNIVERSITY OF JOENSUU
DEPARTMENT OF COMPUTER SCIENCE
Report Series A

sfia - Software for Inspection Automation

Vesa Tenhunen and Jorma Sajaniemi

Report A-2002-3

ACM D.2.5
UDK 681.3
ISSN 0789-7316
ISBN 952-458-114-0

sfia - Software for Inspection Automation

Vesa Tenhunen and Jorma Sajaniemi
University of Joensuu, Department of Computer Science,
P.O. Box 111, FIN-80101 Joensuu, Finland

March 11, 2002

Abstract

A key element in manufacturing quality software is the early detection of defects through various inspection techniques. Required bookkeeping is, however, laborious and automation is needed. We present a new inspection automation tool, *sfia*, that makes the collection of defect data easy and supports all phases of an inspection process. The tool can be used with any document format and is available on many computer platforms.

1 Introduction

A key element in manufacturing quality software is the early detection of defects throughout the whole software production life cycle. If defects are not removed as soon as possible they give rise to other defects which may be hard to trace and are expensive to remove. Early defect detection can be fostered by inspection techniques [2, 3] yielding an effective way to quality [1, 9]. Inspections are laborious, and bookkeeping makes them even more toilsome. This workload can be alleviated by automating the mechanical parts of inspection.

Software process improvement (SPI) is only possible if the state of a process can be measured and articulated in a rigorous way so that the effect of improvement acts can be evaluated [4]. Measurements should be automated as far as possible to get objective data and to relieve developers from extra work [6]. A good solution is to integrate measurement collection into the tools that are used for software production.

Inspection automation tools may provide a solution for both of the above problems. Existing inspection automation tools [5, 10, 8] have their limitations, e.g., no support for various inspection processes, various file formats, or various computer platforms. In this article, we will present *sfia* [11] (software for inspection automation) that allows one to inspect documents and to collect data from defects with ease. *sfia* works on Unix, Linux, Windows, and Macintosh.

Chapter 2 describes the inspection process with various phases and tasks, Chapter 3 gives an example of inspection with *sfia*, Chapter 4 presents the program's user interface and Chapter 5 evaluates the program by comparing it to other products.

2 Inspection Process

Inspection as a method for finding defects within software engineering process was first described by Fagan [2] in the early 1970's. Since then, inspections have become quite popular and they are used during the entire life cycle. Inspections have been tested empirically and they have been found to be an effective way to ensure quality [1, 9].

Despite some variations, most inspection processes still follow closely Fagan's original proposal. Fagan divides the process into five separate stages: (1) Overview, (2) preparation, (3) Inspection, (4) rework, and (5) follow-up.

The people involved are cast into well-defined and distinct roles: moderator, producer (or author), inspectors, reader, and recorder.

Usually the process goes as follows: In the overview, the moderator selects the inspection team members and assigns them their roles. The producer then introduces the product (code, document etc.) that is to be inspected.

During the preparation stage, the inspectors familiarise themselves with the material. In Fagan's original description, the inspectors are supposed to gain an understanding of the work – defect detection is only a by-product. In a variation suggested by Gilb and Graham [3], the inspectors should explicitly focus on finding defects.

In the inspection the moderator oversees the event; the reader (or the producer) paraphrases the document in full. The inspectors can stop the reader at any time and tell their comments which they have found in advance or at the meeting. All comments are written down by the recorder. No attempt to fix the defects is made – the inspection must focus solely on finding defects. The end product of this stage is a document containing all the found defects and problems.

It is also possible to have an inspection as distributed (non-local). The participants "meet" electronically using a suitable conferencing software to send their comments to each other. Another possibility is to replace the meeting by an asynchronous inspection, where the moderator oversees that the inspectors complete their tasks before the inspection moves forward [7].

The next stage is rework, where the producer reviews all defects. In the final stage, the follow-up, the moderator ensures that all comments are addressed and then decides whether there is a need for a full or partial re-inspection.

There are several goals in automating the inspection process. Among the most important ones are increasing rigour, effectivity, and efficiency [7]. The inspection automation tools can also help software process improvement by providing metrics of the inspected product and the inspection process itself.

3 Example of Tool Usage

sfia [11] (software for inspection automation) is a new tool designed to provide help in all aspects of comment handling during an inspection process. With *sfia*, different kinds of documents and source codes (which have been converted into graphics files in gif format) can be inspected in every phase of software engineering process. An inspector can add comments and set different values to them using classifications defined by the document's author. The moderator can monitor defect fixing and create summary reports, and the author can register which comments have been worked out. *sfia* doesn't limit the number of inspectors or comments or the kind of material to be inspected.

Here is an example of the use of *sfia* by various users in different phases of an inspection:

- The author converts pages of the document into gif files with her preferred tools and writes a text file, where she defines the contents of the document and the classifications for comments. The author sends the gif files and the definition file to inspectors.

- Each inspector goes through the document and adds comments as necessary. At the same time she marks the comment's type by selecting a suitable classification value. She can view the legends of the values in a separate window. Finally, she saves the comments and sends the resulting comment file back to the author.
- The author addresses each comment and marks them either as finished or notified, and adds her own comments in the comment texts, if necessary. By creating reports, the author can easily see the status and the amount of the comments.
- At the inspection meeting the author and the inspectors step through the document. The recorder modifies and deletes comments according to the decisions made and resets the comments' status. The recorder also creates reports about comments and their summaries. In the end, the comments are saved and handed back to the author.
- The author opens the document and marks all comments as unfinished. Then she addresses each comment, makes any necessary changes to the original document and marks the comments as finished. She can take care of one inspector's comments at a time by selecting whose comments she wants to be shown.

However, *sfia* doesn't force inspection processes to the form above. For instance, if the inspection meeting is not held, the author can make the appropriate changes as early as she processes the comments for the first time.

4 User Interface

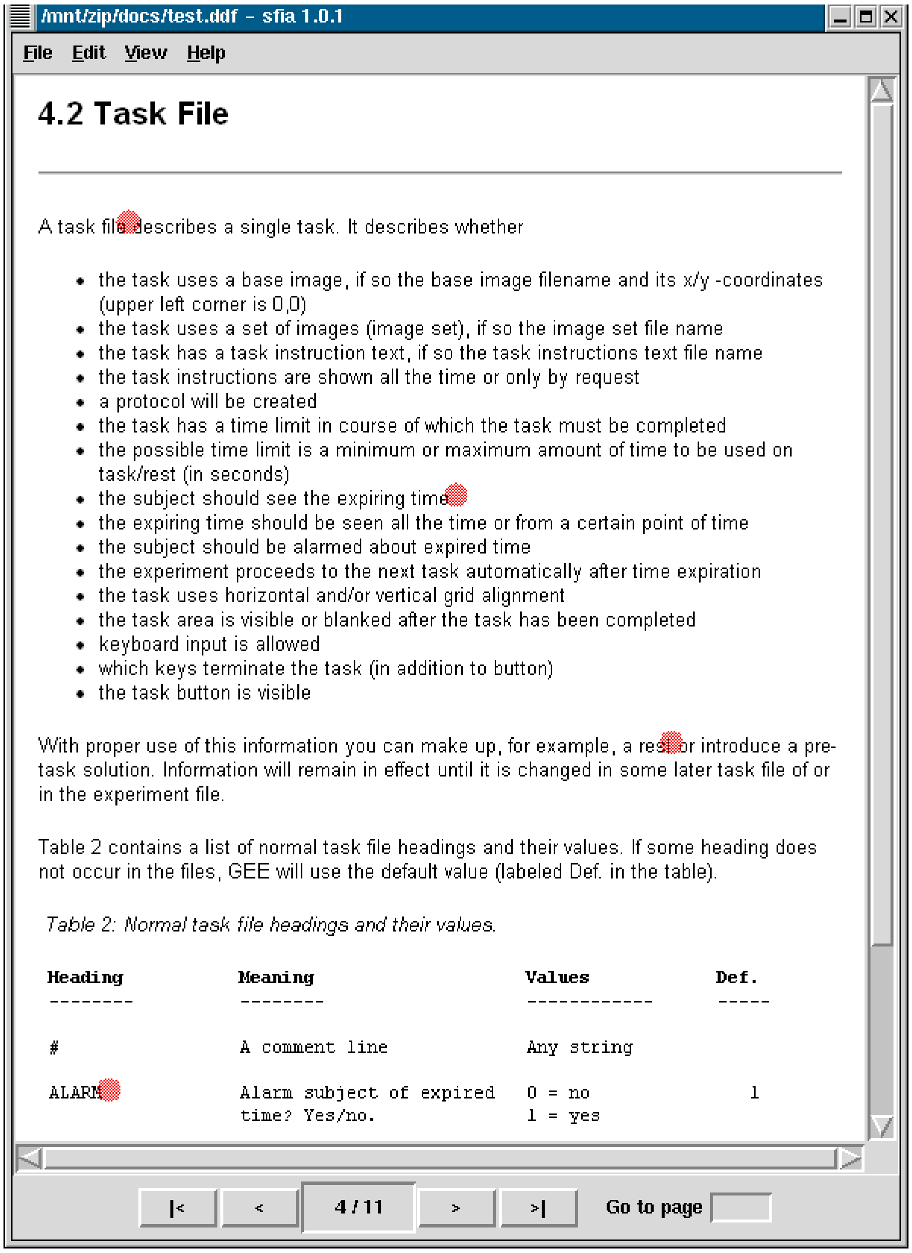
Figure 1 shows a document page with comment marks shown as coloured circles. A new comment can be added by double clicking the appropriate place in the page, which pops up an edit window, as shown in Fig. 2.

The edit window contains the inspector's name, current date, and page number of the comment. The inspector writes her comment in the text area and marks the defect classes with radio buttons. These classifications depend on the document definition composed by the author. If the user is the document's author, there is also visible a fixed classification of comment status, which is used to show whether the comment has been treated. Any number of edit windows may be visible at a time.

All comment texts together with inspectors' names can be seen in the comment list (Fig. 3). By clicking the page numbers or the comments, *sfia* shows the requested page in main window or opens the comment in edit window, respectively. *sfia* uses colours to emphasise the same comment in all three forms (i.e., the circle in the main window, comment text in the comment list, and all comment data in the edit window) when the mouse pointer hovers on them.

sfia produces two kinds of reports: one shows all comment data, and the other (shown in Fig. 4) shows summary information in a table form, categorised by defect classes and comment status.

sfia uses a protection mechanism based on user's status. Every user is, by default, an inspector and so can add, modify and delete her own comments, but only view those made by others. If the user is the author, she has rights to modify texts and statuses of all comments, but not their classifications. User in the role of recorder can pass protections and gain rights to modify every comments' text, status and classifications.



4.2 Task File

A task file describes a single task. It describes whether

- the task uses a base image, if so the base image filename and its x/y -coordinates (upper left corner is 0,0)
- the task uses a set of images (image set), if so the image set file name
- the task has a task instruction text, if so the task instructions text file name
- the task instructions are shown all the time or only by request
- a protocol will be created
- the task has a time limit in course of which the task must be completed
- the possible time limit is a minimum or maximum amount of time to be used on task/rest (in seconds)
- the subject should see the expiring time
- the expiring time should be seen all the time or from a certain point of time
- the subject should be alarmed about expired time
- the experiment proceeds to the next task automatically after time expiration
- the task uses horizontal and/or vertical grid alignment
- the task area is visible or blanked after the task has been completed
- keyboard input is allowed
- which keys terminate the task (in addition to button)
- the task button is visible

With proper use of this information you can make up, for example, a rest or introduce a pre-task solution. Information will remain in effect until it is changed in some later task file of or in the experiment file.

Table 2 contains a list of normal task file headings and their values. If some heading does not occur in the files, GEE will use the default value (labeled Def. in the table).

Table 2: Normal task file headings and their values.

Heading	Meaning	Values	Def.
#	A comment line	Any string	
ALARM	Alarm subject of expired time? Yes/no.	0 = no 1 = yes	1

Figure 1: Comments are marked with coloured circles in the document pages

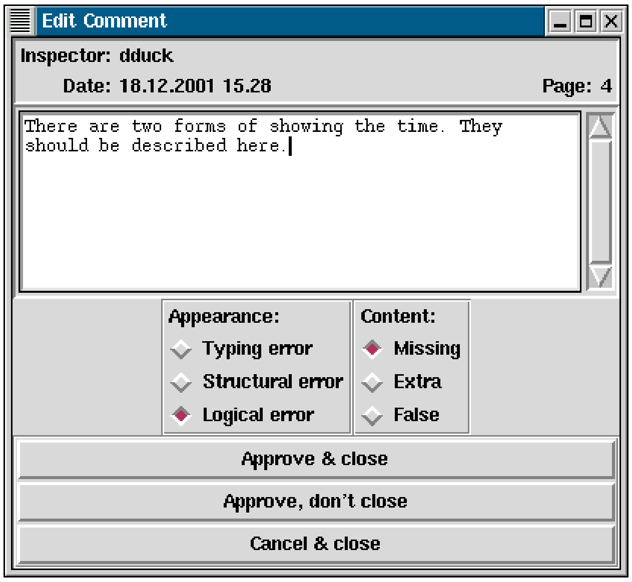


Figure 2: Comments are edited in a separate window

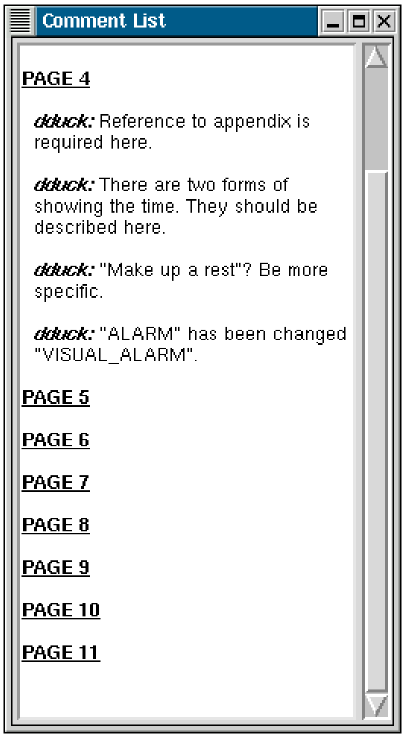


Figure 3: All comments are shown in the comment list

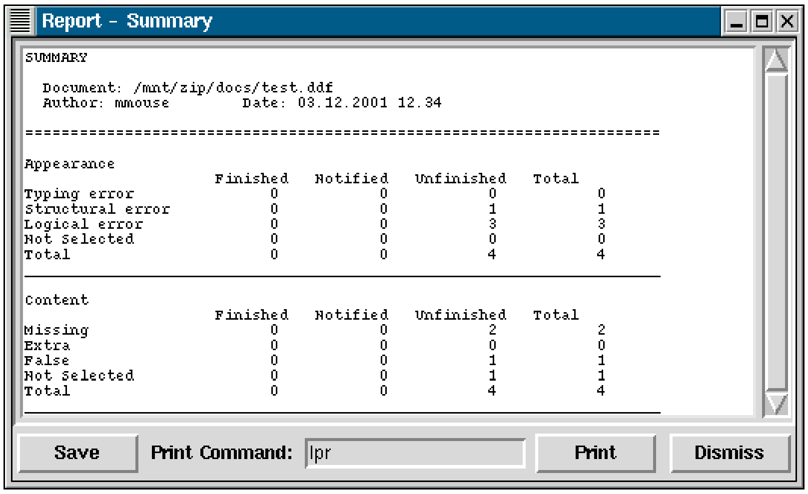


Figure 4: A summary report of the comments

5 Evaluation

Tenhunen and Sajaniemi [12] have suggested a framework for inspection automation tool evaluation that divides tool requirements in two dimensions: phase and viewpoint. The former deals with the successive phases of the process, and the latter considers various issues that must be dealt with during the process.

In this framework, *sfia* is found to work with virtually all systems and to provide all necessary support for comment handling. This includes, e.g., recording of comments with information about the comment's location, classification, explanation and references, and the ability to create summary reports as needed. However, *sfia* leaves process support and content-related issues like search and comprehension to be managed with other – content-aware – tools.

When compared with other inspection automation tools – CSRS [5] that is based on a special inspection method and supports plain ASCII files only; ReviewPro [10] that serves as a versatile note pad with a manual connection to the inspected material; Microsoft Word [8] that is a word processor with no special support for inspections – *sfia* is unique in its ability to accept any file format and its support for comment handling.

References

[1] Doolan, E. P.: Experience with Fagan's inspection method. Software – Practice and Experience, Vol 22, No. 2 (1990) 173–182

[2] Fagan, M. E.: Design and code inspections to reduce errors in program development. IBM System Journal, Vol. 15, No. 3 (1976) 182–211

[3] Gilb, T., Graham, D.: Software inspection. Addison-Wesley (1993)

[4] Humphrey, W. S.: Managing the software process. Addison-Wesley (1989)

[5] Johnson, P. M., Tjahjono, D.: Improving software quality through computer supported collaborative review. Proceedings of the Third European Conference on Computer Supported Cooperative Work (1993)

[6] Kitchenham, B.: Measuring software development. In Software Reliability Handbook, Ed. by P. Rook, Elsevier Applied Science (1990) 303–331

[7] MacDonald, F., Miller, J., Brooks, A., Roper, M., Wood, M.: Automating the software inspection process. Automated Software Engineering Vol. 3, No. 3–4 (1996) 193–218

[8] Microsoft Corporation. http://www.microsoft.com/office/word/

[9] Russell, G. W.: Experience with inspections in ultralarge-scale developments. IEEE Software, Vol. 8, No. 1 (1991) 25–31

[10] SoftWare Development Technologies (SDT). http://www.sdtcorp.com

[11] Tenhunen, V.: sfia – software for inspection automation. http://www.cs.joensuu.fi/pages/saja/se/sfia_intro.html

[12] Tenhunen, V., Sajaniemi. J: An evaluation of inspection automation tools. Accepted to Quality Connection, 7th European Conference on Software Quality, Helsinki Finland 9.-12.6.2002.