C. Wohlin, H. Petersson, A. Aurum, F. Shull and M. Ciolkowski, "Software Inspection Benchmarking - A Qualitative and Quantitative Comparative Opportunity", Proceedings 8th IEEE International Symposium on Software Metrics, pp. 118-127, Ottawa, Canada, June 2002.

Software Inspection Benchmarking - A Qualitative and Quantitative Comparative Opportunity

Claes Wohlin
Dept. of Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520, SE-372 25 Ronneby
Sweden
E-mail: Claes.Wohlin@bth.se

Aybuke Aurum
School of Information Systems, Technology and Management
University of New South Wales
Sydney NSW 2052
Australia
E-mail: aybuke@unsw.edu.au

Håkan Petersson
Dept. of Communication Syst.
Lund University
Box 118
SE-221 00 Lund
Sweden
E-mail: hakanp@tele-com.lth.se

Forrest Shull
Fraunhofer USA Center for Experimental Software Engineering Maryland
University of Maryland
4321 Hartwick Road, Suite 500
College Park, MD 20742-3290
USA
E-mail: fshull@fraunhofer.org

Marcus Ciolkowski
University of Kaiserslautern
Software Engineering Group
Postfach 3049
67653 Kaiserslautern
Germany
E-mail: ciolkows@informatik.uni-kl.de

Abstract

Software inspections are commonly used throughout the software industry, but there are still open questions about the relationship between inspection processes and inspection effectiveness. For example, which techniques work best in various environments? Are requirements specifications inspections and code inspections different in terms of effectiveness? What is the effectiveness in inspections for different group sizes? Benchmarking provides an opportunity to address such issues. This paper discusses how benchmarking may be applied for software inspections. The discussion is illustrated with an empirical study. It is shown how the data can be used to plan and manage software inspections. It is concluded that software inspections are well suited for benchmarking and that software practitioners as well as researchers can learn valuable lessons.

1. Introduction

Software defects are undesired consequences of the software development process. The objective of software inspections is to improve the quality of the product by analyzing the product, detecting defects and removing them before the product is released. It has been documented that software inspection reduces software development costs, increases the quality of software, and improves the productivity as well as the quality of decision making process for management [1, 2, 3]. Despite the documented benefits of inspection, the advantages of inspections are not well perceived by management. A number of social as well as technical factors influence the effectiveness of inspections [4] and comparative studies are necessary to isolate and understand those factors. One approach to doing this is the use of benchmarking, which is discussed in this paper.

It is important for managers and developers to be able to determine if and when an inspection process would be profitable in their environment. However, inspections are labor intensive, and the return from this process is not immediate and obvious. Therefore, there exists a perception that inspections cost more than they are worth. This means that it is valuable to use benchmarking to compare and evaluate different ways of doing inspections to know how to apply them in a cost effective way. The benefit of software inspections in different phases of the software life cycle can also be compared. Thus, inspections are particularly suited for benchmarking. This is because they are possible to characterize and describe and hence are comparable. From a research perspective, benchmarking should allow researchers to make more comprehensive studies and improve our understanding of how software inspections ought to be conducted. It is important to note that benchmarking is a continuous improvement process

rather than a competitive comparison.

This paper discusses the feasibility of software inspection benchmarking. The paper is outlined as follows. Section 2 describes some of the related work, both in benchmarking in general and software engineering in particular. In Section 3, the opportunities to use benchmarking in software inspections are discussed. An illustration of how benchmarking may be applied in software inspections is presented in Section 4. Finally, in Section 5 a discussion and some conclusions are presented.

2. Related work

2.1. Benchmarking in general

Benchmarking is a widely used business practice and has been accepted as a key component for organizations to search for improvement in quality, competitive position or market share. According to a survey in 1992, 31% of US companies were regularly benchmarking their products and services. Another survey in UK (1996) revealed that 85% of the business was using benchmarking practices [5]. In Japan, benchmarking is called “dantotsu”, which means, “striving to be the best of the best” [6]. Here, we would like to define benchmarking of processes as an activity that strives to be the best of the best. Thus, both qualitative and quantitative comparisons with this objective are viewed here as being benchmarking.

The literature describes several types of benchmarks [7, 5, 8]. Sole and Bist, in [7], point out that the level of benchmarking sets the degree of the challenge from a slight improvement in the development process to a radical change in the process. Benchmarking may be divided into different types depending on whom the comparison is made with and the objective of the comparison. Some common types of benchmarking include:

- comparison within the same organizations (internal benchmarking),
- comparison with external organizations (external benchmarking),
- comparison with competitors (industry benchmarking),
- identification of best practices (generic benchmarking),
- comparison of discrete work processes and systems (process benchmarking),
- comparison of performance attributes e.g. price, time to market (performance benchmarking), and
- addressing strategic issues (strategic benchmarking).

2.2. Benchmarking in software development

Benchmarking provides opportunities for comparison, for example, compilers may be compared by compiling the same program on several different compilers and by logging compilation time and errors.

Benchmarking in software development is perceived as an assessment method, which is concerned with the collection of quantitative data on topics such as effectiveness, schedules and costs [9, 10]. It allows the comparison between organizational processes with industry best practices. It also helps managers to determine whether a significant improvement is required to maintain a particular business [10]. Here, the term benchmarking is used both for qualitative and quantitative comparisons as long as the main objective of benchmarking is fulfilled. Thus, a characterization of several processes in qualitative terms would qualify as benchmarking if the objective is to improve these processes. Informally, the following definition of benchmarking is used in this paper. Process benchmarking is the comparison of similar processes in different contexts, and it implies multiple points of comparison, e.g. two data points is not a benchmarking, and it requires a representative sample in terms of, for example, organizations and applications.

Several assessment tools for software benchmarking have also been developed. Maxwell and Forselius [11] developed an Experience database that consists of 206 business software projects from 26 companies in Finland. This database allows managers to compare their projects with the existing projects from the database.

2.3. Benchmarking in software inspection

To the best of our knowledge little work has been done in the area of benchmarking software inspection. One of the few examples from the literature is where Jones [9] argues that function points provide useful metrics on two components of software quality: (a) potential defects, which is the total number of defects found in a work product, and (b) defect removal effectiveness levels, which is the percentage of software defect removed prior to delivery. He reports that in the US the average for potential defects is about five function points, and overall defect removal effectiveness is about 82%. According to one recent study, code inspection reduces life cycle defect detection costs by 39%, and design inspection reduces life cycle defect detection costs by 44% [12].

3. Inspection benchmarking

3.1. Introduction

To enable comparison between inspections, it is necessary to:

- Characterize each inspection process to be compared, including application, environment and people factors (qualitative benchmarking),
- Use comparable measures across inspections (quantitative benchmarking).

It is possible to perform a qualitative benchmarking. Proper characterization also yields the possibility of comparing inspection processes. The addition of measures

means that it is possible to quantitatively compare inspection processes. The measures to compare include effectiveness (how large a proportion of the defects did we find?), efficiency (how many defects did we find over a specific period of time? This is equivalent to defects found per time unit.), and the number of reviewers (how many reviewers do we need to achieve a certain level of effectiveness or efficiency?). It is also interesting to know how severe certain defects are to fully be able to determine effectiveness and efficiency in software inspections.

3.2. Characterization

A key aspect of benchmarking is the characterization, which can either be used as stand-alone qualitative benchmarking or as a part of a quantitative benchmarking, where the characterization can also be used to support identifying the correct quantitative comparisons. The characterization is important to allow for a comparison across different data sets and organizations. Moreover, it is important to enable evaluation of the potential differences observed in data sets, since it provides a baseline for, hopefully, understanding cause and effect relationships.

The characterization includes three perspectives and five aspects that are characterized, see Table 1. The characterization is based on the authors' experience from working with software inspections. The first perspective is the normal working situation, which should capture characteristics related to the working environment and the normal applications developed. The second perspective is related to the resources in the benchmark, i.e. the people participating in the benchmarking and the applied inspection process. The third perspective is basically a characterization of the unique aspects of the benchmark.

From an environmental point of view, it is important to document the type of inspection that is studied as well as the normal notation used in that phase. The type of application normally developed is important. This should not only include the application domain, and also some additional information, for example, whether it is a soft- or hard real-time system that is normally developed.

Next, the people participating in inspections have to be characterized. This includes their native language and experience, both in terms of the application domain and the environment. The inspection process has to be captured. It is important to collect as much relevant information as possible about the process. This includes the type of inspection (e.g. Fagan, or walkthrough), roles used in the inspection, techniques used for individual defect detection, if any, data collection procedure (for example comments are sent by e-mail or collected during a meeting), who is participating (both as reviewers and in any prospective meeting) and whether any tool support is used in the inspections. It is also essential to document how protocols are written and the procedure for re-work. Processes as applied may be different from processes as documented, meaning that ethnographic techniques may be appropriate.

Finally, it is important to document aspects that are related to the actual benchmarking and the artefact used in the benchmarking process. This includes the type of artefact, notation used and the number of defects in the artefact used for benchmarking. Preferably, the artefact is used as developed for the benchmark. In some cases this may be impossible, and it may be necessary to translate it. If this is the case, it has to be documented that it has been translated and also to which language. Since, the objective is to have a seeded artefact (with a known number of defects) for benchmarking, it is important to document the experience of the people in the benchmark application (if different from the normal application). To increase the value of the benchmark, it is important to document the difference in the inspection situation in comparison with how inspections normally are conducted. In other words, the distance from the normal artefacts and the normal inspection situation has to be captured. This is preferably done in a survey after having done the benchmark inspection. The distance should include as many aspects as possible, including application type, language, notation, complexity of the review artefact and also general opinions among developers. The developers could, for example, answer questions such as: Is this inspection representative? Is it easier or more difficult? Have you found more, less or an equal number of defects?

3.3. Quantitative measures

To obtain comparable measures regarding, for example, effectiveness in software inspections, it is necessary to list both the defects that were and were not discovered, as this is needed in order to determine the effectiveness. To achieve this, three opportunities exist:

- A seeded document, i.e. an artefact, is used experimentally to evaluate the inspection process. The document may be either from a generic or a company-specific domain. The advantage of having a document from a generic domain is that it makes comparison easier. The disadvantage is that the document may not give a good picture of how documents normally look in a specific organization. The company specific document may on the other hand make comparison more difficult.
- Defect tracking, i.e. the organization has such a good defect tracking system that it is possible to determine when a defect was introduced and found, and even when it could have been found. The latter means that the defect was present, but it was missed in the inspection. This type of in-depth tracking would allow for determining the number of defects actually present in a specific document, although it will take some time until almost all defects present at a specific inspection have been located.

Table 1. Characterization of software inspection.

Normal working situation		Resources in the benchmark		Unique aspects of the benchmark
Environment	**Application**	**People**	**Process**	**Benchmarking**
Phase	Domain	Native language	Inspection type	Artefact type
Normal notation		Experience in application	Roles	Artefact notation
		Experience in environment	Individual defect detection technique, e.g. reading technique	English or translated
			Meeting	Number of known defects
			Tool support	Experience in benchmark application
			Protocol	Distance from normal artefacts
			Procedure for rework	

- Defect estimation methods, i.e. by applying defect estimation methods after an inspection, it is possible to get a value for the total number of defects in the inspection. If this type of estimate is trusted then it is possible to determine the effectiveness of inspection. This type of defect estimation methods includes capture-recapture and curve-fitting methods [13].

Defect tracking is difficult and hence many organizations are unable to perform tracking according to the above description. Also few companies are using defect estimation methods and hence the focus here is on benchmarking using one or several seeded artefacts. Thus, seeded documents are probably the best option.

The seeded documents may be from any application domain. In case of a standardized artefact, it is preferable to find an area, which is familiar to people, however few developers have actually developed systems in this area. Examples of systems may include an elevator or a reservation system for taxis. Most people have an intuitive feeling for these types of systems, although most developers have not developed systems in these application domains. Systems, for example, in the telecommunication domain are probably not suited since some of the software is hard to understand unless you have worked in the area. It also gives a major advantage to those who have worked with these systems. This makes comparison and hence benchmarking difficult.

Another aspect of the artefacts is the phase they represent. As a first step a requirements specification and code inspections could be benchmarked, since several experiments have been conducted on reviewing requirements specifications and code (see Table 1), and hence baseline data already exist. In addition to this the approach may be extended to other artefacts in the future. The requirements review is especially useful when the specification is written in natural language and hence readable for most developers, i.e. they do not need to have any knowledge in a specific high-level language. Code is also readable for developers even if they are not experts in that specific programming language, but the use of Java, C or C++ allows more developers to be familiar with the language.

3.4. Benchmarking goal

The goal is to be able to compare different software inspection processes. Given the characterization and the standardized artefacts, it is possible to compare, for example, if a specific inspection process is better or worse than another.

Some key concerns regarding benchmarking are scalability, thresholds, simplicity and atypical situations. Scalability should not be a major problem, as long as the inspections scheduled on real projects are of limited size. Since normal recommendations on the length of an inspection meeting are on the order of 2 to 4 hours it is

feasible to benchmark a realistic approximation of the process. Scalability must also be addressed by using documents representative of what is normally seen at the organization, with respect to size and defect density. Then, the objective is not to set quality thresholds on the documents but to provide feedback on effectiveness, efficiency and expectations on these two aspects in terms of group size. Simplicity is also very important because in order to make a benchmark useful, it should be possible to replicate it without having to have a number of experts present. The characterization scheme supports simplicity. Finally, it is often heard about software projects that they are so different from each other that it is not possible to compare them. It may be true that projects are very different, but it should still be possible to compare certain aspects of software projects, for example, software inspections. The differences and similarities should be captured in the characterization and hence atypical inspections should be accounted for in the analysis. Atypical inspections may be important to learn from, but they should not be part of the normal benchmarking data, since it is not anticipated that the same situation will occur again.

4. Illustration

To show the opportunities with benchmarking in software inspections, an empirical study is presented. The study is based on publicly available data, and the main objective is to illustrate how software inspection data may be used to evaluate several important questions regarding software inspections. It must be noted that unfortunately characterizations of the different contexts for the individual data sets are not available and hence the actual empirical results have to be used with caution. Moreover, it is only possible to compare the effectiveness in the inspections, since for most studies there is no information available on time spent and defect severity.

In particular, the results are based on data that was not collected for benchmarking so the actual results are not the main issue here. The key issue is to illustrate how this type of information can be used for benchmarking purposes, if an appropriate characterization is conducted.

The study may be viewed as an attempt to carry out meta-analysis, which means that results from different studies are combined to generate new knowledge. It is acknowledged in the software engineering literature that meta-analysis is needed [14, 15, 16]. In [14], it is stressed that meta-analysis based on raw data is most appropriate. This is the case in our study. Miller [15] discusses when data from experiments can be safely combined. The paper discusses defect detection experiments, which includes both testing techniques and different reading techniques. Hayes [16] takes a similar approach and compares five published studies. A major difference in our study is that the meta-analysis is based on more than 30 data sets and that the analysis is done based on the raw data as mentioned, [14], as one of the most appropriate methods for performing meta-analysis.

4.1. Data

The data in Table 2 is collected from literature and it is actual data from companies or university environments.

The data has primarily been collected as part of formal experiments [17]. For the sake of this illustration, let us imagine that the data is collected from different companies. This is done for illustrative purposes to show the feasibility and opportunity of doing benchmarking in software inspections.

In total, there are 21 data sets from the requirements phase and 10 data sets from code inspections. From the data, all virtual groups are created to obtain as much information as possible from each data set. This means, for example, that 35 “inspection groups” are created with three reviewers from data sets with seven reviewers in total. For the data available, groups of sizes 1-7 have been created to illustrate the opportunities. It should be noted that in the long run, the aim should be to base the benchmark only on real groups to ensure that the conclusions are based on groups that are similar to the ones found in industry.

The disadvantage of virtual groups is that there is a high dependency between the groups, but on the other hand all data sets are treated the same and since the main concern is comparison, it should not be critical for the outcome. It is here worth noting than a preliminary simulation has been conducted to evaluate the use of virtual groups. The outcome indicates that the mean values are approximately the same for virtual groups and real groups. However, the variation among real groups is larger, which is reasonable given the dependence between virtual groups.

A random selection of all combinations is used so that all data sets get a similar weight, otherwise data sets with more reviewers would dominate over the others. It is now possible to create a “database” with experiences from these companies.

The characterization of the data sets is shown in Table 2. For simplicity the data is only characterized regarding the development phase (requirements, code) and reading technique (ad hoc, checklist, perspective-based reading [18]). However, due to the use of virtual groups, the evaluation is not really done of perspective-based reading, since this type of reading technique requires different roles in a group. This is not fulfilled when using virtual groups. Thus, the technique is hereafter referred to as active reading technique, since in all roles the individuals are active, for example, with creating use cases or test cases.

The data provides opportunities to make controlled comparisons to evaluate if, for example inspection rates vary by the development phase or reading technique. Using the data, it is possible to answer, for example, the following benchmark questions:

1. Are there any differences in terms of effectiveness between requirements specification inspections and code inspections?
2. Are there any differences in terms of effectiveness between the different reading techniques?
3. How many inspectors does one need to exceed, for example, 60% effectiveness with a certain probability, for example, 70%?

Table 2. Classification of Data Sets.

No.	No. of reviewers	Doc. Type	Reading Tech.	Reference to source
1	8	Artif. Req[a]	AdH.	Freimut, 1997
2	6	Artif. Req	AdH.	Freimut, 1997
3	6	Artif. Req	AdH.	Freimut, 1997
4	6	Artif. Req	AdH.	Freimut, 1997
5	6	Artif. Req	Chkl	Unpublished[b]
6	7	Req	AdH.	Freimut, 1997
7	6	Req	AdH.	Freimut, 1997
8	6	Req	AdH.	Freimut, 1997
9	6	Req	AdH.	Freimut, 1997
10a[c]	8	Artif. Req	PBR	Regnell 1999
10b	7	Artif. Req	PBR	Regnell 1999
11a	8	Artif. Req	PBR	Regnell 1999
11b	7	Artif. Req	PBR	Regnell 1999
12	6	Artif. Req	PBR	Freimut, 1997
13	6	Artif. Req	PBR	Freimut, 1997
14	6	Req	PBR	Freimut, 1997
15	6	Req	PBR	Freimut, 1997
16	7	Req	PBR	Freimut, 1997
17	6	Req	PBR	Freimut, 1997
18	8	Artif. Req	PBR	Freimut, 1997
19	6	Artif. Req	PBR	Freimut, 1997
20	8	Code	PBR	Freimut, 1997
21	7	Code	PBR	Freimut, 1997
22	8	Code	PBR	Freimut, 1997
23	7	Code	PBR	Freimut, 1997
24	8	Code	PBR	Freimut, 1997
25	7	Code	PBR	Freimut, 1997
26	5	Code	Chkl	Runeson, 1998
27	5	Code	Chkl	Runeson, 1998
28	5	Code	Chkl	Runeson, 1998
29	5	Code	Chkl	Runeson, 1998

a. Artificial requirements specifications, i.e. they have been developed for the experimental studies and they are not “real” specifications. The artificial requirements specifications are treated together with the requirements specifications coming from real software projects.

b. Used in [19] though the data set is not published.

c. The data sets 10 and 11 have been divided into two parts (denoted by “a” and “b” respectively) by random to make them more comparable in size (in terms of number of reviewers) to the other studies.

4. Another opportunity is to assume that company No. 19 and 29 are not in the database, but they would like to compare their inspections with the ones in the database. Thus, it is possible to use the data from the other companies and compare this with these to two companies for requirements and code inspections respectively.

These types of questions and studies can be conducted as more and more data becomes available.

4.1. Analysis

The four questions above are addressed to illustrate how software inspection benchmarking can be used to study a number of issues of interest to software management in their effort to improve the software development process. It is here assumed that all data sets used for comparisons come from “similar” companies, i.e. the characterization

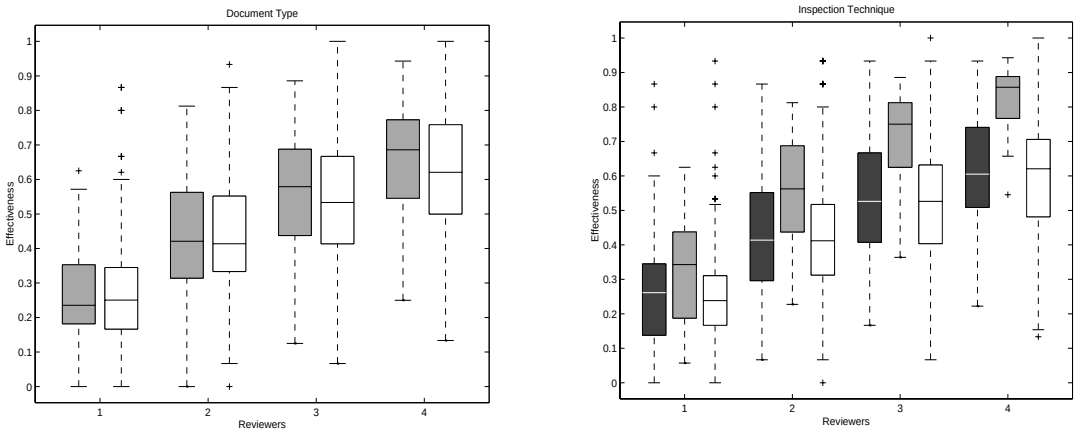

Figure 1. Effectiveness in inspections for different types of documents (left) and for different reading techniques (right). The documents are from the left: requirements specification and code. The reading techniques are from the left: ad hoc, checklists and active reading technique.

shows that it is reasonable to compare the companies.

1. Effectiveness in requirements specification and code inspections respectively

   To the left in Figure 1, the effectiveness in software inspections is shown for different types of documents. The box plots indicate the median value as a line in the box. The box extends from the 25th percentile to the 75th percentile, and the whiskers are determined based on these quartiles and the length of the box multiplied with 1.5. The box plots are shown for 1 to 4 reviewers with requirements specifications inspections to the left and code inspections to the right for the different cases. The focus is on 1 to 4 reviewers since the number of combinations is very few for higher number of reviewer, and the results would depend too much on single data points rather than representing a more general outcome.

   From the box plots, it seems obvious that the differences in terms of effectiveness between requirements specifications inspections and code inspections are minor. To test that there is no significant difference (level 0.05), the non-parametric Mann-Whitney test was applied. The outcome was, as expected, that no significant difference between the two inspection types could be identified. From a benchmarking perspective, this means that we may conclude that we can expect that our effectiveness for different types (or at least for requirements specification inspections and code inspections) of inspections is approximately the same. This information is valuable when planning different types of inspections. In particular, it implies, for example, that any experience regarding effectiveness for code inspections could probably be transferred to inspections of requirements specifications.

2. Effectiveness in inspections using different reading techniques

   To the right in Figure 1, the box plots for the effectiveness for different reading techniques is shown. Once again, the plots provide information for 1 to 4 reviewers with the plots in the following order: ad hoc, checklists and active reading technique. From the box plots, it seems as though checklists may be more effective than the other two techniques. To test this, both a non-parametric Kruskal-Wallis test and a parametric ANOVA test were applied. Both tests showed that there was indeed a statistically significant (level 0.05) difference between some of the reading techniques. A more in-depth analysis showed that the difference was between checklists and the two other reading techniques. From a benchmarking perspective, this tells us that if we have good checklists they ought to outperform ad hoc inspections and the active reading technique. The latter is fairly surprising and from a research perspective. The result should encourage us to further investigate the reasons for this outcome and also to further develop other techniques so that more effective inspections can be obtained.

3. Team size versus effectiveness

   In Table 3, the probability for a certain effectiveness given a certain team size is shown. The table is built from the available data sets, and hence reflect a view across the data sets presented in Table 2. For example, the probability for having a higher effectiveness than 50% with 3 reviewers is 0.57 (see shaded cell). Thus, the columns represent the group size, the rows the effectiveness and the cells in the table the probability for exceeding the effectiveness with the given group size.

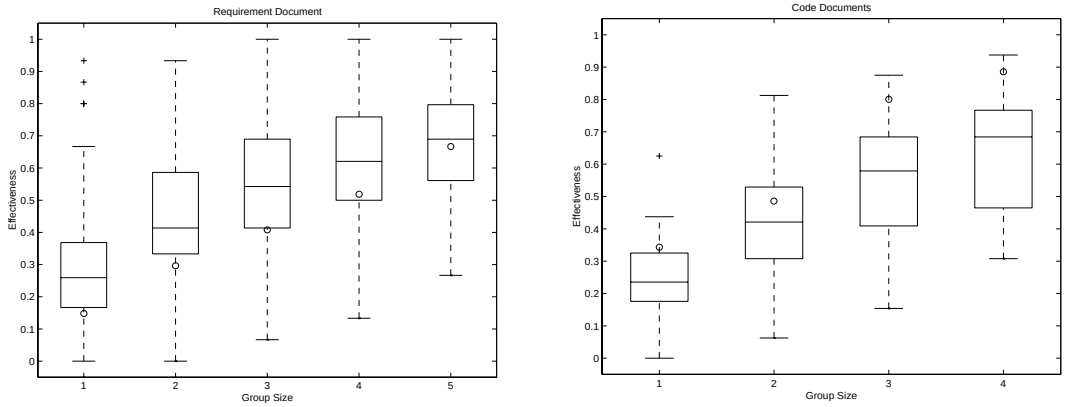

Figure 2. New company scenario for requirements specifications inspections (left) and code inspections (right).

Table 3. Teams size versus effectiveness.

	1	2	3	4	5	6	7[a]
0.1	0.87	0.97	0.99	1.00	1.00	1.00	1.00
0.2	0.64	0.90	0.96	0.99	1.00	1.00	1.00
0.3	0.37	0.75	0.89	0.93	0.97	1.00	1.00
0.4	0.17	0.57	0.77	0.84	0.87	0.91	1.00
0.5	0.10	0.35	0.57	0.74	0.81	0.86	0.95
0.6	0.06	0.22	0.41	0.57	0.65	0.77	0.71
0.7	0.03	0.10	0.23	0.39	0.43	0.55	0.50
0.8	0.03	0.06	0.15	0.24	0.22	0.27	0.27
0.9	0.01	0.02	0.04	0.08	0.11	0.15	0.05
1.0	0.00	0.00	0.00	0.01	0.03	0.04	0.00

a. The lower effectiveness for seven reviewers than for six reviewers is due to the data sets and random effects. For the higher number of reviewers, we have fewer data sets and it should also be noted that the actual combinations making up the table is selected by random, and hence this effect occurs.

The third questions stated in the introduction of Section 4 may now be answered. If asking for an effectiveness level above 0.6 with a probability of 0.7 means that at least 6 reviewers are needed (see black cell with white text). This table provides an opportunity to understand how to plan the number of reviewers to achieve a certain effectiveness with a certain probability. This type of information is very valuable for anyone planning and managing software inspections. In the long run, a table like the one in Table 3 ought to be constructed by organizations that want to improve their ability of planning software inspections.

4. New company scenario

It is here assumed that data sets 19 and 29 are not part of the experience base, and the box plots in Figure 2 are created without these two data sets. In particular, it is assumed that the companies that we compare with are a subset of the companies in the experience base and that they were selected based on the characterization. Thus, the box plots may be seen as the benchmarking experience base that new companies may use for planning and comparing their own software inspection process. The

circles in the box plots represent the two new data sets (or companies from a scenario perspective).

Company No. 19 could use the left diagram in Figure 2 to know what to anticipate when performing inspections. It is possible to see the median value for requirements specification inspections as well as the different quartiles and whiskers. Thus, the company has a good picture of the industry standard for effectiveness of requirements specifications inspections. After having conducted a controlled inspections at the company, it is possible to plot how this company is performing. This is illustrated with the circles in the box plot. It can be seen that company No. 19 in this case performed below median for all group sizes. The difference between how close the circle is to the median depends on the individuals that are added as we increase the group size. It may be particularly interesting to actually study the performance of individuals (although personal integrity has to be taken into account) and hence use this information to put together inspection teams.

To the right in Figure 2, a similar diagram is shown for code inspections. In this case, it is worth noting that company No. 29 is performing better than the industry standard.

In summary, the illustration above shows examples of questions that can be addressed if using benchmarking in software inspections. The actual figures are not the main issue. The illustration is based on the assumption that the data sets are indeed comparable, i.e. the characterizations of the data sets are similar. Anyhow, the figures indicate the level of effectiveness that we can anticipate for different reading techniques, different types of documents and different group sizes.

5. Conclusions

Software inspection benchmarking opens a number of interesting opportunities. The need for empirical studies and experimentation in software engineering is well known. Software inspection is an area where experimentation really is possible. This includes both industry and universities. By agreeing on a number of standardized artefacts or use of company specific artefacts and a characterization schema, it should be possible to a large number of experiments worldwide and hence it should be possible to progress fairly quickly in this area.

Software inspection benchmarking is interesting for both universities and industry. Industry may perform benchmarking as discussed above. Universities may perform experiments allowing for more data to be collected regarding inspections, which would enable us to better understand different aspects of software inspections more fully. This becomes particularly valuable if the universities use the same documents as are used in the benchmarking in industry. For example, universities should be able to include experiments in courses where different reading techniques are compared.

The future work includes a worldwide initiative through the International Software Engineering Research Network (ISERN) to conduct a series of studies to build a software inspections benchmarking experience base. The objective is to provide support to companies along the lines discussed in the illustration of how benchmarking can be used, and to provide researchers with more comprehensive results to build their future efforts on.

6. Acknowledgment

The authors would like to thank Prof. Vic Basili, Prof. Dieter Rombach and Dr. Martin Höst for valuable discussions on the subject of software inspection benchmarking, which have contributed to the ideas presented in this paper.

7. References

[1] M. E. Fagan, “Advances in Software Inspection”, *IEEE Transaction on Software Engineering*, 12(7), July., 1986.

[2] F. A. Ackerman, L. S. Buchwald and F. H. Lewisky, “Software Inspections: An Effective Verification Process”, *IEEE Software*, May 1989, pp. 31-36.

[3] T. Gilb and D. Graham, *Software Inspection*, Addison Wesley Publishing Company, ISBN 0-201-63181-4, 1993.

[4] C. Sauer, D. R. Jeffrey, L. Land and P. Yetton, “The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research”, I*EEE Transactions on Software Engineering*, 26(1), 1998, pp. 1-14.

[5] P. K. Ahmed and M. Rafiq, “Integrated Benchmarking: A Holistic Examination of Selected Techniques for Benchmarking Analysis”, *Benchmarking for Quality & Technology*, 5(3), 1998, pp. 225-242.

[6] L. M. Corbett, “Benchmarking Manufacturing Performance in Australia and New Zealand”, *Benchmarking for Quality Management & Technology*, 5(4), 1998, pp. 271-282.

[7] T. D. Sole and G. Bist, “Benchmarking in Technical Information”, *IEEE Transactions on Professional Communication*, 38(2), 1995, pp. 77-82.

[8] D. Longbottom,“Benchmarking in the UK: An Empirical Study of Practitioners and Academics”. *Benchmarking: An International Journal*. 7(2), 2000, pp. 98-117.

[9] C. Jones, “Software Challenges”, *IEEE Computer*, 28(10), 1995, pp. 102-103.

[10] A. Beitz and I. Wieczorek, “Applying Benchmarking to Learn from Best Practices”, Proceedings 2nd International Conference on Product Focused Software Process Improvement, Oulu, Finland, 2000.

[11] K. D. Maxwell and P. Forselius, “Benchmarking Software Development Productivity”, *IEEE Software*, January/February 2000, pp. 80-88.

[12] L. Briand, K. El Emam, O. Laitenberger and T. Fussbroich, “Using Simulation to Build Inspection Efficiency Benchmarks for Development Process”, Proceedings of 1998 International Conference on Software Engineering, pp. 340-349.

[13] C. Wohlin and P. Runeson, “Defect Content Estimations from Review Data”, Proceedings International Conference on Engineering, pp. 400-409, Kyoto, Japan, 1998.

[14] L. M. Pickard, B. A. Kitchenham, and P. W. Jones, “Combining Empirical Results in Software Engineering”, *Information and Software Technology*, 40, 1998, pp. 811-821.

[15] J. Miller, “Can Results from Software Engineering Experiments be Safely Combined?”, Proceedings IEEE International Software Metrics Symposium, pp. 152-158, Boca Raton, Florida, USA, 1999.

[16] W. Hayes, “Research Synthesis in Software Engineering: A Case for Meta-Analysis”, Proceedings IEEE International Software Metrics Symposium, pp. 143-151, Boca Raton, Florida, USA, 1999,

[17] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell and A. Wesslén, *Experimentation in Software Engineering - An Introduction*, Kluwer Academic Publishers, Boston, USA, 2000.

[18] V. R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sørumgård and M. V. Zelkowitz, “The Empirical Investigation of Perspective-Based Reading”, *Empirical Software Engineering: An International Journal*, 1(2), 1996, pp. 133-164.

[19] B. Regnell, P. Runeson and T. Thelin, “Are the Perspectives Really Different? - Further Experimentation on Scenario-Based Reading of Requirements”, *Empirical Software Engineering: An International Journal*, 5(4), 2000, pp. 331-356.