

Using Group Support Systems for Software Inspections

Michiel van Genuchten, *GroupSupport.com*

Cor van Dijk and Henk Scholten, *Baan*

Doug Vogel, *City University of Hong Kong*

Software engineering sometimes appears to be years behind other disciplines in terms of predictability and quality. But we like to argue that the software industry is facing problems now that other industries have yet to face. Constructing a high-quality, million-line program is daunting. Fortunately, progress is occurring and merging in two important areas: software process improvement and technological support.

One example of software process improvement is the development and execution of inspections of software development documents as a mature technique to detect and prevent software defects. Part of an inspection is a meeting where the participants log detected defects and search for more defects. Unfortunately, the effectiveness and efficiency of the logging meeting are often low compared to those of the preparation for the meeting. This has led software engineers and researchers to debate whether an inspection even needs a meeting.¹

We approach this question from another angle: can we improve logging meetings such that they contribute to more effective and efficient inspections? Improving meetings through technology has been the main goal of the researchers and practitioners working on electronic meeting systems or *group support systems* over the last 20 years.^{2,3} Companies have successfully used these tools for a wide variety of problem-

solving and business process activities—for example, Doug Vogel has described IBM's use of an electronic meeting system.⁴ For more on addressing requirements negotiation, see “Developing Groupware for Requirements Negotiation: Lessons Learned” by Barry Boehm, Paul Grünbacher, and Robert O. Briggs, in this issue.

This article describes our experience implementing a GSS for inspections in an industrial environment. The results confirm our belief that such support can improve the efficiency and effectiveness of inspections, provided the inspections are properly conducted.

Why GSS support for inspections?

An inspection uses standards and checklists to find and fix as many product and process deficiencies as possible. Key characteristics of inspections are individual preparation, data collection, and a fixed syntax to report defects. For those unfamiliar with

Inspections supported by a group support system can help software professionals detect software defects effectively and efficiently.

Inspections

inspections or GSSs, the two sidebars briefly introduce both subjects.

During a traditional logging meeting, the participants review the document and state the defects aloud. The reported defects function as triggers to detect more defects. The moderator leads the meeting, and the scribe records all the defects. Experienced groups claim to find many defects during a meeting.

You could say that this is a primitive way to consolidate defects. The logging meeting's two goals (reporting errors and finding new ones) often conflict. The moderator is usually busy controlling the discussion and the logging rate. Inspectors should take the initiative to find new defects, but they are often distracted by other reported defects. This situation suggests that introducing technology into logging meetings might overcome some of these problems.

There have, indeed, been several attempts to support software inspections with automated tools. Ilkka Tervonen describes a number of experiments with students, mainly using university prototypes.⁵ We were also involved in some experiments with GSS-supported inspections with students.⁶ We concluded that measuring the impact of GSS support is difficult when the group of inspectors is neither trained as inspectors nor experienced in working as a group.

As a prelude to operational use, we did two pilot studies at Philips Electronics and Baan in 1996, involving 14 inspections.⁷ The results in both companies showed that inspectors found more defects and that the logging meeting contributed more to the inspection's overall result. Furthermore, the engineers appreciated GSS support in the logging meeting. The results triggered the large-scale application of GSS technology at Baan, which we now describe.

Applying a GSS at Baan

Baan Development is a medium-size industrial software organization developing enterprise resource planning software. GSS-supported inspections started in Baan Development's Applications Department in the Netherlands, which had 200 engineers. The department had been executing inspections on software documents since 1993. Inspections typically involved three to four engineers, one of whom acted as moderator.

Inspections are a structured way to review software development documents (specifications, the design, or code) with a group of approximately four engineers. Engineers around the world have carried out inspections for more than 20 years. Inspections are widely acknowledged as an important technique to improve software products and processes. An inspection consists of individual preparation by the participants, followed by a meeting in which they log defects and look for more defects, and then reworking of the document by its author. An inspection aims to detect software defects early during development. Inspection results are typically measured in terms of the major and minor defects (*majors* and *minors*) that the inspection detected. Majors would, if undetected, result in a defect in test or in the field. Minors are all other defects.

Inspection results typically involve three performance indicators. *Effectiveness* is the number of defects detected per page.¹ *Efficiency* is the number of defects detected per person-hour invested. Effectiveness and efficiency can be calculated for both the preparation and meeting. *Yield* is the fraction of defects that the inspection detected, as opposed to those that "escape" and are caught in later development phases, in tests, or in the field.²

Well-executed inspections should be able to find 60 to 80 percent of the life-cycle defects before the software goes into test. For example, in a project in which one of the authors was involved, inspections of the specifications, design, and code found 1,170 defects, while the various tests found only 825 defects.³ Mature software groups spend 10 to 20 percent of their resources on inspections. More information on inspections and their results is available elsewhere.^{1,4,5}

References

1. T. Gilb and D. Graham, *Software Inspections*, Addison-Wesley, Reading, Mass., 1993.
2. W.S. Humphrey, *A Discipline for Software Engineering*, Addison-Wesley, Reading, Mass., 1995.
3. J. Rooijmans, H. Aerts, and M. Genuchten, "Software Quality in Consumer Electronics Products," *IEEE Software*, vol. 13, no. 1, Jan. 1996, pp. 55-64.
4. M. Fagan, "Advances in Software Inspections," *IEEE Trans. Software Eng.*, vol. 12, no. 7, July 1986, pp. 741-755.
5. W.S. Humphrey, *Managing the Software Process*, Addison-Wesley, Reading, Mass., 1989.

The GSS inspection process

Baan piloted GSS support for inspections in 1996 and made it available to the engineers as an option in 1997. They could use it in the meeting room only, or in both preparation and the meeting room. We'll call this *GSS inspection* to differentiate it from traditional *paper inspection* (where the scribe records on paper all the detected defects).

GSS inspection's main difference from paper inspection was that engineers electronically delivered the defects they found in preparation, either by email to the moderator or directly into the GSS from their desks. The scribe's and moderator's roles also changed. In a paper inspection, one of the inspectors typically was also the scribe. The GSS made the scribe redundant, relieving one engineer of a boring task. Previously, the moderator spent time preventing two people from speaking at the same time. However, GSS inspection allowed such communication because people

Group Support Systems

GSSs, a form of groupware, have emerged over the past decade and increasingly are being used in a wide variety of business and government organizations. For example, GroupSystems is a GSS developed at the University of Arizona and subsequently commercialized by GroupSystems.com. It provides an integrated set of tools that groups use to generate information, dynamically share comments, classify and prioritize items, and perform a wide variety of other group support functions. Participants can interact both through computers linked on a network and verbally. GSSs (including GroupSystems) can also be used where all participants are not colocated in space or time. As such, GSSs become mechanisms for supporting and sustaining team activity in distributed organizational contexts.

GSS use in organizations has focused on support for a variety of problem-solving and planning activities (for example, quality improvement teams) as well as for business process improvement.¹ GSSs succeed by minimizing the process losses (for example, air time and participant apprehension) and seeking to maximize the process gains (for example, participant synergy) that impact groups.² In a GSS setting, group dynamics play an important role in structuring and electronically supporting group tasks. For example, GSSs support anonymity when participants might be apprehensive about speaking their opinions in the presence of superiors. Organizations report dramatic time savings when using GSSs—for example, over 50 percent in people-hours and up to 90 percent in elapsed project time.³ These savings, coupled with high levels of effectiveness and participant satisfaction, have been replicated in a variety of business and government organizations.² For more information, see www.cmi.arizona.edu, www.groupsystems.com, or www.groupsupport.com.

References

1. J. Nunamaker et al., "Electronic Meeting to Support Group Work," *Comm. ACM*, vol. 34, no. 7, July 1991, pp. 40–61.
2. J. Nunamaker et al., "Electronic Meetings Systems: Ten Years of Lessons Learned," *J. MIS*, vol. 13, no. 3, Winter 1996–1997, pp. 163–207.
3. D. Vogel et al., "Electronic Meeting System Experience at IBM," *J. MIS*, vol. 6, no. 3, Winter 1990, pp. 25–43.

could "speak" via the keyboard. So, the moderator had more time for finding defects. Some moderators could allow limited discussion of major defects because the defect reporting did not require any talking.

Preparation rates

An inspection's preparation rate (measured in lines of code per hour) significantly affects an inspection's effectiveness and efficiency.⁸ Early in 1997, it became clear that some groups had increased their preparation rates at the expense of the effectiveness of the inspections. Consequently, proper preparation rates received more emphasis throughout the year. As a result, we could evaluate how the GSS performed at different inspection rates. We classified the inspections in terms of their preparation rates per engineer. A typical recommended preparation rate for source inspections is 200 lines of noncommented source code per hour.^{8,9} At Baan, the comment lines were also inspected, so we in-

cluded them in the line counts. Therefore, the recommended inspection rate in this case was 200 to 300 lines of code.

Inspections included in this study

This study includes all the inspections of 4GL code in Baan's Application Department during 1997. It excludes 3GL code inspections and all the inspections of designs, specifications, and other development documents. We excluded these inspections to facilitate a clean assessment of the impact of GSS use. Of the 4GL code inspections, we only considered those where a group of engineers inspected their own code. (Frequently, engineers from one development site inspect another site's source code. This is especially important in an organization that is growing rapidly and where, consequently, the levels of experience are mixed.) These criteria let us compare the 87 GSS inspections to 102 paper inspections.

Results

We measured the results in terms of the major defects (*majors*) found. Figure 1 shows the effectiveness of the inspections in terms of majors detected per 1,000 lines of source code. Our strict definition of a major (see the "Inspections" sidebar) means that not all code changes that result from an inspection will be logged as majors. For example, improvements for performance's or clarity's sake are typically not logged as majors. We counted only those defects classified as majors after rework. We did not count false positives—that is, defects that were marked major but turned out to be minor problems (or not problems at all).

Figure 1a gives the effectiveness of the inspections in a scattergram where a point represents every inspection. Figure 1b represents the same data in a bar chart. The chart distinguishes three classes of inspections: 200 to 300, 300 to 400, and 400 to 500 lines of code per hour. As we previously noted, the target preparation rate was 200 to 300 lines. Each of the six bars represents at least 20 inspections. Preparing and logging the 189 inspections took over 1,250 hours. The total meeting effort was 330 person-hours for the paper inspections and 261 person-hours for the GSS inspections. The numbers over the bars indicate the effectiveness of the inspections. We measured effectiveness as the weighted average for all inspections of that class—that is, the total number of majors divided by the to-

tal size of the inspections of that class.

The bar chart indicates that, for a rate of 200 to 300 lines of code, the GSS inspections find 40 percent more defects (12.6 over 9) per KLOC than do the paper inspections. The difference is 46 percent for 300 to 400 lines (8.5 over 5.8), but the difference is gone for 400 to 500 lines (6.6 for both the GSS and paper inspections).

We define efficiency as the number of majors per person-hour for preparation and logging. We do not include the rework hours; we are interested mainly in the defect detection efficiency, and the GSS did not affect defect rework efficiency. Figure 2 gives the weighted average for efficiency. It indicates that the GSS inspections in the 200- to 300-line range are 40 percent more efficient. The difference is only 20 percent in the 300 to 400 range. In the 400 to 500 range, the paper inspections are 17 percent more efficient.

Discussion

The data indicate that the GSS inspections are more effective and efficient for inspections with preparation rates of 200 to 300 and 300 to 400 lines per hour. We believe that a GSS improves the inspection process by supporting it properly and enforcing it. For example, the requirement for inspectors to submit their defect reports before the meeting has encouraged them to contribute more to preparation. However, this is likely not only a tool technology issue. Because all other independent variables cannot be held constant in an industrial setting, other variables could have played a role. Two likely candidates are the preparation rate and the software's quality.

Preparation rate

The preparation rate is known to be a main independent variable for inspection quality. Table 1 shows the average preparation rates for the defined classes. As the table shows, differences in the preparation rate do not explain the differences in effectiveness.

Software quality

As we previously noted, all inspections concerned source code from the same major system. The engineers themselves decided whether to do a GSS inspection. The decision rested not on the source's perceived quality but on more down-to-earth variables, such as the preference of the engineer

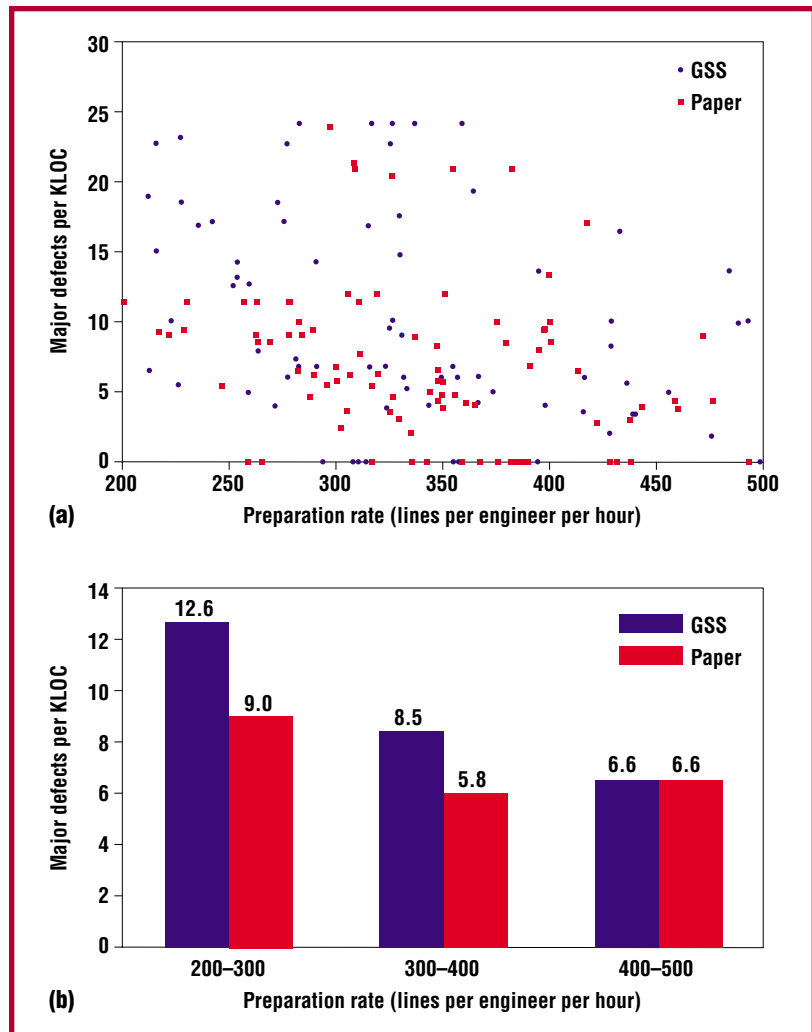


Figure 1. The effectiveness of 87 GSS inspections and 102 paper inspections: (a) a scattergram; (b) a bar chart.

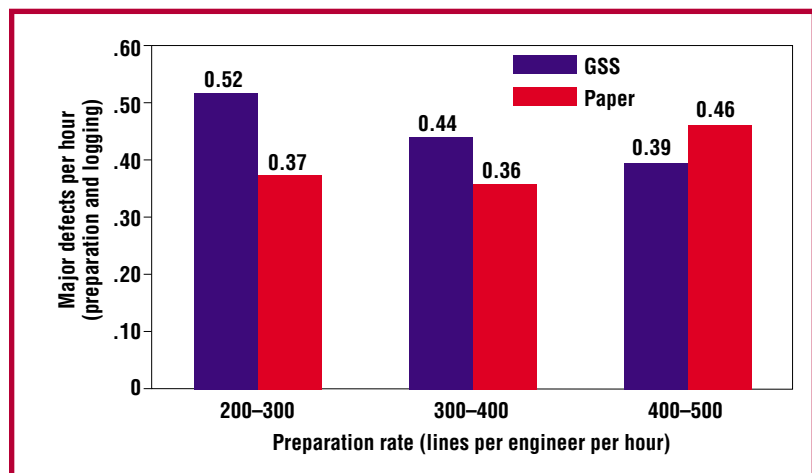


Figure 2. The efficiency of the inspections compared in Figure 1.

who moderated the inspection or the availability of the GSS facility.

One useful measure of inspection results is *yield* (the percentage of detected defects—for more detail, see the “Inspections” sidebar). We do not have complete yield data for a number of reasons, including these:

Table I**Average Preparation Rates for Inspections**

Class	Preparation rates (LOC/hr.)		Difference in rate (percent)	Difference in effectiveness (percent)
	GSS	Paper		
200 to 300	248	238	4	40
300 to 400	342	345	-1	46
400 to 500	429	427	0	0

- The defects found in early tests were not recorded at the same level of abstraction as was the inspection data.
- Our definition of a major makes it hard to compare the number of majors found to the number of test defects detected later.
- Modules were often not inspected as a whole but partially inspected and partially reviewed.

The limited yield data we do have suggest a higher detection yield for the GSS inspections. The code underwent a series of tests after the inspections. The data from the most extensive test show that the code that underwent paper inspection had 87 percent more fixes as a result of detected defects than did the code that underwent GSS inspection.

Implications

This study has implications for both GSS inspections and GSSs in general.

GSS inspections

We see three implications for GSS inspections.

A GSS can contribute to more effective and efficient inspections. This implication is obvious but important, given that mature software groups spend significant time in inspections. Furthermore, it is significant because inspections are an important technique to assure quality software. Making inspection results available electronically might provide earlier and easier opportunities for improving software development.

Distributed inspections are feasible. We do not have much data yet on the impact of distributing participants over different locations. However, our experience with inspectors using the GSS to handle preparation from their own desks encourages us to try distributed inspections. Given the international distribution of many development teams, this is an important opportunity.

A GSS needs to fulfill additional requirements. Supporting software inspections was a new application for GSSs that identified opportunities and exposed weaknesses. On one hand, GSS functionality and architecture allowed quick response to new requirements. On the other, we identified these problems:

- GSSs do not sufficiently support fixed-format input. The input in GSSs is typically free-format text. In some cases, it would be useful to fix the format and then ascertain, for example, that the second attribute is a one-digit attribute that allows distinguishing majors from minors.
- The GSS lacked integrated support for capturing metrics. For example, at a meeting's end we want to know the number of majors and minors, and we want to accumulate such data across sessions and between groups. The GSS could not calculate this automatically.
- Poor interfacing between the GSS and Baan's organizational systems limited expanded operational use. Seamless integration with the existing system and rapid, smooth movement of data between parts of an organization are paramount to operational success. This holds particularly for inspections because of their high frequency and because the participants are engineers. Engineers are very unforgiving if they repeatedly have to work around interface deficiencies. Integration with existing systems at Baan now lets engineers automatically export the sources that are to be inspected from the source code control system to the GSS. After the inspection, the meeting metrics are calculated and the detected defects are exported in the proper format to the defect-tracking system.

Research and development is underway to address some of these weaknesses. In particular, research projects are underway to improve GSS support for inspections, project tracking, and data modeling.⁶ For example, on the basis of the experiences we've described in this article, we are developing a GSS specifically for inspections.

GSSs in general

Software engineering is a particularly challenging application for GSSs. As we mentioned before, we are convinced that today's

About the Authors

software problems might be tomorrow's problems in other fields. So, we feel our results have implications for other GSS applications.

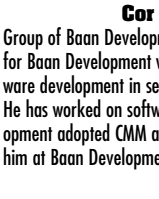
Meeting protocol and tool support. The results show clearly that the benefits of GSS support deteriorate quickly when a proper process is not followed. In other words, when the inspection is not done well, it does not matter how it is supported. This is also a well-known conclusion from implementing information technology support in other fields. It is food for thought for those who have been trying to support meetings with a GSS. How effectively does a GSS support less mature meetings? How sure can we be, given that the outcome of meetings is hardly ever managed in quantitative terms? Increased attention to improving meeting processes and meeting metrics is warranted.

GSSs in primary processes. The application of a GSS to support inspections was one of the first attempts to use a GSS routinely in daily work. Such use puts new and challenging requirements on GSS technology. We have labeled this use of a GSS as *support for primary processes*.¹⁰ Primary processes require "high-frequency" meetings under operational time pressure. In these meetings, participants know what they want to achieve and what is required from them. The meeting structure is mature, and consummating the process requires little to no special facilitation. Examples are the use of a GSS for emergency response or in the classroom for day-to-day teaching. The results of supporting inspections with a GSS give us high expectations for the application of GSSs in other areas.

The demands imposed on software engineering and its practitioners are ever increasing. Fortunately, combinations of methods, techniques, tools, and technology exist that will let us begin to address these demands. In particular, our findings open the door for broader consideration of GSS technology for software process improvement and suggest the development of process-technology hybrids. Many more opportunities exist to combine the best of the worlds of software process improvement and group support technologies. Both worlds will benefit as a result. ☺



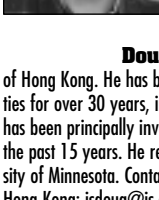
Michiel van Genuchten is CEO of GroupSupport.com, a company that focuses on the development and application of group support systems. He received his MSc and PhD from the Eindhoven University of Technology. He has been involved with software process improvement and GSSs since 1991. He is a member of the IEEE. Contact him at GroupSupport.com, PO Box 80, 5600AB, Eindhoven, Netherlands; michiel@groupsupport.com.



Cor van Dijk is the manager of the Software Engineering Process Group of Baan Development. This group works on process improvements for specific units and for Baan Development worldwide. He studied mechanical engineering and has worked in software development in several different technical and managerial jobs for more than 16 years. He has worked on software process improvement since 1994. Under his guidance, Baan Development adopted CMM and PSP/TSP and has made significant process improvements. Contact him at Baan Development, PO Box 143, 3770 AC Barneveld, Netherlands; cvdijk@baan.nl.



Henk Scholten is a software engineer in the Applications Department of Baan Development. This department provides tools, ideas, and knowledge to help Baan Development write software efficiently. He studied electronics and has worked as a software engineer for many years. Contact him at Baan Development, PO Box 143, 3770 AC Barneveld, Netherlands; hscholten@baan.nl.



Doug Vogel is a professor of information systems at the City University of Hong Kong. He has been involved with computers and computer systems in various capacities for over 30 years, including being president of an electronics manufacturing company. He has been principally involved in the development and application of group support systems for the past 15 years. He received his PhD in management information systems from the University of Minnesota. Contact him at the City Univ. of Hong Kong, Dept. of IS, 83 Tat Chee Ave., Hong Kong; isdoug@is.cityu.edu.hk.



Acknowledgments

We thank Baan's engineers and moderators who participated in the inspections. We also thank the *IEEE Software* reviewers for their suggestions, which led to significant improvements of the article. Finally, we thank Tom Rodgers of Texas A&M University for his comments on earlier versions of this article.

References

1. A. Porter and L. Votta, "What Makes Inspections Work?" *IEEE Software*, vol. 14, no. 6, Nov. 1997, pp. 99-102.
2. J. Nunamaker et al., "Electronic Meetings Systems: Ten Years of Lessons Learned," *J. MIS*, vol. 13, no. 3, Winter 1996-1997, pp. 163-207.
3. J. Nunamaker et al., "Electronic Meeting to Support Group Work," *Comm. ACM*, vol. 34, no. 7, July 1991, pp. 40-61.
4. D. Vogel et al., "Electronic Meeting System Experience at IBM," *J. MIS*, vol. 6, no. 3, Winter 1990, pp. 25-43.
5. I. Tervonen, "Support for Quality-Based Design and Inspection," *IEEE Software*, vol. 13, no. 1, Jan. 1996, pp. 44-54.
6. T. Rodgers et al., "In Search of Theory and Tools to Support Code Inspections," *Proc. 31st Hawaii Int'l Conf. Systems Sciences*, vol. 3, IEEE CS Press, Los Alamitos, Calif., 1998, pp. 370-378.
7. M. Genuchten, W. Cornelissen, and C. van Dijk, "Supporting Inspections with an Electronic Meeting System," *J. MIS*, vol. 14, no. 3, Winter 1997-1998, pp. 165-178.
8. W.S. Humphrey, *Managing the Software Process*, Addison-Wesley, Reading, Mass., 1989.
9. W.S. Humphrey, *A Discipline for Software Engineering*, Addison-Wesley, Reading, Mass., 1995.
10. M. Genuchten, J. Nunamaker, and D. Vogel, "Group Support Systems in Primary Processes," *Proc. 31st Hawaii Int'l Conf. Systems Science*, vol. 1, IEEE CS Press, Los Alamitos, Calif., 1998, pp. 580-589.