



Experiences with software inspections – a case study

J. White

*Co-operative Computing Systems Division,
Hewlett Packard Ltd, Nine Mile Ride,
Wokingham, Berks, RG11 3LL, UK*

ABSTRACT

In recent years many claims have been made for Software Inspections, including the ability to increase the quality of delivered products, and increase the productivity of the software development process.

This paper discusses the use of Software Inspections in a commercial software development environment. In particular, it considers the motivations for using inspections, the practical aspects of introducing the technique, and the results obtained. Both quantitative and qualitative results are considered.

This is a real life case study, based on the experiences of a single division within Hewlett-Packard, showing both the difficulties and barriers to be overcome in introducing inspections, and the benefits to be gained.

It shows that inspections can be up to 8 times more efficient at finding defects than dynamic testing, and that inspections can provide a return on investment both before and after product release.

INTRODUCTION

CCSY-PWD is a software development entity in HP, producing commercial software in the electronic messaging market. Typically, our customers are large multinational companies, with world-wide messaging needs.

This case study covers the period 1991-93. At the start of this period our software development processes could be considered by industry standards as relatively mature with, for example, a structured lifecycle and use of software quality planning and metrics. However, there was still a heavy reliance on dynamic testing at the end of software development. As a result, most of the well publicised woes of the software industry such as missed schedules, expensive back-end testing, and late defect fixes continued to concern us.

A defect detection approach, relying primarily on back-end testing, however well planned, was proving an ineffective and expensive approach. From our historical data, we know that it takes an average of 6.5 hours to find and fix a defect detected

732 Software Quality Management

during dynamic testing, and 2 weeks for a defect detected after release. Clearly, the earlier defects are detected the cheaper they are to eliminate.

Dissatisfaction was visible from both our customers perspective and internally - we had to move even more towards a preventative approach.

As a result, a number of co-ordinated improvement activities were started, including changes to engineer development tools, a formal scheduling process, project quality planning, and organisational structure adjustments to put more focus on prevention. This paper focuses on perhaps the most generic and widely applicable of the techniques we have introduced, and one that can be applied in any software development environment regardless of platform, languages, or tools. That technique is **software inspections**.

Why consider inspections?

Some reviews of documents and code were taking place in CCSY-PWD at this time, but the process was poorly defined, not repeatable, and its effectiveness was difficult to assess.

Typically, internal documents might be distributed to interested parties, who would be asked to feedback comments by a certain date. Informal one-to-one feedback may or may not be provided to the author, and failure to respond was often taken as agreement. Sometimes a review meeting would be held, but data on the costs and benefits of the process was limited, and anecdotal evidence was that the effectiveness of such reviews was generally weak.

A more detailed analysis of the causes of poor reviews revealed that they were not planned, or factored into schedules; that little or no preparation was done by attendees; that the review objectives were unclear; high-level supporting documents were not available; and feedback was not addressed. In summary, no common process existed.

At the same time, results from the use of inspections, both within HP and the industry as a whole [1][2][3], were indicating the potential of this simple technique, when applied within a relatively mature software development environment. We decided to see whether our organisation could also benefit.

In simple terms, our overall objectives were to,

- increase the effectiveness of our review process,
- find problems as early as possible (preventing them causing expensive problems later in the development cycle),
- provide data to show where the development process itself could be improved.

What are Inspections?

Inspections are basically structured reviews of any artefacts produced during development of software products. They are a 'low-tech' technique, that apply engineering brain power and expertise to detecting defects in a well defined process that is designed to be as productive as possible for all concerned. It involves examination of a document (or code) by a team of people led by an independent chairperson (moderator).

The goal is to help the artefact creator develop a high-quality product in the shortest time.

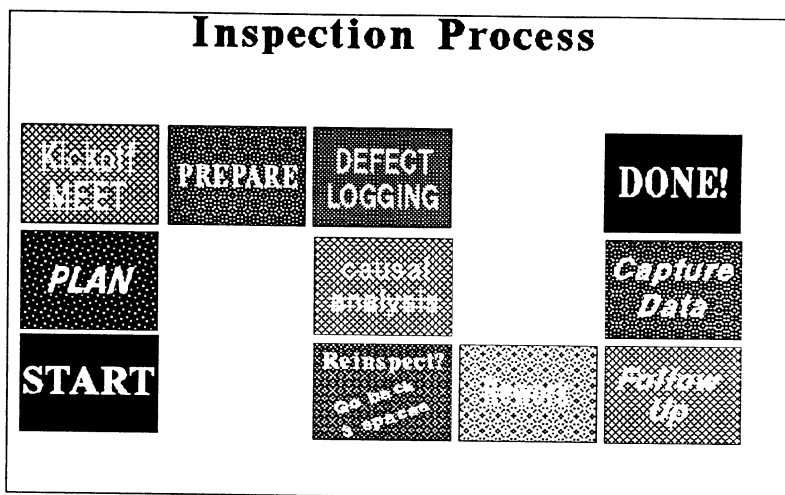


Fig 1. Inspection Process steps.

Our inspection process (Fig 1.) has six steps and uses checklists and role playing as part of the detection techniques. The key steps are,

Plan	The activity of planning the inspection, and ensuring that entry criteria are satisfied - initiated by the Author, but driven by the Moderator.
Kick-off	Brief meeting to ensure that all participants understand the process, their roles, and that everybody has the appropriate documentation.
Prepare	Time allocated to the individual inspectors to work alone from their specific perspective, to understand, study, and find defects in the material.
Meet	Meeting(s) run by Moderator to raise defects and questions, and log them.
Rework	Author responds to all items raised during inspection meeting.
Follow-up	Moderator ensures that all items have been responded to, that data has been recorded, and that the exit criteria are satisfied.

There are plenty of courses, texts and gurus on inspections, for those interested in more details of the process, roles, and philosophy of the technique, so we will not go into further detail here. This paper concentrates firstly on the results of introducing inspections at CCSY-PWD, and then outlines the approach adopted, and some of the barriers and issues that had to be overcome.

It is worth noting that the word 'inspection' itself does tend to paint a picture of a person with a clipboard checking and rejecting individual physical copies of widgets as they roll off a production line. This view of 'inspection' as a back-end detection technique unfortunately conflicts with our use of software inspections as both a method of early defect prevention and process improvement. Perhaps we should find a better term to convey what we are trying to do; in the meantime, it's worth being careful to ensure that all those involved understand how inspections can contribute to this prevention strategy, as well as the nuts and bolts of the inspection process itself.



RESULTS

This section highlights the key results and benefits noted at CCSY-PWD.

Although inspections can vary considerably, Fig 2 summarises the profile of typical code and design inspections held at CCSY-PWD, and indicates the sort of data that is recorded about the inspection process and the defects found. It is this sort of data that is essential if we are to analyse and improve the inspections process and the overall software development process.

Code Inspection	Design Inspection
1100 lines	20 pages
4 people	4 or 5 people
4 hours preparation each	3 hours preparation each
2 x 2 hour meetings	2 x 1½ hour meetings
56 hours total effort	35 hours total effort
Finds (on average): • 15 Major defects • 42 Minor defects	Finds (on average): • 12 Major defects • 39 Minor defects

Fig 2. Profile of typical inspections

From the data collected as part of the inspections process, we have been able to analyse the returns.

The following overall results are from the last 2 years of design and code inspections at CCSY-PWD, from a sample of over 60 inspections covering 1000 pages of design documentation, and 20K lines of code with a total investment of 21 man months in inspections.

This type of analysis has proved invaluable in persuading both engineers and managers of the benefits of inspections. In order to do this, the most severe assumptions have been made in calculating the following results, aimed at showing the absolute minimum returns we can expect from inspections.

For example, all minor defects found by inspections are excluded from the following analysis, but all defects found by dynamic testing are included - this eliminates the possible criticism that inspections find lower severity problems than testing. Also, we assume that a major defect in a design would only lead to a single defect in the code. In practice, it could easily be many more.

The key results are as follows:-

Shorter Time to Market

- **Faster defect discovery**
- **Earlier defect discovery**
- **Faster fixing**
- **Shorter test cycle**

Even when applying the most severe assumptions to the comparison, we have found that inspections are at least twice as efficient at finding defects as dynamic testing. It



takes less than 3 hours to find and fix a major defect using inspections, compared to at least 6.5 hours using conventional dynamic testing.

If all defects found by inspections are included in the calculations, then we are finding and fixing a defect every 0.8 hours using inspections - 8 times as efficient as dynamic testing.

This is comparable with the experience of other entities in Hewlett-Packard, as shown in Fig 3.

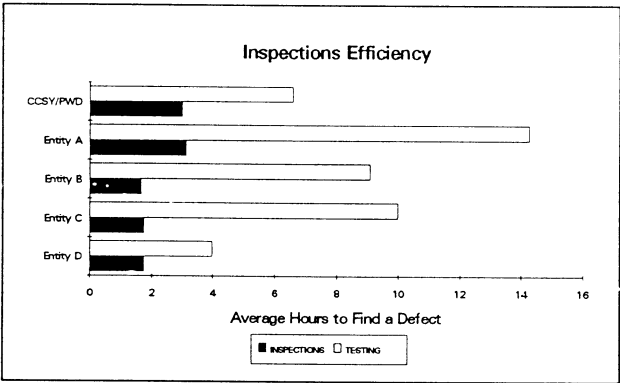


Fig 3. Inspections Efficiency

By detecting defects when they are easier and cheaper to fix, and preventing knock on defects, the rework effort has been reduced.

Our net saving has been at least **one engineering year** of effort.

We also found that the number of testing cycles was reduced, increasing the chances of a project releasing on time. Fig 4 shows the typical overall impact.

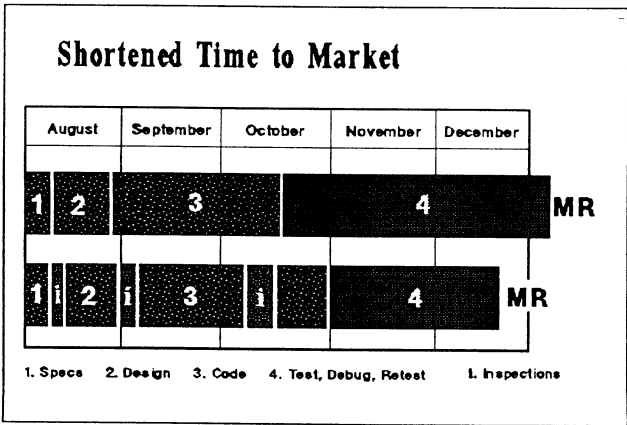


Fig 4. Effect of inspections on Time to Market. [MR=Manufacturing Release]



736 Software Quality Management

In our case the number of tests to be run was unchanged, but the number of attempts required to successfully pass each test declined, resulting a shorter overall test cycle.

Post-release Savings

- **Greater Customer Satisfaction**
- **Lower Defect Fixing and Maintenance Costs**
- **Higher Productivity**

Another way to look at the results of inspections is purely from the post-release savings that may accrue. We used severe assumptions about the number of customer reported problems that inspections may have prevented. For example,

- Regardless of inspections, we assume that dynamic testing would still detect up to 70% of problems.
- We assume that not all defects remaining in the released product are encountered by customers, and of those that are only a proportion require engineering effort to resolve.

Given this model, we estimate that inspections have still reduced the number of major defects that have required fixing by over 80, and saved more than **3 engineering years** worth of maintenance work.

This is a payback of at least twice the amount required to break even.

This is starting to be confirmed by actual incoming defect rates on products that have used inspections, as they are lower than the norm.

Specific examples include one major component that was code inspected after initial unit testing indicated that it was high risk, which has subsequently had no defects reported by customers in the two years since it's release.

Side Benefits

- **Improved teamwork**
- **Shared product knowledge**
- **Improved communication and Engineer training**
- **Pointer to problem areas**

Teams have reported that inspections have provided an excellent method of sharing understanding and expertise, and improving communication and cross-training within the project.

This is not surprising since the inspections process makes public within the team key information such as design and code decisions, allowing people to spot problems not only in the area being inspected, but also in their own areas.

Engineers also report that the resulting 'shared ownership' of code is a benefit. Indeed, once they have experienced them, most engineers are keen to be involved in inspections, providing that they are scheduled and not in addition to their normal activities.

They also provide an opportunity for engineers to learn from each other, and a good way of bringing specialist expertise, such as performance or human factors specialists, to bear on key areas of a project. This has been aided by one minor

change that we have made to the inspections process to allow limited discussions about defects in the inspection meeting when it can help aid understanding (discussion of solutions is still done outside the meeting).

Inspections can also identify error prone items during development, which may require special attention. For example, code modules that might need to be rewritten, or require the planning of more detailed testing.

Other management benefits include better indicators for tracking project milestone progress. For example, "design complete" can be linked to the completion of the last design inspection to give a more reliable indicator that the quality of the design is adequate for subsequent phases.

INTRODUCING INSPECTIONS

Our overall approach to introducing inspections was based on the following steps,

- Gain expertise
- Pilot the process
- Evaluate Pilots
- Eliminate barriers and spread/refine process
- Analyse and act on results

Gain Expertise

We started by identifying one well respected individual to co-ordinate the effort, get trained in the details of the process, and be able to provide hands-on practical assistance to both management and engineers during the pilot phase. Eventually this person become the Chief Moderator and champion of inspections within the division. We made use of the best practice from elsewhere in the industry, and within our own company, and made a conscious decision not to try to modify the inspections process ourselves until we had at least tried it in it's 'standard' form.

Pilot the Process

We communicated the objectives, used industry results and a simple business case, to get initial management interest and agreement to pilot inspections. In parallel, we engaged the help of a few key engineers who were interested in trying potentially better ways of reviewing designs and code.

This allowed us to identify a couple of key projects, most likely to be successful with inspections. For example, they were sufficiently large and complex to allow the inspections teams to be formed primarily from within the team, without calling on other teams, and they were at an early enough stage in their lifecycle to allow inspections to be factored into the plans and schedule.

Once the projects were chosen, brief training of both the managers and engineers involved followed. We then identified specific inspection opportunities within these projects. High risk areas of design or code were chosen, based both on analysis of the likelihood of failure and the consequences of failure. In truth, there was also a large element of guess work at this stage, and if in doubt we tended to select areas that we felt would be easy to inspect. This eased the learning process for all involved, and built up our confidence, since if even randomly selected inspections

738 Software Quality Management

could produce results then we could anticipate even better returns when they were better targeted.

The pilots enabled us to get local data and experience, and helped us address the 'not invented here' syndrome that can often lead to the rejection of such ideas.

Evaluate Pilots

Throughout the pilots we made it clear that the data collected as part of the inspections was going to be used to analyse the process itself, and would not be used in any way to evaluate individual engineers. This was key to getting buy-in, and also in ensuring that the data collected was accurate.

We reported regularly on the use of inspections, communicating both the quantitative and qualitative benefits, and the issues. We were deliberately conservative in our claims - we wanted people to say "if it only gives me this much, I'd still be stupid not to use it".

Thankfully, our results were similar to those experienced by other practitioners, which made the next steps easier. In simple terms, people felt that they were finding more defects, and producing higher quality deliverables than in the past.

The hard data generated by the inspections process proved to be very powerful in evaluating inspections, and putting the case for expanding the use of inspections within the division.

Spread/refine process

Gradually, more teams started to include planned inspections into their developments, and the view of most managers started to change to inspections becoming the expectation rather than the exception.

We did not make inspections mandatory. Our approach was still to persuade people with results, and try to make their own experiences as successful as possible. To this end, the Chief Moderator provided cascade training for additional moderators, and project-centred training for project teams just before their first use of inspections.

We also developed basic support materials, including simple design and code inspection checklists, criteria for selecting inspections, and recommendations on how much and what to inspect.

Other types of review continued to be used on other documents (e.g. specifications), and we encouraged the use of 'elements of inspections' to improve them where possible, although it was important not to consider these as inspections for the purposes of evaluating the technique. As a minimum, we tried to record how many defects, and how much time was spent in such ad hoc reviews, so that we could compare their efficiency.

Two years after the start of the initial pilots, we are at the stage where they are normal practice for all key projects at CCSY-PWD.

Analyse and act on results

Data on each inspection was collected, and stored in a simple spreadsheet to allow regular analysis.

Six monthly return on investment information was provided to the management team.

We used this data to guide,

- Improvements to the inspections process.
e.g. use of a reader role in design inspections, but not in all code inspections; updated checklists.
- How inspections are used.
e.g. typical inspection rates made available to assist in project planning; recommendations to focus on designs and requirements, rather than all code.
- Improvements to the software development process.
e.g. need for better design documentation, and improved change control/commitment management.

ISSUES

However obvious inspections appear, or how simple introducing such a 'low-tech' technique seems, our experience shows that potential pitfalls abound.

This section highlights some of the issues we experienced, and questions we encountered during our program.

Why should we bother with this?

This question, from people at all levels of the organisation, is often the underlying first response to any suggestion of change. The answer has to be tailored to the person concerned, and needs to focus on how the benefits of inspections can make them more successful. For example, for a Project Manager, the possibility of increasing the chances of achieving their schedule to the required quality; for a senior manager, the reduced maintenance load leading to more resources for new projects.

Use of the "because it's good for you" or "because we said so" line of persuasion is not recommended.

Resistance to Change

Institutionalising any change in the software development process, so that it is accepted and used effectively, instead of reluctantly tolerated, can be painfully slow and difficult in most organisations.

We tried to avoid the most obvious mistakes of Change Management in general, such as failing to identify the Sponsor's and Targets of the change, but we still encountered much resistance and questioning of our aims. The 'Yes Minister' syndrome was much in evidence, with everyone agreeing that this was right in principle, but that it clearly didn't apply to them!

From past experience we were cautious of any mandating of inspections, or attempting to introduce them on all projects at once.

Although we would have liked to speed up the whole introduction, there were a number of actions that, with hindsight, helped to make the 2 years of effort worthwhile.

- The inspections were co-ordinated as part of an overall improvement plan - not just 'flavour of the month'.
- The introduction of inspections was handled as a project itself - it needed to be planned and managed.

740 Software Quality Management

- We had data, on which to base a business case for inspections.
- The problem we were trying to solve was generally agreed.
- We got the support of a critical mass of key influencers.
- We used a Chief Moderator, responsible for the effectiveness of the Inspections Process, and identifying areas for improvement.
- The pilot approach involved people in the changes, and allowed them to raise concerns.

Lack of Time

The most frequent concerns expressed were that 'we just don't have time for another activity', or 'my engineers are too busy'. Actual results can help to address these concerns, but the key is to plan the use of inspections into the schedule up front just like any other activity.

In the end there was a need for a certain leap of faith, since we are clearly not able to run almost identical control projects in parallel to show the absolute effect of inspections.

All the indications are that introducing inspections actually reduces the risk to the schedule, and should result in the actual release date being earlier than it would otherwise have been. Unfortunately, this doesn't always convince hardened, overworked Project Managers. The difficulty is that when compared to the most optimistic schedule (the one that has a 1 in 100 likelihood of being met, but which keeps the senior manager happy, at least until the first slip occurs), any realistic schedule looks bad. It is perhaps only natural that when confronted with the possibility of adding additional tasks to a schedule, the immediate reaction is to say that it will take longer.

We found that by determining how many of these unrealistic dates were ever actually met, and making that data clear to all concerned, discussions about the actual impact of inspections were easier to handle.

Finally, although introducing inspections may seem to add new tasks to the schedule, it is worth remembering that some effort was probably going to be spent on informal reviews, chasing people for comments, soliciting inputs, and such like, so the net difference may not be as huge as it appears.

Management buy-in

From the outset, our senior manager was keen to use inspections, but left the individual project managers to make the final decision on whether they would use them on their particular project. For the pilots he even made it clear that he would accept a 10% increase in engineering effort, provided it was used on inspections, to address concerns over the learning curve involved.

It is interesting to note that the major resistance and lack of commitment came from the middle managers, and some project managers. This seems to be because they may not see the direct benefits themselves, and are rewarded in practice (often unintentionally) for reacting to problems and trying to meet tight deadlines rather than the longer term impact on customer satisfaction or cost of quality.

Ideally, there needs to be management buy-in at all levels of the organisation. However, we found that a few examples of demonstrable management commitment is worth any number of managers saying the right thing but acting completely differently. For example, when offered the possibility that a schedule *might* be shortened by 2 weeks if some inspections were dropped, one senior manager made it clear that the best way to secure the schedule would be to continue as planned.

Testing v Inspections

We frequently encountered the issue of whether using inspections should mean that we could do less dynamic testing. Our basic conclusion is that the type and number of tests developed may not change significantly, but that the number of times they need to be run before they pass successfully should be reduced.

It is this test-fix-retest-refix-retest cycle that we hope to impact first, not the size of the test suite.

At present, the majority of our dynamic back-end system testing is still required as part of each projects overall plan for quality. Eventually, as the coverage of inspections increases, this balance may change. At the moment they each tend to be good at finding particular types of defects, and we consider that they are both necessary elements in a comprehensive Quality Plan.

Particular areas where inspections have proved to be particularly effective compared to dynamic testing include identifying problems related to qualities such as portability, maintainability, localisability, and testability.

What to inspect?

Inspections can be used on any deliverable in the software development process, including contracts, specifications, designs, code, test plans, documentation, but we want to do them as early in the development process as possible, so requirements and design documents are prime candidates. If you don't have these, then code inspections are the most practical place to start. When you get your own data to show that it's the defects in the earlier deliverables that are leading to so many problems in the code, you will probably conclude, like many before you, that inspections of earlier deliverables makes more sense.

There is much to be said, however, for going through this learning process in your own organisation. In particular, it is likely to be easier to convince people to start looking at code, and then **their** results will naturally point **them** towards addressing the earlier software development processes and deliverables.

For example, we found one team that was perfectly happy with the designs they were writing until the results of their inspections persuaded them that they needed more consistent designs, resulting in the introduction of design phase checklists.

How much to inspect?

This question seems to be poorly addressed in most inspections courses and literature. There is often an implicit assumption that 'everything' should be inspected. For most commercial organisations this is impractical. We took a simple approach of trying to identify and focus our efforts on the highest risk areas in our products. For example, factors such as module complexity, number of external interfaces, criticality of data, and experience of engineers were all considered in the selection of which code to inspect.

Our current recommendation to teams is to inspect **all** designs plus the highest risk 20% code. A variation on this is to continue inspecting until the results show diminishing returns, based on the defects found and effort invested. Simply by using inspections, this type of data becomes readily available, and can help to guide the development process.



742 Software Quality Management

CONCLUSIONS

Inspections are no silver bullet. They are one of many techniques, but one which is still overlooked by many organisations.

They can provide significant returns, both pre- and post-release, if used as part of a planned software development approach. They can help release resources for other development work, and increase the chances of meeting schedule commitments.

From the experiences of a single organisation, our key recommendations are,

- Use a formal inspection process - only tailor it once you understand it, and have your own data.
- Treat introducing inspections as a project in it's own right - it needs to be planned and managed.
- Identify an organisational champion (Chief Moderator) or group of champions.
- Start small and use an incremental approach.
- Don't try to inspect everything - select high risk items and high level documents if possible.

Software developers who are serious about the quality of their delivered software should look carefully at inspections. Benefits to their customers, and to their own organisations, are there for the taking.

REFERENCES

1. Fagan, M. E. 'Design and Code Inspections to reduce Errors in Program Development' *IBM Systems Journal*, Vol.15, No.3, 1976.
2. Fagan, M. E. 'Advances in Software Inspections' *IEEE Transactions on Software Engineering*, Vol.SE-12, No.7, 1986.
3. Gilb, T. *Principles of Software Engineering Management*, Addison-Wesley, 1988.