Agile Specification Quality Control: Shifting emphasis from cleanup to sampling defects

Tom Gilb
Tom@Gilb.com, www.gilb.com

Abstract. Traditional Inspection is often uneconomic and ties up valuable staff resources. Shifting the emphasis from cleanup (that is, from identifying defects and then removing them), to merely *sampling* the defect level of specifications, produces significant benefits. It enables the *quality level of specifications* to be determined more rapidly. Consequently, the QC can be carried out more frequently. Systems and software engineers rapidly learn, through SQC feedback, to take standards seriously, which in turn reduces defect injection. Further, by analyzing where/how the defects occur continuous process improvement can be supported.

INTRODUCTION

If we carry out inspection of specifications properly (Gilb and Graham 1993) the cost is barely tolerable for some: about one hour of effort, per page[1] checked, per systems engineer or software engineer. The harvest, even if we are skilled, is only to identify between 40-80% of the major defects. That leaves many remaining major defects undetected, and many of these *will* be found, at considerable cost, during testing or in the final released product.

Of course, finding defects using traditional inspection (and fixing them), earlier than the test stage is beneficial, and may even pay off. However, there is a better way: Agile Specification Quality Control (Agile SQC). It ought to appeal to all Spec QC purposes, and especially to the many organizations that have not been able to stomach the high costs, and low effectiveness, of traditional inspection.

The main concept of Agile SQC is to shift emphasis *from* 'finding and fixing defects', to 'estimating the specification defect density', and using this information to motivate systems and software engineers to learn to *avoid* defect injection in the first place. Such a shift permits a dramatic cost saving. When our QC purpose is *measurement*, rather than 'cleanup', we can *sample,* rather than have to check 100% of the specifications. This is the major opportunity that Agile SQC provides. The main *purpose* of Agile SQC is to *motivate* individuals to learn to reduce major defect insertion. Secondary purposes include:

- To prevent uneconomic major-defect density specifications from escaping downstream – and thus to avoid the consequent delays and quality problems. The major tactic to achieve this is to impose a numeric exit-barrier for the specification process, such as 'only a maximum of 1.0 remaining majors per page';
- To teach and reinforce current specification standards.

PROCESS DETAILS

Traditional Inspection Method: The old inspection method (widely practiced in CMM Level 3 as peer reviews) was based on the idea of inspecting 100% of all pages, at optimum rate

[1] A page is defined as 300 words of non-commentary text. Non-commentary text is core specification or background text; it is not notes or other commentary text.

checking (one page per hour), using a review team of between 2 and 5 software and systems engineers. The maximum yield of major defects from such an inspection process is in the range of 40%-80% depending on specification type (For example, a maximum of 60% for software source code specifications, and a maximum of 80% for requirements specifications – in practice however actually more likely that only 30% is achieved since malpractice is common). The reported ability to actually *correctly* correct major defects, once found, is only 5 out of 6 fixes attempted (Fagan 1986 reported in Gilb and Graham 1993). All this amounts to the following:

- The same order-of-magnitude defects *remaining*, as before the quality control process was applied;
- Little or no change in the defect *insertion* density. In requirements specifications, this *regularly* exceeds 100 major defects per 300 lines of specification (Personal experience by field measurement over many years).

New Agile SQC Method: The new 'Agile SQC method' is based on the following:

- *Sampling* of a specification;
- A few (1 to 3) pages at a time;
- Starting early (perhaps once the first 5% of a large specification is written);
- Frequently (every week or so) until the work is completed.

For each individual systems or software engineer (each one must be motivated and trained personally), their sampled specification pages will be checked against a set of a few simple rules – usually about 3 to 7 rules are applied (For example, for initial checks, these could be: *Clear enough to test, Unambiguous to intended readers, and No design options in the requirements).* The reviewers/checkers are asked to identify all deviations from these rules. Any deviation is termed a 'specification defect'. The reviewers/checkers are then asked to classify any specification defect that can potentially lead to loss of time, or significant reduction in product quality, as 'major'. The entire checking session might use only 2 engineers for 30 to 60 minutes This might seem quite a high checking rate, but remember that only a few rules are being used and no other documents are being consulted to check out the original source of material, so we can go faster. In any event, as long as we turn up more defects than the threshold exit level for defects, then exactly how effective we are in detecting defects is a secondary issue.

The major defect findings are reported to a review leader, who calculates the estimated number of defects actually present, based on the total found by the team. An inexperienced team is usually about one third effective, so the estimated total number of majors per page is about three times the total of *unique* majors found by the team. This is a very rough calculation, but it seems to work well in practice.

A pre-arranged standard for exit control (the *fail to exit* level) is set for unacceptable specification major-defect density. Initially, it can be set at 'anything more than 10.0 majors per page'. In the longer term (beyond 6 months of culture change), the aim should be to set the limit at 'anything more than 1.0 majors per page'. To give some examples, IBM reported using a maximum of 0.25 major defects per page (Humphreys 1989). NASA reported a standard of using 0.1 major defects per page (Bhandari et al. 1994). The initial limit set is a matter of trying to get better as fast as humanly possible. Ultimately, it should become a matter of finding the level that pays off, for the class of work you are doing.

Note: There are several limitations to this simplified Agile SQC process:

- It is only a small sample, so the accuracy is not as good as for a 100%, or than a larger-than-few-page sample;
- The team will not have time or experience to get up to speed on the rules and the

concept of major defects;

- A small team of two people does not have the probable greater effectiveness of 3 or 4 people;
- The entire specification will not have been checked, so there will not be the basis for making corrections to the *entire* specification;
- The checking will not have been carried out against all the possible source documents (Usually in the Agile SQC process, no source documents are used, and memory is relied on. While this means that the checking is not nearly as accurate, it does considerably speed up the process).

However, if the sample turns up a defect-density estimation of 50 to 150 major defects per page (which is quite normal), that is more than sufficient to convince the people participating, and their managers, that they have a serious problem.

As discussed earlier, the immediate solution to the problem of high defect density is *not* to set about removing the defects from the document, because the same order of magnitude level of defects would still remain. The best solution for a document with a high defect density is to rewrite it *entirely*, using an individual who does not insert too many defects. Long term, the most effective practical solution is to adopt Agile SQC as part of the corporate process, and most importantly, make sure each individual specification writer takes the defect density criteria (and its 'no exit' consequence) seriously. They will then learn to follow the rules; and as a result will reduce their personal defect injection rate. On average, a personal defect injection rate should fall by about 50% after each experience of using the SQC process. Widespread use of Agile SQC will result in large numbers of systems and software engineers learning to follow the rules. To get to the next level of quality improvement, the next step is to improve the rules themselves.

Agile SQC Process

Tag: Agile SQC. Version: April 18, 2005. Owner: Tom@Gilb.com. Status: Revised Draft.

Entry Conditions

- A group of two, or more, suitable people* to carry out Agile SQC is assembled in a meeting.
- The people have sufficient time to complete an Agile SQC. Total Elapsed Time: 30 to 60 minutes.
- There is a trained SQC team leader at the meeting to manage the process.

Procedure

P1: Identify Checkers: Two people, maybe more, should be identified to carry out the checking.

P2: Select Rules: The group identifies about three rules to use for checking the specification. (My favorites are clarity ('clear enough to test'), unambiguous ('to the intended readership') and completeness ('compared to sources'). For requirements, I also use 'no optional design'.)

P3: Choose Sample(s): The group then selects sample(s) of about one 'logical' page in length (300 non-commentary words). Choosing such a page at random can add *credibility* – so long as it is *representative* of the content that is subject to quality control. The group should decide whether all the checkers should use the same sample, or whether *different* samples are more appropriate.

P4: Instruct Checkers: The SQC team leader briefly instructs the checkers about the rules, the checking time, and how to document any defects, and then determine if they are *major* defects (majors).

P5: Check Sample: The checkers use between 10 and 30 minutes to check their sample against the selected rules. Each checker should 'mark up' their copy of the document as they check (underlining issues, and classifying them as 'major' or not). At the end of checking, each checker should count the number of 'possible majors' (spec defects, rule violations) they have found in their page.

P6: Report Results: The checkers each report to the group their number of 'possible majors.' Each checker determines their number of majors, and reports it.

P7: Analyze Results: The SQC team leader extrapolates from the findings the number of majors in a single page (about 6 times** the most majors found by a single person, or alternatively 3 times the unique majors found by a 2 to 4 person team). This gives the major-defect density estimate. If using more than one sample, you should average the densities found by the group in different pages. The SQC team leader then multiplies the 'average major defects per page density' by the 'total number of pages' to get the 'total number of major defects in the specification' (for dramatic effect!).

P8: Decide Action: If the number of majors per page found is a large one (ten majors or more), then there is little point in the group doing anything, except determining how they are going to get someone to write the specification 'properly', meaning to acceptable exit level. There is no economic point in looking at the other pages to find 'all the defects', or correcting the majors already found. There are simply too many majors not found.

P9: Suggest Cause: The team then chooses any major defect and thinks for a minute why it happened. Then the team agrees a short sentence, or better still a few words, to capture their verdict.

Exit Conditions

• Exit if less than 5 majors per page extrapolated total density, or if an action plan to 'rewrite' the specification has been agreed.

Figure 1: Specification of the Agile SQC Process

Notes:

* A suitable person is anyone, who can correctly interpret the rules and the concept of 'major'.

** Concerning the factor of multiplying by '6 ': We have found by experience (Gilb and Graham 1993: reported by Bernard) that the total unique defects found by a team is approximately twice that of the number found by the person, who finds the most defects in the team. We also find that inexperienced teams using Agile SQC seem to have about one third effectiveness in identifying the major defects that are actually there. So 2 x 3 = 6 is the factor we use (Or 3 x the number of unique majors found by the entire team).

CASE STUDY 1: A FINANCIAL ORGANIZATION

In 2003, a large multinational financial group was a pilot user of this Agile SQC process. It also had combined this with adopting a specification and planning language, Planguage (Gilb 2005). After six months, the organization reported the following for requirements and design specifications:

- Across 18 development projects using the new requirements method, the average major defect rate (per page) on first inspection is 11.2;
- 14 of the 18 development projects exited successfully on first pass Agile SQC. The other 4 development projects failed to meet the exit criteria of 10 major defects per page, the projects' specifications had to be improved, and were then re-inspected;
- A sample of 6 development projects with requirements in the 'old' specification format were tested against the following set of rules:
  - The requirement is uniquely identifiable;
  - All stakeholders are identified;

- The content of the requirement is 'clear and unambiguous';
- A practical test can be applied to validate delivery of the requirement.

The average major defect rate (per page) in this sample was 80.4.
A few months later, as a result of the continuing overall success of the pilot testing, the client decided to spread Agile SQC widely to all types of technical specification.

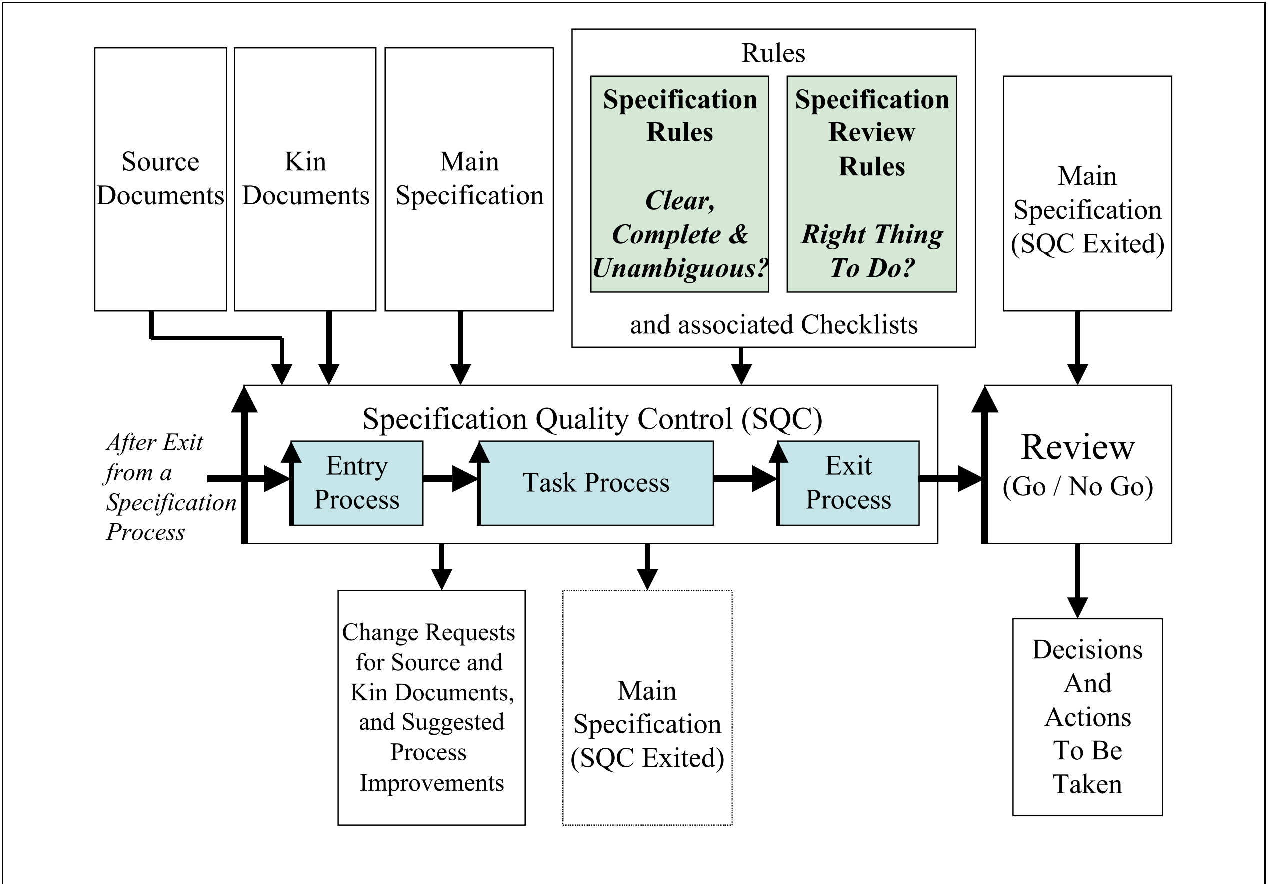
Rules
Specification Rules
Clear, Complete & Unambiguous?
Specification Review Rules
Right Thing To Do?
and associated Checklists
Source Documents
Kin Documents
Main Specification
Main Specification (SQC Exited)
Specification Quality Control (SQC)
After Exit from a Specification Process
Entry Process
Task Process
Exit Process
Review (Go / No Go)
Change Requests for Source and Kin Documents, and Suggested Process Improvements
Main Specification (SQC Exited)
Decisions And Actions To Be Taken

Figure 2. Overview of the SQC Process (Gilb 2005)

CASE STUDY 2: A JET ENGINES MANUFACTURER

At one of my clients, we sampled 2 pages of an 82-page requirements document: four managers checked page 81, and four other managers, who were directly involved with the requirement specifications projects, checked page 82. These pages were all 'non-functional' requirements (such as, security). We agreed to check against the following simple set of requirement specification rules:

1. **Unambiguous** to intended Readership
2. **Clear** enough to test.
3. No **Design** specifications (= 'how to- be good') mixed in

Violation of any one of these rules constituted a specification 'defect' and was classified either as 'major' (likely to result in potential damage to effort or quality) or 'minor' (no way they can harm us, even though they are defects).

We also agreed a specification exit level of 'No more than one *remaining* major defect per page'. They 'agreed' (for demo purposes!) that any manager who signed off (approved) a requirements specification with more than 100 remaining major defects per page should be fired for incompetence. Later that day they themselves were, as we shall see, to provide clear numeric evidence that – they themselves should be 'fired'!

The 8 managers were given 30 minutes to check their page. At the end they reported the following major defects found by themselves individually:

Page 81 (three quarters of a page): 15, 15, 20, and 4 majors.

Page 82 (a full page): 24, 15, 30 and 3 majors.

Estimating the number of major defects found by the team

From the results of this input, we could estimate the number of unique major defects found by the **team**. First we had a hypothetical choice of either logging all the unique major defects (Using non-Agile methods, logging would take 3 minutes for each defect resulting in a 3 hour job), or estimating the result approximately. Not surprisingly, the managers choose the quick estimation. To estimate the number of unique majors (that is, the number of majors that are not duplicated - so if the same defect is found by more than one checker it only counts as one defect); we can double the count of the largest number of majors found by one individual in a small (2-4 people) group. This is based on observations done at Cray Research (Gilb and Graham 1993 pp. 299-301). From personal experience, it works well. In this case, this means that the group working on page 82 had about (2 x 30) 60 majors per page found (±15 majors of course). The group working on page 81 had about 40 total unique major defects they could log if they so chose to log them in detail.

Estimating the total number of defects per page – including those NOT found by the team

But of course inexperienced checkers do not find 100% of the majors defects present - they find only about a third. Remember even experienced checkers carrying out *source code* inspections peak at source code bug-finding effectiveness of 60% (Gilb and Graham 1993 reported by IBM MN), and most groups are not that good. Requirements and design checking tends to have an effectiveness ranging between 30% and 80% or more, depending on a wide range of process factors. These effectiveness factors include, speed of checking, available related project data, use of standards and checklists, and intelligibility of the specification being checked.

Can we verify the level of checking effectiveness in practice?

If you want to prove these estimates, the proof is simple: carry out an inspection, and then remove the major defects you have identified. That should leave twice that number estimated remaining – the two thirds NOT found by checkers (In this example, 80 major defects for page 81, and 120 for page 82). This sounds incredible. How could people miss so many on a single page? The proof comes when you repeat the checking process, and predictably find one third of the remainder (one third of 80); and can prove they *were there* on the first checking pass. Skeptics turn into believers at this point. We have carried out this test on our courses for years, and it always proves the case.

So, how many major defects are there in total on the page?

In this case, the managers accepted my assertion – that the 60 majors on page 82 were an indication of about 180 majors in the page (and 150 majors on page 81, indicative of the same density as page 82). Now this indicates an average of (120 + 180)/2 = 150 majors per page. I asked the managers if they felt this was probably typical for the other ('functional') pages. They said they had no doubts that it did. If managers are skeptical, the solution is simple, take another sample at random. I can assure you that the result found for defect density will be essentially the same order of magnitude.

Then, how can we estimate the total number of major defects in the specification?

Now this leads us to an estimation that we have about 150 (average per physical page) x 82 (total pages) = 12,300 Majors in total. I was initially quite shocked on calculating this number. But the managers were for some strange reason, not as skeptical as I was. I did not know anything about the project beyond that the requirements were just handed to me 45 minutes earlier, and that the managers were somehow responsible.

How many bugs will be generated as a result of these specification major defects?

But let's carry on with the calculations! Now another factor that has to be taken into consideration is that not *all* major defects in specifications lead directly to bugs. The problem being that we don't know exactly *which* of the major specification defects will *actually* cause bugs to be inserted - that depends on the 'sleepiness of the programmers on the day'! Two pieces of research I recall showed that 25% to 35% of the majors actually turn into bugs. For example, to make this plausible, a random guess as to the correct interpretation of an ambiguity with 2 options would give a 50% chance of a bug and 50% not. I have found that a good rule of thumb, that correlates well with observed reality, is that one third of the major defects will cause bugs in the system. So, for this example, that implies that about 4,100 (= 12,300/3) bugs will occur.

What do these major defects cost in project terms? How do they delay the project?
One of my clients (Philips Defence, UK, see the case study in (Gilb and Graham 1993 page 315)) studied about 1,000 major defects found in specification inspection of a wide variety of systems engineering specifications. They determined that the *median* downstream cost of *not* finding the majors would have been 9.3 hours (range up to 80 hours). So I use 10 hours as a rough rounded approximation of the cost of a major if it occurs downstream (at test and field stages).

Well, in this case study, that implies 41,000 hours (10 x 4,100 defects that hit us) effort lost in the project through faulty requirements. I was quite shocked at the implication of this quick estimate based on a small sample. But the managers were quite at home with it. They responded, "Don't worry Tom, we believe you!"

Why? I asked. So they explained, "Because (and we know you did not have any inkling of this) we have 10 people on the project, each using about 2,000 hours per year, and the project is already 1 year late (a total of 20,000 hours) and we have at least one more year of correcting the problems before we can finish."

CASE STUDY 3: AN AIR TRAFFIC CONTROL PROJECT (in Sweden/Germany)

Another client had a seriously delayed software component for an air traffic control simulator. The contract dictated about 80,000 pages of logic specifications. The supplier had written and approved about 40,000 pages of these. The next stage for the logic specifications was writing the software.

The divisional director, Ingvar, gave me the technical managers for a day, to try to sort out the problem. These men had each personally signed off the 40,000 pages. We pulled 3 random pages from the 40,000 and I asked the managers to find logic errors in the specifications – errors in the sense that if coded the ATC system would be wrong. Within an hour of checking, they found 19 major defects in the 3 sample pages. They agreed these pages were representative of the others.

That evening, Ingvar took 30 minutes to check the 19 defects personally, while his managers and I waited in his office. He finally said, ‘If I let one of these defects get out to our customer, the CEO would fire me!

Now the 19 defects found in the 3 pages represent an actual defect density of approximately three times that (that is, they probably did not find two thirds of the existing defects). So the managers had signed off about (20 x 40,000) 0.8 million bugs. And they had only done *half* the contracted logic specification. Well, the *sample* told us a *great deal*.

We started thinking that afternoon about what could have been done better. The conclusion was that we had a ‘factory’ of analysts producing about 20 major defects per page of ATC logic specification. We also concluded that if we had taken such a sample earlier, say after the first dozens of pages written, we might have discovered the systemic defect-density pollution-rate earlier, and have hopefully done something about it.

Too bad that they did not have Agile SQC! The project got completed; but only after being sold off to another corporation. The director lost his job, and it was not just for a single defect.

The irony was that when I first met the director, he told me he had read a book of mine. Too bad he did not practice what he read. His corporation, I later realized, had a bad ingrained habit. They did not review specifications until all pages were completed.

I asked the manager who signed the third signature on the specification approval, why he signed off on what we all acknowledged was a tragedy. He told me it was because ‘the other managers signed it ahead of him’. I guess that is when I lost faith in management approvals.

AGILE SQC ESTIMATIONS AND CALCULATIONS

At this point, it is worthwhile summarizing the overall Agile SQC process of estimating and calculating. See Figures 3 and 4, which show how to arrive at the defect level for a specification and how to calculate the number of remaining defects in a specification respectively. The calculations shown are for yet another case study.

Agile Specification Quality Control (SQC) Form - An Example Filled Out

SQC Date: May 29, 200X. SQC Start Time: _______
SQC Leader: Tom.
Author: Tino. Other Checkers: Artur.

Specification Reference: Test Plan. Specification Date and/or Version: V 2.
Total Physical Pages: 10.
Sample Reference within Specification: Page 3.
Sample Size (Non commentary words): approx. 300.

Rules used for Checking: Generic Rules, Test Plan Rules.
Planned Exit Level (Majors per page): _______ or less.
Checking Time Planned (Minutes): 30. Actual: 25.
Checking Rate Planned (Non commentary pages per hour): 2.
(Note this rate should be less than 2 pages per hour)

Actual Checking Rate (Non commentary words per minute): __________
Number of Defects Identified by each Checker:

Majors: 6, 8, 3. Total Majors Identified in Sample: 17.
Minors: 10, 15, 30.
Estimated Unique Majors Found by Team: 16 ± 5.
(Note 2 x highest number of Majors found by an individual checker)
Estimated Average Majors per Page: ~16 x 3 = 48.
(*A Page = 300 Non commentary words*)
Majors in Relation to Exit Level: 48/1 (47 too many).
Estimated Total Majors in entire Specification: 48 x 10 = 480.

Recommendation for Specification (Exit/Rework/Rewrite): No exit, redo and resubmit.
Suggested Causes (of defect level): Author not familiar with rules.
Actions suggested to mitigate Causes: Author studies rules. All authors given training in rules.
Person responsible for Action: Project Manager.
SQC End Time: 18:08. Total Time taken for SQC: ________

Version: August 15, 2004. Owner: Tom@Gilb.com

Figure 3. A example of an SQC form filled out

Estimating Remaining Major Defect Density

Assumptions:
A logical page (page) is 300 non-commentary words.
Your SQC effectiveness is 33.3% and your SQC is a statistically stable process.
One sixth of your attempts to fix defects fail (One sixth is average failure to fix).
New defects are injected during your attempts to fix defects at 5%.
The uncertainty factor in the estimation of remaining defects is ± 30%.
Probable remaining major defects per page =
'Probable unidentified majors' + 'Bad fix majors' + 'Majors injected'
Let E = Effectiveness expressed as a percentage (%) = 33.3%

If 33 major defects per page have been found during SQC.
Probable unidentified majors =
Major defects total estimated 3 x Found Majors (33) = about 100 ±30

Bad fix majors = One sixth of fixed majors =
Of 30 attempted fixes, 5 major defects in each page are not fixed.
This is useful to recognize.
Even if you found all defects, 1/6 would remain after all were fixed.

Majors injected = 5% of majors attempted to be fixed = 1.5 major defects per page.
(this is not always calculated, since it is small, compared to the error margin)

Probable remaining major defects/page, *after fixing what we found in a sample* =

66 (not found) + 5 (not fixed) = roughly 71 remaining major defects per page.

Taking into account the uncertainty factor of ± 30% and rounding down to the nearest whole number gives 50 Remaining Major Defects per Page
(Minimum = 50, Maximum = 92 remaining major defects per page).

Figure 4. An example of calculating the remaining major defects per page

CONTINUOUS PROCESS IMPROVEMENT

Notice how towards the end of Figure 3 there are two questions concerned with analyzing the origin of the defects (that is, 'Suggested Causes' and 'Actions suggested to mitigate Causes'). The aim of these questions is to identify problems in the work practices that need correction. This approach is identical to Capability Maturity Model Level 5, and to the Defect Prevention Process (see discussion of Mays in (Gilb and Graham 1993)).

In the Raytheon Study (Haley et al. 1995, Dion 1993), this process improvement effort reduced rework costs, within about 7 years, from about 27% of all development costs, down to about 4%. Before that happened though, the individual discipline of software engineers actually following their existing (bad) processes, led to a reduction, in a year, from 43% rework costs to the 27% cited above. So there is lots of short-term improvement available by getting people to follow even simple standards.

Personal experience with SQC is that by merely motivating people to follow the simple rules of 'clear/unambiguous/no design' in requirements, we can reduce the number of major defects inserted into requirements by, in one case an average of 80 majors/page to about an average of 11 majors/page within 6 months. Corporate engineering measurements (Douglas Aircraft 1988) and other examples indicate that the individual rate of reduction of defect insertion is about 50% per learning cycle. So, in about 7 cycles of writing specifications and measuring defects, an individual gets to the exit level of less than one major per page.

SUMMARY

Agile SQC costs very little, but its effect on early control over injected defects is significant. It can drive defect injection down by one and then, with time, two orders of magnitude.

The key Agile SQC concept compared to traditional Spec Inspection methods is to *measure* by *sampling*, and use the information to *motivate* people to 'learn the rules' (that is, the standards and/or best practices), and *reduce their defect injection*.,

Traditional Spec Inspection techniques are doomed to high costs and low effects because:

- they can only hope to find about half the problems (Given 40-80% is the very best in practice);
- they spend approximately 3-4 hours engineering effort *per page* of specification (for full effectiveness)

REFERENCES

Bhandari, I., M.J. Halliday, J. Chaar, R. Chillarege, K. Jones, J.S. Atkinson, C. Lepori-Costello, P.Y. Jasper, E.D. Tarver, C.C. Lewis and M.Yonezawa, In-process improvement through defect data interpretation, *IBM Systems Journal*, Issue 1, Volume 33, page 182, 1994.

Dion, Raymond. July 1993. Process Improvement and the Corporate Balance Sheet. *IEEE*

Software. Pages 28-35.

Fagan. M. E, ‘Advances in Software Inspections’*, IEEE Transactions on Software Engineering*. Vol. SE-12, No. 7, pp 744-751, July 1986.

Gilb, T. and Graham, D., *Software Inspection*, Addison Wesley, 1993.

Gilb, T., *Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*, Elsevier Butterworth-Heinemann, 2005. ISBN: 0750665076.

Haley, T., B. Ireland, Ed. Wojtaszek, D. Nash, R. Dion. Raytheon. 1995. Raytheon Electronic Systems experience in Software Process Improvement. This paper is available on-line at http://www.sei.cmu.edu/publications/documents/95.reports/95.tr.017.html

Humphrey, W. S., *Managing the Software Process*, Addison-Wesley, Reading, MA, 1989.

BIOGRAPHY

Tom Gilb is the author of ‘Competitive Engineering: A Handbook for Systems & Software Engineering Management using Planguage’ (due June 2005), ‘Principles of Software Engineering Management’ (1988) and ‘Software Inspection’ (1993). His book “Software Metrics” (1976) coined the term and, was used as the basis for the Software Engineering Institute Capability Maturity Model Level Four (SEI CMM Level 4). His most recent interests are development of true software engineering and systems engineering methods.

Tom Gilb was born in Pasadena CA in 1940. He moved to England in 1956, and then two years later he joined IBM in Norway. Since 1963, he has been an independent consultant and author. He is a member of INCOSE.

Author Contact: `Tom@Gilb.com, URL www.gilb.com.`

The author is extremely grateful for editing assistance from Lindsey Brodie, Middlesex University, UK, on this paper.

Version: April 18 2005 13:23