Experiences of Applying SPC Techniques to Software Development Processes

Mutsumi Komuro
Hitachi Software Eng.
4-12-7 Higashishinagawa
Shinagawa-ku Tokyo, Japan
+81-3-5780-2085
komuro@hitachisoft.jp

ABSTRACT
Experiences of applying SPC techniques to software development processes are described. Several real examples to apply SPC in Hitachi Software are given. Measures, control charts, and analysis judgment are given. Characteristics of software development processes, their influence on SPC, and lessons learned when applying SPC to software processes are described. In particular, the importance of self-directed and proactive improvement is discussed.

Categories and Subject Descriptors
D.2.8 [Metrics]: Performance measures, Process metrics

General Terms
Management, Measurement, Reliability, Verification, Human Factors, Standardization.

Keywords
Statistical Process Control, Peer Review, Effect Analysis, Statistical Analysis

1. INTRODUCTION
Statistical Process Control (SPC) is known to be a powerful tool to improve processes to enhance quality and productivity[1][2][3]. In Japan many manufacturing companies employ TQM (Total Quality Management)[4] to apply this technique to their products and have achieved successful results. Although an increasing number of software companies are trying to improve their development processes[5] based on Software CMM® or CMMI® [6] which are much influenced by TQM, not too many companies seem to successfully apply SPC to their processes[7].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICSE'06, May 20–28, 2006, Shanghai, China.
Copyright 2006 ACM 1-59593-085-X/06/0005...$5.00.

One big difference between the manufacturing and software processes is that the latter is more human-intensive and creative. Because of this difference, each software development has unique characteristics and the process performances can vary very widely. Therefore we have to implement so many things before we can start applying SPC. For example, there are 25 process areas in CMMI and 21 of them are classified in the maturity level 2 or 3. That is, only remaining four process areas are concerned with level 4 or 5, where the SPC technique is typically supposed to use.

Of course, this difference has influence on all the improvement activities, including those at level 4 and 5. In my opinion, the key to overcome this difference is to promote bottom-up and self-directed improvement. In [7] six theses for process improvement were proposed, including Thesis2 “Developers are motivated for change; if possible, start bottom-up with concrete, “ and Thesis5 “SPI assumes cultural changes, so we need expertise from the social sciences.” In fact, bottom-up and self-directed improvement has always been much encouraged in Kaizen activities or TQM activities in Japan, which seems to be often overlooked in recent improvement initiatives.

In this paper we would like to share our experience of applying SPC to software processes. Our experience also suggests that bottom-up, self-directed improvement would be the key to success in software process improvement.

2. What is SPC?
Statistical Process Control (SPC) is a method of process management through statistical analysis, which includes defining, measuring, controlling, and improving the processes[1].

SPC is also known as TQM in Japanese manufacturing industry and regarded as one of the major factors why Japanese economy achieved such a success[4].

In TQM there are so-called seven basic tools or magnificent seven that are frequently used for statistical analysis, among which control chart is the most important. Typical activities in SPC include the following (i) detecting out-of-control situations based on control limit or other instable patterns associated with control charts, (ii) analyzing special cause for each out-of-control situations, and (iii) stabilizing the process by removing these special causes.
3. Applied Organization

This paper mainly describes our experiences of applying SPC techniques to software development processes in Hitachi Software Engineering or HSE, which is one of the largest software development companies in Japan. HSE develops various kinds of software systems in wide range of customers. HSE has been conducting process improvement movement since its establishment in 1970.

After the end of cold war, the competition in software industry became harder just like in other industries. In order to further improve its processes, HSE has employed CMMI® model [8] and have been conducting company-wide process improvement initiatives since 2001. HSE consists of four major groups and all of them already achieved CMMI® level 3 successfully [9][10]. One of the four groups called Industrial Systems Group or ISG further achieved [10] level 5 in 2004. Most of our experiences of SPC [11] were obtained through these improvement initiatives. The ISG is a group developing and providing information systems and services for manufacturers and distributors, including embedded systems and information appliances. The experiences described in this paper mainly concerns with improvement activities at the ISG.

4. Difficulties to apply SPC to Software Development Processes

It is reported that SPC can be successfully applied to several processes for software development, including monitoring & control, peer review, and test processes[1][12][13] [14]. But, as far as I know, SPC has not yet become widely accepted and commonly used in software industry. This is at least partly because SPC is traditionally so well adopted in manufacturing industry that there is some bias[15][15] toward manufacturing or product-centric activities when applying SPC. In general, software development activities are more process-centric than product-centric, which makes it difficult to apply SPC in a straightforward manner.

In Japan SPC or TQM is known to be a powerful tool to enhance product quality in manufacturing industry[4], but it is still not very popular in software industry. For example, our company Hitachi Software Engineering (HSE) is a subsidiary of Hitachi Ltd., which has strong background in manufacturing business, and our development processes were derived from those of manufacturing processes in Hitachi Ltd. Although there were some quantitative techniques traditionally used in testing process, most processes had not been statistically analyzed or controlled. Moreover some key tools and techniques had not been introduced, including control chart.

In [16] Kan pointed out that a straightforward application of SPC to software development is hard because “the process is complex and involves a high degree of creativity and mental activity.” On the other hand, Florac and Carleton[1] emphasized that it is important to include people, material, energy, and tools in the definition of processes, when dealing with software processes.

I believe that the reason why it is hard to apply SPC to software process stems from the characteristics of software development processes.

4.1 Characteristics of software development processes

In general software development processes have the following characteristics:

(1) human-intensive and creative

Software development processes are more heavily dependent on human activities and require creativity than those of manufacturing processes[17]. It follows that the measurement results of software processes can easily vary and hence become unstable.

(2) Involves multiple common causes

Related to the first point, there are various factors that can affect the performance of the processes, e.g., tools, methods, type of software, and development environment as pointed out by Florac and Carleton[1]. These factors bring about a multiple common cause system. Therefore the data obtained will be a mixed set of common cause results.

(3) Hard to obtain a large set of homogeneous data, especially those data which have important business value.

Unlike manufacturing processes where many parts are daily produced, software development processes usually result in a relatively small set of work products. Furthermore each process execution is a creative and unique activity, which makes it hard to obtain large set of homogeneous data.

Despite of all these characteristics, SPC is still important in software processes. As described in later sections, SPC can give us guidance to stabilize and improve processes. It can also be used to demonstrate the improvement effects.

In our opinion the key to overcome these difficulties is to put more emphasis on processes rather than products. For example, using not only the product data but also process data can increase the available data significantly. Under the assumption that the definition of software process involves people, tools, and working environment as suggested by[1], SPC can help to stabilize human activities. It also helps to identify and separate multiple common causes.

5. Our experiences

The difficulties described in the previous section have influence on the way we apply the SPC technique to software development processes. In this section we will explain this issue through our experiences.

It does not make sense to apply SPC techniques to entire software development processes, which might burden both projects and the organization with measuring activities such as collecting, analyzing, and managing processes. We need to consider the effect of applying SPC from the business point of view, when we apply SPC just like other improvement activities.

5.1 Which process to apply, relationship to business goal

One area of software processes where SPC can be applied is test process. Since test processes check the final work products of the software development, applying SPC to test process seems to be
natural, when you are seeking the analogy with the manufacturing processes.

In HSE we are using various metrics for test process[18], including bug rate and test progress S curve. The bug rate is a defect density at the test process, i.e., defined as (# of bugs identified at the test)/KLOC. Test progress S curve[16] is a graph showing the cumulated number of checked test cases. The bug rate is useful to check the product and process quality of the test process and the test progress S curve is used to check the progress of the test process.

As a preliminary study for applying SPC, we first tried to establish baselines of the bug rate. Traditionally each department has its objective value of the bug rate. This is natural, because each department has different customer and different development environment. In order to establish baselines, we used control chart to detect out-of-control data. Since the bug rate is defect density at testing phase, we chose Z-chart which assumes Poisson distribution.

A typical result is demonstrated in Fig 5.1, plotting bug rates of the released products within a department. The value of each bug rate $b_i$ is normalized by the estimated standard deviation as $(b_i - \bar{b})/\sigma_i$, where $\bar{b} = \text{average of } b_i$'s, $\sigma_i = \sqrt{\frac{\bar{b}}{s_i}}$, and $s_i = \text{size of the } i\text{-th product}$. In Fig 5.1 there is one out-of-control point which is circled. We investigated each such out-of-control point and identified the (special) cause of variation. In most of the cases, the identified causes concern with earlier phases than testing, such as requirement, design, and review, which suggests that these processes are more important than test process in order to control quality.

Another analysis we performed was the relationship between defect removal phase and the quality. In HSE we are using a metric for source code called early bug detection rate, which is defined as the percentage of bugs detected before the unit testing phase in all the bugs.

Figure 5.2 shows the negative correlation between early bug detection rate and bug rate. This graph demonstrates that the higher the early bug detection rate is, the lower the bug rate becomes. It follows that the more you put efforts into earlier phase bug removal activities such as peer review, the higher the quality of work products become. This result also suggests that controlling earlier phase than testing is more efficient way to achieve high quality.

These results coincide with more general principle [1] advocated by Shewhart and Deming: “Focus on the processes that generate the products and services to improve quality and productivity.” Source programs are generated by requirement, design, coding, and review processes. We consider these processes should be more important than test process, when we try to apply SPC technique. In other words, we need to be more process-centric than product-centric.

We realized the importance of SPC, when major groups achieved CMMI level 3 and started implementing higher maturity levels. At that time the ISG had already started ‘Code Inspection Movement’ which was an improvement initiative of peer review on source code. This initiative was started by the advocacy of the general manager, the top person of ISG. The aim was to improve the quality of the products, because peer review is known to be an effective process to improve quality in software development[19][20]. Since fairly good amount of performance data was available for peer review due to this initiative, it was natural for us to select the peer review process as the first target of applying SPC.

5.2 Peer Review Process, Measures, and Control Charts

In HSE peer reviews are conducted in order to assure quality of the work products, including various specification documents and the source code. The following measures are used to monitor and control the peer review processes: (i) Review Speed which is defined as the size of the reviewed work product per hours spent for peer review, (ii) Defect Density at peer review which is defined as the number of errors detected at peer review per the size of the reviewed work product, (iii) Early bug detection rate for source code peer reviews which is a ratio of the number of bugs detected by peer reviews in all the bugs. (iv) Review Efficiency which is defined as the number of errors detected at the peer review per man-hour spent at the review.
Table 5.3  measures used for peer review

<table>
<thead>
<tr>
<th>Measure</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Review Speed</td>
<td>(size of work product)/(hours spent at review)</td>
</tr>
<tr>
<td>Defect Density at PR</td>
<td>(# of detected defects)/(size of work product)</td>
</tr>
<tr>
<td>Early bug detection rate for PR</td>
<td>(# of detected defects at PR)/(all the # of defects)</td>
</tr>
<tr>
<td>Review Efficiency</td>
<td>(# of detected defects)/(time and effort spent at the review)</td>
</tr>
</tbody>
</table>

Review Speed is used to check how well the peer review is conducted, because it is known that the faster the review speed is, the more frequently the reviewers miss the errors.

Defect Density is primarily used to measure the quality of the work product. That is, the lower Defect Density basically means the higher quality of the product. But Defect Density can be affected by the way how the peer review is conducted. There is a possibility that Defect Density is small whereas the quality is bad, because the peer review is poorly conducted. In order to detect this situation, we check both Defect Density and Review Speed for each peer review. Poorly conducted peer review usually has higher Review Speed value.

Early Bug Detection Rate for PR is used to decide how much efforts a project should put into peer reviews on source code. This measure is important, because it relates business goals with activities of projects. In [21] similar measure is defined and called “yield.”

Review Efficiency is used to distinguish a good review method from a bad method in terms of efficiency. This measure is in a sense different from the above three measures. One type is “measure for control” and the other is “measure for improvement.” The former is used to control the process and usually associated with some objective value. Projects are required to use the former type of measures. On the other hand, the latter is used for analysis to identify possible improvement and not necessarily associated with some objective value. Projects are not required to use the measure of the latter type.

Review Speed, Defect Density, and Early bug detection rate are measures for control. But Review Efficiency is a measure for improvement. The rationale for this choice is because forcing efficiency is harmful: it might have bad side effect like trying to review too large work product at once, or counting trivial errors, e.g., typographical errors, into the defects detected at the review.

This choice is consistent with the general principle that the quality should have precedence over the productivity. Please note that this distinction between two types of measures is particularly important in improvement activities for software processes, because software processes are heavily dependent on human activities. The distinction may have psychological effect on improvement activities.

Review Speed and Defect Density are monitored and controlled by XmR and Z control charts respectively. Review Efficiency is analyzed by XmR chart. In order to detect special cause of variations, we use four kinds of tests:

Test1: A single point falls outside the 3-sigma control limits.
Test2: Two out of three successive values falls on the same side of, and more than one sigma away from the centerline.
Test3: Four out of five successive values fall on the same side of, and more than one sigma away from the centerline.
Test4: Nine successive values fall on the same side of the centerline.

The Tests 2-4 are not applied to mR chart of XmR control chart and S chart of X-bar-S control chart when the sample size is small, because in these cases the expected distribution of the data is non-symmetric.

These tests are almost the same as those explained in [1], but we modified Test 4 so that we check nine successive values in stead of eight in [1]. In Japanese Industrial Standard [22] nine values are used. Taking nine values seems to be more appropriate because it gives nearer value of possibility when you compute the possibility under the assumption that the underlining data distribution is normal.

In [4] and [22] several more tests are listed up. But we do not employ these tests for monitor & control, because that would increase the possibility of occurring error of first kind. We will discuss this issue later.

5.3 Example1 – Dealing with individual process data

The basic use of control chart is to detect special cause of variation and to stabilize the process.

Figure 5.4  XmR chart for Review Speed

Figure 5.4 shows an example of XmR chart plotting Review Speed values of document review. There is one error detected by Test 1, whose values are circled.
Figure 5.5 shows Z chart plotting Defect Density values of the same review data. The value from the same review is also circled. This value itself is not an error this time, but is relatively small compared to adjacent values. From Figures 5.4 and 5.5 we can conclude that this review was conducted too fast and there may remain a lot of defects in the reviewed document. Looking into the record of this review, it turned out that the reviewers tried to review too large amount of document, about three times larger than average. Therefore the corrective action was to re-review the document after divided into two or three chunks.

Figure 5.5 Z chart for Defect Density

Repeating this kind of analysis will help us to stabilize the peer review process. It should be noted that we dealt with individual process data in these charts. Since peer review is an example of human-intensive and creative activity, each review may have unique property. We would like to check the performance of each peer review, and take a corrective action as soon as possible.

5.4 Example2 – Dealing with multiple causes

The next example is an organizational level analysis performed at the very early phase of the (high-maturity) improvement movement. We used Review Efficiency to identify and distinguish different review methods.

Here we use XmR chart for organizational level analysis. Since adjacent pair of data are treated as one subgroup in XmR, it is crucial to align similar data to be adjacent. In order for that, we placed the data so that the data from the same projects are sorted by time sequence and the projects are sorted according to the similarity of their developing system, most of the time the order of business departments and sections.

Figure 5.6 shows the XmR control chart plotting Review Efficiency. The circled data is extremely out of the upper control limit.

Looking into the review record of this data, it has the following distinguishing characteristics:

(1) An experienced person was assigned as a moderator of the review and held a preparation meeting.

(2) Checkpoints for the review were discussed and established in the preparation meeting. More precisely the reviewers agreed to check on memory management of the source code at the review.

(3) The review method they chose was inspection-type. That is, the reviewers concentrated on finding defects and did not discuss other issues such as the way how to correct the defects.

These characteristics are similar to those of Selected Aspect Review and Inspections as described in [19]. Since there were some problems concerning operational definition of the measure, as is usually found for the initial analysis, it would be too much to say that all the extreme deviation can be attributed to these characteristics. But these characteristics seem to be at least one of the major causes of the deviation.

Since we identified the special cause for the deviation, we omitted that data and plotted XmR chart, again. There appear new errors which are circled.
Looking at the mR chart, there are two circled data. One is circled by a broken line and the other by an unbroken line. The unbroken line means it really exceeds the control limit and the broken line means it nearly exceeds the limit. It suggests that the moving ranges are not quite stable. Looking at the X chart, there are four data circled which show strong deviations detected by Test 1, Test 2 and Test 3.

Figure 5.8 shows the corresponding data in histogram. There are two or more peaks. There are four points separated from the big peak on the left hand side. These four points correspond to the circled data in Figure 5.4.

Looking into the review records, it turned out that these four data came from the reviews of the same project with the same moderator. They conducted ordinary walk-through type reviews and we could not find any distinguishing characteristics about the review method. The product the project developed was an embedded system and the project developed the system on some particular platform. All the reviewers are very experienced and knew both the product and platform very well.

Later we got new review data from this project and the Review Efficiency of the data was always higher than those of other projects. Although we could not identify any difference on review method, environment and performance of reviews were different. Therefore we classify the data from this project as a different category than others and established different baselines.

Classifying the data from this project into another category, we again plotted the rest of the data, all of whose review methods are classified as walk-through. Figure 5.9 is the result, which shows that the data are in stable status. Thus we established the first baseline for walk-through reviews. After this analysis we got new data from the projects. Plotting these data into the baseline, we found deviations from time to time. By analyzing the special causes of these deviations, we found the following best practices and worst practices about peer review method.

1. Practices of higher review efficiency
   - perform preparation of the peer review, two person review, three person review with experienced moderator, etc
2. Practices of lower review efficiency
   - Wait the peer review until the end of the development phase and review all the work products at once.

Interestingly enough, most of the special causes were about higher review efficiency. It follows that the review method used then had a lot of improvement opportunities.

### 5.5 Example3 - Further Effect Analysis

In section 5.1 we already mentioned that the review has positive effect on quality. Utilizing the baseline data demonstrated in the previous section, we can show the cost reduction effect of peer review.

We have already established a baseline for Review Efficiency of walk-through reviews. Similar techniques could be applied to
testing processes to establish a baseline for testing efforts. Unfortunately effort data for detecting and correcting each bug was not available then, even though the number of bugs and total efforts were recorded. We used regression analysis to estimate the effort spent for each bug. More precisely we calculated the best fit linear equation of type \( y = A x + B \), where \( x \) represents the number of bugs and \( y \) represents the efforts for testing. The idea is that as the number of bugs increases, the efforts should increase accordingly. The coefficient ‘A’ is this ratio and represents the average effort spent for each bug which we wanted to obtain. By the way, the coefficient ‘B’ represents the overhead of testing phase such as the cost to establish testing environment.

From these data we can compute and compare effort per bug for both walk-through reviews and testing. The result is that testing spent twice or more efforts than walk-through reviews did. Figure 5.10 shows this result. We can conclude that putting more effort on peer review, we can save more efforts on testing phase. Therefore peer review is useful for reducing development cost. It should be noted that walk-through reviews were not too efficient type of review methods. Thus we knew that if we improve the review method, we could save much more efforts.

### 5.6 Toward the Kaizen – Bottom-up and Self-directed Improvement

Encouraged by those analysis results at the early phase of improvement activities, we implemented process improvement of peer review as follows:

1. Revised the peer review process definition to include the best practices identified by the analysis.
2. Documented the effect of peer review performance on quality and cost reduction. Defined the objective values of early bug detection rate for PR both at the organization level and each project level.
3. Developed a tool to draw control charts for Review Speed and Defect Density so that each project can analyze the peer review performance data by themselves.
4. Made and delivered training courses on peer review whose topics include process definition, best practices, analysis results, measures, and the tool usage mentioned above.

(5) Provided analysis service for peer review performance in the case when the project was not familiar with the control chart tool in (3).

What we considered most important in this improvement movement was to promote bottom-up improvement activities from the projects as possible as we could. We believe that this is one of the fundamental principles of ‘Kaizen,’ the process improvement movement in Japanese manufacturing industry.

We never imposed or ordered the improvement Instead we tried to support the project to improve its peer review process by itself. For example there was a project which observed a decreasing trend in the control chart for Defect Density, investigated the (root) cause of the problem on their review process, and changed their review style from walkthrough to inspection. The project analyzed the problem and realized the improvement by itself, although inspection style review was recommended as one of the best practices in the training course.

There is actually a test to check trends in one of the Japanese Industry Standard[22], but we did not employ that test for our rule as stated in 5.2. The intension was to avoid occurrence of an error of first kind, i.e., a false alarm. We employed only four tests, because it would increase the possibility of false alarms if we employed too many tests. Instead, we encouraged the projects to watch the peer review process, if there is any pattern which looks unusual in the control chart. We left the detailed action to the project, because we believed it would enhance their motivation. Projects can see the performance difference through the control chart, when they implement any improvements. It is a fun to see the improvement result visually, which led to proactive actions of the projects.

### 5.7 Discussion and comments

We believe that these examples show that SPC is applicable and useful in software development, too. There are some tips and comments, though.

1. Use not only product measurement but also process measurement
   
   This is particularly important in software development, because using the process measurement helps to stabilize the processes including human-intensive, creative activities. In general, the activities in earlier development phase have more important business values and in order to control these activities the process measurements are crucial.

2. Handling individual data becomes more important. For example XmR control chart is more useful than ever, while X-bar-R seems to be useless.

Since it is hard to obtain a large set of homogeneous data in software development, those charts which can plot the individual data are more useful. Thus, XmR, Z or u-charts are useful. On the other hand, X-bar-R chart which was most popular chart of all in TQM seems to be useless for software development processes, because it usually assumes constant size of subgroups. X-bar-S can be used for effect analysis, for example, because it can be used well with variable size of
subgroups[2] and has more accurate control limit than X-bar-R.

(3) Put more emphasis on stability of the data rather than the amount of data
As described in example 2, sometimes we have to separate the data set into different categories to obtain stable data. Common confusion we found about SPC is that people often assume that they always need a very large amount of data. What we want is to establish baselines and models to be able to predict the process performance. Having a large amount of data often helps to establish a baseline in single cause system, but in multiple cause system you can not establish a baseline without separating the data set.

(4) Consider and be aware of the psychological effect
Since software development processes are human-intensive and creative, we need to consider the psychological effect when we conduct improvement movement. This implies that training on processes or SPC is important. You should also have good strategy to convince people about the effectiveness of improvement. In our experience, using and analyzing actual data from the organization is very useful and effective approach.

Another thing we tried was to motivate people to perform improvement activities by themselves and to avoid imposing the activities. To improve is a fun and people have some desire to improve. Providing a tool to visualize the improvement effect (through control chart) is one example to support them.

6. Lessons Learned
In order to apply SPC to software development processes, we need to address several issues including (i) controlling human-intensive and creative activities, (ii) dealing with multiple causes, (iii) analyzing and controlling with only a limited number of available data. Despite of these difficulties, there are evidences that SPC has positive effects on the quality and the productivity (cost reduction) of the products. In order to overcome these difficulties, we need to be more process-centric rather than product-centric.

Since the software development processes are human-intensive and creative, taking the psychological effect (motivation) into consideration is also important.

Alignment with business goal is the key to the successful process improvement. SPC can provide navigation to which direction we should improve our processes in a measured way.

7. REFERENCES