

# Analysis of Inspection Defect (and other) Data

C. C. Shelley

Oxford Software Engineering  
9 Spinners Court, 53 West End, Witney, Oxfordshire, England, OX28 1NH  
shelley@osel.netkonect.co.uk

## ABSTRACT

*Software quality controls, and in particular software inspections performed during the course of software development provide defect data, which in combination with other data, can provide considerable information about the quality of the items inspected, the processes that produced the items, and the prospects for future development. These data can be combined, analysed and presented in many ways. These are reviewed and their limitations and validity assessed. The need for a limited, standard set of robust analyses is proposed together with a suggestion for extracting other information from inspections.*

## 1. Introduction

Software inspections are recognized as the primary software quality control and a major source of the most valuable software measure – defect counts. The inspection process enables the collection of data about the artefact inspected - the defect data primarily, as well as data about the context of the inspection, and the inspection itself. These data are irresistible subjects for analysis. Wide-ranging analyses are possible, but often the analyses are used beyond their applicability producing results of limited value and dubious validity. The uses of defect and supporting data are reviewed and the assumptions underlying their use questioned. There is a need for a review of the analytical techniques currently used.

## 2. Using Defect Data

Software inspections, and other software quality controls, are performed primarily to identify defects, to enable them to be removed. The data recorded from these activities are valuable in their own right.

They increase in value when combined with other data. Radice (Radice, 2000) identifies a minimum set of six data types that can be usefully combined: Lines of Code, or some other size measure; Defects; Engineer months, or equivalent effort measure; Calendar months; Cost; and Test progress. These data are often accompanied by supplementary information that further enhances its value, for example a defect may be accompanied by severity, location, injection point and type information.

Combinations of these data provide a rich source of information: Defect counts can indicate the amount of rework required to make the artefact useable, and the impact on development schedule. Defect ‘densities’ - the number of defects divided by the size - may give indications about the quality of the artefact. When compared with defect data and schedules from similar, previously developed artefacts they can provide predictions about the progress of the development project and the outcomes of future quality controls. The distribution of defects within an artefact can aid in the identification of problem areas or ‘hot spots’. Defect types and modes provide insights into the way the artefact was produced. Defect data are software engineering’s ‘laboratory rat’ providing insights into products, projects and processes.

## 3. SPC in particular

One type of defect analysis receiving particular attention at present is statistical process control (SPC)<sup>1</sup>. This is attractive to software engineers, and software process engineers in particular, for a number of reasons. SPC is one of the production engineer’s tools; and software engineering aspires to be recognized as an engineering discipline and to

---

<sup>1</sup> SPC can encompass a variety of techniques but increasingly is thought of as a synonym for run charts and control charts.

use true engineering tools. Software inspections have their origins in production engineering too, and provide data apparently suitable for SPC. Using SPC with software inspections (and other software QC techniques too) provides opportunities for gaining insights into, and control of, software production processes - and for this to show software engineering's maturation into a true engineering discipline.

Control charts have been developed to meet a wide range of needs and types of data. Two principle classes of control charts are used; control charts where the variable of interest is a continuous measure, and samples have a more or less normal distribution.  $\bar{X}$  R and s charts are of this type. The other type is used where the attribute of interest cannot be readily measured so attributes capable of being counted are used. p, np, c and u charts are of this type. These are described in Juran, (p45.1, Juran 1999).

The use of run charts and control charts in software engineering is discussed by Gilb (p194, Gilb, 1993), Kan (p143, Kan 1995) and Radice (p6-203, Radice 2002). These charts plot sequences or sets of inspection defect counts, and the variability in these activities, as indicated by the variation in the defect densities found, are used to indicate the degree of control over the process used to produce the items in which the defects have been found.

Examination of the use of control charts for software engineering, both proposed and actual, reveals some anomalies. Kan recognizes several limitations to the use of control charts for software engineering: difficulty in specifying metrics related to customer need; software development is not a production process; poorly defined processes; variant processes in use; and rapidly changing processes<sup>2</sup>. He proposes a more relaxed use of 'pseudo-control' charts. Similarly Radice (Radice, 2002) proposes a modified  $\bar{X}$ mR chart and the adoption of charts for both variables and attributes for tracking defect densities, dependent on user comfort. Instances of actual use of SPC type techniques also show difficulties in their adoption.

One of the first difficulties in the adoption of SPC is the selection of appropriate techniques. For example, defect densities would appear to be a continuous measure but this is an effect of the normalization process. The measure itself is discrete. Defects are counted, not measured. Attempts to select the correct type of control chart raise a number of

further questions about the applicability of run and control charts. For a control chart:

What types of quality control or inspection can be included, and what cannot; what are the necessary qualification criteria?

Defect counts from what types of artefacts can be plotted on a chart; how similar or dissimilar can they be?

What are the relationships between these artefacts?

What determines the sequence that the artefacts are plotted?

Assuming discrete data (counts), what degree of evidence is required to build confidence that the distribution is binomial?

Where normalized data (defect density, say) are plotted, what prior analysis is sufficient to demonstrate a linear relationship between the two attributes - and what steps are taken if this is not so?

What is a reasonable sample size when calculating the distribution's standard deviation, and so the control limits?

What process, or process instances, is the control chart intended to control, the production process or the defect detection process, or both<sup>3</sup>?

What value have control limits determined by standard deviation calculations when attempting to control software defect levels - and what action to control the process are to be taken when the limits are exceeded?

And so on. These questions are difficult to answer in general terms, and in specific instances too, because of the many confounding factors. It is difficult to be usefully specific about more than one inspection or quality control, or about the *development* processes that produced the artefacts - as noted by Kan.

This is in contrast to *manufacturing* production processes, guided by control charts, where the items produced are the result of a replication process; where the relationship between manufacturing processes, and the (nearly identical) items they manufacture can be described in terms of production sequences, batches or samples. The variables or attributes of interest of the items produced can be well characterized by mature measures, or by robust

<sup>2</sup> Software development processes, being performed by people, are also self-aware and can react unpredictably to data about themselves in ways manufacturing processes do not.

<sup>3</sup> The data best suited to inform process improvement is not defect counts found by inspection (or other development QC) alone. It needs defect data recorded after the software is released to production or the market.

quality attributes that lend themselves to the analytical techniques of production engineering.

#### 4. The uniqueness of design artefacts and the processes that produce them

In contrast in software development it is reasonable to consider each development process as unique, with different people involved, with difficulty to characterize skills and priorities. The production of a design artefact is a unique event and so, often, is its quality control; each artefact is necessarily unique too. Variation in size and complexity of the artefacts can have non-linear effects on defect levels that complicate the interpretation of standard control charts. There can be many similarities between development processes and the produced artefacts of course, but not in the same way, and not to the same extent as production line processes, which is the replication of nearly identical items (or items with closely specified differences). In production engineering deviations from production norms are undesirable, where uniformity is the aim, and can be analysed and managed systematically. Development artefacts cannot be treated in this way. Where significant deviations from presumed norms are identified they can be problematic, perhaps signalling development or quality control problems, but maybe not. In any case the data is available after the (unique) event. And in the case of defects counts the norm is not acceptable; the intent is no (detected) defects – and this is actually achieved after the defects have been identified and rectified. Where defect data deviates from the norm it signals potentially interesting events that prompt further investigation of that artefact, but not necessarily of the processes that produce those types of artefacts in general. Defect data from development processes is not production control data; it is exploratory data. It is this that leads to the need to modify standard control charts for development artefacts; for Kan's relaxed 'pseudo-control' charts and Radice's selection of SPC techniques based on user comfort.

#### 5. Production Data and Development Data

While the data produced by software quality controls is capable of being analysed by SPC-like techniques this use implies an often unwarranted degree of understanding of development processes and the artefacts they produce. The techniques can also, having been designed for process control, obscure more than they reveal. The control limits may be spurious given that the sequences or sets of defect data from which they are derived can themselves be limited and arbitrary. Control limits or tolerances applied to defect levels are best determined by the consequential costs of the defects,

not uncertain distributions about an inappropriate median. The selection criteria for artefact types to be included in a 'run' will often limit the number of instances to a level that severely reduces any value a run or control chart may have.

So what analytical techniques should be used? Accepting that development processes, and development quality controls, are more complex, difficult to characterize and variable than production processes suggests that techniques that recognize their uniqueness and that of the artefacts they produce, that allow wider ranging investigation of the data and are less constrained by limiting assumptions, would be more appropriate. These techniques are readily available and well known. They are especially well suited to the investigation of software development processes, are more robust, requiring fewer assumptions about the data than specialized production control methods, and they provide more and better information.

#### 6. Analytical techniques for software processes and products.

Like SPC, the techniques that are best suited to the analysis of software development processes and software artefacts are well known to production engineers. They can be found among the seven tools of TQC. The seven tools are:

The process chart

Pareto analysis

The Ishikawa diagram

The histogram

The scatter diagram

The control chart

Check sheets (or tally sheets)

(Other techniques are occasionally listed, Imai (Imai 1986) includes 'graphs', but does not mention process charts.

The two techniques for particular interest to software process engineers are the histogram and the scatter diagram or scattergram. Used well these do all that is required, allowing the data to reveal information in the most effective manner. They are the basis of Tukey's Exploratory Data Analysis (Tukey, 1977). As well as being widely applicable and appropriate for the analysis of complex, poor characterized, non-parametric data the techniques are simple and can be used without error by staff

with little background in statistical analysis or measurement<sup>4</sup>.

Compare the use of the control chart and the histogram and scattergram:

A typical control chart will show normalized variations in defect numbers for a number of artefacts. First, the control chart type should be selected – a standardized c or u chart may be the most appropriate technique. Then the number of samples or instances required for good control limits should be established. This is usually set at about 20 to 25. Now ensure the distribution is, more or less, binomial. Plot the defect densities for each artefact. As the plot proceeds set the control limits. For a u chart the control limits are given by:

$$\bar{u} \pm 3\sqrt{\bar{u}/n}$$

where  $\bar{u}$  is the average number of defects and  $n$  is the average ‘size’ of the item, say LOC. The plot shows the deviation from the median and instances where defect levels exceed the limits, requiring action. Consider whether the definition of ‘control limits’ may need revision for the particular needs the artefacts; setting the norm to zero establishes a target of zero defects, but may mean that defective quality controls are neglected – no QC, no defects. Setting up a control chart requires care, is error prone and may obscure important information.

Using a histogram: plot the number of defects found for each artefact on the y axis, for each instance of quality control of artefacts listed on the x axis. That’s it. The data is un-normalised so larger artefacts will probably show larger number of defects – revealing variability that normalized defect densities obscure. These absolute numbers of defects indicate the amounts of rework, delay and cost. Should the relationship between artefact size and number of defects need to be shown use a scattergram with size (LOC) on the x axis and number of defects on the y axis. Correlations are revealed together with interesting outliers. No control chart type selection is required, there are no sample size limitations, and no control limit calculations are performed, and no ad hoc revisions to the techniques are needed. With absolute, un-normalized, values shown the variation in the nature of the artefacts, and the relationship between size and defect numbers is revealed. The analysis is simpler and more informative.

<sup>4</sup> This is important. Analytical errors are readily detected by potential users of the data, giving rise to uncertainty or discrediting of the data, even if the error itself cannot be explicitly identified and corrected.

Histograms and scattergrams can be used to display, without limitations or constraints, most of the analyses required by software engineers and software process engineers:

Comparative quality - defects vs LOC, scattergram

Comparative rework - defect number vs artefact , histogram

Comparative defect modes/type - defect numbers vs mode/type, histogram

Prediction of defect levels in successor artefacts – histogram of defect numbers vs phase or successor artefact.

Etc.

Many other analyses suggest themselves and can be performed quickly and without error. In each case the data are presented simply and directly with little in the way of mathematical operations, or data and sampling constraints. The data present themselves. In essence, histograms and scattergrams let the figures do the talking. Patterns or trends revealed may suggest further plots, using histograms or scattergrams of course, that provide deeper insights into development activities and the artefacts they produce.

It is proposed here that histograms and scattergrams are promoted as the primary (graphical) analysis tools for software engineering data and that the use of control charts is discouraged – despite the desire of software engineers to use production engineering techniques and tools. The use of simple but powerful traditional graphical techniques that minimize mathematical operations and allow the data to show patterns and trends suggests other graphical techniques that allow numerical data to reveal useful information. These additional techniques should be viewed as supplementing histograms and scattergrams, not as replacements or alternatives - unless this is clearly indicated by the data. Some additional techniques are considered below:

Box plots are another of Tukey’s Exploratory Data Analysis techniques. They are a development of the ‘range bar’ used to compare groups of data. For each group a data point on an x,y plot is made to show the median value, a bar through the point indicating the range and a box indicating the inter-quartile range. (The ‘stock’ chart types in Excel show what they look like.)

A particularly ingenious use of graphics that lets numbers reveal information and that incorporates a histogram is Jeremy Dick’s ‘fault grid’. It is a technique for presenting information about software



The arcs connecting the nodes could also be subject to this treatment too, although that is not shown here.

The assumption when viewing this diagram is that the data is 'actuals'. This need not be the case. Pictures could be drawn to show planned or anticipated attributes, allowing engineering intuition to be expressed quickly and easily and captured as numbers.

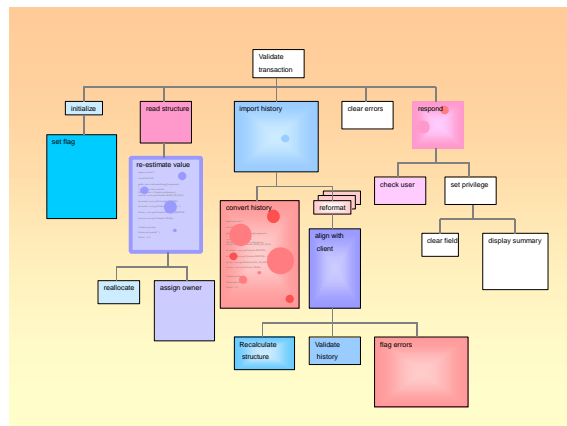


Fig. 3. ...integrated with numerical data.

Readily available technology also provides opportunities to animate the data – to enable programmers and managers to watch the development of a system or the progress of a project planned or actual or both...

## 7. Additional data from software inspections?

The data of primary interest from software inspections is defect data. Analysed with care it can provide information about the software, and the projects and processes that produced it. The value of defect data to software engineering is not surprising. Defect data - whether as defect counts or of measures of deviations from a norm - is also the primary source of information for process control and revision in production engineering.

However there is a difference. Like manufacturing defects, software defects give rise to rework, delay, increased cost, system failures and customer dissatisfaction, but the uniformity, consistency and conformance to standard, so valued in manufacturing's essentially replication processes, are not the only criteria by which software is judged: software development is design; and, as in manufacturing design, excellence is the driver, not uniformity.

Excellence is not assessed solely by the number of defects. A good design will have a certain number of defects, but an excellent design is not defined by fewer. There is something else. Whatever this is it isn't assessed by quality control. But software quality control performed early in the development cycle, particularly inspections, is not quality control alone. It is an aware process performed by people that, as well as recognizing defects, may also recognize excellence: ingenious solutions to difficult problems; prior solutions; opportunities, and the elimination of complexity. This is recognized as one of the additional benefits of software inspections but does not appear to be formally managed or measured. In fact current practice may actively obstruct the capture of this information. Discussion of design options is discouraged, the focus to being fixed on detecting defects alone. This may be a reflection of the origins of inspection as a hardware defect detection technique, used to improve consistency and uniformity.

Software inspections can become even more valuable when adapted for use in a design process; when they are supplemented by a requirement to recognize, acknowledge and record those rare points of excellence that surprise and please (and give rise to the occasional twinge of envy). Like defects these points should be recorded, counted and analysed.

Similar counts and analyses of ingenuity could identify the nature and perhaps sources of excellence. (There doesn't appear to be useable antonym for defect.) The analysis of points of excellence (undefects? proeffects? profects perhaps?) would be similar to that of defects but, unlike defects, the emphasis would be an amplifying or reusing them.

The removal of defects, by reducing rework, delays and dissatisfaction is a major contributor to the delivery of high quality software, but is bounded by the lower limit of zero defects; the amplification of excellence has no limit.

## 8. Closing remarks

Software engineering owes a considerable debt to production engineering for the techniques and tools it has provided, but these tools have been developed for specialized needs and come with limitations and constraints on their use. Where these limitations are unrecognised the benefits they could offer are often not realized.

Software engineering needs its own analytical tools and techniques. By looking more widely for simple and robust techniques, fitted to nature of software development and its products, and by exploiting the

technology it has itself created, to allow investigation and exploration, more rapid dissemination of sound data analysis and improvement techniques can be expected together with faster development of software engineering as a distinct engineering *design* discipline.

## 9. References

Gilb, 1993: Software Inspections, by Tom Gilb and Dot Graham, pub. Addison Wesley.

Imai, 1986: Kaizen, by Masaaki Imai, pub. McGraw Hill.

Juran, 1999: Juran's Quality Handbook, 5<sup>th</sup> Ed., pub. McGraw Hill.

Kan, 1995: Metrics and Models in Software Quality Engineering, by Stephen H. Kan, pub. Addison Wesley.

Radice, 2000: Software Inspections: a Case Study, by Ronald A. Radice, unpublished paper.

Radice, 2002: High Quality Low Cost Software Inspections, by Ronald A. Radice, pub. Paradoxicon Publishing.

Tukey, 1977: Exploratory Data Analysis, by John W. Tukey, pub. Addison Wesley.