Process Modelling to Support Dependability Arguments

Robin Bloomfield[1,2] and Sofia Guerra[1]

[1]Adelard, [2]CSR City University
Drysdale Building, 10 Northampton Square
London EC1V 0HB, UK
{reb,aslg}@adelard.com

Abstract

This paper reports work to support dependability arguments about the future reliability of a product before there is direct empirical evidence. We develop a method for estimating the number of residual faults at the time of release from a "barrier model" of the development process, where in each phase faults are created or detected. These estimates can be used in a conservative theory in which a reliability bound can be obtained or can be used to support arguments of fault freeness.

We present the work done to demonstrate that the model can be applied in practice. A company that develops safety-critical systems provided access to two projects as well as data over a wide range of past projects. The software development process as enacted was determined and we developed a number of probabilistic process models calibrated with generic data from the literature and from the company projects. The predictive power of the various models was compared.

1. Introduction

There are a number of motivations in developing models of the software development process such as process improvement, project management, cost prediction and reliability prediction. In this paper we report work that is motivated by the desire to support dependability arguments about a system. These include arguments about the future reliability of a product before there is direct empirical evidence to support the claim and arguments of fault freeness that to a very high probability no faults are contained in the product. Alternatively, we may wish to use the model to assess other people's arguments of fitness for purpose when they have followed standards such as DO178B or IEC61508.

In [1] we developed a novel theory of reliability growth modelling that enables conservative long term predictions and provides the difficult and unusual link between the development process and long term reliability. The theory states that (given some assumptions) a worst case reliability bound can be obtained from the estimated number of residual faults at the time of release (N) and the operating time (T). The expected value of the mean failure rate is bounded by N/eT, where e is the exponential constant. There are versions of the model for dealing with problems in fault reporting and to changes to the software.

This theory shifts the problem of reliability prediction from estimating reliability directly to one of estimating the value of N. One method for estimating N is to perform a detailed analysis of the software development process. In this method, we develop a "barrier model" of the development process, where in each development phase faults are created or detected. This model is parameterised by the rates of fault creation and detection at each phase of the software lifecycle. It can then be used to estimate the number of residual faults (i.e. those that escape the last barrier).

In this paper we present the work done to demonstrate that this approach can be applied in industrial applications. The process modelling technique has been applied to the software development process of a company that produces safety critical products. From the reliability quantification viewpoint, the main purpose of process modelling is to estimate the number of residual faults (N) in the operational system, and hence predict reliability bounds for the software in field operation. While the *model* is indeed conservative, in practice confidence in the conservativeness of the *predictions* will be based on confidence in the model parameters and the underlying assumptions. A conservative *model* does not necessarily give conservative results: a case by case justification has to be made for the confidence in the predictions.

This work augments the qualitative descriptive analysis of software development processes that might be produced by applying a Hazop style approach to the development process. The difference is that we provide a quantitative analysis of the development process and seek to justify the results from a statistical analysis of the process. The method does not rely so heavily on the use of expert opinion on the efficacy of methods. Another important difference is that we use the process as enacted for the modelling, not as described.

The approach described in this paper provides one solution to the requirements of safety critical standards to justify the software development process (e.g. Clause 7.4 of UK Def Stan 00-55). Also in IEC 61508 there are many techniques and measures proposed (See Part 3 and the normative Annex A) and the standard accepts that it is not possible to give an algorithm for combining them. The process modeling approach provides assistance in justifying the combination of techniques and the requirement to state them in the safety plan.

In this study, the company supplied data from two projects. The project data was analysed to reconstruct the software development process. Using this basic process structure, a range of process models was developed that were calibrated either with generic industrial data or detailed project specific data. The models predict the number of faults at different stages, which are compared against reported field faults and industry generic data. Further investigation was carried out to establish whether fault creation and detection efficiencies derived from one project could predict the performance of other projects in the same company.

The paper is organised as follows. Section 2 gives the background of the work. Section 3 describes the analysis of the process and data. Section 4 describes the different models developed, and Section 5 their results. Section 6 discusses the work and its results. We conclude in Section 7.

2. Barrier model

Process profiling characterises a software development process by measuring the fault discovery both in successive development phases and in operation. This provides a profile of faults detected at each phase. Given a database of past project profiles, the data from a new project can be re-scaled to give estimates of faults detected in later phases and in operation. This approach is related to that used in the Japanese software factory [13], [14] and in projects that investigated high integrity development processes using Bayesian Casual Networks [12] and the more recent work reported in [15].

We have developed a more generic approach where we construct a “barrier model” of the software development process. In the barrier model, a parameterised model is constructed to provide estimation for the number of residual faults N and to assess the quality of the development process with respect to industry norms. A barrier model is developed where each development stage both introduces errors of different types (e.g. requirements, design or code) and includes review or testing to detect errors. This is illustrated in Figure 1.

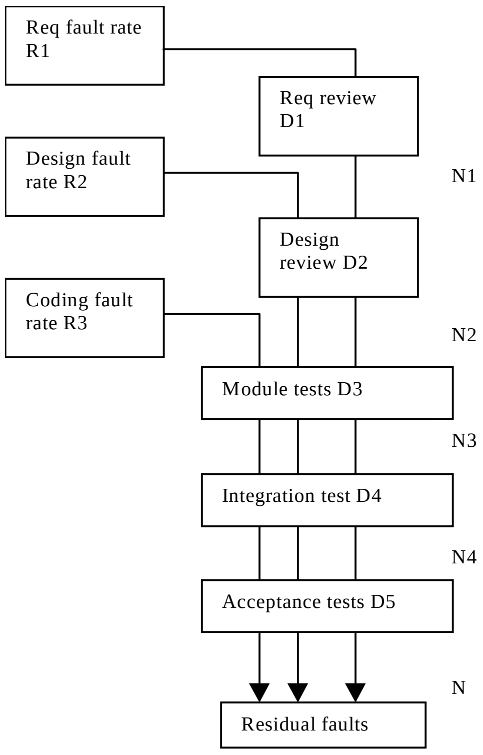

Figure 1: Barrier model

The process model can be used to predict the number of residual faults and the faults detected at intermediate stages. The data required for the model are the fault creation rates per 1000 lines of code at each stage (R1, R2,...) and the detection efficiencies of the various barriers (D1, D2,…). In practice the barriers will have different detection efficiencies for the different types of faults. For example, module testing should have high detection efficiency for code faults, lower efficiency for design faults (e.g. spotted when devising module tests) and an even lower efficiency for requirements faults. Therefore, a set of detection efficiencies are used for the different fault sources, e.g. $D1_{req}$, $D2_{req}$ $D3_{req}$..., $D2_{des,}$ $D3_{des,}$ $D4_{des}$.... etc. The number of faults revealed at each stage is the combined output of these fault streams as they pass through the barriers, e.g.:

$N1 = R1 \times D1_{req}$

$N2 = R1 \times (1- D1_{req}) \times D2_{req} + R2 \times D2_{des}$

$N3 = R1 \times (1- D1_{req}) \times (1-D2_{req}) \times D3_{req} + R2 \times (1-D2_{des}) \times D3_{des} + R3 \times D3_{code}$

After the final stage there is no further barrier, so the number of residual faults (per kloc) is the sum of the different fault streams, attenuated by the sequence of barriers:

$N = \sum R_i \prod (1-D(j)_i) \quad (j \geq i)$,

where i represents the fault source, j represents a barrier, R_i is the fault rate of source i, and $D(j)_i$ is the detection efficiency barrier j for faults of type i.

For any new process, the model parameters, *D* and *R*, have to be determined for all project stages. These can be derived partly by monitoring the process or using data from similar projects. For a novel project it may be necessary to make use of generic data for fault creation and detection. Publicly available industrial process data was collated to support this aspect of the study.

Software systems with similar fault densities can exhibit very different operational reliabilities. This is taken into account in the use of the conservative bound for the MTBF: the conservatism can be extreme but in many cases it is within a factor of 3-10 [1].

There is a balance to be made in the amount of modelling detail used and the possibility of model validation. Clearly software development uses a whole range of skill, rule and knowledge based activities that are dependent on the individual and the context in which the work is undertaken. Even if such detailed models are valid in theory – and many human performance models are heavily debated – we would not be able to find sufficient fine data to calibrate and parameterise them. We side step the problem of such detailed explanatory modelling by using sufficient structure to predict the variables of interest and to use probability density functions that describe the range of performance which may be seen in practice. This is somewhat akin to the modelling in the finance sector where human behaviour is characterised by similar empirical distributions. Moreover, we are modest in our ambitions only seeking to make plausible ranges or bounds on the prediction. For example, in safety arguments often we only seek figures within an order of magnitude.

There are a number of tools for implementing such models. Candidates might be those based on BBNs such as Hugin [17]. However, given the continuous nature of the distributions we use and the problems of validating the conditional node tables in BBN based models, we opted for the tool Analytica [9]. This tool allows parameter uncertainty to be expressed as a probability distribution. Analytica can generate predictions for the number of faults detected at each stage, and the proportions of faults from different sources (requirements, design and coding). The uncertainty is propagated from the initial parameters through to results, where the predictions of residual faults are also represented as probability distributions.

In order to use the model in a specific project it was necessary to discover the software development process to obtain the barriers and phases of the model, and to collect data on creation or detection rates.

3. Process discovery and analysis of fault data

We looked in detail at two projects from the same company that we call Project A and Project B. The projects were dissimilar in both the type of project and the type of data that was available.

The first project was a small but realistic project to assess the feasibility of the barrier model approach. It had the advantage of not reusing any code previously developed. This allowed the analysis of the project from the beginning of the development to its current state without reference to other projects. For example, it was not necessary to consider problems such as how many lines of code were new and reused, or whether a certain fault was generated in the new project or was inherited from the code reused.

The data provided was extracted from an extensive database of quality records that document the development of the system. Although the project was relatively small (approximately 1500 lines of code), the extensive database was a result of its high criticality and the corresponding customer and regulatory oversight that this project received.

The data provided was mainly development process data, as no field data was available at the time of this study. The data encompassed information on the development process, plans, reviews (requirements, design and code), time sheets and other technical documentation including software architecture and components of the software builds. In addition, they included records of modifications and their justifications and details. These modification reports cover documentation problems, quality problems (e.g. lack of traceability) as well as functional problems.

The analysis of the data provided had two main objectives:

- the discovery of the software development process as enacted and how it differed from the idealised process initially described
- the extraction of the number of faults found during the development, the phase of the process in

which they were found and the activity that revealed them

Project B was the third package of a safety-critical system. By the time of the study, and approximately four years after the beginning of this project, twenty-one versions of the software had been developed. In a total of 12 500 significant lines of code, approximately 2500 lines were new, while the remaining code was reused from the previous two packages. These posed problems that were not relevant for Project A: it was necessary to distinguish whether faults were relevant for this study or were originated in the previously developed code.

The type of data supplied for Project B consisted of an electronic database of modification requests. The format of data for this project was more tractable than for the previous. However, for commercial reasons, further information could not be given. Although modelling of this project was performed, it was difficult to carry out full analysis without the support of the development team. In this paper we mainly report on the work done for Project A. Predictions for Project B using Project A rates are discussed in Section 5. The remainder of this section reports on Project A.

3.1. Process discovery

Initial analysis of the data was performed for the elicitation of the development process. The software development plan of the project described a modified form of the waterfall model. Further analysis of the project documentation and meetings with the development team informed the refinement of the initial model of the development process. We used a visualisation technique to provide a clear outline of the actual process (see Figure 2). In this technique, different tasks and documents produced during the software development are categorised according to the different phases of the lifecycle. Using the dates of the documents produced and representing the information graphically provides a clear overview of the process.

Although the phases of the process overlap, the shape of the diagram suggested two iterations of the lifecycle. These two full iterations of the lifecycle were confirmed with the project team: there were two main software builds that followed all the phases of the development process and were delivered to the client. The second of these builds was a result of substantial requirements changes requested by the client. This entailed a large number of changes; approximately 70% of the code was modified. The model includes two repetitions of the waterfall model, as illustrated in Figure 3. This structure is the basis for the barrier model developed for this project.

3.2. Analysis of fault data

The project development documents were further analysed to extract the rates of fault creation and detection of the software. Manual analysis of these documents was performed in order to trace the source of the findings, to see where errors were created and how they propagated through the lifecycle until their detection. It was also necessary to understand the type of modification described, and whether it was an error or simply a quality problem. The manual analysis

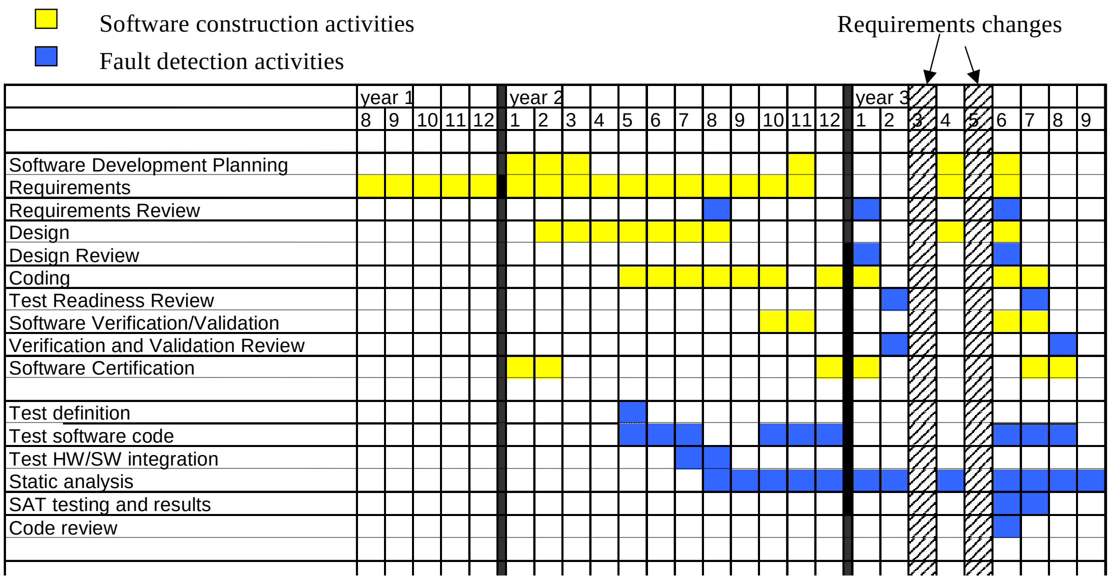

Figure 2: Visualisation of process activities

resulted in the classification of each of the findings according to:

1. the type of problem
2. the detection method
3. the type of modification involved

The *type of problem* corresponds to the activity being performed when the error was generated, and hence to the phase of the development process where the problem was originated. In this case, the type of problem is requirement, design or code.

The *detection method* classifies the finding according to how it was detected, i.e. each detection method corresponds to one of the filters of the barrier model, and it includes:

- reviews: requirements review, design review and code review
- testing
- static analysis

The final classification criterion of the findings is the *type of modification*. The findings are classified in three main categories according to whether they corresponded to error correction, implementation of requirement change or quality improvement. Naturally, error correction is the only type of modification that was considered to be a fault. Most of the modifications reported, however, resulted from improvement of documentation (e.g. traceability, clarification of some figures in the documents) and from changes of the requirements.

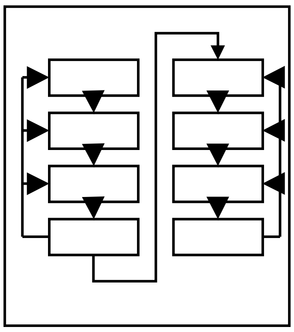
Figure 3: Two iterations of the lifecycle

The analysis of the documents was also used to identify the model barriers, i.e. the activities or methods that were used to detect faults of the different types. The basic barriers are identified in the software development plan, but knowledge of the process was refined by the information obtained from the project development documents.

4. Types of models

In this section we describe several models of the process that were developed. In these models, we varied the fault detection efficiencies and creation rates in order to study the impact of these variations in the number of faults to field predicted. In addition, two alternatives of the structure of the process were explored: one where the two iterations of the process are followed sequentially (as in Figure 3), and other where the two iterations are independent and parallel (see Figure 6). The alternatives considered are explained in the following subsections.

The variations make the models more or less fitted to the specific project data. Naturally, fitted models bring the predictions closer to the actual project data.

4.1. Detection rates

We explored a wide range of data sets on the development process in order to establish typical values for detection efficiency. The sets we have examined are:

- high quality data from software experiments developing critical software (PODS1 [10] and PODS2 [11]) and from a real industrial project, the French SPIN1 reactor protection system documented in considerable detail in [5]
- more generic data from the NASA Software Engineering Laboratory [6], data from SEI and DACS [7], classic data on inspections [4] and the Cleanroom approach [8] and general literature on defect densities [2] and those reviewed in [16]

We have excluded requirements errors from this analysis, since this was not available in most of the data sets. We would expect detection efficiencies for a given technique used in a barrier to be similar across projects, but there will be considerable variations due to the human nature of the task and problem difficulty. This is captured by the use of probability distributions for the detection rates. This also applies to the fault creation rates.

We note that there are doubts about the quality of reporting in the industrial projects such as:

- when the data collection was started
- how faults are classified
- the completeness of the data

The third point is illustrated by the software experiments where a greater fault rate was found.

The analyses of the projects above give generic detection efficiencies. In order to develop a more fitted model, detailed project data was used to calculate review efficiencies. A simple upper bound on the review efficiencies can be calculated from the ratio of the faults found in a phase to the total faults found in the phase and all later phases. This is an upper bound, as residual undetected faults are not included in the calculation. These upper bounds are adjusted by using the model predictions for the number of faults left. Table 1 shows the upper bounds of detection efficiencies for Project A, and it compares the values with industry values. These bounds were used as parameters of the probability distributions for the detection rates.

Activity	*Upper bound*	*Industry values*
Req review	0.50, 0.72, 0.85	0.7–0.85
Code and design review	0.32, 0.30, 0.29	0.7–0.9 (code), 0.8–0.9 (design), 0.4–0.6, 0.75–0.9
Test	0.48, 0.37	0.6–0.9
Static analysis	0.77, 0.60	0.76–0.86
Cust review	0.91	

Table 1: Detection efficiency upper bounds

It can be seen that all the project efficiencies are below the generic industry data for high integrity systems. This may reflect on the rigour of the process or it may reflect the difficulties in dealing with real time software. For example, the lower review efficiency of the static analysis is probably not due to the rigour but reflects the scope of the analysis in dealing with timing and other real time problems. The same might be true for the coding review: the consistently low numbers seem to be significant. As for requirements, it is clear from the analysis of the process that requirements elicitation and change are major issues for the process. In part this may be due to the concurrent engineering that is being performed with software, hardware and engine being developed in parallel, but it may also reflect a need for greater attention to the requirements.

4.2. Fault creation rates

The same data sources were assessed for fault creation rates (see Table 2).. In the data analysed there was a very wide variation in requirements fault creation rates. As a starting point, the requirements creation rate is assessed to be the same as that for code. There is more data on the proportion of coding and design faults.

Proportions	*SPIN*	*PODS1*	*PODS2*	*SEL*	*SEI*	*Average*
Design	0.30	0.42	0.22	0.18	0.35	0.29
Code	0.70	0.58	0.78	0.82	0.65	0.71

Table 2: Generic fault creation rates

The average of these ratios was used in the generic model. For the coding creation rate, the number of projects was taken into account and the knowledge of the spread in the data between projects. The data was weighted by project and distributed according to the judged uncertainty. In this way a histogram was produced for the judged code fault creation rate.

The approach to fitting the distribution is common to all the data analyses (see Figure 4). The distribution to be fitted, in most cases a gamma distribution, was parameterised in terms of the mean and mode. The mean was calculated from the project data and the mode estimated from the histogram of the data.

Data was provided for the faults found in a number of different projects of the same company. The projects

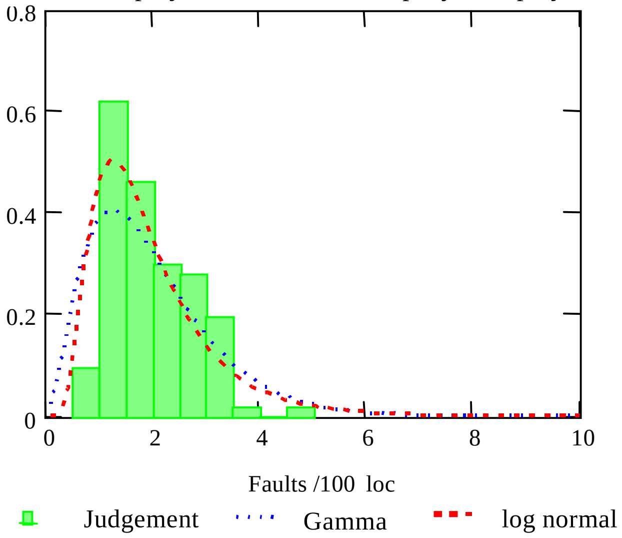

Figure 4: Fitting of judgement on fault rates

cover library modules, support tools and a number of different applications. They followed the same overall process but there were differences in the details of the process and the data collection mechanisms. The collection of projects was treated as examples of the process and a distribution derived accordingly. The graph in Figure 5 shows the fit to a gamma distribution of the projects (note the log scale). This distribution is fitted to the company, but not to the specific project being analysed.

Project specific fitted creation rates vary the

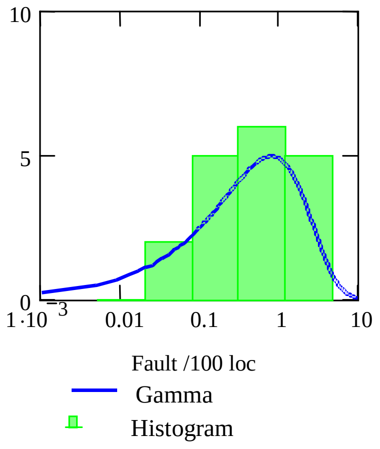

Figure 5: Fit of equipment fault creation

creation for the two iterations of the lifecycle in two different ways. In these two ways, the probability distribution for creation rates is the same in the two iterations, but they are scaled differently:

- More fitted models scale the distribution creation rates of each type of fault per cycle with the number of faults actually detected during the project. For example, 40% of the design faults were found in the first cycle and 60% in the second cycle.
- Alternatively, the creation rates for each cycle are multiplied by the lines of code written in the corresponding build: 1500 for the first build, 1050 for the second.

4.3. Process model

Recall that the development process of the software consisted of two iterations of the lifecycle, as discussed in Section 3.1. Using a sequential model, faults inserted in the first iteration can be detected in the second iteration. An alternative, parallel model, was also developed (see Figure 6). In this model, the phases are based on the software development process of the project, but instead of considering two iterations, only one of the iterations is modelled. Each iteration is treated independently and does not allow for faults inserted in the first iteration to be detected in the second iteration. Hence, the number of residual faults is the sum of the residual faults of each iteration of development.

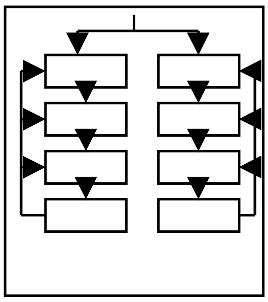
Figure 6: Parallel iterations

5. Results of modelling

For each of the models developed, we compare the actual data with the model estimations for the number of faults detected at each phase. We study further variations of the model where different proportions of faults between requirements, design and code faults are considered, we increase the detection efficiencies of reviews, or remove static analysis.

Table 3 illustrates the comparisons made. The values in the table are faults detected per phase per 100 lines of code. The model in this table uses the project detection efficiencies, the organisation creation rates and the parallel structure. In this approach each iteration is treated independently and does not allow for faults inserted in the first iteration being detected in the second iteration, i.e. we used the model in Figure 6. The phases in the table correspond to the development process: “coding” corresponds to the faults detected during code review, “coding-trr” to the faults found during test readiness review, and “customer review” is the final review performed by the customer of the company that supplied the data.

Phase	*Probability 0.05*	*Probability 0.95*	*Iteration 1*	*Iteration 2*	*Project B*
requirements	0.12	3.18	0.47	2.00	0.12
design	0.05	1.39	0.00	0.19	0.08
coding	0.05	1.40	0.47	0.29	0.04
coding-trr	0.02	0.59	0.33	0.00	0.08
testing	0.03	0.84	0.47	0.10	0.08
static analysis	0.022	0.63	0.67	0.38	0
customer review	0.000	0.01	0.00	0.10	0.08
total	0.91	7.87	2.41	3.06	0.48

Table 3: Model detection efficiencies

As can be seen from the table above, there is considerable variation between iterations and these are compared with the probability bands of the model. This shows that there is considerable variation in the application of the same process. Most of the data is between the 5% and 95% limits with the exception of the customer review (which is based on little data). Given that fractional faults cannot be observed, the observation of any fault during customer review would give a minimum figure of 0.1. The static analysis and design figures are one observation outside the 5–95% bands. The lack of design faults in Project A may be due to classification features of the process as well as the approach to design itself.

Table 3 also compares the iterations of Project A and Project B. Note that Project B has different phases and no static analysis was undertaken. The Project B data resides just within the 5% band making it an unusual project, according to the model.

In addition, different models are compared according to the estimation of faults to field per 1000 lines of code (Table 4). Models fitted to project data have higher predictions for residual faults. In addition, parallel models also predict more faults. This is a result of the fact that in these models faults created in the first iteration are not found in the second iteration. However, the predictions are consistent within a factor of 2.8, which might be sufficiently accurate bearing in mind the propagation of uncertainty in the various parameters of the model. On this basis and using the probability bands of the models developed, there is a high chance that Project A will have between 2-7 faults detected in later phases.

The modelling shows that the total faults to field is predicted within a factor of ~2 using generic data. The fit from one project predicts the second project well: although there were doubts about the process modelling and accuracy of Project B, the model based on Project A predicted that there is a 50% chance that the value of residual faults in Project B is between 2 and 7, with a mid value of ~4 residual faults. In fact, 5 faults were found in the field.

Creation rates (R)	*Detection efficiencies* (D)	*Process Model*	*Mean of faults to field per 100kloc*
Generic	Generic	parallel	1.21
Equipment creation	Generic	parallel	1.27
Equipment creation	Project A efficiencies	parallel	2.69
lines of code	Generic	sequential	0.95
Proportion of faults	Generic	sequential	1.34
lines of code	Project A efficiencies	sequential	1.85
Proportion of faults	Project A efficiencies	sequential	2.10

Table 4: Faults to field per 100kloc

A sensitivity study was undertaken to see how the review efficiencies affect the total faults detected. The results are surprisingly insensitive to the efficacy of the review and detection process. Figure 7 shows the effect of multiplying all the review and detection efficiencies in the model by a scaling factor (the relative efficiency). While the total detected is insensitive to the review efficiency the total faults in the product being deployed is not, as seen in the graph. The flatness of the graph may explain why the fault detection from projects reported in the literature is so consistent.

6. Discussion

We looked at two projects from the same organisation. These projects were dissimilar in both the type of project and the type of data that was available.

We successfully reconstructed the development process as enacted from the extensive documentation collected. This reconstruction was supported by a process visualisation technique based on document changes, fault data and project activities. The visualisation technique was used for both projects, and it provided an accessible overview of the actual process and proved very useful for the general understanding of the process.

The interpretation of the process data was assessed and verified in collaboration with the project development team. Based on this information, we produced a range of process models that were calibrated with generic industrial data, partial data from a collection of projects from the organisation and detailed project specific data.

A detailed understanding of the process is necessary to construct accurate models. The software development documentation is seen as an abstract description of the activities that take place. There is always a difference between this abstract description and the process as enacted. However, it is necessary to understand the disparity from the abstract description to the actual activities in order to model the projects. Some of the discrepancies between the abstract idealised description of the software development process and its implementation are easy to grasp from a superficial analysis. In general, however, this is not the case. Everything needs much explanation and interpretation, and interaction with the project team is needed to comprehend the full process and its activities. In a way we are undertaking an ethnographic study of the development process.

However, even if an accurate process description already exists, the credibility and trust placed in a model depends on this fieldwork. To trust the model there is a need to understand the origin and limitations of the data.

There is always some uncertainty in these models. The process discovery and analysis shows that there is a considerable variability between applications of the same process. There are wide variations between projects and between the iterations of the same project. These variations make the predictions less accurate and make project monitoring and cost prediction more problematic. Some of this variation may be due to inherent randomness in human activity and the perturbations that impact any project. However, in the cases analysed, it is judged that improving the handling of requirement change and improving the consistency of data reporting and classification would reduce this variability.

The total of faults detected in a product line is predicted well from generic industrial data i.e. better than 10%. Models based on generic fault creation rates and detection efficiencies predict faults to field to within a factor of 2–3 when compared with models based on detailed calibrations. The generic models are less accurate at detailed predictions of faults found in the different lifecycle phases: for this fitting the model of the organisation's own detection efficiencies is required.

A model has been developed that adjusts itself to the faults detected in the product line but this was considered to be over fitting given the uncertainties in the data. However in other applications this may be needed where there is not good agreement between the generic data and the company aggregated data.

This approach relies on the maturity of the organisation to collect and analyse development data. While the method might be feasible retrospectively, it will be most readily used in organisations with high level of process maturity (e.g. 3 or above on the CMM scale). If an organisation has no data on the results of its process but does know the structure, the modelling using our generic data would be possible. Generic data will result in a wide spread in results. Nevertheless, it might have an important role in sensitising people to what might be credibly claimed for the system and in rejecting the more outlandish claims.

Similarly for COTS one might use generic data with a generic process to provide some indication of the quality of the product.

The modelling may also provide a more detailed appraisal of the impact of process improvement and in its use of uncertainties, the emphasis on modelling real rather than assumed processes, and the approach to calibration provides a credible approach. The work has led to several recommendations to the data provider for process improvements: it provides an empirically and technically sound support for these recommendations. It was observed that there was a predominance of requirements related issues and consistently low review efficiencies when compared with generic industrial data.

It is clear from the analysis of the process that requirements elicitation and change are major issues for the process. In part this may be due to the concurrent engineering that is being performed with software, hardware and engineering system being developed in parallel but it may also reflect a need for greater attention to the requirements within the process. The faults to field are predicted to be dominated by requirements faults because the performance process barriers are an order of magnitude less effective than those for design and code faults.

The profile of faults found by phase is needed for process improvement measures (not just the total of faults found) as the totals of detected faults is insensitive to improvements in detection efficiencies.

Finally we should consider the use of this model results in the reliability bound model [1]. The process model will produce a distribution function for N and this can be used in a number of ways. It can be used directly to provide a confidence in the bound or a judgement can be made that N is less than a certain figure to a required confidence (e.g. 99% confidence N<x). In making these judgements we need to assess whether the process model itself is underestimating N. This could come from two main sources: overestimate of efficiencies, and underestimate of N used to calibrate the model.

An overestimate of efficiencies might come from ignorance of the actual number of faults left in the product rather than the measured total. In fact the model predictions can be used to adjust for this but for the early phases the impact is small, as the number of faults to field is a relatively small proportion of the total faults found. If the model is being used, the results should be subject to a sensitivity study as we show in Figure 7. If we are only seeking a prediction within a factor of 2-3 then the model is relatively insensitive to review efficiencies (e.g. can vary by ~2). Analytica also provides statistical measures of importance that indicate the major sources of uncertainty in the predictions.

The other concern is some systematic and gross underestimate of N – the iceberg effect – in part due to experience with datasets such as Adams [18] that indicate a substantial number of very low rate faults in a large operating system. Given the safety critical nature of the products, their small size and very extensive field experience we can be confident that there are not a large class of faults undetected and unpredicted by the model. If there were, they would have to had no impact on reliability over many years, over diverse products and not have been detected by the verification techniques deployed.

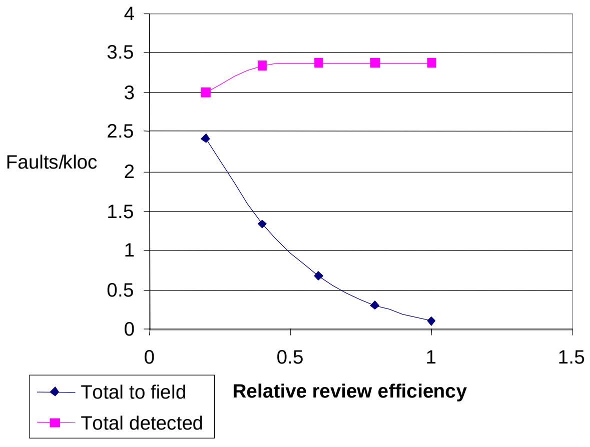

Figure 7: Variation of faults observed and to the field with detection efficiency

There may also be some subtle effects that lead to loss of conservatism. For example, there could be a correlation between a high number of faults inserted and low review efficiencies due to novelty in the problem being addressed or due to project stress. This would increase the tail of the N distribution.

There are also assumptions that the performance on past projects predicts the future. There could be substantial changes to the culture of the company (e.g. changes to safety culture due to project stress, take-overs) and technology and engineering changes that might invalidate the predictions.

7. Conclusion

In this study:

- The actual process followed by the company was successfully reconstructed from the extensive documentation collected.
- A process visualisation technique based on documentation changes, fault data and project activities was developed, which provided an accessible overview of the actual process.
- The interpretation of the process data was assessed and verified in collaboration with the project development team.
- A range of process models was produced that were calibrated with generic industrial data, partial data and detailed project specific data.

We concluded that a detailed understanding of the process is necessary to construct accurate models and that the credibility and trust placed in a model depends on this fieldwork. Models based on generic fault creation rates and detection efficiencies predict faults to field within a factor of 2-3 when compared with models based on detailed calibrations. The generic models are less accurate at detailed predictions of faults found in the different lifecycle phases: for this fitted models to the organisation's own detection efficiencies is required. The work led to several recommendations to the data provider for process improvements.

Based on the experience in this project it was judged that the approach could support a judgement of the likelihood of a system being deployed with faults and hence support the overall safety justification. For larger systems the results could form the basis for estimating "N" in the long-term reliability model and for more mature systems the probability of there being a fault in the delivered product.

Acknowledgments

This work was funded under the HSE Generic Nuclear Safety Research Programme under contract 40029264 and is published with the permission of the Industry Management Committee (IMC). The views expressed in this report are those of the authors and do not necessarily represent the views of the members of the IMC or the Health and Safety Commission/Executive. The IMC does not accept liability for any damage or loss incurred as a result of the information contained in this paper.

References

[1] PG Bishop and RE Bloomfield. "A Conservative Theory for Long Term Reliability Growth Prediction". In *IEEE Trans. Reliability*, vol 45, n 4, pp 550-560, Dec 1996.

[2] F. Akiyama. "An Example of Software System Debugging", *Proc. IFIP Congress'71*,1971. As cited in [3].

[3] M. Shooman. "Software Engineering", International Student Edition, 1983.

[4] M E Fagan, "Design and code inspections to reduce errors in program development", in IBM Systems Journal, Vol 15 No 3, pages 219-248, 1976.

[5] A Jourdil, R Galera, "Methode de developpement d'un logiciel de surete", cahiers technique 117, Merlin-Gerin, France, 1982.

[6] "An overview of the software engineering laboratory", SEL-94-005 NASA Goddard Space Fligth Center, and associated database, 1994.

[7] "A business case for software process improvement", Data Analysis Center for Software, F30602-92-C-0158, 1996.

[8] M Dyer, "The cleanroom approach to quality software development", Wiley 1992, ISBN 0-471-54823-5.

[9] *Analytica User Guide*, http://www.lumina.com.

[10] PG Bishop et al. "PODS a Project in Diverse Software", *IEEE Trans. Software Engineering*, Vol. SE-12, No. 9, pp. 929-940, 1986.

[11] PG Bishop et al. "STEM: a Project of Software Test and Evaluation Methods". In *Safety and Reliability Society Symposium 1987 (SARSS 87)*, Manchester, Elsevier Applied Science, ISBN 1-85166-167-0.

[12] P. Hall et al, "Integrity Prediction during Software Development", *IFAC Symposium on Safety of Computer Control Systems*, Zurich, 1992.

[13] K. Yasuda, "Software Quality Assurance Activities in Japan", *Japanese Perspectives in Software Engineering*, 187-205, Addison-Wesley, 1989.

[14] M.A. Cusumano, *Japan's Software Factories*, Oxford University Press, 1991.

[15] Fenton NE and Neil M, "Bayesian belief nets: a causal model for predicting defect rates and resource requirements", Software Testing and Quality Engineering 2(1), 48-53, 2000.

[16] Fenton NE and Ohlsson N, "Quantitative Analysis of Faults and Failures in a Complex Software System", *IEEE Transactions on Software Engineering*, 26(8), August 2000.

[17] Hugin A/S: http://www.hugin.com.

[18] E.N. Adams, "Optimizing preventive maintenance of software products," IBM Journal of Research and Development, vol. 28, no. 1, pp.2-14, 1984.