ADR Bulletin
ADR Bulletin

Volume 7 | Number 5 Article 1

12-1-2004

Using the Forensic Systems Analysis methodology for expert opinion for online dispute resolution

Stephen Castell

ADR bulletin
The monthly newsletter on dispute resolution

Information contained in this newsletter is current as at December 2004

Volume 7 Number 5

Benefits of ODR in complex software contract disputes

Using the Forensic Systems Analysis methodology for expert opinion for online dispute resolution

Stephen Castell

contents

73
Using the Forensic Systems Analysis methodology for expert opinion for online dispute resolution

79
The transformative effect of mediation in the public arena

85
Mediation in Sweden

92
Diary and happenings

Editor's note: This is an edited version of a presentation given at the Third Annual Forum on Online Dispute Resolution, University of Melbourne, Australia, 5-6 July 2004, in collaboration with the United Nations Economic and Social Commission for Asia and the Pacific.

Introduction

Those who have been involved in litigious disputes over failed computer software projects would agree that, whatever their size in terms of the financial amounts at stake (and whatever the facts and circumstances of the contract between the parties, and the conduct of the software development) software construction and implementation cases present interwoven technical and legal issues which can be both arcane and complex – and therefore prove costly and time-consuming to unravel.

CASTELL Consulting, a professional IT consultancy founded in 1978, has been involved in a wide variety of complex computer software litigation. We have in particular been instructed as expert witnesses (for example in the UK, Europe, the Arabian Gulf, Australasia and the US) in legal actions concerning major software development contracts which have been terminated, with the software rejected amidst allegations of incomplete or inadequate delivery, software errors, shifting user specifications, poor project management, delays and cost over-runs. This work has been conducted on behalf of claimants and defendants, software customers and suppliers, in the High Court (or equivalent), and in arbitration, mediation and other forms of ADR.

Forensic systems analysis

Over years of examining a wide variety of software development disputes as appointed experts, CASTELL has developed techniques for assessing and reading the 'technical entrails' of failed, stalled, delayed or troublesome software development projects. It should be noted that such projects can often be a contractually uncertain mixture of 'customised' software packages *and* 'bespoke' construction. Many articles and papers have been written as a result of such experiences.

The CASTELL inquisitorial method, 'Forensic Systems Analysis', which focuses on testing of the software in dispute, is capable of being developed into a protocol for presenting the objective findings of the IT expert in the context of ODR. I believe that, through online demonstration of the results of Forensic Systems Analysis, the IT expert can provide illuminating and dramatic insights into the state of the software in dispute, and contribute to saving time and costs in, and facilitating rapid settlement of, such technically complex disputes.

I here outline some of the Forensic Systems Analysis components relevant to expert investigation in typical civil litigation over failed software development contracts, the application and findings of which could, I believe, be presented online for ODR purposes.

Software 'quality'

The most common, and arguably most important, issue on which the computer expert is inevitably asked to give a view in software development or implementation cases is: what was the quality of the delivered software and was it fit for the purpose? This raises the question: just what is meant by 'software quality'? The ready answer from the experienced IT expert is that 'quality' can only mean 'fitness for purpose', in the sense of 'does the delivered software meet its stated requirements?'.

The quality of software is essentially dependent on the specification: what the software is expected to do and how the software is expected to perform in its defined environments. In other words, the yardstick for measuring and judging whether software is of appropriate quality and fit for its intended purpose is the Statement of Requirements defining what is required or expected of it.

Testing software against its Statement of Requirements is the practical and universally accepted method of judging the quality of the software and whether or not the software is fit for its intended purpose. This critical focus on testing the software in dispute against its Statement of Requirements has a different emphasis for different specific cases.

- In a case concerning an in-store EFTPOS (electronic funds transfer at point of sale) system for a major national retailer, the crucial issue was whether or not the software supplier was likely to have fixed many outstanding errors and have had the system ready to roll-out in time for the pre-Christmas sales rush. What was the objective technical evidence of the software house's 'bug find and fix' performance? Were the bugs escalating, or was the software emerging into a stable, performing system? Were, rather, the constant changes in customer specification – as alleged by the supplier – to blame for the delays and inability of the software to pass a critical acceptance test?
- A case concerning a large university consortium similarly focused on the apparent inability of the software developer to present a main module of the software system in a state capable of passing formal Repeat Acceptance Tests, with a number of faults appearing at each attempt at 'final' testing (even though three earlier main modules had been successfully developed and accepted). How serious were these faults, and were earlier faults that were thought to have been fixed constantly re-appearing? Was the customer justified in terminating the contract on the grounds of a 'reasonable opinion' that the software supplier would not resolve all the alleged faults in a 'timely and satisfactory manner'? Was the supplier's counter-claim for a large financial amount for 'software extras' valid, and could that explain the inability of the software to converge onto an 'acceptable' system?

Testing incident reports

The computer expert witness – often coming onto the scene of the failed project many months, sometimes years, after it has collapsed – is usually presented with large volumes of project documentation, an important element of which is the set of software testing records. Typically, these are in the form of Testing Incident Reports (TIRs), which can run into many hundreds, if not thousands, for large-scale bespoke software development contracts.

To simplify, the dispute may then come down to this. The customer alleges that the TIRs represented errors in the software which were critical, serious, incapable of being remedied, too numerous in number, or in some other way a material breach of the contract by the software supplier, entitling the customer to reject the software and terminate. The software vendor/developer, on the other hand, retorts that the TIRs did not constitute 'showstopper' faults, they were readily rectifiable, and anyway principally

*a*rose from the many and continuous changes in specification made by the customer – the customer was therefore not entitled to terminate and had repudiated the contract in so doing.

The Forensic Systems Analysis methodology: EFLAT, EAT and FORBAT

The expert hired by either of the parties in the dispute may address these issues using a number of Forensic Systems Analysis components, the most important of which are likely to be EFLAT, EAT and FORBAT.

EFLAT – Expert's Fault Log Analysis Task (material defect)

During software development defects are routinely encountered, and routinely fixed. There is generally nothing alarming about their occurrence. For the purposes of rejection of software and termination of a software development contract, any alleged defect must therefore be assessed using a strict test as to whether or not it is truly a material defect; that is, as to whether or not the contract cannot be considered to have been performed while the defect persists.

EFLAT, developed over the years through debate with many firms of instructing solicitors and counsel, uses what I believe is a sound protocol for testing whether or not any given software fault, in terms of its relevance to a breach and termination of a contract, is a *material defect*. This protocol is essentially that, to be a *material defect*, an alleged software fault must be:

(1) of large consequential (business) effect
(2) impossible, or take (or have taken) a long time, to fix
(3) incapable of any practical workaround.

The customer is entitled to define what a 'large' consequential business effect is; and the supplier may put forward an appropriate sizing for a 'long' time to fix – each from the standpoint of his or her own business/technical knowledge and experience, and in the context of the particular contract/project. Both views ought to be evidentially supportable. Both views – and, also, whether or not there *is* indeed a practical workaround – would be the subject of expert scrutiny and opinion.

EFLAT constitutes a careful re-running of the appropriate acceptance tests, under expert observation, with each TIR (or fault log) raised during the test rigorously assessed according to the *material defect* rule. The outcome is a Scott Schedule (of software defects) with each fault particularised, stating why each was considered a breach of contract (by reference to specific contractually defined requirements), what the consequential effect was estimated to be, what the technical time to fix was (or was estimated to be), and whether or not there was any practical workaround available, giving the expert's independent view on all these individual elements, with an opinion as to whether or not the specific fault in total was a *material defect*.

EFLAT is undertaken with a 'prototyping' orientation, assessing first only a limited proportion of the TIRs, so that experience may be gained as to how difficult the full task is likely to be, how long all the TIRs will take to assess, and whether there may be technical obstacles in, say, reproducing the exact conditions of the Acceptance Test corresponding to those which were obtained when the alleged faults were originally found. Not the least of these obstacles can be the basic evidential uncertainty over whether or not the version of the applications software and/or the database configuration and/or the hardware and systems software environment available to the expert (months or years later) precisely correspond to the system being litigated over.

Obstacles apart, early prototyping of EFLAT enables estimates of how much time (and therefore cost) is likely to be needed to complete the Scott Schedule (of defects) in its entirety, enabling clients and instructing solicitors to take a considered view as to the full extent of expert investigation to be commissioned. Once again, such estimates, properly presented and shared through controlled web 'publication', should be of great benefit in the conduct and costing of ODR.

EAT – 'Extras' Analysis Task

In software development projects of any significant size there may be many 'contract variations' caused by the inevitable shift in the customer's or users' perceptions of what they require as they see the software actually being built and tested. This 'specification drift' or 'constant changes to requirements' is a well-known phenomenon in engineering construction disciplines and presents a challenge to well-ordered project management to ensure that such variations are properly documented and controlled, and that both parties understand and agree on the impact on project scope, timetable and costs of implementing requested software changes.

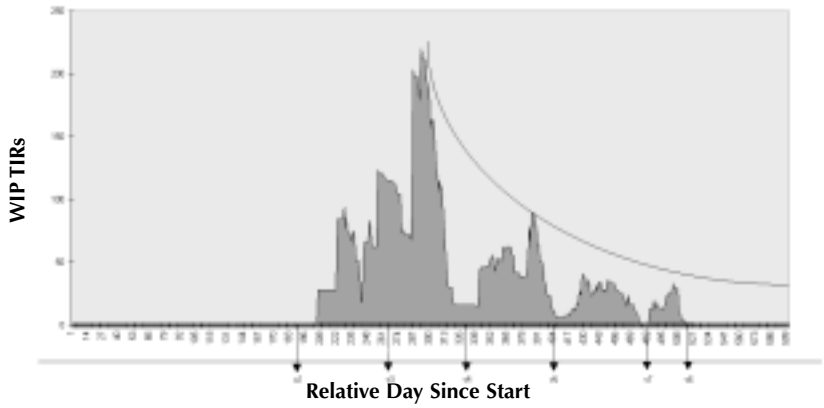
Typically, for the project which collapses and ends in dispute or litigation, the computer expert witness is asked to give an opinion on whether or not there were changes from the originally contracted software, and, if so, what was the quality of the additional software built; and to what financial remuneration (for example, on a *quantum meruit* basis) the supplier may be entitled for providing such software 'extras'.

EAT comprises a methodical analysis of the:

(1) contractual documentation (in

EFLAT constitutes a careful re-running of the appropriate acceptance tests, under expert observation, with each TIR (or fault log) raised during the test rigorously assessed according to the material defect rule.

Graph 1: Work in Progress (Cumulative reported TIRs - Cumulative resolved TIRs)



particular the Statement of Requirements, including any amendments or re-issues thereof during the project)

(2) work records of the software engineers who did the construction of the 'extras'

(3) items of software design, source code, functionality, execution and performance which it is alleged have been produced as a result of the extra work

(4) financial amount claimed, and if it is consistent with (1)-(3) and passes a 'sanity cross-check' such as that provided by assessing the £ or $ per delivered line of source code standard software metric.

FORBAT – FORensic Bug Analysis Task

Always recognising that during software development defects are routinely encountered, and routinely fixed, and there is generally nothing alarming about their occurrence, the overall numbers of such 'bugs' (as logged by the TIRs), and the pattern of their build-up and resolution, are nevertheless important indicators of the progress of software construction and testing.

Such indicators are unfortunately often misread by both the software customer and the software developer: in particular, the dramatic increase in TIRs and apparent 'never-ending increase in bugs' during systems testing can be badly misinterpreted. The point to be made is that systems testing (usually the responsibility of the software developer) *should* find bugs and fix them – it is not being done properly if there is not a large build-up in recorded TIRs. This contrasts with acceptance testing (usually the responsibility of the customer) where 'zero', or only a small number of non-serious bugs, is not an unreasonable expectation, particularly as acceptance testing should be undertaken with the appropriate attitude – for acceptance, not rejection of the software proffered for testing.

FORBAT uses a number of standard quantitative analysis techniques to give an objective graphical presentation of the true 'bug find and fix' performance of the software house, readily understandable to non-technical clients, lawyers or judges. The insights which spring from these presentations are usually vivid (and incidentally can come as something of a surprise to the parties themselves). These are best explained by two examples, both taken from a real software project.

(1) Illustrations of typical FORBAT 'bug find and fix performance' graphs and conclusions reached

Graph 1 represents the cumulative number of TIRs outstanding at any point in time, and is formed by subtracting the cumulative total number of resolved TIRs from the cumulative total number of reported TIRs on a day-by-day basis. A trendline has been added to the graph to indicate the overall tendency of the underlying data, but otherwise the graph is an objective portrayal of basic project statistics.

From this graph it can be seen that the number of outstanding TIRs peaked at more than 200 before the release of version 4, but that these outstanding TIRs had been addressed and resolved, with the exception of fewer than 20, by the time version 4 was released. This is

in keeping with what would be reasonably expected if the software house were efficiently handling the 'errors' reported, eliminating as many bugs as possible before releasing a new version of software. The steep drop-off in the number of outstanding TIRs from day 300 to day 320 is an indication of a significant amount of work performed to address and resolve outstanding TIRs prior to release of a new version of the software. The inferences to be drawn from this graph and its underlying data are, for each major release of the software (ie versions 4, 5, 7 and 8) the:

(1) total cumulative number of outstanding TIRs decreased steadily from more than 200, to less than 100, to less than 50, to around 45
(2) total number of outstanding TIRs reduced sharply before each software release (notice the steep drops in the curve before each major release indicating a 'clean-up' process prior to each release)
(3) total number of outstanding TIRs for each major release decreased from less than 20, to less than 10, to less than 5, to zero.

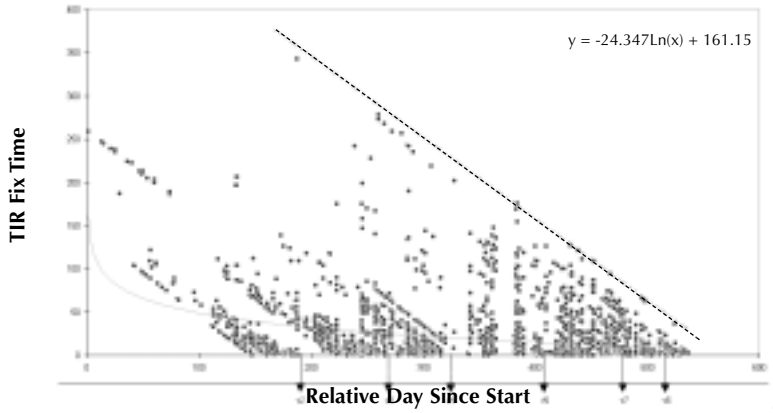
These results are consistent with what is reasonably to be expected in a well-run software development project environment. It is usual in such an environment to see a build-up of reported 'errors' or 'bugs' during systems testing as a major release approaches. It is also usual to see a steep drop-off in the number of these bugs outstanding just before each major release. In summary, Graph 1 illustrates that the process to 'find and fix bugs' was one in which the software house became increasingly proficient; and indicates that the software itself was becoming more and more stable.

Graph 2 depicts all TIRs in scatter format. Each individual TIR is plotted on Graph 2 at the point in time that it was reported versus the elapsed time in days that it took to resolve it. A logarithmic trendline has been calculated based upon the distribution of the base data used, with the resulting equation indicated in the upper right hand portion of Graph 2 and the trendline itself superimposed on the data. This trendline indicates an overall tendency of the data points plotted. Additionally, a 'worst case trendline' has been drawn on Graph 2 (the dotted straight line running from the upper left hand side of the diagram to the lower right hand side). This 'worst case trendline' uses the outlying data points to provide an indication of where the trend is going from the 'worst case' perspective.

Looking at the 'worst case trendline', its steep slope may be noted. It appears that it would intercept the X-axis near project day 550, or one and a half months after the project was terminated. This extrapolated interception, which would have constituted a modest further elapsed time in the context of a software project which had already been running for over 18 months, corresponds to the resolution of all the longest outstanding TIRs within the data population.

I believe that it can be easily appreciated that the vivid and self-evident insights that spring from such an objective graphical portrayal of factual and quantitative software project analyses could with great advantage be readily promulgated and understood within the conduct of an ODR.

Graph 2: FIX TIME - Scatter with Logarithmic Trendline



POLIT – Performance, Operability and Locking InvesTigation

Even when the functionality provided by the software is acceptable, litigious allegations can be made regarding its run-time performance, perhaps expressed as 'response times did not meet those stipulated in the contract'. To form an expert view, a system sizing and performance model may be constructed.

POLIT uses such a model to establish, inter alia, a resource usage profile for key transactions used in testing scripts in order to:

(1) give a measure of the likely performance of a full system workload
(2) identify ways in which performance could be optimised or improved.

Critical resources investigated are disk accesses and CPU (central processing unit) usage.

Key variables examined include:

- resources used by log-in processes
- paging-in of the application software the first time it is used
- impact of different cache sizes
- impact of different batch sizes
- overhead of menu navigation when repeating the same process
- effect of spreading the database
- effect of shadowing (using two identical copies of the database).

POLIT can become deeply technical and abstruse, and it is a challenge to the expert to explain the methodology, the model and the conclusions it delivers in a manner understandable, for example, to lawyers or to the court. However, when that challenge has been met, the findings arising can also be readily shared with all those involved in an ODR, for example by way of interactive web page presentation. Such presentation may allow those involved to carry out their own 'what if' exercises using the expert system sizing and performance model provided.

Some further Forensic Systems Analysis tasks for ODR adaptation are as follows.

FUDDER – FUndamental Database and DEsign Review

Often there are allegations of 'fundamental software design flaws': this task assesses a range of accepted software engineering design parameters, and their documentation, to give an opinion on the completeness, correctness and robustness of the software, database and communications architectures in terms of their suitability for building, testing and implementing a system meeting all required contractual obligations.

PROMADET – PROject MAnagement and Delay Examination Task

'Slippage' in project schedules is extremely common in large software development projects. The expert is often asked to answer the question: "Who was to blame for the delays?" This task examines GANTT charts and the like, drawn and re-drawn throughout the project, together with associated project management documentation (for example, minutes of project board/committee meetings) and work records, as well as the 'fossil record' of the evolution of the construction of the software source code itself.

It should be noted that it can be difficult to 'win a case only on an allegation of delay': the meaning of 'delay', and the evidence and reasons for its occurrence, are usually not easy to determine, or present clearly, for any software project which has gone on for several years. Very often the best that can be said is only that 'everyone was equally responsible' – not a particularly helpful opinion for an expert to give!

EVOCRAT – EVOlution of Changes to Requirements Analysis Task

As I have said, constant contract variation caused by 'specification drift' is a common experience in most sizeable software development projects. The expert is asked to give his or her view as to what such changes amount to in terms of, 'What was the final Statement of Requirements – what was the detailed contractual specification at the end of this process of change?'. EVOCRAT assesses all documents purporting to state new or changed requirements, and consolidates them into a unified specification.

Conclusions

Given that the key to arriving at a useful and helpful expert opinion in complex software contract cases is testing of the software in dispute, I believe that the techniques and findings of the CASTELL Forensic Systems Analysis methodology can readily be applied to resolving disputes in the context of ODR. If the parties involved in ODR can agree to apply these rigorous techniques they can create a growing corpus of shared objective results, using a process along the following lines:

1. Where technically possible, make the software in dispute accessible via the web.
2. Publish/disclose appropriate user guides.
3. Experts on both sides design agreed test plans, scripts, expected results – publish/disclose them.
4. Once the EFLAT and EAT tasks based on such agreed plans have been undertaken, publish/disclose the relevant findings in Scott Schedules in standard/agreed form.
5. Further tests can then be done, and results and expert inferences/ opinions added to the Scott Schedules – eg via hotlinks.

I believe that the benefits of such an approach in complex software contract disputes, presenting the objective findings of the IT expert in the context of ODR, could be substantial. Through online presentation, demonstration and accretion of the results of Forensic Systems Analysis, the IT expert can provide illuminating and dramatic insights into the state of the software in dispute, and contribute mightily to saving time and costs in, and facilitating rapid settlement of, technically complex disputes. ●

Dr Stephen Castell, Chartered IT Professional, Chairman of CASTELL Consulting (founded in 1978), is also an experienced mediator, an ICC Arbitrator and Expert Determiner and a member of the Expert Witness Institute. He can be contacted at cstll01@attglobal.net.

Further details regarding the Forensic Systems Analysis will be available on a future website www.ForensicSystemsAnalysis.com.