Troubleshooting for Project Managers

Resolving technical issues that arise during application project delivery

An Advance7 White Paper

March 2015

CONTENTS

EXECUTIVE SUMMARY

There is a constant pressure on programme and project managers to reduce the time-to-market for a new application. A problem identified during final testing can totally blow the possibility of hitting target dates of an already tight timetable.

Whilst functional errors are relatively easy to identify and fix, intermittent performance and other recurring problems are often much more difficult. What's more, because the cause of such problems is frequently obscure it's even difficult to identify the correct people to tackle the problem.

In this paper we outline some of the warning signs that signal a difficult problem in a project and suggest ways in which a project manager can help the technical teams to resolve the problem. These suggestions are based on experience gained in resolving project problems in real-life situations.

LICENCE

PURPOSE

Projects hit problems; it's a fact of life. The purpose of this paper is to help a project or programme manager to direct a team of technical people to resolve certain types of problems that are detected during testing or early deployment, these being:

- Recurring performance problems
- Intermittent errors
- Intermittent incorrect output

These are problems that, if transitioned into operations, can cause significant IT workload for the service desk and support teams. This is not to say that other problems, such as incorrect or missing functionality, are not also important. These will also add to the support workload. However, although functionality issues might be difficult to resolve, they are typically easy to recreate and debug, whereas the problems listed above usually can't be recreated on demand and so are difficult to diagnose.

The points made in this paper are based around application issues but most of them are equally valid for projects to introduce new infrastructure components or services.

Whilst there is a danger that to some this paper might be stating the blindingly obvious ("… teaching grandmother to suck eggs") we feel that it draws together common issues, and presents straightforward solutions that we know to work.

OBJECTIVE

The objective of this paper is to provide the project manager with enough information to be able to judge the effectiveness of a problem investigation, provide pragmatic solutions to overcome obstacles and assess recommendations made to resolve a problem. Ultimately we hope that project managers can use the information here to help the project team to fix problems quickly and effectively.

SCOPE

Difficult problems may arise at any point during the transition of an application into an operational state, and so this paper deals with issues that arise in:

- UAT
- Preproduction load testing
- Integration testing
- Pilot
- Deployment

Some problems may not be evident until the application is placed under load, some will only appear when the new application is integrated with other production systems, whilst others will only be apparent when certain real-life combinations of transactions occur. In this paper we need to consider all of these possibilities.

THE ISSUE

When faced with a difficult recurring problem, the typical responses of the project team, and the disadvantages of these responses, are:

Response	Disadvantage
Make a series of changes that may or may not solve the problem.	Slow and risky. There is no firm way to re-schedule a launch with this approach as it's difficult to predict when the problem will be solved.
Upgrade the specification of the production infrastructure in the hope that this will either resolve or mask the problem.	Slow and expensive.
Force the project into production in the belief that the problem won't occur in the live environment.	Very risky.
Force the project into production and let the operations staff deal with the fallout.	Bad for the reputation of the IT department and the project manager.

The ability to force a faulty application into an operational state will depend on:

- The urgency of the need for the application
- The political strength of the project sponsor
- The political strength of the operations director

In our experience the majority of faulty applications get forced into production. Sometimes a compromise is agreed whereby the application transitions into production but remains under warranty, meaning that the project team have to continue to support an operational system. Obviously this is likely to cause a cost overrun, and is not good for the reputation of the project team.

In a phased deployment a further problem arises. As the deployment progresses there is a gradual exodus of technical people from the project. Often these people are contractors and so the organisation loses valuable knowledge that could otherwise be used to resolve problems.

Considering these factors, there is a compelling argument to resolve recurring problems as quickly as possible.

Here's a great story for an experienced project manager.

"In one particular instance I had non-specific NFR's (written by consultants who kept decreasing the numbers when [the application] failed testing), an app which continually deadlocked under load, a vendor who was questioning the load tools/scripts and the infrastructure ([the cause] was neither of these), a publicly announced launch date to the market....and so on. [We] delayed the launch once but when it went live we had five P1's on launch day! Took a year to sort out the mess! Trying to get a vendor to sign up to NFR's/KPI's after the event is not easy!!"

EARLY STAGES

In the early stages of an investigation we need to check the basics:

- Is the test or pilot user simply using the system incorrectly or misinterpreting the output?
- Is the load test script issuing invalid commands or incorrectly interpreting a response as an error condition?
- Is the system (application and supporting infrastructure) configured correctly?
- Are we running the correct releases of software on the infrastructure components and at the interface of any systems integrated with the new application?
- Do we have an obvious overload condition?

It's only reasonable to allow technical support teams to check these issues. If the problem persists there may come a point where the method of investigating the approach needs to change. The trick is to spot that point as soon as possible.

THE WARNING SIGNS

How can we tell that the technical teams or a supplier are struggling to solve a problem? There are some phrases and activities that provide an early warning:

- Repeatedly hearing the phrase, "We are just going to try one more thing"
- The problem is passing from team to team
- The project team says that it's a problem with an infrastructure component although they are unable to produce evidence of the root cause
- The developers say the issue is that the database is underpowered or poorly configured even though there is no evidence of an individual query or stored procedure issue

- There are repeated promises that it will be fixed in the next code release
- The investigation is sucking in more and more people but this is just giving rise to more theories as to the cause

It's at this point we should recognise that the investigation is floundering, and we need to determine why.

INHIBITORS AND EFFECTIVE TECHNIQUES

In this section we look at reasons why a project team may struggle to solve a problem and the techniques we can employ to deal with each issue.

Vague Symptom Description

How many times have you heard the complaint, "It's too slow"? To investigate a performance problem we need to know the answer to two fundamental questions:

- Which particular operation is slow?
- Slow compared to what?

Even though we may have a whole range of transactions that a user finds unacceptably slow, we still need to know the answer to these two questions.

If the slow performance is measured by a load test script then getting the details is a pretty simple exercise. However, getting this information from test or pilot users can seem daunting. How many users will we need to survey? What if it's tens of transactions? Anyway, surely we've just got a general performance problem?

The good news is that it doesn't have to be particularly arduous. An effective approach is:

- Focus the team on one symptom
- Gain a detailed understanding of this symptom; right down to the menu selections, data inputs and button clicks that lead up to the problem
- Identify a few users who are experiencing this symptom and who are keen to help resolve the problem
- Devise a plan around those users to capture the information needed to identify the root cause

The first point here is key. Perhaps the biggest mistake made in an investigation is to assume that all symptoms are related. We'll look at this issue in more detail in the next subsection, *Searching for the Silver Bullet*.

If a test user repeatedly gets an error, or a load test script repeatedly fails due to an error, we need a similar level of certainty regarding the symptom:

- Do we have the precise wording of the error?
- Do we know exactly what the error means?

Technical teams will often go to extraordinary lengths to investigate a problem only to discover subsequently that they misunderstood the symptom.

A simple illustration here is something we probably all do every day; open a Word document.

The symptom of a slow performance problem is given as, “When I open a Word document it intermittently takes more than 30 seconds to load”. That sounds fairly precise. We perhaps want to know what the expectation is, and the user says that it usually takes less than 10 seconds. Excellent description – or is it?

There are several ways to open a Word document. With Word started the document can be opened via the File menu, without Word started we can double-click on a document in Windows Explorer or we can double-click on an email attachment.

What happens under the covers in each of these three cases is very different.

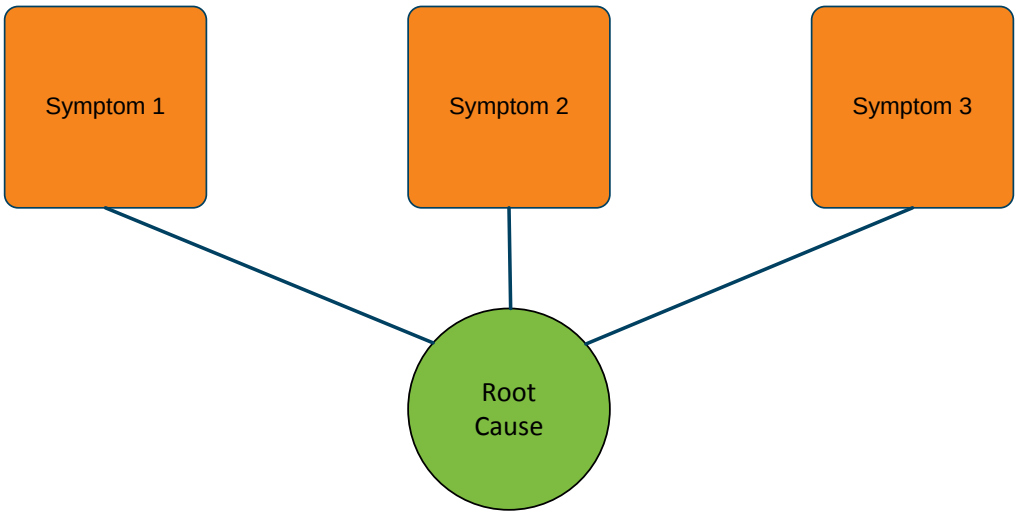
Searching for the Silver Bullet

When a project team is faced with several symptoms they may make the mistake of assuming that they all have the same cause. Perhaps the symptoms are very similar such as the all-encompassing, “The application is slow”, although the variation in scenarios can often be far more subtle.

An investigation based on this type of approach often relies on the mistaken belief that the more symptom information gathered, the better chance the team will have in resolving the problem. There is an easy way to spot this thinking as it is characterised by three activities:

- Constant consideration of new symptom information; even to the extent that details of live incidents are being fed into a call to discuss the problem
- Group discussions focused on theorising about the possible links between the symptoms
- Output from the investigation that is either a set of requirements for more information, or a list of changes to ‘try’, or a list of things to check

The initial mistake is that the team has started out with the belief that all of the symptoms have one root cause.



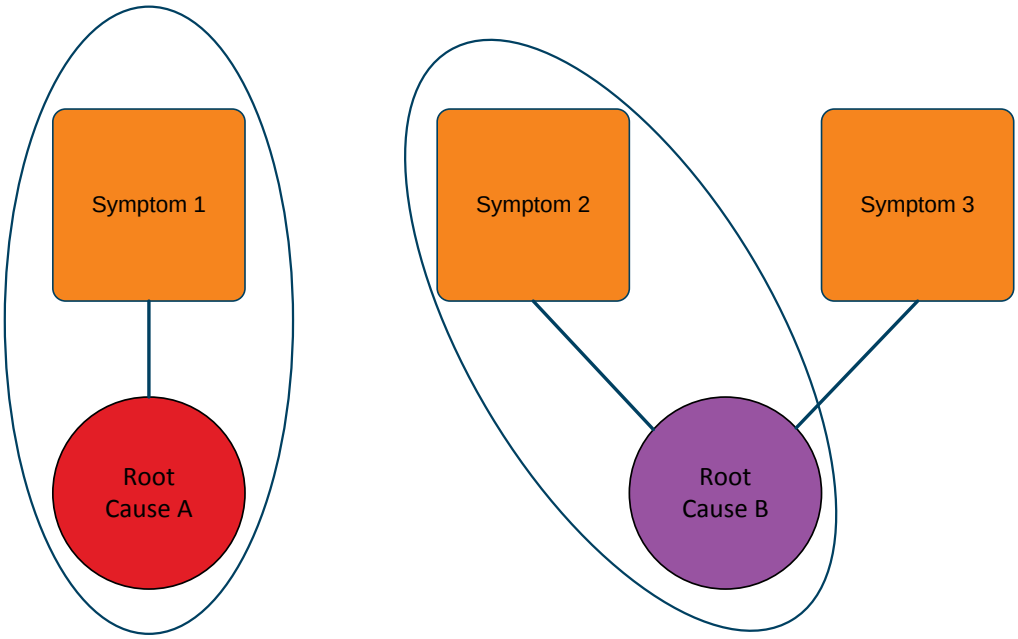
This is a very slow and unreliable way to diagnose a difficult recurring IT problem. If the symptoms do not have a single root cause, the team will just 'spin', trying to determine the link between symptoms that are not actually related.

A good example of this issue is a corporation that hit problems whilst migrating to a new IP telephone system. After deploying about 8,000 new phones to various pilot groups, the service desk was receiving a high rate of complaints. The project team determined that broadly the symptoms fell into two areas; audio quality and call dropouts. They therefore attempted to investigate the issues in the belief that there were two root causes; one for each problem. They spent months trying to determine the factors that linked all of the audio quality issues, and the link between the call dropouts.

There were in fact 52 unique symptoms with 18 root causes.

Although this is an extreme example of the 'silver bullet' issue, it powerfully illustrates the shortcomings of this approach.

If we focus on one symptom at a time we can avoid this trap and resolve difficult problems much more quickly – and the best thing is, we lose nothing.

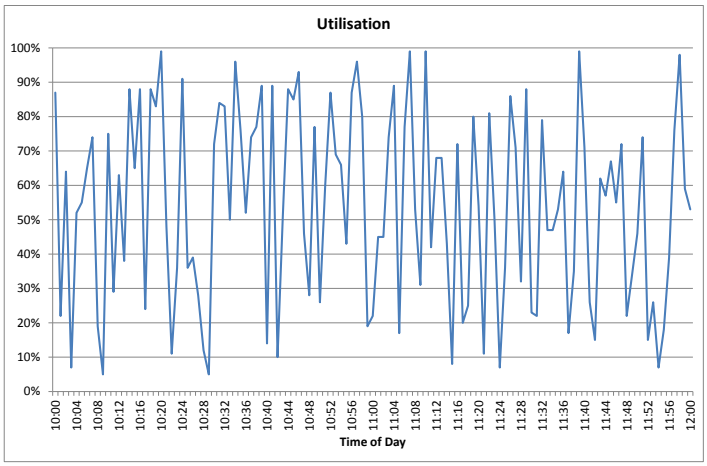


By focusing on Symptom 1 we will avoid the incorrect assumption that the three symptoms are related, and quickly find Root Cause A. If we then move on to investigate Symptom 2 we will find Root Cause B, and by removing this root cause we will fix the cause of Symptom 3.

Unfortunately there is an almost irrational concern that if we don't consider all three symptoms at the same time we will somehow hamper the investigation. Ironically, this is the complete opposite of the truth. Attempting to investigate all symptoms at the same time is the slowest and least effective way to diagnose a problem.

Beware of Wiggly Graphs

The performance of an application in a development environment with one or two developers testing it may be very different from that experienced during pre-production or pilot testing. This is sometimes mistakenly blamed on the infrastructure. It's not that the infrastructure couldn't be the cause, quite obviously it could, but the trouble is that the infrastructure is often blamed with very little evidence.



Evidence may appear in the form of a graph accompanied by an explanation based on 'experience' and other black arts. If the project team is shown a graph like that above, be sure to confirm that the presenter has objective criteria for determining the meaning of the graph.

The questions to ask are:

- At what point along the x-axis (at what time) did the problem occur?
- What is the sample period for each measurement; i.e. what is the averaging interval for each measurement?
- What level of utilisation (or queuing or other y-axis metric) and over how many measurement periods, indicates an overload situation?

There are objective recommendations for metrics like these and so we can't dismiss any conclusions out of hand, we just need to be sure that the presenter has that objectivity.

As a counterpoint, a lack of understanding of these factors can actually lead to failure to identify an overload condition that does exist.

"It will be Fine in Production"

Testing environments often differ from the infrastructure available in production; perhaps fewer or less-powerful servers. Therefore, it may be quite natural to

conclude that performance problems identified during testing will not afflict the production environment.

If the test team propose that a missed performance target will not be a problem once the application is transitioned into production it is worth checking two things:

- Can the load limit for satisfactory performance in test be extrapolated to determine the likely performance in production?
    - For example, if the performance during testing was satisfactory up to 100 concurrent users, are we sure that the system will scale in production to support an expected load of 250 concurrent users?
- What evidence is there to prove that the performance problem is attributable to a physical resources constraint that will be removed in the production environment?

Any constraint that is likely to be removed in production is most likely to be:

- CPU capacity – visible by checking CPU utilisation
- Free memory – visible by checking free memory volume
- Cache capacity – visible by checking cache utilisation and cache hits
- Storage bandwidth – visible by checking disk queuing
- Network bandwidth – visible by checking network utilisation

If the new system misses performance targets under a light load, any performance increase realised in production will simply be due to higher specification infrastructure masking an underlying problem. If so, as use increases and patterns of use change the performance problem is likely to return.

This is an important point: Advance7 has helped with many load test projects, and when a performance issue is identified it is hardly ever due to inadequate equipment specification. Load tests typically fail due to some logical bottleneck such as contention on a database table or the inability of a system to fully use the resources available (e.g. a single-threaded element running on a multi-CPU server).

Load Test Realities

A load test exercise is complex and the quality of the tests can vary considerably. A common failing is a poor or vague definition of Non-Functional Requirements (NFRs). An NFR should be quite specific, such as:

Whilst the system is handling a sustained load of 30 transactions per minute of the expected transaction mix, 95% of all 'Customer Search by Post Code' transactions should complete within 3 seconds.

We sometimes see tests where transactions are fired at a system at a totally unrealistic rate and the system fails to meet an arbitrary performance target. On the other hand we see tests that generate a ridiculously low rate of transactions from a single client and single userid.

A further common issue is the inability of the test team to determine the cause of a test failure (response time or error issues). The internal team or external test service provider may only deliver a pass/fail style result, leaving the project team to determine the cause. As mentioned elsewhere, a set of wiggly graphs (the typical output from a load test) can't be used to determine the root cause of a problem reliably. We need to get the people who would troubleshoot problems in production to fully engaged in the load test exercise. That way we can be sure that there is a plan to capture the right information during testing. There's no point deciding that we need more data when the test is over and the report has been delivered. Like Elvis, the testers may by that time have 'left the building'.

Even with a high quality load test exercise, the testers simply can't test every transaction permutation, and so:

- If you feel the quality of testing was poor, allow adequate time for project technical people to handle problems that will arise during pilot or early deployment
- If you are happy that the testing was conducted to a high quality, make sure system knowledge is available to the operations staff to handle any problems that were not detected in testing

If the root cause of a problem is found and a fix applied, make sure that the system is tested again. The applied fix may not correctly address the root cause found, or there may further problems that simply weren't encountered because the system wasn't fully exercised in the earlier failing tests.

Fixing by Anecdote

Identifying this issue is unfortunately a matter of judgement. Technical people and suppliers will have a tremendous amount of experience, and we want to benefit from it. When a problem arises, someone in the project may have seen the problem before and know the solution. However, people can be so keen to help, it can skew their thinking; an effect described in Daniel Kahneman's excellent book *Thinking, Fast and Slow* (2011).

We have to bear in mind that there are many motivations that can cloud the judgement of the people helping to solve the problem, ranging from a true desire to fix the problem, through meeting sales targets, to a wish to include a technology on a CV.

The solution here is similar to that in earlier subsections. We need to ask what evidence is needed to confirm that this problem has the same cause as another experienced elsewhere, and so how do we get that evidence?

Pattern Method Difficulties

Looking at patterns of symptoms is a natural way to diagnose problems; we all do it in our own lives. In the IT industry the thinking goes something like this:

- The pilot users only experience problems on a Monday morning
- The users who are complaining are all in the Manchester office
- The users who experience the problem are using Windows 7 PCs whereas those with Vista don't report the problem

Based on these 'facts' we might deduce that the problem is caused by something that happens in Manchester on a Monday morning that only affects Windows 7 PCs.

There are several shortcomings in this way of thinking, some of which are:

- The information we are given may not be quite correct, for example we may discover later that London users also occasionally get the problem
- The information is not comparable, for example the users in the other offices rarely use the transaction that has the problem
- There are other unknown differences, for example the profiles of the Manchester users differ from those in other offices due to the installation of another application

If the technical teams are basing the investigation on patterns, watch out for the warning signs described on page 9. This will indicate that the pattern-based approach is no longer viable and the team needs to switch to an evidence-based approach, such as that outlined in the next subsection.

During the investigation of a response time problem with a leisure company's appointment booking system, the developers insisted that the database servers located at each branch were not used for appointment transactions.

It was later found that the database was used, and it was introducing a delay. The developers had forgotten to comment out three lines of code that were no longer needed.

This seems so obvious – surely these were just bad developers. But the fact is we've all done it and if you are investigating a problem right now it's a sure-fire bet that the investigating team has made at least one incorrect assumption. If that assumption has a key role to play as a 'fact' in a pattern-based method, the method will fail.

Switching the Focus

A technical team, support person or supplier may feel under considerable pressure if there is a general belief that their technology is to blame for a problem. Obviously, this state of affairs isn't conducive to rational thinking and effective problem diagnosis.

The issue stems from too much emphasis on what's causing the problem. That may sound counter-intuitive. Of course we want to know what's causing the problem, but an investigation that is too focused on causes can be interpreted as a blame game causing technical teams to spend time defending their technology and their team's actions.

A solution to this issue is to shift the focus to determining how we can get the evidence that will identify the root cause of the problem. This seems a subtle change but is actually a big shift in thinking. It changes the behaviour of technical people as thinking based on evidence is less emotive and tends to appeal to their analytical minds. What's more, taking this approach produces a more reliable result much more quickly than concentrating on the question of "What's the cause?"

Triggering this change is very simple. Convene a meeting with a representative from each technical team involved in the project. Make the statement that we don't know what the cause of the problem is and ask the meeting how we can find the cause; what information do we need?

Occasionally people in the meeting will object to some point made with a phrase that equates to, "That doesn't explain why … ". For example, the database team may say that there is no point collecting data from the database server as it's used for many applications and none of these have a problem. We need the team to assume nothing and so it's useful to state at the beginning of the meeting that the phrase

"That doesn't explain why …" and related statements are banned during this meeting.

Advance7 has considerable experience in dealing with entrenched technical teams. We have found the technique of switching the focus to be very effective.

Reluctant Supplier

There can be a problem getting a solution provider engaged when a problem arises with a new application or upgrade. The provider can be too quick to dismiss the problem with the ubiquitous response, "We don't see this problem at any other customer; it must be your infrastructure".

In defence of the suppliers (at least the honest ones), it's the IT department's responsibility to integrate the components into a working system. Let's imagine that a supplier did actually spend, say, 14 man-days fixing a problem that turned out to be caused by something they did not supply – how would the cost of this investigation be covered?

However, the supplier should be clear on the criteria used if they are to accept that they have a problem that they need to investigate.

Therefore, if the project is faced with a grey problem[1] call the supplier and:

- Explain the problem the team is facing
- State that you don't yet know what is causing the problem
- Tell them that the team are going to collect detailed data at the time of the problem
- Ask what information they would need to prove that the problem was due to their technology

This can save a lot of time as if the project team carefully collects the data requested, and the problem is being caused by the supplier's hardware, software or service we should be able to get them fully engaged very quickly.

[1] Grey problem – a problem where the causing technology is unknown.

Dog with a Bone

Earlier in this paper we cited the use of the phrase “We are just going to try one more thing” as a warning sign. Other variations are, “We are making progress (that can’t be measured) but we have more work to do”, and “We are certain that the next change/upgrade will fix the problem”. This can be a very big issue.

If a technical person has been working on a problem for some time it can be very difficult to get them to change their approach. Technical people don’t like to be beaten by technology, and it’s just as well they feel that way as that’s how we know they are committed. Changing direction and getting others involved can be a bitter pill to swallow, particularly as the investigator may feel that they are about to make a breakthrough.

However, don’t let a problem run on for a long period as it then becomes progressively more difficult to pull people around. If a change of approach is needed, make sure that the original investigator is part of the solution. We need their knowledge and skill, we just need to be careful not to follow their train of thought or we are likely to end up at the same point that they had reached.

IN CONCLUSION

It’s often difficult for a project manager to guide technical support people in the investigation of a problem; they can quickly slip into a state where they think we are trying to teach them how to do their job. In this paper we have looked at effective techniques that can remove the emotion from an investigation and help the project team to gel into an effective problem solving group.

We defined the key techniques to be:

- Identifying signs that conventional troubleshooting is not working and so switching to the approach outlined in this paper
- Gathering detailed and accurate definition of the symptom
- Defining how the symptom differs from the expected behaviour

- Investigating one symptom at a time to avoid the assumption of a single root cause
- Avoiding subjective interpretation of graphs by asking for judgement criteria
- Asking technical people for evidence to support conclusions reached regarding root cause
- Focusing the team on determining how to find the cause of the problem
- Making sure that load tests have been conducted with realistic transaction rate and mix parameters

Advance7 uses all of these techniques regularly and they work.

ACKNOWLEDGEMENTS

In the production of this paper we had help and advice from a number of people. Thanks to all for their contribution:

Mark Bairstow, Advance7

Ian Hersey, JIH Projects

Susannah Mann, Baker McKenzie

David Rowatt, Atos

Dominic Digby, Mizuho Bank

Nicoletta Curtis, Unum

AUTHOR

Paul Offord has had a 37-year career in the IT industry that includes roles in hardware engineering, software engineering and network management. Prior to founding Advance7 in 1989, he worked for IBM, National Semiconductor and Hitachi Data Systems. Paul is now the Development Director at Advance7, and in that role he has led the development of the RPR problem diagnosis method.

Having spent 23 years troubleshooting problems for many of the world's leading companies, Paul now writes and teaches on the subject of problem investigation and diagnosis.

Paul is a Certified IT Professional and a Fellow of the British Computer Society, the fellowship being awarded for his work on IT problem diagnosis.

ADVANCE7

We are passionate about IT troubleshooting. We believe that conventional methods are useful but have limitations and so we've been working hard to find more effective ways to help people resolve issues quickly and effectively. We help our customers by providing:

- Hands-on troubleshooting services – to help project and operations people solve difficult problems
- Education and support services – to train and support IT professionals in the use of advanced troubleshooting methods and techniques

For more information visit our website at www.advance7.com or call us on 01279 211668.

Advance7
Endeavour House
Coopers End Road
Stansted
Essex CM24 1SJ
United Kingdom

Phone: +44 (0) 1279 211668
Web: www.advance7.com

Revision 1.06