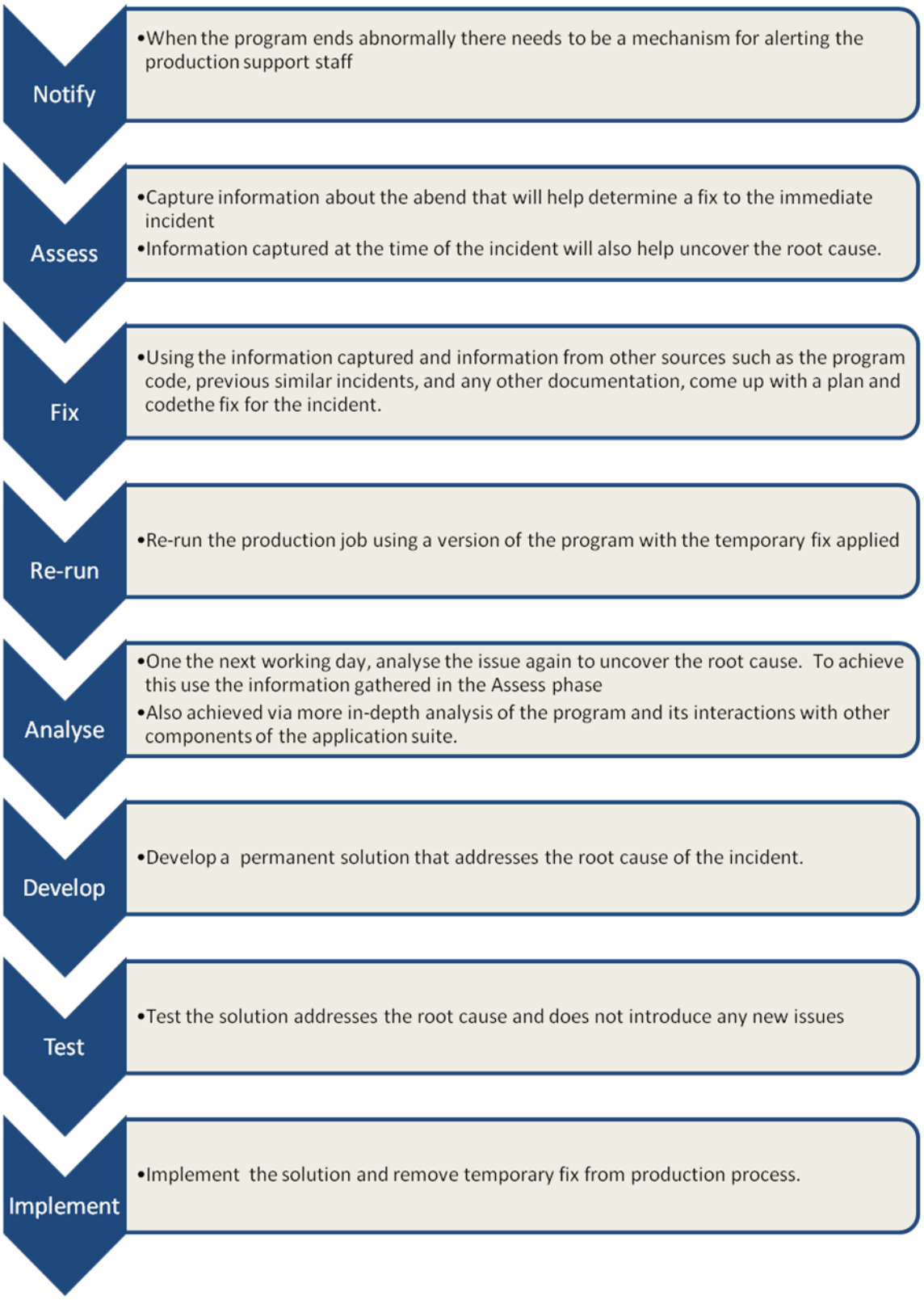
Production Support Use Case

The term Production Support in this context refers to the work done by applications staff to support the successful completion of production batch processing, typically overnight, that forms a vital part of an application. In many sites this batch work must be completed by a certain cut-off time to allow the online system to be fully functional and ready for another day of activity. While there may be a move towards continuous processing and away from the online/batch split, there will always be batch jobs that need to be fixed quickly. Those that support these important batch processes provide a vital service to the organisations that employ them. This chapter shows how the tools and processes of the Optimised Lifecycle can assist Production Support staff address the issues they face.

Production Support Process Flow

Some draw the analogy between the Production Support role in an Information Technology Department and the Triage principles employed in the Emergency Room of a Hospital. While it may not be a matter of life and death the analogy holds true when considering the prioritisation of the work to be done. The issues need to be brought to the attention of those that are best equipped to deal with it. Tasks need to be assessed and prioritised based on their urgency. Jobs on the critical path of the batch stream need to be fixed before jobs that are not. Also it is also important to capture information at the time of the ABEND that will be of value in both dealing with the immediate issue and also later when the root cause of the problem is investigated. Once the immediate concerns are dealt with effectively, the root cause and its solution can be developed, tested and implemented.



(Figure 1)

The Scenario

In this case a batch job has abended and a call has been placed to the on-call production support applications programmer. This is the first of the eight phases that need to occur from this point -

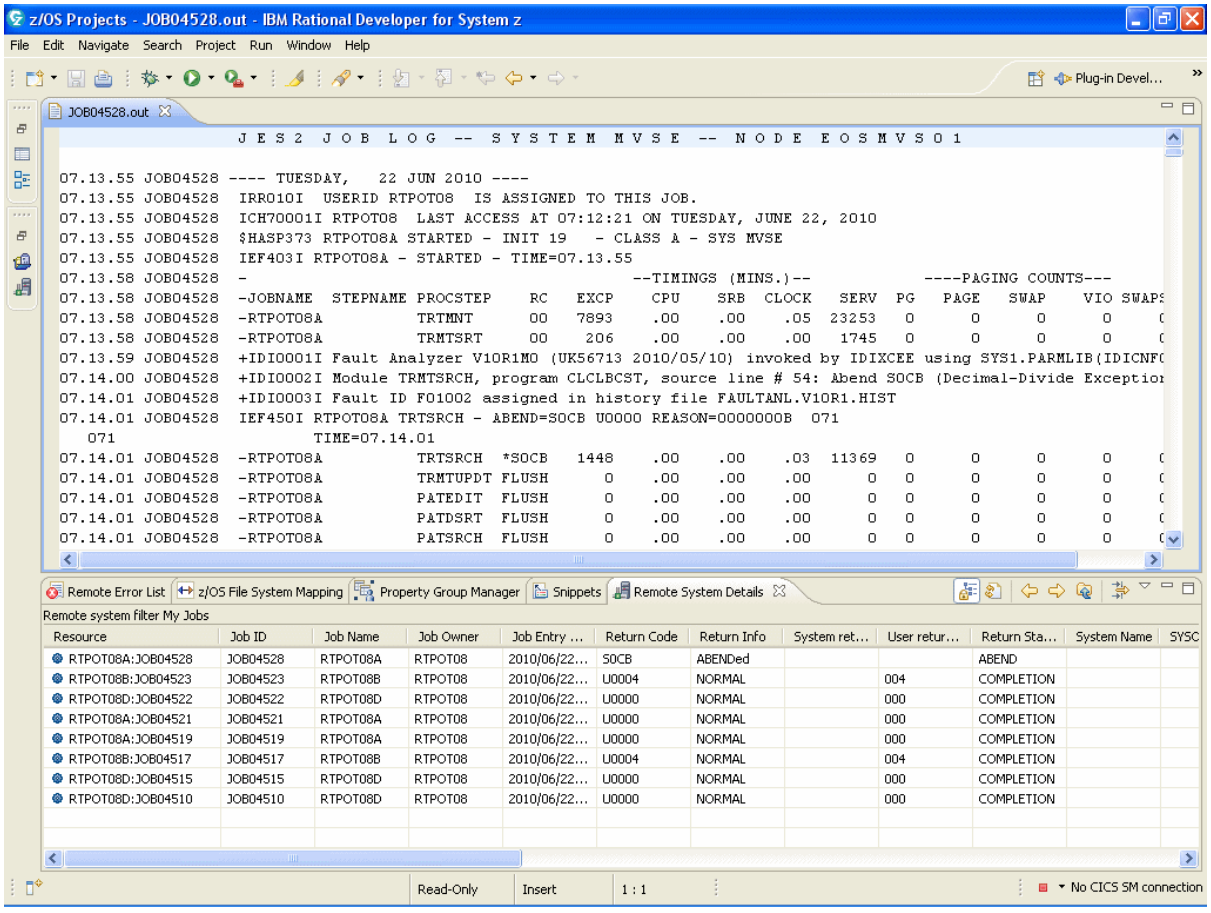
Notify. Notification can be as sophisticated as automation software sending an SMS to on-call staff alerting them to a failure. It could be as simple as a Computer Operator on shift reacting to a non-scrolling message on the system console, issued by the failing job. The main thing is that the notification happens quickly to allow maximum time for the next three steps.

Assess, Fix and Re-run

The Programmer is faced with S0CB ABEND in subprogram CLCLBCST (Figure 2) of TRMTSRCH. The joblog shows that Fault Analyzer has captured the ABEND information. Fault Analyzer lists the following handy information in the job log:

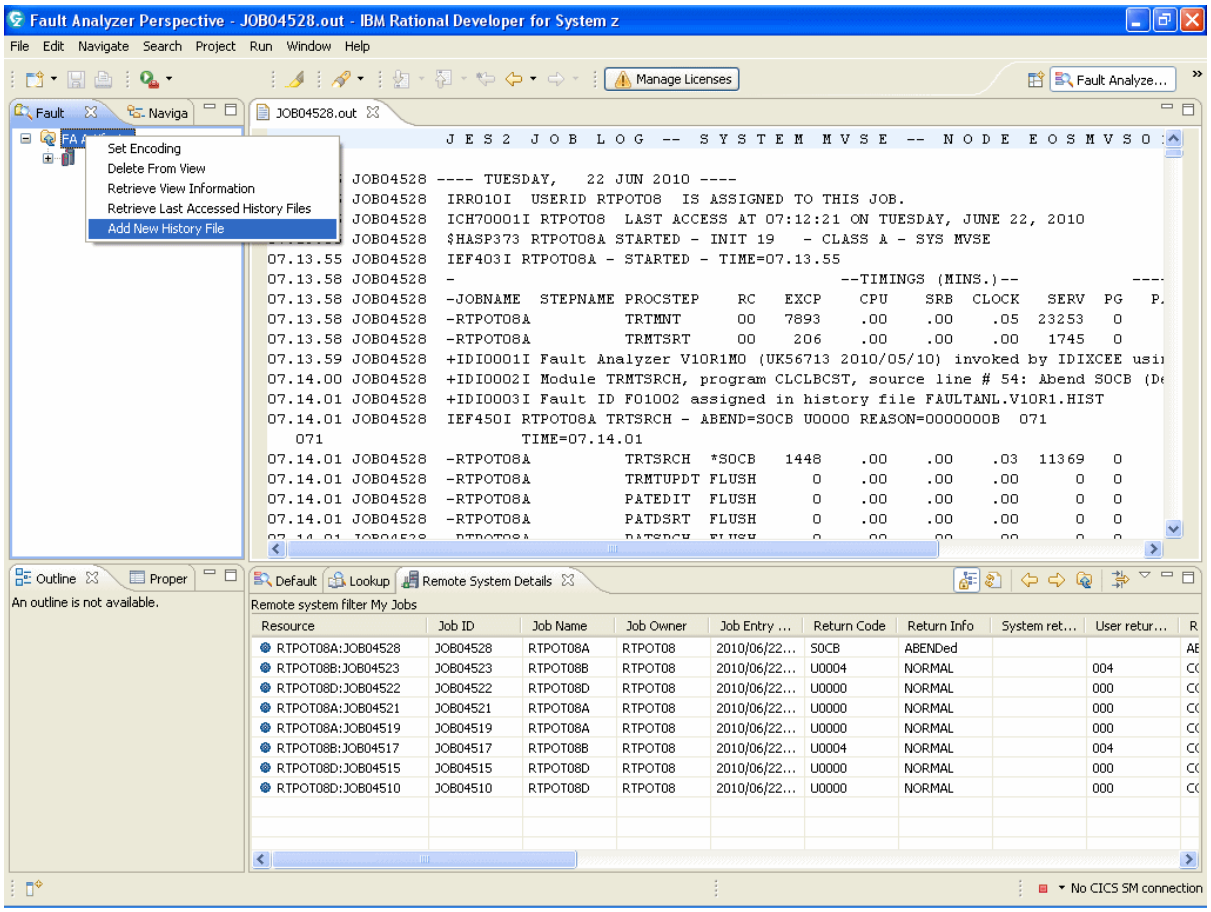
- Program and Module and Source Line number
- ABEND information including a short description (Decimal-Divide Exception).
- The Fault Id and the History File where the abend information can be found

Note: To gain maximum information from the tools insure programs are prepared with the recommended options. Refer to the production documentation for these settings



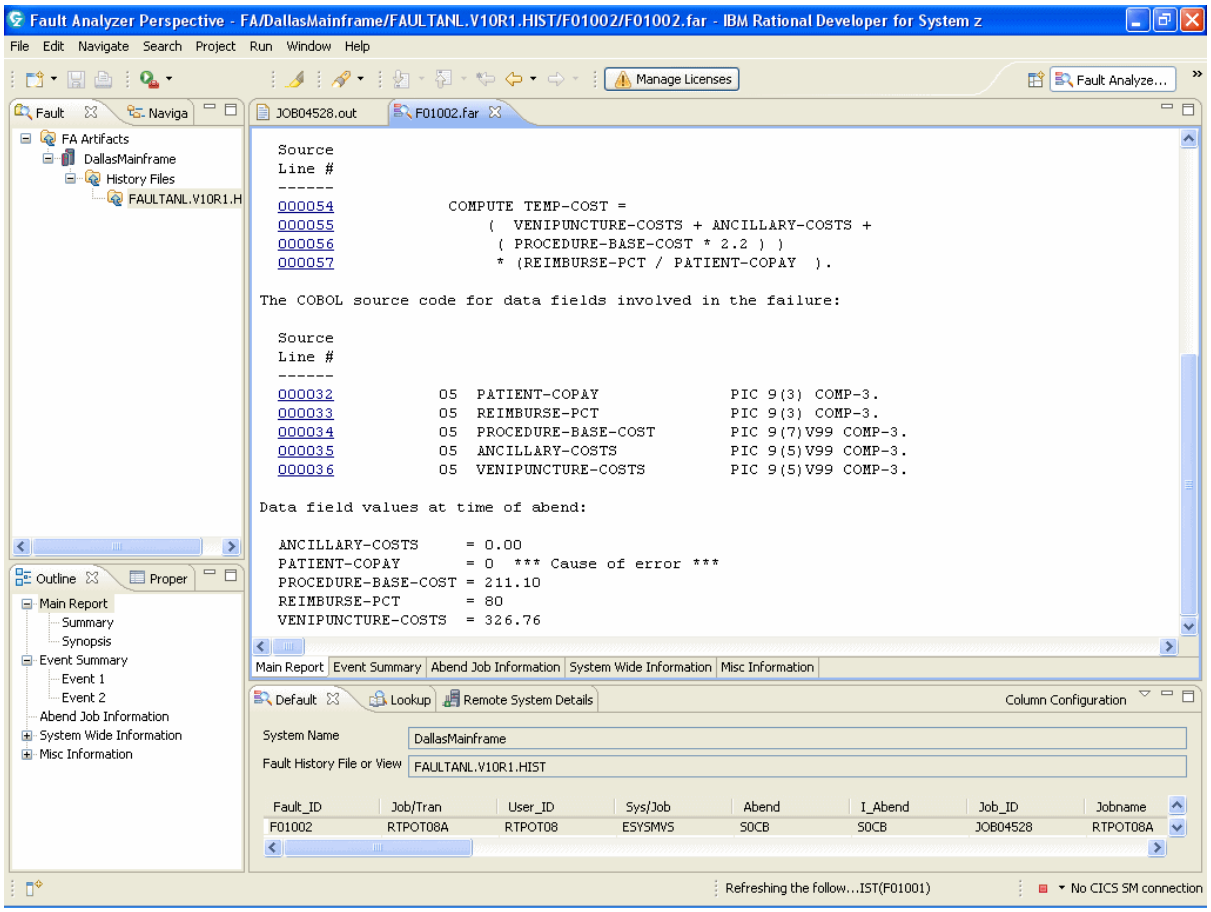
(Figure 2)

To view the Fault Analyzer report the programmer should switch to the Fault Analyzer perspective. As detailed in the PD Tools chapter, the history file displayed in the joblog is added via the context menu (Figure 3)



(Figure 3)

The Fault Analyzer Report (Figure 4) will show the source line that immediately preceded the failure. In this case the divisor in the Compute statement on line 54, PATIENT-COPAY has a value of 0. Hovering over PATIENT-COPAY in RDz would tell the programmer the field is part of the linkage section of the program so is passed to the program from the calling program – in this case TRMTSRCH.



(Figure 4)

The programmer feels they can insert code into the program to check if the field is 0 and move a valid value to it. The programmer consults the application documentation to discover default co-payment factor is 100 but that some states will have a lower factor set. The programmer decides to add a test for the field being zero and if so move 100 to the field to eliminate the S0CB.

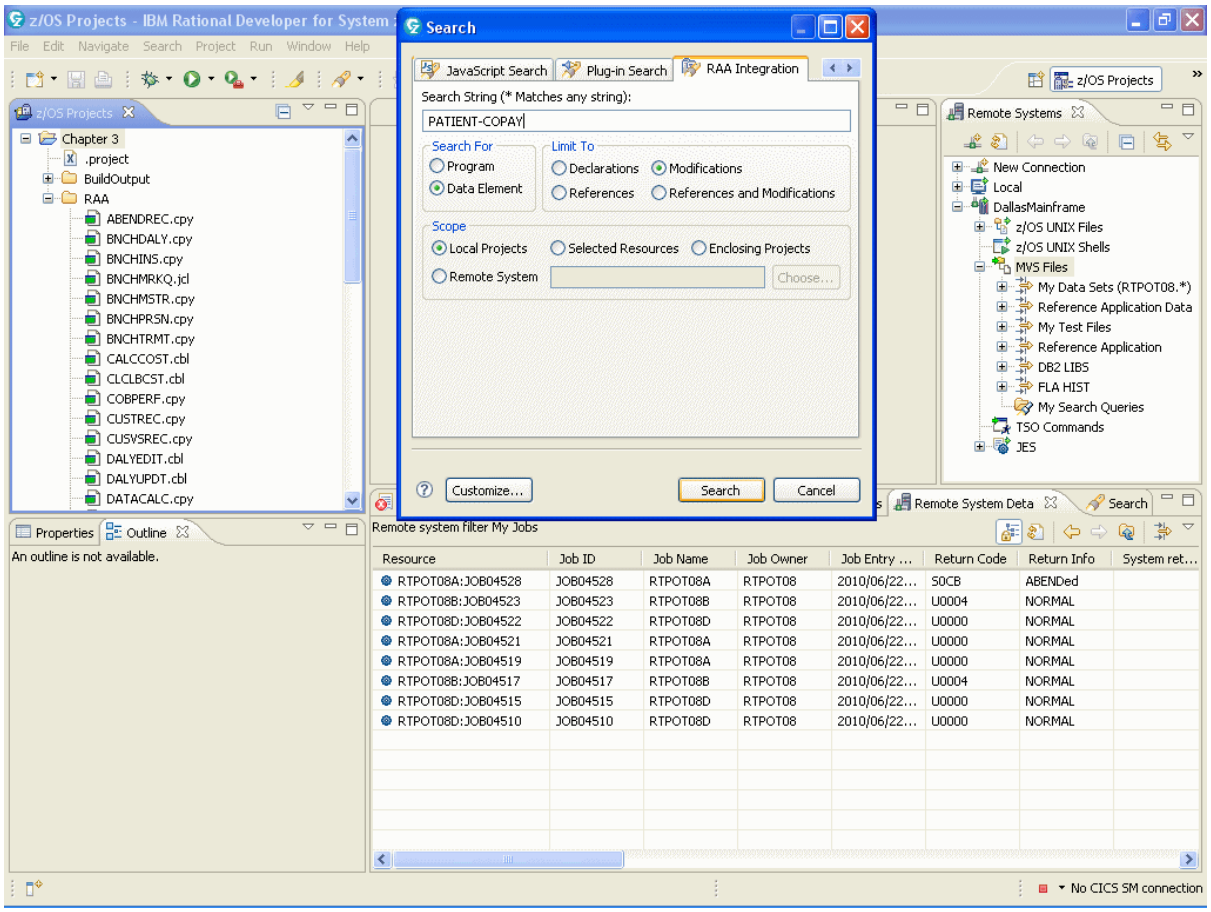
The code is added and the program is recompiled to a temporary library. The production job is re-run successfully from the temporary library. This allows down-stream processing to complete within the window allowed for batch processing under the service level agreement in place.

```
100-CALC-LAB-COSTS.
**  Emergency Fix to allow batch to complete
     IF PATIENT-COPAY = ZERO
            MOVE +100 TO PATIENT-COPAY.
     COMPUTE TEMP-COST =
          (  VENIPUNCTURE-COSTS + ANCILLARY-COSTS +
           ( PROCEDURE-BASE-COST * 2.2 ) )
           * (REIMBURSE-PCT / PATIENT-COPAY  ).
```

Analyse Root Cause and Develop Solution

At the next opportunity an analyst within the applications area supporting the batch job, needs to determine the root cause. This involves trying to understand how and what sets the contents of the PATIENT-COPAY field within the patient master file. RAA can be used to determine where a field is

modified within a set of application components including programs and copybooks. Also RAA can be integrated with RDz so that the searches can be done from within an RDz perspective. Right click on the project that contains the elements that are to be searched. Then from within the search window click on the RAA Integration Tab (Figure 5). Enter the field name, PATIENT-COPAY in the search string, specify to search for a Data Element and limit the search to modifications of the data element.



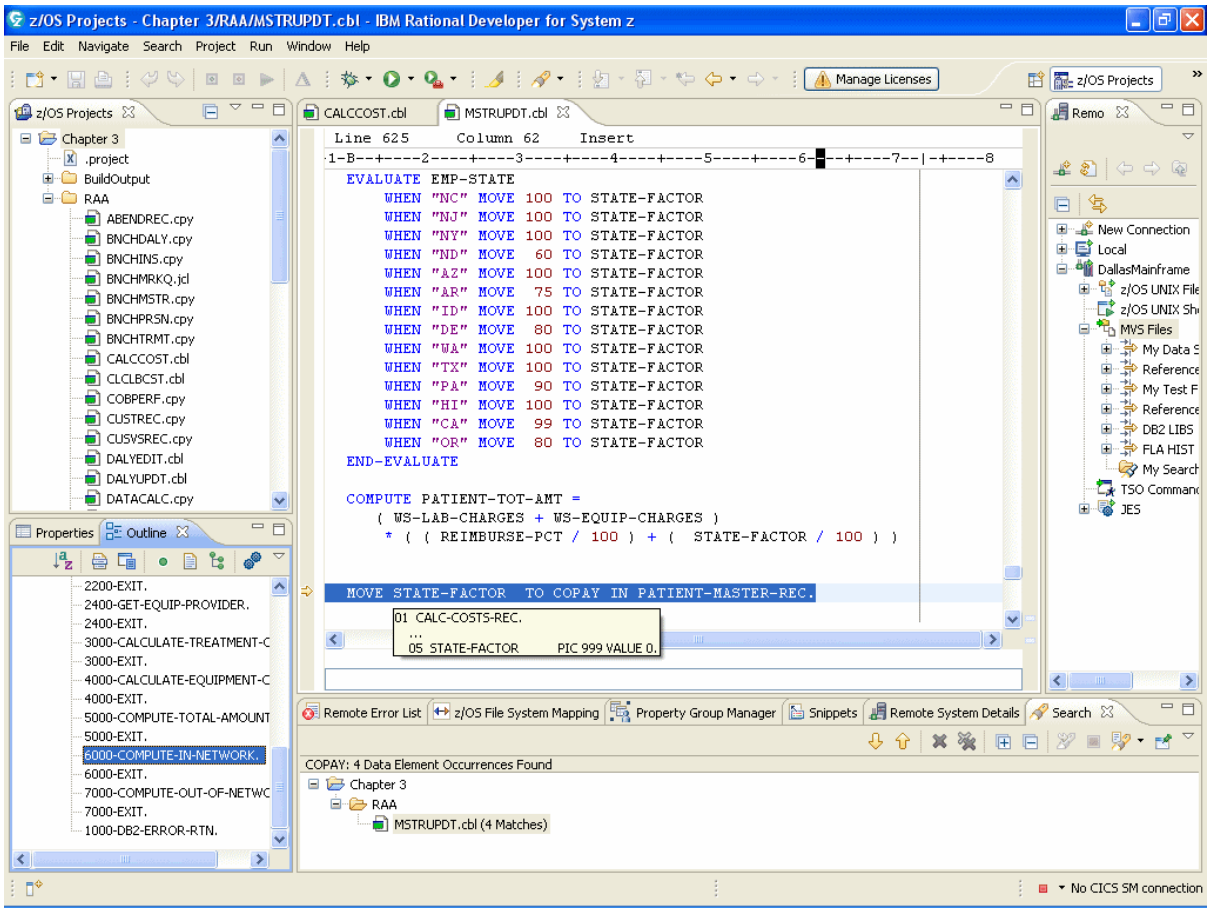
(Figure 5)

The search reveals a number of references. Each reference needs to be investigated. A modification within PATSRCH shows that another field called COPAY from the PATIENT-MASTER-REC (Patient Master File) is used to populate PATIENT-COPAY.

```
       300-CALC-EQUIP-COSTS.
           MOVE "300-CALC-EQUIP-COSTS" TO PARA-NAME.
           MOVE PATIENT-ID IN INPATIENT-DAILY-REC TO
             PATIENT-KEY,
               PATIENT-ID IN CALC-COSTS-REC.
           READ PATMSTR INTO PATIENT-MASTER-REC.
           MOVE COPAY  TO PATIENT-COPAY.
```

The task now is to trace how COPAY in PATIENT-MASTER-REC is populated within the entire application. Again the integration between RDz and RAA (RAAi) can be used to complete an

intelligent search of the application. In a similar manner to the search above for PATIENT-COPAY, the analyst searches for modifications of COPAY. This reveals it is set to STATE-FACTOR which in turn is based on an evaluation of the EMP-STATE field (Figure 6).



(Figure 6)

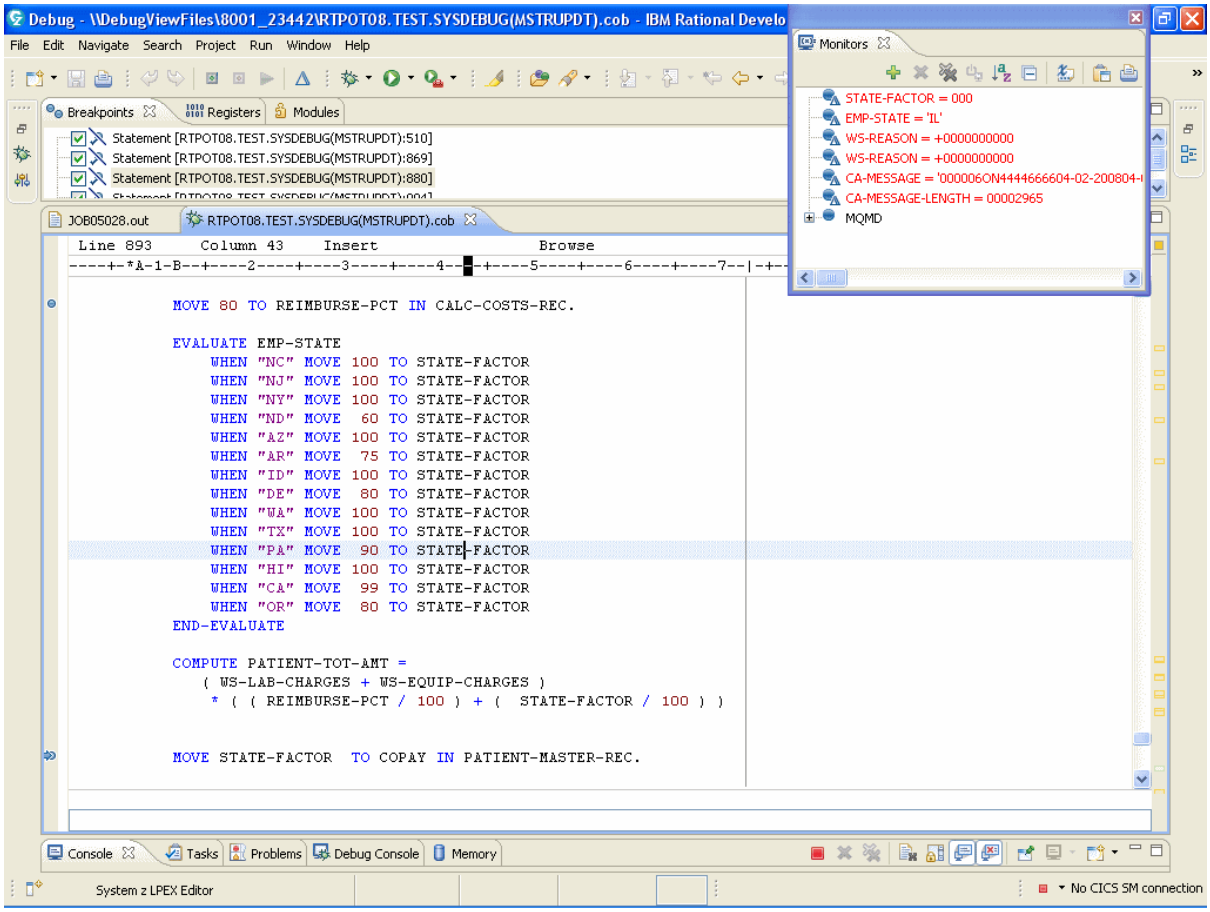
The analyst notices two things from looking at the code excerpt and from hovering over the field:

- The list of State codes is not exhaustive and there is no WHEN OTHER clause on the Evaluate to catch unexpected values.
- The initial value of STATE-FACTOR is set to 0 in the working-storage.

This means that if a State code other than those listed explicitly in the Evaluate statement, COPAY will be left at the initial value of 0 rather than the accepted default value of 100. To test the theory that there are values of EMP-STATE possible that are not catered for, RDz and its Debug perspective are used. Once in the debug session the following is done to setup for the test:

- A breakpoint is set at the following statement:
  `MOVE STATE-FACTOR TO COPAY IN PATIENT-MASTER-REC.`
- Highlight the fields we are interested in (EMP-STATE and STATE-FACTOR), right click and choose to monitor them. In this way we can see their values change as the program is debugged.

Figure 7 shows the results after a couple of records are processed in the debug session. Note the values of EMP-STATE and STATE-FACTOR in the Monitor window.



(Figure 7)

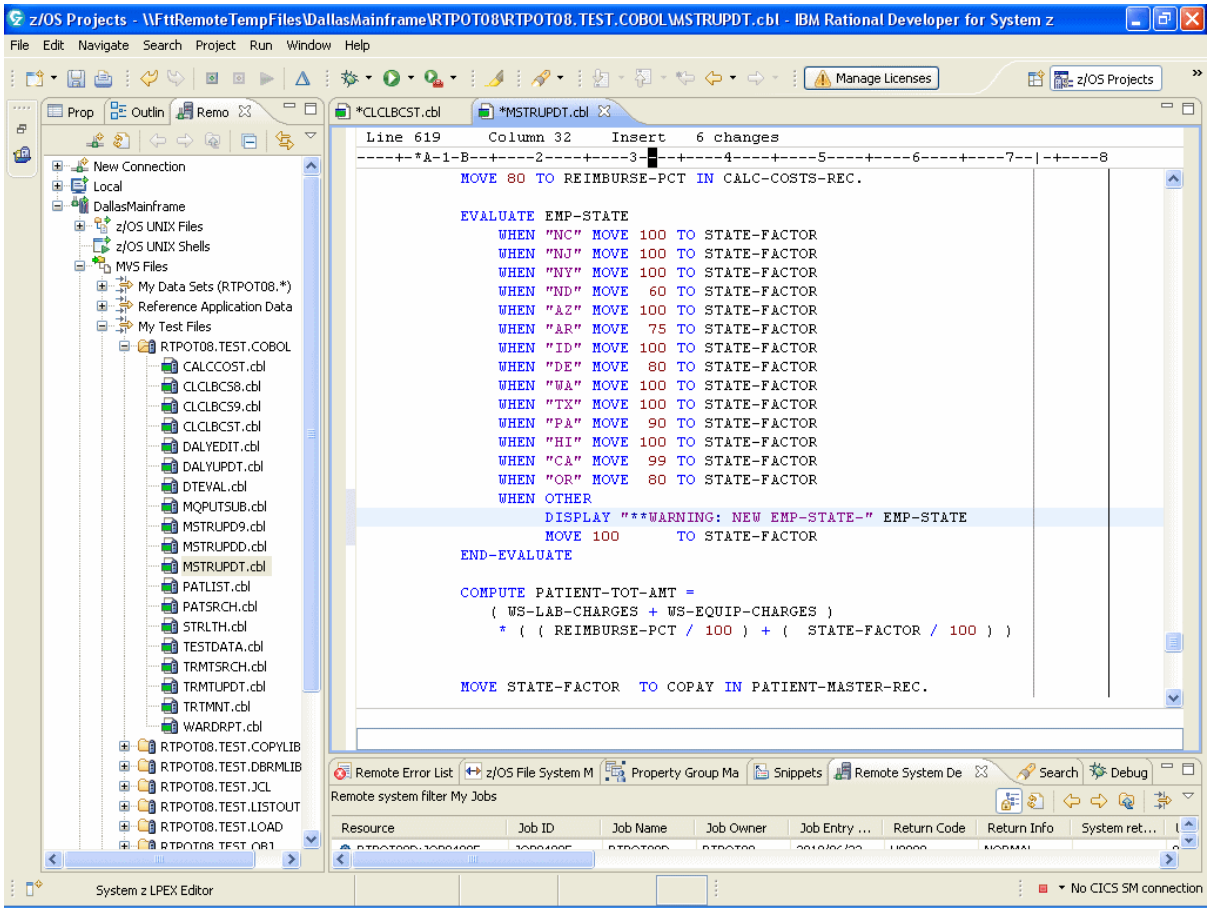
EMP-STATE had a value of “IL”. This is not catered for in the Evaluate statement so at the breakpoint, just before COPAY in PATIENT-MASTER-REC is modified, STATE-FACTOR is still at its initial and invalid value of 0.

It is good practice with EVALUATE statements to always include a WHEN OTHER clause. This makes the program more resilient. While normally the addition of new states from a business perspective may have meant the applications area would be notified so they could make the necessary program changes, this does not always happen. Programs need to be written to expect the unexpected.

The initial intention of the application was to have a default Co-Payment factor of 100 but to allow for lower factors for some states. The change shown in Figure 8 is agreed upon. The change will set the factor to the default value of 100 but also display a warning message on the joblog. It may be better if these exceptions were also written to an error file but for this example we will use a display to the job log only. The idea is to highlight the issue so it can be followed up and addressed, rather than letting it keep occurring and causing unforeseen issues as it did in this scenario.

The fix will have three effects. Firstly the original intention to have a default factor set will be implemented and secondly it will stop any future repetition of this S0CB abend. Thirdly, the warning message will highlight to the applications team that a new state is now incorporated into the database. This will provide an opportunity to discuss the introduction of the new state with a

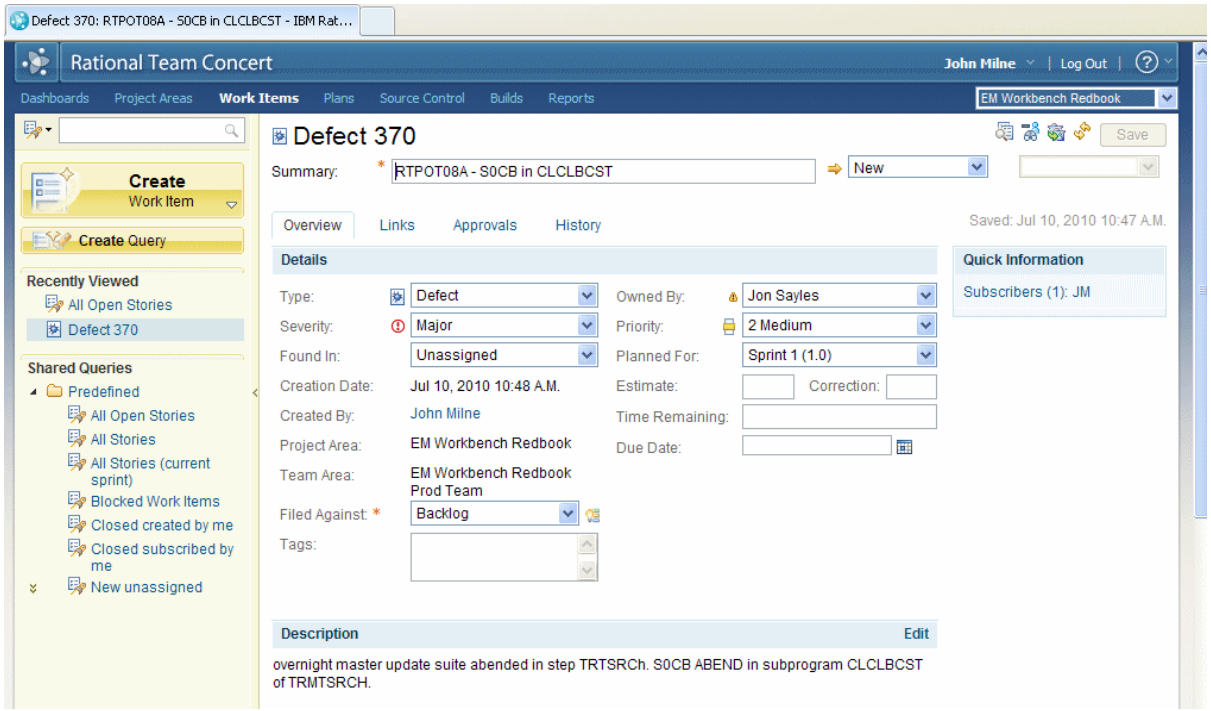
business representative and ensure that the co-payment factor chosen for each is appropriate rather than defaulting.



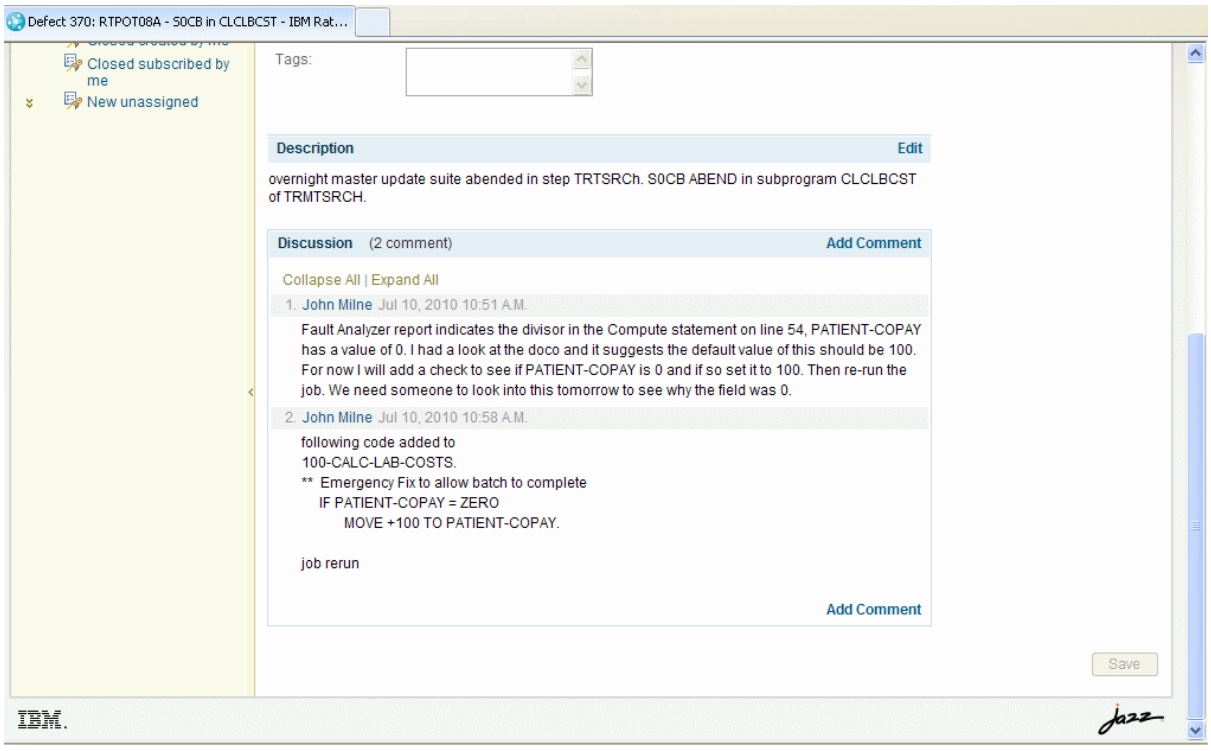
(Figure 8)

Test and Implement Solution

Implementing the solution involves following the development and testing methodology adopted by the site in question. Examples of the methodologies are discussed earlier in Section IV, including: traditional approaches such as the waterfall method; collaborative approaches such as the Iterative Approach using RUP or SCRUM; Agile Methodology which can enabled by use of a collaboration tool such as Relational Team Concert for z (RTCz) (Figure 9 & 10). RTCz like RAA can be integrated with RDz and forms an important part of IBM's Enterprise Modernization Workbench.



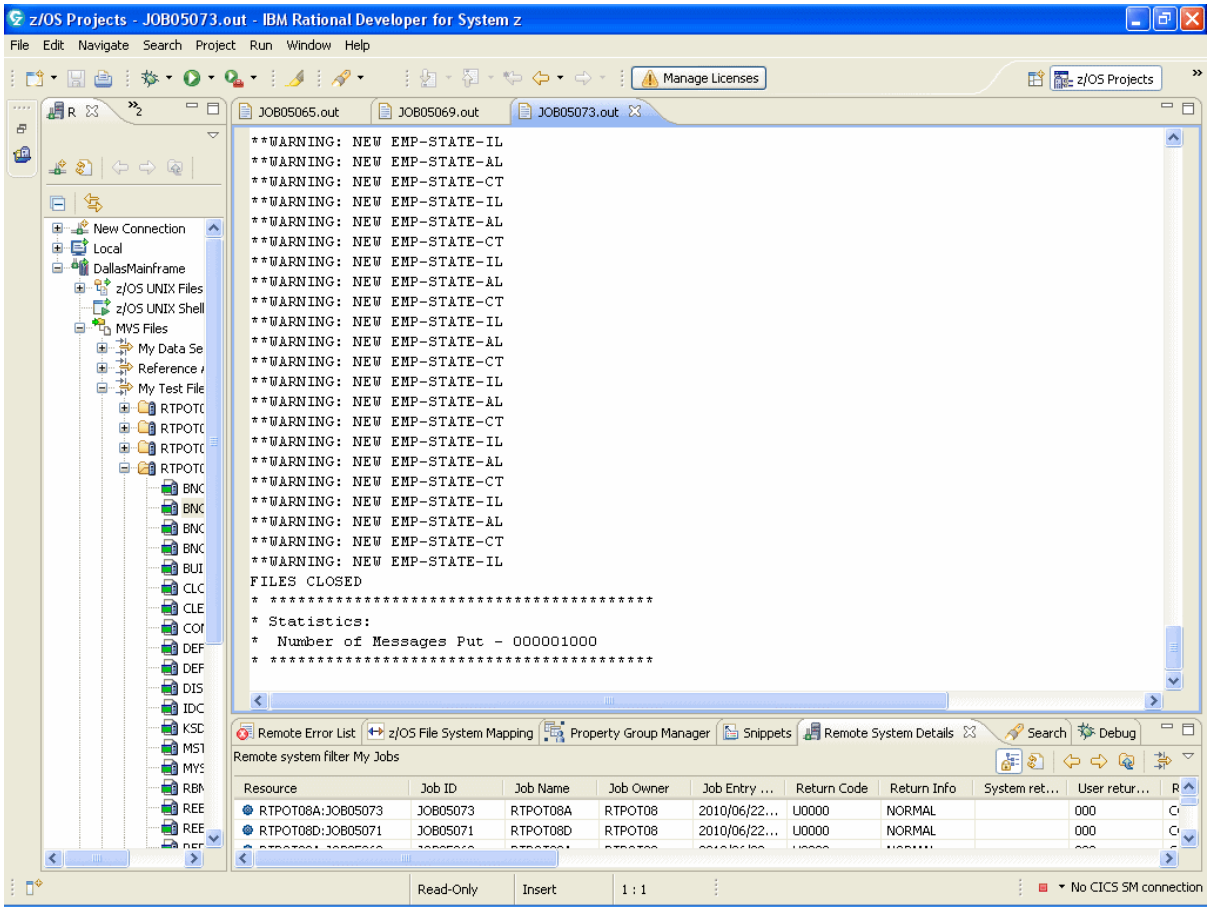
(Figure 9)



(Figure 10)

Irrespective of the approach, it is important to test the changes have the desired results. After retrieving the programs, the changes are made in a test environment. The job is run in that test environment using sample data and/or copies of production data. As expected it reveals a number of states that have been introduced to the database without changes to MSTRUPDT to ensure the COPAY field is correctly set (Figure 11).

There may be more testing required before the program is migrated to production, including a system/integration test of the whole system to ensure this change does not have unforeseen consequences downstream. The other task that needs to be done at the same time the solution to the root cause is migrated to production, is to remove the temporary fix to a version of CLCLBCST to ensure the new, full solution is picked up when the suite next runs.



(Figure 11)

Summary

This chapter has demonstrated how the tools of the Optimised Lifecycle can be used to improve the response to a production support ABEND. It is possible to look into these sorts of issues without the tools discussed and without abend analyzers such as Fault Analyzer, or third party alternatives. COBOL can provide a symbolic dump and if a compile listing, with appropriate options, has been kept it is possible to derive the failing line of code from standard module/offset ABEND information. Also applications staff with deep application knowledge and experience can quickly drill-down on issues without much help. In fact this is how production support issues were investigated and may still be at some sites.

What was demonstrated in this chapter was how the tools now available and the integration between them can benefit those who tackle these issues on a daily basis. These Production Support incidents are often a stressful events for this involved that, if not addressed in a timely manner, can have major negative impacts to the organisations concerned. The value of the tools is that they can

reclaim time that in the past is burned identifying the point of failure and indentifying a fix. This time saved can be the difference between getting systems back on track or not.