Safety Assessment of Systems Embedded with COTS Components by PIP technique

Luping Chen, John May

Safety Systems Research Centre, Faculty of Engineering
University of Bristol, Bristol, BS8 1UB, UK
L.Chen@bristol.ac.uk, J.May@bristol.ac.uk

Abstract: The difficulties to assess reliability of systems that use COTS components are sometimes compounded by the inaccessibility of some COTS codes. This paper develops an approach of Perturbation of Interface Parameters (PIP) to simulate failures of COTS components. It is to validate the use of PIP as a fault-injection technique to test COTS components and surrounding systems. Tests of a nuclear protection system will be presented to demonstrate that PIP can be used to assess and aid safety designs in COTS based software.

1 Introduction

The increasing need for fast, economic methods of software reuse and interoperability between applications has driven the rapid growth of component-based software engineering. Commercial Off-The-Shelf (COTS) components already play an important role in the component development paradigm [BA98]. Component-based software engineering (CBSE) offers the potential of economic and speedy system design and production. At a course grain this can be based on the reuse and integration of high-level COTS components together with bespoke components as elements of a required new system. The use of COTS components offers possibilities for increased productivity and efficiency in software development, but raises new safety issues since the reliability of COTS software components cannot be performed in a one-off manner prior to integration. Furthermore, even if such pre-assurance was a theoretical possibility, it would seldom be available, since COTS components are commonly developed to unknown standards or standards aimed at general use, which are insufficient for safety applications.

A key requirement for such hybrid systems is to show that the use of COTS components (which will be considered as 'black boxes') does not compromise the safety, reliability and (perhaps) security of the overall system, since the reliability of COTS software components cannot be fully assured prior to integration. Currently, developmental processes used by the various COTS suppliers are most likely unacceptable to the mission/safety critical safety and certification communities. Alternate and supplement methods are needed for the certification of these systems [PJ96]. These difficulties are sometimes compounded by the inaccessibility of some COTS code. Where examination of code is not permitted, traditional assurance techniques (with the exception of black box testing) cannot be applied to COTS components post-purchase, to supplement the supplier’s verification and validation (V&V) activities. In general it is necessary to use 'middleware', possibly based on standard infrastructure technologies, to integrate disparate components. This middleware offers an important opportunity to include component adaptation and monitoring strategies, to help ensure fault tolerance, quality of service and security. The overall integration process can be complex, perhaps involving both syntactic and functional adaptation, but this study will have the simple focus of how to assess the value of safety strategies (in the surrounding system) to prevent the propagation of faults or unexpected behaviour from the COTS component to the rest of the system.

Some earlier works have developed the use of fault injection to test the diagnostic fault coverage provided by such COTS wrappers. One approach in [NJ01] investigated the diagnostic value of executable assertions using Software Fault Injection (SFI) inside the COTS component to cover a range of fault sizes (footprint in the input space) and locations. However, where COTS code is inaccessible, fault injection inside COTS components will not be possible. To overcome the limitation of normal SFI when a COTS component is treated as a black box, this paper is to validate that the Perturbation of Interface Parameters (PIP) of a COTS component can simulate faults contained in the COTS component. And we will demonstrate how to use PIP techniques to assess and aid safety designs in COTS based software.

2 Component architecture and interface parameters

We consider the particular case of integrating a COTS component of uncertain integrity into a bespoke system of accepted safety integrity level (justified by the design, development and V&V processes). The objective will be to protect the bespoke system from failure due to COTS faults/uncertainties. Inevitably because of the 'black box' COTS component assumption, the diagnostic options will be restricted to those which can operate on the inter-component communications. Interaction between software components can take place via different means. Procedure calls, shared data stores and message passing can be considered as the primary means of interaction between software modules. Pipes or event broadcasting are likely to be too complex for safety-related applications and in any case also rely on the lower forms of communication. Shared memory can be used if the software components are implemented on single or multiprocessor platforms without the use of 'middleware'. In this case, some protection is often implemented by hardware; for example by restricting the addressing space of the COTS component and/or by making certain memory locations read-only. Interaction via standard procedure calls assumes that the two sub-systems are implemented on a single processor. This architecture is likely to require 'middleware' to handle communications. The 'middleware' provides a collection of procedures to access data or services and must be at the bespoke integrity level. Such a scheme allows some control of data and control flow interactions although the risk of incorrect interaction (interference) via the common physical memory still exists if data is also shared that way. Message passing is the only communication mechanism for distributed implementations without shared memory. Such an architecture has the advantage of greater separation of the sub-systems, with no risk of interference via accidental memory access. The 'middleware' again has to be at the bespoke safety integrity level. A COTS component is required to be designed to have a clear interface with its environment and the appropriate documents are needed to provide a complete description of interface parameters and their specifications. Assuming the basic cross-component linkage between procedure calls and procedure bodies is performed correctly, the interface can be simply defined as the parameters that are used to exchange information between COTS components and the bespoke system,

The conceptual arrangement and the sketch of interface parameters are shown in Fig. 2.1 below:-

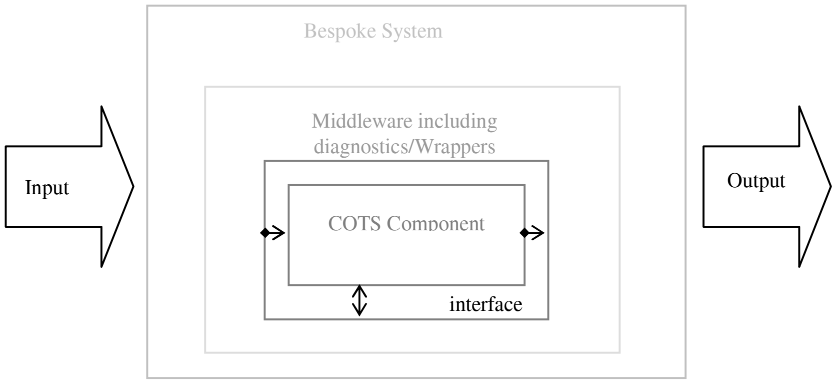


Fig 2.1 Sketch view of the connecting interface
And conceptual view of system components

We distinguish three types of interface parameters:

- the first one is to input information from the bespoke system to the COTS component, which is the single arrow pointing to the COTS component. Such interface parameters is defined as *input parameters of COTS component.*
- the second one is to output information from the COTS component to the bespoke system, which is the single arrow pointing to the bespoke system. Such interface parameters is defined as *output parameters of COTS component*
- the third one is a combined situation.

These interface parameters are the unique routes for information exchange between the bespoke system and the COTS component. *All COTS failure behaviours can only be transferred into the bespoke system by output parts of the interface parameters, and present as some anomalies of the interface parameters*. The PIP is to simulate the faults in COTS component by injecting anomalies at the interface through perturbing those output parameters.

3 Initial validation of PIP technique

Software fault injection (SFI) is used to make the system under test manifest defects in a stressed environment. SFI has been used successfully in safety-critical applications to find failure modes that would have otherwise been extremely difficult to find using standard testing techniques [BJ96]. This makes SFI an attractive technique for testing software diagnostics and in particular, to verify diagnostics function in COTS-based software. However, the physical testing of COTS components is necessarily constrained by the fact that the source code is not available and even if it was, modification could potentially negate its previous history of reliable operation. PIP could remove this problem if it can be shown to provide acceptable fault simulation/coverage.

The fault injection approach is still a new technique and necessary to overcome some practical difficulties. The promise of fault injection and initial idea of PIP as general techniques to assess systems composed of COTS components has already been recognised [VJ98], but how to use these techniques to perform practical, realistic testing is still a matter of research [PD02].

A key requirement in the development of our approach is to examine how to use perturbation of interface parameters (PIP) of a COTS component to simulate faults contained in the COTS component. The foundation of the approach is that the interface parameters are the only route for information to be exchanged between COTS components and the system, so the influence on the system of faults in a COTS component will occur via anomalous values of the interface parameters.

Our application of PIP to assess COTS software based systems begins by investigating how PIP can be used to simulate faults in a COTS component. The aim of this task is to provide evidence to support the validity of PIP but also to see if it is possible to identify a method of simulation that is easy to apply in practise.

3.1 The system for test

In this paper, a nuclear plant protection system is used to provide an example of such properties within a software system [QW91]. The process of PIP can be explained by use of a very simple COTS component 'High_Trip' module. The trip module is designed to monitor if a plant is safe by ensuing that defined high limits on certain physical values are not exceeded. Its connection with the protection system is through four parameters: 3 float variables: *Value, Max and Warn*, and 1 character variable: *high_trip*. The former four are input parameters, which will accept data from sensors on the plant about its current physical state and the boundary values for warning and tripping. The last one is the only output parameter for this module, which tells the system the current status of the monitored object i.e. it is one of a normal, warning or trip status. Therefore, any potential faults contained in this module can only influence the system through returning a false value through the interface parameter *'high_trip'*.

In this case, the number of failure modes is limited. The output parameter has 3 possible values N, W and T. Each time only one of the three values can be a correct output, then the potential failures for each output have two formats. The potential failure formats of this special module only have following cases summarised in the table 3.1:

Table 3.1 failure format of the module

No. of correct output	1	2	3
Values of the correct output	N	W	T
Failure 1	T	T	N
Failure 2	W	N	W

3.2 Comparison at interface level

The general goal in this part is to test if the perturbation of interface parameters can simulate faults inserted into COTS components and their mapping relations by monitoring the interface parameters of the COTS component. There is very little work in the literature describing the relation between the faults in interface parameters and in the COTS components

The test procedures and harness are explained in figure 3.1

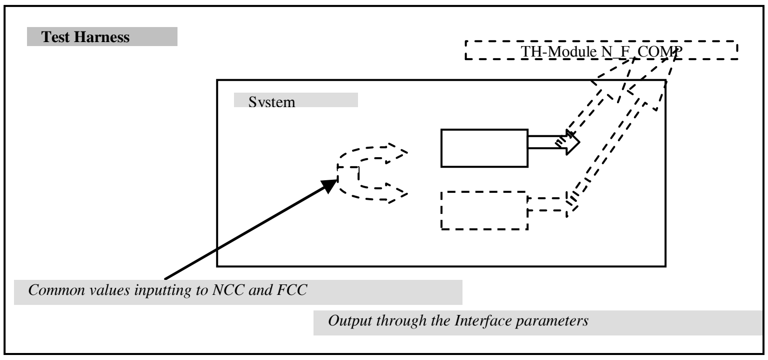

Test Harness
TH-Module N_F_COMP
System
Common values inputting to NCC and FCC
Output through the Interface parameters

Figure 3.1. Test platform for comparison of PIP and SFI in a COTS Component

For this test, the system will embed two modules corresponding to two COTS components. One module is exactly the normal COTS component itself (NCC), the other is the copy of the COTS component but with injected faults (FCC). The values output from NCC and FCC through their interface parameters are both collected by a module N_F_COMP in the test harness to compare and to record their differences. Eight COTS components were simulated and selected from the protection system for the comparison experiments. We used two here to explain the experimental results, which are named as NCC_1 and NCC_2. NCC_1 has 2 output parameters. NCC_2 has 3 output parameters.

Analysis of the results produced the following observations :

1) Any faults injected in the COTS components can be simulated by PIP, but some realistic faults may produce anomalies in several interface parameters. Thus to exactly simulate a real fault in a COTS component it may be necessary to perturb multiple interface parameters at the same time. (However, realism is not the ultimate aim. Powerful, meaningful testing may be achieved by other means. For example, although software mutations do not cover all types of possible real faults, it has been argued that there is equivalence of software mutations and realistic faults at the level of overall system evaluation [GR92]. The experiments in next section illustrate whether single-parameter perturbations constitute sufficient testing.)
2) All interface parameters offer ways to propagate faults inside the components out into the surrounding system.
3) The faults in COTS components can cause an interface parameter to be perturbed to any value. This is obvious unless we can rule out certain types of fault. Bounds on the value can only exist as a result of the imposition of defined limits of these parameters at the interface.
4) Many different internal faults will produce the same, or very similar, failure phenomena at the IPs. Thus PIP may be more efficient than normal SFI when considering the coverage of all possible faults.
5) The rates at which different interface parameters propagate the COTS faults can vary widely. (Given faults that are distributed `evenly` in the component, some parameters are affected by most faults, whilst other parameters are only affected by small subsets of faults). This can be seen by comparing figure 2.6a and 2.6b.

Figure 2.6a The distribution of faults in NCC1 and NCC2

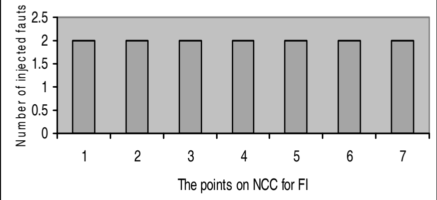
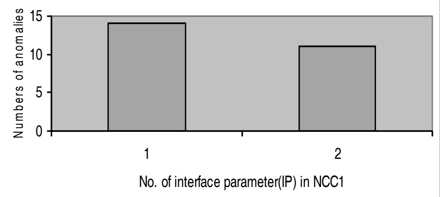


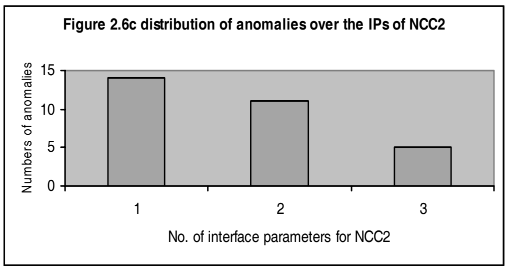
Figure 2.6b distribution of anomalies over the IPs of NCC1



6) Different components may show different operational profiles for propagating faults when similar distributions of faults in the components is assumed, which can be seen by comparing figure 2.6b and 2.6c. Figure 2.6a shows the same distribution of injected faults for both COTS component NCC_1 which has 2 interface parameters, and NCC_2 which has 3 interface parameters. We selected 7 locations on the software structures of both the NCC_1 and NCC_2, and ten faults were injected at each location. We recorded which parameters showed anomalies for each fault with

the test harness as in figure 2.5. Figure 2.6b shows the percentage of the anomalies on interface parameter 1 and parameter 2 in NCC_1. The even distribution in this case appears to mirror the even distribution of faults throughout the code (as pictured in figure 2.6a). However, this will not always be the case. Figure 2.6c shows the percentage of the anomalies over the three interface parameters of NCC_2, showing a distribution that is far from flat.

7) If it is assumed that the faults injected inside the COTS component is a set of 'realistic' faults, the evenly distributed PIP fault set in figure 2.6b simulates the realistic faults. However, in general this will not be the case; 2.6c shows that PIP faults created by perturbing each interface parameters of NCC_2 an equal number of times will not simulate the faults evenly distributed over the structure of COTS component. An intuitive observation from this result is that we can't use this naïve approach to analyse reliability figures because real failure behaviour will be fault distribution-dependent.



In conclusion, the naïve use of PIP may still be the most direct way to make a safety case for software containing COTS components. In our example, relatively simple interface parameter perturbations simulated the effects of all faults injected into the COTS component. A piece of software containing a COTS component should certainly be assessed against a specification of its required fault tolerant functions. PIP can do this conveniently, using direct testing of the implemented system's ability to recognise a wide range of COTS component failures.

3.3 Comparison at system level

If PIP simulation of COTS faults is to be used in a bottom-up (FMEA-style) analysis, it would be encouraging if it produced failure behaviours at the overall system outputs that are similar to those produced by SFI. (PIP simulations of internal COTS faults will not be exact. If they were, this experiment would not be needed because the failures at the system boundary would also be exactly the same). It is possible that in some cases PIP failure simulation, despite appearing superficially close to SFI failures at the COTS interface, produces failures that are not possible in practice. Closeness between the system level failures caused by PIP and SFI would be one way to gain evidence that this was not the case.

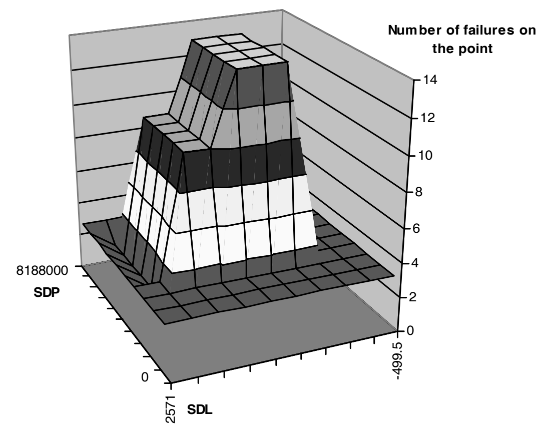
Three aspects are used to characterise a failure namely:

1) <u>Failure rate</u>: in an input space, the ratio of inputs that triggers the failure

2) Failure position: the positions in the input space of the inputs that can trigger the failures
3) Failure severity: the seriousness of the failure consequence to a system.

The test of NCC_2 is introduced as following for assessment of the equivalence of PIP and normal SFI for this component by identifying the above features. To compare failure rate and positions, we recorded footprints (that sub-region of the input space where failure occurs due to a specific fault) in the input space of the tested system by injecting two sets of faults from both PIP and normal SFI. At first, we inserted a set of faults in the COTS component with uniform distribution. The uniform distribution here means to evenly distribute the locations of injected faults over the programme structure of NCC_2. All the resulting failure points and their frequency are displayed in figure 2.7.

Figure 2.7 footprints of SFI fault set for NCC_2



This figure shows that failures occurred on a subspace of the input space. A uniform grid is used to divide the subspace and define the 100 points on the input space which is on the horizontal x-y plane in figure 2.7. The SDL (X-axis) and SDP (Y-axis) on the plane denote the physical values of Steam Drum Level and Steam Drum Pressure as two dimensions of the input space to the tested software. The limits and scale of the SDL and SDP values on the plane are decided according to the requirement specification of DARTS software. Each point in this subspace (which is a point on x-y plain) has a corresponding number (z-axis) which indicates how many faults cause system to fail at this point.

Then the set of PIP faults was injected into the system and the set of footprints obtained for the PIP fault set for NCC_2, which can be seen in figure 2.8.

There are two differences between figure 2.7 and 2.8. One concerns the number of faults causing failure on each input point. The main reason for the differences is that the two sets of SFI and PIP faults were not the same size. The second difference concerns the distribution of the failure points. When we built the fault sets, both the ones for SFI and

PIP have the uniform distribution, which is over the structures of the COTS component for SFI and over the interface parameters for PIP. If we simulate same number of faults in PIP set as in SFI one by uniformly perturbing interface parameters more times, the distribution as in the figure 2.8 will keep the same shape but become higher. But if we increase the number of faults by perturbing the interface parameters by the scale as shown in figure 2.6c, we can get footprints of the failure points for PIP with a new distribution as in figure 2.9.

Figure 2.8 footprints of PIP fault set for NCC_2

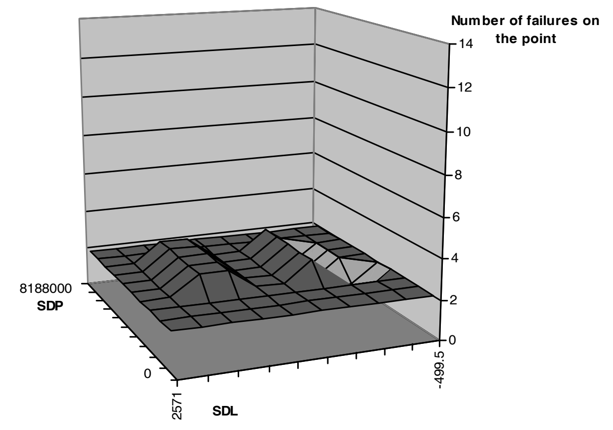
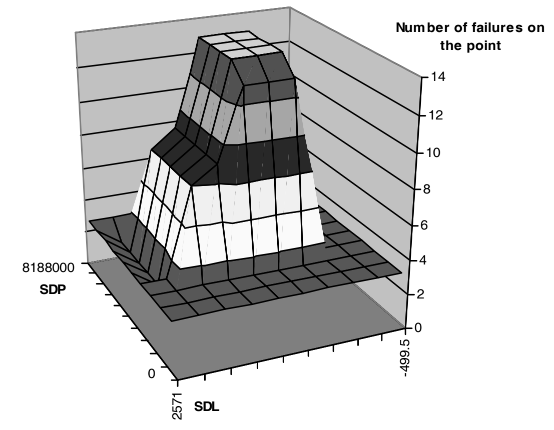


Figure 2.9 footprints of SFI fault set for NCC_2



The distributions of figures 2.9 and 2.7 are quite similar. They would be the same if we had used the same interface perturbations recorded when conducting SFI. (But of course this assumes that SFI testing has been performed.)

This result proves that PIP methods can be used to simulate a set of 'internal' COTS software faults if we consider their overall influence at the system level. Figure 2.7~2.9 shows that the difference between PIP and 'internal' faults can be reduced by putting an appropriate weight on the interface parameters when implementing PIP. This so-called weight is actually information about the possible distribution of faults in the COTS component. The obvious problems here, common to any application of SFI, are:

1) if COTS is black box we will never know figure 2.6c
2) moreover, even if we did know fig.2.6c, there is nothing to say that fig.2.6a (the 'cause' of fig.2.6c) is representative of 'realistic' faults.

Therefore, all that has been demonstrated here is that PIP can reproduce the failure behaviour of a given set of faults. In fact, in principle PIP can simulate all conceivable sets of faults. The difficulty lies in making the choice of faults/failures for PIP to simulate, in order to obtain useful assurance that the fault tolerant functions of the software provide good protection against COTS failures. At first consideration, there appear to be two possibilities:

The first is to choose a distribution of PIP-simulated failures that represents violations of the COTS component specification by a defined amount (tolerance). This implies a bottom-up approach to observing the system-level effects of COTS failures. It is consistent with the approach used in paper [NJ00a]. In this, two strategies were used:

Strategy A. *Contained assertions that check the relationship between the input and output of a behavioural block.*

Strategy B. *Contained assertions that check specific properties of a behavioural block's input or output but not a relationship between the two.*

A practical approach to the design of each assertion strategy was adopted. It was not intended that the assertion strategies in the experiments should provide an optimal diagnostic solution. No method was employed to choose a particular assertion that was believed to produce better failure coverage and produce better failure coverage. The objective of the experiments in [NJ00a, Nj00b] was to use hypothetical fault sets to make a comparative assessment of two different assertions strategies. The aim was to observe the sensitivity of the results to changes in various fault set factors.

The results from the study showed that the assertions which check properties of a behavioural block input or output but not a relationship between the two (strategy B), were particularly sensitive to variations in the fault footprint size. The relative coverage provided by strategy B was shown to decrease considerably as the footprint size decreased. The effectiveness of assertions which check a relationship between the behavioural block input and output (strategy A) was also observed to vary but to a much lesser degree.

A general problem - which became apparent from the experimental study - is that the coverage computation for any given hypothetical fault set will generally be dominated by failures caused by faults with a large footprint. This is not because a fault set will necessarily contain more faults with a large footprint, but because these faults produce more failures.

One argument is that since *real* software faults are known to have a small footprint then a legitimate fault injection approach would be to simply discard or filter out the faults with large footprints. However it is not obvious at this stage that this would be the right thing to do. These results simply revealed that the footprint size is a dominant factor in the choice of 'challenging' hypothetical fault sets.In the COTS case, assertion design would be based on the component specification alone to identify suitable input and output perturbations to represent specification violations.

The second possibility is to choose PIP-simulated failures by working back from possible system level failures (using software fault-tree analysis). This is a top-down approach, using system level failures to define the set of COTS failures of interest. A distribution of PIP-simulated failures could then be chosen in proportion to the severity of the failure consequences at the overall system level. This would mean categorising faults into classes according to the types of failure they cause at system level. In the absence of any other evidence to the contrary, a plausible approach would be to treat the classes as equally likely to occur so that the only reason that we would choose to test using unequal numbers of PIP faults for the different classes is the variation in system level consequences. Higher consequences imply a greater responsibility to guard against them, which in turn implies the need to use more faults.

There is a way to take this approach further, by combining it with statistical testing, to allow a more sophisticated analysis of fault distribution. Suppose it is possible to subdivide the input space according to the fault classes i.e. according to the consequences of the failures that can occur on the different inputs. Suppose also there is previous statistical testing on the COTS component on this input space. Then each input space class (bin) will have a certain numbers of tests in it and we can estimate a bin pfd interval$\in [0,\mu]$ with some confidence $\chi\%$, i.e. we have a reliability associated with each bin [11]. Then in combination with the consequence level for a bin, this reliability could be used to determine the intensity of PIP testing to be used on the bin. Then it might be decided no further risk assurance is required the number of tests in the bin is enough in itself. If not, we can either do more statistical tests, do some PIP tests, or both.

4 Assessment of COTS systems with safety wrappers using PIP

The experimental results in section 3 have indicated that when COTS components have to be treated as black boxes, PIP can be used instead of SFI as a fault simulation method. The experiments also suggest that there is promise in further pursuing the use of PIP for the safety assessment of COTS-based systems.

There are many types of wrappers designed to reduce the risk of system failure caused by potential faults contained in a COTS component. All wrappers are designed to have diagnostic functions to monitor the possible failures of the COTS component and prevent the system failing dangerously (crashing). A PIP fault set must be derived/simulated from the understanding provided by the various design specifications (the ones for COTS components, wrappers and the system) as depicted in figure 4.1, since this is all the information that is available.

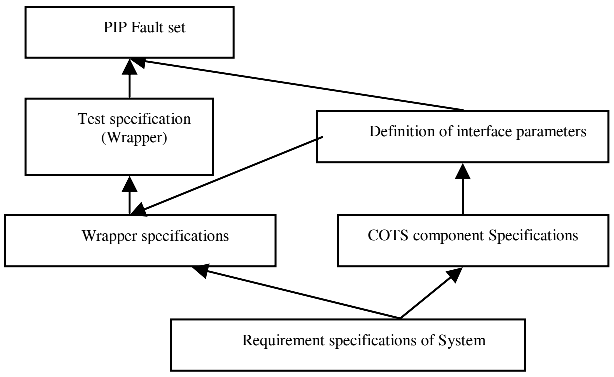

PIP Fault set
Test specification (Wrapper)
Definition of interface parameters
Wrapper specifications
COTS component Specifications
Requirement specifications of System

Figure 4.1 Derivation of the PIP fault set from specifications

The arrows in the figure denote the cause and effect relation. This report studies wrappers that are expected to intervene when a COTS component fails (on particular demands). The test specification for a wrapper is a plan of tests to confirm if all the behaviours designed to be performed by the wrapper have been implemented correctly by the corresponding coding. Figure 4.1 indicates that the derivation of a PIP fault set is based on the test specification for the wrapper and the definition of interface parameters.

The concept of testing a wrapper with real COTS failures is problematical. Demands that trigger these failures can be very rare within the normal system demand-space. If COTS failure were common in the normal usage of the system, the COTS component would not be used. In addition to being rare, system demands that fail the COTS component are unlikely to be known and may not easily be covered by random system testing. Although rare, such demands may still be critical due to the high consequence of failure, and so require additional safeguards. Hence the perceived need for COTS wrappers. Furthermore, with the possible exception of new test techniques still under development [MJ02,RG01], it is unlikely that previous COTS testing in previous applications will be sufficient to justify the COTS integrity (reliability) in a new system. Lastly, COTS are by their nature aimed at the widest possible market, and are therefore seldom developed to safety standards.

In the context of these difficulties with existing methods, PIP must be seen as a promising contender for wrapper assurance. PIP simulates COTS failure and enables direct observation of the response of the wrappers. Interface parameters should have identical definitions in a COTS component's specification and in a wrapper specification for that component. Therefore, the perturbation of the interface parameters does not need to be derived from an understanding of the internal structure of a COTS component. PIP testing can be performed without any contact with, or information on, the physical internals of COTS components. All operations on the interface parameters can be implemented on the wrapper side.

The proposed procedure to use PIP to assess and design the COTS wrappers is displayed in figure 4.2 which relates the wrapper assessment and wrapper construction. PIP can be used purely as an evaluation method to assess the quality of the wrapper or as a tool to use in the development of a wrapper.

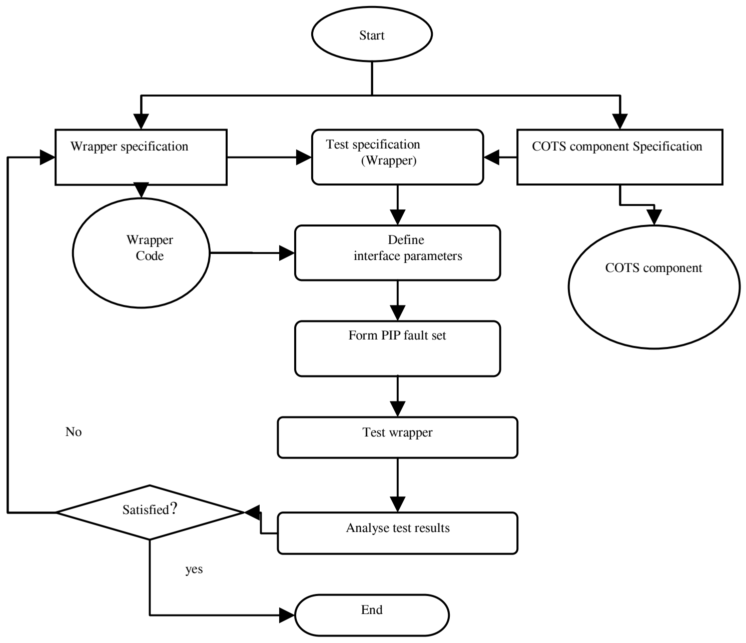


Figure 4.2 Overview of Methodology using PIP to assess wrappers

The issue remaining is that such an assessment ignores the influence of the operational profile on the actual interface parameters that will occur in reality. This information would be required if was necessary to assess the actual reliability enhancement achieved by use of a wrapper. If PIP is required to simulate a realistic frequency of a faults happening in the structure of a COTS component, the operational profile of perturbing parameters will be important. The uniform perturbation of all interface parameters does not represent the uniform injection of faults over the whole structure of the COTS component. In principle, the operational profile could be introduced by using appropriate weighting factors on each parameter to calculate the realistic response of the system. The operational profile could be determined precisely if a COTS component was a white-box however, in most situations, we could get an approximate estimation of the profile by analysing the specification of the COTS component and assuming it is implemented strictly according to the design. To do this, it would be necessary to obtain much relevant information, e.g. the operational profile of the input parameters and fault distributions. At present it is not considered feasible to use the PIP methods in this way because of the limited availability of information about the COTS component.

However, evaluating a system's fault tolerance to a COTS component can be a quantitative approach based on actual testing. For a specific safety design (e.g. a wrapper) and according to the requirement specification of a bespoke system, we can set up a complete set of failure modes of COTS components, then PIP can be used to simulate these failures of COTS component and measure the rates with which the wrappers successfully detect them. Such measurement may be used as a quantitative

5 Conclusions and further works

The PIP methods were validated on two levels. Firstly, by observing the effects *on COTS interface parameters* (IP) caused by a wide range of internal faults in COTS components, it was shown that these anomalies could be simulated by PIP in principle. Secondly, it was possible to confirm that *system level outputs*, resulting from faults inside COTS components, could be simulated by PIP in practice. (The second point does not follow trivially from the first point because, although PIP can be used to exactly reproduce any failure behaviour in the IPs, in practice the COTS failure behaviour is not known and the PIP has to use plausible assumptions about how the IPs should be manipulated. This means choosing plausible perturbations from an infinite set of possible ones. We observed that internal COTS faults produced failures that were covered by the plausible simulated perturbations i.e. at the IPs, the effects of the internal faults did not display any strange properties that were not simulated by the PIP.)

An obvious advantage of the PIP approach is that it has some independence from particular COTS components; the safety wrapper and bespoke system can be tested in the absence of a COTS component using simulated failure modes simulated in the IPs to the bespoke system. Clearly then, PIP can be used to evaluate different design strategies for safety wrappers.

This paper has shown how it is possible to take an empirical approach by fault injection techniques to analyse the structural factors influencing fault tolerant capabilities in a system with COTS components. The experiments introduced have verified the PIP can be used as an effective SFI method to simulate various faulty scenarios in a kind of stress testing. This provides a way to assess various system safety-enhancing design mechanisms. In our empirical example it appears that a complete set of interface parameter perturbations can simulate the effects of all possible faults within the component. However, based on current knowledge, it may not be appropriate to attempt to select PIP faults to try to simulate 'realistic' faults. One example of a way forward is that the PIP can be used to simulate the failures that are implied by the requirement specification of fault tolerant function for COTS-based software. Such use of PIP would directly demonstrate the implemented system's ability to achieve safety when a COTS component fails in certain ways.

COTS wrappers, in common with all forms of fault tolerance, have an intuitive worth based on normal engineering judgement. By trapping COTS failure, they offer real possibilities for reliability enhancement. The simple empirical example of PIP used in the report has demonstrated that evaluating a system's fault tolerance to a COTS component can be a quantitative approach based on actual testing. For a specific safety design (e.g. a wrapper) and according to the requirement specification of a bespoke system, we can consider a set of failure modes of a COTS component, and then PIP can be used to simulate these failures and measure the rates with which the wrappers successfully detect them. Such measurement may be used as a quantitative basis to assess the safety strategies or improve the wrapper design.

In summary, the level of reliability gain achieved with a COTS wrapper cannot be accurately quantified with current methods because of the need to simulate the fault profile of the system and components. However, there are many potential design techniques that can be used to increase safety, and PIP testing offers a plausible method of increasing confidence in the effectiveness of these techniques. Given the difficulties in applying other safety assurance techniques to COTS, PIP testing has the potential to be developed to be an important leg in safety standards for COTS-based software. Future development must include deployment on a significant example(s), and further peer review. If this proves positive, developers of COTS-based software safety systems should be encouraged to apply the technique, and licensing bodies encouraged to accept the assurance it provides. This assurance is not based on measured reliability, but is no less convincing than current safety assurance practices suggested by international software safety standards.

Acknowledgements

The work presented in this paper comprises aspects of a study (NewDISPO2) performed as part of the UK Nuclear Safety Research programme, funded and controlled by the CINIF together with elements from the SSRC Generic Research Programme funded by British Energy, Lloyd's Register, and the Health and Safety Executive

Bibliography

[BA98] Brown, Alan W, and Kurt C. Wallnau, .The Current State of CBSE,. *IEEE Software*, 15(5),37-46 (Sept/Oct 1998)

[BJ96] Bieman, J.M., Dreilinger D., Lin_L., "Using fault injection to increase software test coverage", Proceedings of the International Symposium on Software Reliability Engineering, ISSRE, 1996, pp.166-174

[GR92] Geist, R. & Offutt, J., "Estimation and enhancement of real-time software reliability through mutation analysis", IEEE Trans. Comput., Vol.41, No.5, 1992

[MJ02] J. H. R. May. Testing the reliability of component-based safety critical software. In S. Thomason, editor, *20th International System Safety Conference*, pages 214--224, PO

Box 70, Unionville, Virginia 22567-0070, August 2002. System Safety Society [NJ00a] John Napier, John H. R. May, Gordon Hughes: Empirical Assessment of Software On-Line Diagnostics Using Fault Injection. SAFECOMP 2000: 14-26

[NJ00b] J. Napier, L. Chen, J. May, and G. Hughes. Fault Simulating to validate fault-tolerance in Ada. *International Journla of Computer Systems*, 15(1):61--67, January 2000.

[NJ01] J. Napier. Assessing Diagnostics for Fault Tolerant Software. PhD thesis, Department of Computer Science, University of Bristol, August 2001

[PD02] Panel discussion, How useful is software fault injection for evaluating the security of COTS products. Proceedings of the 17th Annual Computer Security Applications Conference(ACSAC), IEEE Computer Society.2002

[PJ96] Profeta, J.,Andrianos, N., Yu, B., *Safety-Critical Systems Built with COTS*, IEEE Computer, Volume: 29 Issue: 11 , Nov. 1996

[QW91] Quirk, W.J. and Wall, D. N., *"Customer Functional Requirements for the Protection System to be used as the DARTS Example"*, DARTS consortium deliverable report DARTS-032-HAR-160190-G supplied under the HSE programme on Software Reliability, June 1991

[RG01] Richard G. Hamlet, David V. Mason, Denise M. Woit: Theory of Software Reliability Based on Components. ICSE 2001: 361-370

[VJ98] Voas J Certifying Off-The Shelf Software Components. IEE Computer, 31, pp.53-59 June, 1998