

Testing Times

November 1999

The IPL Software Products Newsletter

No 14

Testing: Not Rocket Science, *But When Will We Ever Learn?*

The value of software testing, or rather the cost of not testing, has again been thrown into high relief by a number of spacecraft failures in the last six months. This follows the embarrassingly well-publicised failure of the first Ariane 5 launch in 1996, the cause of which was tracked down to faulty software in the Inertial Reference System. It does make you wonder, 'when will we ever learn?'.

The most expensive of the recent failures has been that of the Titan-Centaur Milstar launch. This occurred in April and resulted in a Milstar satellite being placed in a useless orbit. The satellite was valued at \$800 million and the overall cost has been estimated at \$1.2 billion. These figures disregard the cost to the military of lost intelligence. The cause has been narrowed down to a software fault in the attitude control system, which had somehow escaped the rocket supplier's verification procedures. For those who like details the final blame was pinned on a constant having been entered as -0.1992746 rather than its correct value of -1.992746 (Aviation Week, August 2 1999).

The Deep Space 1 probe (May 1999) had another classic fault. A problem occurred in its 'remote agent' software which should have allowed the craft to operate with a degree of autonomy in space. Unfortunately, a software bug prevented the probe from shutting down its engines correctly. This error took the form of a software variable going out of range and causing the system to reboot. Fortunately the problem was detected and manually overridden. The software is being corrected and will ultimately be uplinked to the probe in space. It has to be said, this remedial work would have been much easier prior to launch (New Scientist, May 29 1999).

In June, a potentially disastrous collision involving the International Space Station (ISS) was only avoided by chance. The incident started when a large piece of space junk was detected to be on course to strike the ISS, and commands were sent from the ground to manoeuvre it out of harm's way. Unfortunately these commands were rejected by the space craft because the software had not been updated to take account of the ISS's extra weight after the US 'Unity' module had been docked to Russia's original FGB module. The accident was only avoided because the junk missed the craft anyway. The root cause is attributed to one of 'software management' (Aviation Week).



Picture Courtesy of NASA

International Space Station - Near Miss

Going a little further back in time (1997) there was the Mars Pathfinder mission, which was undeniably an enormous success both technically and from the public relations point of view. Its success however was nearly marred early on by a single-bit flaw in the software which led to the craft undergoing multiple resets, when it should have been returning data. Fortunately the problem was detected and solved by uplinking the necessary patch.

Not in space, and still unresolved despite the evident wishes of the authorities to treat the matter as 'closed', there are the alleged problems with the

Chinook engine controller (FADEC). This relates to the crash in 1994 of a Chinook helicopter carrying military personnel in Scotland. The initial enquiry blamed the pilots but doubts are still being voiced as to whether the true cause was a fault in the software of the FADEC. These doubts are based, in part, on the investigation into an earlier Chinook crash which implicated poor software design, which in turn was not detected because of an inadequate testing regime.

Should we be surprised by all this? Perhaps not, given the volume of software now permeating every aspect of daily life and the pressures that constantly mitigate against doing 'a proper job'. Three significant space software failures in three successive months should cause eyebrows to be raised. Due reflection must now be given to the true value of software testing. If it has not yet happened to you, this is probably less due to your natural ability to produce perfect software every time, and more due to luck! 'There but for the grace of God...' should be the thought on everybody's mind when they read these stories, and test, test, test should be the result. It's not rocket science, but if the rocket-scientists can't get it right - what hope is there for the rest of us!!

For a digest of computer-related aviation incidents refer to: www.rvs.unibielefeld.de/publications/Incidents/. For a view of (not just computer-related) aviation incidents see: www.aviationweek.com/safety/.

STOP PRESS! Mars Climate Orbiter is the latest satellite to be lost in space. Initial analysis points to a navigation failure, deriving from a failure to convert from imperial to metric units. It seems that the navigation system was expecting newton-seconds and the attitude thruster system was providing pound-seconds! (New Scientist, October 9 1999)

Inside: New Paper on OO Coverage Analysis

OO Context Coverage - Why Traditional Coverage Is Not Enough

The use of structural coverage metrics to measure the thoroughness of a test set is a well-understood technique. However, the application of the technique to object-oriented software presents new challenges.

Traditional structural coverage metrics such as statement coverage, branch coverage and condition coverage measure how well the bodies of each method have been tested. Unfortunately, these traditional metrics do not take into account the object-oriented features of the software under test – specifically the use of polymorphism is effectively ignored. Since the use of polymorphism is a major feature of any object-oriented design, the software under test cannot be considered thoroughly tested unless the polymorphic calls have been exercised.

A New Way to Measure Coverage

To address this Cantata++ introduces a new way to measure coverage – Object-Oriented Context Coverage. By using OO Context Coverage in combination with traditional coverage metrics we can ensure that the structure of the code has been fully exercised, and thus have high confidence in the quality of our test set.

Traditional Coverage Metrics

Why are traditional structural coverage metrics inadequate? Consider the following fragment of code:

```
class Base {
public:    void foo()
         { ... helper(); ... }
         void bar()
         { ... helper(); ... }
private:
         virtual void helper()
         { ... }
};
class Derived : public
Base {
private:
         virtual void helper()
         { ... }
};
void test_driver() {
Base base;
Derived derived;
base.foo(); // Case A1
derived.bar(); // Case A2
}
```

In this example, test case A1 invokes `Base::foo()` on the base object which in turn calls virtual function `helper()` on the base object, invoking

```
Base::helper().
```

In test case A2, note that `bar()` is not overridden in `Derived`, so the inherited method `Base::bar()` is invoked on the derived object, which in turn calls `helper()` on the derived object, invoking `Derived::helper()`.

Does the `test_driver()` function fully exercise the `Base` class? The `Derived` class? Assume for simplicity that all the functions contain linear code only – there are no decisions or loops at all.

Using a traditional structural coverage metric such as statement coverage as our guide we would answer yes to both questions since running `test_driver()` would achieve 100% coverage of `Derived::helper()`, `Base::foo()`, `Base::bar()`, and `Base::helper()`.

What Did We Miss?

We haven't fully tested the interactions between `Base` and `Derived`. Specifically, we have not tested the interactions between `Base::foo()` and `Derived::helper()` or between `Base::bar()` and `Base::helper()`.

Obviously, just because `Base::foo()` works with `Base::helper()` doesn't mean that it will automatically work when used with `Derived::helper()`.

For our testing to be really thorough we should exercise both `foo()` and `bar()` for both the base class and the derived class – almost as if they were overridden in the derived class.

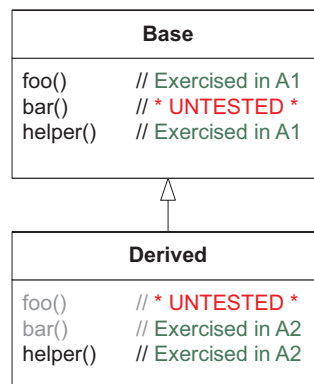


Figure 1: The inherited methods have not been fully exercised

Figure 1 shows that the inherited methods are not fully covered by the original test cases. The diagram includes the "make-believe" versions of `foo()` and `bar()` (shown in lighter text) which represent their inheritance in `Derived`.

We Need Better Tests

To achieve 100% coverage an enhanced test set is required:

```
void better_test_driver() {
Base base;
Derived derived;
base.foo(); // Case B1
base.bar(); // Case B2
derived.foo(); // Case B3
derived.bar(); // Case B4
}
```

The additional test cases help ensure that the interactions between the public methods and the private helper functions have been fully exercised, as shown in figure 2.

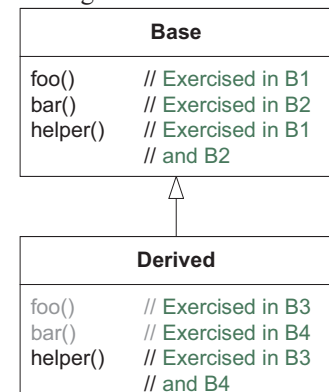


Figure 2: Better test cases give 100% OO coverage

Traditional Coverage Is Not Enough

It clear that when methods are inherited by a derived class some re-testing is necessary – for those methods which have been inherited unchanged as well as those which have been overridden. Thus coverage of an inherited method achieved by execution as if it were part of one derived class (i.e. "in the context of the derived class", by applying the method to an object of the base class) should not be taken as evidence that the method has been fully tested in the context of another derived class.

Unfortunately, traditional structural coverage metrics treat any execution of a routine – whether it is in the context of the base class or a derived class – as equivalent. `Base::bar()` is wrongly considered "100% covered" when it has been exercised in the context of class `Derived` only.

Similarly, code which has been 50% covered in the context of one derived class and 50% covered in the context of another derived class may be wrongly considered 100% covered by traditional

metrics, as shown in figure 3.

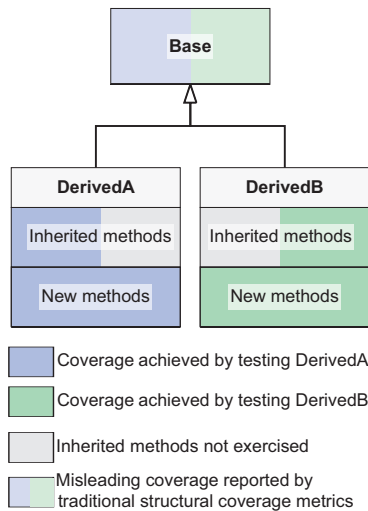


Figure 3: Only 50% coverage of inherited methods in each derived class yet Base appears fully tested using traditional metrics

This problem applies to all the traditional structural coverage metrics – none of them take into account the need for re-test to exercise the interactions between inherited base class methods and the overridden methods in each derived class.

OO Context Coverage

OO Context Coverage is not a single metric, but rather a way of extending the interpretation of (any of) the traditional structural coverage metrics to take into account the need for additional testing when methods are inherited.

OO Context Coverage provides alternative metric definitions which consider the levels of coverage achieved in the context of each class as separate measurements. The OO Context definitions regard execution of the routine in the context of the base class as separate from execution of the routine in the context of a derived class, and regard execution of the routine in the context of one derived class as separate from execution in the context of another derived class.

To achieve 100% OO Context Coverage, the code must be exercised in each appropriate context.

What Are The Contexts?

In the example above the valid contexts for the inherited method `Base::foo()` are "Base" and "Derived". The valid contexts are the same for the other inherited method, `Base::bar()`.

For the method `Base::helper()` the sole valid context is "Base", since the method is not inherited by `Derived` but is instead overridden.

For the overriding definition `Derived::helper()`, the only valid context is "Derived".

Hierarchical Integration Testing

Let us consider the level of testing required for methods which are inherited, using the example shown in figure 4.

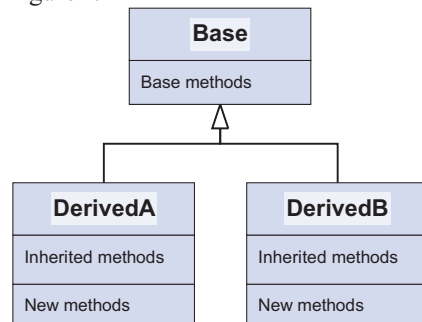


Figure 4: The software under test

The Hierarchical Integration Testing approach to unit testing was proposed by M.J.Harrold as a technique for ensuring thorough class testing. Hierarchical Integration Testing (HIT) recommends that as a first step all methods be tested fully in the context of a particular class (either the base class or, for abstract base classes, a particular derived class).

For a base class this would typically be interpreted as a coverage requirement of 100% decision coverage of each defined method in the context of the particular class. This criterion is also known as "Once-Full" coverage and is equivalent to the minimum traditional coverage requirement for thorough unit testing. This recommendation applies to all classes, so that re-definitions of a method in a derived class ("overridden" methods) are tested with the same thoroughness as the original base class definition.

Interaction Coverage

The HIT approach further recommends that any methods which are inherited by a derived class and which interact with any re-defined methods should be re-tested in the context of the derived class. The focus of this re-testing is to exercise the interactions between the inherited methods and the re-defined method(s) – primarily the calls between the methods. This recommendation could be interpreted as a coverage requirement of 100% call-pair coverage of the inherited methods in the context of each derived class.

Strict Coverage

In practice the integration test cases which exercise the interactions between methods are often spread throughout the complete test suite for the class.

Rather than separating the integration test cases out, sometimes the easiest way to ensure that interactions are thoroughly re-tested is to re-run all of

the base class test cases. This conservative approach can be enforced through the application of a stricter coverage requirement that 100% decision coverage is achieved for each base class method in the context of every derived class by which it is inherited.

Achieving OO Coverage Is Easy

During unit testing the effort required to achieve OO Context Coverage is not significantly greater than that required to achieve coverage according to traditional metrics.

Typically, no additional test cases are required. Instead, test cases already written to test the base class are used to re-test the inherited methods in the context of the derived class. The test cases form a parallel inheritance hierarchy which mirrors the inheritance structure of the software under test and enables base class test cases to be easily re-used to test derived classes (see Figure 5).

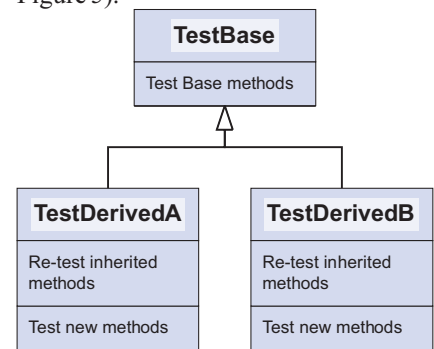


Figure 5: Test classes and software under test form parallel inheritance hierarchies

This re-use of base class test cases has the additional benefit of automatically testing the design for conformance to the Liskov Substitutability Principle (LSP). The LSP is an important object-oriented design principle which helps ensure that inheritance hierarchies are well-defined.

The only costs for this re-use are CPU time and a small initial additional effort to ensure that the test cases are structured so as to be re-usable.

Conclusion

Object-Oriented Context Coverage is a useful addition to the unit tester's tool set. It can be used in conjunction with traditional structural coverage metrics with little additional cost to ensure that interactions between methods are fully tested in each derived class.

References

For more information on OO Context Coverage, HIT, the LSP and the other advanced coverage metrics supported by Cantata++, please see the full paper at <http://www.iplbath.com/tools>.

Customer Quotes

IPL Technical Support staff continue to earn the praise of customers:

"Well, now I can compile, link, download, execute !!!!! ... Without your support, I could never do all this alone. I must say, you have the best support I have ever seen. I wonder how you can answer my emails so quickly and with the right tips. Are you all human ? :)"
DTK, Hamburg, Germany

"..very positive recommendation about AdaTEST PRIMARILY because of its terrific technical support... Our recent problem with a stub being called during elaboration is a good example... With your help, our group is coming along well with AdaTEST. I expected problems in starting to use a new tool set and that is exactly what happened. However, with your help, none of the problems have been show-stoppers."
Lockheed Martin, Ft Worth, USA.

New Customers

New customers who have shown evidence of commitment to higher software standards through purchase of IPL software testing tools over the last twelve months include (in alphabetical order): Alcatel Telecom, Ansaldo Industria, Crowcon Detection Instruments, Danfoss, DSO National Laboratories, ERA (Czech Republic), ICT Automatisering, Medical Industrial Electronics, Philips Medical Systems, Siemens Brasil, Skyforce Avionics, and Unisys.

Meet Us

We will be attending the following events:

Nov 3-4, Quality Week Europe in, Brussels.

Nov 9-11, EuroStar in Barcelona.

Nov 17, Aonix safety-critical systems seminar near Birmingham.

Dec 9, AdaUK Technology Update, Rugby

For further details of any of these events please contact Ian Gilchrist, iang@iplbath.com.

Testing Ted



Millennium Compliance

We have announced this before but here it is for the last time: Our products have been tested and, to the best of our knowledge, current versions of AdaTEST/95 and Cantata++ are millennium-compliant! If you require full details, please contact your supplier. Alternatively, please see www.iplbath.com/p22.htm.

Ports and Versions

Cantata++ is now up to version 2.0-003. This is available for Windows 9x & NT, Sun Solaris, HP-UX, Digital Unix, Solaris x86, and AIX. In most cases the product is available for both the native C++ compiler and g++. We have also formed a relationship with KAI, who provide the leading ISO Standard-compliant C++ compiler. For details see www.iplbath.com/p131.htm.

Cantata version 3.3 has been ported to Red Hat Linux and Borland 5.02 for 16-bit DOS.

AdaTEST 95 has been ported to Aonix ObjectAda on AIX.

For the latest position on availabilities please consult the IPL web site or refer to your supplier.

Support

Don't forget to keep your eye on our product support area at our web site: (www.iplbath.com/prodsupp/) for news of product upgrades and fault-fixes.

Superstitious or what?

The eagle-eyed amongst you may well have spotted that the last issue of Testing Times (April) was number 12 and that this issue is number 14. Well - it's probably just indicative of not wanting to push our luck after such a good year - there's no point in taking chances. However, if we go on like this, you can probably expect to see horoscopes (specially tailored for software testers) in future issues of Testing Times.

Training Dates

Product training courses are intended to teach new users enough about Cantata, Cantata++ and AdaTEST to become productive with the tools. The following are the currently scheduled dates for courses in Bath and Munich through to mid 2000:

Introduction to Cantata:

Bath: Thursday November 25 1999, 2000 - Tuesdays: January 25, March 21, May 16, July 18.

Munich: Tuesdays - November 9 1999, February 8 2000, April 11, June 6.

Introduction to Testing C++ with Cantata++*:

Bath: Tuesday November 23 1999, 2000 - Wednesdays: January 26, March 22, May 17, July 19.

Munich: Wednesdays - November 10 1999, February 9 2000, April 12, June 7.

Using Cantata++*:

Bath: Wednesday November 24 1999, 2000 - Thursdays: January 27, March 23, May 18, July 20.

Munich: Thursdays - November 11 1999, February 10 2000, April 13, June 8.

Introduction to AdaTEST:

Bath only: Fridays - November 26 1999, 2000 - Fridays: January 28, March 24, May 19, July 21.

* These courses are also available as a multi-media CD-ROM on rental terms.

The above courses are of one day duration and are presented at our offices. If you are interested in any of the above, AdaTEST 95 training or on-site training courses, please contact us for further details. Please also see: <http://www.iplbath.com/p72.htm> for further training course information.

Further Information

An information request sheet is enclosed with this newsletter. Also remember that most of our literature is available for immediate access at the web address given below.

IPL Software Products Group,
Eveleigh House, Grove Street, Bath,
BA1 5LR, UK

Tel: +44 (0)1225 475000

Fax: +44 (0)1225 444400

email: tools@iplbath.com

WWW: <http://www.iplbath.com/>

German Office:

Serving Germany, Austria & Switzerland

IPL Software Products Group, Zugspitzstr. 2,
D - 85591 Vaterstetten, Deutschland.

Contact: Michael Strauss

Phone: +49-8106-4034

Fax: +49-8106-4035

email: germany@iplbath.com

Distributors

North America: Quality Checked Software

Contact: Scott Thomas, cst@qcsLtd.com

Tel: +1 503 645 5610, Fax: +1 503 645 5075

Italy: Instrumatic 2000

Contact: Salvatore De Luca, sdl@instrumatic.it

Tel: +39 02 9379 9321, Fax: +39 02 9379 9195

IPL