Source Test-Tools
Arbeit zur LVA "Testen von Softwaresystemen"

Manuela Schrenk, 0055319
David Tanzer, 0055020
Jürgen Thallinger, 0055726
Andrea Tutschek, 0055735

19. November 2004

Inhaltsverzeichnis

1 Einführung

Diese Arbeit behandelt das Thema Source-Test-Tools, welches nur einen kleinen Teil der grossen Applicationtest-Thematik darstellt. Doch so klein es auch klingen mag, ist es doch ein sehr umfangreiches Thema, mit welchem sich schon viele Menschen, welche an steigender Softwarequalität interessiert sind, beschäftigt haben. Diese Arbeit fruchtete sodann nicht selten in ein gut verwendbares Test-Tool, welches es dem Programmierer ermöglichen soll, bereits zur Entwicklungszeit seinen Code auf mögliche Fehler zu prüfen. Und dies ist genau der Sinn bzw. der Job dieser Source-Test-Tools, nämlich während der Entwicklungsphase den Code aktiv zu testen und bereits im frühen Entwicklungsstadium auf gravierende Fehler zu stossen und diese zu beheben.

1.1 Der Softwareentwicklungszyklus

Dieses Thema soll hier nicht ausführlich diskutiert werden. Es soll lediglich dazu dienen, um eine Idee zu bekommen, in welchem Stadium des Softwareentwicklungszyklus Source-Test-Tools in der Praxis zum Einsatz kommen. Dies soll die Darstellung auf der nächsten Seite zum Ausdruck bringen.

Wie die obige Darstellung zeigt, liegt der Aufgabenbereich der Source-Test-Tools in der Implementierungsphase. Das heisst, während der eigentlichen Kodierarbeit wird der Programmierer von einem Tool unterstützt, welches im zeichengenau auflistet, welche Codestücke zu einem eventuellen Fehler führen können. Die Funktionsweise und Arbeit mit dem Tools selbst wird erst in späteren Kapiteln beschrieben. Hier sollen derweil nur die Grundideen dargestellt werden, mit welchen die Tools arbeiten und Fehler erkennen. Diese sind nun folgend aufgelistet:

- MISRA Rules
  MISRA steht für Motor Industry Software Reliability Associatoründ definierte mit den von ihnen aufgestellten Regeln einen quasi Industriestandard für Software in Automobilen. Das Regelwerk umfasst insgesamt 120 definierte Coding-Regeln, welche etwaige gravierende Fehler in Konstruktion usw. weitgehendst ausschliessen soll. Diese Regeln existieren aber nurfür die Programmiersprache C, was aber in Bezug auf MISRA nicht weiter tragisch ist, da in diesem Sektor ohnehin diese Sprache dominiert.

- Naming Conventions
  Jede Programmiersprache hat für sich bestimmte Namenskonventionen, an die

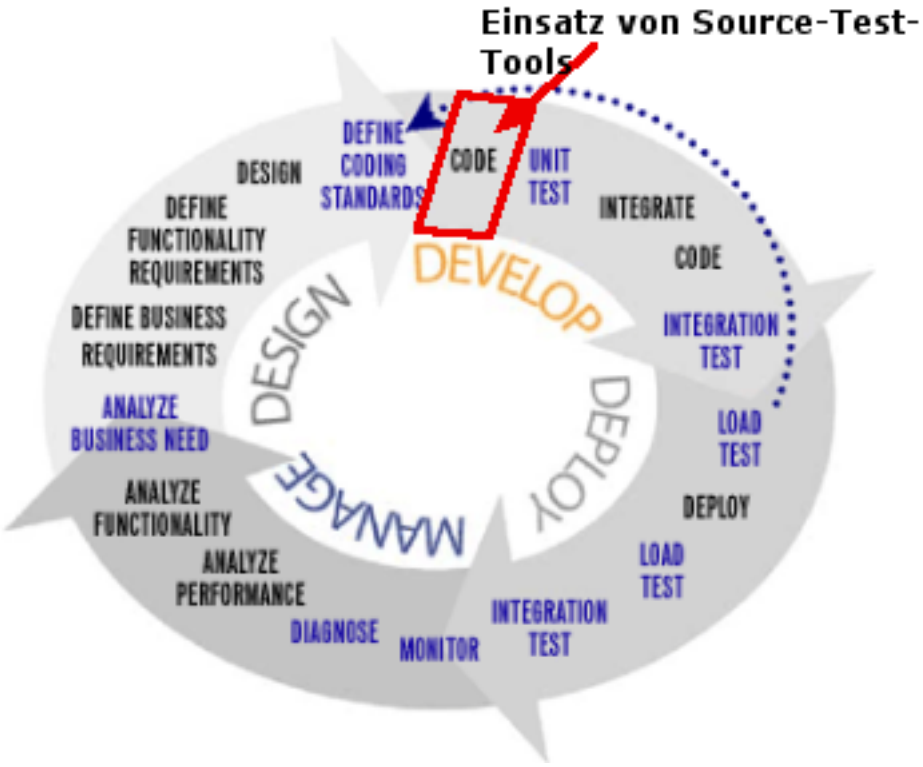

Abbildung 1: Einordnung der Source-Test-Tools in Software-Development-Cycle

sich der Programmierer halten sollte. So gibt es von SUN für JAVA bestimmte Regeln wie Variablen, Klassen und Methoden benannt werden sollen. Auch Microsoft hat solches für Visual Basic und ihr Visual C++ aufgestellt.
Je nachdem welche Programmiersprache vom Tool unterstützt wird, sind diese Namenskonventionen auch in der Fehlererkennung inkludiert. Das heisst der Sourcecode selbst wird nach falsch benannten Variablen usw. durchsucht und das Ergebnis dem Programmierer als Information mitgeteilt.

- Hungarian Notation Diese Notation bzw. Regelsammelsurium wurde von Microsoft's Chief Architect Dr. Charles Simonyi entwickelt, welchem die Idee zugrundeliegt, dass der Datentyp im Namen einer bestimmten Variable vorne angeschrieben wird. Der Grund warum diese Notation sodann üngarische Notation"genannt wird, ist folgender: Aufgrund der Präfixes der Variablen läßt diese Schreibweise auf eine fremdländische Implementierung schließen. Und da der Entwickler selbst, also Simonyi, aus Ungarn stammt, wurde der Notation dieser Name verliehen.

- 32to64 bit Migration-Rules
- teilweise auch: TickIt, IEEE 1012

1.2 Verwendung von Source-Test-Tools

Source Test Tools sind in etlichen Varianten verfügbar. Einige lassen sich direkt in eine IDE einbinden, andere sind Standalone-Programme und müssen extra ausgeführt werden. Falls ein Tool nun integrierbar ist, so bedient man es aus der IDE heraus über eine eigene Toolbar. Grundsätzlich sind die Tools, welche im Folgenden detailiert dargestellt werden, allesamt einfach zu bedienen. Der Unterschied liegt nur an den Fehlerkriterien, nach welchen sie suchen. Die am meisten verwendeten Kriterien wurden obig bereits aufgelistet. Insgesamt klickt man bei den meisten Tools schlichtweg nur einen Button und der Parser des Source Test Tools checkt den Code auf auffällige Stellen. Diese werden sodann in verschiedene "Warning-Levels"gegliedert und dem User in einem eigenen Fenster mit Zeilennummer der auftretenden Klasse versehen aufgelistet. So gibt es folgende "Warning-Levels":

- Information
  Dieses Level ist das niedrigste, es zeigt lediglich an, dass an dieser Codestelle eine eher unbedeutende Regel verletzt wurde, was aber keine weiteren Auswirkungen auf andere Codestücke hat.

- low warning
  Dieses Level zeigt bereits eine Codestück an, welches in seltenen Fällen zu Bugs im weiteren Programmierverlauf führen könnte.

- moderate warning
  Dieses Level warnt den Programmierer, dass einige leichtgewichtige Regeln verletzt wurden und bereits andere Codestücke damit fehlerhaft in der Ausführung laufen würden.

- severe warning
  Dieses Level zeigt einen sicheren Designfehler an einem bestimmten Codestück an.

- fatal warning
  Dieses Level zeigt einen sicheren Designfehler an einem bestimmten Codestück an, welches auch Auswirkungen auf einen oder mehrere damit verbundene Codestücke hat. Diesem Warning-Level wird mit Sicherheit ein schwerer Fehler zugrunde liegen.

Mit diesen Warning-Levels kann der Programmierer genau auch auf versteckte Programmier- und Designfehler hingewiesen werden. Natürlich wird nicht nur das Warning-Level allein ausgegeben, sondern auch detaillierte Angaben, welche Regel genau verletzt wurde durch das auffällige Codestück.
Somit sind Source-Test-Tools wichtig im Implementierungsprozess, da sie bereits hier Designfehler, welche sich in der Programmierarbeit eingeschlichen haben können, schnell aufdecken.

2 Beispiele für Source-Test-Tools: BullseyeCoverage

2.1 Daten

Hersteller URL	Bullseye Testing Technology http://www.bullseye.com/
Hauptfunctionalität Programmiersprachen Platforms	Code Coverage Analyse C / C++ Windows Server 2003, XP, 2000, NT, ME, 9x, CE, .NET; Linux x86, x86-64, Itanium; Solaris; HP-UX; Mac OS X/Darwin; IBM AIX; Tru64 UNIX; FreeBSD; SGI IRIX; NetBSD; Einige Embedded Systeme
Weitere Features	System-level und Kernel mode; Integriert in Microsoft Visual Studio; Run-time source code included

2.2 Arten von Code Coverage

Mehr Informationen zu diesem Theman findet man unter:
http://www.bullseye.com/coverage.html und http://www.bullseye.com/lineCoverage.html

2.2.1 Statement Coverage

Hier wird überprüft, ob jedes ausführbare statement auch wirklich getestet wird. Dieser Test ist auch als “Line Coverage” , “Segment Coverage” oder “basic block coverage” bekannt. Der Vorteil von dieser Methode ist, dass man sie im Objektcode des Programms durchführen kann und nicht unbedingt den Source Code analysieren muss.

Das Hauptproblem mit dieser Methode ist dass sie Kontrollstrukturen und logische Operationen nicht adequat berücksichtigt. Bei einfachen If - Statements (ohne else - Teil) kann man mit der Bedingung “true” bereits volle Anweisungsabdeckung erreichen; den Fall dass die Bedingung “false” ist müsste gar nicht getestet werden. Ein ähnliches Problem ergibt sich bei Kurzschlussauswertung von logischen Operationen. Do-While Schleifen werden **mindestens** einmal betreten, die Bedingung müsste also laut Anweisungsabdeckung auch nicht getestet werden.

2.2.2 Decision Coverage

Diese Methode testet, ob boolsche Ausdrücke in Kontrollstruktuen (wie `if` oder `while`) auch wirklich mit beiden Wahrheitswerten `true` und `false` getestet wurden. Diese Methode ist auch bekannt als "branch coverage" oder "all edges coverage".

Ein gesamter boolscher Ausdruck wird als ein Prädikat betrachtet, egal welche Operatoren darin vorkommen. Ausserdem müssen die cases von `switch` - statements, exception handler und interrupt handler getestet werden. Der Vorteil dieser Methode ist, dass sie einfach ist und die meisten Nachteile der Statement Coverage vermeidet. Auch bei dieser Methode gibt es Probleme mit der Kurzschlussauswertung von logischen Operatoren.

2.2.3 Condition/Decision Coverage

Diese Methode verbindet die Decition Coverage noch mit der sogenannten "Condition Coverage". Dabei wird das Ergebnis jedes boolschen Ausdrucks (durch logisches und und logisches oder getrennt) berücksichtigt. Condition Coverage ist zwar sehr ähnlich zur Decision Coverage, diese Methode alleine garantiert allerdings noch nicht volle Decision Coverage, deshalb werden beide Methoden kombiniert.

2.2.4 Function Coverage

Hier wird nur überprüft, ob jede Funktion des Systems mindestens ein mal aufgerufen wurde. Es geht also nur darum, zumindest jeden Teil des Systems "ein bisschen" abzudecken. Mit dieser Methode können schon gröbere Unzulänglichkeiten der Test Suit schnell gefunden werden.

2.3 Die Messung von Bullseye Coverage

BullseyeCoverage verwendet nur 2 Messungen: Function Coverage als groben Überblick und Condition/Decision Coverage für das detaillierte Testen. Das soll gewährleisten, dass der Tester nicht zu viele Daten bekommt, aus denen er die Wichtigen erst herausfiltern muss.

BullseyeCoverage instrumentiert den Source Code, um diese Messungen durchzuführen. Dadurch kann es auch für eine breite Palette an Compilern und Platformen eingesetzt werden. Laut Hersteller steigt die Ausführungszeit um den Faktor 1.2, die Codegrösse um den Faktor 1.4 und die Compilezeit mit Microsoft C++ um den Faktor 1.5, mit anderen Compilern um den Faktor 3-8.

2.4 Evaluierungs-Version

Eine Evaluierungsversion kann man auf der Homepage des Herstellers beantragen. Die Software steht zum Download bereit, die Evaluierungslizenz muss man beantragen. Man füllt ein Webformular aus, die Daten werden von einem Sachbearbeiter geprüft. Ich haben den Lizenzkey für die Evaluierungsversion bereits ca. 2 Stunden nach abschicken meiner Daten per email zugesandt bekommen.

3 Beispiele für Source-Test-Tools: Insure++

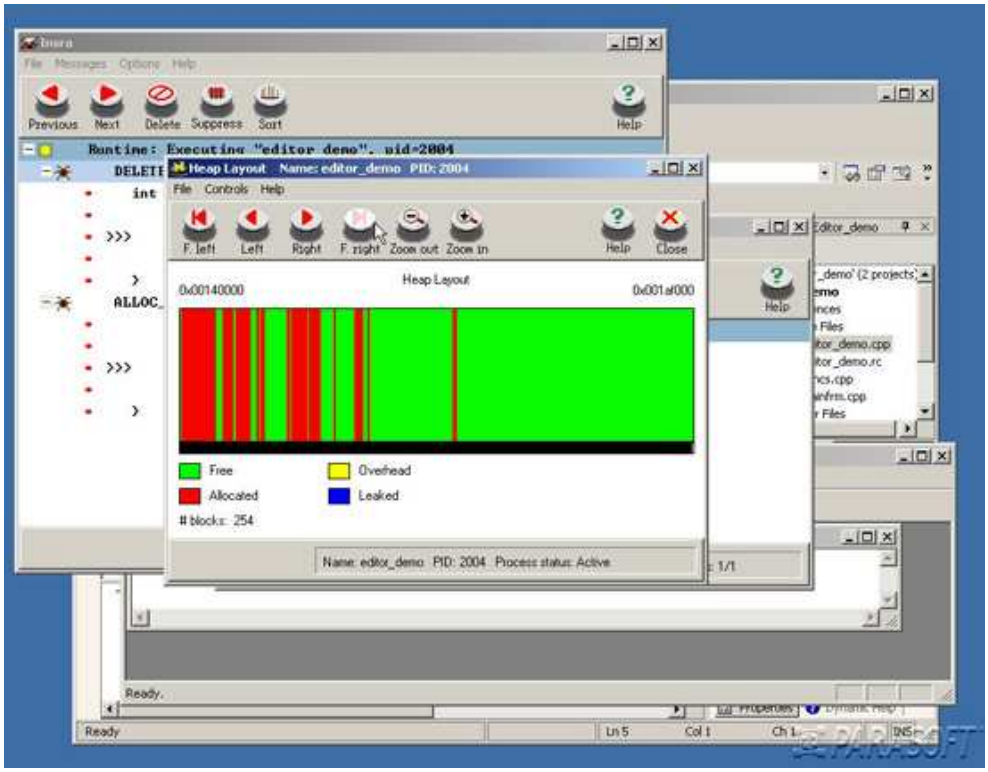

Abbildung 2: Insure++ Screenshot

3.1 Datenauflistung

Hersteller URL	Parasoft http://www.parasoft.com/jsp/smallbusiness/tool_description.jsp?product=Insure
Hauptfunctionalität Programmierspracen Platforms	Finden von Speicherproblemen C/C++ Linux: x86, AMD64, PPC; Solaris 7, 8, 9; IBM AIX; HP-UX; Windows NT, 2000, XP
Weitere Features	Instrumentiert auf Source Code - Ebene; Erzeugt Datenbank von Programmcodeelementen; Mutation Testing

3.2 Daten

Speicherprobleme in C/C++ (memory leaks, memory corruption, nicht initialisierte Variablen) sind beim Testen oft schwer zu finden. Sie treten nur nach sehr langer Zeit, in verbindung mit bestimmten Hard- und Softwarekonfigurationen oder unter anderen schwer zu kontrollierenden Zuständen auf.

Parasoft Insure++ hilft beim Finden der folgenden und weiterer Arten von Fehlern:

- Speicherkorruption
- Momory leaks
- Allocation errors
- Fehlerhafte Variableninitialisierung
- Pointerfehler

Um das zu Testen werden "Source Code Instrumentation" und "Runtime Pointer Tracking" eingesetzt. Ausserdem wird "Mutation Testing" eingesetzt. Insure++ erzeugt Mutanten des codes, die die gleiche Funktion besitzen sollten und führt diese aus. Wenn einer dieser Mutanten ein anderes Ergebnis liefert als das Original dann bedeutet das, dass die Funktionalität des Originalprogramms wahrscheinlich auf impliziten Annahmen beruht, die während der Ausführung vielleicht nicht immer gegeben sind.

Laut einem Artikel, der im Juli 1998 im Linux Journal (http://www.linuxjournal.com) erschienen ist erstellt Insure++ sehr detaillierte Reports und ist einfach zu verwenden. Anstelle des eigentlichen Compilers verwendet man `insure`. Dieser Compiler erzeugt den instrumentierten source code, compiliert ihn mit dem eigentlichen Compiler und löscht ihn wieder. Um möglichst detaillierte Reports zu bekommen sollten auch alle Libraries, die das eigene Programm verwendet, mit `insure` kompiliert sein (soweit das möglich und notwendig ist).

3.3 Evaluierungs-Version

Es Evaluierungsversion ist erhältlich. Dazu muss man sich erst auf der Homepage des Herstellers anmelden. Danach bekommt man per email eine Bestätigung, dass man als Kunde registriert wurde, und einen Link, von dem man sich innerhalb von 24 Stunden die Software herunterladen kann.

Am Tag nach meiner Registrierung bei Parasoft bekam ich eine Email von Parasoft Deutschland, dass mich die zuständige Account Managerin nicht telefonisch erreichen konnte. Darin stand weiters, dass sie erst eine Schlüssel generieren muss, bevor ich das Produkt testen kann und dass sie mit mir erst die Evaluierung telefonisch besprechen will.

4 Beispiele für Source-Test-Tools: Polyspace Technologies: POLYSPACE for C++

Polyspace Technologies hat für die Programmiersprachen C, C++ und ADA eine Entwicklungsumgebung entworfen, die Run-Time-Fehler zur Compilezeit aufdecken soll. Das umfasst eine statische Analyse ohne explizites Testen oder der Durchführung des zu testenden Programms.
Bei einer statischen Analyse von dynamischen Elementen wird jede mögliche Ausführungsvariante einer gegebenen Softwareanwendung durchgespielt und somit Fehler, vor allem Run-Time-Fehler aufgedeckt.
Die Grundidee besteht darin jede mögliche Ausprägung der verwendeten Variablen zu finden und für jeden Punkt in der Applikation zu berechnen.
Beispiel:

```
int tab[100]
int *p= tab;
int i;
for (i=0, i<l; i++, p++)
      p* =0;
*p=5;
```

Polyspace umfasst die folgenden Standards, MISRA, CMM, IEEE1012, ... , und ist für die Betriebssysteme Windows, Linux und Solaris verfügbar.
Von PolySpace abgedeckte Fehlerquellen:

- Lesefehler bei nicht initialisierten Daten
- Null-Pointer und Out-of-Bounds exceptions
- Fehlerhafte arithmetische Operationen (z.B. Division durch Null)
- Overflow, Unterflow von arithmetischen Operatoren (für int und float)
- „gefährliche“ Typcasts
- Zugriffsverletzungen zwischen Threads
- Ungültige dynamische Casts
- Unautorisierte Exceptions

- Aufrufe von abstrakten Methoden
- Arrays mit negativer Größe
- Rückgabewert Null
- Nullpointer auf Memberobjekt
- Falsche Rückgeabewerte
- Nicht terminierende Funktionsaufrufe und Schleifen
- Dead code (nicht erreichbarer Code)

4.1 Anwendung und Aufbau des Programms

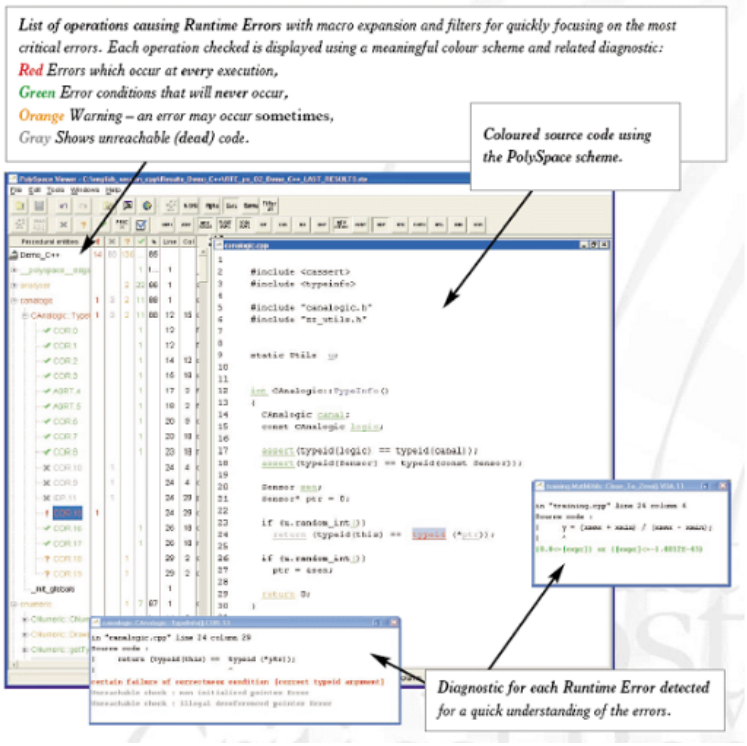

Abbildung 3: Übersicht: Polyspace

Dieser Screenshot zeigt das Interface des Polyspace Tools und einen Einblick in die Interaktion der einzeln Darstellungsformen. In folgenden werden die einzelnen Komponenten separat vorgestellt:

- Ein kompletter und interaktiver Funktionen-/Methoden-Baum:
  Polyspace stellt ein übersichtliches Baumverzeichnis, zur generellen Übersicht der Struktur des Programms, der Fehlererkennung und Fehlerstatistik zur Verfügung. Wie in obigen Bild ersichtlich liefert das Programm eine übersichtliche Auswertung der aufgetretenen Fehler, unterstützt durch ein Farbschema zur Fehlerkategorisierung:

  - ROT: Fehler tritt bei jeder Ausführung des Programms auf
  - GRÜN: Fehlerbedingung kann nie auftreten.
  - ORANGE: Warning – In diesem Codestück kann bei manchen Durchläufen ein Fehler auftreten.
  - GRAU: dead code – Dieser Code ist nicht erreichbar.

  Diese statistische Auswertung ist direkt in ein Excel-Sheet exportierbar und dadurch übersichtlich archivierbar. Wird ein angetretener Fehler im Verzeichnis angeklickt, wechselt auch der Darstellungsraum des Sourcecodes zum betroffenen Codestück.

- Codefenster:
  Das gerade angesprochene Farbschema ist natürlich auch hier in Verwendung und signalisiert die aufgetreten Fehler und deren Intensität. Eine weitere Hilfe bei der Aufdeckung der Run-Time-Fehler umfassen die kleinen weiterführenden Hilfefenster, die bei Anklicken eines jeden Fehlers im Code erscheinen.

- Global Data Dictionary:
  Umfasst eine Auflistung der Datenzugriffe auf globale und gemeinsam genutzte Daten (zwischen Threads, Critical Sections, ... oder Pointerverweise). Auch diese statistische Ausarbeitung ist einfach in ein Excelformat zu exportieren.
  Zur besseren Übersicht werden die gemeinsam genutzten Daten, die Methoden und Funktionen von denen sie benutzt werden und deren Beziehungen graphisch veranschaulicht.

- Zugriffsgraph für jedes Shared-Data-Objekt:

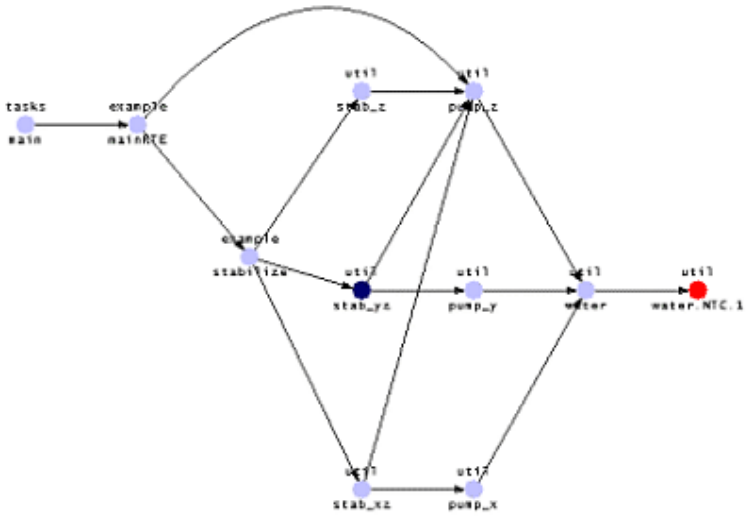

Abbildung 4: Polyspace-Graph

4.2 PolySpace-Produkte im Feld

500 Firmen aus Nordamerika, Europa, Asien und Australien aus den Bereichen Industrie, Automobilentwicklung, Verteidigung, Luftfahrttechnik, Transport und auch Medizin haben eine Version von PolySpace für die Weiterentwicklung ihrer Projekte im Einsatz.

5 Beispiele für Source-Test-Tools: TICS - TIOBE Coding Standards Framework

Dieses Tool der Firma TIOBE-Software wird als Plug-in in Entwicklungsumgebungen, wie Visual Studio 6.0 oder .net integriert oder von der Kommandozeile gestartet und kann dann Sprachen wie C, C++ oder JAVA analysieren, bzw. durch kleine Erweiterungen auch für andere Sprachen verwendet werden, und schwerwiegende Pun-Time-Fehlern schon bei der Kompillierung aufdecken (unterstützte Systeme: Windows NT/2000/XP, bald auch Linux).

Vorteile:

- es können auch Mischungen aus verschiedenen Sprachen und Dateitypen überprüft

werden

- Überprüfung von komplexen kontext-sensitiven Regeln
- Regeln können vor und nach dem Pre-Processing analysiert werden
- gute Performance in der Codeanalyse
- Verletzungen werden geordnet, Duplikate ausgefiltert
- die Ausgabe kann individuell angepasst werden
- mit dem sogenannten Security Level erfolgt eine Unterscheidung in wichtigen und weniger wichtigen Regeln
- es gibt die Möglichkeit ganze Module aus der Analyse auszuklammern (z.B bei externen Softwarepaketen)

5.1 Verwendung von TICS

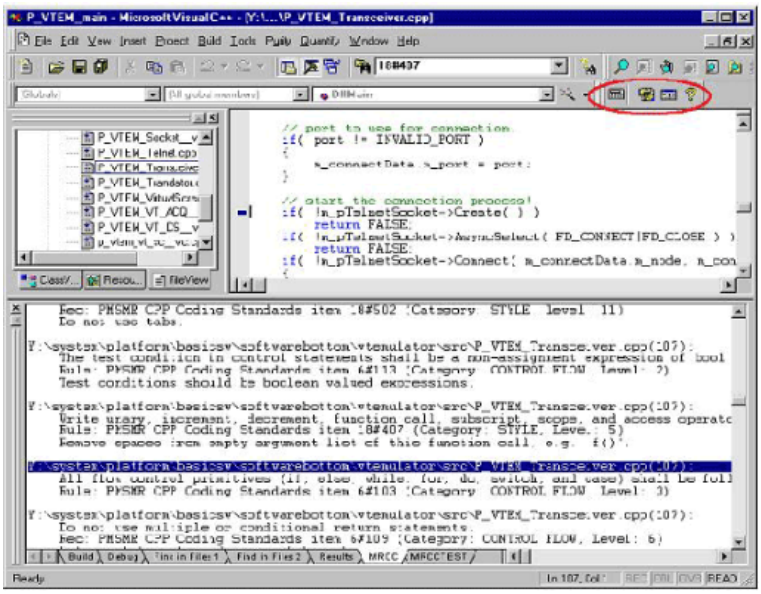

Abbildung 5: Übersicht: TICS

Obiges Bild zeigt die Einbettung von TICS im Visual Studio. Über die eingezeichneten Button können die Regeldefinitionen für jeden Durchlauf geändert werden (siehe Abschnitt Regelkonfiguration).
Nach jedem Testdurchlauf wird eine Auflistung der Fehler, ein Übersicht der Häufigkeit der einzelnen Regelverletzungen und eine Warnung wenn diese einen festgesetzten Schwellwert übersteigen, aber auch die Änderungen des Files seit dem letzten Durchlaufes, ausgegeben. Die Regelverletzungen werden im SDE angezeigt und verweisen beim Anklicken direkt auf die Fehlerstelle im Code.

5.2 Regelkonfiguration

Hier wird das Set der zu überprüfen Regeln festgelegt, es ist möglich Regeln zu aktivieren aber auch wieder aus dem aktuellen Set zu entfernen. Hier wird auch die sogenannte Security Schwelle festgelegt, ein Wert der zur Unterscheidung von schweren und nicht so schwerwiegenden Regeln eingestellt werden kann.

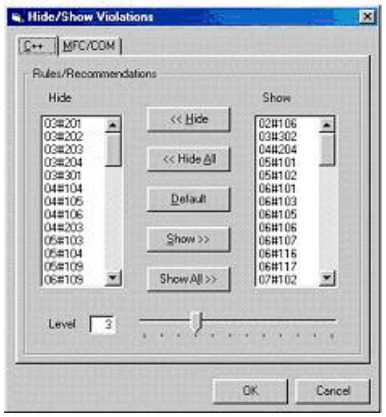

Abbildung 6: Regelkonfiguration

5.3 Qualitätsdatenbank

Ein weiters Feature von TICS stellt die Qualitätsdatenabnk dar. Um einen Überblick über die Entwicklungen eines zu testenden Tools zu bekommen, legt TICS alle Verleztungen der Codeanalyse in eine Qualitätsdatenbank ab und stellt die Entwicklung in einem übersichtlichen, graphischen Rahmen dar.

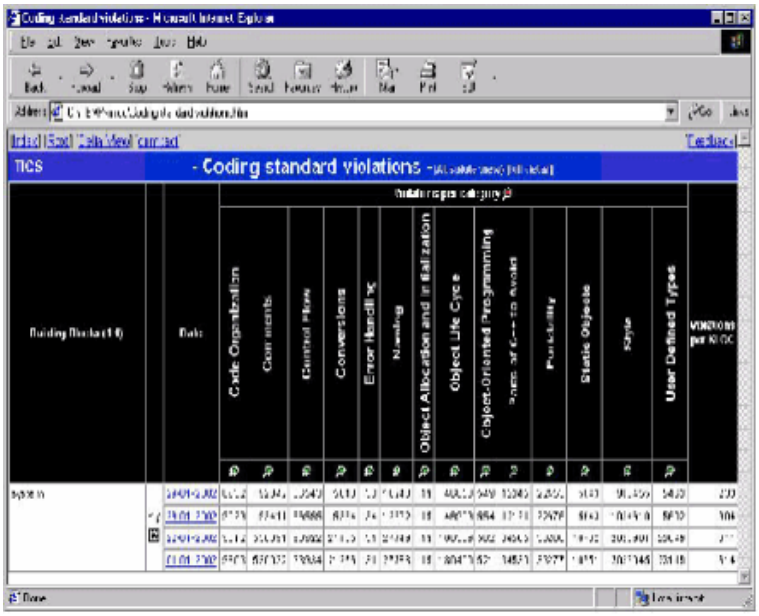

Abbildung 7: Automatische Codeverschönerung

5.4 Automatische Codeverschönerung

Hierbei handelt es sich um eine Funktion, die kleine Fehler im Bezug auf vorgegebene Kodierungsstandards ausbessert und somit für ein übersichtlicheres Codebild sorgt und damit zur Fehlererkennung beträgt.

5.5 Anwendungsbeispiele im Feld

TICS ist das Software-Framework des Lösungskonzeptes SYNtacTICS der Firmen TICS Software und SYNSPACE. Dieses Paket der Qualitätssicherung bietet Kunden ein vollständiges Konzept zur Qualitätsverbesserung ihrer Programm.
Unter anderem ist TICS bei der Philips Medical System Groupin den Bereichen Kernspintomographie, Kardiovasculär und der Common Component Group, in Verwendung.

6 Gegenüberstellung von Source-Test-Tools

In den vorhergegangenen Kapiteln wurden von uns einige Source Test Tools vorgestellt, die wir jetzt kritisch miteienander vergleichen werden. Verglichen werden im speziellen folgende Kriterien:

- Kosten des Produktes
- Unterstützte Plattformen
- Anwendung

.

6.1 Kosten

6.1.1 Bullseye

Eine neue Lizenz erhält man für dieses Produkt ab 800 Dollar, ein Jahr Versions Update inkludiert. Man kann sich allerdings auch bis zu drei Jahre die aktuellste Version sichern, wobei man allerdings für jedes Jahr 100 Dollar zusätzlich zahlen muss.

6.1.2 Insure++

Leider erfährt man hier nichts über den Preis, solange man sich nicht auf der Parasoft Homepage als Kunde registriert hat. Für dies Registrierung ist es nötig, seinen Namen, Firma, Adresse, Telfeonnummer etc. anzugeben, allerdings kann man sich dann gratis eine Demo des Produktes herunterladen.

6.1.3 Polyspace

Ärgerlicherweise erfährt man auch auf dieser Seite nichts über die Kosten des Produktes, erst nach persönlicher Kontaktaufnahme via E-Mail kann man sich über Preise informieren lassen.

6.1.4 SYNtacTIC

Auch hier ist keine Preisangabe auf der Hompage der Hersteller zu finden.

Im Vergleich kann man Aufgrund fehlender Angaben nur feststellen, dass es erstens

so gut wie keine freien Tools gibt und zweitens die Preise für Lizenzen mit Sicherheit im oberen Bereich liegen (wir erinnern: 800 Dollar für Bullseye).

6.2 Plattformen

6.2.1 Bullseye

Bullseye unterstützt ein grosse Auswahl an Plattformen. Hier ein kurze Auflistung: Windows, Linux x86, Linux x86-64, Linux ARM, Linux Itanium,Solaris SPARC Solaris x86, HP-UX PA, HP-UX Itanium, Mac OS X, AIX, FreeBSD, IRIX, NetBSD QNX 6, Tru64 UNIX.
Anmerkung: Beim Erwerb einer Lizenz muss sich der Kunde für eine Plattform entscheiden.

6.2.2 Insure++

Unterstützt werden diverse Lunix und Windows Distributionen, genauso wie Solaris 7, 8, 9, IBM AIX und HP-UX 11.xx.

6.2.3 Polyspace

Im Gegensatz zu den anderen vorgestellten Tools gibt findet man auf der Polyspace Homepage nur sehr schwammige Informationen die Plattformen betreffend. Unterstützt werden Windows, Solaris und Linux.

6.2.4 SYNtacTIC

Unterstützt lediglich Windows NT/2000/XP und seit kurzem auch Linux.

Bullseye und Insure unterstützen im Vergleich die meisten Plattformen. Gleichzeitig definieren die Hersteller auch sehr genau, welche Distributionen im Detail unterstützt werden. Polyspace und Sysntactic fallen in diesem Punkt deutlich ab, sowohl in der Zahl der Plattformen als auch in der genaueren Definition, auf welchem System diese Produkte tatsächlich laufen.

6.3 Sprachen

6.3.1 Bullseye

Hier beschränkt sich Bullseye lediglich auf C und C++, eine im Vergleich eher geringe Ausbeute.

6.3.2 Insure++

Wie der Name bereits erahnen lässt, handelt es sich hier ebenfalls um ein Source Test Tool für C++ und C. Andere Sprachen werden, wie bei Bullseye, nicht unterstützt.

6.3.3 Polyspace

Dieses Tool kann ausser C und C++ auch noch Ada verarbeiten.

6.3.4 SYNtacTIC

Dieses Tool kann laut Hersteller C, C++, C sharp, Java, MFC/COM und IDL analysieren. Angeblich kann man das Tool auf Wunsch des Kunden auch sehr einfach erweiteren, um jede beliebige Sprache parsen zu können.

Im Vergleich hat SYNtacTIC in dieser Sparte eindeutig die Nase vorn. Während die drei anderen Tools lediglich C, C++ und Ada analysieren können, lässt sich SYNtacTIC angeblich sogar auf jede beliebige Programiersprache erweitern.

6.4 Anwendung

Um es gleich vorweg zu nehmen: Um die einfache Anwendbarkeit eines Tools wirklich objektiv bezüglich Einfachheit und hohe Funltionalität vergleichen zu können, müsste man die Tools tatsächlich testen, was uns leider nicht möglich war, da keines der vorgestellten Werkzeuge frei erhältlich ist. Wir werden also lediglich die von den Herstellern selbst genannten Eigenschaften wiedergeben, ohne allerdings kritische Aussagen darüber machen zu können

6.4.1 Bullseye

Abgesehen von einem Kommentar Eric Tannenbaums, dass dieses Tool sehr einfach anzuwenden ist, findet man keinerlei Hinweise auf die Art der Anwendung, weder Screenshots noch konkrete Aussagen oder gar Tutorials. Allgemeine Features sind hingegen beschrieben: So verfügt Bullseye über:

- Function Coverage
- Condition/Decision Coverage
- Source Code Instrumentation
- Automatic Saving

Deatils zu diesen Features lesen sie bitte im Kapitel über Bullseye.

6.4.2 Insure++

Die Hersteller von Insure++ bieten auf ihrer Internetseite ein Flashpräsentation ihres Tools, die einen sehr guten <berblick über die Anwendung und Funktionalität dieses Tools gibt. Hier die wesentlichen Features im <berblick:

- Mutation Testing
- Source Code Instrumentation
- Multiple Error Detection Modes: Ist vergleichbar mit Function Coverage und Condition/Decision Coverage von Bullseye.

6.4.3 Polyspace

Die Hersteller von Polyspace halten sich sehr zurück mit Informationen die Funktionalität oder die Anwendung betreffend. Was Polyspace aber im Gegensatz zu den anderen Tools betont ist, dass zum Testen des Codes keine Testprogramme, kein Eingreifen in den Source Code kein Ausführen des Programmes notwendig sind.

6.4.4 SYNtacTIC

In SYNtacTIC wird die Einfachheit der Anwendung des Programmes besonders hervorgehoben. Dieses Tool lässt sich in SDEs wie Visual Studio als Plugin installieren und testet den Code einfach per Mausclick. Ebenfalls interessant an diesem Tool ist, dass es Mischunge aus verschiedenen Sprachen testen kann (spielt im besonderen auf die .Net Techologie an).

7 Fazit

Wie nicht anders zu erwarten war, unterscheiden sich die Tools alle ziemlich stark, wenn auch nicht unberdingt in der Sprache. Auffällig ist eine Neigung hin zu C und C++, die von allen hier vorgestellten Tools unterstützt werden. Ebenfalls auffällig war die offensichtliche Scheu der Hersteller, konkrete Preise füt ihre Tools anzugeben, eine Ausnahme bildet Bullseye. Freie Tools sind praktisch nicht vorhanden und Demos sind, falls es sie überhaupt gibt, nur nach vorheriger Registrierung beim jeweiligen Anbieter verfügbar. Kundensupport als solcher wird explizit nur von Bullseye angeboten und zwar in Form von regelmässigen Updates des Produktes.
Generell werden solche Testtools für möglichst viele verschiedene Plattformen geschrieben, abgesehen von wenigen Windows Spezialisten wie z.B.: SYNtacTIC, das erst seit kurzem auch Linux unterstützt. Dafür wirbt SYNtacTIC mit extrem einfacher Bedienung.
Eine eindeutige Einteilung der Tools in gut/schlecht fällt sehr schwer, jedes Tool hat gewisse Vor- und Nachteile, im Prinzip hängt es vom Entwickler ab, worauf er Wert legt bzw. was er braucht.

8 Quellenverzeichnis

- Graphik des Softwareentwicklungszyklus:

  `http://www.computerworld.com/printthis/2003/0,4814,83735,00.html`

- Parasoft's Warning-Levels:

  `http://www.parasoft.com/jsp/products/home.jsp?\\`
  `product=Wizard&itemId=62`

- Bullseye: http://www.bullseye.com/
  - Coverage Analysis:

    `http://www.bullseye.com/coverage.html`
  - Line Coverage:

    `http://www.bullseye.com/lineCoverage.html`
  - Product Info: http://www.bullseye.com/productInfo.html

- Insure++:

  `http://www.parasoft.com/jsp/smallbusiness/tool_description.jsp?\\`
  `product=Insure`
  - Screenshots:

    `http://www.parasoft.com/jsp/products/screenshots.jsp?\\`
    `product=Insure&itemId=84`
  - Linux Journal Article:

    `http://www.parasoft.com/jsp/products/article.jsp?\\`
    `articleId=295`
  - X Journal Article:

    `http://www.parasoft.com/jsp/products/article.jsp?\\`
    `articleId=296`