Evaluierung von Software-Verifikationswerkzeugen
Was können und leisten sie?

Ralf Gerlich, Rainer Gerlich, Dr. Rainer Gerlich BSSE System and Software Engineering (BSSE)
Christian R. Prause, Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR)

Mit Softwareverifikationswerkzeugen sollen Fehler in Software gefunden werden. Doch gibt es nur spärliche Information über das, was die Werkzeuge wirklich leisten. Meistens liegt nur die Beschreibung des Herstellers vor, an der sich ein Anwender (grob) orientieren kann. So werden Werkzeuge eher nach Bedienerfreundlichkeit oder Verbreitung in der Software-Community ausgewählt als nach ihren tatsächlichen Fähigkeiten, Fehler zu finden – die von den Herstellerangaben abweichen können. Aber ohne detaillierte Information über die Fähigkeiten eines Werkzeuges ist es objektiv gesehen eigentlich nur ein gutes Gefühl, mehr für die Qualitätssicherung getan zu haben, mindestens etwas mehr als vorher. Doch könnte man vielleicht noch mehr tun? Um diese Frage zu klären, hat das Raumfahrtmanagement des Deutschen Zentrums für Luft- und Raumfahrt (DLR) eine Untersuchung beauftragt an sechs Werkzeugen, angewendet auf repräsentativeMiddleware (Embedded Software) aus der Raumfahrt. Diese Aktivität wird aktuell an weiteren Werkzeugen und weiterer Software fortgeführt. Ergebnisse der abgeschlossenen Untersuchung werden in diesem Beitrag vorgestellt.

Überblick

Die sechs Werkzeuge repräsentieren einen Querschnitt bzgl. der statischen und dynamischen Verifikationsverfahren. Wesentlich für die Auswahl war, dass außer Werkzeugkonfiguration keine manuellen Eingriffe während der Analyse, insbesondere beim dynamischen Verfahren, notwendig sind. Die Werkzeuge verwenden folgende Methoden:

- symbolische Ausführung (2)
- abstrakte Interpretation (2)
- automatisches Testen (1).

Das Testwerkzeug wendet massive Stimulation auf jede Funktion der Middleware an, beobachtet auftretende Anomalien wie Exceptions oder Index-Out-Of-range, und wertet sie aus. Außerdem wurde noch ein Compiler (gcc mit `-Wall` unter mingw) berücksichtigt, um einen Anhaltspunkt zu bekommen, in wie weit die Fähigkeiten heutiger Compiler vergleichbar sind mit denen von StaticAnalyzern.

Der Vergleich der Ergebnisse der verschiedenen Werkzeuge war eine große Herausforderung und konnte nicht wie beabsichtigt automatisch durchgeführt werden: teilweise fehlen Zeilennummern für die Zuordnung, Meldungen zum selben Fehler können für verschiedene Zeilen gemeldet werden, auch doppelt. Im Folgeprojekt, wird nun versucht, den Automatisierungsgrad aufgrund der früheren Erfahrungen zu erhöhen.

Ratschläge für die Werkzeugauswahl und -anwendung
Die folgenden Ratschläge – auf Basis häufiger Fehlannahmen – sollen helfen, Fehler bei der Auswahl und dem Einsatz von Verifikationswerkzeugen zu vermeiden:

Die Werkzeuge unterscheiden sich nicht.

Falsch. Die Werkzeuge implementieren immer eine bestimmte Methode der Fehlererkennung und wenden einen bestimmten Regelsatz an. Daher können sich die Ergebnisse hinsichtlich erkannter Fehler signifikant unterscheiden.

Ein Werkzeug findet alle Fehler.

Falsch. Ein Werkzeug kann nur die Fehler, genauer die Fehlertypen, finden, die die zugrunde liegende Methode und die Implementierung unterstützen. Die Implementierung kann die Bandbreite der Methode noch weiter einschränken.

Einfach das Werkzeug anwenden, das reicht.

Falsch. Für die Anwendung sollte eine Strategie entwickelt werden. Welche Fehler kann ein Werkzeug erkennen, welche sollen erkannt werden, wie hoch darf das Restrisiko sein, welche Werkzeuge sollte man kombinieren?

Eine solche Strategie setzt die Kenntnis der Fähigkeiten voraus.

Alle Werkzeuge haben das Ziel, alle Fehler zu finden.

Falsch. Werkzeuge können bzw. müssen wegen begrenzter Ressourcen (Speicher, Zeit, Genauigkeit)Kompromisse schließen. Manche minimieren die (Warte)Zeit zulasten der Genauigkeit und Vollständigkeit.

Diese schnelle, „oberflächlichere“ Fehlererkennung kann durchaus sinnvoll sein, um viele einfache Fehler zu beseitigen, um die Laufzeiten genauerer / vollständigerer Werkzeuge zu reduzieren, also um den meisten groben Müll wegzuräumen, damit anschließend der feinere Müll besser aufgenommen werden kann, d.h. eine mehrstufige Strategie zur Optimierung.

Die Werkzeuge erzeugen viele unzutreffende Meldungen.

Falsch. Die aktuelle Untersuchung zeigt, dass nur ein kleiner Anteil der Meldungen unzutreffend ist, im Durchschnitt aller Werkzeuge ca. 25%.

Der üblicherweise kommunizierte oder angenommene Wert stellt ein Vielfaches dieses beobachteten Wertes dar.

Das Werkzeug wird nur zur Abnahme am Projektende eingesetzt.

Falsch. Diese Strategie führt erfahrungsgemäß zu einer hohen Anzahl von Meldungen, wobei ein hoher Anteil davon unzutreffend sein kann - im Sinne von „der Fehler kann nicht auftreten“ – oder nur für wenige Randfälle zutrifft – im Sinne von „es ist unwahrscheinlich, dass der Fehler auftritt.“

Jeder gemeldete Fehler kann auch beseitigt werden.

Falsch. Bei vielen Meldungen, d.h. einer Mischung aus kritischen, weniger kritischen und unkritischen Meldungen, können kritische Meldungen übersehen werden. Ziel sollte daher sein, durch Vorsorgemaßnahmen wie Anwendung von „Best Practices“ und

frühzeitigem Werkzeugeinsatz (s.o.) die Anzahl unzutreffender Meldungen zu reduzieren.

Fehler, die von der Implementierung eines Werkzeugs unterstützt werden, werden immer gemeldet.

Falsch. Bestimmte Fehler können die Meldungen für andere Fehler unterdrücken. Einen vollständigen Überblick erhält man erst dann, wenn keine (kritischen) Meldungen mehr erscheinen.

Unzutreffende Meldungen kann man ignorieren.

Falsch. Ob eine Meldung zutrifft, also einen Fehler anzeigt, der beseitigt werden muss, oder nicht, kann erst nach (eingehender) Analyse festgestellt werden. Das erfordert (wahrscheinlich viel) Aufwand.

Ziel sollte also sein, solche Meldungen zu vermeiden durch geeignete Maßnahmen wie durch Anwendung von „Best Practices“ und dem frühzeitigen Werkzeugeinsatz, damit die Ursachen für unzutreffende Meldungen baldmöglichst erkannt werden können.

Meldungen sind immer verständlich und nachvollziehbar.

Falsch. Die Aussage der Meldung ist nicht immer leicht nachvollziehbar. Bei einigen Werkzeugen muss man die zugrunde liegende Methode kennen, um die Meldung verstehen und die Stelle des Fehlers finden zu können.

Die Meldungen sind eindeutig hinsichtlich der Identifikation eines tatsächlichen Fehlers.
Falsch. Die Meldungen müssen meistens interpretiert werden. Ob die Meldungen zutreffen, hängt auch davon ab, welche Sicht das Werkzeug auf die Software hat. Je nachdem, ob Kontext berücksichtigt wird, also einschränkende Bedingungen bspw. in einem integrierten System, oder nicht, kann die Meldung ein True oder False Positive sein (zutreffend oder nicht, TP oder FP). Bei manchen Werkzeugen kann festgelegt werden, ob Kontext berücksichtigt wird oder nicht. Wenn ja, dann ist die Meldung eindeutig bzgl. der Festlegung zum Kontext.

Kennzahlen
Für die Auswertung wurden 20 Fehlertypen identifiziert, auf die alle Meldungen aller Werkzeuge abgebildet werden konnten, und in drei Kategorien unterteilt: kritisch, potenziell kritisch und unkritisch. Als Kennzahlen wurden Sensitivity und Precision bestimmt, über alle Fehlertypen, für die 3 Kategorien und pro Fehlertyp. Beide Werte liegen zwischen 0 und 1. Je näher bei 1, desto besser.

Die Sensitivity charakterisiert die Fähigkeit, Fehler zu finden und ist der Quotient aus der Anzahl gefundener Fehler zu der Gesamtanzahl der Fehler. Da prinzipiell die Anzahl von Fehlern in der geprüften Software nicht bestimmt werden kann, wurde sie durch die Vereinigungsmenge aller mit allen Werkzeugen gefundenen Fehler approximiert, ergänzt durch weitere Fehler, die bei der Analyse der Werkzeugmeldungen manuell gefunden wurden.

Die Precision ist ein Maß für die Treffsicherheit eines Werkzeuges und ist der Quotient aus der Anzahl tatsächlicher Fehler unter allen Meldungen des Werkzeuges (TP / (TP+FP)).

Charakterisierung der Werkzeuge
Die von den Werkzeugen benutzten Verfahren werden in Tab. 1 kurz charakterisiert.

Werkzeug	Typ	Analysemethode
1	statisch	SymbolischeAusführung, Datenflussanalyse
2	statisch	Abstrakte Interpretation
3	dynamisch	Auto-Stimulation
4	statisch	SymbolischeAusführung, Datenflussanalyse
5	statisch	Datenflussanalyse
6	Compiler	Syntax, Semantik

Tab. 1: Charakterisierung der Werkzeuge

Ergebnisse
Die hier dargestellten Ergebnisse spiegeln nur einen kleinen Ausschnitt der erarbeiteten Ergebnisse wider. Sie beruhen auf bestimmten Auswertungskriterien. Bei deren Variation ändern sich auch die Werte für Sensitivity und Precision.

Sensitivity und Precision sind nicht korreliert. Die Werte von Sensitivity variieren von 0,04 bis 0,7, die von Precision von 0,5 bis 0,98. Ein Werkzeug mit geringer Sensitivity kann eine hohe Precision haben.

Insgesamt werden – überraschenderweise - ca. 2/3 der Meldungen nur von einem der Werkzeuge generiert, so dass ein erheblicher Zuwachs für die Sensitivity bei Kombination mehrerer Werkzeuge erwartet werden kann. Bspw. steigt der Wert der höchsten Sensitivity von 0,5 eines einzelnen Werkzeuges auf 0,8 bei Kombination von zwei Werkzeugen.

Ein weiteres Ergebnis der Untersuchungen und Diskussionen mit den Werkzeugherstellern ist in Bild 1 dargestellt. Die Werkzeuge müssen nicht notwendigerweise alle das Ziel haben, möglichst viele Fehler zu finden. Die Implementierung kann auf einem Kompromiss beruhen: Sensitivity gegen Ausführungszeit. So kann es durchaus sinnvoll sein, mit einem Werkzeug mit geringerer Sensitivity schnell bestimmte Fehlertypen zu finden, um dann später umfangreichere und/oder effizientere Analysen für komplexere Fehlertypen durchführen zu können. Man sollte nur wissen, welcher Kompromiss geschlossen wurde.

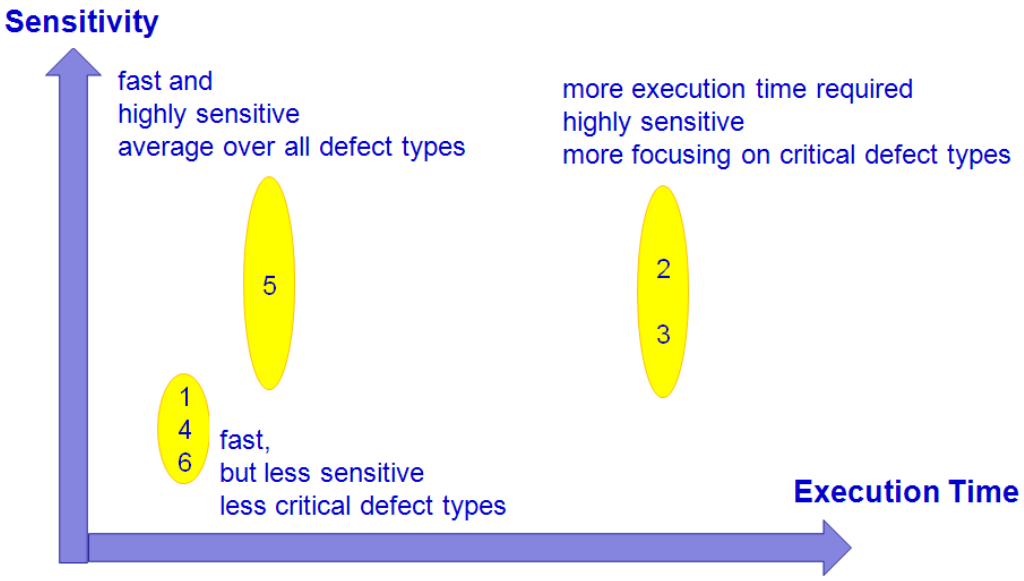

Bild 1: Sensitivity vs. Ausführungszeit

Zusammenfassung

Die Ergebnisse zeigen, dass bei der Planung der Verifikation von Software Kenntnisse über die Eigenschaften der auszuwählenden Werkzeuge unumgänglich sind, wenn das Ziel einer optimalen Qualität der Software erreicht werden soll. Zur weiteren Optimierung und Effizienzsteigerung sollten die Werkzeuge frühzeitig und kontinuierlich sowie ggf. kombiniert über die Codierungsphase eingesetzt werden. Dadurch kann der Codierstil auf die Bedingungen der späteren Verifikation abgestimmt werden.Viele Meldungen können auf diese Weise vermieden werden, inkl. dem vermutlich sonst hohem manuellen Aufwand.

Der Compiler nur mit der Option –Wall ist keine ernsthafte Konkurrenz zu den StaticAnalysern. Eine weitere Untersuchung ist geplant, um zu klären, ob bei Aktivierung der Optimierung bessere Ergebnisse erreicht werden.

Weitere Information zu der Auswertung und den Ergebnissen sind unter [1] und [2] zu finden, oder auf Nachfrage bei den Autoren.

Danksagung

Die Autoren danken den Werkzeugherstellern bzw. -distributoren, für die Werkzeuge 4 und 5, die eine Evaluierungslizenz zur Verfügung gestellt haben.

Der Inhalt dieses Beitrags ist ein Ergebnis des Projektes “Evaluierung von Software-Verifikationsmethoden und -werkzeugen” (ESVW) für das Raumfahrtmanagement des Deutschen Zentrums für Luft- und Raumfahrt (DLR) im Auftrag des Bundesministeriums für Wirtschaft und Energie (BMWi), Fkz . 50PS1505.

Literatur und Quellenverzeichnis

[1] C.R.Prause, R.Gerlich, R.Gerlich, A.Fischer: „Characterizing Verification Tools Through Coding Error Candidates Reported in Space Flight Software”,

Eurospace Symposium DASIA'15 "Data Systems in Aerospace", 19 – 21 May, 2015, Barcelona, Spain

[2] R.Gerlich, R.Gerlich, A.Fischer, M.Pinto, C.R.Prause: „Early Results from Characterizing Verification Tools Through Coding Error Candidates Reported in Space Flight Software", Eurospace Symposium DASIA'16 "Data Systems in Aerospace", 10 – 12 May, 2016, Tallinn, Estonia

Autoren:

Dr. rer.nat. Dipl.-Inf. Ralf Gerlich begann vor 20 Jahren mit der professionellen Softwareentwicklung im Raumfahrtumfeld, zunächst mit Schwerpunkt Systemprogrammierung. Inzwischen beschäftigt er sich hauptsächlich mit Methoden der Softwareverifikation in allen Ebenen der Softwareentwicklung (konstruktiv, organisatorisch und analytisch), sowohl manuell als auch werkzeuggestützt und vollautomatisch. Dabei gilt sein Interesse sowohl der theoretisch-mathematischen als auch der praktisch-pragmatischen Seite. Er ist seit Gründung der Firma Mitarbeiter bei „BSSE System and Software Engineering".

Dr. rer.nat. Dipl.-Phys. Rainer Gerlich verfügt über mehr als 35-jährige Erfahrung im industriellen Software Engineering, davon mehr als 30 Jahre in der Raumfahrt, mit Schwerpunkten in „Automation im Software-Lifecycle", „Requirements Engineering", „Verifikation", „Automatisches Testen, „Embedded Systems", „Sicherheitskritische Anwendungen", „Projekt-management", „Werkzeugentwicklung". Seit 1996 ist er Inhaber von „BSSE System and Software Engineering".

Dr. rer. nat. Dipl.-Inform. Christian Prause beschäftigt sich seit über 10 Jahren professionell mit dem Thema Softwareentwicklung und insbesondere Softwarequalität; zuerst am Fraunhofer Institut für Angewandte Informationstechnik und später als Fachgebietsleiter Softwarequalitätssicherung im Raumfahrt-management des DLR. Hier ist er verantwortlich für die Softwareproduktsicherung in allen größeren Projekten im nationalen Raumfahrtprogramm und die Weiterentwicklung des Fachgebietes. In diesem Rahmen hat er auch das Projekt ESVW beauftragt und auf Auftraggeberseite geleitet. Er ist gespannt auf die Ergebnisse der Folgeaktivitäten.

Kontakt:

Email: Ralf.Gerlich@bsse.biz
Rainer.Gerlich@bsse.biz
Christian.Prause@dlr.de