Warum stürzt mein Programm ab?

Finden von Fehlerursachen in Programm und Prozess

Andreas Zeller
Universität des Saarlandes
zeller@acm.org

Dass manche Programme nicht so laufen, wie sie sollen, ist eins der Grundprobleme der Informatik. Die Forschung hat sich in den vergangenen Dekaden vornehmlich auf die *Vorbeugung* konzentriert – also Verfahren, die das Auftreten von Fehlern von vorneherein ausschließen sollen. In den letzten Jahren hat aber auch das *Heilen* erhebliche Fortschritte gemacht – mit Verfahren, die Fehlerursachen in Programmläufen automatisch bestimmen, und aus Fehlern lernen, um den Entwicklungsprozess zu verbessern.

1 Ursachen im Programmlauf

Die Ursachen für Fehlverhalten lassen sich zunächst im *Programmlauf* suchen. Hier stehen mittlerweile leistungsfähige Verfahren zur Verfügung, die Fehlerursachen automatisch bestimmen. Neben allgemeiner Fortschritte in der dynamischen Programmanalyse und der statistischen Fehlersuche ist hier vor allem das *Delta Debugging*-Verfahren zu nennen, das mittels systematischer Experimente die Fehlerursachen schrittweise einengt.

Wie funktioniert Delta Debugging? Die Grundidee ist einfach. Alles, was wir benötigen, ist ein fehlschlagender automatischer Test, und eine Möglichkeit, die Umstände, die den Programmlauf bestimmen, zu verändern – und zwar so, dass der Test erfolgreich abläuft. Dies kann etwa eine alternative (oft triviale) Eingabe sein, oder eine frühere, funktionierende, Programmversion. Delta Debugging prüft nun systematisch Konfigurationen dieser Umstände, bis es den fehlerverursachenden Unterschied eingeengt hat. Diese Unterschiede können auftreten als

- Fehlerverursachende *Eingaben* („Die Eingabe eines Umlauts sorgt für den Absturz“)
- Fehlerverursachende *Code-Änderungen* („Die Änderung an Zeile 314 durch Entwickler Z sorgt für den fehlschlagenden Test“)
- Fehlerverursachende Thread-Schedules („Der Thread-Wechsel am Beginn von f() sorgt für die Race Condition“)
- Fehlerverursachende *Programmzustände* („Der Zyklus im Syntaxbaum sorgt für die Endlosrekursion des Compilers“)
- Fehlerverursachende *Anweisungen* („Die Anweisung in Zeile 566 verursacht den fehlerhaften Zustand“)

All diese Ursachen lassen sich mit Delta Debugging automatisch bestimmen. Für Eingaben oder Code-Änderungen ist die Implementierung leicht und lässt sich einfach in Entwicklungsumgebungen integrieren. In Eclipse etwa ist noch nicht einmal ein Knopfdruck nötig: Schlägt ein JUnit-Test fehl, setzt sich automatisch ein Delta Debugging-Plugin in Gang, das nach kurzer Zeit auf die Fehlerursache verweist. Das spart Zeit – und ermöglicht den Entwicklern, die ungeliebte Fehlersuche an den Rechner zu delegieren. Gleichzeitig sorgen die Verfahren für ein besseres Verständnis der Fehlersuche und ihrer Grundlagen; wer mehr zum Thema wissen will, dem sei das Buch „Why Programs Fail“ (Morgan Kaufmann und dpunkt.verlag, 2005) empfohlen:

http://www.whyprogramsfail.com/

2 Ursachen im Entwicklungsprozess

Die nächste Herausforderung aber ist die Suche nach Ursachen im *Entwicklungsprozess* – nicht zuletzt, weil Korrekturen hier für langfristige Verbesserungen sorgen können. Hier hat sich in den letzten Jahren viel getan, und zwar, weil wir mittlerweile über aussagekräftige Daten über Entwicklungsprozesse verfügen. Diese Daten stecken in *Software-Archiven,* die die Entstehungsgeschichte eines Systems dokumentieren – in Form von *Versionsarchiven,* die die Änderungen aufzeichnen, und *Fehlerdatenbanken,* die aufgetretene Fehler dokumentieren. Die systematische Analyse und Kopplung solcher Archive kann Fragen beantworten, wie sie in der Software-Entwicklung regelmäßig anfallen:

- Ich habe mein System folgendermaßen strukturiert. Ist dies so sinnvoll?
- Ich möchte die Funktion f() ändern. Muss ich noch mehr ändern?
- Welche meiner Komponenten sollte ich am sorgfältigsten testen?
- Ich habe die Wahl zwischen zwei Änderungen. Welche birgt das geringere Risiko?

Solche Fragen sind im Allgemeinen schwer zu beantworten. Die Analyse der Software-Historie eines Projekts kann jedoch erstaunlich zutreffende Vorhersagen treffen – und das vollautomatisch, ohne dass mit großem Aufwand kontrollierte Experimente oder projektbegleitende Messungen durchgeführt werden müssten. Tatsächlich sind die Voraussetzungen zum Einsatz der Verfahren – ein Versionsarchiv und ggf. eine Fehlerdatenbank – Bestandteile jedweder systematischen Software-Entwicklung. Insofern wurden sie auch in der Vergangenheit genutzt, um Aussagen über Entwicklungsprozesse zu gewinnen. Neu ist jedoch, dass solche Archive nun öffentlich verfügbar sind, was die Forschung und den Vergleich von Ergebnissen erheblich erleichtert.

Zudem gibt es Firmen, die erstaunlich offen sind, was ihren Umgang mit Fehlern angeht. In Zusammenarbeit mit Microsoft Research haben wir für fünf Projekte (u.a. Internet Explorer und Internet Information Server) untersucht, ob gängige *Komplexitätsmetriken* (wie etwa zyklomatische Komplexität, Anzahl der Codezeilen oder Vererbungstiefe) mit Fehlerdichte korrelieren (Nagappan, Ball, Zeller: „Mining Components to Predict Component Failures“, ICSE 2006). Annahme war, dass bestimmte Code-Eigenschaften die Entstehung von Fehlern fördern. Ergebnis: Für jedes Projekt fanden wir signifikant korrelierende Metriken – jedoch für jedes Projekt andere. Keine der Metriken war universell geeignet, Fehlerdichte vorherzusagen.

Andererseits stellte sich heraus, dass *innerhalb* eines Projektes Metriken durchaus geeignet waren, Fehlerdichten vorherzusagen – sofern man eine Linearkombination derjenigen Metriken wählt, die sich über die Historie hinweg als gute Prädiktoren erwiesen haben. Das heißt: Hat man einmal anhand der Geschichte herausgefunden, welche Metriken für das Projekt geeignet sind, kann man mit guter Vorhersagekraft rechnen.

Neben Komplexitätsmetriken lassen sich weitere Code-Eigenschaften auf die Korrelation mit Fehlerdichte untersuchen. Wer etwa wissen möchte, wie sich die Verwendung von Zusicherungen, Vererbung, Parallelität oder anderen Sprachfeatures auf die Fehlerdichte auswirkt, kann eine entsprechende Untersuchung in wenigen Stunden implementieren und durchführen. Auch wenn nicht sicher ist, dass hierbei *universelle* Aussagen enstehen – projekt-spezifische Korrelationen lassen sich in jedem Fall bestimmen, und auch statistisch hinsichtlich der Vorhersagekraft bewerten.

3 Fazit

Alles in allem hat das Gebiet der Fehlersuche in den letzten Jahren gewaltige Fortschritte gemacht – sowohl was die Suche in *Programmen* angeht, als auch, was die Suche in *Prozessen* angeht. Die große Herausforderung der Zukunft wird sein, die großen Datenmengen, die in Programmläufen und während der Programmentwicklung anfallen, zu organisieren, und systematisch zu durchsuchen – eine Goldgrube nicht nur für Softwaretechniker, sondern auch für Forscher in Statistik, Data Mining, und Machine Learning.

Weiterführende Verweise zu Projekten und Werkzeugen finden Sie auf unserer Homepage

`http://www.st.cs.uni-sb.de/`