

■ Software-Qualität:

## Ein Tool, das den Überblick behält

Die US-Firma Coverity, gegründet von jungen Forschern der Stanford-Universität, hat sich auf die Analyse großer Quellcode-Bestände spezialisiert. Bei umfangreichen Software-Projekten mit Millionen Zeilen Code ist es praktisch unmöglich, manuell einen Überblick über die Abhängigkeiten zwischen den Modulen und Dateien zu behalten. Die Werkzeuge von Coverity decken Fehler und Schwachstellen auf, die sich ins Code-Dickicht eingeschlichen haben.

Fehler in Software-Projekten liegen in der Natur der Sache. Wenn Steven Machernis, der Europa-Manager von Coverity ([www.coverity.com](http://www.coverity.com)) das Tool „Prevent SQS“ einem Kunden vorführt und zur Probe den Quellcode des Kunden analysiert, findet er „durchschnittlich einen Fehler pro 1000 Zeilen Quellcode“. „Projekte mit sorgfältig gepflegtem Code enthalten weniger Fehler – etwa pro 10 000 Zeilen Quellcode tritt hier ein Fehler auf“, berichtet er. Doch bei großen Projekten mit Millionen Zeilen Code summieren sich auch diese seltenen Fehler zu einem beträchtlichen „Überraschungspotential“ für die spätere Anwendung. Im Industrieinsatz kann aus so einer „Überraschung“ schnell ein Sicherheitsrisiko werden.

Dabei programmieren die Kunden, die Coverity nutzen, keineswegs Spaghet-

ti-Code. Denn mit den Speicherpreisen fallen auch die Hemmschwellen, immer umfangreichere und komplexere

Software einzusetzen. Ein Beispiel aus dem PC-Bereich: Windows NT 4.0, erschienen 1999, hatte noch 15 Millionen Zeilen Quellcode. Bei Windows Vista, sieben Jahre später, sind es fast 50 Millionen Zeilen. Im gleichen Trend liegt auch Geräte-Software jeglicher Art – seien es Steuergeräte im Fahrzeug oder vernetzte Büro- und Unterhaltungsgeräte. Die Komplexität steigt jedoch nicht linear mit dem Code-Umfang, sondern exponentiell.

Im Jahr 1999 begann an der Stanford University in Kalifornien ein For-

### Code-Analyse für C/C++

Diese Probleme deckt die Coverity-Code-Analyse ab:

- ▶ Path Flow Engine: untersucht alle Verzweigungen und alle Ausführungspfade in jeder Funktion.
- ▶ Acceleration Engine: analysiert jeden Ausführungspfad einzeln.
- ▶ False Path Engine: deckt Code auf, der zur Laufzeit niemals ausgeführt wird, weil die dafür nötigen Bedingungen nie gegeben sind.
- ▶ Data Tracking Engine: verfolgt die möglichen Werte von Integer- und Boole'schen Variablen. Sorgt z.B. auch dafür, dass Arrays nicht überfüllt werden.
- ▶ Data Propagation Engine: überwacht, wie skalare und Pointer-Werte übergeben werden und sich über Aufrufe verbreiten. Schutz vor Null-Pointern.
- ▶ Interprocedural Summary Engine: kontextbezogene Analyse eines gesamten Software-Systems, verfolgt Referenzen und Querverbindungen.
- ▶ Incremental Analysis Engine: verbessert die Performance durch Zwischenspeicherung von Analyse-Ergebnissen.
- ▶ Type Flow Engine: überwacht alle Typ-Beziehungen, gewährleistet, dass die Typen in Ausdrücken und Variablen zusammenpassen inklusive Klassenhierarchien.

