29 Rechnergestützte Diagnose in Software-Entwicklung und -Test

Dierk Ehmke
Hochstr. 59, D-64285 Darmstadt
mail@d-ehmke.de

Zusammenfassung

PeriPlus wurde für die rechnergestützte Diagnose in Software-Entwicklung und -Test komplexer Systeme entwickelt. Einsatzgebiete sind etwa Daily-Build und Test. PeriPlus erkennt und analysiert Probleme, benachrichtigt gezielt Administratoren oder Problemverursacher, bereitet Ausgaben auf und stellt Metriken bereit. Ein Prototyp wird erfolgreich für große Softwaresysteme eingesetzt.

29.1 Beobachtungen

Im Folgenden konzentrieren wir uns auf die Routinetätigkeiten im Daily-Build. Routine schließt hier unter anderem ein:

- Administratoren schauen häufig nach, ob der Daily-Build noch läuft.
- Im Fehlerfall werden große Log-Dateien (mehrere MB groß, verteilt auf viele Dateien) analysiert, um das Problem zu finden. Hinzu kommt, dass das Herausfiltern oft vorkommender doppelter oder vernachlässigbarer Fehlermeldungen ebenfalls Zeit kostet. Weit verstreute Einträge in verschiedenen Logdateien führen dazu, dass Fehlerhinweise übersehen werden. Diese Arbeiten erfordern viel Disziplin, wirken demotivierend und ihre häufige Wiederholung ist bekannte Ursache für weitere Fehlleistungen.
- Ermitteln derjenigen Person, die für dieses Problem verantwortlich ist, sowie deren Benachrichtigung.
- Nach dem Lauf automatisierter Smoke-Tests entscheiden, ob Testeingangskriterien erfüllt wurden.
- Überwachung der Code-Qualität etwa mit Lint, Purify und Style-Guide-Checkern. Anhand der Logfiles werden die Entwickler ermittelt, die ihren Code überarbeiten müssen. Das wird in vielen Organisationen sporadisch gemacht und gerät leicht in Vergessenheit. Andere ‘schleichende’ Trends: Entwickler stellen Code nicht zurück, die Testabdeckung sinkt, Entwickler kommen mit ihren Aufgaben nicht voran.
- Viele Entwickler schauen sporadisch nach dem Status des Daily-Build und unterbrechen zu diesem Zweck ihre Arbeit.
- Da der Projektfortschritt wesentlich vom Daily-Build abhängt und jederzeit Probleme auftreten können, werden besonders kompetente Mitarbeiter für die Administration benötigt.

Wichtig hier: Es gibt kaum Werkzeugunterstützung für die aufgezählten Tätigkeiten.

Teilweise liegt die Ausführungsdauer dieser Tätigkeiten im Bereich weniger Minuten. [DeMarco, Timothy Lister] beschreibt, dass es nach kleinen Unterbrechungen fünfzehn Minuten dauert, bis die ursprüngliche Aufgabe weiterbearbeitet wird.

29.2 Zielsetzung

Manche der beschriebenen Routinetätigkeiten sind komplex und bedürfen intensiver Expertenarbeit, viele sind trivial. Wir schätzen, dass die Paretoregel (80-20-Regel) zutrifft, das heißt 80% der Probleme sind eher trivial und 20% sind Expertenprobleme.

Komplexe Probleme lassen sich schwerlich automatisch diagnostizieren, spätestens wenn sie semantischen Ursprungs sind. Triviale Probleme dagegen scheinen geeignete Kandidaten für eine rechnergestützte Lösung zu sein.

Ziel war es, zunächst diese trivialen Probleme in den Griff zu bekommen und Administratoren davon zu entlasten. Dabei sollte möglichst nicht in die bestehenden Umgebungen eingegriffen werden und möglichst viele elektronisch vorliegende Informationen/Daten genutzt werden. Dieses minimal invasive Vorgehen ist wichtig, da Daily-Build-Umgebungen komplex sind, Durchläufe viel Zeit benötigen (häufig mehr als zwölf Stunden, [McConnell] liefert ein Beispiel mit 36 Stunden) und Abbrüche sehr teuer sind.

29.3 Grobstruktur des Diagnosesystems PeriPlus

PeriPlus gliedert sich in die drei Komponenten Fakten-Scanner, Diagnose und Aktoren, die im Folgenden beschrieben werden.

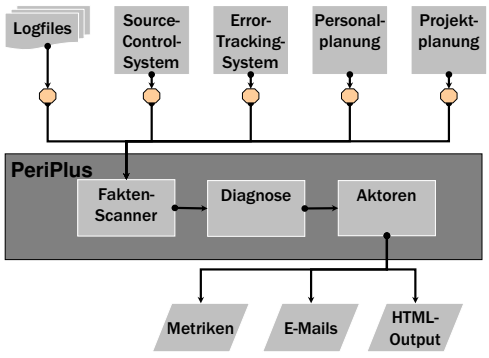

Abbildung 29.1: Grobstruktur des Diagnosesystems PeriPlus

29.3.1 Fakten-Scanner

Fakten-Scanner lesen und filtern die Daten aus der Systemumgebung und konvertieren sie in eine interne Repräsentation. Beispiele für Fakten-Scanner:

- Log-Dateien von Compilern, Lint, Purify, Style-Guide-Checkern und aus automatischen oder manuellen Tests
- Source-Control- bzw. Versions-Management- Systeme
- Daten aus dem Fehlerverfolgungssystem und Werkzeugen für die Personal- und Projektplanung

29.3.2 Diagnose

Die Diagnosekomponente erstellt aufgrund der Fakten Diagnosen. Beispiele dafür sind:

- Zuordnung: Bereits beim Quellcode ist die Zuordnung nicht eindeutig. Quellcode-Owner kann diejenige Person sein, die die Datei im Source-Control-System angelegt hat, oder diejenige, die sie zuletzt geändert hat. Danach kann aber auch die Zuständigkeit gewechselt bzw. die Person die Organisation verlassen haben. Schwieriger wird es bei Testfällen, die in der Regel unabhängig von der Quellcodestruktur sind und in gröbere funktionale Bereiche gegliedert sind. Schließlich muss für temporäre Abwesenheiten (Krankheit, Urlaub etc.) eine Stellvertreterregelung beachtet werden.
- Fehlerdiagnose: neue (das heißt, beim letzten Daily Build trat kein Fehler auf) Fehler im Quellcode, Netzprobleme, Linkfehler, Installationsfehler, Fehler in den Skripten und Fehler in Testfällen.
- Qualitätsdiagnose: Anhand der Daily-Build- Ergebnisse inklusive Smoke-Tests wird entschieden, ob Testeingangskriterien für aufwändigere Tests erfüllt sind oder das Produkt releasefähig ist.

29.3.3 Aktoren

Aktoren machen die Diagnose-Ergebnisse in der Umgebung wirksam. Beispiele dafür sind:

- Nachrichten versenden an Verursacher, Prozessverantwortliche, Projektleiter und Management.
- Erzeugung übersichtlicher Ausgaben
- Erklärungskomponente: Wir streben nicht an, dass das System hundertprozentig zutreffende Diagnosen erstellt. Das macht eine Erklärungskomponente erforderlich, mit deren Hilfe Entwickler die Diagnosen nachvollziehen und kontrollieren können.
- Fehlerverfolgungssystem: Einträge generieren für Probleme. So münden Diagnoseergebnisse direkt in den etablierten Workflow der Entwicklungsorganisation.
- Reparieren von Problemen
- Metriken-Erstellung
- Projektstatus, Tracking: Bei testgetriebener Entwicklung kann automatisch aus der Testabdeckung der Projektstatus ermittelt werden.

29.4 Ergebnisse

Der Prototyp wird in der Entwicklung zweier komplexer, strategischer Produkte seit vier Jahren eingesetzt. Das eine Produkt besteht aus 25.000 Quelldateien und ist überwiegend in Java geschrieben. Hier werden Diagnosen für den Daily-Build inklusive automatisierter Smoke- Tests erstellt, per E-Mail versand und HTML-Ausgaben erzeugt. Das andere hat einen Umfang von 3,4 Millionen LOC und ist überwiegend in C und C++ entwickelt. Hier werden basierend auf Lint- und Purify-Ausgaben kritische Code-Segmente ermittelt und die zuständigen Entwickler darüber informiert.

Es hat sich gezeigt, dass der beschriebene Ansatz die Erwartungen erfüllt. Triviale Probleme werden vom System diagnostiziert. Problemverursacher können oft zutreffend ermittelt werden.

Anstatt den Prozess ständig zu überwachen und aus umfangreichen Logfiles Probleme zu diagnostizieren, warten Administratoren den Empfang einer E-Mail ab und überprüfen das vom Diagnosesystem ausgegebene Resultat.

Für das Nadelöhr Kommunikation haben schon kleine Verbesserungen große Wirkung. So entfällt das sporadische Durchsuchen einer Unmenge von Log-Files, weil alle sich darauf verlassen können, per E-Mail relevante Informationen zu erhalten. Zudem bleiben nicht betroffene Entwickler gänzlich unbehelligt, weil gezielt nur die Verursacher der Probleme benachrichtigt werden.

29.5 Diskussion

Entscheidend für den Erfolg dürfte die Beschränkung auf die trivialen Probleme gewesen sein. Bereits hierfür kann ein signifikanter Nutzen verzeichnet werden. Wir erwarten, mehr verallgemeinerbare Regeln zu finden, die immer komplexere Sachverhalte diagnostizieren können. Schlussfolgerungen

Bedenkt man die Bedeutung zunehmender Komplexität und Kritikalität sowie der Faktoren Ressourcenknappheit und Kommunikationsaufwand in Softwareprojekten und berücksichtigt die Tatsache, dass viele Projekte scheitern, legen die gezeigten Ergebnisse nahe, den beschriebenen Ansatz weiterzuentwickeln.

Unter dem Namen PeriPlus wird aktuell ein Nachfolgesystem entwickelt, das voraussichtlich im 3. Quartal 2004 in einer ersten Version vorliegen wird.

29.6 Literaturverzeichnis

[DeMarco, Timothy Lister] DeMarco, Timothy Lister. Wien wartet auf Dich! Carl Hanser Verlag, München, Wien, 1991.

[Gyhra] N. Gyhra. Einsatz eines Expertensystems im Qualitätssicherungsprozess eines komplexen Software- Produktes. Fachhochschule Darmstadt, Fachbereich Informatik, Darmstadt, 2001.

[McConnell] Steve McConnell, `http://www.stevemcconnell.com/ieeesoftware/bp04.htm`, Stand 25. März 2004.

[Puppe] F. Puppe. Einführung in Expertensysteme. Springer Verlag, Berlin, 2. Auflage, 1991.