Code Refactoring

“Refactoring ist eine Technik zur Umstrukturierung eines vorhandenen Codes, bei der die interne Struktur geändert wird, ohne das externe Verhalten zu ändern.”

Martin Fowler, www.refactoring.com

Durch Verbesserungen und Korrekturen wird ein ursprünglich sauberes Code-Design oft verwässert. Diese Verschlechterung erhöht die Wartungskosten und verringert Performance und Stabilität. In der Vergangenheit hat man sich oft vorgenommen, schnell erfolgte Änderungen mit dem Release einer neuen Version neu zu schreiben. Die Praxis zeigt jedoch, dass diese Pläne oft zu ehrgeizig und mit vorhandener Zeit und gegebenem Budget nicht umsetzbar sind bzw. die Funktionalitäten wieder eingeschränken würden.
Beim Refactoring wird Code verbessert, indem eine Reihe kleiner Änderungen über die Zeit verteilt werden. Jede Änderung kann dabei sofort überprüft werden, ohne dass auf ein größeres Produktrelease gewartet werden muss.
Aber wie finden wir die „Kandidaten“ für das Refactoring?
Über Quellenprüfungen kann man Code finden, der zwar für Compiler akzeptabel ist, aber im Allgemeinen als „Bad Practice“ angesehen wird. Beispiele sind Type Casts oder fehlende Default-Cases in Switch Statements.
Softwaremetriken zeigen Codebereiche, die besonders betrachtet werden sollten. Prüfungen der Komplexität des Kontrollflusses wie die McCabe Cyclomatic Number zeigen uns, welche Funktionen, Klassen oder Dateien schwierig zu verstehen sind. Kopplungs- und Kohäsionsmetriken für objektorientierte Systeme bieten Einblicke in Klassen, die aus ihren sauberen Designs „herausgewachsen“ sind.
Metriken wie die „Halstead Program Difficulty” kombinieren verschiedene Messungen um komplexen Code aufzudecken.
Architekturmetriken sind insbesondere zu Beginn eines Refactoringprozesses hilfreich. Die Nichtbeachtung von Architekturhierarchien und „stability metric points“ von Subsystemen bedürfen besonderer Beachtung. Mit einem Architektur-Drilldown können wir die zugrunde liegenden Codeteile finden, die wir uns näher ansehen sollten.
Die Einführung einer expliziten Typisierung zur Verbesserung der Lesbarkeit und Robustheit von Programmen ist eine weitere Refactoring-Aufgabe. Statt int für alles zu benutzen, kann man ein typedef oder struct verwenden, deren Struktur jeweils nur ein Element enthält. Hierdurch können Variablen klar unterschieden werden. Um effektiv sicherzustellen, dass der Typ konsistent eingeführt wird, müssen die Variablen und ihre Abhängigkeiten zu anderen Variablen und Parametern angezeigt werden.

Ein weiteres evolutionäres Downgrade von Software resultiert aus kopierten Codeteilen. Im Laufe der Zeit werden einige Kopien für Korrekturen geändert, andere jedoch vergessen. Eine Möglichkeit, ähnliche Funktionen und Codeteile zu erkennen, kann sehr hilfreich sein, um die Kopien zu überarbeiten und zusammenzuführen.
Eine typische Refactoring-Anforderung besteht darin, Daten zusammenzufassen und die Anzahl der globalen Variablen im System zu beschränken. Dies kann mit der Anwendung einfacher Metriken beginnen, um die Anzahl der Lesevorgänge und Sätze globaler und statischer Variablen zu überprüfen. Besser wären Diagramme, die die Variable und alle Funktionen zeigen, die sie direkt und indirekt verwenden. Insbesondere beim Refactoring von C-Code in C ++ - Code bietet diese Ansicht eine Entscheidungshilfe, ob dies ein Kandidat für eine Klasse ist. Mehr Werkzeugunterstützung ist erforderlich, wenn Pointer dieser Variablen involviert sind; in diesen Fällen ist eine Datenflussanalyse erforderlich, um die variablen Abhängigkeiten und tatsächlichen Verwendungen beim Durchlaufen von Pointern zu ermitteln. Selbst wenn keine Absicht besteht, Klassen zu erstellen, ist das Refactoring globaler Variablen wichtig, da sich ihre Verwendung über die Lebensdauer des Programms erstreckt. Sehr oft werden neu eingeführte Variablen aufgrund von Zeitdruck und Bedenken hinsichtlich unerwünschter Nebenwirkungen an vorhandene Variablen gekoppelt. Um dies zu regeln, muss das Refactoring gekoppelte Variablen, all ihre Verwendungen und ihre gegenseitigen Abhängigkeiten finden.

Abbildung 1 zeigt ein einfaches Beispiel für dieses Refactoring.

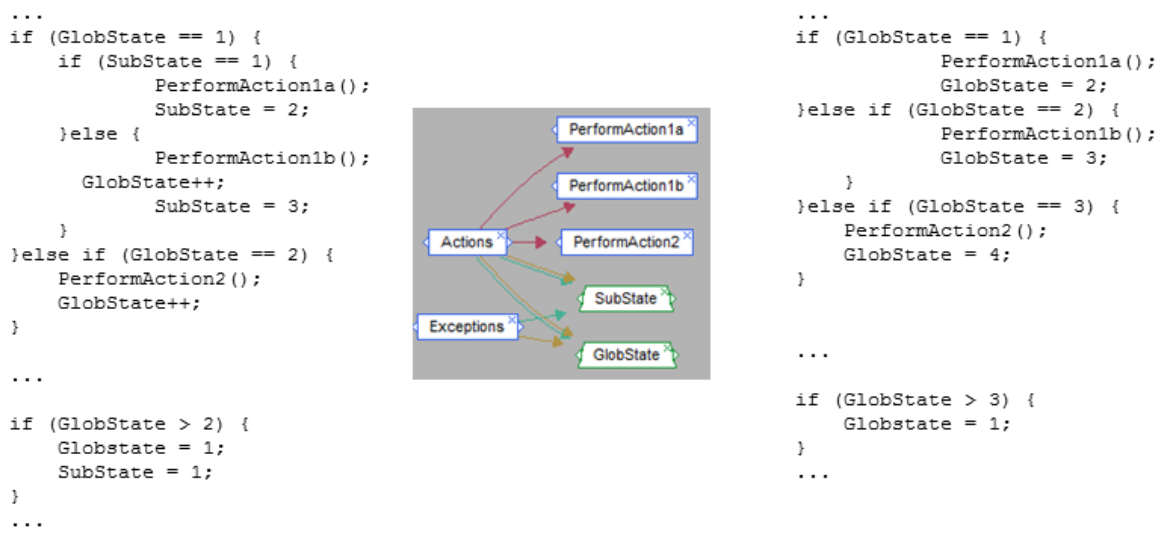

Abbildung 1. Refactoring gekoppelter Variablen: Zuordnung von SubState zu GlobState mithilfe einer vollständigen Ansicht der Verwendungen

Eine Variation des globalen Variablen-Refactorings findet sich in Multitasking- oder Multithreading-Programmen. Hier führt die übermäßige Verwendung gemeinsam genutzter Variablen zu Leistungseinbußen, da der Zugriff in kritischen Regionen geschützt werden muss. Andere Bedenken sind das erhöhte Risiko von Race Conditions, die durch fehlenden Schutz verursacht werden, und Deadlocks, die durch die große Anzahl kritischer Regionen verursacht werden. Um diese umzugestalten, muss die Kopplung zwischen Variablen identifiziert werden. In diesem Zusammenhang kommt auch der über die Tasks verteilte Zugriff auf die Variablen ins Spiel.

Zur Unterstützung von Refactoring-Aktionen ist das Tool Imagix 4D der kalifornischen Firma Imagix Corp. sehr hilfreich. Das Tool ist auch geeignet ein besseres Verständnis für eigenen und fremden Code zu bekommen.

Weitere Informationen sind auf https://www.verifysoft.com/de_imagix4d.html verfügbar.

Übersetzung eines Blogbeitrags, Quelle: https://www.imagix.com/blog/code-refactoring/