Software anderer Leute verstehen

Software kommt heute von zahlreichen Entwicklern. Oft werden in der Softwareentwicklung auch Open-Source-Komponenten genutzt. Durch Jobwechsel und Pensionierungen sind die Entwickler von Code-Komponenten nicht mehr greifbar und können somit keine Auskunft über ihre Software geben. Trotzdem muss der Software-Entwickler, der ein Projekt wartet oder weiterentwickelt, mit dem „geerbten“ Code zurechtkommen.

Eine vernünftige Dokumentation des Quellcodes ist hierbei hilfreich, in der Praxis aber leider nicht immer in ausreichend vorhanden. Im schlimmsten Fall muss der Entwickler mit dem Quellcode arbeiten und sich hierin zurechtfinden.

Der Code hat eine „natürliche“ Struktur, die von Verzeichnissen und Dateien geprägt ist. Darunter bilden Funktionen und globale Variablen bzw. Klassen, Name Spaces/Packages die nächste Strukturebene. Calls und Beziehungen von Daten sind weitere strukturelle Elemente. Alle diese Strukturen können als physische Architektur der Software bezeichnet werden.

Architekturdiagramme, welche die physische Struktur der Software darstellen, sind gute Hilfsmittel für den Software-Engineer, um die Elemente der Software und ihre Abhängigkeiten zu verstehen. Um nützlich zu sein, müssen die Architekturdiagramme interaktiv sein und dem Nutzer ermöglichen, sich frei innerhalb der verschiedenen Abstraktionsebenen zu bewegen.

Im Gegensatz zu herkömmlichen Straßen- oder Wanderkarten, bei denen bei Vergrößerungen nur bestimmte Bereiche gezeigt werden und die außerhalb liegenden Gebiete verschwinden, können Architekturdiagramme die umgebenden Bereiche auf höheren Abstraktionsebenen anzeigen, während der Drilldown im ausgewählten Bereich durchgeführt wird. Dies unterstützt Programmierer, die in der Regel auf allen Abstraktionsebenen arbeiten.

Der traditionelle Call-Graph ist als Abstraktion der Ausführungsreihenfolge von Anweisungen nach wie vor zum Programmverständnis sehr hilfreich. Ein Startpunkt könnte der Einstiegspunkt des Programms sein. Aktuell nicht interessante Stellen in Subsystemen können bei der Analyse vernachlässigt werden. Da Programmierer auf allen Abstraktionsebenen arbeiten, müssen die von ihnen verwendeten Tools alle relevanten Ebenen bereitstellen, für die sie dann Strukturen und Kontrollflüsse anzeigen und erlauben, diese zu kombinieren bzw. zwischen diesen zu wechseln. Im Idealfall sind solche Ebenen auf die Anweisungsblockebene beschränkt, die in Form von Flussdiagrammen angezeigt wird.

Je nach Ziel der Programmerkundung kann ein Datenflussansatz zu einem schnelleren Verständnis der relevanten Teile führen. In seiner einfachsten Form beginnt die Datenflussanalyse mit Datenvariablen und betrachtet alle Funktionen, die direkt oder indirekt auf diese Einfluss nehmen. Ein weiterentwickelter Ansatz besteht darin, zu verfolgen, wie Datenwerte aus anderen Datenwerten berechnet werden. Hiermit kann theoretisch gezeigt werden, wie jede Datenvariable aus Eingabewerten berechnet wird.

Ein weiterer Schlüssel zum Programmverständnis sind Nebenläufigkeiten. Für Programme mit rein linearem Steuerfluss sind die beiden oben beschriebenen Analysen ausreichend. Insbesondere bei Embedded Systemen sind Programme mit parallel ablaufenden Tasks sehr verbreitet, so dass für deren Verständnis die Analyse von Nebenläufigkeiten erforderlich ist. Der erste Schritt ist hierbei die Tasks in diesen Programmen zu identifizieren. Die verschiedenen Tasks werden in der Regel durch den Aufruf einer Systemroutine gestartet und können daher mit einfachen Call-Graph-Suchen ermittelt werden. Der nächste Schritt besteht darin, zu bestimmen, wie die Tasks über gemeinsam genutzte Datenvariablen, Ereignisse oder Signale miteinander interagieren. Hierfür sind zusätzliche grafische Modelle in Form von Task-Flow- oder Kollaborationsdiagrammen erforderlich.

Zusammenfassend kann gesagt werden, dass für das Verständnis unbekannten Codes Modelle hilfreich sind. Ziel der Modelle ist es, die im Programm verwendeten Konzepte zu abstrahieren und zu visualisieren. Um Programmierer beim Verständnis des Codes zu unterstützen müssen verschiedene Abstraktionsebenen kombiniert werden und der Wechsel zwischen diesen Ebenen auf einfache Weise möglich sein.

Hierfür stehen dem Programmierer Tools wie Imagix 4D
https://www.verifysoft.com/de_imagix4d.html zur komfortablen Quellcodeanalyse zur Verfügung.

Übersetzung eines Beitrags aus https://www.imagix.com/blog/understanding-other-peoples-programs/