Physikalisch-Technische Bundesanstalt, Berlin (Adlershof)

Vorlesung: Wissenschaftliche Softwareentwicklung

2023/24

Dr. habil. Till Biskup

— Glossar zu Lektion 19: „Codeoptimierung“ —

Hinweis: Die nachfolgend genannten Begriffe und Definitionen erheben keinen Anspruch auf formale Korrektheit, sondern dienen lediglich dem besseren Verständnis der in der Vorlesung behandelten Themen und sind im jeweiligen Kontext zu sehen. Mehrfache, voneinander abweichende Definitionen in unterschiedlichen Kontexten sind daher möglich. Englische Begriffe werden zwar nach Möglichkeit übersetzt, erscheinen aber ggf. unter ihrem englischen Namen in der Liste. Verweise untereinander sind durch ↑ gekennzeichnet.

Clean Code „sauberer Code“, letztlich lesbarer Code, der insbesondere im Kontext der naturwissenschaftlichen Datenauswertung die essentiellen Kriterien von Wiederverwendbarkeit, Zuverlässigkeit und Überprüfbarkeit erfüllt.

Codeoptimierung Veränderung der ↑Laufzeit und des ↑Ressourcenverbrauchs von Quellcode ohne Einfluss auf sein von außen erkennbares Verhalten. Strukturiertes und systematisches Vorgehen, das die Wahrscheinlichkeit, Fehler einzuführen, minimiert. Im Gegensatz zum ↑Refactoring steht nicht die verbesserte Lesbarkeit und Kommunikation des Quellcodes im Fokus, sondern die ↑Laufzeit und der ↑Ressourcenverbrauch. In dieser Hinsicht optimierter Quellcode wird i.d.R. weniger gut lesbar sein. Deshalb sollte Codeoptimierung nur wenn und soweit wie unbedingt notwendig durchgeführt werden. Setzt zwingend ausreichende (automatisierte) ↑Tests, idealerweise ↑Unittests, und die verlässliche Messung der ↑Laufzeit und des ↑Ressourcenverbrauchs, meist mit Hilfe eines ↑Profilers, voraus.

Compiler Im Deutschen meist als „Übersetzer“ bezeichnetes Programm, das den Quellcode eines Programms in direkt auf der Hardware ausführbaren Maschinencode übersetzt. Kompilierte Sprachen sind im Gegensatz zu interpretierten Sprachen (↑Interpreter) meist deutlich schneller, aber in binärer Form (Maschinencode) an eine spezifische Hardwareplattform gebunden. Moderne Compiler beinhalten häufig einen ↑Linker.

Flag „Markierung“, hier: Bezeichnung für eine Option (einen Schalter), die einem ↑Compiler übergeben wird und ggf. zu optimiertem Binärcode führen kann. Oft sind die Standardeinstellungen von Compilern auf größtmögliche Kompatibilität ausgerichtet, nicht auf maximale Effizienz, geringste ↑Laufzeit (1.) und geringsten ↑Ressourcenverbrauch.

Interpreter Im Gegensatz zum ↑Compiler ein Programm, das den Quellcode eines Programms nicht in direkt ausführbaren Maschinencode übersetzt, sondern den Quellcode einliest, analysiert und ausführt. Die Übersetzung des Quellcodes erfolgt entsprechend zur Laufzeit des Programmes, was die Ausführungsgeschwindigkeit gegenüber kompilierten Programmen (↑Compiler) in der Regel deutlich reduziert. ↑Skriptsprachen werden in der Regel interpretiert und nur selten kompiliert.

Landau-Symbole Quantifizierung des asymptotischen Verhaltens von Funktionen und Folgen. Werden in der Informatik bei der Analyse von Algorithmen verwendet und geben ein Maß für die Anzahl der Elementarschritte oder der Speichereinheiten in Abhängigkeit

von der Größe der Eingangsvariablen an. Vgl. ↑Laufzeit, ↑Ressourcenverbrauch.

Laufzeit *run time*, 1. Zeit, die ein Programmteil oder Programm für seine Ausführung benötigt. In dieser Bedeutung ist die Laufzeit relevant für die ↑Codeoptimierung mit dem Ziel einer möglichst kurzen Laufzeit. Sie wird oft über die ↑Landau-Symbole quantifiziert und so generell vergleichbar. 2. Zeit, während der ein Programm (aktiv) ausgeführt wird.

Linker Programm, das einzelne Teile einer Software zu einem ausführbaren Programm verbindet. Folgt meist auf die Kompilierung (↑Compiler). Oftmals wird aus Gründen der Modularität und Ökonomie auf Funktionalität in Programmbibliotheken zurückgegriffen. Die Verweise auf die Funktionalität in diesen Bibliotheken können entweder statisch (statisches Linken) oder dynamisch zur jeweiligen Laufzeit (dynamisches Linken) eingebunden werden.

maschinennahe Sprache eine Programmiersprache, die nah an der Realisierung des ausführbaren Binärcodes in Maschinenbefehlen für den Prozessor (Assembler-Code) ist. C ist ein Beispiel für eine vergleichsweise maschinennahe Sprache, und daraus bezieht sie auch ihre hohe Geschwindigkeit. Viele moderne Programmiersprachen fügen zusätzliche Abstraktionsebenen zwischen Programm und Hardware ein, die das Programmieren ganz wesentlich erleichtern, aber eben auch im Allgemeinen zu einer langsameren Ausführung des Programms führen.

Plattform Zusammenspiel aus Betriebssystem und Hardware-Architektur. Betriebssysteme können bis zu einem gewissen Grad unterschiedliche Hardware-Architekturen abstrahieren, so dass der gleiche Binärcode auf unterschiedlicher Hardware lauffähig ist, ohne neu kompiliert werden zu müssen. Für maximale Effizienz (insbesondere bzgl. der ↑Laufzeit, 1.) ist die Kompilierung für eine spezifische Plattform, insbesondere Hardware, unabdingbar.

Profiler Programmierwerkzeug zur Analyse der ↑Laufzeit (1.) und ggf. des ↑Ressourcenverbrauchs von Software. Wichtige Voraussetzung für die ↑Codeoptimierung, da nur ein Profiler erlaubt, die für eine Optimierung relevanten Codebereiche zu bestimmen und den Effekt der Optimierung zu quantifizieren.

Refactoring Verbesserung der Qualität des Quellcodes einer Software ohne Einfluss auf ihr von außen erkennbares Verhalten. Diszipliniertes Vorgehen zum Aufräumen von Quellcode, das die Wahrscheinlichkeit, Fehler einzuführen, minimiert. Setzt zwingend ausreichende (automatisierte) ↑Tests, idealerweise ↑Unittests, voraus.

Ressourcenverbrauch Bedarf einer Software an Speicher. Unterschiedliche Algorithmen haben einen unterschiedlichen Ressourcenverbrauch und weisen vor allem Unterschiede hinsichtlich der Abhängigkeit von der Komplexität der Eingabe auf. Diese Abhängigkeit lässt sich allgemein theoretisch mit Hilfe der ↑Landau-Symbole quantifizieren und vergleichen.

Skriptsprache Meist vergleichsweise einfach zu erlernende Programmiersprache, die i.d.R. interpretiert (↑Interpreter) und nur selten kompiliert (↑Compiler) wird. Die Unterschiede zwischen Skriptsprache und interpretierter Programmiersprache sind fließend. Beispiele sind: Python, Perl, PHP, bash.

Test hier: strukturiertes Vorgehen, eine Software zu überprüfen. Setzt die Definition klarer Anfangs- und Endbedingungen (Eingabe und Ergebnis) voraus und sollte idealerweise vollständig automatisiert ablaufen können. Vgl. ↑Unittest.

Unittest ↑Test eines Codeblocks in Isolation. Ein Unittest überprüft von außen, ohne den Quellcode des zu testenden Systems zu kennen oder zu benötigen. Die getesteten Codeblöcke sind i.d.R. klein. Zwingende Voraussetzung ist, dass das (erwünschte) Verhalten des zu testenden Codeblocks eindeutig definierbar (und seinerseits in Form von Quellcode formalisierbar) ist. Unittests sind gewissermaßen die unterste Ebene (automatisierter) ↑Tests.