Physikalisch-Technische Bundesanstalt, Berlin (Adlershof)

Vorlesung: Wissenschaftliche Softwareentwicklung

2023/24

Dr. habil. Till Biskup

— Glossar zu Lektion 04: „Editoren / IDEs“ —

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.

Automatisierung *automation* Strategie, um sich die manuelle Durchführung repetitiver und meist langweiliger Prozesse zu ersparen, indem sie an Maschinen ausgelagert wird. Automatisierung sorgt für Konsistenz (aber nicht Fehlerfreiheit) und ermöglicht es den menschlichen Akteuren, ihre dadurch freiwerdende Kapazität auf die eigentlichen intellektuellen Aufgaben, die weder automatisiert noch von Algorithmen übernommen werden können, zu verwenden.

Voraussetzungen für die Automatisierung sind u.a. hinreichend uniforme und wiederkehrende Prozess(schritt)e und die intellektuelle Durchdringung der zu automatisierenden Abläufe.

Im weiteren Sinn umfasst Automatisierung auch die so weitreichende Formalisierung von (manuellen) Abläufen, dass keine Denkleistung für deren Durchführung mehr erforderlich ist.

Build-Umgebung Satz an Werkzeugen, die dafür notwendig sind, ein Programm auszuführen bzw. in von der Hardware (Prozessor) direkt ausführbaren Binärcode (Maschinencode) zu überführen. Welche Werkzeuge dazu gehören und inwieweit es einzelne, voneinander getrennt ansprechbare Werkzeuge sind, hängt stark von der gewählten Programmiersprache und dem verwendeten Betriebssystem ab. Vergleiche die Eintragungen zu ↑Compiler, ↑Interpreter und ↑Linker

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.

Debugger Werkzeug zur Diagnose und zum Auffinden von Fehlern (↑Bug) in Programmen. Debugger bringen eine Reihe hilfreicher Funktionalitäten zur Analyse eines in einem Programmablauf aufgetretenen Fehlers mit und sind oft in einer ↑IDE integriert.

Editor Programm zum Erstellen von Quellcode, der dann entweder kompiliert oder interpretiert und ausgeführt werden kann. Grundsätzlich ist für (fast) alle Programmiersprachen ein reiner Texteditor ausreichend. Oftmals steigern integrierte Entwicklungsumgebungen (↑IDE) die Produktivität allerdings ganz erheblich.

IDE integrierte Entwicklungsumgebung, engl. *integrated development environment*; Software

zur Programmierung (Quellcode-Erstellung), die neben einem ↑Editor als Hauptkomponente noch diverse weitere Werkzeuge integriert und so die Softwareentwicklung möglichst ohne Medienbrüche ermöglicht. Häufig integriert sind der Zugriff auf die ↑Versionsverwaltung, die ↑Build-Umgebung und ein ↑Debugger sowie Werkzeuge zur ↑statischen Codeanalyse. Darüber hinaus unterstützt die ↑Editor-Komponente einer IDE nicht nur (komplexe) Syntax-Hervorhebung, sondern auch Aspekte wie ↑Refactoring. Nachteile sind die steile Lernkurve aufgrund der erheblichen Komplexität dieser Programme. Eine gute IDE und ihre souveräne Beherrschung durch einen Programmierer haben andererseits erheblichen Einfluss auf Qualität und Geschwindigkeit des Programmierens.

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.

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.

Makro Folge von Anweisungen, die unter einer bestimmten Bezeichnung zusammengefasst sind und sich mit nur einem einfachen Aufruf ausführen lassen. Häufig in Skriptsprachen formuliert und entsprechend einfach zu erstellen. Dienen der Automatisierung wiederkehrender Abläufe in Programmen, z.B. in ↑IDEs.

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.

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.

Syntax Im Kontext formaler Sprachen (zu denen Programmiersprachen gehören) ein System von Regeln, wie aus einem grundlegenden Zeichenvorrat (Alphabet) syntaktisch korrekte („wohlgeformte“) Ausdrücke gebildet werden können.

VCS siehe ↑Versionsverwaltung

Versionsverwaltung engl. *version control system*, VCS; Software zur Verwaltung unterschiedlicher Versionen von Dateien und Programmen, die den Zugriff auf beliebige ältere als Versionen gespeicherte Zustände ermöglicht. Gleichzeitig ein wichtiges Werkzeug für die Softwareentwicklung und wesentlicher Aspekt einer Projektinfrastruktur.