Physikalisch-Technische Bundesanstalt, Berlin (Adlershof)

Vorlesung: Wissenschaftliche Softwareentwicklung

2023/24

Dr. habil. Till Biskup

— Glossar zu Lektion 03: „Infrastruktur“ —

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.

Bug (engl. für „Wanze, Käfer“) Programmfehler oder Softwarefehler, allgemein ein Fehlverhalten von Computerprogrammen.

Bug-Verwaltung engl. *bug tracker*; Software, die spezifisch auf die Handhabung von Fehlern (↑Bug) in Programmen zugeschnitten ist. Strukturiert und vereinfacht den Umgang mit Fehlermeldungen (*bug report*), dokumentiert aufgetretene Fehler und entkoppelt Nutzer und Entwickler zeitlich und räumlich. Darüber kann sie der Diskussion der weiteren Entwicklung dienen, das ist aber eher der Bereich eines ↑Issue Trackers oder dedizierter Projektmanagement-Software.

Build-Umgebung Satz an Werkzeugen, die dafür notwendig sind, ein Programm auszuführen bzw. in von der Hardware (Prozessor) direkt ausführbaren Binärcode 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.

Codeanalyse i.d.R. zumindest teilweise automatisch durchgeführte Analyse des Quellcodes einer Software vor oder während ihrer Ausführung. Die Analyse ohne Ausführung des Quellcodes heißt ↑statische Codeanalyse, jene während der Ausführung ↑dynamische Codeanalyse. Eng verwandt mit dem systematischen Testen von Software (vgl. ↑Unittest).

Code Review Gemeinsame Begutachtung eines Quellcode-Abschnitts eines Projektes durch mehrere Projektbeteiligte. Ein Code Review ist formaler als das ↑Pair Programming, außerdem wird beim Code Review in der Regel *nicht* programmiert, sondern nur über den Quellcode diskutiert. Ziele eines Code Re-

views sind u.a.: Wissensvermittlung innerhalb des Teams, Sicherstellen eines gemeinsamen Kenntnisstands, Entwicklung von Ideen für die Weiterentwicklung. Eine Verbesserung des diskutierten Codes geht meist damit einher, allerdings sollte Kritik immer konstruktiv sein.

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.

dynamische Codeanalyse ↑Codeanalyse während der Ausführung des Quellcodes. Ein Programm oder Teile davon werden kontrolliert ausgeführt, um so mögliches Fehlverhalten aufzuspüren. Kommt in unterschiedlichen Varianten vor, u.a. als ↑Test gegen eine Spezifikation (vgl. ↑Unittest). Gegenstück: ↑statische Codeanalyse.

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.

externe Dokumentation Dokumentation außerhalb des Quellcodes, meist von Konzepten etc. Das Problem externer Dokumentation ist noch viel mehr als bei Dokumentation im Quellcode (↑Schnittstellendokumentation), dass sie nur mit großem Aufwand synchron mit dem Quellcode gehalten werden kann.

freie Software Software, die Endnutzern die Freiheiten der Nutzung, des Überprüfens/Studierens, des Teilens und des Modifizierens der Software gewährt. Das erfordert zwingend die Offenlegung des Quellcodes der Software. Der Begriff „frei“ bezieht sich hier auf die Freiheiten des Nutzers, nicht auf die Kosten der Software. Die Begriffe „*open source*“ und „freie Software“ sind nicht gänzlich deckungsgleich.

größeres Projekt hier: Alles, was mehr als zwei Wochen Arbeit kostet und deutlich mehr als zweihundert Zeilen (reinen) Quellcode bzw. mehr als eine Handvoll Unterfunktionen umfasst. Wichtig ist der Fokus: Sobald ein Programm über längere Zeit und/oder von anderen verwendet werden soll (was eher die Regel statt die Ausnahme ist), ist es ein größeres Projekt.

IDE integrierte Entwicklungsumgebung, engl. *integrated development environment*; Software zur Programmierung (Quellcode-Erstellung), die neben einem ↑Editor als Hauptkomponente noch diverse weitere Werkzeuge integriert. 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 einer IDE 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.

Infrastruktur Personelle, sachliche und finanzielle Ausstattung, um ein angestrebtes Ziel zu erreichen. Im Kontext der Softwareentwicklung die Gesamtheit der Hilfsmittel, die (manche) Abläufe formalisieren und für Struktur und Überprüfbarkeit sorgen. Erleichtert die Arbeit des Programmierers, indem sie viele Aspekte festlegt, die so zur Routine werden (und keine Denkleistung absorbieren).

Issue Tracker Software zum Empfang, zur Bestätigung, Klassifizierung und Bearbeitung von Kundenanfragen. Erklärtes Ziel ist, dem Verlust von Nachrichten vorzubeugen und jederzeit einen Gesamtüberblick über die zu bearbeitenden Vorgänge zu ermöglichen. Ein Issue Tracker erlaubt die Zuweisung einer Anfrage (*ticket*) an einen Verantwortlichen zur weiteren Bearbeitung bis zur Lösung. Die Grenzen zur ↑Bug-Verwaltung sind mitun-

ter fließend, andererseits bilden Issue Tracker mitunter einen integralen Bestandteil einer Projektmanagement-Software.

Konvention innerhalb einer Gruppe oder einem (lokalen) Kontext getroffene (temporäre) Festlegung. Ziel von Konventionen ist die Vereinheitlichung und damit einhergehend die Befreiung von der Notwendigkeit, jedesmal aufs Neue nachdenken zu müssen, wie z.B. gewisse Prozesse durchgeführt oder Objekte benannt werden sollen. Konventionen sind im Gegensatz zu ↑Standards weniger verbindlich und deutlich flexibler sowie *ad hoc* innerhalb einer Gruppe einführbar.

Lizenz Nutzungsrecht; Software ist *per se* vom Urheberrecht geschützt, unabhängig von ihrer Funktionalität. Lizenzen übertragen Nutzungsrechte vom Urheber der Software an ihren Nutzer.

Pair Programming „Paarprogrammierung", enge und aktive Zusammenarbeit zweier Programmierer bei der Programmierung von Software. Eine Person sitzt an der Tastatur und programmiert, während die andere aktiv über den entstehenden Code und das Problem nachdenkt und ggf. korrigierend eingreift. Auffallende Probleme werden sofort angesprochen und diskutiert. Die Rollen sollten häufig wechseln (mindestens innerhalb einer Stunde), idealerweise ebenso die Zusammensetzung der Paare.

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.

Schnittstelle hier: Verbindung zwischen einem Stück Software (Programm, Routine) und seiner Umgebung. Dient der Trennung von Verantwortlichkeiten und ermöglicht Modularisierung. Ist deshalb ein wesentlicher Aspekt der Softwarearchitektur.

Schnittstellendokumentation Beschreibung der Verwendung einer ↑Schnittstelle direkt im Quellcode der Software, insbesondere der Ein- und Rückgabeparameter. Diese Dokumentation kann häufig über spezielle Werkzeuge automatisch in einem gut lesbaren Format ausgegeben werden.

Standard von einem oft internationalen und anerkannten Gremium definierte Festlegung. Standards sind im Gegensatz zu ↑Konventionen sehr viel starrer und nicht *ad hoc* von einer Gruppe einführbar.

statische Codeanalyse ↑Codeanalyse ohne Ausführung des Quellcodes. Kann neben Syntaxfehlern und Einhaltung von Konventionen mitunter auch komplexere Tests durchführen. Häufig in einer ↑IDE integriert, so dass bereits beim Schreiben des Quellcodes auf mögliche Probleme hingewiesen wird. Gegenstück: ↑dynamische Codeanalyse.

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. ↑dynamische Codenanalyse und ↑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.

Urheberrecht siehe ↑Lizenz

Versionsnummer hier: eindeutige Bezeichnung einer Version einer Software, deren Kenntnis es erlaubt, auf genau diese Version der Software Bezug zu nehmen.

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.