1. Einführung

1.1. Aufbau dieses Kurses

Der Aufbau dieses Kurses richtet sich nach der zeitlichen Reihenfolge, in der die Werkzeuge eingesetzt werden. Dieser Kurs besteht aus sieben Teilen, von denen jeder ein bestimmtes Problemumfeld behandelt. Die einzelnen Teile sind unabhängig voneinander, so dass jeder die für ihn interessantesten Teile auswählen kann. Jedem Werkzeug ist dabei ein eigenes Kapitel gewidmet.

Versionskontrolle

- Wir beginnen mit Werkzeugen zur *Versionskontrolle*, die typischerweise zu Beginn eines Projekts eingerichtet werden. Zunächst gilt es, *Änderungen* zu erkennen, beschrieben in Kapitel 2 „Änderungsverwaltung mit DIFF und PATCH“. Dann zeigen wir, wie diese Änderungen kontrolliert *verwaltet* werden – zunächst auf einzelnen Dateien (Kapitel 3 „Revisionsverwaltung mit RCS“) und dann auf Verzeichnissen (Kapitel 4 „Parallele Programmentwicklung mit CVS“).

Eingabeverarbeitung

- Das eigentliche Erstellen von Programmen ist ebenfalls Gegenstand dieses Kurses – zumindest, wenn es um das *automatische Erstellen* geht. In Kapitel 5 „Lexikalische Analyse mit LEX“, Kapitel 6 „Syntaktische Analyse mit YACC“ und Kapitel 7 „Lexikalische und syntaktische Analyse mit ANTLR “ zeigen wir, wie man aus abstrakten Strukturbeschreibungen Programme zur *Eingabeverarbeitung* generiert.

Programmbau

- Nach dem Programmieren kommt das *Bauen des Programms* – das Erzeugen eines fertigen Produkts aus den manuell erstellten Quelltexten. Kapitel 8 „Programme bauen mit MAKE“ zeigt, wie man den Bau des Programms automatisiert und die richtigen und notwendigen Konstruktionsschritte ausführen lässt. Kapitel 9 „Software automatisch konfigurieren mit AUTOCONF“ ergänzt dies um Techniken zur automatischen *Portierung* in neue Umgebungen. Schließlich betrachten wir in Kapitel 10 „Programme dokumentieren mit JAVADOC “ spezielle Verfahren, um die Dokumentation aus dem Programmcode zu generieren und so auf dem neuesten Stand zu halten.

Prototypen

- Ebenfalls Bestandteil des Programmierens ist das Erstellen von *Prototypen.* Kapitel 11 „Prototypen erstellen mit Tcl/Tk“ gibt eine Einführung in *Tcl,* eine einfache Skriptsprache, und *Tk,* ein Werkzeugkasten zum schnellen Erstellen grafisch-interaktiver Programme. Die Sprachen PERL und PYTHON kommen ebenfalls zu Wort.

Testen und Debuggen

- Ist das Programm schließlich geschrieben, muss es *getestet* werden. Kapitel 12 „Systemtests mit DEJAGNU“ beschreibt, wie man mit DEJAGNU ganze Programme *automatisch testet;* Kapitel 13 „Komponententests mit JUNIT“ zeigt dies für einzelne Programmkomponenten. Tritt ein Fehler auf – ob beim Test oder beim Kunden –, muss er dokumentiert und verfolgt werden. Diesen Prozess unterstützen Werkzeuge zur *Problemverwaltung*, besprochen in Kapitel 14 „Problemverwaltung mit BUGZILLA“. Schließlich muss der Fehler untersucht und beseitigt werden – etwa mit den *Debuggern* aus Kapitel 15 „Fehlersuche mit GDB und DDD“.

Programmanalyse

- Um allgemeines Wissen über Programme zu gewinnen, dazu dienen Werkzeuge der *Programmanalyse.* Wie man Probleme der *Programmleistung* angeht, zeigt Kapitel 16 „Laufzeitanalyse mit GPROF und GCOV“. Prüfungen, ob Programme gewissen Richtlinien genügen, wird in Kapitel 17 „Stilprüfungen mit CHECKSTYLE “ gezeigt. In Kapitel 18 „Statische Programmanalyse mit LINT“ beschreiben wir, wie man anhand des *Datenflusses* etwa uninitialisierte Variablen aufdeckt. Kapitel 19 „Program-Slicing mit UNRAVEL“ zeigt, wie man *Slices* bestimmt – Anweisungen, die den Wert einer Variablen an einer bestimmten Stelle beeinflussen.

Integrierte Umgebungen

- Zum Schluss wollen wir die betrachteten Werkzeuge in einer *integrierten Umgebung* verfügbar machen. Kapitel 20 „Integrierte Entwicklung mit ECLIPSE“ stellt die universelle Entwicklungsumgebung ECLIPSE vor, die Programmierwerkzeuge in einer *grafischen Oberfläche* integriert.

Jedes Kapitel schließt mit:

- *Konzepten* als Zusammenfassung des Inhalts
- *Bibliografischen Hinweisen* zur Vertiefung Ihres Wissens

1.2. Schreibweisen und Konventionen

Wir benutzen im diesem Kurs eine durchgängige Notation:

- *Kursive Schreibweise* hebt Begriffe hervor.
- Fremdsprachige Begriffe sind ebenfalls *kursiv* gesetzt.
- *Programm-* und *Systemnamen* sind in GROSSBUCHSTABEN gesetzt.
- Codestücke und Daten sind in `Schreibmaschinenschrift` gesetzt.
- In Beispielen mit Benutzer-Interaktion sind Benutzer-*Eingaben* in **`fetter Schreibmaschinenschrift`** gesetzt; *Ausgaben* sind durch `gewöhnliche Schreibmaschinenschrift` gekennzeichnet. Das Zeichen „`$`“ steht dabei für die UNIX-Eingabeaufforderung, das Zeichen „⊠“ für den Eingabecursor.

Schreibweisen wie etwa *der Anwender* werden in einer allgemeinen und geschlechtsneutralen Bedeutung verwendet.

1.3. Einsatz der Werkzeuge unter UNIX und LINUX

Alle Beispiele in diesem Kurs wurden unter UNIX getestet und sollten unverändert auf allen UNIX-Varianten laufen. Besonders praktisch ist es, wenn Sie eine LINUX-Distribution auf Ihrem PC installiert haben – in den meisten LINUX-Distributionen sind alle GNU-Werkzeuge von vorneherein enthalten und können über Ihre LINUX-Distribution einfach installiert werden.

Auf UNIX-Rechnern finden Sie neben den GNU-Werkzeugen oft auch noch die gleichnamigen Implementierungen Ihres UNIX-Herstellers, die sich in Funktionalität und Robustheit unterscheiden.

Soweit nicht anders angegeben, bekommen Sie die Quellcodes der GNU-Werkzeuge von der *Free Software Foundation* unter:

```
http://www.gnu.org/
```

In der Regel müssen Sie den Quellcode noch für Ihre Maschine übersetzen, was aber keine große Hürde darstellt – in der Regel genügt die Befehlsfolge

```
$ ./configure; make install
```

(In Kapitel 9 lernen Sie, was `configure` tut; Kapitel 8 beschreibt `make`.)

1.4. Einsatz der Werkzeuge unter WINDOWS

Die in diesem Kurs beschriebenen Werkzeuge sind fast alle auch für das WINDOWS-Betriebssystem erhältlich. Wie sich diese Werkzeuge unabhängig von dem Betriebssystem in einem modernen Gewand kleiden, zeigen wir in Kapitel 20 anhand von ECLIPSE. Diese integrierte Umgebung basiert auf den in diesem Kurs vorgestellten Programmierwerkzeugen – mehr noch, einige dieser Klassiker wie DIFF, RCS und MAKE sind integraler Bestandteil.

Die meisten Werkzeuge sind aber auch als eigenständige Programme unter WINDOWS verfügbar. Das wichtigste Projekt in diesem Zusammenhang ist CYGWIN. Dieses Projekt bietet eine POSIX-Schnittstelle in Form einer Bibliothek und darauf aufbauend fast alle GNU-Werkzeuge, die diese Schnittstelle inzwischen direkt unterstützen.

Viele der vorgestellten Werkzeuge sind nicht nur in Portierungen erhältlich, sondern werden von den Entwicklern aktiv unter WINDOWS unterstützt.

1.5. Literatur

Am Ende jedes Kapitels finden Sie einen Abschnitt „Bibliografische Hinweise“, der auf weiter gehendes Material verweist.

Wenn Sie sich nach dem Lesen dieses Kurses für allgemeine Grundprinzipien der Softwarekonstruktion interessieren, können wir Ihnen diese Bücher empfehlen:

- Als generelle Lehrbücher zum Umfeld der *Programmierung im Kleinen* empfehlen wir die Bücher von McConnell (1993) und Kernighan und Pike (1999). Beide beschreiben allgemeine Prinzipien aus der Praxis der individuellen Softwarekonstruktion.
- Herausragend aus den Lehrbüchern zum *Programmieren im Großen* ist das *Lehrbuch der Softwaretechnik* von Balzert (1996, 1997). Es ist nicht nur außerordentlich umfassend, sondern auch didaktisch hervorragend aufgebaut und auf dem neuesten Stand der praktischen Softwareentwicklung.
- Durchgehend empfehlenswert, wenn auch englischsprachig, sind das Buch von Sommerville (1996b), das einen guten informalen Überblick verschafft, und das Buch von Ghezzi et al. (1991), das die Grundlagen der Softwaretechnik behandelt.

Jedes dieser Werke behandelt Programmierwerkzeuge als typische Realisierungen von Prinzipien der Softwarekonstruktion; das Werk von Balzert enthält zudem eine CD-ROM mit nützlichen Werkzeugen.