Hagen Hübner
Seminar: Open-Source-Softwareentwicklung
HS2014
21.10.2014

Handout Vortrag Tools

Leitideen

- Die allermeisten Aufgaben lassen sich mit unterschiedlichen Tools mit unterschiedlichen Lizenzen lösen.
- Toolkompetenzen in punkto Kommunikation und Entwicklung sind elementar für die Mitarbeit an jedem FLOSS-Projekt.
- Das gewünschte Projekt schränkt die Toolauswahl oft ein. Ist ein Projekt etwa in einer bestimmten Sprache geschrieben, muss die Entwicklung mit passenden Tools geschehen.

Kommunikation

Bei Open-Source Projekten wird über eine Vielzahl von Kanälen kommunziert. Unidirektional per Websites, bidirektional über E-Mails (Mailinglisten), Chats (verbreitet: irc-channels) und social-media. Auch zum Einsatz kommen Tools für Screensharing, VOIP-Telefonate oder Videokonferenzen. Bei eigenen Projekten lohnt es sich, auf verbreitete Dienste und Protokolle zu setzen, und sich nicht auf eine Technologie, etwa einen social-media Kanal oder nur eine Mailinglist zu beschränken.

Programmiertoolchain

Editor: Elementar für jede Codeerstellung. Editoren unterscheiden sich hauptsächlich durch integrierte Zusatzfunktionalität (Syntax-Highlighting. Automatisches Einrücken. Autovervollständigung.) und Erweiterbarkeit (Erweiterung um neue Sprachen. Plugins wie Compiler oder Versionskontrollen-Integration.)

Versionskontrolle: Ermöglichen verteiltes und gleichzeitiges Arbeiten an Quellcode. Sichern Konsistenz und Rückkehr zu stabilen Stadien des Projekts. Es exisiteren unterschiedliche Open-Source und proprietäre Systeme und Tools.

Compiler / Interpreter: Je nach Sprache herrscht eine grössere Auswahl an Quellcode übersetzenden Tools. Unterscheiden sich in Bedienung, Lizenz und Standartunterstützung.

Debugger / Testing / Profiler: Helfen bei der Beseitigung von Bugs, Herstellung von funktionierendem Code und dienen der Runtimeanalyse und anschliessenden Optimierung des kompilierten Programmes.

Sonstiges: Bei der Entwicklung für Endgeräte oder CPU oder Betriebssystem-Architekturen welche man selber nicht benutzt sind Emulatoren notwendig.
Es gibt Tools welche aus Quellcodekommentaren Dokumentation erstellen können. Zu nennen sind etwa javadoc für Java und Doxygen für C/C++/Python und weitere Sprachen.
Pakete für unterschiedliche Linuxdistributionen und Betriebssysteme müssen auf bestimmte Art und Weise gepackt werden. Verschiedene, an das jeweilige OS angepasste Tools ermöglichen dies.

Für eigene Projekte lohnt es sich, nur Dateiformate zu verwenden welche von frei verfügbarer Software verarbeitet werden können. Proprietäre Dateiformate stellen eine Hürde dar und beschränken somit die Zahl potentieller Mitarbeitender.

IDE's vs. Einzelprogramme

IDE's scheinen mit ihren ausgeklügelten GUI's zunächst wesentlich intuitiver zu sein. Allerdings sind die Grenzen ihrer Erweiterbarkeit um einiges enger als wenn man für die Entwicklung einzelne Programme benutzt. *"Throughout a Unix system, easy things are easy and hard things are at least possible."* Um die benötigten Programme kennenzulernen ist einiges an Zeit nötig. Der Lernaufwand für eine solche modulare Herangehensweise kann sich allerdings langfristig lohnen. Neue Programmiersprachen und Werkzeuge können leicht integriert werden. Die Zeit welche man beispielsweise benötigt um mit einem Editor vertraut zu werden kann sich lohnen, wenn man diesen dann für alle Codeerstellungs- und Editieraufgaben verwendet.

Grenzen Open-Source - Proprietäre System

Es sind Projekte denkbar, bei denen zwar der Quellcode Open-Source ist, jedoch weder Open-Source Tools zur Kompilation verfügbar sind, noch die Distribution frei funktioniert. Dies ist beispielsweise bei der Entwicklung für IOS Anwendungen so. Das von Apple zur Verfügung gestellte Entwicklungskit ist nicht Open-Source. Es ist ausserdem nur unter Mac OSX lauffähig. Möchte man entwickelte Software auf seinem eignene IOS Gerät testen, oder Programme in den AppStore laden wird zudem eine Jahresgebühr von 99$ fällig. Zwar gibt es alternative Möglichkeiten IOS Programme zu kompilieren, allerdings können diese dann nicht im offiziellen Store veröffentlicht werden; der Endanwenderkreis ist dadurch erheblich eingeschränkt.

Verwendete Literatur

- Andrew Hunt / David Thomas: The Pragmatic Programmer, 1999.
- Raymond Eric Steven: The Art of Unix Programming, 2003. `http://www.catb.org/esr/writings/taoup/html/`