Stichwortliste zum Vortrag „Freie Programme attraktiv dokumentiert mittels `noweb`“ von R. Stanowsky anlässlich des Software Freedom Day am 15.09.2012 vorgetragen in der Dingfabrik Köln

1 Freie Programme

- Das vorgestellte Beispielprogramm `simCIR` wäre undokumentiert formal „frei und offen“ (*Recht* zur Benutzung, zur Weitergabe, zum Studium, zur Verbesserung), aber ziemlich nutzlos
- „frei und offen“ im Sinne von mitteilsam, einladend geht über formale Freiheiten hinaus und beschreibt mehr eine Einstellung
- Mittlerweile geschätzte 90% neu erstellter formal freier Software dienen kommerziellen Zielen
  - Verhinderung marktbeherrschender Stellung eines Produkts
  - Versuch zum Markteinstieg mit begrenztem Risiko
  - Supporteinsparung nicht mehr weiterentwickelter Software
  - Angriff auf Konkurrenten ...
- Um Ziele zu erreichen ist diese Software gerade so frei, dass am Markt noch akzeptiert
- Meist besteht gar kein Interesse an guter Dokumentation freier Software bis zu Interesse an keiner Dokumentation
  - Mehraufwand → Kapitalrendite↘
  - Kontrollverlust durch Weiterentwickler
  - Möglichkeit der de facto Rückholung durch unfreie Nachfolgeentwicklung (z.B. bei Markterfolg)
- Nichtkommerzielle freie Software ist wichtig: Nicht überall gibt es ausreichend kommerzielles Interesse
  - Wo ist der Suchmaschinen-Nach-/Feinfilter für den eigenen Rechner, der z,B. vorgetäuschte Treffer unterdrückt oder reguläre Ausdrücke beherrscht?

2 Stellenwert attraktiver Dokumentation

- Bedeutung der restlichen 10% nicht-kommerzieller freier Software kann durch einladende, attraktive Dokumentation über ihren Mengenanteil hinausgehen
- Gute Dokumentation steigert den Nutzen der freien Software erheblich
  - fremdes Programm verstehen kann schwerer sein als neu programmieren
- ... und erhöht die zu erwartende Qualität des Programms
  - systematischere Programmierung im Hinblick auf Erklärbarkeit
  - intensivere Beschäftigung, weniger (Flüchtigkeits-)Fehler
- Ein guter Programmierer kann sein Können leichter belegen, der Nutzer kann es leichter nachprüfen
  - Akzeptanz freier Software↗
  - Erreichen einer kritischen Masse von Nutzern wird wahrscheinlicher
  - Schnellere Fehlerbereinigung und Reifung/Fortentwicklung der freien Software
  - Anreiz für Fachleute aber Amateurprogrammierer sich die Funktionsweise eines freien Programms anzusehen und vielleicht zu verbessern oder Verbesserungen vorzuschlagen

3 Literate Programming

- ... zielt darauf ab die Funktionsweise eines ganzen Programms systematisch und übersichtlich zu vermitteln (nicht bloß der Bedienung)
- Vereint Dokumentation und Programm in einem Dokument
  - keine Versionsdifferenzen/-unsicherheiten
- Das gemeinsame Dokument bestimmt Abfolge und Struktur der extrahierten Dokumentation
  - Eingebetteter Programmcode wird besonders dargestellt
  - üblicherweise werden Verzeichnisse der Code-Abschnitte, verwendeter Programmstrukturen/-vereinbarungen sowie Querverweise (teil-)automatisch erstellt oder manuelle Erstellung wird unterstützt
- Der nach Verständnisbedürfnis eingestreute Programmcode wird beim Extrahieren dagegen anhand der Programmstruktur in die richtige Reihenfolge (i.d.R. zum Kompilieren) umsortiert
- Abschnittsweise Dokumentation beeinflusst Programmierstil
  - möglichst klare Abgrenzung des Codes in Blöcke
  - klarer Programmstrukturbaum
  - klare Unterscheidung Block-lokaler und Block-übergreifender Vereinbarungen, ...
- Nachträgliches Anwenden auf ein Programm bedeutet Code-Umbau oder kein Ausschöpfen des Literate Programming
- Nachteil: Anwendung von Literate Programming und Begleit-Werkzeugen (z.B. LaTeX) bedeutet neue Einstiegshürde
- Zudem möchten manche Autoren vielleicht nur den Quellcode des Programms, nicht jedoch der Dokumentation offen legen
  - z.B. Wissenschaftler mit Veröffentlichungsdruck in kommerziellen/unfreien Medien, Plagiatserschwerung
- Der Mehraufwand beträgt grob 50–100%

4 `noweb`

- Literate Programming Werkzeug `noweb` ist absichtlich einfach:
  - wenig Quellcode
  - wenige Befehle
  - kein Pretty-Printing (d.h. graphische Differenzierung von Programmelementen in Dokumentation wie bei (`C`)`WEB`)

  → schnell zu erlernen
- Dokumentationssprachenunabhängig (hauptsächlich LaTeX, manchmal HTML)
- Programmiersprachenabhängig, auch mehrere parallel
- Im wesentlichen über Skripte `noweave` und `notangle` zu benutzen die gemeinsames `noweb`-Dokument (typisch: `.nw`) in mehreren Schritten mit ausführbaren Programmen verschiedener Sprachen bearbeiten
  - Modularität durch Option die Zwischenergebnisse mit separaten Standard- oder eigenen „Filtern“ zu bearbeiten
- `noweb` ist freie und offene Software
- ... aber selbst (inkl. vorhandener Optionen) leider nicht gut dokumentiert, etwa mit Literate Programming

4.1 `noweave`

- ... erzeugt Dokumentation inklusive eingebetteten Code-Abschnitten („chunks“) in unveränderter Abfolge formatiert
  - Eingabe aus mehreren Einzeldateien möglich (z.B. LaTeX-Befehl `\input`)
  - genau eine Ausgabe auf Standardausgabe (normalerweise in Datei umgeleitet)
- Doku-chunks ohne/mit Start eines neuen Absatzes werden mit `@␣`/`@`<*newline*> begonnen (Doku-/Code-chunks werden durch Beginn eines neuen chunks beendet)
- Automatisches Verzeichnis der Code-chunks in LaTeX/HTML mit `\nowebchunks` bzw. `<nowebchunks>`
- Verzeichnis von Programmobjekten (Variablen, Sprungmarken,...) in LaTeX/HTML mit `\nowebindex` bzw. `<nowebindex>`
  - Automatische Indexerstellung mit zusätzlichen (Standard-) Filtern (für C z.B. `noweave -autodefs c -index`...) möglich
  - händische Aufnahme in Index mit `@ %def` ...
- Code-Zitate in Dokumentationstext mit `[[ ... ]]`

4.2 notangle

- `notangle -Rxyz` ... extrahiert Code-chunk `xyz` und alle zitierten/in Code-chunk-Strukturbaum untergeordneten und fügt die äußeren Blätter in zitierter Reihenfolge aneinander
  - Wieder Eingabe aus mehreren Einzeldateien möglich (z.B. in C mit `#include`) und genau eine Ausgabe auf Standardausgabe
  - Aber: mehrere „root-chunks“ (d.h. Wurzel eines Code-chunk-Baums, selbst unzitiert) möglich, auch in verschiedenen Programmiersprachen
- Code-chunks werden mit `<<`*chunk-Name*`>>=` definiert und mit `<<`*chunk-Name*`>>` in anderen Code-chunks zitiert/eingefügt
- Mehrfache Definitionen desselben Code-chunks werden im Ausgabe-Quelltext aneinandergehängt und in Dokumentation als Fortsetzung markiert
- Einrückungen zitierter Code-chunks bleiben erhalten (d.h. gehen auf Inhalt über) → extrahierter Programmcode ist lesbar, auch geeignet für Sprachen mit syntaktisch bedeutsamen Einrückungen (anders als (`C`)`WEB`)

5.1 noweb und LaTeX

- Kombination mit LaTeX als Dokumentationssprache ideal weil LaTeX in reinem Textformat in die `noweb`-Datei einzubinden ist, sehr ansehnliche Ergebnisse liefert und mit den üblichen LaTeX-Paketen anpassbar bleibt → Standardeinstellung und meist verwandt unter `noweave`
- LaTeX-Paket `noweb.sty` wird mit `noweb` geliefert: `\usepackage{noweb}`, `\noweboptions{smallcode,...}` (`\usepackage[smallcode,...]{noweb}` geht nicht)
- Automatischen Standardwrapper für LaTeX (`\documentstyle{...} \usepackage{noweb} \begin{document}` ... `end{document}`) am besten mit `noweave -delay ...` abwählen und individuelle LaTeX-Präambel schreiben
- Mit `noweave` extrahierte `.tex` -Datei weist keine Zeilendifferenzen zur `.nw` -Datei auf → LaTeX-Fehlermeldungen unverändert nutzbar
- `noweb.sty` tut sich mit Seitenumbruch etwas schwer: versucht Code-chunks nicht umzubrechen und unmittelbar vorangehenden Kommentar möglichst auf dieselbe Seite zu bringen

5.2 ... und gcc

- Mit `notangle -L`*Format* ... werden extrahiertem Quellcode Präprozessordirektiven eingefügt mit Information über ursprüngliche Zeilennummer in `.nw`-Datei; ohne *Format*-Angabe standardmäßig für den C-Präprozessor, über *Format* anpassbar an Übersetzer/Präprozessoren anderer Programmiersprachen → Fehlermeldungen von gcc geben die ursprünglichen Zeilen an
- Damit kann Übersetzungs-/Fehlersuch-/Editier-Zyklus mit üblichen Werkzeugen weitergehen; Editiert wird direkt in `.nw`-Datei (z.B. QuickFix-Modus von Vim)

5.3 ... und Vim

- Es gibt ein „syntax“ script für `noweb`-Dateien sowie eine umfangreiche „LaTeX-Suite“

5.4 ... und make

- Jedes `noweb`-Dokument könnte Code-chunk `<<makefile>>=`... enthalten um das Extrahieren und Formatieren/Übersetzen von Dokumentation und Programm(en) zu vereinfachen
- Wäre schöne Konvention; Mit festem Befehl (z.B. `notangle -t -Rmakefile % > makefile` im Vim) könnte jeder mit wenigen Tastendrücken das `makefile` und damit die gewünschten Dateien (`dvi-`, `pdf-`, Programmcode-, ...) erzeugen

6 Anmerkungen zum Beispielprogramm `simCIR`

- Vorabbemerkung: Warum ist `simCIR` in C und nicht in ein paar Zeilen R geschrieben?
  - Wollte auch den mathematischen Hintergrund verstehen
  - Funktionssicherheit: Befehl `R <- Rst[,-1]` aus dem R-Code im Anhang mehr durch Versuch und Zufall gefunden, auch nachträglich keine klare Dokumentation gefunden → Zufalls-Nebeneffekt eines Befehls?, Funktion in anderen (zukünftigen) R-Versionen?
    - Gerade mit (Pseudo-)Zufallszahlen ist jederzeit korrekte Funktion schwierig festzustellen
  - Fähigkeit von R extrem große Ausgabedateien zu erzeugen war vorab zweifelhaft
  - pnorm() aufrufen ist bequem, aber was passiert dann? Keine Aussage in R-Doku über Verfahren und damit keine Idee von Qualität/Nachteilen der Näherung → man müsste in Quellcode sehen, recherchieren und verstehen kann länger dauern als neu programmieren (s.o.)
  - Performance

  Fazit: R ist ein tolles Werkzeug, aber leider nicht so toll dokumentiert, z.B. mit Literate Programming
- `simCIR` ist Beispiel für ein nachträglich mit `noweb` dokumentiertes Programm. Dabei veränderte sich das Programm erheblich
  - Klarere Struktur im Hinblick auf Vermittelbarkeit
  - Neue Variablennamen in Übereinstimmung mit üblichen Bezeichnungen in Formeln der Dokumentation
  - Unbedeutende Variablen (Schleifenzähler) ohne Wechselwirkung mit anderen Code-chunks zur Verdeutlichung so lokal wie möglich definiert
- Freiheiten von Literate Programming/`noweb` wurden für eigenen Stil genutzt/ausprobiert
  - Zur gewählten Erklärungstiefe
    - Ich wollte ein Programm dokumentieren, keinen Aufsatz über die Simulation von Marktgrößen und keinen Programmierleitfaden für C, R, make ... schreiben.
    - Ich wollte meinen Rechenweg ohne zeitaufwändige Literaturrecherche direkt nachvollziehbar machen
  - Gewählte „Erzähl“-Reihenfolge: Programmüberblick, dann zentraler/namensgebender Teil, dann Nebenrechnungen und schließlich nötige Initialisierungen/Aufräumarbeiten/Bibliotheken; Bewusst abweichend von Rechenreihenfolge
  - Programmelemente (Variablen, Unterprogramm) werden im Code möglichst direkt nach ihrer Einführung/Erklärung im Text definiert/deklariert
  - Unbedeutende Variablen (s.o.) sind nicht in den händisch erstellten Index aufgenommen
  - Im Anhang wird zur Funktionsprüfung des Programms mit Beispieldaten und R animiert
- Ich habe mich auch verändert
  - Neues gelernt über `noweb`, LaTeX, C/gcc, R, make, man, Vim
  - Erst bei der Erklärung der Box-Müller-Methode habe ich sie wirklich verstanden, vorher hatte ich nur geglaubt verstanden zu haben
- Es gibt immer noch etwas zu verbessern: Funktion, Stil, `noweb` → Suchaufgabe

Viel Spaß beim Lesen und Ausprobieren. Über reichlich Reaktionen würde ich mich freuen.

7 Weiterführende Literatur

noweb http://www.cs.tufts.edu/~nr/noweb/

LaTeX DANTE, Deutschsprachige Anwendervereinigung TeXe.V., http://www.dante.de

C/gcc http://www.gnu.org/software/gnu-c-manual/, .../software/gcc/

R http://www.gnu.org/software/r/, http://www.r-project.org

make http://www.gnu.org/software/make/

man man(1), man(7), http://www.tldp.org/manpages/man.html, http://kernel.org/doc/man-pages/, http://www.infodrom.org/projects/manpages-de/, http://www.schweikhardt.net/man_page_howto.html

vim http://www.vim.org/, .../scripts/ → nach „noweb“/„latex-suite“ suchen