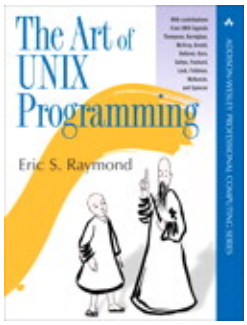
Buchbesprechung: The Art of UNIX Programming



by Edgar Hernández Zúñiga
<edgar(en)linuxfocus.org>

About the author:

Ich habe keine Biografie, auch keine ganz kleine ...

Translated to English by:
Edgar Hernández Zúñiga
<edgar(en)linuxfocus.org>

Abstract:

Diese Buchbesprechung legt den Schwerpunkt auf die Haupt−Themen des Buches. Wenn Sie diesen Text lesen, wird das Buch in den Geschäften erhältlich sein. Diese Buchbesprechung basiert auf Version 0.87 des Buches, einer Vorveröffentlichung, die uns vor der Publikation zur Evaluierung zur Verfügung gestellt wurde. Beim Schreiben dieses Artikels habe ich erkannt, wie wichtig das Thema des Buches ist. "The Art of UNIX Programming" verdient eigentlich einen eigenen Artikel. Das Buch ist sehr gut geschrieben und Sie werden sehen, dass Eric weiss, worüber er spricht.

Besprochenes Buch:	The Art of UNIX Programming.
Autor(en):	Eric S. Raymond.
Beiträge von:	Thompson, Kernighan, McIlroy, Arnold, Bellovin, Korn, Gettys, Packard, Lesk, Feldman, McKusick, Spencer.
Seiten:	550 (in dieser Version).
Verleger:	Addison Wesley (http://www.awprofessional.com)

Einführung

Eric S. Raymond, bekannt für seine Veröffentlichung "The Cathedral and the Bazaar", hat ein außergewöhnliches Buch geschrieben. Dieses Buch gibt uns einen Einblick in die Techniken und Elemente des Unix−Systems.

Das Werk von Eric S. Raymond wurde von anderen bekannten Namen aus der Unix−Welt unterstützt, die

Beiträge zu diesem Buch geleistet haben, unter ihnen finden sich Ken Thompson, Brian Kernighan und Dennis Ritchie.

Das Buch ist in vier Hauptteile gegliedert:

- Context
- Design
- Implementation/Tools
- Community

Jeder dieser Teile enthält verschiedene Themen, die von den *Basics of the Unix Philosophy* zu *Best Practices for Working with Open−Source developers* reichen und dann Konzepte wie *Modularity, Design of protocols for applications, Transparency, Mini−languages and Complexity* umfassen. Es geht weiter mit *Languages and Tools* im Teil, der sich mit der Implementation befasst. Zusätzlich finden Sie praktische Beispiele, die dem Leser ein besseres Verständnis des Themas ermöglichen.

Nachstehend finden Sie das Inhaltsverzeichnis des Buches. Es gibt nur einen kurzen Überblick, kann aber trotzdem nützlich sein, um eine Vorstellung über die in diesem Buch behandelten Themen zu bekommen.

Inhaltsverzeichnis

Coding for modularity.

5. Textuality.

The Importance of Being Textual.
Data file meta-formats.
Application protocol design.
Application protocol meta-formats.

6. Transparency.

Some case studies.
Designing for transparency and discoverability.
Designing for maintainability.

7. Multiprogramming.

Separating complexity control from performance tuning.
Taxonomy of Unix IPC methods.
Problems and methods to avoid.
Process partitioning at the design level.

8. Minilanguages.

Taxonomy of languages.
Applying mini-languages.
Designing mini-languages.

9. Transformation.

Data-driven programming.
Ad-hoc code generation.

10. Configuration.

What should be configurable?
Where configurations live.
Run-control files.
Environment variables.
Command-line options.
How to choose among configuration-setting methods.
On breaking these rules.

11. Interfaces.

Applying the Rule of Least Surprise.
History of interface design on Unix.
Evaluating interface designs.
Tradeoffs between CLI and visual interfaces.
Transparency, expressiveness, and configurability.
Unix interface design patterns.
Applying Unix interface-design patterns.
The Web browser as universal front end.
Silence is golden.

12. Optimization.

Don't just do something, stand there!
Measure before optimizing.
Non-locality considered harmful.
Throughput vs. latency.

13. Complexity.

Speaking of complexity.
A Tale of Five Editors.
The right size for an editor.
The right size of software.

III. IMPLEMENTATION.

14. Languages.

Unix's Cornucopia of Languages.
Why Not C?
Interpreted Languages and Mixed Strategies.
Language evaluations.
Trends for the Future.
Choosing an X toolkit.

15. Tools.

A developer-friendly operating system.
Choosing an editor.
Special-purpose code generators.
Make in non-C/C++ Development.
Version-control systems.
Run-time debugging.
Profiling.
Emacs as the universal front end.

16. Re-Use.

The tale of J. Random Newbie.
Transparency as the key to re-use.
From re-use to open source.
The best things in life are open.
Where should I look?
What are the issues in using open-source software?
Licensing issues.

IV. COMMUNITY.

17. Portability.

Evolution of C.
Unix standards.
Specifications as DNA, code as RNA.
Programming for Portability.
Internationalization.
Portability, open standards and open source.

18. Documentation.

Documentation concepts.
The Unix style.
The zoo of Unix documentation formats.
The present chaos and a possible way out.
The DocBook tool chain.
How to write Unix documentation.

19. Open Source.

Unix and open source.
Best practices for working with open-source developers.
The logic of licenses: how to pick one.
Why you should use a standard license.
Varieties of Open-Source Licensing.

20. Futures.

```
    Essence and accident in Unix tradition.
    Problems in the design of Unix.
    Problems in the environment of Unix.
    Problems in the culture of Unix.
    Reasons to believe.
```

A. Glossary of Abbreviations.

B. References.

C. Contributors.

Kultur und die Unix–Philosophie

Für Menschen mit Erfahrung in der Unix–Welt oder in anderen, von Unix abgeleiteten Betriebssystemen, ist es nicht ungewöhnlich, den Begriff *Philosophie* zu hören. Unix ist eine Philosophie, neben einer Kultur ist es ein Lebensstil, eine Art, Dinge zu tun, zu strukturieren und zu programmieren.

Unix basiert auf einer mächtigen Philosophie und einem Design, die es ihm seit seiner Geburt in 1969 erlaubt haben, als Referenz für die Entwicklung einer großen Anzahl von Betriebssystemen zu dienen.

Grundlagen der Unix–Philosophie

Die Unix–Philosophie startete offensichtlich mit Ken Thompson, als dieser ein kleines, aber sehr fähiges Betriebssystem entwerfen wollte. Danach haben viele Menschen dazu beigetragen und Unix enthält heute die Erfahrungen verschiedenster Quellen.

Das Buch enthält eine Anzahl wichtiger Themen, deren Studium sich lohnt. Ein Teil, den ich besonders nützlich fand, ist der Abschnitt, der die Leserin dazu ermutigt, das Design vor dem Start zu abstrahieren und über die Grundideen nachzudenken, welche die Unix–Philosophie ausmachen:

- *Modularität:* Schreiben Sie einfache Teile, die über saubere Schnittstellen miteinander verbunden sind.
- *Zusammensetzung:* Gestalten Sie Programme für die Verbindung mit anderen Programmen.
- *Wirtschaftlichkeit:* Programmiererzeit ist teuer; gehen Sie im Vergleich zur Maschinenzeit sparsam damit um.
- *Optimierung:* Prototyp vor Verschönerung. Stellen Sie vor Optimierungen die Funktionsfähigkeit sicher.
- *Erweiterbarkeit:* Entwickeln Sie für die Zukunft, weil sie schneller kommt, als Sie annehmen.

Schlußfolgerung und Empfehlungen

Es ist, wie ich definitiv sagen kann, ein exzellentes Buch. Eric Steven Raymond lehrt uns, gute Software zu schreiben und über die Konzepte des Designs nachzudenken, bevor man anfängt. Dieses Buch sollte für alle diejenigen Programmierer sehr interessant sein, die bereits Software in C, C++, Java usw. schreiben, aber von anderen Nicht−Unix−Systemen kommen.

Es enthält eine sehr ausführliche Bibliografie, die Ihnen die Erweiterung Ihres Horizonts ermöglicht. Ich persönlich war sehr erfreut, auch Referenzen zu Dokumentation über Versions−Steuerungssysteme zu finden.

Ich hoffe, einen Artikel geschrieben zu haben, der eine tiefere Analyse des Buches darstellt, und würde mich freuen, wenn Ihnen diese kurze Einführung gefallen hat.

2005−01−11, generated by lfparser_pdf version 2.51