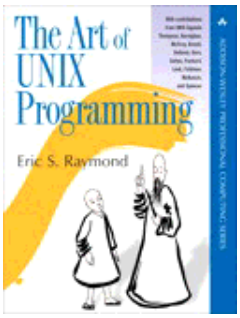
LinuxFocus article number 313
http://linuxfocus.org

# Book review: The Art of UNIX Programming




by Edgar Hernández Zúñiga

<edgar(en)linuxfocus.org>

*About the author:*

I don’t have a biography, not a small biography...

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

*Abstract*:

This bibliographic review intends to provide a clear and concise perspective of the key topics included in the book which will probably be offered for sale at the time you read this article.

This review is based on version 0.87 of the book, a text that we received just for evaluation before its publication.

While I was writing this article I realized that the subject of the book was so wide and the way it is presented was so valuable that perhaps we should include another article to provide an in deepth evaluation of the topics of this book.

| | |
|---|---|
| Reviewed title: | The Art of UNIX Programming. |
| Author(s): | Eric S. Raymond. |
| Contributions: | Thompson, Kernighan, McIlroy, Arnold, Bellovin, Korn, Gettys, Packard, Lesk, Feldman, McKusick, Spencer. |
| Pages: | 550 in this version. |
| Publisher: | Addison Wesley ( http://www.awprofessional.com ) |

## Introduction

Eric S. Raymond, widely known by The Cathedral and the Bazaar, did an exceptional work. This book

will be published soon but it is also possible to find it on line. In fact, this great compilation based on many hours of research and processes lets us have a general idea about the many different technologies and elements related to Unix systems as well as some of their results.

Eric S. Raymonds work is supported by the contribution of many important co-workers, such as Ken Thompson, Brian Kernighan and Dennis Ritchie. Remarkably the author admits that the first one and the latter one encouraged him to do this work.

The book is divided in four main parts:

- Context
- Design
- Implementation/Tools
- Community
-

  Each part contains different topics that include from *Basics of the Unix philosophy*, under the chapter Context, to *Best Practices for Working with Open-Source developers*, as well as concepts of *Modularity, Design protocols for Applications, Transparency, Minilanguages and Complexity* under the chapter Design, and *Languages and Tools* under Implementation. Furthermore illustrative, practical examples are included whenever they are necessary to provide the reader a better comprehension.

  To my regret, as I prefer offering meaningful reviews of books rather than mentioning their contents and adding just some brief comments, I decided to include the general index of this book. I thought that the readers would find it useful.

# Table of contents

The open-source movement: 1998 and onward.
The lessons of Unix history.

**3. Contrasts.**
The elements of operating-system style.
Operating-system comparisons.
What goes around, comes around.

**II. DESIGN.**

**4. Modularity.**
Encapsulation and optimal module size.
Compactness and orthogonality.
Libraries.
Unix and object-oriented languages.
Coding for modularity.

**5. Textuality.**
The Importance of Being Textual.
Data file metaformats.
Application protocol design.
Application protocol metaformats.

**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 minilanguages.
Designing minilanguages.

**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 toolchain.
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.**

# Unix Culture and Philosophy

As I mentioned before, apart from giving a personal opinion on this book, I think that it is also advantageous considering a systematic analysis that lets us provide not only the contents of this book but

also a clear idea about it for those readers who will not be able to read it all.

For those readers who have already gone through the Unix world as well as some operating systems related to it, the expression *Philosophy* will not be odd. Unix is not only a philosophy but a culture, a life style and a way of doing things. It is, for instance, an approach to a completely different method of programming.

Unix is based on a powerful philosophy of design which, from its starting point in 1969, let it be a reference for the creation of further operating systems.

## Basics of Unix Philosophy

Unix philosophy, undoubtly first brought into existence by Ken Thompson, who was interested in creating a simple but also highly competent operating system, and its lessons learned and provided by different sources, cannot be considered a formal method of design but a philosophy based on experience.

This book will provide the reader with great topics to study. One of its most advantageous parts is that one where we are encouraged to leave our thoughts aside and just keep in mind the following basic ideas which are part of Unix philosophy:

- *Modularity:* Write simple parts connected by clean interfaces.
- *Composition:* Design programs to be connected to other programs.
- *Economy:* Programmer time is expensive; conserve it an preference to machine time.
- *Optimization:* Prototype before polishing. Get in working before you optimize it.
- *Extensibility:* Design for the future, because it will be here soones than you think.

## Conclusion and recommendations

Undoubtly, this is an excellent book. Eric Steven Raymond has done a great compilation in a truly good style added to the references acknowledged, such as Unix Programming Environment by Kernighan and Pike, Unix Philosophy by Gancarz and The Pragmatic Programmer by Hunt and Thomas, among others. All this makes its reading necessary.

The book intends to help those programmers who are just beginners to acquire the necessary experience to improve themselves but will also be useful for programmers who are not familiar with Unix but have fair knowledge of languages such as C, C++ and Java.

The book includes an extensive bibliography for those readers who might be interested in gaining more

skills. I personally enjoyed reading the topic about standards for the creation of documentation and version control systems.

Hopefully I will be able to write an article that allows us to know much more about this book. For the time being, I hope you have enjoyed this brief introduction.

---



2005-01-14, generated by lfparser_pdf version 2.51