

Przedmowa (13)

Część I Kontekst (19)

Rozdział 1. Filozofia: filozofia ma znaczenie (21)

- 1.1. Kultura? Jaka kultura? (21)
- 1.2. Trwałość Uniksa (22)
- 1.3. Argumenty przeciwko nauce kultury uniksowej (23)
- 1.4. Co w Uniksie jest złe? (24)
- 1.5. Co w Uniksie jest dobre? (25)
 - 1.5.1. Oprogramowanie o otwartych źródłach (25)
 - 1.5.2. Międzyplatformowa przenośność i otwarte standardy (25)
 - 1.5.3. Internet i World Wide Web (26)
 - 1.5.4. Społeczność Open Source (26)
 - 1.5.5. Prawdziwa elastyczność (27)
 - 1.5.6. Programowanie Uniksa jest przyjemne (27)
 - 1.5.7. Doświadczenie zdobyte w Uniksie można zastosować gdzie indziej (28)
- 1.6. Podstawy filozofii uniksowej (29)
 - 1.6.1. Reguła modularności: pisz proste części połączone przejrzystymi interfejsami (31)
 - 1.6.2. Reguła przejrzystości: przejrzystość jest lepsza niż spryt (32)
 - 1.6.3. Reguła kompozycji: projektuj programy tak, aby dało się je łączyć z innymi (32)
 - 1.6.4. Reguła oddzielania: oddzielaj politykę od mechanizmu; oddzielaj interfejsy od głównej części programu (33)
 - 1.6.5. Reguła prostoty: projektuj pod kątem prostoty; uciekaj się do złożoności tylko tam, gdzie to konieczne (34)
 - 1.6.6. Reguła powściągliwości: pisz duży program tylko wtedy, gdy zostanie jasno udowodnione, że nie da się inaczej (35)
 - 1.6.7. Reguła przezroczystości: dbaj o zrozumiałość kodu, aby ułatwić badanie i debugowanie programów (35)
 - 1.6.8. Reguła odporności: odporność jest pochodną przezroczystości i prostoty (36)
 - 1.6.9. Reguła reprezentacji: przełóż wiedzę na dane, aby logika programu mogła być prosta i odporna (37)
 - 1.6.10. Reguła najmniejszego zaskoczenia: projektując interfejs, zawsze postępuj w najmniej zaskakujący sposób (37)
 - 1.6.11. Reguła milczenia: kiedy program nie ma nic nieoczekiwanego do powiedzenia, nie powinien mówić nic (38)
 - 1.6.12. Reguła naprawy: naprawiaj, co się da, ale jeśli program musi zawieść, niech zawiedzie z hukiem i jak najszybciej (38)
 - 1.6.13. Reguła ekonomii: czas programisty jest drogi; oszczędzaj go zamiast czasu komputera (39)
 - 1.6.14. Reguła generacji: unikaj programowania ręcznego; jeśli to możliwe, pisz programy piszące programy (40)
 - 1.6.15. Reguła optymalizacji: napisz prototyp, zanim zaczniesz dopracowywać program. Sprawdź, czy działa, zanim zaczniesz go optymalizować (40)
 - 1.6.16. Reguła różnorodności: nie ufaj żadnym deklaracjom o "jedynym słusznym sposobie" (42)
 - 1.6.17. Reguła rozszerzalności: projektuj programy z myślą o przyszłości, bo nadejdzie ona wcześniej, niż się spodziewasz (42)

- 1.7. Filozofia uniksowa w jednej lekcji (43)
- 1.8. Stosowanie filozofii uniksowej (43)
- 1.9. Liczy się też nastawienie (44)

Rozdział 2. Historia: opowieść o dwóch kulturach (45)

- 2.1. Pochodzenie i historia Uniksa, lata 1969 - 1995 (45)
 - 2.1.1. Geneza, lata 1969 - 1971 (46)
 - 2.1.2. Exodus, lata 1971 - 1980 (48)
 - 2.1.3. TCP/IP i Wojny Uniksowe, lata 1980 - 1990 (50)
 - 2.1.4. Uderzenia w Imperium, lata 1991 - 1995 (56)
- 2.2. Pochodzenie i historia hakerów, lata 1961 - 1995 (58)
 - 2.2.1. Zabawa w gajach Akademii, lata 1961 - 1980 (58)
 - 2.2.2. Fuzja internetowa i ruch wolnego oprogramowania, lata 1981 - 1991 (60)
 - 2.2.3. Linux i reakcja pragmatyków, lata 1991 - 1998 (62)
- 2.3. Ruch Open Source, od roku 1998 do chwili obecnej (64)
- 2.4. Lekcje płynące z historii Uniksa (66)

Rozdział 3. Kontrasty: porównanie filozofii uniksowej z innymi (67)

- 3.1. Elementy stylu systemu operacyjnego (67)
 - 3.1.1. Jaka jest idea unifikująca system operacyjny? (68)
 - 3.1.2. Wielozadaniowość (68)
 - 3.1.3. Współpracujące procesy (69)
 - 3.1.4. Granice wewnętrzne (70)
 - 3.1.5. Atrybuty plików i struktury rekordów (71)
 - 3.1.6. Binarne formaty plików (72)
 - 3.1.7. Preferowany styl interfejsu użytkownika (72)
 - 3.1.8. Zamierzone grono odbiorców (73)
 - 3.1.9. Bariera oddzielająca użytkownika od programisty (73)
- 3.2. Porównanie systemów operacyjnych (74)
 - 3.2.1. VMS (76)
 - 3.2.2. MacOS (77)
 - 3.2.3. OS/2 (78)
 - 3.2.4. Windows NT (80)
 - 3.2.5. BeOS (83)
 - 3.2.6. MVS (85)
 - 3.2.7. VM/CMS (87)
 - 3.2.8. Linux (89)
- 3.3. Co odchodzi, to wraca (90)

Część II Projekt (93)

Rozdział 4. Modularność: czystość i prostota (95)

- 4.1. Hermetyzacja i optymalny rozmiar modułu (97)
- 4.2. Zwartość i ortogonalność (98)
 - 4.2.1. Zwartość (99)
 - 4.2.2. Ortogonalność (100)
 - 4.2.3. Reguła SPOT (102)

- 4.2.4. Zwartość i jedno silne centrum (103)
 - 4.2.5. Zalety niezaangażowania (105)
- 4.3. Oprogramowanie ma wiele warstw (105)
 - 4.3.1. Od góry w dół czy od dołu w górę? (106)
 - 4.3.2. Warstwy spajające (108)
 - 4.3.3. Studium przypadku: język C jako cienka warstwa kleju (108)
- 4.4. Biblioteki (110)
 - 4.4.1. Studium przypadku: wtyczki programu GIMP (111)
- 4.5. Unix i języki obiektowe (112)
- 4.6. Kodowanie z myślą o modularności (114)

Rozdział 5. Tekstowość: dobre protokoły to dobra praktyka (115)

- 5.1. Dlaczego tekstowość jest ważna? (117)
 - 5.1.1. Studium przypadku: format uniksowego pliku haseł (118)
 - 5.1.2. Studium przypadku: format pliku .newsrc (120)
 - 5.1.3. Studium przypadku: format pliku graficznego PNG (121)
- 5.2. Metaformaty plików danych (122)
 - 5.2.1. Styl DSV (122)
 - 5.2.2. Format RFC 822 (123)
 - 5.2.3. Format "słoika ciasteczek" (124)
 - 5.2.4. Format "słoika rekordów" (125)
 - 5.2.5. XML (126)
 - 5.2.6. Format plików INI systemu Windows (128)
 - 5.2.7. Uniksowe konwencje dotyczące formatu plików tekstowych (129)
 - 5.2.8. Zalety i wady kompresji plików (130)
- 5.3. Projektowanie protokołów aplikacyjnych (131)
 - 5.3.1. Studium przypadku: SMTP, protokół transferu poczty (132)
 - 5.3.2. Studium przypadku: POP3, protokół skrzynki pocztowej (133)
 - 5.3.3. Studium przypadku: IMAP, internetowy protokół dostępu do poczty (134)
- 5.4. Metaformaty protokołów aplikacyjnych (135)
 - 5.4.1. Klasyczny internetowy metaprotokół aplikacyjny (136)
 - 5.4.2. HTTP jako uniwersalny protokół aplikacyjny (136)
 - 5.4.3. BEEP: Blocks Extensible Exchange Protocol (138)
 - 5.4.4. XML-RPC, SOAP i Jabber (139)

Rozdział 6. Przezroczystość: niech stanie się światłość (141)

- 6.1. Studia przypadków (143)
 - 6.1.1. Studium przypadku: audacity (143)
 - 6.1.2. Studium przypadku: opcja -v programu fetchmail (144)
 - 6.1.3. Studium przypadku: GCC (146)
 - 6.1.4. Studium przypadku: kmail (147)
 - 6.1.5. Studium przypadku: SNG (148)
 - 6.1.6. Studium przypadku: baza danych terminfo (150)
 - 6.1.7. Studium przypadku: pliki danych gry Freeciv (153)
- 6.2. Projektowanie pod kątem przezroczystości i odkrywalności (154)
 - 6.2.1. Zen przezroczystości (155)
 - 6.2.2. Kodowanie pod kątem przezroczystości i odkrywalności (156)

- 6.2.3. Przezroczystość i unikanie nadopiekuńczości (157)
- 6.2.4. Przezroczystość i edytowalne reprezentacje (158)
- 6.2.5. Przezroczystość, diagnozowanie błędów i usuwanie skutków błędu (159)
- 6.3. Projektowanie pod kątem konserwowalności (160)

Rozdział 7. Wieloprogramowość: wyodrębnianie procesów w celu oddzielenia funkcji (163)

- 7.1. Oddzielanie kontroli złożoności od dostrajania wydajności (165)
- 7.2. Taksonomia uniksowych metod IPC (166)
 - 7.2.1. Przydzielanie zadań wyspecjalizowanym programom (166)
 - 7.2.2. Potoki, przekierowania i filtry (167)
 - 7.2.3. Nakładki (171)
 - 7.2.4. Nakładki zabezpieczające i łączenie Bernsteina (172)
 - 7.2.5. Procesy podrzędne (174)
 - 7.2.6. Równorzędna komunikacja międzyprocesowa (174)
- 7.3. Problemy i metody, których należy unikać (181)
 - 7.3.1. Przestarzałe uniksowe metody IPC (182)
 - 7.3.2. Zdalne wywołania procedur (183)
 - 7.3.3. Wątki - groźba czy niebezpieczeństwo? (185)
- 7.4. Dzielenie procesów na poziomie projektu (186)

Rozdział 8. Minijęzyki: jak znaleźć notację, która śpiewa (189)

- 8.1. Taksonomia języków (191)
- 8.2. Stosowanie minijęzyków (193)
 - 8.2.1. Studium przypadku: sng (193)
 - 8.2.2. Studium przypadku: wyrażenia regularne (193)
 - 8.2.3. Studium przypadku: Glade (196)
 - 8.2.4. Studium przypadku: m4 (198)
 - 8.2.5. Studium przypadku: XSLT (198)
 - 8.2.6. Studium przypadku: warsztat dokumentatora (199)
 - 8.2.7. Studium przypadku: składnia pliku kontrolnego programu fetchmail (204)
 - 8.2.8. Studium przypadku: awk (205)
 - 8.2.9. Studium przypadku: PostScript (206)
 - 8.2.10. Studium przypadku: bc i dc (207)
 - 8.2.11. Studium przypadku: Emacs Lisp (209)
 - 8.2.12. Studium przypadku: JavaScript (209)
- 8.3. Projektowanie minijęzyków (210)
 - 8.3.1. Wybór odpowiedniego poziomu złożoności (210)
 - 8.3.2. Rozszerzanie i osadzanie języków (212)
 - 8.3.3. Pisanie własnej gramatyki (213)
 - 8.3.4. Makra - strzeż się! (214)
 - 8.3.5. Język czy protokół aplikacyjny? (215)

Rozdział 9. Generacja: podwyższanie poziomu specyfikacji (217)

- 9.1. Programowanie sterowane danymi (218)
 - 9.1.1. Studium przypadku: ascii (219)

- 9.1.2. Studium przypadku: statystyczne filtrowanie spamu (220)
- 9.1.3. Studium przypadku: modyfikowanie metaklas w programie fetchmailconf (221)
- 9.2. Doraźna generacja kodu (226)
 - 9.2.1. Studium przypadku: generowanie kodu wyświetlającego tabelę znaków w programie ascii (226)
 - 9.2.2. Studium przypadku: generowanie kodu HTML na podstawie listy tabelarycznej (228)

Rozdział 10. Konfiguracja: jak zacząć od właściwej nogi (231)

- 10.1. Co powinno być konfigurowalne? (231)
- 10.2. Gdzie znajdują się dane konfiguracyjne? (233)
- 10.3. Pliki kontrolne (234)
 - 10.3.1. Studium przypadku: plik .netrc (236)
 - 10.3.2. Przenoszenie do innych systemów operacyjnych (237)
- 10.4. Zmienne środowiskowe (237)
 - 10.4.1. Systemowe zmienne środowiskowe (238)
 - 10.4.2. Zmienne środowiskowe definiowane przez użytkownika (239)
 - 10.4.3. Kiedy używać zmiennych środowiskowych? (240)
 - 10.4.4. Przenoszenie do innych systemów operacyjnych (241)
- 10.5. Opcje wiersza polecenia (241)
 - 10.5.1. Opcje wiersza polecenia od -a do -z (242)
 - 10.5.2. Przenoszenie do innych systemów operacyjnych (247)
- 10.6. Którą metodę wybrać? (247)
 - 10.6.1. Studium przypadku: fetchmail (248)
 - 10.6.2. Studium przypadku: serwer XFree86 (249)
- 10.7. O naruszaniu tych reguł (251)

Rozdział 11. Interfejsy: wzorce projektowe interfejsu użytkownika w środowisku uniksowym (253)

- 11.1. Stosowanie Reguły Najmniejszego Zaskoczenia (254)
- 11.2. Historia projektowania interfejsów w systemie Unix (256)
- 11.3. Ocena projektów interfejsów (257)
- 11.4. Różnice między CLI a interfejsami wizualnymi (259)
 - 11.4.1. Studium: dwa sposoby pisania programu kalkulatora (263)
- 11.5. Przezroczystość, wyrazistość i konfigurowalność (264)
- 11.6. Uniksowe wzorce projektowe interfejsów (266)
 - 11.6.1. Wzorzec filtra (266)
 - 11.6.2. Wzorzec cantrip (268)
 - 11.6.3. Wzorzec źródła (source) (269)
 - 11.6.4. Wzorzec drewna (sink) (269)
 - 11.6.5. Wzorzec kompilatora (269)
 - 11.6.6. Wzorzec ed (270)
 - 11.6.7. Wzorzec roguelike (271)
 - 11.6.8. Wzorzec "rozdzielenia mechanizmu od interfejsu" (273)
 - 11.6.9. Wzorzec serwera CLI (278)
 - 11.6.10. Wzorce interfejsów oparte na językach (279)
- 11.7. Stosowanie uniksowych wzorców projektowania interfejsów (280)

- 11.7.1. Wzorzec programu poliwalencyjnego (wielowartościowego) (281)
- 11.8. Przeglądarka internetowa i uniwersalny Front End (282)
- 11.9. Milczenie jest złotem (284)

Rozdział 12. Optymalizacja (287)

- 12.1. Jeżeli masz zrobić cokolwiek, lepiej nie rób nic (287)
- 12.2. Zmierz przed optymalizacją (288)
- 12.3. Nielokalność bywa szkodliwa (290)
- 12.4. Przepustowość i opóźnienia (291)
 - 12.4.1. Grupowanie operacji (292)
 - 12.4.2. Nakładające się operacje (293)
 - 12.4.3. Buforowanie wyników operacji (293)

Rozdział 13. Złożoność: tak prosto, jak tylko można, ale nie prościej (295)

- 13.1. Mówiąc o złożoności (295)
 - 13.1.1. Trzy źródła złożoności (296)
 - 13.1.2. Wybór między złożonością interfejsu a złożonością implementacji (298)
 - 13.1.3. Złożoność niezbędna, opcjonalna i przypadkowa (299)
 - 13.1.4. Mapowanie złożoności (300)
 - 13.1.5. Gdy prostota nie wystarcza (301)
- 13.2. Opowieść o pięciu edytorach (302)
 - 13.2.1. ed (303)
 - 13.2.2. vi (304)
 - 13.2.3. Sam (305)
 - 13.2.4. Emacs (306)
 - 13.2.5. Wily (307)
- 13.3. Właściwy rozmiar edytora (308)
 - 13.3.1. Identyfikowanie problemów ze złożonością (308)
 - 13.3.2. Nici z kompromisu (312)
 - 13.3.3. Czy Emacs jest argumentem przeciwko tradycji Uniksa? (313)
- 13.4. Właściwy rozmiar programu (315)

Część III Implementacja (317)

Rozdział 14. Języki: w C albo nie w C? (319)

- 14.1. Uniksowy róg obfitości języków (319)
- 14.2. Dlaczego nie C? (320)
- 14.3. Języki interpretowane i strategie mieszane (322)
- 14.4. Ocena języków (323)
 - 14.4.1. C (323)
 - 14.4.2. C++ (325)
 - 14.4.3. Powłoka (327)
 - 14.4.4. Perl (330)
 - 14.4.5. Tcl (332)
 - 14.4.6. Python (334)
 - 14.4.7. Java (338)
 - 14.4.8. Emacs Lisp (341)

- 14.5. Trendy na przyszłość (342)
- 14.6. Wybór biblioteki systemu X (344)

Rozdział 15. Narzędzia: taktyki rozwoju (347)

- 15.1. System operacyjny przyjazny dla programisty (347)
- 15.2. Wybór edytora (348)
 - 15.2.1. Co należy wiedzieć o vi (349)
 - 15.2.2. Co należy wiedzieć o Emacsie (349)
 - 15.2.3. Wybór przeciw religii: używaj obu (350)
- 15.3. Generatory kodu do zadań specjalnych (351)
 - 15.3.1. yacc i lex (351)
 - 15.3.2. Studium: Glade (354)
- 15.4. make: automatyzacja przepisów (355)
 - 15.4.1. Podstawowa teoria make (355)
 - 15.4.2. Make w językach innych niż C i C++ (357)
 - 15.4.3. Produkcje użytkowe (357)
 - 15.4.4. Generowanie plików makefile (359)
- 15.5. Systemy kontroli wersji (362)
 - 15.5.1. Po co kontrolować wersje? (362)
 - 15.5.2. Ręczna kontrola wersji (363)
 - 15.5.3. Automatyczna kontrola wersji (363)
 - 15.5.4. Uniksowe narzędzia kontroli wersji (364)
- 15.6. Debugowanie w czasie działania programu (367)
- 15.7. Profilowanie (368)
- 15.8. Łączenie narzędzi z Emacsem (368)
 - 15.8.1. Emacs i make (369)
 - 15.8.2. Emacs i debugowanie w czasie działania programu (369)
 - 15.8.3. Emacs i kontrola wersji (370)
 - 15.8.4. Emacs i profilowanie (370)
 - 15.8.5. Jak IDE, ale lepsze (371)

Rozdział 16. Ponowne wykorzystanie: nie wyważajmy otwartych drzwi (373)

- 16.1. Opowieść o Janie Nowicjuszu (374)
- 16.2. Przewidywalność jako klucz do ponownego użycia kodu (377)
- 16.3. Od ponownego wykorzystania do otwartych źródeł (378)
- 16.4. Najlepsze rzeczy w życiu są otwarte (380)
- 16.5. Gdzie szukać? (382)
- 16.6. Kwestie związane z używaniem otwartego oprogramowania (383)
- 16.7. Licencje (384)
 - 16.7.1. Co można uznać za otwarte oprogramowanie (385)
 - 16.7.2. Standardowe licencje otwartego oprogramowania (386)
 - 16.7.3. Kiedy potrzebny jest prawnik? (388)

Część IV Społeczność (391)

Rozdział 17. Przenośność: przenośność oprogramowania i utrzymywanie standardów (393)

- 17.1. Ewolucja języka C (394)

- 17.1.1. Wczesna historia języka C (395)
 - 17.1.2. Standardy języka C (396)
- 17.2. Standardy Uniksa (398)
 - 17.2.1. Standardy i Wojny Uniksów (398)
 - 17.2.2. Duch na uczcie zwycięstwa (401)
 - 17.2.3. Standardy Uniksa w świecie otwartych źródeł (402)
- 17.3. IETF i Proces Standaryzacji RFC (403)
- 17.4. Specyfikacja to DNA, kod to RNA (406)
- 17.5. Programowanie ukierunkowane na przenośność (408)
 - 17.5.1. Przenośność i wybór języka (409)
 - 17.5.2. Omijanie zależności od systemu (412)
 - 17.5.3. Narzędzia umożliwiające przenośność (413)
- 17.6. Internacjonalizacja (413)
- 17.7. Przenośność, otwarte standardy i otwarte źródła (414)

Rozdział 18. Dokumentacja: objaśnianie kodu w świecie WWW (417)

- 18.1. Koncepcje dokumentacji (418)
- 18.2. Styl Uniksa (420)
 - 18.2.1. Skłonność do wielkich dokumentów (420)
 - 18.2.2. Styl kulturowy (421)
- 18.3. Zwierzyniec uniksowych formatów dokumentacji (422)
 - 18.3.1. troff i narzędzia z Warsztatu Dokumentatora (422)
 - 18.3.2. TEX (424)
 - 18.3.3. Texinfo (425)
 - 18.3.4. POD (425)
 - 18.3.5. HTML (425)
 - 18.3.6. DocBook (426)
- 18.4. Istniejący chaos i możliwe rozwiązania (426)
- 18.5. DocBook (427)
 - 18.5.1. Definicje typu dokumentu (427)
 - 18.5.2. Inne definicje DTD (428)
 - 18.5.3. Łańcuch narzędzi DocBook (429)
 - 18.5.4. Narzędzia do migracji (431)
 - 18.5.5. Narzędzia do edycji (432)
 - 18.5.6. Pokrewne standardy i praktyki (432)
 - 18.5.7. SGML (433)
 - 18.5.8. Bibliografia formatu XML-DocBook (433)
- 18.6. Najlepsze praktyki pisania dokumentacji uniksowej (434)

Rozdział 19. Otwarte źródła: programowanie w nowej społeczności Uniksa (437)

- 19.1. Unix i otwarte źródła (438)
- 19.2. Najlepsze metody pracy z twórcami otwartego oprogramowania (440)
 - 19.2.1. Dobre praktyki korygowania programów (440)
 - 19.2.2. Dobre praktyki nazywania projektów i archiwów (444)
 - 19.2.3. Dobre praktyki rozwoju projektu (447)
 - 19.2.4. Dobre praktyki tworzenia dystrybucji (450)
 - 19.2.5. Dobre praktyki komunikacji (454)
- 19.3. Logika licencji: jak wybrać (456)

- 19.4. Dlaczego należy stosować standardowe licencje (457)
- 19.5. Zróżnicowanie licencji otwartego źródła (457)
 - 19.5.1. Licencja MIT lub X Consortium (457)
 - 19.5.2. Klasyczna licencja BSD (458)
 - 19.5.3. Licencja Artistic (458)
 - 19.5.4. Licencja GPL (458)
 - 19.5.5. Licencja Mozilla Public License (459)

Rozdział 20. Przyszłość: zagrożenia i możliwości (461)

- 20.1. Zasadność i przypadki w tradycji Uniksa (461)
- 20.2. Plan 9: tak wyglądała przyszłość (464)
- 20.3. Problemy w konstrukcji Uniksa (466)
 - 20.3.1. Plik w Uniksie jest tylko wielkim workiem bajtów (466)
 - 20.3.2. Unix słabo obsługuje graficzne interfejsy użytkownika (468)
 - 20.3.3. Plik usunięty na zawsze (469)
 - 20.3.4. Unix zakłada istnienie statycznego systemu plików (469)
 - 20.3.5. Projekt kontroli zadań jest pełnym nieporozumieniem (469)
 - 20.3.6. API Uniksa nie stosuje wyjątków (470)
 - 20.3.7. Za wywołania ioctl(2) i fcntl(2) należy się wstydzić (471)
 - 20.3.8. Model bezpieczeństwa w Uniksie może być zbyt prosty (472)
 - 20.3.9. W Uniksie jest zbyt wiele różnych rodzajów nazw (472)
 - 20.3.10. Systemy plików można by uznać za szkodliwe (472)
 - 20.3.11. W kierunku Globalnej Przestrzeni Adresowej Internetu (473)
- 20.4. Problemy w środowisku Uniksa (473)
- 20.5. Problemy w kulturze Uniksa (475)
- 20.6. Źródła nadziei (478)

Dodatki (479)

Dodatek A Słownik skrótów (481)

Dodatek B Bibliografia (485)

Dodatek C Współpracownicy (495)

Dodatek D Korzeń bez korzenia: uniksowe koany Mistrza Foo (497)

Skorowidz (505)