JAIST Repository
https://dspace.jaist.ac.jp/

Title	プロジェクトの公開データを用いたオープンソースソフトウェア開発への参加支援
Author(s)	藤田，充典
Citation	
Issue Date	2004-06
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/958
Rights	
Description	Supervisor:落水 浩一郎，情報科学研究科，博士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

博 士 論 文

プロジェクトの公開データを用いた
オープンソースソフトウェア開発への参加支援

指導教官　落水 浩一郎 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

藤田 充典

2004 年 6 月

Copyright © 2004 by Mitsunori Fujita

要 旨

本論文は、オープンソースソフトウェア開発 (OSSD) プロジェクトが提供している公開情報からプロジェクトの新規参加者に有用な情報を抽出して提供する手法を提案する。プロジェクトの公開情報とは、OSSD において一般的に利用されている、版管理システム、バグデータベース、メーリングリストから得られる情報を指す。新規参加者に有用な情報とは、プロジェクトに参加する際の障壁となる「プロジェクトの状況把握が難しい」、「実践的なプログラミング技術が要求される」という、2 つの問題を解決するために役立つ情報を指す。

本論文で述べる手法は、前者の問題に対して、版管理システムから得られる作業履歴を基に、新規参加者が必要とするプロジェクトの現状を反映する情報を提供する。後者の問題に対しては、バグデータベースから得られる既に解決された問題報告と、版管理システムから得られる問題解決の際の作業履歴を基に、実践的なプログラミング技術を学習するための演習問題を提供する。

本手法を用いることで、新規参加者はプロジェクトが蓄積した膨大な情報の中から必要な情報を捜し出すコストを削減し、大勢の開発者達によって洗練された問題の修正方法を学習できる。また、本手法はプロジェクトの開発スタイルに手を加える必要が無く、一般的な OSSD が提供している公開情報を用いるため、多くの OSSD プロジェクトに適用することが可能である。

目 次

1 はじめに 1

1.1 研究の背景 1

1.2 研究の目的 3

1.2.1 OSSD をサポートする有効性 3

1.3 本研究のアプローチ 6

1.4 本論文の構成 8

2 議論 9

2.1 OSSD に関する研究 9

2.2 CVS の情報を利用するツール 10

2.3 プログラミング演習問題作成技術 10

3 作業履歴の抽出／分析によるプロジェクト情報の獲得 12

3.1 OSSD への参加の際に求められる情報 12

3.2 プロジェクト情報を獲得する必要性 14

3.3 新規参加者に求められる情報 17

3.4 公開情報からのプロジェクト情報の抽出 19

3.5 プロジェクト情報を獲得可能な作業空間 (エディタ) 20

3.5.1 プロジェクト情報分析ツール: CVSLogMonitor 20

3.5.2 CVSLogMonitor を起動可能なエディタ 25

3.6 評価実験 25

3.6.1 Gimp プロジェクトの概要 26

3.6.2 実験 1: 開発者の活動の様子 27

3.6.3 実験 2: 新規参加者に求められる情報の抽出 28

3.7 実験結果の考察 30

3.8 実験のまとめ 31
3.8.1 今後の展望 31

4 問題報告を利用した実践的プログラミング演習問題作成手法 32
4.1 高度なプログラミング技術の習得 32
4.1.1 従来の学習方法 32
4.1.2 プログラミング演習問題による学習 33
4.2 公開情報を利用した演習問題の作成方法 34
4.2.1 プログラミング技術の習得に適した問題 35
4.2.2 バグデータベースの問題報告 36
4.2.3 演習問題構成アルゴリズム 37
4.2.4 期待できる効果 38
4.2.5 プログラミング技術学習に関する研究の現状 39
4.3 プログラミング演習問題集「ひよこ」システム 40
4.4 演習問題の評価 41
4.4.1 演習問題の作成 42
4.4.2 Apache の問題報告 (PR) の抽出 42
4.4.3 作成した問題の例 43
4.4.4 演習問題作成のコスト 44
4.4.5 演習問題の評価実験 47
4.4.6 実験結果の考察 50
4.5 実験のまとめ 50
4.5.1 今後の展望 50

5 おわりに 53
5.1 本研究のまとめ 53
5.2 今後の展望 54

謝辞 56

参考文献 57

本研究に関する発表論文 **65**

A オープンソースソフトウェア **68**

A.1 オープンソースソフトウェアとは 68

A.2 オープンソースソフトウェアの普及 71

A.3 オープンソースソフトウェア開発をサポートすることの有効性 . . . 72

A.4 オープンソースソフトウェア開発の特徴 73

A.5 オープンソースソフトウェア開発プロジェクトの公開情報 74

B CVS の log コマンドの出力 **83**

C 演習問題作成に関する Apache の問題報告の評価 **87**

C.1 問題報告の評価 87

図 目 次

1.1 公開情報を用いた新規参加者サポートの概念図 7

3.1 メーリングリストのスレッド表示例 15
3.2 新規参加者に求められる情報と抽出手順 17
3.3 CVSLogMonitor の実行画面 21
3.4 CVS リポジトリの履歴情報を利用した情報抽出 22
3.5 HTML 形式の出力 23
3.6 ファイルに関する月ごとの更新の様子 24
3.7 CVSLogMonitor を起動する Emacs-Lisp 25
3.8 開発者ごとの更新量の違い 28
3.9 版管理システムから得られる更新時に残されたコメント 29

4.1 問題作成手法の概念図 34
4.2 ひよこ の演習問題 40
4.3 問題の例 (PR812) 43
4.4 修正差分から抽出した解答例 45
4.5 提出された修正差分 48
4.6 自動生成した演習問題の初期状態 51

A.1 ソフトウェアの分類 70
A.2 BitMover がホスティングするプロジェクト 78
A.3 Mailman のメーリングリストアーカイブ 80
A.4 Bugzilla の検索画面 82

B.1 CVS の log コマンドの出力 (前半部分) 84
B.2 CVS の log コマンドの出力 (後半部分) 85

表 目 次

3.1 同時に更新されたファイル数 29

4.1 公開情報から得られる情報 35
4.2 Apache の問題報告で用いられているタグ 36

A.1 主要な版管理システム 77
A.2 各バグデータベースの特長 81

C.1 問題報告の評価 1 88
C.2 問題報告の評価 2 92

第 1章

はじめに

ソフトウェアのソースコードを公開して開発を進めるオープンソースソフトウェア開発 (OSSD) と呼ばれる開発手法がある。従来のプロプライエタリソフトウェアの開発ではソースコードを外部に公開せず、ソフトウェアを特徴づける機能の実現方法を隠すことが常であった。一方、OSSD では従来のソフトウェア開発手法と比べて比較的オープンな方針で開発が進められる。オープンソースソフトェア (OSS) の開発者達はソースコードを公開し、ユーザが自身の修正や改良をソフトウェアに施せるようにしている。また、開発者達はユーザからの問題報告や機能要求を奨励し、ユーザが様々な形で開発に関わることを呼びかけている。

本論文では、この新たな OSS 開発者を増やすために、OSSD を行っている開発者集団 (OSSD プロジェクト) が公開している情報から、新規に参加する者 (新規参加者) にとって有用となるプロジェクトに関する情報を獲得/提供する手法について述べる。

1.1 研究の背景

従来、開発の途中で人員を増やすことは開発が遅れる原因とされてきたが、OSSD では新たな開発者を取り込むことで発展を続けている。この「ユーザを新たな開発者として OSSD に取り込む活動」が OSS 発展の原動力となっている。これに関して、Raymond は `The Cathedral and the Bazaar`[59] の中で Linux が開発者を集めながら発展してきた理由として以下のものを挙げている:

- ユーザを大事な資産として扱うこと
- 早めのリリース、頻繁なリリースを行い、そしてユーザの声を聞くこと
- 賢いデータ構造を持つこと

また、國領は Linux のようなオープンアーキテクチャの協働モデルが成り立つ条件として以下のものを挙げている [34]:

- 多くの人が無償で共有されることが望ましいと思う対象であること
- 作業が片手間で行えるように高度なモジュール化が施されていること

実際、Linux や Apache、Mozilla といった有名な OSSD プロジェクトはこれらの特徴を備えている。これらのプロジェクトでは膨れ上がるソフトウェアを個々のモジュールに切り分け、それらを開発経験を積んだ優れた開発者達に管理させることで並行して開発を進めている。他にも、開発者達が開発に取り組むモチベーションの高さや予め備えている技術力の高さ、時間的制約の少なさ、作業内容の重複を認める開発スタイルなどが従来の開発手法との違いとして挙げられる。筆者は特に開発者達のモチベーションの高さに注目し、新規参加者達の参加に対するモチベーションを維持させることが OSS 発展の重要な課題だと考えている。OSS が提供する機能に不満を持ったユーザが、障害報告やコードの改良など様々な形で、モチベーションを下げないまま手軽にプロジェクトへ貢献できることが望ましい。

しかし、現実には OSS の機能に不満を持ったユーザ全てが OSSD に貢献する訳ではない。特に新規参加者にとっては、プロジェクトへ参加する際にモチベーションの低下につながる障壁が存在する。例えば、プロジェクトに参加して OSSD を進めるためには、マスターリポジトリへのアクセス権限の取得や、ソフトウェアへの変更を取り込んでもらうための手続きを調べる必要がある。また、OSSD ではソフトウェアを構成する個々のモジュール及びソフトウェア自体の構成管理を行うコアメンバを中心として開発/デバッグ/テストが並行して繰り返し行われるため、新規参加者がソフトウェアの構造及び、開発の現状を把握する作業を困難にする。これらの作業の手間が障壁となって新規参加者のモチベーションを下げ、OSSD 開発者の増加を妨げていると考えられる。

以下では、この問題を解決するために OSSD を行っている開発者集団 (OSSD プロジェクト) が公開している情報から、新規に OSSD へ参加する際に必要な情報を獲得/提供する手法について述べる。

1.2 研究の目的

本論文における研究の目的は、情報社会において欠かせない存在となった OSS の開発を進める OSSD プロジェクトの参加者を増やすこと及び OSS 開発者を育てるためのサポートツールを開発することである。具体的には、OSSD の開発に新規に参加する際に必要となる情報を提供し、OSSD プロジェクトが抱える問題の一つである、新規に参加する際の障壁を下げることである。

1.2.1 OSSD をサポートする有効性

OSSD をサポートする有効性には以下のものが挙げられる[1]:

- 質の高いソフトウェアが増加する
- ソフトウェア技術者の不足を解消する
- ソフトウェアに対する正しい理解を広める
- OSS 導入の動きを助ける

現在の OSSD が抱えている問題を以下に示す:

1. プロジェクトに関する情報獲得が難しい
   多くの OSSD プロジェクトは独自の開発スタイルで進められているため、プロジェクトに関する調査が必要になるが、OSSD では蓄積される情報量が莫

[1]詳細は付録 A に示す

大になり、知りたいと望む情報を得るのが難しい。また、プロジェクトが提供するドキュメントはソフトウェアの更新に追いつかず、現状を反映していない場合がある [70, 29]。必要な情報を容易に獲得する手段が求められる。

2. OSSD を進めて行くための技術習得が困難
   OSSD に参加して多くの優秀な開発者達と共に問題を解決していくためには、高いプログラミング能力が求められる。また、実践的なプログラミング能力だけではなく、リーディング能力や、他人のスタイル [15, 79] に合わせてプログラミングを行う技術も求められる。

3. ソフトウェアの構成を把握する難しさ
   たとえソースコードが公開されていても、巨大なソフトウェアの構成を理解することは非常に困難である [82]。

筆者はここで示した問題の中で 1 と 2 が新規参加者にとって特に問題になると考えている。その理由を以下に示す:

- 1 の問題について
  OSSD プロジェクトの発展に伴って利用者と開発者の人数が増えることで、プロジェクトの内外で交わされるコミュニケーション量が増大する。また、プロジェクトのコアメンバ達が決定するソフトウェアの開発方針や、プロジェクトの運営方針が変更される場合がある。これらはプロジェクトに関する情報 (プロジェクト情報) としてドキュメントやメーリングリストのアーカイブとして管理される。しかし、プロジェクトの規模に応じて蓄積される情報は莫大なものになる。そのため利用者が求める情報を探すことが困難になる。また、管理者が情報の正しさ／新しさを確認することが困難になる。

  新規参加者はプロジェクト情報を正しく把握し、自身の貢献がプロジェクトの方針に沿うかどうかを確認できることが望ましい。そのため、新規参加者がプロジェクトに参加する際に必要な情報を抽出／提示することで新規参加者の手間を減らし、モチベーションを下げずに開発に参加させる支援が望まれる。また、多くのプロジェクトに対応するために、軽量で汎用的な情報抽出手法を用いることが望ましい。ここでの軽量とは、プロジェクトに関して

あらゆる情報を抽出するのではなく、新規参加者が OSSD プロジェクトへ参加する「きっかけ」となる程度の情報を抽出することを指す。

- 2 の問題について

  OSS のソースコードはユーザからの貢献とプロジェクトに参加する多数の開発者達によって更新され続けている。更新を繰り返すことによって追加／洗練されたソースコードは、たとえ可読性が考慮されていても見る者が容易に把握できるとは限らない。そのため、開発に参加する際には複雑なソースコードを読み取る (リーディング) 能力が要求される。また、プロジェクトに参加して共同開発を進めていくためには、ソースコード間の依存関係を把握した上で、既存のソースコードに対して悪影響を与えないコーディング技術が求められる。更には、セキュリティ・ホールや脆弱性を極力埋め込まない技術と知識を身に付けることも求められる。

  OSSD プロジェクトについて、新規参加者自身が開発を行う際に求められる能力を把握し、それらを効果的に身に付ける手段が望まれる。

- 3 の問題について

  巨大なソフトウェアの構成把握をサポートするために、多くの IDE (Integrated Development Environment) や Web 上で利用可能な依存解析／情報視覚化ツールが開発されている [20, 73, 50]。しかし、これらのツールの多くは計算コストの高い静的な解析結果を利用するため、提供される情報が更新に追いつかず現状を反映していない場合がある。そのため新規参加者は自分の望む変更に対して良いアドバイスを得られる現役開発者、もしくは現状を反映した情報源を探すことが望ましい。

  ソフトウェアの構成把握をサポートすることは重要な課題であるが、解析コストが高く、サポート可能なプロジェクトが限られるため本論文では対象としない。

以降、本論文では OSSD を支援するために以下の 2 つの問題を扱う:

1. プロジェクトに関する情報獲得が難しい
2. OSSD を進めて行くための技術習得が困難

これらの問題を解決するアプローチについて次節で述べる。

1.3 本研究のアプローチ

様々な OSSD プロジェクトを対象に新規参加者をサポートするために、以下の特徴を備えた手法が望まれる:

- 多くのプロジェクトに適用可能な汎用性を持つ
  現状の OSSD プロジェクトの開発プロセスに手を加えることなく、各プロジェクトが提供する一般的な情報から新規参加者に有用な情報を抽出する。

- 情報の抽出及び提供を自動化できる
  獲得可能な情報から自動化に適した情報を選択して抽出する手順、それらを用いて必要な情報を構成する手順、収集した情報を提供する手法を確立する。

そこで本研究では、OSSD プロジェクトで一般的に利用されているシステムに着目し、アクセスが容易で書式が定まったシステムの出力及びシステムが自動的に記録するログ情報を利用した。以後、これらの OSSD プロジェクトから得られる情報を公開情報 (Public Data) と呼ぶ。

公開情報からプロジェクトに関する情報を獲得して分析を行うことで開発者達に有用な情報を提供する概念図を図 1.1 に示す。この手法は OSSD プロジェクトの開発スタイルに何ら変更を加えることなく、各プロジェクトに有用な情報を提供することを目的としている。

ここでの公開情報にはソフトウェアのソースコードの他に、ソフトウェアの機能やプロジェクトの目的等を紹介する Web ページ、開発過程の作業履歴が記録される版管理システム (Version Control System)、開発者及びユーザの主要なコミュニケーション手段であるメーリングリストのアーカイブ (ML)、修正すべき既知のソフトウェアに関する問題が登録されたバグデータベース (BUGDB) などから得られる情報が含まれる。これらの公開情報は、プロジェクトの Web ページ上などで公開され、誰でも自由にアクセスして情報を獲得することができる。

公開情報に蓄積された開発の作業履歴や、メーリングリストを介して行われるコミュニケーションの履歴は、プロジェクト及びソフトウェアが辿ってきた発展

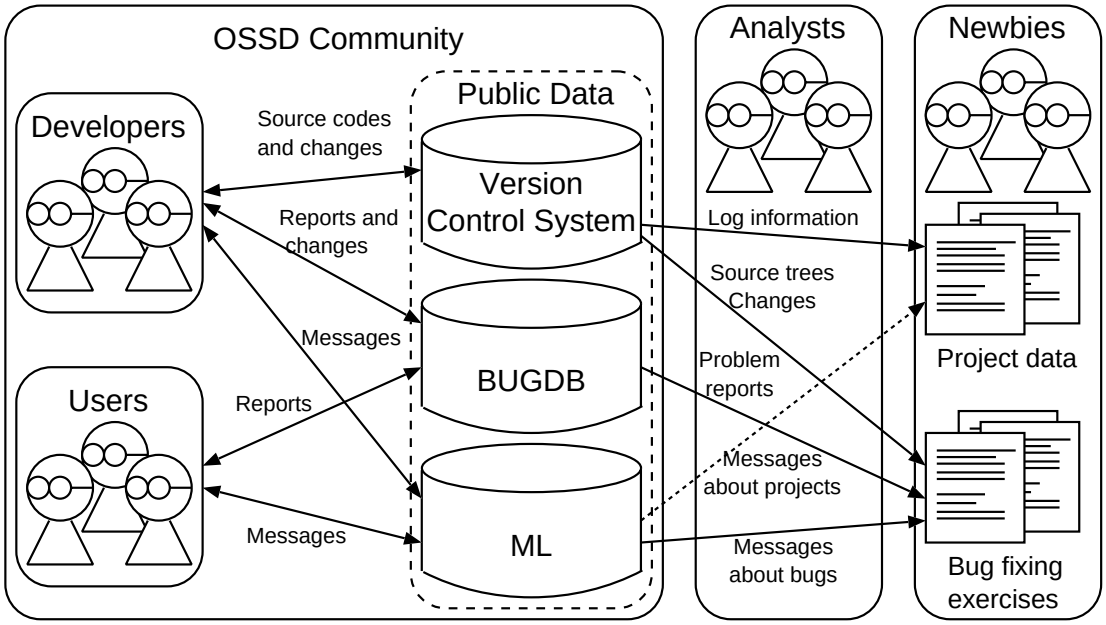


図 1.1: 公開情報を用いた新規参加者サポートの概念図

の過程を記録している。これらの情報を獲得して分析を行うことで新規参加者達に有用な情報を抽出できると考えている。

しかし、メーリングリストのアーカイブに対する自然言語解析はコストが高いため、プロジェクト毎に解析を行い続けることは現実的でない。そこで本研究では、書式の定まった出力を容易に得られる版管理システムとバグデータベースに焦点を当て、メーリングリストについては文字列検索及び、版管理システムとバグデータベースの出力に含まれる更新日時を基にした参照に留めている。

先に示した新規参加者の障壁となる 2 つの問題を解決するために以下のことを行う:

1. 版管理システムから得られる作業履歴から抽出可能なプロジェクト情報を提供する

   作業履歴に残された更新内容を基に、各更新の行われた日時や更新の頻度、同時に更新されているファイル、更新の担当者などを自動的に抽出／解析してプロジェクト情報を生成する。

2. バグデータベースから得られる既に解決された問題報告と版管理システムか

ら得られる修正内容から、実践的なプログラミング技術を学習するための演習問題を作成して提供する

問題報告の内容からバグの詳細や再現方法、修正が行われた期間を自動的に抽出して演習問題を生成し、該当する期間の修正内容を版管理システムから獲得することで解答例を生成する。

1.4 本論文の構成

本論文は、5 つの章から構成される。

まず、第 2 章では、OSSD と開発支援ツール、演習問題作成に関する関連研究について議論する。次に、第 3 章では、OSSD プロジェクトに関するプロジェクト固有の情報を獲得する手法の説明と具体例を示す。そして、第 4 章では、実際に OSSD で生じたバグを利用する演習問題作成手法を示し、演習問題の管理、提供手段を紹介する。最後に、第 5 章において本論文のまとめと残された課題および今後の展望を述べる。

第 2章

議論

本章では本研究を進めるにあたって調査した OSSD に関する研究と他の研究者によって行われている版管理システム CVS から得られる情報を利用したツール、演習問題作成技術に関する研究について述べる。

2.1 OSSD に関する研究

Raymond の `The Cathedral and the Bazaar`[59] によって OSSD の持つ特異なプロセスが研究者の関心を集め、いくつかの論文が発表されている。しかし、それらの多くは特定の OSSD プロジェクトの特長やプロセスを客観的に述べているに過ぎない [11, 46]。McConnell は `Open-Source Methodology: Ready for Prime Time?`[39] で OSSD に関して次のように述べている:

> ”オープンソース方法論の決定的な描写はまだない。Raymond の The Cathedral and the Bazaar が最も良い唯一の描写であるがスケッチに過ぎない。”

多くの OSSD プロジェクトは、リーダーやコアメンバ達によって決定された独自の開発スタイルで進められるため、一括りに扱うことはできない。多くの OSSD をサポートするには特定の OSS に縛られない汎用的な手法が望まれる。本論文で述べたアプローチは、OSSD プロジェクトの開発スタイルに依存せずに一般的な公開情報からプロジェクト固有の情報を抽出するため、様々なプロジェクトに対応できる。

本研究と同様に OSSD プロジェクトの公開情報に着目した研究がいくつか行われている。Mockus は Apache プロジェクトを対象としてプロジェクトの採用している開発プロセスを紹介した上で、公開情報から得られる情報を基に OSSD と従来の開発スタイルとの比較を行った [44]。また、Mockus は調査対象に Mozilla を加えた同様の研究も発表している [45]。Koch は Gnome プロジェクトを対象として、公開情報であるメーリングリストと CVS から得られる情報を基に、更新が行われた行数やプロジェクトの参加人数等の時間軸上の変化や関係をグラフ化することで OSSD の持つ特徴を示した [33]。しかし、彼等の研究は特定の OSSD プロジェクトを対象とした個々のプロジェクトの特徴の調査に留まっている。本研究は OSSD のサポートを目的として、そこで役立つ情報の調査と抽出を行っている。

2.2 CVS の情報を利用するツール

CVS を利用した OSSD をサポートするツールとして、CVS のリポジトリから得られる情報を視覚化する CVSWeb[83] やリポジトリに取り込まれたソフトウェアの状態を視覚化する Tinderbox[57] が知られている。これらは多くの OSSD プロジェクトで利用されているが、リポジトリの情報を CGI によって直接利用するためサーバ側で設定／運用を行う必要がある。

筆者はより多くの OSS を対象とするために、3.5.1 節で紹介する CVSLogMonitor を開発した。CVSLogMonitor は CVS のリポジトリさえ公開されていればプロジェクト情報の分析に必要な情報を獲得することができる。

2.3 プログラミング演習問題作成技術

半自動化可能なプログラミング演習問題作成技術について、柏原らによるプログラム空欄補充問題の作成 [31] に関する研究がある。彼等は空欄補充問題がアルゴリズム理解に貢献するという見地から、Program Dependence Graph を用いることで、C 言語で書かれたアルゴリズムの内容に立ち入らずに要所を決定し、空欄補充問題を作成可能にしている。彼等の手法を OSS に適用可能であれば、ソフトウェアの構造上の要点を扱う問題が作成できる筈である。しかし、この手法で

は作成可能な問題の種類が空欄補充問題に限られるため、実践的な演習問題作成には向かないと考えている。

また、中村 [49] は、プログラミング言語の教材について分析を行い、効果的な教材作成に必要な要件を明らかにする研究を行った。中村は 12 冊の C 言語の教科書について、教授系列や取り扱っている概念が各々異なることを示した。また、C 言語を教授するためのコンセプトマップ [65] が作成者[1]ごとに異なることを示すことで、" C 言語においては、単一の「よい」教授系列が存在しない" と結論づけている。そのため、プログラミング言語の学習教材については、教授者及び学習者ごとの学習方針を採用できることが望ましいと考えている。4.3 節で述べるプログラミング演習問題集「ひよこ」システムでは、問題作成と管理の手順を明らかにすることで演習問題を募り、カテゴリに分けて提示することで利用者ごとの異なる学習方針に応えることができる。

[1]C 言語に熟達した学生 7 名が作成

第 3章

作業履歴の抽出／分析によるプロジェクト情報の獲得

この章では、新規参加者が必要とするプロジェクト情報の説明とその抽出方法について述べる。

3.1 OSSD への参加の際に求められる情報

OSSD プロジェクトの 1 つである Mozilla プロジェクトでは、版管理システムの書き込み権限を求める参加者に対して以下のものを求めている [56]：

- 作業中のコードや、その作業で影響を受けるかもしれない他のコードに関する能力
- チェックイン、ビルド、および他のプロセスに関する能力
- 基本的で質の高いコーディングの経験

これらはソフトウェア開発を続けることで養われるものであるが、プロジェクト固有の前提となる知識や技術を獲得して磨いていくために、筆者は先人の作業に習うことが有効な方法だと考えている。そこで新規参加者にはプロジェクトに関する以下の情報が必要になると考える:

- 参加手順または参加手順に関する最新の議論
  OSSD プロジェクトはソースコードを公開しているだけでなく、ユーザや新

規参加者にとってもオープンな対応を行っている。プロジェクトによっては開発者及びユーザの目的に応じた複数のメーリングリストや特定の議論を専門に行う手段 (ニュースグループ、掲示板、IRC) を提供している。これらのコミュニケーション手段は多くの場合、プロジェクトのホームページで紹介されている。しかし、ホームページの情報はプロジェクトの経過とともに増え続けるため、古いままで更新されない情報も多い。継続的にプロジェクトに関わっている開発者達はこの変化に対応できるが、ホームページの情報を信頼するしかない新規参加者は、正しい情報に到達できない、もしくは誤ったメーリングリストに用件を送信してしまう。このような関係のない大勢のメーリングリスト参加者に送られるメールは、受信者に不快な思いをさせるだけではなく、他の議論を邪魔する場合がある。

- 類似事例を検索するための情報

  OSSD は不特定多数の開発者達によって非同期に進められる分散共同開発であるため、問題の修正や新しい機能の追加を試みる際に、既に他の開発者によって重複した作業が行われている場合がある。開発の宣言をメーリングリストなどで行った後で開発を進めるスタイルもあるが、プロジェクトの様々な要件について議論が行われているメーリングリストでは、メールの山に埋もれて認知されない場合があり、作業の詳細や担当者を知ることは困難である。また、暗黙の内に外部のプロジェクトで開発が進められているものや、プロジェクトには反映されないまま開発が進められているものもある (ローカリゼーション、特殊パッチ)。さらに、類似した作業が既に行われているのであれば、保守性を維持するためにソースコードの再利用やスタイルの統一などが求められる。

- ファイル間の関連

  他のファイルに副作用のある変更を行った際には、新たな問題を生じさせないために関連するファイル (機能) についても調査し、動作を確認する必要がある。高機能な依存解析ツール [20, 50] が開発されているが依存解析ツールを度々起動していたのではコストが高い。より軽快に関連ファイルを見つける方法が望まれる。

- モジュールやファイルのオーナーまたは主な更新者
  ファイルやモジュールのオーナーが分かっていれば、開発に参加する際に次に示す効果を期待できる。(1) 自分の変更作業を反映して貰うことができる。(2) 変更に関するアドバイスを受けることができる。(3) オーナーが把握している他の作業内容との重複を避けることができる。

これらの情報を各 OSSD プロジェクト毎に Web 上で公開するシステムを構築することで、新規参加者が自ら調査を行う手間を軽減し、モチベーションを維持したまま自分に適した OSS を選択して参加するきっかけを与えられると考えている。

3.2 プロジェクト情報を獲得する必要性

ソフトウェア開発において、開発の進捗状況を把握することは重要な課題である。Apache Software Foundation[17] では、付録 A に記したツール群を用いて開発を進めている。しかし、Cubranic は現在の OSSD で一般に用いられている開発ツール群について、次のように述べている [9]:

> ”これらのツールだけではグループの意思決定、ナレッジマネジメント、タスクスケジューリング、プログレストラッキングなどのより高度なコーディネーション (higher-level coordination) のサポートがない。また、個々の開発者が遭遇している開発の進捗状況を追跡する手段がなく、設計に関するドキュメントや開発の計画などはファイルに書きなぐられる程度で、ソースツリーや Web サイトのどこかにまぎれてしまうことが多い。情報を効果的に管理し、参照できるツールが要求される。更に、メーリングリストはメールで溢れ返ることが多い。より効果的に情報量の管理と参照を行うためには、図 3.1 に示したような基本的なスレッド表示 (reply-threading) ではなく、より洗練されたツールが必要となる。”

溢れ返ったメーリングリストのメールを扱う手段としては、村越ら [48, 80, 77] の成果がある。村越らはメーリングリストで交わされる日本語による議論につい

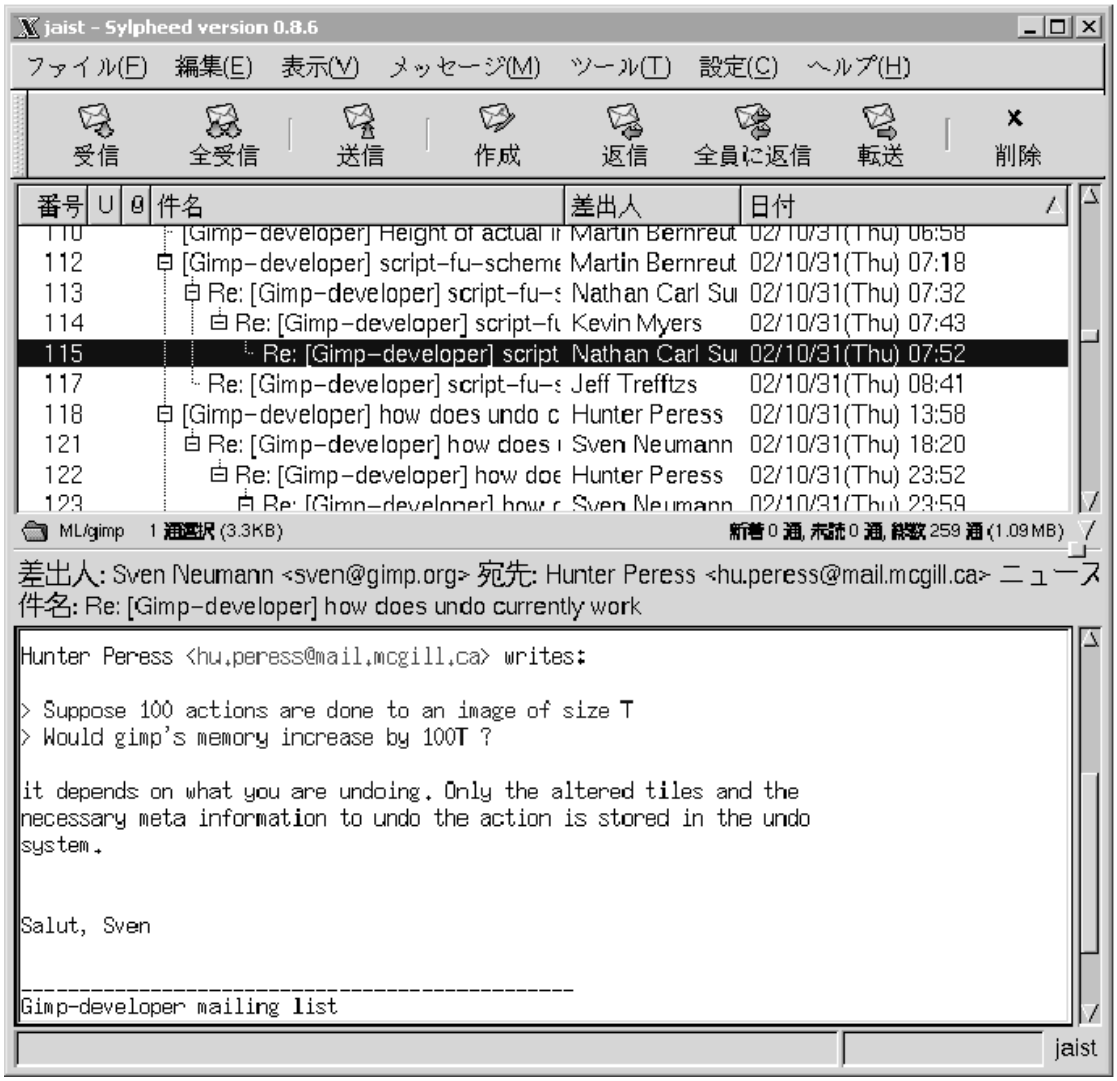


図 3.1: メーリングリストのスレッド表示例

て、「手がかり語辞書」を用いることで、 議論が終っていない未終結な討議スレッドを特定する手法及び、議論の内容を要約する手法の提案と実現を行った。終結した議論の内容を要約して提供することで情報を調べるコストを下げることが可能になり、未終結な議論を強調することで曖昧になっている情報を正すことができる。

しかし、村越らの研究は日本語に特化した「手がかり語辞書」を用いているため、英語で議論が交わされることが多い OSSD プロジェクトのメーリングリストでは対象が限られてしまう。また、解析のコストが高いため進行中の複数のプロジェクトについて現状を反映することは難しい。加えて、OSS 開発者達はパブリックなメーリングリストではなく、プライベートなコミュニケーション手段によって意志決定を行う場合がある [11]。そのためメーリングリストから得られる情報だけで全てが解決する訳ではない。

多くの OSSD プロジェクトが公開している情報は、ソフトウェアの開発に関する決定事項や情報源だけしか与えていない。このことは新規参加者の開発に参加するモチベーションを下げる障壁の 1 つとなる。ボランティアの参加と新しい血に頼る OSSD プロジェクトにおいてこの参加の障壁は深刻な問題になる [9]。

この障壁を取り除き、新規参加者が OSSD に参加／貢献できる基盤を築くために、以下のようなプロジェクトに関する情報、手順を効果的に獲得できることが必要だと考えている:

- 開発に参加するために必要な情報
- 類似事例を検索するための情報
- 関連するファイル
- モジュールやファイルのオーナーまたは主な更新者

いくつかの OSSD プロジェクトはこれらの情報をドキュメントやホームページ上に示しているが、整理・更新されていないものが多い。また、ファイル間の関連やファイルのオーナーに関する情報は、直接ソースコードを読んで把握するか、過去のメーリングリストの記事などから調べなければならない。

3.3 新規参加者に求められる情報

前節で触れた新規に OSSD プロジェクトに参加する際に望まれる情報について詳細を述べる (図 3.2)。

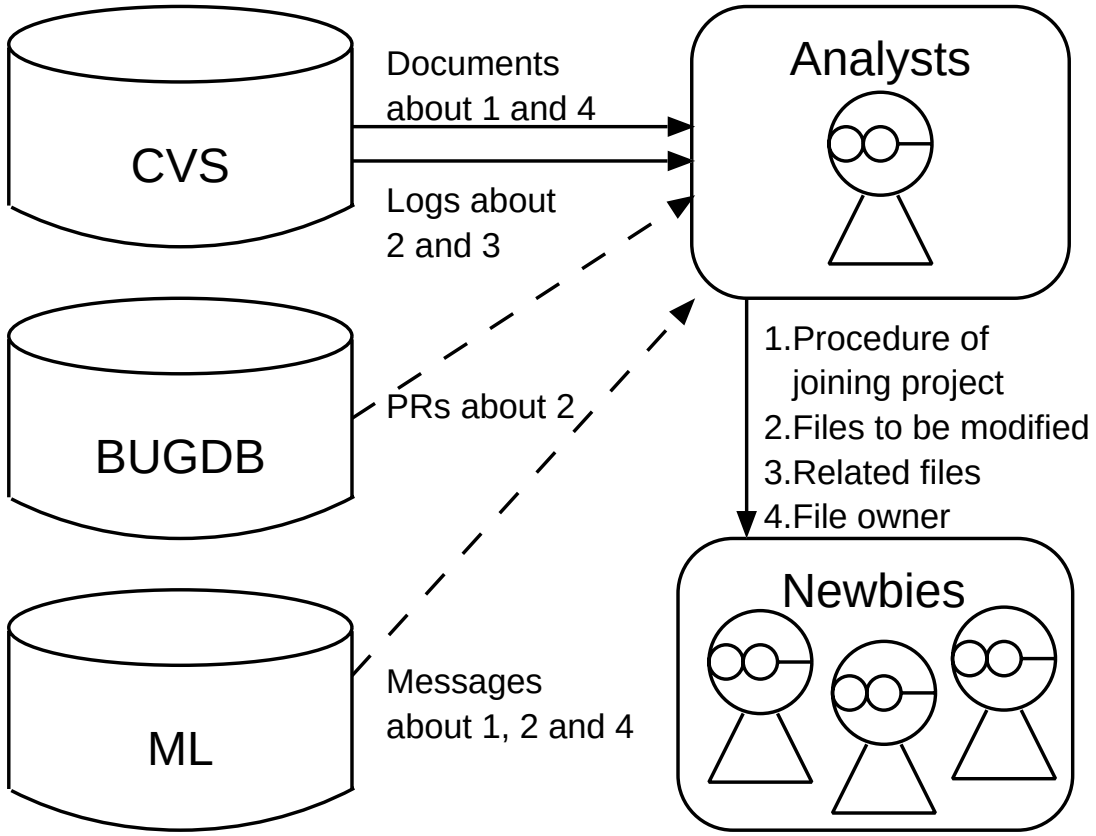


図 3.2: 新規参加者に求められる情報と抽出手順

新規参加者に求められる情報について現状の分析を以下に示す:

- 開発に参加するために必要な情報
  開発に参加するための手順や手段はプロジェクトごとにドキュメントや Web 上のデータとして公開されていることが多い。しかし、ドキュメントでは表現できない細かいニュアンスの存在や、ドキュメントの内容が必ずしも現状と一致しない場合がある。そのため、リポジトリから得られるドキュメントの最新の内容と併せて、メーリングリストで行われた「参加」に関する最近の議論を検索して提供する。

- 類似事例を検索するための情報

  数百から数万のファイルから構成される OSS では、新機能を追加するために編集するファイルを特定することや、修正すべきバグを含んだファイルを特定することが難しい。ソフトウェアの構成やファイルの命名規則、バグデータベースの出力などから容易に特定できる場合を除いて、新しい機能の実現やバグの修正を行うためには、まず、編集すべきファイルを特定する必要がある。そこで版管理システムの更新履歴とメーリングリストのアーカイブ、バグデータベースの修正履歴から過去の同様の事例について検索を行い、該当したファイルを候補として提供する。

- 関連するファイル

  複数の開発者の意図が絡み合う OSSD では、新機能の追加やバグの修正が一つのファイルで可能とは限らない。そのため機能の追加／バグの修正を行う際には関連するファイルも特定する必要がある。そこで検索されたファイルについて、リポジトリから得られる情報から同時に更新されたファイルの存在を調べる。そして、その更新内容に対するコメントのマッチングと同時に更新された回数などからファイル間の関連の強さを定め、関連の強いファイルを候補として提供する。

- モジュールやファイルのオーナー

  ファイルの更新を行うためには、まず、ファイルのオーナー、もしくはそのファイルが属するモジュールのオーナーを特定し、修正を施すためのパッチを送るか、リポジトリへの書き込み権限を得て自ら修正を加える必要がある。OSSD で作成されたファイルには、ライセンスによって守られた権利を主張するためにオーナーの名前が記述されている場合が多い。しかし、プロジェクトの途中でオーナーが代わっても情報が更新されていない場合がある。また、実質的なオーナーが異なる場合がある。そこでドキュメントから得られるオーナーに関する情報に併せて、メーリングリストでファイルの内容について触れている投稿者、該当するファイルを頻繁に更新している主な更新者を候補として提供する。

3.4 公開情報からのプロジェクト情報の抽出

本研究では公開情報から得られる作業履歴、特に版管理システム CVS (Concurrent Versions System) から獲得できる情報を主に扱っている。

公開情報から得られる情報を分析することで新規参加者に有用な情報を提供する手順を図 3.2 で示した。以下に、公開情報からプロジェクト情報を抽出する方法について述べる。

- CVS から得られる情報
  OSS の版管理システムとして一般的に用いられる CVS からは、CVS の内部コマンドを利用することでリポジトリに登録されたモジュールやファイルに関する更新の内容や、更新者、回数、日時等、様々な情報を獲得することができる。これらの情報から特定のファイルに対する更新の回数が最も多い開発者、頻繁に更新を行っている開発者について調べることで、ファイルの実質的なオーナーを調べることができる。あるファイルが他のファイルと同時に更新される回数を調べることで、統計的にそれらのファイル間の関連の強さを調べることができる。

- 開発者メーリングリストのアーカイブ
  OSSD では、開発に関する話し合いや取り決め事をメーリングリストを利用して行っている。一般には、パブリックなメーリングリストの他に、開発者専用やバグ報告専用など、複数のメーリングリストが存在する。メーリングリストのアーカイブから得られる情報には、メールのサブジェクト、投稿者、日付、内容、返信の有無などが挙げられる。これらの情報から特定のサブジェクトに関する議論が行われた期間や、その議論に参加した投稿者、議論によって導かれた結論などを調べることができる。

- バグデータベースに登録された情報
  ユーザからの問題報告はメーリングリストへの投稿の他に、一般に公開されたバグデータベースによって行われる。バグデータベースから得られる情報には、問題の登録番号、影響するモジュール、問題の状況、投稿者、日付、コメントなどが挙げられる。これらの情報からモジュールごとに問題の有無、問題の発生していた期間、発生する問題の種類などを調べることができる。

3.5 プロジェクト情報を獲得可能な作業空間 (エディタ)

研究を進めるに当り版管理システム CVS からプロジェクト情報を獲得するツールを開発し、OSS 開発に用いられるエディタとの統合を行った。

3.5.1 プロジェクト情報分析ツール: CVSLogMonitor

OSSD を進めるために、多くのプロジェクトが CVS を利用している [12] ことが知られている。そこで CVS のリポジトリから得られる作業履歴を利用することで、分析に必要なデータを大量に獲得することができると考えた。このため、作業履歴の獲得と分析作業をサポートするログ解析ツール CVSLogMonitor を開発した (図 3.3)。

CVSLogMonitor は図 3.4 に示すように CVS の *log* コマンドの出力 (付録 B) として得られる作業履歴を解析することにより、更新履歴、更新頻度、更新の関連性等の情報を抽出するツールである。CVSLogMonitor はネットワークを介して CVS のリポジトリサーバに直接接続するか、CVS の *log* コマンドの出力を保存したファイルを指定することで作業履歴を読み込むことができる。

また、このツールはリポジトリから直に作業履歴を獲得することができるため、ソースコードをチェックアウトする手間を省くことができる。獲得された作業履歴はファイルの更新単位であるリビジョン毎にパースされ、後から参照できるようにファイルにも保存される。CVSLogMonitor はこの作業履歴からファイル名や登録された日時、開発者の名前など用いて絞り込みを行い、それを出力する。表示可能なデータには以下のものがある。

- リポジトリに登録された全てのファイル名とその情報
- ファイルのリビジョンごとの情報
- リビジョンが更新された日時
- 開発者の名前 (アカウント名)
- 更新によって追加された行数／削除された行数

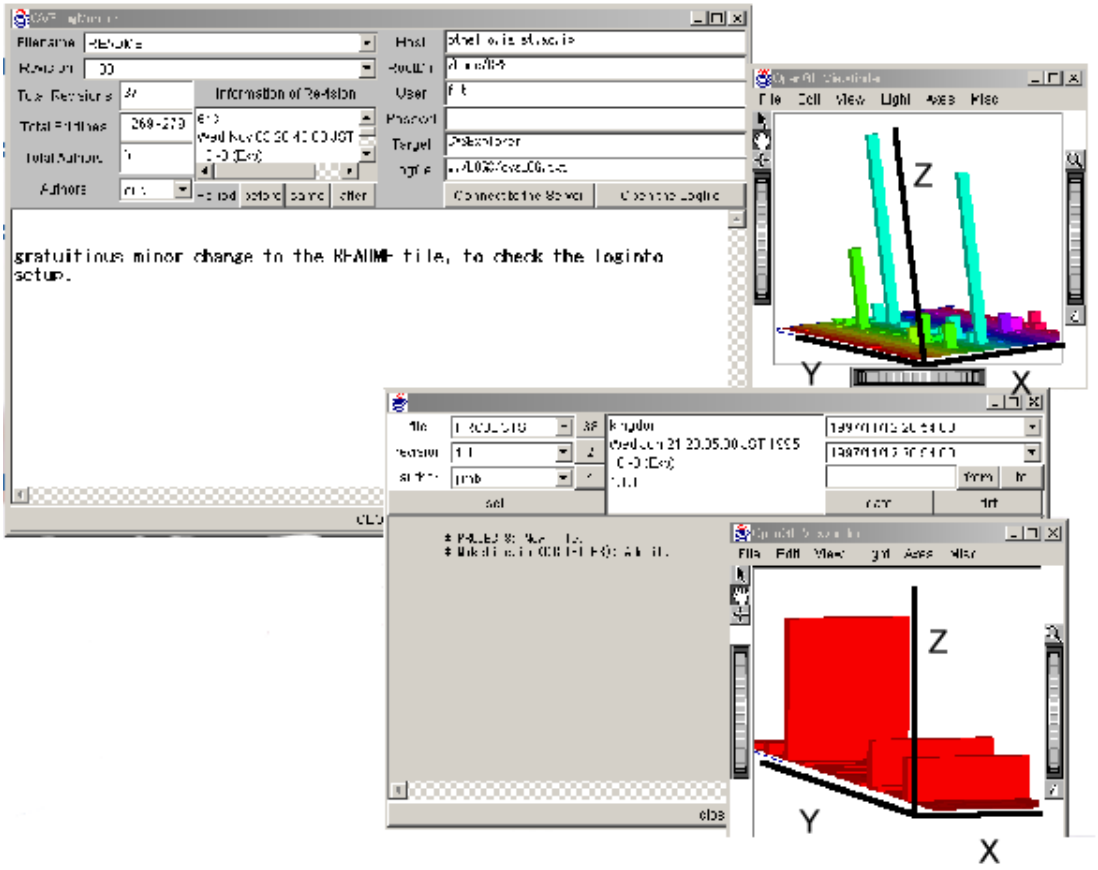


図 3.3: CVSLogMonitor の実行画面

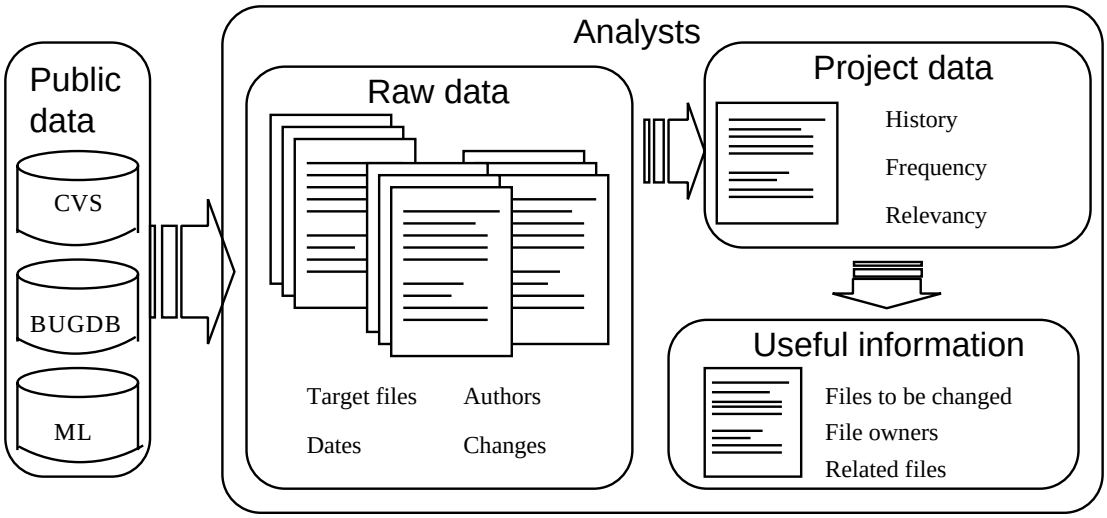


図 3.4: CVS リポジトリの履歴情報を利用した情報抽出

- リビジョンから派生しているブランチの有無
- 開発者が更新時に書き残したコメント

現実装ではファイル毎のリビジョンツリーに残された作業履歴から得た数値データを視覚化することで直観的な分析をサポートする。図 3.3 の中に示した二つのグラフでは、X 軸に開発者、Y 軸に更新を行ったファイル、Z 軸に更新を行った行数を割り当てることで各開発者について更新対象の傾向を調べることができる。上下のグラフはそれぞれ複数の開発者と、一人の開発者についての更新の様子である。CVSLogMonitor の主となる機能は、ファイルとそのリビジョンごとの変更情報を出力することである。また、その出力フォーマットとして、ファイル名や更新者、更新日時による条件選択やソートといった、作業履歴の分析に有用と考えられる機能を備えている。例えば、現行の CVS(1.11) の *log* コマンドではタグを指定して絞り込みをかける機能が提供されていないが、CVSLogMonitor ではタグのリストを利用することができる。結果出力についても、図 3.3 に示したような GUI を用いた表示だけでなく、図 3.5 のように HTML 形式で出力し、その内容を Web ブラウザで確認することができる (図 3.6)。

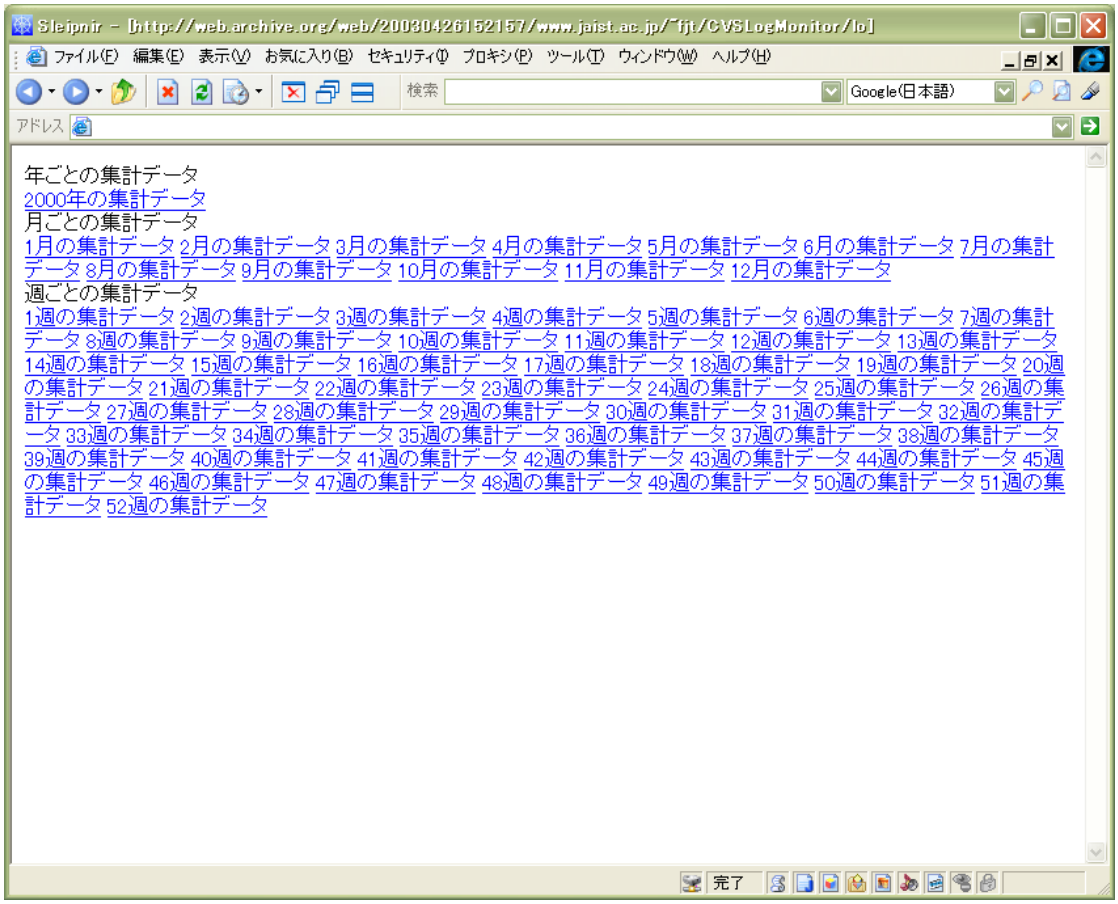


図 3.5: HTML 形式の出力

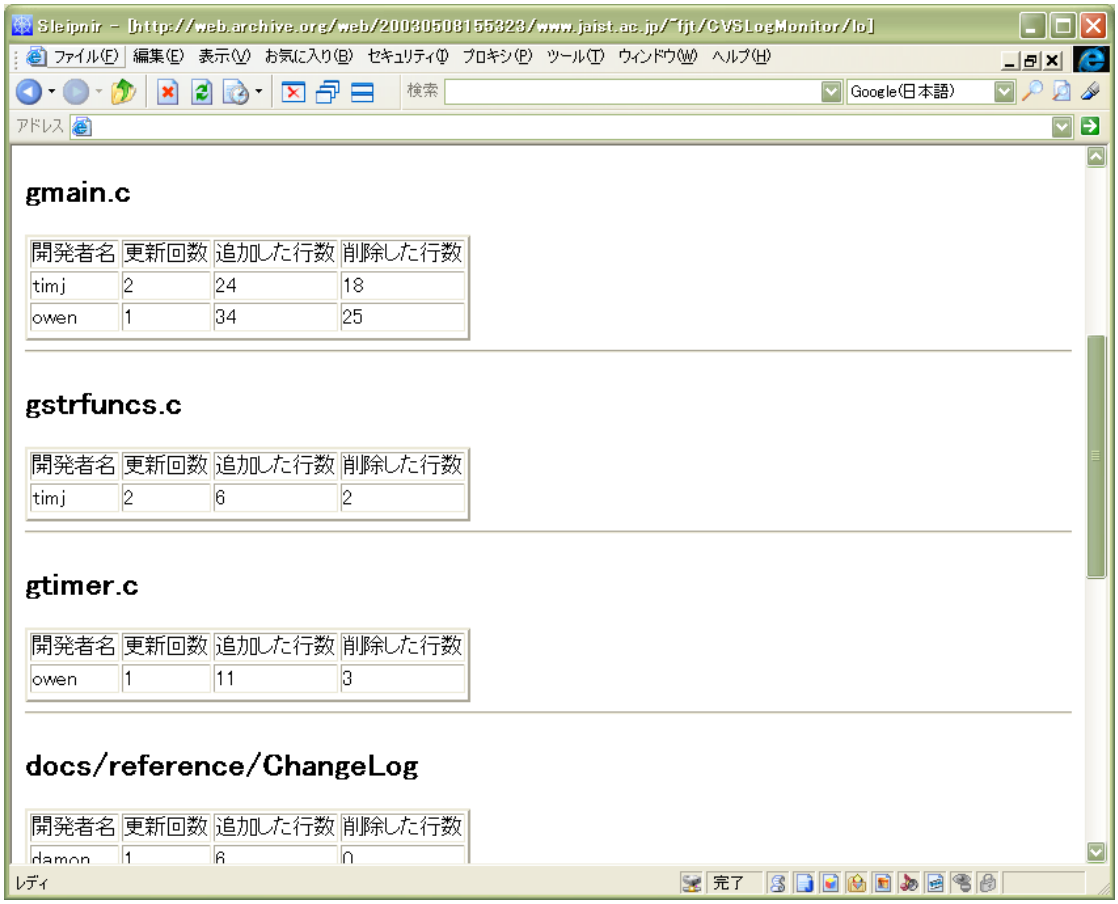


図 3.6: ファイルに関する月ごとの更新の様子

```
(defun cvslogmonitor ()
  (interactive)
  (let ((buffer (get-buffer-create "*Java Output*")))
    (set-buffer buffer)
    (erase-buffer)
    (start-process "java" buffer "java" \
    "-cp" "/home/fs043/fjt/develop/CVSLogMonitor" \
    "jaist.fjt.CVSLogMonitor.CVSLogMonitor")
    (display-buffer buffer)))
```

図 3.7: CVSLogMonitor を起動する Emacs-Lisp

3.5.2 CVSLogMonitor を起動可能なエディタ

OSS 開発者達に良く使われているエディタである Emacs[3] と Vim[22] を CVSLogMonitor と連携させることでプロジェクト毎に必要な情報を (public data から) 分析可能にした。ここでの必要な情報には以下のものが含まれる:

- 修正の履歴
- 変更の影響範囲
- ファイルのオーナー

CVSLogMonitor をエディタ (Emacs, XEmacs v20 以降で確認) から起動可能にする設定を図 3.7 に示す。CVSLogMonitor 自体は別ウィンドウで実行される。エディタは CVSLogMonitor の出力をエディタ自身のバッファで受け取ることができる。

3.6 評価実験

これまでに本研究の作業履歴を利用するアプローチと CVSLogMonitor の有効性を検討するため、開発者が更新を行なっているファイルの種類とその更新量に着目し、開発者の役割や関わりの深さを調べる実験を行った [42]。この実験では

Gimp プロジェクト [72] と関連するライブラリの 1 つである glib の作業履歴を獲得して次の分析実験を行った:

1. 開発者の開発活動の調査
2. 新規参加者に求められる情報の抽出

Gimp プロジェクトの概要と各実験の詳細を以下に示す。

3.6.1 Gimp プロジェクトの概要

Gimp (GNU Image Manipulation Program) は様々なプラットフォームで利用可能な OSS のグラフィックツールである。Gimp プロジェクトでは、一般的な公開情報 (CVS、メーリングリスト、バグデータベース) に加えて、Netnews のニュースグループ[1]や Wiki[2] を用いてユーザからの質問や要望を受けつけている。

Gimp プロジェクトでは新しいバージョンに向けた新機能の追加やバグの修正を行うと共に、追加機能を提供するライブラリや非安定版プラグインの開発を複数のモジュールに分けて進めている。そのため、新しい機能やプラグインの追加・改良を行う際には、プロジェクトの状況を把握した上で関連するライブラリの開発状況について知る必要がある。Gimp プロジェクトが開発者に向けて公開している Web ページ [71] では以下の情報を提供している[3]:

1. CVS について
   - 登録されているモジュール
   - 最近の更新内容
   - リリース時の変更点
   - Gimp の構築に必要なソースコード及びライブラリの入手法とコンパイル手順等
2. バグデータベースについて

[1] comp.graphics.apps.gimp
[2] Web ブラウザ上で情報交換が行えるシステム
[3] これらの情報は論文執筆時のものである。

- バグデータベースへの登録方法
- 未解決のバグに関するバグデータベースへのリンク
- 拡張予定の機能に関するバグデータベースへのリンク

3. メーリングリストについて

- 4 つのメーリングリスト (アナウンス専用、ユーザ向け、開発者向け、Web 開発用) のアーカイブへのリンク
- 上記の各メーリングリストへの参加方法
- Gimp に関連する他のメーリングリストへの参加方法

4. プラグイン開発について

- Gimp のライブラリ及びアプリケーションの API リファレンス
- プラグインのテンプレート

これらの情報の中には質問の仕方やパッチの送り先などが紹介されているが、プロジェクトへの参加方法やファイルのオーナー、開発状況に関する具体的な情報はない。提供された情報に従う場合、開発への参加希望者が必要な情報を得るためには、まずメーリングリストのアーカイブの検索を行い、求める情報が得られないことを確認した後に、質問を投稿して誰かが答えてくれるの待たねばならない。

3.6.2 実験 1: 開発者の活動の様子

Gimp を構築する際に必要なライブラリ glib の作業履歴を獲得して各開発者の活動に関する分析実験を行った。実験には Gimp プロジェクトが公開している CVS リポジトリから得られる 2000 年に行われた glib に関する作業履歴を用いた。

分析の結果からは、更新を行なった開発者が 35 人、全体での更新行数が 45562 行、そのうち C プログラムに関する更新が 22072 行であることが分かった。ここでは開発者毎の全ての更新行数をグラフ化したものと、ドキュメントを除いた、ソースコードの更新行数だけをグラフ化したものを図 3.8 に示す。

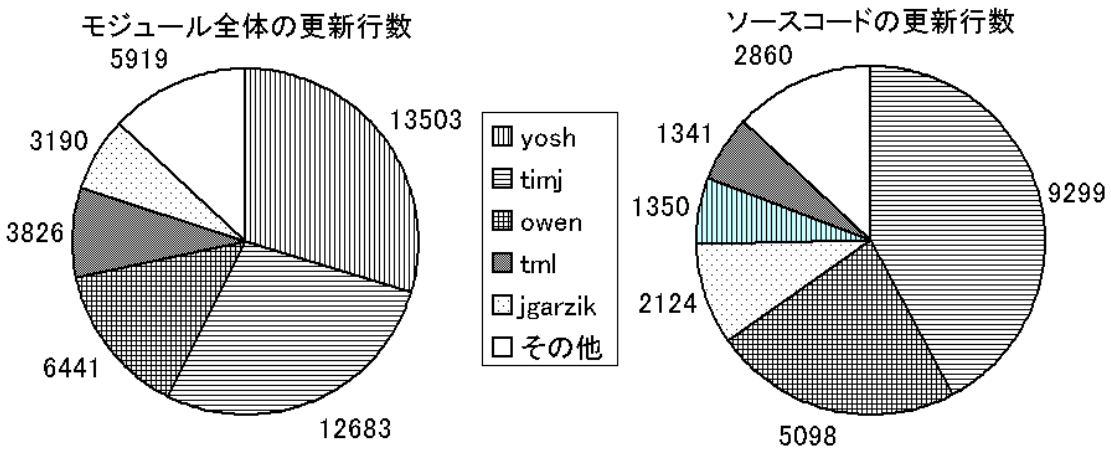


図 3.8: 開発者ごとの更新量の違い

このグラフからは、全体で行われた更新行数の半数以上が yosh と timj という 2 人の開発者によって行われていることが分かる。また、2 つのグラフを比較して見ると、全体では 2 番目に更新量が多かった timj がソースコードに関しては最も多くの更新を行なっていること、全体では最も多くの更新を行なっていた yosh がソースコードに関してはそれほど更新を行なっていないなど、開発者毎の更新対象の違いを見ることができる。

3.6.3 実験 2: 新規参加者に求められる情報の抽出

「新たに OSSD に参加して新しい機能の実装を試みる」という仮定の下に、機能追加の際に必要になると考えている情報の抽出実験を行った。

この実験では Gimp プロジェクトの 2000 年 1 月 1 日から 2000 年 12 月 31 日までの 1 年間に行われた 36 人の開発者達による約 1200 件の更新履歴について以下の情報抽出を行った：

1. 新しく機能が追加されたファイル
2. 1 で抽出したファイルと関連の強いファイル
3. 1 と 2 で抽出したファイルのオーナー

```
* app/commands.c
* app/layer_select.[ch]: small new feature: display a layer preview
popup when switching the active layer using PageUp/PageDown. Could
need a little tweaking...
```

図 3.9: 版管理システムから得られる更新時に残されたコメント

ここで、「新しく機能が追加されたファイル」とは、類似事例の検索結果として得られたものとする。

新しく機能が追加されたファイル　更新時のコメントから機能追加に関する検索を行い、最初に該当した「layer_select.c」を調査の対象として選んだ。

この変更は 2000/12/28 17:31 に Sven Neuman (neo) によって CVS に登録されている。登録時に彼が残したコメントを図 3.9 に示す。コメントからも読みとることができるが、同時に更新されたファイルには「commands.c」と「layer_select.h」があった。

関連の強いファイル　「関連の強いファイルは同時に更新される」という仮定の下で、更新された日時と更新者の名前から同時に更新されたファイルの数を求めた。表 3.1 は、「layer_select.c」と「layers_dialog.c」について同時に更新されたファイルの数を回数ごとに示したものである。表 3.1 のファイル名の横に記された括弧の

表 3.1: 同時に更新されたファイル数

	同時に更新されたファイルの数				
ファイル名	1 回	2 回	3 回	4 回	5 回以上
layer_select.c (6)	131	35	9	0	1
layers_dialog.c (46)	129	52	32	16	23

中の数字は期間中に更新された回数を示している。「layer_select.c」と同時に更新された回数が最も多かったファイルが「layers_dialog.c」である。「layers_dialog.c」は「layer_select.c」の 6 回更新の内、5 回同時に更新されていることが分かった。

また、「layers_dialog.c」は期間中に 46 回更新されていて、最も多く同時に更新されたファイルは 10 回の「gimpimage.h」であった。

この結果から「layer_select.c」と「layers_dialog.c」の 2 つのファイルの間には強い関連があると考えられる。そこで「layer_select.c」を編集する際には、「layers_dialog.c」を調査すべきファイルとして提示する。

ファイルのオーナー　更新履歴からは「layer_select.c」が 2 人の開発者 neo (4 回) と mitch (2 回) によって計 6 回更新されていることと、最後に更新を行ったのが mitch であることが分かった。また、「layers_dialog.c」については 46 の更新の内、26 回が mitch によって行われ、残りは neo が 14 回、yosh と alt が 2 回ずつ、garo と simon が 1 回ずつ更新を行っていた。最後に更新を行ったのは mitch であった。

この結果からオーナーを断定するのは難しいが、他の開発者達に比べて多くの更新を行っている mitch と neo を候補として挙げることができる。CVS のリポジトリに登録されたファイルの編集を望む際には、彼等からアドバイスや変更の同意を得るべきである。

3.7　実験結果の考察

開発者の活動の様子を調べる実験では、作業履歴を分析することで活発に開発を進めている開発者達を特定することができた。リポジトリへの書き込み権限を持ち、自ら頻繁に更新を行う彼等はコアメンバかそれに近い立場の開発者であると考えられる。プロジェクトにおいて強い決定権を持つコアメンバの動向を追うことは、プロジェクトの方向性を調べる際に有効な方法の 1 つであると考えている。

新規参加者に求められる情報を抽出する実験では、新機能を追加する際に必要な情報の獲得を行った。ここで得られた結果は、分析によって抽出した数字であるため実世界での検証が必要になるが、度々同時に更新されるファイルが強い関連を持ち、最も頻繁にファイルの更新を行う開発者をファイルのオーナーと見なすことは、詳細な分析によって十分な説得力を持ち得ると考えている。

3.8 実験のまとめ

本実験では OSSD プロジェクトの公開情報である CVS の作業履歴を用いて以下に示すプロジェクト情報の抽出を行った:

1. プロジェクトの主要な開発者
2. 新しく機能が追加されたファイル (類似事例の検索)
3. 関連するファイル
4. ファイルのオーナー

これらの情報は、OSSD プロジェクトの CVS のリポジトリサーバと対象となるソフトウェア名を設定することで自動的に抽出して蓄積できる。そして蓄積した情報から任意のプロジェクト情報を検索する手段を提供することで、OSSD に関心を持った新規参加者に対して先述のプロジェクト情報を提供できる。

3.8.1 今後の展望

3.1 節で触れた情報提供サイトを構築するために、プロジェクト情報を獲得するツールの改善及び、獲得したプロジェクト情報の妥当性について評価を行い、新規参加者にとってより有用で詳細な情報とその抽出手法を確立する。

第 4章

問題報告を利用した実践的プログラミング演習問題作成手法

本章では OSSD プロジェクトのバグデータベースから抽出した問題報告とその修正差分を利用した実践的プログラミング演習問題の作成手法を示す。

4.1 高度なプログラミング技術の習得

基盤ソフトウェアを保守していくためには、システムやプログラミングに関する高度な技術が求められる。OSS の修正や拡張を行う際には他者が記述したソースコードを理解し、他の機能に悪影響を与えないソースコードを記述する必要があるからである。また、プロジェクトの方針に合わせたコーディングスタイルや命名規則に従い保守性を考慮する必要がある。これらの技術は従来の学習方法による習得が困難であり、効果的に身につける学習法が望まれる。

4.1.1 従来の学習方法

一般的なプログラミング技術の学習方法には、参考書を利用する方法と既存のソースコードを参照する方法が挙げられる。それぞれの特徴を以下に示す。

- 参考書の利用
  参考書はプログラミング言語の特徴や、文法を学習するのに有効である。し

かし、このような参考書の多くはプログラミング言語そのものを解説するにとどまっている。保守作業を考慮したプログラミング技術を解説した参考書 [36, 38, 79] もいくつか存在するが、具体的な保守作業を教える教科書やデバッグ問題集はほとんどない。高度なプログラミング技術の習得を前提とした場合に、学習者にとって適切な内容、難易度の参考書を探すことが困難である。

- 既存のソースコードの参照
  企業やグループ内で開発された既存のソフトウェアのコードレビューを行うことで、習熟した開発者の技術を学ぶことができる。しかし、この方法では既存のソースコードの他に、適切な技術と知識を備えた指導者の存在が前提となる。また、個人で学習のための資源と指導者を得ることは困難である。

4.1.2 プログラミング演習問題による学習

筆者は OSS 開発で現実に発生したバグを扱った実践的なプログラミング演習問題による学習法が高度なプログラミング技術の習得に有用だと考えている。

オープンソースソフトウェア開発プロジェクトでは、優秀なプログラマ達が中心となってソフトウェアのプログラミングを行っている。作成されたソースコードは他の多くの開発者やユーザ達の貢献によって洗練され、バグが減少していくことが知られている [59]。このような OSSD における作業内容の多くは、次節で述べるように版管理システムやバグデータベースなどのプロジェクトの公開情報から調べることが可能である [43]。この作業内容から得られる OSS 開発者達のプログラミング技術と洗練されたソースコードはプログラミング技術を学ぶ者にとって良い手本となる。また、多くの OSSD プロジェクトではバグデータベース [55, 69] を利用してユーザからの問題報告を一般に公開している [32]。この問題報告から演習課題を作成し、実際に OSS 開発者達によって施された修正内容から解答例を作成する手法の詳細を次節で述べる。

4.2 公開情報を利用した演習問題の作成方法

本手法は、OSS 開発で一般に利用されている版管理システム (CVS)、バグデータベース、メーリングリストから得られる情報を利用して問題を作成するものである[1]。その概念を図 4.1 に示す。

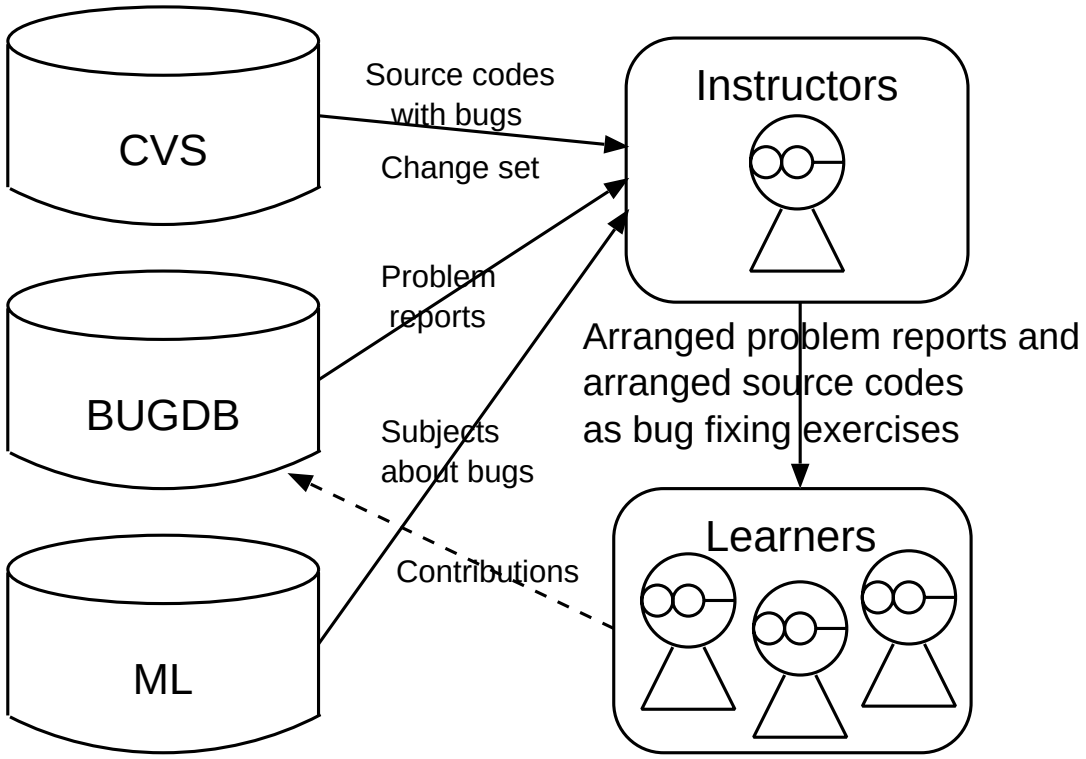


図 4.1: 問題作成手法の概念図

この手法で作成された演習問題は、演習問題としての体裁を整える必要があるが、学習者が関心を持った OSS に存在した人為的でないバグの解決に取り組める。この演習問題はバグを含むバージョンのソースコードと、バグを引き起こすための設定ファイル及びスクリプトからなる。学習者はこれらを用いて実際にソフトウェアのコンパイルを行い、再現されたバグの状態を確かめながらデバッグ作業を行うことができる。

ほとんどの OSS 開発では、開発者や関心を持ったユーザに対してソースコードの他にも開発者向けのメーリングリストのアーカイブや、バグデータベースを

[1]厳密には変更要求とバグの修正に分類できるが本論文ではバグの修正のみを対象とする

Web 上で公開している。本手法では、この公開情報から問題作成に必要なデータを抽出する (表 4.1)。これらは Web 上で公開された CGI に要求を送ることで獲

表 4.1: 公開情報から得られる情報

公開情報	獲得できる情報
版管理システム	ソースコード、変更差分、作業履歴等
バグデータベース	報告されたバグの一覧と現状等
メーリングリスト	ユーザ向け、開発者向け、バグ報告等

得できるため、簡単なスクリプトで必要なデータを収集することができる。そのため、手法が確立すればほとんどの作業を自動化できる可能性がある。

4.2.1 プログラミング技術の習得に適した問題

プログラミング技術の習得に適した問題として、次のような特徴を備えていることが望ましいと考える。

- 有名なバグを扱った問題
  有名なバグに対する一般的な対処法を学習できる演習問題が求められる。有名なバグとしてバッファーオーバーランやメモリリーク等がある。これらのバグに対する絶対的な解決方法はないが、同種のバグを蓄積することで問題解決のリファレンスになると考えている。これらのバグを解決する能力は多くのソフトウェア開発で求められている。また、ソースコードを書く際に同様の間違いを犯さないためにも学習の必要性がある。

- 修正の目的が明確な問題
  修正すべき内容が明確な演習問題が求められる。ここでの明確さとは構文エラーの様に「修正箇所」が明確であることを指すのではなく、コンパイルは可能だが目的通りに動作していないソースコードの修正等、「修正の目的」が明確であることを指している。このような演習問題を利用することで、ただ修正とコンパイルを繰り返すだけの作業ではなく、動作の理解を前提とした修正作業を学習することができる。

- 質の高い模範解答が存在する問題

  演習問題には十分な根拠を持った解答例が求められる。ある問題に対するソースコードの修正方法は、多くの場合、1 つではなく複数の方法が考えられる。そのような複数の修正方法の中から十分な根拠を持つ修正方法を解答例として示すことで、周辺のソースコードと調和した修正方法を学習することができる。

参考書の利用や既存のソースコードの参照では学習が困難なプログラミング技術習得のサポートを目的として、これらの特徴を備えた演習問題を提供する。

問題作成の際に利用する情報について 4.2.2 節で述べる。

4.2.2 バグデータベースの問題報告

バグデータベースに登録された問題報告には、ユーザの環境で発生したバグの詳細や再現方法、開発者達の対応が記述されている (表 4.2)。これらの情報を利用

表 4.2: Apache の問題報告で用いられているタグ

タグ	意味	例 または 種類
Number	問題の識別子	812 (数字)
Category	問題の分類	mod_proxy (問題があるモジュールの名前)
Synopsis	問題の概要	(問題の症状)
Severity	問題の危険度	critical, serious, non-critical
State	問題の状態	open, analyzed, suspended, feedback, closed
Arrival-Date	報告された日時	Tue Oct 21 15:30:01 PDT 1997
Last-Modified	最後に更新された日時	Thu Nov 13 20:00:59 PST 1997
Release	影響のあるバージョン	1.3b2 (リリースタグ)
Environment	ユーザの環境	FreeBSD 2.2.1-RELEASE
Description	問題の内容	(問題報告)
How-To-Repeat	問題の再現方法	(手順)
Fix	ユーザからの修正案	(パッチ or 修正案)

することで、バグを再現するための環境と問題解決に関する作業履歴を得ること

ができる。

1. 問題を含むソースツリーのバージョン
2. 問題の特徴と再現方法
3. 問題が報告された日時と修正された日時

4.2.3 演習問題構成アルゴリズム

公開情報からプログラミング演習問題を作成する演習問題構成アルゴリズムの具体的な手順を以下に示す。

1. State (問題が解決したか否か) や Severity (問題の危険度)、Category (問題の分類) を指定して該当するバグの一覧を要求し、内容を吟味した上で問題としての体裁を整える。
   例: closed, serious, mod_proxy
2. Release (影響のあるバージョン) を指定してバグを含むバージョンのソースツリーを獲得する。
   例: *cvs -d (*リポジトリの場所*) co -r (Release*に該当するタグ*) (*モジュール名*)*
3. バグが発生する環境に合わせて Makefile を編集することで、獲得したソースコードをいつでもコンパイルできる状態にする。
   例: *configure –prefix=/tmp*
4. How-To-Repeat (問題の再現方法) を利用してバグを再現するための設定ファイルやスクリプトを作成する。
5. Arrival-Date (報告された日時) と Last-Modified (最後に更新された日時) の間に行われた変更を獲得する。
   例: *cvs diff -D(Arrival-Date*の日時*) -D(Last-Modified*の日時*)*
6. 単純に *diff* を取ると莫大な量になるため、該当するバグに関する修正を特定して絞り込むことで解答例を作成する。

例*:* メーリングリストのアーカイブや *CVS* の *log* コマンドの出力から *PR* に関する話題を検索し、修正が行われた具体的な箇所や日時を調べることで該当箇所を絞りこむ。

これらの手順の中で 1 と 4 と 6 の作業は、一般的に自然言語で書かれたテキストを解釈する必要があるため、手作業で行わなければならない。しかし、バグを再現するスクリプトの記載を義務づけるなど、PR の記述スタイルを厳密に定めることで 1 と 4 については自動化できる可能性がある。2 と 5 の作業はスクリプトを用意していくつかのパラメタを変更するだけで自動化できる。3 の作業は特殊なライブラリや設定が必要のない場合にのみ自動化できる。

4.2.4 期待できる効果

この手法によって作成された問題は修正箇所が 1 つとは限らず、修正内容が他の箇所に影響を与える場合がある。そのため関連ファイルについても、その内容を理解する必要性が生じ、プログラミングに重要な能力であるリーディングの技術 [76] を磨くことができる。また、解答例について、実際には複数の修正方法が考えられる中から、多くの優秀な OSS の開発者達が採用した解答例を参考にすることで、より実践的な問題解決の方法を学ぶことができる。加えて、この修正方法に関する詳細な情報は粒度にばらつきはあるものの、版管理システムに残されたコメントやバグデータベースの更新情報、メーリングリストのアーカイブからキーワード検索によって獲得できる。これらを併せて用いればコードレビューによって得ることができた経験に基づく知識や、状況に応じたプログラミングスタイルを学習することができる。他にも次のような効果が期待される。

- 問題の解決方法の比較
  複数の解答例を用意して比較することでコードレビューを行う。他のソースコードとの一貫性や提供されている変数、関数の利用方法を互いに比較することで、より実践的なソースコードの書き方を学ぶ機会を得る。

- 未解決のバグへの対応
  バグデータベースに登録されているバグで、未だに修正されていないバグ

について演習問題を作成し、修正可能であればそのままプロジェクトに貢献する。

4.2.5 プログラミング技術学習に関する研究の現状

従来のプログラミング技術学習に関する研究は次のような制約を持つ:

1. 学習に適した特徴を持つ特定のプログラミング言語を利用している
2. 特定の環境でプログラミング作業を行う必要がある [73]
3. プログラミング情報の視覚化にとどまる [19]
4. 簡単な演習問題を作成して提供する
5. 解析のために特定のフォーマットに従う必要がある [40, 75]

1 及び 2 については、利用可能なプログラミング言語や OS に制限が設けられるために既存の OSSD に参加する際には役立たない。本研究で提案する演習問題構成アルゴリズムは、多くの OSSD プロジェクトで利用可能なバグデータベースと CVS から得た PR と作業履歴を用いて演習問題を作成できる。そのため、演習問題の作成コストは伴うが、各プロジェクトで実装に用いているプログラミング言語及び開発対象となるソフトウェアに関する演習問題を作成し、提供することが可能である。

3 及び 4 については、プログラミングまたはその理解を容易にする研究であり、高度な技術を習得することが目的ではない。本研究の手法では、OSS で実際に発生したバグを再現した演習問題を利用することで、バグの潜んでいる箇所を探すために、リーディング技術を身に付けることができる。また、OSSD の開発者達によって洗練された修正方法を参照することで、高度なプログラミング技術を学ぶ機会が得られる。

5 については、既存のソースコードに変更を加える必要があるため適用性が低い。本研究では、演習問題として利用する OSS のソースコード及び PR について、何ら変更を加えずに演習問題を作成することができる。演習問題の利用者についても、解答のための作業ディレクトリとバグを再現するための設定ファイル

を用いることで、オリジナルのソースコードや設定ファイルを直接編集することなく問題の解決に取り組むことができる。

4.3 プログラミング演習問題集「ひよこ」システム

この方法で作成した演習問題の公開と管理を行うためのシステム「ひよこ」(図 4.2) を構築した。学習者は「ひよこ」を利用することで、オープンソースソフトウェア開発で現実に起こった問題解決に取り組むことができる。

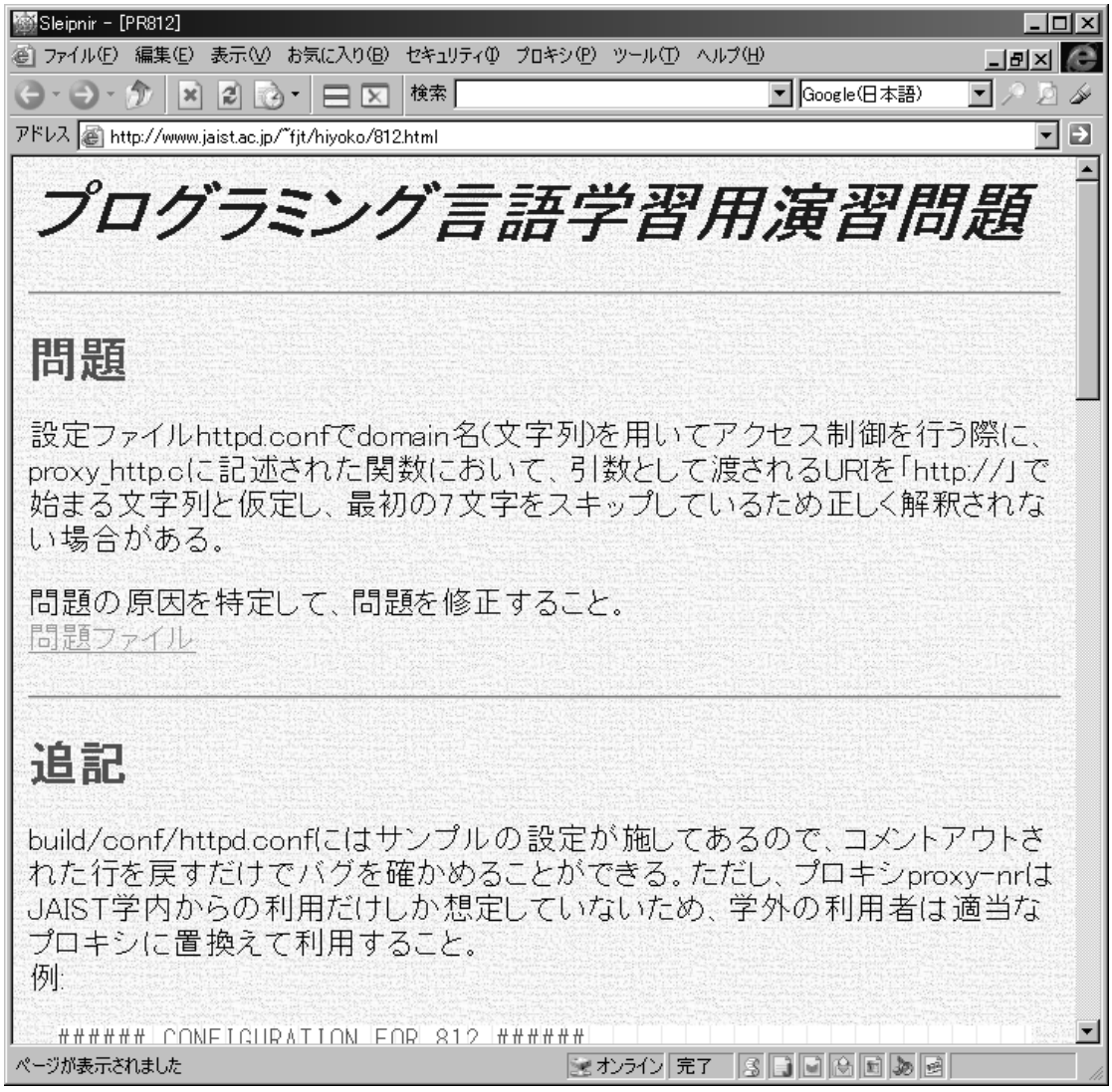


図 4.2: ひよこ の演習問題

「ひよこ」は Apache HTTP サーバ上で実現されていて、以下に示す 3 種類の Web ページで構成されている:

1. 問題を選択するための Web ページ
   学習者はキーワードから演習問題を選択することができる。

2. 問題を説明するための Web ページ
   学習者は指示に従うだけでバグを再現することができる。

3. 解答を送るための Web ページ
   学習者は Web のフォームを利用して解答を送信し、解答例を受け取ることができる。

論文執筆時現在、「ひよこ」の問題は以下のフォーマットに従っている:

- 問題及びバグを含む問題ファイルが入手可能な URI
- 追記 (バグの再現方法)
- 利用方法 (問題ファイルの展開からコンパイルまでの手順)
- 解答の作成方法 (スタイルを統一するために *diff* の利用方法を指定する)
- 解答の提出方法 (解答フォームへの URI)

問題の解答を送るための解答フォームには、問題の再評価と分析を行うためのアンケート項目が含まれている。また、作成した解答ファイルはサーバ側で起動される CGI と、標準的な Web ブラウザの機能だけを使って送信される。解答例を受け取る際も同様である。これによって学習者は、わざわざメーラーを起動して必要事項を書き込んだメールを作成することなく、問題の利用と解答を気軽に行うことができる。

4.4 演習問題の評価

本手法を評価するために実際に演習問題の作成を行い、被験者に解かせた。

4.4.1 演習問題の作成

問題作成のソースとして、今回は Apache HTTP サーバプロジェクト [16] のモジュールの 1 つである apache-1.3 のソースツリーを利用した。

以下、問題報告の抽出と実際に作成した問題の具体例、問題作成時に得られた知見について述べる。

4.4.2 Apache の問題報告 (PR) の抽出

プロジェクトが公開している GNATS[69] を利用したバグデータベース [32] から、apache-1.3 に関する PR の調査を行った。この際、サーバへの負荷を考慮して 2002 年 4 月時点で 5000 件以上報告されている apache-1.3 に関する PR の内、登録が早いものから 100 件について PR と該当する期間の修正内容を獲得してバグの内容と解決方法を確認した C.1。そして PR に含まれる情報 (表 4.2) に着目して、問題作成に利用する PR を選択し、問題と解答例の抽出を行った。

PR には先に示した情報の他に、報告者と開発者との間でのメッセージのやり取りと、それに伴う State の変化が記録されている。このやり取りの基本的な流れを次に示す。

1. ユーザからバグが報告された (open) 後に開発者がバグの確認を行う。この時、バグが再現されない場合には状況確認のために報告者とのやりとりが続く。

2. ユーザの間違いが確認された場合にはその旨を報告して終了 (closed) する。Apache グループの場合、報告者からの返事がないまま一定期間が過ぎると問題が解決したとみなして終了となる。また、既存の報告に同様の内容が見つかった場合には該当する報告を示して終了する。

3. バグが確認された (analyzed) 場合には、開発者及び、報告者、ユーザのいずれかによってバグが修正されるのを待つ。深刻なバグではなくてユーザの願望が強いものであった場合には、フィードバック待ち (feedback) となる。

4. バグが修正された場合には対応したバージョンを報告して終了する。現在は

対象としない問題であった場合には中断 (suspended) される。

ここからは、State が open から analyzed を経て closed へと変化した場合、すなわち、問題が報告されて確認された後に終了している PR を調べれば、問題作成に必要な情報が得られることが分かる。

今回は、PR の内容と規模を考慮して、1 つのファイルの編集のみで修正が可能と判断した PR812、PR1155、PR1294 の 3 つの PR から問題を作成した。

4.4.3 作成した問題の例

ここでは PR812[2] を基にして作成した問題と解答例について、内容とその作成方法を示す。

問題 PR812

問題を作成するために、まず PR812 に記述された Description の内容の要約を行い体裁を整えた。和訳したものを図 4.3 に示す。この例では予め設定ファイル

> 「設定ファイル httpd.conf で domain 名 (文字列) を用いてアクセス制御を行う際に、*proxy_http.c* に記述された関数において、引数として渡される URI を「http://」で始まる文字列と仮定し、最初の 7 文字をスキップしているため正しく解釈されない場合がある。」

図 4.3: 問題の例 (PR812)

(httpd.conf) に正しく処理されない domain 名を記述しておくことで、3 節で挙げた特徴の一つである「修正の目的が明確な問題」を作ることができる。

次に、Release に書かれたバージョンを選択してチェックアウトすることでバグを含むソースツリーを獲得した。

```
cvs co -r APACHE_1_3a1 apache-1.3
```

実際には PR812 が該当するバージョンは「1.3a2」であるが、PR1155 が該当するバージョンが「1.3a1」であったため、関連箇所に変更がないことを確認した上で

[2] http://bugs.apache.org/index.cgi/full/812

「1.3a1」のソースツリーを共用している。このソースツリーに含まれる C 言語に関するファイル全て (*.[ch]) の行数の総和は 58992 行である。その内、PR で指摘されている *proxy_http.c* の行数は 426 行、修正対象となる関数 proxy_http_handler の行数は 285 行である。

最後に、獲得したソースツリーに対して解答者が手を加えることなく問題に取り組めるように、作業用のディレクトリを作成して必要なファイルをコピーし、バグを再現するための設定ファイル (*httpd.conf*) の書き換えや、コンパイルと実行に必要なスクリプト (*Configure*) の準備を行った。この作業によって、解答者は作業用ディレクトリ上でそのまま *make* コマンドを実行することでコンパイルを行い、apache-1.3 の実態である *httpd* を実行するだけでバグを確認することができる。

解答例 PR812

PR812 の問題を解決するために実際に行われた修正内容を *diff* によって出力したものを図 4.4 に示す。図 4.4 の出力を得るためには、まず、次のコマンドによって Arrival-Date で示された日時から Last-Modified で示された日時までの間に修正された内容を抽出する。

```
cvs diff -D"Tue Jul  1 18:10:01 1997" -D"Mon Aug 18 02:43:48 PDT 1997"
```

続いて、得られた修正差分 (23418 行) から該当箇所を探し出し、該当箇所の影響範囲の特定と、作業履歴の分析 [41] を行うことで PR812 に関する修正箇所だけに絞り込んだ。

この修正差分では、渡された URI に「://」が含まれている場合に、該当箇所までスキップして得られた文字列を、新たに宣言した urlptr に渡して利用するように修正している。

4.4.4 演習問題作成のコスト

演習問題を作成するために、最初に 2 週間を費やして 100 件の PR と変更内容の評価を行った (付録 C.1) が、多くの PR は報告者の間違いとして処理されたか、PR から抽出した期間中に修正が行われていないことが分かった。また、自動的

```
Index: proxy_http.c
===================================================================
RCS file: /home/cvspublic/apache-1.3/src/modules/proxy/proxy_http.c,v
retrieving revision 1.26
retrieving revision 1.27
diff -r1.26 -r1.27
158a159
>     char *urlptr = NULL;
172c173,175
<     url += 7;  /* skip http:// */
---
>     urlptr = strstr(url,"://");
>     if (urlptr == NULL) return BAD_REQUEST;
>     urlptr += 3;
174c177
<     p = strchr(url, '/');
---
>     p = strchr(urlptr, '/');
177,178c180,181
<         desthost = pstrdup(pool, url);
<         url = "/";
---
>         desthost = pstrdup(pool, urlptr);
>         urlptr = "/";
181,184c184,187
<         char *q = palloc(pool, p-url+1);
<         memcpy(q, url, p-url);
<         q[p-url] = '\0';
<         url = p;
---
>         char *q = palloc(pool, p-urlptr+1);
>         memcpy(q, urlptr, p-urlptr);
>         q[p-urlptr] = '\0';
>         urlptr = p;
210d212
<       url = r->uri;                   /* restore original URL */
```

図 4.4: 修正差分から抽出した解答例

に抽出した変更差分から修正内容を探し出す作業は、数万行のファイルから数十行を特定する非常に困難なものとなった。実際には、先述のように修正が行われていない場合もあり、作業の後で Apache のバグとして認められる PR は 32 件であった。ここでバグを再現するために特別な環境が必要な PR を除いた上で、演習問題に相応しい特徴を備えた PR を選択した結果として 3 件の PR を選択した。

この結果を踏まえて、新たに CVS の作業履歴中のコメントに修正の報告があった 100 件の PR について同様の評価を行っている (付録 C.2)。しかし、修正の報告があった PR の中にも実際には改善が見られない PR が残っていたために、思うように候補を選択することが出来なかった。結果として、10 件の PR を問題作成の候補として選んだ。

PR を調査する過程で得られた知見を以下にまとめる。

- PR が有効なものとは限らない
  報告者の間違いとして処理された PR やバグが解決されないまま処理された PR がある。

- バグレポートの open から closed までの期間
  この期間が長くなると *diff* によって得られる差分が大きくなるため、修正箇所の特定が難しくなる。例えば PR909[3] の場合、Arrival-Date が「Thu Jul 24 07:10:01 1997」、Last-Modified が「Sun Dec 14 07:02:54 PST 1997」となっている。この期間 (約 140 日間) に行われた変更の *diff* による出力は 70070 行にもなる。解決したと思われたバグが再発した場合に期間が長くなることが多い。

- 該当する修正が含まれているとは限らない
  closed までの期間が短い PR の中には、開発者が「次のリリースで修正する」として、実際には修正が終わっていないものがある。また、「修正した」と書いているにも関わらず、公開されているソースツリーに反映されていない場合がある。

- 無用な変更差分
  ソースツリー全体の変更差分を獲得するため、タイプミスの修正や改行、コー

[3] http://bugs.apache.org/index.cgi/full/909

ディングスタイルの変更、ソフトウェアのバージョンの書き換えなど、分析には不必要な差分が多量に含まれてしまう。

- 開発者が備忘録として利用
  開発者が自分のやるべきことを PR として報告し、自分自身で終了させている場合がある。この場合、PR の State が open から analyzed を経ずに closed に変わるため、問題抽出の自動化を行う際に妨げとなる。

4.4.5 演習問題の評価実験

作成した問題を評価するために実際に被験者に解かせた。出題の際には、作成した問題の他にバグが報告されたバージョンに該当するソースコードのアーカイブと、バグを再現するための環境 (計算機、OS) と手段 (設定ファイル、実行スクリプト) を与えた。

実験方法

この実験の目的は、本番の評価実験に備えて問題の難易度と時間設定を決める材料にすることと、作成した問題を客観的に評価して問題作成の参考にすることである。被験者は C 言語に習熟した修士課程の学生 1 名である。

被験者には、作成した PR812 に関する問題と、バグを再現するための設定ファイルとスクリプトを含むソースツリーを渡している。また、この問題は OS や計算機のアーキテクチャに依存しないため、被験者が普段使いなれている計算機と OS (NetBSD 1.5ZB, i386) で実験を行った。

実験結果

被験者は問題を受け取ってから 30 分程で修正を終えた。提出された修正差分を図 4.5 に示す。提出された修正差分は図 4.4 に示した解答例と形式は異なるが、問題の箇所について同等の修正を施している。

次に、被験者に解答例を渡した後でそれぞれの修正方法の違いを比較してコードレビューを行った。以下、レビューの結果として得られたコメントと解説を示す:

```
--- proxy_http.c        Wed May  8 15:07:48 2002
+++ proxy_http.c.ryoh   Fri May 10 02:52:34 2002
@@ -169,7 +169,12 @@

 /* We break the URL into host, port, path-search */

-    url += 7;  /* skip http:// */
+    for (p=url; ; p++) {  /* skip protocol part ("...://") */
+           if (*p == ':' && *(p+1) == '/' && *(p+2) == '/') {
+                   url = p + 3;
+                   break;
+           }
+    }
     destport = DEFAULT_PORT;
     p = strchr(url, '/');
     if (p == NULL)
```

図 4.5: 提出された修正差分

- *strstr(3)* を使った修正と本質的には一緒である
  PR812 の問題を解決するためには、指定したファイル proxy_http.c の関数 proxy_http_handler について、引数として渡された URI からドメイン名を正しく抜き出す必要がある。バグを含むソースコードでは、URI に書かれたプロトコルの種類を示す「http://」の部分をスキップするために、整数値 7 を指定してポインタを進めている。解答例及び被験者が提出した修正差分では、他の (「://」を含めて 7 文字以外の) プロトコルに対応できるように、URI が代入された文字列から「://」を検索することでポインタを進めている。しかし、解答例と比較した場合、被験者が提出した修正差分では「://」が存在しない場合を考慮していないことが分かる。このような陥り易いミスを学習者に意識させることで、プログラミング技術の向上につながると考えている。

- ポインタを利用するよりも、標準 C ライブラリの *strstr(3)* を使った方がよいかもしれない

プログラミング言語の標準関数や既に実装／提供されているライブラリを利用することで複雑な操作を単純に記述できる。単純な記述は OSSD を含む共同開発で求められる可読性を高めることにつながり、再利用性を向上する。しかし、このような標準関数やライブラリの使い方を学習する機会は少ない。OSS で採用された解答例を基に、学習者がこれらの利用法を学習する機会を与えることが有効だと考えている。

- *proxy_http.c* 以外のファイルで渡される URL を「http://」だけに限定する箇所がコメントアウトされていたが題意と違うと判断して利用しなかった
  問題となっている関数 proxy_http_handler はバージョン apache-1.3a1 に含まれるファイル mod_proxy.c の関数から呼び出されている。mod_proxy.c には「扱うプロトコルを HTTP だけに制限する」という趣旨のコメントと共に、プロトコルが「http://」であることを確認する条件式がコメントアウトされている。被験者はコメントアウトを解除することで、仕様を制限する方法を検討していた。プログラムの実現／修正方法は 1 つとは限らず、与えられた条件の中で最適な方法を見つけることが望ましい。OSS を利用した演習問題では、ソフトウェアを構成する全てのソースコードと関連するドキュメントが提供されるため、被験者は得られる情報から問題の修正方法を検討することができる。

- バグが修正できたかどうかを確かめられる点がよい
  作成した演習問題にはバグを再現するための設定ファイルが含まれているため、容易にバグの動作を確認し、修正の結果を確かめることができた。実際の OSS を用いることでソフトウェアの実行状態を確かめながら演習問題に取り組むことでデバッグ慣れることができる。これは空欄補充問題や多岐選択肢式の問題には無い特徴である。

- バグの及ぼす影響が重大なものであるほうがやる気が出る
  被験者が解いた演習問題のバグは、アクセスを拒否しているドメインから、HTTP 以外のプロトコルによるアクセスを認めてしまうものであった。被験者はバグの引き起こす問題の大きさを理解し、その問題を解消することに興味を覚えた。学習者のモチベーションを維持できることは、教材の持つべき

重要な特徴だと考えている。

4.4.6 実験結果の考察

修正作業は想定していたよりも短時間で完了し、得られた結果からはコードレビューによって被験者が自身の修正内容について考察できたことが分かる。また、バグの状態を確認しながら修正を行うことで、問題に取り組む意欲が沸くという知見を得た。

4.5 実験のまとめ

本実験では OSSD プロジェクトで実際に発生したバグを利用してプログラミング演習問題を作成し、被験者に問題を解かせることで演習問題の評価を行った。問題の作成にはコストがかかるが、学習者の学習意欲を刺激し、学習を継続させられる問題が望まれる。

4.5.1 今後の展望

課題として問題の作成と、質の向上、保守を進めていく手段が求められる。これらの作業を「ひよこ」でサポートするために、Web 上でコンテンツの編集を可能にするシステム [81] を利用して、問題作成者が演習問題を自由に追加・編集可能な機能を実現した。演習問題と共に編集機能を公開することで以下のようなフィードバックを期待している:

- 新しい演習問題
- 演習問題に対する評価・修正案
- 理由の示された別解

「ひよこ」の演習問題が多くの人の目に触れ、十分なフィードバックを受けることで OSS が発展するプロセスと似た効果が得られると考えている。利用者からのフィードバックや演習問題構成アルゴリズムによって作成された演習問題の洗練プロセスを以下に示す:

1. **初期状態**　演習問題と解答例が正当であるかどうか不明な状態 (図 4.6)
2. **演習問題・解答例の確認**　演習問題の内容が OSS 開発者によって実際に修正が行われたバグであり、該当箇所の修正内容が解答例に含まれることが確認 (期間の特定) された状態
3. **演習問題の修正**　バグが発生した環境の確認、問題文の整形・修正および省略された語の追加、バグの再現方法の修正・カスタマイズが行われた状態
4. **解答例の修正**　解答例をバグに該当する修正内容だけに絞り込み、変更理由が調査された状態
5. **表現のカスタマイズ**　問題文の表現を変える (バグを含むファイル名、関数名の追加・削除など) ことで難易度を変化させた課題が複数用意された状態

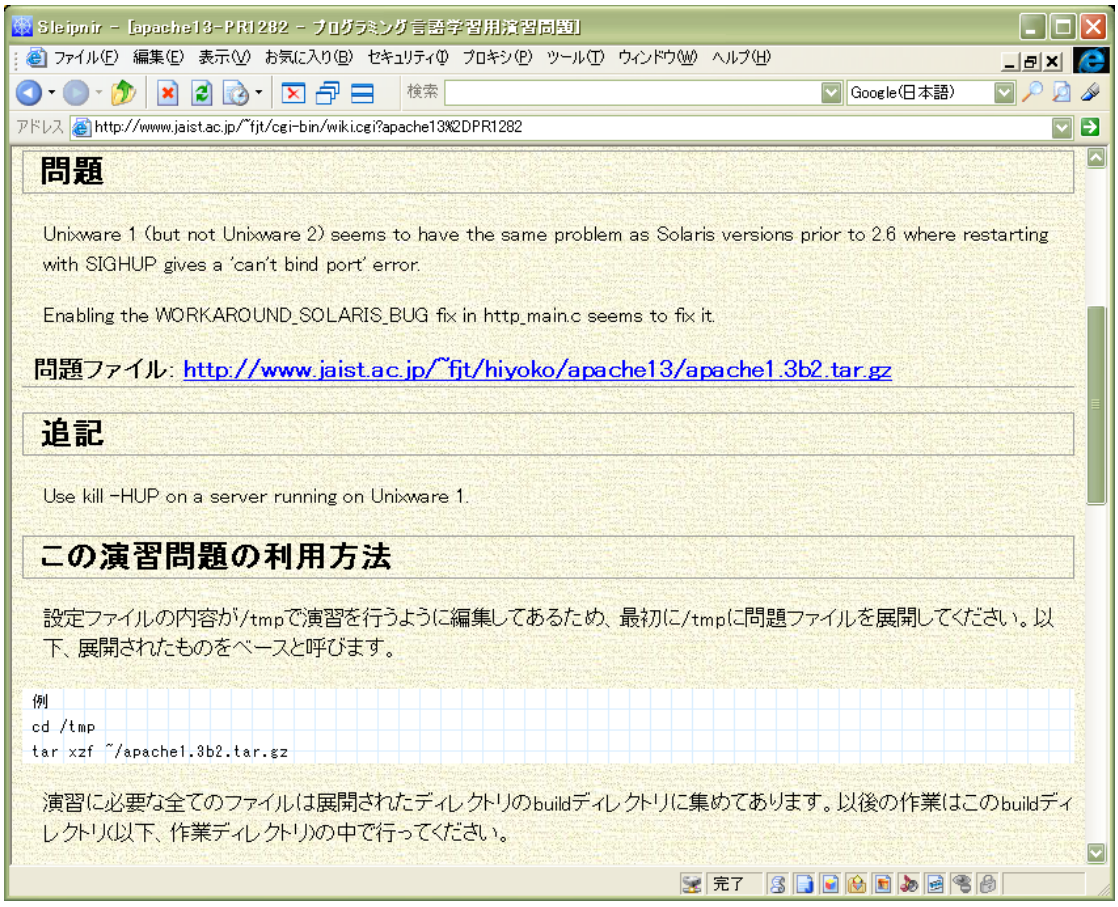


図 4.6: 自動生成した演習問題の初期状態

今後は、「ひよこ」を用いた問題作成の自動化及び、質を洗練するプロセスの効果を実験によって確かめながら改善・運用する。また、参考書による学習との比較等、学習効果に関する評価実験を行う。

第 5章

おわりに

本章では、まとめと今後の展望について記す。

5.1 本研究のまとめ

本研究では、OSSD の抱える問題について要求を明らかにし、OSSD の特徴を捉えつつ彼等の開発スタイルに影響を与えることなく、新規参加者を支援するアプローチを示した。

新規参加者が OSSD プロジェクトに参加する際に有用な情報を特定するために、OSSD プロジェクトの公開情報を得る源となる版管理システム、メーリングリスト、バグデータベースの 3 つのシステムについて、利用形態とそこから得られる情報について調査を行った。

プロジェクトに関する情報が得にくいという問題に対して、公開情報の 1 つである版管理システム CVS から得られる作業履歴を分析して、新規参加者に求められる情報を抽出する研究を行った。その際に、プロジェクト情報の抽出と分析をサポートするツール、CVSLogMonitor を開発し、OSSD で広く用いられているエディタとの連携を図った。本研究のアプローチと CVSLogMonitor の有効性を確かめる実験では、Gimp プロジェクトにおける主要な開発者達の開発対象の調査を行った。また、プロジェクトに参加して新機能を追加するという仮定の下で行った実験では、関連ファイルの特定とそれらのファイルのオーナーの特定を行った。

高度なプログラミング技術が要求されるという問題に対して、学習のために実

践的なプログラミング演習問題を提供するアプローチを示した。作成した演習問題を公開して評価するために、Web ページを利用して演習問題を提供するシステム「ひよこ」を立ち上げた。また、実際に作成した演習問題について被験者 (1 名) に解かせることで評価を行った。演習問題の作成コストは高いが、演習問題の内容については被験者から良い評価を得られた。

本論文で示したアプローチは、OSSD プロジェクトの開発スタイルに何の変更も求めず、彼等が一般的に利用しているシステムから自動的に獲得できる公開情報だけを用いるため OSSD プロジェクトに関して汎用性が高い。また、分析コストが低く、自動化が可能なため、多数のプロジェクトに適用できるため、SourceForge や Savannah に登録されている数万の OSSD プロジェクトに適用できる。

本研究の成果によって、世界中に散らばっている潜在的な OSS 開発者達の参加障壁を下げ、実際に OSSD プロジェクトに参加させることができれば OSS の更なる進化と発展が期待できる。

5.2 今後の展望

本論文で示した実験結果は公開情報だけを基に、明文化されていないプロジェクトの主要な開発者、新しく機能が追加されたファイル、ファイルのオーナーなどのプロジェクト情報を抽出した。版管理システムから得られた情報だけで新規参加者を十分にサポートできるとは言い難いが、本論文で述べたプロジェクト情報の獲得手法は以下の利点がある:

- OSSD で一般的に用いられている版管理システムを利用するため汎用性が高い
- 書式の整った版管理システムの出力を利用するため情報抽出の自動化が容易

今後は他の公開情報と組み合わせて得られるプロジェクト情報についても分析を行い、併用することが望ましい。その上で、各 OSSD プロジェクトから得たプロジェクト情報を蓄積して新規参加者のための情報提供サイトを構築する。また、OSSD に参加している開発者に実際に CVSLogMonitor を使ってもらうことで、プロジェクト情報の有用性を評価し、求められる情報と獲得手法の洗練及び CVSLogMonitor

の改良を進める。

「ひよこ」を用いた演習問題の提供に留まらず、演習問題作成と管理のためのパタンランゲージを記述することでガイドラインを示し、演習問題及び演習問題の質の改善を目指す。正しく分類されたカテゴリによって検索可能な演習問題を用意することで、プログラミング技術の学習に用いるだけではなく、類似問題を検索することで現実の問題修正の参考になるデバッグ・カタログとして機能させる。

謝辞

本研究を行なうにあたり、終始御指導を賜わった 落水 浩一郎 教授に深謝致します。また博士論文審査において有益な御助言、御指導を賜わりました 片山 卓也 教授、本学情報科学センター篠田 陽一 教授、 南山大学の 野呂 昌満 教授、東京工業大学の 権藤 克彦 助教授に深く感謝致します。

本研究を進めるにあたり、有益な御助言をいただき、多面に渡って励ましていただいた 本研究室 助手 藤枝 和宏博士に深く感謝致します。

日頃から研究の進め方をはじめ、悩みごとや個人的な相談にも快く応じてくださった 故 村越 広享 博士に深く感謝するとともに、故人のご冥福を心よりお祈りいたします。

多面に渡って励ましていただいた 日本テレコム株式会社研究所 猪俣 敦夫 博士、就学費に困っていた折に翻訳の仕事に誘ってくれた IIJ 技術研究所 宇夫 陽次朗氏、友人として日頃から励ましてくれた 富士通株式会社 斉藤 賢二氏、株式会社ビック東海 横山 浩一氏、株式会社 IRI(インターネット総合研究所) 田淵 貴昭氏、研究に必要な多くの時間を与えてくれた 本研究室 助手 服部 哲 博士に深く感謝いたします。

最後までわがままを許してくれた両親達と応援してくれていた姉夫婦に深く感謝致します。

研究のテーマであるオープンソースソフトウェアの普及を進めた Free Software Foundation 及び Open Source Initiative の皆様、全てのオープンソースソースソフトウェア開発者達に感謝致します。

最後に、本論文をまとめるに当たって御協力いただいた全ての方達、特に落水・篠田研究室及び片山・権藤研究室の諸兄に厚く御礼申し上げます。

参考文献

[1] Brian Berliner. Cvs ii: Parallelizing software development. In *1990 Winter USENIX Conference, Washington D.C.*, 1990.

[2] BitMover. Bitkeeper - the scalable software configuration management system. availed at `http://www.bitkeeper.com/`, 1997.

[3] Debra Cameron, Bill Rosenblatt, and Eric S. Raymond. *Learning GNU Emacs*. O'Relly & Associates, 1996.

[4] D. Brent Chapman. Majordomo: How i manage 17 mailing lists without answering "-request" mail. In *6th Large Installation Systems Administration Conference*, 1992.

[5] CollabNet. Project home page. availed at `http://ccvs.cvshome.org/servlets/ProjectHome`, 1999.

[6] CollabNet. subversion.tigris.org. availed at `http://subversion.tigris.org/`, 1999.

[7] Microsoft Corporation. Visual source safe web site. availed at `http://www.microsoft.com/japan/msdn/ssafe/`, 2001.

[8] Bryan Costales and Eric Allman. *sendmail, 2nd Edition*. O'Relly & Associates, 1997.

[9] Davor Cubranic and Kellogg S. Booth. Coordination in open-source software development. In *IEEE 7th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp. 61–66, 1999.

[10] Chris DiBona, Sam Ockman, and Mark Stone. *OPENSOURCES: Voices from the Open Source Revolution*. O'Relly & Associates, 1999.

[11] Roy T. Fielding and Gail E. Kaiser. The apache http server project. *IEEE Internet Computing*, Vol. 1, No. 4, 1997.

[12] Karl Fogel. *Open Source Development With CVS*. CoriolisOpenPress, 1999.

[13] Free Software Foundation. The free software definition. availed at `http://www.gnu.org/philosophy/free-sw.html`, 1996.

[14] Free Software Foundation. Savannah: Welcome. availed at `http://savannah.gnu.org/`, 2000.

[15] Free Software Foundation. Gnu coding standards. availed at `http://www.gnu.org/prep/standards.html`, 2002.

[16] The Apache Software Foundation. The apache http server project. availed at `http://httpd.apache.org/`, 1999.

[17] The Apache Software Foundation. The apache software foundation. availed at `http://www.apache.org/`, 1999.

[18] Ken'ichi Fukamachi. "fml" mailing list server package. availed at `http://www.fml.org/software/fml/index-e.html`, 1993.

[19] Emden R. Gansner and Stephen C. North. An open graph visualization system and its applications to software engineering. *SOFTWARE Practice and Experience*, Vol. 30, No. 11, pp. 1203–1233, 2000.

[20] Arne Georg Gleditsch and Per Kristian Gjermshus. Lxr cross referencer. availed at `http://sourceforge.net/projects/lxr`, 2001.

[21] Object Management Group. Omg home. availed at `http://www.omg.org/`, 1997.

[22] Sven Guckes. The vim (vi improved) home page. availed at http://www.vim.org/, 1998.

[23] 比屋根一雄. オープンソースと政府. availed at http://oss.mri.co.jp/, 2002.

[24] James W. Hunt and M. D. McIlroy. An algorithm for differential file comparison. Technical report, Comp. Sci. Tech. Rep. # 41, Bell Telephone Laboratories, 1976.

[25] Open Source Initiative. The open source definition. availed at http://opensource.org/docs/index.php, 2002.

[26] Open Source Initiative. Open source initiative - licensing. availed at http://opensource.org/licenses/index.php, 2002.

[27] Open Source Initiative. Open source initiative - welcome. availed at http://www.opensource.org/, 2002.

[28] Ian Jackson. Debian bug tracking software. availed at http://www.chiark.greenend.org.uk/~ian/debbugs/, 1994.

[29] Michael K. Johnson. The linux kernel hacker's guide. availed at http://www.tldp.org/LDP/khg/HyperNews/get/khg.html, 1996.

[30] Brian Kantor and Phil Lapsley. Rfc 977: Network news transfer protocol. availed at http://www.ietf.org/rfc/rfc0977.txt, 1986.

[31] 柏原昭博, 寺井淳裕, 豊田順一. いかにプログラム空欄補充問題を作るか？ 人工知能学会研究会資料, 第 SIG-IES-990102 巻, pp. 9–16, 1999.

[32] Dan Kegel and Huy Le. Apache problem report database. availed at http://bugs.apache.org/, 1999.

[33] Stefan Koch and Georg Schneider. Results from software engineering research into open source development projects using public data. Diskus-

sionspapiere zum Tätigkeitsfeld Informationsverarbeitung und Informationswirtschaft, H.R. Hansen und W.H. Janko (Hrsg.), Nr. 22, Wirtschaftsuniversität Wien, 2000.

[34] 國領二郎. オープン・アーキテクチャ戦略—ネットワーク時代の協働モデル. ダイヤモンド社, 1999.

[35] Cricket Liu and Paul Albitz. *DNS and BIND, 2nd Edition*. O'Relly & Associates, 1996.

[36] SteveMaguire 著, 関本健太郎訳. WRITING SOLID CODE -バグのないプログラミングを目指して-. アスキー, 1995.

[37] Ken Manheimer, Barry Warsaw, and John Viega. Mailman - an extensible mailing list manager using python. In *Proceedings of the 7th International Python Conference*, 1998.

[38] Steve McConnell. *Code Complete*. Microsoft Press, 1993.

[39] Steve McConnell. Open-source methodology: Ready for prime time? *IEEE Software*, Vol. July/August, No. 4, pp. 6–8, 1999.

[40] Sun Microsystems. Javadoc tool home page. availed at `http://java.sun.com/j2se/javadoc/`, 1995.

[41] 藤田充典, 藤枝和宏, 落水浩一郎. オープンソース開発におけるプロジェクトモニタリング手 法. In *Software Symposium 2000*, pp. 126–130. ソフトウェア技術者協会, 2000.

[42] 藤田充典, 藤枝和宏, 落水浩一郎. Cvslogmonitor: オープンソース開発のためのモニタリングツール. 情報処理学会シンポジウムシリーズ, pp. 149–150, 2001.

[43] 藤田充典, 藤枝和宏, 落水浩一郎. オープンソース開発への参加支援. 第 18 回大会論文集. 日本ソフトウェア科学会, 2001.

[44] Audris Mockus, Roy T. Fielding, and James Herbsleb. A case study of open source software development: the apache server. In *International Conference on Software Engineering*, pp. 263–272, 2000.

[45] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transaction on Software Engineering and Methodlogy*, Vol. 11, No. 3, pp. 309–346, 2002.

[46] Audris Mockus and James Herbsleb. Why not improve coordination in distributed software development by stealing good ideas from open source. In *ICSE '02 Workshop on Open Source Software Engineering*, pp. 35–37, 2002.

[47] Geoff Mulligan, 宇夫 陽次朗+藤田 充典訳. SPAM の撃退. ピアソン・エデュケーション, 1999.

[48] Hiroyuki Murakoshi, Akira Shimazu, and Koichiro Ochimizu. Construction of deliberation structure in e-mail communication. In *Pacific Association for Computational Linguistics*, pp. 16–28, 1999.

[49] 中村謙. プログラミング言語教材の構成法に関する実験と考察. Master's thesis, 北陸先端科学技術大学院大学 情報科学研究科情報システム学専攻, 2002.

[50] 福安直樹, 山本晋一郎, 阿草清滋. 細粒度リポジトリに基づいた case ツール・プラットフォーム sapid. 情報処理学会論文誌, Vol. 39, No. 6, 1998.

[51] Netcraft. Netcraft web server survey. availed at `http://www.netcraft.com/survey/`, October 1995.

[52] Open Source Development Network. Sourceforge.net: Welcome. availed at `http://sourceforge.net/`, 2002.

[53] J. Oikarinen and D. Reed. Rfc 1459: Internet relay chat protocol. availed at `http://www.ietf.org/rfc/rfc1459.txt`, 1993.

[54] OpenBSD. Openssh. availed at `http://www.openssh.com/`, 1999.

[55] The Mozilla Organization. Bugzilla project home page. availed at http://www.bugzilla.org/, 1998.

[56] The Mozilla Organization. Getting cvs write access to mozilla. availed at http://www.mozilla.org/hacking/getting-cvs-write-access.html, 1998.

[57] The Mozilla Organization. tinderbox. availed at http://www.mozilla.org/tinderbox.html, 1998.

[58] Rational. Software configuration management and effective team development ith rational. availed at http://www.rational.com/products/clearcase/, 2002.

[59] Eric S. Raymond. The cathedral and the bazaar. availed at http://catb.org/~esr/writings/cathedral-bazaar/, May 1997.

[60] Eric S. Raymond. fetchmail home page. availed at http://catb.org/~esr/fetchmail/, 1999.

[61] リチャードブラム著, コスモプラネット訳. Postfix メールサーバの構築. アスキー, 2002.

[62] リチャードブラム著, コスモプラネット訳. qmail メールサーバの構築. アスキー, 2002.

[63] JanSandred 著, でびあんぐる監訳. オープンソースプロジェクトの管理と運営. オーム社, 2001.

[64] Atsushi Sato. メーリングリスト管理ソフト一覧. availed at http://www.kt.rim.or.jp/~atsato/ml/basic/software.html, 1997.

[65] 佐藤隆博. 構造学習法の入門〜コンセプトマッピング・アプローチ〜. 明治図書出版株式会社, 1996.

[66] IETF Secretariat. Ietf home page. availed at http://www.ietf.org/, 1996.

[67] Perforce Software. Perforce software - the fast software configuration management system. availed at http://www.perforce.com/index.html, 1996.

[68] Richard M. Stallman. Categories of free and non-free software. availed at http://www.gnu.org/philosophy/categories.html, 1996.

[69] Yngve Svendsen and Milan Zamazal. Gnats. availed at http://www.gnu.org/software/gnats/gnats.html, 2001.

[70] The GCC Team. Simple gcc projects. availed at http://gcc.gnu.org/projects/beginner.html, 2002.

[71] The GIMP Team. Gimp developer resources. availed at http://developer.gimp.org/, 2003.

[72] The GIMP Team. The gimp homepage. availed at http://www.gimp.org/, 2003.

[73] The Source-Navigator team. Source-navigator. availed at http://sources.redhat.com/sourcenav/, 2002.

[74] Andrew Tridgell and Dan Shearer. Jitterbug - main page. availed at http://samba.org/jitterbug/, 1997.

[75] Dimitri van Heesch. Doxygen. availed at http://www.stack.nl/~dimitri/doxygen/, 1997.

[76] A. von Mayrhauser and A. M. Vans. Program understanding - a survey. Technical report, Technical Report CS-94-120 Colorado State University, 1994.

[77] 渡邊大貴. 電子メールコミュニケーションにおける討議内容の要約と呈示法について. Master's thesis, 北陸先端科学技術大学院大学 情報科学研究科情報システム学専攻, 2002.

[78] Brian A. White. *Software Configuration Management Strategies and Rational ClearCase: A practical Introduction*. Addison-Wesley, 2000.

[79] Brian W.Kernighan and Rob Pike. *The Practice of Programming*. Addison Wesley, 1999.

[80] 山見太郎. 電子メールを用いたコミュニケーションに対する支援環境の設計と実現. Master's thesis, 北陸先端科学技術大学院大学 情報科学研究科情報システム学専攻, 2001.

[81] 結城浩. Yukiwiki. availed at `http://www.hyuki.com/yukiwiki/`, 2000.

[82] Jamie Zawinski. resignation and postmortem. availed at `http://www.jwz.org/gruntle/nomo.html`, 1999.

[83] Henner Zuhause. Hen's cvsweb cvs repository. availed at `http://stud.fh-heilbronn.de/~zeller/cgi/cvsweb.cgi/`, 1996.

本研究に関する発表論文

論文

[1] 藤田 充典、藤枝 和宏、 落水 浩一郎: “オープンソース開発におけるプロジェクトモニタリング手法”，ソフトウェア・シンポジウム 2000 論文集，pp.126-130(平成 12 年 6 月).

[2] 藤田 充典、権藤 克彦、 落水 浩一郎: “現実的なプログラミング問題作成手法の提案”，日本ソフトウェア科学会 FOSE 2002，pp.95-106(平成 14 年 11 月).

[3] 藤田 充典、権藤 克彦、 落水 浩一郎: “オープンソースプログラミング演習問題の構築”，情報技術レターズ (情報科学技術フォーラム (FIT) 講演論文集, pp.41-42(平成 15 年 9 月).

国際会議

[1] M. Fujita, K. Fujieda, K. Gondow, K. Ochimizu: “Support for Joining in Open Source Software Development Projects Using Public Data”, ISFST, CD-ROM(平成 14 年 10 月).

[2] K. Ochimizu, H. Murakoshi, K. Fujieda, M. Fujita: “Sharing Instability of a Distributed Cooperative Work”, ISPSE 2000, pp.36-45(平成 12 年 11 月).

解説記事

[1] 藤田 充典、藤枝 和宏、山本 晋一郎、落水 浩一郎、阿草 清滋: “特集: ソフトウェア発展 オープンソースとソフトウェア進化”，bit, vol.32, No.12, pp.21-27(平成 12 年 12 月).

口頭発表

[1] 藤田 充典、藤枝 和宏、落水 浩一郎: “共同開発の作業履歴から観測される事象とその考察”，電子情報通信学会技術報告，SS99-34(平成 11 年 9 月).

[2] 藤田 充典、藤枝 和宏、 落水 浩一郎: “オープンソース開発における作業履歴からの情報抽出と視覚化”，電子情報通信学会技術報告，SS99-50(平成 12 年 1 月).

[3] 藤田 充典、藤枝 和宏、 落水 浩一郎: “オープンソース開発におけるプロジェクトモニタリング手法”，電気関係学会北陸支部連合大会，平成 12 年度 講演論文集, (平成 12 年 9 月).

[4] 藤田 充典、藤枝 和宏、 落水 浩一郎: “オープンソースソフトウェアの開発プロセスについて”，情報処理学会シンポジウムシリーズ，Vol. 2001，No. 2，pp.45-46(平成 13 年 1 月).

[5] 藤田 充典、藤枝 和宏、 落水 浩一郎: “CVSLogMonitor: オープンソース開発のためのモニタリングツール”，情報処理学会シンポジウムシリーズ，Vol. 2001，No. 5，pp.149-150(平成 13 年 3 月).

[6] 藤田 充典、藤枝 和宏、 落水 浩一郎: “オープンソース開発への参加支援”，日本ソフトウェア科学会第 18 回大会，CD-ROM(平成 13 年 9 月).

[7] 藤田 充典、権藤 克彦、 落水 浩一郎: “バグトラッキングシステムを用いたプログラミング技術学習法について”，日本ソフトウェア科学会第 20 回大会, CD-ROM(平成 15 年 9 月).

出版

[1] ジェフ・モリガン (著)、宇夫 陽次朗、藤田 充典 (訳): “SPAM の撃退 (Sendmail, Procmail の設定とメールフィルタリング)”，ピアソン・エデュケーション，(平成 11 年 12 月).

第 A 章

オープンソースソフトウェア

A.1　オープンソースソフトウェアとは

オープンソースソフトウェア (OSS) の解釈は多々あるが、一言で述べるならば「ソースコードを公開しているソフトウェア」のことを指す。OSS には以下を含む:

- ソースコード
  ソフトウェアを構成するプログラムやライブラリなどを実現する
- ドキュメント
  ソフトウェアの利用マニュアル、構築方法、ライセンスなどを説明する

ソースコードを公開してソフトウェアを開発する活動をオープンソースソフトウェア開発 (OSSD) と呼ぶ。本論文で扱う OSSD とは単にソフトウェアのソースコードを公開しているだけではなく、開発の作業履歴や旧バージョンのソースコード、メーリングリストのアーカイブなど開発の行程で蓄積された情報についても公開している開発スタイルを指す。

オープンソースソフトウェアの利点

ソースコードまで吟味して理想に近いソフトウェアを選ぶことができる。自分でバグの修正や改良を施せる。セキュリティの欠点を見付けられる。

Open Source Initiative による定義

オープンソースソフトウェアを普及させ、その開発戦略を練るための団体として発足した Open Source Initiative[27] は、OSD (Open Source Definition) と呼ばれるオープンソースの定義 [25] を作成している [63]。OSD によれば、オープンソースとはソースコードへのアクセスだけを意味するのではなく、オープンソース (のプログラム) として認められるためには以下に示す項目の条件を満たす必要がある:

1. 再頒布の自由
2. ソースコード
3. 派生ソフトウェア
4. 作者のソースコードの完全性
5. 個人やグループに対する差別の禁止
6. 利用する分野に対する差別の禁止
7. ライセンスの分配
8. 特定製品でのみ有効なライセンスの禁止
9. 他のソフトウェアを制限するライセンスの禁止

Free Software Foundation による定義

Free Software Foundation (FSF) はソフトウェアのソースコードを公開し、ソフトウェアを自由に利用できるようにするフリーソフトウェア運動を最初に起こした団体である [10]。FSF では GNU Software と呼ばれるソフトウェア群の開発、公開、保守及びそのサポートを行っている。FSF では以下に示す 4 つの自由を有するソフトウェアをフリーソフトウェアと定義している [13]:

- 目的を問わず、プログラムを実行する自由 (第 0 の自由)
- プログラムがどのように動作しているか研究し、そのプログラムにあなたの必要に応じて修正を加え、採り入れる自由 (第 1 の自由)

- 身近な人を助けられるよう、コピーを再頒布する自由 (第 2 の自由)
- プログラムを改良し、コミュニティ全体がその恩恵を受けられるようあなたの改良点を公衆に発表する自由 (第 3 の自由)

この中で第 1 と第 3 の自由によってフリーソフトウェアのソースコードが公開されていることが保証されている。図 A.1 に示すのはフリーソフトウェアとオープンソース、他のライセンスの関係を説明する際に良く用いられる Chao-Kuei によるソフトウェアの分類である [68]。また、FSF では GPL (The GNU General Public License) や LGPL (The GNU Lesser General Public License) などのライセンスを提唱し、ユーザがフリーソフトウェアやドキュメントを自由に利用する権利を保証する活動を行っている。

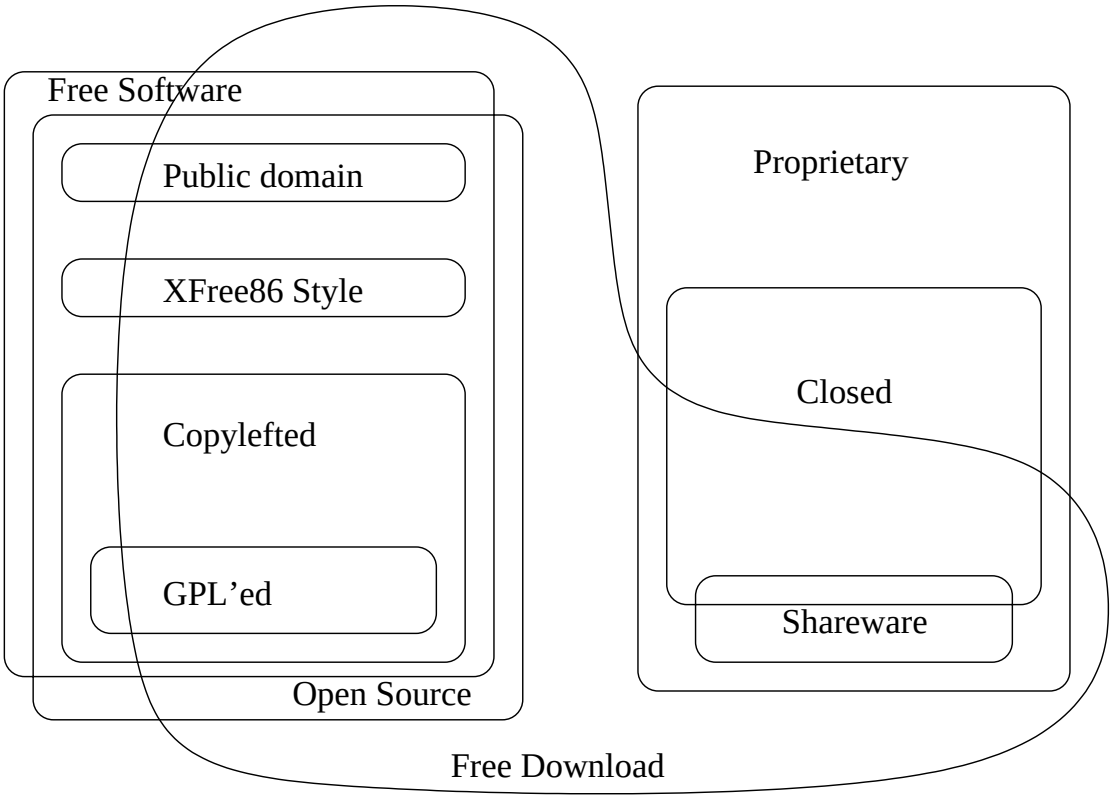


図 A.1: ソフトウェアの分類

A.2　オープンソースソフトウェアの普及

今日の情報社会において OSS は既に欠かせないものになっている。

コンピュータの普及と通信技術の発達に伴って、取り引き先やオフィス間の連絡が電子メールや電子会議システムで行われるなど、コンピュータ間の相互接続が進められている。そのため異なるベンダによって開発されたソフトウェアやオペレーティングシステムの間でも通信が可能であることが強く望まれている。

このようなコンピュータ間の相互接続のための通信規格は、特定企業に市場を独占されないために IETF (The Internet Engineering Task Force)[66] や OMG (Object Management Group)[21] などの比較的オープンな団体もしくは、複数の企業が合同で設立した団体によって策定されている。同様にインターネットの基盤技術を支えるソフトウェアに関しても、特定企業の方針に振り回されることがない、オープンなものが望まれている。例えば、インターネット上に点在するコンピュータの名前解決を行う DNS (Domain Name System) サービスには、Internet Software Consortium の OSS である BIND[35] が最も多く使われている[1]。インターネット技術の中で最も有用なサービスの 1 つに電子メールがある。この電子メールを配信する MTA (Mail Transfer Agent) にも、OSS である Sendmail[8] や postfix[61]、qmail[62] が広く使われている。また、Web ページを公開するための HTTP (Hyper Text Transfer Protocol) サービスにおいても、やはりオープンソースソフトウェアの Apache[16] が最も広いシェアを持っていることが Netcraft の調査 [51] で明らかになっている。

さらに、近年ではイギリスやフランス、中国、インドを始めとする各国で政府主導による OSS 推進の動きが見られるようになった [23]。また、日本やドイツでも OSS を見直す動きが起こっている。

情報社会の基盤を支えている OSS を改めて見直し、その開発を支えることは非常に有益なことだと考えられる。

[1] OSS ではない実装 djbdns も知られている:http://www.linux.or.jp/JF/JFdocs/DNS-HOWTO-2.html

A.3 オープンソースソフトウェア開発をサポートすることの有効性

OSSD はその多くが無償の活動であるため、単なるボランティア活動だと思われがちである。しかし、OSS が広く一般に知れ渡ることで、社会的な名声を得ること以外にも参加者達に有益な効果をもたらすと考えている。OSS の開発者を増やし、育てることの意義を以下に示す:

- 質の高いソフトウェアを増やす
  高機能でバグの少ない OSS が開発されるようになる。

- ソフトウェア技術者の不足を解消する
  OSSD に参加することで優れた開発者達の技術を習得し、業務に役立てることができる。

- ソフトウェアに対する正しい理解を広める
  特定のベンダの製品を使い続けることで、良いソフトウェアを選ぶという当然の認識が崩れている。OSS を利用し、開発に参加することで OSS の機能性や安定性、導入コストなどを知ることができる。

- OSS 導入の動きを助ける
  OSS の安定性や安全さを確認することで、政府が OSS を選択した正しい理由を知ることができる。

OSSD をサポートする動き

オープンソースソフトウェア開発を支援を目的として、オープンソースソフトウェア開発に必要とされているシステムを無償で提供する OSDN (Open Source Development Network) の SourceForge.net[52] や FSF の Savannah[14]、BitMover のホスティングサービス [2] などがある。

A.4　オープンソースソフトウェア開発の特徴

OSSD では開発者達がユーザからの貢献を受け取り、採り入れることでソフトウェアを発展させるという特徴を持つ。ここでの貢献には以下のものが含まれる:

- 問題報告
- ソフトウェアに対する改良案
- ソースコード
- 時間

Eric Raymond は自身が fetchmail[60] という OSSD プロジェクトを率いることで、ソフトウェアのユーザを開発者としてプロジェクトに採り入れるという、OSS の開発スタイルの有効性を示した [59]。

OSSD では以下に示すようなサイクルを生み出すことでソフトウェアの発展が期待されている:

- OSSD プロジェクトは、一からソフトウェアを開発するのではなく、誰かが作成したソフトウェアの開発及び保守を共同でサポートする目的で立ち上げられる場合が多い。彼等は、自分達が開発しているソフトウェアの魅力と、自分達の活動をアピールするためにプロジェクトの Web ページ (ホームページ) を作成する。
- ホームページにはソフトウェアに関する説明、最近の出来事、開発の現状、ソフトウェアの入手方法、コミュニティとの連絡の取り方、貢献の仕方などが紹介されている。プロジェクトの活動によって、ホームページがインターネット上の検索エンジンやニュースサービスなどから辿れるようになると、潜在的なユーザの目に触れる機会が増える。
- ユーザは OSS に不満を感じると他のソフトウェアを探すか、プロジェクトに問題の修正や機能の追加を求める。この際、ソースコードが明らかであるためにユーザ自身が改良を行うことが可能であり、その変更内容をプロジェクトに送ることができる。ユーザはやがて開発者としてプロジェクトに参加し、ソフトウェアの機能を高めることで新たなユーザを引き寄せる。

A.5 オープンソースソフトウェア開発プロジェクトの公開情報

Apache Software Foundation のメンバーでもある Fielding ら [11] によれば、効果的な大規模コラボレーションを進めて行くには次のものが要求される:

- 将来の貢献者を含む大勢の人々
- 少なくても 1 つの共通目標
- 後で参照可能なパブリックな目的及びプライベートな目的のコミュニケーション手段
- 現在と過去の成果物及びコミュニケーションの履歴にアクセスできる共用の情報空間
- 上記の項目を協調させるコーディネーション

そして Apache の開発では以下のツール及び意志決定の手段を採用している:

- メーリングリスト (Developer Mailing List)
  主要なコミュニケーションの手段として用いられる。 普段は日に 50 通程度だがリリースが近くなると 200 通を越える。その内容には、新機能のデザイン、バグの修正、ユーザの抱える問題、Web コミュニティやリリースの日程やプロジェクトの strategy などのニュースがある。
- Change Control
  開発の初期にはプロジェクトに関する議論をメールと FTP を用いて行い、決定事項をメーリングリストに流していた。投票による意志決定を行う。
- 版管理システム (CVS)
- バグデータベース (GNATS)[2]

[2]現在は Bugzilla を採用している

- SSH (Secure SHell)[54]
  ネットワークを介して情報にアクセスする際にセキュリティのリスクが高まるため、CVS リポジトリへのアクセスの暗号化を可能にする SSH のような技術が望まれる。

以降、OSSD プロジェクトで一般的に用いられている版管理システム、メーリングリスト、バグデータベースについて紹介する。

版管理システム

版管理システムはソフトウェアを構成するソースコードやドキュメントの管理を行う。版管理システムの主な役割は、ソフトウェアのリリースのサポートと、登録されたソースコードやドキュメントなどの変更管理である。また、版管理システムの管理情報やコンテンツを格納する場所をリポジトリと呼んでいる。

ソースコードの変更管理

版管理システムはソフトウェアを構成する各ソースコードのバージョンを管理する。版管理システムにおけるソースコードのバージョンとは、仕様の変更や問題の修正に伴って編集されたソースコードを区別する。版管理システムに登録されたソースコードに何らかの変更を加えることをバージョンの更新と呼ぶ。あるバージョンのソースコードと別のバージョンのソースコード間の違いを変更差分と呼ぶ。変更差分には様々な表現方法があるが本論文の趣旨とは異なるので子細は省く。一般には *diff* コマンド [24] で得られるソースコード間の行単位の違いの集合を指す場合が多い。

ソフトウェアのリリース

ソフトウェア開発におけるリリースとは、開発目標として掲げた仕様を実現するバージョンのソースコード集合を構成することである。この構成をリリースバー

ジョンと呼び、ソフトウェアの名前に英数字や記号として付加されることが多い[3]。ソフトウェア開発者はリリースバージョンを指定することで版管理システムから望むソフトウェアの構成を得ることができる。リリースにはソフトウェアに大規模な変更が加えられた場合や、大きな目標を達成した際に用いられるメジャーリリース、メジャーリリースを基に機能追加や安定性の向上を進めるマイナーリリース、小規模な変更でリリースを繰り返すために用いられるマイクロリリースがある。多くのオープンソースソフトウェアでは、「ソフトウェア名-メジャーリリース. マイナーリリース. マイクロリリース」(例: apache-1.3.1) の様にバージョンが示されている。

オープンソースソフトウェア開発に用いられる版管理システム

一般的な版管理システムの分類を以下に示す。

1. 企業が開発した有償のシステム
   有償のライセンス契約を結ぶことで利用可能な、主にソースコードを公開しないソフトウェア開発で用いられる版管理システムである。他製品と連携することで、ソフトウェア開発の一貫したサポートを可能にするシステムもある。トランザクションやセキュリティ管理等、高度な機能を備えるものや、専門家によるサポートやトレーニング等がライセンスに含まれるものがある。

2. 企業の提示するライセンスに従うことを条件に無償で利用できるシステム
   企業の提示する条件を満たすことで無償で利用できる有償のシステムと同等の機能を備えた版管理システムである。主な条件は (1) 商用ソフトウェアの開発に利用しないこと、(2) ソースを公開すること、(3) 特定のオープンソースライセンス [26] に従っていること等である。

3. 企業が無償で提供しているシステム
   企業によって開発され、無償で提供されている版管理システムである。オープンソースライセンスの下でソースコードが公開されている場合もある。

[3]新しさを示す基準として一般にインクリメンタルな数字が用いられるが、Tex のように桁数が増えるものや Microsoft 社の OS のように表記が変わるものもある

4. オープンソースソフトウェアとして開発されたシステム
   オープンソースソフトウェア開発プロジェクトで開発された版管理システムである。野心的な機能の実現を掲げたシステムや、先進的な機能が取り込まれているシステムがあるが、実験的な意味合いが強いものが多い。

分散共同開発が行える主要な版管理システムの一覧を表 A.1 に示す。表 A.1 で

表 A.1: 主要な版管理システム

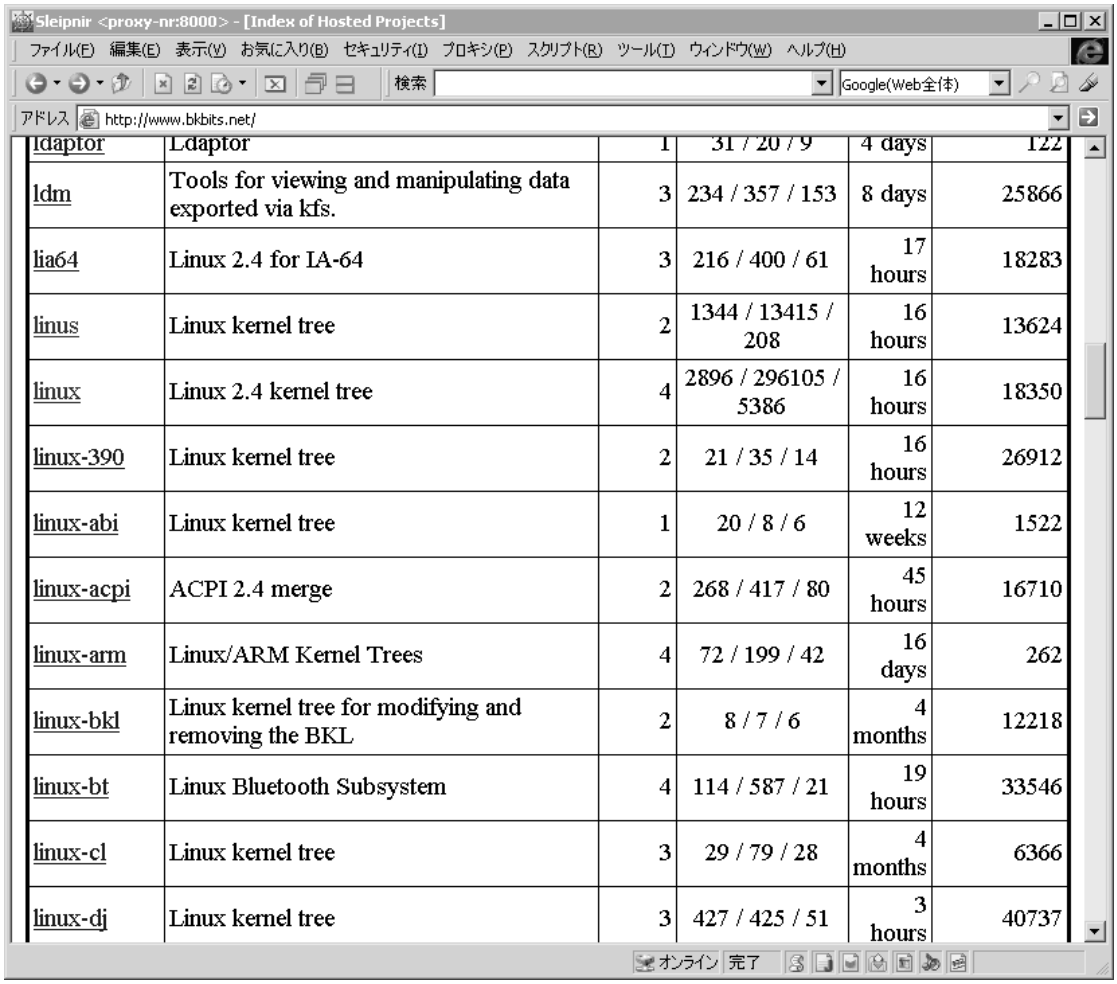
分類	名称	提供者 or ホスティング	ソース	利用条件	OSSD での利用
1	Visual SourceSafe[7]	Microsoft Corporation	非公開	有償	ない
1	ClearCase[58, 78]	Rational	非公開	有償	ない
2	BitKeeper([2]	BitMover	非公開	制限あり	少ない
2	Perforce[67]	Perforce Software	非公開	制限あり	少ない
3	CVS[1, 5]	CollabNet	公開	無償	多い
4	Subversion[6]	CollabNet	公開	無償	少ない

は先に述べた版管理システムの分類、版管理システムの名前、システムの提供もしくは開発のホスティングを行っている社名、ソースコードが公開されているか否か、OSSD で利用する条件、採用している OSSD プロジェクトの多さを示している。BitKeeper は BitMover の提供するホスティングサービスの条件を満たし、ホステッドプロジェクトとして登録された場合にのみ無償で利用できる (図 A.2)。また、Perforce については OSD の定義を満たす OSS に限り無償で利用できる。

メーリングリスト

メーリングリストとは、単一のアドレスを用いて複数の受信者に電子メールを配送する仕組みである [47]。特定の話題に関する議論や情報提供などに用いられることが多い。

ほとんどのメーリングリストはメーリングリスト管理ソフトを用いて運営されている。OSSD などで用いられているメーリングリスト管理ソフトの例を以下に示す [64]。



ldaptor	Ldaptor	1	31 / 20 / 9	4 days	122
ldm	Tools for viewing and manipulating data exported via kfs.	3	234 / 357 / 153	8 days	25866
lia64	Linux 2.4 for IA-64	3	216 / 400 / 61	17 hours	18283
linus	Linux kernel tree	2	1344 / 13415 / 208	16 hours	13624
linux	Linux 2.4 kernel tree	4	2896 / 296105 / 5386	16 hours	18350
linux-390	Linux kernel tree	2	21 / 35 / 14	16 hours	26912
linux-abi	Linux kernel tree	1	20 / 8 / 6	12 weeks	1522
linux-acpi	ACPI 2.4 merge	2	268 / 417 / 80	45 hours	16710
linux-arm	Linux/ARM Kernel Trees	4	72 / 199 / 42	16 days	262
linux-bkl	Linux kernel tree for modifying and removing the BKL	2	8 / 7 / 6	4 months	12218
linux-bt	Linux Bluetooth Subsystem	4	114 / 587 / 21	19 hours	33546
linux-cl	Linux kernel tree	3	29 / 79 / 28	4 months	6366
linux-dj	Linux kernel tree	3	427 / 425 / 51	3 hours	40737

図 A.2: BitMover がホスティングするプロジェクト

- Majordomo[4]
  世界中のメーリングリストで最も多く利用されているメーリングリスト管理ソフト。自動登録、ダイジェスト機能など豊富な機能を有する。Perl で実装されている。

- GNU Mailman[37]
  多くのオープンソースソフトウェア開発を抱える SourceForge や Savannah で採用されているメーリングリスト管理ソフト。全ての管理作業を Web ページを通して行うことができる。Python で実装されている。

- fml[18]
  日本で作られた高機能メーリングリスト管理ソフト。メールを使った自動管理機能や投稿集計など運営に役立つ豊富な機能を備える。Perl で実装されている。

OSSD プロジェクトではメーリングリストを以下に示す目的で用いることが多い:

- ソフトウェアに関する質問及び議論[4]
- ユーザからの要望や問題報告の受付
- プロジェクトからのアナウンス
- 主要開発者達による意志決定
- 版管理システムに登録されたソースコードの更新通知

多くの OSSD プロジェクトではメーリングリストで行われた議論の内容を後から参照できる様にアーカイブとしてまとめて公開している場合が多い (図 A.3)。

[4]他に IRC (Internet Relay Chat)[53] や USENET[30] の適切なニュースグループ (例: comp.graphics.apps.gimp など)、プロジェクトが提供する他の討議システムを用いて議論を行っている場合がある

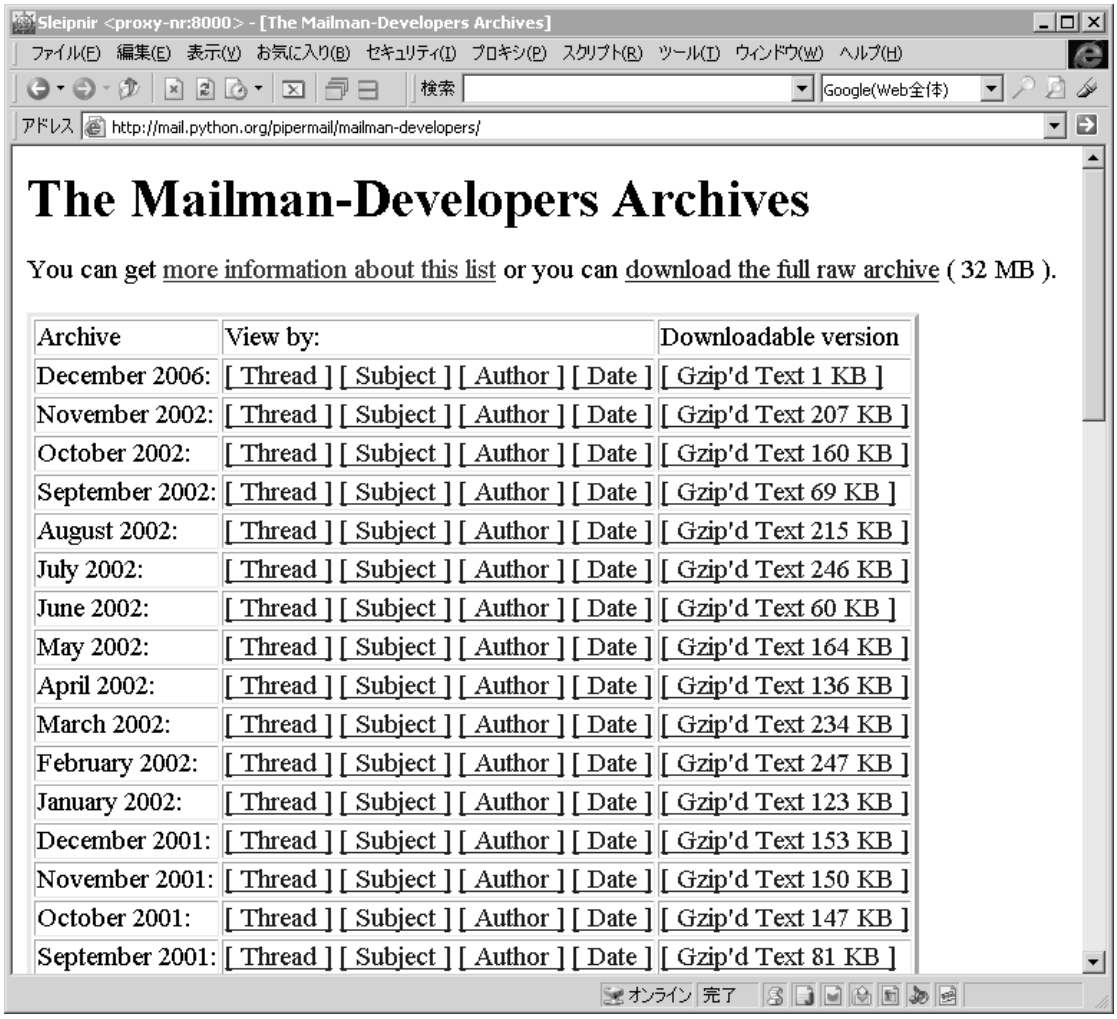


図 A.3: Mailman のメーリングリストアーカイブ

バグデータベース

バグデータベースとは、ソフトウェアに関する発見された問題 (バグ) を登録し、後から参照可能にするシステムである。問題報告以外にも改良案や新機能の提案など利用されることがある。

OSS のバグデータベースの実装としては GNU Software である GNATS[69]、Mozilla の問題報告を扱うために開発された Bugzilla[55]、Samba の問題報告を扱うために開発された JitterBug[74]、Debian の問題報告を扱うために開発された debbugs[28] などが知られている。表 A.2 に各々の特長を示す。ここで挙げたバグ

表 A.2: 各バグデータベースの特長

名称	ライセンス	報告手段	検索機能	匿名アクセス	採用しているプロジェクト
GNATS	GPL	Web	高機能	不可	Apache
Bugzilla	MPL	Web	高機能	閲覧のみ	Mozilla, Linux kernel, Apache
JitterBug	GPL	Web	弱い	閲覧のみ	linux-patches
debbugs	GPL	メール	高機能	可能	Debian

データベースと連携して、バグの報告や閲覧をサポートするソフトウェアも存在する。

GPL と比較してライセンスの制約が少ない MPL (Mozilla Public License)[5] を採用していることに加え、他のバグデータベースと比較して開発が盛んで、検索などの機能が豊富な Bugzilla が一般によく採用されている (図 A.4)。

[5]新規に追加したソースコードの公開義務がない

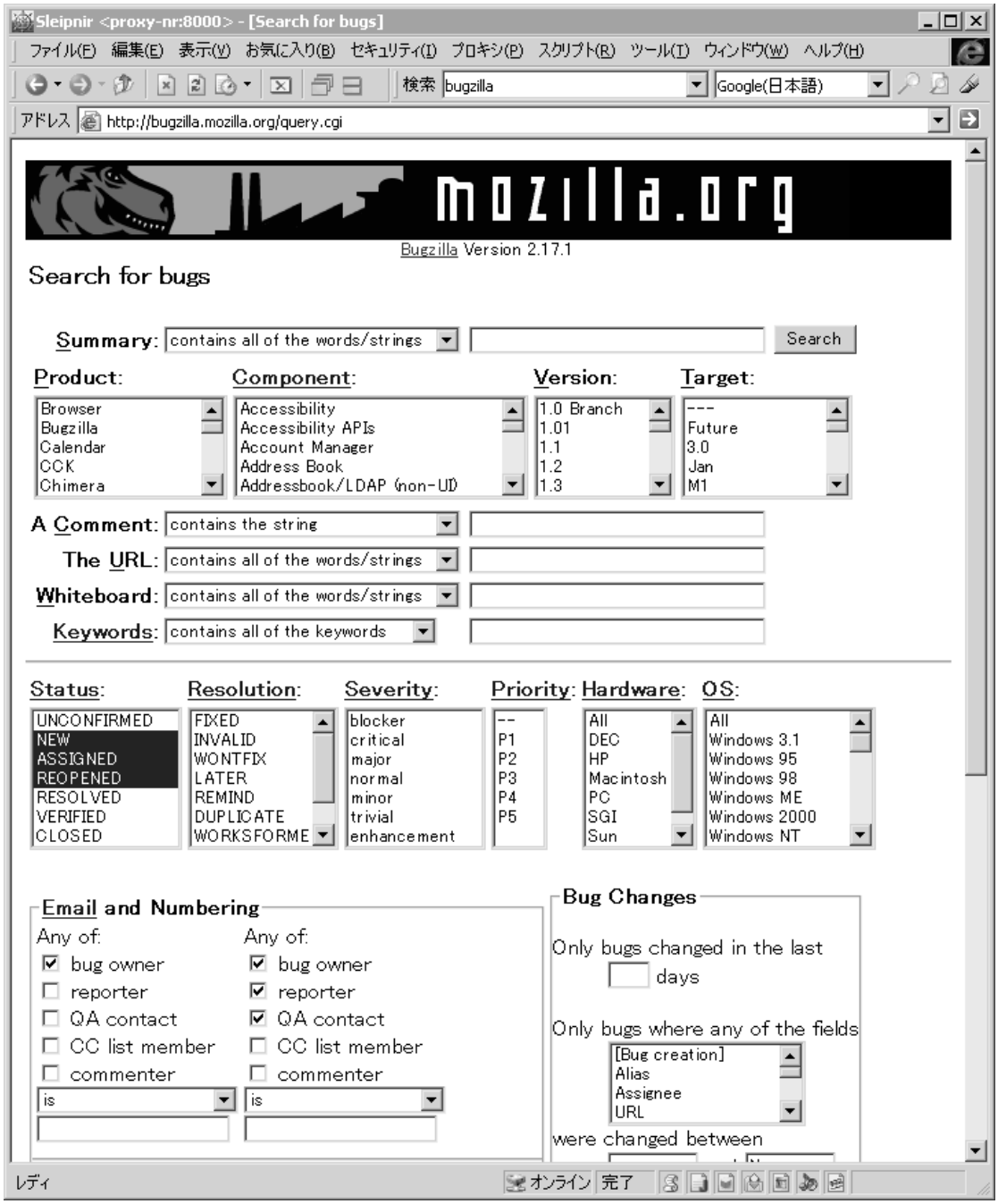


図 A.4: Bugzilla の検索画面

第 B章

CVS の log コマンドの出力

CVS の *log* コマンドによる出力はファイルに関する情報が含まれた前半部分と、リビジョンツリーに相当する後半部分に分けられる。後半部分の出力順はツリー毎に新しいリビジョンから古いリビジョンへと並べられる。また、ブランチに関してはツリーごとに後から別れたブランチほど後方に並べられる。

以下、*log* コマンドによって出力される前半部分と後半部分のそれぞれの情報について説明を加える。

前半部分に含まれる情報の説明

図 B.1 に示すのは *log* コマンドによる前半部分の出力例である。なお、左端の数字は後で説明を加えるための行番号であり、実際の出力には含まれない。

前半部分の出力に含まれている情報について説明を加える。括弧内の数字は先に示した出力例の該当する行番号を示す。

- RCS ファイル (1)
  実際にリポジトリに保存されている RCS ファイルを示している。例ではリポジトリサーバの「/home/CVSLog/EditLine.java,v」に格納されている。

- ファイル名 (2)
  作業ディレクトリにおけるファイル名である。例では「EditLine.java」が該当する。

```
1:    RCS file: /home/CVS/CVSLog/EditLine.java,v
2:    Working file: EditLine.java
3:    head: 1.2
4:    branch:
5:    locks: strict
6:    access list:
7:    symbolic names:
8:            test: 1.2
9:            beta: 1.1.1.1
10:           start: 1.1.1.1
11:           begin: 1.1.1
12:   keyword substitution: kv
13:   total revisions: 3;     selected revisions: 3
14:   description:
```

図 B.1: CVS の log コマンドの出力 (前半部分)

- ヘッドリビジョン番号 (3)
  最後に更新されたリビジョンに割り付けられた番号である。例では「1.2」がヘッドリビジョン番号である。

- ブランチのリビジョン番号 (4)
  ファイルから派生したブランチに割り当てられるリビジョン番号である。例では、ブランチが派生していないため空欄である。

- ロック制御 (5)
  ファイルのロック機構の扱いを決める。指定が無ければ例の様に「strict」が指定される。

- アクセスリスト (6)
  ファイルに対する権限を持つ開発者のアカウントのリストである。ファイルアクセスを制限する場合に利用される。例では指定が無いため空欄である。

- タグ情報 (7〜11)
  モジュールをリリースする時に付けられるタグのラベルと、そのタグが割り当てられたリビジョン番号のリスト。例では「test」、「beta」、「start」、「begin」というタグが、それぞれ":"の右側に記されたリビジョンに割り当てられていることを示す。

- キーワード (12)

  checkout コマンドの実行時オプションによって指定されるキーワードである。特に指定が無い場合は例の様に「kv」が指定される。

- リビジョンの総数 (13)

  リポジトリに記録されている該当するファイルのリビジョンの総数である。例では「total revisions: 3」の部分が該当し、ファイルが 3 つのリビジョンを持つことを示している。

- 該当したリビジョンの数 (13)

  log コマンドの実行オプションによる絞り込みにマッチしたリビジョンの数である。例では「selected revisions: 3」の部分が該当する。絞り込みを行わなければリビジョンの総数と等しくなる。

- デスクリプション (14)

  ファイルに対して付加されたコメントである。例では何もコメントが付けられていないため空欄である。

出力の後半部分に含まれる情報の説明

図 B.2 は *log* コマンドの後半部分の出力例である。なお、実際の出力では複数のリビジョンが連結して出力される。そのため、例の 8 行目に出力された横線が境界線としての役目を果たしている。

```
1:    revision 1.41
2:    date: 1999/02/24 06:13:57;  author: amundson;  state: Exp;  lines: +7 -0
3:    branches:  1.41.2;
4:    Wed Feb 24 00:08:42 CST 1999 Shawn T. Amundson <amundson@gtk.org>
5:
6:        * *.[ch]: inserted additional note to look for ChangeLog and
7:           AUTHORS file for a log of modifications.
8:    ----------------------------
```

図 B.2: CVS の log コマンドの出力 (後半部分)

後半部分の出力に含まれている情報について説明を加える。括弧内の数字は先

に示した出力例の該当する行番号を示す。

- リビジョン番号 (1)
  ファイルの更新情報に対して割り当てられる番号である。例では「1.41」がリビジョン番号を示している。

- 更新日時 (2)
  ファイルの更新が行われた日時が記録されている。例では「date: 1999/02/24 06:13:57;」の部分が該当する。なお、指定が無い限りタイムゾーンは GMT が採用される。

- 更新者名 (2)
  更新を行なった開発者のアカウント名が記録されている。例では「author: amundson;」の部分が該当する。

- ファイルの状態 (2)
  ファイルが開発中のものであるか、既に削除されたものであるかを表している。例では「state: Exp;」の部分が該当し、更新時のファイルが開発中であることを示している。なお、ファイルが既にリポジトリから削除されている場合には、「state: dead;」と出力される。

- 編集された行数 (2)
  更新によって追加／編集された行数と削除された行数を表している。例では「lines: +7 -0」の部分が該当し、”+”で始まる数字が追加／編集された行数を表し、”-”で始まる数字が削除された行数を表している。

- ブランチのリビジョン番号 (3)
  このリビジョンから派生したブランチツリのリビジョン番号を表す。例では「1.41.2」がこれに該当する。派生したブランチはこのリビジョン番号を元にブランチツリーを構成する。

- メッセージ (4〜7)
  更新者がリポジトリに更新を反映させる時に残したコメントである。例では更新日時から始まり、更新者の名前や更新内容についてのコメントを残している。

第 C章

演習問題作成に関する Apache の問題報告の評価

C.1 問題報告の評価

表 C.1,C.2 は Apache Problem Report Database[32] から獲得した問題報告 (pr) について、演習問題作成に関する評価を行った結果である。表の見方を以下に示す:

- PR
  バグデータベースによって付与された PR の識別番号
- Severity
  バグデータベースに登録された PR の深刻度
- 評価
  演習問題を作成するために次のような評価を行った
  - ◎:4.2.1 節の特徴を備えている PR
  - ○:解答例となり得るパッチが付与されている PR
  - △:開発者達によって正しく処理されていない PR
  - ×:抽出ミス
  - 表記無し:演習問題に相応しくないと判断した PR

- 要約と備考

  問題の発生するプラットフォーム、問題報告の Description の要約と対応状況、パッチの有無

表 C.1: 問題報告の評価 1

PR	Severity	評価	要約と備考
1030	non-critical	○	cgi のパラメタによって apache がクラッシュする。 提案されたパッチによって修正された。
1086	non-critical	△	あるサイズ以上の大きさの画像の読み込みに失敗する。 最近アナウンスされたバージョンで修正済みである。
1007	non-critical		mod_unique_id のドキュメントが欲しい。 追加した。
1008	non-critical		MAP_THRESHOLD, MMAP_SEGMENT_SIZE のドキュメントが欲しい。 追加した。
974	non-critical	△	ProxyRemote で外のホストにアクセスできないことがある。 1.3 のツリーに修正を加えた。
1131	non-critical	△	CGI から gethostbyname() を利用すると失敗する。 次のリリースで修正の予定。
1003	non-critical		新しい実装についてのドキュメントが欲しい。 追加した。
1208	non-critical	△	ErrorLog の指定について書かれたドキュメントが間違えている。 更新した。
1309	non-critical	△	make で出力される build time の出力が正しくない。 報告者からパッチが送られた。

812	non-critical	◎	他のプロキシにフォワードする際に「http://」で始まるURL だけしか正しく処理されない。 報告通りのバグなので修正された。
1205	non-critical		mod_mime_magic に関する修正案集。 これから修正される。
1265	non-critical	△	proxy 周りの設定がおかしい。 MS Windows に関連したバグで、開発者用ツリーでは修正されている。
1282	non-critical	○	unixware で SIGHUP を送った後の挙動がおかしいのでパッチを作成しました。 機種依存の設定が必要になる。パッチを当てて修正された。
1155	non-critical	◎	RedirectMatch で URL を指定する時に、%20(スペース) が入るとうまく処理されないのでパッチを作成しました。 報告通りのバグなのでパッチを当てて修正された。
1214	non-critical		SunOS には strtoul() がありません。 使わないように修正した。
1211	non-critical		unblock_alarms() で競合が起こってもすぐ終了しないようにすべき。 改善した。
1228	non-critical		palloc() は失敗しないので条件文は無駄になる。 そのように修正した。
1294	non-critical	◎	MMAP_THRESHOLD より小さなファイルがことごとくエラーになるのはバグだと思う。 報告通りのバグなので修正するパッチを紹介。
1258	non-critical	△	Windows NT で make 時にエラーがでる。 次のリリースで修正。

1261	non-critical		SuppressColumnSorting オプションが欲しい。 修正した。
1273	non-critical	○	Monitoring system によってシステムが落ちていると報告される。 紹介したパッチを試すか次のバージョンでの修正を待ってもらう。
1248	non-critical	△	Location の中で IP によるアクセス制御がうまくいかない。 記述方法に問題があるかもしれない。パッチによって修正された。
1293	non-critical		ログのエラーメッセージにはもっと情報が含まれていた方が良い。 1.3b3 で修正の予定。
1249	non-critical		syslog、openlog への参照が解決しない。 OS に依存する問題。修正した。
904	non-critical		mod_mime_magic が圧縮したファイルを読むと SIGSEGV によって終了してしまう。 次のリリースで修正予定。
928	non-critical		scoreboard ファイルを開けないため apache が起動しない。 開発者用ツリーで修正した。
1094	non-critical		mod_status が機能しない。 すぐに出る次のリリースで修正の予定。
1266	non-critical		DirectoryIndex が MS Windows95 でうまく機能しない。 この問題は次の MS Windows 用のリリースで解決する予定[1]。
1160	non-critical		Proxy が正しく動作しない。 CacheDirLength の設定を 20 以下にする必要がある。

[1]diff の出力を調べると KnownBUGS に含められている

1284	non-critical		UNIQUE_ID がエクスポートされていないのはおかしい。 1.3b3 に向けて修正した。
1244	non-critical		CGI がテキストモードで処理される。 次のリリースで修正の予定。
1009	non-critical		post_read_request が未対応のままである。 他の方法で対応した。

表 C.2: 問題報告の評価 2

PR	Severity	評価	要約と備考
1282	critical		unixware 限定の問題。 パッチが報告されていて修正可能。
1291	serious	○	CGI のエラーが全部 ”500 Internal Server Error” になるのはおかしい。 本質ではないが修正が必要な場合もある。パッチ付き。
1293	non-critical		エラーメッセージの情報量をもっと増やして欲しい。 1.3b3 で修正の予定。
1317	non-critical	○	aix 限定の問題。AIX 1.x はファイルやディレクトリの名前に ”@” を使えない。 パッチ付き。1.3b3 で取り込まれる予定。
1319	critical	◎	RedirectMatch で ”/” を指定されると SIGSEGV が発生する。 パッチ付き。パッチは 1.3b4 で適用される予定。
1328	serious	○	access.conf で指定した通りにうまく動かない。 パッチ付き。1.3b3-dev に修正が反映される。
1342	non-critical	○	3000 年になったときに問題が発生する[2]。 パッチ付き。
1343	non-critical	○	suexec log に 2000 年問題がある。 パッチ付き。1.3b4-dev で修正の予定。
1348	non-critical		ドキュメントの修正案。 1.3b6 に向けて修正の予定。
1366	non-critical	○	send_fd と send_fd_length が適切な値を返さない。 パッチ付き。1.3b4 で修正の予定。

[2]実際には 2100 年

1373	non-critical		Fancy Indexing を有効にしないと ReadmeName が表示されない。 1.3b2 以降で修正の予定。
1384	critical		PR1328 を参照。
1399	non-critical	×	PR の選択ミス。 1.2.6 で修正の予定。
1423	non-critical		ドキュメントの修正案。
1440	non-critical		Rewrite がうまくいかない。 ユーザ側のミス。
1445	non-critical		ドキュメントの修正案。
1454	serious	△	報告された内容は解釈に問題がある。 対応状況が不明。
1479	non-critical		Module はログを出力するべき。 1.3b3 以降のリリースにメッセージを追加した。
1480	non-critical		主要でないかバイナリに組み込まれる指示子は .conf-dist ファイルで保護されるべき。 1.3b3 以降のリリースにメッセージを追加した。
1482	non-critical		os2 限定の問題。パスワードファイルにユーザが追加できない。 cp コマンドの文法の問題。1.3b4-dev と以降のベータバージョンで修正した。
1491	non-critical	○	mmap_handler がサイズ 0 のエラーを出力する。 パッチ付き。1.3b4-dev と以降のベータバージョンで修正した。
1499	critical		unixware 限定の問題。再起動を繰り返していると怪しい挙動を示す。 1.3b4 から は unixware 用 に USE_FCNTL_SERIALIZED_ACCEPT を 追 加 した。

1500	critical	◎	必要なとこまでメモリを解放してしまっているのではないか。 パッチ付き。1.3b4-dev から取り込まれる予定。
1511	non-critical		ドキュメントのパスが間違っている。 ドキュメントを修正した。
1512	non-critical		ドキュメントの例が間違っている。 ドキュメントを修正した。
1534	serious		使えない機能が記述されている。 ドキュメントを修正した。
1542	non-critical	○	linux (sparc) 限定の問題。rlim_t の typedef が重複している。 パッチ付き。1.3b4-dev 以降で修正の予定。
1543	serious	○	suexec 失敗時の厳密なエラーメッセージが欲しい。 パッチ付き。1.3b4-dev 以降で修正の予定。
1544	serious		例がふさわしくない。 既にドキュメントを修正した。
1553	serious		unixware 限定の問題。サーバが固まる。 unixware 用に USE_FCNTL_SERIALIZED_ACCEPT を追加。1.3b4 で修正の予定。
1559	non-critical	△	”index.html.en” は正常に解釈されるが ”index.en.html” がうまく解釈されない。 報告者からの報告がないまま時間切れで終了。
1574	non-critical		ReadmeName と HeaderName が SSI から利用できない。 1.3.6 で修正した。PR#4256 と重複。
1599	non-critical	○	mod_digest が Authorization だけを参照して Proxy-Authorization を参照しないため、digest Authentication を proxy authentication に利用すると必ず失敗する。 パッチ付き。パッチによって修正された。

1613	non-critical		next 限定の問題。コンパイルできない。 1.3b4-dev 以降で修正の予定。
1614	non-critical		next 限定の問題。コンパイルできない。 1.3b4-dev 以降で修正の予定。
1639	non-critical	△	hpux 限定の問題。コンパイルできない。 修正内容は不明。
1644	non-critical		mod_rewrite で Vary header フィールド等が使えない。 1.3.8 で修正した。
1645	non-critical	△	CustomLog フォーマットでタブがスペースに置換えられてしまう。 パッチ付き。1.3.1 で修正の予定。修正内容は不明。
1666	non-critical	○	basic authentication を使う時に "Basic" と書かなければいけない[3]。 パッチ付き。1.3 と 1.2-dev で修正の予定。
1667	non-critical	○	plain text が HeaderName で指定されていた場合に pre が preamble の後では無くて前に来てしまう。 パッチ付き。1.3b3 以降で修正の予定。
1683	non-critical	○	cgi が "Connection: close" と出力した際に close してしまう。 パッチ付き。1.3 と 1.2-dev に取り込まれる予定。
1684	non-critical		RewriteLog を使うと "HostnameLookups on" と同等になる。 1.3b3 以降で修正の予定。
1699	non-critical		並び替えの規準の標準設定を変えたい。 1.3.3 で機能を追加した。
1701	non-critical	◎	UserDir を使った時に " ." や " .." と記述されると障害が起こる。 1.3b6-dev で修正した。

[3] "basic" と書けない

1717	non-critical	○	solaris 限定の問題。コンパイルできない。 パッチ付き。修正内容は不明。
1738	non-critical		変数 cwd が初期化されずに使われている。 1.3b3 で修正の予定。
1771	non-critical	○	poly borders の処理がおかしい。 パッチ付き。修正した。
1772	non-critical	◎	gzip と x-gzip のように同等なものがある場合、同じ拡張子を割り当てられないから統一すべきである。 パッチ付き。修正した。
1799	non-critical	×	問題は継続したまま。
1816	non-critical	○	linux 限定の問題。RLimitMEM が Linux で動かない。 パッチ付き。1.3b6-devl で修正の予定。Linux 側のバグ。
1817	serious		1.2 の Files が 1.3 と互換性がない。 ドキュメントを修正した。
1825	non-critical	○	BrowserMatch がクォートされていても無視される。 パッチ付きだが十分なものではない。後で修正された。
1827	serious	◎	エラーログにエラーメッセージ ”read failed” だけでなくファイル名も出力したい。 パッチ付き。修正した。
1847	serious	◎	ap_cpystrn がおかしい。 パッチ付き。修正した。
1854	critical	△	solaris 限定の問題。PR#1425 と PR#1367 と重複。 デフォルトの設定を変更した。修正されたかどうかは不明のまま。
1888	non-critical	◎	VirtualHost にあわせて TZ を変えたい。 パッチ付き。1.3b6 で修正の予定。
1898	non-critical		ドキュメントではワイルドカード指定が使えなければならない。 1.3.6 以降で修正の予定。

1900	non-critical	○	rputs の計算が正しくない。 パッチ付き。1.3b6-dev 以降で修正の予定。
1904	serious		mod_include から call_exec が呼ばれた際に正しく動作しない。 標準設定を変更した。
1908	non-critical	△	ログの %p が正しく解釈されない。 ドキュメントを修正した。修正されたかどうかは不明のまま。
1921	non-critical	○	linux 限定の問題。regex の EOL が sunos ではマッチするが linux ではマッチしない。 パッチ付き。修正した。
1925	non-critical	○	HostnameLookups が off の場合に間違った環境変数を定義している。 パッチ付き。1.3b6 で修正した。
1963	non-critical		ドキュメントへのリンクが無い。 ドキュメントを修正した。
1973	serious	○	ncr 限定の問題。詳細は不明。 パッチ付き。修正した。
2000	non-critical	◎	multiple Listen を使っていると SIGHUP を送っても子プロセスが古いソケットを継承している。 パッチ付き。1.3b6-dev 以降で修正の予定。
2056	non-critical		SERVER_VERSION に OS の名前が入っているべきである。 1.3b6 以降で AddVersionPlatform 指示子を追加する予定。
2065	non-critical		find_token() の動作がおかしい。 1.3b5-dev 以降で修正の予定。
2078	non-critical	△	VirtualHost の設定がうまくいかない。 ユーザの間違い。

2090	serious	○	LogFormat の内容が継承されない。 パッチ付き。修正されたかどうかは不明のまま。
2093	critical	△	aix 限定の問題。コンパイルできない。 バージョン 4.1 以上なら対応した筈。直ったかどうかは不明のまま。
2147	non-critical	△	ドキュメントの内容がおかしい。 返事がないまま時間切れで終了。
2188	non-critical		compat.h で #define md5 ap_md5 が抜けている。 1.3b6 以降で修正の予定。
2229	non-critical	◎	log の中に正しい IP が残されていないことがある。 パッチ付き。1.3.1 以降で修正の予定。
2249	non-critical		PR#1921 と重複。
2252	non-critical		add_cgi_vars() がメモリリークしている。 1.3b8-dev で修正の予定。
2267	non-critical	○	typo。 パッチ付き。修正した。
2293	critical		next 限定の問題。PR#2316 と重複。
2300	non-critical		ユーザの間違い。
2303	serious		User が正しく動かない。 User 指示子は win32 では作用しない。ユーザの設定次第。
2310	non-critical	○	ncr 限定の問題。OS が認識されない。 パッチ付き。1.3.1-dev で修正の予定。
2312	serious		aix 限定の問題。2534, 2664, 2853, 3042 と重複。 コンパイラの問題。
2316	critical	○	next 限定の問題。PR#2293 の絡みでコンパイルできない。 パッチ付き。1.3.1-dev で修正の予定。

2361	non-critical		linux 限定の問題。dlfcn.h が無い。PR#2434, PR#2377 と重複。 ディストリビューションの問題。
2366	non-critical		PR#2229 と重複。
2377	critical		PR#2361 と重複。
2434	non-critical		PR#2361 と重複。
2524	non-critical		PR#2525 と重複。2361 とも重複か。
2525	serious	△	PR#2434 と重複するが dlfcn.h がない。 1.3.1 以降ではチェックを加える。修正されたかどうかは不明のまま。
2533	non-critical	○	sco 限定の問題。SCO 用の設定が不十分。 パッチ付き。修正した。
2534	serious		aix 限定の問題。Directory が機能しない。 PR#3042 等でも触れるが AIX のコンパイラの問題。
2569	non-critical		linux 限定の問題。PR#2361 と関連する問題。
2664	critical		aix 限定の問題。PR#2312 と同じ。コンパイラの問題。
2853	critical		aix 限定の問題。PR#2312 と同じ。コンパイラの問題。
3026	non-critical		ReadName/HeaderName が固定だと .htm と記述する DOS 等で問題になる。いくつかの PR と重複。 最新版では .htm にも対応する。
3042	serious		aix 限定の問題。PR#2312 と重複。 コンパイラの問題。
3072	non-critical		*で該当しないものまでマッチしてしまう。 1.3.6 以降で修正の予定。
3178	serious	◎	50M 以上のファイルを proxy に通すと落ちてしまう。 PR#2950 に似ている。 パッチ付き。1.3.7 で修正の予定。

3529	serious	△	前にも出ているかもしれないが HeaderName と ReadmeName が直感的に機能しない。 未解決のまま。
3569	non-critical		HeaderName で MultiViews が有効にならない。 修正した。
3657	serious	○	画像が更新されているのに 304 を返す (更新されない)。 1.3.4 以降で修正の予定。
4256	serious	○	PR#1574 と重複。mod_autoindex が正しく動かない場合がある。 パッチ付き。修正した。