

ソフトウェア工学特論

オープンソースソフトウェア(2)

システム工学部 情報通信システム学科 大平 雅雄

スケジュール（大平担当分）

- ▶ 10月3日：オープンソースソフトウェア（1）
- ▶ 10月10日：休み（体育の日）
- ▶ 10月17日：オープンソースソフトウェア（2）
- ▶ 10月24日：リポジトリマイニング（1）
- ▶ 10月31日：リポジトリマイニング（2）

本講義のねらい

- ▶ 情報通信システム開発におけるOSSの重要性を理解し、OSS開発に関する基礎知識を習得すること
 - ▶ オープンソース開発の歴史
 - ▶ **オープンソース開発の方法論**
 - ▶ **コミュニティマネジメント**
 - ▶ オープンソース開発・利用支援に関する研究
- ▶ 注意点
 - ▶ オープンソースの利用を勧めたり、オープンソース開発にまつわる思想を植え付けるための講義ではありません
 - ▶ OSSの前提・基礎知識を学んでもらい、将来、エンジニアとしてOSSテクノロジーを利用する可能性の高い諸君に心構えをしてもらうことが目的です

本日の内容

▶ OSSの開発方法論

▶ 環境

- ▶ OSS開発コミュニティのモデル
 - 伽藍とバザール (The Cathedral and the Bazaar)
- ▶ OSSコミュニティの新旧比較

▶ 人 (参加者)

- ▶ 動機
- ▶ 昇格プロセス

▶ 開発支援ツール

- ▶ バージョン管理, 不具合管理, コミュニケーション

▶ プロセス

- ▶ コード (パッチ) レビュー
- ▶ 不具合管理

オープンソースソフトウェア開発の方法論

コミュニティ主導のソフトウェア開発

- ▶ 世界中のボランティア開発者がオンライン上にコミュニティを形成し，共同でソフトウェアを開発する



オープンソースソフトウェア開発の方法論

伽藍（がらん）とバザール

- ▶ 企業とOSSコミュニティとでのソフトウェア開発方式の違いを説明するための例え
 - ▶ エリック・レイモンド著「伽藍とバザール」



伽藍方式



神？

MS:ビル・ゲイツ

Apple: スティーブ・ジョブズ



バザール方式

オープンソースソフトウェア開発の方法論

OSSコミュニティの開発形態（基本編）：その1

- ▶ **開発者の自由**な参加／脱退が可能
 - ▶ **不特定多数**の開発者が自由に参加／脱退することが可能である
 - ▶ プロジェクトの解散・消滅を招く原因となることもある
- ▶ **ボランティア**での参加が基本
 - ▶ 参加する開発者への対価として金銭的な報酬がある場合はごく稀
 - ▶ 参加動機：**個人の知的好奇心の充足**による場合が多い
- ▶ 厳格な指揮系統が存在しない
 - ▶ 開発者が自由に参加するため、**個人中心で自由な環境**で開発が行われる

オープンソースソフトウェア開発の方法論

OSSコミュニティの開発形態（基本編）：その2

- ▶ ユーザ参加型コミュニティ
 - ▶ ユーザは成果物の享受者として捉えられる
 - ▶ 開発者へのモチベーションの提供者
 - ▶ ユーザによる欠陥の発見と報告という行為の重要性から、共同開発者と位置付ける場合も多い
- ▶ ネットワークを介した分散開発環境
 - ▶ メーリングリスト（ML）や掲示板などの非対面であつ非同期な媒体が用いられる

オープンソースソフトウェア開発の方法論

OSSコミュニティの開発形態（応用編）：その1

▶ 財団主導のOSS開発

- ▶ 関連企業や組織から得た寄付金を使って専任の開発者を雇用する
- ▶ 財団の貢献
 - ▶ 安定した資金源の確保
 - ▶ 専任開発者の雇用によるOSS開発の信頼性・安定性の確保
- ▶ 代表例
 - ▶ Linux Foundation
 - ▶ Mozilla Foundation
 - ▶ Apache Software Foundation (ASF)
 - ▶ Free Software Foundation (GNU)

オープンソースソフトウェア開発の方法論

OSSコミュニティの開発形態（応用編）：その2

▶ 企業主導のOSS開発

- ▶ 商用ソフトウェアをオープンソース化しコミュニティに提供
- ▶ コミュニティの協力を得て製品の機能・品質を向上させる

▶ 企業の貢献

- ▶ 社員をオープンソース開発に専念させる
- ▶ サーバ等の開発環境を提供する

▶ 代表例

- ▶ Netscapeプロジェクト（Netscape→AOL→解散）
- ▶ Eclipseプロジェクト（IBM）
- ▶ Android（Google）

オープンソースソフトウェア開発の方法論

OSSコミュニティの開発形態（応用編）：その3

▶ 企業連合型のOSS開発

- ▶ 利害関係の一致する複数の企業が共同でOSSプロダクトを開発する
- ▶ 既存の標準的（デファクトスタンダード）なプロダクトに対抗しうる製品を開発する
 - ▶ 一社単独では開発コストあるいは開発スピードで競合他社には敵わない領域で近年顕著にみられる協業形態

▶ 企業の貢献

- ▶ 社員をプロジェクトに参加させる
- ▶ スポンサーとして出資する

▶ 代表例

- ▶ OpenStack

本日の内容

▶ OSSの開発方法論

▶ 環境

- ▶ OSS開発コミュニティのモデル
 - 伽藍とバザール (The Cathedral and the Bazaar)
- ▶ OSSコミュニティの新旧比較

▶ 人 (参加者)

- ▶ 動機
- ▶ 昇格プロセス

▶ 開発支援ツール

- ▶ バージョン管理, 不具合管理, コミュニケーション

▶ プロセス

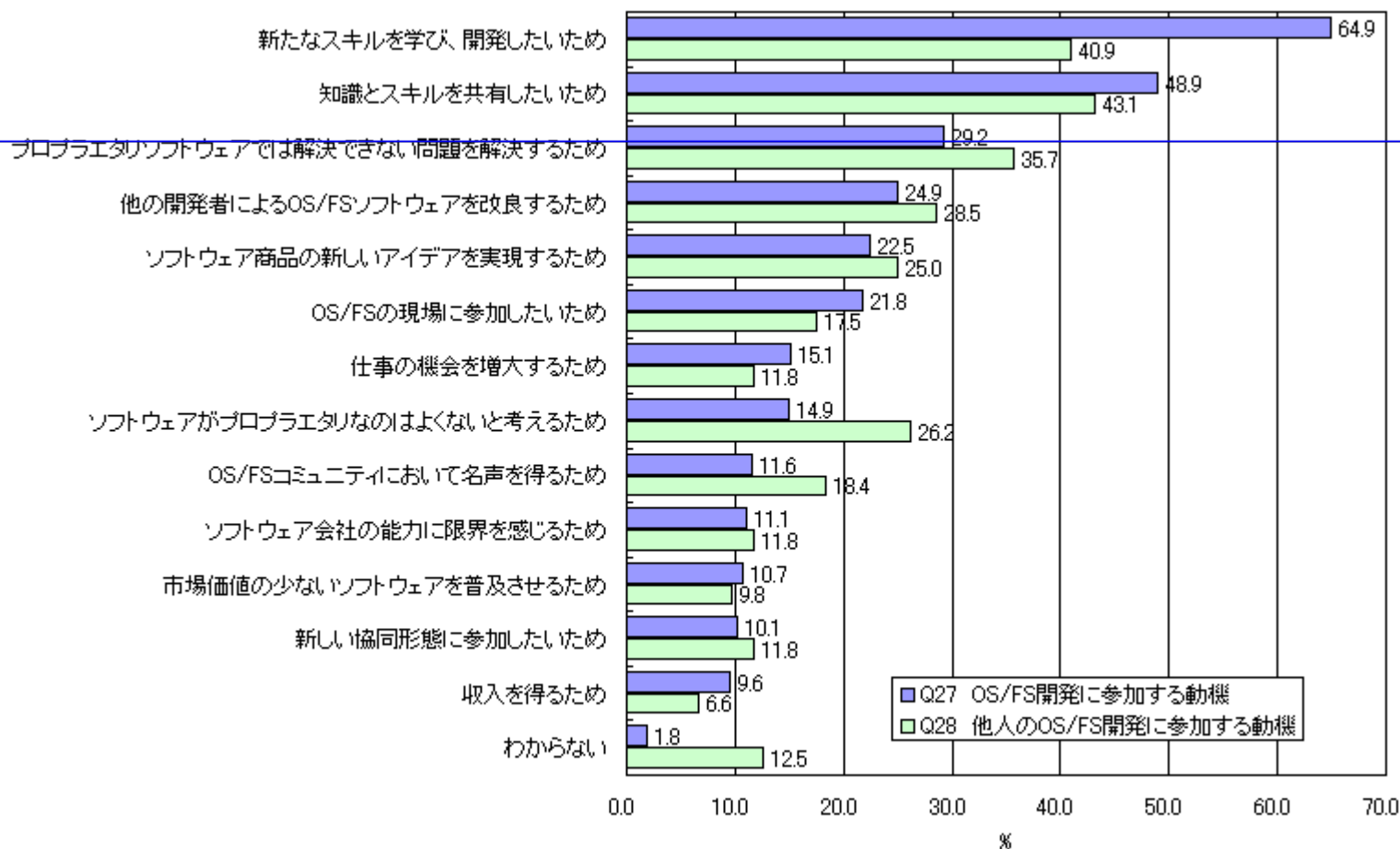
- ▶ コード (パッチ) レビュー
- ▶ 不具合管理

オープンソースソフトウェア開発の方法論

開発者のモチベーション

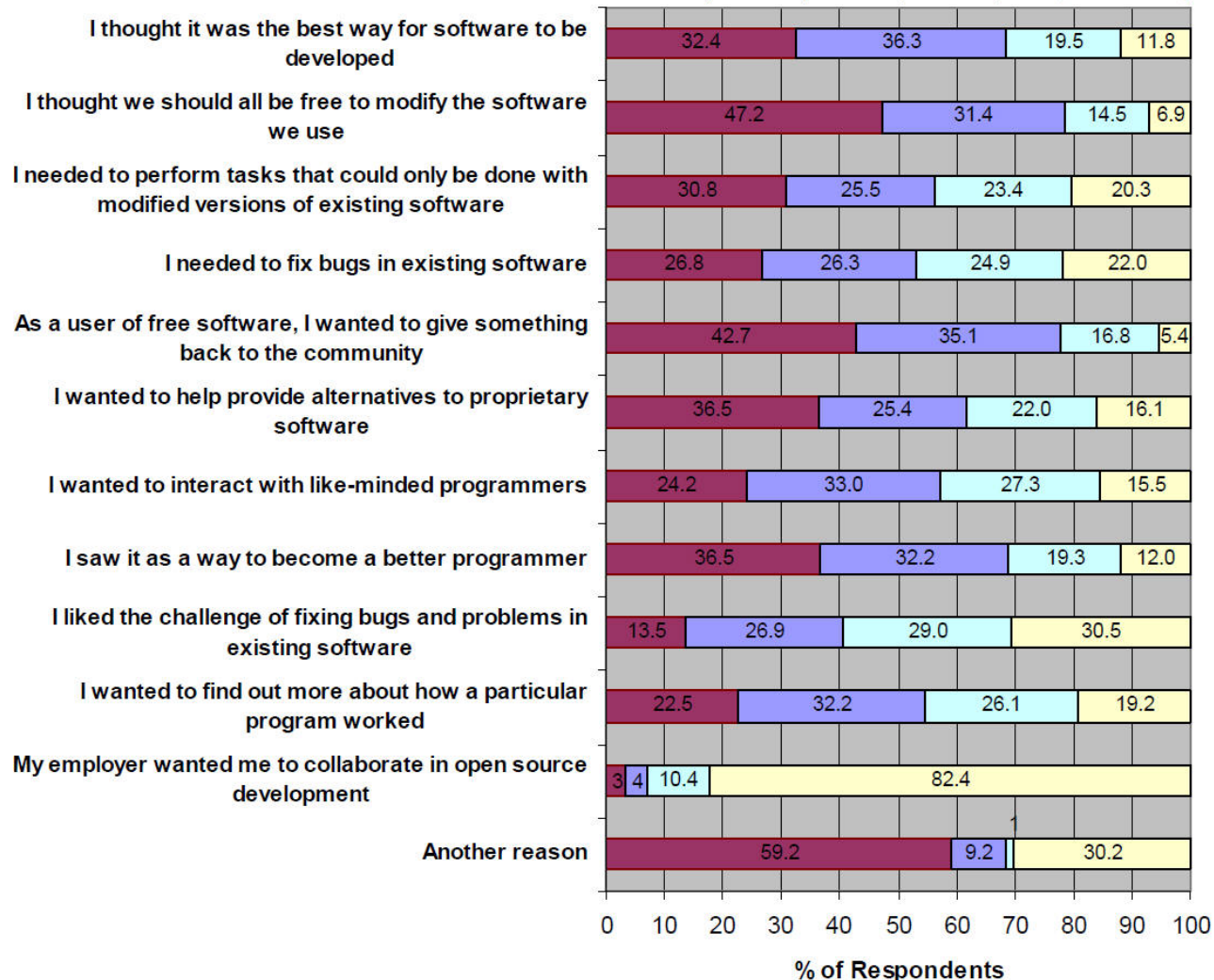
▶ なぜOSS開発に参加しているのか？

「オープンソースソフトウェア技術者の人材開発に関する調査（株式会社三菱総合研究所）」



オープンソースソフトウェア開発の方法論 開発者のモチベーション (FLOSS-US)

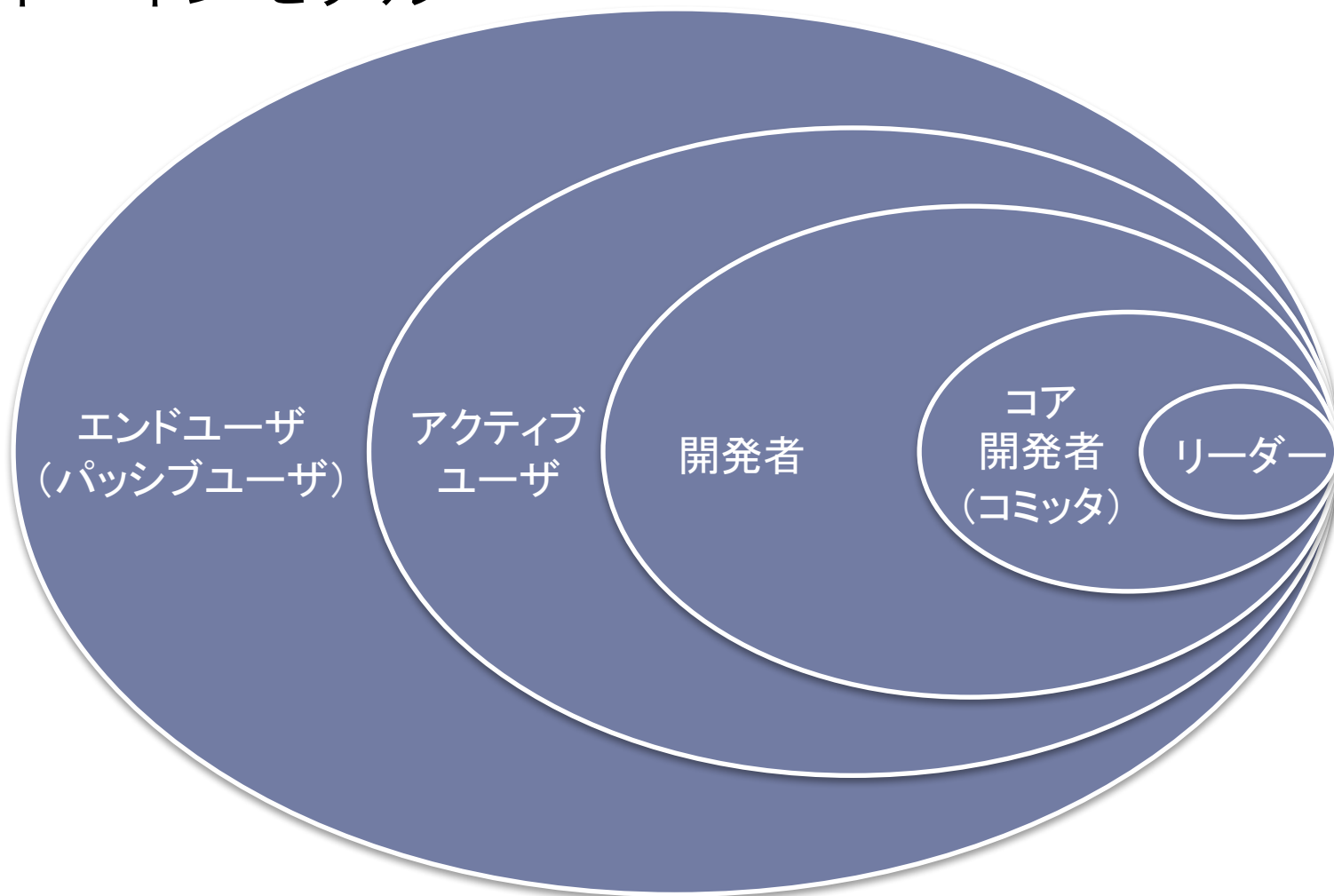
(1540 respondents; June 17, 2003; FLOSS-US)



オープンソースソフトウェア開発の方法論

OSSコミュニティの社会構造

▶ オニオンモデル



Proc. of 25th International Conference on Software Engineering
(ICSE'03), pp.419-429, 2003.

Toward an Understanding of the Motivation of Open Source Software Developers

Yunwen Ye and Kouichi Kishida

Structure of an OSS community

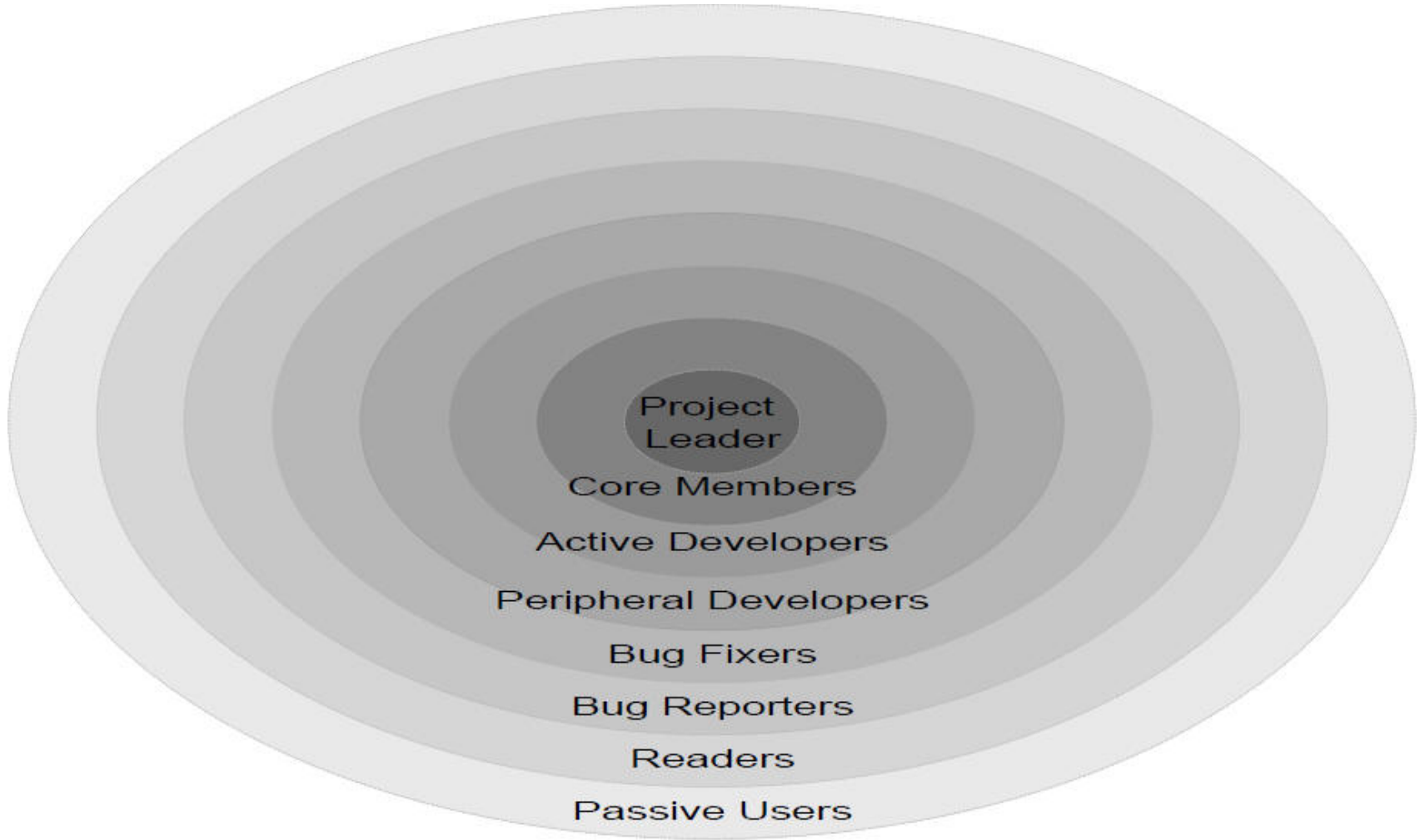


Figure 1: General structure of an OSS community

Learning through the co-evolving process of OSS systems and communities

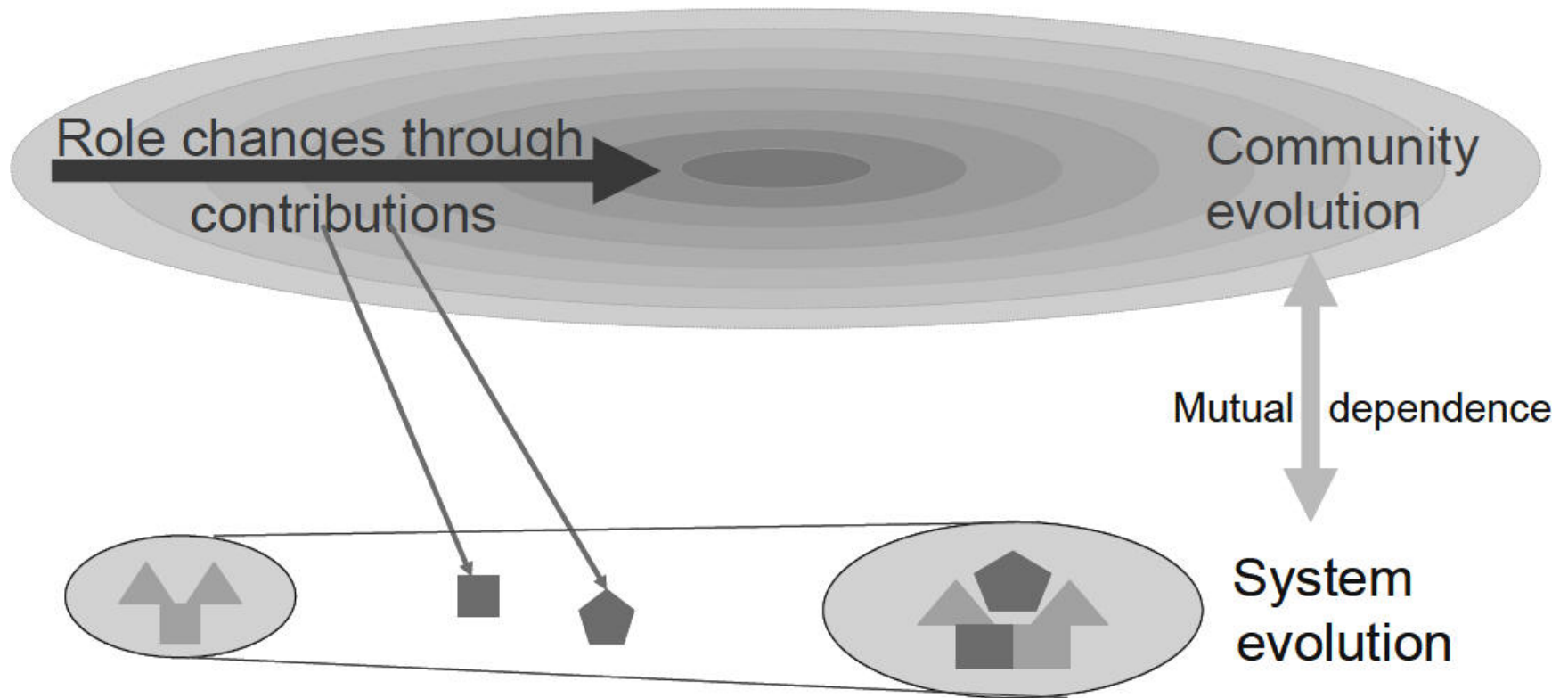


Figure 2: The co-evolution of OSS systems and OSS communities

Learning as the Motivation (1)

- ▶ OSSプロジェクト創始者のモチベーション（プロジェクト開始理由）
 - ▶ Explorative learning
 - ▶ 新しいやり方や問題解決の方法を見つける
 - ▶ 探索的（手探りで試行錯誤する中で）様々な発見が得られる
 - 成功するための要件があらかじめ明確ではないという点では、大学の研究プロジェクトに近い感覚
 - ▶ Learning by doing
 - ▶ 自らの知識や技術を具体的に試しながら、特定の領域に対する理解を深める
 - 不足している知識や技術が明確になるので、習得すべきスキルが明らかになる（失敗することにも価値がある）

Learning as the Motivation (2)

- ▶ プロジェクト開始後に参加する人のモチベーション
 - ▶ Learning by doing (peripheral participation : 周辺的参加)
 - エキスパートプログラマのソースコードを読んで学習する
 - 不具合を報告したり修正したりする
 - ▶ Recognized contributions
 - コミュニティへの貢献（不具合修正など）が認められるにつれて、より大きく複雑なタスクが与えられるようになる
 - その結果、コミュニティの中でより中心的な役割を与えられると同時に、コミュニティに対する責任感が生じる

Learning as the Motivation (3)

- ▶ 社会的側面：Social Aspects of Extrinsic Motivations (外発的動機)
 - ▶ コミュニティ内で評判が高くなると、他の開発者からより信頼されるため共同でタスクを遂行する機会が増える
 - ▶ 最終的に、プロジェクト創始者と同等の立場でOSS開発コミュニティの現状と課題について考えるようになる
 - ▶ この時点の参加者は、周辺的に参加している人々よりもかなり中心的な存在になっている
 - ▶ 開発の方向性に影響を与え得るコアメンバになるためには、コミュニティに対してさらなる貢献をし続けなくてはならない

Proc. of 29th International Conference on Software Engineering
(ICSE'07), pp. 364-374, 2007.

Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study

Chris Jensen and Walt Scacchi

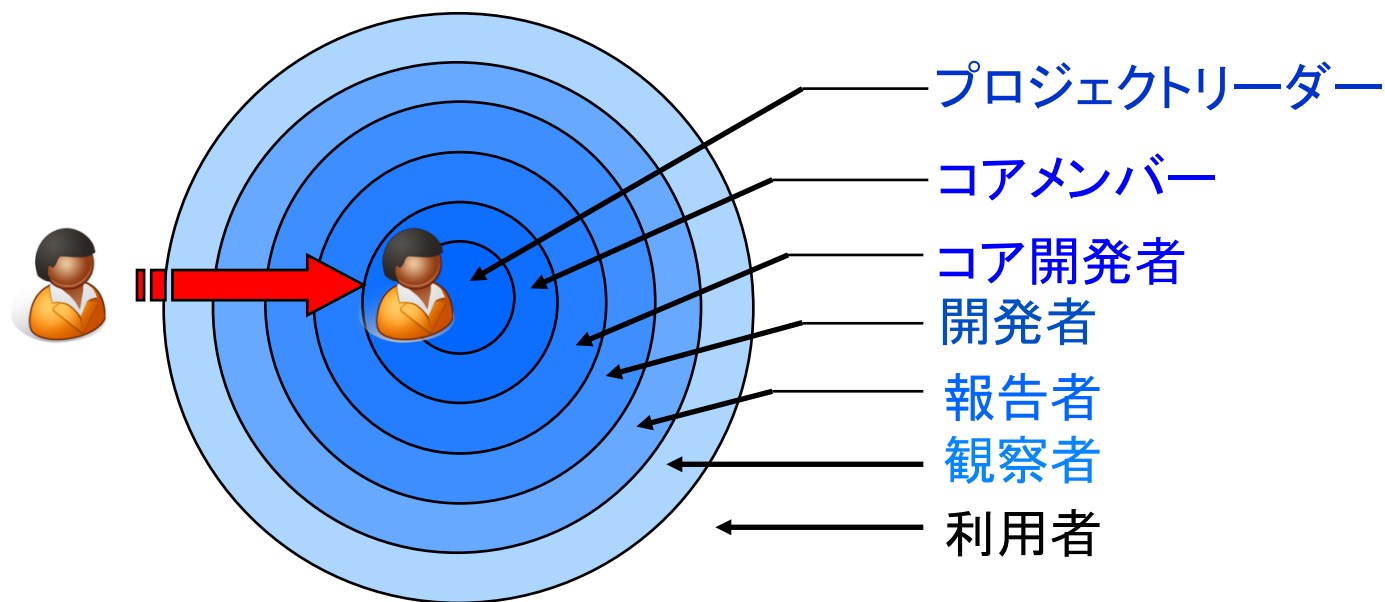
Introduction & Background

- ▶ Mozilla, Apache, NetBeansプロジェクトを対象に，OSSプロジェクトの組織構造や技術的役割，役割移動の実態に関する調査を行った.
- ▶ 参加者へのインタビューを実施

Introduction & Background

- “Onion” organizational hierarchy

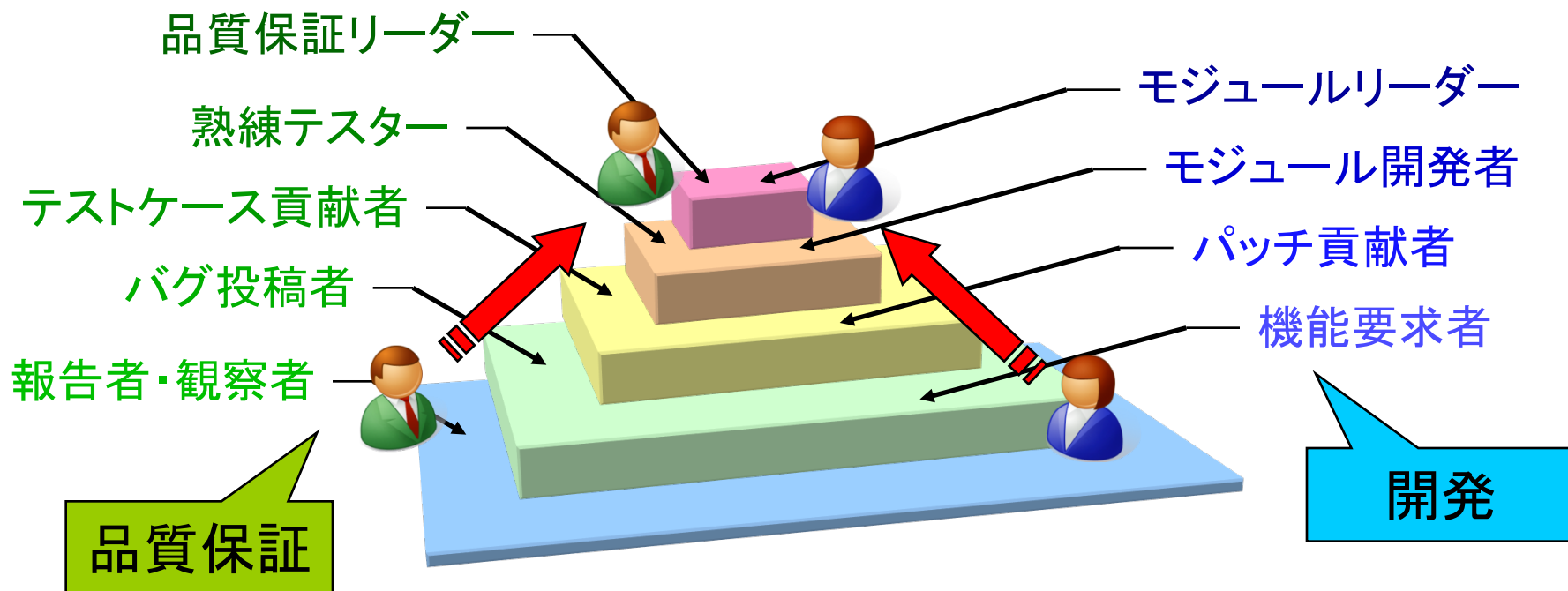
- OSS開発における開発者の組織構造は、一般的に
オニオン型組織と捉えられている
- 参加者の役割移動は一方方向しかない



Introduction & Background

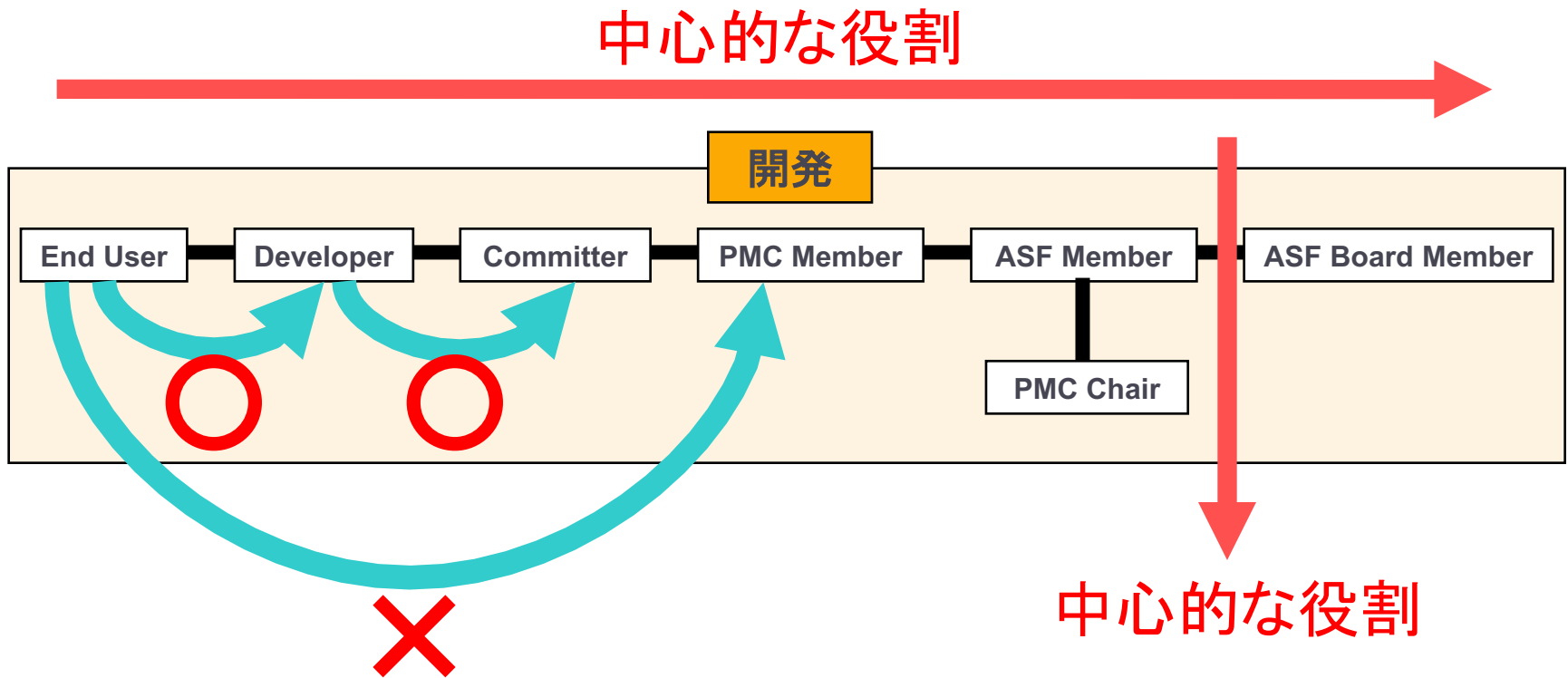
- “Onion Pyramid” organizational hierarchy

- 大きなOSS開発では**ピラミッド型組織**で構成されている
 - 1つの層に**複数の役割**が存在する
 - 役割の種類ごとに**グループ化**している
- 役割によって移動する方向が違う



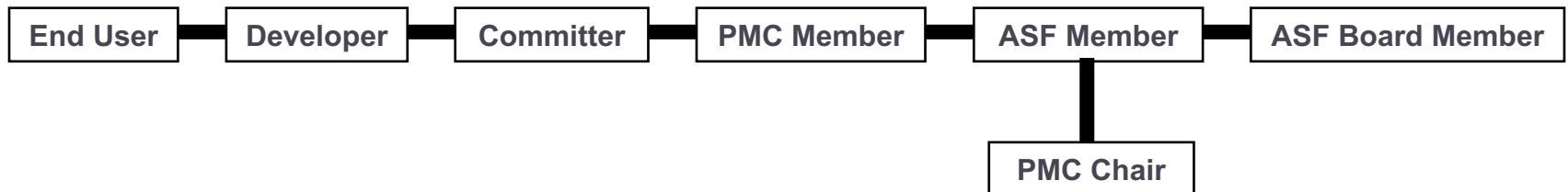
Case 1: Apache.org

- ▶ 1個の役割グループから構成されている



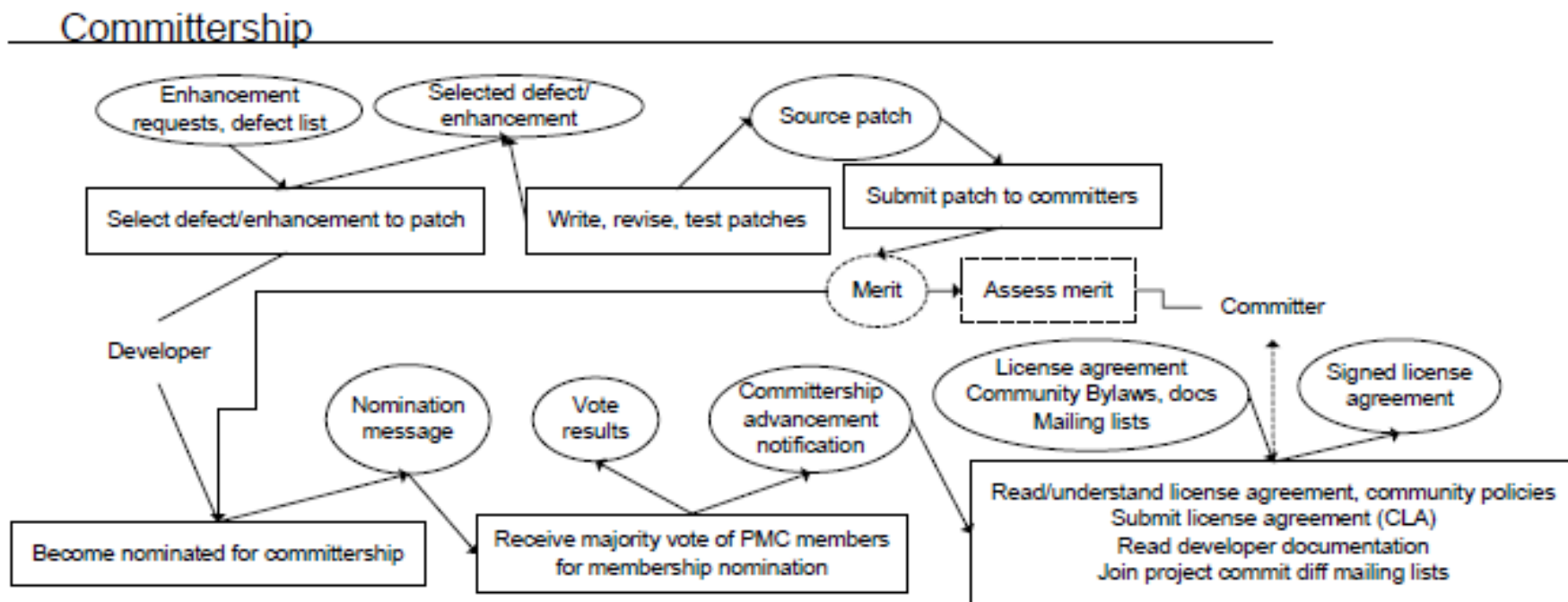
Case 1: Apache.org

- ▶ 参加者は開発に対して技術的な貢献（バグ報告やコード修正など）をすることで、中心の役割へと移動する
- ▶ Committer以上の役割は、承認あるいは選出制度が用いられている
 - ▶ PMC Member が Developer の貢献を認め、その Developer にコミット権限を与え、Developer は Committer となる
 - ▶ ASF Member が Committer の中から新しい PMC Member を選出する



Case 1: Apache.org

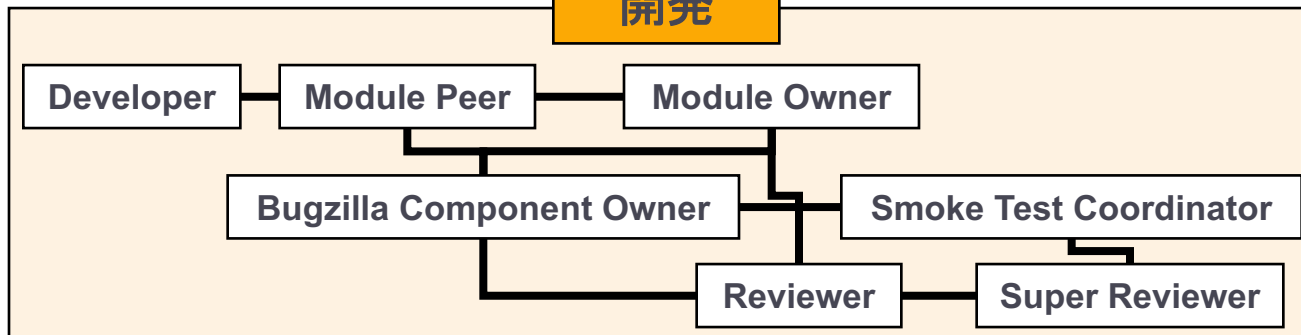
▶ Committerになるまでの流れ



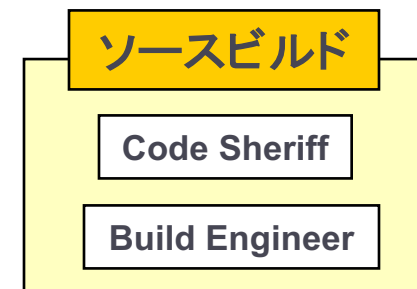
Case 2: Mozilla.org

▶ 4個の役割グループから構成されている

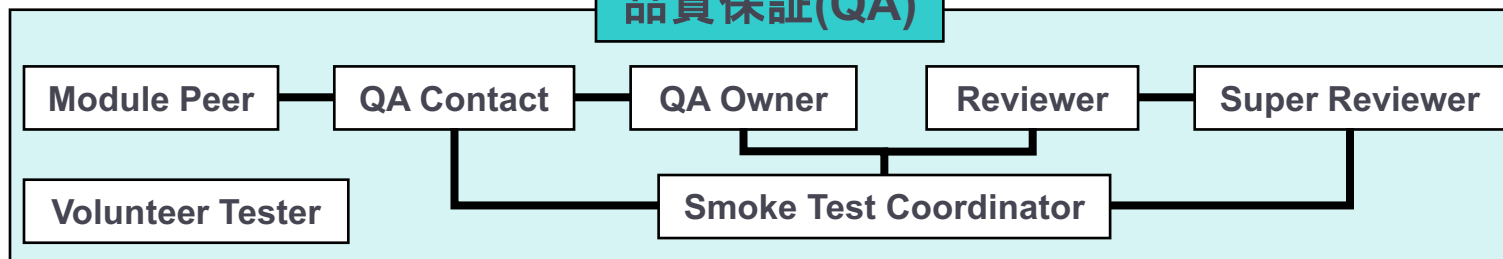
開発



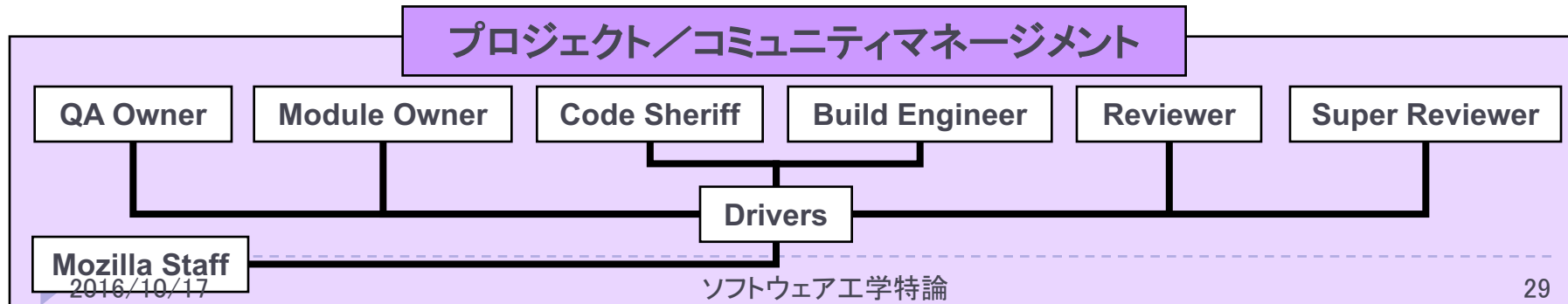
ソースビルド



品質保証(QA)

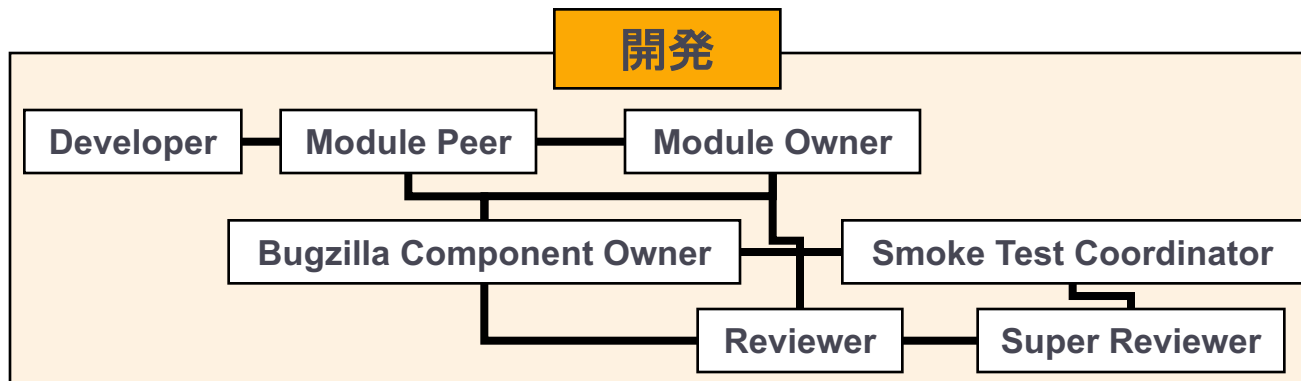


プロジェクト／コミュニティマネージメント



Case 2: Mozilla.org

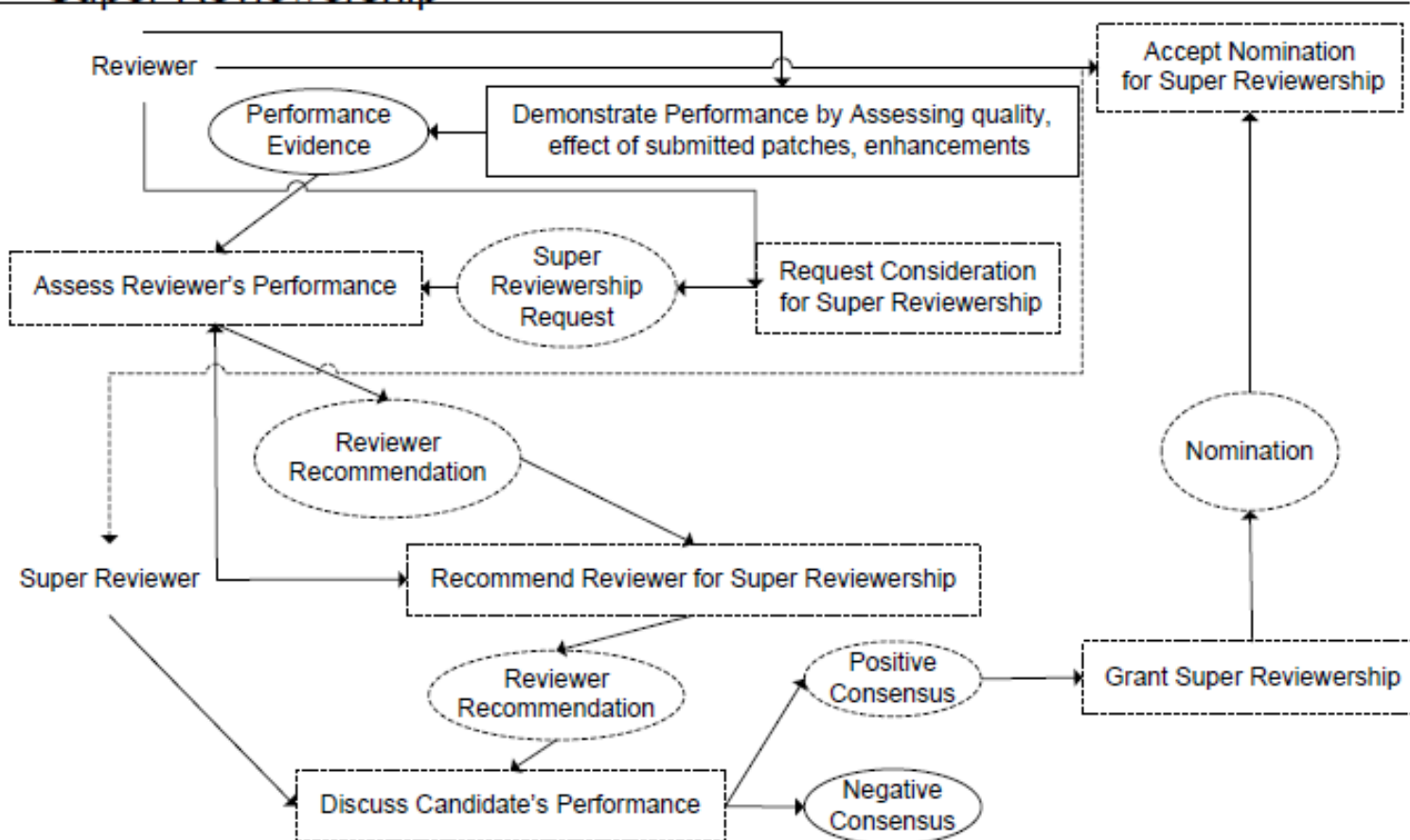
- ▶ 参加者は開発に対して技術的な貢献をしたり、社会的な貢献(開発の指示etc.)をすることで、中心の役割へと移動する
- ▶ 開発グループ内で優れた技術を示すことで、Super Reviewerになることができる



Case 2: Mozilla.org

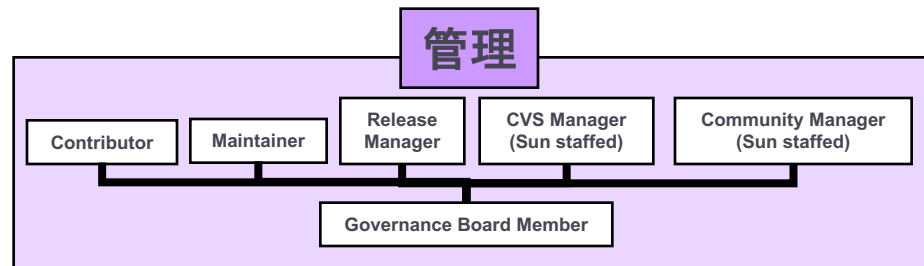
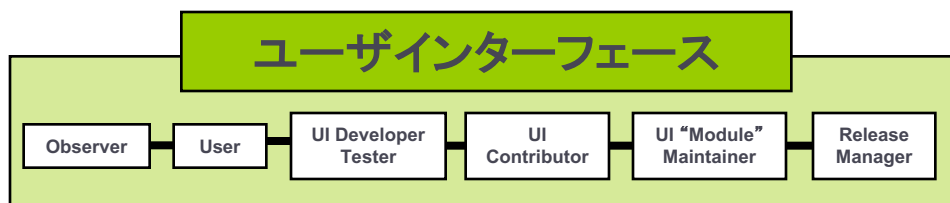
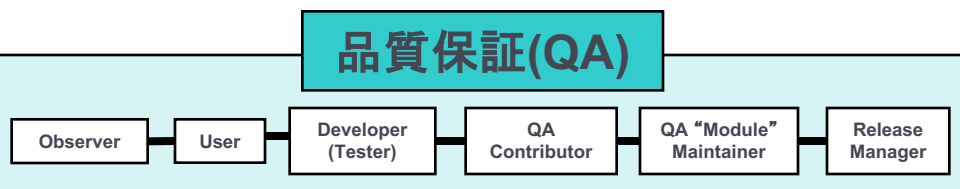
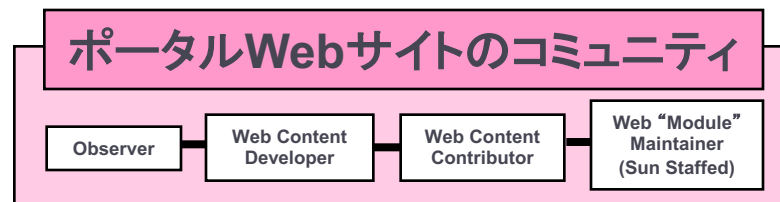
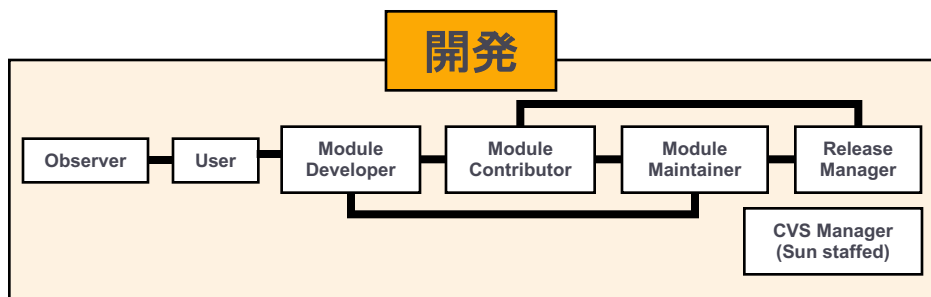
▶ Super Reviewerになるまでの流れ

Super Reviewership



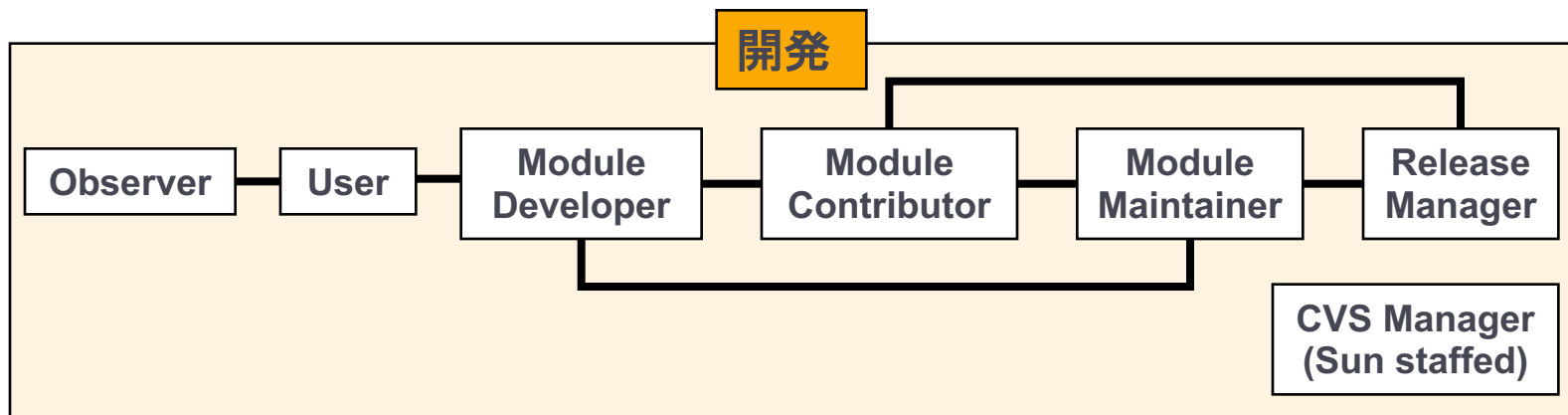
Case 3: NetBeans.org

▶ 6個の役割グループから構成されている



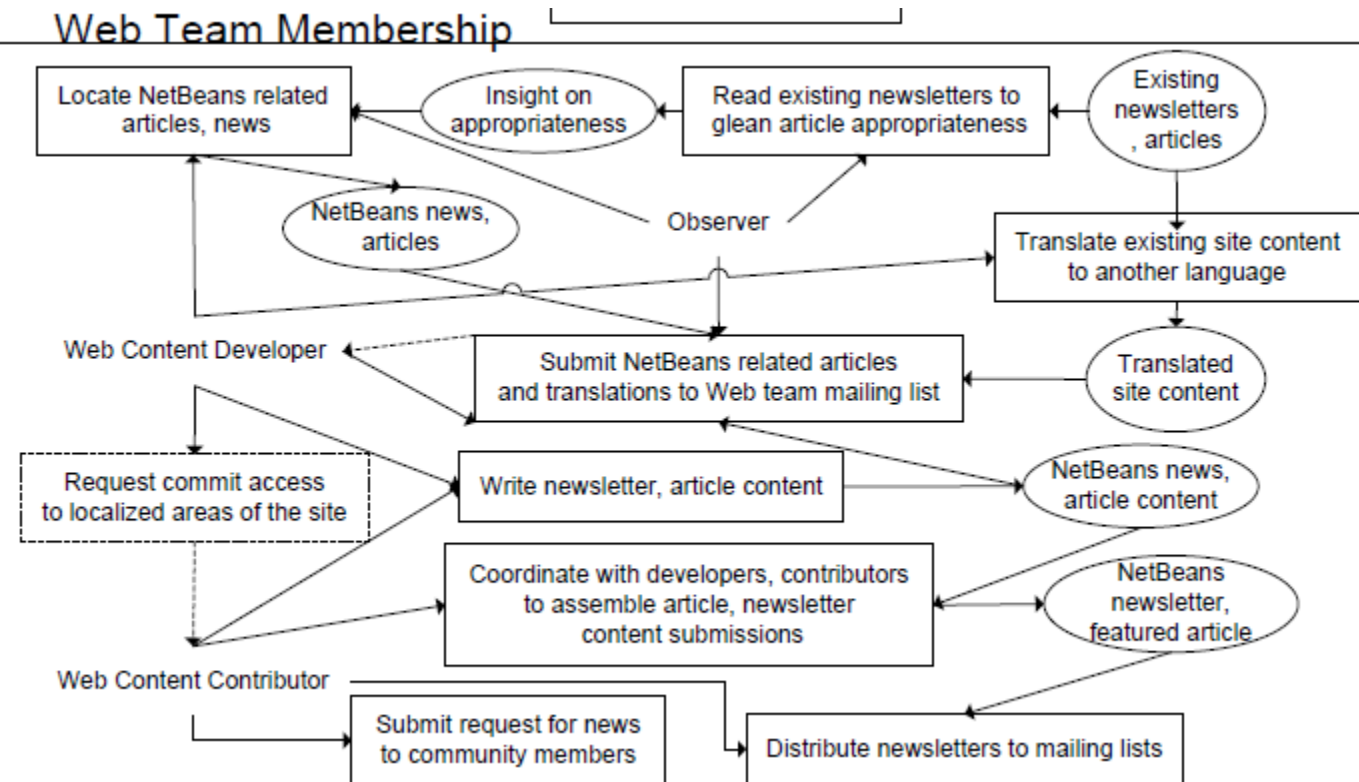
Case 3: NetBeans.org

- ▶ ほとんどの中心メンバーはSun Microsystems社（Sun）の社員で構成されている
- ▶ Sunの社員にのみ移動が許されている役割が存在する
 - ▶ CVS Managerなど
- ▶ 中心的な役割にはSunの社員の任命や指名が必要である
 - ▶ Module DeveloperがコミッタになるためにはCVS Managerへの申請が必要である



Case 3: NetBeans.org

▶ Web Teamの役割移動の流れ



Conclusions

- ▶ 一般的なOSS開発のオニオン型やピラミッド型だけでは表現しきれない複雑な組織構造を確認した
- ▶ 周囲の役割から中心的な立場へと変わる役割移動の実態を明らかにした

役割獲得方法	種類
ボランティア	ボランティアとして参加（貢献）する 例：Mozillaのボランティアテスター
許可	コアメンバから特定の役割を与えられる 例：Apacheのコミッター
指名	コアメンバに特定の役割を指名される 例：NetBeansのコミュニティマネージャー
選出	コミュニティの委員会で投票される 例：Apache Software Foundationの理事メンバー

本日の内容

▶ OSSの開発方法論

▶ 環境

- ▶ OSS開発コミュニティのモデル
 - 伽藍とバザール (The Cathedral and the Bazaar)
- ▶ OSSコミュニティの新旧比較

▶ 人 (参加者)

- ▶ 動機
- ▶ 昇格プロセス

▶ 開発支援ツール

- ▶ バージョン管理, 不具合管理, コミュニケーション

▶ プロセス

- ▶ コード (パッチ) レビュー
- ▶ 不具合管理

オープンソースソフトウェア開発の方法論

三種の神器

- ▶ オープンソース開発のためのツール
 - ▶ コミュニケーション
 - ▶ メーリングリスト
 - ▶ 掲示板
 - ▶ IRC
 - ▶ バージョン管理（構成管理）
 - ▶ CVS
 - ▶ Subversion
 - ▶ Git
 - ▶ 不具合（バグ）管理
 - ▶ Bugzilla

本日の内容

▶ OSSの開発方法論

▶ 環境

- ▶ OSS開発コミュニティのモデル
 - 伽藍とバザール (The Cathedral and the Bazaar)
- ▶ OSSコミュニティの新旧比較

▶ 人 (参加者)

- ▶ 動機
- ▶ 昇格プロセス

▶ 開発支援ツール

- ▶ バージョン管理, 不具合管理, コミュニケーション

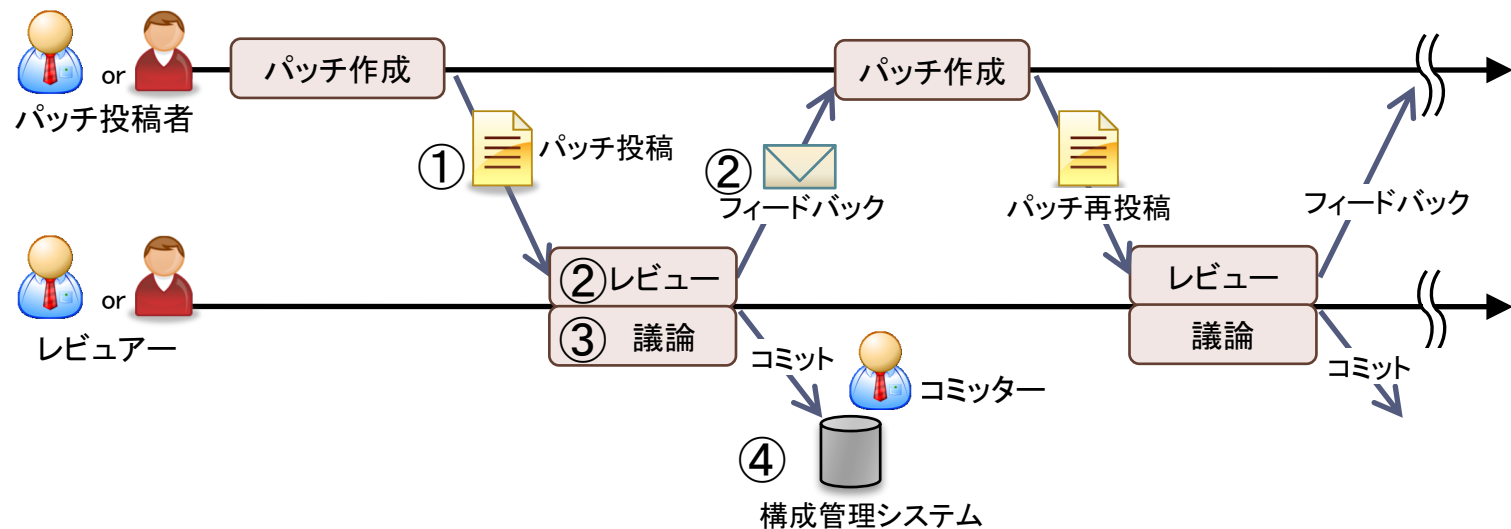
▶ プロセス

- ▶ コード (パッチ) レビュー
- ▶ 不具合管理

オープンソースソフトウェア開発の方法論

OSS開発の流れ：パッチレビュー

- ▶ ソフトウェアに対する変更はパッチの適用により行われる
 - ▶ パッチ (patch)：機能追加・不具合修正のためのコード差分
 - ▶ 複数の開発者によってレビュー（検査）され妥当性が検証されたパッチのみが適用される



①パッチの作成・投稿

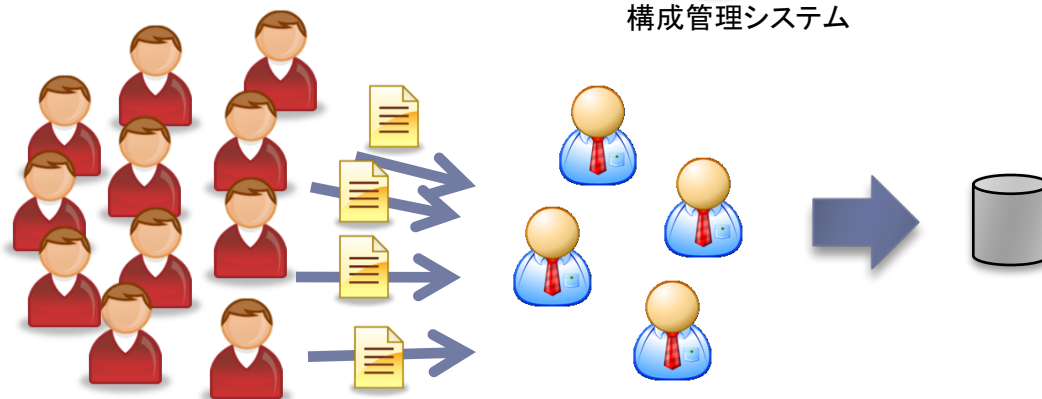
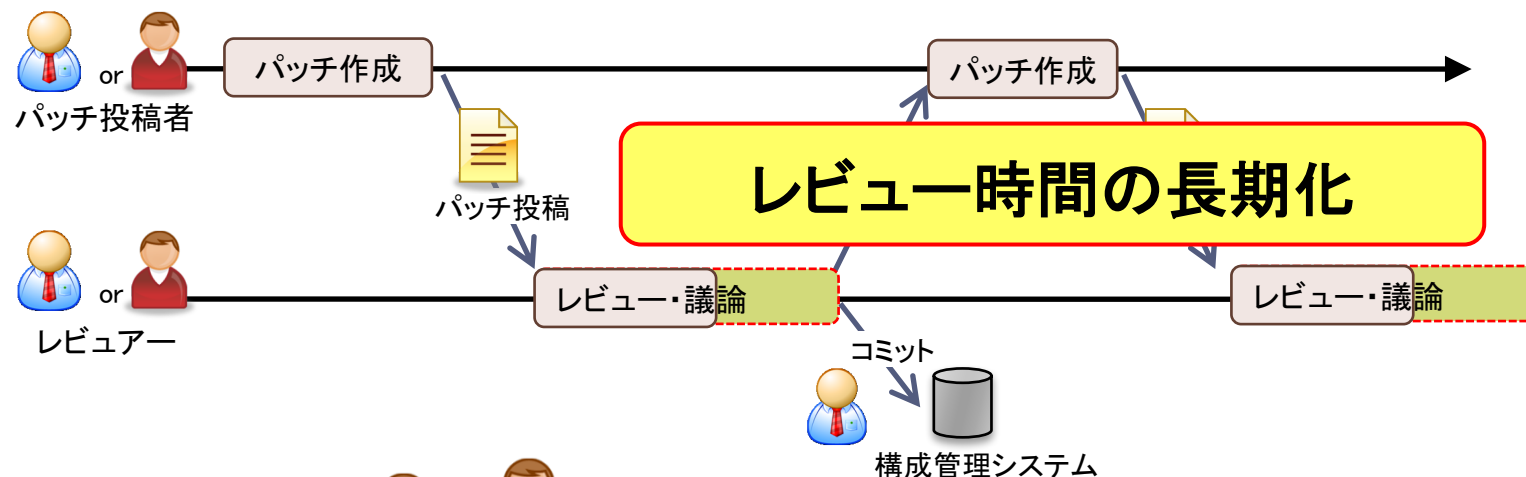
②投稿されたパッチのレビュー&フィードバック

③不具合に対する議論(コメントの投稿)

④パッチの妥当性確認後、構成管理システムへコミット

オープンソースソフトウェア開発の方法論

OSS開発の流れ：パッチレビューの問題

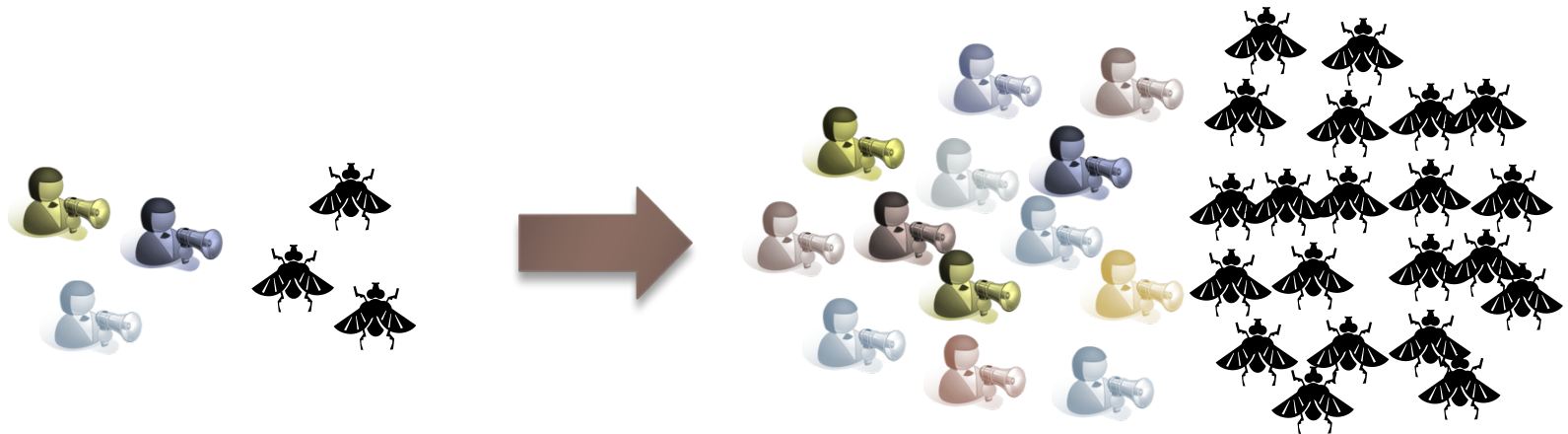


少数のコミッターですべての作業を行うのが困難

オープンソースソフトウェア開発の方法論

不具合管理プロセス (1)

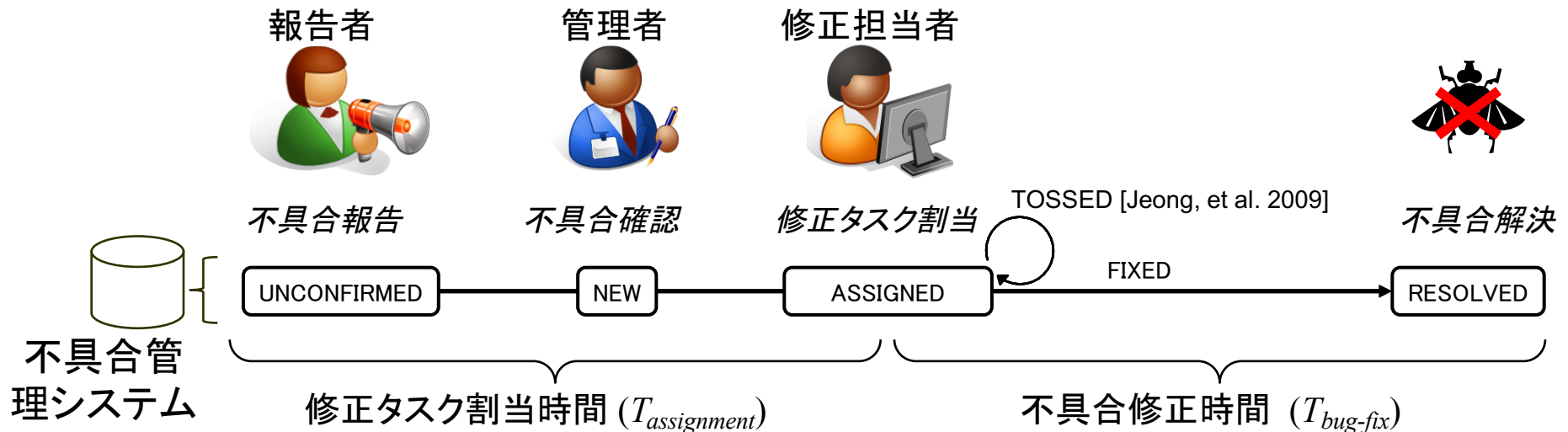
- ▶ 近年の大規模OSS開発では、効率的な不具合管理プロセスが必須である
 - ▶ 不具合管理プロセス：不具合の報告から不具合修正タスクの割当ておよび修正にいたるまでの一連のプロセス
 - ▶ 理由：ユーザ数の増加に伴い、OSSプロジェクトに報告される不具合の数が大幅に増加したため



オープンソースソフトウェア開発の方法論

不具合管理プロセス (2)

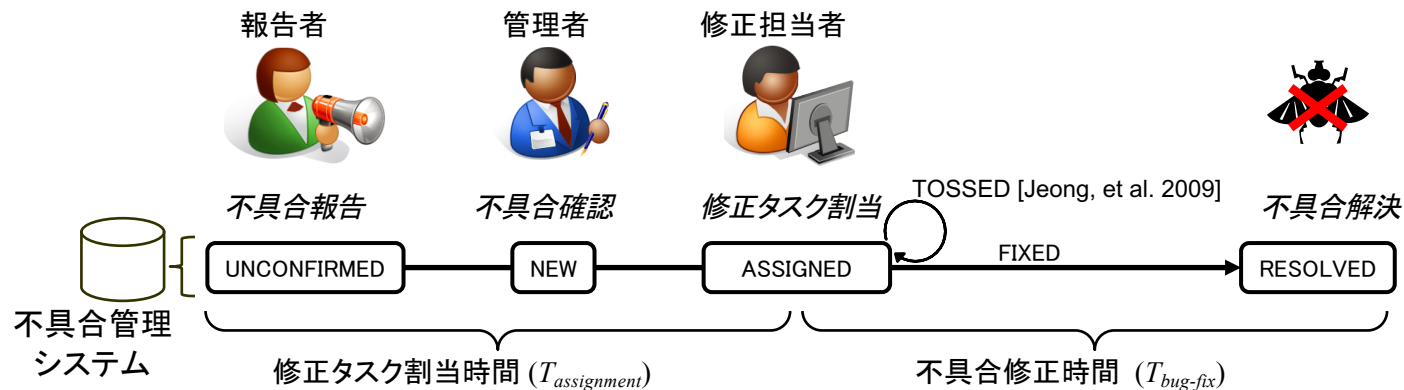
▶ 一般的な不具合管理プロセス



- ▶ 大規模OSSプロジェクト（MozillaやEclipseなど）では、一日に100件以上の不具合が報告されることも珍しくない
- ▶ 管理者のタスク
 - ▶ 大量の不具合報告を理解する
 - ▶ すでに報告された不具合でないかどうかを確認する
 - ▶ 不具合修正に適任の開発者に依頼する

オープンソースソフトウェア開発の方法論

不具合管理プロセスの問題点

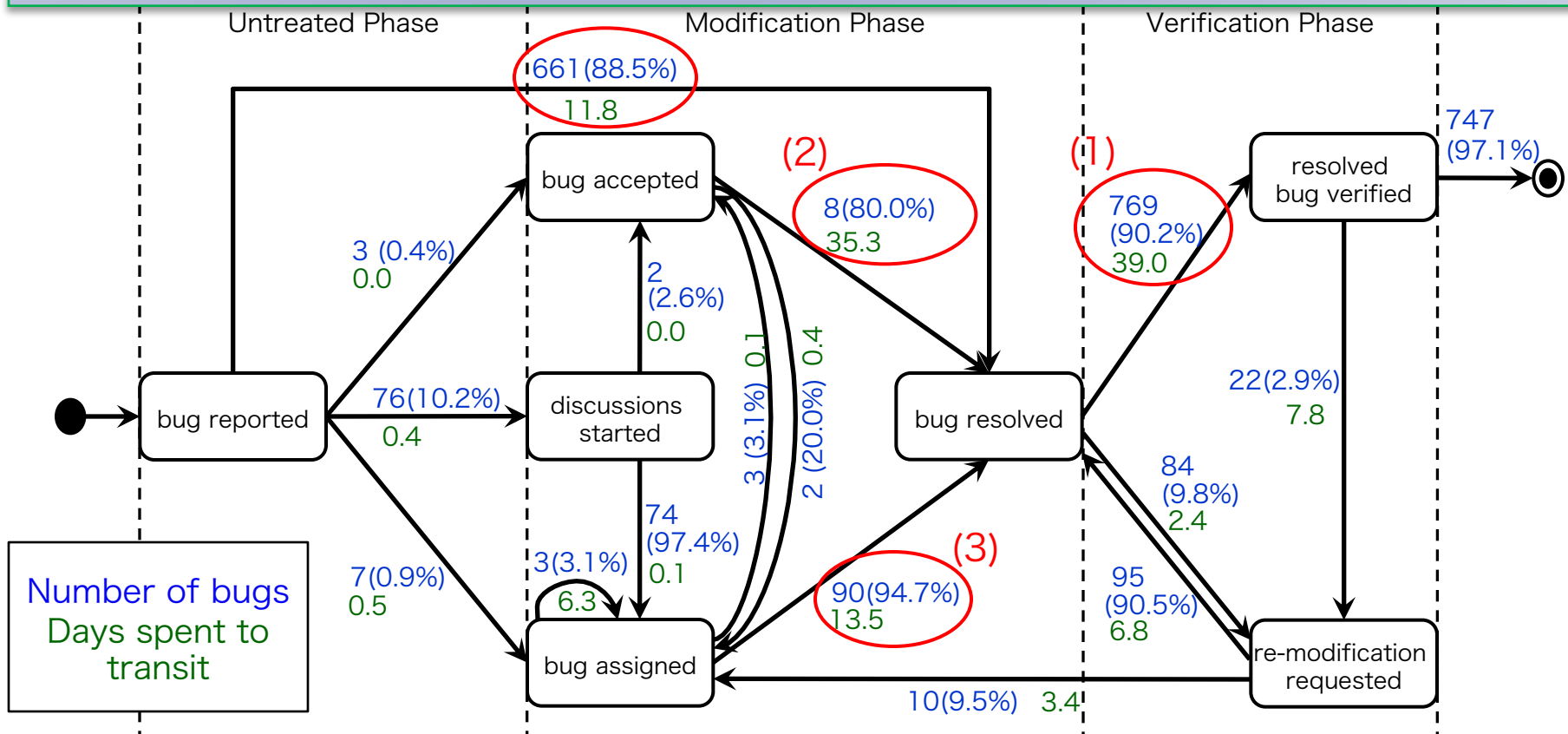


- ▶ エンドユーザは開発者が修正に必要な情報を報告できない（どんな情報が必要か分からない）
 - ▶ 不具合が確認できず修正できない
 - ▶ 不具合を修正する担当者がなかなか決まらない
- ▶ 個々の不具合に対して適任の担当者を選ぶのが難しい
 - ▶ 修正担当者が何度も入れ替わる必要が出てくる
 - ▶ 修正が完了した（と思っていた）不具合が後日再度報告される

不具合管理プロセスの実態

Apache

Many bugs were resolved without going through “bug accepted” or “bug assigned”

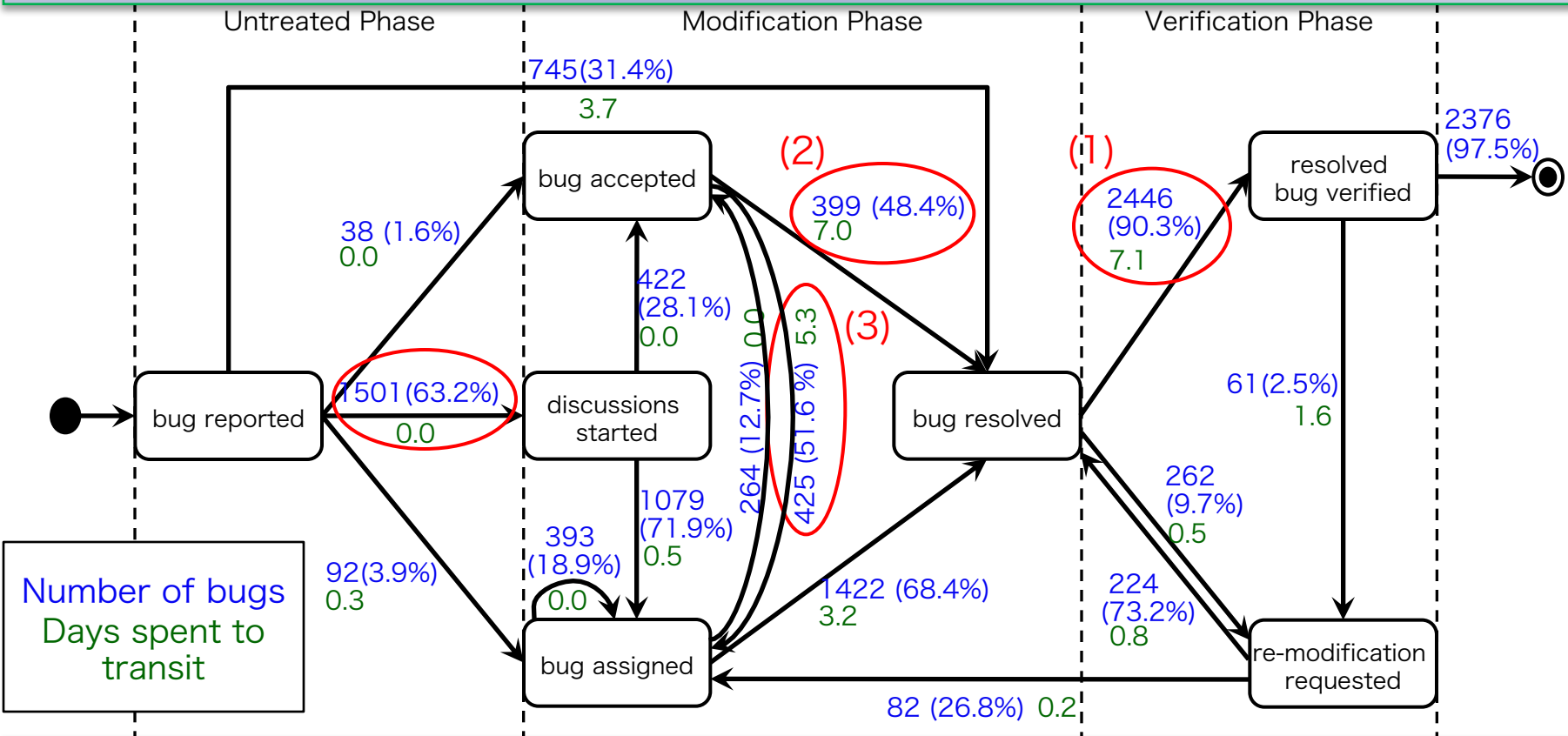


(1) “bug resolved” to “resolved bug verified”, (2) “bug accepted” to “bug resolved” and (3) “bug assigned” to “bug resolved” were bottlenecks

不具合管理プロセスの実態

Firefox

Many bugs were resolved after “bug accepted” or “bug assigned”



(1) “bug resolved” to “resolved bug verified”, (2) “bug accepted” to “bug resolved” and (3) “bug accepted” to “bug assigned” were bottlenecks

本講義のねらい

- ▶ 情報通信システム開発におけるOSSの重要性を理解し、OSS開発に関する基礎知識を習得すること
 - ▶ オープンソース開発の歴史
 - ▶ オープンソース開発の方法論
 - ▶ コミュニティマネジメント
 - ▶ オープンソース開発・利用支援に関する研究
- ▶ 注意点
 - ▶ オープンソースの利用を勧めたり、オープンソース開発にまつわる思想を植え付けるための講義ではありません
 - ▶ OSSの前提・基礎知識を学んでもらい、将来、エンジニアとしてOSSテクノロジーを利用する可能性の高い諸君に心構えをしてもらうことが目的です

本日の内容

- ▶ コミュニティマネジメント
 - ▶ コミュニケーションとコーディネーション
 - ▶ 参加者間のコミュニケーション
 - ▶ コーディネーション：地理的分散と時間的分散
 - ▶ コラボレーション
 - ▶ べき乗の法則 (Power law relationships)
 - コミュニティの分布
 - コミュニケーションの分布
 - コントリビューション（貢献）の分布
 - ▶ コントリビューション（貢献）
 - 参加者のライフタイム
 - 期待されている貢献の種類
 - 参加者の社会的関係の影響

コミュニケーション: OSSコミュニティのコミュニケーション構造 (1)

- ▶ オニオン構造を持つOSSコミュニティにおいて、
周縁的参加者（エンドユーザ）はコア開発者から離れた存在
- ▶ コア開発者はユーザのニーズをどうやって知るのか？
- ▶ ユーザはコア開発者に要望を伝えれば良いのか？

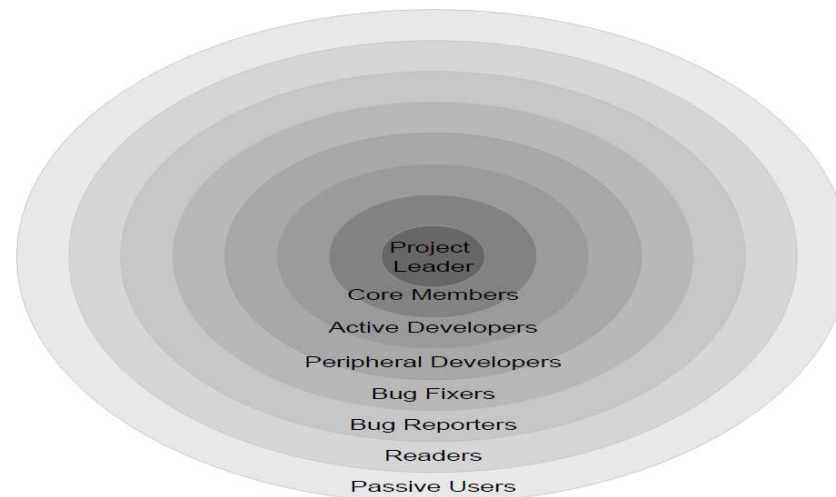


Figure 1: General structure of an OSS community

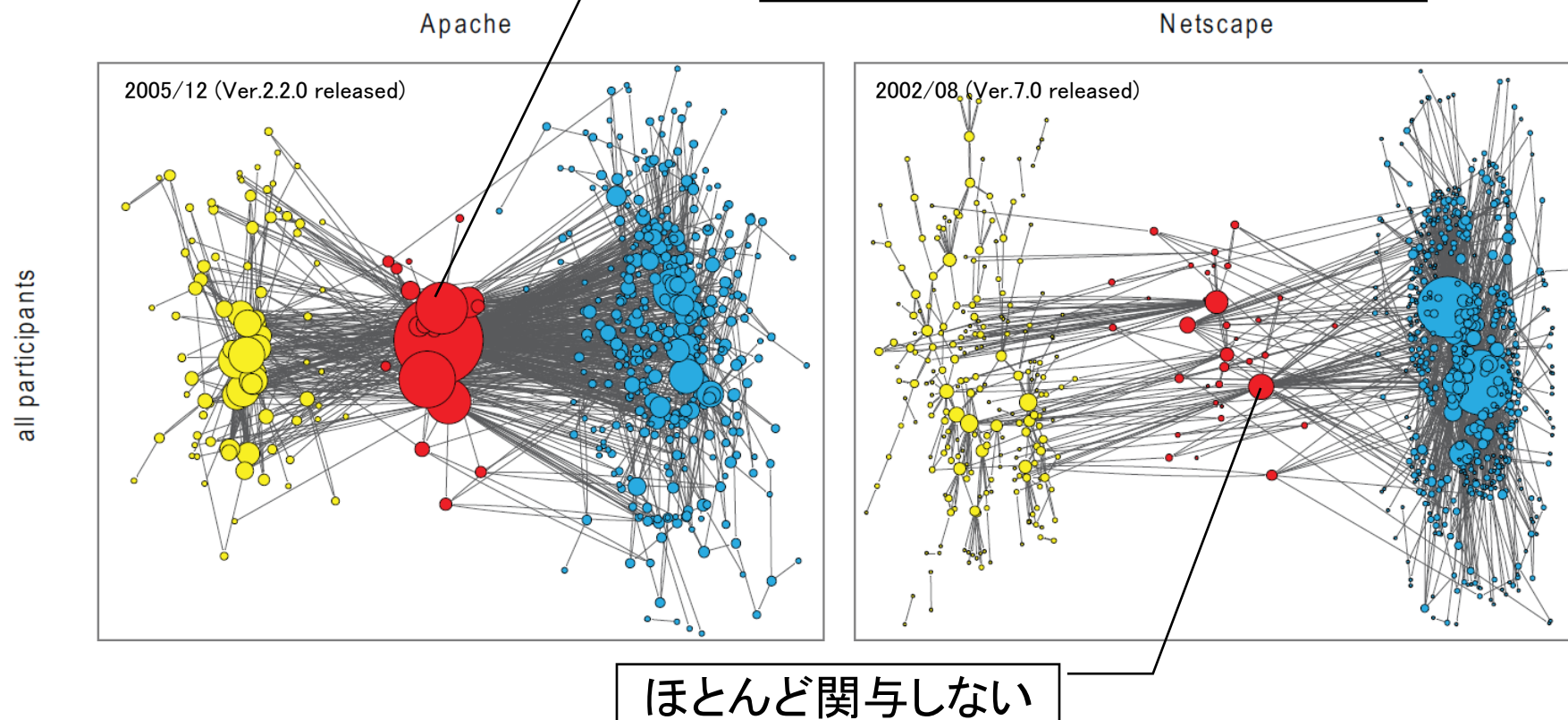
Y. Ye and K. Kishida, Toward an Understanding of the Motivation of Open Source Software Developers, Proc. of 25th International Conference on Software Engineering (ICSE'03), pp.419-429, 2003.

コミュニケーション: OSSコミュニティのコミュニケーション構造 (1)

▶ ApacheとNetscapeのコミュニケーション構造

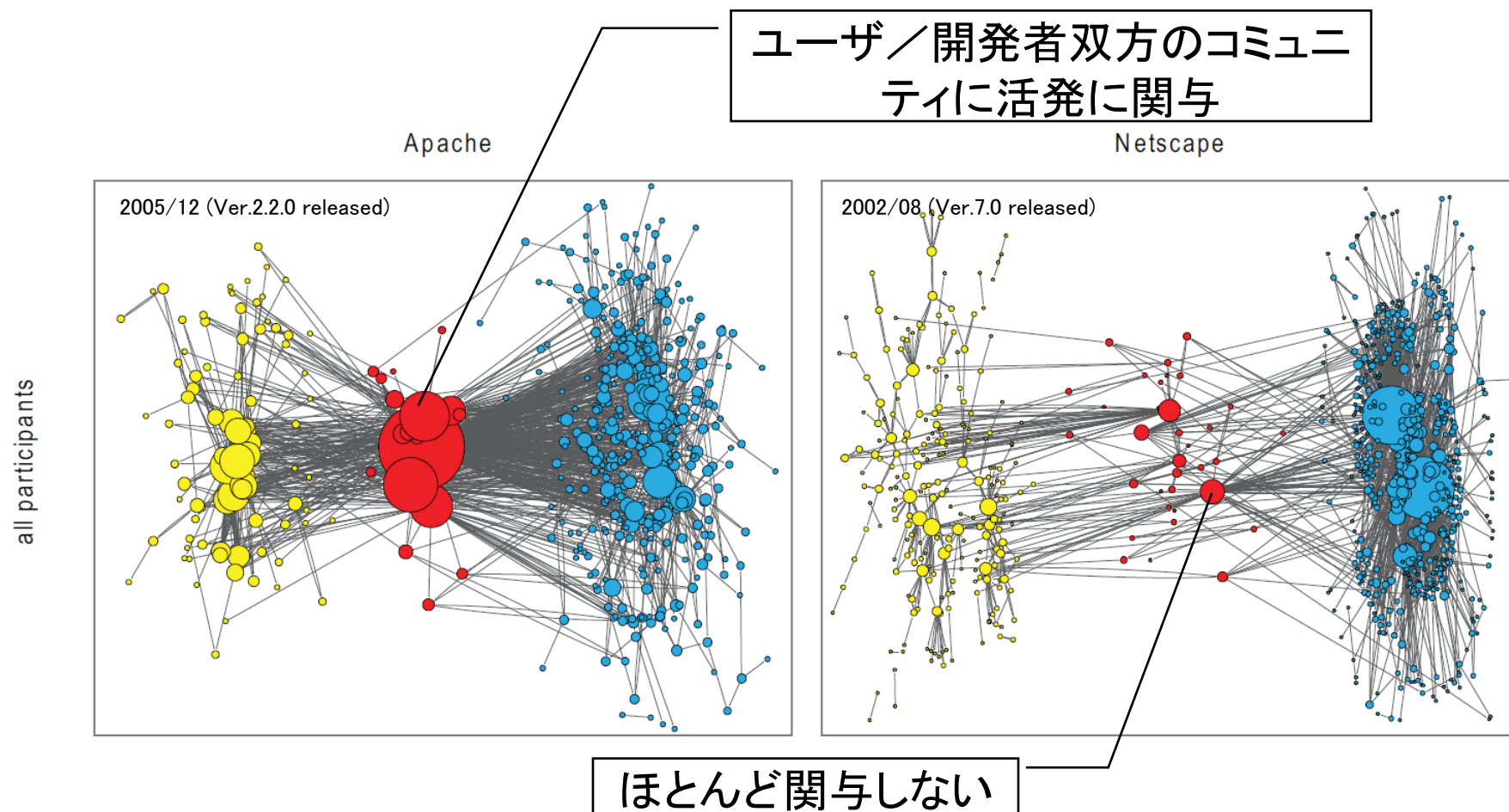
▶ コア開発者（赤色のノード）：2つのMLに参加

ユーザ／開発者双方のコミュニティに活発に関与



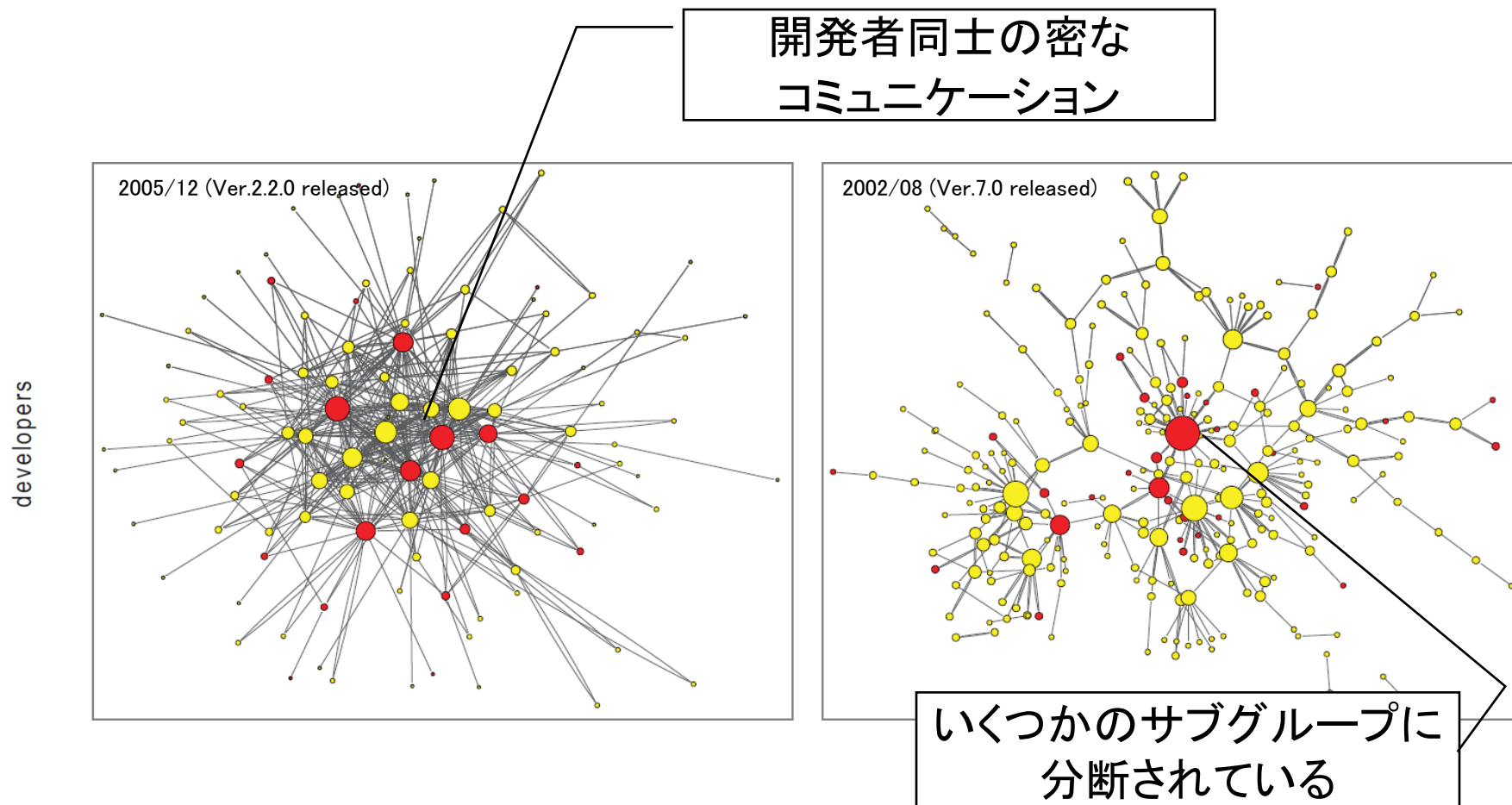
コミュニケーション: OSSコミュニティのコミュニケーション構造 (2)

▶ コミュニティ内で何が起きているのか？



コミュニケーション: OSSコミュニティのコミュニケーション構造 (3)

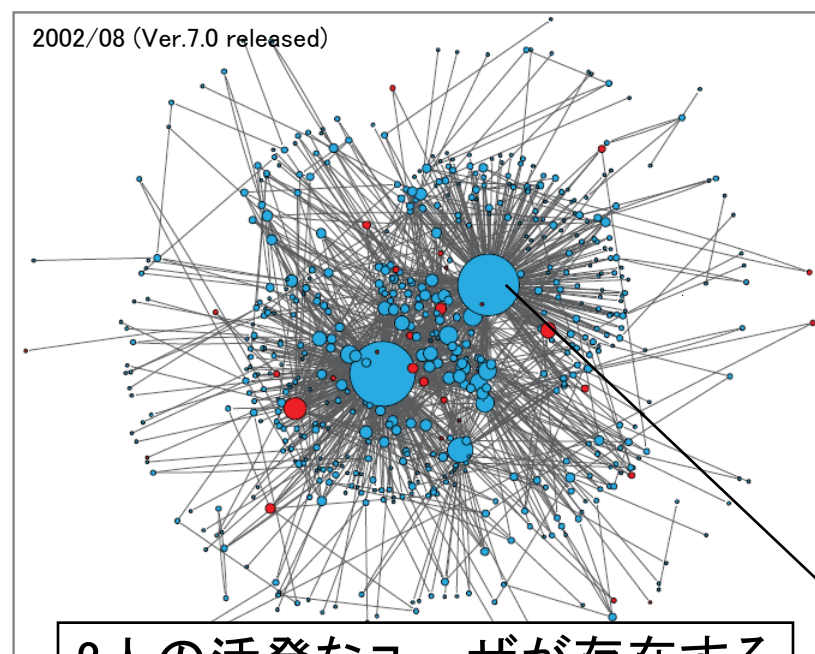
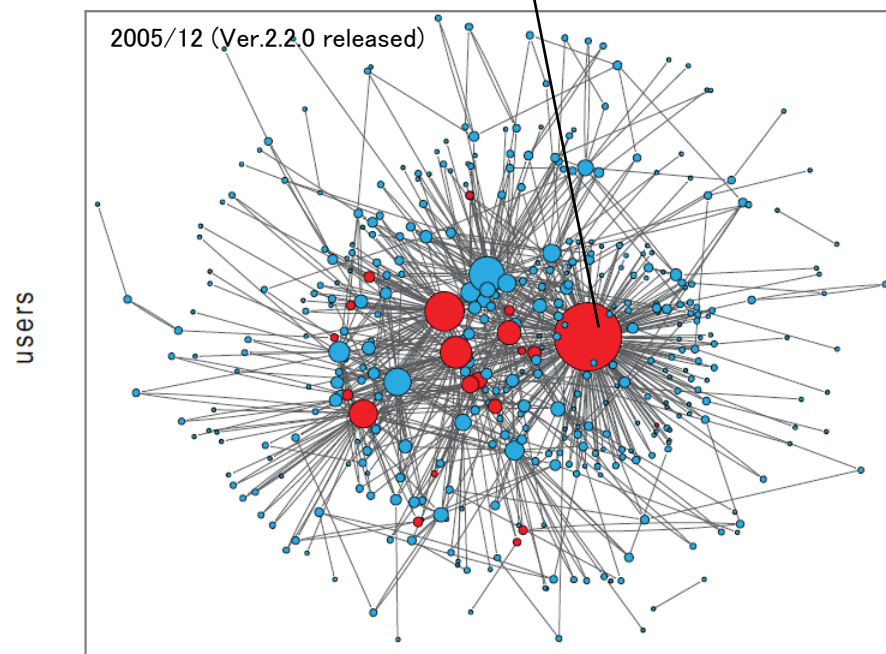
- ▶ ApacheとNetscapeのコミュニケーション構造
 - ▶ 開発者 (黄色のノード)



コミュニケーション: OSSコミュニティのコミュニケーション構造 (4)

- ▶ ApacheとNetscapeのコミュニケーション構造
 - ▶ ユーザ (青色のノード)

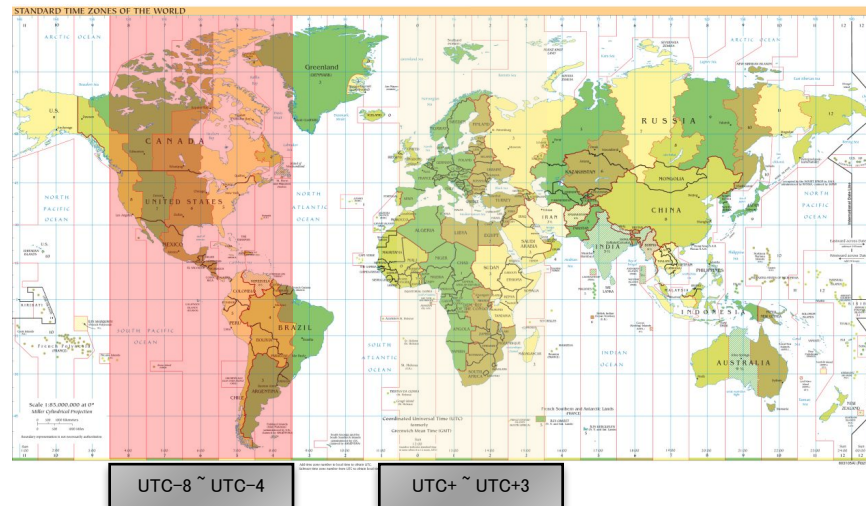
コア開発者がユーザを
積極的にサポート



2人の活発なユーザが存在する
が、コア開発者の関与が少ない

コミュニケーション: 地理的分散と時間的分散 (1)

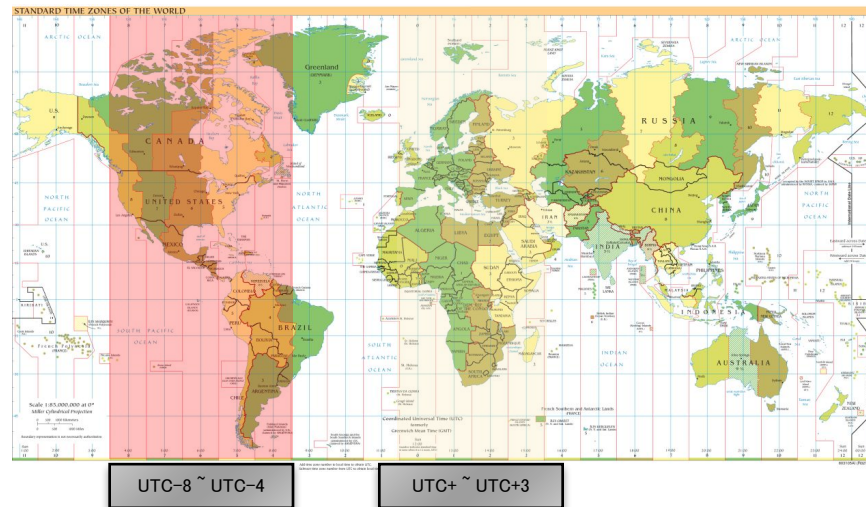
- ▶ OSS開発では、地理的に分散した環境下で開発が行われる



- ▶ 2種類の時間的分散が発生する
 - ▶ 地理的分散 => 異なるタイムゾーン
 - ▶ 作業する時間帯に制約がない => 同じタイムゾーンでも異なる作業時間帯が存在する
 - ▶ 学生なら一日中
 - ▶ 社会人なら終業後あるいは週末のみ, など

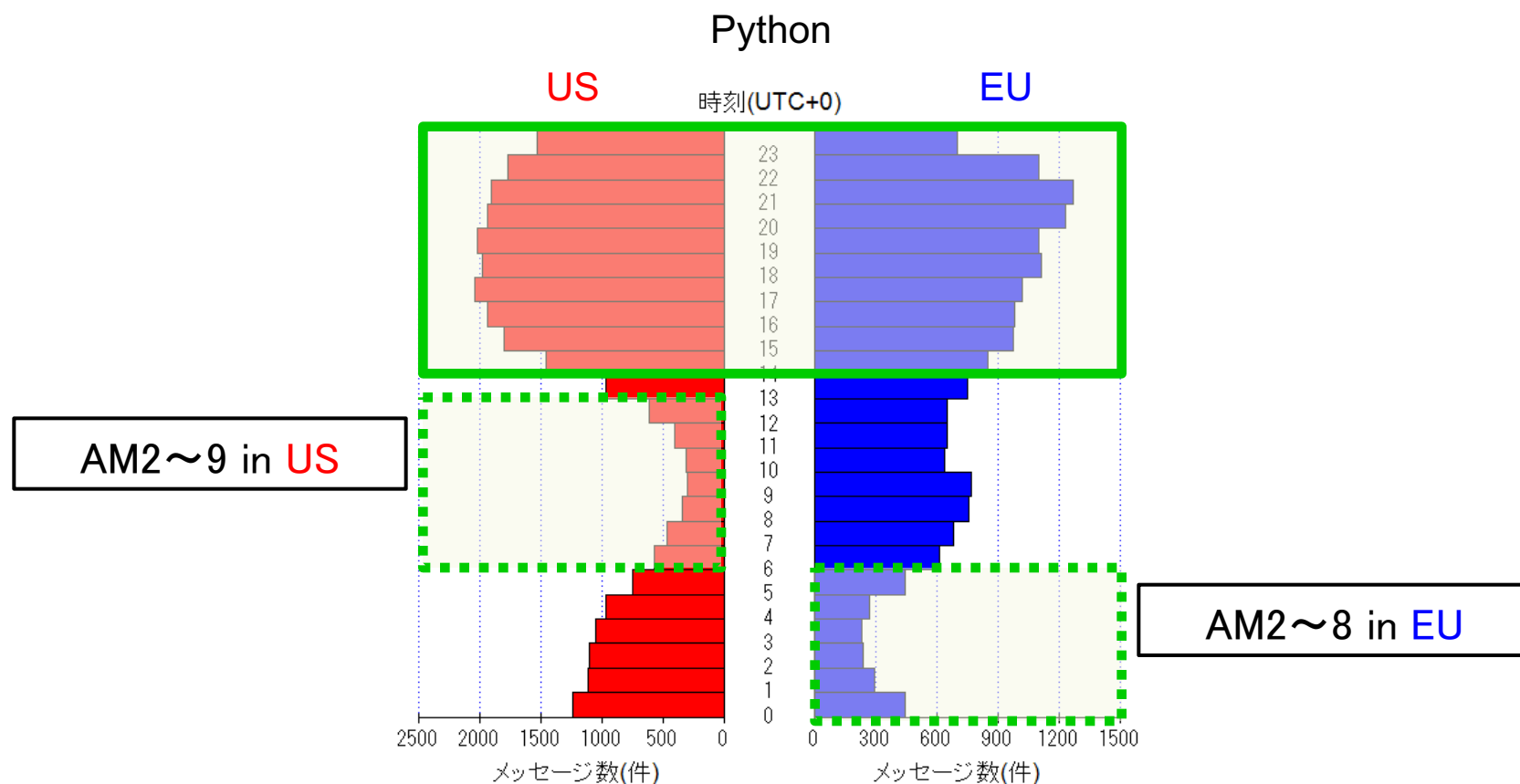
コミュニケーション: 地理的分散と時間的分散 (2)

- ▶ 複雑な「時差」の存在をどのように克服しているか？



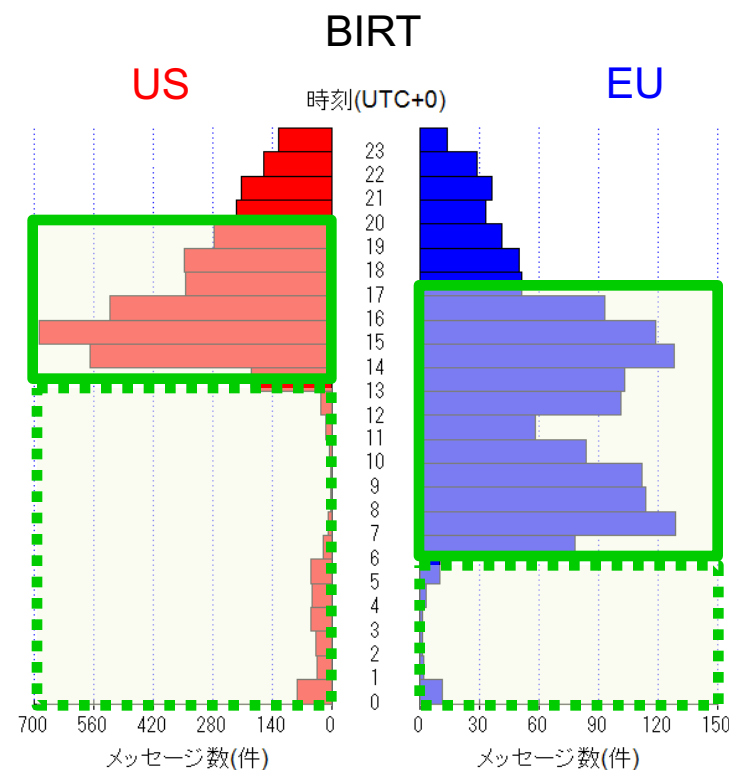
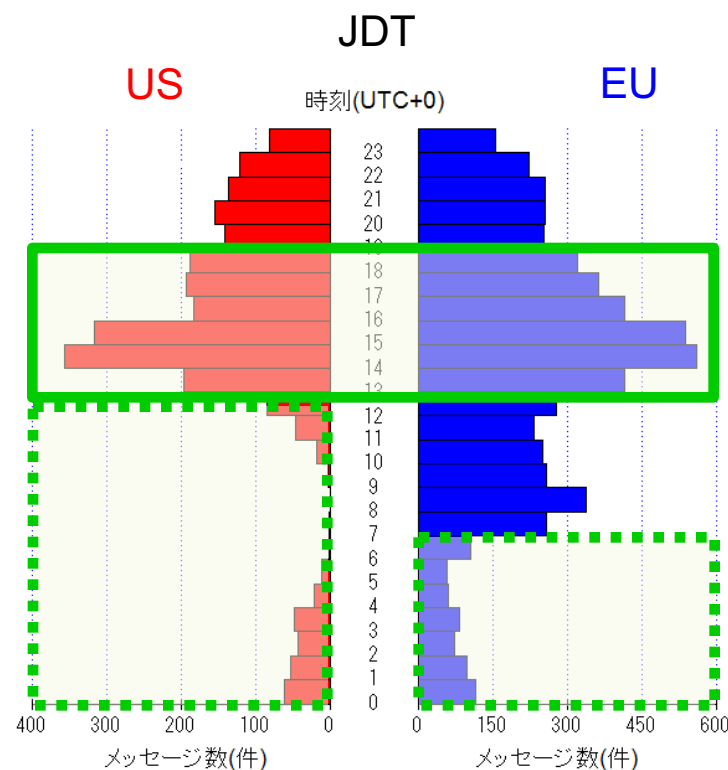
コミュニケーション: 地理的分散と時間的分散 (3)

▶ Pythonプロジェクト(“ピュアな” OSSプロジェクト)



コミュニケーション: 地理的分散と時間的分散 (4)

▶ Eclipseプロジェクト(企業主体のプロジェクト)



OSS研究成果の社会還元

- ▶ Modest @ Mozilla Japan
 - ▶ 開発者と利用者が共に開発のアイデアを出し合うための場を開設
- ▶ OpenOffice.org日本語プロジェクト
 - ▶ これまで存在しなかったユーザーメーリングリストを開設。ユーザー同士の助け合いから、さらに一歩進んで積極的にコミュニティへ貢献する人材の育成を目指す

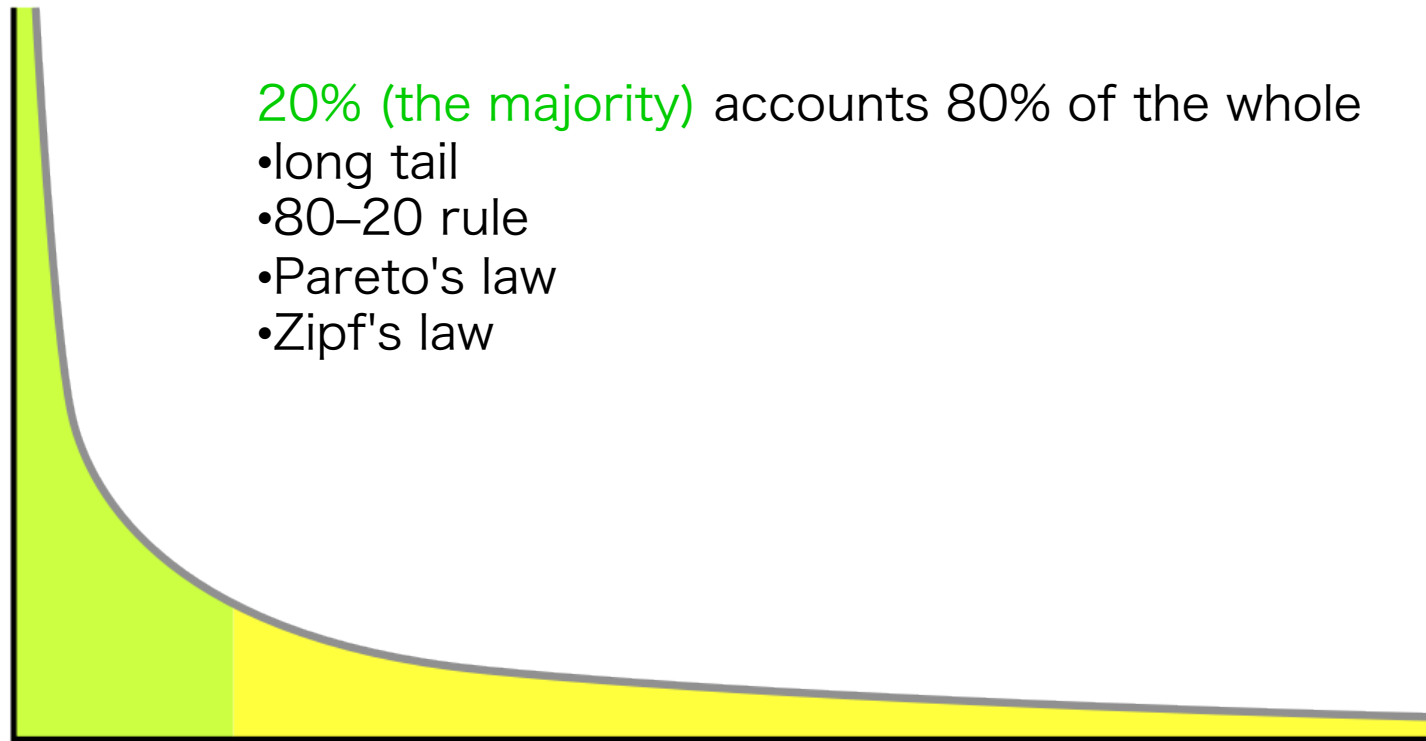


本日の内容

- ▶ コミュニティマネジメント
 - ▶ コミュニケーションとコーディネーション
 - ▶ 参加者間のコミュニケーション
 - ▶ コーディネーション：地理的分散と時間的分散
 - ▶ コラボレーション
 - ▶ べき乗の法則 (Power law relationships)
 - コミュニティの分布
 - コミュニケーションの分布
 - コントリビューション (貢献) の分布
 - ▶ コントリビューション (貢献)
 - 参加者のライフタイム
 - 期待されている貢献の種類
 - 参加者の社会的関係の影響

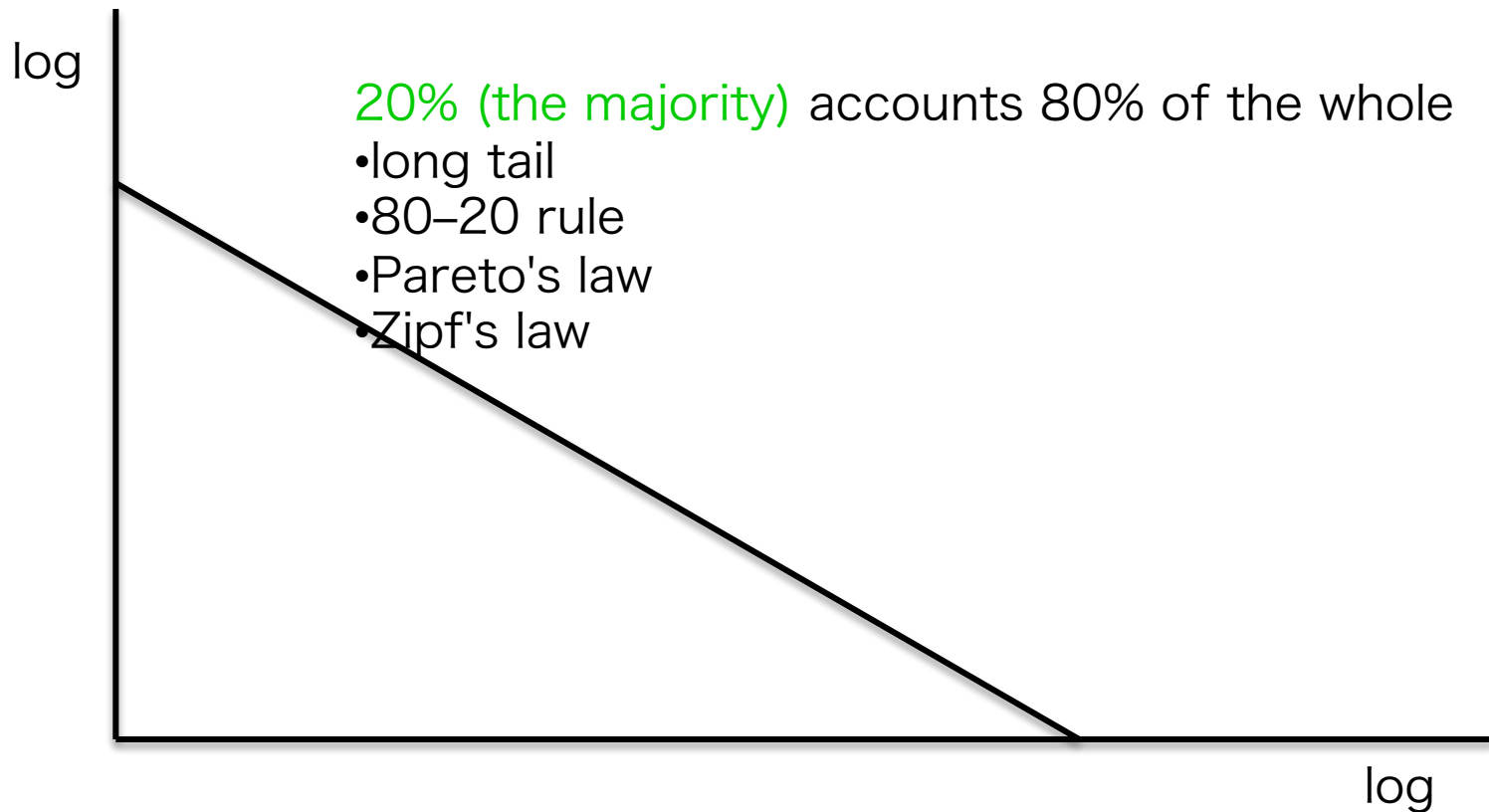
コラボレーション: べき乗の法則 (1)

- ▶ OSS開発のあらゆる側面に「べき乗の法則」を観察することができる



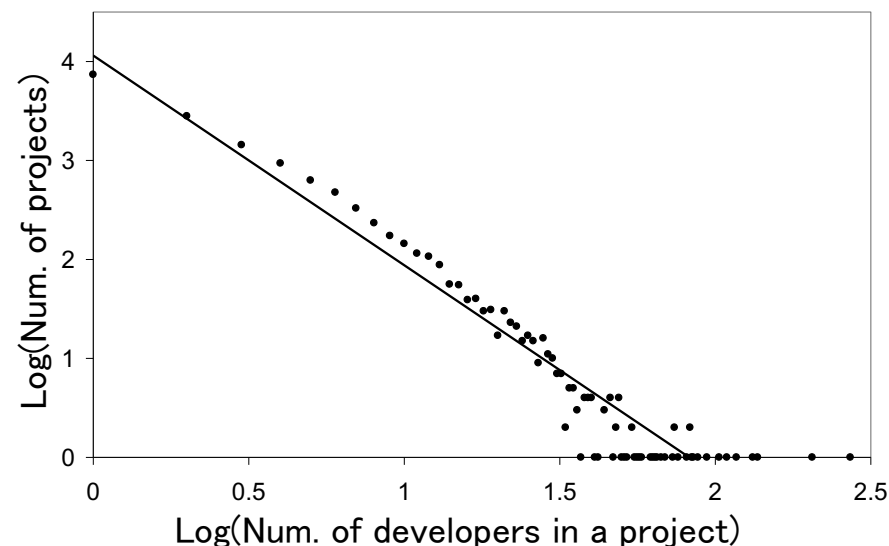
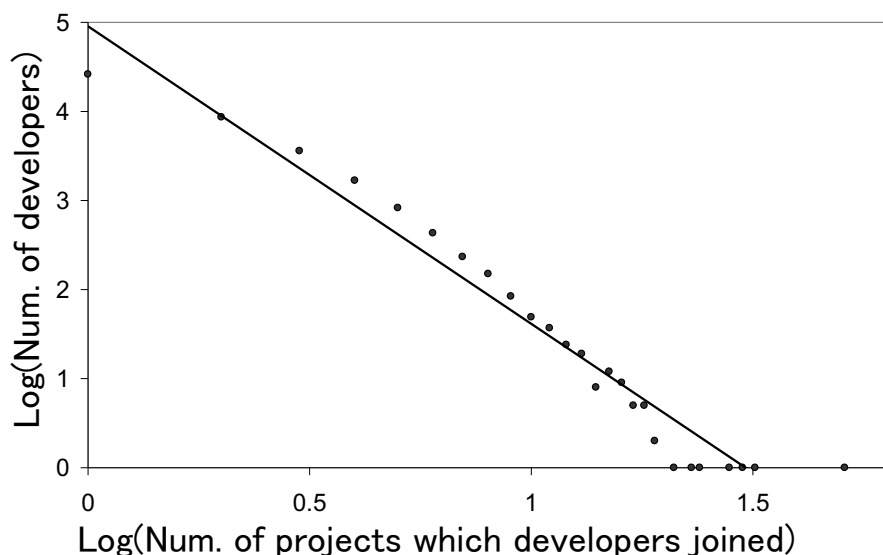
コラボレーション: べき乗の法則 (2)

- ▶ OSS開発のあらゆる側面に「べき乗の法則」を観察することができる



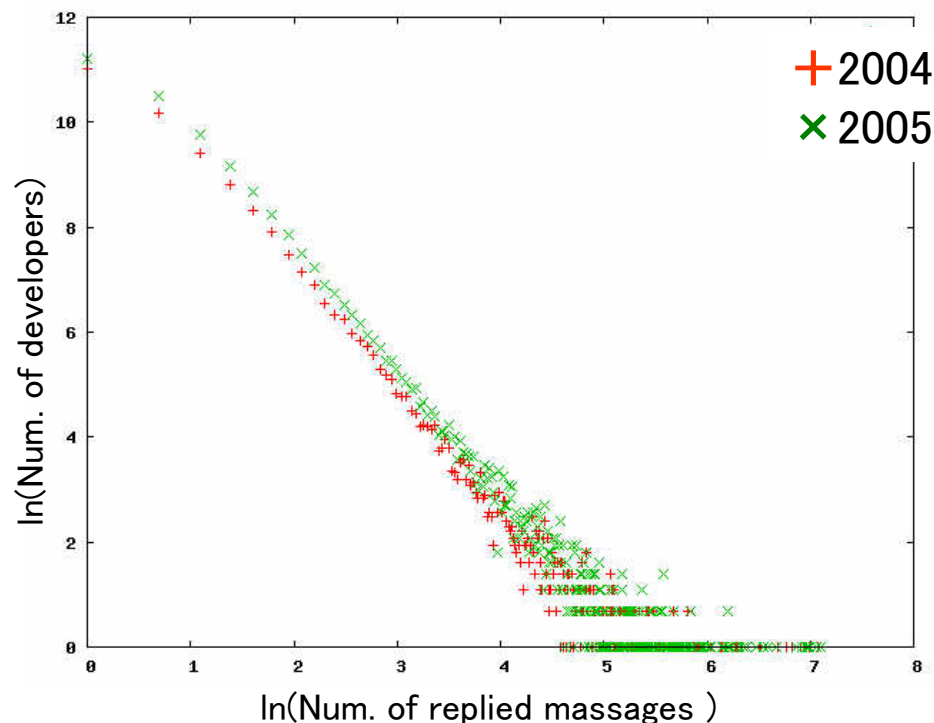
コラボレーション: べき乗の法則 (3)

- ▶ 所属関係
 - ▶ SourceForge.net
 - ▶ 130,000人の開発者
 - ▶ 90,000件のプロジェクト



コラボレーション: べき乗の法則 (4)

- ▶ 返信関係
 - ▶ SourceForge.netのBBS
 - ▶ 167,000人の開発者
 - ▶ 1,230,000件の返信数 (リプライ)



コラボレーション: べき乗の法則 (5)

- ▶ Apacheにおけるコミット数（コード変更回数）
 - ▶ 上位15人のコア開発者のコミット数が全体の80%以上を占める

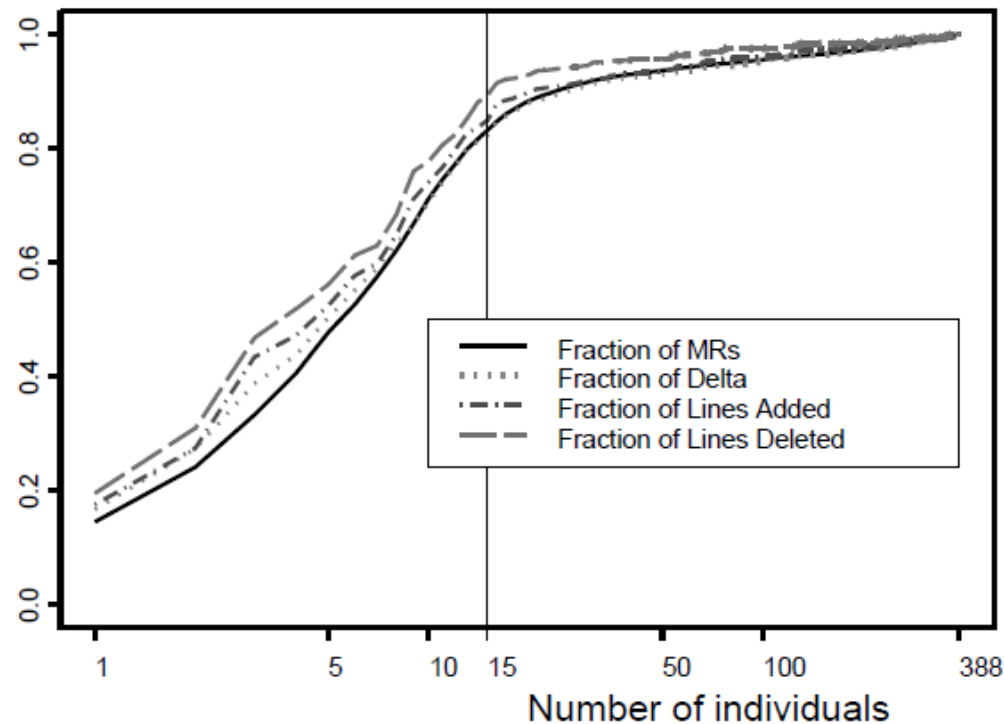


Fig. 1. The cumulative distribution of contributions to the code base.

コラボレーション: べき乗の法則 (6)

- ▶ Apacheにおける不具合修正数
 - ▶ 上位15人のコア開発者の不具合修正数が全体の60%以上を占める

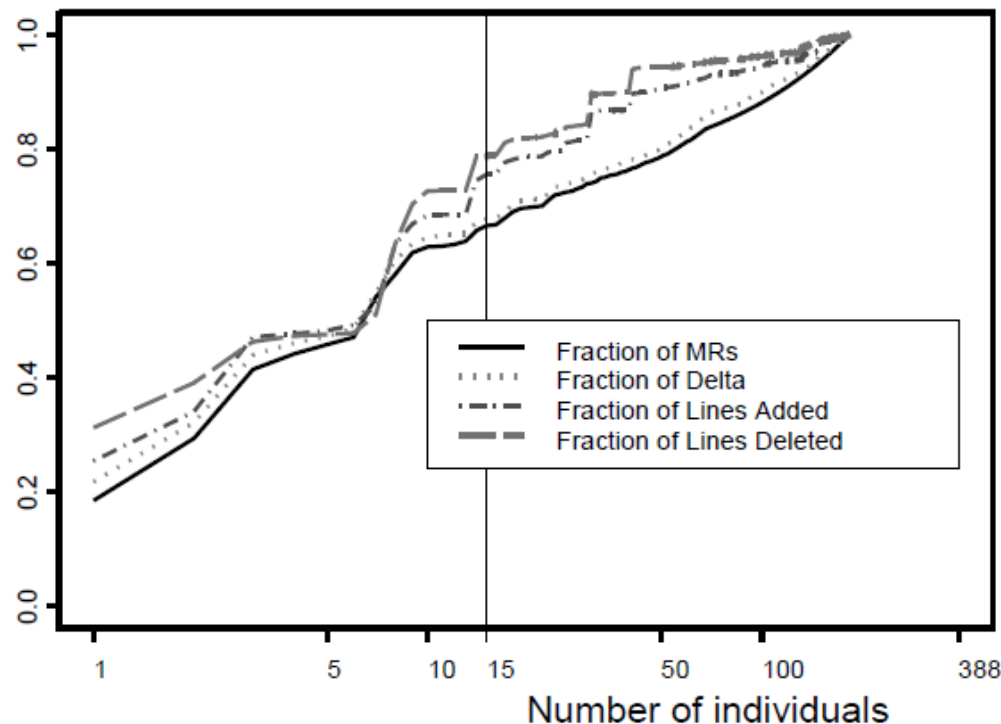


Fig. 2. Cumulative distribution of fixes.

本日の内容

- ▶ コミュニティマネジメント
 - ▶ コミュニケーションとコーディネーション
 - ▶ 参加者間のコミュニケーション
 - ▶ コーディネーション：地理的分散と時間的分散
 - ▶ コラボレーション
 - ▶ べき乗の法則 (Power law relationships)
 - コミュニティの分布
 - コミュニケーションの分布
 - コントリビューション (貢献) の分布
 - ▶ **コントリビューション (貢献)**
 - 参加者のライフタイム
 - 期待されている貢献の種類
 - 参加者の社会的関係の影響

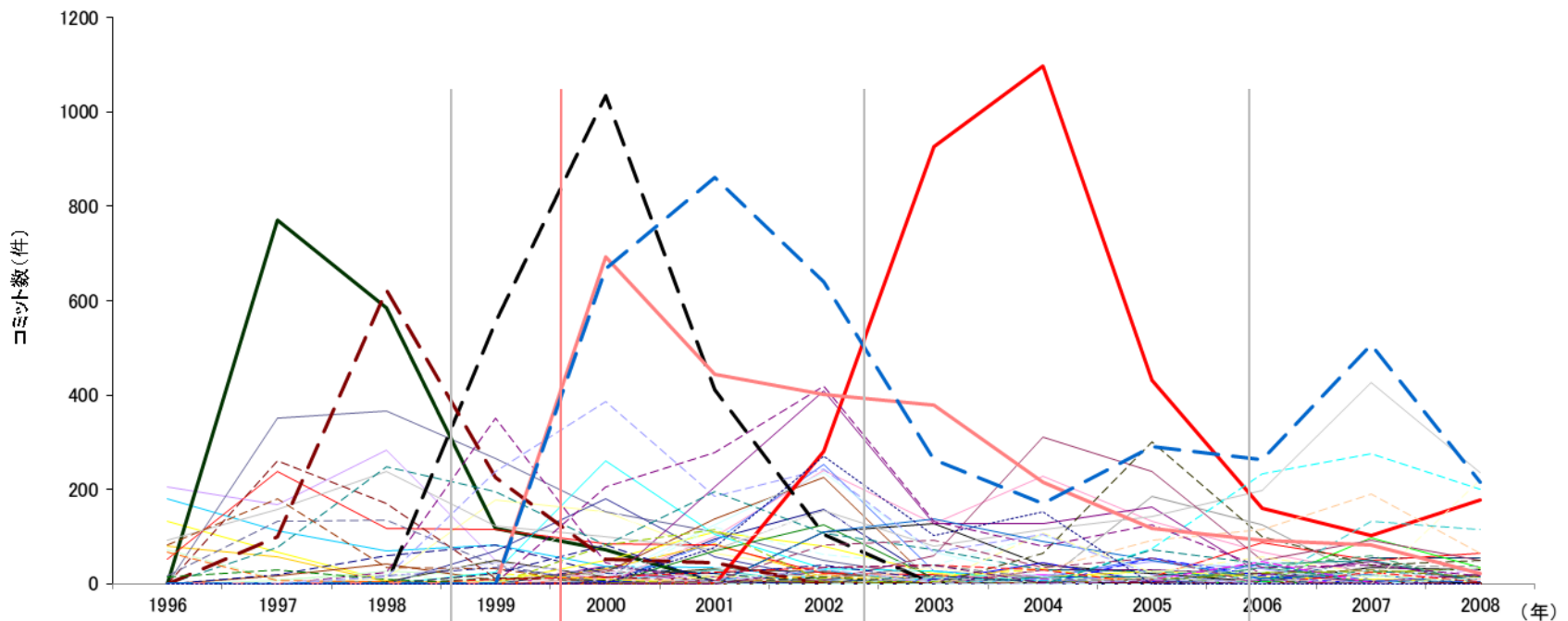
コラボレーション：コントリビューション (1)

- ▶ 参加者のライフタイム
 - ▶ 伝統的なOSSコミュニティでは、ほとんどの開発者はボランティアである
 - ▶ ボランティア開発者は自由に参加／離脱することができる
- ▶ プロジェクトに参加している期間はどれくらいか？
- ▶ いつ／なぜプロジェクトを去るのか？

コラボレーション：コントリビューション (2)

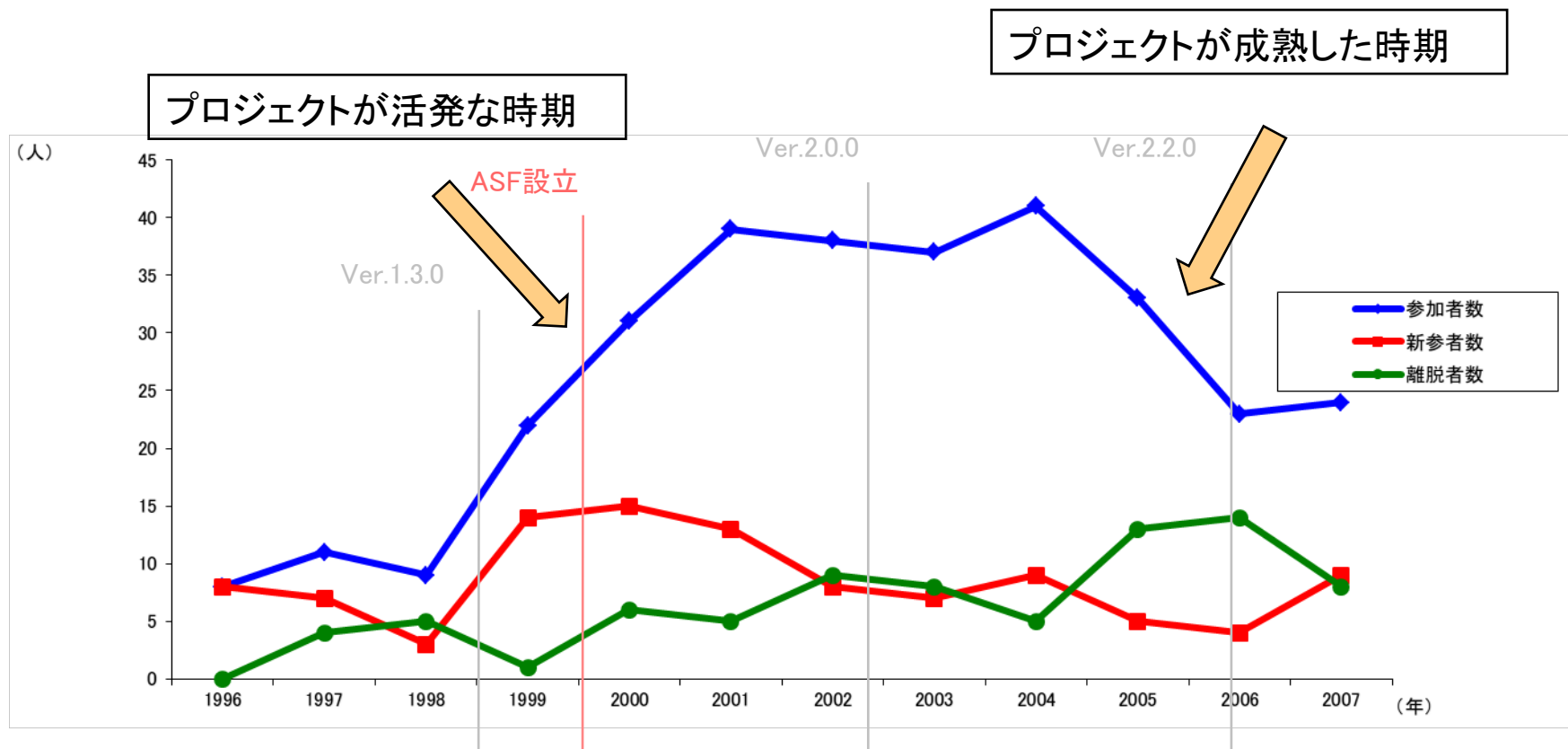
▶ Apacheにおけるコミットのライフタイム

- 多くのコミットはすぐにやめてしまう
- 開発者の活動はどこかの時点にピークがある



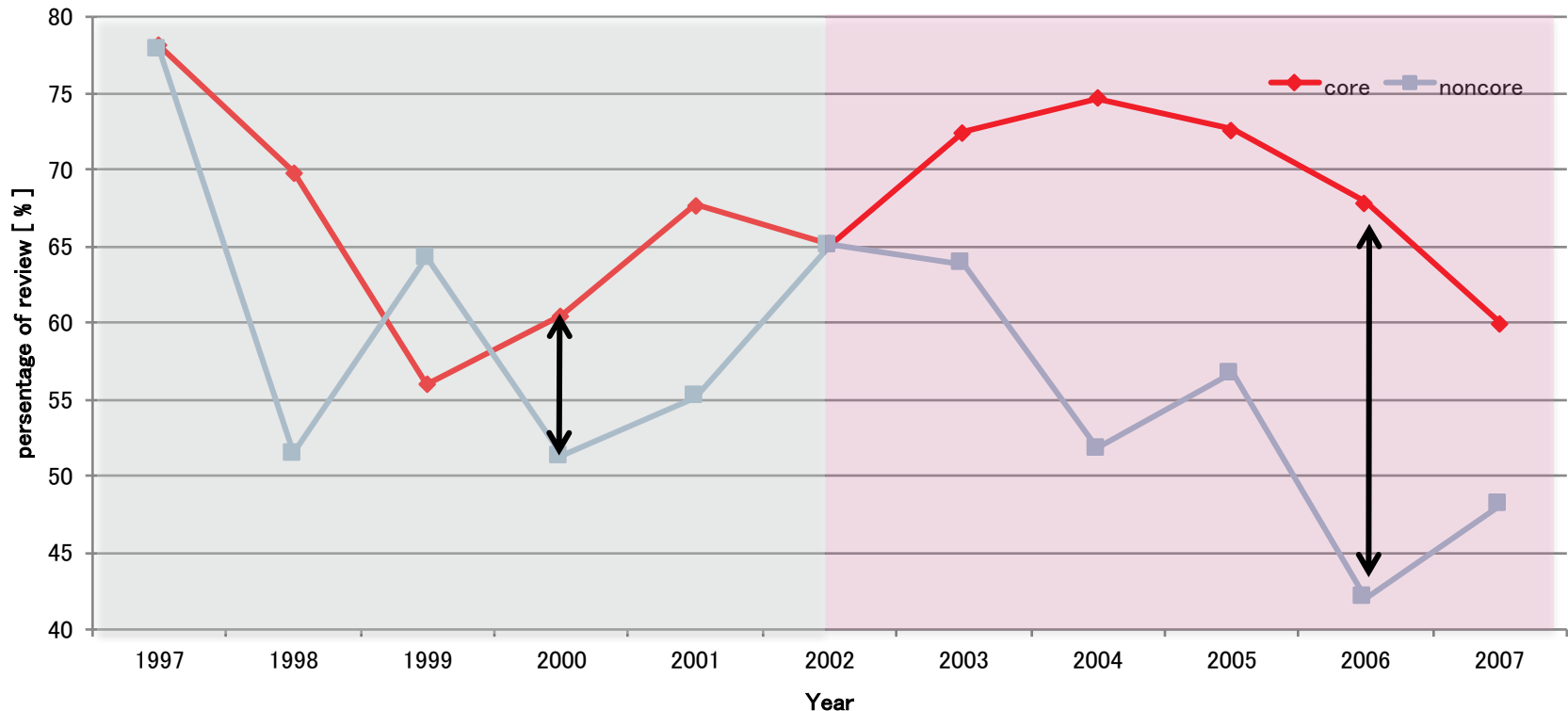
コラボレーション：コントリビューション (3)

▶ Apacheにおけるコミッタのライフタイム



コラボレーション：コントリビューション (4)

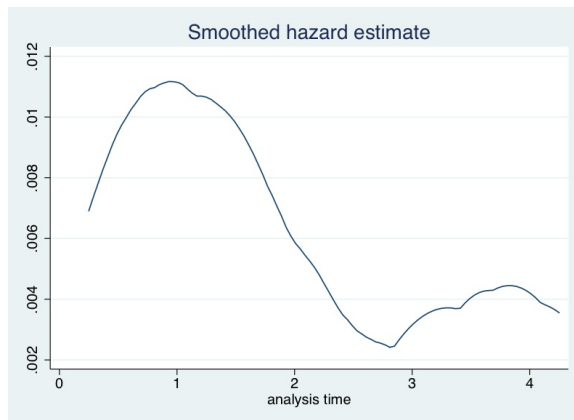
- ▶ Apacheにおけるパッチレビュー率の比較
 - ▶ コア開発者 vs. 一般開発者
 - ▶ プロジェクトが成熟するにつれて差がでてくる？



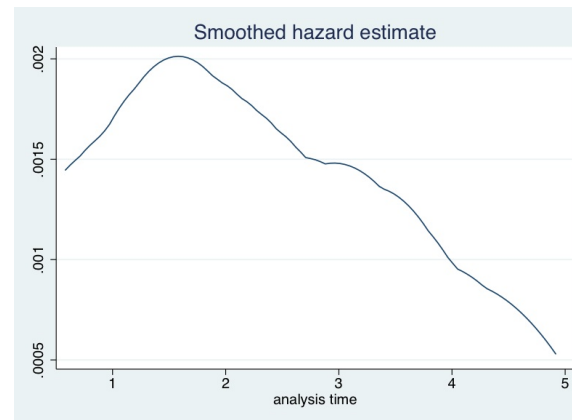
コラボレーション：コントリビューション (5)

- ▶ 参加者のライフタイム
 - ▶ 生存分析 (Hazard Rate Analysis) [Bird2007]

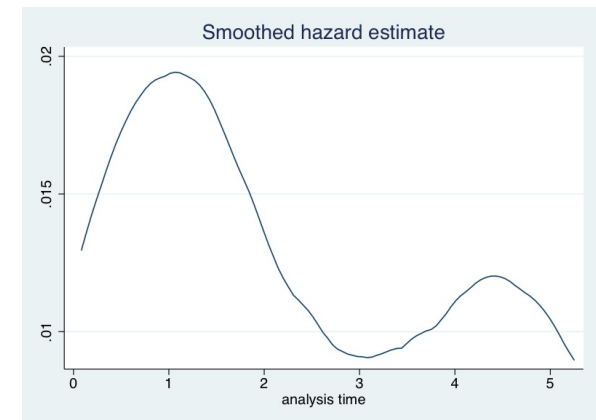
Christian Bird, Alex Gourley and Prem Devanbu, "Open borders? immigration in open source projects," Proceedings of the Fourth International Workshop on Mining Software Repositories (MSR'07), No.6, pp.2007.



Apache



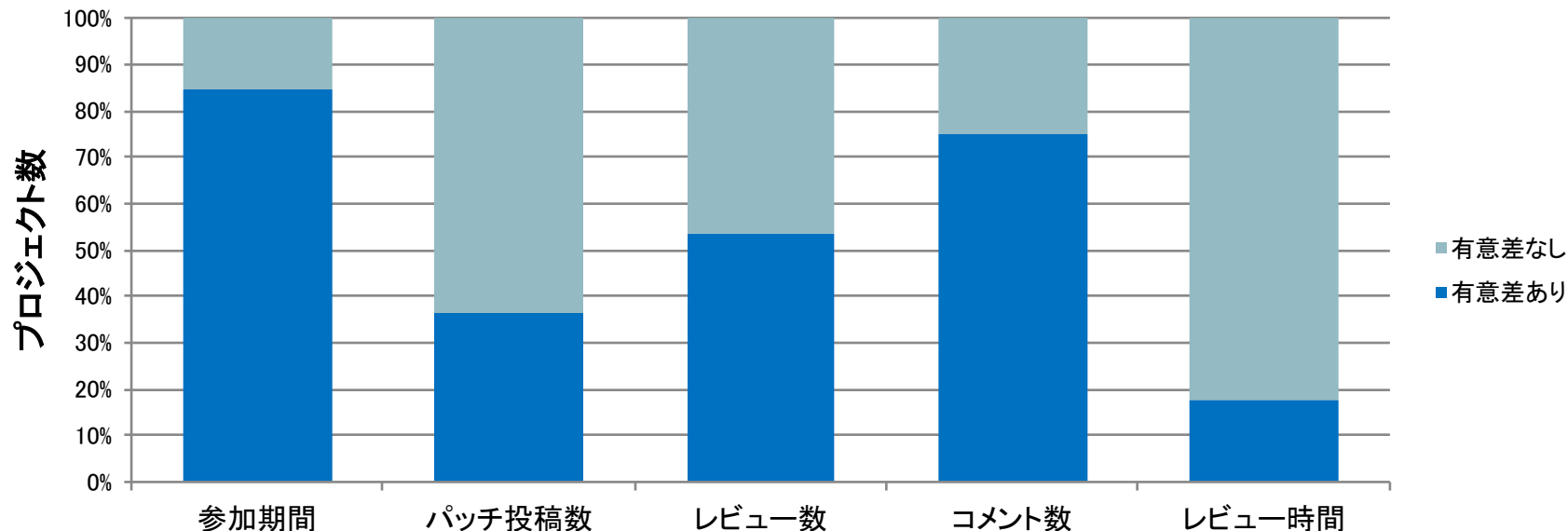
Postgres



Pyhthon

コラボレーション：コントリビューション (6)

- ▶ 期待されている貢献の種類（コアメンバとなるために）
 - ▶ どの様な開発者がコア開発者に昇格されるのか？
 - ▶ どの様な貢献が昇格されるために望まれているのか？
- ▶ 32プロジェクトの開発者の分析
 - ▶ コミッタ昇格前の開発者の活動量とコミッタ昇格との関係



レポート課題：Social Coding

▶ Github

- ▶ 1,000万人以上のユーザが利用する，現在最も人気のある開発者向けホスティングサービス
 - ▶ 1000万以上のリポジトリ（gitリポジトリ）
 - ▶ プルリクエスト型のソフトウェア開発
 - ▶ ソーシャルコーディング

▶ 課題

- ▶ Githubの人気の理由を，次のキーワードの説明を含めてA4x2枚程度で考察しなさい.
 - ▶ ソーシャルコーディング
 - ▶ プルリクエスト
 - ▶ 分散バージョン型管理システム（Git）
 - ▶ ブランチ（forkあるいはclone）

レポート課題：Social Coding

- ▶ 前回のレポート課題AとBの両方を提出した人は提出免除
- ▶ 提出先：reports@oss.sys.wakayama-u.ac.jp
- ▶ メールタイトル：ソフトウェア工学特論
- ▶ 期限：10月31日（月）12:00まで

スケジュール（大平担当分）

- ▶ 10月3日：オープンソースソフトウェア（1）
- ▶ 10月10日：休み（体育の日）
- ▶ 10月17日：オープンソースソフトウェア（2）
- ▶ 10月24日：リポジトリマイニング（1）
- ▶ 10月31日：リポジトリマイニング（2）