日本ソフトウェア科学会第33回大会 チュートリアル
2016年9月6日　於：東北大学

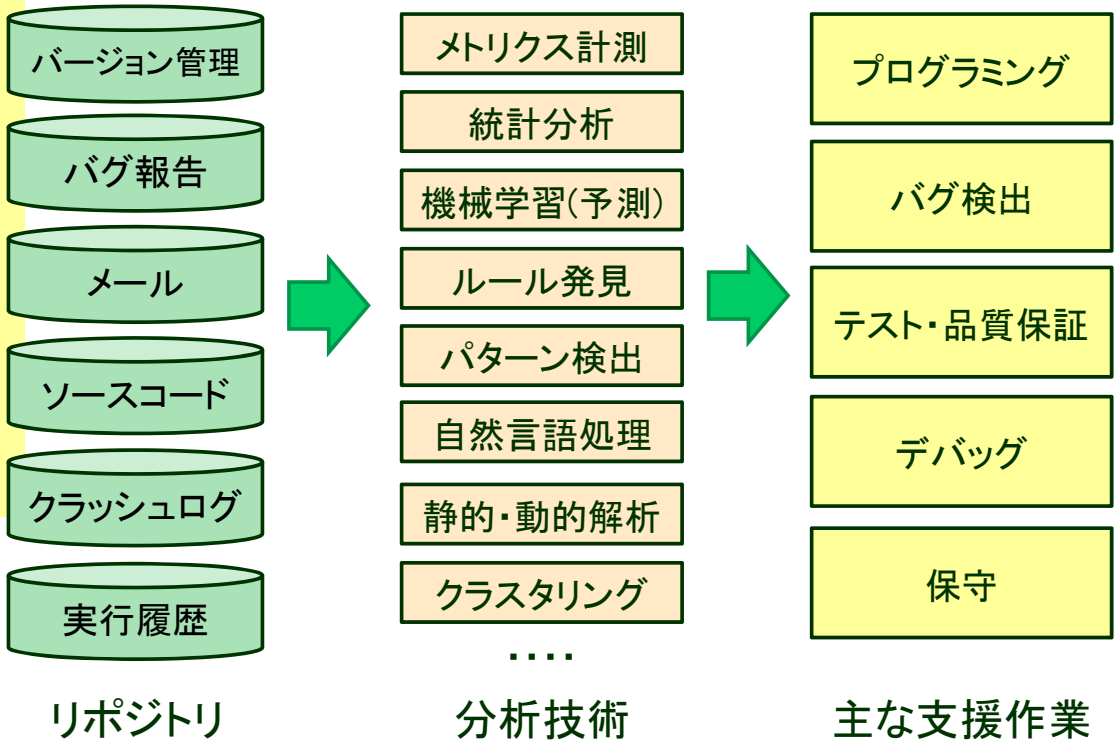
ソフトウェアリポジトリマイニングのすすめ

岡山大学大学院自然科学研究科
門田 暁人
monden@okayama-u.ac.jp

リポジトリマイニングとは

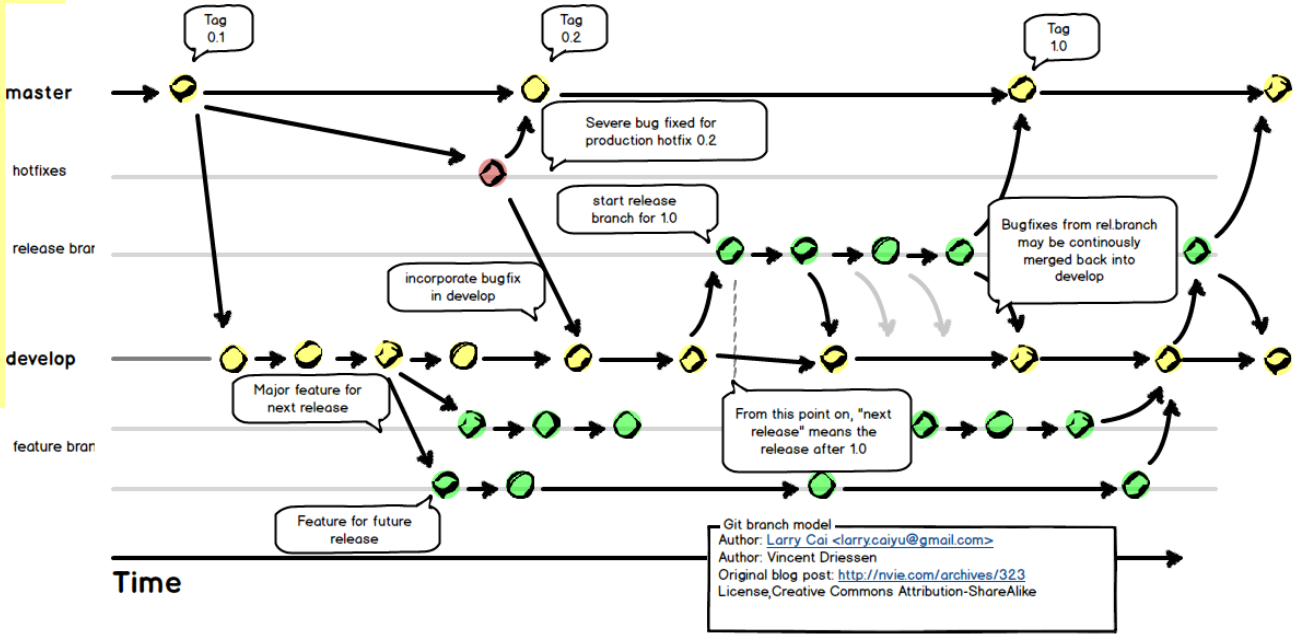
ソフトウェアリポジトリという鉱山から鉱石や鉱脈を掘り当てる.

リポジトリマイニングとは



バージョン管理システム(Git)

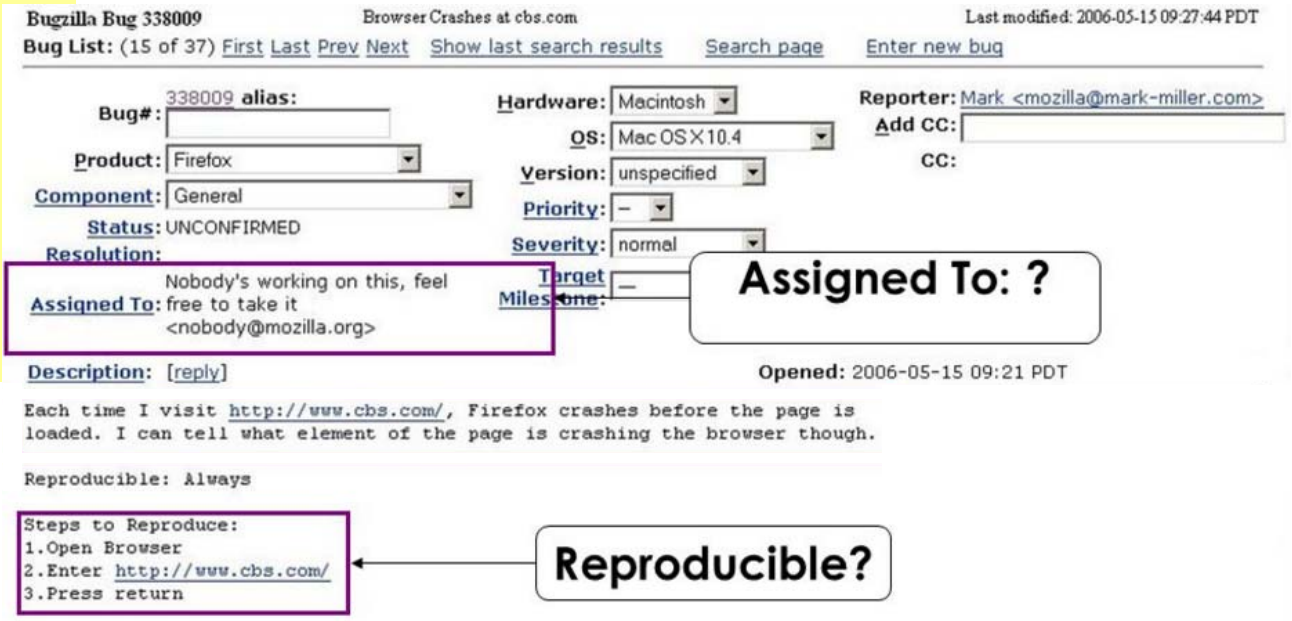
- 変更履歴をブランチ，ファイル単位で記録(ソースコード，開発者，変更時期，コメント等)．プルリクエスト，マージの管理．



https://mockupstogo.mybalsamiq.com/projects/diagrams/Git%20Workflow

バグ管理システム（Bugzilla）

- 対象プロダクト，コンポーネント，バージョン，優先順位，重要度，報告者，修正者，修正状況，バグの症状等が記録される．



L. Tan, T. Xie, http://slideplayer.com/slide/4130379/

リポジトリマイニングとは

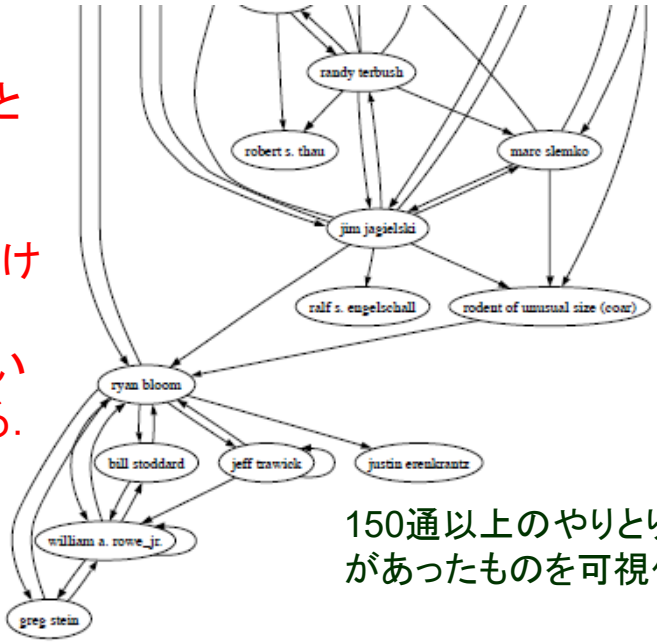
大きな鉱脈を掘り当てるのが研究の醍醐味！

鉱脈の例

■E-mailに基づくソーシャルネットワーク分析 [1]

Google Scholarによると
被引用回数449 ！！

・何かを解決しているわけ
ではない.
・テーマが新しくて面白い
・中身もしっかりしている.



150通以上のやりとり
があったものを可視化

[1] C. Bird, A. Gourley, P. Devanbu, M. Gertz, A. Swaminathan, Mining email social network, MSR2006. (Acceptance rate = 48%)

鉱脈の例

■ストーリー

- 対話と協調は，巨大プロジェクトでは重要である.
- 従来，商用システム開発では分析が難しかった.
- OSSではメールベースで対話するので分析が可能となった.
- E-mailのやりとりをソーシャルネットワーク化するにあたって，「名寄せ」問題を解決する. →その後の論文でよく引用される.
- 分析(Apache HTTP Serverが対象)
  - スケールフリー性（メッセージ数-人数がlog-log関係）
  - Betweennessの分析

→SE分野におけるソーシャルネットワーク分析の先駆けとしてよく引用される.

[1] C. Bird, A. Gourley, P. Devanbu, M. Gertz, A. Swaminathan, Mining email social network, MSR2006. (Acceptance rate = 48%)

鉱脈の例

- 派生した鉱脈
  - SNAに基づくバグトリアージ[1]（被引用数238）
  - SNAに基づく組織構造の分析 [2]（被引用数208）
  - SNAに基づくバグ予測[3]（被引用数148）
  - SNAに基づくビルド失敗予測[4]（被引用数146）
  - ・・・

ソーシャルネットワーク分析を用いた○○，は一大ブームとなった．

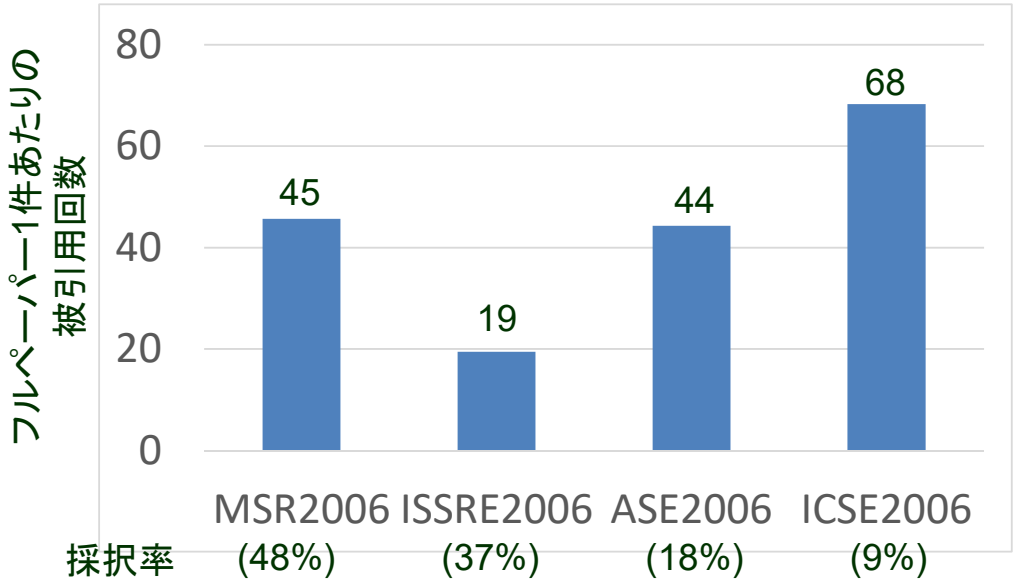
[1] G. Jeong, S. Kim, Improving bug triage with bug tossing graphs, FSE2009.
[2] C. Bird, D. Pattison, R. D'Souza, V. Filkov, P. Devanbu, Latent social structure in open source projects, FSE2008.
[3] A. Meneely, L. Williams, W. Snipes, J. Osborne, Predicting failures with developer networks and social network analysis, FSE2008.
[4] T Wolf, A Schroter, D Damian, T Nguyen, Predicting build failures using social network analysis on developer communication, ICSE2009.

リポジトリマイニングの現状

- ソフトウェア工学分野の一大テーマとなっている．
  - 若手研究者が，ICSE，FSE，ASEなどトップカンファレンスに次々と論文を発表している．
    - Ahmed E. Hassan, Tao Xie, Sunghun Kim, Tien Nguyen, Yann-Gaël Guéhéneuc , Bram Adams , Abram Hindle, ・・・など．
  - 研究者コミュニティが大きくなり，関連する会議も増えた．
    - MSR，MSR Summit
    - WCRE + CSMR → SANER
    - IWESEP
    - SWAN
    - SoftwareMining

リポジトリマイニングの現状

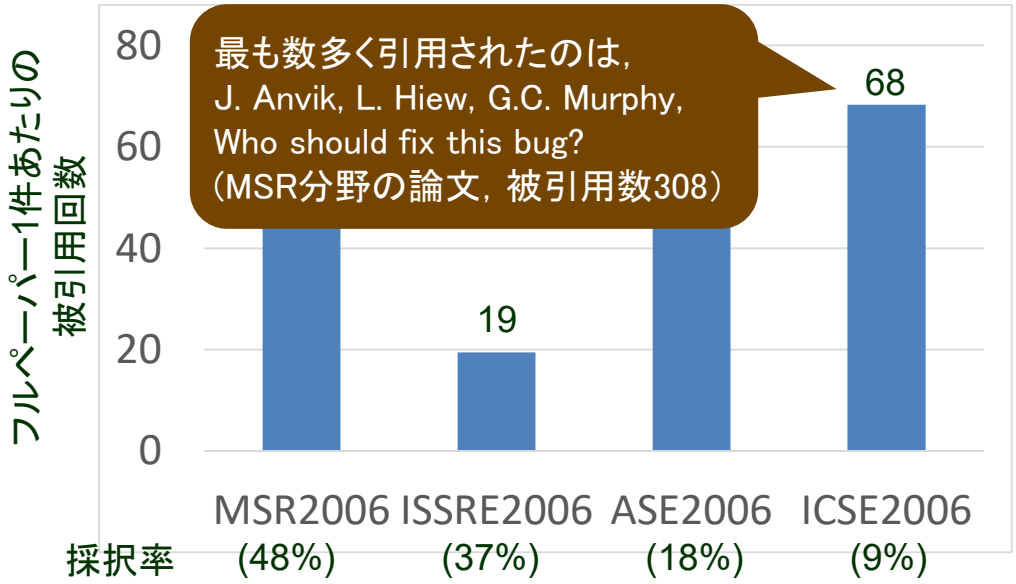
- 引用されやすい.



[1] 門田暁人, 伊原彰紀, 松本健一: ソフトウェアリポジトリマイニング,コンピュータソフトウェア, Vol. 30, No. 2, pp.52-65, May 2013.

リポジトリマイニングの現状

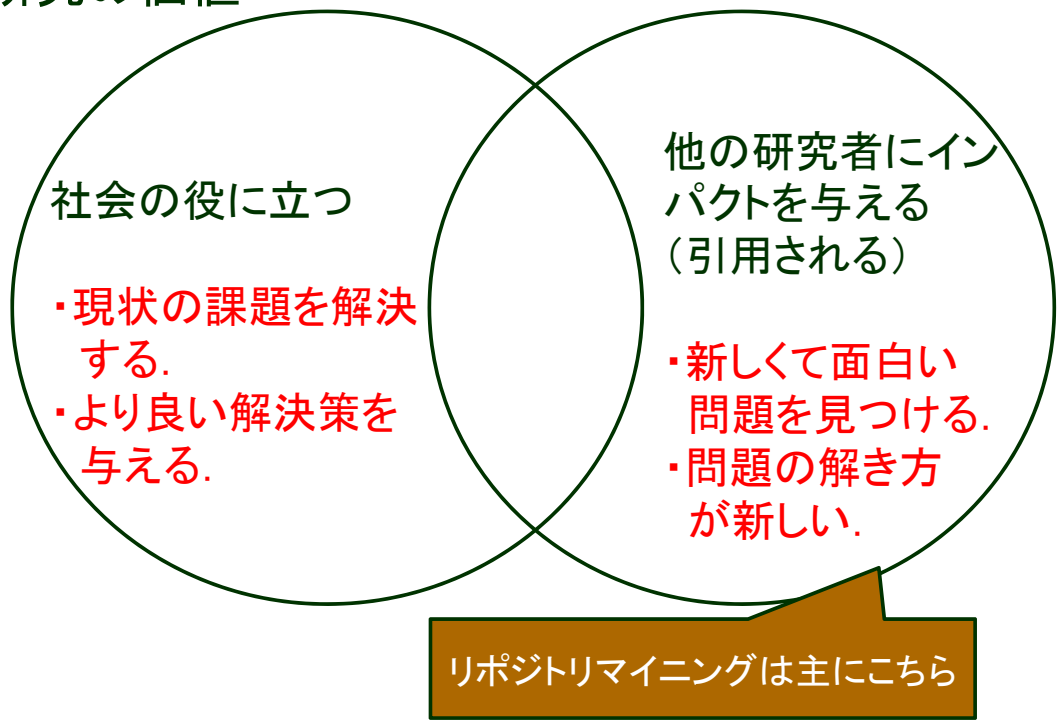
- 引用されやすい.



[1] 門田暁人, 伊原彰紀, 松本健一: ソフトウェアリポジトリマイニング,コンピュータソフトウェア, Vol. 30, No. 2, pp.52-65, May 2013.

話はそれますが. . .

■研究の価値



なぜリポジトリマイニングが流行っているのか？

■オープンソースソフトウェア（OSS）の普及

- リポジトリベースの開発
  - ◆バージョン管理リポジトリ
  - ◆バグ管理リポジトリ
  - ◆メーリングリスト
- OSSはソフトウェア業界の必需品
  - ◆従来

    「OSS開発は趣味の世界」

    「OSSの研究は実際の（企業の）開発現場では使えない」
  - ◆今日

    （企業の）ソフトウェア開発はOSSから始まる[1]

[1] 野村佳秀: 米国大学および国内企業でのOSSプロジェクト活用の実際, ソフトウェアエンジニアリングシンポジウム2013ワークショップ, オープンソースソフトウェア工学セッション (2013). https://sites.google.com/site/sesjp2013/workshop#ws-5

なぜリポジトリマイニングが流行っているのか？

■デジタルテレビに使用されているOSS[1]

カテゴリー	使用OSS例	コード量概数（千コード）
OS	Linux kernel	13,920
ランタイム処理	busybox	314
	parted	364
	e2fsprogs	331
	xfsprogs	218
ネットワーク処理	OpenSSL	615
	netfilter/iptables	45
	WIDE-DHCPv6	37
グラフィックス処理	DirectFB	452
	flare	45
DBMS	SQLite	205
ライブラリ	glibc	2,606
	newlib	1,719
	libupnp	131
小　計		21,002

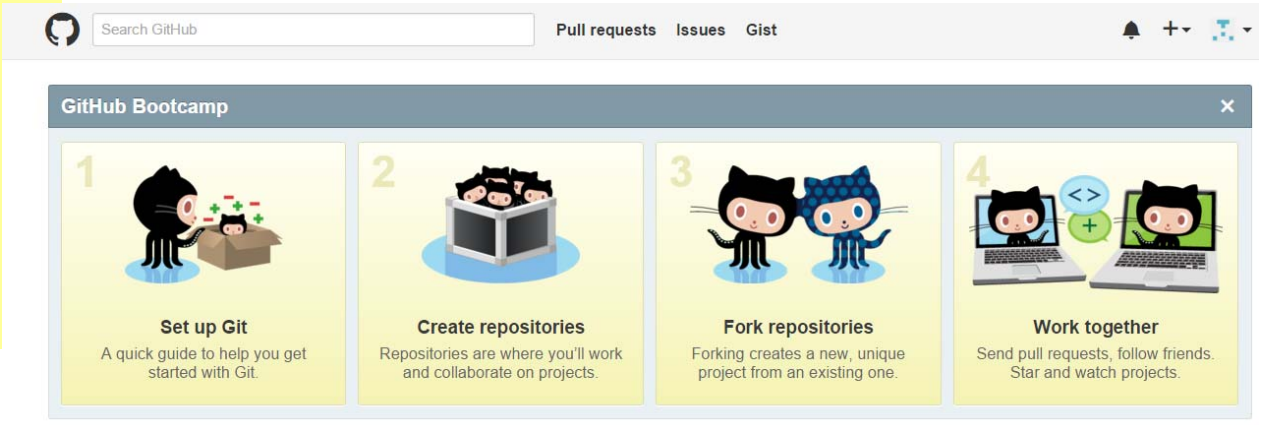
- 10種類以上，2千万行を超えるOSSを活用している．これらをベースに独自機能を追加開発している．
- 東芝では，ほぼ全てのデジタルテレビとHDDレコーダにLinuxを搭載している（いた）．

[1] 長谷川，野末，“OSSを活用したソフトウェア開発の動向”,東芝レビュー，Vol.67, No.8, pp.2-6, 2012.

なぜリポジトリマイニングが流行っているのか？

■リポジトリホスティングサービスの発展

- GitHub（約2000万Gitリポジトリ，100万人）
- データの宝庫！



ソフトウェア工学分野におけるビッグデータ時代の到来！

なぜリポジトリマイニングが流行っているのか？

■ソーシャルコーディング時代の到来

□Git，GitHubにおける革新的ソフトウェア開発

	GitHub	従来開発
機能	思いもよらない斬新な機能やしくみの追加(Pull request)	低コストで仕様を満たす．
開発効率	コアメンバー ＋ 不特定多数による開発	限られた人間による開発
	非同期の開発・レビュー（分散リポジトリ）	日程調整
品質	不特定多数によるバグ報告	テストケース異存
	「見られる意識」による品質向上	「動けばよい」開発
モチベーション	（技術のある）個人に活躍の場	人月単価，外注，派遣
	社会への貢献	組織に閉じた貢献
技術向上	他人のコードを見ることによる技術向上	担当外の仕事をやる気にならない

なぜリポジトリマイニングが流行っているのか？

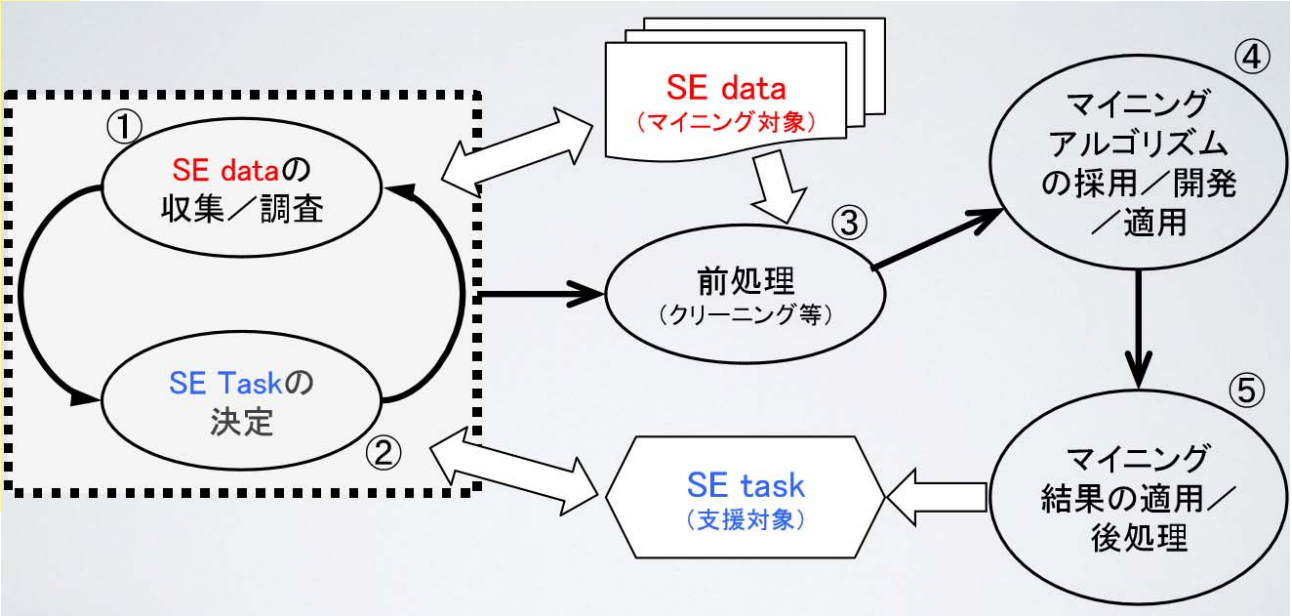
■ソーシャルコーディング時代の到来

□Git，GitHubにおける革新的ソフトウェア開発

	GitHub	従来開発
機能	思いもよらない斬新な機能やしくみの追加(Pull request)	低コストで仕様を満たす．
開発効率	コアメンバー ＋ 不特定多数による開発	限られた人間による開発
	非同期の開発・レビュー（分散リポジトリ）	日程調整
品質		
モチベーション		
	社会への貢献	組織に閉じた貢献
技術向上	他人のコードを見ることによる技術向上	担当外の仕事をやる気にならない

研究の主戦場を，従来のウォーターフォール開発からリポジトリベースの分散開発に移行するのは理に適っている？？

リポジトリマイニングの手順



大平雅雄, 開発データマイニングのススメ:「見える化」の次のステップ, SECセミナー 定量的なプロジェクト管理・プログラム管理のススメ～見える化による生産性・信頼性の向上～, 2015年10月30日
http://sec.ipa.go.jp/users/seminar/seminar_tokyo_20151030-02.pdf

リポジトリマイニングの要素技術

ソフトウェア（コード）解析

静的／動的解析, 形式手法, ソフトウェアメトリクス

統計解析

平均, 最頻値, 中央値, 分散, 標準偏差, 共分散, 相関係数, 誤差, 偏差値, 無作為抽出（ランダムサンプリング）期待値, 不偏分散, 有意差, 尤度関数, 回帰分析, 重回帰分析因子分析, 主成分分析, 判別分析, 共分散構造分析

データマイニング

頻出パターン（相関ルール分析）, 判別分析（単純ベイズ分類器, 決定木, サポートベクターマシン）, 回帰分析（線形回帰、ロジスティック回帰）, クラスタリング

テキストマイニング

形態素解析, N-gram, 共起関係, tf-idf, LSA (LSI), LDA, 主成分分析, クラスター分析

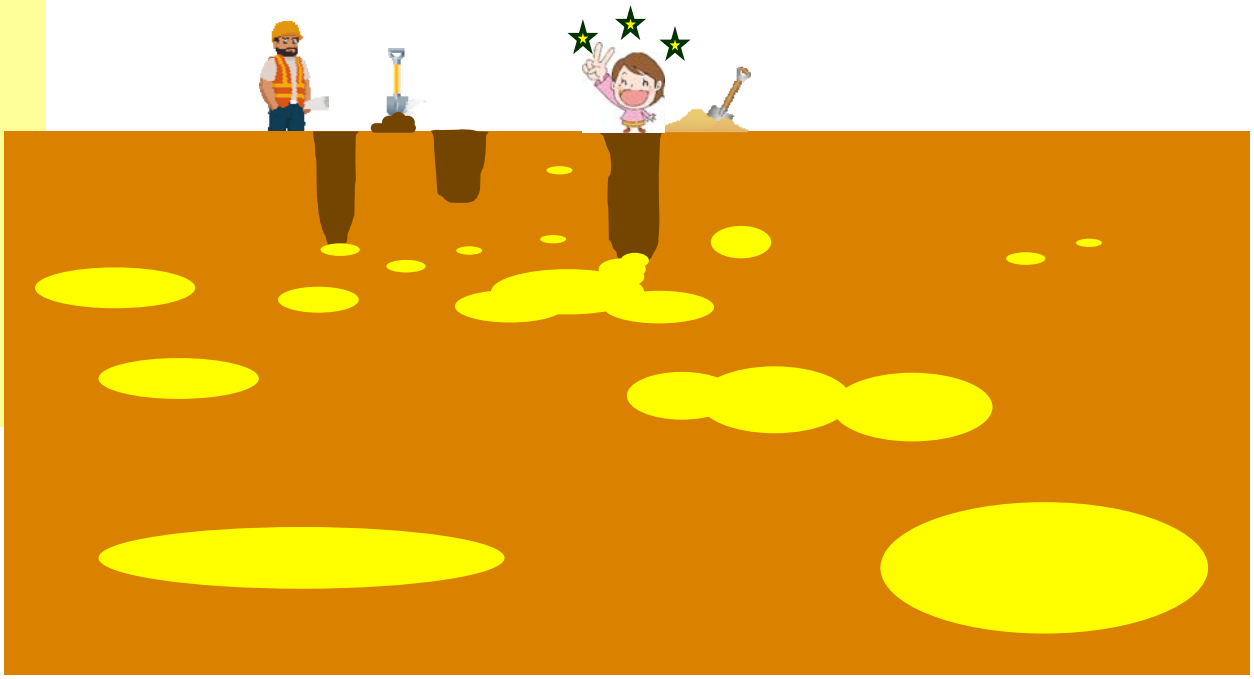
ソーシャルネットワーク分析

グラフ理論, 次数分布, 平均経路長, クラスタ係数, スケールフリー, 次数中心性, 近接性, 媒介性

大平雅雄, 開発データマイニングのススメ:「見える化」の次のステップ, SECセミナー 定量的なプロジェクト管理・プログラム管理のススメ～見える化による生産性・信頼性の向上～, 2015年10月30日
http://sec.ipa.go.jp/users/seminar/seminar_tokyo_20151030-02.pdf

リポジトリマイニングのすすめ

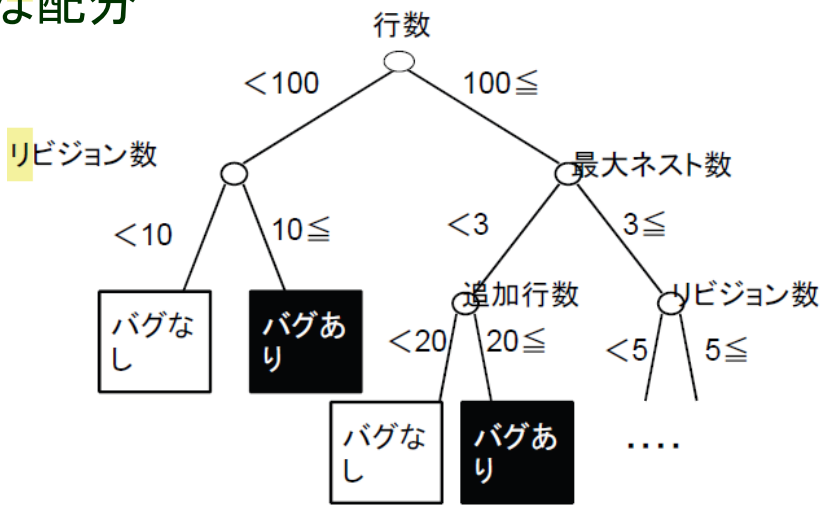
大きな鉱脈がたくさんある（研究テーマの宝庫）．
しかも，まだまだありそう．



リポジトリマイニングの研究テーマ

ソフトウェア品質：バグモジュール予測

- バグを含むモジュール（機能，ソースファイル）の予測[1]
- 80%のバグが20%のコードに含まれている[2]
- テスト工数の最適な配分



[1]畑秀明, 水野修, 菊野亨: 不具合予測に関するメトリクスについての研究論文の系統的レビュー, コンピュータソフトウェア, Vol.29, No.1 (2012), pp.106–117.
[2] Robert L. Glass, “Facts and Fallacies of Software Engineering,” Addison-Wesley, 2003.

ソフトウェア品質：バグモジュール予測

- 必要となるデータセットの例

リリース後に検出されたバグ

ファイル名	バグ数	TLOC	NBD	VG	NOM	・・・	ADD	DEL	PRE
MarkupToEclipseToc.java	0	143	2	18	13	・・・	0	0	0
BugzillaClientFactory.java	3	40	1	3	0	・・・	30	34	3
BugzillaEditingMonitor.java	0	46	2	6	2	・・・	11	12	3
ConfluenceLanguage.java	0	132	0	5	5	・・・	0	0	0
ViewSourceHandler.java	0	103	4	14	6	・・・	0	0	0
IRepositoryConstants.java	1	42	0	0	0	・・・	38	15	0
CommonColors.java	0	43	0	1	0	・・・	0	0	0
CollapseAllAction.java	1	44	1	3	2	・・・	7	9	2
・・・・	0	79	2	7	2	・・・	8	13	0

（リリース前に）予測したい値

ソースコードの複雑さ

変更行数

前バージョンのバグ数

ソフトウェア品質：バグモジュール予測

■必要となるデータセットの例

バグ票とバージョン管理システムより

ソースコードからメトリクス計測

バージョン管理システムから記録

ファイル名	バグ数	TLOC	NBD	VG	NOM	・・・	ADD	DEL	PRE
MarkupToEclipseToc.java	0	143	2	18	13	・・・	0	0	0
BugzillaClientFactory.java	3	40	1	3	0	・・・	30	34	3
BugzillaEditingMonitor.java	0	46	2	6	2	・・・	11	12	3
ConfluenceLanguage.java	0	132	0	5	5	・・・	0	0	0
ViewSourceHandler.java	0	103	4	14	6	・・・	0	0	0
IRepositoryConstants.java	1	42	0	0	0	・・・	38	15	0
CommonColors.java	0	43	0	1	0	・・・	0	0	0
CollapseAllAction.java	1	44	1	3	2	・・・	7	9	2
・・・・	0	79	2	7	2	・・・	8	13	0

（リリース前に）予測したい値

ソースコードの複雑さ

変更行数

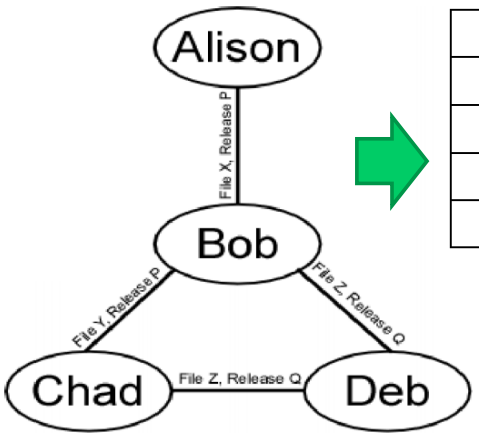
前バージョンのバグ数

ソフトウェア品質：バグモジュール予測

■MSRならではの発展

□開発者ネットワークメトリクスに基づくバグ予測[1]

◆SNAメトリクス（Closeness，Betweennessなど）の採用



Developer	Degree	Closeness	Betweenness
Alison	1	5/3 ≈ 1.67	3/6
Bob	3	1	5/6
Chad	2	4/3 ≈ 1.33	3/6
Deb	2	4/3 ≈ 1.33	3/6

File	Sum of Degree	Sum of Closeness	Sum of Betweenness
File X	4	8/3 ≈ 2.67	8/6 ≈ 1.33
File Y	6	4	11/6 ≈ 1.83
File Z	7	11/3 ≈ 3.67	11/6 ≈ 1.83

[1] A. Meneely, L. Williams, W. Snipes, J. Osborne, Predicting failures with developer networks and social network analysis, FSE2008.

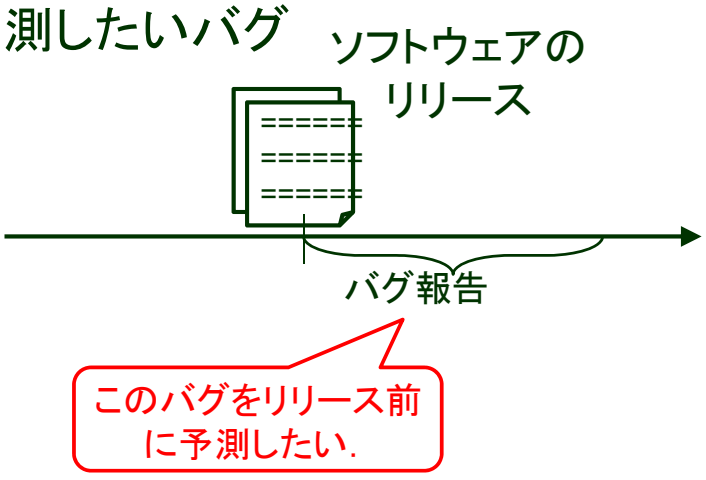
ソフトウェア品質：バグモジュール予測

- MSRならではの発展
  - リポジトリメトリクスに基づくバグ予測[1]
    - ファイル変更履歴の複雑さに基づくメトリクスの採用
  - プロジェクト間予測[2]，Transfer Defect Learning[3]
  - Just-in-timeバグ予測[4]
  - 個人向けバグ予測[5]

[1] A.E. Hassan, Predicting faults using the complexity of code changes, ICSE2009.
[2] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, B. Murphy, Cross-project defect prediction – a large scale experiment on data vs domain vs process, FSE2009.
[3] J Nam, SJ Pan, S Kim, Transfer defect learning, ICSE2013.
[4] Y. Kamei et al., A Large-Scale Empirical Study of Just-In-Time Quality Assurance, IEEE Trans. Soft. Eng. Vol.39, No.6, pp.757-773, June, 2013.
[5] T Jiang, L Tan, S Kim, Personalized defect prediction, ASE2013.

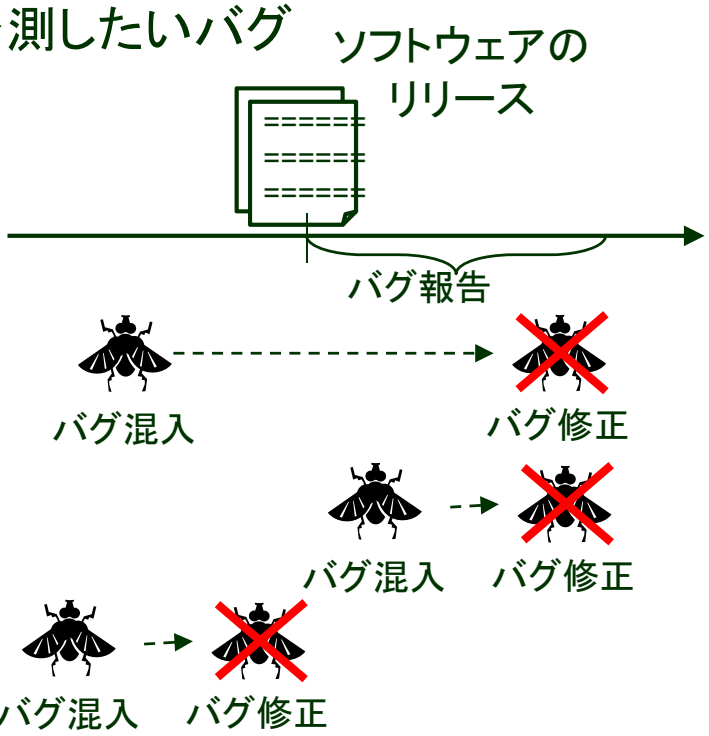
ソフトウェア品質：バグモジュール予測

- 予測したいバグ



ソフトウェア品質：バグモジュール予測

- 予測したいバグ



バグ混入時期，修正時期を推定する必要がある.

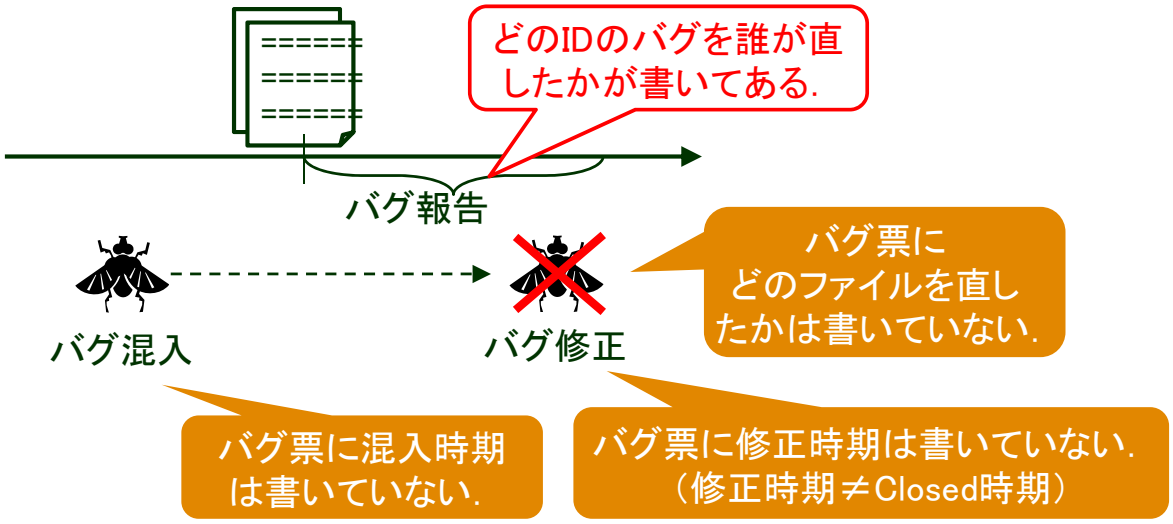
ソフトウェア品質：バグモジュール予測

- SZZアルゴリズム[1]
  - いつ，だれが，どこにバグを混入したかを推定する.
  - 多くの研究で用いられている.（被引用回数＝494）

[1] J. Sliwerski, T. Zimmermann, A. Zeller, When do changes induce fixes?, MSR2005.

ソフトウェア品質：バグモジュール予測

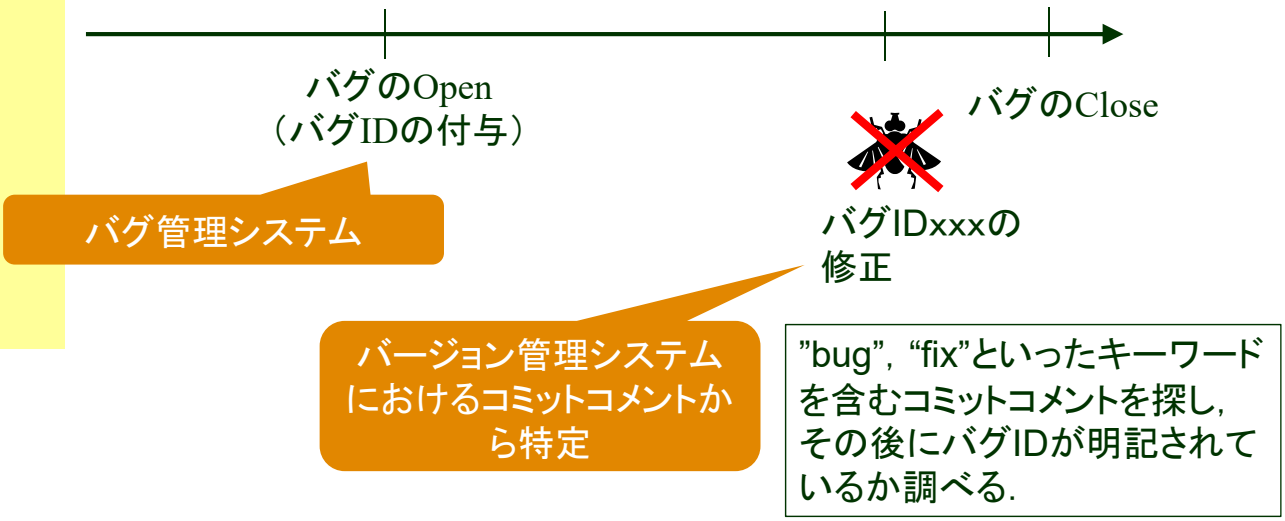
- SZZアルゴリズム[1]
  - いつ，だれが，どこにバグを混入したかを推定する．



[1] J. Sliwerski, T. Zimmermann, A. Zeller, When do changes induce fixes?, MSR2005.

ソフトウェア品質：バグモジュール予測

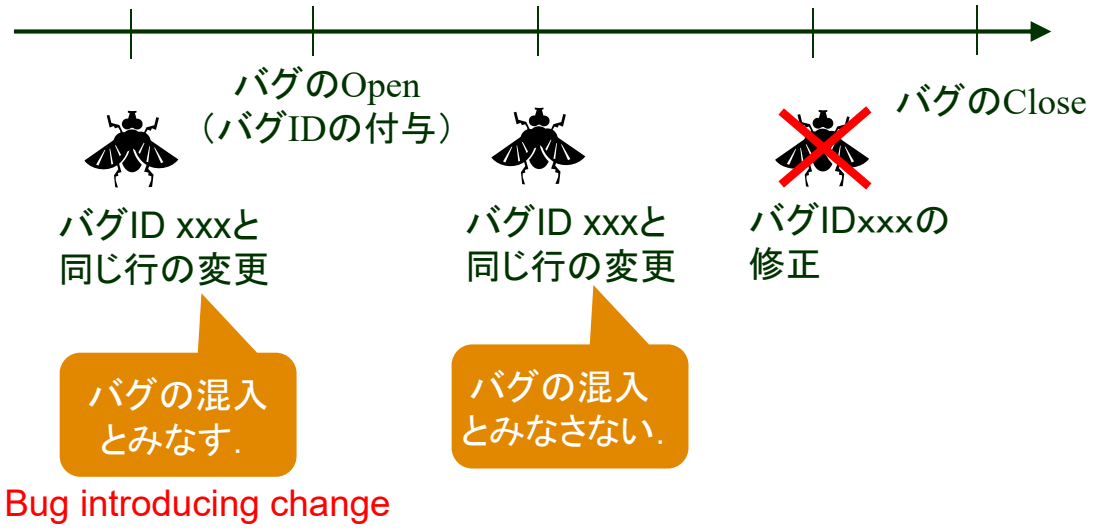
- SZZアルゴリズム[1]
  - いつ，だれが，どこにバグを混入したかを推定する．



[1] J. Sliwerski, T. Zimmermann, A. Zeller, When do changes induce fixes?, MSR2005.

ソフトウェア品質：バグモジュール予測

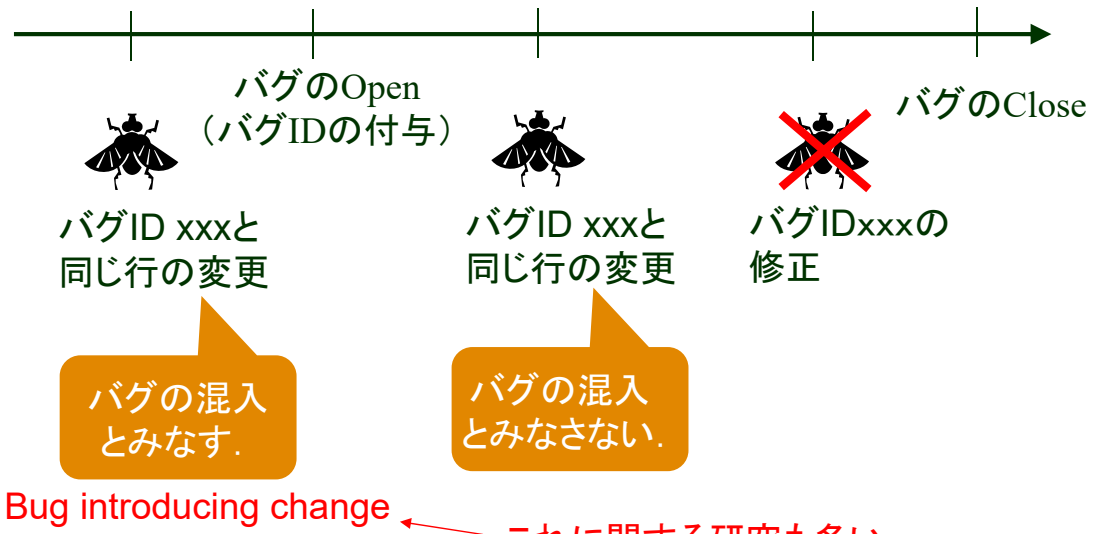
- SZZアルゴリズム[1]
  - いつ，だれが，どこにバグを混入したかを推定する.



詳しくは，http://pages.cpsc.ucalgary.ca/~zimmerth/talks/ase2006.pdf

ソフトウェア品質：バグモジュール予測

- SZZアルゴリズム[1]
  - いつ，だれが，どこにバグを混入したかを推定する.



詳しくは，http://pages.cpsc.ucalgary.ca/~zimmerth/talks/ase2006.pdf

ソフトウェア品質：バグモジュール予測

- SZZアルゴリズムの発展
  - ReLink[1]
  - Multi-layered approach[2]
  - Trustrace[3]
- Missing Linkの分析[4]

[1] R Wu, H Zhang, S Kim, SC Cheung, Relink: recovering links between bugs and changes, FSE2011.
[2] AT Nguyen, TT Nguyen, HA Nguyen, TN Nguyen,
Multi-layered approach for recovering links between bug reports and fixes, FSE2012.
[3] N Ali, YG Guéhéneuc, G Antoniol, Trustrace: Mining Software Repositories to Improve the Accuracy of Requirement Traceability Links, IEEE Trans. Soft. Eng., Vol.39, No.5, May 2013.
[4] A Bachmann, C Bird, F Rahman, P Devanbu, A Bernstein, The missing links: bugs and bug-fix commits, FSE2010.

ソフトウェア品質：バグモジュール予測

- 必要となるデータセットの例

バグ票とバージョン管理システムより

ソースコードからメトリクス計測

バージョン管理システムから記録

ファイル名	バグ数	TLOC	NBD	VG	NOM	・・・	ADD	DEL	PRE
MarkupToEclipseToc.java	0	143	2	18	13	・・・	0	0	0
BugzillaClientFactory.java	3	40	1	3	0	・・・	30	34	3
BugzillaEditingMonitor.java	0	46	2	6	2	・・・	11	12	3
ConfluenceLanguage.java	0	132	0	5	5	・・・	0	0	0
ViewSourceHandler.java	0	103	4	14	6	・・・	0	0	0
IRepositoryConstants.java	1	42	0	0	0	・・・	38	15	0
CommonColors.java	0	43	0	1	0	・・・	0	0	0
CollapseAllAction.java	1	44	1	3	2	・・・	7	9	2
・・・・	0	79	2	7	2	・・・	8	13	0

（リリース前に）予測したい値

ソースコードの複雑さ

変更行数

前バージョンのバグ数

ソフトウェア品質：バグモジュール予測

- ソースコードメトリクス
  - メトリクス計測ツールを使う.
  - 例：Understand, QAC, ・・・

- 公開されているデータセットもある.
  - Eclipse Bug Data! Release 2.0a, 2007-12-28
  - https://www.st.cs.uni-saarland.de/softevo/bug-data/eclipse/

ソフトウェア品質：Bug Localization

- バグ票 → （ソースコード上の）バグの位置の推定[1]

バグの症状

Bug ID: 80720
Summary: **Pinned console** does not remain on top
Description:
*Open two **console views**, ... **Pin** one **console.** Launch another program that produces output. Both **consoles display** the last launch. The **pinned console** should remain **pinned.***

原因となるソースコード

Source code file: **ConsoleView.java**

```
public class ConsoleView extends PageBookView
    implements IConsoleView, IConsoleListener {...
      public void display(IConsole console) {
          if (fPinned && fActiveConsole != null) { return;}
      } …
      public void pin(IConsole console) {
            if (console == null) {  setPinned(false);
            } else {
               if (isPinned()) { setPinned(false); }
              display(console);
              setPinned(true);
            }
      }
}
```

  - 各ドキュメントをVector Space Model(VSM)で表現する.
  - 類似のソースコードを検索する.
  - 過去の類似のバグも参考にする.

[1] J. Zhou, H. Zhang, and D. Lo. Where should the bugs be fixed? ICSE2012.

ソフトウェア品質：Bug Localization

- 発展
  - Two-phase予測[1]
  - ソースコードの構文情報を利用した予測[2]
  - ファイルの同時変更情報を利用した予測[3]

[1] D Kim, Y Tao, S Kim, A Zeller, Where should we fix this bug? a two-phase recommendation model, IEEE Trans. Soft. Eng. Vol.39, No.11, 2013.
[2] RK Saha, M Lease, S Khurshid, DE Perry, Improving bug localization using structured information retrieval, ASE2013
[3] C. Tantithamthavorn, A. Ihara, K. Matsumoto, Using co-change histories to improve bug localization performance, SNPD2013.

ソフトウェア品質：バグの要因分析

- 変更とバグの分析
  - 変更の多いファイルはバグが多い．（多数の論文あり）
- 使用ライブラリに着目した分析[1]
  - Eclipse.jdt.internal.compilerパッケージをimportしている71%のファイルは後にバグ修正されている．
  - uiパッケージの場合は14%
- 開発者とバグ混入の関係の分析[2]
  - Eclipse3.0, 3.1, 3.2を分析
  - どのバージョンでもバグ混入率が高い／低い開発者が存在する．
  - 複数の開発者が変更したファイルはバグ混入率が高い．

[1] A. Schroter , Predicting Defects and Changes with Import Relations, MSR2007.
[2] S. Matsumoto, Y. Kamei, A. Monden, K. Matsumoto, M. Nakamura, An analysis of developer metrics for fault prediction, PROMISE'10.

ソフトウェア品質：再修正バグの分析

- 再修正が必要となるバグ（デグレード，エンバグ）の要因分析・予測[1][2]
  - Eclipseでは約16％のバグがreopenされた．（Apache HTTPは6.5%，OpenOfficeは26.3%）
    - バグレポートの内容の影響が最も大きい[2].
      - 再修正：“control”，“background”,“debugging”，“breakpoint”，“blocked” and “platforms”
      - 問題なし：“verified”，“duplicate”,“screenshot”，“important”，“testing” and “warning”

[1] T Zimmermann, N Nagappan, PJ Guo, B. Murphy, Characterizing and predicting which bugs get reopened, ICSE2012.
[2] Shihab, E., Ihara, A., Kamei, Y., Ibrahim, W.M., Ohira, M., Adams, B., Hassan, A. E. and Matsumoto, K.: Studying Re-Opened Bugs in Open Source Software, Empirical Software Engineering,18(5),2013.

ソフトウェア品質：バグ修正時間の予測

- バグ修正にかかる時間の予測[1]
  - 入力：バグレポート　→過去の類似バグレポート検索
  - 出力：修正時間
- 発展
  - 修正されるバグの予測[2]
  - バグ修正priorityの予測[3]，bug triage[4]
  - Blockingバグの予測[5]

[1] C Weiss, R Premraj, T Zimmermann, A. Zeller, How long will it take to fix this bug? MSR2007.
[2] PJ Guo, T Zimmermann, N Nagappan, B. Murphy, Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows, ICSE2010.
[3] Y Tian, D Lo, C Sun, Drone: Predicting priority of reported bugs by multi-factor analysis, ICSM2013.
[4] D Čubranić, Automatic bug triage using text categorization, SEKE2004.
[5] H Valdivia Garcia, E Shihab, Characterizing and predicting blocking bugs in open source projects, MSR2014.

ソフトウェア品質：バグレポート分析

- 良いバグレポートの書き方[1][2]
- バグ分類間違いの分析[3]
- 重複するバグレポートの検出[4]
- バグレポートの自動分類（セキュリティバグ）[5]
- バグ修正者の自動割り当て[6]
- バグ修正者の再割り当ての分析[7]

[1] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, T. Zimmermann, What makes a good bug report? FSE2008.
[2] P Hooimeijer, W Weimer, Modeling bug report quality, ASE2007.
[3] K Herzig, S Just, A Zeller, It's not a bug, it's a feature: how misclassification impacts bug prediction, ICSE2013.
[4] AT Nguyen, TT Nguyen, TN Nguyen, D. Lo, C. Sun, Duplicate bug report detection with a combination of information retrieval and topic modeling, ASE2012.
[5] M Gegick, P Rotella, T Xie, Identifying security bug reports via text mining: An industrial case study, MSR2010.
[6] J Anvik, Automating bug report assignment, ICSE2006.
[7] PJ Guo, T Zimmermann, N Nagappan, B. Muphy, Not my bug! and other reasons for software bug report reassignments, CSCW2011.

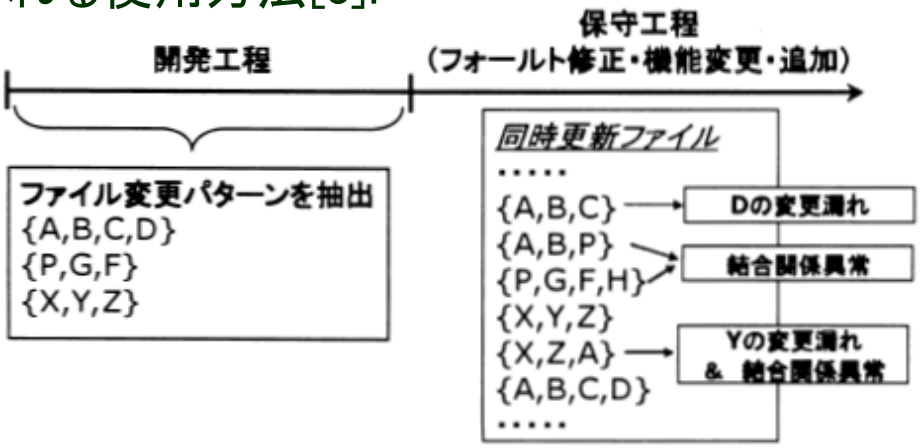
開発プロセス：変更の分析・予測

- 大きな変更はなぜ起こるか？[1]
- 小さな変更の分析[2]
- どのような変更でバグが入るか？[3]
- 典型的なコミットは何か？[4]
- 変更要求（change request）の自動分類[5]
- 変更の予測[6]

[1] A. Hindle, D.M. German, R. Holt, What do large commits tell us?: a taxonomical study of large commits, MSR2008.
[2] R. Purushothaman, D.E. Perry, Toward understanding the rhetoric of small source code changes, IEEE Trans. Soft. Eng., Vol. 31, No. 6, 2005.
[3] J Śliwerski, T Zimmermann, A Zeller, When do changes induce fixes? MSR2005.
[4] A Alali, H Kagdi, JI Maletic, What's a typical commit? A characterization of open source software repositories, ICPC2008.
[5] G Antoniol, K Ayari, M Di Penta, F Khomh, Y-G Guéhéneuc, Is it a bug or an enhancement?: a text-based approach to classify change requests, CASCON '08.
[6] M. Askari, R. Holt, Information theoretic evaluation of change prediction models for large-scale software, MSR2006.

開発プロセス：Change coupling分析

- 同時に変更されることの多いソースファイル群を特定し，開発時の同時変更のし忘れを警告する[1][2].
- 想定される使用方法[3]:



[1] M. Fischer, M. Pinzger, H. Gall, Populating a release history database from version control and bug tracking systems, ICSM2003.
[2] Zimmermann, T, Weisgerber, P, Diehl, S, Zeller, A., Mining version histories to guide software changes, ICSE2004. (被引用数1051)
[3] 松村,横森,大杉,川口,松下, ファイルの同時変更パターンと変更差分の分析による論理的結合関係の自動抽出, ソフトウェアシンポジウム2005, 104-112, June 2005.

開発プロセス：Code Review

- 開発者へのインタビュー，レビューコメント分析[1]
- レビュー担当者の自動推薦[2]
- レビューにより修正されたコードの分析[3]
- プルリクエストのレビュー担当者推薦[4]
- 受理される／されないパッチの分析[5]

[1] A Bacchelli, C Bird, Expectations, outcomes, and challenges of modern code review, A Bacchelli, C Bird, ICSE2013.
[2] P Thongtanunam, C Tantithamthavorn, R.G. Kula, N. Yoshida, H. Iida, K. Matsumoto, Who should review my code? A file location-based code-reviewer recommendation approach for modern code review, SANER2015.
[3] M Beller, A Bacchelli, A Zaidman, E. Juergens, Modern code reviews in open-source projects: which problems do they fix? MSR2014.
[4] Y Yu, H Wang, G Yin, CX Ling, Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration, APSEC2014.
[5] Y. Jiang, B. Adams, D.M. German, D. M., Will My Patch Make It? And How Fast? Case Study on the Linux Kernel, MSR2013.

開発プロセス：Branch, Merge, Conflict

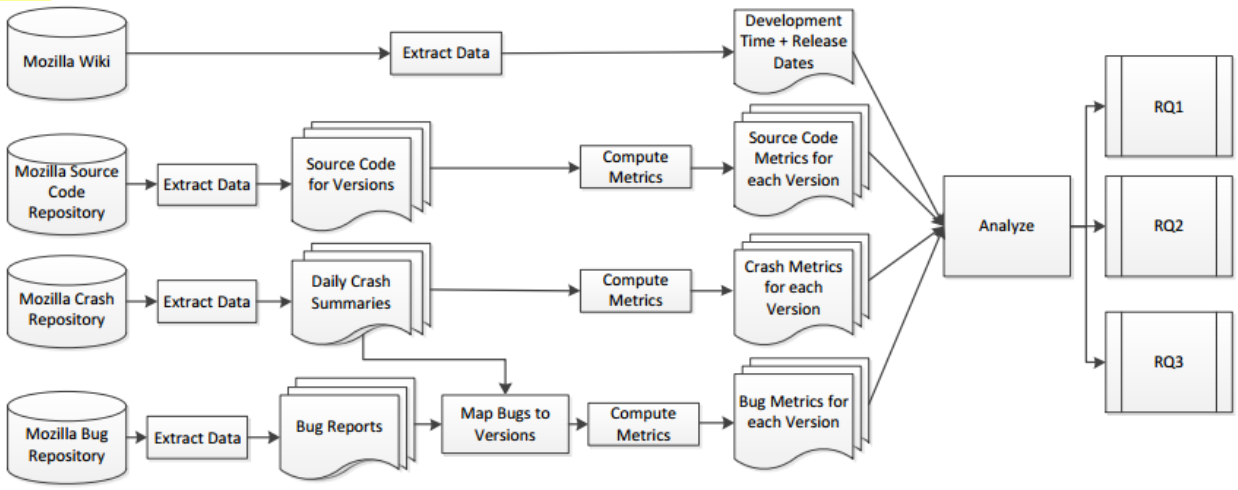
- ブランチとマージの実態調査[1]
- コンフリクト発生時の修正すべきメソッドの優先順位付け[2]
- コンフリクト解消方法の分析（右表）[3]

project	# of conflict methods	Causes			Resolutions	
		CHANGE SAME POSITION	DELET-ION	RENAM-ING	1-WAY	OTHER
james	115	114	1	0	115	0
jrobin	593	168	363	62	592	1
maven.plugins	34	31	3	0	34	0
org.eclipse.ajdt	2	2	0	0	1	1
org.eclipse. gmp.graphiti	4	2	1	1	3	1
org.eclipse. jubula.core	3	3	0	0	3	0
org.eclipse. paho.mqtt.java	6	0	4	2	6	0
org.eclipse. scout.sdk	7	4	3	0	6	1
org.eclipse. stardust.ui.web	9	9	0	0	9	0
org.eclipse.uml2	6	6	0	0	2	4
total	779	339	375	65	771	8

[1] S Phillips, J Sillito, R Walker, Branching and merging: an investigation into current version control practices, CHASE2011.
[2] N Niu, F Yang, JRC Cheng, S. Reddivari, A cost-benefit approach to recommending conflict resolution for parallel software development, RSSE2012.
[3] R Yuzuki, H Hata, K Matsumoto, How we resolve conflict: an empirical study of method-level conflict resolution, SWAN2015.

開発プロセス：リリース分析

- リリースの間隔に関する分析[1]
  - Firefoxの場合，早いリリースだからといってリリース後のバグが増えるわけではないが，クラッシュの間隔は短くなる．



[1] F. Khomh, T. Dhaliwal, Y. Zou, B. Adams, Do faster releases improve software quality? – an empirical case study of Mozilla Firefox- , MSR2012.

プログラミング：コーディングパターン検出

- ライブラリのAPIの呼び出しパターンの検出[1]
- メソッド呼び出しパターンの検出[2]

例：jEditでは，

- openNodeScope/jjtreeOpenNodeScope/closeNodeScope/jjtreeCloseNodeScope

というパターンが55回出現している．

- 要素技術：シーケンシャルパターンマイニング
  - 要素列から頻出する部分列をパターンとして抽出する．
  - 部分列の個々の出現は，順序が同一でなければならないが，不連続なものでよい．

[1] T. Xie and J. Pei: MAPO: Mining API Usages from Open Source Repositories, MSR2006, pp.54–57, 2006.
[2] 石尾,伊達,三宅,井上,"シーケンシャルパターンマイニングを用いたコーディングパターン抽出," 情報処理学会論文誌, Vol.50, No.2, pp.860-871, 2009.

プログラミング：ソースコードコメントの分析

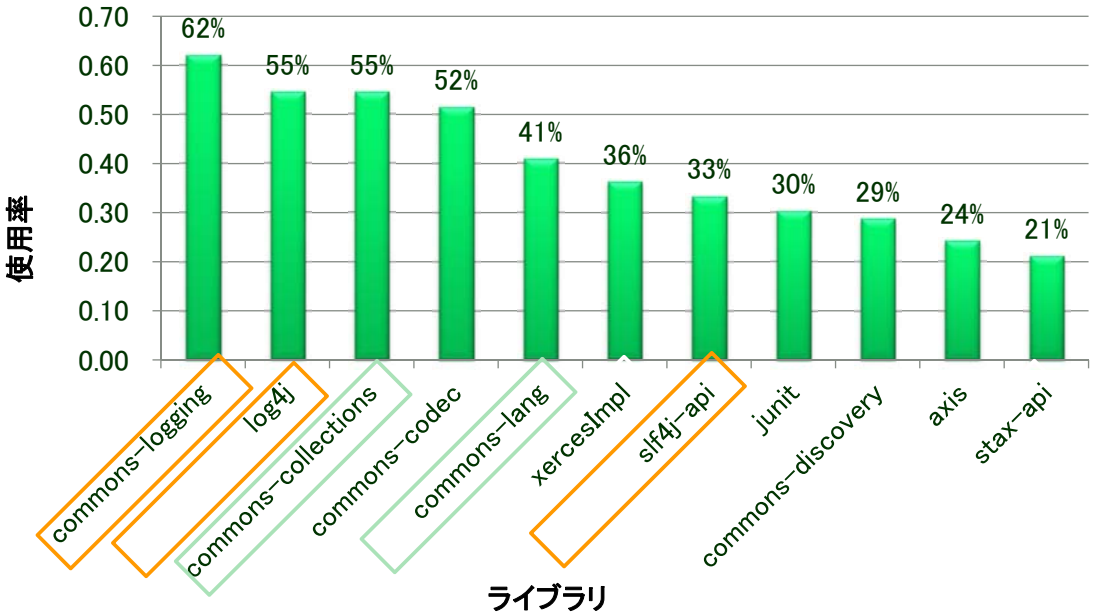
- コメントの分類[1]
  - 6プロジェクト，2100コメントを7つの観点（What，Who，Where，When，Time，Evolution）から分類した．
- "TODO"コメントの分析[2]
- コメントとバグの関係の分析[3]

- 関連研究： 変数名の付け方の分析・推薦[4]

[1] Y. Padioleau, L. Tau, Y. Zhou, Listening to programmer – taxonomies and characteristics of comments in operating system code, ICSE2009.
[2] M.A. Storey, J. Ryall, R.I. Bull, D. Myers, J. Siner, TODO or to bug: exploring how task annotations play a role in the work practices of software developers, ICSE2008.
[3] H. Aman, S. Amasaki, T. Sasaki, M. Kawahara, Lines of Comments as a Noteworthy Metric for Analyzing Fault-Proneness in Methods, IEICE Trans. Inf. & Syst., E98-D(12), 2015.
[4] Y. Kashiwabara, T. Ishio, H. Hata, K. Inoue, Method Verb Recommendation Using Association Rule Mining in a Set of Existing Projects, IEICE Trans. Inf. & Syst., E98-D(3), 2015.

プログラミング：ライブラリ使用状況

■“Enterprise”ドメインの場合（Java）



ロギングライブラリ，基本ライブラリ，XMLパーサの利用が多い．

[1] 砂田隆浩，門田暁人，松本健一，"Javaソフトウエア開発における使用ライブラリの分析，" ソフトウェア信頼性研究会 第8回ワークショップ，Nov. 2012.

プログラミング：ライブラリ使用状況

■各ドメインでどのようなライブラリが使用されているか？

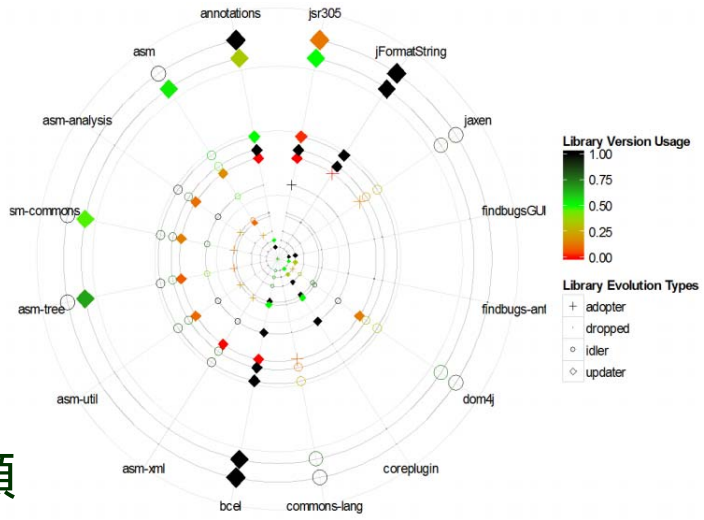
ドメイン	使用頻度の高いライブラリ	
	特有ライブラリ	共通ライブラリ
Testing	asm	commons-logging（7） Log4j（6） commons-codec（5） commons-collections（4） commons-lang（4） junit（3） xercesImpl（3）
Bio-Informatics	Jama/mysql-connector-java	
HTTP/WWW		
Chat	Mina-core	
Enterprise	mail/commons-httpclient	
Networking	Commons-net	
Sound/Audio	tritonus_share/mp3api	

プログラミング：ライブラリ使用状況

- バージョンごとのライブラリ使用頻度の分析[1]

Library name and version	Times used
junit 3.8.1	60
junit 3.8.2	9
junit 4.4	7
log4j 1.2.8	10
log4j 1.2.14	9
log4j 1.2.15	0
servlet-api 2.3	4
servlet-api 2.5	1
derby 10.1	6
derby 10.2	1

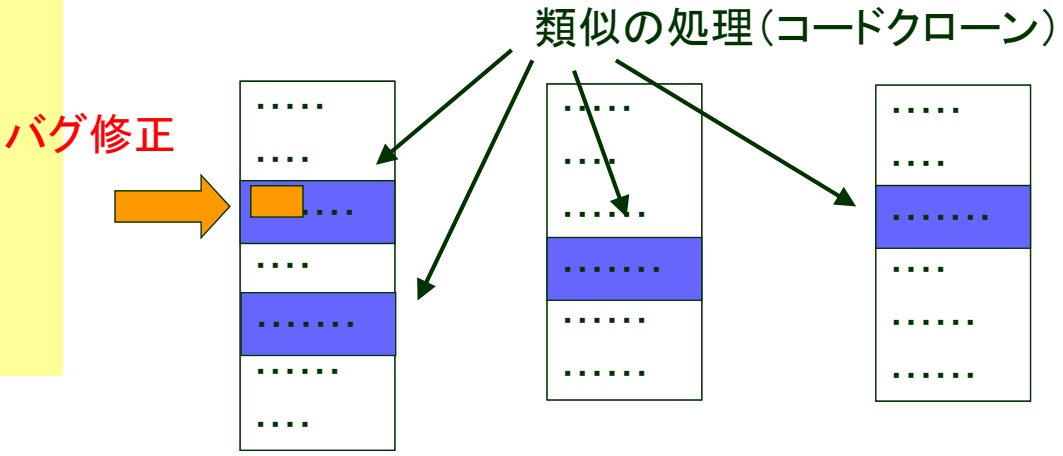
- 異なるバージョンのライブラリ間の依存と使用頻度の可視化[2]



[1] YM Mileva, V Dallmeier, M Burger, A Zeller, Mining Trends of Library Usage, IWPSE2009.
[2] R.G. Kula and C. De Roover, D.M. German, T. Ishio, K. Inoue, Visualizing the Evolution of Systems and Their Library Dependencies, VISSOFT 2014.

プログラミング：コードクローン

- Patchを当てるべきコードの発見 [1]
- プロジェクト間コードクローンのベンチマーキング[2][3]



[1] Jang, J., Agrawal, A., and Brumley, D.: ReDe-Bug: Finding Unpatched Code Clones in Entire OS Distributions, Proc. 33rd Symposium on Security and Privacy (2012), pp.48-62.
[2] J Svajlenko, JF Islam, I Keivanloo, CK Roy, MM Mia, Towards a Big Data Curated Benchmark of Inter-project Code Clones, ICSME2014.
[3] https://github.com/clonebench/BigCloneBench

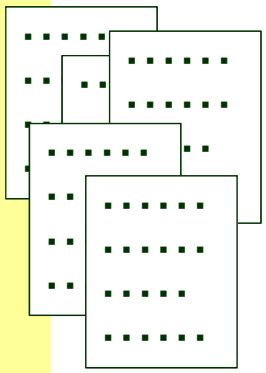
プログラミング：ソフトウェア部品検索

- キーワードベースのソフトウェア部品検索エンジン
  - SPARS[1], Koders, Jarhoo
- 入出力ベースの検索エンジン Prospector[2]
- コードクローンベースの検索エンジン[3]
- ランキング手法[4]
- バイナリプログラムを入力としたソースコード検索[5]

[1] SPARS Project, http://sel.ist.osaka-u.ac.jp/SPARS/
[2] D. Mandelin, L. Xu, R. Bodík, D. Kimelman, “Jungloid mining: helping to navigate the API jungle,” PLDI2005, pp.48-61, 2005.
[3] P. Xia, Y. Manabe, N. Yoshida, K. Inoue: Development of a code clone search tool for open source repositories, コンピュータソフトウェア, Vol.29, No.3, pp.181-187, 2012.
[4] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, S. Kusumoto: Ranking Significance of Software Components Based on Use Relations, IEEE Transactions on Software Engineering, Vol.31, No.3, pp.213–225, 2005.
[5] Wei Ming Khoo, Alan Mycroft, and Ross Anderson, Rendezvous: A Search Engine for Binary Code, MSR2013.

プログラミング：トピック分析

- ソフトウェアからのトピック抽出と自動分類[1]
- ソースファイルからのトピック抽出[2]



ドキュメント
（ソースファイル，バグ票，．．．）

特徴量
の抽出

- Bag of Words（単語の集合）
- N-Gram
- トピックモデリング（LDA等）

分析・
予測

[1] K Tian, M Revelle, D Poshyvanyk, sing latent dirichlet allocation for automatic categorization of software, MSR2009.
[2] A Kuhn, S Ducasse, T Gírba, Semantic clustering: Identifying topics in source code, Information and Software Technology, 2007.

プログラミング：学術論文の引用

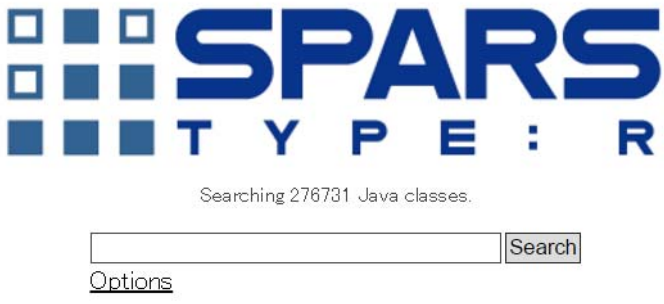
■プログラマはどのような論文を参照しているのか，Ohloh上の7万件のJavaプロジェクトを調査した[1].

順位	引用されたプロジェクト数	論文名（発表年）	論文概要
1	30	Computing Standard Deviations: Accuracy	標準偏差を計算する4アルゴリズムの比較
2	21	Random Number Generators: Good Ones Are Hard to Find	疑似乱数生成アルゴリズム
3	18	A Fast Parallel Algorithm for Thinning Digital Patterns	デジタルパターンの細線化アルゴリズム
4	17	Generating Gamma Variates by a Modified Rejection Technique	ガンマ変量生成アルゴリズム
5	15	A Fast Algorithm for Finding Dominators in a Flowgraph	制御フローグラフにおける支配ノード発見アルゴリズム
6	13	Literate Programming	プログラミングスタイル

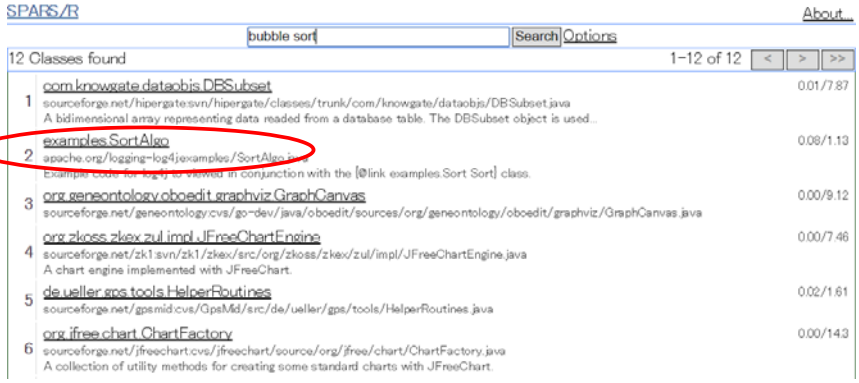
[1] 小西, 門田, 畑, 松本, オープンソースソフトウェアにおける学術論文の引用状況の分析, ソフトウェア工学の基礎XX, FOSE2013.

プログラミング：ソフトウェア部品検索

■SPARS-J　http://demo.spars.info/r/

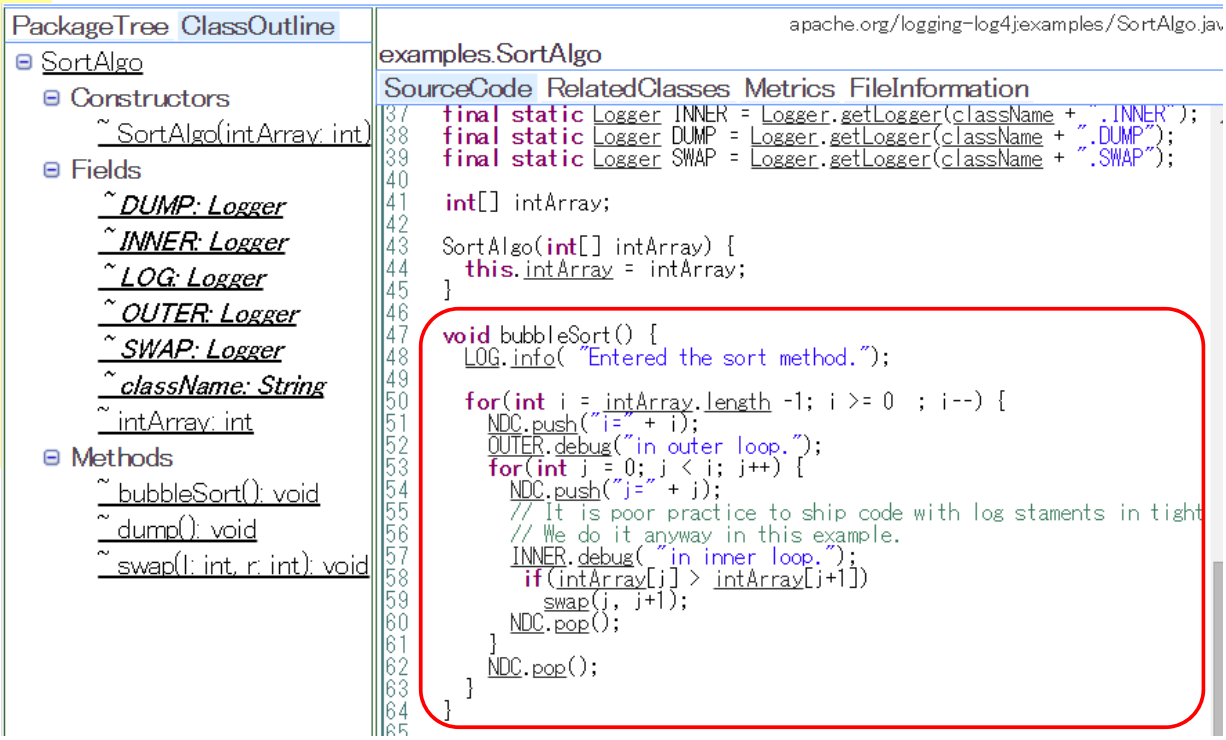


“bubble sort”で検索



プログラミング：ソフトウェア部品検索

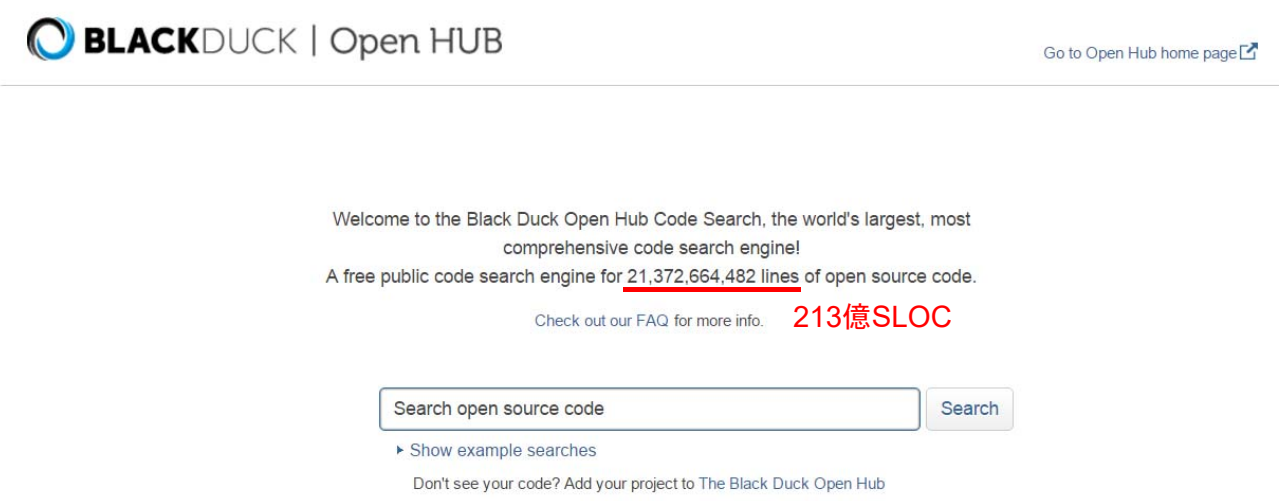
■SPARS-J　http://demo.spars.info/r/



プログラミング：ソフトウェア部品検索

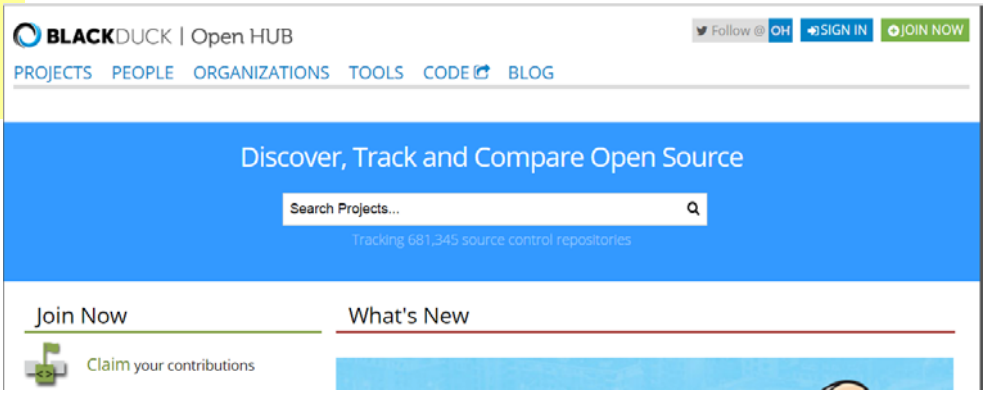
■Open HUB Code Search（旧Ohloh）

- OSSソースコード全文検索（60万以上のプロジェクト）



Open HUB（旧：Ohloh）

- OSS開発者コミュニティサイト
  - OSSプロジェクト検索
  - 類似プロジェクト検索
  - プロジェクト活動状況
    - Commit量の推移
    - Contributor，メンバー数の推移など



Open HUB（旧：Ohloh）

Popular Projects

- Mozilla Firefox 13202 users
- Apache HTTP Server 9420 users
- MySQL 9133 users
- Apache Subversion 8674 users
- PHP 7746 users
- Linux Kernel 7087 users
- Bash 6271 users
- Firebug 5954 users

Active Projects

- Chromium (Google Chrome) 4553 commits
- OpenStack 2752 commits
- Liferay Portal 4345 commits
- Mozilla Core 0 commits
- Mozilla Firefox 4558 commits
- azure-content 4094 commits
- Arch Linux Packages 4892 commits
- FreeBSD Ports 2974 commits

Active Contributors

- Felix Yan 1730 commits
- Translation updater bot 1638 commits
- luca suriano 1013 commits
- Manfred Touron 720 commits
- Petr Hejl 594 commits
- Stephan Bergmann 554 commits
- Juan RP 532 commits
- y4m4 514 commits

Open HUB（旧：Ohloh）

MySQL

Settings | Report Duplicate

Very High Activity

9,133

I Use This!

Analyzed 1 day ago. based on code collected 1 day ago.

Project Summary

MySQL, the most popular Open Source SQL database management system, is developed, distributed, and supported by Oracle Corporation.

Tags

solaris net php python udf dbms xml mysql relational java oltp lamp gis replication web software_development posix sql embedded engine server linux transactions javascript clustering jdbc dbi database odbc windows rdbms database_server acid unicode c perl macosx

Share

いいね！ 2 Tweet G+1 2 Share 7

Quick Reference

Project Links: Homepage, Community, Documentation, Download, Forums, Issue Trackers, Mailing Lists

Code Locations: git://github.com/mysql/mysql-server

Licenses: GPL-2.0+

Similar Projects: HyperSQL Dat..., Ingres Database, PostgreSQL D..., Firebird

Managers: yinsigan

Browse Code

Open HUB（旧：Ohloh）

In a Nutshell, MySQL...

... has had 119,390 commits made by 1,331 contributors representing 2,591,481 lines of code

... is mostly written in C++ with an average number of source code comments

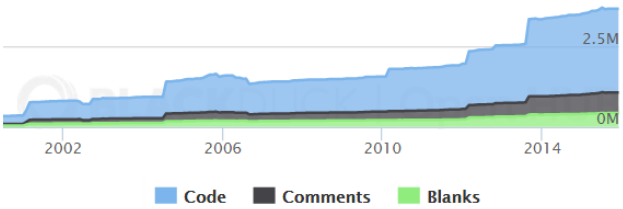
... has a well established, mature codebase maintained by a very large development team with decreasing Y-O-Y commits

... took an estimated 761 years of effort (COCOMO model) starting with its first commit in July, 2000 ending with its most recent commit about 2 months ago

Languages

C++	55%	C	19%
JavaScript	17%	22 Other	9%

Lines of Code



Activity

30 Day Summary

Nov 7 2015 — Dec 7 2015

0 Commits

0 Contributors

12 Month Summary

Dec 7 2014 — Dec 7 2015

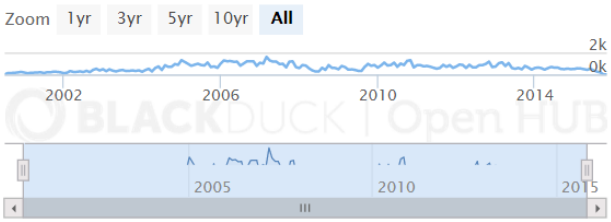
4288 Commits

Down -2578 (37%) from previous 12 months

177 Contributors

Down -1 (0%) from previous 12 months

Commits per Month



人的要因：エキスパート推薦・分析

- バグ修正依頼　→　バグ修正者の推薦[1]
- Failテストケース　→　バグ修正者の推薦[2]
- 変更要求　→　開発者の推薦[3]
- パッチ投稿　→　レビュー担当者の推薦[4]
- コミッターの分析 [5]

[1] J. Anvik, L. Hiew, G. C. Murphy, Who should fix this bug? ICSE'06, pp.361-370, 2006.
[2] F. Servant, J.A. Jones: WhoseFault: Automatic Developer-to-Fault Assignment Through Fault Localization, ICSE2012, pp.36-46, 2012.
[3] H. Kagdi, D. Poshyvanyk: Who can help me with this change request?, ICPC 2009, pp.273–277, 2009.
[4] J.B. Lee, A. Ihara, A. Monden, K. Matsumoto: Patch Reviewer Recommendation in OSS Projects. APSEC2013.
[5] A. Jongyindee, M. Ohira, A. Ihara, K. Matsumoto: Good or Bad Committers? -- A Case Study of Committer's Activities on the Eclipse's Bug Fixing Process. IEICE Trans. 95-D(9), 2012.

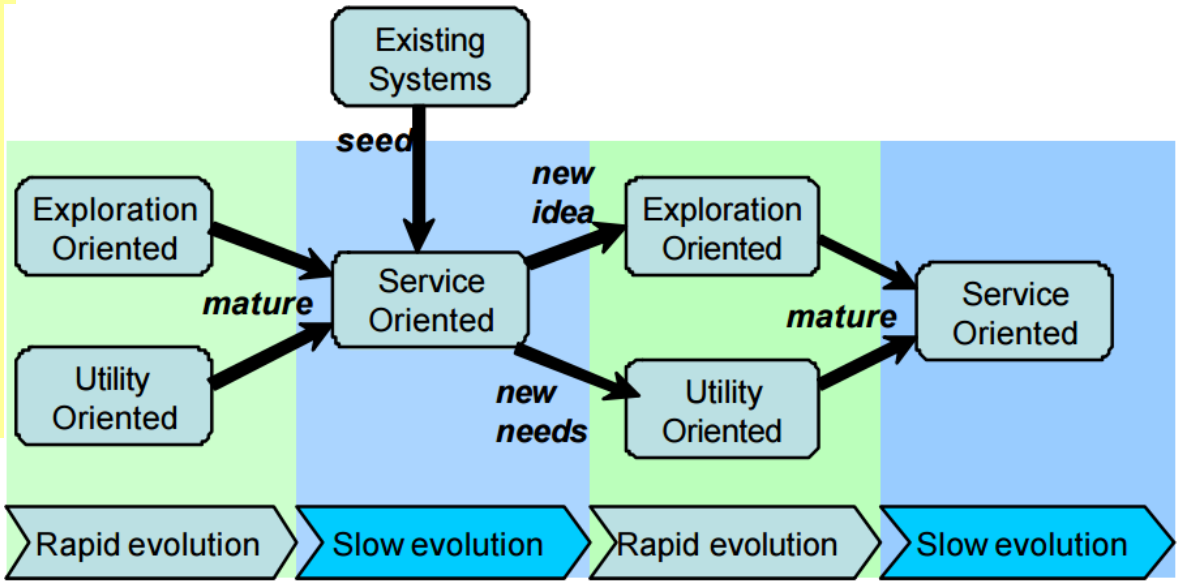
人的要因：コミュニティ分析

- 事例
  - 一世を風靡したNetscapeは2008年3月に全てのサポートを終了した.
    - 終わりそうなプロジェクトには兆候がある.
  - GIMPプロジェクトでは，プロジェクトリーダーがプロジェクトから離脱したため，正式版がリリースされることなく開発が約20か月停滞した[1].
    - その後，別の開発者がプロジェクトリーダーの役割を引き受けた.

[1]Y. Ye, K. Kishida, Toward an understanding of the motivation of open source software developers, ICSE2003.

人的要因：コミュニティ分析

■OSSプロジェクトの進化パターンの分析[1]



[1] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, Y. Ye, Evolution patterns of open-source software systems and communities, IWPSE2002.

人的要因：コミュニティ分析

■OSSプロジェクトの成功の定義 [1]

Level 1	Level 2
User	Satisfaction
	Involvement
Product	Meets requirements
	Code quality
	Portability
	Availability
Process	Activity
	Adherence to process
	Bug Fixing
	Time
	Age
Developers	Involvement
	Varied developers
	Satisfaction
	Enjoyment
Use	Competition
	Number of users
	Downloads
Recognition	Referral
	Attention and recognition
	Spin offs
Influence	

[1] K Crowston, H Annabi, J Howison, Defining open source software project success, ICIS2003.

人的要因：コミュニティ分析

- GitHub上の100,000プロジェクト，30,000開発者についてのネットワーク分析[1]
  - 共通の開発者のいるプロジェクト＝繋がっている
  - 共通のプロジェクトに参加する開発者＝繋がっている

Project url	PageRank
https://github.com/mxcl/homebrew	0.0009862
https://github.com/rails/rails	0.0006378
https://github.com/lifo/docrails	0.0006370
https://github.com/joyent/node	0.0002161
https://github.com/rubinius/rubinius	0.0001678

Developer		PageRank
Joshua Peek	josh[AT]joshpeek.com	0.00009536
Aman Gupta	aman[AT]tmm1.net	0.00008860
Steve Richert	steve.richert[AT]gmail.com	0.00008850
Michael Klishin	michaelklishin[AT]me.com	0.00008170
Josh Kalderimis	josh.kalderimis[AT]gmail.com	0.00008163

[1] F Thung, TF Bissyande, D Lo, L. Jiang, Network structure of social coding in github, CSMR2013.

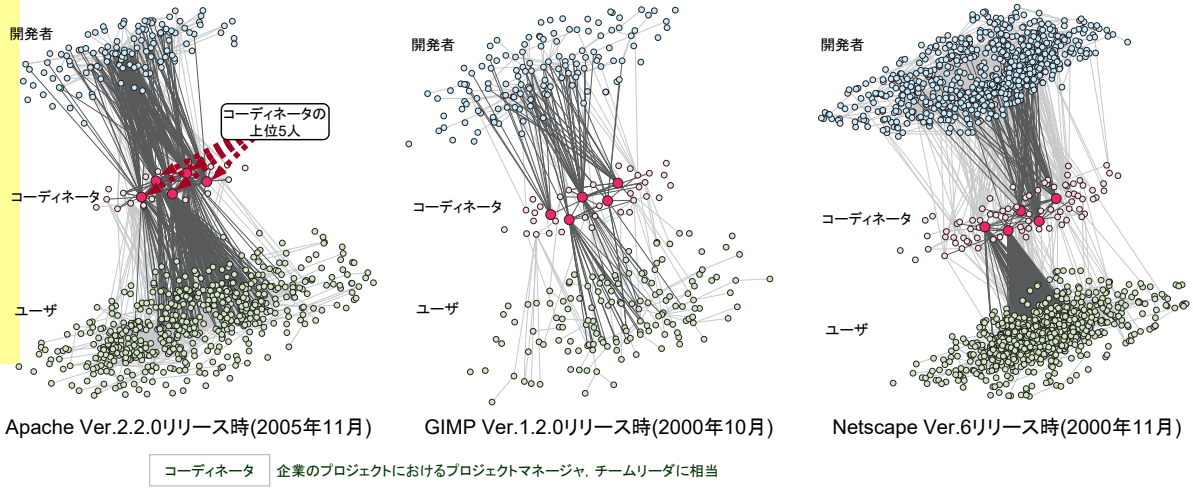
人的要因：コミュニティ分析

- TransparencyとCommunicationの分析[1]
  - Transparency：GitHub上で見えている他者の活動
  - 24名のGitHubユーザへのインタビュー
  - 結論
    - 人々はGitHub上の他者の活動情報から，各人のスキル，コード修正の意図，どのプロジェクトが盛況等をうまく推測している．

[1] L Dabbish, C Stuart, J Tsay, J Herbsleb, Social coding in GitHub: transparency and collaboration in an open software repository, CSCW2012. (被引用数 315)

人的要因：コミュニティ分析

■コーディネーターの活動の分析 [1]



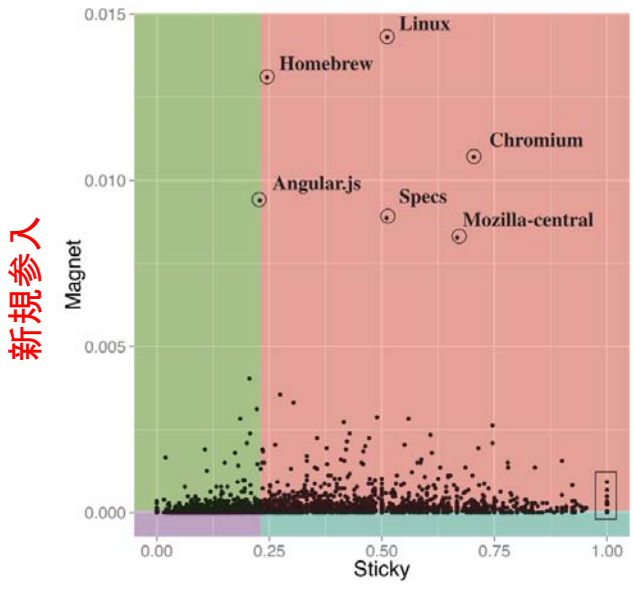
Apache Ver.2.2.0リリース時(2005年11月)　GIMP Ver.1.2.0リリース時(2000年10月)　Netscape Ver.6リリース時(2000年11月)

コーディネータ　企業のプロジェクトにおけるプロジェクトマネージャ，チームリーダに相当

[1] S. Matsumoto, Y. Kamei, M. Ohira, and K. Matsumoto. A comparison study on the coordination between developers and users in FOSS communities. STC'08, pp.1-9, 2008.

人的要因：コミュニティ分析

■Magnet or Sticky [1][2]



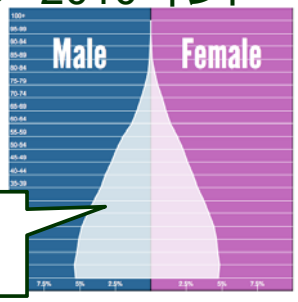
新規参入

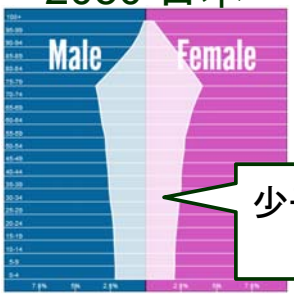
継続参加

[1] K. Yamashita, S. McIntosh, Y. Kamei, N. Ubayashi, Magnet or Sticky?: An OSS Project-by-Project Typology, MSR2014.
[2] K. Yamashita, S. McIntosh, Y. Kamei, N. Ubayashi, A. E. Hassan, N. Ubayashi, Magnet or Sticky? Measuring Project Characteristics from the Perspective of Developer Attraction and Retention ,Journal of Information Processing, 2016.

人的要因：コミュニティ分析

- Software Population Pyramid [1]
- 一般的な人口ピラミッドの場合は．．．
  縦軸に年齢，左右に男女別の人口を棒グラフで表す
  - 国の人口状態を示し，政治や社会，経済に関する様々な考察に使われる
  - 出生率や男女比などをもとに，将来の人口ピラミッドの形状予測が行われる　2010 インド　　2050 日本





発展途上国に
よく見られる形状

少子高齢化が進む国に
よく見られる形状

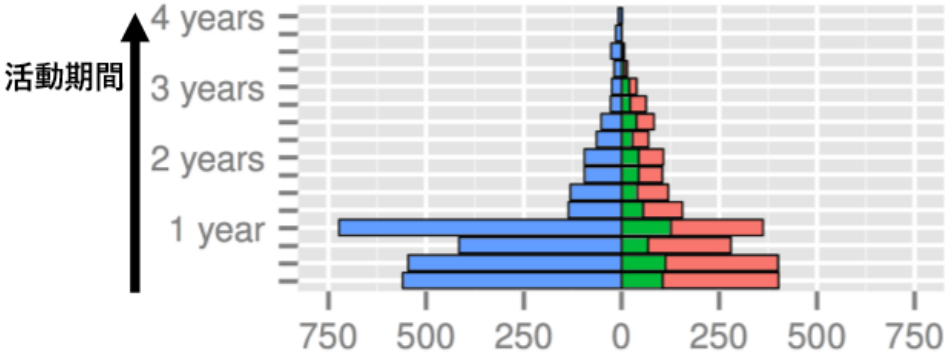
http://populationpyramid.net/

[1] S. Onoue, H. Hata, A. Monden, K. Matsumoto, Investigating and Projecting Population Structures in Open Source Software Projects: A Case Study of Projects in GitHub. IEICE Transactions 99-D(5) 2016.

人的要因：コミュニティ分析

- Software Population Pyramid [1]

- Coding: コーディングを1回以上行った貢献者
- Non-coding: コメントやバグ報告のみを行った貢献者
- Moved: Non-codingからCodingに転じた貢献者

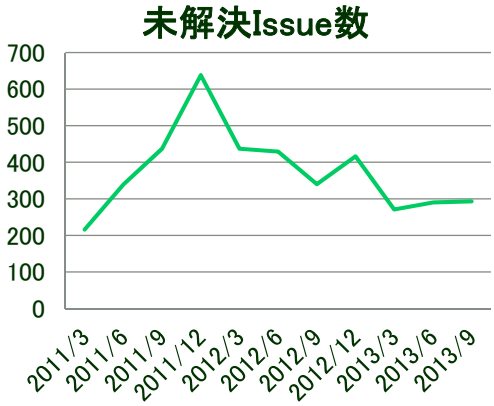


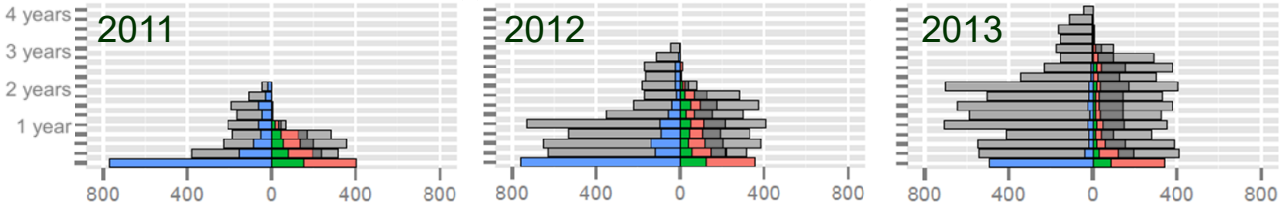
[1] S. Onoue, H. Hata, A. Monden, K. Matsumoto, Investigating and Projecting Population Structures in Open Source Software Projects: A Case Study of Projects in GitHub. IEICE Transactions 99-D(5) 2016.

Software Population Pyramid

◆homebrewプロジェクト

- 未解決issue数が2011年末をピークに減少している
  - 修正するバグが減ったので貢献者が離脱した？
  - 修正したいバグが修正できたので貢献者が離脱した？

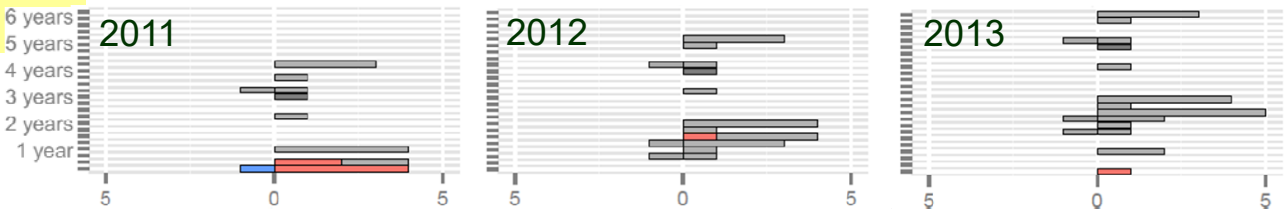




Software Population Pyramid

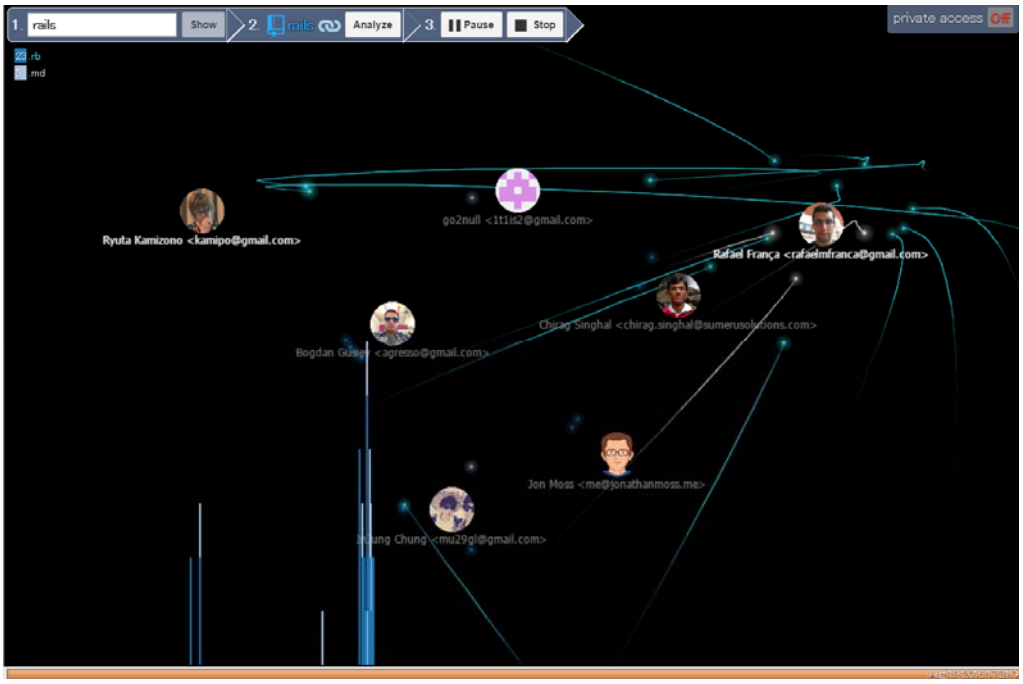
■blueprint-css

- □少数ではあるが貢献者がプロジェクトに参入してる
  - ◆pull request(コミット依頼)を行う新規貢献者がいた
- □コミッター含む長期貢献者が離脱しているため，pull request が承認されなかった
  - ◆新規貢献者がすぐにプロジェクトを離脱してしまった



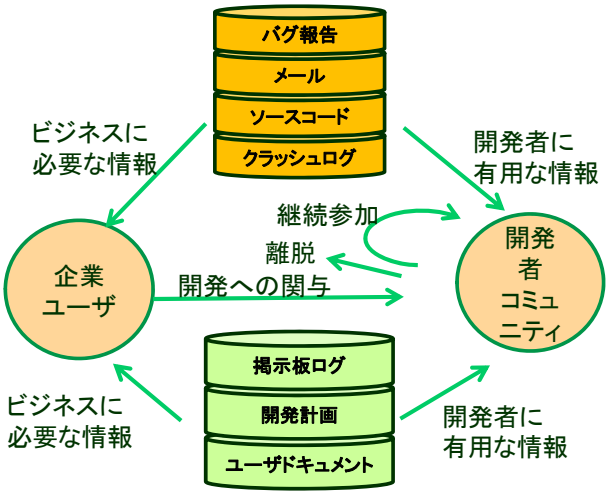
人的要因：コミュニティ分析

- 可視化
  - GitHub Visualizer http://ghv.artzub.com/



人的要因：コミュニティ分析

- Information and Software Technology Special Issue on Software Ecosystem http://www.sciencedirect.com/science/journal/09505849/56/11
  - エコシステム＝
    生態系，循環系，産業構造



- 発展：
  - ゲーム理論に基づくモデル化・分析 [1]

[1] H. Hata, T. Todo, S. Onoue, K. Matsumoto, Characteristics of Sustainable OSS Projects: A Theoretical and Empirical Study, CHASE2015.

人的要因：人材育成

- OSS人材育成
  - 情報処理推進機構 オープンソフトウェア・センター
    - OSSモデルカリキュラム
- 研究
  - 新人を教育する「メンター」の分析 [1]
    - SNAに基づくメンター自動推薦アルゴリズムYoda

System	Top 1 Correct	Top 1 Wrong	Top 2 Correct	Top 2 Wrong
Apache	11 (85%)	2 (15%)	21 (81%)	5 (19%)
FreeBSD	10 (30%)	23 (70%)	16 (24%)	50 (76%)
PostgreSQL	7 (100%)	0 (0%)	14 (100%)	0 (0%)
Python	20 (65%)	11 (35%)	48 (77%)	14 (23%)
Samba	31 (94%)	2 (6%)	54 (82%)	12 (18%)

  - 新人がプロジェクトを去る原因の分析[2]
    - 開発者間の会話の内容に着目して分析している.

[1] G Canfora, M Di Penta, R Oliveto, S. Panichella, Who is going to mentor newcomers in open source projects? FSE2012.
[2] I Steinmacher, I Wiese, AP Chaves, Why do newcomers abandon open source software projects?, CHASE2013.

人的要因：Emotion Mining

- Apache software foundationの117 OSSプロジェクトのissue reportにおける感情の分析[1]
  - ランダムサンプリングした約800のコメントを最大16名でタグ付け
  - Love: Thanks for your input!
  - Joy: great work you guys!
  - Surprise: Oops. It needs to be added to Makefile
  - Anger: This is an – ugly – workaround
  - Sadness: Sorry for delay
  - Fear: I'm most concerned with …

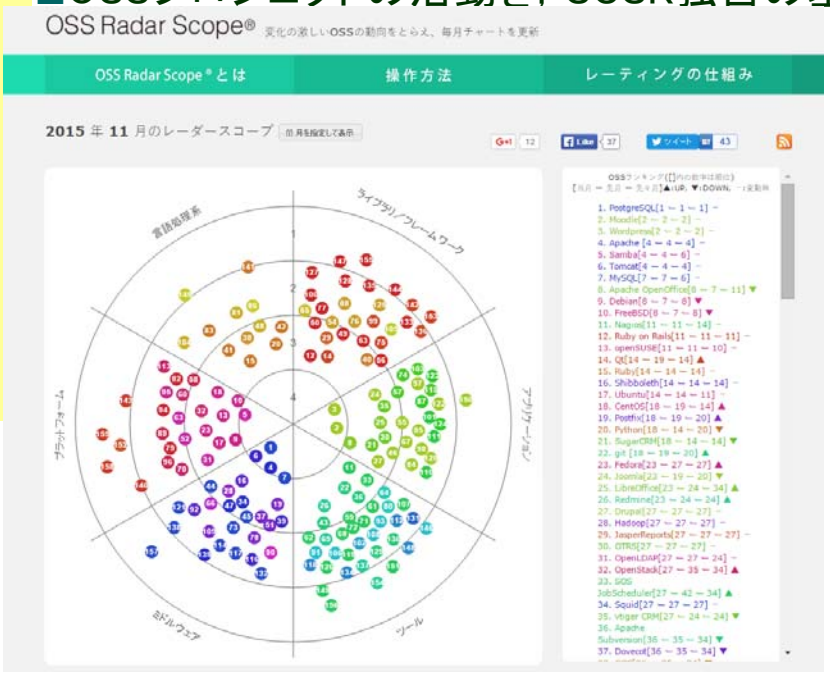
[1] A. Murgia, P. Tourani, B. Adams, M. Ortu, Do developers feel emotions? an exploratory analysis of emotions in software artifacts, MSR2011.

人的要因:OSS評価

- EU:Quality Platform for OSS (QualiPSo)
  - OSS成熟度モデル，OSS信頼性評価モデル
- アジア:東北アジアOSS推進フォーラム(NEAOSS)
  RepOSS

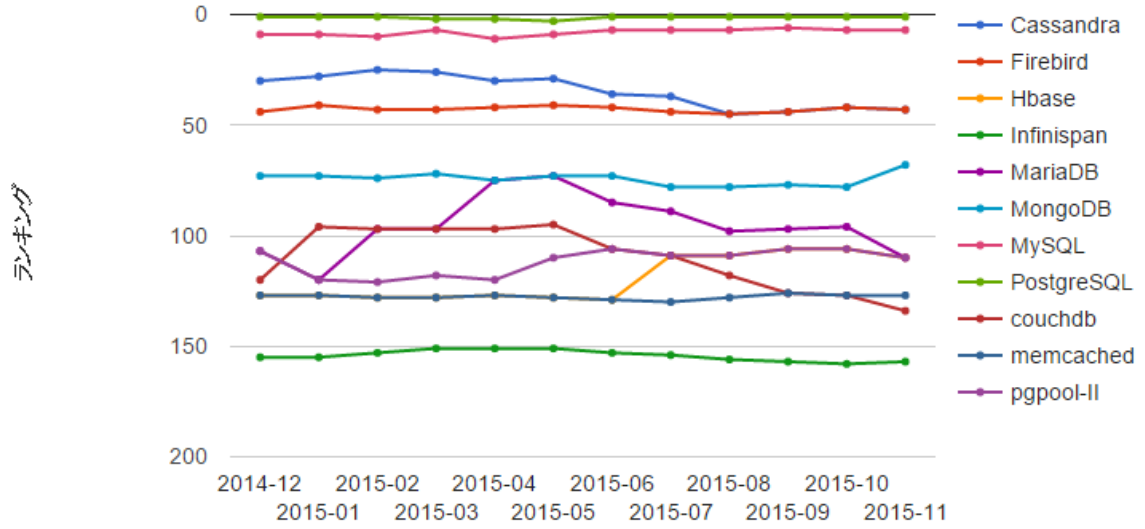
人的要因:OSS評価

- OSSレーダースコープ　http://radar.oss.scsk.info/
  - OSSプロジェクトの活動を，SCSK独自の基準で評価している.



人的要因：OSS評価

■ OSSのランキング月次推移（DBMS）



- PostgreSQL，MySQLが2強．
- 次いで，Cassandra，Firebird
- MongoDBが浮上してきた．

人的要因：OSS評価

■ レーティング方法

コミュニティの活動状況

開発者間でのオープンなコミュニケーション頻度が高いのは良質のOSSである可能性が高い。
ユーザーからのフィードバック（質問含む）が多く、共有されているのは良質のOSSである可能性が高い。

リリース履歴

計画的なバージョンアップを経ているOSSは良質のOSSである可能性が高い。
パッチリリースの頻度が高いOSSは良質のOSSである可能性が高い。
開発の経過に関する情報を包み隠さず公開しているプロジェクトは良質のOSSを生み出す可能性が高い。

ドキュメント／関連出版物

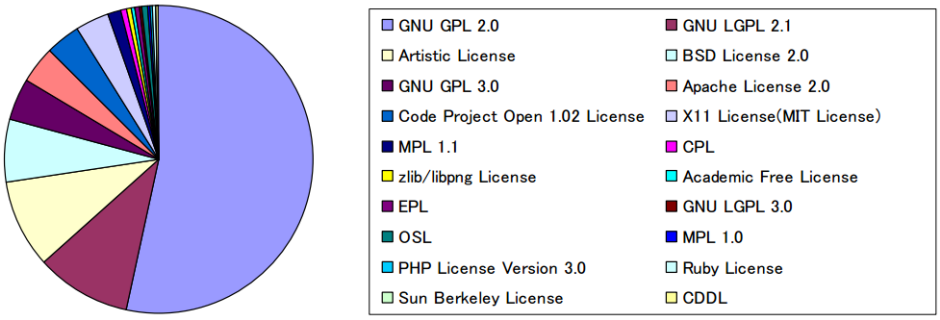
ドキュメントの質／量に力を入れているのは良質のOSSである可能性が高い。
出版された書籍の冊数が多いOSSは良質のOSSである可能性が高い。

サポート情報

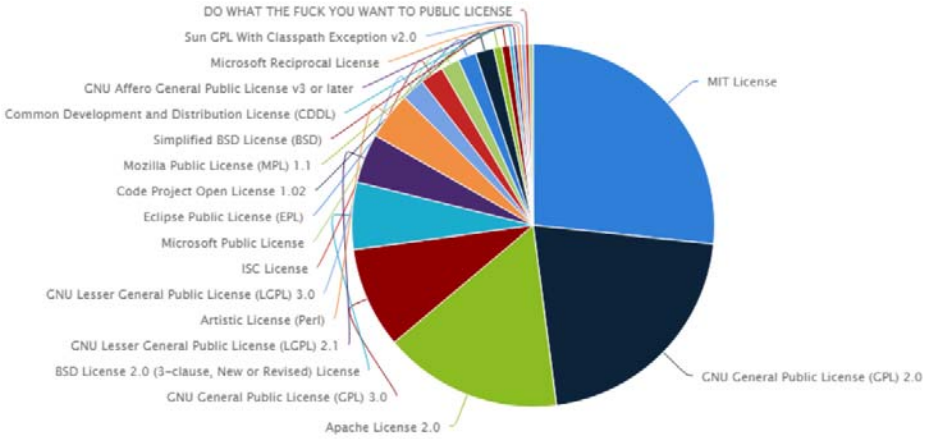
サービスを提供する「ベンダーの数」×「提供されるサービスの品揃え」と、OSSの質は比例する。

OSSライセンス

■2009年



■2016年



[1] Black Duck Software: 2009 http://www.blackducksoftware.com/oss/licenses
[2] Black Duck Software: 2016 https://www.blackducksoftware.com/top-open-source-licenses

OSSライセンス

■Open Source Software
= Source Code + License

		複製・再頒布可能	改変可能	改変部分のソース公開要	組み合わせた他ソフトウェアのソースコード公開も必要
OSS	コピーレフト型（GPL等）	○	○	○	○
	準コピーレフト型（MPL等）	○	○	○	×
	非コピーレフト型（BSDライセンス等）	○	○	×	×
フリーソフトウェア		○	×	—	—
商用ソフトウェア		×	×	—	—

OSSライセンス

- ライセンスマッチングシステム [1]
  - ソースコードを入力として，OSSのライセンスを自動的に同定する.
- ライセンス自動分析サービス
  - Black Duck Software社のProtex[2]
    - 200,000以上のOSSプロダクトをベースにチェック
  - Palamida社のPalamida[3]
    - コードのマッチングによってライセンス違反を検出

[1] D. M. German, Y. Manabe, K. Inoue. A Sentence-Matching Method for Automatic License Identification of Source Code Files. ASE'10, pp. 437-446, 2010.
[2] www.blackducksoftware.com/protex
[3] www.palamida.com

OSSライセンス

- 意図しないOSSライセンス違反の例

1. LinksysはBroadcom社にWRT54GLというソフトウェアの開発を発注した.
2. LinksysはWRT54GLを販売した.
3. CISCOはLinksysを買収した.
4. CISCOはSoftware Freedom Law Center (SFLC)に訴えられた.
   - WRT54GLがBusyBoxのライセンスに違反している.
   - BusyBox：組込みLinux向けのツール群

分析対象リポジトリの広がり

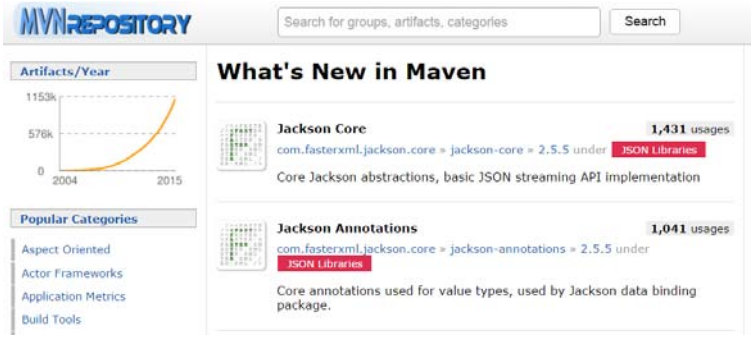
分析対象リポジトリの広がり

- クラッシュレポートリポジトリ
  - 一部のOSSプロジェクトでは，クラッシュレポートシステムを導入している.
    - FirefoxやThunderbirdでは，Socorroサーバを利用[1]
  - 研究テーマ例
    - クラッシュレポートの自動分類[2]
    - バグ修正の優先順位付け[3]
    - 不具合箇所の自動特定[4]

[1] Firefoxのクラッシュサーバ, https://crash-analysis.mozilla.com/
[2] Y. Dang, R. Wu, H. Zhang, D. Zhang, P. Nobel, Rebucket: a method for clustering duplicate crash reports based on call stack similarity.
[3] D. Kim, X. Wang, S. Kim, A. Zeller, S.C. Cheung, S. Park, Which crashes should I fix first?: predicting top crashes at an early stage to prioritize debugging effort, IEEE Trans. Soft. Eng., 2011.
[4] 亀井, 長本, シャシャンク, 小須田, 伊原, 鵜林, クラッシュリポジトリマイニング－ソースコードの欠陥個所の特定に向けて－, ソフトウェア工学の基礎XX, FOSE2013.

分析対象リポジトリの広がり

- ライブラリリポジトリ
  - Maven[1][2]：Javaの依存ライブラリの管理を行う．
    - ライブラリの自動ダウンロード，ソースコードの自動ビルド，自動テスト，プロジェクトWebサイトの作成などが行える．
    - pom.xmlに必要なライブラリを記述する．
  - ライブラリ群は，Maven Central Repositoryで管理されている．
- ライブラリやその依存関係を分析するのに便利[3]．



[1] http://search.maven.org/
[2] http://mvnrepository.com/
[3] 砂田，門田，松本，Javaソフトウエア開発における使用ライブラリの分析，ソフトウェア信頼性研究会，2012.

分析対象リポジトリの広がり

- Stack Overflow　http://stackoverflow.com/
  - プログラミング技術に関するQ&Aサイト
  - ゲーミフィケーションの導入
    - 良質な回答者への評価ポイント，バッジの付与
  - XMLファイルでダウンロード可能．日本語版のサイトもある．
- 研究テーマ例
  - 良い質問の分析（良いコード例）[1]
  - 高い評価を得るためには？[2]
  - Stack Overflowでの活動とGitHub上での活動の関係の分析[3]

[1] SM Nasehi, J Sillito, F Maurer, C. Bums, What makes a good code example?: A study of programming Q&A in StackOverflow, ICSM2012.
[2] A Bosu, CS Corley, D Heaton, D Chatterji, JC Carver, NA Kraft, Building reputation in stackoverflow: an empirical investigation, MSR2013.
[3] B Vasilescu, V Filkov, A. Serebrenik, StackOverflow and GitHub: Associations between software development and crowdsourced knowledge, SocialCom2013.

分析対象リポジトリの広がり

- Stack Overflow上の活動とGitHub上の活動の関係の分析[3]
  - SO: July 2008 – Aug. 2012. (1,295,622ユーザ)
  - Github: GHTorrent経由でJul.2011–Apr.2012. (39万ユーザ)
  - 主な分析結果

Active GitHub committers ask fewer questions on StackOverflow than others.
More active GitHub committers provide more answers on StackOverflow.
More active StackOverflow answerers make more commits on GitHub.

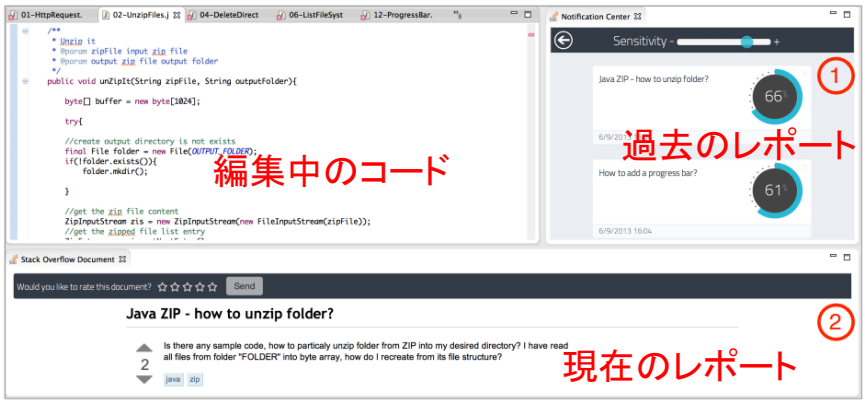
[3] B Vasilescu, V Filkov, A. Serebrenik, StackOverflow and GitHub: Associations between software development and crowdsourced knowledge, SocialCom2013.

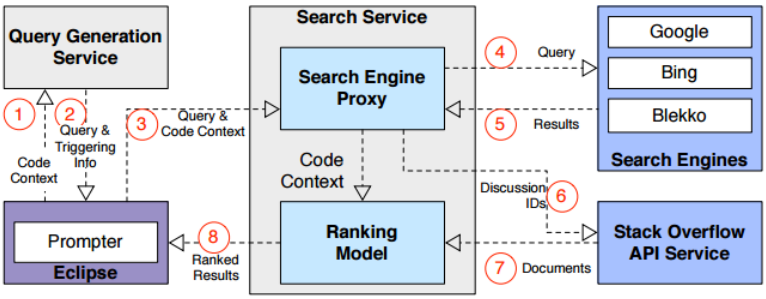
分析対象リポジトリの広がり

- Stack Overflow documentation
  - http://stackoverflow.com/documentation
  - サンプルコードを中心とした解説

分析対象リポジトリの広がり

- 研究テーマ例
  - IDEとStack Over flowの統合 [1]





[1] L Ponzanelli, G Bavota, M Di Penta, R Oliveto, M Lanza, Mining StackOverflow to turn the IDE into a self-confident programming prompter, MSR2014.

分析対象リポジトリの広がり

- 継続的インテグレーション（CI)ツール
  - Travis-CI, Jenkinsなど
- 研究テーマ例
  - GitHubにおけるTravis-CI利用状況の分析[1][2]
  - Build statusに影響する要因（コミットコメント文面）の分析[2]

[1] B Vasilescu, S Van Schuylenburg, J Wulms, A Serebrenik, MGJ van den Brand, Continuous integration in a social-coding world: Empirical evidence from GitHub, arXiv:1512.01862 (ICSME2014).
[2] M Beller, G Gousios, A Zaidman, Oops, my tests broke the build: An analysis of Travis CI builds with GitHub, PeerJ Preprints 4:e1984v1, 2016.
[2] EA Santos, A Hindle, Judging a commit by its cover: correlating commit message entropy with build status on travis-CI, MSR2016.

分析対象リポジトリの広がり

- コードレビューリポジトリ
  - Gerrit Code Review，Rietveldなど
  - GitHubでは，Gerritと連携可能
- 公開データ
  - Android Open Source Project，OpenStack，QT，Chromium Projectsのレビューデータ[1][2]

[1] http://sdlab.naist.jp/reviewmining/
[2] K Hamasaki, RG Kula, N Yoshida, AEC Cruz, K Fujiwara, H Iida, Who does what during a code review? datasets of OSS peer review repositories, MSR2013.

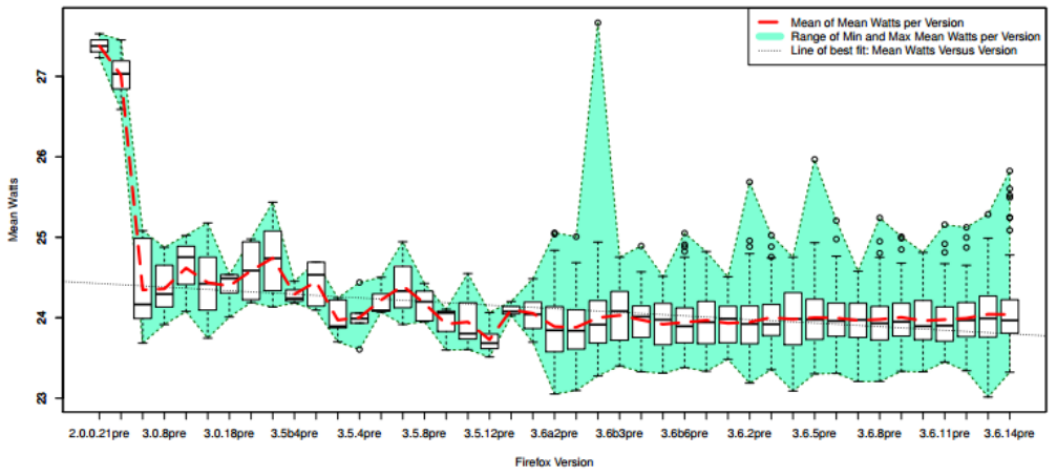
分析対象リポジトリの広がり

- Twitter，Blog，Wikiなど
  - ハッシュタグ#csharp，#java，#javascript，#dotnet，#jquery，#azure，#scrum，#testing，#opensourceのついているツイートの分析[1]
  - Drupalプロジェクトにおけるtwitterの活用状況の分析[2]

[1] Y Tian, P Achananuparp, IN Lubis, D. Lo, E-P Lim, What does software engineering community microblogs about?, MSR2012.
X Wang, I Kuzmickaja, KJ Stol, P Abrahamsson, B Fitzgerald, Microblogging in open source software development: The case of drupal using twitter, IEEE Software, 29(3), 2014.

分析対象リポジトリの広がり

- Green Mining[1][2]
  - モバイル端末の消費電力を計測する.
    - ソフトウェアのバージョンごとの消費電力[2]



[1] A. Hindle, Green mining: Investigating power consumption across versions, ICSE2012 NIER Track, 2012, http://ur1.ca/84vh4
[2] A Hindle, Green mining: A methodology of relating software change to power consumption, MSR2012.

分析対象リポジトリの広がり

- モバイルアプリ
  - Androidアプリの期待動作と実際の挙動との差異の分析[1]
    - Google playより，Top 150アプリ×30カテゴリをダウンロードして分析している.
  - Androidアプリのマルウェア分析[2]
    - 1200のmalware app data set
    - http://www.malgenomeproject.org/ (2015.12.21 まで)
  - Blackberry app storeの分析[3]

[1] A. Gorla, I. Tavecchia, F. Gross, A. Zeller, Checking app behavior against app descriptions, ICSE2014.
[2] Y. Zhou, X. Jiang, Dissecting Android malware: characterization and evolution, SP2012.
[3] M. Harman, Y. Jia, Y. Zhang, App store mining and analysis: MSR for app stores, MSR2012.

分析対象リポジトリの広がり

- オンラインジャッジシステム
  - Aizu Online Judge[1], PKU judgeonline[2], TopCoder[3]など
  - プログラミングの問題，過去に提出されたソースコード（の履歴）がリポジトリに保存されている．
  - 研究例：初心者の学習支援[4]
- プログラミングのe-learningシステム[5]

[1] http://jdge.u-aizu.ac.jp/onlinejudge/
[2] http://poj.org/
[3] https://www.topcoder.com/
[4] 藤原新, プログラミング学習者向け漸進的ソースコード提示システムTAMBA, 奈良先端科学技術大学院大学修士論文, NAIST-IS-MT1451091, 2016.
[5] http://www.bluej.org/

公開データ

データ取得＆クリーニングの難しさ

- 2006年当時：Audris Mockus先生曰く[1]：
  データの“cleaning”には95%の労力を要する－分析はわずか1～5%

- 現在：新規研究参入者向けに，公開データセットが増えつつある．
  - MSR Data Showcase
    - データ提供者も被引用回数が増えるためWin-Winとなる．
  - MSR Challenge
    - データを公開して分析論文を募集している．

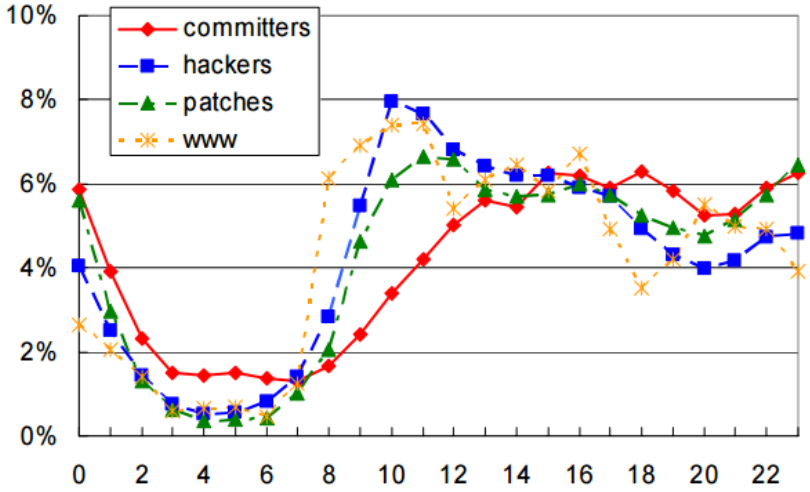
[1] A. Mockus, How to run empirical studies using project repositories, IASESE2006.

公開データの例

- MSR Mining Challenge 2016
  - Boa dataset http://boa.cs.iastate.edu/
  - 700,00 SourceForge projects, 8,000,000 GitHub repositories
- MSR Mining Challenge 2015
  - Stack Overflow data http://2015.msrconf.org/challenge_data/
- MSR Mining Challenge 2014
  - 90 GitHub projects http://2014.msrconf.org/challenge.php
- MSR Mining Challenge 2013
  - Stack Overflow data http://2013.msrconf.org/challenge.php
- MSR Mining Challenge 2012
  - Android Platform change report, bug report
    http://2012.msrconf.org/challenge.php#challenge_data

公開データの例

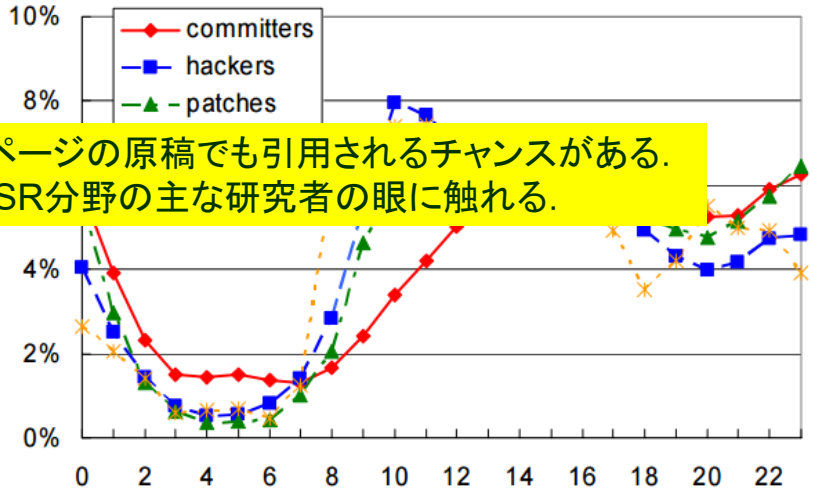
- MSR Mining Challenge 2006への参加[1]
  - E-mailに基づく開発者の活動時間の分析
  - PostgreSQLが対象，原稿2ページ，被引用数15



[1] M Tsunoda, A Monden, T Kakimoto, Y Kamei, K Matsumoto, Analyzing OSS developers' working time using mailing lists archives, MSR2006.

公開データの例

- MSR Mining Challenge 2006への参加[1]
  - E-mailに基づく開発者の活動時間の分析
  - PostgreSQLが対象，原稿2ページ，被引用数15



[1] M Tsunoda, A Monden, T Kakimoto, Y Kamei, K Matsumoto, Analyzing OSS developers' working time using mailing lists archives, MSR2006.

公開データの例

MSR2015 Data Showcase paper

- A Repository with 44 Years of Unix Evolution
- The Debsources Dataset: Two Decades of Debian Source Code Metadata
- A Dataset of the Activity of the git superrepository of Linux
- StORMeD: Stack Overflow Ready Made Data
- The MetricsGrimoire Database Collection
- Landfill: an Open Dataset of Code Smells with Public Evaluation
- Fuse: A Reproducible, Extendable, Internet-scale Corpus of Spreadsheets
- Dataset of developer-labeled commit messages for task classification validation
- A Novel Industry Grade Dataset for Fault Prediction based on Model-Driven Developed Automotive Embedded Software
- The Firefox Defect Temporal Dataset
- An Architectural Evolution Dataset
- A Dataset For API Usage
- Generating the Blueprints of the Java Ecosystem
- A Data Set for Social Diversity Studies of GitHub Teams
- A Dataset of High Impact Bugs: Manually-Classified Issue Reports
- A Dataset of Open Source Android Applications

公開データの例

MSR2015 Data Showcase paper

- A Dataset of High Impact Bugs: Manually-Classified Issue Reports

  By Masao Ohira, Yutaro Kashiwa, Yosuke Yamatani, Hayato Yoshiyuki, Yoshiya Maeda, Nachai Limsettho, Keisuke Fujino, Hideaki Hata, Akinori Ihara and Kenichi Matsumoto

  http://oss.sys.wakayama-u.ac.jp/?p=1009

issue_ic	type	status	resoluti	compor	priority	reporte	created	assigne	assigne	resolve	created	assigne	summa	descrip	a
12	Improveme	Resolved	Fixed	wicket	Major	J.W. Jans	2006/10/	2009/10/	Juergen D	2009/10/	1100.071	0	open Mod	Wicket 1.2	2
16	Bug	Closed	Fixed	wicket	Major	Korbinian	2006/10/	2006/10/	Frank Bille	2006/10/	0.387014	0.149734	missing ba	yesterday	2
31	Bug	Resolved	Fixed	wicket	Minor	Bernhard I	2006/11/(	2006/11/(	Martijn Da	2006/11/(	0.512269	0.505174	Wrong sou	The paths	1
38	Bug	Resolved	Fixed	wicket	Minor	Alexander	2006/11/(	2007/02/	Jean-Bap	2007/03/	126.6015	17.42028	WicketTes	Basics of	2
81	Improveme	Resolved	Fixed	wicket	Major	Martijn Da	2006/11/	null	Unassigne	2007/01/(	49.54426	null	Make debu	Currently	2
82	Improveme	Closed	Fixed	wicket	Major	Igor Vaynt	2006/11/	2006/11/	Igor Vaynt	2006/11/	0.006169	0.006169	Support c	http://ww	1
85	Bug	Closed	Fixed	wicket	Major	Jean-Bap	2006/11/	null	Unassigne	2006/11/	2.511944	null	FormTeste	When usin	1
88	Bug	Resolved	Fixed	wicket	Major	Juergen D	2006/11/	2006/11/	Igor Vaynt	2006/11/	0.392373	0.066667	NPE in Ch	The Forms	2
91	Improveme	Closed	Fixed	wicket	Trivial	Dmitry Ka	2006/11/	null	Unassigne	2006/11/	0.310579	null	Russian A		2
100	Improveme	Resolved	Fixed		Major	Eelco Hille	2006/11/	2006/11/	Eelco Hille	2006/11/	4.045706	4.045648	extract IP	If we extra	2
101	Bug	Resolved	Fixed		Minor	Alastair M	2006/11/	2006/11/	Alastair M	2006/11/	0.008692	0.008692	Lazy-regis	If you mou	2
105	Improveme	Resolved	Fixed	wicket	Trivial	Martijn Da	2006/11/	2006/12/	Igor Vaynt	2006/12/	36.35197	5.79E-05	Remove R	To cut dov	2
142	Bug	Closed	Fixed	wicket	Major	Ingram Ch	2006/12/(	2006/12/(	Igor Vaynt	2006/12/(	3.49875	0.082292	NullPointe	wicket.fee	1
144	Bug	Resolved	Fixed	wicket	Major	Eelco Hille	2006/12/(	2006/12/(	Eelco Hille	2006/12/(	3.711771	3.711771	wrong han	Resource'	2
149	Bug	Resolved	Fixed	wicket	Major	Nathan Ha	2006/12/(	2006/12/	Alastair M	2007/01/	41.90734	17.93069	mounted l	The path i	1

- High impact bugをblocking, surprising, dormant, security, performance, breakage に手動分類している．人海戦術で一人当たり数百のバグを分析
- High impact bugの分析に役立つ

公開データの例

- GHTorrent http://ghtorrent.org/
  - GitHub上の全イベントを取得し，DBに蓄積している．
  - Gitリポジトリは全履歴を保持しているが，GitHubから全履歴を取得することは難しい（GitHub APIの制約）．
    - 例：PULL Requestは，GitではなくGitHubの機能であるため，GitHubからデータを取得する必要がある．
  - GHTorrent経由でデータを取得すると便利

- GitHub Archive https://www.githubarchive.org/
  - GHTorrentと同様

公開データの例

- tera-PROMISE http://openscience.us/repo/
  - The largest repositories of SE research data
  - OSSに限らない．商用開発のデータも多い．NASAにおける開発データなども．古いデータから新しいデータまで．
  - コード分析，バグ，開発工数，リファクタリング，要求分析，ソフトウェア保守，テスト生成など

公開データの例

- FLOSSmole http://flossmole.org/
  - tera-PromiseのOSSプロジェクト版
  - 多数のOSSプロジェクトのデータが公開されている.
  - データソース
    - Sourceforge
    - Freecode (Freshmeat)
    - Rubyforge
    - GitHub
    - ・・・・
- Software-artifact infrastructure repository http://sir.unl.edu/portal/
  - 複数バージョンのソースコード, テストケース, バグ, スクリプトなどが整理されている(らしい).

公開データの例

- Eclipse Bug Data! Release 2.0a, 2007-12-28
  - https://www.st.cs.uni-saarland.de/softevo/bug-data/eclipse/

plugin	filename	pre	post	ACD	FOUT_avg	FOUT_ma:	FOUT_sun	MLOC_avg	MLOC_ma	MLOC_sur
org.eclipse	/org.eclips	1	0	0	6.75	29	54	9.25	32	74
org.eclipse	/org.eclips	1	0	0	12.5	13	25	16	18	32
org.eclipse	/org.eclips	0	0	0	5.333333	10	16	12.66667	29	38
org.eclipse	/org.eclips	2	0	0	7.333333	16	88	9.666667	28	116
org.eclipse	/org.eclips	2	0	4	6.210526	27	118	9.894737	55	188
org.eclipse	/org.eclips	1	0	0	1	1	2	5	8	10
org.eclipse	/org.eclips	0	0	0	1.333333	4	8	3.666667	12	22
org.eclipse	/org.eclips	0	0	0	0	0	0	0	0	0
org.eclipse	/org.eclips	1	0	0	0	0	0	0	0	0
org.eclipse	/org.eclips	2	0	0	5.090909	22	168	8.484848	32	280
org.eclipse	/org.eclips	1	0	0	4	7	12	8.333333	13	25

- ファイル毎のバグ数がカウントされている.
- リリース前(pre)とリリース後(post)のバグ数がカウントされている.
- 多数のソースコードメトリクスが計測されている.

公開データの例

- 他分野：NII　情報学研究データリポジトリ
  http://www.nii.ac.jp/dsc/idr/datalist.html
  - Yahooデータセット
  - 楽天データセット
  - ニコニコデータセット
  - リクルートデータセット
  - クックパッドデータセット
  - ・・・・

分析ツール

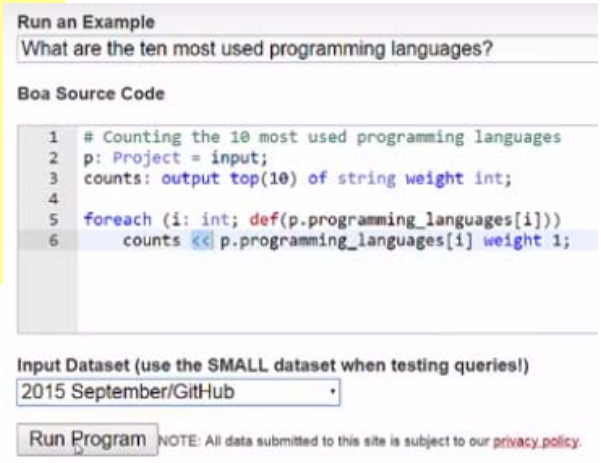
分析ツール

- メトリクス計測
  - Understand https://www.techmatrix.co.jp/quality/understand/
  - SonarQube http://www.sonarqube.org/
  - MASU https://github.com/kusumotolab/MASU
- コードクローン計測
  - CCFinderX http://www.ccfinder.net/
  - DECKARD https://github.com/skyhover/Deckard
- テストカバレッジ計測
  - EclEmma(JaCoCo) http://www.eclemma.org/jacoco/
- GitHubからのデータ計測
  - RepositoryProbe[1] https://github.com/exKAZUu/RepositoryProbe

[1] 高澤, 坂本, 鷲崎, 深澤, RepositoryProbe: リポジトリマイニングのためのデータセット作成支援ツール, コンピュータ ソフトウェア, 32(4), 2015.

分析ツール

- マイニング支援
  - BOA: An enabling language and infrastructure of ultra-large-scale MSR studies http://boa.cs.iastate.edu/
  - マイニングのためのプログラミング環境＋データセット



[1] R Dyer, HA Nguyen, H Rajan, TN Nguyen, Boa: A language and infrastructure for analyzing ultra-large-scale software repositories, ICSE2013.

分析ツール

- マイニング支援
  - Grimoire Lab http://grimoirelab.github.io/
    - CVS/SVN/メーリングリストからデータ抽出するためのツールLibresoft toolsetを起源としている.
    - その後，ソフトウェア開発，コミュニティ分析ツールMetricsGrimoireとなり，Grimoire Labへと発展した（らしい）.

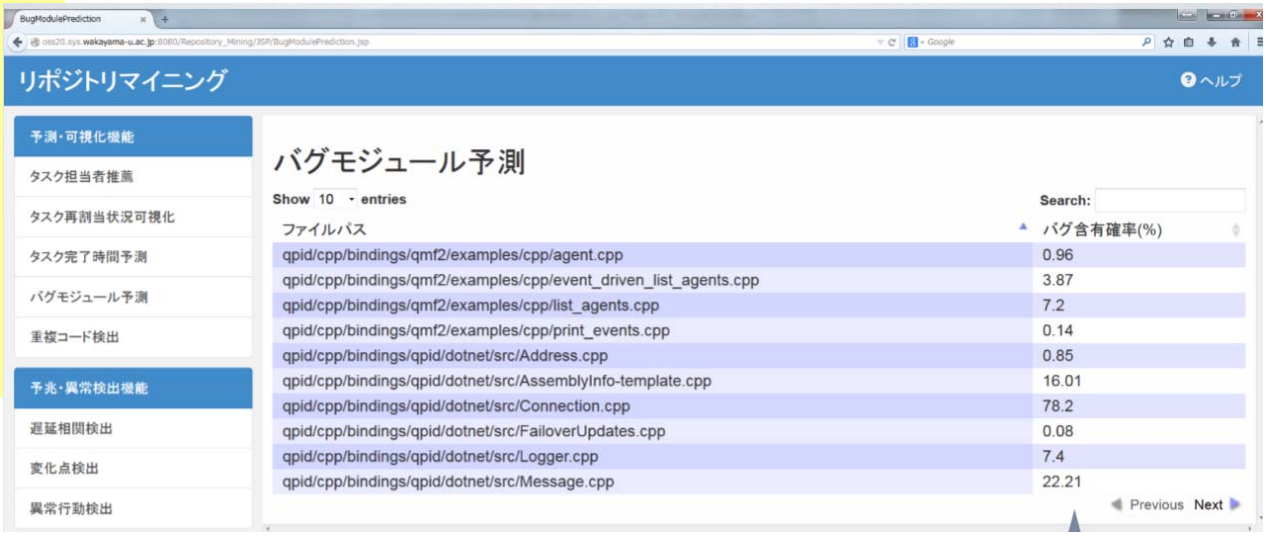
分析ツール

- リポジトリマイニングプラグイン for EPM-X[1]
  http://oss.sys.wakayama-u.ac.jp/msr/
  - MSRの研究成果を実開発プロジェクトで利用できる.
    - タスク担当者推薦，タスク再割当状況可視化，タスク完了時間予測，バグモジュール予測，コードクローン検出
    - 遅延相関検出，変化点検出，影響波及分析
- 定量的プロジェクト管理ツールEPM-X
  http://www.ipa.go.jp/sec/softwareengineering/tool/ipf/
  - プロジェクト管理ツール(Redmine, Trac)，構成管理ツール（Subversion, GIT）から自動データ集計・グラフ化を行う.

[1] 大平雅雄, 開発データマイニングのススメ:「見える化」の次のステップ, SECセミナー 定量的なプロジェクト管理・プログラム管理のススメ～見える化による生産性・信頼性の向上～, 2015年10月30日
http://sec.ipa.go.jp/users/seminar/seminar_tokyo_20151030-02.pdf

分析ツール

- リポジトリマイニングプラグイン for EPM-X[1]
  http://oss.sys.wakayama-u.ac.jp/msr/



[1] 大平雅雄, 開発データマイニングのススメ:「見える化」の次のステップ, SECセミナー 定量的なプロジェクト管理・プログラム管理のススメ～見える化による生産性・信頼性の向上～, 2015年10月30日
http://sec.ipa.go.jp/users/seminar/seminar_tokyo_20151030-02.pdf

分析ツール

- メソッドレベルのバージョン管理ツールHistorage[1][2]
  - Gitリポジトリからの変換ツールKenja[3]
  - Historageのホスティングサービスkataribe[4][5]
- 応用例
  - メソッドレベルバグ予測[6]

[1] H. Hata, O. Mizuno, T. Kikuno, Historage: Fine-grained Version Control System for Java, IWPSE-EVOL2011.
[2] K. Uemura, Y. Saito, S. Fujiwara, D. Tanaka, K. Fujiwara, H. Hata, H. Iida, K. Matsumoto, A Hosting Service of Multi-Language Historage Repositories, ICIS2016.
[3] https://github.com/niyaton/kenja
[4] K. Fujiwara, H. Hata, E. Makihara, Y. Fujihara, N. Nakayama, H. Iida, K. Matsumoto, Kataribe: a hosting service of historage repositories, MSR2014.
[5] http://sdlab.naist.jp/kataribe/
[6] H. Hata, O. Mizuno, T. Kikuno, Bug prediction based on fine-grained module histories, ICSE2012.

分析ツール

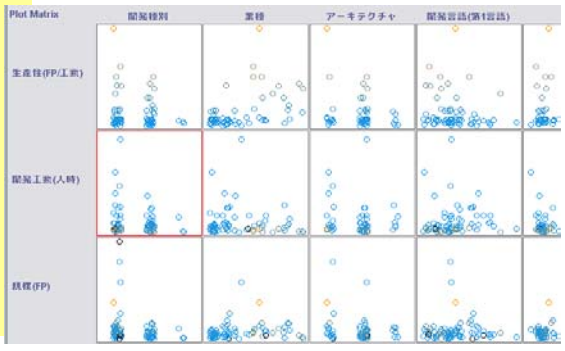
- R https://www.r-project.org/ 分析に必須！
  - 統計パッケージ＆データマイニング用の言語
  - 統計分析，モデリング，クラスタリング，自然言語処理，ルールマイニング，可視化など
  - 多数のライブラリ
- ランダムフォレストによるモデル化，予測の例

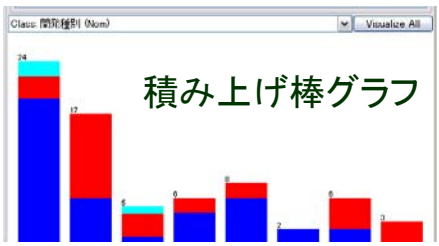
```
library(randomForest)
fitdata = read.table("学習データ.csv“, header = TRUE, sep = ",", quote="¥"",
dec=".", fill = TRUE)
testdata = read.table("評価データ.csv", header = TRUE, sep = ",", quote="¥"",
dec=".", fill = TRUE)

model <- randomForest(BUG~., data=fitdata)
result <- predict(model, testdata, type="response")
```

分析ツール

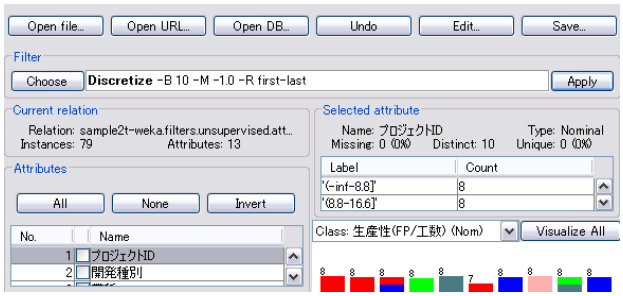
- WEKA http://www.cs.waikato.ac.nz/ml/weka/
  - 定番データマイニング＆可視化ツール

散布図マトリクス

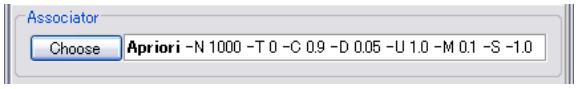




データの加工（名義尺度への変換）

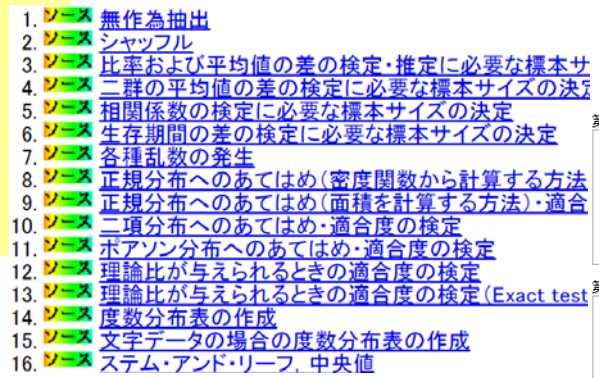


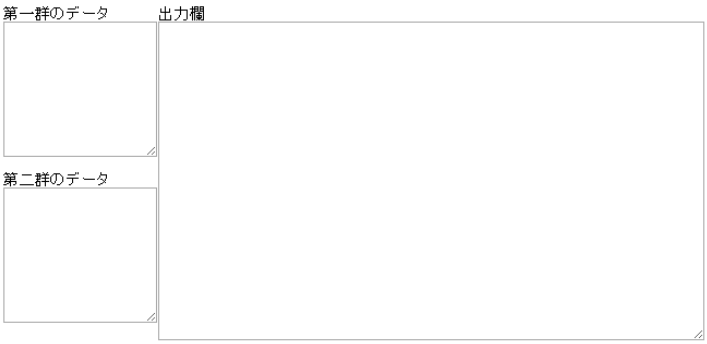
相関ルールマイニング



分析ツール

- 群馬大学の青木繁伸先生のWebサイト[1]
  - Web上の統計分析ツール
  - 統計学自習ノート，用語辞典
  - Rの使い方なども





[1] http://aoki2.si.gunma-u.ac.jp/

分析には気合いも必要

- データのクリーニング
  - 例：Mining Challengeの Stack Overflowデータから，コードクローンに関する話題を抽出して分析した[1]
    - キーワード検索の後，1654件を精査し，925件を抽出した．
- データの分析
  - 2100のコードコメントを分類した[2]
  - 約800のコメントについてEmotionをタグ付けした[3]

- ただし，システマティックに（2名以上で合意を得る等）

[1] E. Choi, R.G. Kula, N. Yoshida, K. Inoue, What do practitioners ask about code clone? A preliminary investigation of stack overflow, IWSC2015.
[1] Y. Padioleau, L. Tau, Y. Zhou, Listening to programmer – taxonomies and characteristics of comments in operating system code, ICSE2009.
[2] A. Murgia, P. Tourani, B. Adams, M. Ortu, Do developers feel emotions? an exploratory analysis of emotions in software artifacts, MSR2011.

Analyze This!

Analyze this!

- Andrew BegelとThomas Zimmermannが，1000名以上のMicrosoft社員に，MSRとして分析して欲しいテーマを尋ねた結果をまとめたもの[1]
  - 質問１：MSRの専門家に尋ねたい5つの質問を列挙せよ.
  - 質問２：質問１の結果をレーティングしてください.

[1] A. Begel T. Zimmermann, Analyze this! 145 questions for data scientists in software engineering, ICSE2014.
[2] https://www.microsoft.com/en-us/research/publication/analyze-this-145-questions-for-data-scientists-in-software-engineering/

Analyze this! 質問の例

- バグ計測
  - バグは開発のどの段階で混入し，検出されたのか．またどの段階で検出すべきだったのか？
  - Hotfixとして提供しなければならないようなバグが混入するとき，開発者はどのような種類のミスを犯しているのか？（例えば，date/timeのミス，メモリ確保のミスなどがあるのか？）
- 開発
  - プログラムに分析用コードを追加して役立つ場合とそうでない場合（既知事実しか得られない）の違いをどう見極める？
  - コードコメントは他者がコードを理解するのに役立つと言われているが，果たして真実なのか？
  - 「Many eyes make bugs shallow（多くの人間がいれば全てのバグを発見できる）」と言われているが，損益分岐点は？

Analyze this! High Rank Questions

- 上位ランクの質問

1. How do users typically use my application?
2. What parts of software products are most used and/or loved by customers?
3. How effective are the quality gates we run at checkin?
4. How can we improve collaboration and sharing between teams?
5. . . .

マイニングの観点

マイニングの観点

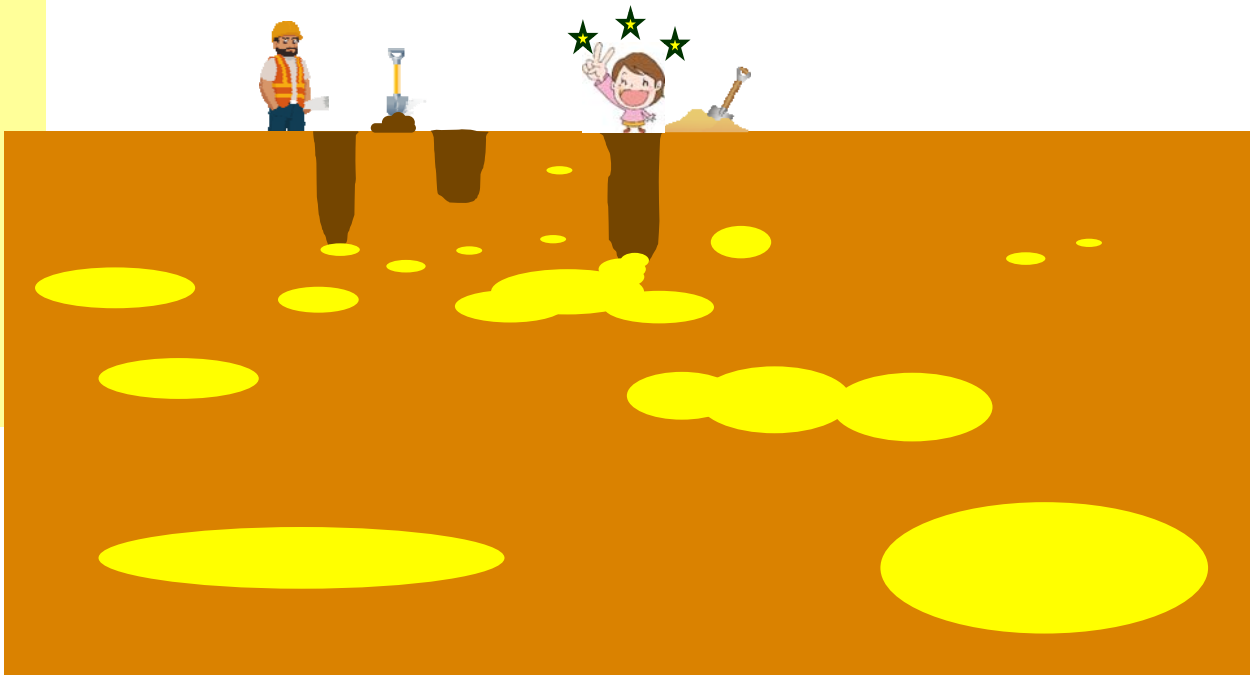
- 分析
  - 現状把握（例：ライブラリの使用），モデル化
  - 仮説，リサーチクエスチョン（〇〇に△△が影響する）の検証
  - メトリクスの定義，可視化
  - 良いもの vs 悪いもの（例：受理される／されないパッチ）
  - 自動分類（プログラム，バグ，クラッシュログ，．．．）
- 予測，判別
  - 題材，目的変数，コンテキスト，予測モデル，精度評価，予測前／後のアクション
- Replicated Study
  - データを変えて，規模を変えて，・・・
- ・・・

参考書

リポジトリマイニングの参考書

- Tim Menzies, Laurie Williams, Thomas Zimmermann, Perspectives on Data Science for Software Engineering, Morgan Kaufmann, 2016.
- Christian Bird, Tim Menzies, Thomas Zimmermann, The Art and Science of Analyzing Software Data, Morgan Kaufmann, 2015.
- Tim Menzies, Ekrem Kocaguneli, Burak Turhan, Leandro Minku, Fayola Peters, Sharing Data and Models in Software Engineering, Morgan Kaufmann, 2014.

リポジトリマイニングのすすめ



リポジトリマイニングのすすめ

- 大きな鉱脈はまだまだありそう.
- 小さな鉱脈は無数にありそう.
- 1プロジェクトのデータから始められる.
  - Eclipse, Firefoxが多い.
- 様々なランクの会議
  - トップ：ICSE, FSE
  - 中堅：MSR, ICSME, SANER, ESE
  - エントリー：CHASE, SWAN, IWESEP, MSR Challenge

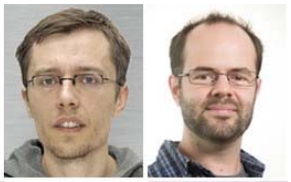
Let's Mining Software Repositories!

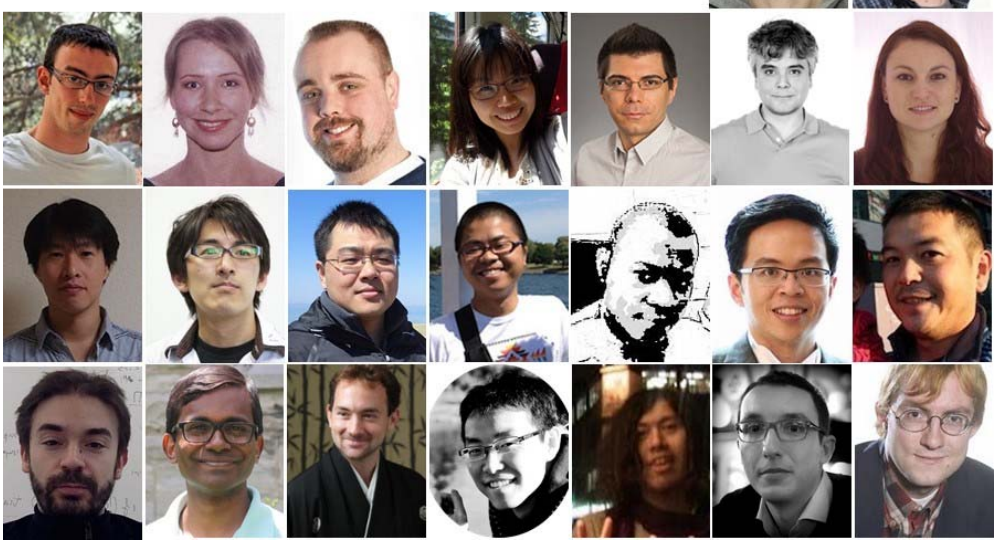
日本発のMSR関連国際会議(IWESEP)

- 国際ワークショップ IEEE IWESEP (International Workshop on Empirical Software Engineering in Practice)
  - エンピリカルソフトウェアに関する国際会議. 2016年で7回目.
  - 2016年, IEEEスポンサーシップを取得.
  - 国内外の実務者・研究者約50人～70人が参加
  - 若手研究者による会議運営
  - PCチェアに国際的に活躍している若手研究者を招へい

日本発のMSR関連国際会議(IWESEP2016)

- General Chair: Hata Hideaki (NAIST)
- ProgramCo-Chairs:
  - Shinpei Hayashi (TITech)
  - Shane McIntosh (McGill U.)





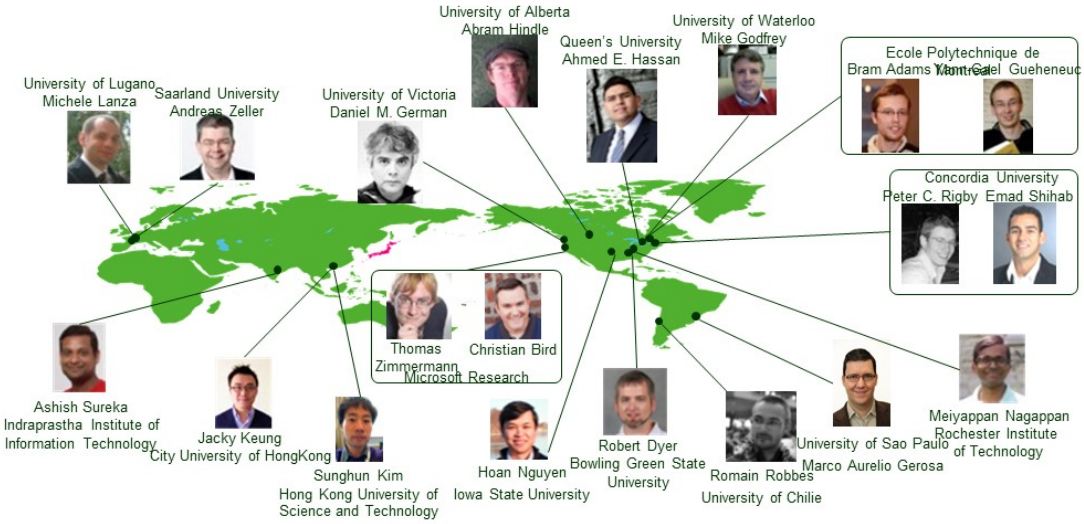
日本発のMSR関連国際会議(IWESEP2017)

- IWESEP2017　https://iwesep2017.github.io/

  - 近々，CFPがアナウンスされる予定

日本発のMSR関連国際会議

- MSR ASIA Summit　http://msrsummit2015.se-naist.jp/
  - MSRのトップ研究者を多数招聘し，ホットトピックをライトニングトークで紹介する1dayイベント（2013, 2014, 2015に開催）
  - 国内外から約50-70名が参加
  - オーガナイザ：亀井靖高（九州大），伊原　彰紀（NAIST）



MSR関連の国内ワークショップ

- ソフトウェア工学の基礎ワークショップ
  http://fose.jssst.or.jp/fose2016/
  - MSRに限定しない．ソフトウェア科学・工学全般
  - 参加者100名超．2016年で32回目．2泊3日．
  - 今年は，香川県ことひら温泉にて12月1～3日に開催されます．

- ソフトウェアエンジニアリングシンポジウム，およびその併設ワークショップ　http://ses.sigse.jp/2016/
  - 例年8月末 or 9月頭に東京にて開催
  - WSは，ソフトウェア開発データの分析（という名称が多い）
- ウィンターワークショップ　http://wws.sigse.jp/2017/
  - 例年1月末 or 2月頭に1泊2日