

Graph-Based Analysis and Prediction for Software Evolution

Pamela Bhattacharya

Marios Iliofotou

Iulian Neamtii

Michalis Faloutsos

概要

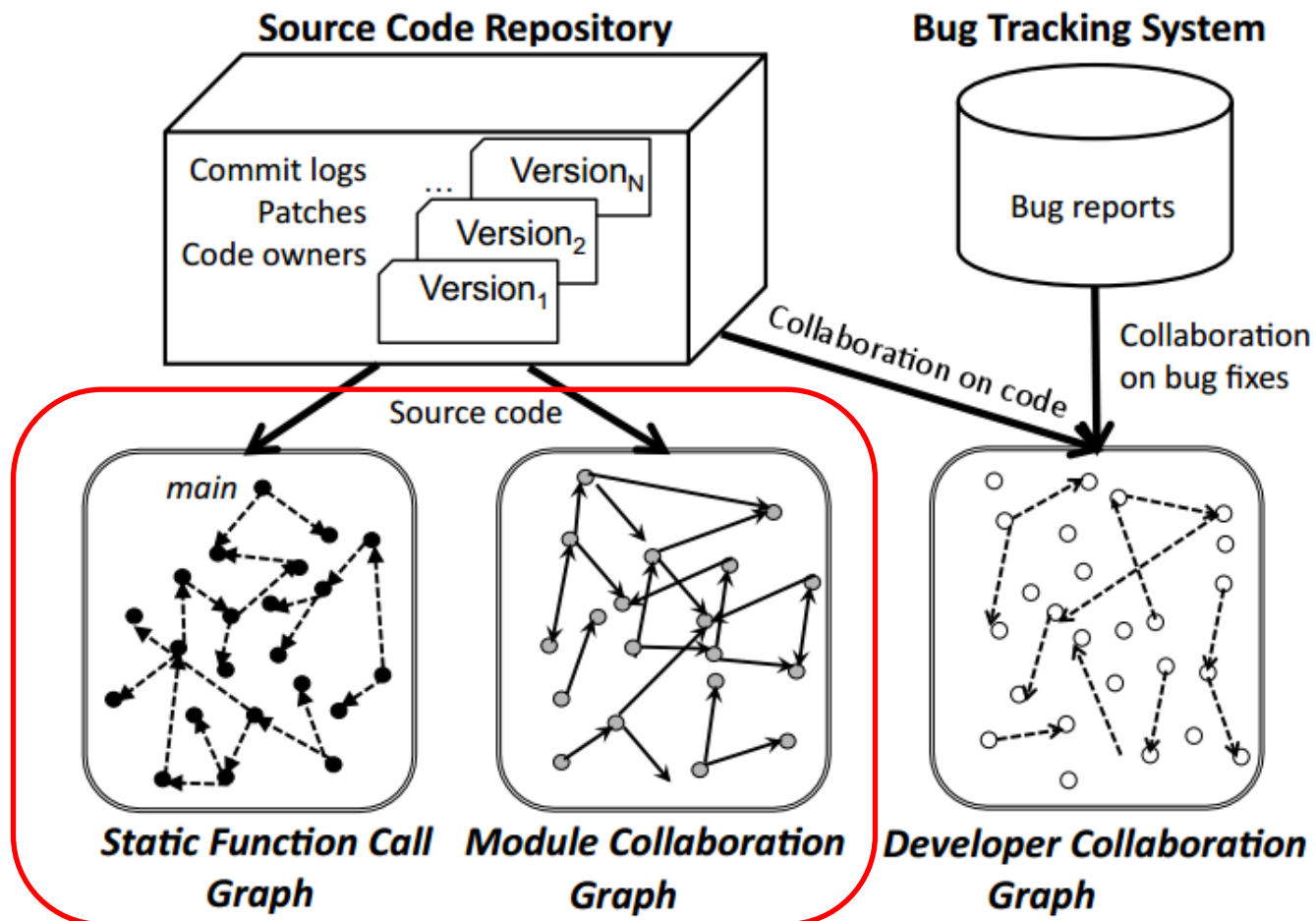
- 彼らの主張
 - 進化を分析するにはネットワーク分析が1番！
 - グラフに関するメトリクスの有効性を示した
- 彼らが分析したソフトウェア進化
 - グラフメトリクスを用いて以下のことができると示した
 - バグの重要度の見積もり
 - メンテナンスにかかる工数の予測
 - バグを含みやすいリリースの予測

背景

- 複雑な構造を分析するためにネットワーク分析
 - ウェブのトラフィック構造
 - コミュニケーション などなど
- グラフ理論でソフトウェア開発を支援できないか？
 - 関数のコールグラフ, モジュール間の参照グラフを使って3つの調査
 - 開発者間の関係グラフ2種を使って1つの調査

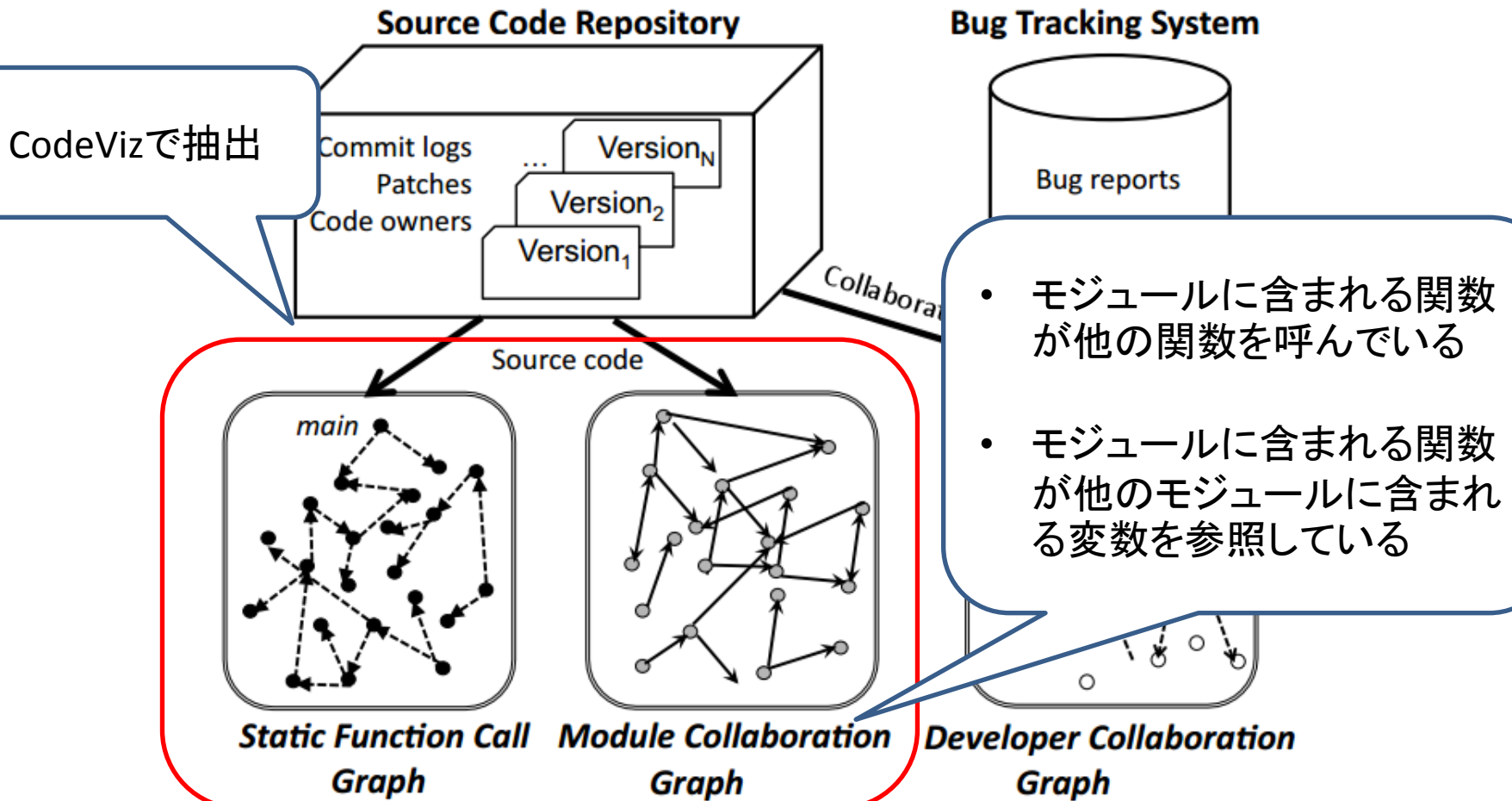
グラフの作成

- ソースコードベースのグラフ



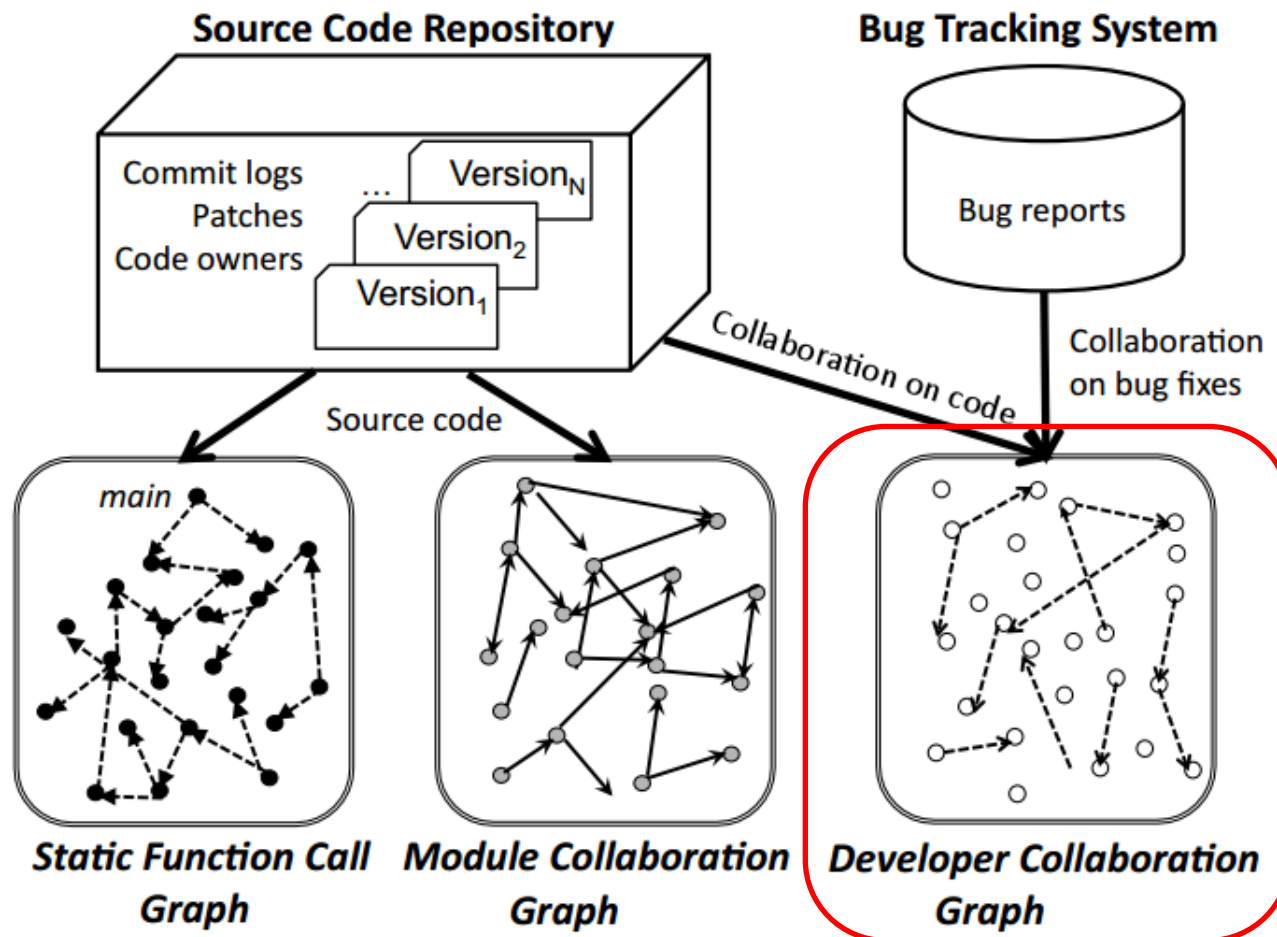
グラフの作成

- ソースコードベースのグラフ



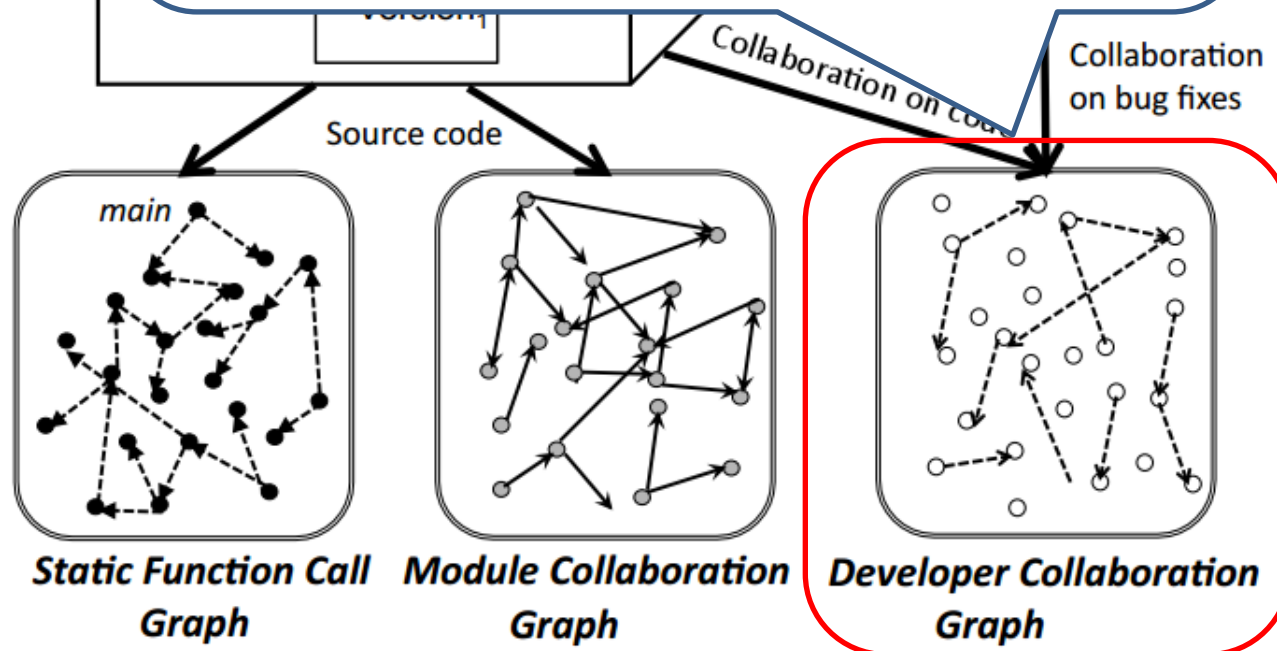
グラフの作成

- 開発者のコラボレーションのグラフ



グラフの作成

- 開発者の2種類のグラフ
 - バグ管理システムの開発者
 - 再割当のグラフ
 - バージョン管理システムの開発者
 - 同じファイルを変更した開発者



対象とするプロジェクト

Application	Time span	Re-lea-ses	Lan-guage	Size (kLOC)		Use
				first release	last release	
Firefox	1998-2010	92	C,C++	1,976	3,780	↗, 1, 2, 3
Blender	2001-2009	28	C,C++	253	1,144	↗, 1, 2
VLC	1998-2009	83	C,C++	144	293	↗, 1, 2
MySQL	2000-2009	13	C,C++	815	991	↗, 1, 2
Samba	1993-2009	78	C	5	1,045	↗, 1
Bind	2000-2009	171	C	169	321	↗
Sendmail	1993-2009	55	C	25	87	↗
OpenSSH	1999-2009	77	C	12	52	↗, 1
SQLite	2000-2009	169	C	17	65	↗
Vsftpd	2001-2009	59	C	6	15	↗
Eclipse	2001-2010	27	Java	828	1,903	3

グラフに対するメトリクス

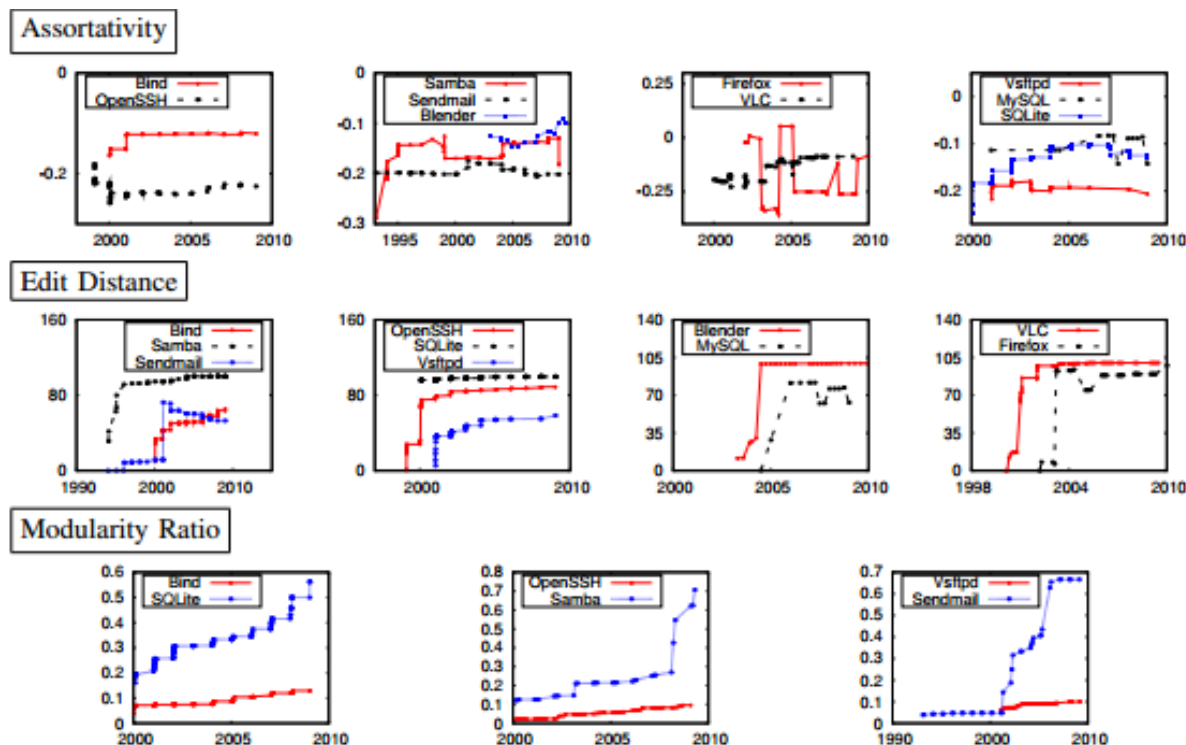
- Graph diameter
 - グラフの任意の2頂点間での最短経路長の最大値
- Assortativity
 - 各辺が接続している2頂点が同じぐらいの次数を持っているかどうかを判定する値
- Number of nodes in cycles
 - グラフ内で、有向辺で循環的に接続された頂点の数
- Edit Distance
 - 値が大きいほどシステム構造が大きく変更されている

頂点に対するメトリクス

- Clustering Coefficient
 - ある頂点 u に接続されている頂点が, 互いに接続されている確率 (密結合なグラフで値が高い)
- Node Rank
 - Page Rank 系メトリクス. 多数のノードから直接・間接的に参照されているノードほど値が高い
- Modularity Ratio
 - モジュール間よりもモジュール内の呼び出し・変数アクセスが多いほど値が高い

結果1

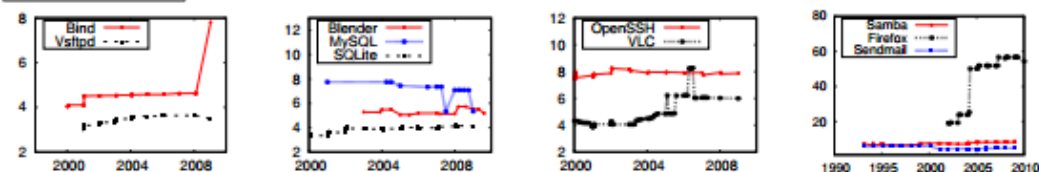
- 関数のコールグラフに関するメトリクスは、ソフトウェアが違っていても似たような値の変動をする



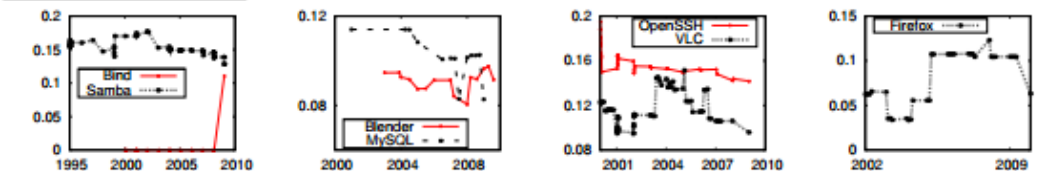
結果1

- 関数のコールグラフに関するメトリクスは、ソフトウェアが違っていても似たような値の変動をする

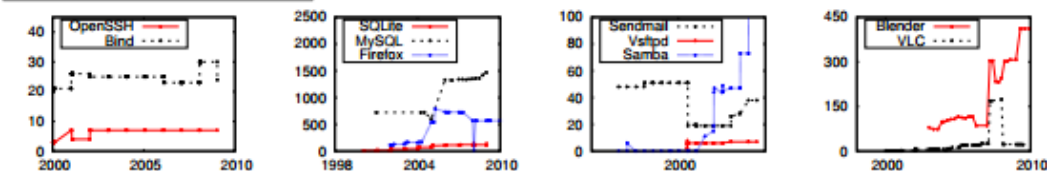
Average Degree



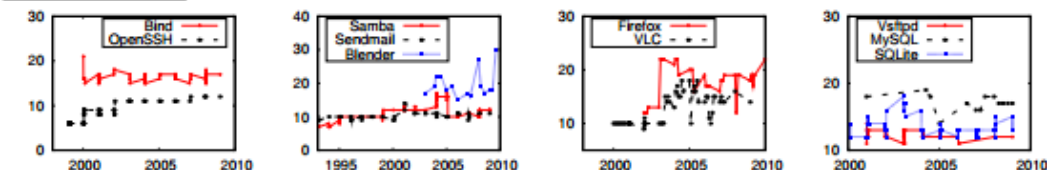
Clustering Coefficient



Number of Nodes in Cycles



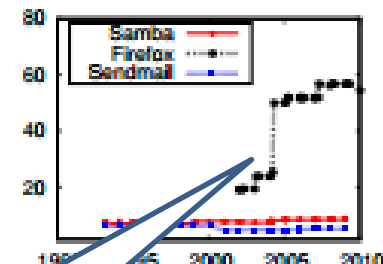
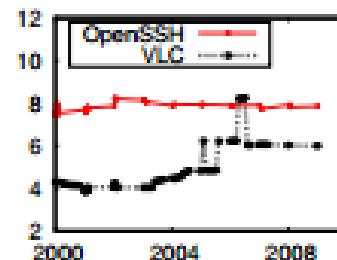
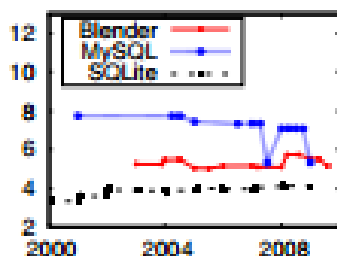
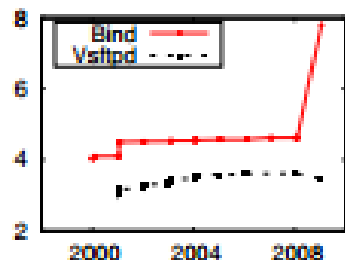
Graph Diameter



結果1

- 関数のコールグラフに関するメトリクスは、ソフトウェアが違ってても似たような値の変動をする

Average Degree



アーキテクチャが
大幅に変更された

メトリクスの突然の変化はシステム構造の大きな変更に対応する

結果2

- 関数・モジュールは多くの関数・モジュールから参照されているものほど発生したバグの重要度が高い

Program	NodeRank	In degree	Out degree
Blender	0.79	-0.04	-0.0008
VLC	0.82	0.21	-0.11
MySQL	0.73	-0.20	-0.24
Samba	0.78	-0.02	-0.10
OpenSSH	0.65	-0.22	-0.19
Firefox	0.704	-0.17	-0.38

Table V

CORRELATION OF *BugSeverity* WITH OTHER METRICS FOR TOP 1%
NodeRank MODULES (P-VALUE = 0.01).

結果2

- 関数・モジュールは多くの関数・モジュールから参照されているものほど発生したバグの重要度が高い

Program	NodeRank	In degree	Out degree
Blender	0.79	-0.04	-0.0008
VLC	0.82	0.21	-0.11
MySQL	0.73	-0.20	-0.24
Samba	0.78	-0.02	-0.10
OpenSSH	0.65	-0.22	-0.19
Firefox	0.704	-0.17	-0.38

Table V

CORRELATION OF *BugSeverity* WITH OTHER METRICS FOR TOP 1%

NodeRankはバグの重要度予測に効果的なメトリクスである

結果3

- ModularityRatioの値が高いほど、そのモジュールのメンテナンスにかかる工数は小さい



Figure 4. Change in *ModularityRatio* with change in *Effort*; *x*-axis represents time.

Program	Lag	F-prob
Blender	1	0.0000015
VLC	3	0.3673
Firefox	1	0.00056

Table VI

GRANGER CAUSALITY TEST RESULTS FOR *H2*.

結果3

- ModularityRatioの値が高いほど、そのモジュールのメンテナンスにかかる工数は小さい

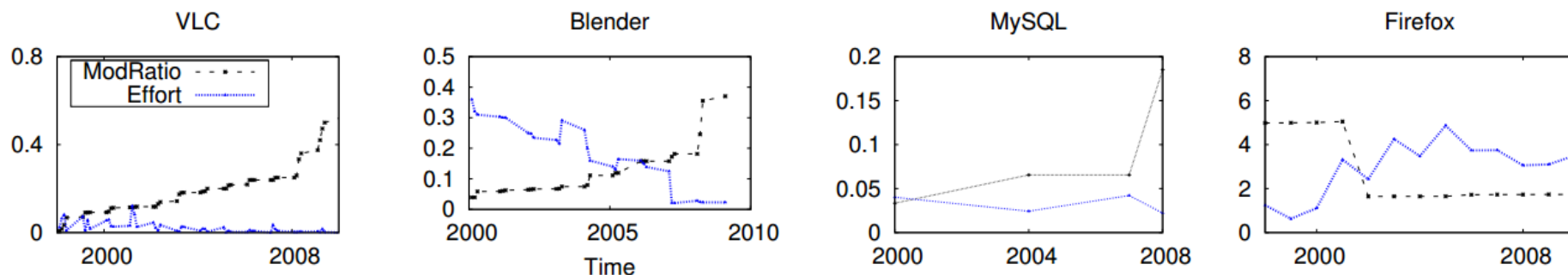


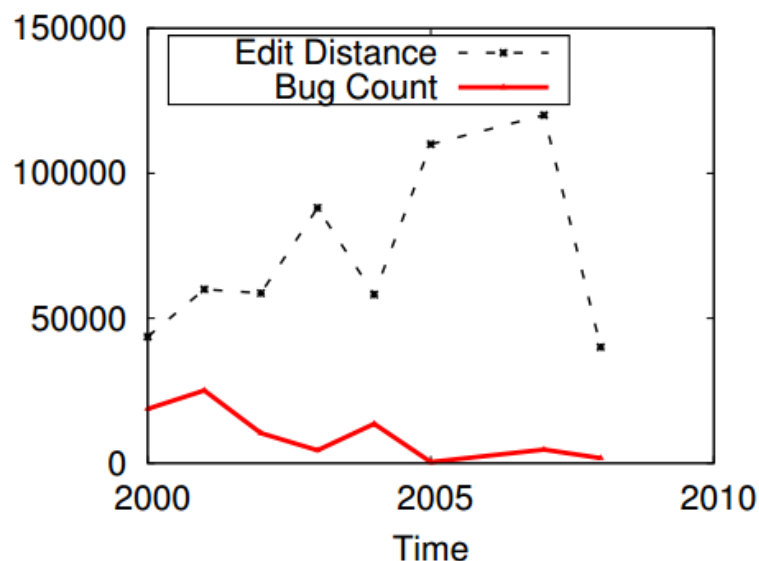
Figure 4. Change in *ModularityRatio* with change in *Effort*; *x*-axis represents time.

Program	Lag	F-prob
Blender	1	0.0000015
VLC	3	0.3673

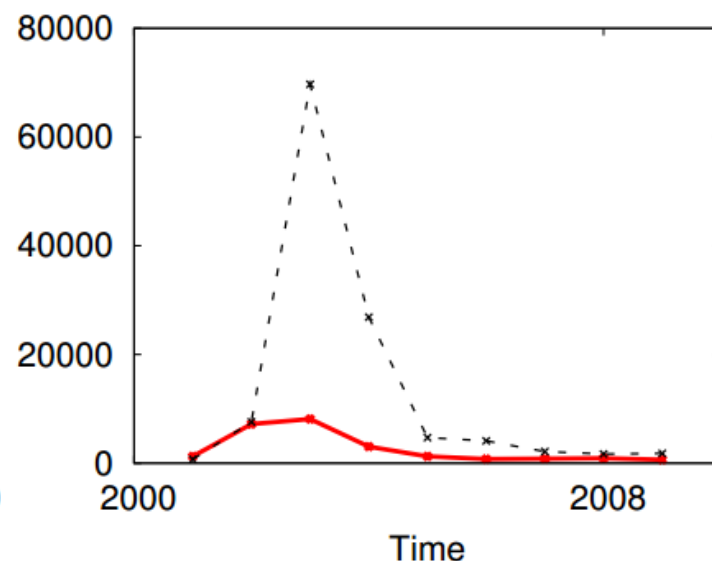
ModularityRatioによって次リリースの工数が決まる(Blender, Firefox)

結果4

- 開発者間でのバグ対応の協力関係グラフ(1年単位)が変化するほど、そのシステムはバグが多い



(a) Eclipse, correlation = 0.504



(b) Firefox, correlation = 0.759

Figure 5. Change in collaboration graph *Edit Distance* v. *Defect Count*.

まとめ

- グラフに関するメトリクスの用途を4つ示した
 - 関数のコールグラフに関するメトリクスは、ソフトウェアが違っても似たような値の変動をする
 - 関数・モジュールは、多くの関数・モジュールから参照されているものほど、そこで発生したバグの重要度が高い
 - モジュール性を表現する ModularityRatio メトリクスが高いほど、そのモジュールのメンテナンスにかかる工数は小さい
 - 開発者間でのバグ対応の協力関係グラフ(1年単位)が変化するほど、そのシステムはバグが多い

所感

- やっていることがかなり似ていると思った
- Granger Causality Test 使えそうだと思った
- グラフに関するメトリクスも使ってみようと思った
- メトリクスの値がなぜ著しく変化しているのか, など
しっかりとした分析がなされていると感じた