

中間成果物のサンプリングによる全体品質の推測にむけた分析

Analysis toward total quality estimation using sampling deliverables

三菱電機株式会社 通信機製作所 奈良先端科学技術大学院大学 情報科学研究科
 Mitsubishi Electric Co. Communication Networks Center
 Graduate School of Information Science Nara Institute of Science and Technology
 ○笠井 則充¹⁾ 森崎 修司²⁾ 松本 健一³⁾
 ○Norimitsu Kasai¹⁾ Shuji Morisaki²⁾ Ken-ichi Matsumoto³⁾

Abstract This paper analyzes the way of sampling software deliverables to estimate total quality of software products. The purpose of the analysis is to obtain the samples of deliverables efficiently, and low in cost. In the analysis, we use measured and derived metrics of business software consists of C++ and VB.NET. Functions/methods are chosen with average metrics in C++, lower in cyclomatic complexity per code, higher in calls per code in VB.NET to measure total quality of the software efficiently.

1. はじめに

近年ソフトウェアの複雑化，短納期化が進んでおり，生産性を維持しつつソフトウェア品質を確保する目的で外部委託によってこれに対応する傾向がある[1]．一方で，ソフトウェアの外部委託によって品質が低下するという報告があり[2]，納期と品質の両方を確保することが重要である．しかし現実には担当者が複数の業務を並行的にこなす，かつ外部委託を行って委託者の管理を行いながら納期を確保しつつ成果物の品質を確保することが必要なため，実現は容易ではない．外部委託により生産されるソフトウェア品質を低下させずに生産性を維持するには，委託先とうまくリスクを共有していくことが必要である．

ソフトウェア開発において外部委託を行う場合，要求分析やシステム設計等 V 字モデルの上位プロセスを委託元で行い，下部プロセスの一部または全部を外部委託することが多い．また，委託先で担当するプロセスの品質はプロセスごとの要素成果物に対するデザインレビューによって判定し，最終納品物であるソフトウェアの品質は委託作業が終了した後にシステム総合試験で判定する場合が多い．このとき最終納品物の品質が期待したものよりも低い場合は，手戻りなどの作業が発生し全体工程を圧迫することとなる．これを防ぐためシステム総合試験の前に委託元によって最終テストを実施し，バグを発見して対処することがあるが，全体工程の状況などにより必ずしも十分な対応ができていないとは言えない．

このことから，最終納品物で詳細に判定するのではなく，委託先の担当する各プロセスにおける中間時点での要素成果物（中間成果物）によってソフトウェアの最終品質を推測することで全体工程を圧迫させることなく早めに対応することが可能となる[3][4]．委託先の担当する各プロセスの品質は中間成果物に対するレビューや定期的な報告によって把握するが，報告は様々な事実や状況をまとめたものであることが多く，まとめる過程で情報が欠落したり詳細な事実を反映できなかったりすることがあり，必ずしも十分に機能できていない．しかし，すべての中間成果物を委託元で詳細に確認することはコスト面や時間的な制約で現実的ではないことが多い．

本研究では，委託先で開発作業が進行している段階で，中間成果物や成果物の代表的なものをサンプリングして抜き出し，検査，レビューすることにより最終的な全体品質を推測しようとするものである．本論文では委託先より納品されたソースコードに着目し，基準に従ってサンプルを抽出，抽出したソースコードを詳細に検査確認することで全体品質を推定することを考える．

1) 三菱電機株式会社 通信機製作所 〒661-0001 兵庫県尼崎市塚口本町 8-1-1
 Mitsubishi Electric Co. Communication Networks Center, 8-1-1 Tsukaguchi Hommachi,
 Amagasaki, Hyogo, Japan

1) 2) 3) 奈良先端科学技術大学院大学 情報科学研究科 〒630-0192 奈良県生駒市高山町 8916-5
 Nara Institute of Science and Technology 8916-5, Takayama, Ikoma Nara, Japan

以降、2章で実施概要を述べ、3章で対象としたソフトウェアと具体的評価手順と結果を述べ、4章で考察し、5章でまとめる。

2. 実施概要

2.1 概要

本研究では、ソフトウェアの品質を推測するためのサンプルとして中間成果物や成果物の代表的なものを抜き取ることを考える。この前提として人手による検査に先立って中間成果物、成果物の属性及び情報を利用する。本論文では、実際の商用開発のソースコードとそこから得られたソースコードメトリクスを対象として代表の抽出方法を実証的に評価し、考察を加える。この流れを図1に示す。うまく代表を選び出すには、どのような観点や属性が必要かを検討する。ソースコードメトリクスとしてサイクロマチック数、実行数、関数/メソッド呼出の回数、最大ネスト数を計測し、さらにこれらの導出メトリクスとして行数あたりのサイクロマチック数（サイクロマチック数密度）、行数あたりの関数/メソッド呼出数（呼出密度）を考え、これらとシステム総合試験で見つかったバグの改修回数、及びそのバグが与えるインパクト（リスク）との関係を散布図としてプロットし評価する。メトリクス値、導出メトリクス値によって改修回数やインパクトの特徴を得て、これに開発当事者による考察を加えることにより、今後、メトリクス値によって品質を代表するソースコードの選び方やその粒度を考察する。

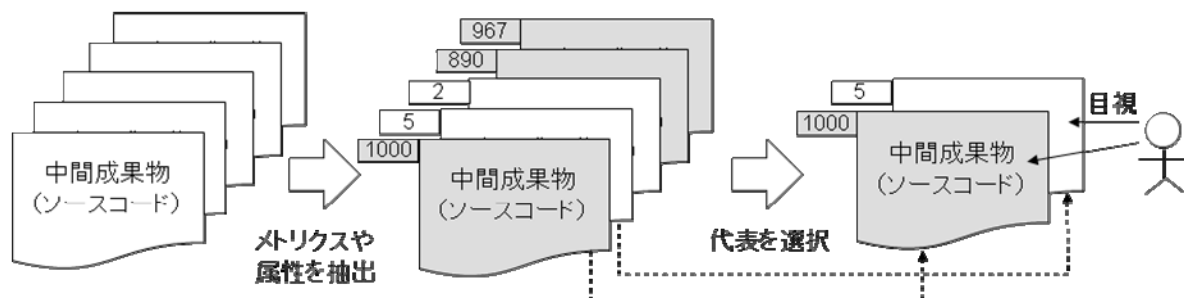


図1 中間成果物のメトリクスや属性による品質推定サンプルの抽出

3. 適用評価

3.1 対象

対象とするアプリケーションは、ある通信システムの設備機器を監視制御するものであり、Windows XP上で動作する3つのプロセスから構成される。これらはVisualStudio2005で開発し、2つはC++、1つはGUIを持つVB.NETで記述している。規模はそれぞれ実行数で6.7K、8.4K、38.7K[LOC]である。システム総合試験によって発見された不具合は不具合管理システムによって不具合発生日、対策方針、アプリケーションに改修を実施した場合は改修実施日や対応した版等、経緯や結果が管理される。結合試験を完了した初版リリースのソースコードより前記メトリクスを計測し、出荷判定に合格した最終第12版に至るまでに発見されたバグについて、関数/メソッド単位で改修を行った回数（改版数）、バグが与えるインパクト（リスク度合）を表1の基準で5段階評価した。関数の数は769、このうちバグが見つかったものは41であった。

表1 リスク度合（発見されたバグが与えるインパクト）と内容

リスク度合	内容
0	全く影響無し
1	運用上影響無し
2	一部の表示等に影響
3	機能の一部が不全
4	動作が停止

3.2 評価

対象とするアプリケーションは構成上2つの異なる言語で実装されている。VB.NET で作成したプロセスは監視した結果を画面上にグラフィカルに表示し、オペレータの操作を受付けて対象装置に対して制御を実施し結果を画面上に表示する。C++で作成したものはGUIは無く他装置とインターフェースを取り監視情報を取得してGUI側へ渡し、GUIからの制御操作の情報から対象装置に対して制御コマンドを送出する。これらの機能上の差異、特にGUIの有無が計測したメトリクスに影響を与えている可能性を考慮し、本評価においてはこれらを別々に分けてソースコードメトリクスを計測し評価する。VB.NET と C++で作成したモジュール間で計測したソースコードメトリクスの平均に有意な差があるかを検定した。結果を表2に示す。t検定の前提として実施したF検定の結果、最大ネスト以外は不等分散と判定された。

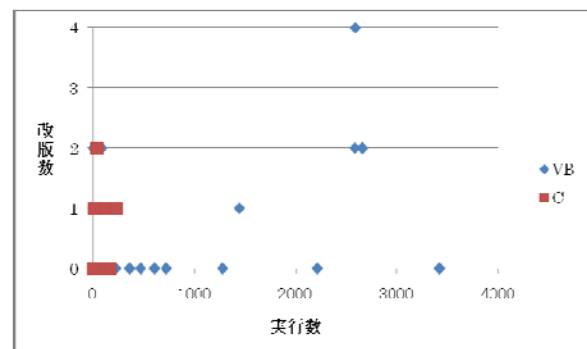
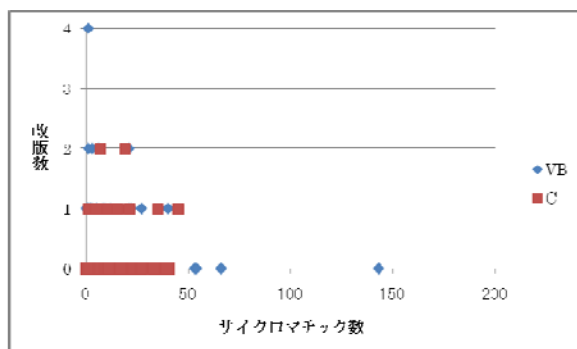
計測したソースコードメトリクスのうち、サイクロマチック数と関数呼出数については実行数に対する割合としてサイクロマチック数密度、関数呼出数密度を導出メトリクスとして考える。

表2 開発言語によるソフトウェアメトリクスのt検定結果

開発言語	モジュール数	サイクロマチック数			実行数		
		平均	分散	P 値	平均	分散	P 値
VB.NET	544	5.184	85.609	p<0.01	56.013	84295.3	p<0.05
C++	224	6.848	55.771		26.469	978.5	

開発言語	モジュール数	最大ネスト			関数呼出数		
		平均	分散	P 値	平均	分散	P 値
VB.NET	544	3.596	2.466	p<0.01	29.689	16149.7	p<0.01
C++	224	2.955	2.375		10.469	196.8	

図2は、モジュールに含まれる関数を1単位とし、関数毎に計測したメトリクス及び導出メトリクスを横軸で表し、縦軸は結合試験を完了した初版リリースから最終第12版に至るまでの改版数を表し、散布図としてプロットしたものである。1つの点が1関数を表し、ひし形の点はVB.NETの関数、四角の点はC++の関数を表す。図3は横軸については図2と同様であり、縦軸は発見したバグがこのアプリケーションに与えるインパクトとしてリスク度合を考え、これを5段階評価したものである。



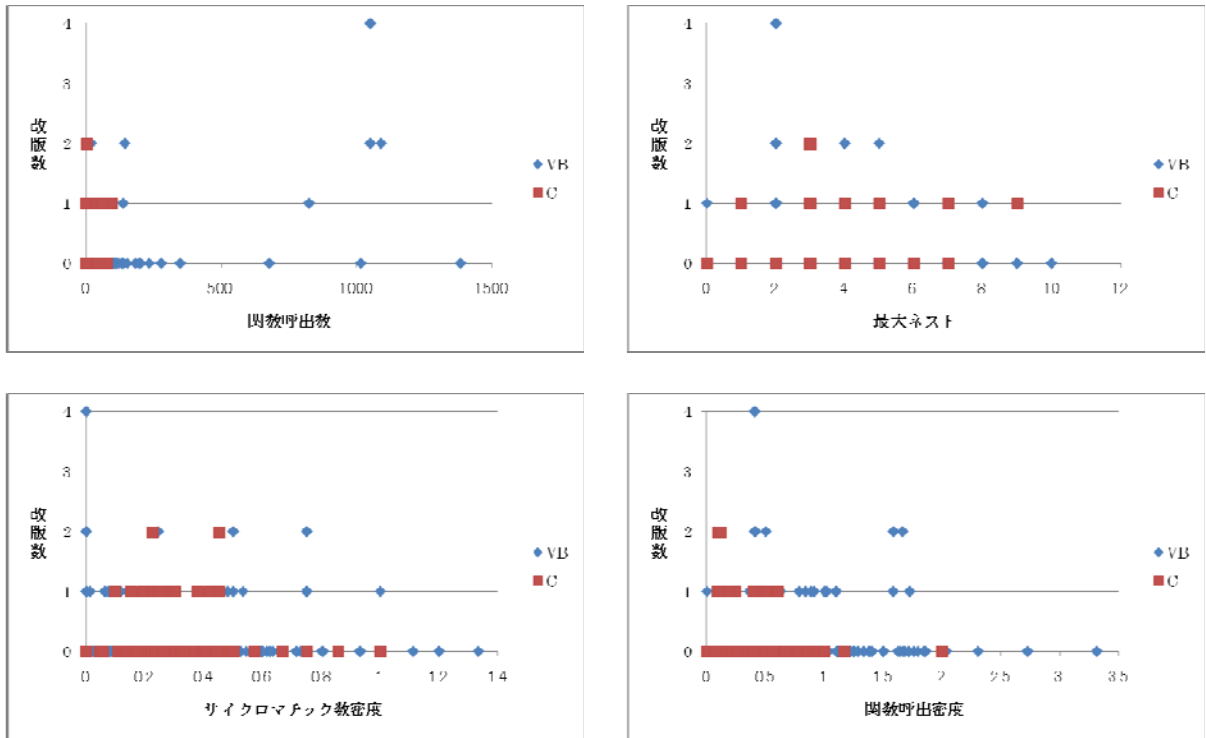
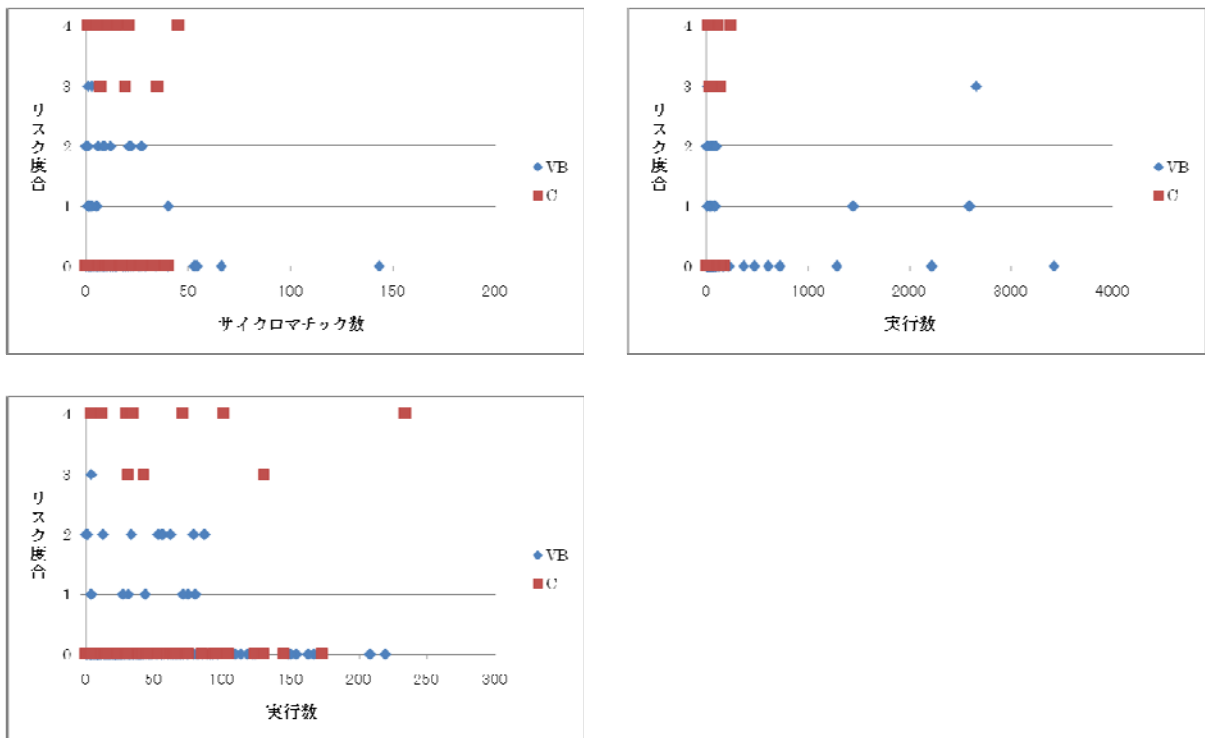


図2 ソースコードメトリクスと改版数



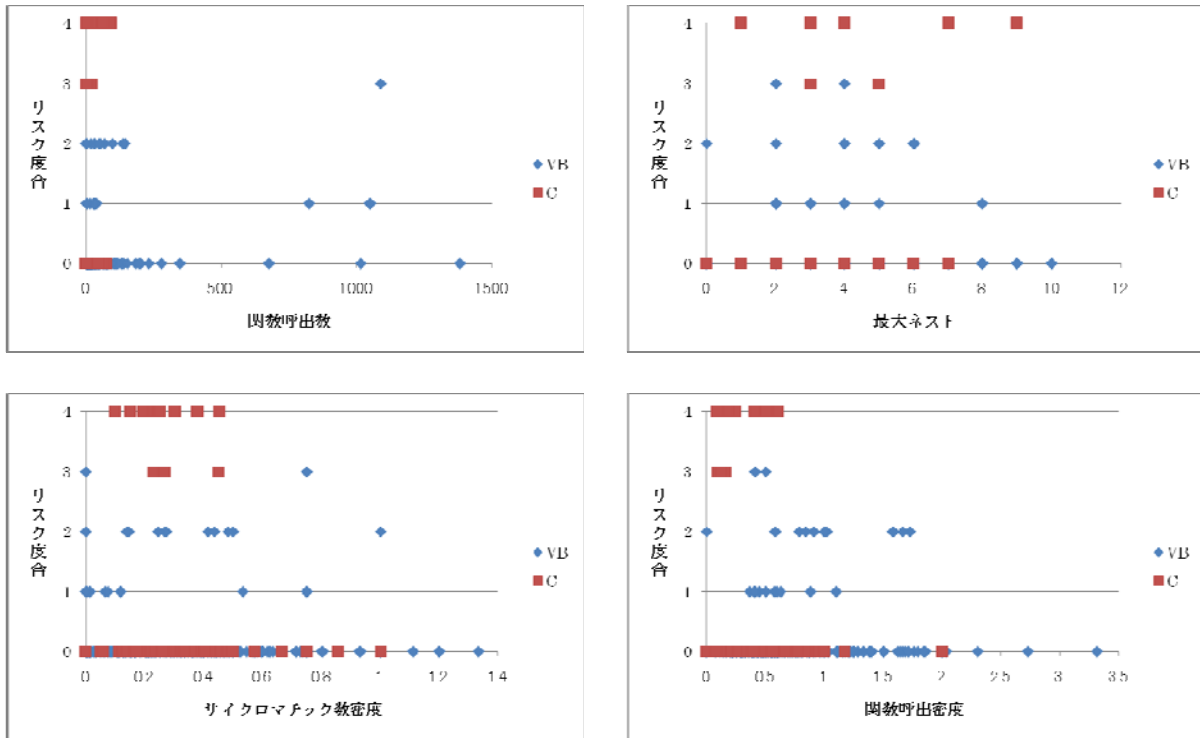


図3 ソースコードメトリクスとリスク度合

図2より、関数呼出し数が多いVB.NETの関数、実行行数の多いVB.NETの関数にはバグ修正に伴う改版が多く含まれていることがわかる。サンプリングの際に、この部分を含めれば、効果的な品質推測が容易になる可能性がある。また、図3からは、実行行数が大きいVB.NETの関数では、テストにおいて発見が見逃された場合にリスクの高いバグが出やすいことが推測される。C++で作成された関数においても実行行数が極端に大きなものでリスクの高いバグが発生しており、同様の傾向が推測される。最大ネスト数が多いC++の関数について、リスクの高いバグが出やすいと推測される。サンプリングの際、これらの部分を含めることで改修インパクトの大きなモジュールを効果的に抽出できる可能性がある。

また、初版モジュールにおいてVB.NETとC++それぞれのモジュールについて求めたサイクロマチック数の密度、関数/メソッド呼出し数の密度をシステム総合試験で発見したバグ対応による改修の有無で分類し、これらの平均値に有意差があるかをt検定した。結果を表3に示す。計測したメトリクスのうち、開発言語がVB.NETの場合は、サイクロマチック数密度の平均値は改修を行ったモジュールが有意に低く、関数/メソッド呼出し密度の平均値は改修を行ったモジュールが有意に高いことが判明した。このことから、VB.NETで開発したモジュールについてはサイクロマチック数密度は平均値より低く、かつ関数/メソッド呼出し密度は平均値より高いモジュールに改修モジュールがより含まれることが判明した。一方、C++のモジュールについてはいずれのメトリクスについても改修有無のモジュール間で平均値に有意差が認められなかった。

表3 サイクロマチック数密度、関数/メソッド呼出し密度の改修有無モジュールt検定結果

開発言語	改修有無	モジュール数	サイクロマチック数密度			関数/メソッド呼出し密度		
			平均	分散	P値	平均	分散	P値
VB.NET	無	518	0.51	0.095	p<0.01	0.580	0.214	p<0.05
	有	26	0.29	0.084		0.807	0.183	
C++	無	209	0.33	0.038	n.s	0.432	0.077	n.s
	有	15	0.29	0.014		0.321	0.042	

4. 考察

今回の適用例において、評価対象のアプリケーションに対しては GUI を持つ VB.NET とそうでない C++ のモジュールにはソフトウェア構成上の差異を仮定し評価を行った。差異の要因として、C++ で作成したモジュールはほぼすべてのソースコードが実装段階で制作担当者によって入力したものであるのに対し、VB.NET で作成したコードは Visual Studio2005 のコンパイラによって自動生成されたものを含む点が挙げられる。具体的には画面表示に関わるウインドウ、ラベル、ボタン等のコンポーネントは制作担当者が画面設計時にマウスで配置場所や大きさを決定するが、これらに該当するソースコードはコンパイラによって自動的に生成される。この箇所のソースコードは、画面内に配置したコンポーネントの大きさや色と言った属性（プロパティ）を羅列した構造となっており、ネストが 0、実行行数が比較的多い、といった特徴が有る。

表 4 対象ソースコードの特徴と評価結果から考えられるサンプリング箇所

	VB.NET	C++
サンプリング箇所の候補	<ul style="list-style-type: none"> ・ サイクロマチック密度が平均より低めの関数群 ・ 関数/メソッド呼出し密度が平均より高めの関数群 	全域からメトリクス値が均等となるよう選択

サンプリングにあたっては、表 4 に示すように、VB.NET の場合、サイクロマチック密度は全体平均より低め、関数/メソッド呼出し密度は全体平均より高めのモジュールを選び、C++ の場合は、全モジュールからこれらのメトリクス値が均等となるようにサンプリングすることで、より効率的に全体品質を推測できると考える。

対象のアプリケーションでは、VB.NET で主に GUI を、C++ で外部装置とのインターフェースや内部情報管理等の処理分担を行っている。バグの有ったモジュールの解析の結果、C++ モジュールでバグ発生時のリスク度合いが高かった。

今回の適用例を他のソースコードにそのまま適用するには更なる検討と議論が必要であるが、保守開発や派生開発など類似のソースコードを変更し、新しいバージョンとするような開発の場合には、過去のバージョンでの改版数とメトリクスの傾向を用いることにより、本適用例と同様の手順で効果的なサンプリングができる可能性がある。

5. まとめ

本研究では、ソフトウェアの全体品質の推測を行うにあたり中間成果物をサンプリングすることを考える。この過程として本論文ではソースコードに着目し、実際の商用ソフトウェアからソースコードメトリクスを計測し、作成した散布図を元に分析を行った。今回の適用例の分析結果から中間成果物の全体品質の推測に向けたサンプリング方法の傾向が示された。

今後は、ソースコードメトリクス以外の属性やソースコード以外を対象としたサンプリングを検討するとともに、統計的サンプリング手法の適用可否を検討する予定である。

参考文献

- [1] 角田 雅照, 門田 暁人, 宿久 洋, 菊地 奈穂美, 松本 健一, “外部委託率に着目したソフトウェアプロジェクトの生産性分析”, 信学技報, pp. 19-24 (2006-04)
- [2] 経済産業省 独立行政法人 情報処理推進機構, “2005年版組込みソフトウェア産業実態調査 報告書”, 2005年6月
- [3] 金村 一弘, 水野 修, 菊野 亨, 高木 徳生, 坂本 啓司, “レビュー作業の質に着目したソフトウェア最終品質の推定”, 電子情報通信学会技術研究報告. SS, ソフトウェアサイエンス, 99(683) pp.1-7 20000314
- [4] Thomas Thelin, Håkan Petersson, Per Runeson, Claes Wohlin, “Applying sampling to improve software inspections”, Journal of Systems and Software archive Volume 73, Issue 2 pp. 257-269 (October 2004)