特集 ソフトウェアレビュー／ソフトウェアインスペクションと欠陥予防の現在

2 ソフトウェアインスペクションの効果と効率

野中 誠　東洋大学経営学部

本稿では，ソフトウェアインスペクションにおける実務的な課題を 3 つ挙げ，これらの課題に対する取り組みについて説明する．3 つの課題とは，インスペクションの定量的管理，有効性と効率性の改善，アプリケーション領域知識の組織的活用である．それぞれについて，具体例や手法を挙げながら解説する．

ソフトウェアインスペクションの実務的課題

「ソフトウェアインスペクションを実施すると，ソフトウェア成果物に含まれる欠陥を効果的かつ効率的に発見できる」――このステートメントは，ソフトウェアエンジニアリング分野において繰り返し述べられてきた．インスペクションを標準プロセスにしている，あるいは導入を試みているソフトウェア組織は多く，ソフトウェア品質の向上に取り組んでいる実務者にとって，インスペクションは関心の高いテーマの 1 つである．

しかし，インスペクションの効果を十分に享受できているソフトウェア組織は，筆者が知るかぎり，必ずしも多くない．それは，CMMI（能力成熟度モデル統合）のレベル 1 相当の組織だけではなく，レベル 5 相当の組織であっても同様である．開発プロセスが組織内で標準化されておらず，開発プロセスのデータが適切に測定されていない組織がインスペクションを実践しても，場当たり的なインスペクションしか実施できない．一方，定量的管理の体制が整った組織であっても，アプリケーション領域の知識を組織内で共有できていなければ，インスペクションを通じた品質の作り込みは限定的になる．

インスペクションを効率的に実施し，その効果を十分に享受するためには，克服すべきいくつかの課題がある．ここでは，実務的な視点から次の 3 点の課題を挙げ，それぞれについて必要な取り組みの具体例を述べる．

(1) **定量的管理の実践**：事後のデータ分析に耐えられる精度で，インスペクションプロセスを測定する．インスペクション状態を把握するための適切な評価メトリクスを用いる．それぞれのメトリクスについて，過去の実績に基づいた妥当な目標値を設定する．目標値と実績値を比較してプロジェクトを制御する．測定結果に基づいて，インスペクションプロセスを継続的に改善する．

(2) **有効性と効率性の改善**：欠陥指摘が目的のはずが仕様説明に終始してしまったり，成果物の一部分に大半の時間を浪費したりすることのないよう，モデレータが中心となって理解・指摘フェーズ[☆1]の進行を制御する．適切なリーディング手法を組み合わせて適用し，欠陥の指摘漏れが生じないようにする．指摘漏れが生じた場合には，見逃した欠陥のそれぞれについて根本原因分析を行い，インスペクションのチェックリストに反映するなどして再発防止に努める．

(3) **アプリケーション領域知識の組織的活用**：ソフトウェア開発全般に言えることであるが，エンジニアリング技法と，アプリケーション領域知識の両方がなければ，品質の高いソフトウェアは開発できない．アプリケーション領域の専門知識を有したメンバをインスペクションに含める，経験の浅いメンバを参加させる，欠陥指摘に必要な暗黙知をチェックリストや標準プロセスとして形式知化するなどの取り組みが求められる．

以下，ここで指摘した 3 点の課題に関して，特に，インスペクションの効果と効率を定量的に把握する具体的な技法を中心に紹介する．

☆1 フェーズの定義は，本特集「1. ソフトウェアインスペクションの動向」を参照されたい．

メトリクス（単位）	定義
欠陥除去率（%）	除去欠陥数 ÷（除去欠陥数 ＋ 除去漏れ欠陥数）
欠陥除去規模密度（欠陥数／規模）	除去欠陥数 ÷ 成果物規模
インスペクション速度（規模／時間）	成果物規模 ÷ インスペクション所要時間
欠陥指摘工数密度（欠陥数／工数）	指摘欠陥数 ÷ 計画フェーズから収集フェーズまでの全工数

表 -1　主なインスペクション評価メトリクス

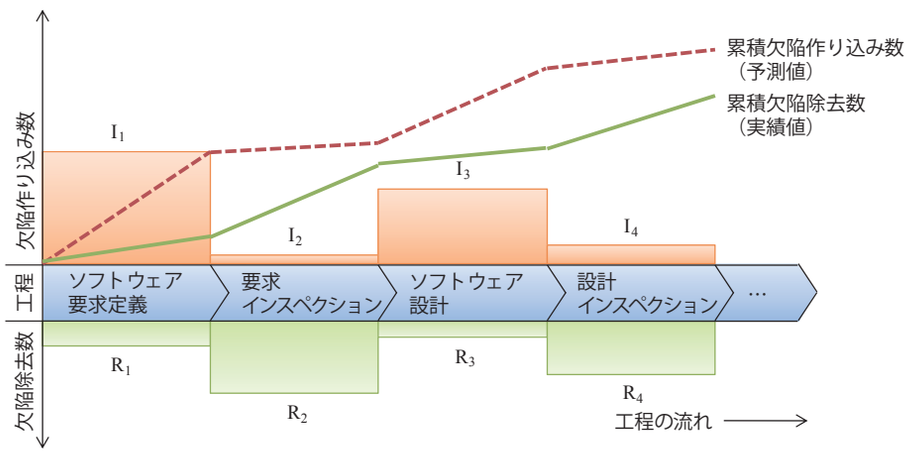

図 -1　欠陥作り込み・除去モデル

▶インスペクションの定量的管理

■インスペクション評価メトリクス

まずは，インスペクションの効果と効率を定量的に把握することが基本である．表 -1 に，本稿で紹介するインスペクション評価メトリクスを示す（その他のメトリクスについては，文献 1），2），5）などが参考になる）．これらのうち，欠陥除去率と欠陥除去規模密度は有効性，インスペクション速度と欠陥指摘工数密度は効率性に関するメトリクスに分類できる☆2．

欠陥除去率

欠陥除去率☆3 は，インスペクションで除去した欠陥数を，除去した欠陥数と除去し損ねた欠陥数☆4 の合計で割った値である．例として，図 -1 の棒グラフで示した欠陥作り込み・除去モデルにおけるインスペクションの欠陥除去率を考える．ソフトウェア要求定義で I_1 個の欠陥を作り込み，R_1 個の欠陥を除去したとする．また，要求インスペクションで R_2 個の欠陥を除去したが，インスペクションの過程で I_2 個の欠陥を新たに作り込んでしまったとする．このとき，要求インスペクションの欠陥除去率は $R_2 \div (\Sigma_{i=1}^{2} I_i - R_1)$ で与えられる．同様に，設計インスペクションの欠陥除去率は $R_4 \div (\Sigma_{i=1}^{4} I_i - \Sigma_{i=1}^{3} R_i)$ で与えられる．

欠陥除去率を算出するには，発見したすべての欠陥について，その欠陥がどの工程で作り込まれたのかを確認・記録しなければならない．図 -1 の例で，要求インスペクションが終わった時点での累積欠陥作り込み数と累積欠陥除去数の差に相当する部分は，要求インスペクション以降に発見された欠陥である．したがって，これらの欠陥が発見されたときに，ソフトウェア要求定義または要求インスペクションで作り込まれた欠陥であることが確認されなければ，要求インスペクションの欠陥除去率を正確に求められない☆5．

欠陥数としてカウントする範囲は，顧客から報告される欠陥も含まれる．したがって，欠陥除去率の値が確定するのは，ソフトウェア出荷後の一定期間（通常は 1 年）

☆2 欠陥除去規模密度と欠陥指摘工数密度は，標準的な名称があるわけではなく，文献によってさまざまな名称で呼ばれている．

☆3 欠陥除去率の定義は文献によって多少異なるが，本稿では，一般に DRE（Defect Removal Efficiency）と呼ばれるメトリクス[3] について言及している．インスペクションの理解・指摘フェーズにおける目的は，通常，欠陥の“除去”ではなく“指摘”であることから，このメトリクスを欠陥指摘率として定義してもよい．本稿では，指摘だけでなく除去まで含めたインスペクション全体の有効性指標という意味で，欠陥除去率という名称を用いている．

☆4 除去し損ねた欠陥として記録する範囲は，後述の通り，ソフトウェア出荷後 1 年までとすることが多い．

☆5 個々の欠陥について作り込み工程を記録するのは，実務では無理があると思われるかもしれない．しかし，欠陥の再発防止を本気で考えるならば，個々の欠陥について根本原因分析を行い，同じ欠陥を二度と見逃さない，または作り込まないようにプロセスを改善しなければならない．事実，このレベルでの根本原因分析を実施できているソフトウェア組織では，ソフトウェア品質が格段によい．

が過ぎた時点となる．そのため，このメトリクスは，プロジェクトの進行中におけるフィードバック（たとえば，工程移行の可否を判断する根拠とするなど）に用いるよりは，むしろ，プロジェクトの事後分析に用いられる．

ただし，累積欠陥作り込み数を十分な精度で予測できるのであれば，プロジェクト進行中でのフィードバックに用いることができる．図 -1 の線グラフで示したように，過去の実績，ソフトウェア規模，プロジェクト特性などに基づいて累積欠陥作り込み数の予測値が得られ，これに累積欠陥除去数の実績推移を重ねれば，欠陥除去率の予測値が得られる．この値を目標値と比較することで，プロジェクト管理へとフィードバックできる．

欠陥除去率は，インスペクションに限らず，テストの有効性評価にも用いられる重要なメトリクスである．ソフトウェアのライフサイクル全体におけるすべての欠陥除去工程について，このメトリクスを用いて欠陥除去の有効性を把握し，定量的管理を行うことが望ましい．

欠陥除去規模密度

欠陥除去規模密度は，インスペクションで除去できた欠陥数を，インスペクション対象の成果物規模で割った値である．欠陥除去規模密度は，欠陥除去率とは異なり，インスペクション終了時に値が確定する．したがって，インスペクション終了時に実績値と目標値を比べることで，工程移行の可否判断などに用いることができる．

欠陥除去規模密度が目標値の下方管理限界を下回る場合は，成果物の品質が平均レベルよりも優れているか，またはインスペクションで十分な量の欠陥を指摘できていない可能性が高い．この場合，インスペクションプロセスを点検し，適切なインスペクションが実施されたことを確認した上で，必要に応じて再度インスペクションを実施する必要がある．

逆に，欠陥除去規模密度が目標値の上方管理限界を上回る場合は，成果物の品質が平均レベルよりも劣っている可能性が高い．このような状況でインスペクションを実施しても十分な効果が期待できず，インスペクションへの投入工数が無駄になる可能性が高い．したがって，この傾向が早期に確認できた場合は，インスペクションを中断して成果物作成プロセスへと差し戻し，成果物の品質改善を促すのが望ましい．

インスペクション速度

インスペクション速度は，インスペクション対象の成果物規模を，理解・指摘フェーズにおける所要時間で割った値である．ここで用いる値は所要時間であって，インスペクション参加者の工数合計ではない点に留意されたい．

実績値が目標値の下方管理限界を下回る場合は，インスペクションの効率に問題があることを示唆しており，理解・指摘フェーズにおけるプロセスを見直す必要がある．逆に，上方管理限界を上回る場合には，インスペクションで十分な数の欠陥が指摘できていない可能性がある．この場合，有効性を表すメトリクスと合わせて傾向を把握する必要がある．

欠陥指摘工数密度

欠陥指摘工数密度は，理解・指摘フェーズにおいて指摘した欠陥数を，計画フェーズから収集フェーズまでに投入した全工数で割った値である．インスペクション速度とは異なり，欠陥指摘工数密度では，これらのフェーズに投入したメンバ全員の工数合計を用いている．

欠陥指摘工数密度を欠陥“除去”工数密度へと拡張し，計画フェーズからフォローアップフェーズに至るインスペクションプロセス全体の工数合計を測定してもよい．同様に，単体テスト，結合テスト，受入テストなどのテスト工程も測定し，インスペクションの欠陥除去工数密度と比較すると，インスペクションとテストのどちらの工程が効率的に欠陥を除去できているか比較できる．もちろん，インスペクションとテストでは発見または修正できる欠陥の性質に違いがあり，完全に等価な比較ができるわけではない．とはいえ，工程間での欠陥除去の効率性を評価するのには，十分に役立つ情報が得られる．典型的には，インスペクションの効率はテストに比べて 2 ～ 10 倍，文献によっては 30 倍良い．

■基本メトリクス

以上で紹介したメトリクスは，いずれも，欠陥，成果物の規模，時間，および工数という変数の組合せで算出されていた．これらは，ソフトウェア開発におけるエンティティ（成果物やプロセスなど）の属性を直接的に表す変数である．これらは，一般に，基本メトリクス（または基本測定量）と呼ばれる．一方，これまでに紹介したインスペクション評価メトリクスは，基本メトリクスを組み合わせて得られることから，一般に，導出メトリクス（導出測定量）と呼ばれる．本稿で示した基本メトリクスと導出メトリクスの対応関係を図 -2 に示す．

本稿で紹介したインスペクション評価メトリクスの数値が意味を持つためには，その前提として，基本メトリクスの測定方法が一貫している必要がある．すなわち，開発プロセスが標準化され安定していること，成果物の仕様書には規定フォーマットが用いられていること，成果物規模の測定方法について合意が得られていること，欠陥の定義が明確になっていることなどが必要である．

たとえば，欠陥（または障害）の測定と故障の測定は，

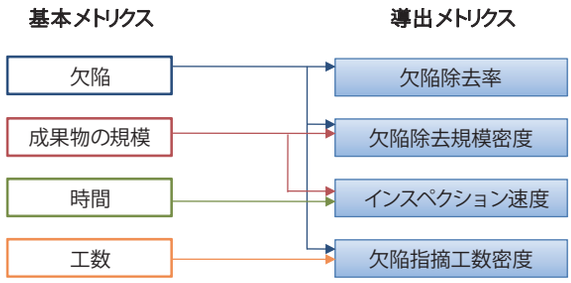

図 -2　基本メトリクスと導出メトリクスの対応関係

明確に区別しなければならない．欠陥（defect）または障害（fault）とは，成果物の中にある不適切な部分を意味する．プログラム実行時に欠陥部分が実行され，期待とは異なる出力が得られた場合，この状態を故障（failure）と呼ぶ．インスペクションは静的解析技術であるため，故障のカウントは原理的に不可能である．したがって欠陥（障害）をカウントする．一方，テストは，故障を検出するのが主な目的である．故障が検出されると，1 件の故障に対して 1 枚の“バグ票”が起票されることが，実務の場面でしばしばある．管理の目的にもよるが，インスペクションの効果と効率を評価するという観点からは，故障と欠陥（障害）の混同は避けねばならない．テストにおける測定の単位が故障ではなく欠陥（障害）となるよう，組織内で基準を設けておく必要がある．

また，欠陥としてカウントする対象も明確にしておかなければならない．一般には，主要欠陥とマイナー欠陥に分けて，主要欠陥のみをインスペクション評価メトリクスに用いる．

このように，基本メトリクスの定義を明確化すること，導出メトリクスの算出に用いる基本メトリクスの定義を明確化することが必要である．

■ メトリクスの目標値

表 -2 に，本稿で紹介したインスペクション評価メトリクスの目標値例を，要求，設計，およびコードインスペクションのそれぞれについて示す．これらの数値は TSP（Team Software Process）[2] に示された数値に基づいた例示であり，絶対的な基準としての数値ではない．特に，欠陥指摘工数密度の数値はインスペクション参加人数によって値が異なる．

表 -2 に示した数値は参考程度にとどめ，実際には，組織内でデータを蓄積し，実績に裏付けされた妥当な目標値を設定することが求められる．

▶インスペクションの有効性・効率性の改善

■ プロジェクト全体の工数削減：改善効果の試算

インスペクションの定量的管理が行えるようになると，プロセス改善の効果を試算したり，プロジェクトの品質計画を実績値に基づいて立案できるようになる．インスペクションというプラクティスがソフトウェア開発において推奨される理由の 1 つは，欠陥の早期発見による手戻り工数の削減である．ここでは，インスペクションの改善効果が品質コストに与える影響の試算を示す．

例として，ソースコード行数が 50KLOC のソフトウェアを新規に開発する場合を考える．このプロジェクトの工数を，工数見積りモデル COCOMO II を用いて見積もった結果，230.8 人月と算出できる．このプロジェクトにおける手戻り工数の比率を全体で 30% と仮定し，その比率は後工程になるほど高くなるとする．手戻り工数比率に関する調査結果として客観的なものは多くはないが，仕様変更への対応，欠陥修正，再作業などを手戻りと考えると，30% という値は現実的な推定値であろう．

開発工程を，要求（インスペクションを含む），設計（同様），実装，および統合の 4 工程とし，各工程の稼働工数（手戻りを除いた工数），手戻り工数，およびそれらの合計，ならびに手戻りの原因工程別の手戻り工数比率を表 -3 の通りに仮定する．たとえば，設計工程の場合，合計で 36.9 人月の稼働工数に対して手戻り工数が 9.2 人月であり，その手戻り工数のうち 55% は要求工程および要求インスペクションで見逃した欠陥が原因であることを意味している．また，残りの 45% は，設計工程内で作り込んだ欠陥が原因で生じた手戻り工数とする．なお，原因工程別の手戻り工数比率の値は，IPA/SEC

メトリクス	要求インスペクション	設計インスペクション	コードインスペクション
欠陥除去率	70%	70%	70%
欠陥除去規模密度	1.5 欠陥／ページ	1.0 欠陥／ページ	5.0 欠陥／ KLOC
インスペクション速度	2 ページ／時間	5 ページ／時間	200LOC ／時間
欠陥指摘工数密度	0.5 欠陥／人時	0.5 欠陥／人時	1.0 欠陥／人時

注）欠陥指摘工数密度の値は，インスペクション参加者が 5 名程度の場合として算出している．

表 -2　インスペクション評価メトリクスの目標値例（文献 2）を参考に例示）

工程	初期見積工数（人月）			原因工程別の手戻り工数比率（%）				改善後工数（人月）		
	稼働	手戻り	合計	要求	設計	実装	統合	稼働	手戻り	合計
要求	19.6	3.5	23.1	100				21.6	3.5	25.0
設計	36.9	9.2	46.2	55	45			40.6	5.4	46.0
実装	60.0	32.3	92.3	35	30	35		60.0	17.4	77.4
統合	45.0	24.2	69.2	20	25	25	30	45.0	16.6	61.6
合計	161.6	69.2	230.8					167.2	42.8	210.0

表 -3　インスペクションの改善による手戻り工数削減効果の試算例

『2007 年度版組込みソフトウェア産業実態調査』の調査結果を参考に，筆者が任意に設定した．

さて，表 -3 の状況において，要求インスペクションと設計インスペクションを強化し，欠陥見逃し件数をそれぞれ現状の 1/4 および 1/3 にまで低減できたとする．インスペクションの強化には，短期的にはコストがかかる．たとえば，インスペクション参加者を増やす，準備工数を増やす，インスペクション会議の時間を増やすことにより，品質の確保が期待できるが，その分のコストが増加する．ここでは，要求工程（または要求インスペクション）と，設計工程（同様）における稼働工数を，それぞれ 10% 増やすことで，欠陥除去率の向上が実現できるとする．

以上の仮定に基づいて，改善後の工数を試算した結果が表 -3 の右側部分である．元の見積り工数が 230.8 人月であったのに対して，改善後の工数は 210.0 人月であり，約 10% の工数削減効果が算出されている．要求工程と設計工程の稼働工数は増加しているが，要求工程以降の手戻り工数の減少がそれらを上回っているため，全体では工数削減に結びついている．

この試算例では，インスペクションコストをかけることで，手戻りコストの削減をもたらす例を示した．このように，評価コストを増やして失敗コスト（およびそのリスク）を削減するのは，品質コスト投資におけるセオリーの第 1 段階である．次の段階に求められるのは，インスペクションプロセスの改善などの予防コストをかけることで，失敗コストの削減，さらには評価コストの圧縮を図る必要がある．

■ リーディング手法に着目した有効性の改善

インスペクションプロセス，特に理解・指摘フェーズを改善するには，大別すると，成果物を調べる手法（リーディング手法）の改善と，会議の進め方の改善の 2 つに分けられる．ここでは，技術的な観点での改善に焦点を絞って話を進める．

リーディング手法の違いによって，欠陥指摘の有効性に統計的有意差があることが，多くの研究により示されている（たとえば文献 4)）．ただし，あるリーディング手法が他に比べて“一貫して”有効性が高いという結果が示されているわけではない．文献 4）では，パースペクティブベースドリーディング(後述)と，アドホックまたはチェックリストを用いた方法の有効性を比較した 8 件の文献を比較しているが，このうち，統計的に有意な差が示されたのは 3 件のみであったことを示している．

とはいえ，これらのリーディング手法に関する研究に共通する含意は，アドホックな手法に比べてシステマティックなリーディング手法を適用した方が効果的であることと，リーディング手法によって指摘しやすい欠陥の種類に違いがあることの 2 点である．前者については，あらためて議論するまでもないだろう．一方，後者について，リーディング手法の違いによって欠陥指摘のしやすさに差が生じるのは，論理的に考えて納得がいく．

チェックリストベースドリーディングの場合，過去に見逃した欠陥に関する経験的知識を網羅的に適用できるという利点がある．しかし，具体的なシナリオにおけるシステム利用フローにおいて生じ得る問題を見いだすのには不向きである．また，チェックリストが肥大化しがちで，すべてのチェック項目を確認するのが容易ではない，チェック項目の記述が抽象的だとインスペクタがその真意を認識できない，チェックリストの維持がなされないと形骸化するなどの課題がある．

ユースケースベースドリーディングでは，具体的なシステム利用シナリオを用いるので，要求インスペクションにおける妥当性（validation）の確認において効果を発揮する．また，この派生手法として，テストケースに基づいたリーディング手法が挙げられる．筆者がこの手法を用いて設計およびコードレビューの実験を行った際には，モジュール間インタフェース（たとえば，呼出元の変数の値によって呼出先関数の振舞いが変わるなど）にかかわる欠陥を，具体的なテストケースを用いたために容易に検出できたという結果を得た [6)]．しかし，ユースケースについて，想定可能なすべてのシナリオを漏れなく抽出するのは容易ではない．テストケースの場合はカバレッジを考慮できるが，レビュー時に用いるテストケ

ースをうまく縮約しないと同じ実行パスを何度も確認することになり，無駄が生じる．

パースペクティブ（観点）ベースドリーディングでは，インスペクタが役割分担をすることで，多様な観点から効率よく成果物を確認できる．その際，ユーザ視点で調べるときにはユースケースを用いる，設計視点では縮約済みのテストケースを用いる，テスト視点では経験的知識に基づいたチェックリストを用いるなど，さまざまなリーディング手法を組み合わせて適用するのがよい．

先述した通り，リーディング手法の有効性や効率性は，研究論文によって結果に一貫性がない場合がある．これは主に，実験環境の問題に起因していると思われる．リーディング手法に関して，実践という視点に限れば，論文に有効性の根拠を求めるよりは，むしろ論理的な思考に基づいて適用方法を工夫するのがよいと思われる．

アプリケーション領域知識の組織的活用

ここまで述べた内容は，ソフトウェアエンジニアリングの技法的側面についてであった．しかし，インスペクションの効果と効率は，技法のみで解決できる範囲は限られている．決定的に重要な要因として，アプリケーション領域知識の適切な運用が必要である．

ここで，1つの事例を挙げて説明する．2006年に，高齢者用集合住宅で入居者が死亡後1カ月経ってから発見されるという事故があった．この事故の引き金となったのは，入居者が在宅中でも外からの施錠でシステムに「不在」と表示される仕様上の問題であった．入居者の様子を見に警備員が来て，警備員が帰るときに外から施錠したため，在宅中にもかかわらずシステム上は「不在」となった．これにより，すべての異常検出機能が無効となり，結果的に，死亡事故にまで至った．

この事例を見て，多くの読者は，なぜそのような仕様にしてしまったのか疑問に思うことであろう．それは，我々が住宅での在室・不在処理というアプリケーション領域について，一定レベル以上の知識を有しているため，仕様上の不備にすぐ気がつくのである．しかし，このように多くの人が知識を共有できるアプリケーション領域の方がむしろ稀で，現実の製品開発では，すべてのインスペクタが十分な知識を保持していない場合も多々ある．組織内で知識を共有し，ソフトウェア開発のあらゆる局面でこれを活用しなければ，先に挙げたような事故事例を繰り返すことになる．

このように，インスペクションの効果と効率を高めるエンジニアリング技法の適用と，アプリケーション領域に固有の知識を組織内で共有し活用することは，いずれもソフトウェア品質の確保において決定的に重要である．

おわりに

以上，限られた紙幅の中ではあるが，本稿の冒頭に述べたステートメント「ソフトウェアインスペクションを実施すると，ソフトウェア成果物に含まれる欠陥を効果的かつ効率的に発見できる」を実現するために必要な取り組みのための，3つの課題について述べた．これら3つの課題は，いずれも，ソフトウェア開発にかかわる人の要素によるところが大きい．ややもすると，現状のプロセスを変えることなく，これらの課題への取り組みをやり過ごしてしまうかもしれない．しかし，本稿で紹介した事故事例をはじめとした，ソフトウェア品質の問題に取り組むには，インスペクションの効果と効率を継続的に改善することが不可欠である．そのような取り組みの際に，本稿を参考にしていただければ幸いである．

参考文献

1) Gilb, T. and Graham, D. : Software Inspection (1993). (伊土誠一，富野　壽監訳，ソフトウェアインスペクション，共立出版 (1999)).
2) Humphrey, W. S. : The Team Software Process^{SM} (TSP^{SM}), CMU/SEI-2000-TR- 023 (2000).
3) Laird, L. M. and Brennan, M. C. : Software Measurement and Estimation : A Practical Approach, IEEE Computer Society, Wiley-Interscience (2006).
4) Thelin, T., et al. : An Experimental Comparison of Usage-Based and Checklist-Based Reading, IEEE Trans. Softw. Eng., Vol.29, No.8, pp.687-704 (2003).
5) Wiegers, K. E. : Peer Reviews in Software : A Practical Guide, Addison-Wesley (2002). (大久保雅一監訳：ピアレビュー，日経BPソフトプレス (2004)).
6) 野中　誠：設計・ソースコードを対象とした個人レビュー手法の比較実験，情報処理学会研究報告 SE-146-4, pp.25-32 (2004).

(平成21年3月12日受付)

野中 誠（正会員）　nonaka-m@toyonet.toyo.ac.jp

1972年生．早稲田大学理工学部工業経営学科卒業，同大学院理工学研究科経営システム工学専門分野修了，博士後期課程単位取得退学．早稲田大学助手を経て，2003年東洋大学経営学部専任講師，2006年現職．