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

1 ソフトウェアインスペクションの動向

森崎 修司　奈良先端科学技術大学院大学 情報科学研究科

ソフトウェアの品質向上に向けた活動

ソフトウェア，情報システムの社会への広範な浸透に伴い，ソフトウェア品質が社会に与える影響の大きさは増加の一途をたどっている．また，開発期間の短縮がビジネス面での優位性をもたらす場合も多く，短い期間で高い品質という一見矛盾するような要求がつきつけられている．ソフトウェアの品質を高める施策の１つとして，評価による欠陥の発見と修正があり，ソフトウェアテスト，ソフトウェアレビュー／ソフトウェアインスペクションが含まれる．

ソフトウェアレビュー／ソフトウェアインスペクションはソフトウェアの中間成果物，成果物（本稿ではソースコードを含め，ドキュメントと呼ぶ）を対象として，対象の作成担当者以外の目視によって欠陥を発見，指摘する活動であり，ソフトウェアの静的解析の１つと位置づけられる．ソフトウェアテストと同様にソフトウェアの誤りを発見する活動であるが，プログラムが実行可能状態（コンパイル，ビルド可能）にない段階でも実施できる点でソフトウェアテストとは異なる．近年では，多くのソフトウェア開発プロジェクトにおいて品質向上活動としてソフトウェアレビュー／ソフトウェアインスペクションが実施されており，多くの組織において標準プロセスの１つとして位置づけられている．また，オープンソースソフトウェアにおいても，ソフトウェア品質の維持，向上を目的として，ソースコードをプロジェクトのリポジトリにコミットする前に作成担当者以外のメンバによるコードレビューを義務づけているプロジェクトも多い．

ソフトウェアレビュー／ソフトウェアインスペクションの歴史は古く，1976 年に Fagan が “Design and code inspections to reduce errors in program development”[3] として提案したのが最初とされている．その後，研究者や実務者による提案，検討が積み重ねられ，近年では，その目的，形態，名称，対象も多岐にわたり，ソフトウェアインスペクションの裾野は広がりを見せている．たとえば，ソフトウェアレビュー／ソフトウェアインスペクションを定義している IEEE 1028 Software Reviews and Audits の 1998 年版では，対象ドキュメントとして，設計書，ソースコードをはじめとして 7 項目が対象ドキュメントの例として挙げられているが，2008 年版 1.6 節では，39 項目が対象ドキュメントの例として列挙されている．

図 -1　ソフトウェアレビュー／ソフトウェアインスペクション会議の様子

ソフトウェアレビュー／ソフトウェアインスペクションの形態はさまざまであり，その詳細は後述するが，代表的な実施方法の１つは，図 -1 のように会議形式で作成者，レビューア／インスペクタが参加し，欠陥と推測される部分，疑問点，今後欠陥になり得ると推測される

局面	対象ドキュメントと指摘例
計画	文書 ID PS-20081121-1 の顧客向け提案書 8 ページに記されている冗長化を実現するためには，13 ページのシステム構成では計算機が足りない．
要求	文書 ID CS-1002-4 の要件定義書 15 ページ 24 行目「リクエストから 5 秒以内に結果もしくは，タイムアウトしたことを示す結果をユーザに提示すること」と 22 ページ 12 行目「国際ネットワークを通じた処理の場合，待機時間 7 秒以上とすること」を同時に満たせない場合がある．
設計	文書 ID D-1002-12 の詳細設計書 42 ページ 5 行目，および，表 3-2 の「次の状態遷移まで現行の値を保持する」という記述があるが，47, 52, 53 ページ，および，表 4-5，図 4-1 には状態を保持するための仕組みが記述されていない．
コーディング	receive_request.c 727 行目，関数 send_status への引数が変数 status となっているが，現在の状況は status ではなく，current_status に入っている．
テスト（テスト設計）	文書 ID TP-1002-122 のテスト計画書 3 ページ項目 B-10 のテスト項目「ステータス 0xBBA8 に対して例外処理が実行されることを確認」とあるが，該当する例外処理が複数あるため，例外処理コード等の具体的な記述が必要である．
運用	文書 ID OP-1002-202 復旧手順書 133 ページ ハードウェア障害復帰時のデータ回復のために必要な情報がウィークリーバックアップとされているが，このバックアップは差分のみを記録しているため，元のデータを復元できない．

表 -1　ソフトウェアレビュー／ソフトウェアインスペクションにおける欠陥指摘の例

部分を指摘するものである．表 -1 は局面，対象ドキュメントと指摘の例である．ソフトウェアレビュー／ソフトウェアインスペクションでは，一般に，欠陥の指摘が中心であり対応策については必ずしも検討する必要はない．

ソフトウェアレビュー／ソフトウェアインスペクションは人間中心の自由度の高い活動であるため，その効果がばらつきやすい傾向にある．ソフトウェアレビュー／ソフトウェアインスペクションの研究は，実施形態，進め方を定義することにより，より安定した効果，高い効果を得ることを目指している．本稿では，ソフトウェアレビュー／ソフトウェアインスペクションの動向を紹介することを目的とし，最初に一般的な定義と期待される効果を述べる．次に，これまで検討されてきた実施形態や実施フェーズを紹介する．最後に今後の研究の方向性を示す．

ソフトウェアレビュー／ソフトウェアインスペクションの定義と効果

■ 定義

文献 1) や文献 6) でも指摘されているとおり，ソフトウェアレビュー／ソフトウェアインスペクションの定義でコンセンサスの得られたものは筆者の知る限り存在しない．また，レビュー／インスペクションについて触れられている標準は多数存在するが，その定義には十分な一貫性が確保されているわけではない．本稿では，標準文書の中でも具体的な定義がなされており，SWEBOK(SoftWare Engineering Body of Knowledge) のソフトウェア品質マネジメントプロセスでも参照されている IEEE 1028 Software Reviews and Audits 2008 での定義を紹介する．IEEE 1028 では，Management review, Technical review, Insepction, Walk-through, Audit が定義されている．Management review は，管理職が状況把握と意思決定をすることを目的としている．Audit はソフトウェアが規定，標準，ガイドライン，手順等に従っているかどうかを確認することを目的としている．表 -2 は，学術研究における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関連の強い Technical review, Inspection, Walk-through を比較した表を IEEE 1028 から抜粋したものである．

表 -2 からも，うかがい知ることができるが，ソフトウェアインスペクション（Inspection）が最も公式（事前に定義された形式に沿って実施）であり，レビュー（Technical review），ウォークスルー（Walk-through）の順にカジュアルになっていく．ソフトウェアインスペクションでは，意思決定はその場で検討されることは少なく，あらかじめ定義された選択肢のいずれかを選ぶ．また，指摘された欠陥は修正されることが前提である．レビューでは技術リーダーが参加メンバの推薦に基づいて意思決定する．ウォークスルーでは参加メンバと相談しながら作成者が意思決定することができる．ソフトウェアインスペクションではソフトウェアインスペクション対象ドキュメントの欠陥指摘を目的としているのに対し，レビューでは仕様，計画とレビュー対象ドキュメントの一致を評価，ウォークスルーでは，よりよい実現／実装方法の検討や参加メンバの学習のための議論も目的とされている．

研究分野においては，IEEE 1028 をはじめその他の定義に基づいてインスペクション，レビュー，ウォークスルー等の呼称が使い分けられていることは多くない．海外ジャーナルや国際会議の学術論文では，ほと

	Inspection	Technical review	Walk-through
目的	欠陥の発見，修正の検証，ソフトウェア品質の検証	仕様，計画との一致の評価．変更の一貫性の判断	欠陥の発見，代替案の検討，製品の改善，習得のための議論
意思決定	事前に定義された対処方法を選択．発見された欠陥は修正	参加メンバの提案に基づいて管理職や技術リーダが判断	作成者の判断を参加メンバが合意
参加人数	3～6名	3名以上	2～7名
主導者	訓練を受けた進行役	技術リーダ	進行役，作成者
データ収集	必須	プロジェクト要件ではないが参加者で共有されることが多い	推奨される
成果物	指摘欠陥リスト，実施報告書	実施報告書（担当，完了予定日が記録された対応予定を含む）	指摘欠陥リスト，対応予定，決定事項
管理職の参加	なし	管理職参加のエビデンスが必要なとき	なし
顧客（ユーザ代表）の参加	可	可	可

表 -2　Inspection, Technical review, Walk-through の比較表（出典：IEEE 1028 一部抜粋）

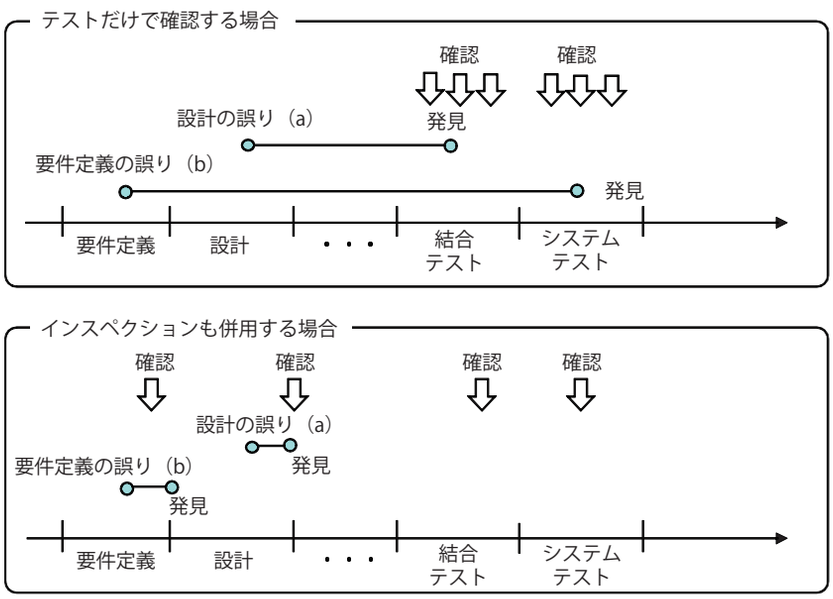

図 -2　インスペクションの効果

んどの場合，software inspection あるいは inspection と言及されている．本稿でも，以降では，ソフトウェアインスペクション，あるいは単にインスペクションと呼ぶ．筆者が Working group of International Research Cooperation on Software Inspections[☆1] の活動を通じて海外の研究者や実務者に聞いた範囲でも，その定義は曖昧であり，実務者の間では review が比較的多く使われているという反応を得ている．

■効果

混入した欠陥を早期に発見，修正することにより，やり直しの手間や時間がテストと比較して小さくなることによって，インスペクションの効果がもたらされる．図 -2 はウォータフォールモデル開発での欠陥の混入と除去の様子を表している．図 -2 上はテストによる欠陥の発見を，下はインスペクションとテストを併用する場合である．設計段階で欠陥 (a) が混入され，テストのみで欠陥を検出しようとする場合，誤った設計に基づいてコーディング，単体テストが実施された後，結合テストで欠陥が発見され，修正される．修正時には，誤った設計に基づいて作成されたソースコードが修正され，修正後のソースコードに対して，再度単体テストを実施する必要がある．設計インスペクションで欠陥 (a) を発見，修正できれば，誤った設計に基づいたソースコードの作

☆1　http://www.software-inspection-wg.org/

成，単体テスト実施の手間を省くことができる．同様に要件定義の誤り(b)を早期に発見，修正できれば，(b)に基づいた設計，実装，単体テスト，結合テストの手間を省くことができる．

ウォータフォールモデル型の開発だけでなく，繰返し開発でも同様のことがいえ，以下のように一般化できる．

ソフトウェア開発の過程を要求からプログラムへの段階的な詳細化とし，各段階で欠陥が混入される可能性があり，かつ，混入された欠陥は詳細化が完了したテストでのみ発見されるとするならば，ソフトウェアに存在する欠陥の集合 D を詳細化の段階 i で分類でき，次のように表すことができる．

$$D = \{ D_1, ..., D_i, ..., D_m \}$$

m はテストが実施されるまでの詳細化の段階数を表す．

さらに，詳細化段階 i においてソフトウェアに存在する欠陥の集合 D_i の要素である欠陥 d は次のように表すことができる．

$$D_i = \{ d_{i1}, d_{i2}, ..., d_{in_i} \}$$

ここで n_i は詳細化段階 i においてソフトウェアに存在する欠陥の総数である．また，ある詳細化段階 i でソフトウェアに存在した欠陥 d_{ij} に基づいて詳細化された欠陥は詳細化段階 $i+1$ において $d_{(i+1)j}$ として存在するとする．

ここで，詳細化段階 i において欠陥 d_{ij} を修正するコストを $cost(d_{ij})$ と表す．インスペクションはテストに先立ってある詳細化段階 i において欠陥を修正するため，$\Sigma cost(d_{kj})$ $(i < k \leq m)$ の修正コストを削減することができる．

特に，i が増えるに従って，$cost(d_{ij})$ が大きくなる欠陥を修正したり欠陥の発見数を増やすことにより，インスペクションの効果が大きくなることが期待できる．

実施形態

インスペクションの活動は人手によるところが大きく，実施の自由度が大きい．多くの研究で，本章で示す「実施形態」，次章で示す「フェーズ」において，一定の手順や制約を導入することによりインスペクションの効率を上げ，結果として修正コストを削減することを目指している．

■ 実施回数・実施グループ

実施回数は以下のとおり．ここでの実施回数は，同一ドキュメントに対する実施回数である．また，会議として複数回に分かれている場合であっても，一連の会議であれば，1回と数えるものとする．また，実施グループが異なる場合もある．

• 同一グループで1回

対象ドキュメントが完成した時点で1回インスペクションを実施する．欠陥指摘への対応(修正等)が完了したら次工程へ移行する．対象ドキュメントのうち，完成した部分から先にインスペクションを実施する場合もこれに該当する．実施にかかる手間とコストが他と比較して小さくて済むところが利点である．相互に依存する欠陥指摘がされている場合に，対応がうまくなされない可能性があることが欠点である．

• 同一グループで複数回

フェーズつきインスペクション(phased inspection)[5]，アジャイルインスペクション(agile inspection)[4]が該当する．フェーズつきインスペクションは，同一の対象ドキュメントに対して異なる観点から複数回インスペクションを実施する．k 回目のインスペクションの欠陥指摘への対応が完了しないと $k+1$ 回目のインスペクションは実施されない．それぞれのインスペクションには観点を設定する．たとえば，網羅性の観点でのインスペクションを実施，修正後に，保守性の観点でインスペクションを実施するなどである．フェーズつきインスペクションの利点は，相互に依存する欠陥指摘が減るため，指摘欠陥への対応が正確に実施されやすくなるところである．欠点は，実施コストが高い点，完了までに時間がかかる点である．

アジャイルインスペクションは品質計測を目的として，対象ドキュメントのごく一部を抜き取りインスペクションを実施する．計測した品質指標（単位ドキュメント規模あたりの重要欠陥数等）が目標値となるまで，作成者に修正を依頼する．他のインスペクションと異なり，指摘欠陥の修正だけでなく，それ以外の部分についても作成者が類推し修正する必要がある．ごく一部を取り出し，何度も実施する点からアジャイルという名前がついており，必ずしもアジャイル開発プロセスで利用されるものではない．アジャイルインスペクションの利点は，実施コストが小さいこと，対象ドキュメントの一部が完成すれば実施できる点である．修正を依頼された作成者が，指摘された欠陥以外の類似の欠陥も類推して修正することが前提となっているため，作成者に一部の欠陥から類似の欠陥を類推するためのスキルが求められることが欠点である．

• 複数グループで1回ずつ

N 重インスペクション（N-fold inspection[7]）と呼ばれる．同一の対象ドキュメントに対して異なるメンバから構成されるグループでインスペクションを実施する．指摘後，指摘結果をグループ間で共有する．グループの間

でなるべく観点が重ならないようグループを構成する．利点は指摘の網羅性が大きくなる点である．欠点は実施コストが高い点，グループ間での指摘結果をまとめる必要がある点である．

■ 実施メンバ

• 開発メンバ

対象ドキュメントの作成者以外で開発に従事しているメンバがインスペクタとなり，開発当事者としての知識を利用しながら欠陥を指摘する．当事者には，プロジェクトマネージャ，リーダー，仕様書作成／設計／コーディング／テスト／運用／保守の担当等が含まれるが，欠陥の指摘は必ずしも同じ担当どうしで実施する必要はない．たとえば，対象ドキュメントが仕様書の場合に，インスペクタは必ずしも仕様書を作成している別の担当である必要はなく，テスト担当や運用担当がその担当の立場からインスペクションを実施してもよい．対象ソフトウェア，ドキュメントをよく理解しているメンバで実施するので，対象への理解や説明に必要な時間を省くことができる利点がある．開発側の視点に限定された欠陥指摘になるので，ユーザが求めているソフトウェアとなっているかどうかなどの指摘されにくい欠陥が残る場合があることが欠点である．

• ユーザと開発メンバ

ユーザがインスペクタとして参加し，ユーザの期待するものが開発されているかどうかを確認する．ソフトウェアがユーザにとって必要なものとなっているかどうかを中心に指摘する．ユーザが求めているものとなっているかどうかの確認，ユーザと開発状況を共有できる利点がある．ユーザに負担がかかる点，ユーザにソフトウェア(特にソフトウェア内部)に関する説明をするためのコストがかかる点が欠点である．

• 第三者と開発メンバ

開発当事者としての知識を持たないが，指摘に必要な固有の知識を持つ第三者がインスペクタとして参加する．固有の知識には，標準，法令への準拠，セキュリティ面での問題をはじめとして，当事者では気づきにくい欠陥のパターン等，欠陥指摘固有の知識も含まれる．開発者側にはない観点からインスペクションを実施できる利点がある．ソフトウェアに関する説明をするためのコストがかかる点が欠点である．

■ 実施時間の同期／非同期

実施時間を同期する同期型，同期しない非同期型がある．後述するフェーズごとに選択することができる．

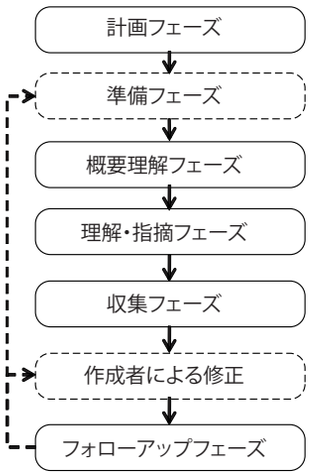

図 -3　インスペクションのフェーズ

• 同期型

実施の時間帯を揃え，参加メンバが同じ会議に参加する．同一の場所で実施されることが多いが，遠隔で電話，テレビ会議，支援ツール等を介して実施することもある．指摘をとりまとめるのが容易であること，参加メンバ間で知識共有が容易にできるところが利点である．参加メンバの時間を拘束してしまう点，参加メンバの都合のよい日時までの待ち時間が発生することが欠点である．

• 非同期型

各々の参加メンバの指摘結果を提出する締切を設定し，締切までにそれぞれの都合のよい時間に実施する．参加メンバの都合のよい時間を選んで実施できる点，時差のあるような遠隔地とでも容易に実施できる点が利点である．指摘結果の重複を取り除くなど，指摘をとりまとめる手間がかかる点が欠点である．

▶ フェーズ

インスペクションのフェーズを図 -3 に示す．計画フェーズはソフトウェア開発プロセス全体でのインスペクションの位置づけを計画するフェーズである．その後，日程調整をはじめとしたインスペクションの準備フェーズがあり，その後，個々のインスペクションでの指摘をするための概要理解，理解・指摘，収集フェーズがある．収集フェーズではインスペクションの結果が集められ作成者による修正の後に，フォローアップフェーズがある．フォローアップフェーズの結果により，再度，作成者による修正を依頼したり，インスペクション自体を再実施する場合がある．

■ 計画フェーズ

計画フェーズでは，ソフトウェア開発全体の計画の中

でのインスペクションの目的，実施条件，得られた結果の使い方を定義し，前章で述べた実施形態の各項目の予定を立てる．インスペクションの結果が他のソフトウェア開発の作業に影響を与える場合が多いため，本フェーズはソフトウェア開発全体の計画時点で実施することが多い．同様の理由で，本フェーズは開発プロジェクトの責任者を交えたり，責任者自身によって実施される場合が多い．インスペクションで得られた結果を，その後の開発作業でどのように利用するかも本フェーズで決定しておく．多くは指摘された欠陥の修正であるが，品質推測の判断材料の１つとして，リリースや工程移行の判定に使ったり，再インスペクションやインスペクタの変更の判断に使われる場合もあるので，事前に計画に含めておく．

インスペクションの目的は“保守性を向上する”，“性能問題が起こらないようにする”等の欠陥指摘の大まかな方向性を示すものとする．実施条件は特定の条件を満たしたときだけインスペクションを実施するといったものや特定の条件を満たしたときにインスペクタを追加，変更するといったものである．たとえば，あるソースコードメトリクスの閾値を設定しておき，ソースコードが完成した時点でそのソースコードメトリクスを計測し，閾値よりも大きくなった場合にはコードインスペクションを実施するなどである．また，欠陥指摘密度など，インスペクションの効果を示すメトリクスを設定しておき，下限値を下回ったときに，再インスペクションする場合もある．

■概要理解フェーズ

概要理解フェーズは対象ドキュメントが揃った時点で実施される．対象ドキュメントの構造や全体像を理解するためのフェーズである．設計インスペクションであれば，外部仕様書，概要設計書，画面設計書，シーケンスダイアグラム等がどのようなファイルに保管され，それぞれがどのような依存関係にあるかをインスペクションの参加者が理解する．対象ドキュメントの量が多い場合には，この時点で対象ドキュメントの一部を選び出す場合もある．ユーザによるインスペクション，第三者インスペクションにおいても，対象ドキュメントの一部を選択する場合が多い．対象ドキュメントの品質を予測するモデル（Fault-prone model）が多数提案されており，それらを使って選択する場合もある．

■理解・指摘フェーズ

理解・指摘フェーズでは，インスペクタが対象ドキュメントを実際に理解しながら，欠陥と思われる部分を指摘していく．必ずしも対象ドキュメントのはじまりから順番に読んでいく必要はなく，インスペクタが疑わしいと考える部分から理解をはじめ，欠陥を発見すれば指摘する．たとえば，非機能要求に定量的な指標が含まれているか，サブシステム間のインタフェースに不整合がないか，ユーザからの入力値の妥当性確認を怠っていないか等の観点で関係する部分から理解をはじめる．

理解・指摘フェーズの進め方としてリーディング技法が数多く提案されている．リーディング技法は，インスペクタ全員で同じやり方で進めていくもの，インスペクタごとにシナリオを割り当てるもの，の２種類に大別できる．前者の代表的なものとして，チェックリストリーディング，ユースケースベースドリーディングがある．チェックリストリーディングは，事前に準備したチェックリスト（欠陥指摘に役立つ質問の集合）に答えていくことで欠陥を指摘する．ユースケースベースドリーディングは，ユーザの利用手順であるユースケースに沿って対象ドキュメントを読み進め，欠陥を指摘する．

シナリオを用いたリーディング技法の代表的なものとしとして，パースペクティブベースドリーディング，ディフェクトベースドリーディングがある．パースペクティブベースドリーディングでは，インスペクタごとに役割を割り当てる．役割には，テスタ，ユーザといった開発プロジェクトに対する立場を選ぶ．テスタの役割を割り当てられたインスペクタは，対象ドキュメントからテストケースを作れるかを確認し，欠陥や曖昧な部分を指摘する．たとえば，仕様書に“長時間かかる場合にはその進捗をユーザに提示する”という機能の定義がある場合，これをテストするテストケースを作成するためには“長時間”の定義が必要になるため，“長時間の定義がない”という指摘ができる．ディフェクトベースドリーディングは，過去に起こった不具合を ODC（Orthogonal Defect Classification：直交欠陥分類法）等を用いて分類し，不具合の分類ごとに対象ドキュメントに含まれている欠陥を指摘する．

■収集フェーズ

理解・指摘フェーズで指摘された欠陥を精査し，重複する指摘の除去，誤った指摘の除去を実施し，指摘欠陥リストを作成する．各々の欠陥に対する分析や計測も本フェーズで実施される．分析では，指摘欠陥を見逃した場合のインパクトをもとに深刻度を決定したり，欠陥が混入された時期（工程），理由，本来検出すべきであった時期（工程）を推測したりする．計測では，対象ドキュメントの規模，インスペクションの実施時間，要した工数等をもとに指摘欠陥密度，インスペクション速度，単位工数あたりの指摘欠陥数等の品質，生産性メトリクスを算出する場合もある．Fault-prone モデルや Capture-

recapture モデルをはじめとして，メトリクスから対象ソフトウェアの品質を予測するためのモデルが多数提案されている．得られたメトリクスをもとに計画フェーズで定義された条件で再インスペクションを実施する判断をする場合もある．

インスペクションでは，原則的に指摘欠陥に対する修正方法の検討はしないが，指摘の内容や作成者のスキルによっては修正方針を検討することがある．その場合，本フェーズで実施される．

■ フォローアップフェーズ

収集フェーズで作成された指摘欠陥リストをもとに修正した後に本フェーズを実施する．指摘欠陥リストに記載されている欠陥が修正されているかどうかを確認する．特に修正に伴って全体との整合性が失われていないか，該当個所の修正に伴って，他の部分に波及していないか，を確認する．指摘されたすべての欠陥が修正されれば，そのインスペクションは完了となる．

▶今後のインスペクション研究の方向性

インスペクション研究の今後の方向性は大きく 2 つある．1 つは，限られた時間で効率的に欠陥を指摘することの支援であり，テストでの検出とインスペクションの検出の棲み分けにつながる試みである．もう 1 つは，プロジェクトの事情やソフトウェアに求められる要件にあわせたインスペクションのテーラリング(カスタマイズ)である．

■ 時間制約つきインスペクション

これまでのインスペクション研究では，インスペクションで欠陥を発見し修正することにより修正コストが小さくなるという前提で，網羅的な発見を支援するものが多かった．しかしながら，現実にはインスペクションで指摘，修正した場合とテストで発見，修正された場合で，それほど修正，確認コストが変わらないものがある．たとえば，ユーザに提示するエラーメッセージの誤字は，インスペクションで指摘しても，テストで発見しても修正コストが大きく変わらない．

1990 年代までに提案されたリーディング技法の多くが欠陥指摘の網羅性を大きくするためのものであった．たとえば，チェックリストリーディングやパースペクティブベースドリーディングは欠陥指摘の網羅性を高めることを目的としている．しかしながら，対象ドキュメント規模の増大，開発期間短縮の要望が強くなり，インスペクションで指摘すべき欠陥の種類を明らかにし，その種類の欠陥を優先的に指摘しようとする動きが出てきている．

Petersen らは，ユースケースベースドリーディングのユースケースに優先順位づけをし，優先順位の高いものに理解・指摘フェーズの時間を多く割り当てる Time controlled reading を提案している[8]．ユースケースには概要理解フェーズにおいてユーザの利用頻度に応じて理解・指摘フェーズの時間が割り当てられ，優先順位の高いユースケースほど，長い時間をかけてインスペクションが実施される．

筆者らの研究グループでも，インスペクションで見逃され，テストで発見された場合に必要となる修正コストや波及範囲が大きい欠陥を優先的に指摘することを目的としたインスペクション技法を提案している．たとえば，文献 9）では，テストで発見，修正された場合に必要となる再テストの件数が大きいと推測されるものから指摘していく手法を提案し，その効果を確認している．

アジャイルインスペクションも時間制約つきインスペクションの延長線上にあると考えることができる．ドキュメントの一部分を対象とし，欠陥指摘し，作成者に修正を依頼することで欠陥指摘をしていないドキュメントの他の部分についても修正されることが期待されるため，インスペクション時間を短縮することができる．

■ インスペクションのテーラリング

インスペクションの主な活動は人手による欠陥の発見である．画一的な手法でインスペクションの効果を向上させるのは難しいという前提に立ち，インスペクションをカスタマイズする手法が提案されている．Denger らは文献 2）で，インスペクションをテーラリングするための手法 TAQtIC を提案している．TAQtIC では計画フェーズにおいて，プロジェクトの特徴，事情，関係者を明確化し，特徴，事情に合わせて，インスペクションの目的，リーディング技法を設定する．たとえば，インスペクタのスキルや経験に合わせてシナリオを割り当て，関係者の日々の活動とインスペクション活動との対応を明確にすることでインスペクタへの動機づけとすることを目指している．

表 -3 は文献 2）で紹介されているシナリオの例である．シナリオは，前提，質問から構成され，ここでは，経験が浅いプログラマと経験豊富なプログラマそれぞれに別のシナリオが割り当てられている．

▶まとめ

ソフトウェアレビュー／ソフトウェアインスペクションは 1970 年代に提案されて以来，オープンソースソフトウェア，国内外を問わず，大多数の商用開発において

	初級プログラマ	エキスパートプログラマ
前提	自身を対象システムに関する知識が少ないプログラマだとして，ユースケースを中心に読み進めてください．	対象ソフトウェアに関する豊富な知識を持つプログラマだとして，システム全体でどのクラスが最も重要か，またどのクラスが最も複雑かを考え，最も複雑，重要なクラスから進めてください．
質問	・機能を理解するときやソースコードが正しいかどうかを判断するために，ソースコードのコメントやドキュメントは十分でしたか？ ・ユーザの操作が関与する部分すべてにおいてエラー処理や例外処理はありますか？ ・エラーやデバッグのためのログ出力は適切な場所で行われていますか？	・データ構造は適切に使われていますか？ また，そのデータ構造は実行速度やメモリ使用量とのトレードオフが考慮されていますか？ ・これまでの経験から，このクラスで実現されている機能は今後の開発でも必要になりそうですか？ もしそうなら，再利用性を高めるための追加コストを投じる価値がありそうですか？

表 -3　TAQtIC 適用時のシナリオ例

実施されており，その効果は認められつつある．しかしながら，その研究成果やベストプラクティスの共有は十分に行われているとは言い難い状況にある．本稿がそれらの情報共有に寄与することを願う．本稿の執筆にあたって，NTT データ MSE 佐々木健介氏，Fraunhofer Institute for Experimental Software Engineering Frank Elberzhager 氏，奈良先端科学技術大学院大学 亀井氏，下村氏，瀧氏，保田氏にご協力，アドバイスをいただいた．ここに感謝の意を表する．

参考文献

1) Aurum, A., Petersson, H. and Wohlin, C. : State-of-the-art : Software Inspections After 25 Years, Software Testing, Verification and Reliability, Vol.12, No.3, pp.133-154 (2002).
2) Denger, C. and Shull, F. : A Practical Approach for Quality-Driven Inspections, IEEE Software, Vol.24, No.2, pp.79-86 (2007).
3) Fagan, M. E. : Design and Code Inspections to Reduce Errors in Program Development, IBM Systems Journal, Vol.15, No.3, pp.182-211 (1976).
4) Gilb, T. : Agile Specification Quality Control, Cutter IT Journal, Vol.18, No.1, pp.35-39 (2005).
5) Knight, C. J. and Myers, A. E. : Phased Inspections and Their Implementation, ACM SIGSOFT Software Engineering Notes, Vol.16, No.3, pp.29-35 (1991).
6) Laitenberger, O. and DeBaud, J. : An Encompassing Life Cycle Centric Survey of Software Inspection, Journal of Systems and Software, Vol.50, No.1, pp.5-31 (2000).
7) Martin, J. and Tsai, W. : N-Fold Inspection : A Requirements Analysis Technique, Communications of the ACM, Vol.33, No.2, pp.223-232 (1990).
8) Petersen, K., Ronkko, K. and Wohlin, C. : The Impact of Time Controlled Reading on Software Inspection Effectiveness and Efficiency : A Controlled Experiment., In Proc. of the International Symposium on Empirical Software Engineering and Measurement, pp.139-148 (2008).
9) 田村晃一，亀井靖高，上野秀剛，森崎修司，松村知子，松本健一：見逃し欠陥の回帰テスト件数を考慮したコードレビュー手法，電子情報通信学会技術研究報告，Vol.108, No.173, pp.61-66 (2008).

（平成 21 年 3 月 4 日受付）

森崎 修司（正会員）　smrs@is.naist.jp

2001 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．同年（株）インターネットイニシアティブ入社後，オンラインストレージサービスの立ち上げ／マネージメント／開発，RFID ソフトウェアの国際標準策定活動に従事．2007 年奈良先端科学技術大学院大学助教．博士（工学）．ソフトウェア計測，ソフトウェアインスペクションの研究，文部科学省「次世代 IT 基盤構築のための研究開発：StagE プロジェクト」に従事．