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

5 第三者インスペクションによる品質検査と欠陥予防

細川 宣啓　日本アイ・ビー・エム（株）

ソフトウェアインスペクションによる品質向上活動

ソフトウェアの開発においては，開発早期（要求定義や設計段階）におけるドキュメントの欠陥の修正漏れが原因で，後にコスト・スケジュール上の問題がしばしば引き起こされる．現在，ドキュメントの欠陥を調べる手段は基本的に人間による目視に依存している．これらは具体的な手順や形式によって，インスペクション，レビュー，ウォークスルーなどに分類され，実際に欠陥の早期発見に効果があることが歴史上，実証的に示されている．

Fagan により提案されたソフトウェアインスペクション[1]は，ソフトウェア品質の向上に大きく寄与することがさまざまな文献で報告されている[2]．インスペクションとは，日本語では単純に検査と訳されることが多いが，一般的には仕様やプログラムコードの品質確認検査，特に複数人数での目視検査を中心とした品質レビュー活動を意味する．外部音が遮断できる部屋に３～４人のエンジニアを集め，欠陥検出および除去方針まで決定する形態が最も一般的である．この際，モデレータと呼ばれる会議の主催者，対象を読み上げるリーダ（Reader），読み上げられた対象を検査するバリデータといった，役割（ロール）を決定し，検査を行う．

インスペクション，特にフォーマルインスペクションと呼ばれる品質検査手法は，その欠陥検査精度がきわめて高く仕様書作成時点の上流フェーズから実施することで，欠陥除去効率を著しく向上させるといわれている．しかしながら，大規模なプロジェクトではドキュメントの量が膨大であり，また人間による目視は作業負荷が高いため，すべてのプロジェクト・すべてのドキュメントについて実行するには非現実的な労力と時間を要する．この結果，現実にはドキュメントの欠陥を十分調べることができないまま上記のような問題を避ける機会を失うことになる．

筆者が率いているクオリティインスペクション部門は，インスペクションの専門家により構成され，第三者インスペクションの実施，および，その効率的な実施に必要なソフトウェア品質に関する仮説や品質測定のためのメトリクスの研究開発を主要な業務としている．第三者インスペクションの実施メンバは，過去にソフトウェアの（中間）成果物の作成に従事し，十分に教育され，スキルを認められた者から構成されるが，実施メンバとなった時点でソフトウェアの（中間）成果物の作成は担当しない．インスペクション専任のメンバとすることにより，通常のソフトウェア開発担当と比較して著しく多数の欠陥と接することができ，起こりがちな欠陥の傾向やその兆候に関する知識を蓄積することができる．

本稿では，日本 IBM で行っている第三者インスペクションの活動の一部を紹介することにより，第三者によるインスペクションの有効性や欠陥防止の効果を紹介することを目的としている．

開発メンバ自身によるインスペクションの課題

開発メンバが自身の作成したドキュメントを対象としてインスペクションを実施する場合の課題は３つある．１つは開発メンバの持つ欠陥指摘スキルである．２つ目は，客観性である．３つ目はインスペクション実施に要する時間である．

・欠陥指摘スキル

欠陥指摘の作業は，まず対象を理解すること，それから，実際に欠陥を指摘することの２つから構成される．

前者は普段からソフトウェア開発に携わっていれば，自然と身につくスキルであるといえるが，後者には，別のスキルが必要である．ソフトウェア開発者として作業を進めていれば，前者のスキルはソフトウェアに接しているかぎり身についていくが，後者のスキルを身につける機会はそれほど多くない．

• 客観性

欠陥指摘を開発メンバで実施するためには，自らが開発したものを客観的な立場で見直す必要がある．インスペクションの対象となるドキュメントに作成者自身が欠陥を指摘しようとすると，記述のない部分，および，曖昧な部分を開発メンバの持っている知識で補ってしまう．また，誤解されやすい部分への配慮が欠ける傾向がある．

• インスペクションに必要な開発メンバの作業時間

欠陥指摘を開発メンバが実施すれば，その間，開発メンバの成果物作成作業が止まる．また，1つ目の項目と関連するが，開発メンバは必ずしも十分な欠陥指摘スキルを持たないため，欠陥指摘に要する時間も長くなる傾向にある．短納期化が進んでいる現行のソフトウェア開発では大きな問題といえる．

▶日本 IBM での第三者インスペクションの取り組み

■ 概要

開発メンバ以外の外部の品質検査の専門家に，中間成果物，または，成果物の品質検査を依頼することを第三者品質検証，第三者インスペクションと呼ぶ．この形態はソフトウェア開発固有の形式ではなく，医療分野での検査部，検査技師の存在，製造業の品質保証部の役割と同様のものである．

ソフトウェア開発における第三者インスペクションは，先に述べたインスペクションの3つの課題を次のように解決する．

• 欠陥指摘スキル

第三者インスペクションのメンバは，自身は成果物作成作業はほとんど担当せず，成果物作成作業に従事する開発メンバと比較してきわめて多くの開発プロジェクトにおいて欠陥指摘を実施できる．たとえば，開発プロジェクトの開発期間が半年～1年であるとすると，その中でインスペクションの実施回数は数回であり，成果物作成作業に従事するメンバが欠陥指摘に関する知識を得る機会もこの程度の回数となる．これに対して，第三者インスペクションを実施するメンバは，組織内で同時進行しているプロジェクトのインスペクション活動に関与することができるため，半年～1年で数十から数百のプロジェクトを経験することが可能である．その結果，ドメイン知識，アーキテクチャ，一般的に陥りがちな落とし穴等，欠陥指摘に必要な知識を得る機会が，成果物作成作業に従事する開発メンバと比較すると，著しく多い．また，指摘後に正しい対処（修正）方法，欠陥除去時の優先順位（ある欠陥Aを除去した後に欠陥Bを除去しないとAが再発する可能性が高い，といった望ましい除去順序のこと）に関する知識を得る機会も，通常の開発メンバと比較して，より多く持つことができる．また，筆者らの第三者インスペクションでは，欠陥の存在個所の局所化にソフトウェアメトリクスを用いている．これも多数の対象ドキュメントに接しているからこそできる特徴の1つである．

• プロジェクト外部の第三者による客観性

第三者インスペクションのメンバは，インスペクション対象のソフトウェアの(中間)成果物の作成を行わない．そのため，開発メンバの思い入れ・正しいはずだという思い込み等がほとんど存在しない．そのため，仕様や処理の誤解釈を誘発する原因となる欠陥因子を網羅的に指摘することができ，欠陥自体とそれが及ぼす影響／被害のみに専念し客観性のある検査を行うことができる．

• インスペクションに必要な開発メンバの作業時間

第三者が実施することにより，インスペクションを実施している最中でも，開発メンバは他の開発作業ができる．多くの場合，成果物の作成作業とインスペクション作業を並行して実施することができるため，開発期間短縮につながることが多い．また，インスペクションの実施時間についても，開発メンバ自身が実施する場合と比較すると，第三者インスペクションのメンバのほうが早く完了することができる．

■ 実施時期と実施プロセス

図-1に示す通り第三者インスペクションは成果物の開発途中に実施する場合と，ある工程の終盤にまとめて実施する場合の両方がある．一般には，工程終了間際に一度だけ実施し，大量の欠陥が指摘されるインスペクションでは工程終了予定を過ぎても修正が終わらないことがあるため，小規模なインスペクションを複数回実施するほうが望ましい．小規模なインスペクションを複数回実施するメリットはほかにもある．工程の途中であっても，欠陥を早期に発見，修正すれば，その欠陥に依存する他の部分の欠陥を予防することができる．(中間)成果物ができた部分からインスペクションを実施することで，早期に欠陥を修正できる．欠陥指摘だけではなく，各開発メンバ間のばらつきや開発標準違反を早期に指摘することで，終盤の工程終了間際のインスペクションの負荷を低減することができる．

インスペクションを複数回実施する場合の実施回数は

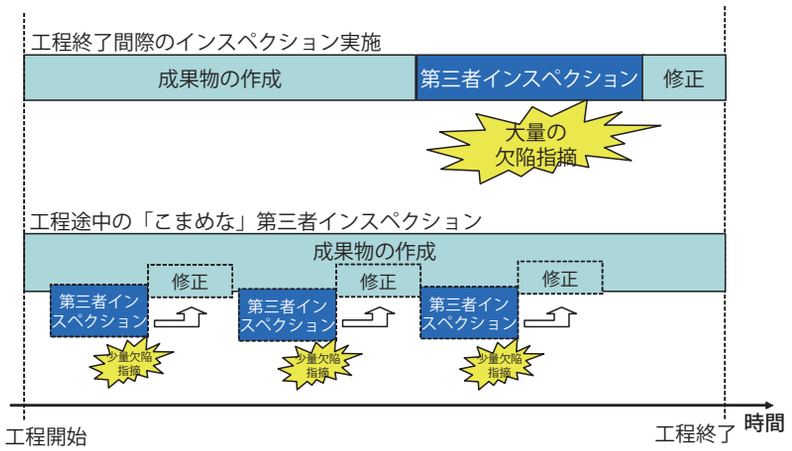

図 -1　第三者インスペクションの実施タイミング

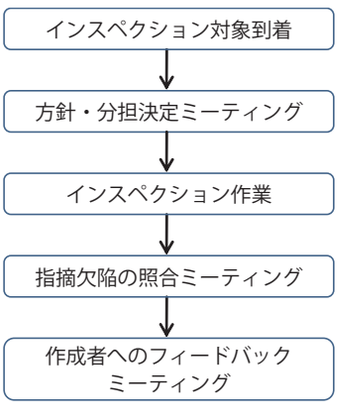

図 -2　第三者インスペクションのプロセス

プロジェクト規模に依存する．開発期間が長期にわたる大規模システム開発においては，工程途中のインスペクションと終了間際のインスペクションの両方を実施する場合もある．

図 -2 に第三者インスペクションのプロセスを示す．図 -2 は 1 回のインスペクションのプロセスを表している．開発プロジェクトよりインスペクション対象（仕様書やソースコード）を受け取る．次に，第三者インスペクションチームにおいて，インスペクション対象をもとに方針と分担を決定するためのミーティングを実施する．インスペクション作業は，複数のインスペクタによる目視検査であり，品質要求に応じて，会議形式で同期して実施されるフォーマルインスペクション，バディーチェック（2 人 1 組で実施する確認検査），過去の欠陥事例，傾向との照合を実施するパターン検査，複数のインスペクタで非同期での実施，から選択する．また，インスペクション作業にはメトリクス計測が含まれ，メトリクスによる品質推定やそれに基づく重点調査等が実施される．インスペクション作業が完了したら，それぞれの指摘欠陥を照合するミーティングを実施する．ここで，それぞれの指摘結果を収集，整理し，指摘欠陥のリストと対策のアクションを開発プロジェクトにフィードバックする．

▶実施例

本章では，前章までで述べた実施形態でインスペクション対象ごとに，筆者らが具体的にどのようなインスペクションを実施しているかを一部紹介する．

■要件定義フェーズのドキュメント

要件定義フェーズのドキュメント（要求仕様書等）は自然言語や特定の記法で記されるのが通常であり，欠陥指摘において留意すべき点がある．本節では，その留意点を示すとともに，その他の着眼点として文に表れる曖昧さ，および，文のコピー率を紹介する．

留意点

1. 全体把握すること

要件定義フェーズのドキュメントでは，最初からドキュメントの特定の部分を詳細に理解しようとしてしまい全体を見失いやすくなる傾向がある．1 種類のドキュメントを見て全体を OK，NG と判定してしまうのが最たる例である．さまざまなドキュメントから総合的に判断して要件を表現できているか検証すべきであるが，一般的に大量の要件定義フェーズのドキュメントを目の前にすると中心となるシナリオだけ選択してインスペクションしようとしてしまうことが少なくない．筆者らは，すべての種類のドキュメントから均一に対象を選んでおり，特に作成チームまたは作成者別に対象を選んでいる．

2. 記述の目的，存在理由を常に確認すること

ドキュメントの一部を選び，大勢で目視しながら検査する場合，該当個所がドキュメント全体においてどのように位置づけられるのか，プロジェクト全体の目的を満たすかどうか，という点を忘れてしまう場合が少なくない．インスペクション作業の実施中に“これは要件か？”，“これはプロジェクトの目的達成のために必須か？”，“これが実現できなかったらどうなる？”といった質問を何度も繰り返す必要がある．また，チェックリストをはじめとするインスペクションのツールにこのような確認ポイントを列挙しておき，インスペクション時に利用することもある．

文の曖昧さ

仕様書を構成する文の曖昧さを文に含まれる読点の数から推測する方法を例として述べる．1 文に読点が多い場合，通常その文は長くなっていることが多く，欠陥

が含まれる可能性が高い．筆者がテキストマイニングの専門家の方にアドバイスをもらったのがこの点を調べるようになったきっかけであるが，作成者が確信をもって書いていない文，作成者がよく対象を理解していない文は叙述的になる傾向があり，読点が多用される長い文となることが多い．これは要件が理解されていない，または確定していないことを，可能なかぎり表現しておこう，記述しておこうという作成者の思惑の表れの１つである．実際に１文中の読点の数を数え，その数が多い文を調べると，高確率で欠陥を検出することができる．このような文には，論理の矛盾，記述どうしの不整合，一貫性欠如といった要件定義フェーズのドキュメントに典型的に含まれるものが多数指摘できる．

処理機能記述の例）　長さ

1.3，ファイルIDのチェック	15
1.1で取得したファイルIDが指定されたファイルIDであること	31
1.3.1，OKの場合，	12
チェック結果のメッセージフラグ="OK"	20
へ.	6
合，	12
チェック結果のメッセージフラグ="E"	19
チェック結果のメッセージリストにエラーメッセージ（ID="xxxx01"）をセット	41
チェック結果を返し，処理を終了する.	18
2.1，タイトル行から項目数と初日を読み込み.	23
初日 = cDate	8
2.2，需要カレンダーの日付チェック	18
メソットcalCheck（引数A, 引数B, 引数C, 引数D）を呼び出す，チェック結す.	48
2.2.1，OKの場合，	12
チェック結果のメッセージフラグ="OK"	20
へ.	6
合，	12
チェック結果のメッセージフラグ="E"	19
チェック結果のメッセージリストにエラーメッセージ（ID="xxxx02"）をセット	41
チェック結果を返し，処理を終了する.	18

処理機能記述の例）　長さ

次の処理へ.	6
次の処理へ.	6
初日 = cDate	8
1.3.1，OKの場合，	12
1.3.2，異なる場合，	12
2.2.1，OKの場合，	12
2.2.2，異なる場合，	12
1.3，ファイルIDのチェック，	15
2.2，需要カレンダーの日付チェック	18
チェック結果を返し，処理を終了する.	18
チェック結果のメッセージフラグ= "E".	19
チェック結果のメッセージフラグ= "E".	19
チェック結果のメッセージフラグ= "OK".	20
チェック結果のメッセージフラグ= "OK".	20
2.1，タイトル行から項目数と初日を読み込み.	23
1.1で取得したファイルIDが指定されたファイルIDであること	31
チェック結果を返し，処理を終了する.	18
チェック結果のメッセージリストにエラーメッセージ（ID="xxxx01"）をセット	41
チェック結果のメッセージリストにエラーメッセージ（ID="xxxx02"）をセット	41
メソットcalCheck（引数A, 引数B, 引数C, 引数D）を呼び出す，チェック結果	48

図 -3　要件定義フェーズのドキュメントに含まれるコピーされた文

文のコピー率

文を安易にコピーすると，文のコピー先個所の前後に欠陥が生じやすくなる．作成者が文をコピーする際には，その内容をあまり理解していない場合が多く，これが不整合や矛盾の原因となる．そのため，コピーした文を特定し，周辺との整合性を調べれば，欠陥指摘につながる可能性が高い．要件定義フェーズのドキュメントに含まれるすべての文を文の長さでソートすると比較的簡単に文のコピーを見つけることができる．図 -3 はそのような，文のコピーを探そうとしている例であり，ドキュメントに含まれるすべての文とその長さを計測したもの（図 -3 上），と文の長さでソートしたもの（図 -3 下）である．文の長さでソートすることにより，コピーされた文が前後に並ぶため，文のコピーを比較的容易に検出することができる．

■設計フェーズのドキュメント

設計フェーズのドキュメントも要件定義フェーズのドキュメントと同様に，内容を深く理解しようとしすぎず全体を把握することが重要である．また，設計フェーズのドキュメントの記述の詳細さも重要である．

全体把握

要件定義フェーズのドキュメントと同様に，設計フェーズのドキュメントについても内容を深く理解しようとして，インスペクション作業そのものが遅々として進まない場合がある．要件定義フェーズと同様に，インスペクタは微細な内容に注視しすぎないよう注意する必要がある．筆者らの部門に配属された新人には，次のように指示をすることによって，注視しすぎない感覚を養ってもらうことにしている．メソッド記述や処理機能定義の中から“場合”と“時”という文字列を検索することによって，分岐やエラー処理を発見し“そうでない場合”が記述されているかどうかを確認する．一般に“～の場合”，“～のとき”の後ろには必ず“そうでない場合”，“～

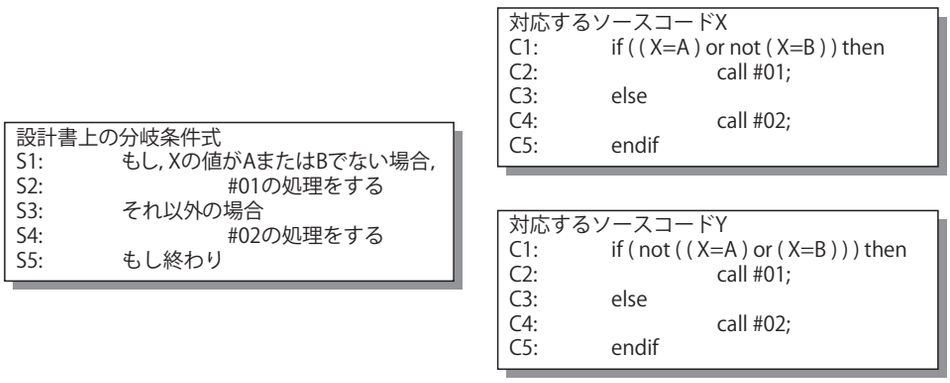

図 -4 設計ドキュメントとソースコードの対応例

でないとき ” が記述されることが期待される．しかしながら，設計フェーズのドキュメントではこれらの記述が欠落している場合が多いためである．この作業により正常系の定義だけでなく異常系(例外，エラー処理)の定義も書いてあるかを確認することができる．また比較的高い割合で欠陥が指摘できるとともに，微細な内容を注視しない訓練になる．

記述の詳細さ

設計フェーズのドキュメントでは，内容の漏れや矛盾に加え，どの程度まで詳細に書けば完了したと判断できるかが難しい．いいかえると，何が揃っていれば，設計書として十分かという点が曖昧である．そこで，その対象ドキュメントからテストケースが導出できるかどうかを判断基準としている．実際にテストケースを導出するにはそれなりのコストが必要になるが，設計フェーズのドキュメントの一部を複数選び，その記述でテストケースが導出できるかどうかを確認しテストケース数の概算を算出する．選んだ部分ごとにテストケースを集計し，それらの間で極端にテストケース数が多い，あるいは，少ない部分がないかを確認する．もしも存在する場合には，その原因を調査する．

■ ソースコード

コードインスペクションでは，ソースコード全体の目視による全体確認，設計ドキュメントとの対応から分かる情報，ツール等を利用して定量的に計測できる情報，の 3 点を中心に検査する．

全体確認

ソースコードファイルの中には，自動生成されたコード，過去のバージョンのもの，パッケージ等を流用しているものが含まれる．インスペクション対象のソースコードファイル全体について，今回のバーションで作成されたものか，人手によって作成されたものかどうかをまず把握し，人手によって作成された部分を優先する．自動生成されたコード，過去のバージョンのソースコードも場合によってはインスペクションの対象となるが今回のリリースに向けて書かれた部分の優先順位が高い．対象が決まったら，以下のような観点でインスペクションを進めていく．

1. ソースコードファイルの規模(総行数)
2. 構造化されているか
3. 1 人で書いているものか，複数人で修正しているか
4. 複雑なロジック，入出力が存在するか
5. 末尾にコーディング工程終盤に変更したロジックが存在しないか

設計ドキュメントとの対応

ソースコードファイルをエディタで開き，末尾から先頭に向けて，ざっとスクロールするとこれら 5 項目を大まかに知ることができる．特に，書き方が異なる部分を見つけたら，そこは他と異なる開発メンバによる編集の痕跡と推測される．一部を選び出して詳細に見る場合には，選び出すための観点となり得る．

ツールにより計測する定量的情報

ソースコードと設計書との対応を確認するにはコストがかかるが，少なくとも数個所を選び，確認する．その例の 1 つを図 -4 に示す．図 -4 は，設計書の記述とそれに対応する可能性のあるソースコードの対応を表している．図 -4 の中で特に注目しなければならないのは，S1 と C1 の対応である．図中の対応するソースコード X においては，“ もし，X の値が A または B でない場合 ” を “if ((X = A) or not (X = B)) then” と実装している．一方ソースコード Y では，“if (not ((X = A) or (X = B))) then” としている．どちらが正しいかを図 -4 だけで判断することはできないが，設計ドキュメントの他の部分

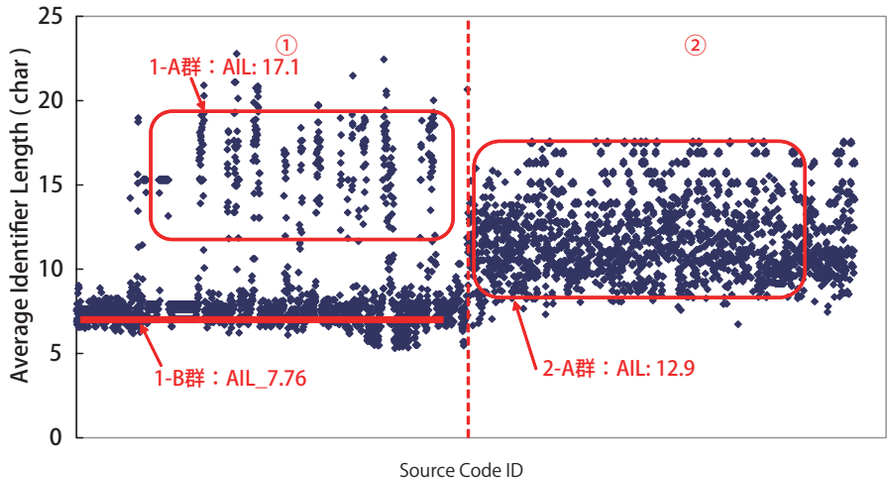

図 -5　ソースコードファイルごとの平均変数名長（AIL）（文献 3）より引用）

AIL	プログラマのプログラミング言語経験	注目点の例	図 -5 中の対応個所
8 文字以下	COBOL が長い	例外処理（try ～ catch）の漏れ	1-B 群
9 ～ 17 文字	C/C++ が長い	Null Pointer エラー	2-A 群
18 文字以上	Java のみ	リソース解放忘れ，new の多用	1-A 群

表 -1　AIL とプログラマとプログラミング言語の経験

を見ればどちらが正しいかを推測することができる．

ソースコードメトリクス等，ソースコードから定量的に計測できる情報は多い．ソースコードメトリクスを活用し，コードインスペクションを実施しており，その詳細は文献 2）でも報告している．ここでは，コードインスペクションに役立つメトリクスである平均変数名長（AIL : Average Identifier Length）を紹介する．

コードインスペクションの対象となるソースコードファイルごとに，そのファイルに含まれる変数名の文字列の長さ（AIL）の平均を算出する．その値をプロットすると図 -4 のようになる．図中の縦軸は AIL である．横軸はソースコードファイルに対応する．グラフ中の 1 つの点はあるソースコードファイルの AIL に対応する．横軸のソースコードファイルは同一パッケージ，同一モジュール，同一ディレクトリ等のものが隣り合うように並べる．

図 -5 から得られる情報は多くあるが，そのうちの 1 つは，ソースコードを書いたプログラマがどのようなプログラミング言語の経験を持っているかである．プログラミング言語の経験が分かれば，典型的なミスを推測することができ，効率的な欠陥指摘ができる．たとえば，AIL が 8 以下である場合には，COBOL の経験が長いことが推測される．図 -5 の例では，1-B 群は COBOL の経験が長いプログラマが書いたソースコードである可能性が高い．また，AIL が 9 以上 12 以下の場合には，C/C++ の経験が長いプログラマである可能性が高い．図 -5 の例では，2-A 群は C/C++ の経験が長いプログラマが書いた可能性がある．AIL が 18 以上になると Java 以外のプログラミング言語の経験がないプログラマの可能性が高くなる．図 -5 の例では，1-A 群に対応する．AIL の閾値は筆者らが多数のソースコードを対象に経験的に得たものであり，対象ソフトウェア，プロジェクト，コーディングルール等によって調整する必要のある値である．

これらのプログラマのプログラミング言語の経験の情報をもとに，表 -1 のような注目点でコードインスペクションを実施する．今回，ソースコードは Java で書かれているものとする．COBOL の経験の長いプログラマが担当したソースコードでは，COBOL には存在しない例外処理（try ～ catch）構文の漏れに着目する．C/C++ 言語の経験の長いプログラマならば，Null Pointer エラーに注目する．また，Java の経験しかないプログラマならば，Java 言語ではランタイムが実施してくれることの多い，リソース解放忘れや，new の多用による性能劣化の可能性がないかに注目する．

AIL のようなメトリクスから常に期待通りの情報を

得られるとは限らないが，筆者らは自動計測が可能なあらゆるメトリクスを収集し，欠陥との関連を恒常的に観察し，その知見を積み上げている．その理由は，メトリクスがより早い欠陥指摘につながる場合が多いことを経験的に確認しているからである．その原理／原則はモデル化とそのモデルの効果と限界を実証的に検証することを繰り返すエンピリカルソフトウェア工学の手法と同様である．

■工程間，工程内でのばらつき

工程間，同一工程内でのばらつきに注目することにより，得られる情報は多い．要件定義，設計(基本設計，詳細設計）では，対象成果物が段階的に詳細化されていくため，一般には，文章の記述粒度が粗い状態から細かい状態へと変化し，その規模が大きくなる．そのため，文書のページ数や行数は増加する．しかしながら，時間がなくなるなど，何らかの外圧があった場合には，同一の要件や機能に関する記述が前の工程の文章より減ってしまうことがある．ここには潜在的な欠陥が含まれる可能性が高い．

同一工程内でも，工程の最初と最後で記述の粒度や詳細化の度合いが大きくばらついている場合には，時間がなくなるなど，何らかの外圧があった可能性が高い．多くの作成者は，自身が理解しているもの，主要な業務から書き始め，理解しにくい例外処理等は最後に記述する傾向がある．この傾向のある文書では，最初に作成したページと最後のページを見比べることにより，時間が差し迫った状態で記述したかどうかを判断することができる．時間が差し迫った状態で記述しているようであれば，まずは最後の部分に欠陥が含まれていないかを調査する．

▶今後に向けて

■インスペクションの今後の課題

本稿では，開発メンバが他の開発メンバの作成したドキュメントに存在する欠陥を指摘するインスペクションの課題を述べ，欠陥指摘スキルを持った開発メンバ以外の第三者によるインスペクションの実施により期待される効果を述べた．また，筆者らが実践している第三者インスペクションの実施方法と具体例の一部を紹介した．

第三者インスペクションは通常のインスペクションと排他的な関係にはなく，併用することができる．また，対象プロジェクトの開発プロセスによらず適用することができる．アジャイル開発においても，今回紹介したような方法で短期間のビルドサイクルにあわせたインスペクションを実施することができるであろう．

第三者インスペクションの課題の１つはインスペクタの確保である．第三者インスペクションの実施メンバであるインスペクタ間には均質なスキルと分担が可能な経験／スキルが必要になるが，そのようなインスペクタの確保は困難であるのが実情である．現状，インスペクタになるためには，開発経験が必要であり，対象プロジェクトとなり得るドメインや開発方法論等に精通している必要もある．これらのスキルの体系化や役割分担は今後の課題である．

■ヒューマンエラーの認識と対応

ヒューマンエラーとは，システム最終成果に対して確実に欠陥を埋め込むことになる，開発者が当然行うべき必要な行動の欠如または誤りのことである．ヒューマンエラーの原因を突き止めないかぎり，今，目の前で除去した欠陥が必ず再発する．換言すれば，もし誰かが犯したヒューマンエラーを，自動的に検出／指摘することができたら，確実に埋め込まれてしまう未来の欠陥を事前に予防することができると期待できる．たとえば前述のAIL のような属人的指標は，開発作業の開始に先立って行われるべき標準化・ガイドの質に依存して数値が推移する「兆候」の１つであるともいえる．長期間これらの定量データを収集し欠陥を予測すること，および予測した欠陥種類を参考に目視インスペクションを行うことで，検出効率を高めることができると筆者は考える．なぜなら“存在するはずの欠陥”を探すため，インスペクション作業の生産性が飛躍的に向上するからである．今後これらヒューマンエラーに関する研究とその兆候データの収集は日本から諸外国に向けて発表すべき研究として継続していく予定である．短時間を武器にすることと“計測は力なり”という考え方を広く世界に広めたいと考えている．

参考文献

1) Fagan, M. E. : Design and Code Inspections to Reduce Errors in Program Development, IBM Systems Journal, Vol.15, No.3, pp.182-211 (1976).
2) Gilb, T. and Graham, D. (著), 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, ISBN 4-320-09727-0 (1999).
3) 細川宣啓, 渡辺千恵子, 原佑貴子, 徳　隆宏 : 定量メトリクス・データによる Defect Prevention の実践, ソフトウェアテストシンポジウム東京 2009 予稿集, pp.134-139 (2008).

(平成 21 年 3 月 9 日受付)

細川 宣啓　CARVIN@jp.ibm.com

1992 年日本アイ・ビー・エム（株）に入社．SE を経て 1999 年より同社品質保証組織にて Quality Inspection チームを立上げ．品質工学および上流フェーズ欠陥検出技術の社内外への展開を手がける．IEEE Associate Member．経済産業省 IPA/SEC 価値指向マネジメントワーキンググループメンバ．2007 年 ASTA Korea，2008 年 4WCSQ にて発表．