第 18 章　レビューについて

レビューの種類

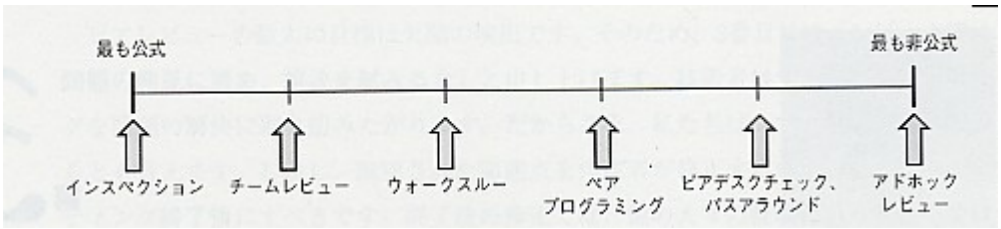
ソフトウェアのレビューとは、要件定義書や設計書、ソース・プログラムなどの成果物が作成された直後に、作成者も含めた何人かの人がその成果物をチェックして、欠陥を発見する作業をいう。

当然発見した欠陥は、その直後に修正されなければならない。これを通して、ソフトウェアの品質の向上を図ることがレビューの最終的な目的である。しかしレビューそのものの直接の目的は、この「欠陥の発見」に限定することが重要である。つまり「いかにしてその欠陥を取り除くのか」ということを、レビューのテーマにするべきではない。それについては、また後で述べる。

レビューには、いくつかの種類がある。アメリカの学会である IEEE（The Institute of Electric and Electronic Engineers）が決めたソフトウェア・エンジニアリング関係の標準に IEEE std 1028-2008（IEEE Standard for Software Reviews and Audit）というものがあって、ここでは次の 5 つについてその内容やレビューの方法が取り上げられていた[1][IEEE08a]。

① マネジメント・レビュー
② テクニカル・レビュー
③ インスペクション
④ ウォーク・スルー
⑤ 監査

またカール・ウィーガースはその著「ピアレビュー」の中で、図表 18-1 として転載した図を掲載してさまざまなレビューの種類を明示している。



図表 18-1　レビューの種類
（[WIE04] 36 ページより）

IEEE の標準にあるマネジメント・レビューとテクニカル・レビュー、および監査はここで述べたいこととはやや異なる目的を持つレビューであり、ウィガースがいうペアプログラミングはアジャイルソフトウェア開発のエクストリーム・プログラミングのところで議論した[2]。したがって、ここではインスペクションとウォーク・スルー、およびアドホック・レビューについて取り上げる。

アドホック・レビュー、つまり非公式のレビューは、コーヒーのベンディング・マシンの前

[1] IEEE　Std 1028-2008 は、この原稿を修正している時点（2017 年（平成 29 年）1 月 13 日）では、廃止されてしまっている。
[2] アジャイル・ソフトウェア開発については、第 14 章で議論した。

のベンチに座って、買いたての缶コーヒーを飲みながら友達にプログラムを見てもらい、間違いを指摘してもらうようなものから始まる。これでも予想以上の充分な効果があると、ワインバーグはその著書の中で述べている[WEI71]。一般にソフトウェア技術者やプログラマは、成果物を作った直後に自分自身でその成果物に埋めこんだ欠陥を発見することがたいへんに苦手である。非公式のレビューでもこの問題の解決に、充分な効果がある。

しかし公式のレビューは非公式のレビューと比較すると、もっと効果が大きい。この章ではまず、最も公式のレビューである「インスペクション」から議論を始め、その後でもっと簡便なウォーク・スルーなどについての議論に進みたい。

インスペクションの手順

マイケル・ファガン（Michael E. Fagan）がインスペクションについての有名な論文[FAG76]を書いてから、この原稿を書いている時点（2005 年（平成 17 年）7 月）までで、およそ 30 年が経過した。この間に、レビューやインスペクションについての多くの本が書かれ、そのうちの何冊かが日本語に翻訳されている（例えば、[FRE82]、[YOU89a]、[GIL93]など）。しかしまだ引き続き、インスペクションについての新しい本が書かれている（例えば、[WIE02]）。

これは、何を意味するのだろうか。多分、「どうすればインスペクションをより効率良くすることができるのか」などについての研究がまだ盛んに行われていることに加え、その一方で「正式の」インスペクションが期待しているほど広がっていない、つまり別のいい方をすれば、アメリカでもまだインスペクションの仕方を変えることでもっと欠陥の除去を効率よくできると考えている人たちがいる、ということだと私は考える。

しかしインスペクションの手順そのものは、ファガンの論文[FAG76]からカール・ウィーガースの最新の本[WIE02]に至るまで、表面的にはほとんど変わっていない。まず、その手順についての議論から始めたい。

何かの成果物を完成してインスペクションにかけたいと考えた人（以下では「作成者」と呼ぶ）は、インスペクション・リーダ（以下では単に「リーダ」と呼ぶ）を務められる人たちのところに行ってインスペクションの実施を願い出ることから、一連のインスペクションの手順が始まる。

リーダはその成果物がインスペクションに耐えうる状態まで仕上がっていることを確認の上、チェックをしてくれる人たち（以下では「インスペクタ」と呼ぶ）を選ぶ。この時リーダはインスペクタとして、特定の技術領域に強い人を選ぶことがある。その技術領域の立場から充分なレビューを行いたいというリーダの考えの表れである。インスペクタの人数は、そう多くある必要はない。多分 4 人ぐらいまでで充分だろう。この人たちを集めて、リーダは最初にキックオフ・ミーティングを開く。

キックオフ・ミーティングでリーダは、インスペクタにチェックの視点を明確に指示することがある。あるいは作成者からインスペクタに、成果物についての簡単な説明を行うことがあるかもしれない。いずれにしろキックオフ・ミーティングが終われば、インスペクタが個々に成果物のチェックを始められる状態にする。それがキックオフ・ミーティングの目的である。

その 2 日後位に予定されるインスペクション・ミーティングまでの間に、それぞれのインスペクタはその成果物を丹念にチェックし、欠陥の発見に努める。この発見した欠陥を、インスペクタがインスペクション・ミーティングで発表することになる。

インスペクション・ミーティングでリーダは最初に、個々のインスペクタに「個別のチェッ

クを行ったかどうか」を確認し、行った場合はその作業にかけた時間を聞いて、別途任命した「書記」に記録を依頼する[3]。個別のチェックを行わなかったインスペクタがいれば、リーダはその人に書記を依頼するか、あるいはインスペクション・ミーティングへの参加を拒否する。事前のチェックを行わない状態でミーティングに参加することは、時間の無駄だからである。複数のインスペクタが参加できなくなってインスペクション・ミーティングが意味をなさなくなれば、リーダは別途ミーティングを設営し直す必要がある。

リーダはミーティングで特別の「読み手」を指名し、その人が資料を声を出して読みながら議論を進めることがある。読み手がいることで、議論のポイントをより明確にすることができる。インスペクタはそれぞれが発見した欠陥を報告しあうことで、全体としての欠陥の発見に努める。単にこれまで発見した欠陥を報告するだけではなく、他のメンバーの報告を聞きながら新たな欠陥の発見に努めることが重要である。書記は、この報告された欠陥を余さず記録する。

ミーティング期間中のテーマは、前述の通り欠陥の発見に限定する。発見した特定の欠陥をどう除去するかというようなことに話題が移ることがあるかもしれない。しかし、それは厳禁である。そのようなことがあればリーダは、毅然とした態度でその話題を打ち切らなければならない。欠陥の発見以外のことをテーマにすることは、時間の無駄以外の何者でもない。また発見した欠陥をどう取り除くのがよいかということは、作成者が一番よく知っている。言うまでもなく、作成者がこの成果物を作成したからである。したがってこれをミーティングのテーマにすることは、作成者の知識と経験を充分に認めていないことにもなる。

ミーティングの時間は、2 時間を超えるようなことがあってはならない。「人間が集中力を持続できるのは、2 時間が限度」といわれているからである。ロバート・グラスはその著の中で、「インスペクションは 1 時間でくたくたになる」と書いている[GLA03]。

インスペクション・ミーティングの後、作成者は「欠陥」と指摘されたものが本当に欠陥であるのかどうかをチェックし、欠陥であればそれを除去する。リーダは、作成者のこの作業を管理する。

さらにリーダは、このインスペクションの報告書を作成し、管理者に報告する。

以上が一連の、一般的なインスペクションの手順である。

インスペクションを成功させるための組織の文化

インスペクションに限らず一般にレビューは上で述べたように、同僚に間違いを指摘してもらう作業である。単純に考えれば「人間とは間違いを犯すものであり、ソフトウェアは本来間違いを許されないものであるから、ソフトウェア技術者は自分の間違いを見つけて除去するためには手段を選ばない」とするのが正当である。

しかしソフトウェア技術者は本来「誇りの高い」人たちであり、同僚に自分の間違いをしられるようなことを快く思わない傾向がある。ここに、インスペクションをはじめとするレビューの難しさがある。ソフトウェアの開発組織の中に、同僚に弱みを見せ、間違いを指摘してもらうことを認める文化がなければ、インスペクションなどを行わない方がよいとウィガースを初めとする人たちは指摘している（[WIE02]、[GIL93]）。ウィガースのこの指摘はある意味で逆説であり、「このような文化の組織は、何とかその文化を変えてインスペクションを行うべき」

[3] ここで記録した内容は、ソフトウェア・メトリクスでの分析の対象になる。ソフトウェア・メトリクスについては、第 9 章で述べた。

と、彼は言いたいに違いない[4]。

この文化に関連し、インスペクションなどを実施する時に注意しなければならないことがいくつかある。例えば、次のようなものである。

- 作成者を非難するようなことを、一切言ってはならない。
- 淡々と誤りだけを指摘し、それ以外のことにはふれない。
- 管理者は、レビュー・ミーティングには出席しない。

管理者とは部下の昇級や昇格、あるいは解雇などに責任と権限を持っている人である。だから管理者がレビュー・ミーティングに出席するとアメリカなどでは、レビューアは自分が発見した欠陥がいかに重大なものであるかということを管理者に印象づけようとし、逆に誤りを埋め込んだ作成者はそれがさほど重大なものではないことを管理者に印象づけようとするようである。これは本来のレビューにとって必要なことでも好ましいことでもなく、さらにこれが高じると作成者がレビューを受けようとする意欲をそぐことにもなりかねない。これは本来、レビューの実施に当たって避けなければならないことである。

ソフトウェア開発組織にこの文化が根付いていることが、インスペクションなどを成功させるために不可欠なことである。これは管理者と次に述べるインスペクション・リーダの責任である。

韓国やミャンマーでは、年少のものが年長者を厚く敬う習慣がある。このこと自体はすばらしいことだが、逆にレビューで年長者が犯した誤りを年少者が指摘できないということが起きている。これを払拭できなければ、このような文化を持つ国ではソフトウェア産業を健全に育成することが難しいだろう。

インスペクション・リーダの役割

インスペクションの実施に当たって、インスペクション・リーダの役割はたいへんに大きい。インスペクションが成功するかどうかは、このリーダの進め方によるといっても言いすぎではない。

リーダが行うべきことは、既に述べた。あえてここでそれを繰り返すことはしないが、リーダは作成者への配慮を欠いてはならない。仮に作成者への配慮が十分ではないインスペクタがいれば、リーダは即座に毅然とした態度でそのインスペクタに注意を与えなければならない。さらにそれでそのインスペクタの態度が改まらない場合、そのインスペクタを退席させるとか、レビュー・ミーティングを中止するとかまで行わなければならないかもしれない。

またリーダは、インスペクションの効率向上に努めなければならない。人間が集中力を持続できる時間に限界があることは、既に述べた。リーダはインスペクタが集中力を持続している間に、できるだけ多くの欠陥を発見するよう努めなければならない。そのために必要なら、ゲーム感覚で単位時間内に発見できる欠陥数を競うような文化が、その開発組織にあっても良いと指摘する人もいる。

このようなことを実現するために、アメリカではインスペクション・リーダを養成するセミナーを開いている組織があるようである。しかし残念ながら、日本でそのようなセミナーがあるという話を聞かない。日本はアメリカに比べて、インスペクション実施の割合が高くないの

---

[4] 特に作成者が若いソフトウェア技術者の場合、インスペクション・リーダとインスペクタはその技術者の将来に配慮して、そのやる気や発想力をつぶさないように充分に気をつけてほしい。

かもしれない。あるいはインスペクションの重要性についての認識が、まだ充分ではないのかもしれない。

良いカリキュラムが作られていて、それに基づいたセミナーがあって、それを受けることでインスペクション・リーダとして必要なことが容易に学べるということは、素晴らしいことである。しかし上で述べたように、インスペクション・リーダとして学ばなければならないことは必ずしもむずかしいことではない。つまり外部のセミナーが利用できないなら、社内でインスペクション・リーダを養成することは可能である。ソフトウェアの品質を高めたいと考えている企業は、何らかの方法で積極的にインスペクション・リーダを養成し、必要なら企業文化を変えて、真剣にインスペクションの実施に取り組まなければならない。

ウォーク・スルーについて

インスペクションと並ぶもう一つの公式のレビューに、ウォーク・スルーがある。ウォーク・スルーは、次のような手順で行われるのが普通である。

成果物を作成した作成者がその成果物をウォーク・スルーにかけたいとき、作成者が直接レビューアを選び、レビュー・ミーティングへの参加を要請する。レビューアが決まると作成者は成果物をレビューアに渡し、その 2 日ぐらい後にレビュー・ミーティングを開く。個々のレビューアはそれまでの間に、その成果物をチェックして欠陥の発見に努める。

レビュー・ミーティングでは作成者が主導して、欠陥の発見に努める。インスペクションの場合の「読み手」と同じように、作成者が典型的なケースについての説明を行うことで議論のポイントを明確にすることもある。発見された欠陥は作成者自身か、あるいは作成者が依頼した書記が記録する。レビュー・ミーティングの時間も、2 時間を限度とする。

ミーティングの後作成者は管理者宛ウォーク・スルーの報告書を作成し、さらに発見された欠陥の除去に努める。

結局ウォーク・スルーは、インスペクションの簡易版ということができる。

レビューでどれくらいの欠陥を発見できるか

それではこのような公式のレビューで、どれくらいの欠陥が発見できるのだろうか。

いずれもうまくいった場合という条件の下での話であるが、インスペクションでは 80%の欠陥の発見が可能であり、ウォーク・スルーでも 60%の欠陥の発見が可能との報告がある。

ちなみにテストの場合、一段階だけでのテストでの欠陥の発見率は 50%以下、通常は 30%程度と報告されている。テストと比較すると、レビューでの欠陥発見はたいへん効率的であるということができる。

インドにあるモトローラのソフトウェア開発組織では、欠陥発見率が 85%以下になったことが明らかになれば、レビューの方法が良くないと判断して、その作業を詳細にチェックして改善を行い、欠陥発見率の向上に努めるという。

しかしレビューの効果は、この欠陥発見率の高さだけではない。先にも述べたように、レビューでは欠陥が埋め込まれた直後にそれを発見することができる。一方のテストでは、そうはいかない。

仮に要件定義の段階で欠陥が埋め込まれ、それが発見されないまま後の工程に持ち越されると、間違った要件定義書を基に正しい設計を行ってもその設計は間違いであり、その設計を基に正しくプログラミングしても､そのプログラムは間違っているということになる｡テストは、

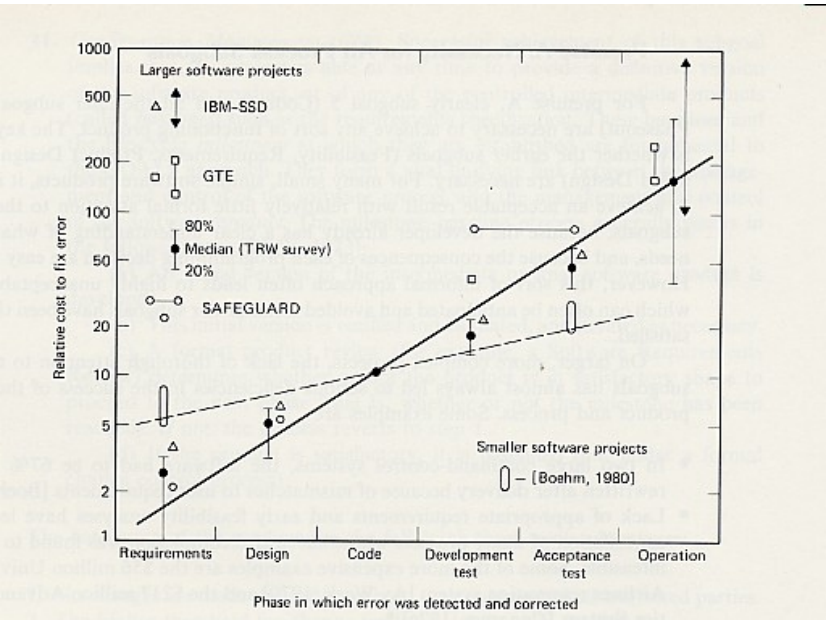
このプログラム作成後の段階で欠陥を発見する。これを正しくするためには、要件定義の段階まで遡って作業をやり直し、その後設計をやり直し、プログラミングもやり直して、再度テストして欠陥が取り除かれていることを確認することになる。つまり、たいへん大きな手戻りが発生することになる。さらにこれらの間違った要件定義書や設計書を参照して他の部分の設計などが行われていると、手戻りの範囲はもっと広くなる。

ソフトウェア・エンジニアリング界の大御所の一人であるバリー・ベーム（Barry w. Boehm）は、その著「Software Engineering Economics」の中で、図表 18-2 として転載した図表を示して、この事実を明記している[BOE81]。

つまりベームは、要件定義の段階で持ち込まれた欠陥をそのフェーズの間に修正する場合のコストを 1 とした場合、設計段階での修正コストは 5、プログラミング段階では 10、稼働開始後は 100 になるといっている。

レビューではいうまでもなく、要件定義段階の欠陥は要件定義段階で発見して取り除くことができる。全体の欠陥の中プログラミングの段階で埋め込まれるものは 35%程度であり、残りはシステム分析や設計などの上流工程で埋め込まれるとの指摘もある[JON96]。この観点からのレビューでは、手戻りをテストの場合に比べるとたいへんに少なくすることができる。

トヨタは自動車を作る際の生産性向上のために、最終の製品を作ることに関わりを持たない作業を極力排除するという[LIK04]。理想は、全ての工程の全ての作業が最終製品を作るための作業であることが望ましい。いかにトヨタといえどもこれは不可能だが、しかしトヨタの社内では常にこの方向に向けての作業工程の改革が進んでいる。



図表 18-2　開発フェーズの進展に伴うエラー修正コストの増加
（[BOE81] 40 ページより）

最終製品作りに役立たない作業を行うことは、小さなマイナスである。しかし手戻りは、明らかに大きなマイナスである。手戻りを少なくすることは、ソフトウェア開発の生産性を向上させる上で、たいへんに効果が大きい。

このようにレビューの役割は、欠陥除去のためだけでなく開発生産性向上のためにもたいへんに大きい。

非公式のレビューについて

非公式のレビューとはインスペクションやウォーク・スルーのような手続きを踏まず、友人やチームメンバーに依頼して成果物をレビューしてもらうことをいう。何もコーヒーのベンディング・マシンの前だけに限ることはない。

この非公式のレビューでも、欠陥除去のためにはたいへんに有効である。ワインバーグはその名著「プログラミングの心理学」の中で、非公式のレビューを繰り返して、結果として欠陥をゼロにしたある開発組織の話を書いている[WEI71]。

個人的な話で恐縮だが、私もプログラマとしての現役の時には、いつも非公式のレビューをお願いする相棒がいた。欠陥があることが明らかなのに自分自身でそれを発見できない時、私はよく彼を私の机のところに来てもらい、彼に私のプログラムでの処理の仕方を説明した。彼は時々、たいへん鋭い質問をしてくれた。その質問を契機に、欠陥を発見できることが多かっている。

パーソナル･チェック

ワッツ･ハンフリー博士が提唱している PSP（Personal Software Process）の中に、パーソナル･チェックがある。設計書やプログラムなどの成果物を作成した人が、その成果物の完成直後にチェックリストを使って自分自身で行うレビューである。これも効果が大きいと考えられるが、この内容は PSP について記述する章[5]で改めて記述することにする。

何をレビューの対象にするか

スティーヴ・マコネルはその著で、ソフトウェア・プロジェクトの成果物全てをレビューの対象にするべきと述べている[MCC98]。特にプログラムに関わるものは、「要件定義書」、各種の「設計書」、「ソース・プログラム」を全てレビューの対象にすることとしている。

レビューを省略して、その結果プログラムの品質低下を招くより、時間やワークロードをかけてでもしっかりとレビューを行い、品質を高いままに保つ方が結果的に早く、安く、ソフトウェアをカットトーバーできるというのが、彼の主張である。

私は全面的に、マコネルの主張に同意する。このことについては、すぐ後で再度述べる。

なお、日本情報システム・ユーザー協会（JUAS）は毎年日本の主要な企業を対象にソフトウェアの作り方についての調査を行い、その報告書を発行している。2008 年 7 月に発行された報告書には、「開発にかけた時間の 15%以上をレビューに費やしたプロジェクトでは、高い品質のソフトウェアが開発できた」との調査結果が掲載されている[JUAS08b]。

レビューの CMMI／CMM での取り扱い

レビューは、カーネギー・メロン大学の SW-CMM（Software Capability Maturity Model：能力成熟度モデル）では、「ピアレビュー」という名前で、第 3 段階（定義段階）での KPA（Key Process Area）の 1 つに位置づけされていた。つまり CMM でレベル 3 の認定を取るためには、開発組織の中でレビューがしっかりと行われていることが不可欠の要件であった。

---

[5] PSP については、第 44 章で述べる。

CMM が拡張された「開発のための CMMI（Capability Maturity Model Integration-DEV）v3」[6]では、表面的に「ピアレビュー」という言葉は消えて、それに代えて「検証（Verification）」と「妥当性確認（Validation）」いう言葉がレベル 3 のプロセスアリアにある。そしてこの「検証」活動の中に、「ピアレビュー」が位置づけされているという形になっている。つまり表面的には違う形になっているが、実質は変わらない[CMM10a]。

このことからもレビューは、ソフトウェアを開発するという活動では、非常に基本的な、不可欠の活動であるということができる。

なお「検証」とは、本来作るべきものとして記述されている文書（例えば要件定義書や設計書）に記載されている通りにソフトウェアが作成されているかを確認するもの、「妥当性確認」とはソフトウェアが開発している組織にとって適切なものであるかどうかを確認するもの、である。つまり妥当性確認で要件定義書がその組織にとって適切なものであることを確認した後、検証でソフトウェアが要求仕様書に記述されている通りかどうかを確認する、という手順を踏むのが正しい。

検証も妥当性確認も、レビューとテストで確認することになる。

レビューの省略は絶対に行ってはならない

ソフトウェアを開発するに当たり、管理者やユーザが開発チームに強い圧力をかけて、早期の稼働開始を実現させようとすることがある。管理者やユーザは強い圧力をかけると開発チームは一層頑張って、早期の稼働が可能になると単純に信じている。そして開発チームはこの圧力に負けて、スケジュールを短縮しようとする。この時に、レビューを省略してしまうことがある。これは、絶対に行ってはならない。

レビューの省略を行うと、プロジェクトはテスト開始までの間欠陥をそのソフトウェアの中に溜め込み、テストの最初の段階でそれが一挙に表に出て、たいへんな混乱を引き起こすことになる。プロジェクトはまさにブルックスが言う「コールタールの沼に足を取られた」ような状態になり、もがき続ける[BRO75]。そして結果としてこのプロジェクトは「デスマーチ・プロジェクト」になり[YOU97]、最悪の場合最終の製品を作り出すことができず、解散させられてしまうようなことになる。

私も現役のソフトウェア技術者だった頃に、よくこの圧力をかけられた。私の場合はスケジュールの短縮と引き替えに当面の開発量の圧縮（一部の機能の開発を先延ばし）を提案し、この要求を通して開発中のレビューの省略を行ったことは一切無かった。

ソフトウェア技術者は、理不尽な開発期間の短縮の圧力に絶対に負けてはならない。

プログラム内の残存欠陥数の推定

インスペクションなどの結果を使って、そのインスペクションなどを実施したプログラム内での残存欠陥数を推定することができる。ここでは、その方法について考えてみたい[7]。

この考え方は標本再捕法と呼ばれるもので、池の中にいる魚の数を推定することで説明されることが多い。この場合具体的には、次のような実験を行う。まず、池から魚を捕らえる。この魚に印を付けて、池に放す。次に、充分時間が経った後再度魚を捕らえてその中で印が付い

---

[6] CMM と CMMI については、第 40 章で述べる。

[7] 今開発しているソフトウェア全体での残存欠陥数の推定方法については、別途第 32 章で述べる。

ている魚の割合を調べることで、池の中にいる魚の総数を推定することができる。

具体的には、次のようになる。最初に捕まえた魚の数が、20 匹だったとする。そして 2 回目に 25 匹捕まえて、そのうちの 5 匹に印が付いていたとする。そうすると、2 回目に捕まえた魚の 5 分の 1 に印が付いていたわけであるから、最初の捕まえた魚の数 20 匹の 5 倍、つまり 100 匹の魚がその池にいると推定することができる。

これを応用して、あるプログラムのインスペクションを 2 人のエンジニアが行った場合のそのプログラムの残存欠陥数を推定することができる。その手順は、以下の通りである[HUM00a]。

1. 最初のエンジニアが見つけた欠陥の数を A とする。
2. 2 人目のエンジニアが見つけた欠陥の数を B とする。
3. さらに両方のエンジニアが見つけた欠陥の数を C とする。
4. そうすると、そのプログラムの中にある欠陥の数は A×B／C として見積もることができる。
5. すでに発見された欠陥の数は A＋B―C である。
6. このことから、残存する欠陥数は （A×B／C）－(A＋B―C) となる。
7. さらにこのインスペクションでの欠陥発見率は、次の式で求められる。

   欠陥発見率 ＝ (100×(A＋B―C) ×C)／(A×B)

エンジニアの数がもっと多い場合でも、この考え方を一部修正して行うことができる。

キィワード

インスペクション、ウォーク・スルー、レビュー、残存欠陥数、検証、妥当性確認

略語

PSP：Personal Software Process
SW-CMM：Software Capability Maturity Model
KPA：Key Process Area
CMMI：Capability Maturity Model Integration

人名

マイケル・ファガン（Michael E. Fagan）、バリー・ベーム（Barry w. Boehm）

参考文献とリンク先

[BOE81] Barry W. Boehm, “Software Engineering Economics,” Prentice-Hall, 1981.

[BRO75] F.P.ブルックス Jr. 著、山内正彌訳、「ソフトウェア開発の神話」、企画センター、昭和 52 年.

この本の内容は、以下の本にそっくり含まれている。

フレデリック・P・ブルックス，Jr. 著、滝沢徹、牧野祐子、富澤昇訳、「人月の神話：狼人間を撃つ銀の弾丸はない －原著発行 20 周年記念増訂版－」、アジソン・ウェスレイ・パブリッシャーズ・ジャパン、1996 年.

この本の原書は、以下のものである。

Frederic Phillips Brooks Jr., “The Mythical man-month：essays on software engineering,” Anniversary edition, Addison Wesley, 1995.

[CMM10a] CMMI 成果物チーム、「開発のためのCMMI® 1.3 版 CMMI-DEV, V1.23 CMU/SEI-2010-TR-033 ESC-TR-2010-033 より良い成果物のためのプロセス改善」、カーネギー・メロン大学ソフトウェア工学研究所、2010年
この資料は、次の URL からダウンロードできる（確認日：2017 年（平成 29 年）1 月 25 日）。
http://cmmiinstitute.com/resource/japanese-language-translation-of-cmmi-for-development-v1-3/

[FAG76] M. E. Fagan, “Design and code inspections to reduce errors in program development,” pp182-211, IBM System Journal, Vol.15 No.3, 1976, IBM.
このペーパは次の URL で、IBM のホームページからダウンロードすることができる。（確認日：2017 年（平成 29 年）1 月 13 日）
http://www.research.ibm.com/journal/sj/382/fagan.pdf

[FRE82] ダニエル・P. フリードマン、ジェラルド・M. ワインバーグ著、岡田正志監訳、「ソフトウェア技術レビューハンドブック：実践的ノウハウに関する Q&A」、TBS 出版会、1987 年.
この本の原書は、以下のものである。
Daniel P. Freedman, Gerald M. Weinberg “Handbook of Walkthroughs, Inspections, and Technical Reviews Evaluating Programs, Projects, and Products,” Dorset House, 1982.

[GIL93] Tom Gilb、Dorothy Graham 著、伊土誠一、富野壽監訳、「ソフトウェアインスペクション」、構造計画研究所、1999 年.
この本の原書は、以下のものである。
Tom Gilb、Dorothy Graham, “Software Inspection,” Pearson Education, 1993.

[GLA03] ロバート・L. グラス著、山浦恒央訳、「ソフトウェア開発 55 の真実と 10 のウソ」、日経 BP 社、2004 年.
この本の原書は、以下のものである。
Robert L. Glass, “Facts and Fallacies of Software Engineering,” Pearson Education, 2003.

[HUM00a] ワッツ・S. ハンフリー著、秋山義博監訳、JASPIC-TSP 研究会訳、「TSPi ガイドブック ソフトウェア開発の課題 10」、翔泳社、2008 年 6 月 18 日.
この本の原書は、以下のものである。
Watts S. Humphrey, “Introduction to Team Software Process, “ Addison-Wesley, 2000.

[IEEE08a] IEEE Computer Society Standards Coordinating Committee, “IEEE Standard for Software Reviews and Audit IEEE Std 1028-2008,” IEEE, 2008.

[JUAS08b] 日本情報システム・ユーザー協会、「ユーザー企業ソフトウェアメトリクス調査 2008 ソフトウェアの開発・保守・運用の評価指標」、日本情報システム・ユーザー協会、平成20年7月.

[LIK04] ジェフリー・K・ライカー著、稲垣公夫訳、「ザ・トヨタウェイ（上、下）」、日経 BP社、2004年.
この本の原書は、以下のものである。
Jeffrey K. Liker, ‘The Toyota Way,” MACGRAW-HILL, 2004.

[MCC98] Steve McConnell 著、（株）アルテア・ジャパン訳、久手堅憲之監修、「新訳ソフトウェアプロジェクトサバイバルガイド」、日経 BP ソフトプレス、2005 年.

この本の原書は、以下のものである。
Steve McConnell, “Software Project Survival Guide,” Microsoft, 1998.
[WEI71] ジェラルド・M・ワインバーグ著、木村泉他訳、「プログラミングの心理学 または、ハイテクノロジーの人間学」、技術評論社、平成6年.
この本の原書は、次のものである。
Gerald M. Weinberg, “The Psychology of Computer Programming,” Van Nostrand Reinhold Co., 1971.
[WIE02] Karl E. Wiegers 著、大久保雅一監訳、「ピアレビュー 高品質ソフトウェア開発のために」、日経 BP ソフトプレス、2004 年.
この本の原書は、次のものである。
Karl E. Wiegers, “Peer Review in Software: A Practical Guide,” Addison Wesley Professional, 2002.
[YOU89a] E.ヨードン著、国友義久、千田正彦訳、「ソフトウェアの構造化ウォークスルー [第2版]」、近代科学社、1991 年.
この本の原書は、以下のものである。
Edward Yourdon, “Structured Walk-Throughs,” Yourdon Press, 1989.
[YOU97] E.ヨードン著、松原友夫訳、「デスマーチ：なぜソフトウェア・プロジェクトは混乱するのか」、トッパン、1998 年.
この本の原書は、以下のものである。
Edward Yourdon, “Death March The Complete Software Developer’s Guide to Surviving “Mission Impossible” Projects,” Prentice Hall, 1997.
またこの本は第 2 版が発行されている。第 2 版は、以下のものである。
エドワード・ヨードン著、松原友夫／山浦恒央訳、「デスマーチ 第 2 版：ソフトウェア開発プロジェクトはなぜ混乱するのか」、日経 BP 社、2006 年.
さらに、この本の原書は以下のものである。
Edward Yourdon, “Death March 2nd Edition,” Pearson Education, 2004.

（2005 年（平成 17 年）7 月 1 日　初稿作成）
（2006 年（平成 18 年）8 月 17 日　一部修正）
（2008 年（平成 20 年）9 月 7 日　一部修正）
（2010 年（平成 22 年）7 月 19 日　一部修正）
（2014 年（平成 26 年）4 月 29 日　一部修正）
（2016 年（平成 28 年）4 月 15 日　一部修正）
（2017 年（平成 29 年）1 月 13 日　一部修正）