大規模で短期間のソフトウェア開発における品質進捗へのアプローチ

--- Quality Mind and Skill Scoring System ---

An Approach to Quality Progress in Large-Scale, Short-Term Software Development

--- Quality Mind and Skill Scoring System ---

（株）日立製作所　システム＆サービスビジネス統括本部

Hitachi, Ltd. Systems & Services Business Division

○羽田達哉	床直紀 [1)]	中山仁 [2)]	石井尚 [3)]	古田莉央 [4)]
○Tatsuya Hada	Naoki Toko	Hitoshi Nakayama	Takashi Ishii	Rio Furuta

Abstract

In software development, the progress of the development work can be evaluated, but the quality of the product (how many bugs/defects the code contains) is difficult to know until testing is completed. Thus, there are often more software bugs than planned, and it takes a lot of time to notice that the amount of time and effort needed to fix the bugs is greater than planned, and that replanning is necessary in regards to the future. In large-scale, short-term software development in particular, the risk is high that there will be many bugs, that delivery will be delayed, and that the cost will overrun. For this reason, a cycle that can detect and improve areas poor in quality (improve the process) at an early stage is important.

An effective way to conventionalize good processes is to simply and honestly implement reviews and self-checks (a standard check perspective) within each process. Work quality can be evaluated by quantifying the status of implementation. By using this gauge of work quality and therefore an "abnormal value = poor work quality" module, it is possible to objectively detect the omissions (dishonest statements) and lack of understanding of development personnel and to provide skill-improving guidance.

We call this technique the "Quality Mind and Skill Scoring System", or QM3S. This technique for quantitatively evaluating the quality of development work has been confirmed to be effective in ensuring software quality. In the future, it will be utilized as a technique for ensuring product quality from the development work stage.

（株）日立製作所　システム＆サービスビジネス統括本部　品質保証本部　公共品質保証部

Hitachi, Ltd. Systems & Services Business Division

Quality Assurance Division　　Public Quality Assurance Department

東京都品川区南大井 6-23-1(日立大森ビル)　Tel:03-5471-4567　Email:tatsuya.hada.rf@hitachi.com

6-23-1,Minami-Oi,Shinagawa-ku, Tokyo, Japan

同　課長

1)　Senior Engineer of Quality Assurance Department

同　部長

2)　Director of Quality Assurance Department

同　技師

3)　QA Leader of Quality Assurance Department

同　担当

4)　QA of Quality Assurance Department

1. 序論

ソフトウェア開発では、プロダクト品質は開発プロセス、開発要員と相関関係があることが一般的に知られている。例えば、CMMI では「よいプロダクトは、よいプロセスから生まれる」と記載されている。そこで私たちは「開発プロセス（作業上の規則）に従って作業すると開発作業の精度や質が良くなる。」「開発作業の精度や質が良くなるとソフトウェア品質が良くなる。」すなわち「開発プロセスに従って作業するとソフトゥエア品質が良くなる。」と考える。この考え方に基づき、開発プロセスが遵守される（されやすい）手法の検討、試行を実施した。本論はその検討過程および試行結果を述べるものである。

2. プロダクト品質を予測する

私たちはソフトウェアのテストでプロダクト品質を評価するために、様々な測定指標を使っている。代表的な測定指標は、バグ密度（コードの規模当たりのバグ数）が挙げられる。バグ密度は長年の開発経験から標準値（期待される値）が算出されており、マネージャーはバグ密度の標準値を元にプロジェクト計画（必要となる期間、開発要員数、コスト）を作成し、ソフトウェア開発を管理している。

テスト開始後にソフトウェアのプロダクト品質が悪い（バグが標準値より多く存在する）と判明した場合は、バグ修正に掛かる時間と労力が計画より増えるため、開発スケジュールは遅延し、開発コストはオーバーすることになる。特に大規模で短期間のソフトウェア開発は、大量の要員（色々な経験とスキル？）が一斉に開発する（プロセス通り？）ため、標準値よりバグが多くなるリスク、納期に遅延するリスク、コストが増えるリスクが大きい。

これらの状況を突破するために、私たちはテスト開始前にプロダクト品質を予測することが重要であると考えた。

3.作業プロセスがプロダクト品質に与える影響

3.1 良いプロセスの検討実験

あるプロジェクトにおいて複数チームが同じソフトウェア（開発中でバグが存在している）を対象にソフトウェアテストを並行して実施したところ、テスト結果（バグ件数）や進捗（遅延日数）が同一にならず差が生じた。チーム間での要員のスキル差は無い、または無視できるレベルであったことから、原因はテスト作業における作業プロセスの差異であると分かった。この実験結果を元にどのプロセスがプロダクト品質の差異の主要因であるか（「良いプロセス」であるか）を分析した。

3.2 分析対象（測定データ）

分析に用いる対象データを表 3.2 に示す。

各チームのテストで見つかったバグ件数、見つかるはずが見つけられなかったバグ件数（見逃しバグ件数（※））、テストで使用したチェックリスト件数などを品質指標として分類した。※見逃しバグ件数は後工程にて計測

テスト完了までに当初予定より遅延した日数（遅延日数）、当初予定より遅延した作業量（遅延工数）を進捗として分類した。

各チームにおける詳細プロセスの差異（チェックリストレビューの回数／時間、朝会/夕会の回数／時間、懸案管理など）を作業プロセスとして分類した。

表 3.2 分析対象データ

#	大項目	データ	取得元
1	品質指標	見逃しバグ件数	バグ票
2		見逃しバグ重要度（A+B）の件数	バグ票
3		バグ密度	バグ票
4		バグ重要度（A）の件数	バグ票
5		チェックリスト密度	チェックリスト
6		チェックリスト区分（異常＋限界）の件数	チェックリスト
7	進捗指標	遅延日数	進捗報告書
8		遅延工数	進捗報告書
9	作業プロセス	チェックリストレビュー回数	指摘票
10		チェックリストレビュー時間	レビュー票
11		チェックリスト区分（異常、限界）に関する指摘の割合	指摘票
12		小日程の有無	ヒアリング
13		朝会/夕会の実施回数	ヒアリング
14		朝会/夕会の実施時間	ヒアリング
15		懸案件数	懸案管理台帳
16		懸案対応の平均遅延日数	懸案管理台帳
17		バグ内容の情報共有の場の有無	ヒアリング
18		類似見直し対象が全範囲の割合	バグ票

これらのデータを用いて、「品質指標と相関の強い作業プロセス」「進捗指標と相関の強い作業プロセス」を相関分析にて分析した。

3.3 プロセス分析結果

各指標と作業プロセス内容の間に強い相関が見られた作業プロセスの上位3つを、プロダクト品質の向上のために効果的な作業プロセスであると判断した。以下に、その作業プロセスを示す。また、当該作業プロセスについて、品質指標値が最も良いチームが実施した具体的な作業プロセスを表 3.3 に示す。

表 3.3 効果的な作業プロセス

#	分類	指標	作業プロセス
1	品質	バグ密度・バグ重要度(A)の件数	セルフチェックリストを用いてチェックリストレビューを実施し、1件1件観点漏れがないか確認する。
2			バグ内容の情報共有の場を設ける。 (例:朝会/夕会の実施)
3		チェックリスト区分(異常+限界)の割合	セルフチェックリストを用いてチェックリストレビューを実施し、1件1件観点漏れがないか確認する。
4			担当者ごとに1本目が完了した時点でチェックリストレビューを実施し、2本目以降に指摘内容を反映させる。
5			チェックリストレビュー時にチェックリスト区分の基準値(正常:60% 異常:20% 限界:20%)を満たしているか、確認する。
6	進捗	遅延日数・遅延工数	早期に発見したバグをチーム内で情報共有できる状態で管理する。
7			懸案期限日の根拠を明確にした上で、懸案期限日を設定する。(懸案対策の優先順位を明確にし、作業に取り組む。)

3.4 良いプロセスとは

実験の分析結果より私たちはよいプロダクトを生み出す「よいプロセス」とは以下を満たすと分かった。

・セルフチェックリスト（ノウハウから抽出した標準的なチェック観点）を適切に使用する

・適切なレビューを実施する

・開発要員同士でお互いが持つ品質に関する情報を共有する

上記は長年のソフトウェア開発経験と一致しており、私たちは妥当であると考える。

4 新施策　QM3S

大規模開発で短期間のソフトウェア開発では大量の開発要員を一気に集めるため、要員ごとの品質マインドやスキルにバラツキがある。品質マインドやスキルが低い開発要員はプロセスを適切に実施することが出来ず、開発したソフトウェアの品質は悪い。プロセスの実施状況を数値化し、定量評価することで、テスト前に品質を確認・改善することを私たちは検討した。私たちは本施策を QM3S”Quality Mind and Skill Scoring System”と呼称している。

4.1 QM3S　手順概要

ソフトウェア開発（設計、コーディング、テストといった各工程）における新たなやり方（QM3S）のプロセスフローを図 4.1 に示す。

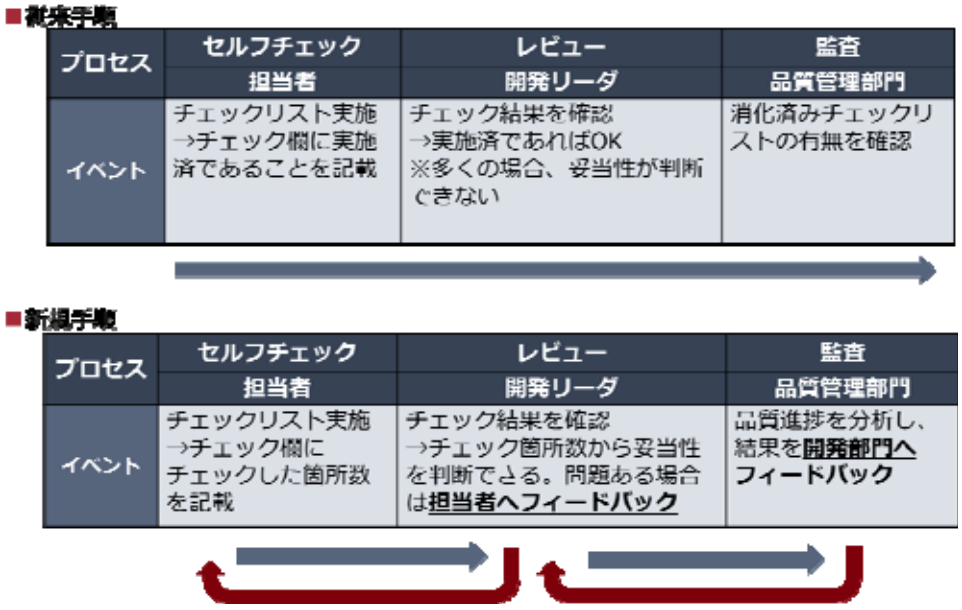

図 4.1-1　プロセスフロー

担当者は作業後にセルフチェックリストを利用して確認を実施する。今までの手順は確認したチェックリストにチェック結果（レ点）を記載するであった。確認箇所数（数値）を記載するという新たな手順を検討した。その利点は以下の通り。

・確認箇所はコピー＆ペーストでは記載できないので、セルフチェックリストをどれだけ使用したかが分かる

・担当者が確認するついでに箇所数をカウント出来るため、工数負荷がほとんど発生しない。

開発リーダーはレビューにおいて、担当者のセルフチェック結果が妥当であるかを確認する。今までの手順は確認したチェックリストに審査者(Name)を記載するであった。開発リーダーがチェックした確認箇所数との差分箇所数(数値)を記載するという新たな手順を検討した。その利点は以下の通り。

・担当者より報告された箇所数を確認することで、担当者の理解不足や嘘を簡単に検知できる。
・担当者の誤り箇所が具体的に理解できるため、担当者への指導が効率的にできる。
・開発リーダーより報告された追加箇所数を確認することで、どれだけ担当者を指導したか分かる。

品質保証部門(第三者)は報告されたセルフチェックの確認箇所数(担当者／開発リーダー)からセルフチェック、レビューの確認量を数値化(数値化された確認量を"品質進捗"と定義)する。そして品質保証部門は品質進捗の監視と制御を行う。セルフチェックリストの変更前後を図 4.1-2 に示す。

図 4.1-2　セルフチェックリスト

4.2 品質進捗

前項 4.1 で定義した"品質進捗"について具体的な算出方法を述べる。

品質進捗 ＝ 確認箇所数 × 観点ごとの係数

確認箇所数：担当者および開発リーダの確認箇所数の和

観点ごとの係数：セルフチェック観点の重要度（※）に応じて設定した係数

※後工程で摘出しづらい、不良を作り込んだ際の修正工数が大きくなる観点ほど重要度を上げる

4.3 QM3S 適用結果

複数機能からなるソフトウェアを機能単位にチームを分けて開発した。そのソフトウェア開発のコーディング工程において、以下の3つの視点で本施策の適用結果を纏めた。

・品質進捗の測定結果
・バグ密度
・品質進捗と残バグ密度

本施策を適用した3チームにおいて、工程内で定期的に品質指標を測定した。各チーム単位での品質指標の測定結果を図 4.3-1 に示す。

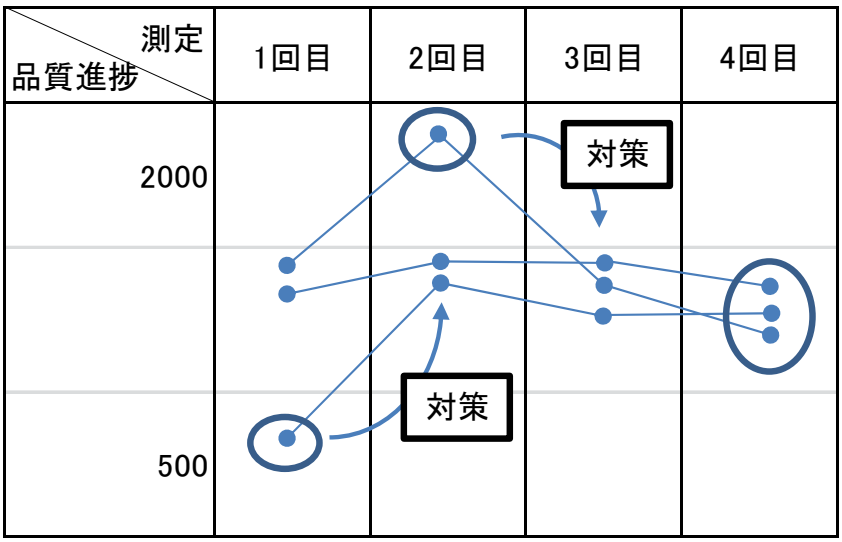

図 4.3-1　品質指標の測定結果

1回目の測定において、チーム1の品質進捗は他の2チームと比較して低い値であった。この原因を調査したところ、チーム1の開発要員はセルフチェックリストに記載されたチェック内容を正しく理解できていなかった（理解できなかった項目は何も確認しなかった）と判明した。そこで、品質保証部門（第三者）より開発リーダー、開発要員に対してチェック内容を詳細に追加説明した。その結果、2回目の測定において、チーム1の品質進捗は適正な値に改善された。

2回目の測定において、チーム3の品質進捗は他の2チームと比較して高い値であった。この原因を調査したところ、チーム3の開発要員がセルフチェック結果を不正報告していた（過大な確認箇所数を報告した）と判明した。そこで、品質保証部門（第三者）にて不正報告した開発要員を特定し開発リーダーを交えて厳重注意を行った。その結果、3回目の測定において、チーム3の品質進捗は適正な値に改善された。

1～4回目の測定結果より、プロセスの実施状況が十分に適切であれば、各チームの品質進捗はほぼ同じ値に集約することが分かった。

本施策を適用したチームと未適用のチームで、セルフチェック、レビュー、次工程のテスト（単体テスト）でのバグ密度（ソースボリューム当たりの摘出されたバグ件数）がどう違うかを測定した。バグ密度の測定結果を表 4.3-2 に示す。

表 4.3-2　バグ密度の測定結果

チーム	ｾﾙﾌﾁｪｯｸ	レビュー	テスト
施策適用	2.8件/ks	0.5件/ks	4.3件/ks
施策未適用	0.9件/ks	0.4件/ks	7.4件/ks
ＰＪ計画値	未定義	未定義	4.3/ks

セルフチェックでのバグ密度は適用したチームの 2.8 件/Ks に対し、未適用のチームは 0.9 件/Ks であった。単体テストでのバグ密度は適用したチームの 4.3 件/Ks に対し、未適用のチームは 7.4 件/Ks であった。このことから、本施策を適用することで開発作業中に見つけるバグ件数が多くなり、テストで発見されるバグ件数（残バグ件数）をPJ計画値に近づけられることが出来ると分かった。

また施適用したチームの担当者からはセルフチェックに対する Q&A が 30 件発生した。未適用のチームの担当者からはセルフチェックに対する Q&A は発生しなかった。Q&A の内容はチェック観点やカウント方法などであったことから、本施策によりセルフチェックやレビューの作業品質が上がったことが原因といえる。

次に私たちは品質進捗がソフトウェアのプロダクト品質と相関関係があると考えた。プロダクト品質は存在するバグ密度で表される。今回の場合は残存バグ（その後のテストでの発見されたバグ密度）であると私たちは考える。品質進捗と残存バグ密度の相関を図 4.3-3 に示す。

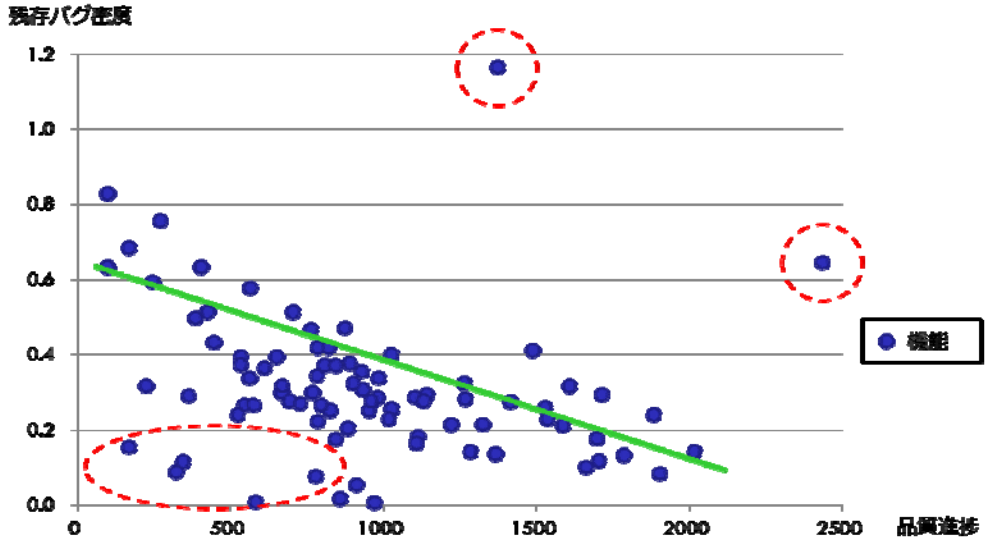

図 4.3-3　品質指標と残存バグ密度

殆どの機能で以下の傾向があることが分かった。（補助線で示した部分）

・殆どの機能で品質進捗が大きい場合はバグ密度が低い（ソフトウェア品質が良い）

・殆どの機能で品質進捗が小さい場合はバグ密度が高い（ソフトウェア品質が悪い）

一部の機能で以下が分かった。（○で囲った部分）

・セルフチェックリストのチェック項目では摘出できないバグが多く潜在していた場合は、品質指標と品質の相関関係が低い。セルフチェックリストのチェック項目の精度に課題がある。

・ソースのボリュームが小さい機能は品質進捗と品質の相関関係が低い。

4.4 プロダクト品質は作業プロセスから予測できる

開発要員が自分の作業結果に対し適切に確認したかどうかは、セルフチェックリストの確認量で判断できる。そのため私たちはセルフチェックリストの確認箇所数（担当者／開発リーダー）から算出した品質進捗を利用することで、開発したソフトウェアの品質をある程度予測できると分かった。

5.結論

私たちは品質進捗を活用することで開発要員の確認不足や理解不足といった問題を検知することができた。テスト前（例えば、設計・コーディング）に品質進捗を測定することで、テスト前からソフトウェア品質を予測し改善できると考える。

テスト前にソフトウェア品質を予測し改善することで、私たちは納期に遅延するリスク、コストが増えるリスクを減らすことができる。特に大規模で短期間のソフトウェア開発において、この手法は効果が大きいと私たちは考える。

6.参考文献

なし

以上