第 6 章　高品質のソフトウェアを作る方法

第 5 章で我々は、品質とは「ユーザなどの要求に応じる程度」と定義し、その上に立ってソフトウェアの品質とは何かということを見てきた。

第 6 章では、その高品質のソフトウェアを作る方法について考えてみたい。その前に、高品質のソフトウェアがもたらす便益を見ておきたい。

高品質のソフトウェアの便益

既に第 3 章で述べたように、品質の低いソフトウェアは、それ自身ソフトウェア危機の 1 つの症状である。しかしそれと同時に、他の全てのソフトウェア危機の要因でもある。つまり低品質のままでは顧客に出荷できないことから、開発者はより品質の高いものに仕上げようとテストに拍車をかける。このことが、開発コストの増大とスケジュールの遅延を招く。さらにコストの増大とスケジュールの遅延が激しくなって、ユーザがこのままではこのプロジェクトは製品を仕上げることができないと判断すると、プロジェクトは最終製品を作ることなく解散させられることがある。

高品質のソフトウェアはこの逆である。製品の品質が良いことから、テストを適切に切り上げることができる。その結果一般にコストはリーズナブルな範囲に留まり、製品も順調に出荷される。つまりソフトウェア危機の症状の多くは、高品質のソフトウェアを作ることで解消されることが期待できる[1]。

要件定義について

ソフトウェアの開発の手順を図表 6-1 で示すように、先ず 2 つの部分に分けて考えてみたい。ユーザの要求を明確にする段階と、そこで明確になったユーザの要求を実現するべく実際に動くソフトウェアを開発する段階の 2 つである。

第 5 章で述べたように、品質の高いソフトウェアとはこのユーザの要求を高いレベルで実現したソフトウェアであるから、先ずこの最初の段階、つまりユーザの要求を明確にする段階で、そのレベルが高いことが必要である。この作業は、「ソフトウェア要求仕様書の作成」と「ソフトウェアの要件定義」と名付けられている作業の中で行われる[2]。

ここでたいへん残念なことに、ユーザは自分のソフトウェアへの要求を常に明確に意識しているとは限らない。仮に意識していても、それを的確に表現できるとは限らない。時には、自分が考えていることとは逆の表現をすることすらある。しかもこの要求は、時間とともに変化することがある。ユーザが変われば、当然その要求が変わる[3]。

第 5 章の品質の定義で、「要求」とは明確に述べられたり書かれたりしたものだけでなく、述

---

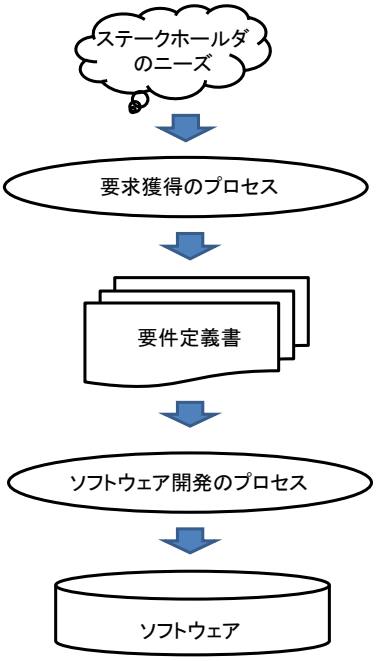
[1] 高品質のソフトウェアを作るだけでは残念ながら、ソフトウェア危機の全てが解消するわけではない。ソフトウェア危機の解消法については、第 3 章で論じた。そこでは、ソフトウェアプロジェクトの見積もりが適切になされることも重要な要件であると述べた。

[2] ソフトウェアの要求仕様書の作成は第 20 章で、要件定義書の作成は第 21 章で、それぞれ述べる。

[3] 「要求」とはユーザなどのステークホールダ（利害関係者）が、明確にでも漠然とでも、そのソフトウェアなどに「期待しているもの」を表す。一方の「要件」は、「要求」の中で実現可能であることが確認できたもの、検証できるものなど、より明確に、具体的になったものを指す。

べられなかったもの、あるいは書かれなかったものも含み、ユーザが自覚していないものまでも含むと述べた。この言葉に、ソフトウェアの要件定義の難しさがある。ソフトウェア工学の範疇でこの部分は「要求工学（Requirement Engineering）」と、「工学（Engineering）」の言葉を付けた名前で呼ばれている。ここにその対応の難しさが表れていると、私は考える。

ユーザの要求は、機能レベルのものだけであってはいけない、「品質」への要求など[4]も含まなければならないと、ドイチェらは述べている[DEU88]。第 5 章で述べたようにソフトウェアの品質は、いま有効な国際規格である ISO／IEC 25010：2011（日本の規格は JIS X 25010：2013）では、「利用時の品質」で 5 つの特性、「外部特徴」と「内部特徴」では 8 つの特性と 30 の副特性があげられている。全てのソフトウェアでこれらの特性／副特性全部が必要になるとは思わない。しかし必要な特性と副特性について、ユーザが明確に品質の要求についても意識することが必要である。別の言い方をすれば、ソフトウェアの要件定義の作業でユーザと一緒に仕事をして要求を明確にするソフトウェア技術者は、機能面だけでなく品質などの非機能面についても、ユーザの意向をよく把握しておかなければならない。



図表 6-1　ソフトウェア開発のプロセス（[SAN94]より）

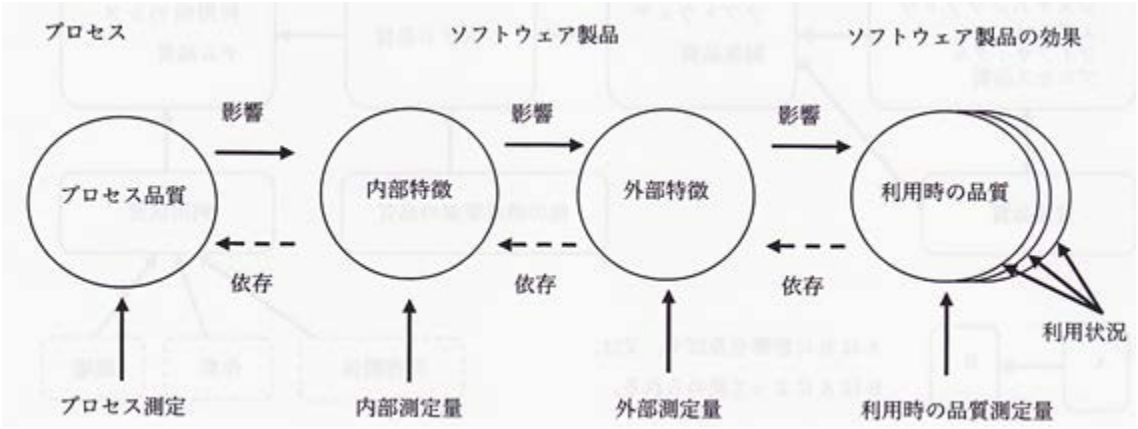
ソフトウェアへの要求が明らかになった後は、この要求が書かれた文書（「要件定義書」などと呼ばれる）に基づいて、ソフトウェア開発の作業が行われることになる。これらの作業のレベルも、当然高いことが要求されることになる。これについては次の項で、別の図を使って議論したい。

ソフトウェア・プロセスの改善

図表 6-2 は、図表 5-4 として示したものと同じものである。この図については第 5 章でも述

[4] 品質への要求など機能以外の要求を「非機能要求」という。機能要求と非機能要求については、第 21 章の「要件定義書の作成」の章で述べる。

べたが、ソフトウェアを開発するプロセス（ソフトウェアの作り方）の品質が良ければ、その結果構築されるソフトウェアの品質が良くなることを示している。この考え方は、何もソフトウェア開発に限る話ではない。自動車やテレビ、コンピュータ、工作機械など、全ての二次産業の製品に適用されている考え方である。今のソフトウェアの品質保証の考え方は、ここから来ている[5]。



図表 6-2　ソフトウェアに関わる品質（[JIS13a]より）

つまりソフトウェアの品質を良くするためには、ソフトウェアのプロセス、つまりソフトウェアの作り方が良くなければならない。一回限りの成功で良ければ、ソフトウェア・プロセスの品質が良くなくてもよい場合がある。ある優れたソフトウェア技術者の大奮闘があれば、たまたま高品質のソフトウェアを作ることができるかもしれない。しかし高品質のソフトウェアを継続して作り続けるためには､組織としてのソフトウェアのプロセスが良くなければならず､ソフトウェアのプロセスを良い状態にするためには、それを常時改善してゆく必要があることになる。これがソフトウェア・プロセス改善の「こころ」である[HUM89][6]。

さらにその組織を構成しているソフトウェア技術者のレベルが高いことも、必要不可欠である。そのために組織はその構成員に対して適切な教育を行う必要があり、構成員はその知識・技術を修得するだけでなく、自らソフトウェア技術者としての倫理を身につけ、適切に仕事ができるよう自らを習慣づけなければならない[7]。

レビューとテスト

ソフトウェア・プロセスの改善という問題は、ソフトウェア開発の全部の作業を対象としている。ソフトウェアの作り方を良くするためには、その開発に関わる全部の作業を改善することが必要ということは分かるにしても、これではあまりにも範囲が広く、遠大でありすぎる。

---

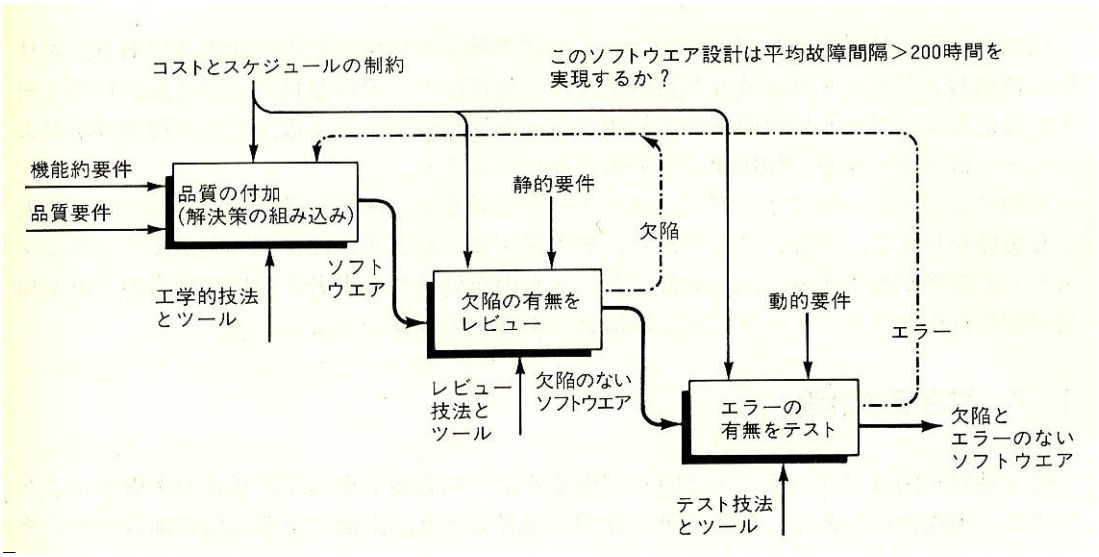
[5] ソフトウェアの品質保証については、第 7 章で述べる。

[6] ソフトウェア・プロセスの改善については、第 17 部の 3 つの章（第 39 章から第 41 章）で述べる。

[7] ソフトウェア技術者の倫理については第 48 章で、PSP（パーソナル・ソフトウェア・プロセス）と TSP（チーム・ソフトウェア・プロセス）については第 44 章で、それぞれ述べる。

もっと直接的、端的に改善作業を始めることができないのかという疑問があるかもしれない。この項では、その疑問に応えたい。

ソフトウェアの品質を構成する要素は第 5 章で述べたように、多岐にわたる。しかしその中でポイントになる要素は信頼性に関わる部分、つまりソフトウェアに欠陥が少ないことだとケーパース・ジョーンズ（Capers Jones）は述べている[JON96b]。いくつもの欠陥を抱えているソフトウェアが、これを除く他の要素で高いレベルにあるとは考えにくいし、逆に欠陥がほとんどないソフトウェアが他の要素でも高いレベルにある、ということが普通である。したがってソフトウェアの品質を構成する要素の数は多いものの、つまるところ我々はこれについて我々の常識のレベルに戻って、とりあえず欠陥を少なくすることにポイントを置けばよい、ということになる。



図表 6-3　ソフトウェア品質ライフサイクル（[DUE88]より）

最終的にソフトウェアに欠陥を持ち込まない方法は、基本的には 2 つある。レビューとテストである。この関係を図示したものを、図表 6-3 に示す。

普通に考える場合、製品から欠陥を取り除くのはテストによって行うということだろう。しかし理想をいえば、製品を作る過程で全ての欠陥を取り除き、高い品質を製品に埋め込むというスタンスで製品を作り上げて、テストでは「欠陥がないことを確認する」というスタンスを取るべきである。良くない表現かもしれないが、「適当に製品を作り上げ、それをテストして見つかった欠陥を取り除く」というスタンスでは、高い品質の製品を作り上げることはできない。ソフトウェア以外の製品でもそうだが、特にソフトウェアについてこのことは、次のようにいえる。

1. テストは、ソースプログラム（プログラマがプログラム言語で記述したプログラム）が完成した後でなければできない。したがってテストを行うだけでは、システム要件定義のフェーズ（要求仕様を固める段階）や設計以降のフェーズで埋め込まれた欠陥もテストの段階まで発見する機会を持つことはできない。間違った要求仕様を基に正しい設計

と正しいプログラミングを行っても、間違ったプログラムが生まれる。このような欠陥について、テストでそれを発見した時に、手戻りの量が非常に多くなることになる。手戻りが多いと、それだけで作業の生産性が低下する。手戻りの量を少なくするためには、欠陥が埋め込まれた後、できるだけ早い時期にそれを見つけて、取り除くことである。

2. テストでは、テストデータを用意した範囲内しか欠陥を発見できない。全てのケースにテストデータを用意することは、不可能である。

したがって、欠陥の少ない（理想的には欠陥のない）ソフトウェアを構築しようとすれば、要件定義や設計、プログラミングなどの作業が進行中に、その成果物、つまり要件定義のフェーズでは「要件定義書」、設計のフェーズでは各種の「設計書」、プログラミングではプログラムそのものなど、作業の結果作られる文書類をレビューして、その中に欠陥が埋め込まれていないことをチェックすることが必要である。当然発見した欠陥は、除去しなければならない。

埋め込まれている欠陥を発見し、除去する割合を欠陥除去率という。レビューの場合はそのレビューのやり方にもよるが、良いレビューでは欠陥除去率は 60 パーセントから 80 パーセントにも達するといわれている。一方テストでの欠陥除去率はだいたい 30 パーセント見当である。欠陥除去率の観点から見ても、レビューの方がテストより効率的である。つまり高品質のソフトウェアを開発する上で、レビューの作業は欠かせないということになる。

しかしいくら欠陥除去率が高いからといってもやはり、レビューだけを行い、テストなしで高品質のソフトウェアを実現することは難しい。レビューとテストには、それぞれの特徴がある。高品質のソフトウェアを開発するためには、この特徴を良く見極めて、レビューとテストの両方をうまく使い分けることが必要になる[8]。

キィワード

要件定義、要求工学、Requirement Engineering、ソフトウェアの品質保証、ソフトウェア・プロセス、ソフトウェア・プロセス改善、ソフトウェアの欠陥、レビュー、テスト、欠陥除去率

規格

ISO／IEC 25010：2011、 JIS X 25010：2013

人名

ケーパース・ジョーンズ（Capers Jones）

参考文献とリンク先

[DEU88] Michael S. Deutsch、Ronald R. Willis 著、成田光彰訳、「ソフトウェア品質工学ー技術・管理両面からの総合的アプローチ」、日経 BP 社、1990 年.

この本の原書は、以下のものである。

Michael S. Deutsch, Ronald R. Willis, “Software Quality Engineering,” Prentice-Hall, 1985.

[HUM89] Watts S. Humphrey 著、藤野喜一監訳、日本電気ソフトウェアプロセス研究会訳、

[8] レビューについては第 18 章で、テストについては第 12 部の 3 つの章（第 30 章から第 32 章）で、それぞれ述べる。

「ソフトウェアプロセス成熟度の改善」、日科技連、1991 年.
この本の原書は、以下のものである。
Watts S. Humphrey, “Managing the Software Process,” Addison-Wesley, 1989.

[JIS13a] 日本工業標準調査会審議、「JIS システム及びソフトウェア製品の品質要求及び評価ーシステム及びソフトウェア品質モデル JIS X 25010 : 2013 （ISO／IEC 25010 : 2011）」、日本規格協会、平成 25 年.

[JON96b] Capers Jones 著、富野壽監訳、「ソフトウェア品質のガイドライン」、（株）構造計画研究所、1999 年.
この本の原書は、以下のものである。
Capers Jones, ‘Software Quality Analysis and Guideline for Success,” International Thomson Publishing, 1996.

[SAN94] J. サンダース、E. カラン著、原田曄他訳、「ソフトウェア品質向上のすすめー新しいソフトウェア開発の標準」、（株）トッパン、1996 年.
この本の原書は、以下のものである。
Joc Sanders, Eugene Curran, “Software Quality A Framework for Success in Software Development and Support,” Addison-Wesley Publishing, 1994.

（2004 年（平成 16 年）5 月 30 日　初稿作成）
（2014 年（平成 26 年）3 月 1 日　一部修正）