解説

ソフトウェア工学

山　本　哲　男
Tetsuo YAMAMOTO

理工学部情報メディア学科　准教授
Associate Professor, Department of Media Informatics

1. はじめに

2018 年 4 月に着任した情報メディア学科の山本哲男です．専門はソフトウェア工学と呼ばれる分野であり，ソフトウェア開発における種々の問題点を解決するために研究をしています．本稿では，そのソフトウェア工学の解説と問題点を列挙し，読者の今後のソフトウェア開発に生かせればと思います．

2. ソフトウェア工学

他の理工系分野に比べて情報系は歴史が浅く，多くの問題が課題となっています．とりわけ，ソフトウェアに関する分野では，職人と呼ばれる一部の人たちによる開発に頼り切る傾向が見られます．IT 全盛時代になりつつある現代においてもこの傾向はなかなか改善されていません．ソフトウェア工学に関しても多くの書物が出版されています[1][2]．まずは，歴史を振り返りながら，ソフトウェアの位置づけを見ていきたいと思います．

2.1　歴史

・1960 年前半まで

コンピューター創世記においてはソフトウェアはハードウェアに付加されたサービス的な機能と位置づけられており，多くの人たちにとってソフトウェアには無関心の領域でした．

・1968 年

1960 年代後半になるとコンピューターが扱う問題の複雑性化にともない，ソフトウェアの重要性が求められます．ソフトウェアを正しく動作させるためにはどのようにしていかなければならないか等を考える必要が出てきます．この時代に「ソフトウェア危機」と呼ばれる言葉が誕生しています．また，NATO の第 1 回ソフトウェア工学会議もこの時期に開催され，ソフトウェアを工学的見地から考えていく時代に入ります．

つまり，ソフトウェア工学という分野が誕生して，まだ 50 年しかたっていません．50 年の間に以下の様々な問題が起こっています．

・ソフトウェアの品質悪化，保守の対応
・ソフトウェア技術者の不足
・障害が及ぼす社会的波及問題

これらの問題を工学として解決していこうというのがソフトウェア工学です．そもそも，工学とは理論的に体系化された定理や科学的な定義や法則などを，実践的な工業技術へ結びつけることであり，既

存の職人芸，その場しのぎの解決法を打開するための学問です．今後ますます重要となるソフトウェアを，いかに工学にするかがソフトウェア工学の研究者の使命でもあります．

2.2 ソフトウェアの一般的特性

本節では，ソフトウェアを見つめ直すことでソフトウェアならではの特性を述べていきます．

1. ソフトウェアは目に見えない製品

ハードウェアといった物理的に存在する物は人間の目に見え，その形や大きさが分かりますが，ソフトウェアは目に見ません．もちろん，ソフトウェアに関するドキュメントやソースコードは目に見えますが，表現手法の一つであり，本質的にはソフトウェアは見えないものです．

そのため，ソフトウェア全貌を俯瞰的に理解することは難しい問題がでてきます．例えば，建物の建築であれば，いまどの程度完成しているか目に見えて分かる場合がありますが，ソフトウェアの場合は見えませんので，いまどこまで進んでいるかを把握することは困難です．

2. 品質の劣化なし，向上していく傾向

ソフトウェアはハードウェアと違い経年劣化という概念が存在しません．作成後の品質は一定で何年経過しても下がりはしません．もちろん，不具合は存在しますが，その不具合を修正することで品質は向上していく傾向にあります．劣化しないということは，一度導入すると，永遠に利用できる可能性があるということです．もちろん，社会情勢の変化等によりソフトウェアが使い物にならない場合はありますが，そうでない場合はソフトウェアのライフサイクルはハードウェアに比べ長くなります．

3. 潜在的バグが潜んでいる

ソフトウェア開発は機械を用いて作成する工程はすくなく，人手の作業による作成がほとんどです．そのため，開発者の能力に依存する傾向が高く，潜在的な不具合を含む可能性が高くなります．不具合がないソフトウェアかどうかを定量的に評価することは困難であり，100% 不具合がないと言うことは出来ません．そのため，完璧なソフトウェア構築は不可能であり，バグを内包する製品を出荷してしまうことになります．もちろん，ソフトウェアの規模が大きくなるにつれて，含まれている潜在的なバグの数も多くなります．

ただし，インターネットの普及により，ソフトウェアの種類によっては不具合があること前提に出荷することがあり，出荷後不具合が発見された場合は速やかに修正版を配布するという開発も行われます．

4. 修正可能

ハードウェアに比べてソフトウェアは変更や破棄が容易になります．実態がなく材料がいらないという特徴から修正自体は簡単に実施できます．もちろん，どのように修正しなければならないかは別の問題になります．そのため，修正や破棄が容易な事からソフトウェアの改良や刷新が簡単な事であると勘違いする場合もあります．

5. 複写可能

ハードウェアと異なり製品の量産，流通コストが低くなります．ソフトウェア自身の複写はいくらでも可能ですが，ソフトウェアを配布する媒体を量産する必要が出てきます．ただし，配布媒体もインターネットの普及により低コスト化してきており，物理的な商品の流通網を利用することなく，配布が可能になります．ただし，これらは違法コピーを容易にしてしまうという危険性につながり難しい問題となっています．しかし，ソフトウェアのコピーが容易であるということは，新規に開発する際にソフトウェアの再利用化を促進させる面もあり，一概に複写出来ないようにすることは避けなければなりません．

6. 要求される機能は社会情勢とともに絶えず変化

ソフトウェアに求められる機能拡張，性能改善，環境適合に関する要求が当たり前となり，常にソフトウェアが進化できる状態で開発しておかなければなりません．例えば，税金に関するソフトウェアの

場合，税率や仕組みが変わる前提でソフトウェアを構築しておく必要があり，そのように開発をしておくべきです．つまり，ソフトウェア進化が前提な開発をすることが求められています．

7. 波及効果が分かりにくい

ソフトウェアのある箇所を変更すれば，他の部分に新しいバグが作り込まれる可能性も大いにあります．その多くの原因は，あいまいな仕様，構造化されていない設計によるものですが，本質的にソフトウェアは複雑であり，修正箇所からの影響波及が多岐にわたります．さらに，この波及した箇所の特定が困難であるため，ソフトウェアの大幅な修正を困難なものとしています．動作しているソフトウェアの改善をすることで，新たな不具合が生まれてしまうのもそのためです．

8. バグは本人より第三者のほうが見つけやすい

ソフトウェア開発者の多くは，自分がコーディングしたプログラムに完璧さを望んでおり，エラーがないよう心がけてきた心理的安心感とプライドを持っています．そのため，思い込みがあり不具合が発生せいた場合もその修正箇所の特定に時間がかかる場合が多いです．第三者の開発者の方が発見を短時間に行える場合もあり，テストや不具合修正行程をどのような体制で実施するかは難しい点となっています．

9. 作成者の思想

ソフトウェア自体が複雑なものであり，また，その構築には様々な方法があります．多くのソフトウェアは，人間のノウハウ，経験，技能，考え方などに依存した部分が多く作成者しか解読できない場合も多く存在します．開発者が 10 人いれば，10 通りの実現方法が存在し，同様に動作するソフトウェアも中身は全く異なる事があります．つまり，ソフトウェアを構築する過程に置いて，その制作者の思想が反映されるため，他人の作成したソフトウェアの読解が困難となるため，作成者がいなければ変更が出来ないという状況が生まれます．

2.3 よいソフトウェア

それでは，良いソフトウェアはどのようなソフトウェアなのでしょうか．以下に説明していきます．

1. ユーザの要求仕様が正しく反映されているソフトウェア

どんなに高度な技術を駆使して作成されたソフトウェアでも，ユーザニーズを満足していなければ価値効果がありません．ユーザのニーズをいかに正しく形式化し，ソフトウェアとして構成できるかがポイントになります．

2. 潜在的なバグが少ないソフトウェア

開発の初期の段階から極力不具合を少なくするという方策があるソフトウェアが良いソフトウェアになります．開発体制や各開発工程（ソフトウェアプロセス）が事前に定義してあり，その通り実施できる場合は不具合が少なくなる傾向にあります．ソフトウェアのテスト工程においても，ある基準に従いテストを実施し信頼度を高める必要があります．

3. 見積もりコスト以内のソフトウェア

商用ソフトウェアにおける重要な要素として開発コストが上げられます．開発コストとは単純に考えるならば，開発人数と開発日数の積で求まる値になります．もちろん，開発コストを費やせば費やすほどよいソフトウェアができる可能性がありますが，ソフトウェアの規模と機能内容の複雑さに対して，適切なコストを設定しなければ商売として成り立ちません．想定された見積もり工数内のコストでソフトウェアが完成し，納期内に開発が完了することがよいソフトウェアと言えます．

4. 運用，保守が容易

要求仕様通りの機能が実現出来ているソフトウェアでも，運用しやすいソフトウェアであることや利用が簡単であることがよいソフトウェアと考えられます．運用・保守も含めて考えられたソフトウェアを構築することが求められています．不具合が存在しないソフトウェアを作成することが困難な状況であるため，障害回復がしやすいソフトウェアを開発することが重要になってきています．

5. 安全性が高いこと

ソフトウェアにおけるセキュリティ対策を実施しているかも重要な要素となります．重要なデータを扱う場合の機密保護対策や障害が発生した場合の安全対策を事前に組み込んでいるソフトウェアはよいソフトウェアと言えます．

さらに，一昔前と現在では「よい」という観点が変化してきています．例えば，コンピューターのCPU やメモリといったリソースが高価で性能の低い時代においては，処理効率性が求められていました．処理効率を求める箇所では，アセンブリ言語と呼ばれる低級言語でソフトウェアの記述を行うことが多くありました．計算時間効率を高めるために，ファイルやデータベースへのアクセス時間の短縮化や処理手順の最適化し，資源効率を求めるために，CPU やメモリやディスクを効果的に利用するソフトウェアを作成する必要がありました．また，プログラムの保存領域の圧縮，プログラム実行時のCPU メモリ領域の最適化といった，ソフトウェアの本質的なでない処理に時間をかけていました．

しかしながら，現在のソフトウェア開発では理解容易性を求める傾向にあります．ソフトウェアを管理する工程では，わかりやすさが第一に要求します．複雑なソフトウェアを長期に渡って管理維持していくためには，多大な労力と経費が費やされるため，構成や設計構造が分かりやすいソフトウェアがよいソフトウェアになります．第三者が見ても理解しやすいよう作成することが重要な課題です．また，検査がしやすいソフトウェアを目指すことも重要になってきています．品質検査は必須な工程であり，見やすくわかりやすいソフトウェア記述であることや部品化が行われ機能の独立性が保たれていることが現代のソフトウェアにとっては重要な要素となります．さらに，高品質のドキュメントがあることも重要になりますが，ドキュメントがなくとも利用できるソフトウェアの方が望ましくなってきています．

人間の直感的に分かりやすいソフトウェアを作成する必要があり，場合によっては，操作に必要なチュートリアルを作成する必要も生じます．

3. ソフトウェア開発における問題点

ソフトウェア開発を取り巻く環境の変化は速く，世の中で提案・実施されている開発手法や技術は絶えず進化し続けています．数年前の技術が過去の物ということはよくあることです．常に新しい技術が提供される一方で，ソフトウェアを発注する顧客の要望は日々高くなっています．開発コストの増大やソフトウェアの複雑化に伴い，当初の計画通りに完了しないことも多々見られます．まだまだ人間に依存している部分が多く，決定打となる解決方法は見つかりません．ソフトウェアに開発に「銀の弾丸」は存在しません[3]．

また，そもそも「何をすべきか」を決めることが非常に困難なソフトウェア開発の中で，高度な IT 技術者不足から多段下請け構造に走っているソフトウェア開発の世界にも問題があります．これには，ソフトウェアの本質的な難しさビジネス構造上の問題があります．情報系以外の他の工学分野においては，これまでの経験と研究成果からやるべきことが明確になっており，それをいかに達成するかという視点から実施され，それだけの基盤ができています．しかし，ソフトウェアは何をどこまでやるかの合意形成を関係者間で取ることが本質的に困難な状況です．それは，人間が本来持っている曖昧な部分とソフトウェア開発等に必要な明確な部分とのギャップが原因と考えられます．この本質的な問題が常に存在し，この解決を図るために様々なソフトウェア工学の試みがあります．

ソフトウェア開発プロジェクトの実態を風刺した有名な画像があります．図 1 にその風刺画を示します．

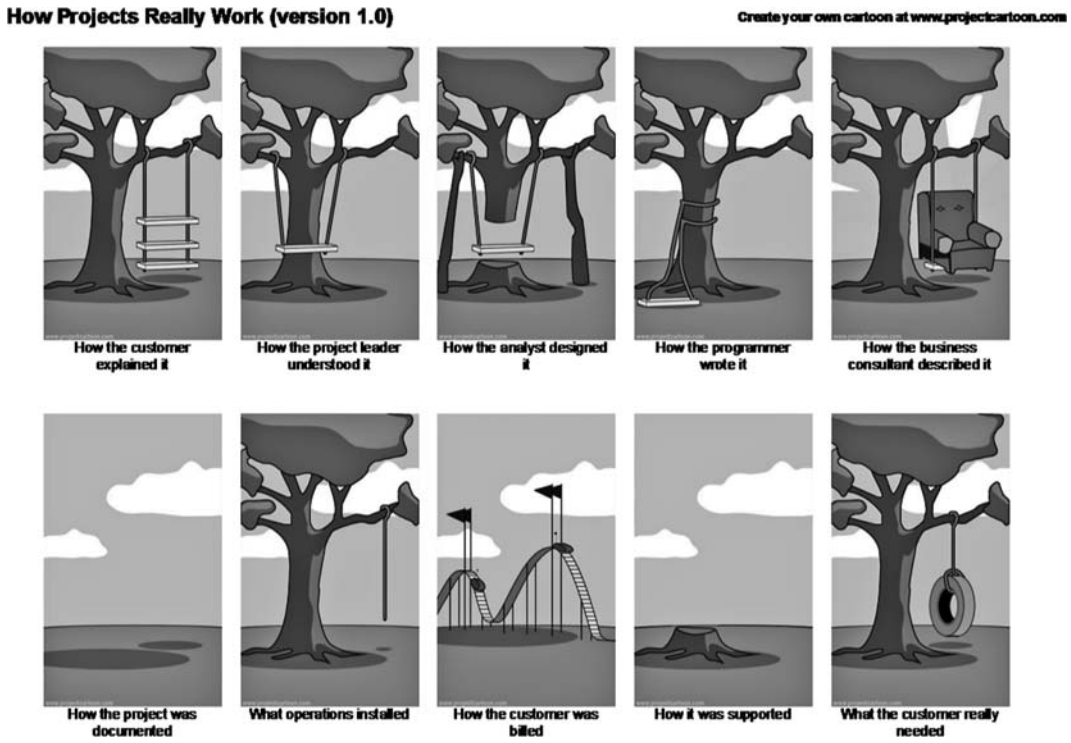

図 1　顧客が本当に欲しかった物（http : //www.projectcartoon.com/cartoon/ より引用）

図の左上の図から説明していきます.

・顧客が説明した要件

顧客自身がソフトウェア開発担当者に言った説明になります．もしくは担当者が聞き出した結果になります．この段階では板のブランコが欲しいという要求と見て取れます．ただし，よく絵を見るとブランコにしては板の枚数が多く，乗りにくい気もします.

この段階ですでにずれているというところでしょうか.

・プロジェクトリーダの理解

聞き出した要求をプロジェクトリーダーが理解した図です．たしかに形はブランコになっていますが，木の幹が邪魔をしてこげない状態になっています．プロジェクトリーダーは表面上の要求を理解しますが，その中身まで深く考えないことが見て取れます.

・アナリストのデザイン

アナリストが分析した結果のデザインになります.

機能的にはブランコという点は満たすようになりましたが，半ば強引にブランコに仕上げた状態になっています．要求から振り返らず，都合良く改変したものと考えられます.

・プログラマのコード

確かに「木の板」木に付けられてブランコのようなものに仕上がっています．その点では間違ってはいませんが，そもそも機能要求を満たしているのかと考えると満たしていない事になります．なんとか作成はしたが，まったく使い物にならないということを表しています.

・営業の表現，約束

往々にして営業は過大な表現を顧客に説明して仕事を取ってきます．その様子を表す一枚になっています．自分の所でソフトウェアを開発してもらえる

と，こんなにも素晴らしい物が出来ますよというアピールをするということです．

・プロジェクトの書類（ドキュメント）

ドキュメント化の状態を表しています．そんなものはまったく存在せず，後回しにした結果，結局作成されていないとうことです．

・実装された運用

結局木の枝にロープが付いた状態までしか実装されておらず，ブランコとして運用はできていない状態を表しています．

・顧客への請求金額

開発を重ねた結果，最終的にはジェットコースターが作れる程度の金額だった事を表しています．多くの時間と人件費がかかり，請求が膨大になってしまうことが見てとれます．

・得られたサポート

作成したソフトウェアに対して実際に得られるサポートはこの程度であることを表しています．ほとんどサポートが得られない状態です．

・顧客が本当に欲しかった物

実は顧客が本当に欲しかった物は，木の枝にロープを下げて，その先にタイヤが付いているものが欲しかったというオチです．木のブランコ程のものではなく，タイヤで十分だったということです．これは，顧客自身も何が欲しいのかうまく伝えられない事を意味しており非常に重要な観点となっています．

顧客自身もそもそもの要求を理解しておらず，その要求を使える時点で待ち上がっており，その後，多くの関係者にその要求が伝わる中で，各担当者の理解のずれやスキル不足等から全く異なるものが出来上がってしまうという問題を表現しています．

4. ソフトウェアテストにおける問題点

次にソフトウェアの具体的な難しさを説明するためにテスト工程の例を挙げて説明します．テストの底で有名な話として「マイヤーズの三角形問題」[4]というものがあり，その例を用いたいと思います．

4.1 マイヤーズの三角形問題

マイヤーズの三角形問題とは，「ソフトウェアをテストするのに十分と思われる一連のテストケース（すなわち十分なデータのセット）を書いてください．このソフトウェアは 3 つの整数を入力とし，この 3 つの値はそれぞれ三角形の 3 辺の長さを表すものとする．ソフトウェアは三角形が不等辺三角形・二等辺三角形・正三角形のうちどれであるかを決めるメッセージを出力する」という問題です．

さて，このソフトウェアが不具合なく動作することをテストするためには，どの程度の入力を考えてテストしていかなければならないのでしょうか．以下にその解答例を列挙していきます．なお，3 辺の入力をそれぞれ A, B, C として説明していきます．

1．有効な不等辺三角形として成立するかのテスト（A と B と C がすべて異なるとき）

例 ［A＝3, B＝4, C＝5］

2．有効な正三角形として成立するかのテスト（A＝B＝C のとき）

例 ［A＝1, B＝1, C＝1］

3．有効な二等辺三角形として成立するかのテスト（A と B が同じとき）

例 ［A＝2, B＝2, C＝3］

4．有効な二等辺三角形として成立し，2 つの等辺を含む 3 種類の組み合わせのテスト（A＝B，B＝C, C＝A のとき］

例 ［A＝3, B＝3, C＝4］［A＝3, B＝4, C＝3］［A＝4, B＝3, C＝3］

5．入力した項目が 0 の場合のテスト

例 ［A＝0，B＝3，C＝4　他 1 つが 0 の場合の 3 パターン］［A＝0，B＝0，C＝1　他 2 つが 0 の場合の 3 パターン］

6．入力した項目がマイナスの場合のテスト（入力が整数であるためマイナスも考慮してテストする必要がある）

例 ［A＝－1，B＝3，C＝4　他 1 つがマイナスの場合の 3 パターン］［A＝－1，B＝－1，C＝1　他 2 つがマイナスの場合の 3 パターン］［A

＝－1, B＝－1, C＝－1］

7．全ての項目が 0 を超える整数だが 2 つの項目の和が残りの 1 つの項目と等しい場合のテスト（A＋B＝C のとき）

例　［A＝1, B＝2, C＝3］

8．「7」の項目において 3 種類の組み合わせのテスト

例　［A＝1, B＝2, C＝3］［A＝1, B＝3, C＝2］［A＝3, B＝2, C＝1］

9．全ての項目が 0 を超える整数だが 2 つの項目の和が残り 1 つの項目より小さい場合のテスト（A＋B＜C のとき）

例　［A＝1, B＝2, C＝4］

10.「9」の項目において 3 種類の組み合わせのテスト（A＋B＜C, B＋C＜A, C＋A＜B）

例　［A＝1, B＝2, C＝4］, ［A＝1, B＝4, C＝2］, ［A＝4, B＝2, C＝1］

11. 全ての項目が 0 の場合のテスト（A＝B＝C＝0）

例　［A＝0, B＝0, C＝0］ 0 のテストは特別視すべき項目であり，必ずテストする必要があります．なぜなら，ソフトウェア内部の判定箇所で 0 を含めるか復命内科の判定は人間が間違えやすい点の一つです．

12. 半角数字の整数ではない値が入力されている場合のテスト（整数以外が強引に入力されたとき）

例　［小数点，制御コード，半角／全角スペース，半角／全角英大小文字，半角／全角カナ，全角数字，かな，漢字　等］

13. 入力されている値が 3 個に満たない場合

例　［A が空，B＝3，C＝4　他 1 つが空欄の場合 3 パターン］, ［A が空，B が空，C＝1　他 2 つが空欄の場合 3 パターン］, ［A が空，B が空，C が空］

14. 13 種類のテストパターンを試さなければならない

また，入力した値に対して期待される表示結果と実際の表示結果が合っているかも必要になってきます．

以上のように三角形の状態を確認するだけの問題においてもいくつものテストケースを準備して検証しなければなりません．どこでミスを犯しているか事前には分からないため，様々なパターンを準備してテストをする必要があります．

5. おわりに

ソフトウェア開発の特性やソフトウェア開発における問題点を説明してきました．今後，皆さんがソフトウェア開発にたずさわる場面になりましたら，少し立ち止どまり，これらの事について今一度考えてみてはいかがでしょうか．ますます高度かつ複雑化する情報化社会において，ソフトウェアの重要度が増していくのは間違いありません．ソフトウェアエンジニアを目指す際は，よりよいソフトウェアの開発を目指していただければ幸いです．

参考文献

［1］ シャリ・ローレンス　プリーガー，ソフトウェア工学－理論と実践，ピアソンエデュケーション，2001.

［2］ ロジャー S. プレスマン，実践ソフトウェアエンジニアリング，日科技連出版社，2005.

［3］ フレデリック・P・ブルックス　Jr.，人月の神話，丸善出版，2014.

［4］ マイヤーズ，J.，ソフトウェア・テストの技法，近代科学社，2006.