WHITE PAPER | 2015 年 9 月

コストと複雑さ、網羅率

要求を使用した高品質なアプリケーション・デリバリの正確なコスト予測

ca technologies

目次

セクション 1

概要

ソフトウェア開発コミュニティでは近年、従来の手法の弱点に対応するための新しいワーキング・モデルを開発・導入しようという協調した取り組みが行われてきました。おそらくその最も顕著な例は、従来のウォーターフォール開発モデルの柔軟性のなさや並行作業ができないことに対応して起きた、アジャイル開発プラクティスへの移行です。

ただし、新しい柔軟性に優れた手法のメリットが論理的にも直観的にも明白である一方、そのことを証明するメトリクスがいまだにないことも明らかです。では、信頼できる客観的なメトリクスを使用してソフトウェア・プロジェクトのコストを予測できるようにするにはどうすればいいでしょうか？ソフトウェア・プロジェクトの 61% にキャンセルや問題（納期の遅れ、予算超過、機能の欠如）が発生するという事実から判断すると、それを予測するメトリクスは現在のところ現実には存在しないと考えるのが妥当でしょう。[1]

本書ではスケジュールどおりに高品質なソフトウェアをデリバーするコストを正確に予測できるようにするために、要求とその要求に関して定義できるメトリクスを使用して開発の成果を測定する合理的なアプローチについて説明します。また本書では 1 つの特定のパラダイム、フローチャートの必要性について説明し、アジャイル環境における要求の設計、コスト予測、変更管理などの課題に対してフローチャーがいかに魅力的なソリューションを提供するかについて例を示します。

セクション 2

要求：ソフトウェア開発の第一章

ソフトウェア業界では一般に、要求はソフトウェア開発ライフサイクル（SDLC）の根幹であると認識されています。要求は残りのプロセスを予測する元となる枠組みです（ただし後述するようにテストは例外で、これを修正しようという向きもあります）。このことを考慮すれば、開発が始まる前に要求のリストが存在すると想定するのは当然のことだといえます（これは多くの場合、この段階での唯一の情報です）。したがって、できるだけ早い時期に開発コストを予測する必要があるため、「この要求は見積もりの役に立つか？」と問うことは理にかなっています。

ソフトウェア・プロジェクトを考えてみましょう。たとえばオンライン小売業者のアカウント管理システムの要求を定義しようとしているところだとします。まず始めに高いレベルの設計が立案され、検討を要する主な領域についての概要が示されます。たとえば以下のようになります。

最も高いレベルの要求
アカウント管理
セキュリティ
決済
注文システムとの連携

最も高いレベルの要求がすべて特定されると、各項目について詳細な分析が行われ、次のレベルの要求が特定されます。さきほどの例は以下のようになります。

最も高いレベルの要求	次のレベル
アカウント管理	アカウントの追加 アカウントの削除 …
セキュリティ	パスワードのセキュリティ 決済詳細のセキュリティ …

ここまでは一般に受け入れられているベスト・プラクティスと一致しています。しかしコストの観点から見た場合、これはどのように役に立つのでしょうか？要求は粒度に従って分割されているため、以下のように開発の総コストを計算するための手法が提供されます。

1. 最も低いレベルの要求それぞれについて、実装のコスト（および / または複雑性）を計算する
2. 上のレベルそれぞれのコストは、次のレベルの要求のコストの合計に等しい
3. 最も高いレベルのすべての要求のコストの計算が済むまで手順 2 を繰り返す
4. 総コストは、最も高いレベルの要求のコストの合計に等しい

先ほどの例を使用して、以下の要求があると仮定します。

最も高いレベルの要求	次のレベル	最も低いレベル
アカウント管理	アカウントの追加	顧客の名 顧客の姓 顧客の住所 …
	アカウントの削除	アカウント番号
セキュリティ	パスワードのセキュリティ	AES 暗号化 SHA-256 パスワード・ハッシュ …
	決済詳細のセキュリティ	RSA 暗号化 …

最も低いレベルのコストを最初に計算し、入力します（ピンク色）。

最も高いレベルの要求	次のレベル	最も低いレベル
アカウント管理	アカウントの追加	顧客の名 (2) 顧客の姓 (2) 顧客の住所 (6) … (35)
	アカウントの削除	アカウント番号 (2)
セキュリティ	パスワードのセキュリティ	AES 暗号化 (70) SHA-256 パスワード・ハッシュ (40) … (125)
	決済詳細のセキュリティ	RSA 暗号化 (60) … (250)

これらを特定し入力したら、次のレベルでは、その下のコストの合計を計算します。

最も高いレベルの要求	次のレベル	最も低いレベル
アカウント管理 (47)	アカウントの追加 (45)	顧客の名 (2) 顧客の姓 (2) 顧客の住所 (6) … (35)
	アカウントの削除 (2)	アカウント番号 (2)
セキュリティ (545)	パスワードのセキュリティ (235)	AES 暗号化 (70) SHA-256 パスワード・ハッシュ (40) … (125)
	決済詳細のセキュリティ (310)	RSA 暗号化 (60) … (250)

したがって、この例では総コストは 592（つまり 47 + 545）になります。実際にはコストは通常、固定費を割り当てた単位を使用して表されます（1 単位 =1,000 ドルなど）。この例では「セキュリティ =545,000 ドル」です。

この例では数値は任意で、説明のためにのみ使用されています。実際に使用される場合は、個別の要求のコストを計算する必要があります（本書で後述する内部の複雑性に関する説明を参照してください）。

上記では、コストに関してマクロ的な観点から説明しました。ここからはよく聞かれる質問、「最も低いレベルの要求のコストをどのように計算するか」について説明します。この場合はマクロ的な観点に加え、別のアプローチを使用します。

セクション 3

要求に関する（認識されている）問題

まず、要求が主観的であるという考えを払拭することが必要です。すべてのエンジニアリング分野の中で、この考えは完全にソフトウェアだけに限られているようです。たとえば、基本的な仕様にエラーの余地があるとしたら、エンジニアが橋の建設を進めることは期待しないでしょう。しかしソフトウェア業界ではこれが標準的な状況です。筆者の経験もこの事実を確かに証明しています。なぜこのような状況になったのかは興味深い（しかしまた別の）問題です。

ただし、先の土木工学との比較は、より成熟した工学分野から手法を「借用」できる可能性を提起しています。

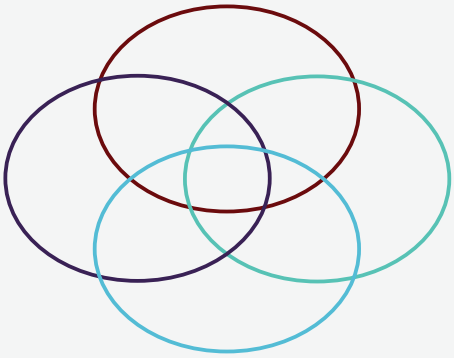
1970 年初期に登場した要求ベースのテスト（RBT）は、そのような例です。このテストはハードウェア・テストの中心の数学に基づいており、基礎となる論理によって 100% の機能網羅率を確保する、（比較的小規模な）テストケース・スイートを確立できる方法です。テスト品質の向上にこのテストを適用できるかどうかに関心がある方は、この参考資料を参照してください。[2] ただしここでもっと注目すべきは、要求の曖昧さに対する RBT の見方です。RBT は私たちが払拭しようとしている考えの基礎を形成しているからです。

RBT で使用する論理モデルを適切に構築するには、テストを実施できるようになる前に要求からすべての曖昧さを除去する必要があります。同じことがここにも当てはまり、各要求のコストを正確に計算するには、要求についてただ一つの解釈しか存在しないようにする必要があるのです。このトピックは本書が扱う範囲を超えていますが、曖昧性のレビュー・プロセスについての詳細をご希望の方は以下のリンクを参照してください。[3]

> Bender, R.、http://benderrbt.com/Bender-Requirements%20Based%20Testing%20Process%20Overview.pdf

曖昧さを排除する理由は、コストと複雑さを最小化し定量化するには、それを実装する全員の間で要求の解釈が一貫している必要があるためです。たとえば、1 つの要求に複数の解釈が行われることはあり得ます（ビジネス・アナリスト、1 人以上の開発者、テスト担当者による解釈など）。誤った解釈があるたびに、過りを修正するため一定量の再作業が必要になり、その結果コスト予測の精度が低下します。曖昧な要求の解釈は完全に主観的なものであるため、解釈の誤りが起こる数を予測することは不可能で、その数の上限と下限が得られるだけに過ぎません。

「解釈」をベン図として考えると、次のようになります。



円の重なり合っている部分は同じように解釈された要求で、重なっていない部分は同じ解釈が行われなかった要求です。ここで重要なのは重なる部分を最大化し、各グループが要求を同じように解釈することです。しかしこれを達成するための唯一の方法は、そもそも要求に曖昧さがないようにすることです。

有限のハウスドルフ測度として知られる数学的構成を使用すると、解釈の各セット間の「距離」を測定できます。曖昧性を測定する一貫した客観的な方法はまだ確立されていないため（このセクションで前述したプロセスは除きます。曖昧性レビューの結果を定量化する単一の方法としては十分ではありません）、これは本書が扱う範囲を超えています。しかし解釈間の「距離」という考えは、それがコスト予測の誤差の範囲にどのように影響するかに関して数学的根拠を示すために十分です。このセクションでは、解釈をお互いに近づけるために必要な再作業と、これをどのようにコスト予測に組み込むべきかについて考慮することで十分です。付録 A では、要求の曖昧さをコスト予測の誤差の範囲としてどのようにして数学モデルに組み込めるかについて説明しています。

セクション 4

内部の複雑さ

その他のアプローチ [4] では、ソフトウェアのサイジングが「テスト可能な要求」の数から直接測定可能であることが提唱されています。これは称賛に値するアプローチで、従来のコード行数（LOC）やファンクション・ポイント（FP）法が大きく改善されています。しかし、この特定のインスタンスを選ぶのではなく、現在までのいくつかの欠点を指摘する必要があります。

以前のアプローチは多くの場合、各要求の内部の（固有の）複雑さを無視していました。つまり、すべてのテスト可能な要求に同等の重みがあるように扱っていたのです。これはかなり単純な想定で、総コストを計算する際に大きな誤差に発展する可能性があります。最初の例では、複雑さとコストを反映するために各要求には個別の重みが付けられています。各要求に同等の重みが付けられるとしたら、要求間の違いは反映されないことになります。直観的に、コストの高い要求の実装にはより多くのコーディングが必要であると考えられます。

したがって、要求はそれぞれ固有の価値によって評価する必要があります。これには、「要求 X のコストが 1,000 ドルなら、要求 Y のコストは 2,000 ドルになる」というような想定を避けることも含まれます。なぜなら、コストと共に、個別の要求の「誤差の範囲」も考慮に入れる必要があるためです。ご推察のとおり、「誤差の範囲」にはほとんど一貫性がありません。

セクション 5

品質の測定としての予測力

上記を考慮に入れると、予測コストの誤差の範囲についどう考えるべきか、そしてこれは品質の解釈として使用できるのかという疑問が自然に浮かんできます。

もちろん予測は予測に過ぎず、予測が当っていたかどうかは後にならなければ判断できません。また、予測にはそれぞれの誤差の範囲が伴います。しかし通常、予測数値は高い値と低い値の間の範囲内で設定され、実際の値はこの 2 つの値の間のどこかになることが予測されます。天候モデルはそのよい例で、「予測値」（予測された実際の値）と、モデルで誤差の範囲として考慮される値の幅があります。

この考え方は、本書で使用している例にも適用できます。各要求に対して 10% の誤差範囲を考慮に入れるとします。これは予測コスト全体にどのように影響するでしょうか？下記の表は複雑性のメトリクス（赤）と、10% の誤差範囲（上限は青、下限は緑）を示しています。

最も高いレベルの要求	次のレベル	最も低いレベル
アカウント管理 **(47) (42.3) (51.7)**	アカウントの追加 **(45) (40.5) (49.5)**	顧客の名 **(2) (1.8) (2.2)** 顧客の姓 **(2) (1.8) (2.2)** 顧客の住所 **(6) (5.4) (6.6)** … **(35) (31.5) (38.5)**
	アカウントの削除 **(2) (1.8) (2.2)**	アカウント番号 **(2) (1.8) (2.2)**
セキュリティ **(545) (490.5) (589.5)**	パスワードのセキュリティ **(235) (211.5) (258.5)**	AES 暗号化 **(70) (63) (77)** SHA-256 パスワード・ハッシュ **(40) (36) (44)** … **(125) (117.5) (187.5)**
	決済詳細のセキュリティ **(310) (279) (341)**	RSA 暗号化 **(60) (54) (66)** … **(250) (225) (275)**

この例では予測コストの総額は 592 ± 59.2 になります（つまり誤差範囲は全体で 10%）。

統計的な観点からすると、誤差の範囲とは任意の差異のある確率変数と同じです。そこで各コストについて、誤差の要素を関連づけ、コストを以下のように表します。

$$C = c + \varepsilon$$

（ここで c は固定値で ε は誤差の要素とします）。統計的に言えば誤差の要素の予測値は「0」です。これについては直観的に、誤差が正の値と負の値になる確立は同等であると考えられます。誤差の重大さはその差異で表されます。一般的なモデル化手法を使用するために、すべての誤差要素は互いに独立して同じように分布し、すべて 0 と σ^2（σ は標準偏差）を予測値とする正規分布（ガウス分布）に従うものとみなします。上述の通り、この誤差範囲は曖昧性の「大きさ」に比例します（付録 A）。

統計における基本的な結果（中心極限定理 [5]）から以下のようになります。

$$Standard\ Deviation\ of\ Total\ Cost = \sqrt{n} * Standard\ Deviation\ of\ Single\ Requirement$$

従って、考慮する要求が多ければ多いほど、誤差の範囲の合計はほとんど直線的に増加します。

セクション 6

フローチャート手法の適応 — テストのコスト

要求の定義手法のひとつであるフローチャート手法は、ここで説明した複雑さのメトリクスを活用します。まず、3 つの所見があります。

1. 数学の用語では、フローチャートは有向グラフに過ぎません。フローチャートのブロックが頂点になり、その間の矢印は有向辺を形成します。
2. 複雑さメトリクスはそれぞれのブロックに付与されます。数学用語ではこれはグラフの各頂点に「重み」を適用することと言い換えられます。
3. テストケースはグラフを通して経路に変換できます。

これらの 3 つの所見は、私たちは十分に試行された多くのグラフ理論的概念を数学から活用できる可能性があることを意味します。グラフ理論はグラフ状の構造（エッジで結び付けられた一連のノードとして説明できるすべての構造）のみを扱う数学の分野で、数学分野のすべての範囲にわたってさまざまに応用されています。グラフ理論的概念をテストケース作成およびテストケースの作成と最適化の基礎としてどのように使用できるかについての詳細な説明は、こちらをご覧ください。[6] 本書では以下で上記の頂点、エッジ、経路の特性を使用するだけで十分です。

そのような 1 つの分野がオペレーションズ・リサーチとして知られる数学分野です。これはスケジューリングや経路の最適化などの問題を解決するためのアルゴリズムの応用に関する数学の分野です。たとえば「巡回セールスマン」問題はオペレーションズ・リサーチにおける有名な問題です。テストケースの設計における問題を解決するために経路に基づくアルゴリズムを適用する明確な場合だけでなく [7]、スケジューリング用の既存のアルゴリズムを活用するとプロジェクト・マネージャにとって非常に便利です。本書ではテストのコストの定量化に焦点を当てているため、このトピックは将来の記事に譲ることにします。

ここまでは実装コストのケースについてのみ検討してきました。当然、総コスト予測の全体像をつかむには、テストのコストについても考慮する必要があります。上記の第 3 の所見が理由で、グラフ理論はテストのコストに対して真価を発揮します。この目的に向けて、上記で詳述したようにコストを 2 つの別々のコスト、実装コストとテストのコストに分割します。

ここでの分析では、要求 j の実装コスト $i(Rj)$ と、要求 j のテストのコスト $t(Rj)$ について検討し、総コスト $C(Rj)$ がこれら 2 つの合計と等しくなるようにします。すなわち以下のようになります。

$$C(Rj) = i(Rj) + t(Rj)$$

テストケースは経路に過ぎないため、経路は以下のようにエッジの順序集合として表せます。

$$p = (E1, E2, \cdots, Enp)$$

ここでは完全な経路について言及しているため、各経路は始点ノードから始まり、終点ノードで終了するものとします。上記は頂点（ノード）の観点から以下のように再定義できます。

$$p = ((\pi, v1), (v1, v2), \cdots, (vnp - 1, vnp) , (vnp, \mu))$$

（ここで π は始点ノードを表し、μ は終点ノードを表します）。

経路の総テストのコストは単純に各ノードのテストのコストの合計として示せます。すなわち以下のようになります。

$$t(p) = t(\pi) + t(\mu) + \sum_{i=1}^{n_p} t(v_i)$$

コストの標準偏差は、正規分布した誤差要因を想定すると、以下のように要約できます。

$$stddev(t(p)) = \left(\sqrt{n_p + 2}\right)\sigma$$

これで各経路のテストのコストが計算できたので、次は経路の数（テストケースの数）に注目します。テストケースの数について文字どおり広く受け入れられている概念は循環的複雑度の概念[8]、つまりグラフ全体の線形的に独立した経路の数です。グラフ内の 2 つの経路は、ループ・バック・エッジを考慮に入れず、エッジの独自の集合で構成されているとき、かつそのときのみ、線形的に独立していると考えられます。このメトリクスをマクロ的始点と調和させる別の項目があります（実際には同じように規模変更できないため、また、McCabe はもともとこのメトリクスをコードの複雑性つまりマクロ的視点のために設計しました）が、ここではグラフ理論との関係のため、コスト予測の下限としてはこれで十分です。数学好きの方のために、この関係は循環的複雑度の正式な定義によるものです。これは終点ノードに関するグラフの最初のホモロジーのサイズ（最初の相対ベッチ数）として定義されます。つまりこれがグラフ全体の線形的に独立した経路の数です。これは、100% の分岐網羅を達成するために必要なテストケースの最小数と一致します。

分析のために、$P(G)$ はグラフ G 全体で可能な経路の総数を表し、$L(G)$ は G 全体の線形的に独立した経路の最小セットとすると、以下が成り立ちます。

$$L(G) \subseteq P(G)$$

$$M = |L(G)|$$

ここで、M は G の循環的複雑度を示します。100% の分岐網羅を満たす任意のテスト戦略について考慮します – これを $S(G)$ で表します。したがって、これは 100% の分岐網羅を満たすため、以下が成り立ちます。

$$L(G) \subseteq S(G) \subseteq P(G)$$

ここで経路の集合のコストを、個別の各経路のコストの合計であると定義します。すなわち、以下のようになります。

$$t(S(G)) = \sum_{i=1}^{|S(G)|} t(p_i)$$

コストは線形演算子（経路の集合の合計）であるため、以下も成り立ちます。

$$t(L(G)) \le t(S(G)) \le t(P(G))$$

また、循環的複雑度をまとめると、以下の上限と下限を直接導き出せます。

$$M * \min_{p \in L(G)} t(p) \le t(S(G)) \le |P(G)| * \max_{p \in P(G)} t(p)$$

簡単に言えば、下限は循環的複雑度 M に最も低コストの経路のテストのコストを乗算したもので、上限は経路の総数に最も高コストの経路のテストのコストを乗算したものです。

セクション 7

現実世界への適用

ここまでは理論的な説明が中心で、主に優れた要求を正確なコスト予測に変換する方法についての議論の構築に焦点を当ててきました。プロジェクト・マネージャなら誰でもわかることですが、これを実践事項に変換するには説得力のある論拠だけでは十分ではありません。ここでは優れたプロセスが絶対的な鍵となります。この理論の適用性は大部分がフローチャート作成手法に関連しているため、フローチャート作成ツールの使用をこのプロセスの中心とするのは理にかなっています。

要求が関与する場面で優れたプロセスを構築する鍵は、ビジネス・アナリストが思考と論理を明確に形成できるような手段を持つことです。以下は、要求の形式（ユーザストーリー、古いスタイルの要求管理手法など）に関わらず同様に適用可能です。

1. 要求が長々と書かれた文のかたまりである場合、まずそれを個別の箇条書きに分けることが必要です。
2. 箇条書きに分けるプロセスは反復可能であることが必要で、箇条書きの各項目に 1 つの（そして唯一の）ステートメントがあるだけになるまで分割を続けます。
3. フローチャートの枠組みではユーザストーリーとユースケースが経路に対応するため、それらがどのように相互に影響するかを考慮することも必要です。ユーザストーリーとユースケースにはある程度の共通性（共通するプロセス・ブロックなど）があり、そのことはフローチャートの全体的な設計で考慮に入れる必要があります。ユーザストーリーがすべての可能性を網羅することはほとんどないため、これは欠けているテストケースを後処理で特定する際に必要なことです。

簡潔で明確な箇条書きで要求を策定したら、これらの箇条書きの項目をフローチャートにマッピングすることは難しくありません。1つの項目を正確に1つのブロックにマップすることが重要です。練習として、その後フローチャートから要求のテキストを再構築することは、非常に有益です（全体的として行っても経路ごとに行ってもかまいません。重要なのはフローチャートが理解しやすいことで、この方法でそれを確認できます）。優秀な設計ツールでは、自動的にフローチャートから直接、要求のテキストを再構築できます。

次のフェーズではフローチャートが設計できたら、ツールによって作成されたすべての経路に目を通し、「筋が通っている」ことを確認します。これは「テストケース・レビュー・プロセス」（per (1)）と呼ばれますが、筆者の経験では多くの場合これは要求の検証においてきわめて貴重な作業です。このプロセスによって元の要求に重要なことが欠けていたケース（ビジネス・アナリストが考え付かなかったり「無意味」だと思われていたりするが、要求ごとに起こりうるユースケース）を多数発見できたことを筆者は証言できます。

フローチャートを作成したら、次に、個別の要求（プロセスまたは意思決定ブロック）のコスト（または複雑性）を評価する必要があります。前述のとおり、この「コスト」は実装コストとテストのコストに分けられます。これは最も重要ですが、プロジェクト・マネージャにとっては実行するのが最も難しいことです。これらのコストはある程度の精度で評価しなければ、全体的なテスト予測も不正確になってしまいます（上記参照）。そのため、これらの個別要求の実装とテストの両方に精通した各分野のエキスパートからの情報が必要です。ここで指摘

しておかなければならないのは、要求が詳細で明確であればあるほど、SME にとってはコスト（および必要に応じて誤差の範囲）の予測が容易になることです。

個別のコストの説明が付けば、プロジェクト・マネージャは最初の 2 つのセクションで注目した戦略を使用して総コストを評価でき、誤差の範囲を使用して上限と下限を予測できます。適切な要求ツールがあれば、ユーザからの情報なしでこの分析を迅速に実行できる可能性が高くなります。

セクション 8

変化する要求

要求が変化しないままでいることはほとんどありません。要求を明確化し曖昧性を排除するプロセスでは、ビジネス・アナリストとユーザの間で必然的に要求が受け渡され、それが繰り返されるたびに変更が行われます。ここで疑問が起こります。コスト・メトリクスでは差異はどのように説明できるでしょうか？

優れたフローチャート作成ツールは変更を一連の要求に保管し追跡する機能を備えていることも必要で、これは「改訂履歴」または「バージョン管理」機能として考えることができます。これによってビジネス・アナリストは「バージョン」を通して要求の変更を追跡でき、新しい情報に従ってコストを調整できます。要求の各バージョンが進歩するのに応じて、コスト・メトリクスを要求の各バージョンに対して計算でき、それによって各バージョン間の差異を計算できます。これはビジネス・アナリストとユーザの間で行われる改良プロセスにとって非常に貴重です。ユーザは「特定の機能を微調整するとしたらコストはどのくらいかかるか」を知ることができ、ビジネス・アナリストはこれをフローチャートに組み込んでコストを修正できるからです。

また、ここから派生したメリットとして、要求が変更された場合の全体的なテスト作業が軽減されます。どのテストケースを実行する必要があるかを割り出すために手間をかけるより、テストケース全体を再実行する方が簡単だとテスト・チームが考えた場合に隠れたコストが発生することがよくありますが、このアプローチを適用することで以下が可能になります。

- すべてのテストを実行するのではなく、数多くあるテストのうちどれを実行する必要があるか自動的に識別する
- テストの自動修復する。これは現在多くが手作業で行われているプロセスであり継続的な開発戦略において役立ちません。
- 作成する必要がある新しいテストケースを容易に特定する
- 最後に、冗長な、または重複するテストを排除する。テスト・チームは網羅率に関する全体的な影響について心配するため、決してテストケースを削除しません。

セクション 9

まとめ

本書では、要求がプロジェクトのコスト予測に与える影響と、堅固で明確な要求は欠落した情報が少ないため誤差の範囲を狭められることついて概説してきました。

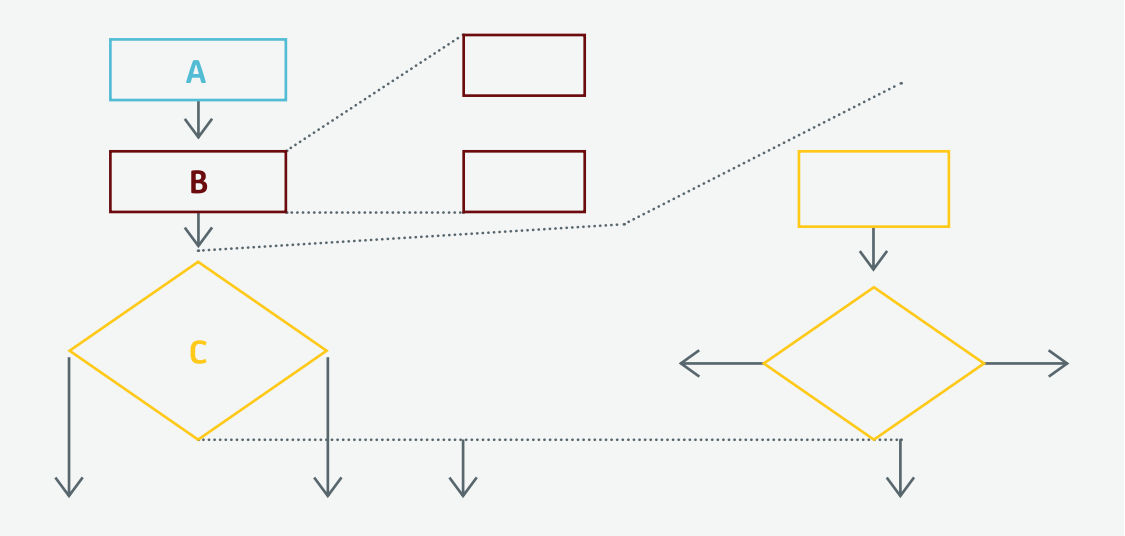
また、テストのコストを実装コストから切り離す戦略と、テストのコストを個別に計算する方法、そしてフローチャートを利用すればテストのコストと実装コストを容易に計算できることについて説明してきました。さらに、「要求が変わればコストはどのように変化するか？」という重要な問いが戦略全体から浮かび上がり、要求設計におけるフローチャート・パラダイムの優れた事例として位置付けられました。

セクション 10

付録 A

1 つのフローチャート（μ個の終点ノード）は n 個のノードで構成されます。$n_S = n_P + n_D + 1$ 個のサブフローチャート、ここで n_P はプロセス･サブフローチャートの数で、n_D は意思決定サブフローチャートの数を示します（したがって明らかに、$n > n_S$）。+ 1 とは、メイン･フローチャートはそれ自身のサブフローチャートであるためです。

例として以下の簡単な例について考えてみてください。



この例では、メインの（拡張されていない）フローチャートが左側にあり、B に対応するサブフローチャートが中央に赤色で描かれ、C に対応するサブフローチャートが右側にある黄色の部分です。この例では、

$$np = 1$$

$$nd = 1$$

$$n_S = 3$$

それぞれのサブフローチャート F_i に N_i 個のノードと E_i 個のエッジ、そして P_i 個の接続されたコンポーネントがあるとします。接続されたコンポーネントの数 P_i は出口ノードの数に等しいため、m 個の出力がある意思決定では P_i は m に等しくなります。

M_i は各サブフローチャートの循環的複雑度を表します。したがって以下のようになります。

$$M_i = E_i - N_i + 2P_i$$

すべてのフローチャートの総循環的複雑度は以下に等しくなります。

$$M_T = \sum_{i=1}^{n} E_i - (\sum_{i=1}^{n} N_i - n_p - n_d) + 2$$

（それぞれの意思決定サブフローチャートとプロセス・サブフローチャートが 1 つの大きいフローチャートに組み戻されるとき、1 つのブロックがなくなります）。

再配置して P_i の条件を導入すると以下のようになります。

$$M_T = \sum_{i=1}^{n} (E_i - N_i + 2P_i) - 2(\sum_{i=1}^{n} P_i) + n_p + n_d + 2$$

各意思決定には d_i 個の出口ポイントがあり、各プロセスには 1 つの出口ポイントのみがあり、メイン・フローチャートには μ 個の出口ポイントがあります。

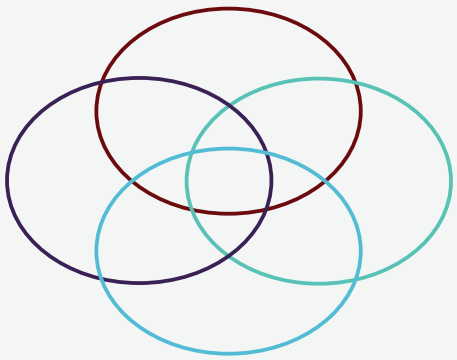
$$M_T = \sum_{i=1}^{n}(E_i - N_i + 2P_i) - 2\left(\sum_{i=1}^{n_d} d_i\right) - 2\left(\sum_{i=1}^{n_p} 1\right) - 2\mu + n_p + n_d + 2$$

したがって以下のようになります。

$$M_T = \sum_{i=1}^{n}(E_i - N_i + 2P_i) - 2\left(\sum_{i=1}^{n_d} d_i\right) - n_p + n_d + 2(1 - \mu)$$

$$M_T = \sum_{i=1}^{n} M_i - 2\left(\sum_{i=1}^{n_d} d_i\right) - n_p + n_d + 2(1 - \mu)$$

セクション 11



ユーザが把握している自らの要望

アナリストが把握しているユーザの要望

プログラマが作成するコード

テスト担当者によるソリューションのテスト

付録 B：コスト予測へのハウスドルフ測度の適用

この分析では、「要求のセット」を単純に特定のグループが把握している要求のセットとしてモデル化します（たとえばユーザの解釈のセットは赤色で示され、アナリストの解釈のセットは紫色で示され、交わった部分はその 2 グループが共通して理解していることです）。私たちは各グループの解釈間の差異を測定するためにハウスドルフ測度を導入します（基本的にセットを幾何空間として扱い、ハウスドルフ測度を使用して、幾何空間の間の距離を測定するようにセットとセットの間の距離を測定します）。

I_U、I_B、I_P、I_T はそれぞれユーザ、ビジネス・アナリスト、プログラマ、テスト担当者の「要求セット」または「解釈」を表すものとします。

以下であるとします。

$$\Theta = \{IU, IB, IP, IT\}$$

（すなわちすべての解釈のセット）

この場合、$X, Y \in \Theta$ について、ハウスドルフ測度は以下のように定義されます。

$$d_H(X,Y) = \max\left\{\sup_{x \in X, y \in Y} inf\, d_H(x,y), \sup_{y \in Y, x \in X} inf\, d_H(x,y)\right\}$$

直観的に、任意のポイントを仮定すると、これはサブ空間 X からサブ空間 Y への移動に必要な最大距離として考えることができます。

したがって X, Y は有限です。

$$d_H(X,Y) = \max\left\{\max_{x \in X, y \in Y} d_H(x,y), \max_{y \in Y, x \in X} d_H(x,y)\right\}$$

言い換えると、有限のサブ空間に対する d_H の制約です。なぜなら有限のサブ空間では以下のようになるからです。

$$sup\ inf = sup\ sup = max$$

全体空間 Θ では、再作業をサブ空間の各ペアの d_H を最小化（理想的にはゼロに）するために必要な作業として表すことができます。

作業 $\omega_{X,Y}$ を、X と Y の間のハウスドルフ測度を最小化するために必要な作業量として定義します。以下のように表せます。

$$\omega_{XY} = c_{XY} d_H(X,Y)$$

任意の作業定数 cX,Y について。

P_θ は同値関係のもと Θ のすべてのペアのセットとします（すなわち Θ の正の 2 乗から重複を除いたもので、したがって $(X,Y) \in \Theta$ と $(Y,X) \in \Theta$ は同値）。基本的に P_θ は $(X,Y) = (Y,X)$ の関係のもと、Θ の正の 2 乗の同値クラスのセットと同等です。この例では以下のようになります。

$$P_\theta = \{(I_U, I_B), (I_U, I_P), (I_U, I_T), (I_B, I_P), (I_B, I_T), (I_P, I_T)\}$$

したがって総作業 $\Omega\Theta$ は以下のように表せます。

$$\Omega_\theta = \sum_{(X,Y) \in P_\theta} \omega_{X,Y} = \sum_{(X,Y) \in P_\theta} c_{X,Y} d_H(X,Y)$$

これは再作業のファクターが各解釈間の距離に比例するという主張の基盤を形成します。この再作業のファクターは全体的なコストの変動性に織り込む必要があります。ここでは、前述した標準偏差 σ は再作業のコストに比例しています。特に単一の要求 R を仮定すると、変動性は以下に比例します。

$$\sigma_R \propto \Omega_{\theta_R}$$

セクション 12

参考資料

1 Standish Group、「Chaos Manifesto」、2013 年

2 「Bender RBT, Requirements Based Testing Process Overview」 2009 年

3 AmbiguityProcess. benderrbt.com. (オンライン) http://benderrbt.com/Ambiguityprocess.pdf

4 Wilson, Peter B、「Sizing Software With Testable Requirements」

5 Kallenberg, O.、「Foundations of Modern Probability」、ニューヨーク、Springer-Verlag、1997 年

6 Jorgensen、「Software Testing: A Craftman’s Approach」、2002 年

7 Leyzorek, M. 他、「A Study of Model Techniques for Communication Systems」、オハイオ州クリーブランド、Case Institute of Technology、Investigation of Model Techniques、1957 年

8 McCabe, Thomas J. (Snr.)、「A Complexity Measure, s.l.: IEEE Transactions on Software Engineering」、1976 年

セクション 13

CA Technologies のメリット

CA Technologies（NASDAQ：CA）は、複雑な IT 環境の管理と保護に役立つ IT 管理ソリューションを提供し、アジャイル開発のビジネス・サービスを支援します。CA Technologies のソフトウェアと SaaS ソリューションを活用することで、データセンタからクラウドに至るまで革新を加速し、インフラストラクチャを変革し、データとアイデンティティを保護できます。CA Technologies はそのテクノロジにより、お客様が必要な成果と期待どおりのビジネス・バリューを実現できるようにします。お客様を成功に導くプログラムの詳細については、**ca.com/jp/customer-success** をご覧ください。CA Technologies の詳細については、**ca.com/jp** を参照してください。

ca.com/jp/でCA Technologiesにアクセスしてください

CA Technologies（NASDAQ:CA）は、企業の変革を推進するソフトウェアを作成し、アプリケーション・エコノミーにおいて企業がビジネス・チャンスを獲得できるよう支援します。ソフトウェアはあらゆる業界であらゆるビジネスの中核を担っています。プランニングから開発、管理、セキュリティまで、CA は世界中の企業と協力し、モバイル、プライベート・クラウドやパブリック・クラウド、分散環境、メインフレーム環境にわたって、人々の生活やビジネス、コミュニケーションの方法に変化をもたらしています。詳細については **ca.com/jp** をご覧ください。

CS200-154748