テストマネジメント虎の巻 第七回

日本ヒューレット・パッカード株式会社
HP ソフトウェア事業統括 湯本剛

テストの見積り

皆さんこんにちは。今回のテストマネジメント虎の巻はテストの見積りです。テストの見積りは、工数と経費が予算やスケジュールに合うかどうかを判断し確定させる作業です。計画の中の一部の作業とも考えられますが、計画は目的を達成する方法を考えることで、見積りは計画が実現可能かを考えることであるため、分けて考えたほうが良いです。

見積りはいつするか

見積りには、工数と経費(テスト機材などの調達や出張旅費など)があります。経費のほうは、「第四回:計画編その 2 ～テストする環境とやり方をきめる～」にて解説したテストする環境を決めたところで、だいたい何が必要なのかがわかるため、それをベースに計算していきます。工数のほうは計画編の全部の情報(つまり、「範囲」「やり方」「スケジュール」)をもとにして見積ります。本来はこれら全部の情報がわかってきてから見積もったほうが正確に工数を予測できるでしょう。しかし、現実的にはそこまで分からない状態でも見積りをださなければならないことが多いと思います。特に受託開発の場合には、受注前の見積りをしなければならないことがあり、本当に微々たる情報で「どれくらいかざっくりと見積もっておいて！」などと言われたりします。こういう場合は仕方がないので、わかる範囲の情報で見積りをします。その後、不明な部分が明らかになった場合や、仕様変更などで状況が変わった場合に再見積りをして、コストとスケジュールの兼ね合いで実施可能なテスト内容へと微調整(時には大幅な修正)をしていきます。

また、計画の全部の情報をもとに見積りができればその見積りは完ぺきかといえばそうとも言えません。見積りは予測であり、不確実なものだからです。計画を作った後に判明する見積り不足のリスクもあります。例えば、プログラム設計やコードの複雑具合などのテストのしやすさにかかわる部分を早い段階で見積りに考慮するのは困難です。また、仕様変更の発生度合いや不具合検出件数などの見積り違いによる工数のブレもあります。とにかく、見積りは一度作ってからの見直しが何度も入ることを前提にして、再見積りをしやすい仕掛けを入れるなどの工夫をするようにしましょう。

まとめると見積りのタイミングには大きく以下の 3 段階があるといえます。

計画前の工数見積り

受注前の見積りは前述したように、微々たる情報で見積りをしなければならないことが多くあります。そういう場合は経験則をベースにした比率などを使って見積りをします。経験則は、今まで経験したプロジェクトの実績をもとにする方法です。

筆者は見積りに使う経験則の種類として以下のものを使っていました。

＜見積りに使う経験則＞

- 過去の開発工数とテスト工数の割合
- 過去のテスト実行工数とそれ以外のテストにかかわる工数の割合（もしくは作業ごとの割合）
- 過去の 1 テストケースあたりの実行工数
- 過去の不具合の発見件数
- 過去の仕様変更の発生件数

最もざっくり計算する場合は、開発工数とテスト工数の割合です。例えば、「開発工数におけるテストの工数の平均が、単体テストが 5%、統合テストが 15%、システムテストが 10%であれば、テスト工数は 30%となり、開発工数を 70 人日で見積もっているのなら、テスト工数は 30 人日確保する」といった具合になります。これは PM やエンドユーザが使う指標であり、ある程度の目安にはなりますが、現場の感覚では、実際にこの割合で仕事をやりきれるかはかなり疑問が残るでしょう。

もう少し現実味のある見積り方法としては、テストケース数を見積り、過去のテストケース実行工数からテスト実行フェーズの工数を割り出して、後はテスト実行工数以外の工数との比率を使ってテスト工数を見積もるというものです。例えば、「第五回：計画編その 3 ～スケジューリング（やることの具体化）～」で登場した旅行予約システムのバージョンアップ案件で考えてみましょう。まずは、新規画面であるオプショナルツアー選択画面の入力フィールドがいくつあるか、DB アクセスは何回するか、他の機能へ影響をどの程度与えるか、といった 3 段階で、それらの数の 3 倍~7 倍といったように見積ります。これはいわゆる「画面単体のテスト」であり、コードレベルの単体テストなら、if の数をベースに 2 倍のテストケース数という具合で算出します。また、シナリオテストはユースケースや業務フロー数、その他のテストは画面レベルの画面のテストの 0.2 倍の工数を確保するといったように算出します。* JSTQBFL シラバスでは、テストにはメトリクスベースと経験ベースの 2 種類があるとしていますが、メトリクスも自分たちの過去の経験から得た実績データであるため、本稿では総称して「経験則」としました。 ただし、ここで書いた、「3 倍～7 倍」、「2 倍」「0.2 倍」という数字は、これまでの筆者の経験（過去の実績を計測し続けた結果）の積み上げです。しかし、過去の自分がかかわったテスト作業の実績を計測し続けた結果をもとにしているため、ただの「勘」ではありません。ただの「勘」にならないようするためにはテスト実行時に実績工数を計測し続けることが非常に重要です。また、その工数になるテストケースの書き方、テスト実行の仕方もきめておかなければなりません。見積りの根拠は、自分たちのプロジェクトのテストの仕方に完全に依存するからです。自分たちにそういう過去の実績がまるで残っていない場合は、一般的な数字（Capers Jones の「見積りのすべて」）が有名です）を使うこともあります。ただ、それは皆さんのプロジェクトの現実を全く無視した数字であることを忘れてはいけません。テストケース一つのサイズも違うでしょうし、テスト実行の仕方もまるで異なるので、「外れて当然」だと思うぐらいでちょうどよいでしょう。そして、実績工数を必ず計測し、適時見直しをすることを強くお勧めします。

テスト実行の工数が出たら、過去の実行工数と実行工数以外の比率データを使い、テスト工数を算出します。これも一般的な指標としては実行 4 割、その他 6 割というデータがあります。(「テストプロセス改善」より引用していますが、この指標は筆者の経験ともかなり近いものになっています)

テストケース１つあたりの実行工数をもとにする理由は、他のフェーズよりも実績データを計測していることが多いからです。テスト実行フェーズの進捗状況の確認や遅れを挽回するときの計画にも非常に重要な数値であり、実績工数を

計測することでいろいろなことに役立つためです。筆者も現場にいた時は必ず計測していました。

計画時の工数見積り

計画前の見積りは精度も荒く、誤差も出ます。(そのため、組織によっては、過去の類似データの平均だけではなく、中央値と標準偏差も加味しているところもあるようです。)しかし、計画時の工数見積りではもっと多くのことが分かってくるので、細かく見積りをしなおします。一つの方法として、WBS(Work Breakdown Structure)を書いて、最下層のタスクの工数を見積り、その数字を積み上げていくという方法があります。WBS のポイントは、タスクを洗いざらいあげることと、どこまで細かくタスクを分解するかです。筆者の経験から、見積りとしての精度が高くなるためには以下の基準を満たすように最下層のタスクまで分解するとよいと考えています。

＜理想的な WBS の最下層タスク＞

- どのようなタスクなのかが具体的にイメージでき、説明できる。
- タスクから導かれる成果物がはっきりしている。
- そのタスクの担当者が特定できる。(人数は 1 人が理想。多いほど誤差が出る)
- タスクの実行期間を短く設定できる。(1 日の成果が明確になるのが理想。長いほど誤差が出る)
- タスクの実施中に他のタスクに影響を受けない。

上記の基準が完全に満たされる見積りを作るのは困難ですし、それが必須ではないと思っています。ただ、基準を満たせていないところに誤差が出るということをあらかじめ認識し、テストが始まったら調整していくことが大事になります。

また、不具合にかかる工数を見積もることも忘れてはいけません。そのために不具合の件数を見積もらないといけません。ここでも過去の実績データを使うことになります。ただ、テストの実行工数の見積りと違い、不具合の件数の精度は過去の実績データで推測するのは極めて困難です。なぜなら、不具合がどの程度出るかは、開発の状況次第であり、テストのやり方だけではコントロールできないからです。不具合の件数は 10 件なのか、100 件なのか、1000 件なのか、といった桁数単位で推測が当たる程度でしょう。筆者は、見積りをスケジュールに落とすときに、毎日不具合のための時間をテスト実行者 1 人当たり 1 時間入れるようにして計画を立てて、何件不具合が見つかるまで見積りを見直さなくてもよいかという数値を見積りに追記しておくようにしていました。例えば、不具合 1 件当たりにかかる工数を 15～30 分としておくと、1 週間で 1 人当たり 10～40 件までの不具合であれば残業なしでテスト実行ができるといった具合です。(これとは別に修正された不具合を確認するための工数も必要です。そちらは、検出した不具合の件数に見合うだけの修正工数が確保できる日程をスケジュールの中に明確に入れ込みます)なぜそのようなことを行うかというと、予測した不具合件数にリスクがあることを周りの関係者に理解してもらうためです。

上記の工数見積りの後、工数をスケジュールに割り当ててみて人員が何人必要かを精査します。工数の積み上げで人数を決めても、実際はスケジュールによって短期間であれば多くの人数が必要になりますし、長期間であれば少ない人数で済むからです。また、1 日の作業工数を 8 時間で計算してはいけません。なぜなら人間はロボットではないので、例えば、8 時間常にテスト実行を集中して同じスピードで行うということはできないからです。筆者は 1 日の作業工数を 6 時間でスケジュールに落としていました。後の 2 時間は、1 時間は不具合が出た時のための工数であり、もう 1 時間は人として必要になる工数(人と会話したり、トイレに行ったりといった工数)です。また、メンバーの数が増えるほどコミュニケーションのための工数を別途確保しないといけません。メンバーの数が 10 人以上であれば、1 日のテスト

実行工数を 5 時間で見積り、1 時間はコミュニケーションのための時間にあてるという対応が必要だというのが筆者の経験則です。

またアサインされるメンバーのスキルをどう補うかも計画に盛り込まないといけません。初心者であれば勉強の日程を確保しておかなければなりませんし、初心者の成果物レビューのために他のメンバーが工数をとられることも加味しておかなければなりません。具体的には、初心者の成果物レビューを担当する経験豊富なメンバーの作業工数を 1 日 3 時間でスケジュールに落としていくといったことを行いました。

計画後の工数見積り

計画後は、工数見積りした予定に沿って仕事をしていきます。まず、各メンバーには、計画時の WBS をメンバー自身でもっと具体的にしてもらい、見積りどおりに仕事ができるかを確認してもらいます。不安なところがあれば、申告してもらうようにします。そして、マネージャは見積り時にイメージした作業の仕方を説明し、メンバーのイメージと差異がないかを意識合わせします。見積りを納得してもらい、仕事を始めたら、今度はその見積りを守るように仕事をしてもらわないといけないので、事前の意識合わせは非常に重要です。実作業に入ったら、見積りは「メンバー各自が守るべき目標作業時間」になるのです。例えば不具合 1 件にかかる時間を 15 分～30 分で見積もっているということは、不具合が出たら 30 分以内に不具合レポートを起票し終わるように仕事をするということです。不具合の再現確認を 20 分やっても上手くいかない場合は他のメンバーに相談をして、対処方法を決めるようするといった対応で、30 分から大きくずれないようにすることが必要です。そして、テストケースの実行、不具合の起票といった、定義したタスクごとにどの程度工数を使ってどの程度の数をこなしているのかを計測し続けます。計測結果をみて、目標どおりにできているか、そうでないのかがわかってきます。

ただし、いくら努力をしても見積りどおりにいかないことがあります。前工程が遅れていたり、想定以上に不具合が出たりといった要因もありますが、自分たちが本来やるべきタスクが WBS から漏れている場合もあります。それらの場合は見積りを修正します。前述したように、不具合に関しては、何件見つかったら見積りの見直しをするということを計画時に宣言しておけば、今度は見積りの見直しをするかどうかを予測することができ、予め他の関係者と調整ができます。この時期だと追加予算は難しいため、予算の総量からテストに割り振ってもらうなどの対応が必要になるため、工数オーバーの予測は非常に重要です。

どちらにしろ、見積りの見直しは定常的に起こるものですので、必ず見直しができるように準備しておきます。見積りのパラメータとなるテストケース実行工数や不具合の件数などは計算式を作っておきスピーディに再見積りをしたり、作業順序の見直しができるように計画を立てる（例えばある不具合のために今テストすべきテストケースができなくなったときに別の日程で行うテストを前倒しして行う）といったことが該当します。

見積りの精度を上げるコツ

最も大きなコツは、前述した「見積りを守るように仕事をする」ことです。メンバー1 人 1 人の意識が見積りの精度を決定します。また、メンバーが守れるような見積りにするためには、実作業をイメージして見積りをすることが大事です。イメージできるということはいろいろな作業の仕方が確立できているということです。いわばテスト技術の向上が見積りの精度を高くします。また、そのやり方をメンバー間で共有できることが重要です。いわばコミュニケーションです。

というように、見積りの精度は計算式の正確さが重要なのではなく、見積りの数字に紐付くタスクを共有できるコミュニケーションの正確さが重要なのです。そしてもう一つ忘れてはいけないのが、多くの人たちを巻き込むようなタスクを作らないような努力をすることです。これは不具合をテストで見つけるのではなく、レビューなどの上流工程で見つけることが見積り精度を上げるということつながります。要求そのものを誤解していたという不具合であっても、設計前であれば要求関連のドキュメントを直せばよいだけです。テスト開始後、終盤に入って要求そのものを誤解していたことが発覚すると、要求分析の修正、ソフトウェア設計の修正（影響範囲の見直し）、コードの修正といったようにタスクにかかわる人が増えていき、見積りを大幅に修正しなければならなくなるからです。

まとめると以下のようになるでしょう。

＜見積りの精度を上げるコツ＞

- 実作業をイメージした見積りをする。
- 実作業の仕方を確立する。
- 実作業の仕方を共有する。（コミュニケーション手段の確保）
- 見積りを守るように仕事をする。
- 多くの人たちを巻き込むようなタスクを増やさない。

今回のテーマは見積りでしたが、テスト工数の見積りに焦点を当てて説明しました。かなりの部分が筆者の経験をベースにしていますので、賛同、もしくは意義ありといったコメントをいただければ幸いです。次回は、見積りの続きで「テストの投資効果」についての説明をしたいと思います。