WHITEPAPER

開発プロジェクトを成功に導く ベストプラクティス

本ホワイトペーパーでは、Qtプロフェッショナルサービスのエンジニアリングチームが過去数年間で発見し、現在も活用しているベストプラクティスの一部をご紹介します。これらのベストプラクティスを導入することで、開発プロジェクトのごく初期段階から自信に満ちた意思決定を行い、必要以上の時間と労力をかけることなくアジャイルかつ効率的にプロジェクトを成功に導くことができます。
なお、本ホワイトペーパーでご紹介する事例の多くはQtフレームワークに言及していますが、それ以外でも多くのソフトウェア開発プロジェクトに応用していただくことが可能です。

目次

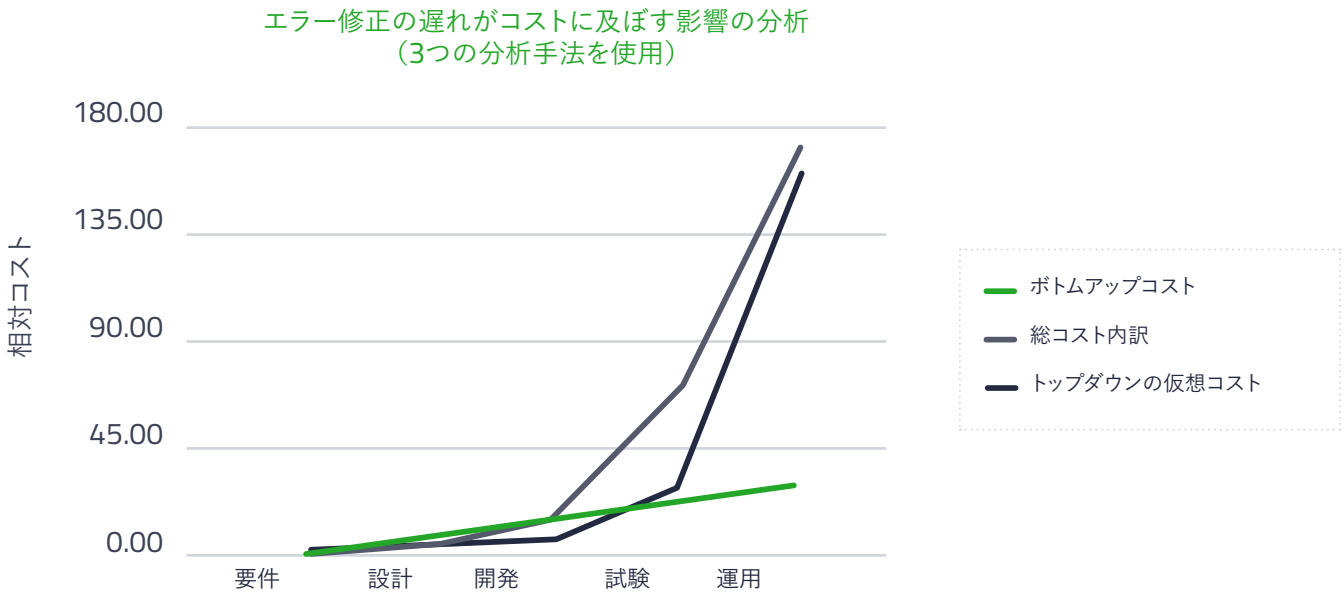
はじめに

入念に計画された開発プロジェクトであっても、製品リリースまでの過程を成功に導くのは容易ではありません。要件の定義ミスやチーム間の連携不足、その他さまざまな課題に直面するソフトウェア開発プロジェクトであればなおさらです。
このような困難な開発環境において、ベストプラクティスを導入せずに高い品質や迅速なデリバリーを実現することは不可能です。ベストプラクティスを絶えず応用することで、プログラマーとコードのアジャイル性を高め、拡張が容易で信頼性に優れたコードを書き、エラーの発生やスケジュールの遅延によるコストの増加を未然に防ぐことができます。

エラーのコスト

こんな話を耳にしたことがあるはずです…
「エラーのコストはプロジェクト進行のなかで、
発見が遅れるほど劇的に拡大し、最大で100倍に達する」[1]

NASAのチーム[2]も、大型宇宙探査機、軍用機、通信衛星という3つの開発プロジェクトの分析を行った際に、上記と同じ結論に達しました。チームは3種類の異なる分析方法（ボトムアップコスト、総コスト内訳、トップダウン仮想コスト）を用い、各プロジェクトライフサイクルのエラー発生時期に基づいたエラー修正コストの見積もりを行いました。3つの方法のいずれを用いた分析においても、コスト増とプロジェクトの段階の間に確かな関連性が明らかになり、場合によっては著しいコスト拡大が確認されました。



これらの分析結果は、開発初期段階でのエラーの発見が鍵を握ることを改めて裏付けるものです。また、この調査研究では指摘されていませんが、コスト増の問題がR&D活動以外の領域に及ぼす影響も見過ごせません。市場投入の時期が遅れれば、ユーザーは別のソリューションを探すようになり、結果的に企業の業績全体に負の影響が及ぶからです。

[1] Boehm, B. W. (1981). Software Engineering Economics, Prentice-Hall, Englewood Cliffs, NJ.

[2] Stecklein, J. M., Dabney, J., Brandon, D, Haskins, B., Lowell, R., & Moroney, G. (2004). Error Cost Escalation Through the Project Life Cycle. Report JSC-CN-8435 (Houston, TX: NASA Johnson Space Center, 2004).

要件を正しく定義する

開発要件の量と質はいずれも、製品のアーキテクチャに直接影響します。そこからさらに、実装とエラー修正のコストにも影響が及ぶことになります。周知の通り、設計初期段階でのエラー修正は、そもそも存在するべきではない問題に対する大量の次善策的コードを書くよりも、大幅なコスト削減になります。開発要件の定義に際して最も重要なのは、構築しようとしているシステムが有用性に対するニーズにマッチしているかどうかです。これは、開発手法がアジャイルであれウォーターフォールであれ、ほかの手法であっても変わりません。従って要件定義に当たっては、エンドユーザーと市場のニーズ分析を最初に行い、その結果を詳細に文書化することが大切です。これらのニーズを明らかにした後に、複数のユースケースを策定して、これから新たに開発するシステムをユーザーがどのように活用するかを記述します。アジャイルもしくは反復的なプロジェクト管理メソッドを用いるのであれば、初期のプロトタイピングフェーズで仮説の検証を行う必要もあります。この検証プロセスを経ることで、ハードウェアの制約がある中で製品の要件を確実に満たすことも可能になります。ハードウェアが手元にない場合、Qtを使った組み込み開発の初期段階は、デスクトップでエミュレータを利用して進めることになります。

プロトタイプについては、全体的な開発手法にかかわらず複数作ることをお勧めします。プロジェクトを成功させるためには、反復と改良が欠かせません。QMLを使えば、モックアップやプロトタイプの構築も簡単です。ここで忘れないでいただきたいのは、高品質な、あるいは完成品に求められる適切な構造を備えたプロトタイプはめったにないという事実です。通常、プロトタイプは数回の反復と改良を必要とします。そうすることにより、プロセスで再利用できるロバストなコンポーネントを作ることが可能になり、確立されたコーディングに従ってそれらをサルベージしたり、リファクタリングを行ったりできるでしょう。

未来を見据えた開発

コードについては、要件に書かれたプラットフォームでのみ実行できれば問題ないと思いがちです。しかしながらコードは長期間にわたって使われるものであり、その過程で絶えず変化が生じます。数年後には、アプリケーションを携帯電話やタブレット、その他のデバイスで実行したいと思うユーザーが出てくるかもしれません。

従って、アプリケーションの開発時には画面サイズや解像度、アスペクト比の異なる複数のプラットフォームで実行する必要性を最初から念頭に置くことをおすすめします。そのためには、ビジネスロジックとプレゼンテーションロジックを切り離して考えるのがベストです。これにより、アプリケーションの機能を損なう心配なく、後からUIを書き換えることが可能になります。これは、デスクトップからタッチスクリーン型デバイスに移行するような場合に特に有益です。このようなケースではUIのデザインアプローチも見直す必要が生じますが（Qt　WidgetsからQMLに移行するなど）、その際にビジネスロジックがUIレイヤーと切り離されていないと、作業が一層複雑になってしまいます。UIの拡張性に配慮しながら未来を見据えたアプリケーション開発を行うための詳細については、弊社ウェブサイトの拡張性に関するドキュメントをご参照ください　（http://doc.qt.io/qt-5/scalability.html）。

もう1つのベストプラクティスとしてご紹介するのは、新しいプラットフォームの導入後も保守が可能な、一貫性のあるルック&フィールを開発することです。一貫性のあるビジュアルは製品ポートフォリオのブランド認知度の向上および維持の面で効果的です。それにもかかわらず、デバイスごとにまったく見た目の異なるアプリケーションをリリースする企業は少なくありません。その場合は、カスタムメイドのビジュアル要素を開発した後、PhotoshopやMaya、MODO、Blenderといったサードパーティのデザインツールからデザインをインポートすることも可能です。

Qtを使ったUIデザインの詳細については、弊社ウェブサイトのGUIコンセプトに関するドキュメント（https://doc.qt.io/qt-5/qt-gui-concepts.html）をご参照になるか、弊社のエンジニアリング&プロダクトスタッフ（https://www.qt.io/contact-us）にお問い合わせください。

ハードウェアの選択は慎重に

ハードウェアの適切な評価は極めて重要なプロセスであり、このプロセスなしでは多くのプロジェクトが頓挫します。開発者は通常、組み込みハードウェアの完成を待つ間に、デスクトップやソフトウェアエミュレータで開発をスタートします。もちろん、この手順自体に問題はありません。重要なのは、実際のハードウェアや評価ボード上でのアプリケーションの実行をなるべく早く実施し、ハードウェアを期待通りに実行できるかどうかを評価することです。(Boot2Qt　参照：http://doc.qt.io/QtForDeviceCreation/qtb2-index.html　のようなソフトウェアスタックがあればプロトタイプの評価が容易に行えます。)また評価ボードを使う場合は、最終的なターゲットハードウェアに可能な限り近いものを選び、コストやパフォーマンス、リソース消費を適切に評価することが大切です

開発プロセスのなるべく早い段階でハードウェアの評価を行うことを怠ると、開発コストの拡大や開発の遅延、ユーザーへの普及の遅れといったさまざまな問題が生じる大きな要因となります。たとえば、コストを理由に低出力プロセッサを選択した場合、きれいなアニメーションや高処理能力のグラフィックは多用できません。もちろん、十分な労力と時間を投じれば、低出力のチップから多少なりとも優れたパフォーマンスを引き出せますが、ハードウェアの制約は常につきまといます。プラットフォームのメモリの制約が大きな課題となるなら、Qt Lite Configuration Tool(https://doc.qt.io/QtForDeviceCreation/qt-configuration-tool.html)を使えば、不要な機能を削除して実用性のある小さなバージョンを作り、小さなフラッシュとRAMフットプリントを使うことが可能になります。この方法なら大抵、部品表(BOM)全体のコストも削減できます。ローエンドハードウェア向けのデザインに関するその他のヒントは、弊社のブログをご覧ください(https://blog.qt.io/)。

Qtの強みの1つはマルチプラットフォームサポートなので、最適なハードウェアを探すのが簡単なことです。Qtのバージョンごとのサポートデバイスおよびコンフィギュレーションをこちらにまとめました(http://doc.qt.io/qt-5/supported-platforms.html)。弊社では小型化した最新のハードウェアのパフォーマンスベンチマーキングを行っており、どのハードウェアがお客様のプロジェクトに最適かお勧めできるので、コストのかかる実験工程を省いていただけます。ご興味のあるプラットフォームやコンフィギュレーションが上記のリンクページにない場合には、弊社にお問い合わせください(https://www.qt.io/contact-us)。リストに記載のないコンフィギュレーションについても豊富な導入／保守の経験があるのでご安心ください。

スタートは適切なトレーニングから

Qtは万能かつ柔軟性に優れたプラットフォームとして、あらゆる問題を解決する多種多様なソリューションを提供します。お客様のプロジェクトを適切にスタートさせるには、アプローチごとの長所と短所を正しく見極めなければなりません。Qtの基本について最低限のトレーニングを受講いただいてから導入に着手されるようお勧めしているのもそのためです。弊社では以下のように、多彩なトレーニングをご用意しています。

- Qtトレーニング（https://www.qt.io/qt-training/）
  – カスタマイズ可能で効果的
- フォーラム（https://forum.qt.io/）や
  ブログ（https://blog.qt.io/）などの各種コミュニティ
- 事例豊富な高品質ドキュメント
  （http://doc.qt.io/qt-5/supported-platforms.html）
- 自習用資料（https://www.qt.io/qt-training-materials/）

継続的インテグレーションの実践

継続的インテグレーション（CI、Continuous integration）とは、全開発者のワーキングコピーを早い段階で頻繁にメインラインに統合するプロセスを言います。開発チームがCIを実践することで、「統合地獄」とも呼ばれる統合上の問題に陥るのを防止できます。1日に1回または数回のCIを行うだけで、長期的に見た開発の時間とコストの最小化を図ることが可能です。

お客様のアプリケーションにパフォーマンス関連の要件（スピーディな起動やコンテクストスイッチなど）がある場合、それらの要件とターゲットハードウェアとの互換性を継続的に確認することをお勧めします。それにより、パフォーマンスに負の影響を及ぼす可能性のある変更を、より素早く容易に発見できます。
Qtベースのアプリケーションやライブラリのユニットテスト用フレームワークであるQt Test（http://doc.qt.io/qt-5/qtest-overview.html）の導入をご検討いただくと良いケースもあります。

Qt Testはユニットテスト用フレームワークに必要とされる全機能を提供するほか、グラフィックUIのテスト用の拡張機能も備えているので、統合時の様々な頭痛の種をなくすことができます。詳細は弊社ウェブサイトのチュートリアル（http://doc.qt.io/qt-5/qtest-tutorial.html）でご確認ください。

ビルドプロセスに包括的なユニットテストフレームワークを追加することで、低品質なコミットをブロックできるほか、網羅的なコンフィギュレーションテストを定期的に実行することができます。テスト結果を開発者に明快かつ簡潔に、ユーザーフレンドリーな形で伝えれば、効率的かつ即応性に優れたチームを育て、継続的インテグレーションへの投資を最小限に抑えられるでしょう。

チームの連携

いかに才能あふれたエンジニアリングチームがいたとしても、開発プロジェクトを複数チームで分担して進める場合、「連携」が不足しているという非常に大きな問題を抱えてしまうことがあります。個々のチームが常に効果的な相互コミュニケーションを図らなければ、APIのミスマッチや境界条件の定義不足、誤ったモジュールの利用といった負の要素をアプリケーションから排除することはできません。大規模なプロジェクトの各領域に複数のチームがサイロ化された環境で個別に取り組んだり、それぞれが独自のメソッドやデバイス、ツールを使って作業を進めるといった状況は珍しくありません。

このように分断された開発プロセスを数カ月にわたって続けた後、チームごとの成果を統合しようとしたところで、計画通りに事は運びません。このような事態を避けるには、各開発チームに同じツールチェーンとフレームワークコンポーネントを提供し（Qtであれば容易にできます）、各ツールや機能で可能な限りコードを再利用する必要があります。これを実践する効果的な方法の1つが、Qtを使ってカスタムSDKを作るというアプローチです。このアプローチなら、多数のチームメンバーとプロジェクトの間で一貫性を保つことができます。

無用な一からの内製を避ける

開発者が必要もないのにコードを一から書くことは珍しくありません。開発者や企業が社内開発のソリューションを好む例は実に多く、ソフトウェア開発上の問題解決に最適な社外の既製のソリューションがあっても内製することを選ぶ傾向があります。しかしこれは、特にQtが多数ご用意している既製のテスト済みコンポーネントと比べると、大変なコスト増につながります。Qtなら、コミュニティを通じてさまざまな専門知識を得ることも可能です。

新しいコードを書く場合でも、Qtが提供するあらゆるツールを活用することをお勧めします。たとえば、Qtには極めてロバストなクロスプラットフォームの抽象レイヤが備わっているため、プラットフォームやプロセッサ、OS専用のコードを書く必要がありません。“#ifdef OS_LINUX”でコードを書いているなら、その時点ですでに何らかの問題が発生します。

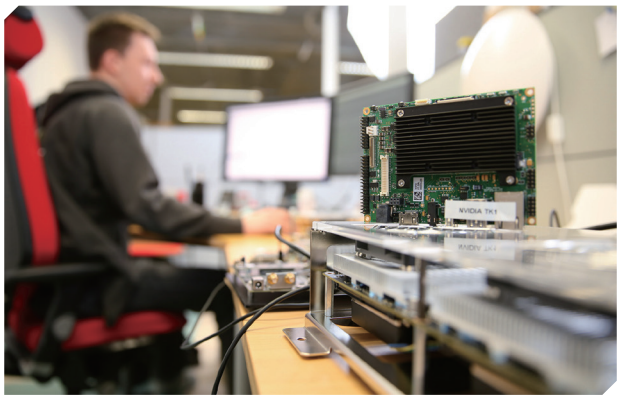
では、ハードウェアやドライバ、OSサポート（サウンド、Bluetooth、スレッド関数など）が異なる複数のターゲットにコードを書く場合はどうでしょうか？Qtは抽象レイヤを提供しているので、プラグインでプラットフォーム固有のソリューションを追加し、Qt APIで常時アクセスすることができます。これによりモバイルや組み込みといった新たなデバイスへのサポートを追加するのが容易になり、シミュレータがなくてもデスクトップ上で開発が行えるようになります。

必要なものはすでに社外で開発されている場合があり、それらを利用すれば、開発の時間と労力を節約できます。欲しい機能がQtに搭載されていない場合は、ぜひQtのロードマップをチェックしてください。お客様の現在のプロジェクトにちょうど間に合うよう、リリースされるものもあるかもしれません。

パフォーマンスの最適化とリファクタリング

最適化のタイミングとして適切かどうかを判断するに当たっては必ず、プロジェクト全体に及ぶ影響を第一に考えてください。多くの場合、デザインを行い、コードを書き、開発中のコードのプロファイリング／ベンチマーキングを行って、どの部分を最適化するか検討するのが有効です。

開発段階では、(コストと時間を浪費しない)妥当なパフォーマンスを検討し、ベストプラクティスに従って、繰り返しテストを実施するのがベストだと私たちは考えています。このアプローチを念頭に置き、最終的に不要になるかもしれないコードのプロファイリングや最適化には、無駄な時間を費やさないのがコツです。機能的な開発基盤を準備しておけば、後から最適化のプロセスを繰り返す時間と労力を節約できます。一般には、最適化は最後に行うことをお勧めしています。ただし、自動車業界のように起動時間が鍵を握る場合は、プロジェクトの着手段階で起動時間の最適化に特に注意を払う必要があります。起動の速いシステムを作るには、適切なアーキテクチャデザインが求められるからです。Qtベースのデバイスに極めてスピーディな起動性能を持たせるのは優れた設計性能(およびQt Quickならではの多彩な機能、適切なハードウェア、システム画像の最適化)があれば可能ですが、早い段階で目標を



明確に定め、これを早期に達成して、開発プロセスを通して維持し続けることが大切です。弊社の提唱するベストプラクティス(http://doc.qt.io/qt-5/qtquick-performance.html)をご確認の上、特に問題が発生しやすい「Models and Views」のセクションをよくお読みください。弊社ではコードのリファクタリングも、可読性を高めて複雑性を軽減し、最終的に保守や拡張を容易にするプロセスとして推奨しています。定期的なコードのクリーニングを適切に実行することで、基盤ロジックを簡素化し、不要な複雑性を解消して、目に見えない潜在的なバグや脆弱性を取り除くことができます。なお、リファクタリングは100%手作業で行う必要はありません。Qt Creatorなら、コンピュータのガイダンスに沿って迅速かつ簡単にコードのリファクタリングができます。

まとめ

開発者の多くは、自身が手掛けるソフトウェアそのものと、そのための開発プロセスの両方を改善する方法を探し求めています。具体的な改善方法としては、開発工程から当て推量を取り除いてくれる新たなプロセスやメソッドを追加する、あるいは、日々の実践を通してソフトウェアを書く作業を楽にしてくれるようなテクニックを体得するといったことが考えられます。しかし開発者にもっとお勧めしたいのは、弊社のスキルセットを活用し、チームとしてより効果的に協働し、より信頼性に優れた製品を迅速に開発するという方法です。

The Qt Companyでは、継続的なソフトウェアの改善の必要性を痛感しています。本ペーパーでご紹介するベストプラクティスを、お客様の手掛ける開発プロジェクトや会社全体のプロセスにもぜひご活用ください。ご自身のプロジェクトや会社全体のソフトウェア開発の改善に向けて一層のサポートを必要とされる場合は、弊社にお問い合わせください（https://www.qt.io/contact-us）。お客様のプロセス改善の実現に向けて、弊社ならではの専門知識をご提供いたします。