開発と並行したリバースエンジニアリングの実践

須藤 暢也†

概要:本来、開発済みのソフトウェアに対して行われるリバースエンジニアリングの手法を仕様変更が発生する開発状況にある組込みソフトウェアに適用した。実施したリバースエンジニアリングの実施工程とその中で直面した問題について報告する。

Practice of the reverse engineering in parallel with the software development

1. はじめに

2018 年度の組込み/IoT に関する動向調査[1]によると、組込みシステムの開発において、設計品質や開発能力の向上に対する課題意識が高いことが読み取れる。

本稿では、度重なる仕様変更と短期間での開発が求められたことにより、仕様に基づいた設計と開発資料の整備が十分に行えず、要求と仕様/実装との紐付けが困難となった開発中のソフトウェアに対して実施したリバースエンジニアリングについて報告する。

2. リバースエンジニアリングの目的と実施工程

2.1. 背景と目的

リバースエンジニアリングを実施することとなったソフトウェアは様々な事情により開発中の仕様見直しを繰り返された結果、設計書などの開発に必要とされるドキュメントの整備が追いつかず、要求との紐付けが整理されていない状況にあった。

そのような開発状況であるため、ソフトウェアの検証の際に適した検証ケースを用意できない、バグが確認されても解析に時間を要するという事態に陥っていた。

そこで、「ソフトウェアの内部構造とその振る舞いを整理し、検証結果の解析効率を上げる」、「コードから抽出した処理とソフトウェアに求められている要求・仕様を突き合わせることでその整合性をとる」、「変更に対して品質を担保できる検証ケースを整備する」ことを目的に、開発中のソフトウェアに対してリバースエンジニアリングを実施することとした。

† 日本アイ・ビー・エム株式会社 グローバルサービス事業部
Global Business Services, IBM Japan, Ltd

2.2. 実施工程

実施したリバースエンジニアリングの工程は「ソフトウェアの構造把握」と「検証ケースの策定」の二つに大きく分けられる。

まず、「ソフトウェアの構造把握」では既製のコード解析ツールを用いてコード上に存在する関数を洗い出し、その呼び出し関係を確認する。確認後は各関数の処理とその責務を確認し、検証ケースの作成も視野に入れた各関数の条件分岐(if, switch 等)及び各関数の処理で用いられる変数と関数間で受け渡される変数の確認を実施することで、対象となるソフトウェアコードの振る舞いに対して理解を深める。その上で、ソフトウェアの開発陣が作成した要求仕様書と突き合わせて、実装漏れや実装不備を確認する。

次に、「検証ケースの策定」では前述の作業でまとめた関数の条件分岐を組み合わせることで検証すべきケースを洗い出す。無論、機械的に組み合わせるのではなく、処理上意味ある分岐の組み合わせのみである。

洗い出した組み合わせは前述の作業で実装漏れや実装不備が無ければ、仕様通りにソフトウェアが動作するか確認できるものになっているはずである。

検証結果は、検証環境の都合上、事前に整理した各関数で用いられる変数の中で観測可能な変数の入出力の組み合わせで確認を実施することとした。

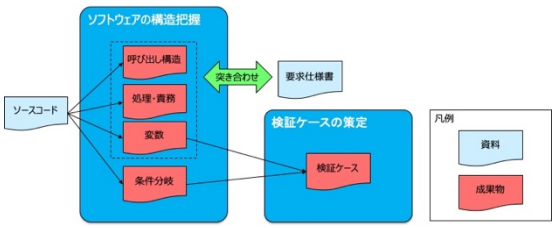

図.2.1 資料と成果物の関係簡易図

作業で利用される資料、作成される成果物は上記の通りである。

3. 直面した問題

3.1. 要求仕様と実装との不十分な突き合わせ

実装漏れや実装不備の確認のために要求仕様書とソフトウェアの実装の突き合わせを行ったところ、要求仕様書に記載されていない実装が多く存在した。

確認したところ、開発陣が作成した要求仕様書は過去のある時期の実装コードから逆起こししたものであり、以降の仕様変更に対する追いつきが完全にはできていないためであった。

結果的に突き合わせで行えたのは、要求仕様書と実装の食い違い、違和感の指摘がほとんどであり、本来目的とした実装漏れを確認できる状態ではなかった。

開発陣と確認した結果を反映した最新の要求仕様書の整備を依頼しているが、結局はコードからの逆起こしとなることは変わりないため、上位の要求とソフトウェアの仕様を結び付けられなければソフトウェアの品質を保証できないのではと懸念される。

改めて要求から仕様を書き起こせれば良いが、開発者・有識者が目先の開発に手一杯の中ではそれも難しいため、既存資料に対して都度発生する仕様の変更内容を反映し、上位の要求と結びつける作業を開発工程に組み込む等、ソフトウェアの品質確保のための工程を計画に盛り込んでいく必要があると考えている。

3.2. 更新への追いつき対応

2 章でも述べた通り開発中のソフトウェアに対してリバースエンジニアリングを実施したため、作業中にも仕様変更に伴うソフトウェアの更新が頻繁に実施された。

検証結果を元にソースコードに大規模の改変が突発的に行われることもあり、開発陣も変更するコード量を予想できる状況ではなかった。

そのため、開発のマイルストーンに向けてリバースエンジニアリングを実施したが、期間的猶予に対して作業対象とするコード量も膨大となり、期間内でのリバースエンジニアリングが困難となることが多々あった。

関数の呼び出し関係の精査などツールが適用可能な工程は自動化することで必要な工数を少なく一定にすることができたが、処理の振る舞いの把握はどうしても人手に頼らざるを得ずステップ数に比例して必要となる工数が大幅に増えた。

ツールを利用した振る舞いの把握も検討したいが、処理の意図の理解が重要であるため、全ての作業の自動化は難しいと考えている。しかし、コードの構造把握はある程度ツールでできる部分もあるため、その方向性で工数の削減に繋げられないか検討したいと考えている。

4. 最後に

近年、複雑なソフトウェアを短期間で開発することが求められている。また、今回のように検証結果を即座にソフトウェアに反映させなければならない開発では開発資料が陳腐化しやすく、開発が属人化することが考えられる。

今回はそのような状況に陥った開発プロジェクトの支援のために開発中のソフトウェアに対してリバースエンジニアリングを実施した。結果的に把握できる範囲での要求仕様の確認及び検証ケースの作成を達成した。

しかし、継続的に開発陣で同様の作業を実施できるようにするには可能な限り開発作業に費やす時間を減らさない形での開発資料及び検証ケースの最新化が重要となってくると考えられる。今回のリバースエンジニアリングの工程や問題に対するアイディアについて討議できれば幸いである。

参考文献

[1] IPA「組込み/IoT に関する動向調査」, 2018 年度
https://www.ipa.go.jp/ikc/reports/20190327.html