Vol.2014-SLDM-165 No.10
Vol.2014-EMB-32 No.10
2014/3/15

組み込みソフトウェア開発の効率化適用事例紹介

山上　真広 , 前田　善行
富士通九州ネットワークテクノロジーズ㈱
{ m12yama, maeda.yoshiyuki } @jp.fujitsu.com

要旨

弊社は通信システムの開発会社であり，スマートフォンのような通信端末からキャリア向けの通信装置まで，通信システム全般の開発を広く行っている．

通信装置内に搭載されるソフトウェアは，ネットワークの高速・大容量化に伴い，開発規模が拡大している一方，通信市場のニーズ応じたスピード開発が求められ，さらなる品質・生産性を向上させる必要がある．

本稿では，弊社における組み込みソフトウェア開発力強化の取り組みとして，「上流仕様（要求仕様/設計物）の品質向上」，「成果物作成の効率向上」の観点で，それぞれの施策について報告する．

1. はじめに

ネットワークの高速・大容量化を実現する通信装置において，その制御の中枢を司るソフトウェアの開発規模が拡大の一途をたどっている．そのような状況の中，市場ニーズに応えるべくスピード開発を実行し，さらに高品質・高生産性を実現することが期待されている．

近年の大規模ソフトウェア開発は，図1に示すような類似機種装置のソフトウェアをベース流用する「流用開発」が主流となっている．本開発手法では，ソースレベルで必要な部分を改造し，必要な部分を流用する形態であるため，全体構造を見ない追加・修正を実施することによる構造の複雑化(構造劣化)が発生する．

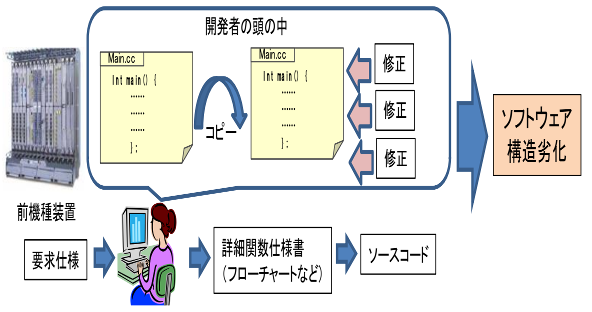

図1．大規模ソフトウェア開発における流用開発

またソフトウェア開発の理想は，いかに上流工程で品質を担保することである．しかしながら，実際のソフトウェア開発は，後工程であるテスト工程依存の開発が実情である．図2にソフトウェア開発の工程毎の開発工数分布図を示す．

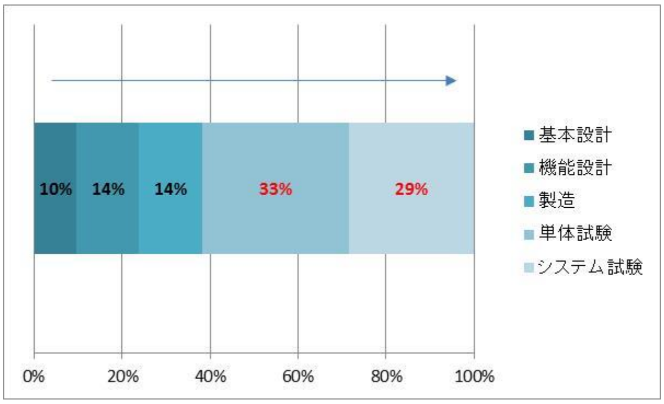

図2．開発工程別工数分布

図2のような工数分布となる背景としては以下がある．

① 技術進歩により，上流工程での仕様確定が難易度が高く，テスト工程による実動作をベースに仕様確定を実施せざるを得ない．
② ①の影響で，ソフトウェアの修正回数が増加となる．

図1及び図2で説明した状況から，ソフトウェア開発力を強化する為には，

■流用開発母体の綿密な解析の実行．解析結果をもとに上流工程における品質確保の実践．
■ソフトウェア変更対応工数の削減．さらに品質向上・効率向上の実現．

本稿では，弊社が取り組んでいるソフトウェアの「高品質」「高生産性」の施策の一例について述べる．

2. 取り組み施策

2.1. 流用開発の品質向上施策

組み込みソフトウェアでは，小規模であった頃の開発文化であるソース･コード中心の開発が未だ残っており，ソフトウェアを流用する際には，ドキュメントがほとんどない状況で開発を推進せざるを得なくなっている．

こういった状況を打破する為，弊社では図3に示すリバースエンジニアリング手法を適用した．
リバースエンジニアリング手法とは，流用ソフトウェアの動作を分析･解析し，その仕様，目的，構成部品，要素技術などを明らかにすることである．

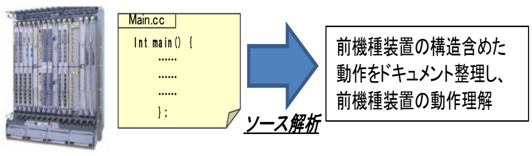

図3. リバースエンジニアリング手法

弊社で実施したリバースエンジニアリング手法の実施内容を以下に整理し，効果について記載する．

[1]UMLモデリングツールの活用

流用装置の大規模ソフトウェアをリバースエンジニアリング手法を用いて解析するに辺り，以下市販ツールを利用して，UMLモデル図作成を実施した．図4に概略図を記載する．

【市販ツール】
製品名：Enterprise Architect（略して EA）
製造元：スパークスシステムズ ジャパン株式会社
（https://www.sparxsystems.jp/inquiry.htm）

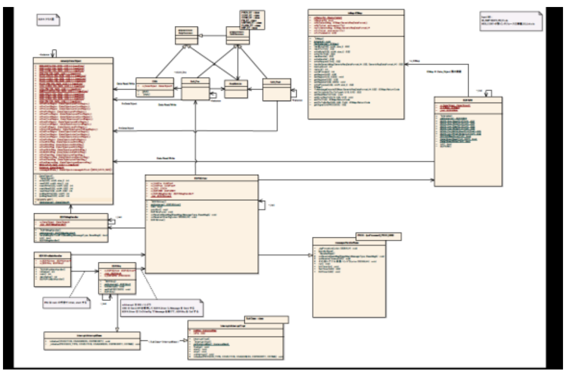

図4. EA ツールを用いた UML モデル図

本 UML ツールの特徴としてソースコードの生成・読み込みの機能がある．ソースコードを取り込むことでクラス図を自動生成することができ，変更を行う場合はクラス図を変更することにより，再度ソースコードとして出力し直すことが可能となる．

本UMLツールを使用したことによるメリット/デメリットを以下に整理する．

【メリット】
① ソフトウェアの全体構成図が理解でき，ソフトウェア構造劣化の防止．
② ソフトウェア解析スピードの向上．

【デメリット】
① 自動生成できる成果物はクラス図のみが対象となっている．
② ソフトウェア全体構造図が理解できる反面，シーケンス制御（例えばハードウェアとのインタフェースなど）を解析するには不向き．

上記デメリットが発生する為，弊社では施策【2】を新たに実施した．

[2]ハードソフトインタフェース部シーケンス図作成

UML モデリングツールで解析不能な箇所においては，図5に示すようなシーケンス図を記載した．

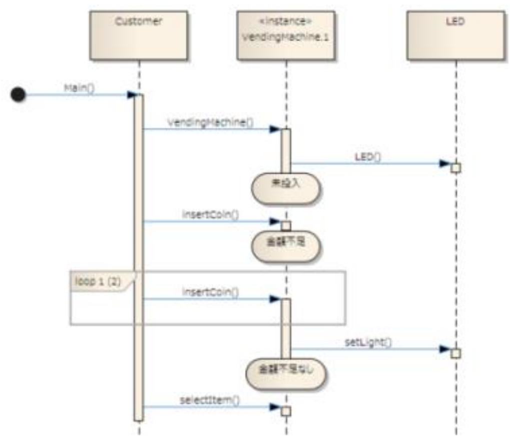

図5. ハードソフトインタフェース部分のシーケンス図

本作業を実施することで，リバースエンジニアリング作業を実施したソフトウェア開発チームが，流用装置のハードウェア制御動作を理解することが可能となった．
このようなメリットを活かして，弊社ではさらに施策【3】を実施した．

【3】ソフトウェア要件であるハード・ソフトインタフェース仕様書のソース自動生成対応

現状の開発では，ハードとソフトにおける分業化が進んでおり，技術領域（回路設計中心のハード開発,プログラム設計中心のソフト開発）の違いから両社の間には大きな溝がある．そのため，ハード/ソフト両者の観点で相互理解を行い，ハード・ソフトのバランスのとれた仕様書作成を行うことが課題である．

施策【1】【2】を実施したことにより，ソフトウェア構造を理解した状態で，ハード・ソフトインタフェース仕様書を作成することが可能となる．
さらに，ハード・ソフトインタフェース仕様書に，ソース自動生成機能を盛り込み，ソフトウェア開発自体の効率化を実現した．図6に従来と本施策による開発プロセスの差異，図7に例として，ハード・ソフトインタフェース仕様書の一部であるデバイス設定仕様をベースにソースを自動生成する機能例を記載する．

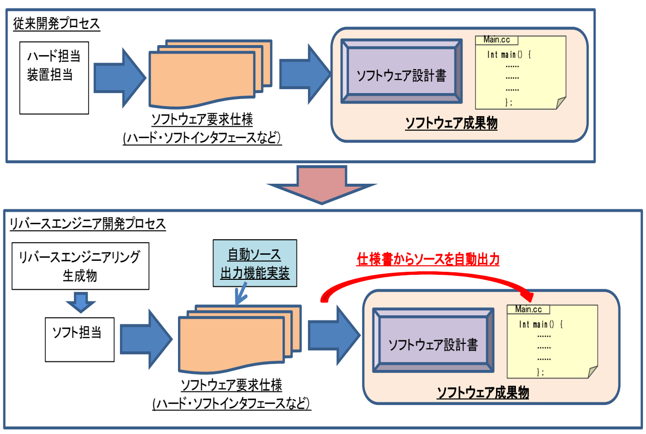

図6．本施策を実施した開発プロセス

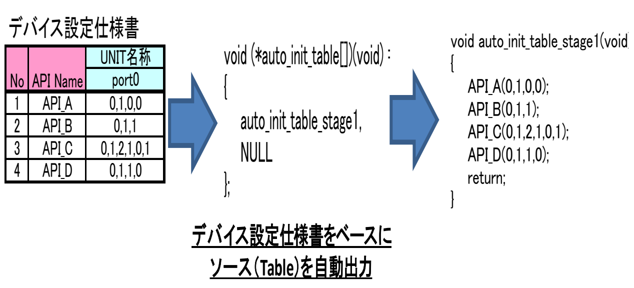

図7．ソフト要求仕様書からソース出力する例

【まとめ】
上記３つの施策を実施し，課題であった流用ソフトウェア開発の品質確保を実現した．

2.2. 成果物作成過程における品質強化／効率化施策

ソフトウェア変更対応工数の削減，テスト工程における品質確保，効率向上を目指して弊社が実施している手法を説明する．弊社では，オープンソースのプロジェクト管理ソフトウェアである Redmine (http://redmine.jp)を用いて，タスク管理/進捗管理/不具合管理を実施している．Redmine の特徴であるカスタマイズの柔軟性を活用することにより，プロジェクトの実態に合わせた運用を行うことが可能となる．Redmine による一元管理の元，施策【1】にて Redmine 自身のカスタマイズによる取り組み.,施策【２】～【４】にて外部ソフトウェア (Jenkins (http://jenkins-ci.org/))との連携による取り組みを紹介する．

【1】工程にあわせたトラッカーカスタマイズ

Redmine はタスク管理を『チケット』という単位で管理を行う．そのチケットの種別を表す『トラッカー』をカスタマイズすることにより，状況に応じた柔軟なタスク管理が可能となる．チケット駆動型開発による仕様設計から評価までの作業管理の一元化を行うことで作業・確認漏れなどの手戻りを防止している．

表 1.Redmine トラッカーの作成例

種別	内容	必須項目
AI管理	調査などの作業管理に使用	「担当者」「開始日」「終了日」「進捗」*1
BD/FD(DD)	設計書作成作業に使用	基本情報(*1)に加えて 「設計書枚数」
MK	コーディング作業に使用	基本情報(*1)
CT/IT	評価項目書作成/実施に使用	基本情報(*1)に加えて 「設計書(項目書)枚数」
レビュー管理	レビュー記録に使用	基本情報(*1)に加えて 「プロセス」「レビュー指摘数」 「レビュー参加人数」「レビュー時間」
変更管理	仕様変更/改善作業管理に使用	基本情報(*1)に加えて 「原因分類」「発生版数」「原因内容」 「混入工程」「対処内容」「確認内容」 「確認版数」「リリース版数」
問題管理	問題管理に使用	変更管理同様

本取り組みにより，評価時だけでなく開発全工程を網羅した品質分析が可能となる．

【2】ソースコードバージョン管理システムとの連携

Redmine には，CVS や Subversion といったバージョン管理システムと連携する機能がある．弊社では，本機能を活用することにより，問題と修正内容を容易に紐付けし，チケット駆動型開発が可能となる．以下にその効果を記載する．

【効果】

① コミット単位の明確化
チケットがないコミットは原則認めない．そのため，チケット単位で作業状況を全て一元管理することが可能となる．

② ソースコードの見える化
変更内容がチケット上で確認することができ，チケット記載内容との対応付け，開発メンバ内の変更内容の共有化が容易となる.

連携イメージ図を図8に記載する.

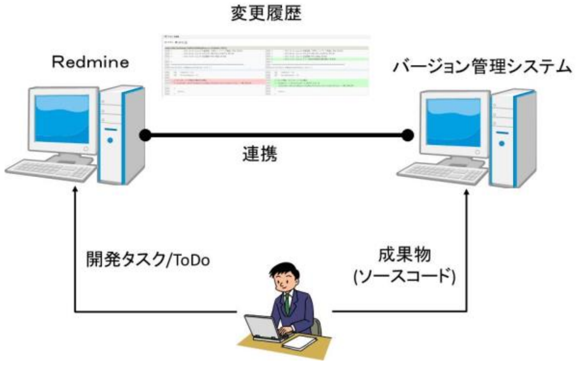

図 8.バージョン管理システムとの連携図

【3】ソースコード構成管理手順(コミット)の統一化

弊社では，前述のバージョン管理システムとの連携も含め，ソースコミット時の手順を統一化することで，コミット時の効率化/オペミス防止/手戻り防止に取り組んでいる. また，静的コード解析もコミット手順の一つとして実施することにより品質確保を行っている. 表 2 に示す「CommitTool」を活用することにより，コミット時の効率化/オペミス防止/手戻り防止を実現している.

名称	CommitTool
説明	ソースコミットに用いるTOOL 実行ファイルとプロジェクト環境ファイルによって構成される
規模	シェルスクリプト　2kstep
実行内容	①メニュー選択 ②文字コード/改行コードチェック ③コンパイル確認 ④静的コード解析 ⑤Redmine番号入力 ⑥ソース差分確認 ⑦コミット ※各箇所でNG時にはエラーとなり、コミット不可

表 2.CommitTool 概要

【4】継続的インテグレーションツール Jenkins との連携

継続的インテグレーションツールである Jenkins とプラグイン導入により Redmine 連携が可能となる.
図 9 に定期品質チェックの図を示す. Jenkins のジョブの定期実行機能を活用し，日単位で静的解析ツールを実行することにより，ソースコードの品質を保証し，誤ったコミットやオペミスによる手戻りを最小限に抑える取り組みを行っている.
また，チェックを行う時間に関しても，通常勤務時間に影響を与えず，PC 稼動負荷を抑えることができる深夜/早朝の時間に実施を行っている.

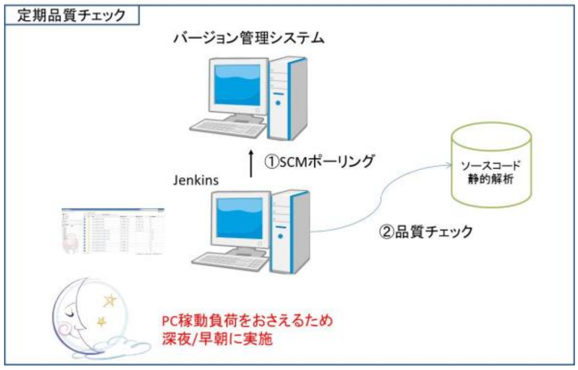

図 9.Jenkis を用いた定期品質チェック

ソフトウェアの正式なビルド作業に関しても，同様に Jenkins ジョブを作成し，管理を行っている.
図 10 にビルド手順の図を示す. Redmine 上からボタンを押すだけで，全ての作業を自動で実行する.

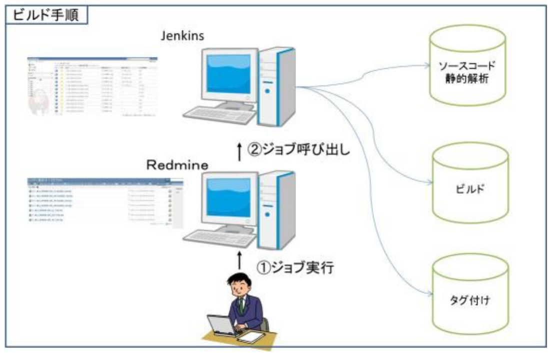

図 10.Jenkis を用いたビルド手順

静的解析～ソースコードのタグ付けまで一連の流れで実行するため，作業の簡略化/オペミスの防止に繋がっている. 実行後，成果物は規定の格納先に格納され，コミットログから変更一覧が自動生成されるようになっている.

3. まとめ

組み込みソフトウェア開発力強化の取り組みとして，「上流仕様(要求仕様/設計物)の品質向上」，「成果物作成の効率向上」という観点での施策をそれぞれ述べた. 施策普及の為に，最も注力している事は，教科書的な施策ではなく開発実態に即した施策を実施することが重要と考えている. 今後も，本施策をさらに強化していく取り組みを実施していきたいと考える.