Panasonic Technical Journal Vol. 56 No. 3 Oct. 2010

白物家電ソフトウェアにおけるリファクタリングの取り組み

Software Refactoring Activities in Home Appliances Software Development

野 田 桂 子* 山 崎 直 紀*
Keiko Noda Naoki Yamasaki

コード行数，関数複雑度などのコードメトリクスおよび構造分析結果を活用し，複雑化したソフトウェア構造の課題を定量化するとともに，構造の可視化分析に基づいたリファクタリング方法を形式化することにより，効率よくリファクタリングを行うことができた。

We quantified the complicated software structure's issues by using the results of structure analysis and metrics such as lines of code and cyclomatic complexity. And we explicitly define our refactoring procedures based on software structure visualization analysis. By using these procedures, we were able to perform refactoring efficiently.

1. リファクタリングの必要性と難しさ

近年，白物家電も，高機能化，多機能化に伴い，ソフトウェア規模が年々増大し，その構造も複雑になってきた。

白物家電ソフトウェアの場合，アセンブラで記述していた小規模な頃の資産を継承している場合が多く，以下のような特徴があり，仕様変更時の実装漏れやミスにつながりやすかった。

- 外部変数による依存関係が多い
- 関数，ファイル，ディレクトリが，適切な単位で分離されていないことが多い

このため，仕様変更に柔軟に対応でき，再利用性の高いソフトウェア構造への再設計（以下，リファクタリングと記す）を行うことが急務となってきた。

しかし，年々開発期間が短縮される中，量産開発の中でのリファクタリング実施は難しい。今回は，リファクタリングプロセスを明確化し，このプロセスに基づいてエンジニアリングを担当する当部署と開発部署が一体推進することで，品質の低下を招くことなく，リファクタリングを実施することができた。

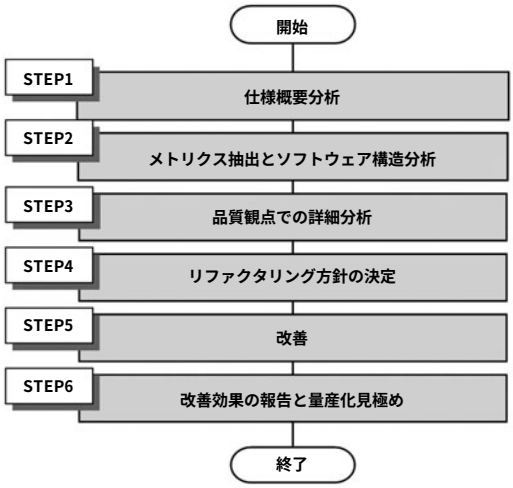
2. リファクタリングプロセス

リファクタリングを実現するために，**第1図**に示したプロセスを白物家電共通で手順化した。

〔1〕STEP1：仕様概要分析

仕様書の分析を行い，UML（Unified Modeling Language）のユースケース図を使用し，「機能仕様の図式化」を行う。

〔2〕STEP2：メトリクス抽出とソフトウェア構造分析



第1図 リファクタリングプロセス
Fig. 1 Software refactoring process

解析ツールを用いて，以下のようなメトリクス抽出を行い，課題を定量化し，これらを元にソフトウェア構造を分析する。

- コード行数，複雑度など
- 外部変数の数や参照関係
- 関数の参照関係，関数構造図，関数階層など

〔3〕STEP3：品質観点での詳細分析

ソースコードの詳細な分析を行って抽出した課題項目について，ソフトウェア品質特性（ISO/IEC-9126）[1]の観点からマトリクス化し，課題項目の品質への影響度を定性的に分析する。

〔4〕STEP4：リファクタリング方針の決定

STEP2とSTEP3の分析結果を総合的に評価し，「リファクタリングの優先度と目標設定」を行う。課題の大きさと，量産実績のあるソフトウェアを変更する品質リスクとを比較して，優先度や範囲を決め，改善の目標値を設定する。

開発部署の技術責任者に，可視化した課題の認識，方針の承認を得るための報告を実施する。

＊ ホームアプライアンス社 技術本部
Corporate Engineering Div., Home Appliances Company

〔5〕STEP5：改善

STEP4で決定したリファクタリング方針に基づき，改善を行い，リファクタリング前のソフトウェアから品質低下がないことの検証を行う。

〔6〕STEP6：改善効果の報告と量産化見極め

改善による品質検証結果，STEP5の改善の効果などについて報告を行い，開発部署の技術責任者から量産化への見極めを得る。

3. 改善手順

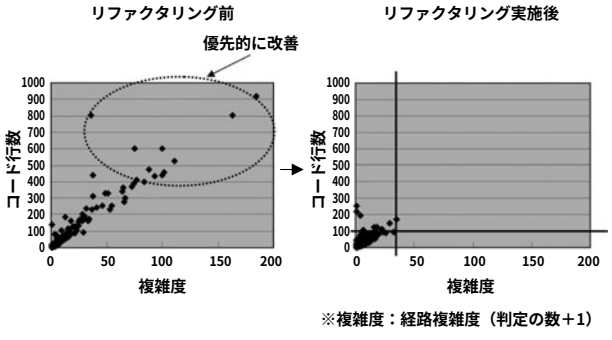
STEP5での改善は，決定した方針に基づいて，2段階の手順で行う。現行のソフトウェア構造の枠組み内で行うコードの改善と，ソフトウェア構造の改善である。

3.1 コードの改善

コードメトリクスのうち，特に保守性の観点から，関数単位のコード行数と複雑度に注目した改善を行う。コード行数は100行，複雑度は30を目標値として，たとえば，**第2図**左のリファクタリング前の散布図の場合，楕円部分の関数のコード改善を優先して行った。

課題の多い関数は，モジュール強度[2)]（モジュール内の関連性の強さ）の低いものが多いため，関連の強い機能単位に関数分割する。さらに，同じコードパターンの繰り返しであるクローンは，共通部分を1つの関数として集約する。このようなコード改善の中，パターン化できるものは，今後に備えコーディング規約に反映させる。

上記の改善により，ある事例では，コード行数や複雑度は，約1/4に改善され（**第2図**右），コードクローンなどの集約により，コードのメモリー使用量を約2割削減できた。



第2図 関数ごとのコード行数と複雑度の改善

Fig. 2 Improvement of lines of code and cyclomatic complexity

3.2 構造の改善

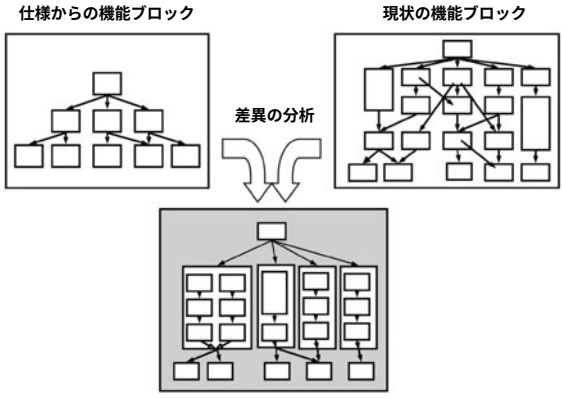
コードの改善実施後，構造の改善を行う。

STEP1で図式化した機能仕様から設計される構造と，現状の関数やファイル間の相互依存関係から可視化される構造の差異を分析し，適切な機能単位にソフトウェア構造を再構築する（**第3図**）。

ディレクトリやファイルの分割単位が，外部変数や関数の参照関係から見ると妥当でない場合も多いので，ディレクトリ構造を見直す。相互依存関係が密なものは，1つのディレクトリにパッキングする。

また，仕様変更の多いものや機種依存のものは，ディレクトリで独立させ，新たにOSを搭載する場合は，機能単位からタスク単位のディレクトリ構造に再編する。

さらに，おのおののディレクトリについて，修正などの影響を限定できるように，外部変数や関数の公開範囲をより狭くするとともに，相互依存関係を整理し，ディレクトリ間の独立性を高めるようにした。



第3図 構造の改善

Fig. 3 Software structure refactoring

4. 今後の展開

今回，従来，暗黙で進められていたリファクタリング方法を形式化し，優先度や範囲を見極めることにより，品質の低下を招くことなく，リファクタリングを実施できた。

今後，改善したコードや構造が容易に崩れないようにするためのルール整備や，より上流の設計で作りこむ枠組みを実現していく予定である。

参考文献

1）ISO/IEC9126-1:2001, Software engineering - Product quality - Part1: Quality model (2001).

2）Glenford J. Myers（国友義久　他 訳）: ソフトウェアの複合／構造化設計 （近代科学社） p.39 (1979).