ソフトウエア大規模再利用の為の共通フレームワーク構築

Common Framework Construction for The Software Large Scale Reusing

森　勇仁*　　榎並　崇史*
Yujin MORI　　Takashi ENAMI

要　　旨

エンジン制御ソフトウエア開発において，UML（Unified Modeling Language）を用いたオブジェクト指向により，品質向上・開発効率向上を行ってきたが近年の開発製品数や開発機能の増加に伴い，開発工数の増加が課題となってきた.

そこで，従来製品ごとにソフトウエアを流用していたソフトウエア開発プロセスから，制御の本質に着目した共通フレームワークを構築し，その共通フレームワークと製品差分との組み合わせにより多製品同時開発を行う方法に切り替えた．また，モデルとソースコードだけでなく，テスト環境，製品展開マニュアルも含めてコア資産化し，それを再利用することで多製品への展開を行った.

その結果，開発プロセスの変更と，共通フレームワークの構築により，更なる品質・開発効率の向上を実現した.

ABSTRACT

To improve the software quality and the development efficiency, we have used Object Oriented Method with UML (Unified Modeling Language) to develop the engine control software. However, the development man-hour has increased because of the increase of the number of products and functions that should develop.

Then, we changed the approach to develop the engine control software. Previously, we developed engine control software of a product by reusing software of other products. Currently, we develop the common framework that focus on the essence of the control, and develop software for various products by combining the common framework and differences according to the products. And we also make core assets including not only develop models and source code, but also the test environment and the manuals of development using those deliverables. In this way, we can develop software for various products by reusing those core assets.

By changing the development process, and having constructed the common framework, the software quality and the development efficiency are improved still more.

* MFP事業本部　第二設計センター
2nd Designing Center, MFP Business Group

1. 背景，目的

複写機やプリンタのエンジンを制御するソフトウエアの開発において，オブジェクト指向によるモデリング設計主体のソフトウエア開発を行ってきた．しかし，近年同時並行で開発する製品数が増えた為，更なる生産性向上と品質向上が課題となった．

本取り組みでは，複数製品で利用可能な共通フレームワークを基本とした，ソフトウエア開発のコア資産を構築し，それを大規模再利用することにより，多製品並行開発における生産性向上と品質向上を目指す．

2. 大規模再利用の為の共通フレームワーク構築

従来から，ソフトウエアの開発工数削減の為に，ソフトウエアを再利用することが試みられてきた．しかし，小規模のライブラリなどは再利用できても，ソフトウエアの規模が大きくなると，再利用は上手くできなかった．

その理由としては，従来は，ある製品開発が終了し，次の製品開発が始まる段階で，前製品との差分を調査し，ソフトウエアの変更点を検討していた．前製品との差分が小さければ変更は部分的で済むが，差分が大きくなると変更個所が多くなる．変更箇所が多くなるに従い，ソフトウエアの品質も低下し，メンテナンスも難しくなってくる為，最終的には全部作り直した方が良い，ということになる．

本取り組みでは，ソフトウエアの大規模再利用を行う為，エンジン制御に共通な本質的機能とは何かの分析から行った．それにより，製品ごとの制御方式の違い，ハードウエアやレイアウトの違いに影響しない，多製品展開可能な共通フレームワークを構築した．共通フレームワークを再利用し，レイアウト情報などの製品固有の差分（製品変動部）を開発し，共通部と変動部を組み合わせることで，製品開発を行うようにした．

共通フレームワークを構築する為に，ソフトウエアの開発方法をどのように変えたかを，エンジン制御ソフトウエアの一部分である，スキャナのDF（Document Feeder）制御ソフトウエアの開発で，説明する．

最初に開発する製品（1製品目）のDFはシートスルー方式のDFを搭載し，次に開発する製品（2製品目）のDFはフラットベッド方式のDFを搭載する場合の開発例を示す．

シートスルー方式（Fig.1）：スキャナキャリッジ位置は固定で，原稿を搬送させながら読み取る．長い原稿も読み取り可能で，装置をコンパクトにできる．

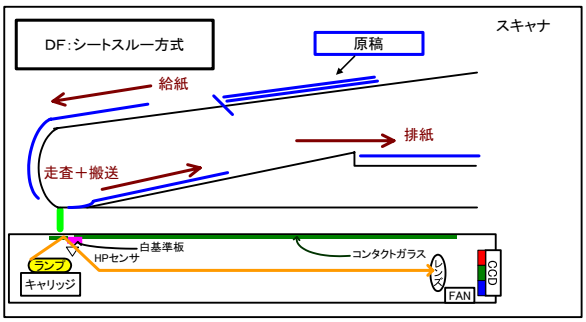


Fig.1　Sheet-through document feeder.

フラットベッド方式（Fig.2）：原稿を読み取り台（コンタクトガラス）上に搬送させ，原稿は動かさずスキャナキャリッジ位置を動かしながら読み取る．読み取り原稿の最大サイズに応じて装置が大きくなる．

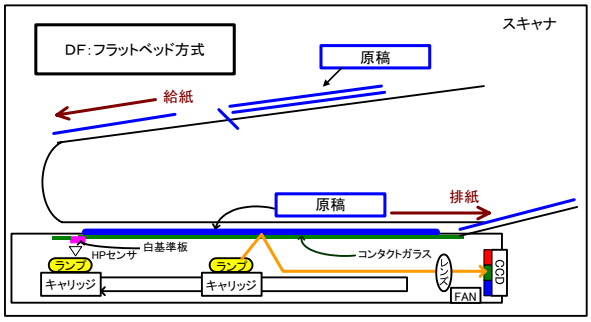


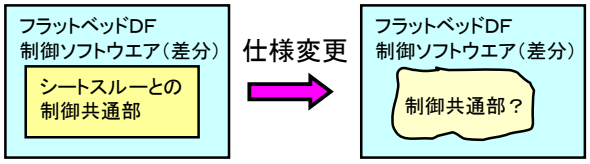
Fig.2　Flat bed document feeder.

従来の開発方法では，

① 1製品目：シートスルー方式のDFを制御するソフトウエアを，制御仕様通りに動作するように開発．（この時点では，共通部・変動部の切り分けは意識していない．）

② 2製品目：フラットベッド方式のDFの制御と，シートスルー方式のDF制御との共通部・変更部を検討．共通部を流用し，フラットベッド方式の制御ソフトウエアを開発する．

この方法では，1製品目と2製品目の制御仕様を基に，共通部／変更部の切り分けを行う為，共通部の作りや範囲は，個々の製品の制御仕様によって決まる．開発の初期には共通部としていたものが，その後製品の仕様変更により，共通部を変更することもありえる．共通部が製品仕様の影響を受けると，共通部の範囲が小さくなり，共通部/変動部の切り分けが曖昧になる．



・共通部は，個々の製品仕様によって決まる．
・共通部／変動部の分離が曖昧になる．

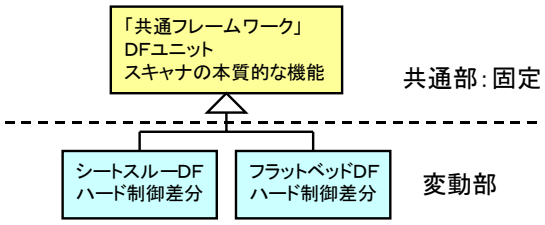
Fig.3 Example of remaking software.

本取り組みでの開発方法では，

① 本質的な共通の機能を分析する．シートスルー方式やフラットベッド方式など，DFの方式は色々あるが，そもそもDFとは何を行うものかの分析から行う．結果，DFの方式やハードウエアの違いに影響しない，共通の機能を見出す．本質的な共通の機能を実現するソフトウエアの設計を行い，共通フレームワークを構築する．

② 1製品目：共通フレームワーク（共通部）を使用し，シートスルー方式のDFの製品差分（変動部）を開発．共通部と変動部の組み合わせにより，製品開発を行う．

③ 2製品目：フラットベッド方式のDFも1製品目と同様に製品差分を開発し，共通部と変動部の組み合わせにより，製品開発を行う．



・共通部は個々の製品仕様に影響されなく，固定．
・共通部／変動部の分離が明確

Fig.4 Example of reusing the common framework.

この方法では，共通の機能を分析し，共通フレームワークを構築している為，共通部は，個々の製品仕様によって影響されること無く，固定にすることができる．共通部が固定である為，共通部と製品変動部との分離が明確である．従来のように，制御仕様の変更により，共通部に変更が加わり，共通部/変動部の切り分けが曖昧になる事も無い．

今回は，DF制御ソフトウエアの開発を例に説明したが，本取り組みでは，Fig.5に示すように共通フレームワーク構築をエンジン制御全体で行うことで，共通部を増やし，ソフトウエアの大規模再利用を実現した．

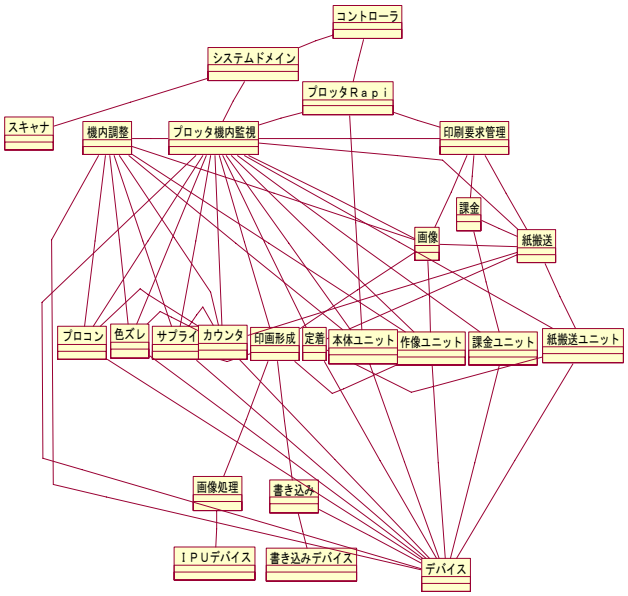


Fig.5 Class diagram of common framework.

3. 多製品展開の為のコア資産の構築

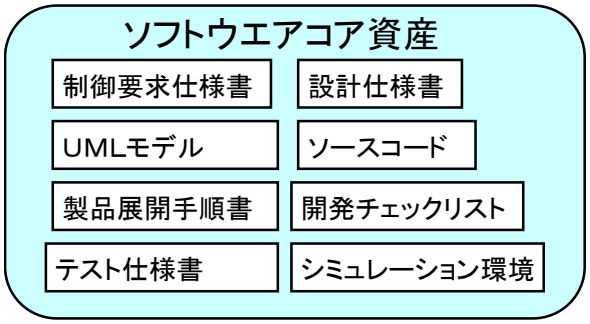


Fig.6　Contents of core assets.

従来の開発手法では，ソフトウエアの共通部の多くは製品の仕様に依存していた．その為，ソフトウエア共通部の製品展開の仕方や，運用の方法が製品ごとにばらばらであり，また，ソフトウエアの品質や開発効率は，開発者のスキルに依存する部分が多かった．

本取り組みでは，製品の仕様に影響を受けない共通フレームワークを構築したことにより，その共通フレームワークを製品展開する仕方や，運用方法に関しても共通化し手順化することが可能となった．具体的には，共通フレームワークの設計仕様書，UMLモデルとソースコードだけではなく，制御要求仕様書，製品開発に展開する為の手順書，開発チェックリスト，またテスト仕様書や実機の動作をPC上でシミュレーションする環境も併せてコア資産化し，製品開発で再利用を行った．その結果，対象ドメインに精通していない開発者でも，従来に比べ，より短期間で品質の高いソフトウエア開発をすることができるようになった．

実際に製品開発に展開した結果をフィードバックし，洗練化しつづけることで，コア資産の品質をより向上させることができる．

本取り組みでは，コア資産構築直後に複数の製品に展開するのではなく，まずは先行開発製品（1製品目）に展開し，十分品質確保した後，他の多くの製品開発に展開した．

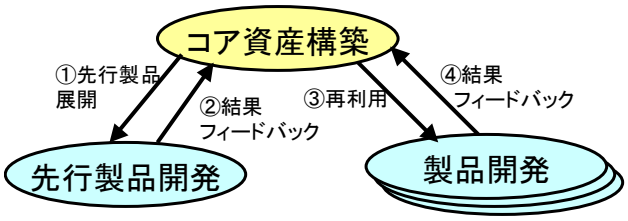


Fig.7　Core assets construction and various products development.

4. 共通フレームワーク構築，製品展開の為の開発体制

共通フレームワークを構築し，それを効率的に製品展開する為に，開発体制の見直しも行った．

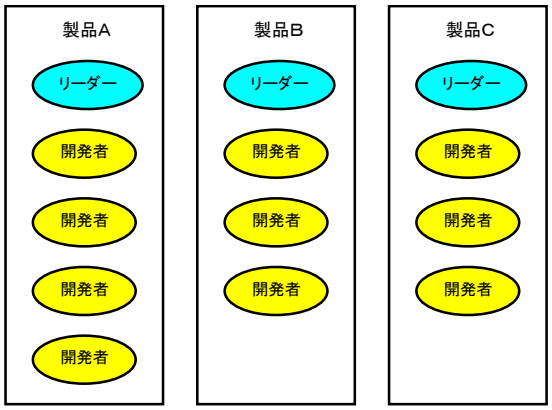


Fig.8　System of development of each product.

従来の開発体制では，Fig.8のように製品ごとに開発者を割り振っていた．この体制では開発製品数が少ない場合には効率的だが，開発製品数が多くなると，同じようなソフトウエアを複数の開発者が作成することになり，非効率である．また開発者は担当製品の対応に専念し，他製品の情報を知らない為，先に述べた共通フレームワークの構築ができない．またある製品で行った改善や不具合対策を他の製品に展開することが難しく，品質向上の点でも課題があった．

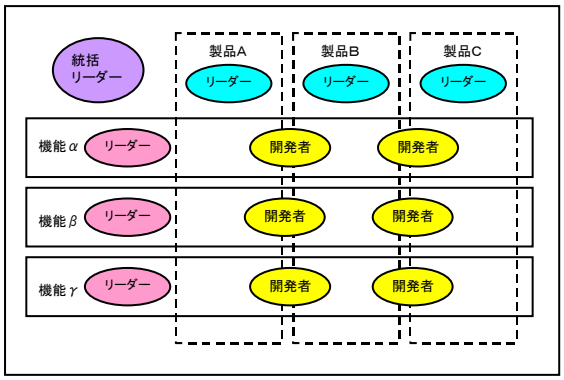


Fig.9　System of development of each function.

本取り組みでは，Fig.9のように機能ごとの開発チームが，複数製品の開発を担当する体制とした．

これにより，開発者は製品の共通部や変動部を把握しやすくなり，ソフトウエアの重複開発も防げる．またソフトウエアの改善や不具合対策の水平展開も容易となった．

5. ソフトウエア開発の上流工程と設計工程の分離

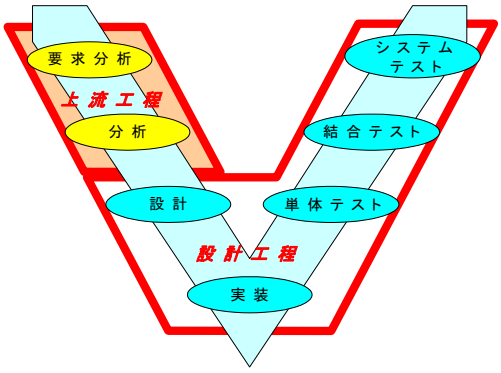


Fig.10　Separation of software development process.

従来の全ての開発者が一連の工程（要求分析からシステムテストまで）を行う開発プロセスでは，製品開発の状況により設計工程以降に工数が取られることが多く，上流工程がおろそかになり，結果として共通フレームワーク構築に支障をきたす傾向にあった．

この問題を解決する為に，機能開発チームを要求分析・分析を行う上流工程担当者と，設計・実装・テストを行う設計工程担当者に分離し，開発を行った．

まず，上流工程担当者が，ある機能に対しソフトウエアの分析まで行う（Fig.11 分析1）．その分析結果を受け，設計工程担当者は，同じ機能に対しソフトウエア設計から実装，テストまでを行う（Fig.11 実装1）．設計工程担当者が実装1を行っている間，上流工程担当者は別の機能に対してソフトウエアの分析まで行う（Fig.11 分析2）．

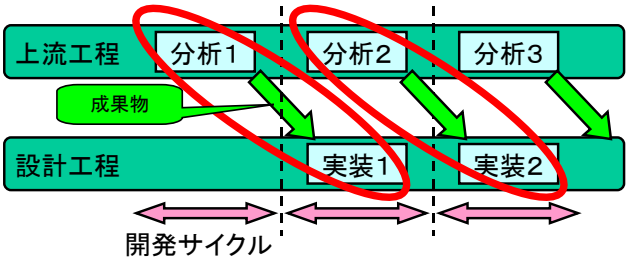


Fig.11　The upstream process charge and design process charge.

このように，上流工程と設計工程を分離する事で，製品開発に影響される事無く，共通フレームワーク構築が加速した．更には，複数人で成果物を共有する為，1人で一連の工程を進めていたときの曖昧さが無くなり，品質向上にも繋がった．また上流工程と設計工程を並行に進めることで，開発期間短縮にも繋がった．

6. 結果

6-1　開発工数

Fig.12において，製品開発初期の開発工数の改善例を示す．

試作機械を入手後，その機械で立ち上げから1枚印刷などの基本動作を確認する実機確認において，前製品で品質が保証されているコア資産を再利用した結果，開発手順や評価内容が明確となり，障害が少ない為，工数を従来の1/3に抑えることができた．

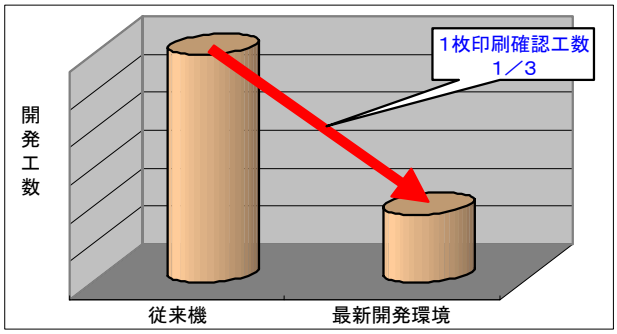


Fig.12　Development man-hour.

またFig.13，14において，コア資産を多製品に展開することで，開発工数全体が低減した例を示す．

1製品目の開発では，コア資産の構築により工数が多くなっている．しかし2製品目以降は，コア資産構築工数が不要となり，コア資産を再利用した結果，全ての製品開発において開発工数が1/3に削減できた．(Fig.13)

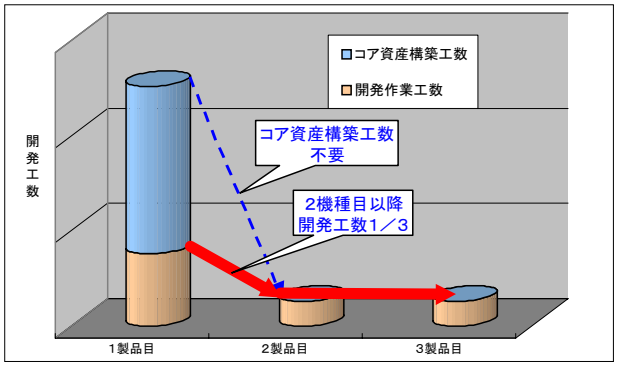


Fig.13　Development man-hour.

このように，コア資産構築にかかる初期投資（工数）は必要であるが，数製品に再利用する事で回収できる．また，製品への再利用が増えるほど削減効果も増えていくことになる．(Fig.14)

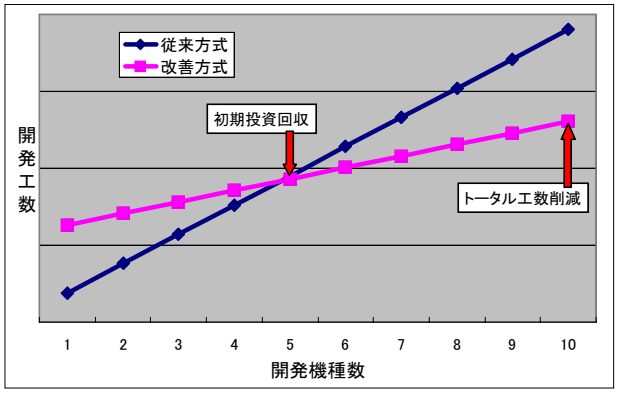


Fig.14　Development man-hour reduction by reusing core assets.

6-2　ソフトウエア品質

Fig.15にコア資産を構築したときの製品，及びそれを再利用した製品での，開発時に発生した障害の量の推移を示す．障害が少ないほど，品質が良いといえる．

グラフ上①のフレームワーク構築では，上流工程に工数をかけ，障害総合指数は大幅に低減した．次にグラフ上②の再利用第1世代では，コア資産の構築を行った事で品質が更に向上した．グラフ上③の再利用第2世代では，品質が安定しているコア資産を多製品展開した事により，全ての製品において品質向上を達成する結果が得られた．

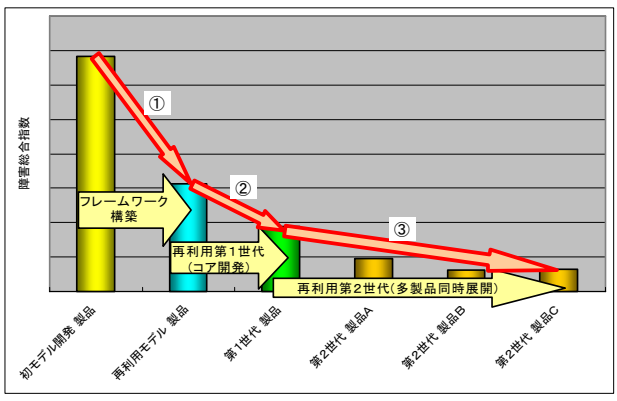


Fig.15　Transition of software quality.

7. 結論

半期ごとのソフトウエア開発工数の推移をFig.16に示す.

グラフの左半分①の時期は，開発製品数の増加，コア資産構築により工数は増加していた．次にグラフの右半分②の時期は，開発製品数は増加しているが，コア資産の再利用により，工数は減少傾向を示している.

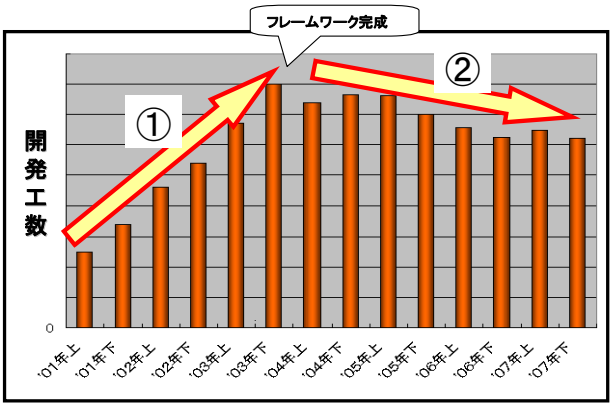


Fig.16　Transition of development man-hour.

共通部と変動部の組み合わせによる製品開発プロセスに変更すること．また，品質を確保したコア資産を多製品に展開すること．これら2点の考え方によりソフトウエア大規模再利用を可能とし，多製品並行開発においても，品質を向上させ，更に開発効率をも向上させることが実証できた.