モデルベース開発ツールを活用した際のコストの

効果検証

実施報告書

2013 年 2 月

独立行政法人情報処理推進機構
Information-technology Promotion Agency, Japan

はじめに

IPA/SEC では、ソフトウェア品質説明力を強化すべく様々な観点からの検討を実施してきました。その一環として、ソフトウェア品質を説明するための手法等について具体的な実施方法、そのための作業量、実施にあたっての課題等を整理し、実際にソフトウェア品質を説明する際の参考とできるようにするために、公募により、観点ごとに分けられた実験を別々に実施しました。本書は、それらの結果を、実験ごとにまとめた報告書のうちの１つです。

本報告書の実験は、「2011 年度システムエンジニアリング実践拠点事業」として、東芝情報システム株式会社に委託し実施しました。

「モデルベース開発ツールを活用した際のコストの効果検証」
【実施報告書】

独立行政法人情報処理推進機構

目次

1. 実験の概要 1
2. モデルベース開発の概要 3
2.1 ソフトウェア開発の現状と課題 3
2.1.1 設計品質の向上について 4
2.1.2 開発コストの削減について 6
2.1.3 開発期間の短縮について 7
2.1.4 課題への対応 8
2.2 モデルベース開発とは 9
2.2.1 システム設計プロセス 10
2.2.2 ソフトウェア設計プロセス 11
2.2.3 ソフトウェア作成プロセス 11
2.2.4 ソフトウェア検証プロセス 11
2.2.5 適合評価プロセス 11
2.3 モデルベース開発の予想される利点 12
2.3.1 工数の削減 12
2.3.2 コーディングの短縮 12
2.3.3 レビュー時間の短縮 13
2.3.4 手戻りの軽減 13
3. 実験の前提 15
3.1 実験の目的 15
3.2 実験の範囲と手法 15
3.3 実験対象の詳細 15
3.3.1 開発対象システムの概要 15
3.3.2 開発対象システムの仕様 17
3.3.3 レガシー開発環境と手順 19
3.3.4 モデルベース開発環境と手順 20
3.4 実験項目 22
4. 適用レベルの定義と工数削減効果の想定 23
4.1 適用レベルの定義 23
4.2 各適用レベルの工数削減効果の想定 23
4.2.1 適用レベル 0 23
4.2.2 適用レベル 1 23
4.2.3 適用レベル 2 26

4.2.4 適用レベル 3 27
4.3 実験で実施した適用レベル 28
5. 品質指標 29
5.1 システムプロファイリングの決定 29
5.2 プロジェクトプロファイリングによる補正値の算出 30
5.3 各品質指標の目標値算出 31
5.3.1 設計レビュー作業実施率の目標値 31
5.3.2 コードレビュー作業実施率の目標値 32
5.3.3 テスト密度の目標値 34
5.4 実験対象開発における品質指標の目標値のまとめ 35
6. 実験の実施 36
6.1 レガシー開発の実験データ 36
6.1.1 設計レビューの品質 37
6.1.2 コードレビューの品質 37
6.1.3 検証の品質 37
6.2 モデルベース開発の実験データ 38
6.3 実験結果 40
7. 考察 42
7.1 各実験項目についての考察 42
7.1.1 工数の削減 42
7.1.2 コーディングの短縮 43
7.1.3 レビュー時間の短縮 43
7.1.4 手戻りの軽減 45
7.1.5 モデルベース開発初期導入に掛かる工数 46
7.2 適用レベルの妥当性の証明 47
7.2.1 フロントローディングによる効果について 47
7.2.2 ソフトウェア検証プロセスへのモデルベース開発導入について 48
7.2.3 コード自動生成について 48
7.2.4 レビュー工数について 48
7.3 適用レベルの工数削減率 48
7.3.1 適用レベル 3 48
7.3.2 適用レベル 2 49
7.3.3 適用レベル 1 49
7.3.4 適用レベル 0 50
7.3.5 適用レベルの全体工数削減率 51
7.4 まとめ 51

8. モデルベース開発導入による工数以外の効果 52
8.1 モデル作成による保守性の強化 52
8.2 開発における管理作業の軽減 52
8.3 開発成果物の均一化 52
9. モデルベース開発定着後の効果 52
9.1 プラントモデル再利用における効果 53
9.2 モデル再利用における効果 53
10. 参考文献 54

図目次

図 1 実験の流れ 2
図 2 組込みソフトウェア開発の課題 3
図 3 製品における不具合の要因 4
図 4 ソフトウェア不具合発見プロセス別件数比率と発生プロセスの内訳 5
図 5 製品開発における開発費の割合(2008 年) 6
図 6 製品開発における開発費の割合(2010 年) 6
図 7 新規開発行数(2008 年) 7
図 8 新規開発行数(2010 年) 8
図 9 レガシー開発 9
図 10 モデルベース開発 10
図 11 工数の削減 12
図 12 コーディングの短縮 12
図 13 レビュー時間の短縮 13
図 14 手戻りの軽減 14
図 15 ボンディングシステムの構成図 16
図 16 ボンディング装置用制御コントローラの構図 17
図 17 ボンディング装置用制御コントローラのシーケンス 17
図 18 ボンディング装置用制御コントローラのシステム構成図 18
図 19 システム構成I/O 19
図 20 適用レベル 1 システム設計プロセス適用例 24
図 21 適用レベル 1 ソフトウェア設計プロセス適用例 25
図 22 適用レベル 1 ソフトウェア検証プロセス適用例 26
図 23 適用レベル 2 適用例 27
図 24 適用レベル 3 適用例 27

図 25 システムタイプ一覧 29
図 26 SCPチェック手順 30
図 27 プロジェクトプロファイルチェック 30
図 28 設計レビュー作業実施率の指標 31
図 29 コードレビュー作業実施率の指標 33
図 30 テスト密度の指標 34
図 31 モデルベース開発とレガシー開発の工数 40
図 32 モデルベース開発とレガシー開発の工数（再掲） 42
図 33 コーディング短縮による効果 43
図 34 レビュー時間の短縮による効果 44
図 35 手戻りの軽減による効果 45
図 36 モデルベース開発定着後のプロセス 53

表目次

表 1 適用レベルの定義 2
表 2 実験の範囲 15
表 3 レガシー開発概要 19
表 4 プロセス毎のレガシー開発作業 20
表 5 モデルベース開発概要 21
表 6 プロセス毎のモデルベース開発作業 21
表 7 適用レベル 23
表 8 実験対象開発における指標 35
表 9 レガシー開発のプロセス別工数 36
表 10 レガシー開発のレビュー時間と修正時間 36
表 11 レガシー開発の単体検証 36
表 12 レガシー開発の結合検証 36
表 13 モデルベース開発のプロセス別工数と導入工数 38
表 14 モデルベース開発のレビュー時間と修正時間 38
表 15 モデルベース開発のシステム設計での検証 39
表 16 モデルベース開発のソフトウェア設計での検証 39
表 17 モデルベース開発のソフトウェア検証での検証 39
表 18 適用レベル 3 の評価 40
表 19 全体工数削減率 43
表 20 レビュー総工数 44
表 21 教育計画 46

表 22 適用レベル 3 の工数削減率.....49
表 23 適用レベル 2 の工数削減率.....49
表 24 適用レベル 1 の工数削減率.....50
表 25 適用レベル 0 の工数削減率.....51
表 26 適用レベルの全体工数削減率.....51

用語定義

用語	解説
KStep	コードの行数の単位。1 行 1Step。1000Step = = 1KStep。
MATLAB	米国 MathWorks 社の製品で、行列計算及びグラフィックスに優れたソフトウェア。世界的に広く使われているツールである。
Simulink	MATLAB の演算機能を活用して動作する、モデル化・シミュレーション・解析を行うツール。
V 字プロセス	ソフトウェア開発工程。ソフトウェア開発中で実施する内容とその成果物を規定している。
アンローダー	ボンディング基板をレールから降ろす装置。
ウエハカメラ	ピックアップステージ周りの位置確認用カメラ。
基板カメラ	ボンディング基板周りの位置確認用カメラ。
検証プロセス	V 字プロセスのソフトウェア検証プロセス、適合評価プロセスの 2 つのプロセスを指す。
構造レイヤ	制御ロジックの構造を機能毎に切り分けて表現する階層。
スタブ	ソフトウェアテスト時において、下位モジュールの代用として用意する仮のモジュール。
ステッピングモータ	パルスに同期して動作する同期電動機。正確な位置決めが実現できるモータである。
設計プロセス	V 字プロセスのシステム設計プロセス、ソフトウェア設計プロセスの 2 つのプロセスを指す。
ダイトランスファ	ピックアップステージからプリサイサステージへチップを運ぶヘッド。
チップ	集積回路。特定の複雑な機能を果たすための電子部品。
突き上げ軸	ピックアップステージからチップを持ち上げる軸。
データフローレイヤ	制御ロジックを表現する階層。
トップレイヤ	モデルの入出力を記述する階層。
ヒューマンエラー	人為的に混在した失敗や誤りによるエラー。
ピックアップステージ	ウエハからチップを取り外すステージ。
プラントモデル	ソフトウェア開発時に利用する、制御対象のハードの動作をモデルで実現したもの。
プリサイサカメラ	プリサイサステージ周りの位置確認用カメラ。
プリサイサステージ	ピックアップステージから運んだチップが置かれるステージ。

ボンディング	プリサイサステージからボンディング基板へチップを運び、ボンディング基板にチップを貼り付けるヘッド。
ボンディング基板	チップを貼りつける基板。
ボンディングシステム	チップと LSI パッケージの端子を接続するシステムのこと。
モデル	MATLAB/Simulink などモデルベース開発で使用されるツールで扱う抽象的な機能ブロックの集まり。
モデル IP	特定の機能を持つモデル。
モデルベース開発	モデルを使用し V 字プロセスのフロントローディングを行う開発方法。
モーションコントローラ	各種モータの位置制御をする装置のこと。
レガシー開発	仕様書を作成し、ハンドコーディングでソフトウェア開発を行う従来の開発方法。
ローダ	ボンディング基板をレールに乗せる装置。
ローダマガジン供給軸	ボンディング基板を運ぶ軸。

１．実験の概要

本実験は、品質説明力を強化する手法として、モデルベース開発に注目し、同一の製品について、モデルベース開発を採用した場合と、採用しない従来の開発方法の場合とを比較し、モデルベース開発の導入効果とそれに伴う負荷を評価した実験である。実験概要を以下に説明する。詳細については後述する。

【対象の対象】

本実験は、半導体製造装置で使用される制御コントローラの開発作業を対象として実施する。

【説明力を強化したい品質】

従来のレガシー開発にてV字プロセスを適用した場合、V字左の設計プロセスやV字底の実装プロセスで混入した不具合はV字右の検証プロセスで検出されることになり、設計プロセスや実装プロセスへの大幅な手戻りが発生する。

これに対して、モデルベース開発ツールを同プロセスに適用した場合、設計プロセスで混入した不具合は同プロセス内でのシミュレーションによって検出が可能になり、検証プロセスで検出されたときのような手戻り作業で発生する工数の削減が可能となるとともに、不具合の検出漏れの可能性を抑えることもできる。また、モデルベース開発ツールを適用することにより、ソースコードの自動生成が可能となるため、本来実装プロセスで行うコーディング作業が不要となり、人的なコーディングバグの混入がなくなり、全体としての品質の向上となることが広く知られている。このような効果により、モデルベース開発を採用することによって、製品の品質説明力が強化される。

【実験方法】

半導体製造装置内のボンディング制御コントローラ機能の開発を対象に、設計、実装の各プロセスにて混入した不具合を、検証プロセスにて検出した際の修正が必要となるプロセスを明確にするとともに、その修正に掛かった工数をプロセスごとに算出し、レガシー開発とモデルベース開発それぞれでの最終的なプロセス別の生産性を算出し比較する。また、上記実験を対象として、従来のレガシー開発をモデルベース開発に移行する際のモデルベース開発教育工数、設計（見直し）工数を含めた初期工数（差分工数）を算出し、導入時の全体工数に対しての増加工数を算出する。

【実験の目的】

あらかじめ定義されているボンディング制御コントローラ機能の要求仕様を基に従来のC言語によるレガシー開発にて設計、実装、検証プロセスを実施した場合とモデルベース開発ツールを適用した開発にて設計、実装、検証プロセスを実施した場合の生産性を算出し、後者のツールを活用することの効果を検証する。また、モデルベース開発教育、及び上記のモデルベース開発ツールによる再設計（見直し）を実施し、開発手法ごとの手戻り作業の削減

コストを含めた全体工数の比較を行い、モデルベース開発の効果を検証する。

【適用レベル】

モデルベース開発を適用するプロセス（工程）の数を基準に適用レベルを設定し、適用レベル毎に、作業コストの削減効果による評価を行う。

表 1 適用レベルの定義

適用レベル	定義
L0	モデルベース開発非導入
L1	モデルベース開発導入（1 プロセス以上に導入）
L2	モデルベース開発導入（2 プロセス以上に導入）
L3	モデルベース開発導入（全てのプロセスに導入）

本報告書では、表 1 で定義した適用レベルごとに、レガシー開発と比較したモデルベース開発の工数削減率を用いて、コスト評価を行う。そのため、本実験では、同じ製品の開発を、レガシー開発とモデルベース開発でそれぞれ開発したときの各工数データを収集する。実験で取得したレガシー開発のデータは、モデルベース開発との比較対象であるレガシー開発の品質妥当性を確認するため、品質指標を利用する。この実験の大まかな流れを図 1 に示す。

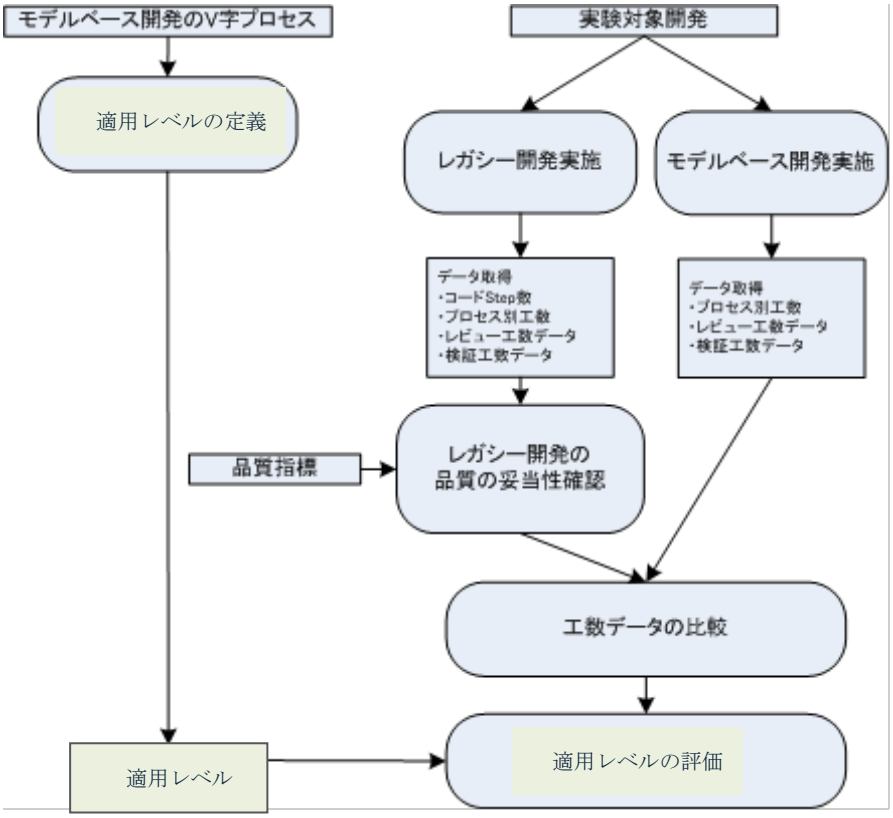

図 1 実験の流れ

２．モデルベース開発の概要

組込みソフトウェア開発の現状と課題を、経済産業省の「2010 年版組込みソフトウェア産業実態調査報告書」「2008 年版組込みソフトウェア産業実態調査報告書」におけるデータで整理し、課題を解決するための手段として、モデルベース開発について説明する。

２.１ ソフトウェア開発の現状と課題

図 2 は「2010 年版組込みソフトウェア産業実態調査報告書」における組込みソフトウェア開発の課題を示したグラフである。これを見ると、ソフトウェア開発において最も改善の必要がある課題は順に以下のようになることが分かる。

1. 設計品質の向上
2. 開発コストの削減
3. 開発期間の短縮

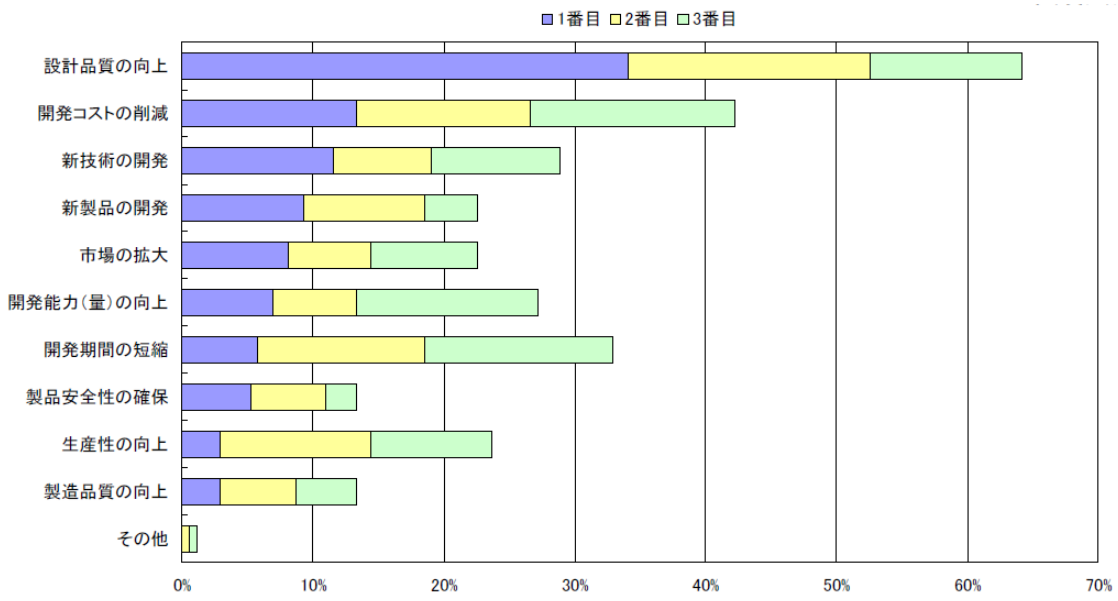

図 2 組込みソフトウェア開発の課題

2010 年版組込みソフトウェア産業実態調査報告書より抜粋

この上位 3 つの課題に関係しているデータを確認し、その要因や動向について整理する。

２．１．１　　設計品質の向上について

図 3 は「2010 年版組込みソフトウェア産業実態調査報告書」における製品の不具合の要因の割合を示したグラフである。組込みソフトウェアの不具合の製品開発全体における含有率は、ハードウェア設計の不具合や製品企画・仕様の不具合に比べて非常に多く、全体の約 60%を占めていることが分かる。

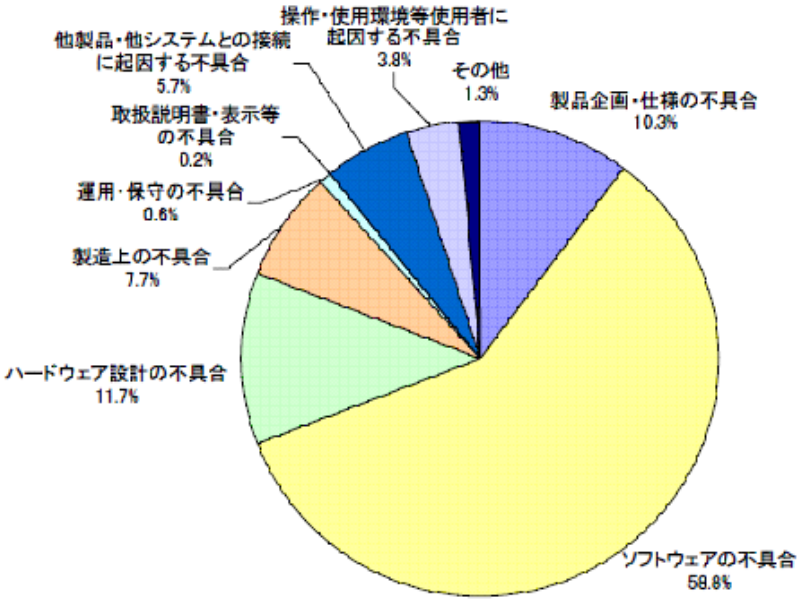

図 3　製品における不具合の要因

2010 年版組込みソフトウェア産業実態調査報告書より抜粋

図 4 は「2010 年版組込みソフトウェア産業実態調査報告書」における、組込みソフトウェアの各プロセスで発見される不具合をプロセス別に示したグラフである。ソフトウェアの不具合は主にソフトウェア実装・単体テストのプロセス以降で発見されており、バグの要因は前のプロセスであるソフトウェア要求定義、システムアーキテクチャ設計が特に多く見受けられる。要求定義・設計段階のプロセスで混入する不具合を削減することができれば、ソフトウェア開発における不具合発生件数を減らし、設計品質を向上できる可能性がある。

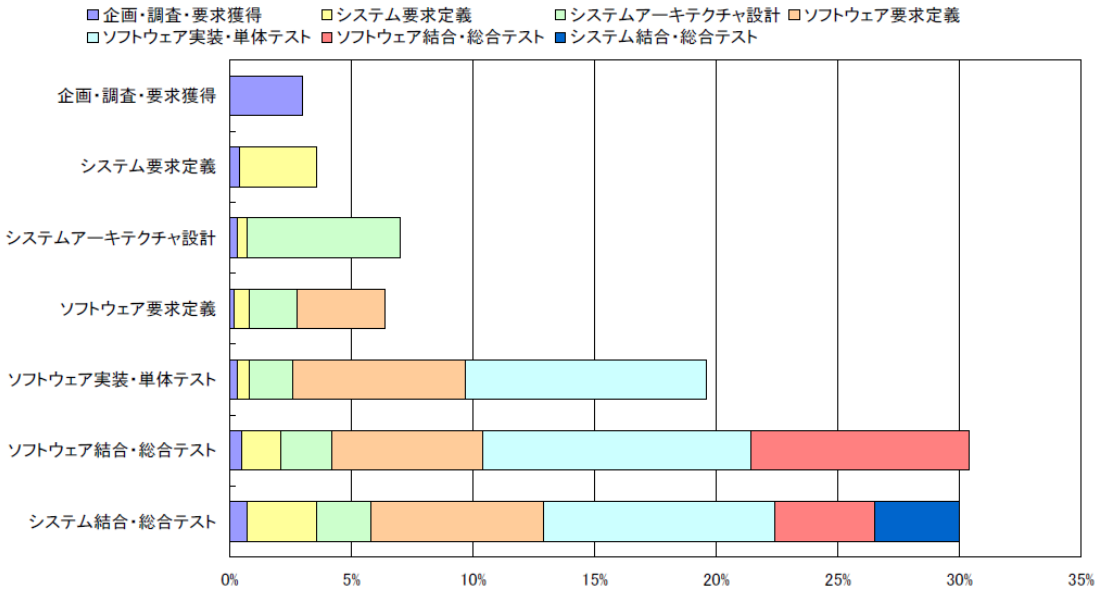

図 4 ソフトウェア不具合発見プロセス別件数比率と発生プロセスの内訳

2010 年版組込みソフトウェア産業実態調査報告書より抜粋

２．１．２　　開発コストの削減について

図 5、図 6 は「組込みソフトウェア産業実態調査報告書」における 2008 年と 2010 年の製品開発費の割合を示したグラフである。2008 年の製品開発におけるソフトウェア開発費は全体の 42.4%と、他の開発に比べて開発費の多くを費やされているが、2010 年には製品開発全体の 60.9%と、2008 年に比べてさらに開発費が増加傾向にある。製品製造における開発費は製品価格や利益に影響してくるため、開発費の削減は重要な課題である。

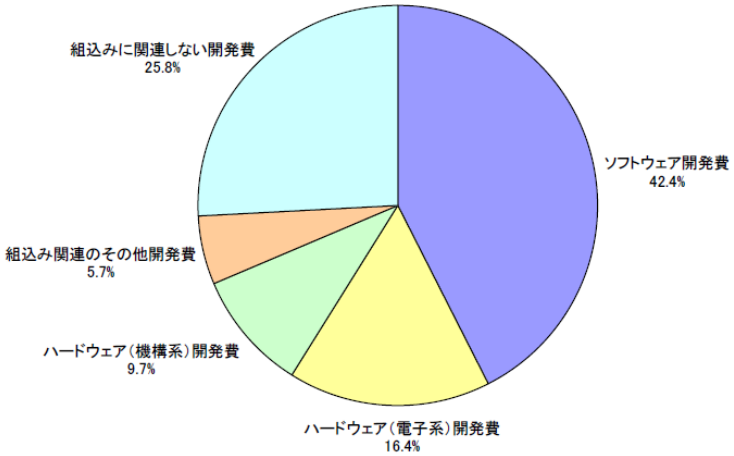

図 5　製品開発における開発費の割合(2008 年)

2008 年版組込みソフトウェア産業実態調査報告書より抜粋

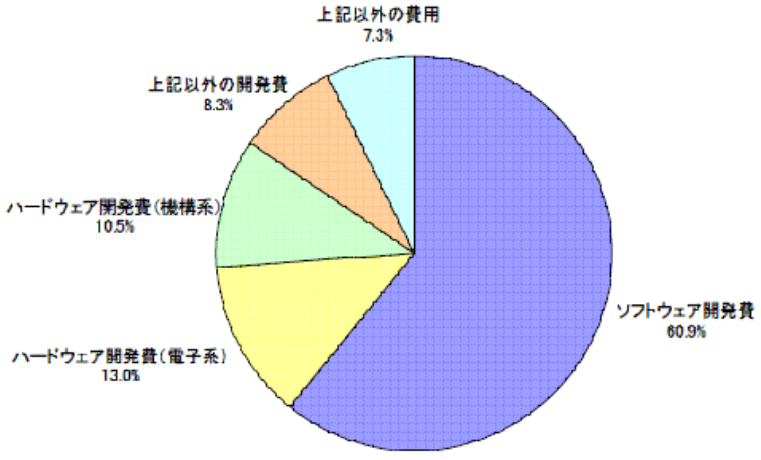

図 6　製品開発における開発費の割合(2010 年)

2010 年版組込みソフトウェア産業実態調査報告書より抜粋

２．１．３　　開発期間の短縮について

図 7、図 8 は「組込みソフトウェア産業実態調査報告書」における 2008 年と 2010 年の新規開発行数を示したグラフである。2008 年の新規開発行数の平均が 25.2 万行に対し、2010 年は平均 45.4 万行と 2 年間で 2 倍近く、開発行数が増加していることが分かる。この要因として、組込み製品の多機能化・多様化に伴う開発ソフトウェアサイズの肥大化、複雑化が考えられる。近年では、消費者のニーズに迅速に応えるため製品のライフサイクルが短くなっており、開発の期間短縮が課題になっている。

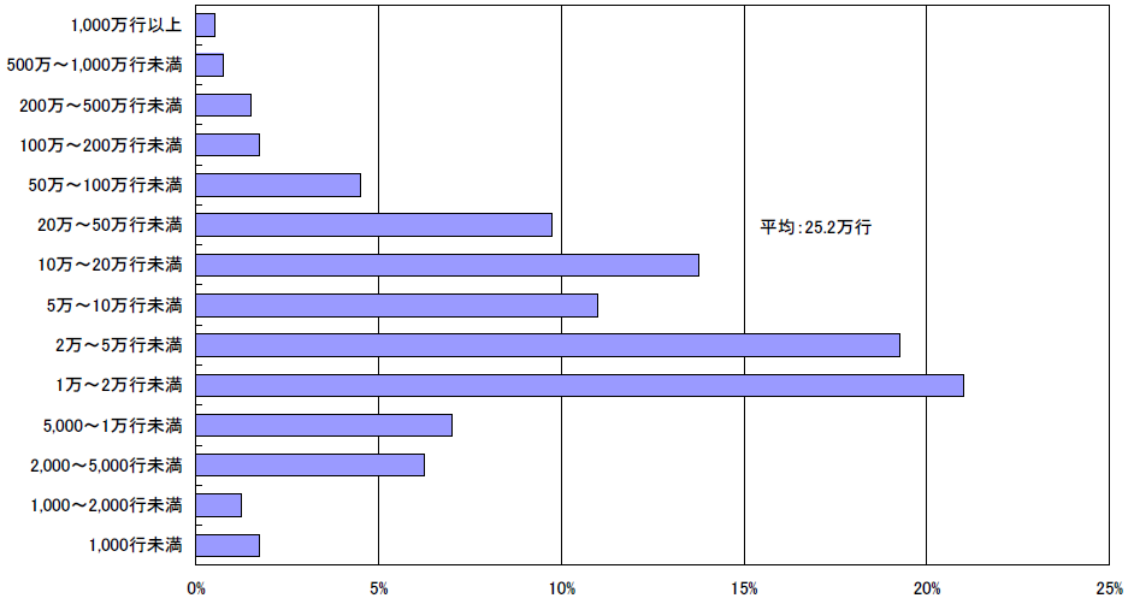

図 7　新規開発行数(2008 年)

2008 年版組込みソフトウェア産業実態調査報告書より抜粋

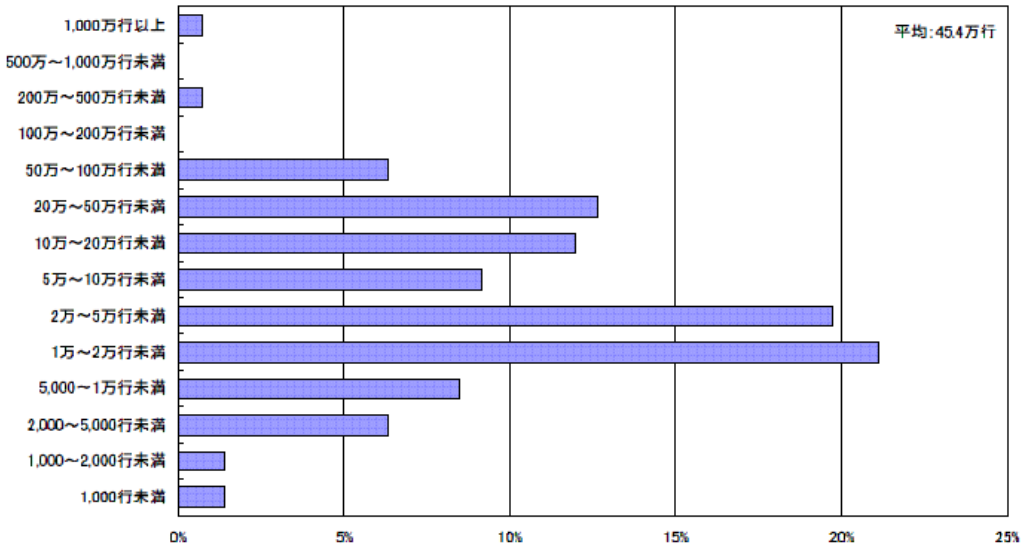

図 8 新規開発行数(2010 年)

2010 年版組込みソフトウェア産業実態調査報告書より抜粋

２．１．４　　課題への対応

前述した課題に対応するため、「設計品質の向上」「開発コストの削減」「開発期間の短縮」の 3 つの課題を解決することができる開発手法として、モデルベース開発を導入することが考えられる。モデルベース開発は、ソフトウェア開発プロセスの初期段階でシミュレーションによって不具合を早期発見することにより大きな手戻りを防ぐことができる。また、本実験でも使用したモデルベース開発の開発ツール製品である MATLAB/Simulink は、プログラムコード自動生成機能を持ち、この機能を使用することにより、ソフトウェア作成時にヒューマンエラーが混入しないため、迅速に正確なソフトウェア開発を行うことができる。

２.２　モデルベース開発とは

図 9 にソフトウェア開発のＶ字プロセスを利用した、レガシー開発の概要図を示す。レガシー開発は、システム設計・ソフトウェア設計プロセスでシステム設計やソフトウェア設計を実施後、これを基にソース作成プロセスでコーディングしたコードに対して検証・評価を行う。検証・評価プロセスでは、システム設計・ソフトウェア設計・コードが要求通りに作成されていることを、コードを動作させて確認する。不具合が出た場合は、修正されるまでＶ字プロセスを繰り返し実施する。

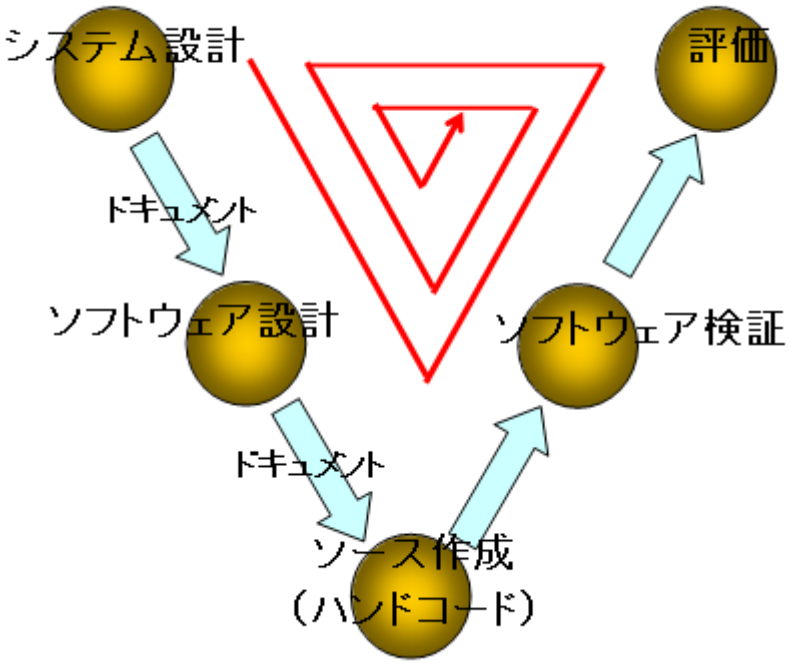

図 9　レガシー開発

図 10 にソフトウェア開発の V 字プロセスを利用した、モデルベース開発の概要図を示す。モデルベース開発では、システム設計・ソフトウェア設計プロセスの段階でシミュレーションを導入し、繰り返し動作検証を行いながらシステム設計・ソフトウェア設計をモデルで実施する。このため、ソフトウェア検証プロセスや適合・評価プロセス実施後にシステム設計・ソフトウェア設計・ソース作成プロセスへの手戻りは発生しない。なお、本実験では、モデル作成ツール製品として米国 Mathworks 社の MATLAB/Simulink を使用する。

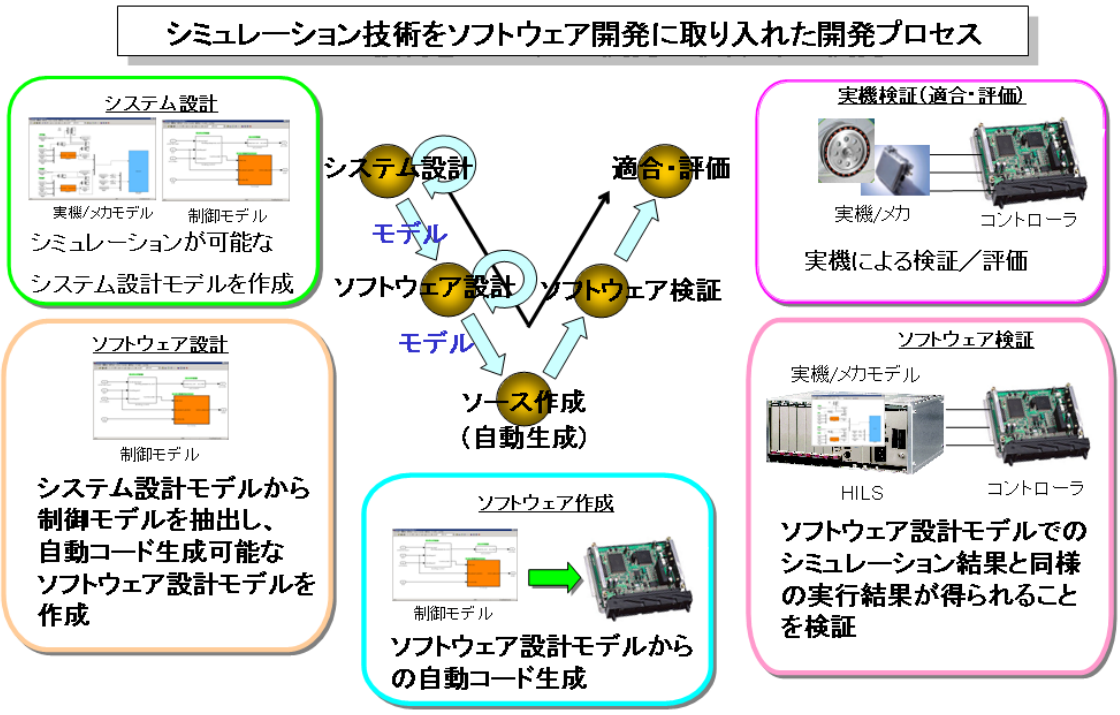

図 10 モデルベース開発

以降で、V 字プロセスのシステム設計・ソフトウェア設計・ソフトウェア作成・ソフトウェア検証・適合評価について、レガシー開発と比較したときのモデルベース開発における特徴をプロセス毎に記述する。

2.2.1 システム設計プロセス

レガシー開発では、机上で検討した内容をシステム設計プロセスで設計書としてシステム設計書にまとめており、ソフトウェア作成プロセス後に初めて設計したコードを動作可能になる。モデルベース開発では、システム設計の段階から MATLAB/Simulink のシミュレーション機能を導入し、モデルで設計をしながら動作確認ができることが特徴である。これによって、システム設計プロセスでのミスや不具合を設計段階で発見し、システム設計の品質を上げることができる。

２.２.２　　ソフトウェア設計プロセス

レガシー開発でのソフトウェア設計プロセスでは、設計書として基本設計書、詳細設計書などを作成し、これを基にソフトウェア作成プロセスでコーディングを行っていた。モデルベース開発を導入することにより、MATLAB/Simulink のシミュレーション機能で動作確認を行いながら、コントロールなどの詳細設計を行うことができる。これによってソフトウェア設計プロセスでのミスや不具合を設計段階で発見し、ソフトウェア設計の品質を上げることができる。

２.２.３　　ソフトウェア作成プロセス

モデルベース開発のソフトウェア作成プロセスでは、システム設計・ソフトウェア設計プロセスで作成したモデルを、MATLAB/Simulink の自動コード生成機能を使用してＣコードを作成する。レガシー開発ではシステム設計・ソフトウェア設計プロセスで作成した設計書を基にプログラマがプログラムを作成するため、文法ミスやロジックエラーなどが起こる可能性があった。モデルから自動的に C コードを生成することにより、ヒューマンエラーが混入しない正確なプログラミングを期待することができる。また、システム設計・ソフトウェア設計プロセスですでにミスや不具合を修正した段階にあるため、品質を保つことができる。さらに、システム設計・ソフトウェア設計プロセスで作成したモデルを自動コード生成で使用するため、本来レガシー開発では必要なハンドコード・単体検証の工数を削減することができる。

２.２.４　　ソフトウェア検証プロセス

レガシー開発では、検証する制御コントローラ以外はスタブやドライブなどの仮想環境を使用して検証を行うが、モデルベース開発では HILS(Hardware In the Loop Simulation)環境を利用する。実際の ECU に作成した制御コントローラなどのコードを乗せ、制御対象を仮想環境として用意することで、リアルタイムな検証を行うことができる。また、制御対象側を仮想環境で実施する利点としては、実機が高価である場合、実機がシビアである場合（危険性など）、実機の故障や経年劣化など、様々な条件を仮想的に実施できることなどがある。

２.２.５　　適合評価プロセス

適合評価プロセスはレガシー開発と同じく、作成した制御コントローラのコードを実際の ECU に乗せ、制御対象の実機を用意し、運用するのと同等の環境で検証を行う。

２.３　モデルベース開発の予想される利点

2.2節で説明したようなモデルベース開発の特徴から、レガシー開発と比較したときに予想される利点を考える。

２.３.１　　工数の削減

図 11 に示す通り、コーディングの短縮、レビュー時間の短縮、手戻りの軽減の効果により、全体的な工数はレガシー開発よりもモデルベース開発のほうが短い。

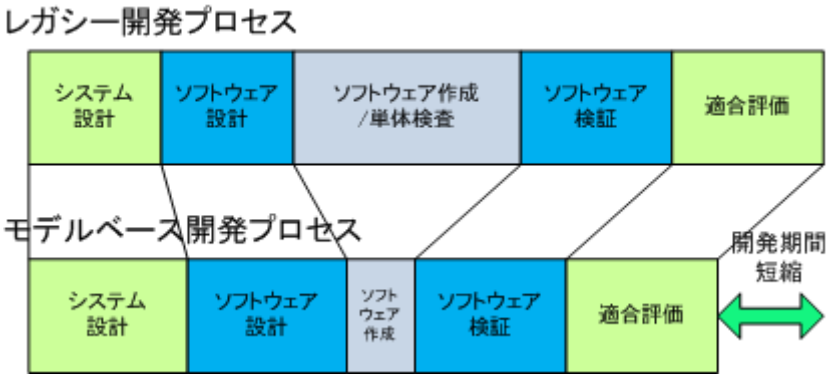

図 11　工数の削減

２.３.２　　コーディングの短縮

図 12 に示す通り、モデルベース開発のコード作成は自動であり、ヒューマンエラーが入り込まない。また、自動生成のため、コードを作成するための工数が必要ない。

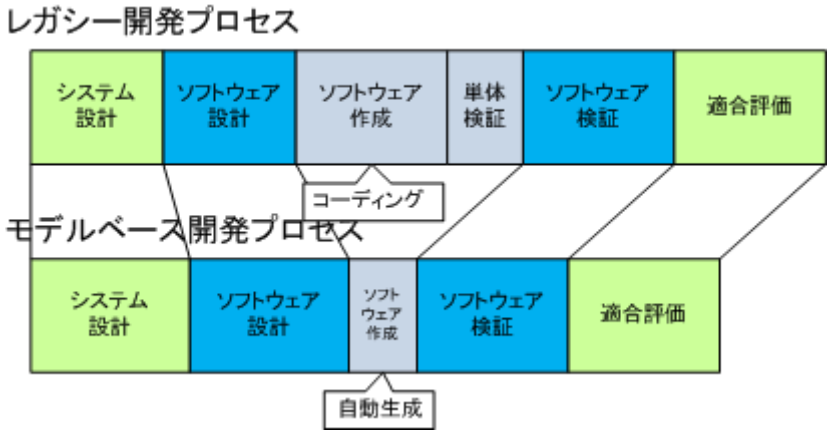

図 12　コーディングの短縮

２.３.３　　レビュー時間の短縮

モデルベース開発で作成するモデルは、抽象的な図を使用するため、コードよりも見易い。このため、レビューを行う第三者にも理解し易く、図 13 に示す通りレビューに掛かる時間を少なくすることができる。

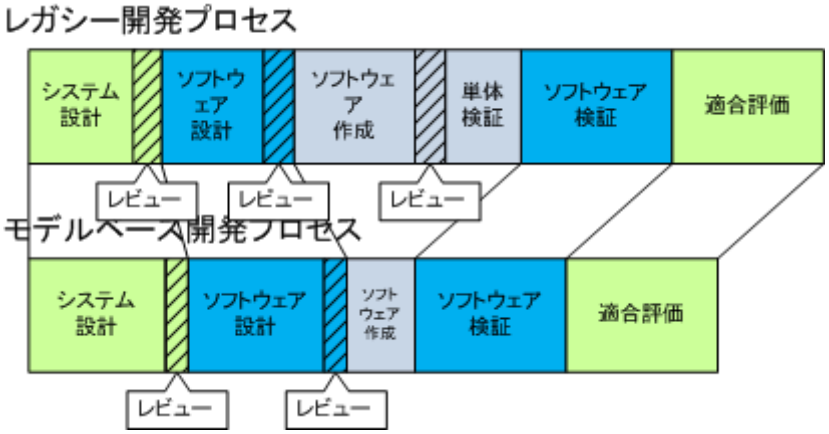

図 13 レビュー時間の短縮

２.３.４　　手戻りの軽減

モデルベース開発では、PC 上でモデルを使ってシミュレートし、検証を行いながらの設計となる。そのため、システム設計プロセスでは、作成目的である制御コードなどの動作確認をするためのハードウェアにあたるプラントモデルを作成する必要がある。また、ソフトウェア設計プロセスでは、システム設計プロセスで作成したプラントモデルを使用してシミュレートしながら設計を行う。このため、図 14 に示す通りシステム設計・ソフトウェア設計プロセスはレガシー開発よりも期間が長くなるが、アルゴリズムの不具合をコーディング前に発見することができるため、単体検証からの手戻りに掛かる工数を軽減することができる。

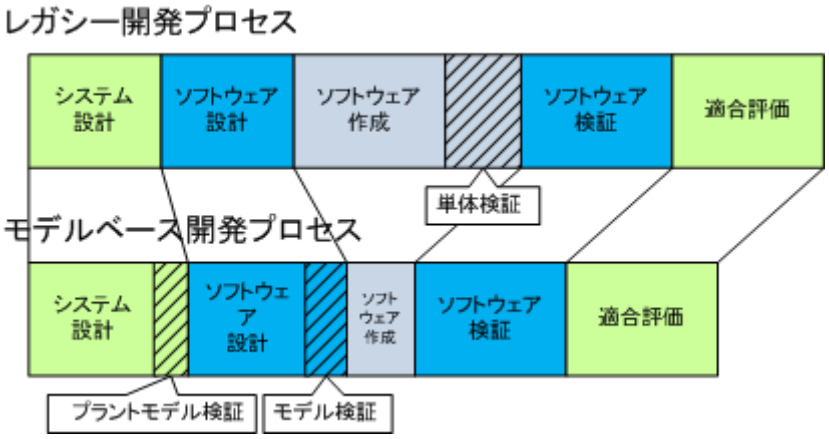
レガシー開発プロセス
システム設計
ソフトウェア設計
ソフトウェア作成
ソフトウェア検証
適合評価
単体検証
モデルベース開発プロセス
システム設計
ソフトウェア設計
ソフトウェア作成
ソフトウェア検証
適合評価
プラントモデル検証
モデル検証

図 14 手戻りの軽減

３．実験の前提

３.１ 実験の目的

レガシー開発における生産性指標には様々なものがあるが、手法の有効性の評価に活用できるほどのモデルベース開発に関する明確な指標はまだない。ソフトウェア品質説明力強化のための実効性のある開発技術を探るために、モデルベース開発の有効性を評価することが本実験の目的である。

３.２ 実験の範囲と手法

本実験の範囲を、表 2 に示す。

表 2 実験の範囲

実験対象	ボンディング装置用制御コントローラ開発作業
ライフサイクルの範囲	システム設計～ソフトウェア検証
評価の尺度	工数削減率 ・全体から見たプロセス毎の工数削減率（%） ・全体の工数削減率（%）

実験対象は「ボンディング装置用制御コントローラの開発作業」であり、開発作業の V 字プロセスにおけるシステム設計～ソフトウェア検証までのプロセスをデータ収集の範囲とする。

本実験では、モデルベース開発の導入度合を示す「適用レベル」を定義し、適用レベルごとに、レガシー開発とモデルベース開発の要した工数を比較し、工数削減率を割り出す。これを尺度として評価を行う。なお、レガシー開発は過去の開発であり、モデルベース開発を実施した組織とは異なるため、開発経験（学習）による工数削減効果を考慮する必要はない。

３.３ 実験対象の詳細

本節では、実験が対象とした開発作業が開発対象としたシステム（以下開発対象システム）と、開発作業（レガシー開発とモデルベース開発）について説明する。

３.３.１ 開発対象システムの概要

ボンディング装置用制御コントローラは、ボンディングシステムの一部の機能である。図 15 はボンディングシステムの構成図である。ボンディングシステムは半導体製造装置の 1 つで、ボンディング基板上にチップを乗せる作業を行う装置のことである。以下にこのボンディングシステムの

動作を簡単に示す。

1. ボンディング基板がローダマガジン供給軸からレールに流される
2. ピックアップステージにあるチップを、ダイトランスファでプリサイサステージに運ぶ
3. プリサイサステージにあるチップをボンディングでボンディング基板に貼る
4. チップの貼りつけを完了したボンディング基板をローダマガジン供給軸でレールから排出する

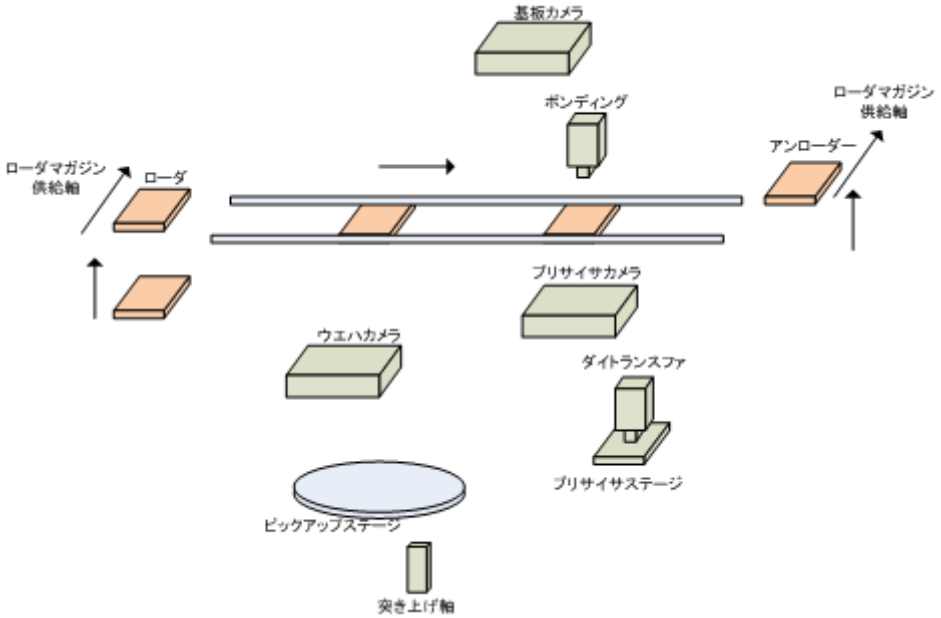

図 15 ボンディングシステムの構成図

本報告書の実験対象であるボンディング装置用制御コントローラ開発は、もともとレガシー開発で作成したものであったが、この機能に対するモデルベース開発の実現性について検証をするために再度ボンディング装置用制御コントローラ開発を行った。

モデルベース開発で開発した機能は、ボンディングシステムでも特にリアルタイム性がシビアなボンディング・ダイトランスファ・突き上げ軸の制御（ボンディング装置用制御コントローラ）である。また、レガシー開発で作成された現行システムでは、ソフトウェアの複雑化・肥大化により、それらをベースに開発を継続することは限界に近づいているという課題もある。そのため、モデルベース開発で設計から作り直すことによってこれらの課題を解消する。さらに、開発効率の向上と、作成した機能の再利用性を高め、実用性を示すことで、ボンディング・ダイトランスファ・突き上げ軸の制御だけでなくボンディングシステム全体に対してモデルベース開発導入を進めていくことを目的として開発を進めた。

３.３.２　　開発対象システムの仕様

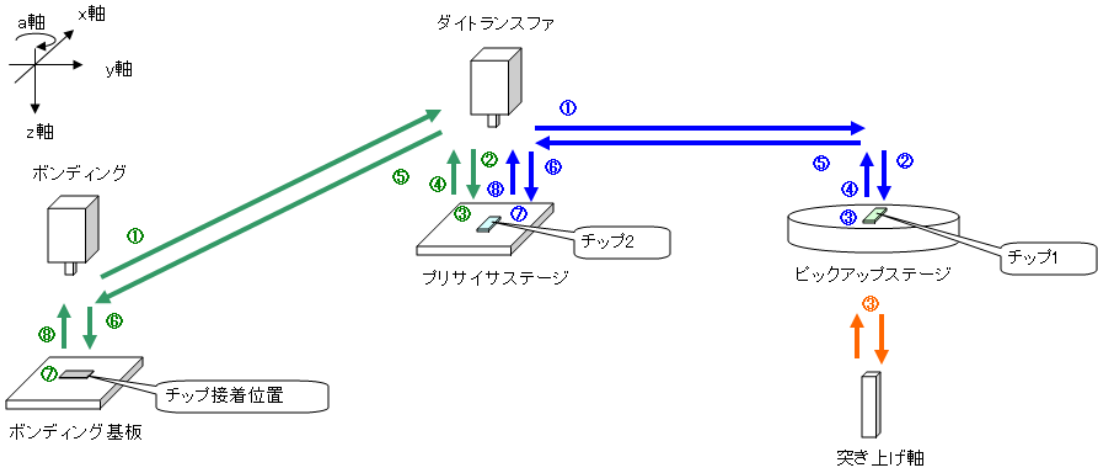

図　16　ボンディング装置用制御コントローラの構図

図　16　は、ボンディング・ダイトランスファ・突き上げ軸により、ピックアップステージにあるチップをボンディング基板に接着する部分の構図である。以下にその動作を示す。

1. ダイトランスファがピックアップステージに移動する
2. 突き上げ軸でピックアップステージにあるチップを持ち上げる
3. 持ち上げられたチップをダイトランスファが受け取り、プリサイサステージに置く
4. ボンディングはプリサイサステージに置かれたチップをボンディング基板に接着する

この一連の動作のシーケンスを図　17　に示す。ボンディング、ダイトランスファ、突き上げ軸はそれぞれ複数のステッピングモータで動作する。

シーケンス番号		1	2	3	4	5	6	7	8
ボンディング	x・y・a軸	プリサイサへ				ボンディングへ			
	z軸		下降		上昇		下降		上昇
	吸着			吸着				解除	
	接着							接着	
ダイトランスファ	y軸	ピックアップへ				プリサイサへ			
	z軸		下降		上昇		下降		上昇
	吸着			吸着				解除	
突き上げ	z軸			突き上げ					

図　17　ボンディング装置用制御コントローラのシーケンス

ボンディング装置用制御コントローラの開発では、この装置のボンディング、ダイトランスファ、突き上げ軸の 3 つの動作させるためのステッピングモータを制御するコントローラを作成する。

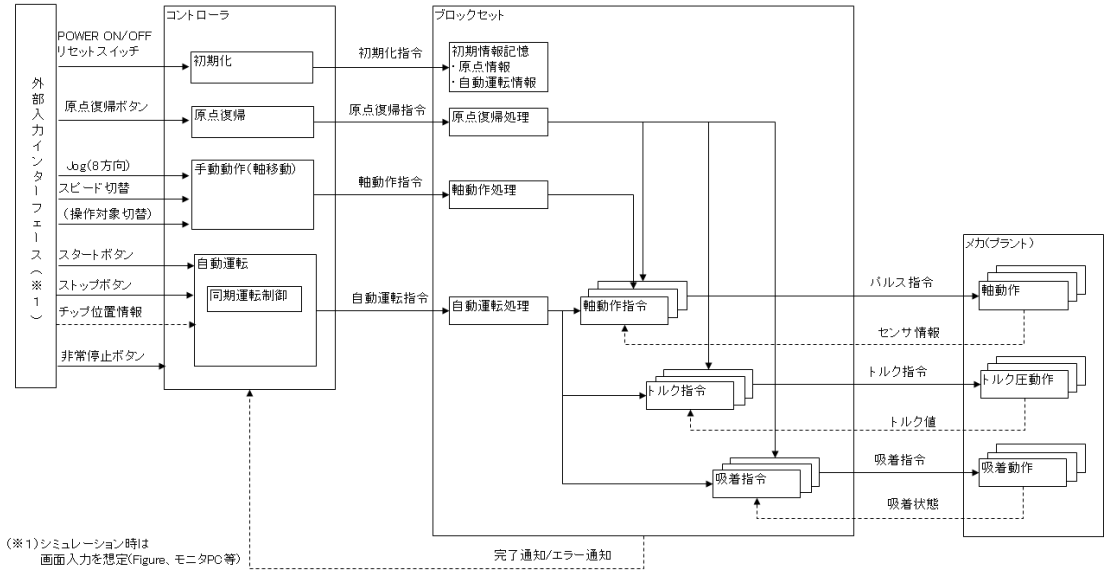

図 18 ボンディング装置用制御コントローラのシステム構成図

図 18 は、ボンディング装置用制御コントローラのシステム構成図である。外部インターフェースで入力された値から、図 18 中のコントローラでボンディング、ダイトランスファ、突き上げ軸の操作の種類を決定し、決定した操作で動作させるための必要なデータを図 18 中のブロックセットに渡す。ブロックセットはモーションコントローラのような役割を持ち、モータを動作させるために必要なパルス指令やトルク値などのデータを作成して図 18 中のメカ（プラント）に指令を出す。メカ（プラント）はボンディング、ダイトランスファ、突き上げ軸を物理的に動作させるステッピングモータにあたり、図 18 中のブロックセットで指令されたパルス指令やトルク値に従い動作する。本報告書で実験対象としたボンディング装置用制御コントローラは図 18 中のコントローラにあたる。

図 18 中のコントローラの入力インターフェースを図 19 の入力種別の外部入力に、出力インターフェースを図 19 の入力種別のコントローラ⇔ブロックセットに示す。

入力種別	項目	項目詳細	In/Out	値	補足
外部入力	POWER ON/OFF		In	1/0	1/0:ON/OFF
	リセットスイッチ		In	1/0	1/0:ON/OFF
	原点復帰ボタン		In	1/0	1/0:ON/OFF
	Jog	(8方向)	In	0～256	現状は3段階のみだが拡張性を考慮して汎用的な値とする
	スピード切替		In	0/1/2	0/1/2:Low/High/ウェハピッチ
	操作対象切替		In	0/1/2	0/1/2:ボンディング/ダイトラ/突き上げ
	スタートボタン		In	1/0	1/0:ON/OFF
	ストップボタン		In	1/0	1/0:ON/OFF
	チップ位置情報	X座標補正値	In	0～999	単位は0.1mm
		Y座標補正値	In	0～999	単位は0.1mm
		回転角補正値	In	0～3600	単位は0.1度
	非常停止ボタン		In	1/0	1/0:ON/OFF
コントローラ⇔ブロックセット	初期化指令		In	1/0	指令を受けてブロックセットが初期化情報を読み込み
	原点復帰指令		In	1/0	ティーチングがないため原点情報は固定値
	軸動作指令	制御軸種別	In	0～10	0を除き、7軸+トルク軸+吸着2軸(今回の制御は7軸のみ)
		移動速度	In	0～2000	単位はmm/s
	自動運転指令	自動運転開始	In	1/0	1/0:自動運転状態/自動運転解除
		チップX座標補正値	In	0～999	単位は0.1mm(ピックアップ、プリサイサ)
		チップY座標補正値	In	0～999	単位は0.1mm(ピックアップ、プリサイサ)
		チップ回転角補正値	In	0～3600	単位は0.1度(ピックアップ、プリサイサ)
	完了通知	制御軸種別	Out	0～10	0を除き、7軸+トルク軸+吸着2軸
		シーケンス番号	Out	0～16	各軸ごとにどのシーケンスが終了したかを通知
	エラー通知	エラー発生有無	Out	0～10	0は正常、エラー発生軸1～10(7軸+トルク軸+吸着2軸)
		エラー内容	Out	文字列	エラー内容
	一時停止指令	一時停止制御軸	In	16bit	1/0:一旦停止/停止解除
	非常停止指令		In	1/0	1/0:ON/OFF
ブロックセット⇔プラント	パルス指令	回転方向	In	1/0	1/0:正/負
		移動量	In	0～99999	パルス数
		スピード	In	0～9999	Hz
		加速時間	In	0～9999	ms
		減速時間	In	0～9999	ms
		直線補間	In	1/0	1/0:有/無
	トルク指令		In	0～99999	単位要確認
	吸着指令		In	1/0	1/0:ON/OFF
	センサ情報		Out	??	センサ数要確認
	トルク値		Out	0～99999	単位要確認
	吸着状態		Out	16bit	1/0:吸着中/吸着解除
	一旦停止指令		In	16bit	1/0:一旦停止/停止解除
	非常停止指令		In	1/0	1/0:ON/OFF

図 19 システム構成 I/O

3.3.3 レガシー開発環境と手順

3.3.2項の仕様を基に、レガシー開発を行った環境とその概要を表 3 に示す。

表 3 レガシー開発概要

目的	ボンディング装置用制御コントローラの設計、実装、検証
開発環境	Windows XP
必要ツール	gcc （C コンパイラ）
収集データ	コード Step 数 プロセス別工数(システム設計・ソフトウェア設計・ソフトウェア作成/単体検証・ソフトウェア検証) レビューデータ(レビュー時間・レビュー指摘件数・指摘項目修正時間) 検証データ(検証項目・不具合件数・不具合修正時間)
検証方法	単体検証、結合検証

3.3.2項の仕様を基に、レガシー開発を行ったときの、プロセス毎の作業を表 4 に示す。

表 4 プロセス毎のレガシー開発作業

プロセス	作業
システム設計	システム設計
ソフトウェア設計	基本設計、詳細設計
ソフトウェア作成/単体検証	コーディング、単体検証
ソフトウェア検証	結合検証

作業毎に、具体的な作業内容を説明する。

- システム設計

発注元で定義されたボンディング装置用制御コントローラの仕様を基に、機能を分け、各機能の入出力インターフェースを設計する。

- 基本設計、詳細設計

システム内部の設計を行う。システム設計で分けた機能ごとのシステム（以降、サブシステム）を実現するための処理ロジックやデータ設計を行う。

- コーディング、単体検証

外部設計、内部設計を基に、C 言語を用いてコーディングを行い、サブシステムごとに単体検証を実施する。

- 結合検証

コーディングしたサブシステム間の入出力インターフェースを繋いで、検証を実施する。

３.３.４　　モデルベース開発環境と手順

3.3.2項の仕様を基に、モデルベース開発を行った環境とその概要を表 5 に示す。

表 5 モデルベース開発概要

目的	ボンディング装置用制御コントローラの設計、実装、検証
開発環境	Windows XP
必要ツール	MATLAB/Simulink 2007b
収集データ	プロセス別工数(システム設計・ソフトウェア設計・ソフトウェア作成・ソフトウェア検証) レビューデータ(レビュー時間・レビュー指摘件数・指摘項目修正時間) 検証データ(検証項目・不具合件数・不具合修正時間)
検証方法	単体検証、結合検証

3.3.2項の仕様を基に、モデルベース開発を行ったときの、プロセス毎の作業を表 6 に示す。

表 6 プロセス毎のモデルベース開発作業

プロセス	作業
システム設計	トップレイヤ作成、プラントモデル作成、単体検証
ソフトウェア設計	構造レイヤ作成、データフローレイヤ作成、単体検証
ソフトウェア作成	ツールによるコード自動生成
ソフトウェア検証	結合検証

作業毎に、具体的な作業内容を説明する。

- トップレイヤ作成、単体検証

MATLAB/Simulink のモデルを使用して階層構造にサブシステムを作成していく。トップレイヤは、半導体製造装置用制御コントローラシステムの最上位階層のサブシステムのことであり、モデルの機能説明や、半導体製造装置用制御コントローラへの入出力を記述する。作成したトップレイヤの動作が発注元で定義されたボンディング装置用制御コントローラの仕様通りの動作をしていることを確認しながら設計を行う。

- データフローレイヤ作成、構造レイヤ作成、単体検証

構造レイヤは、トップレイヤより下階層のサブシステムである。このサブシステムは半導体製造装置用制御コントローラを機能毎に分割した塊であり、その機能に必要な入出力を配線する。構造レイヤ作成で作成した最下層のサブシステムに、制御ロジックを記述する。作成したモデルの動作が発注元で定義されたボンディング装置用制御コントローラの仕様通りの動作をしていることを確認しながら設計を行う。

- ツールによるコード自動生成

トップレイヤ作成、構造レイヤ作成、データフローレイヤ作成したモデル一式から、MATLAB/Simulink ツールの機能を利用してコードに自動生成する。

- 結合検証

MATLAB/Simulink ツールにより自動生成したコードを実際の ECU に乗せ、HILS 環境を用いて検証を実施する。

３.４ 実験項目

2.3節でも挙げたモデルベース開発の以下のメリットに注目し、これらを実験項目として、実験を行う。

- 工数の削減
- 手戻りの軽減
- コーディングの短縮
- レビュー時間の短縮

また、モデルベース開発を実施するにあたり、作業者へのモデルベース開発手法や、開発ツールの使用方法を教育する工数が必要になる。このときに必要な工数についても実験項目に加える。

- モデルベース開発初期導入に掛かる工数

４．適用レベルの定義と工数削減効果の想定

モデルベース開発における信頼性・安全性の度合いを表すための以下に説明する適用レベルを定義し、各レベルの工数削減の評価を想定する。

４.１ 適用レベルの定義

モデルベース開発の工数削減効果を最も得られるのは、開発における全てのプロセスにモデルベース開発を導入することである。しかし、これまでの開発手法やノウハウを手放すことにもなるため、開発時のリスクも大きくなる。一般的に、モデルベース開発は段階的に導入していくケースが多いため、モデルベース開発非導入、導入段階（1プロセスに適用）、過渡期（複数のプロセスに適用）、完成期（全プロセスに適用）の 4 つの段階を設け、これを適用レベルとして表 7 のように定義する。

表 7 適用レベル

適用レベル	定義
L0	モデルベース開発非導入
L1	モデルベース開発導入（1 プロセス以上に導入）
L2	モデルベース開発導入（2 プロセス以上に導入）
L3	モデルベース開発導入（全てのプロセスに導入）

４.２ 各適用レベルの工数削減効果の想定

V 字プロセスの各プロセスを単位として、モデルベース開発を導入するプロセス数で適用レベルを L0（レベル 0）～L3（レベル 3）までの 4 段階に分割する。以下に各適用レベルの工数削減の評価を想定する。

４.２.１ 適用レベル０

適用レベル 0 は、モデルベース開発を導入していない開発のため、工数削減はされない。

４.２.２ 適用レベル１

1 プロセスにモデルベース開発を導入したときの効果を以下に想定する。

● システム設計プロセスへの導入

システム設計プロセスへモデルベース開発を導入した場合、トップレイヤの設計・検証を繰り返し行う。このため、システム設計プロセスの工数が増加するが、本来適合評価で行う検証をシステム設計プロセスで前倒しに実施しているため適合評価の工数が減り、全体的な工数は実質レガシー開発と変化がないと考える。しかし、工数には直接効果は表れないが、システム設計プロセスでの検証をシステム設計段階で行うことにより、V字プロセス1回の実施における製品の品質が上がり、V 字プロセスの繰り返し実施回数がレガシー開発よりも少なくなると想定する。

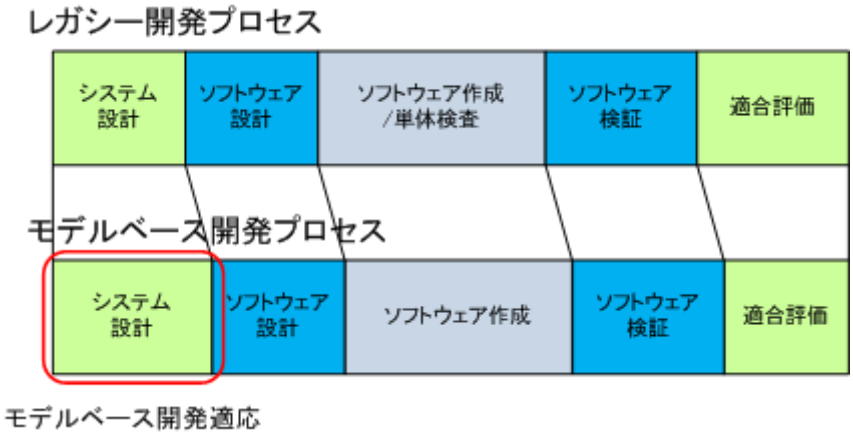

図 20 適用レベル 1 システム設計プロセス適用例

● ソフトウェア設計プロセスへの導入

ソフトウェア設計プロセスへモデルベース開発を導入した場合、構造レイヤ、データフローレイヤの設計・検証を繰り返し行う。このため、ソフトウェア設計プロセスの工数が増加するが、本来ソフトウェア検証プロセスで行う検証をソフトウェア設計プロセスで前倒しに実施しているためソフトウェア検証プロセスの工数が減り、全体的な工数は実質レガシー開発と変化がないと考える。しかし、システム設計プロセスへモデルベース開発を導入した場合と同様に、ソフトウェア設計プロセスでの検証をソフトウェア設計段階で行うことにより、V 字プロセス 1 回の実施における製品の品質が上がり、V 字プロセスの繰り返し実施回数がレガシー開発よりも少なくなると想定する。

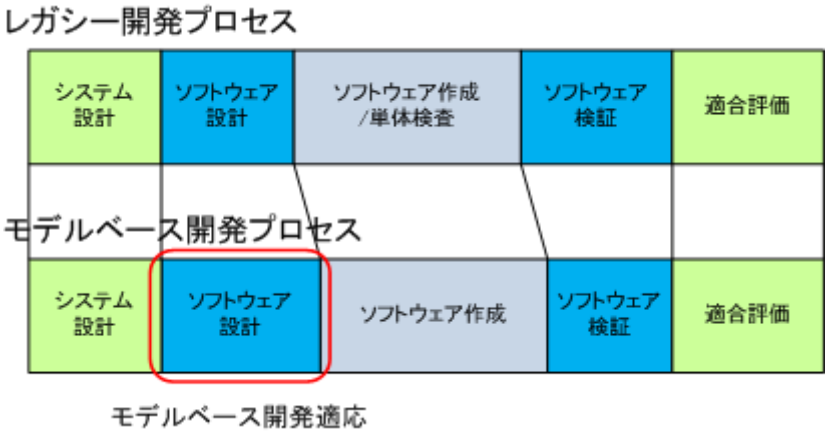

図 21 適用レベル 1 ソフトウェア設計プロセス適用例

- ソフトウェア作成プロセスへの導入

ソフトウェア作成プロセスへモデルベース開発を導入するには、コード自動生成可能な制御モデルが成果物として作成されていることが前提であるため、ソフトウェア設計プロセスにモデルベース開発を適用せずにソフトウェア作成プロセスにモデルベース開発を適用することはできない。

- ソフトウェア検証プロセスへの導入

ソフトウェア検証プロセスへモデルベース開発を導入した場合、ソフトウェア検証では HILS を利用して制御対象を仮想的に用意し、リアルタイムに実行・検証する。これにより、ソフトウェア検証プロセスに掛かる工数が削減できるため、レガシー開発に比べてＶ字プロセス１回の実施に掛かる工数を少なくすることができると考える。しかし、設計プロセスへモデルベース開発を導入していないため、１回のＶ字プロセス実施における品質は上がらず、Ｖ字プロセスの繰り返し実施回数はレガシー開発と変わらないと想定する。

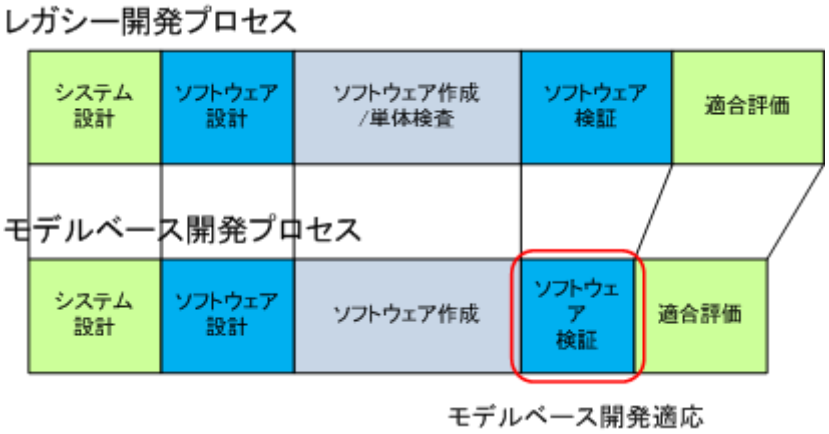

図 22 適用レベル 1 ソフトウェア検証プロセス適用例

- 適合評価プロセスへの導入

適合評価プロセスはモデルベース開発でもレガシー開発と同じ作業を行うため、適用対象外。

4.2.3 適用レベル 2

単純に V 字プロセスの 2 つ以上のプロセスの組合せは複数考えられるが、プロセスごとのモデルベース開発の作業により、選択するプロセスは決まってくる。以下に、2 個以上のプロセスの選択例を示す。

- システム設計プロセス、ソフトウェア設計プロセス、ソフトウェア検証プロセスへの導入

システム設計・ソフトウェア設計プロセスへモデルベース開発を導入した場合、設計プロセスは全てフロントローディングで実施する。また、ソフトウェア検証プロセスへモデルベース開発を導入することで、HILS を利用した検証を行う。この組合せでは、1 回の V 字プロセスにおける工数削減と、V 字プロセスの繰り返し回数削減の両方を実施することができると想定する。

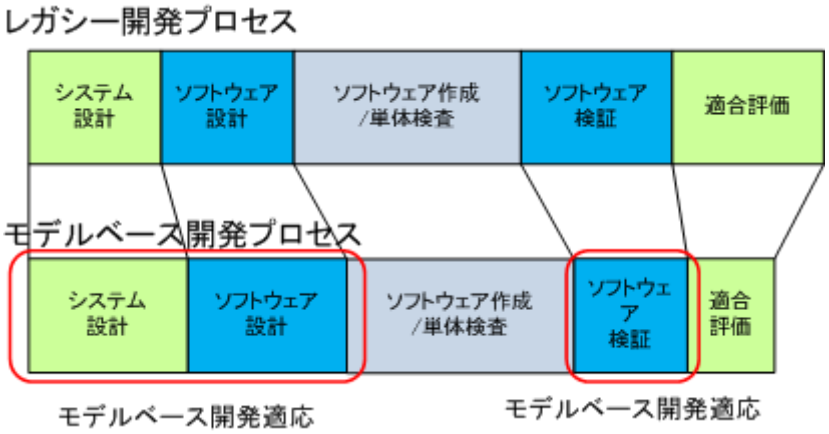

図 23 適用レベル 2 適用例

4.2.4 適用レベル 3

適用レベル 3 は、適用レベル 2 で挙げた例に加えて、ソフトウェア作成プロセスにもモデルベース開発を導入させた形となる。ソフトウェア作成プロセスにモデルベース開発を導入した場合、設計プロセスで検証・設計を繰り返し行ったモデルを、コード自動生成機能を使ってコード生成を行う。このため、ソフトウェア作成プロセスで混入するようなヒューマンエラーなどの不具合がなくなり、この分さらに V 字プロセスの繰り返し回数を減らすことができると考える。また、ソフトウェア作成プロセスにおけるハンドコードの工数も削減できるため、1 回のＶ字プロセスにおける工数もさらに削減できると想定する。

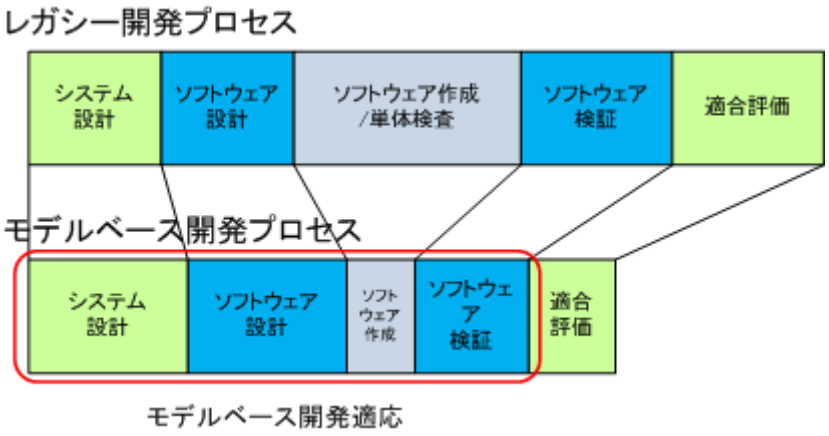

図 24 適用レベル 3 適用例

４.３ 実験で実施した適用レベル

全ての適用レベルを評価するには、開発未経験者の確保と、開発を設計から実施できる期間・工数が必要であり、実験期間内での実施は難しい。そこで本実験では適用レベル 3 を計測の対象とし、全てのプロセスに対してモデルベース開発を導入する。適用レベル 0～適用レベル 2 についての評価は、適用レベル 3 の実験結果から考察する。

５．品質指標

想定した適用レベルの妥当性を明らかにするため、実績のあるレガシー開発と比較を行う。比較対象であるレガシー開発の品質妥当性を確認するために、本報告書では SEC BOOKS「組込みソフトウェア開発向け品質作り込みガイド」の品質指標の目標値を利用する。

５.１ システムプロファイリングの決定

品質指標の目標値を定めるため、実験対象であるボンディング装置用制御コントローラ開発が図 25 のどのシステムタイプに該当するか検討する。

システムのタイプ：SCP (System Characteristics Profiling)
Type-1：Normal：通常レベルの品質や信頼性が求められるシステム
Type-2：Normal Quality Required：通常以上に高い品質や信頼性が求められるシステム
Type-3：Critical：高い品質や信頼性が求められるシステム
Type-4：Highly Critical：きわめて高い品質や信頼性が求められるシステム

図 25 システムタイプ一覧

SEC BOOKS「組込みソフトウェア開発向け品質作り込みガイド」より抜粋

以下の 2 つの判断によって、このタイプを決定する。

1. 人的損失の判定
2. 経済損失の算定

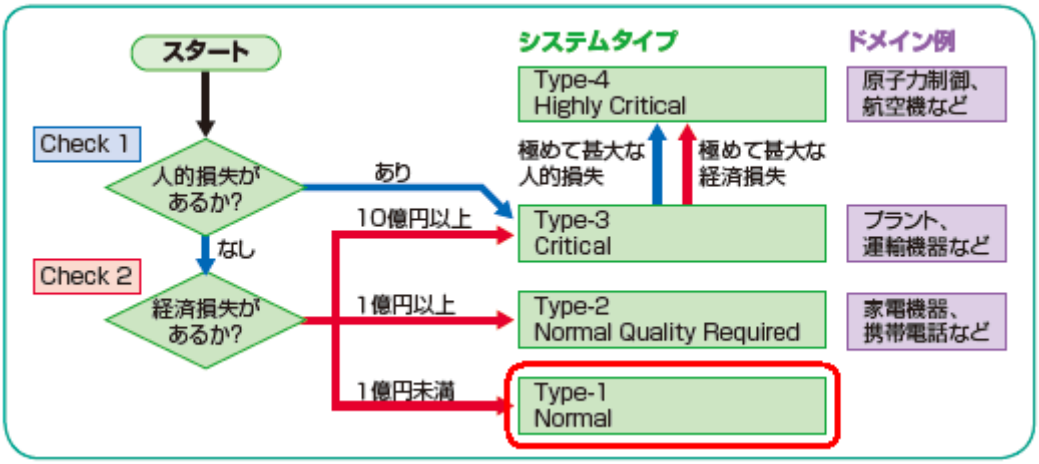

図 26 SCP チェック手順

SEC BOOKS「組込みソフトウェア開発向け品質作り込みガイド」より抜粋

本報告書での実験対象のボンディング装置用制御コントローラ開発は、人的損失が無く、経済損失が 1 億円未満であるため、図 26 から、システムタイプを Type-1 Normal と定める。

5.2 プロジェクトプロファイリングによる補正値の算出

次に図 27 のプロジェクトプロファイルのチェック表から、補正係数を求める。

ファクター		マイナス補正（－1）		基本		プラス補正（＋1）
①	ソフトウェア規模	■ 極めて小さい	□	普通	□	極めて大きい
②	ソフトウェアの複雑さ	■ 極めて単純	□	普通	□	極めて複雑
③	システム制約条件の厳しさ	□ 制約ゆるい	□	普通	■	制約厳しい
④	仕様の明確度合い	■ 極めて明確	□	普通	□	明確になっていない
⑤	再利用するソフトウェアの品質レベル	□ 極めて高品質	■	普通	□	極めて品質低い
⑥	開発プロセスの整備度合い	□ 整備できている	■	普通	□	整備できていない
⑦	開発組織の分業化・階層化の度合い	■ 開発組織が単純	□	普通	□	開発組織が複雑
⑧	開発メンバのスキル	□ メンバスキル高い	■	普通	□	メンバスキル低い
⑨	プロジェクトマネージャの経験とスキル	■ ＰＭスキル高い	□	普通	□	ＰＭスキル低い
⑩	システム障害時のメーカ側損失額	■ 極めて小さい	□	普通	□	極めて大きい
小計		−6		0		+1
合計ポイント数		−5				

図 27 プロジェクトプロファイルチェック

SEC BOOKS「組込みソフトウェア開発向け品質作り込みガイド」より抜粋

これより、補正係数を−5 と定める。

５.３　各品質指標の目標値算出

5.1節、5.2節のシステムタイプと補正係数から、各品質指標の目標値を定める。

５.３.１　　設計レビュー作業実施率の目標値

図 28 の設計レビュー作業実施率の指標から、目標値を求める。

ＩＤ	PR21
名称	設計レビュー作業実施率
略称	ERDR
名称（英語表記）	Execution Ratio of the Design Review

参考値	N	NQ	C	HC	補正ベース値
	7.20	9.60	12.00	14.40	2.40
参考値の範囲	4.80 ～ 9.60	7.20 ～ 12.00	9.60 ～ 14.40	12.00 ～ 16.80	
計測単位	人時／ KLOC				
許容誤差	有効数字上位3桁まで				
指標値の意味	● 設計のレビューにどれだけ工数をかけているかをプロジェクト規模とのバランスで表します。 ● 安全性、信頼性を要求されるシステムほどレビューにかける工数は多くなります。				
計算方法	設計レビュー工数/ソースコード全行数 ERDR = REDE/TLOC				
指標の利用法	● 設計のレビューには一定以上の適切な工数をかけて行うことが求められます。 ● 本指標値が参考値より小さい場合には、設計のレビューが甘く結果として設計の漏れや抜けが残りやすくなります。 ● 逆に、本指標値が参考値より大きい場合には、仕様が曖昧なために設計レビューがしづらくなっている、設計構造が複雑すぎる、機能が多い、といったプロジェクト上のリスクが入っていることが考えられます。				
備考： 参考値の解釈	● 本指標の参考値はすべて新規に作成した場合を想定して算出しています。 ● ソフトウェアの再利用率が高い場合はレビュー対象が規模に対して少なくなると考えられるので、参考値より少なく見積もることができます。 ● また、本指標はプロジェクトの求める品質レベル、構造の複雑さ、異常系の処理などにより、値が変化します。				

図 28　設計レビュー作業実施率の指標

SEC BOOKS「組込みソフトウェア開発向け品質作り込みガイド」より抜粋

システムタイプが Type-1 Normal であるため、参考値の N の 7.20%を基準とする。補正係数と参考値を基に、実験対象開発の設計レビュー作業実施率を求める。

実験対象開発の設計レビュー作業実施率
=当該タイプの設計レビュー作業実施率+補正係数/10×補正ベース値
=7.20+(-5)/10×2.40
=**6.00**

５.３.２　　コードレビュー作業実施率の目標値

図 29 のコードレビュー作業実施率の指標から、目標値を求める。

ＩＤ	PR22
名称	コードレビュー作業実施率
略称	ERCR
名称（英語表記）	Execution Ratio of the Code Review

参考値	N	NQ	C	HC	補正ベース値
	3.60	4.80	6.00	7.20	1.20
参考値の範囲	2.40 ～ 4.80	3.60 ～ 6.00	4.80 ～ 7.20	6.00 ～ 8.40	
計測単位	人時／ KLOC				
許容誤差	有効数字上位3桁まで				
指標値の意味	● ソースコードのレビューにどれだけ工数をかけているかをプロジェクト規模とのバランスで表します。 ● 安全性、信頼性を要求されるシステムほどレビューにかける工数は多くなります。				
計算方法	コードレビュー工数/ソースコード全行数 ERCR = RECO/TLOC				
指標の利用法	● ソースコードのレビューには一定以上の適切な工数をかけて行うことが求められます。 ● 本指標値が参考値より小さい場合には、対象ソースコードの必要な部分に十分目が行き届かずソースコード上の不具合が残りやすくなります。 ● 逆に、本指標値が参考値より大きい場合には、ソースコードの理解性、保守性等が良くない、レビューの効率が良くないなどのプロジェクト上のリスクが入っていることが考えられます。				
備考： 参考値の解釈	● 本指標の参考値はすべて新規に作成した場合を想定して算出しています。 ● ソフトウェアの再利用率が高い場合はレビュー対象が規模に対して少なくなると考えられるので、参考値より少なく見積もることができます。 ● また、本指標はプロジェクトの求める品質レベル、構造の複雑さ、異常系の処理などにより、値が変化します。				

図 29 コードレビュー作業実施率の指標

SEC BOOKS「組込みソフトウェア開発向け品質作り込みガイド」より抜粋

システムタイプが Type−1 Normal であるため、参考値の N の 3.60%を基準とする。補正係数と参考値を基に、実験対象開発のコードレビュー作業実施率を求める。

実験対象開発のコードレビュー作業実施率

=当該タイプのコードレビュー作業実施率+補正係数/10×補正ベース値

=3.60+(−5)/10×1.20

=**3.00**

５.３.３　　テスト密度の目標値

図 30 のテスト密度の指標から、目標値を求める。

ＩＤ	PD40
名称	テスト密度
略称	DOTI
名称（英語表記）	Density Of Test Items

参考値	N	NQ	C	HC	補正ベース値
	25.00	50.00	75.00	100.00	25.00
参考値の範囲	0.00 ～ 50.00	25.00 ～ 75.00	50.00 ～ 100.00	75.00 ～ 125.00	
計測単位	項目/KLOC				
許容誤差	有効数字上位2桁まで				
指標値の意味	● ソース規模あたりのテスト実施項目数を示します。この値は動的テストの十分性、ソースに対するテスト網羅性の目安となります。				
計測方法	テスト項目数 / ソースコード全行数 DOTI＝NOTI/TLOC				
指標の利用法	● テスト密度は対象の規模が大きくなれば、組み合わせの数などが増えるので増加する傾向にあります。ただし、やみくもに増やせばよいというわけではなく、効果のある組み合わせを考えてテストを設計することが必要です。また、対象システムの複雑さ、入出力の数などにより、テスト密度はそれ以上に変化しますので、補正係数などを考慮して十分バランスをとるようにすることが必要です。 ● 本指標が参考値より小さい場合は、対象システムの規模に対して適切な数のテストが実施されていないことを意味し、テストによる不具合の検出が不十分になる可能性があると考えることができます。 ● 逆に本指標が参考値より大きい場合は、余計なテストを実施しており、テストの効率が良くないなどの可能性があります。				
備考： 参考値の解釈	● 参考値は、関数1つの行数を120行とした場合、NQで関数1つに対し、6項目のテストを用意するといった目安で算出しています。 ● 本指標値をさらに有効活用するには、ソフトウェア全体のテスト密度を測るだけでなく、モジュールごと、機能ごとに難易度、複雑度が高い部分、入出力の多い部分などを考慮してテスト密度が適当であるかなどを見ると良いでしょう。				

図 30　テスト密度の指標

SEC BOOKS「組込みソフトウェア開発向け品質作り込みガイド」より抜粋

システムタイプが Type-1 Normal であるため、参考値の N の 25.00%を基準とする。補正係数と参考値を基に、実験対象開発のテスト密度を求める。

実験対象開発のテスト密度
=当該タイプのテスト密度+補正係数/10×補正ベース値
=25.00+(-5)/10×25.00
=**12.5**

５.４ 実験対象開発における品質指標の目標値のまとめ

実験対象開発における品質指標の参考値、参考値の範囲、算出した目標値を以下の表 8 にまとめる。

表 8 実験対象開発における指標

品質指標	参考値	参考値の範囲	目標値
設計レビュー作業実施率	7.20	4.80～9.60	6.00
コードレビュー作業実施率	3.60	2.40～4.80	3.00
テスト密度	25.00	0.00～50.00	12.5

６．実験の実施

実験対象をレガシー開発とモデルベース開発とし、各開発のデータの収集を実施する。

６.１　レガシー開発の実験データ

表 9、表 10、表 11、表 12 に、レガシー開発の実験データをまとめる。表 11、表 12 の不具合件数は、不具合が起因するプロセス毎にまとめる。

表 9 レガシー開発のプロセス別工数

プロセス	工数[人月]
システム設計	0.59
ソフトウェア設計	1.87
ソフトウェア作成/単体検証	2.72
ソフトウェア検証	1.23

表 10 レガシー開発のレビュー時間と修正時間

プロセス	レビュー時間[人時]	修正時間[人時]
システム設計	13	10
ソフトウェア設計	17	20
ソフトウェア作成・単体検証	15	5

表 11 レガシー開発の単体検証

検証項目数[件]	351	不具合件数[件]	修正時間[人時]
不具合件数	システム設計	2	6
	ソフトウェア設計	6	4
	ソフトウェア作成	6	2
	合計	14	12

表 12 レガシー開発の結合検証

検証項目数[件]	67	不具合件数[件]	修正時間[人時]
不具合件数	システム設計	0	0
	ソフトウェア設計	8	10
	ソフトウェア作成	0	0
	合計	8	10

６．１．１　　設計レビューの品質

表 8 の品質指標を使用して、レガシー開発で実施した設計レビューの品質の妥当性を確認する。なお、設計レビューには、システム設計プロセスとソフトウェア設計プロセスのレビューを両方含む。

以下の計算式から、実験対象の設計レビューの作業実施率を求める。算出方法は図 28 の計測単位を使用する。

設計レビュー作業実施率
=(システム設計レビュー時間+ソフトウェア設計レビュー時間)/ソースコード全行数
=(13+17)/5.442
=**5.51**

この値は参考値の範囲内であり、かつ目標値に近い値であるため、ボンディング装置用制御コントローラ開発で実施した設計レビューは品質として問題ない。

６．１．２　　コードレビューの品質

表 8 の品質指標を使用して、レガシー開発で実施したコードレビューの品質の妥当性を確認する。

以下の計算式から、実験対象のコードレビューの作業実施率を求める。算出方法は図 29 の計測単位を使用する。

コードレビュー作業実施率
=(ソフトウェア作成・単体検証レビュー時間)/ソースコード全行数
=15/5.442
=**2.76**

この値は参考値の範囲内であり、かつ目標値に近い値であるため、ボンディング装置用制御コントローラ開発で実施したコードレビューは品質として問題がない。

６．１．３　　検証の品質

表 8 の品質指標を使用して、レガシー開発で実施した検証の品質の妥当性を確認する。なお、品質指標の「テスト」という言葉は本報告書に記載している「検証」と考える。また、テスト密度は結合検証・総合検証の合計から求めるため、今回は結合のみで実施する。

以下の計算式から、実験対象の検証の作業実施率を求める。算出方法は図 30 の計測単位を使用する。

テスト密度
=レガシー開発結合検証の検査項目数/ソースコード全行数
=67/5.442
=**12.3**

この値は参考値の範囲内であり、かつ目標値に近い値であるため、ボンディング装置用制御コントローラ開発で実施した結合検証は品質として問題がない。

６.２ モデルベース開発の実験データ

表 13、表 14、表 15、表 16、表 17 に、モデルベース開発の実験データをまとめる。表 15、表 16、表 17 の不具合件数は、不具合が起因するプロセス毎にまとめる。

表 13 モデルベース開発のプロセス別工数と導入工数

プロセス	工数[人月]
システム設計	0.78
ソフトウェア設計	2.18
ソフトウェア作成	0.94
ソフトウェア検証	0.49
導入工数	0.63

表 14 モデルベース開発のレビュー時間と修正時間

プロセス	レビュー時間[人時]	修正時間[人時]
システム設計	3	8
ソフトウェア設計	3	16
ソフトウェア作成	0	0

表 15 モデルベース開発のシステム設計での検証

検証項目数[件]	50	不具合件数[件]	修正時間[人時]
不具合件数	システム設計	5	2
	合計	5	2

表 16 モデルベース開発のソフトウェア設計での検証

検証項目数[件]	302	不具合件数[件]	修正時間[人時]
不具合件数	システム設計	0	0
	ソフトウェア設計	18	5
	合計	18	5

表 17 モデルベース開発のソフトウェア検証での検証

検証項目数[件]	60	不具合件数[件]	修正時間[人時]
不具合件数	システム設計	0	0
	ソフトウェア設計	3	3
	ソフトウェア検証	0	0
	合計	3	3

６.３ 実験結果

実験データから工数削減率を求め、適用レベルの評価を行う。今回の開発では、モデルベース開発を全てのプロセスに適用しているため、適用レベル 3 が対象となる。適用レベル 0～2 については、7章の考察で実験の結果から述べることとする。図 31 のボンディング装置用制御コントローラ開発における各プロセスの工数から、全体工数に対する各プロセスの削減率と全体の削減率を計算し、表 18 の評価欄に示す。結果は小数点以下を四捨五入する。

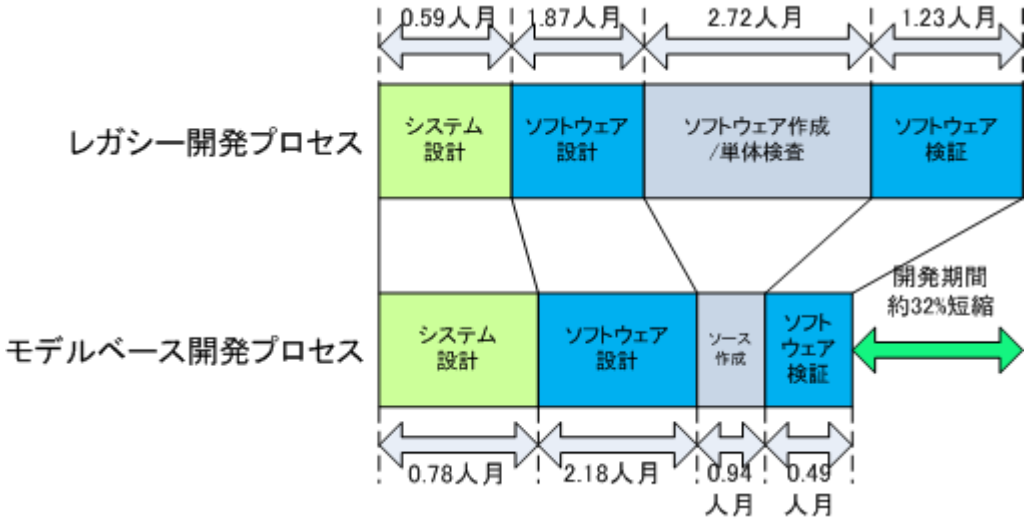

図 31 モデルベース開発とレガシー開発の工数

表 18 適用レベル 3 の評価

※ ○は適用、×は非適用 適合・評価プロセスは実験で実施していないため － とする

適用レベル	適用プロセス		評価（削減率）	
L3	システム設計	○	−3%	全体
	ソフトウェア設計	○	−5%	32%
	ソフトウェア作成	○	28%	
	ソフトウェア検証	○	12%	
	適合・評価	−	−	

この結果より、ソフトウェア開発に適用レベル 3 を適用すると約 32%の工数を削減することができると分かった。その内訳をみると、システム設計・ソフトウェア設計プロセスの工数がレガシー開発と比較して増加しており、その反面ソフトウェア作成・ソフトウェア検証プロセスの工数が設計プロセスの工数増加を上回る削減率であることが分かる。

※期間/費用の関係上、本実験では 1 サイクルのみで評価を行った。2 サイクル目以降は開発者の学習効果などにより工数の削減率はより高くなると予想されるので、本実験では実施しない

こととした。

７．考察

６.１節、６.２節の実験データと6.３節の実験結果から、４章の適用レベルの定義と工数削減効果の想定で想定した工数削減効果の評価について考察する。

７.１　各実験項目についての考察

レガシー開発とモデルベース開発のそれぞれの実験結果を実験項目ごとに比較し、考察する。

７.１.１　　工数の削減

表 9 と表 13 から、レガシー開発とモデルベース開発それぞれに掛かった工数を、プロセス毎にまとめて比較し、その差を図 32 に示す。MATLAB/Simulink を使用したシステム・ソフトウェア設計では、レガシー開発に比べて 0.2、0.3 人月分の工数が掛かっている。対照的に、ソフトウェア作成/単体検証におけるモデルベース開発の工数はレガシー開発の同プロセスに比べると顕著に工数が減少していることが分かる。

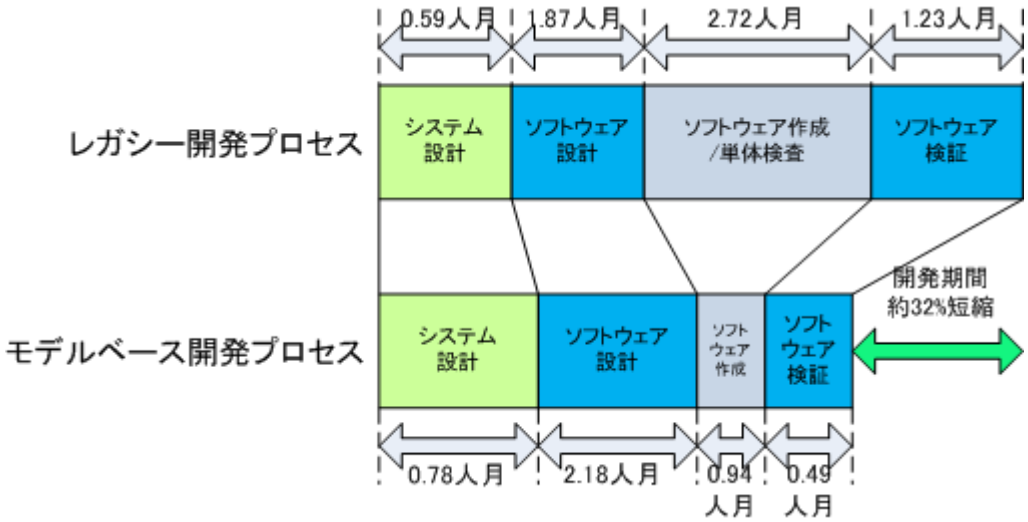

図 32　モデルベース開発とレガシー開発の工数（再掲）

レガシー開発期間とモデルベース開発期間から、工数削減率を求め表 19 に示す。

表 19 全体工数削減率

レガシー開発期間	6.41 人月
モデルベース開発期間	4.39 人月
削減率	約 32%

７.１.２　　コーディングの短縮

表 9 と表 13 から、レガシー開発とモデルベース開発のソフトウェア作成プロセスに掛かった工数を図 33 に示す。レガシー開発では、コーディングと動作確認の工数を合わせて 2.72 人月掛かった。一方、モデルベース開発は 0.94 人月と、レガシー開発の工数の半分以下であった。

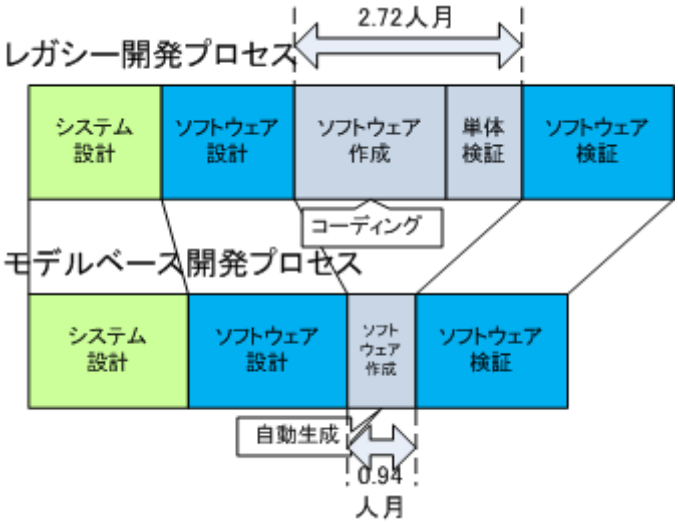

図 33 コーディング短縮による効果

モデルベース開発のソフトウェア作成プロセスで掛かる工数は、レガシー開発に比べて大幅に工数を削減することができる。モデルベース開発のソフトウェア作成プロセスに掛かっている工数は、コード自動生成するために必要なツールへの設定によるものである。

７.１.３　　レビュー時間の短縮

モデルベース開発を導入することで、システム設計・ソフトウェア設計・ソフトウェア作成プロセスにおけるレビュー工数と、レビューによる指摘事項の修正工数が短縮することを確認する。

表 10 と表 14 のレビュー時間、修正時間から、以下の計算式で各プロセスに掛かったレビュー工数を求める。なお、算出された工数は四捨五入し、1 日 7.75 時間、1 人月 20 人日として換算する。

（レビュー時間＋修正時間）/(20*7.75)=レビュー工数

レガシー開発とモデルベース開発の各プロセスに掛かったレビュー工数を、図 34 に示す。

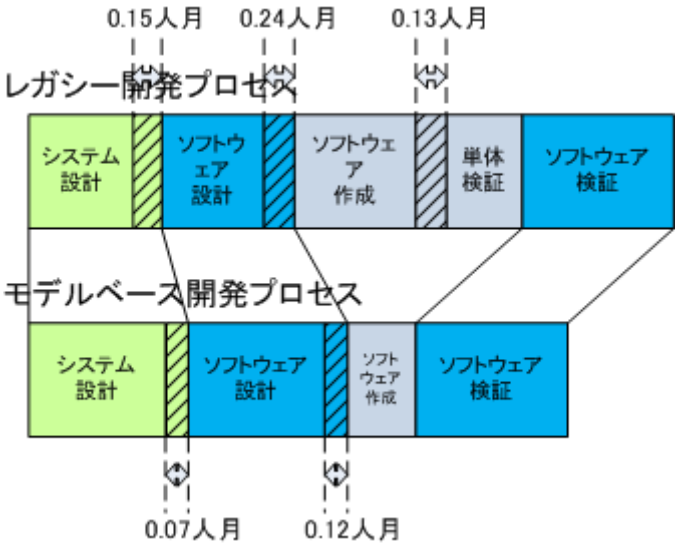

図 34 レビュー時間の短縮による効果

レガシー開発とモデルベース開発それぞれのレビューの総工数を表 20 に示す。

表 20 レビュー総工数

プロセス	レビューに掛かる全体工数
レガシー開発	0.52 人月
モデルベース開発	0.19 人月

レガシー開発とモデルベース開発のシステム設計・ソフトウェア設計プロセスを比較すると、レビュー工数に掛かる工数がそれぞれ 0.08 人月と 0.12 人月少なくなっていることが分かる。また、モデルベース開発のソフトウェア作成プロセスではツールのコード自動生成で行われ、ヒューマンエラーが入り込まないため、レビューの必要がない。これにより、全体的にモデルベース開発のレビュー実施工数は 0.33 人月削減されている。また、モデルベース開発は、機能を抽象化して表すモデルを利用しているため第三者が見易く、理解し易い。システム設計・ソフトウェア設計プロセスでのモデルベース設計のレビュー時間がレガシー開発よりも少なくなっているのは、このためと考えられる。

７．１．４　　手戻りの軽減

モデルベース開発を導入することで、検証を実施した際に発見された不具合の修正対応に掛かる工数が軽減することを確認する。

表 11、表 12、表 15、表 16、表 17 における不具合の修正時間から、以下の計算式で各プロセスに掛かった修正工数を求める。なお、算出された工数は四捨五入し、1 日 7.75 時間、1 人月 20 人日として換算する。

（修正時間）/(20*7.75)=修正工数

レガシー開発とモデルベース開発に掛かった不具合修正工数を図 35 に表す。

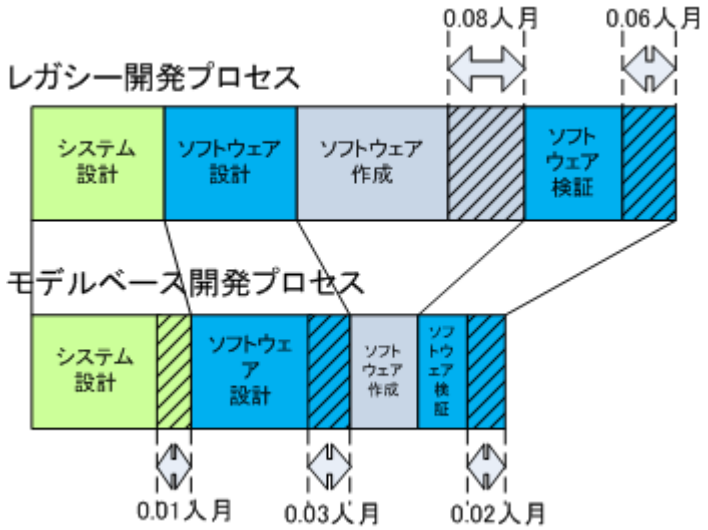

図 35 手戻りの軽減による効果

レガシー開発の不具合修正時間を見ると、不具合の原因が前プロセスであるほど不具合 1 件当たりの修正工数が多くなっていることが分かる。モデルベース開発では、不具合修正時間が全体的に少ない。モデルベース開発の不具合の発見されたプロセスを確認すると、システム設計・ソフトウェア設計プロセスで発見された不具合件数がレガシー開発より多いことが分かる。逆に、ソフトウェア検証プロセスで発見された不具合件数はレガシー開発よりも少ないことが分かる。これより、モデルベース開発は、設計プロセスへのフロントローディングによって従来検証プロセスで発見される設計プロセス側の不具合を設計段階で解決し、手戻りを軽減することによりソフトウェア検証プロセスの工数が削減されていることが分かる。

また、各開発の検証項目数、不具合件数、不具合修正時間に着目すると、レガシー開発とモデ

ルベース開発における全体的な検証項目数、不具合件数に大きな違いはないが、修正時間の合計を見ると、モデルベース開発で実施された修正時間の工数が少ないことが分かる。このことから、モデルベース開発で実施された検証は、レガシー開発と同等のものであり、その品質を保ったまま開発に掛かる工数を削減できていると言える。

７．１．５　　モデルベース開発初期導入に掛かる工数

モデルベース開発を導入するにあたり、開発者に対してモデルベース開発に必要な教育を行う必要がある。そこで、表 21 のような教育計画を立てて実施した。

表 21 教育計画

教育項目	項目	内容
モデルベース開発基礎教育	MATLAB/Simulink 習得	開発ツールである The Mathworks 社の MATLAB/Simulink の習得（E-learning/書籍学習/社外講習）
	モデリング規約把握	モデル記述規約である JMAAB(Japan MATLAB Automotive Advisory Board)制定のモデリングガイドラインの把握
モデルベース開発実践	モデル設計/作成	モデリング技術の習得（パイロット JOB/有識者による指導）
	モデル検証/チューニング	
	M-ファイルコード作成	

教育の内容は主にモデルベース開発時に使用するツール MATLAB/Simulink の習得が主である。まず開発者は教育項目「基礎教育」で MATLAB/Simulink のツールの利用方法を学習する。その後教育項目「実践」で実際にモデルベース開発の V 字プロセスを、有識者の指導の下、実施する。

教育計画に対する工数発生の考え方として、「実践」については OJT で対応するため、モデルベース開発の初期導入工数としては考えない。「基礎教育」については、開発に取り掛かる前段階で実施するため、工数が別途必要である。

「基礎教育」にある E-learning/書籍学習/社外講習の計画について、それぞれの内容と実施工数(人日)を示す。

- E-learning

The MathWorks が提供している無料の E-learning（オンデマンド Web セミナー）により、MATLAB/Simulink、モデルベースデザイン、制御系設計の概要を学び、今後の習得が必要な技術の方向性を把握する。実施工数は 1.5 人日である。

- 書籍学習

社外研修の代替として、MATLAB/Simulink の操作方法や開発方法を書籍より習得する。なお、学習の際には MATLAB/Simulink を実際に操作しながら学習を進める。実施工数は 6 人日である。

- 社外講習

The MathWorks 主催のトレーニングにより、MATLAB/Simulink の使用方法基礎、制御設計方法など応用方法について習得する。また、社外研修を受けたメンバは、メンバに対して社内教育という形で研修内容を展開していく。実施工数は 5 人日である。

これらの基礎教育の実施工数を合わせたものが、表 13 の導入工数に対応する。人月に換算すると、以下の式から約 0.63 人月掛かることが分かる（1 人月 20 人日として換算）。

1.5+6+5=12.5 人日→0.625 人月≒0.63 人月

本報告書で示したモデルベース開発初期導入教育の工数は、モデルベース開発経験者がメンバにいない環境で実施した結果である。モデルベース開発経験者を増やすことで、今回示した工数をさらに削減することができる可能性がある。

７.２ 適用レベルの妥当性の証明

各実験項目で考察した内容を基に、適用レベルで想定していた内容の妥当性を証明する。

７.２.１ フロントローディングによる効果について

ソフトウェア設計プロセスにモデルベース開発を導入することにより、従来検証プロセスで行っていたシステム検証プロセス～ソフトウェア作成プロセス間の検証を、ソフトウェア設計プロセスに負担させ、設計・検証を繰り返すことによって、ソフトウェア検証プロセスの工数を削減できる。7.1.4項の考察から明らかである。これより、適用レベル 1～3 のソフトウェア検証プロセスへのモデルベース開発導入によるフロントローディングの効果は妥当性があると言える。しかし今回の実験では適合評価プロセスを実施していないため、システム設計プロセスへのモデルベース開発導入による効果を計ることができない。これについては別途検証が必要である。

７.２.２　　ソフトウェア検証プロセスへのモデルベース開発導入について

7.1.1項のソフトウェア検証プロセス全体に掛かった工数と、7.1.4項のソフトウェア検証プロセスの不具合修正に掛かった工数から、ソフトウェア検証の実施に掛かった工数を算出すると、レガシー開発では 1.17 人月、モデルベース開発では 0.47 人月掛かっていることが分かる。これより、モデルベース開発ではレガシー開発に比べてソフトウェア検証の実施工数を 0.7 人月削減できていると言える。よって、適用レベル 1 におけるソフトウェア検証プロセスへのモデルベース導入の想定は妥当であると言える。

７.２.３　　コード自動生成について

7.1.2項の考察から、適用レベル 2･3 でソフトウェア作成プロセスにモデルベース開発を導入することで、直接的な V 字プロセス内の工数削減に繋げることができる。また、6.1節の実験結果から、レガシー開発の検証プロセスでソフトウェア作成プロセスによる不具合が検出されていることが分かる。コード自動生成を実施することによってこの不具合の修正時間も削減できていると言える。よって、適用レベル 3 におけるソフトウェア作成プロセスの想定は妥当であると言える。

７.２.４　　レビュー工数について

適用レベルの想定段階ではレビュー工数について考慮していなかったが、7.1.3項の考察から、モデルベース開発を導入することによって可読性が向上し、また、ソフトウェア作成プロセスでのコードレビューが不要になるため、レビュー工数削減にも寄与できることが分かった。

７.３ 適用レベルの工数削減率

6.3節の適用レベル 3 の実験結果と、4.2節の各適用レベルの工数削減効果の想定から、適用レベル毎の工数削減率を考察する。なお、適用レベル 3 の実験結果を基に適用レベル 0～2 の工数削減率について考察するため、適用レベル 3 から順に示していく。

７.３.１　　適用レベル３

6.3節の適用レベル 3 の実験結果から、適用レベル 3 の工数削減率を表 22 に示す。

表 22 適用レベル 3 の工数削減率

※ ○は適用、×は非適用 適合・評価プロセスは実験で実施していないため － とする

適用レベル	適用プロセス		評価(削減率)	
L3	システム設計	○	−3%	全体
	ソフトウェア設計	○	−5%	32%
	ソフトウェア作成	○	28%	
	ソフトウェア検証	○	12%	
	適合・評価	−	−	

７.３.２ 適用レベル２

6.3節の適用レベル 3 の実験結果から、4.2.3項の想定を基に、適用レベル 2 の工数削減率を表 23 に示す。適用レベル 2 では、ソフトウェア作成プロセスにモデルベース開発を導入していないため、ソフトウェア作成プロセスの工数削減効率は 0%になる。適用レベル 2 のソフトウェア検証プロセスにおける、ソフトウェア設計プロセスにモデルベース開発を導入したことによるフロントローディング効果と、ソフトウェア検証プロセスにモデルベース開発を導入したことによる工数削減効果は適用レベル 3 と同様の効果を期待できるため、工数削減率を 12%とした。これより、この適用レベル 2 の全体工数削減率は 4%となる。

表 23 適用レベル 2 の工数削減率

※ ○は適用、×は非適用 適合・評価プロセスは実験で実施していないため － とする

適用レベル	適用プロセス		評価(削減率)	
L2	システム設計	○	−3%	全体
	ソフトウェア設計	○	−5%	4%
	ソフトウェア作成	×	0%	
	ソフトウェア検証	○	12%	
	適合・評価	−	−	

７.３.３ 適用レベル１

6.3節の適用レベル 3 の実験結果から、4.2.2項の想定を基に、適用レベル 1 の工数削減率を表 24 に示す。それぞれシステム設計・ソフトウェア設計・ソフトウェア検証プロセスにモデルベース開発を導入した場合の効果を示している。

システム設計プロセスにモデルベース開発を導入した場合、ソフトウェア設計～ソフトウェア検証プロセスでの工数削減効果は表れないため、システム設計プロセス以外の工数削減率は 0%となる。これより、この場合の全体工数削減率は−3%となる。

ソフトウェア設計プロセスにモデルベース開発を導入した場合、ソフトウェア検証プロセスの工数削減効果は、ソフトウェア設計プロセスにモデルベース開発を導入したことによるフロントローディングの効果だけであるため、適用レベル 3 のソフトウェア検証プロセスの工数削減率よりも少ないと考えることができる。このため、モデルベース開発を導入しているため少なくとも 0%以上、適用レベル 3 のソフトウェア検証プロセスの削減率 12%未満の効果が期待できるとした。これより、この場合の全体工数削減率は-5%以上 7%未満となる。

ソフトウェア検証プロセスにモデルベース開発を導入した場合、ソフトウェア検証プロセスの工数削減効果は、ソフトウェア検証プロセスにモデルベース開発を導入したことによる削減効果だけであるため、ソフトウェア設計プロセスにモデルベース開発を導入した場合と同様に考えることができる。このため、ソフトウェア検証プロセスの工数削減率を 0%以上 12%未満とした。これより、この場合の全体工数削減率は 0%以上 12%未満となる。

表 24 適用レベル 1 の工数削減率

※ ○は適用、×は非適用 適合・評価プロセスは実験で実施していないため - とする

適用レベル	適用プロセス		評価(削減率)	
L1	システム設計	○	-3%	全体
	ソフトウェア設計	×	0%	-3%
	ソフトウェア作成	×	0%	
	ソフトウェア検証	×	0%	
	適合・評価	-	-	
L1	システム設計	×	0%	全体
	ソフトウェア設計	○	-5%	-5%以上 7%未満
	ソフトウェア作成	×	0%	
	ソフトウェア検証	×	0%以上 12%未満	
	適合・評価	-	-	
L1	システム設計	×	0%	全体
	ソフトウェア設計	×	0%	0%以上 12%未満
	ソフトウェア作成	×	0%	
	ソフトウェア検証	○	0%以上 12%未満	
	適合・評価	-	-	

7.3.4 適用レベル 0

適用レベル 0 はモデルベース開発を導入せず、開発に掛かる工数は今回の実験のレガシー開発と同じであるため、工数削減率は表 25 のようになる。

表 25 適用レベル 0 の工数削減率

※ ○は適用、×は非適用　適合・評価プロセスは実験で実施していないため － とする

適用レベル	適用プロセス		評価	
L0	システム設計	×	0%	0%
	ソフトウェア設計	×	0%	
	ソフトウェア作成	×	0%	
	ソフトウェア検証	×	0%	
	適合・評価	－	－	

７.３.５　　適用レベルの全体工数削減率

7.3.1項から7.3.4項の考察より、適用レベルごとの全体工数削減率を表 26 にまとめる。なお、適用レベル 1 の工数削減率は、システム設計・ソフトウェア設計・ソフトウェア検証プロセスにモデルベース開発を導入した場合のプロセス全体の工数削減率を比較した最小値と最大値を採用した。

表 26 適用レベルの全体工数削減率

適用レベル	評価
L0	0%
L1	−5%以上 12%未満
L2	4%
L3	32%

７.４ まとめ

今回の実験では、1回のＶ字プロセス実施におけるレガシー開発とモデルベース開発の工数削減効果の妥当性を示すことができた。しかし、今回実験対象とした開発ではＶ字プロセスを繰り返し実施していないため、V 字プロセスを継続して実施した場合の品質効果は別途検証する必要がある。また、システム設計プロセスにモデルベースを導入したことによる検証プロセスでの効果を調査するために、実機を用いた検証を行う開発での実験が必要である。

８．モデルベース開発導入による工数以外の効果

7章までは、工数に着目してモデルベース開発を導入したことによる効果を述べてきたが、工数以外にもレガシー開発と比較したときにメリットとして考えられる点を挙げる。

８.１　モデル作成による保守性の強化

モデルベース開発で使用するMATLAB/Simulinkツールでの設計は、そのモデリングという作成方法から必然的に構造化された設計かつ視覚的に分かり易い設計になる。

このため機能の修正・追加が発生した場合、修正箇所の特定やレビューも効率よく実施でき、その修正範囲もコンパクトに収めることができると考える。

８.２　開発における管理作業の軽減

設計後にソースコードの自動生成を使うことで、ソースコード作成や詳細設計書の作成が不要となるといった作業工数が削減されることに加え、ソフトウェア設計以降の管理作業が軽減される。1 つは不要となった詳細設計書やモデルから自動生成されるソースコードを構成管理の対象からはずすことができ、その上それらのトレーサビリティを管理することが不要となる。

またモデルのファイルはソースコードと違い、複数のファイルに分割しなくとも 1 ファイル内に複数のブロックを階層的に収めることができるため、ソースコードの管理に比べ大幅に管理しやすくなると言える。

８.３　開発成果物の均一化

モデリングというその手法の特徴から、必然的に構造化された設計になることを述べたが、それにともなって成果物の内容が均一になると言える。レガシー開発で使用する C 言語等では、モデルより自由度が高い設計が可能で、その成果物の内容は開発者に依存する度合いが高いと言える。品質という観点でも、モデルベース開発はそのバラつきが抑えられると言える。

９．モデルベース開発定着後の効果

モデルベース開発を繰り返し行うことで、導入時の効果に加えて、モデル資源が増えたことによりさらに効果的な活用を考えることができる。

９.１　プラントモデル再利用における効果

本報告書の実験対象である半導体製造装置用制御コントローラ（ボンディング装置用）の制御対象モータのようなプラントのモデルを IP（Intellectual Property）として作成しておくことによって、新たな制御コントローラ開発を行う場合に制御対象モデルを再利用する。これによって、本報告書の実験結果ではレガシー開発に比較して増加してしまったシステム設計工数において、プラントモデルの作成/検証工数を削減することができる。モデル IP 導入後のプロセスイメージを図 36 のモデルベース開発プロセス（導入時）に示す。

９.２　モデル再利用における効果

9.1節でも述べたように似たような機能をもつ製品を繰返し開発するケースにおいては、それぞれで共通となる機能が存在し、かつその割合が高いことが見込まれる。このような場合は、初期導入時に作成した実績あるモデルを活用することができ、作成及び検証の期間を短縮することが可能となる。

またそれらの機能がシステムドメインに非依存な機能であれば、様々な分野の開発に利用することができ、それらシステムの開発効率を向上させることができる。

以下、プラントモデル再利用及びモデル再利用における効果のイメージを示す。

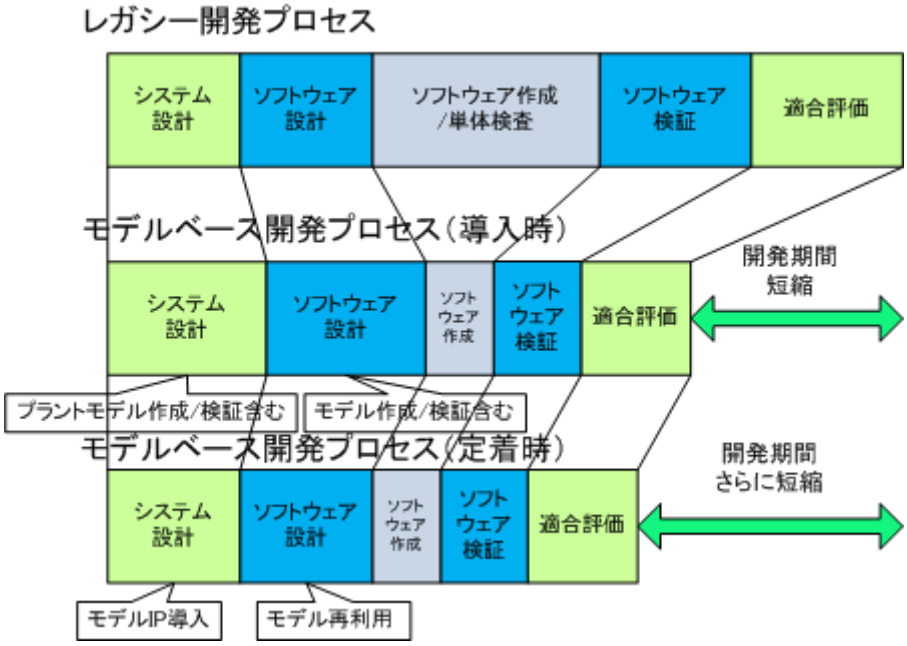

図 36　モデルベース開発定着後のプロセス

１０．　参考文献

- 2010 年版組込みソフトウェア産業実態調査報告書
- 2008 年版組込みソフトウェア産業実態調査報告書
- 独立行政法人　情報処理推進機構　技術本部
  ソフトウェア・エンジニアリング・センター　編著
  「組込みソフトウェア開発向け品質作り込みガイド」