ソフトウェアテストシンポジウム 2004
JaSST'04: Japan Symposium on Software Testing 2004

組み込み商品群におけるソフトウェアの妥当性確認

－限られた時間の中で効率的に組み込みソフトウェアの妥当性確認を実施する－

酒井 由夫†

†組込みソフトウェア管理者・技術者育成研究会(SESSAME)

〒113-8656 東京都文京区本郷 7-3-1 東京大学大学院 工学系研究科 化学システム工学専攻 飯塚研究室
E-mail: †xx_sakai@d3.dion.ne.jp

あらまし 日本の産業が競争力を保ち続けるには，高付加価値・高信頼性を持つ組み込み商品の創出が必要である．特に短期間で効率的な妥当性確認が重要となる．本論文では，まず組み込み機器を商品群（プロダクトライン）と捉え，安全性や信頼性をリスク分析により確保することを提案する．次に組み込みソフトウェアのコアとなる部分とバリエーションを実現する部分を明確に分離し，異なるテスト手法を適用するというアプローチを提案する．

キーワード 妥当性確認，検証，テスト，組み込みソフトウェア，プロダクトライン

Validation method for embedded software product line

Yoshio SAKAI†

† Society of Embedded Software Skill Acquisition for Managers and Engineers (SESSAME)

Building 5 of the Faculty of Engineering (Hongo Campus) 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656 Japan
E-mail: †xx_sakai@d3.dion.ne.jp

Abstract Effective and efficient validation is the most key technology. This paper presents a risk-based validation method focusing on the product line methodology discussed by SEI/CMU. The author also proposes two approaches to validate embedded software. One is to separate core components and replaceable components. The other is to apply different validation method to them.

Keyword Validation , Verification, Test, Embedded software, Product line

1. 組み込み機器の効率的な妥当性確認の方法

1.1. 組み込みソフトウェアを取り巻く環境

日本のお家芸である組み込み分野においても，アジアの国々がその勢力を伸ばしつつある．そのような環境の中で日本の組み込み産業が目指すべき道は高付加価値・高信頼性の組み込み商品をプロデュースするということであろう．コストダウンだけを目指したのではアジアの諸外国との値引き合戦になってしまう．高付加価値・高信頼性を保ちながら安値感のある商品こそがヒットする時代であり，日本の組み込み産業の競争力を保ち続ける鍵であると考える．

高付加価値・高信頼性の商品を実現するためには，商品に搭載する組み込みソフトウェアが高付加価値・高信頼性を有していることが重要な要件となる．当然のことながら高付加価値・高信頼性のソフトウェアを作成するためには，プロジェクトに参加するエンジニアには高度な分析能力，実装能力，検証能力が要求され，必然的に費用と時間も通常の開発より多くかかる．

しかし，開発費用の増加は許容できても，開発期間の延長による商品の市場への投入の遅れは致命的な打撃を受けないとも限らない．したがって，我々は限られた時間の中で高付加価値・高信頼性を持つソフトウェアを生み出す方法を考えなければならない．

そこで，本論文では組み込み機器を商品群ととらえ，まず，組み込み機器商品群に対する安全性や信頼性をリスク分析によって確保し，次に組み込みシステムのソフトウェアのコアとなる部分とバリエーションを実現する部分を明確に分離しテストの手法も別々の考え方を適用するという二つのアプローチにより，短期間で効率的な組み込みソフトウェアの妥当性確認を実施する方法について検討する．

1.2. 組み込み機器におけるプロダクトライン戦略

組み込み機器はローエンドからハイエンドまでといった幅広いグレードを持つなど企業内でさまざまな商品群が構成されている場合が多く，機器に組み込まれるソフトウェア以外にも人材や商品開発のプロセスの再利用率が高いという特徴を持っている．商品群のコア資産を共有し，再利用して新しい商品を開発する戦略は**プロダクトライン**と呼ばれている．**プロダクトライン**戦略によって摘出するコア資産は，ソフトウェアの資産だけでなく，商品開発のプロセスや開発環境，ドキュメント，テスト手法など同じ商品群で繰り返し使う知識や手法なども含まれる．

1.3. プロダクトライン戦略に基づく安全性・信頼性の確保

組み込み商品群の安全性・信頼性を確保するという目的で**プロダクトライン**戦略を適用した場合，まず，最も優先度が高く再利用の必要性が求められるのが，商品または商品群に対して行われたリスク分析の結果と対策である．組み込み商品群に対するリスク分析の結果と対策は長い期間マネージメントしていくべき対象であり，組み込み商品群にとって貴重なコア資産である．なぜなら，組み込み商品群に対して長年蓄積されたリスク分析の結果と対策によって，使用するハードウェアデバイスの曖昧さを許容することが可能となり，また，実際にフィールドで発生した不具合や事故の対策をもとに無限に存在するヒューマンエラーに対する効果的な対策を打つことができる．（図１）

リスク分析の次に行うべき作業は**プロダクトライン**戦略に基づいて商品群のソフトウェアをドメイン分析し，ソフトウェアとしてのコア資産を摘出することによって，商品群の中で再利用性の高い変わりにくいドメインと再利用性の低い変わりやすいドメインを分割することである．分割したそれらのドメインに対してのテストアプローチを変えることで，限られた開発期間の中で優先順位をつけた効率の高いソフトウェアの検証作業を実施することができる．

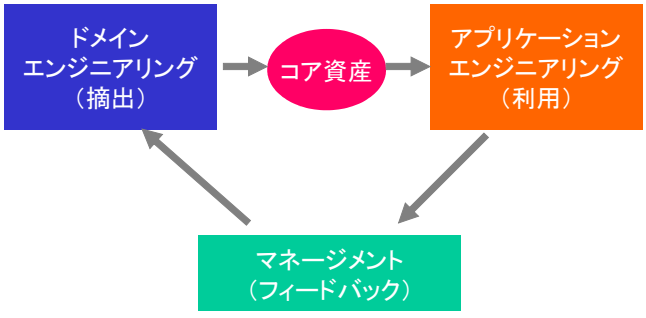

図１　プロダクトラインにおける３つの活動

2. 組み込みソフトウェアに対するリスク分析

2.1. Validation(妥当性確認)と Verification(検証)

組み込み機器の安全性・信頼性の確保を効果的に実施するためには Validation (妥当性確認)と Verification（検証）の違いを明確にしておくことが有効である．

ソフトウェア Verification とは，ソフトウェア開発のライフサイクルのある段階での設計のアウトプットが，その段階における明確な要求事項にすべて適合しているということの客観的な証拠を提供するということである．ソフトウェア Verification では，ソフトウェアが開発される過程における一貫性，完全性，正確性が証明されることと，また，それを補間する文書が存在することが期待されている．そして，その積み重ねの結果が Validation されているという結論に結びつくこととなる．ソフトウェアテストは，ソフトウェア開発のアウトプットが，そのインプット要求事項に適合することを確実にするための検証活動のひとつである．他の検証活動には，さまざまな統計的，動的分析，ソースコードや文書の検査，ウォークスルー，レビューなどが含まれる．（図２）

開発者はソフトウェアの品質を高めるために製品に対するテストを永遠に続けることはできない．いつかは製品をリリースしなければならない．実際には十分にテストを行う間もなく製品を発売する期限が刻一刻と迫ってくるというのが現実であろう．そう考えるとソフトウェアの Validation はどこまで，また，いつまでやればよいのだろうか．現実を考えると求められているユーザー要求に製品が適合しているという「自信」が十分なレベルに達するまでソフトウェア Validation は行う必要があると考えられる．仕様書の中で見つけられた間違いの修正数や，残された不具合の評価，テストカバレッジの結果等は製品を出荷してよいかどうかの確信のレベルを得るために使われる．また，自信のレベルと必要とされるソフトウェア Validation，検証，テスト作業の程度は製品の安全上のリスク（障害）に起因すべきである．想定したリスク（障害）に対して漏れのないように Validation を行う必要があり，また，リスク（障害）がユーザーに与える影響が大きければ大きいほど Validation は慎重かつ確実に行われる必要がある．

商品としての安全性・信頼性を確保するためには，第一にリスク分析の結果による対策を優先させるべきである．

2.2. リスクマネージメントの重要性

組み込みシステムにとってリスク分析の結果と対策は再利用するべき重要な資産である．場合によっては，要求仕様を実現することよりもリスク分析の対策

の方が重要なシステムさえある．組み込みシステムの用途は明確である場合が多く，使いやすければやすいほど人は組み込みシステムに頼ることになり，組み込みシステムは潜在的に高い安全性・信頼性が求められることになる．このような組み込み商品群に対するリスク分析の結果と対策の多くは商品の要求仕様が変化していくこととは対照的に不変的であり代々蓄えられながら継承されていく性質を持っている．組み込み商品群はフィールドで発生した問題に対して対策を行いながら安全性や信頼性を向上させていくといっても過言ではない．

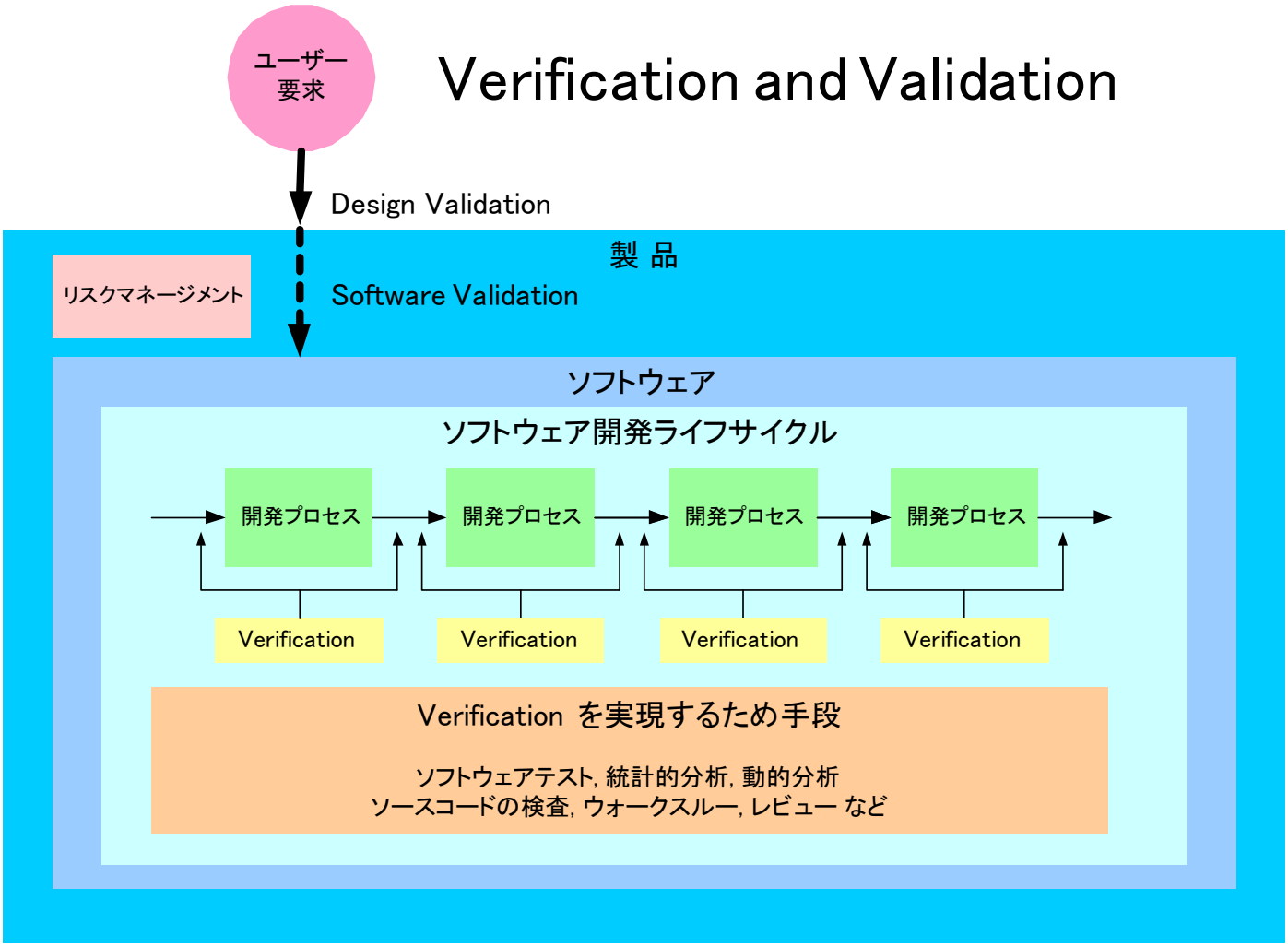

図２　Verification（検証）と Validation（妥当性確認）の関係

2.3. リスク分析の方法

リスク分析の内容は，一部要求仕様書に書かれている場合もあるが，安全性や信頼性の観点から設計者以外のメンバーを交えてレビューを行いできるだけたくさんのリスク（障害）の可能性を検討し対策を講じる必要がある．また，このようにして構築したリスク分析と対策のノウハウは単一機種だけでなく，商品群として企業の製品全体に水平展開すべきコア資産であることから，データベース化することが望ましい．データベース化されたリスク分析のノウハウには，フィールドで発生した障害に対する対策を追加したり，より安全性・信頼性が高くなるような対策への修正を行ったりしながら徐々に充実させていく．

表１にリスク分析の結果と対策をまとめるためのリスク分析表を紹介する．このリスク分析表はもともとアメリカの FDA（Food and Drug Administration 食品医薬品局）が輸血用の血液型を自動判別装置に対してリスク分析した例を電子ポットのリスク分析に適用したものである．

表１　電子ポットのリスク分析表

番号	障害	原因	重要度	発生の可能性／故障率	対策	実施確認の方法	チェック
No.	Hazard	Cause	Level of Concern	Likelihood/ Failure Rate	Method of Control	Trace	Check
A-1	ヒータの異常加熱により火災が起こる	・サーミスタの故障 ・ヒータの故障	High	1/10000 （故障率）	**・ハードウェアによる対策** 温度ヒューズによるヒータへの回路切断 **・ソフトウェアによる対策** ブザーによる注意喚起とエラー表示（３０秒）を行う.	設計書番号 #001 テスト計画 #001	□ □
		・水の量が少ないに加熱した	High	たまにあり (Moderate)	**・ソフトウェアによる対策** 第１水位センサがオフ状態ならば，ヒータや沸騰ボタンは動作しない.	設計書番号 #002 テスト計画 #002	□ □
A-2	ポット内の水温が上がらないのでお湯が使えない	・ヒータが動作しない ・ヒータの動作が不安定	Low	まれ(Low)	**・ソフトウェアによる対策** ヒータ制御中に１分周期で水温を検出し，目標温度よりも水温が５℃下がり，かつ前回検出した水温よりも今回検出した水温の方が低い場合，ヒータ電源をオフにしブザーによる注意喚起とエラー表示（３０秒）を行う.	設計書番号 #003 テスト計画 #003	□ □
A-3	お湯で火傷する	・蓋が開いたのにヒータがオンのままになっている	High	まれ(Low)	**・ソフトウェアによる対策** 蓋センサがアクティブになったら，ヒータを停止し，沸騰ボタンを効かないようにする. ブザーによる注意喚起とエラー表示（３０秒）を行う.	設計書番号 #004 テスト計画 #004	□ □
		・お湯を出すつもりがないのに誤って給湯ボタンを押してしまった	High	たまにあり (Moderate)	**・ソフトウェアによる対策** 給湯はロック解除ボタンを押さないと効かないようにする 給湯してから５分たったら給湯ボタンをロックする.	設計書番号 #005 テスト計画 #005	□ □
		・子どもがいたずらし，給湯してしまった.	High	まれ(Low)	**・ソフトウェアによる対策** 給湯はロック解除ボタンを押さないと効かないようにする 給湯してから５分たったら給湯ボタンをロックする. **・ユーザー自身による対策** 「子どもの手の届かないところにポットを設置する」ように取り扱い説明書に記載し注意を促す.	設計書番号 #006 テスト計画 #006	□ □
A-4	蒸気で火傷する	・ポットの蓋を開けるときに蒸気で火傷する	High	たまにあり (Moderate)	**・ユーザー自身による対策** 「上蓋を開けるときは，蒸気にご注意ください」のラベルを上蓋付近に貼って注意を促す	設計書番号 #007 テスト計画 #007	□ □
A-4	ポットの中で雑菌が繁殖しお腹をこわす	・保温設定で６０℃を選択したため耐熱性の雑菌が繁殖した	High	まれ(Low)	**・ソフトウェアによる対策** 保温モードで９０℃と６０℃を選んだときも一度沸騰させてから目標温度まで冷やす.	設計書番号 #008 テスト計画 #008	□ □

2.4. リスク分析表の書き方

リスク分析表は想定されるリスク（障害）がどのような原因で発生するのかを分析し，そのリスク（障害）がユーザーに及ぼす危険の重要度と頻度を割り出し，リスク（障害）に対する対策を立て，対策がきちんと行われたかどうかを確認するために用いる．

- 障害→操作者が被る障害の内容
- 原因→障害を引き起こす原因
- 重要度→障害の重要度
- 発生の可能性または故障率→部品の故障率のように故障率がはっきり分かっている場合は，故障率をそのまま書き，人為的ミスのような発生の確率がはっきりしない場合には，「たまにあり」「まれ」などといった表現で発生の可能性を記入する．
- 対策→対策はハードウェアによる対策やソフトウェアのよる対策，または，取り扱い説明書や注意ラベルによってユーザー自身に注意を喚起し，ユーザー自身による対策のいずれかの分類と具体的な方法と書く．
- 実施確認の方法→設計書の番号またはテスト計画のドキュメント番号
- チェック→実施確認を行ったチェック

2.5. リスク分析表の効果

リスク分析表またはリスク分析データベースをマネージメントすることによって，商品の安全性・信頼性は効率よく確保されることになる．また，リスク分析の結果や対策の内容を分類し，商品群共通のリスク分析項目を拾い上げ水平展開することで商品群全体としての安全性・信頼性を向上することが可能となる．

3. 組み込みシステムのテスト計画

3.1. 組み込み商品群に対するテストの問題点

組込みシステムによってはハードウェアを制御する部分がいたる所に存在する場合がある．ハードウェアをさわる部分をドライバとしアプリケーション層と分離しても，アプリケーション層をテストするときにドライバを呼んでいれば結局はハードウェア制御へたどり着いてしまうので，ハードウェアのスタブを作らないとアプリケーション層のテストができない．このような状況から組み込みシステムのソフトウェアをパソコンのアプリケーションソフトのように全体をおしなべて均一にテストしようとすることは難しい．組み込みシステムのソフトウェア全体をパソコンでシミュレーションしようとした場合，リアルタイムＯＳやハードウェアをシミュレーションしなければならず，このシミュレーションの部分を作成するだけで開発工数の多くを使わざるを得ない．

3.2. 組み込みシステムソフトウェアのドメイン分割

そこで，本研究では組み込み商品群のシステムソフトウェアを「再利用可能なコア資産」と「商品のバリエーションを構成する部分」に分けてテストを行う際のアプローチを変えることを検討する．（図３）

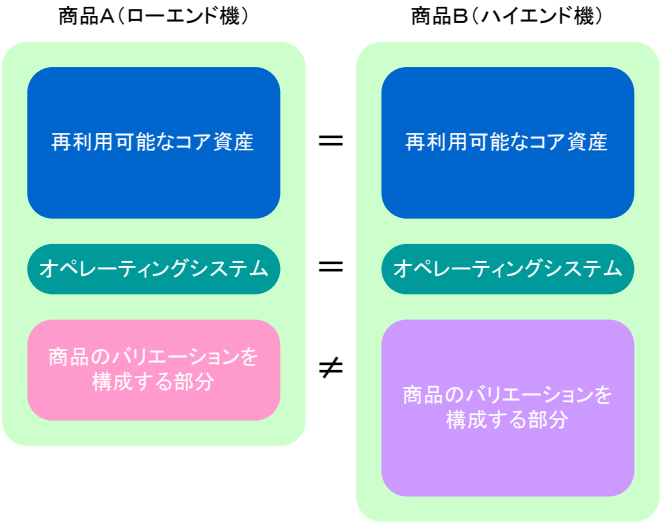

図３　組み込み商品群の内部構成例

具体的には，以下のような手順で組み込みシステムに対するドメイン分析と行い，コア資産となるドメインと商品のバリエーションを構成する部分のドメイン（受動的なドメイン，ハードウェアの制御を含むドメイン）を分類する．

1. 組込みシステムをドメイン分割する
2. ドメインに分割したときに，各ドメインの依存関係が最小になるようによく精査する．
3. 特にハードウェアと関わる部分はアプリケーション層と違うドメインになるように心がける．
4. コア資産となりうるドメイン群と，パッシブなドメイン群，ハードウェアの制御を含むドメイン群を分類する．

3.3. コア資産に対するテストアプローチ

ソフトウェアコア資産としての重要性や再利用性，コア資産に含まれるアルゴリズムをさらにアップグレードする可能性を考えると，ハードウェアデバイスの制御やソフトウェアのアルゴリズムをシミュレーションによって検証する環境を整えておく必要がある．ハードウェアデバイスの置き換えが多数必要な場合は，シミュレーション環境を構築するときに多くの労力を

必要とするが，コア資産を商品群として再利用することを考えれば，構築したシミュレーション環境も再利用することになり，２回目，３回目の開発では開発の初期段階で商品の心臓部を検証することが可能となる．

したがって，ソフトウェアコア資産に対しては，OSのラッピングを行ったり，ハードウェアデバイスのスタブの作成あるいはハードウェア制御部分の一部入れ替え等を行ったりして，コア資産をシミュレーション環境でテストできるようにする．（図４）

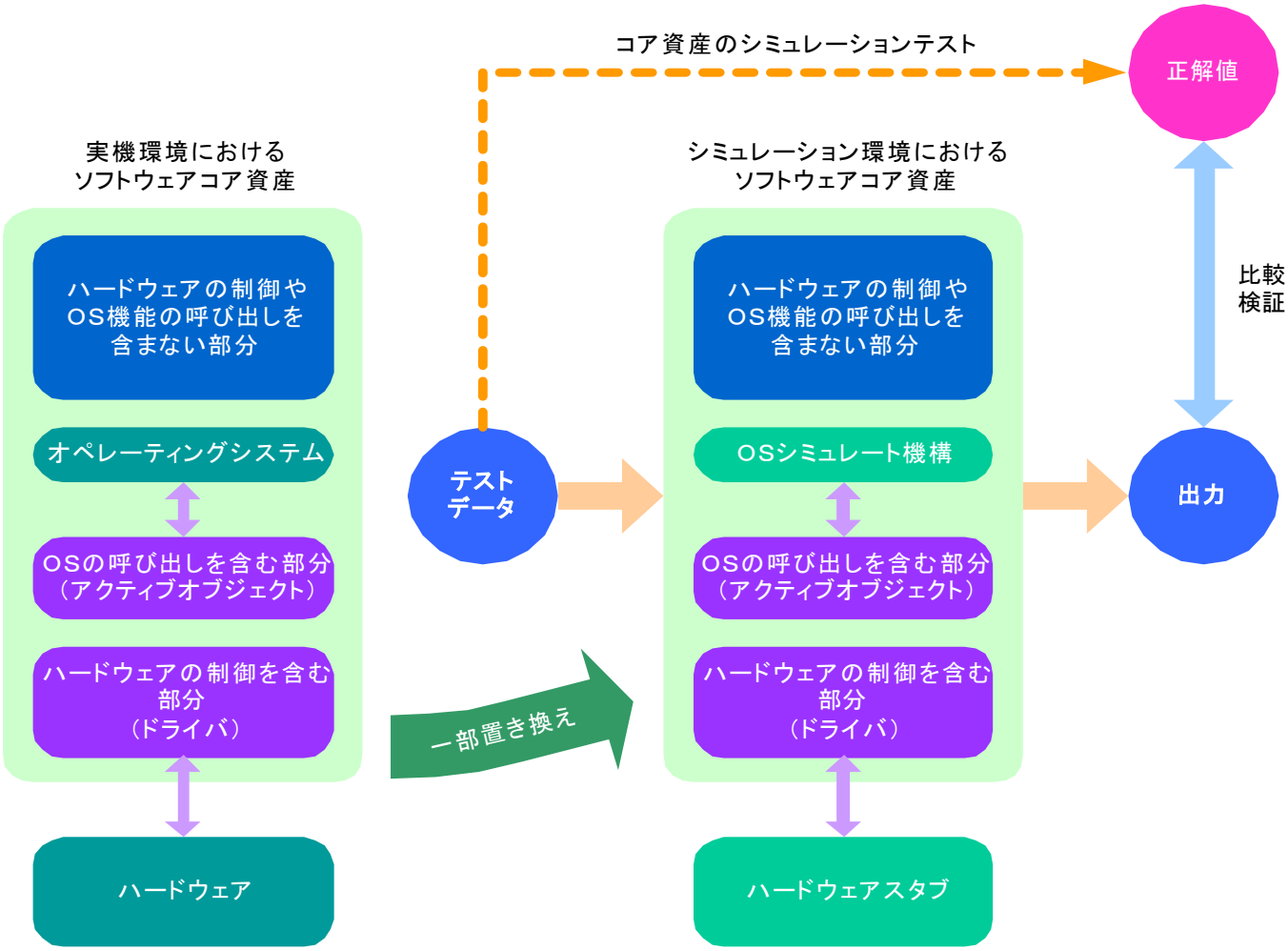

図４　組み込み商品群におけるコア資産のシミュレーションテスト

3.4. 受動的なオブジェクトに対するテストアプローチ

パッシブなオブジェクト（ソフトウェアモジュール)は,パソコンのアプリケーションソフトと同様に,繰り返し型のテストアプローチを取ることができる．少しの修正を行っただけでもシミュレーション環境によるテストが容易に実施できるので，xUnit などのテストフレームワークを使うことも可能であり，また，XP(eXtreme Programming)で提唱されているようなテストファーストの設計も試みることも有効である．（図５）

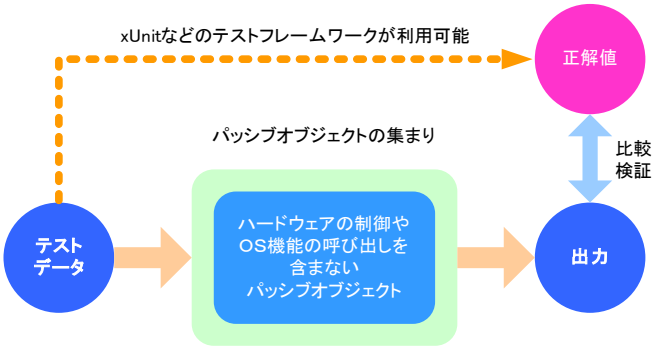

図５　パッシブなオブジェクトのテスト

3.5. ハードウェアを含む部分のテストアプローチ

コア資産以外のハードウェアを含む部分に対しては，基本的にはシミュレーションを行わずにターゲットボードとＩＣＥ（インサーキットエミュレータ）等を使った実機環境でテストを行うことを考える．（図６）

コア資産以外のハードウェアを含む部分としては，ＧＵＩ（グラフィック・ユーザー・インターフェース）を含んだ操作系のユーザーインターフェースなども想定される．専用のキーによる操作や液晶画面等への表示等の検証は試作のハードウェアが用意されているのであれば，実機環境で動作を繰り返し確認していく方法がシミュレーション環境を構築するよりも早く検証ができる場合がある．

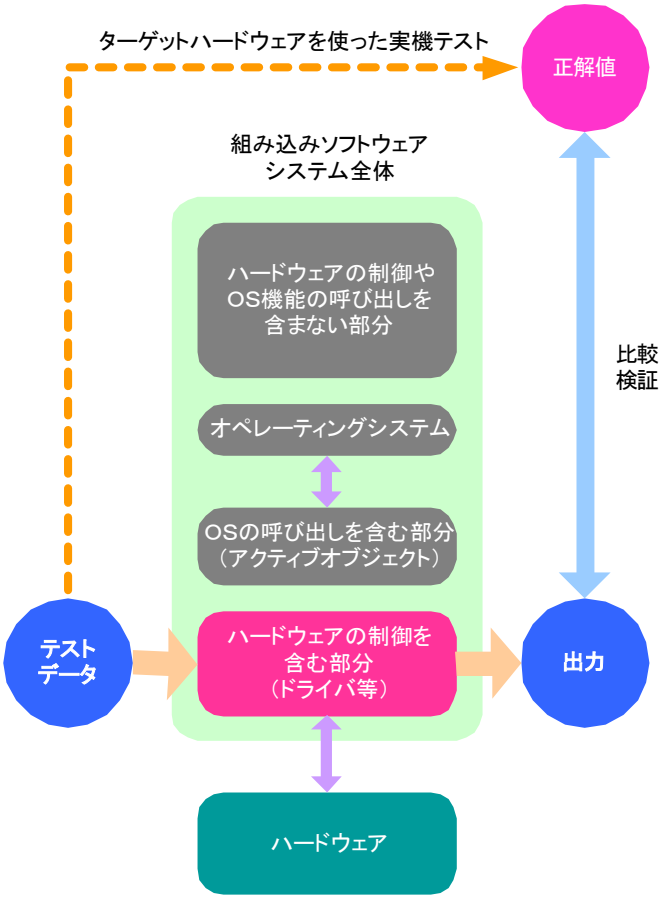

図６　ハードウェアドライバ等のテスト

3.6. 組み込みソフトの単体テストと結合テスト

組み込みシステムの開発は基本的にはＶ字のウォーターフォールモデルの開発工程になることが多い．もちろん，プロトタイピングを行ったり，スパイラルあるいは繰り返し型の設計工程をとったりすることもあるが，商品開発という大きなスコープでソフトウェア全体のライフサイクルプロセスを眺めれば，要求分析・要求定義から始まり徐々に階層を深めるにしたがって仕様が詳細化し，詳細仕様の検証からソフトウェアとハードウェアの結合，システム全体に対する検証へというＶ字の流れは必ず存在する．（図３）

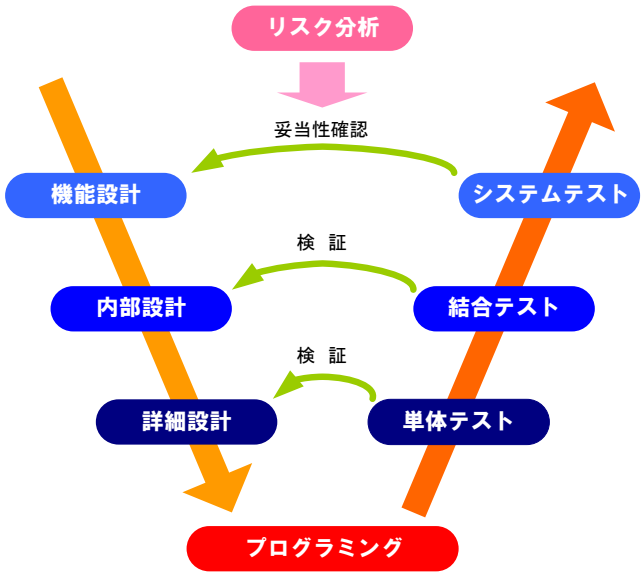

図３　組み込みシステムの開発工程

しかし，これまで述べてきたように組み込みシステム全体に対して均一にテストを行うことは，リアルタイムＯＳやハードウェアの存在があるだけに難しい．そこで，リアルタイムＯＳやハードウェア制御が含まれたソフトウェアに対して実施する単体テスト，結合テストの内容を具体的にどのようにすればよいかを考える．

3.7. 静的テストツールの使用について

組み込みシステムのソフトウェア開発には信頼性の高いソフトウェアを作成したいという希望はあるものの，テストにかける人的工数をできるだけ減らしたいという要望も存在する．テストの工数を削減する方法として，ソースコードの文法的チェックや複雑性を計測する静的テストツールがある．

静的テストツールにソフトウェアを通すということは，テストケースの入力に対する出力と正解値を比較するという方法とは異なるが，文法的に好ましくないソースコードを発見したり，複雑でわかりにくいソフトウェアを事前にスクリーニングしたりするという効果を期待できる．静的テストツールはルーチンワークとして日々のソフトウェア作成作業の間に組み込むことが可能であるため，簡便かつ頻繁に実施できるブラックボックステストの一つととらえると有効である．

3.8. 組み込み単体テスト･結合テストの考え方

これまで，組み込みシステムのソフトウェアを「コア資産」,「パッシブなオブジェクトの集まり」,「ハードウェア制御を含む部分」の３つの対象に分類しテストのアプローチを行う方法を紹介した．

ここからは，Ｖ字のソフトウェアライフサイクルプロセスにおける，単体テスト，結合テストとこのテストアプローチをどう関連づけるのかについて考察する．

静的テストツールを使ったテストは，テストケースの入力に対する出力と正解値を比較検証していないので，厳密には詳細設計に対する検証であるとは言い難い．しかし，単体テスト・結合テストを合わせたステージで詳細設計・内部設計に対する検証を実施すると考えれば静的テストツールを通すという行為も単体テストの一種であると考えることができる．静的テストツールを単体テストととらえた場合，「コア資産」,「パッシブなオブジェクトの集まり」,「ハードウェア制御を含む部分」の３つの対象に対する単体テスト・結合テストの内容を表２のように定義し実践すると最小の工数で組み込みソフトウェアシステムの信頼性を検証できると考えられる．

まず，コア資産に対しては静的テストツールを通す行為を単体テストととらえ，結合テストにおいて，ＯＳやハードウェアをシミュレーションし，その機能を検証する．

次にパッシブなオブジェクトの集まりに対してはテストファーストでプログラミングを行ったり，xUnit のようなテストフレームワークを使ったりして，ソースコードを修正するたびに積極的にテストを実施する．

最後にコア資産ではないハードウェアを含んだソフトウェアに対しては，コア資産と同様に静的テストツールを通す行為を単体テストととらえ，ターゲットハードウェアとＩＣＥ(インサーキットエミュレータ)などで実機上のテストを行うことで信頼性の検証を行う．

表２　単体テスト・結合テストの方針

	単体テスト	結合テスト
コア資産	自動テスト	テストケースを作成
	静的テストツールを利用	機能単位でシミュレーションを実施する
パッシブなオブジェクトの集まり	テストケースを作成	テストケースを作成
	xUnit などのテストフレームワークを利用	テスト項目は基本的なものとする
コア資産でない，ハードウェア制御を含んだドメイン	自動テスト	テストケースを作成
	静的テストツールを利用	実機テストで十分に検証する

工数小（スクリーニング）
工数中（設計検証）　工数大（設計検証）

4. まとめ

組み込み機器を商品群ととらえ，まず，組み込み機器商品群に対する安全性や信頼性をリスク分析によって確保し，次に組み込みシステムのソフトウェアのコアとなる部分とバリエーションを実現する部分を明確に分離しテストの手法も別々の考え方を適用するという二つのアプローチにより，短期間で効率的に組み込みソフトウェアの妥当性確認を実施する方法について論じた．

この手法はシステム全体のソフトウェアを均一なもののして扱わずに，組み込みソフトウェアをその特徴によって分割し，それぞれの特徴にあったテストを実施することで，商品群としての再利用性と，商品としての安全性・信頼性の両方を効率的に確保しようとするものである．テストは優先度をつけながら実施していくため，開発において時間的な余裕ができたならば，さらに深いテストを実施するというアプローチをとることもできる．商品に対する要求を分析し，その特徴に応じた検証作業や妥当性確認を実施することで，限られた時間の中で商品の安全性・信頼性を最大にすることが可能になると考える．

文　献

[1] 酒井由夫他，“具体例で学ぶ組み込みソフトの再利用技術”, Interface，2003 年 12 月号，67-137pp, Nov. 2003.

[2] Paul Clements, Linda Northrop, ソフトウェアプロダクトライン，前田卓雄（訳），日刊工業新聞社，Sep. 2003

[3] General Principles of Software Validation; Final Guidance for Industry and FDA Staff 3.1.2 Verification and Validation, http://www.fda.gov/cdrh/comp/guidance/938.html

[4] Guidance for FDA Reviewers – Pre market Notification Submissions for Automated Testing Instruments Used in Blood Establishments http://www.fda.gov/cber/gdlns/pmaautotest.pdf

[5] 組込みソフトウェア管理者・技術者育成研究会(SEASAME) 話題沸騰ポット要求仕様書 http://www.sessame.jp/workinggroup/WorkingGroup2/POT_Specification.htm