ソフトウェアテストシンポジウム 2005
JaSST'05: Japan Symposium on Software Testing 2005

組込みソフトウェア品質向上のための Activity Mapping
－組込みソフトウェア製品群における効率的な品質向上施策の分析－

酒井 由夫†

†NPO 法人 組込みソフトウェア管理者・技術者育成研究会（SESSAME）

〒103-0007 東京都中央区日本橋浜町 1-8-12-8F(社)日本システムハウス協会内
E-mail: †sakai@sessame.jp

あらまし ソフトウェアの品質向上のための Activity を製品開発のプロセスにマッピングし，組織，プロジェクト，技術者個人といった視点から組込みソフトウェア製品群を眺め，最も効率的で効果の高いソフトウェア品質向上のアプローチ方法を提案する．

キーワード ヒューマンエラー，妥当性確認，検証，プロダクトライン，コア資産，SLCP，技術者教育

QA Activity Mapping for Embedded Software Product Line

Yoshio SAKAI†

† Society of Embedded Software Skill Acquisition for Managers and Engineers (SESSAME)
1-8-12 8F Hamacho, Nihonbashi, Chuo-ku, Tokyo, 103-0007 Japan
E-mail: †sakai@sessame.jp

Abstract This paper presents a method of mapping activities to improve quality assurance on software product line. The author also proposes an effective and efficient approach to achieving strategy to improve quality assurance under the scope of the organization, project and team member.

Keyword Human Error, Validation, Verification, Product Line, Core Asset, SLCP, Engineer Education

1. ソフトウェアの特徴

1.1. ソフトウェアの不具合を作り込む原因

ソフトウェアは技術者の日々の作業の積み重ねで築き上げられる成果であるため，人間のミスが入り込みやすいという特徴を持っている．[1]

ソフトウェア品質向上の Activity は「設計時に不具合の入り込みを少なくする Activity」と「作成した成果物に入り込んだ不具合を摘出する Activity」の２種類に分類できる．この「設計努力」と「摘出努力」は，もともと人間が原因となって作り込む仕様の曖昧さやコミュニケーション不足，あるいは成果物に混入したバグなどを解消するためのアクションである．（図１）

1.2. 過ちを犯しやすい人間の活動を制御する

ソフトウェアの品質向上を目的とした Activity を，過ちを犯しやすい人間の活動をコントロールするという視点で眺め分類すると次のようになる．

- 人間の過ちを軽減する取り組み
  - a) プロセスの定義
  - b) リスク分析
  - c) コーディングルールの定義
  - d) 静的コードテスト
  - e) メトリクス分析による指導
  - f) テスト，レビュー，インスペクション
  - g) ソフトウェア資産の再利用
  - h) 要求仕様の明確化
  - i) 不具合・仕様変更データベースの整備
  - j) UML によるシステム構造の階層化
  - k) 不具合発生・対策曲線の分析

- 人間の介在を少なくする取り組み
  - l) MDD（Model Driven Development）
  - m) ソフトウェア資産の再利用
  - n) 優れたアーキテクチャ，フレームワークの採用

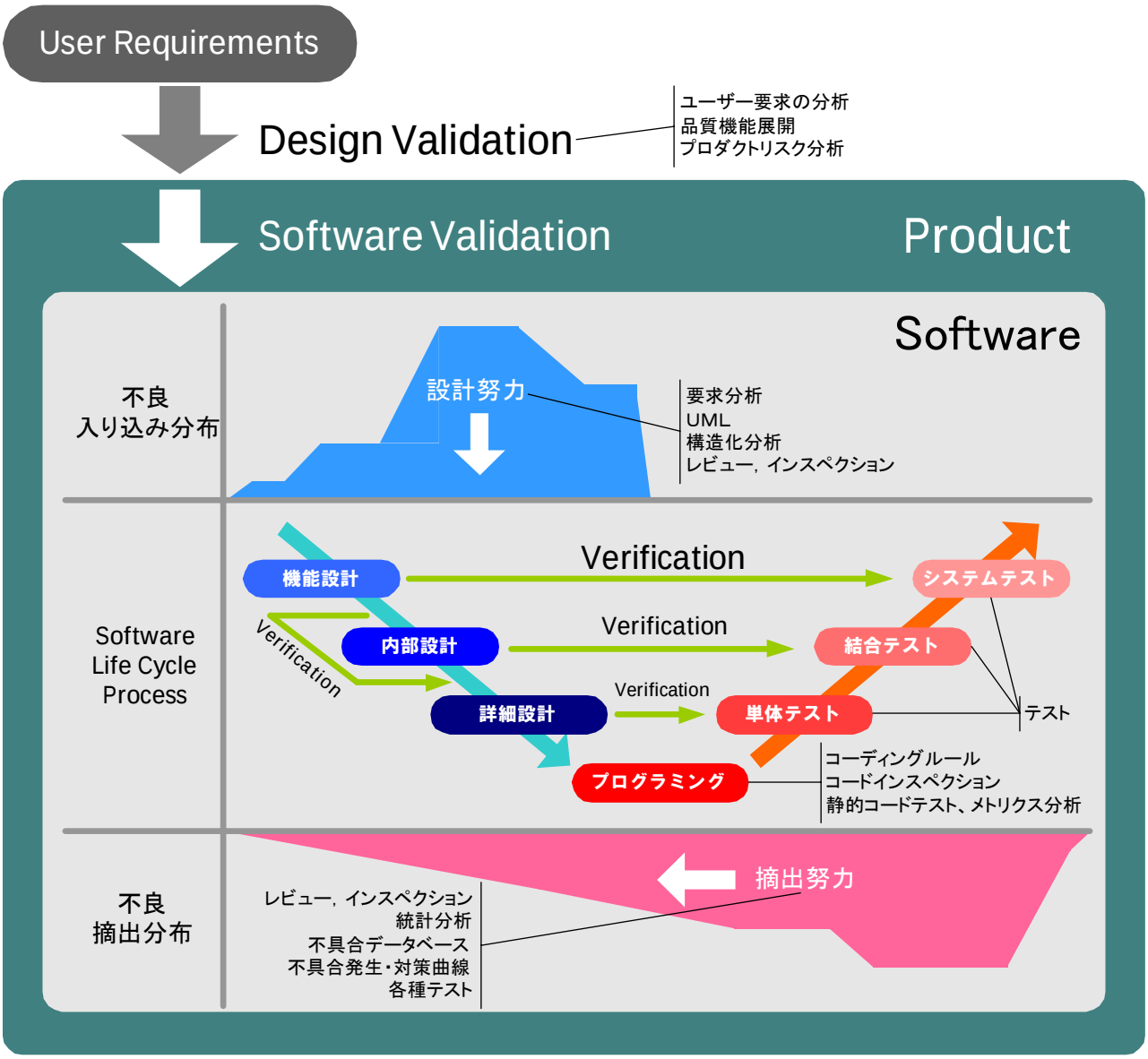


図１ Product と Process，Validation と Verification の関係に配置した品質向上のための Activity Map
（文献[5] P36 図 1.10「不良低減のための方針」をベースに品質向上のための Activity を Mapping したもの）

これらの Activity が組込みソフトウェアの品質向上の活動の中でどのような役割を果たし，また，顧客満足のためにどれくらいの効果を発揮しているのかを把握することが，効果的な品質向上のアプローチにつながる．また，これらの品質向上のための Activity は人間の主観性をできるだけ排除するための活動であると考えることができ，静的コードテストなど，簡便に自動で実施できる Activity や，明確な役割を定め，ルールに従って実施するインスペクションなどの Activity を積極的に採用することによって客観性が高まり，不良を摘出する確率を上げることができる．

2. Validation の必要性

米 FDA(Food and Drug Administration) は 1997 年に Software Validation についての一般原則を公開している．[4]

文献[4]によれば，効果的かつ効率的な組込みソフトウェアの品質向上施策を考える上で，Validation（妥当性確認）の役割は特に重要であると考えられる．なぜなら担当者は Validation の過程で，作り上げた製品がユーザー要求に適合しているかどうかを確認するため，Validation によってユーザーニーズやユーザーの安全性を直接チェックすることができるからである．

また，FDA は Validation（妥当性確認）と Verification

（検証）を分けて考えることも重要であると主張している．Verification とは，ソフトウェアのライフサイクルプロセスにおいて，プロセスのインプットとアウトプットの整合性を確認することである．この検証作業を積み重ねることによって，製品の妥当性が確認されているという確証につながる．

ソフトウェアシステムまたは組込み製品に組み込まれたソフトウェアがユーザー要求に適合しているかどうかということは Verification だけでは完全に確認できるとは言い切れない．たとえば，組込みシステムに利用される RTOS(Real Time Operating System)も，航空，宇宙，自動車，鉄道，医療機器などの機器に組み込まれる場合は，期待される信頼度は非常に高くなる．

このように高い信頼性が要求される機器にも対応できるような高信頼性ソフトウェアモジュールを作成し，これを幅広い用途に使うことは可能である．しかし，高信頼性ソフトウェアを作成するには，当然，コストや工数が高信頼性を求められていないソフトウェアを作成するよりも何倍もかかる．だからといって，高信頼性が要求される機器のすべてが，市販のソフトウェアモジュールを使わないという選択はできないため，実際にはいくつかの商用で即利用可能なソフトウェア部品（COTS: Commercial Off-The-Shelf）を利用することになる．このとき COTS を利用する組織は COTS を実装する製品の用途，ユーザー要求に合わせて Verification（検証）し，安全性・信頼性を確保するために製品全体を Validate しなければならない．

安全性・信頼性を確保する必要性があるからこそ，ユーザー要求を分析した上での Validation は欠かすことができないのである．Validation は製品を開発するプロセスの中で，見落とされた真のユーザーニーズや，安全性や信頼性に関わる危険性を摘出し対策するために役立つ．

3. 品質向上のための Activity のマッピング

ソフトウェアの品質向上のための Activity を組込み製品及び組込みソフトウェアの Validation & Verification，ソフトウェアのライフサイクルプロセス，不良作り込み摘出分布の関係図にマッピングした図が図１となる．図１を見ると，ソフトウェアの品質向上のための Activity がソフトウェアのライフサイクルプロセスの中で，また，組込みソフトウェア製品の Validation & Verification の取り組みの中でどのような位置付けになっているのかがわかる．このソフトウェア品質向上の Activity Map を製品開発のプロセスの中で常に思い浮かべながら，現在実施している Activity について「不足がないか」また「バランスが偏っていないか」を分析する．

4. Product と Sub System の関係

4.1. 高凝集と疎結合の効果

ソフトウェアシステムは，サブシステムの集合体である場合が多い．（図２）

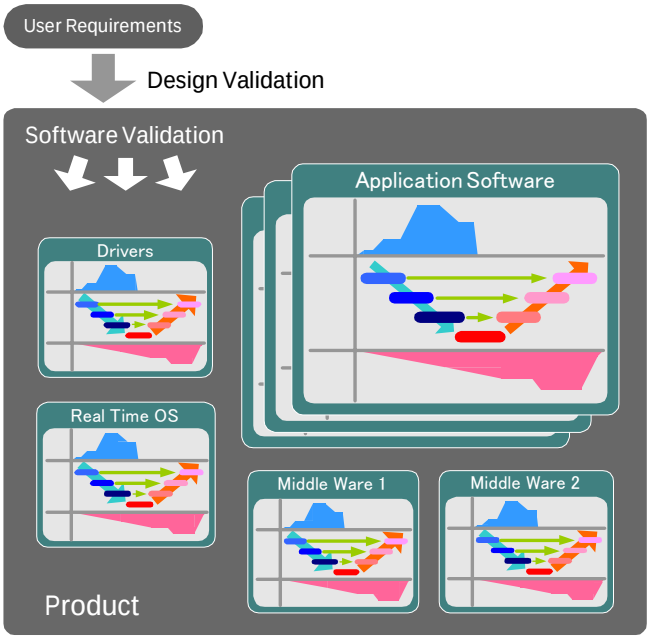


図２ Product の中に存在する Software Subsystem と Validation, Verification の関係

サブシステムの中には，オペレーティングシステムやミドルウェア，ドライバ，ライブラリ，アプリケーションソフトウェアなどがある．それぞれのサブシステムの独立性はできるだけ高く，また，サブシステム間の依存関係が小さくなるように設計することが望ましい．このような高凝集(high cohesion)で疎結合 (low coupling) なサブシステム，ソフトウェアモジュールの設計を実現することができれば，ソフトウェアシステム全体の構造をシンプルにし，機能性，信頼性，使用性，効率性，保守容易性，移植性を向上させることが可能になる．

4.2. Sub System に求められる信頼性の度合い

システムに採用するサブシステムの中で RTOS(Real Time Operating System)などは再利用性が特に高く，システムの中で重要な役割を果たしている．

製品の中に配置されたサブシステムに求められるソフトウェアの信頼性のレベルは，どのような製品に組込むのかによっても異なる．同じ RTOS でも高い信頼性が要求される機器に組み込まれる場合，RTOS をベンダーから購入する際に，その OS が十分に検証されているかどうかを示すための証拠が必要となる．

サブシステムの作成の過程と Verification の内容は，図２のように内部が明らかになっていることが望まし

いが，COTS やレガシーシステムなど作成されたプロセスを明確にすることが期待できない場合はサブシステムの機能と役割から該当のサブシステムをブラックボックスとして検証するしかない．
そして，ブラックボックステストの結果や，採用の実績を分析するなどして，信頼性が確保されたという確信に到達した場合は，このサブシステムを製品群に適用し，再利用する．

5. 組込みシステムと高信頼性ソフトウェアの関係

5.1. テストによる不具合摘出の実際

10 万行を超えるようなソフトウェアに対して，完全な組み合わせテストや 100%のカバレッジを行うことは難しい．また，このような施策はかける費用に対して効果が高いとは言えない．

ツールを使った静的コードテストやメトリクス分析は非常に有効ではあるが，プログラムの論理的な間違いや要求仕様が正しく実装されているかどうかを自動的に検証することはできない．

結局，プログラムの中に人間が作り込む不具合は，それを作ったソフトウェア技術者の”癖”を考慮することなく，効果的に摘出ことは難しいのである．

10 人を超えるプロジェクトチームが作ったソフトウェアでは，限られた時間，限られた工数で作り込まれた不具合のすべてを摘出することはできないと考えた方がよい．逆に組込みシステムが利用される環境においてユーザーに危害や不利益を与えるようなリスクを，重大なものから徐々に低減していくという視点で Validation（妥当性確認）を実施することが高信頼性ソフトウェアの実現につながる．

5.2. 要求仕様定義の限界

ユーザーインターフェースを持ったある程度の規模の組込みソフトウェアにおいてはそもそも 100%正しい要求仕様などはあり得ない．なぜなら，組込みシステムの開発メーカーにおける開発者やステークホルダーが要求仕様のすべてを細部に渡ってチェックし意思統一することはできないし，一方の仕様を優先させたために，もう一方の仕様を削るといったことは組込みシステムでは資源の制約からよくあり，その場合最初に想定した要求仕様のすべてを満たすことはできないからである．

また，開発者やステークホルダーがＯＫを出した仕様であっても製品が市場に出た後で，ユーザーがその仕様に対して製品を購入しないということで「ＮＯ」を突きつけることもある．直接的なクライアントが存在しない組込み製品において，ユーザー要求はメーカー側が想定するしかなく,”正しい”要求仕様というものを定義できないのである．

絶対的な要求仕様の定義者がいないのであれば，現在市場で受け入れられている仕様が新しい製品の仕様にも大きな影響を与える．したがって，製品や製品群の要求品質を機能展開し，基本要件を実現しているソフトウェアサブシステムやモジュールをコア資産として特定し，その寿命を延ばすことが重要となる．

5.3. ソフトウェア再利用の有効性

組込み製品は特定の市場に製品を投入し続けるため，製品群に組み込まれるソフトウェアモジュール，サブシステムは繰り返し再利用される場合が多い．このような特定の市場をターゲットとしたインクリメンタルな繰り返し型の開発では，製品の価値を凝縮した重要なコア資産がいかに”枯れた”モジュールとなっているかどうかが高信頼性のバロメータとなる．

千ページを超えるテストのエビデンスといっしょに提出されたソフトウェアモジュールと，市場ですでにユーザーに使われている製品に組み込まれているソフトウェアモジュールのどちらを採用するかと言われれば，組込みソフトウェアアーキテクトは同じ機能，同じインターフェースを持ったソフトウェアモジュールであれば後者を選ぶのである．

組込みアーキテクトがそのような選択を行う理由は，ソフトウェアを作り込む人間がどのような不具合を作り込むのかを予想することは非常に難しく，市場で大きな不具合を起こすようなバグをテストで優先的に取り除けるという確信がないため，市場で長い間使い込まれ大きなリスクが入り込んでいないことを証明されたモジュールの方に安全性を感じるからである．

組込み製品群のソフトウェアの場合，過去にフィールドで不具合を起こしていない，もしくは不具合対策を積み重ねたソフトウェアモジュールを再利用することが高信頼性ソフトウェアの実現につながっている．また，プロジェクトマネージャはこのような信頼性の高いソフトウェアモジュールを作ったエンジニアを継続的に起用する．逆に言えば，高信頼性ソフトウェアを実現するためには，組込みシステムの市場性や製品群の特徴を考慮し，サブシステムやモジュールを再利用しやすいアーキテクチャを，組込みシステムに採用することが重要だということである．

組込みシステムにおいて「“枯れた”ソフトウェアの再利用」はソフトウェアの信頼性を高めることに対して最も有効に働く．組込みソフトウェアエンジニアは，この「再利用」というカードを場当たり的に使うのではなく，体系的にかつ効果的に使わなければならない．

6. Product Line と Core Asset

6.1. Product Line と開発工数の関係

ソフトウェアプロダクトラインは再利用可能なソフトウェアのコア資産を開発する「ドメインエンジニアリング」，コア資産を使って派生製品を開発する「アプリケーションエンジニアリング」，コア資産を管理する「マネージメント」の３つの活動から成り立つ．[2][3]（図３）

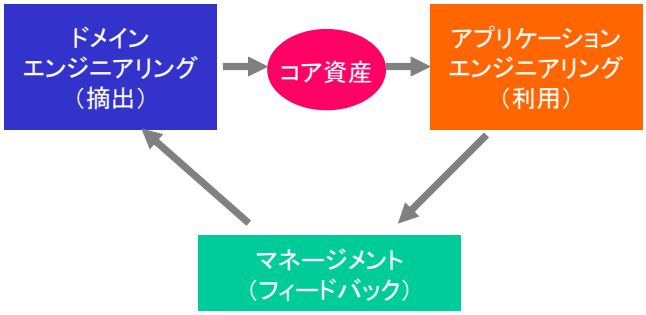


図３　プロダクトラインにおける３つの活動

コア資産は組織内のドメインエンジニアリングの活動から生み出され，投入される市場に特化した製品群の価値が凝縮された重要な資産である．

図２に示した Product は単体の製品であるが，組込み機器の場合，派生開発を行い製品群のラインナップが存在する場合が多い．（図４）

また，組込み製品群のラインナップでは各製品に共通のコア資産が存在することも少なくない．逆に言えば，製品群のラインナップをよく分析し，コア資産を摘出して，コア資産をベースに派生開発を行うと製品を何機種か開発していくうちに，単独で製品を開発するよりもトータルの開発費を下げることができる．（表１）

また，コア資産を他のサブシステムと分離し独立性を高めることで，コア資産の信頼性が高まり，製品群全体の信頼性の向上に貢献することができる

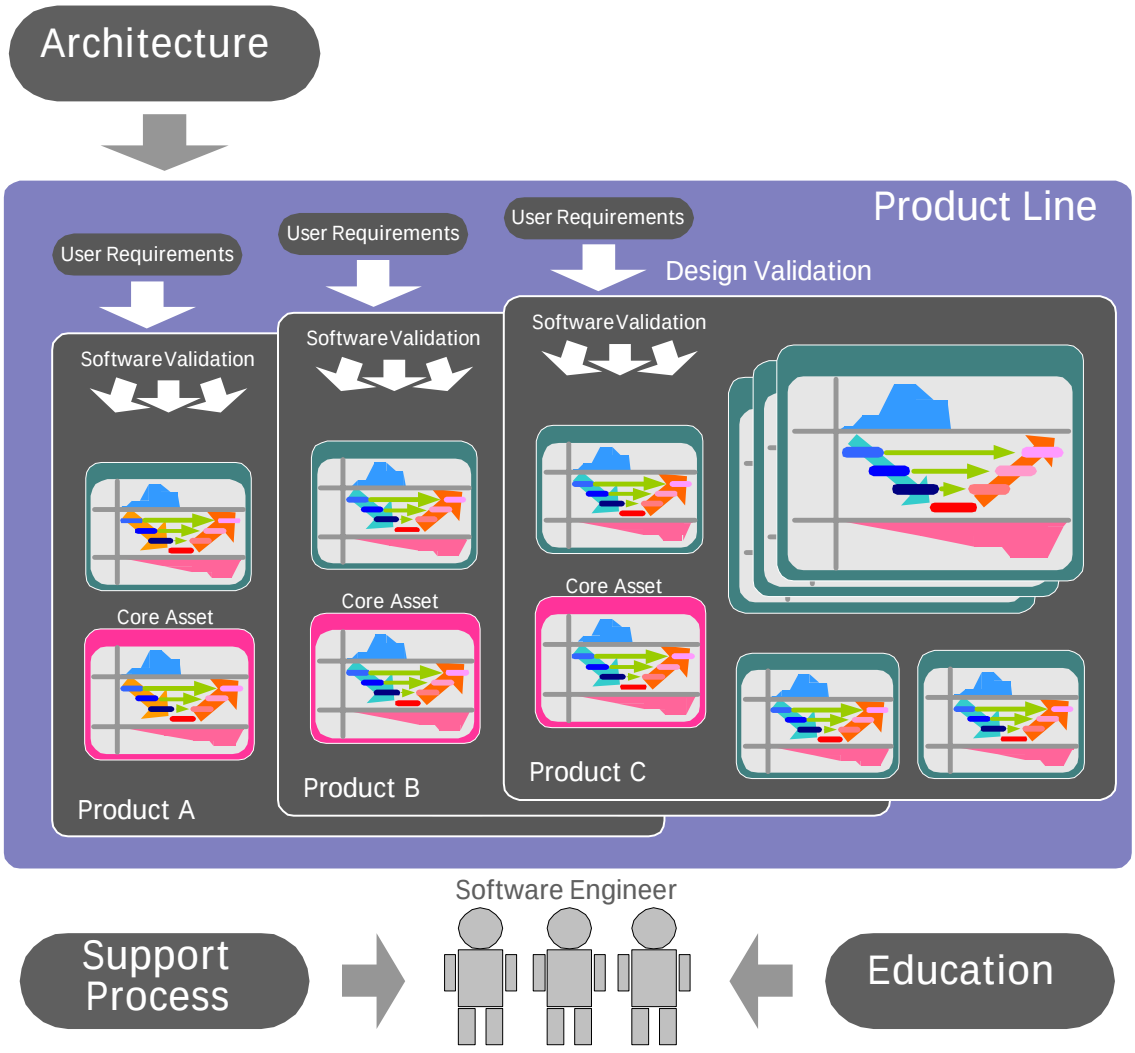


図４　製品群における Product と Core Asset および Architecture，と技術者教育の関係

コア資産を最初に構築するときには，再利用を考慮しないモジュールを構築するときの約1．5倍程度の工数がかかる．しかし，コア資産を３回以上再利用し，事前に分析したリスクへの対策やフィールドで発生した不具合の対策を蓄積させることで，コア資産の開発にかけた工数以上の信頼性の付加価値をコア資産の中に封入することができる．（表２）

表１　プロダクトライン開発の工数の変化の例

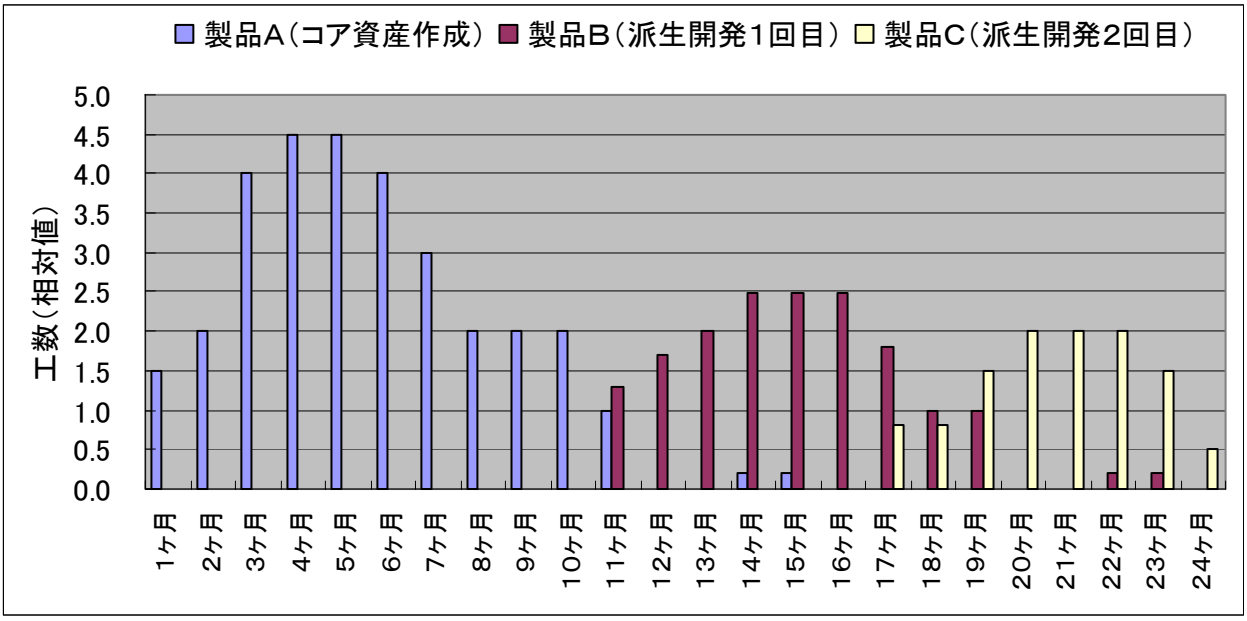
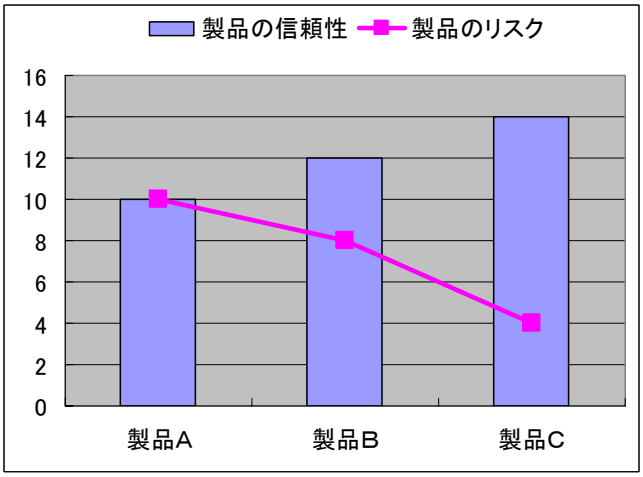


表２　プロダクトライン開発実行時の
信頼性とリスクの変化の例



製品の信頼性に対する要求は多くの場合潜在的なものであるが，不具合の発生により一旦顧客の信用を損うと，その製品のみならず企業全体にダメージを与えかねない．そのため，コア資産の信頼性を向上させることは特に重要度が高い．コア資産を再利用することは，製品の価値を確固たるものにすることが可能になるとともに，製品群全体の信頼性を向上させることにもつながるのである．

7. 品質向上アプローチの例

製品に求められる安全性や信頼性のレベルは製品の種別や投入する市場によって異なる．

また，製品のリコールが企業にとって大きなダメージとなりうる昨今では，安全性や信頼性の向上は潜在的ではあるが製品のアドバンテージとなりうる．

しかし，製品開発にかけられる開発コスト，期間，リソースなどの制約条件は組織の条件やプロジェクトやエンジニア個人の技術力によってもさまざまなケースがある．さらに，同一組織内においてもプロジェクトが異なれば製品の品質を高めるための QA Activity の実力も同じであるとは限らない．

そこで，品質向上のための Activity を組織の成熟度や Activity の実施能力などの視点で評価し，各視点での重要度を分類しマッピングした表を示す．（表３）

7.1. 組織の成熟度レベルでの分類

まず，品質向上に対する組織成熟度のレベルごとに必要な Activity について考察する．

表３　視点別 ＱＡ向上のための Activity Map

必要な施策→ ※括弧内は 1.2. で掲げた「過ちを犯しやすい人間の活動を制御する取り組み」の分類 ↓さまざまな視点	開発プロセスの定義（a）	Validation の実施（b）	エンジニアの教育（a～n）	不具合を作り込まないための Activity（c, h, j, l）	不具合を摘出する Activity（d, e, f）	支援プロセスの整備（l, k）	アーキテクチャの適用（n）	再利用の推進（g, m）
組織成熟度レベル低	要	要	要	C	B	C	C	C
↑	要	要	要	B	B	C	C	C
↓	要	要	要	B	A	B	C	C
	要	要	要	A	A	A	B	B
組織成熟度レベル高	要	要	要	A	A	A	A	A
組織	◎	◎	◎	◎	○	◎	○	○
プロジェクト	○	◎	◎	◎	◎	○	◎	◎
技術者個人	○	○	◎	◎	◎	△	○	○

※「要」は必要性を表す
※A,B,C は要求レベルを表す
※◎○△は優先度を表す

どのようなレベルの組織においても，ソフトウェアが組み込まれた製品をユーザーにリリースし，製品を保守する責任を負うのであれば「開発プロセスの定義」「Validation の実施」「エンジニアの教育」は最低限実行する必要がある．

品質マネージメントシステムの構築には ISO9001 等の国際規格の認証を得ることが最も確実であるが，認証を得なくてもその内容を理解し実践することは有効である．

次に，製品の品質向上に必要な Activity を組織の成熟度のレベルによって習得する目標を定め，徐々に成熟度のレベルを上げていく．品質向上に対する成熟度レベルが低い組織に，いきなり高信頼性ソフトウェアを要求しても作成されたソフトウェアが高品質になる可能性は低い．したがって，組織は自分たちの成熟度レベルを把握し，その成熟度に合った Activity に力を注ぐことが，その時点での総合的かつ効果的な製品の品質向上につながる．

品質向上に対する組織成熟度と，注力すべき Activity の関係図を図５に示す．

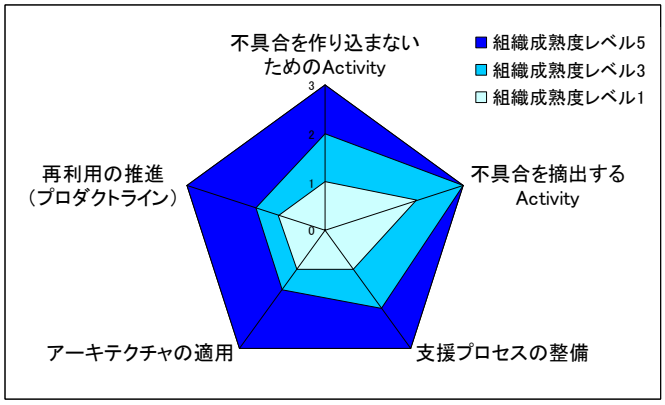


図５ 組織成熟度と注力すべき Activity の関係

品質向上に対する成熟度の低い組織は「開発プロセスの定義」「Validation の実施」「エンジニアの教育」を実施した上で，不具合を摘出する Activity（テスト，レビュー，インスペクション，不具合データベースの構築等）を強化する．その後，組織の品質向上に対する成熟度のレベルが高くなってきたら，不具合を作り込まないための Activity（要求分析，ＵＭＬ，構造化分析など）や支援プロセス（インフラストラクチャの整備，構成管理ツールの導入，不具合評価方法の習得など）の整備に努める．

優れたアーキテクチャの適用や再利用の推進（プロダクトライン）などは，高度な技術を必要とするため，組織の設計能力の成熟度が高くなってから取り組むか，もしくは能力の高いアーキテクトを含めたパイロットプロジェクトを編成し，実験的な製品開発を行って，そこで得たノウハウを組織全体にフィードバックする．

また，成熟度の高い組織が作成したソフトウェアサブシステムを製品に採用し再利用することが，製品の品質を向上させるためには有効である．品質向上に対する成熟度の高い組織・チームがプロダクトライン開発におけるドメインエンジニアリングの活動を行えばさらに効果は高まる．

7.2. 組織，プロジェクト，技術者個人での分類

製品の品質向上に必要な施策は「組織」「プロジェクト」「技術者個人」によって重要度や責任範囲が変化するため，それぞれの立場において何を優先的に実施すればよいのかを認識する必要がある．

例えば，すべてのソフトウェア開発組織において「開発プロセスの定義」や「Validation の実施」，「エンジニアの教育」は組織全体で確実に取り組む．なお，イントラネットやファイルサーバー，企業内ポータルサイトなどのインフラストラクチャ，構成管理ツール，不具合管理データベースなどのツール，統計的不具合

分析手法などの支援プロセスは組織が共通資産また共通技術として用意する.

プロジェクトは，製品開発の中で不具合を作り込まない Activity, 不具合を摘出する Activity を中心に活動を行い，品質を高め不具合が入りにくい高凝集(high cohesion)で疎結合(low coupling)なサブシステムを作成し，プロダクトラインの３つの活動を進め，コア資産の信頼性を高める.

技術者個人は，不具合を作り込まない Activity として設計・分析技術を，また，不具合を摘出する Activity としてコーディングルールやテスト技術，レビュー技術を習得する.（表３）

文献[10]で安富らが提言しているこのような開発者に向けてソリューションマップを示し，組織や立場の違いにより段階的に解決技術を導入する手法は実効効果が高いと考えられる.

7.3. 信頼性要求の高い製品での分類

信頼性要求の高い組込み製品を扱う組織は，組織の成熟度レベルが低い状態で製品をリリースすることは許されない．信頼性要求の高い組込み製品を扱う組織は必要な Activity を実施できる能力を備えてから製品の開発を開始すべきである.

製品に求められる安全性や信頼性のレベルと開発コスト, 期間, リソースの制約条件及び QA Activity を実施する組織の実力は，組織や組織が扱う製品，市場によってさまざまなパターンがあるため，これらの条件をよく分析し，表３「視点別 ＱＡ向上のための Activity Map」を参照しながら，今，重点的に取り組むべき Activity が何かを考えることが効果的である.

8. まとめ

組込み製品群におけるソフトウェアの品質向上には，以下を実施することが有効である.

1. Validation と Verification の意味を理解し，ユーザーニーズ及び製品の信頼性やユーザーの安全に関わる Validation（妥当性確認）の Activity（リスク分析など）を確実に実施する.
2. SQA(Software Quality Assurance)の計画における，Software Life Cycle Process の定義や各種 Activity がソフトウェアの品質向上にどのようにまた，どれくらいの効果を与えているかを分析する.
3. 現在行っているソフトウェア品質向上のための Activity が全体の SQA 活動の中でどの部分に位置づけられているのかを常に把握しながら作業を進める.（図１，図２，図３）
4. 人間のミスを客観的にスクリーニングするための自動テストや，ソースコードの自動生成機能を利用する.
5. ソフトウェアプロダクトラインに基づき，ソフトウェアモジュールやサブシステムの再利用を推進し，コア資産の信頼性を高め，不必要な改変を減らして不具合の入り込みを防ぐ.
6. ソフトウェア技術者に対し，ソフトウェアの品質向上についての考え方と技術を教育する.
7. 自組織の製品の品質向上に対する成熟度のレベルを把握し，そのレベルによって今取り組むべき Activity が何か，また強化すべき Activity が何かを判断し，組織全体として現時点で最も効果の高い Activity を実施する.

なお，プロジェクト単位で組込みソフトウェア品質向上を考える場合，1〜3 を理解した上で，製品の制約条件（開発期間やリソースの制限）を 分析し，4 と 5 を制約条件の中で効率的に実施することが有効である. また，6 の技術者への教育は一朝一夕では達成できないため，長期的な教育計画に基づき継続的に行う必要がある.

文　献

[1] 酒井由夫，“人間の考え方，コンピュータの考え方”，Software People，vol.4，112p-120p，技術評論社, May. 2004

[2] 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, ＣＱ出版社, Nov. 2003.

[3] 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

[4] 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

[5] 保田勝通，ソフトウェア品質保証の考え方と実際，日科技連，1995 年

[6] 松本吉弘，ソフトウェアエンジニアリング基礎知識体系-SWEBOK-，オーム社，2003 年

[7] 酒井由夫，「組み込み商品群におけるソフトウェアの妥当性確認」, JaSST’04: Japan Symposium on Software Testing 2004, Tokyo, Japan, Jan, 2004

[8] JIS TR X 0018:1999 「ソフトウェア・ライフサイクル・プロセスー構成管理」,Japan, 1999

[9] ISO 9001:2000,Quality management systems – Requirements, 2000

[10] 安富大輔, 川井奈央, 今関剛, 渡辺晴美, 佐藤啓太, 「組込みソフトウェア開発技術体系化に基づく技術導入」，組込みソフトウェアシンポジウム 2003, Tokyo, Japan, Nov, 2003

[11] 組込みソフトウェア管理者・技術者育成研究会 (SESSAME) セミナー資料, http://www.sessame.jp/