

15-A-20

組込みソフトウェアのアーキテクチャ設計の可視化¹

1. 適用した技術や手法の概要

組込みソフトウェアの開発は、デジタルカメラやデジタル家電などに代表されるデジタル機器の普及、進化に伴い、急速に大規模化、複雑化、短納期化、多機種化してきている。しかし、組込みソフトウェアの大規模化、複雑化に対応した設計プロセスや設計技法などの適用は不十分なままである。開発リソースの関係から従来の少人数での小規模なソフトウェア開発のプロセスを継続している現場も多く、設計資料等を十分作成せず、ソースコード中心の後工程依存の開発スタイルとなっている場合が多い。また、組込みソフトウェアの短納期化、多機種化に応えるためには開発期間の短縮が必要であり、これに対処するため既存ソフトウェア資産に対し機能追加を繰り返す派生開発も多い。ところが、設計方針を明確にしないまま機能追加を繰り返す派生開発が多いため、複雑で分かりにくく、全体像の把握が困難なソフトウェアとなる傾向が強い。これら理由のため、品質が安定せず、生産性も悪化するという問題があった。

このような状況を打開するためには、図 15-A-20-1 に示すように、既存コードのリファクタリングを進めるボトムアップの取り組みに加えて、開発戦略や設計思想をシステム仕様に落とすことで上流工程の完成度を上げることを重視したトップダウンの取り組みが必要であると考え。特に、「デジタル機器向けの組込みソフトウェアの開発」においては、派生開発がほぼ避けられない状況であるが、上流工程の品質を高めることで、設計の抜けに伴う手戻りの削減や、見つかったバグの影響範囲の明確化・限定化が実現でき、結果、検証工程を含む後工程の品質が安定し生産性も向上できることが期待できる。

¹ 事例提供: ビースラッシュ株式会社 山田 大介 氏

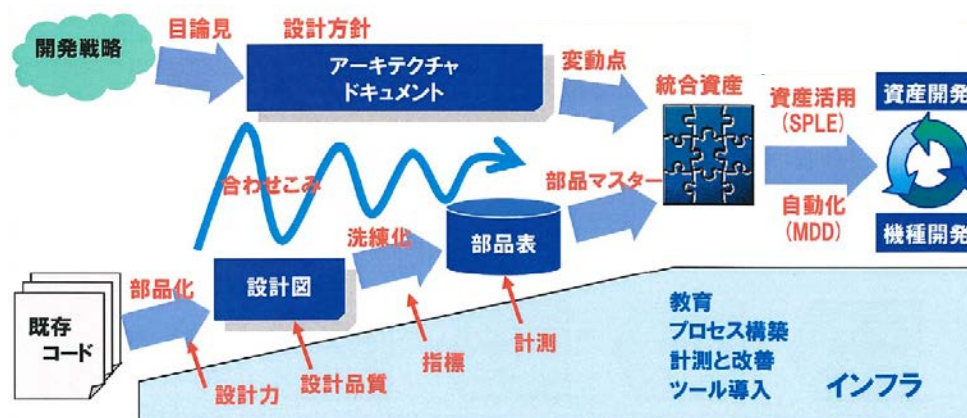


図 15-A-20-1 よりよいソフトウェアを開発するための
トップダウンとボトムアップの取り組み

そこで、本事例では、「デジタル機器向けの組込みソフトウェアの開発」において、上流工程の完成度を上げることを重視したトップダウンの取り組みとして、「アーキテクチャ設計を可視化する取り組み」について紹介する。一般的に、アーキテクチャ設計は、ソフトウェアの基本構造を決定する作業を指す。具体的には、システムに要求されている機能・性能を、システムを構成する要素に配分して構成要素の仕様を明確にするとともに、構成要素間のインターフェースを明確化することを言う。機能設計と物理設計からなることが多く、開発工程で言うと、詳細設計開始までに実施すべき工程、つまり上流工程になる。ソフトウェアの基本構造がどうなっているかということはソフトウェアの品質に大きな影響を与える。

本事例におけるアーキテクチャ設計では、開発戦略の策定、目論見（開発戦略から製品の特徴や売りを端的に表現したもの）の作成、設計方針の策定、構造設計（静的・動的）、可視化という手順で開発を進めて行く。開発プロジェクトでは、アーキテクト、アーキテクチャ設計書を読む開発要員、実装プログラマの各要員で開発を実施した。

なお、リファクタリングはソースコードの質を高めることでソフトウェアの資産価値を高める施策ではあるが、それなりの開発パワーがかかり経営判断を仰ぐケースも多いことから、本事例ではトップダウンの取り組みのみを取り上げた。

2. 導入の背景とアーキテクチャ設計の手順・効果

2.1. 従来開発の問題

組込みソフトウェアの開発では従来、「1.適用した技術や手法の概要」でふれたように、ソースコードを中心とした後工程依存の開発スタイルで行われる場合が多い。この開発スタイルは設計方針を明確にせず、加えて設計ドキュメントを十分に作成せずに進められるケースが多い。このような開発スタイルの問題点には、以下のようなものが挙げられる。

(1) 不明確な設計意図

- ・ なぜ、現状のコーディングになっているのか、設計方針が分からない
- ・ コードレベルでの修正で対処してしまうため、複雑な構造になる
- ・ 設計意図をドキュメントに残す方法があいまいで、正確な判断ができない
- ・ 複数の設計情報間の一貫性の確保、抜け漏れの防止、トレーサビリティ確保などの方法が不明確で、バージョンアップに伴って内容に齟齬が発生し、陳腐化する
- ・ 重要なトレードオフ（例：性能と消費電力）などの設計情報間にまたがる判断を明確に記載する手段がない

(2) 全体が見えない

- ・ 有識者の経験値に頼る場面が多い（属人性が強い）
- ・ 修正の影響範囲が分からず、影響範囲特定のコード解析に時間がかかる
- ・ どの部分が再利用の範囲か不明確であり、流用は行っているが生産性は低下している
- ・ 組込みソフトウェア設計に関する設計技法を習得していないことがある

(3) 上位視点のドキュメントが不足

- ・ 暗黙の了解が多く、技術情報が形式値化されていない
- ・ 他人が理解できるドキュメントが書かれていない
- ・ インターフェース不整合時に、都度打ち合わせで解決している
- ・ 新規参加者の即戦力化が難しい
- ・ 外部委託時に請負型で委託できない
- ・ 上位視点での設計に慣れていないエンジニアが多い
- ・ 設計する際、設計書を書くモチベーションが低下

2.2. 本事例で導入したアーキテクチャ設計の手順

本事例で導入したアーキテクチャ設計の手順を図 15-A-20-2 に示す。具体的には、以下の手順で開発プロセスを実行していき、最終的にアーキテクチャ設計書を完成させる。

- ① 開発戦略から製品の特徴や売りを端的に表現した「目論見」（要求の重点化）の作成
- ② 目論見を実現するための「設計方針」の策定
- ③ 開発ソフトウェアの機能単位としての「静的構造設計」（機能分割）
- ④ 開発ソフトウェアの実行要素単位としての「動的構造設計」
- ⑤ 「アーキテクチャ設計書」の作成

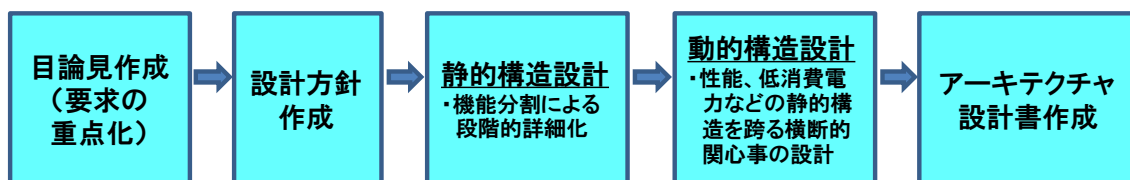


図 15-A-20-2 本事例で導入したアーキテクチャ設計の手順

2.3. 本事例で導入したアーキテクチャ設計の導入効果

2.2 で説明した手順を取り入れることによって、システム／ソフトウェア全体を俯瞰できる構造図が作成されるので、システム／ソフトウェア全体の「可視化(見える化)」ができる。

この結果、開発量の抑制、重複開発の抑制、計画的再利用、戦略的再利用の促進、「擦り合わせ」から「組み合わせ」開発の促進が可能となる(図 15-A-20-3)。

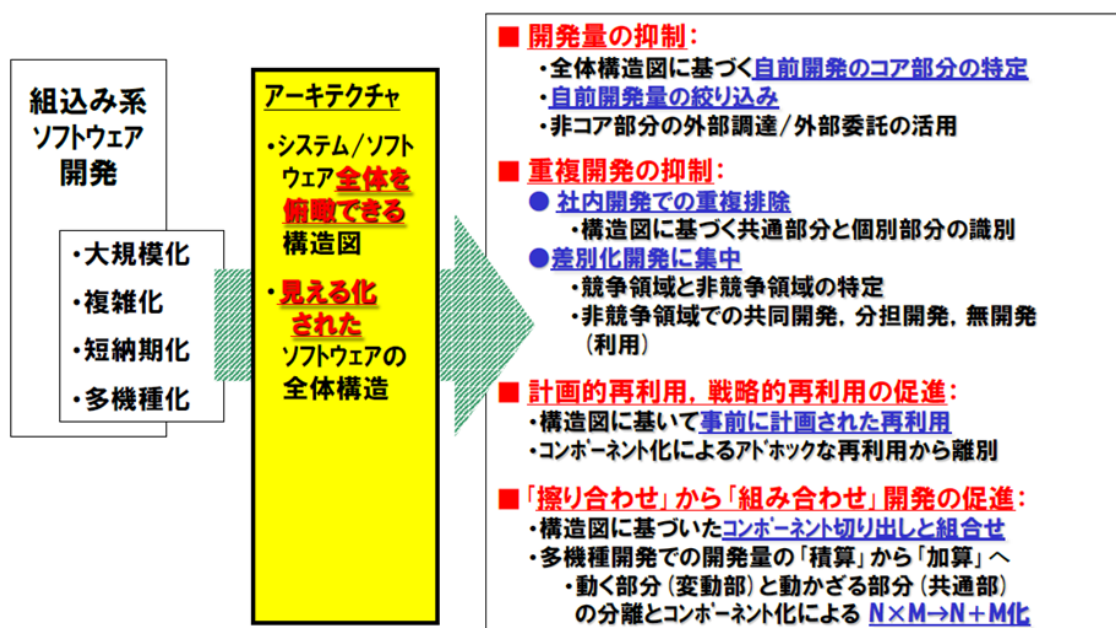


図 15-A-20-3 本事例で導入したアーキテクチャ設計の導入効果

3. 技術や手法導入のための事前準備や工夫

3.1. 設計内容の可視化

アーキテクチャ設計手法に触れた世の中の教科書の多くは概念的な説明が多く、開発の現場でアーキテクチャ設計手法を成功させるためには、設計方法や設計内容を可視化して、設計意図を開発者に伝達することが鍵となる。そのためには、まずアーキテクチャ設計の手順を明確化し、全体を俯瞰する必要がある。

(1) アーキテクチャ設計手順の明確化

手順の明確化を意識した全体の流れ、ポイントを以下に示す。

① 目論見

「目論見」とは「製品戦略と開発戦略の要点を記述したもの」であり、アーキテクチャ設計上重要な要求を整理・重点化し「目論見」としてまとめ、一覧を作成して管理する（図 15-A-20-4）。

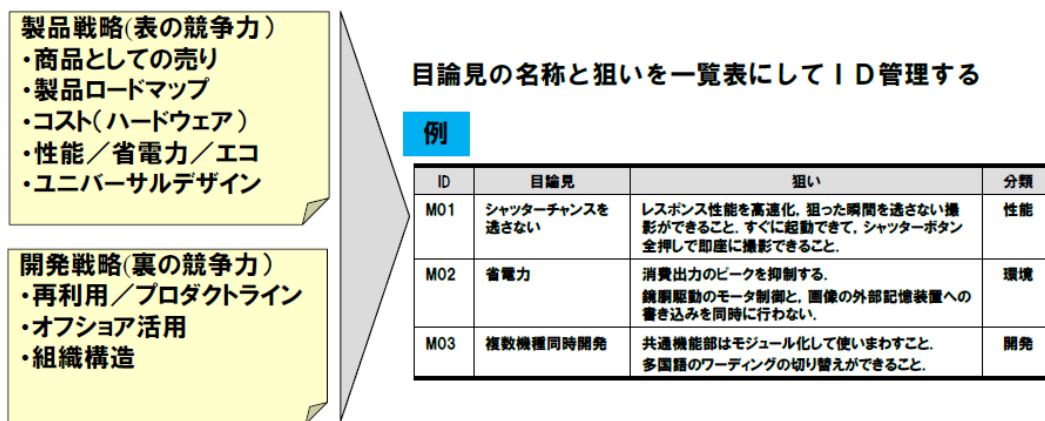


図 15-A-20-4 「目論見」と狙いの明確化

② 設計方針

「目論見」を実現するための「設計方針」を策定する。設計方針では、分割方針、ブロック化方針、制御方針などを策定していく。

③ 静的構造設計

「設計方針」に従って、「静的構造設計」を行う。この設計では段階的に機能を分割し詳細化を実施していく。

④ 動的構造設計

「静的構造設計」の次に、「動的構造設計」を行う。この設計では性能、低消費電力などの複数のモジュールや部品（静的構造）に跨る横断的な関心に関する設計を行う。

⑤ アーキテクチャ設計書の作成

「アーキテクチャ設計書」のフォーマットを規定し、これに基づいて「アーキテクチャ設計書」を作成していく。開発者に対して、アーキテクチャ設計の手順と「アーキテクチャ設計書」を周知し、可視化を図る。

(2) 全体の俯瞰（設計内容の整合）

可視化をより確かなものにする工夫として、「観点マトリクス」と名付けた独自の手法を導入した。(1) で作成された「目論見」、「設計方針」、「構造設計」を組み合わせ「観点マトリクス」で表現する。設計内容の整合については、静的構造を中心に、動的構造と実装構造を設計する。その際、ビジネス視点と開発者の視点の整合を図るため、「観点マトリクス」を使用して整合化する（図 15-A-20-5）。

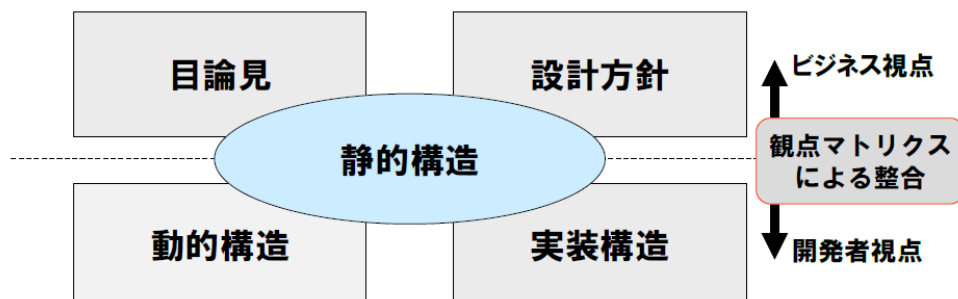


図 15-A-20-5 「観点マトリクス」による設計内容の整合

「観点マトリクス」は複数の視点（ビジネス視点・開発者視点）で設計内容を整合させるためのものであり、「目論見」を起点とした追跡が可能である。また、静的構造図を中心として設計要素間の関係を可視化するとともに、二項間の組合せの表で管理している（図 15-A-20-6）。

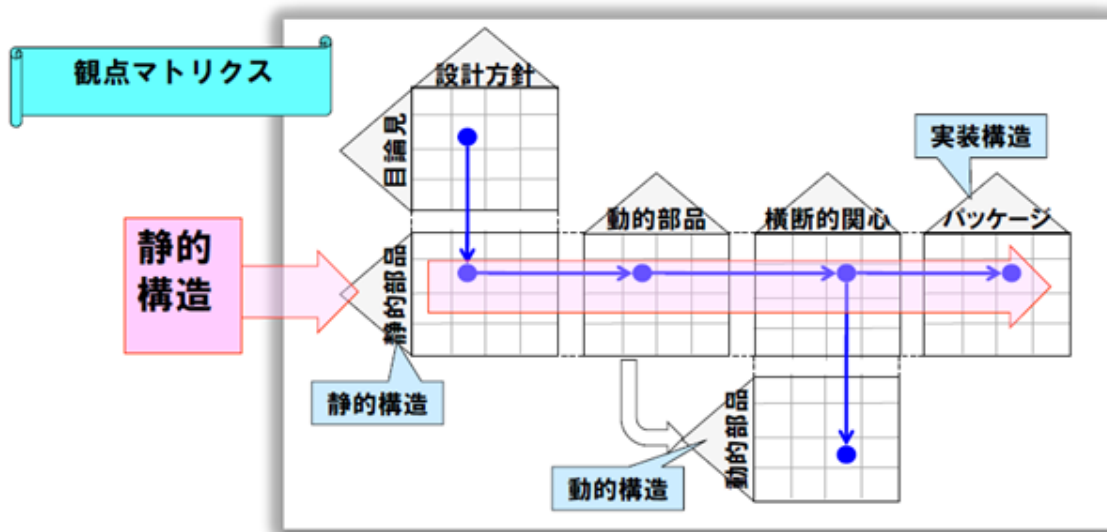


図 15-A-20-6 観点マトリクス

「観点マトリクス」を用いた、目論見から動的部品までの整合方法について、その手順を以下に示す。

① 目論見と設計方針の整合

まず、目論見×設計方針の「観点マトリクス」を作成する。これで、目論見を実現するための設計方針を抽出するとともに、設計方針間のトレードオフを検討し、優先順位を付ける。目論見と設計方針の整合のための「観点マトリクス」の例を図 15-A-20-7 に示す。行方向がビジネス視点（目論見）であり、列方向が開発者視点（設計方針）のマトリクスである。

<div>例</div> <div>観略表</div> <div>方針</div> <div>目論見</div>			分割方針		ブロック化方針			制御方針		
			P01	P02	P03	P04	P05	P06	P07	P08
			レイヤー化	UI分離	図処理部の独立	独立・取替の	外部UIモジュールの	起動順序の最適化	性能チューニング	全押し後の動作調停
シャッターチャンス を逃さない	M01	すぐに起動できること	▲							
	M02	撮影ボタン全押しですぐに撮影すること	▲						○	
きれいな画質	M03	映像処理エンジンの取り換えが容易であること			○					
省電力	M04	動作時の消費電力を抑える								○
複数機種同時開発	M05	共通機能部はモジュール化して使いまわすこと	○	○			○			
	M06	多国語のワーディングが切り替えできること				○				

図 15-A-20-7 目論見と設計方針の整合のための「観点マトリクス」の例

② 設計方針と静的部品の整合

次に、設計方針×静的部品の「観点マトリクス」を作成する。これで、設計方針の実現網羅性を確認し、静的部品間の協調動作の必要性を見極める。

③ 静的部品と動的部品の整合

最後に、静的部品×動的部品の「観点マトリクス」を作成する。これで、並行動作やパフォーマンスのボトルネックを明確化し、機能追加や修正の影響範囲を検討する。

④ 横断的関心の検討

3.2 に記述。

⑤ 実装構造設計

最後に、開発戦略に従い、実装単位を明確にする。

- ・ 新規開発するのか流用開発するのかを決定する
- ・ 静的部品×実装構造の「観点マトリクス」を作成する

3.2. 「横断的関心」の検討

前述の「観点マトリクス」に加えて、可視化をより確かなものにするもう一つの工夫として、「横断的関心」に着目した独自の手法を導入した。「横断的関心」とは、ある機能の設計が一つのモジュールや部品に閉じず、複数のモジュールや部品で実現される機能や非機能要件のことである。特に、組込み製品では、パフォーマンス、省電力、応答性、リアルタイム制約、例外処理（割り込み、中断など）、高信頼性・安定性などの特有の非機能要件の実現が求められることが多い。これらの特性を考慮した上で組込みソフトウェアのアーキテク

設計を行うことが重要である。

「横断的関心」を識別し、構造設計との関係をマトリクスで俯瞰することで、プロアクティブに擦り合わせを行う。

「横断的関心」と静的部品および動的部品をマッピングすることで、次のような効果を得ることができる。

- ・ 変更影響度が把握できる
- ・ 修正漏れを防止できる
- ・ 設計意図を伝達できる
- ・ 性能のチューニングができる
- ・ チューニング意図を明確にできる
- ・ 性能のボトルネックを発見できる

具体的な手順を以下に示す。

① 横断的関心と静的部品の整合

まず、静的部品×横断的関心の「観点マトリクス」を作成（図 15-A-20-8）し、仕様変更に対する影響範囲を見極めるとともに、省電力設計の影響範囲を明らかにする。この結果、変更影響度の把握に活用できるとともに、横断的関心がどの静的部品に関係するのかが分かる。

② 横断的関心と動的部品の整合

次に、動的部品×横断的関心の「観点マトリクス」を作成する。これにより、実行時の設計意図が明確になるとともに、パフォーマンスのチューニングやパフォーマンスのボトルネックの把握に役立つ。

関心事 静的部品		シナリオ		横断的関心		
		CS01	CS02	CC01	CC02	CC03
		撮影シナリオ	再生シナリオ	起動時間	省電力	全押し後制御
アプリ層	C01 撮影	○				○
	C02 再生		○			
	C03 設定					
	C04 画面遷移	○	○			
ミドル層	C05 自動焦点	○				
	C06 画像ストリーム	○			○	○
	C07 画像ファイル管理		○			
	C08 コンフィグ管理					
	C09 ウィンドウ管理		○		○	
ドライバ層	C10 鏡胴駆動	○		○	○	
	C11 レンズ駆動	○		○		
	C12 CCD駆動	○		○	○	○
	C13 画像処理エンジン	○		○		○
	C14 カードIF	○	○	○	○	
	C15 キースキャン			○	○	
	C16 液晶モニタドライバ	○	○	○	○	

図 15-A-20-8 静的部品と横断的関心の「観点マトリクス」の例

3.3. 設計力向上のための教育の実施

本事例で導入するアーキテクチャ設計を実行に移すためには、設計者にその手法をあらかじめ習得させる必要がある。このため、構造的な設計図の読み書きと全体を俯瞰する経験を積むための教育を実施した。研修内容について図 15-A-20-9 に示す。

デジタルカメラを例に開発者業務に近い形の題材を用いて講義と演習を実施した。図 15-A-20-9 の上段に示しているアーキテクチャ設計手順（目論見の作成、設計方針の策定、静的構造設計、動的構造設計）の講義を実施し、下段に示している中間成果物（目論見リスト、方針リスト、静的構造図、動的構造図）を作成する演習（研修）を行った。その演習中に観点マトリクスの活用方法を説明し、アーキテクチャ設計書の作成方法を教育した。

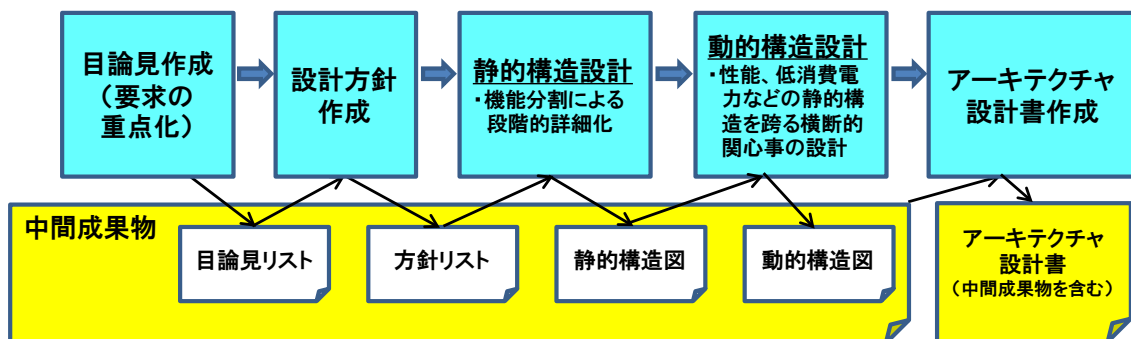


図 15-A-20-9 研修内容

4. 適用した技術や手法の効果測定の方法とその結果

4.1. 本事例で導入したアーキテクチャ設計の導入効果

製品戦略や開発戦略に沿った目論見が策定され、これに基づき設計方針が明確になったため、作成された設計ドキュメントは一貫したものになった。また、観点マトリクスの使用でビジネス視点と開発者視点での整合が図られたため、品質の高い組込みソフトウェアが開発できた。特に組込みソフトウェアに求められる特性であるパフォーマンス、省電力、応答性、リアルタイム制約、例外処理（割り込み、中断など）、高信頼性・安定性などが考慮されたものになった。

4.2. 得られたドキュメント体系

得られたドキュメント体系を図 15-A-20-10 に示す。アーキテクチャ設計書のフォーマットを規定し、設計書への記載内容を統一するために記載項目を規定した。構造設計内容の記載に加え、設計のインプット情報の記載を重視し、各項目間の機能要件のトレーサビリティを確保する方法を具体化した。これにより、開発現場におけるアーキテクチャ設計実施の負担軽減を実現できた。

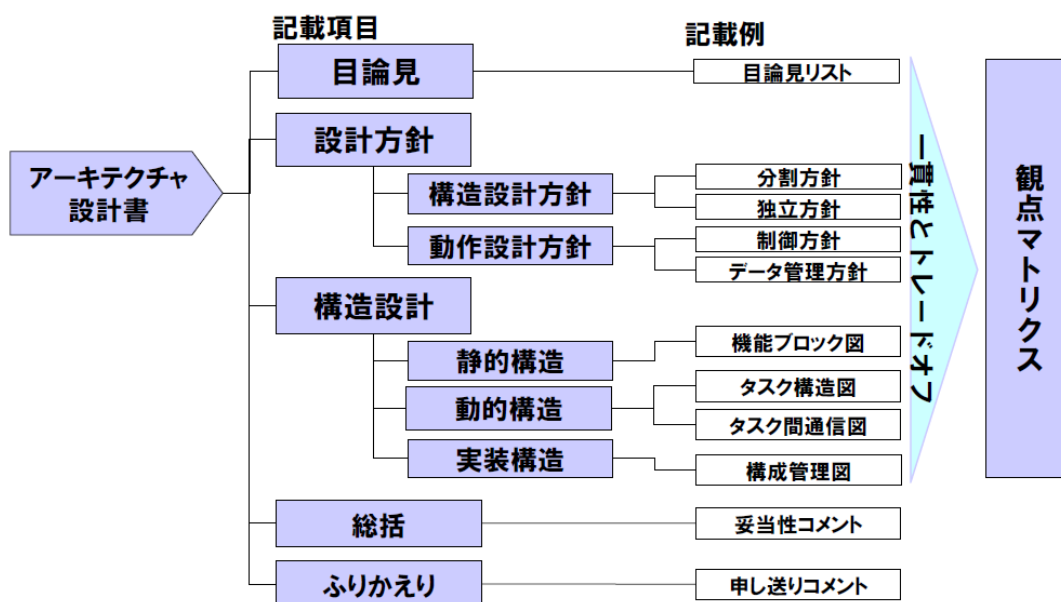


図 15-A-20-10 得られたドキュメント体系

4.3. 現場導入時の課題

現場導入時の課題として以下の2点が挙げられる。

- (1) アーキテクチャ設計を実施するにはアーキテクトの活躍が重要になるが、アーキテクトの役割や権限が不明確になりがちである。プロジェクト計画時にアーキテクトの役割と権限を明確にしておく必要がある。
- (2) アーキテクチャ設計を採用する当初は不慣れなこともあり、これを実施するには時間がかかることが多い。これをプロジェクト計画時に織り込んでおく必要がある。

5. 適用した技術や手法の導入後の改善活動、今後の課題等

今後の展開として、多機種開発への適用と、組込みソフトウェア開発のアーキテクトの戦略的な育成を検討している。

5.1. 多機種開発への適用

多機種化に応えるためには開発期間の短縮化が不可欠であり、そのためには派生開発を一層効率化する必要がある。図 15-A-20-6 で示した観点マトリクスに機種別や顧客別の条件(変動点)を加味した、派生開発に一層効果が期待できる「観点マトリクス」を検討中である(図 15-A-20-11)。

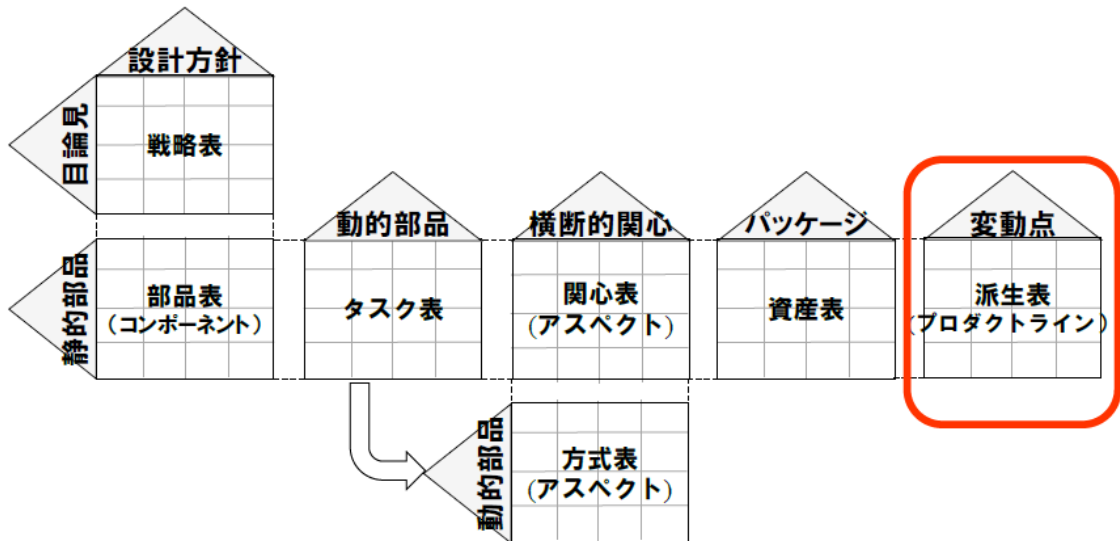


図 15-A-20-11 派生開発に一層効果が期待できる「観点マトリクス」の例

5.2. 組込みソフトウェア開発のアーキテクトの戦略的な育成

アーキテクチャ設計手順を熟知し、アーキテクチャ設計書を作成することができ、『何を、どのように作るのか』を正しく判断できる人材、加えて、リーダーシップを発揮できる人材、これがアーキテクトである。特に、組込みソフトウェア開発のアーキテクトに課せられた役目は、様々な利害関係者と調整し、価値のある製品開発を先導するとともに、開発メンバーを教育・育成し、アーキテクチャ設計書を作成できるようにすることである。このアーキテクトの選抜と育成の仕組みが必要である。(図 15-A-20-12)。

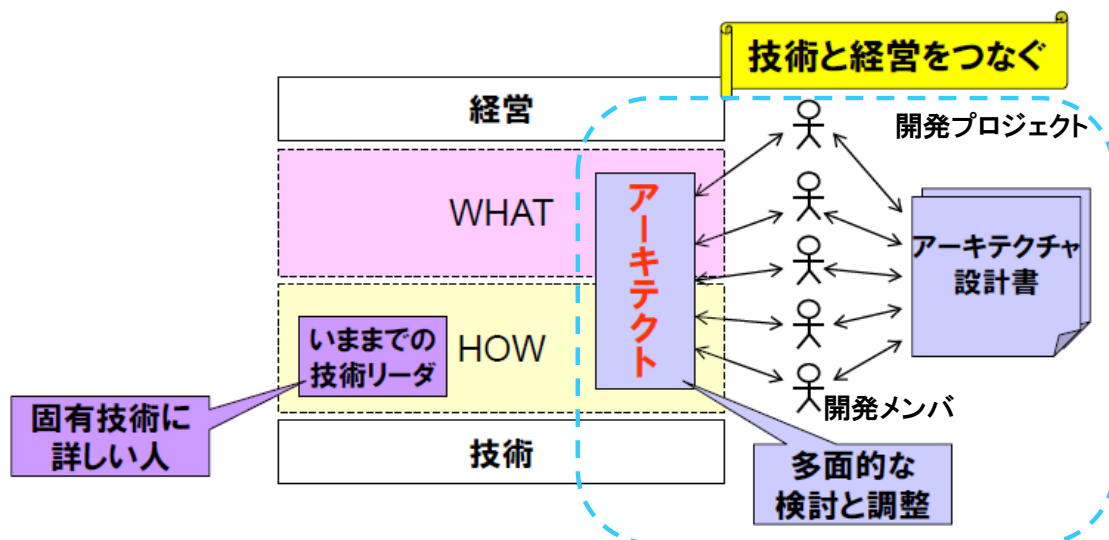


図 15-A-20-12 組込みソフトウェア開発のアーキテクトの戦略的な育成

6. アーキテクチャ設計の効果と水平展開

6.1. 品質について

本事例で導入したアーキテクチャ設計により、製品戦略や開発戦略に沿った目論見が策定され、これに基づき設計方針が明確になり、作成された設計ドキュメントは一貫したものになった。また、観点マトリクスの使用でビジネス視点と開発者視点での整合が図られた。その結果、十分な品質の確保と、性能においても十分なレベルに達成することができた。品質が向上したことにより、他の開発案件に対しても、計画的な再利用や戦略的な再利用の促進が期待できる。また、設計意図を他の人へ伝達するコストが分かることにより、実装時の手戻り作業が減少した。

6.2. 生産性について

本事例では、そもそも（一般論で言うところの）アーキテクチャ設計手法に開発の現場が不慣れだったこともあり、生産性は従来とほぼ同等レベルにとどまった。

しかし、派生開発を活用する多機種開発においては、今後、手法の習熟度が進むはずなので高い生産性が得られることが期待できる。

6.3. 検証への影響

上流工程の完成度が高まり、検証時には、スムーズな検証が実施できた。バグが検出されてもその影響範囲が明確であり、対応時間が少なくなった。

6.4. 水平展開状況

本事例における開発対象は、デジタルカメラの組込みソフトウェアであったが、その後、デジタルカメラの組込みソフトウェアの派生開発やデジタルカメラ以外の機種の組込みソフトウェアにも水平展開を実施している。

7. まとめ

本事例では、派生開発がほぼ避けられない「デジタル機器向けの組込みソフトウェアの開発」において、上流工程の完成度を上げることを重視したトップダウンの取り組みとして、「アーキテクチャ設計を可視化する取り組み」について紹介した。今後は多機種開発への適用を展開するとともに、アーキテクチャ設計を推進する組込みソフトウェア開発のアーキテクトを育成することが重要である。

参考文献

- [1] 山田大介、春名修介、古山：組込みソフトウェアのアーキテクチャ設計方法の可視化の試み、JISA SPES2010 事例研究
- [2] 尾山壮一、福嶋慎一、春名修介 他：平成 19 年度ソフトウェアに関する調査報告書Ⅱ 組込み系ソフトウェア開発の課題分析と提言 ～大規模化、複雑化、短納期化、多機種化の波にどのように立ち向かうべきか～、(社) 電子情報技術産業協会 ソフトウェア事業委員会、p.77-127、2008 年 3 月
- [3] 福嶋慎一、春名修介 他：平成 20 年度ソフトウェアに関する調査報告書Ⅱ 組込み系ソフトウェア開発の課題分析と提言 ～開発スピードアップを阻害する要因の実態分析～、(社) 電子情報技術産業協会 ソフトウェア事業委員会、p.119-140、2009 年 3 月
- [4] 春名修介、課題認識とワークショップ 2009 の狙い ～組込み系ソフトウェア開発のキモは何か～、JEITA 組込み系ソフトウェア・ワークショップ 2009、(社) 電子情報技術産業協会 ソフトウェア事業委員会、
<http://home.jeita.or.jp/is/committee/software/091020/materials/1-2.pdf>
- [5] 春名修介、CEATEC2009 インダストリアルシステムトラック講演 組込み系ソフトウェア開発をスピードアップ！(JEITA 活動報告) ～組込み系ソフトウェア開発のキモは何か～、
<http://home.jeita.or.jp/is/committee/software/091009/materials/haruna.pdf>
- [6] Glenford J.Myers (著)、国友義久(訳)、伊藤武夫(訳)、ソフトウェアの複合／構造化設計、近代科学社
- [7] ANSI/IEEE 1471-2000、Recommended Practice for Architecture Description of Software-Intensive Systems
- [8] Nick Rozanski、Eoin Woods (著)、榊原彰 (監修)、牧野祐子 (訳)、システムアーキテクチャ構築の原理 IT アーキテクトが持つべき 3 つの思考、翔泳社

掲載されている会社名・製品名などは、各社の登録商標または商標です。

独立行政法人情報処理推進機構 技術本部 ソフトウェア高信頼化センター (IPA/SEC)