機能安全規格への対応から見えた、ソフトウェア品質の見える化
～レビュー手法の見直し～

2012年10月11日

パナソニック アドバンストテクノロジー（株）

大阪開発センター　機能安全基盤技術開発チーム

川崎　雅弘

PADの会社概要

http://panasonic.co.jp/pad/

商号(社名)	パナソニック アドバンストテクノロジー株式会社 (英文) Panasonic Advanced Technology Development Co., Ltd.
事業目的	デジタルネットワーク関連商品・システム(ネット家電)及び そのソフトウェアの研究・開発・設計・販売
事業規模	開発委託、業務委託等収入:65.7億円(11年度実績) 従業員数:427名(2012年4月1日現在)

- PAD
  - 人事G、経理G
  - 戦略企画室
  - エンジニアリングセンター
  - 大阪開発センター
    - 機能安全基盤技術開発チーム
  - 名古屋研究所
  - 広島研究所

本社(大阪府門真市)
大阪開発センター
エンジニアリングセンター

広島研究所(東広島市)

名古屋研究所(名古屋市)

Panasonic ideas for life

パナソニックグループ内の位置づけ

本社R＆D部門

要素開発

製品開発

事業ドメイン

デジタル・AVC

携帯電話

生活家電

医療

ヘルスケア

車載

デバイス

ロボット

産業機械

デジタルネットワーク
関連商品

**パナソニック
アドバンストテクノロジー㈱**

高信頼性／安全性が求められる分野への展開

－ 組織として安全対応能力を構築する

SGS TUV GmbH (ドイツ) の
ISO26262 ソフトウェアプロセス認証を単独取得 (2012/8)

Panasonic ideas for life

本発表の概要

車載用機能安全規格 ISO26262では、「説明責任」が求められる。

しかし、設計通りに実装しているという証明は、単にチェックリストを使ったレビューをしているという説明だけでは許容しにくい。 それは、レビュワーの能力依存など、見えない要素があるためである。

本発表でのレビュー手法は、「再現可能なレビュー」をイメージし、具体的な手順を定義し、準拠性と網羅性を説明できるようにした。

■ポイント

再現可能なレビューのために、以下のように確認方法を特定する。

- 上位設計文書と厳格に比較し、不整合を検出する
- テンプレートと整合させたチェックリストに基づき確認する
- NG指摘だけでなく、OKの記録も残す

本レビュー手法を実際のプロジェクトで施行し、効果を確認した。
また、そこでの気づきと、現場への展開活動についても紹介する。

ISO26262規格上のレビュー要求

規格要求はこれだけ。しかし、「アーキテクチャ設計が安全要件を遵守していること」は、単に「チェックリストでレビューをした」「トレーサビリティを取っている」では説明できない

■例：ソフトウェアアーキテクチャ設計の検証に関する記述

ISO26262-6:2011
7.4.18 ソフトウェアアーキテクチャ設計は以下に示す特性を証明するために、ISO26262-8:2011, 9章に従い、かつ表6に示されるソフトウェア設計検証方法を用いて検証されること
a) ソフトウェア安全要件の遵守
b) ターゲットハードウェアとの互換性
c) 設計ガイドラインの順守

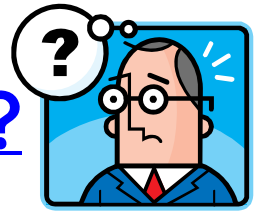
ISO26262-8:2011, 9章
9.1 目的
検証の目的は、作業成果物の要件への準拠を確実にすることである。
9.2 General ※抜粋
b) 製品開発フェーズにおいて、検証は下記のように様々な形で実施される。
1) 設計フェーズにおいて、検証は要求仕様、アーキテクチャ設計、モデル、ソフトウェアコードなどの作業成果物の評価である。そのため、作業成果物は正確性、完全性、一貫性のために、既に確立している要件に準拠している。評価はレビューやシミュレーション、解析技術によって実施される。評価はシステマティックな方法により、計画され、定義され、実施され、ドキュメント化される。

表6 ソフトウェアアーキテクチャ設計の検証方法

	方法	A	B	C	D
1a	設計のウォークスルー	++	+	o	o
1b	設計のインスペクション	+	++	++	++
1c	設計の動的部分のシミュレーション	+	+	+	+
1d	プロトタイプ作成	o	o	+	+
1e	形式手法による検証	o	o	+	+
1f	制御フロー分析	+	+	++	++
1g	データフロー分析	+	+	++	++

「++」強い推奨、「+」推奨、「o」利用に意見なし

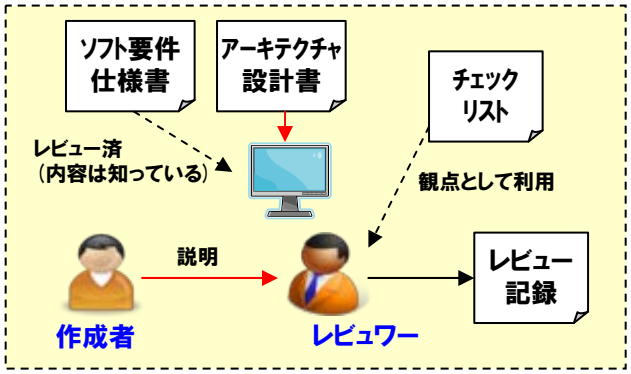
あなたが品質/安全性の責任者なら、何を確認しますか？



なぜ「レビューをした」では説明できないか

レビュー記録を見ても、レビューの網羅性や十分性が判断できない

従来のレビューと課題



- **時間内に終わるように制御され、質にムラがある**
  - 気になる部分中心で、十分に見ていない部分もある

- **レビュワーの能力に依存する**
  - 知識や記憶に頼る指摘が多く、NGの根拠が不明瞭

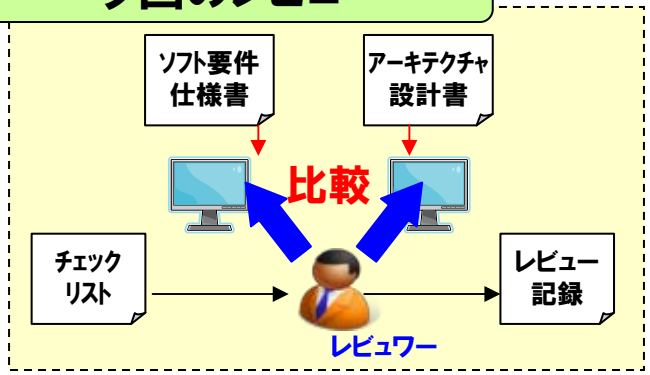
- **NGの記録はあるが、OKの記録が無い**
  - チェックリストの項目が成果物の複数個所にあっても、全部確認しているかは不明

品質/安全性の責任者として、
納得できるようなレビューを自分で実施してみる

今回のレビューの取り組み

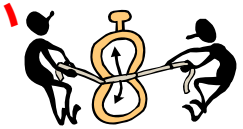
レビューを再現可能なように、確認方法を限定し、確認箇所を記録する

今回のレビュー



●上位文書と厳格に比較し、不整合を検出

- 漏れだけでなく、根拠の無い追加／変更も対象
- どんなに時間がかかっても、省略はしない



●チェックリストとテンプレートを整合

- チェックリスト通りに確認していけば、レビュー完了

■レビューチェックリスト（抜粋）

分類 Category		チェック項目 Check items	該当項目 Check field	チェック箇所 Checked Part)
0.全体 Total	1	文書名、文書番号、承認者名、作成者名、発行日、発行部署が明確になっているか		
	2	改定履歴の日付、内容が明確になっており、SPMによって承認されているか。		
	3	組織の資産として設計書を活用する視点で、設計書に対する構成管理、資産管理が明確になっているか ・設計書を再利用する際の注意点が明確になっているか 例)開発に適用したプロジェクトの振り返りシートの注意事項を確認している		
	4	要求、要件、設計の表現方法は適切か		
1.概要 Outline	1.1	設計書の目的と位置付けが明確になっているか		
	1.2	参照ドキュメントが明確になっているか		
	1.2	要件、システム要件仕様やシステムアーキテクチャ設計項目とソフトウェア設計項目(コンポーネントの対応付けが明確になっているか		
	1.2	ソフトウェア設計書が適切なバージョンのシステムアーキテクチャ設計書に基づき作成されているか		
	1.3	必要な用語、略語などが明確になっているか		
2.制約条件 Constraints	2.1	構成管理の制約は明確になっているか ・ディレクトリ構造は明確になっているか		
	2.2	ソフトウェアアーキテクチャに関する制約は明確になっているか ・要件仕様、安全要件 ・システムとハードウェアの設定(コンフィグレーション)の仕様 ・ハードウェアーソフトウェアインターフェイス仕様 ・ハードウェア設計仕様の関連要件 ・タイミング制約 ・ユーザーインターフェイス		
3.コンポーネント詳細記述 Detailed component		ソフトウェアの安全分析(FTA, FMEA)により、安全メカニズムを検証しているか		
		・コンポーネント間の結合度(関連度合いが弱くなるインターフェースを定義しているか		
	3.2	並行処理(要求される機器性能を実現する並行処理制御)の動作を定義しているか		
		コンポーネントの機能(機能構造)が明確になっているか		

観点

実際に見た場所

テンプレート上の、確認すべき箇所

■レビュー確認箇所の記録（抜粋）

レビュー管理シート Review Management Sheet

成果物名 Work Product: J1109200.SWAD_バッテリ電圧監視センサ_ソフトウェアアーキ

目次 Item ／イベント番号 Event No	レビュー結果 Review Result 1	2	3	4
表紙 Cover	#1	OK	#44	OK
改訂履歴 Change history	#2	#15	OK	
1. 概要 Outline	OK			
1.1 目的 Objectives	OK			
1.2 対象機種 Product	OK			
1.3 参照ドキュメント Reference documents.	#3	#16	#42	OK
1.4 用語、略語などの定義 The definition of terms, abbreviations, and etc.	#11	#17	OK	
2. 設計方針 Design strategy.	OK			
2.1 基本構造 Basic Structure	OK			
2.2 ソフトウェア安全要件の実現 Realization of Software safety requirement	OK	#18	#43	OK
2.3 その他の設計方針 Other design strategy	OK			
3. 制約条件 Constraint conditions.	OK			
3.1 ハードウェアからの制約 The constraints of hardware.	OK			
3.2 ソフトウェアの制約 The constraints of software.	OK			
4. ソフトウェア構成 Software structure.	OK	#19	OK	
4.1 ソフトウェアの機能(機能構造) Functions of software (	#12	#14	#45	
4.2 ソフトウェア構造(静的) Software architecture (st	OK			
4.2.1 ソフトウェアアーキ Software architecture	#4	#20,22	OK	
4.2.2 ソフトウェアコンポー 間のインターフェイス In	OK			
4.2.3 ソフトウェアの外部 イス External interface of software	#13	#23	OK	
4.2.4 ソフトウェアコンポ General of Software components	OK	#21,24-37	#46	
4.3 ration(dynamic architecture)	#6-#10		OK	
		#38	OK	
02] Priority management for tasks and ISRs		OK		
vice routines		OK		
itecture of software (state machine diagram)		#39-41	OK	
5. 性能見積もり Performance design and estimation.		OK		
5.1 時間制約見積もりTime constrains design and estimation.		OK		
5.2 メモリマップと使用量見積もり【SWPE002】Memory map and memory size estimation.		OK		

OKの記録

NGの記録
（指摘内容は
レビュー記録に記載）

設計書の目次

レビューから属人性を排除 ⇒ 説明可能な品質（品質の見える化）

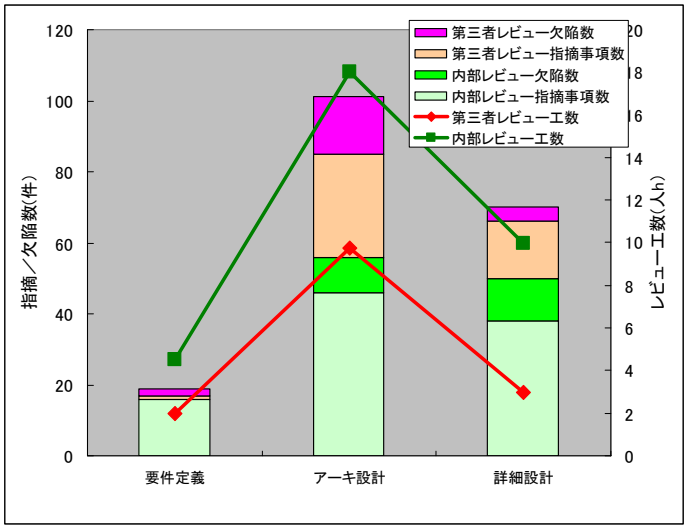
Panasonic ideas for life

レビューの結果と効果

一貫性/遵守性において、従来レビューでは未検出のものが多く検出された

■従来のレビューの後に本レビュー方法を実施し、指摘／欠陥検出の度合いから有効性を分析

	内部レビュー（従来レビュー相当）	第三者レビュー（本レビュー）
レビュー目的	ソフトウェア要件定義書との一貫性を確認する	
レビュー手法	ウォークスルー	机上チェック
人数	2名（レビュワー＋作成者）	1名
チェックシート	同じものを使用	
特記事項	未ﾚﾋﾞｭｰ箇所は無いこと	上位文書と厳格に比較



■成果物規模

ソフトウェア要件定義書	7（Pages）
ソフトウェアアーキテクチャ設計書	47（Pages）
ソフトウェア詳細設計書	59（Pages）
ソースコード（C言語）	3.5（KSteps）

※いずれもレビュー時間は同等となった（工数は人数差による）
※両レビューの結果、単体テストでの欠陥は7件、結合テスト以降は0件である

■指摘結果から見られる傾向

内部レビューの指摘傾向 ⇒ 信頼性/妥当性	第三者レビューの指摘傾向 ⇒ 一貫性/遵守性
・記載ミス、記載漏れ ・記述不足、紛らわしい記述（理解できない） ※レビュワーによって、設計書の「質」を上げる効果を出している	・上位文書に記載がなく根拠が不明 ・上位文書からの情報の欠落 ※これらは、必ずしも異常挙動に繋がらないものや、テストで検出されるものも多く含まれる

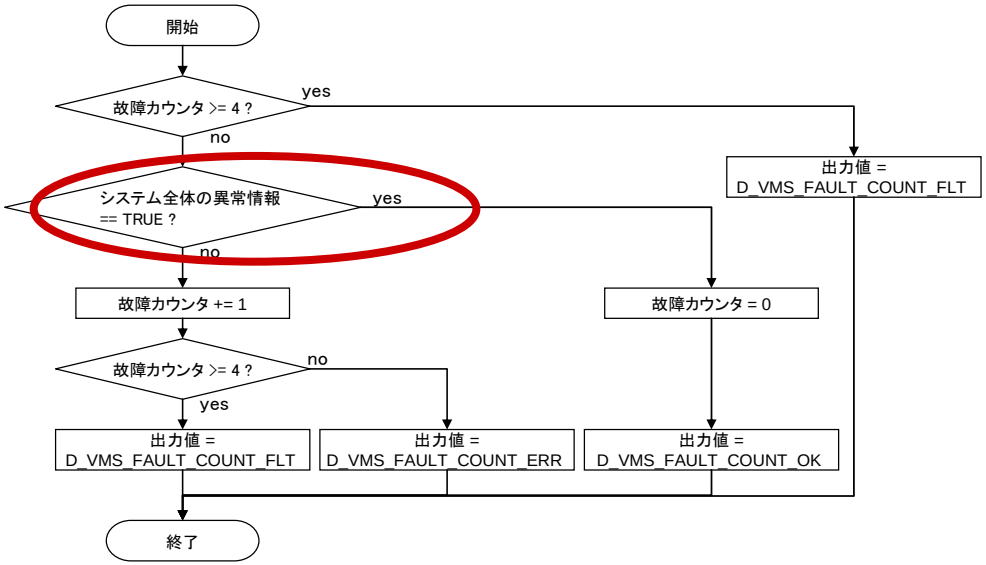
品質/安全性の「説明」をするには、本レビューの「上位文書との厳格な比較」は有効

厳格な比較をする上での課題

本レビューで、異常挙動には繋がらないが上位文書との不整合がいくつも検出された

不整合の例

・設計書ではTRUEかどうかで判断しているが、コードではFALSEかどうかで判断している




```
/*
 * 故障カウンタ制御
 */
static void VMS_Flt_CountFault ( USHORT usFault, USHORT *pusFaultInfo )
{
   if ( g_ucFaultCount >= D_VMS_FLT_COUNT_MAX ) { /* 故障確定状態 */
      *pusFaultInfo = D_VMS_FAULT_COUNT_FLT;
   } else { /* 正常・異常状態 */
      if ( FALSE == usFault ) { /* 異常発生 */
         g_ucFaultCount++; /* 異常発生のためカウントアップ */
      if ( g_ucFaultCount >= D_VMS_FLT_COUNT_MAX ) { /* 故障確定か？ */
            *pusFaultInfo = D_VMS_FAULT_COUNT_FLT;/* 故障確定状態 */
         } else {
            *pusFaultInfo = D_VMS_FAULT_COUNT_ERR;
         }
       } else { /* 異常なし */
         g_ucFaultCount = 0; /* 故障カウントリセット */
         *pusFaultInfo = D_VMS_FAULT_COUNT_OK;
       }
   }
}
```

■関数としての動作は同じだが、レビューで厳格に比較すると差異として検出される
- 機能安全規格上はここまで厳格な一致を求めていないし、レビュー結果としてNGでなくても良いが、このような差異が多発すると、**レビューが極めて非効率**になる
- もし、設計通りの実装だと効率面などで問題があるならば、設計書を修正すべき

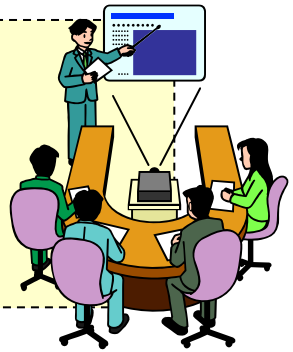
開発者がこれを「問題ない」と思っていることで、遵守性が信用できなくなる

本レビュー手法の展開

・プロジェクトメンバに対して、レビューしにくい不整合の事例を共有

- 再発防止で欠陥事例の共有は良くあるが、**開発者の意識を変える**ことが目的

■メンバの感想
- ・言われなければ、それが問題になるとは気づかなかった。
- ・これはレビューでNGとしたほうが良い。そのほうが、意識が変わってくると思う。
- ・特に若手の開発者のトレーニングに取り入れてはどうか。



・新規の車載関連プロジェクトのメンバに対して、本レビュー手法を試行

- 説明では理解してもらえないと思い、小規模なものを実際にレビューしてもらった

■メンバの感想
- ・カルチャーショックを感じた。今までのレビューでは不十分だと理解できた。
- ・「アーキテクチャ設計が要件通りか」を確認するには、確かにこのようにしないと説明できない。

まとめ

■検証レビュー（Verification）では上位文書との比較をすることを徹底する

- 記憶に頼らず、二画面を使って厳格に比較を行う。
- レビュー箇所を記録し、時間的に未確認部分は別途確認する。

説明できる

■実際に試してみて効果を実感する

- 経験すると考え方が変わる。品質／安全性の責任者自身も体験すべき。

※説明だけだと従来レビューの課題が実感できず、本レビューを非常に重く感じてしまう。
（むしろ、テスト欠陥が激減するので、結果として工数削減の可能性もある）

実施できる

■開発者の意識改革をし、品質／安全性責任者との思いを合わせる

- 品質／安全性責任者や開発者の実施結果を事例としてトレーニングする。
  - －開発者は気づいていても、それが問題だと思っていないかもしれない。
  - －欠陥だけでなく、レビュー効率低下なども問題として意識させる。

納得できる

このような活動が徹底されれば、品質は安定し、予測も し易くなる

一人ひとりが、品質／安全性の意識を持つことが重要

安全文化

Panasonic ideas for life

ご清聴ありがとうございました

Panasonic

ideas for life