第3分科会（Aグループ）

組み込みソフトウェア開発における品質向上への取り組み

～要求仕様書に起因する下流工程での不具合ゼロに向けて～

Quality Improvement in Embedded Software Development
- For Zero Failures in lower process from Requirements Specifications -

主査　飯泉 紀子　株式会社日立ハイテクノロジーズ
副主査　田所 孝文　株式会社山武
研究員　大立 薫　オリンパス株式会社
平島 直人　アンリツエンジニアリング株式会社
井上 博史　三菱電機コントロールソフトウェア株式会社

1. 研究概要

近年、組み込みソフトウェア規模が肥大化する一方で、開発期間や開発コストは短縮される傾向にある。また、ソフトウェア品質に対する要求は一層厳しさを増しており、開発期間や開発コストを厳守しつつソフトウェア品質を向上させることが求められている。しかし、実際のソフトウェア開発現場では、下流工程での不具合検出に注力しているにも関わらず、市場導入後の不具合発生が後を絶たない状況である。これは、ハードウェアと結合しなければ分からない不透明な仕様を上流工程で排除しきれないという、組込みソフトウェア特有の問題が絡んでいると考えられる。

本研究では、上流工程において仕様書の品質を向上させることが、ソフトウェア品質向上の根本施策であるという信念のもと、仕様書の品質低下を引き起こしている3つの要因、「仕様不明瞭」、「仕様記載ミス」、「仕様記載漏れ」に対し、これらを防止する施策を適用結果とともに提言する。

Abstract

In recent years, software scale has been getting larger. In spite of this situation, on software development, shorter schedule and smaller cost are required. And higher software quality is also required. This means software quality is required to improve within prearranged schedule and cost. However, actually, more problems are found out than expected in lower process, and also problems have been not eliminated after delivery due to insufficient problem finding. This is attributed to the embedded software problem that is difficult to solve specification in upper process because the uncertain specification is related with connection to hardware.

In this report, we recognize that quality improvement of specification in upper process is essential for software quality improvement. Basing on this reorganization, we focus three causes of quality reduction of specification; inconsistency specification, mistakes drawing on specification, omission of specification; and suggest methods to solve the quality reduction along with some examples and their results.

2. 背景(研究課題を選定した理由)

近年、組み込みソフトウェア開発規模の増大が進む一方、開発期間の短縮や開発コストの低減を要求されている。組み込みソフトウェアの不具合が原因で、社会インフラシステムが停止してしまうような事故も多く報告され、社会へ与える影響も大きくなっている。そのため、ソフトウェアの品質に対する要求は一層厳しさを増している。

組み込みソフトウェア開発特有の問題として、ハードウェアとの関連が強く、ソフトウェアの仕様を早い段階では確定しきれないということがある。ハードウェア開発と平行させてソフトウェア要件定義やソフトウェア詳細設計を行うと、開発途中での仕様変更や仕様追加が頻発し品質確保が難しくなる。このように、組み込みソフトウェア開発では上流工程での品質作りこみを十分に実施し難い状況のため、テスト段階で品質を確保せざるを得なくなり、テストやデバッグに多くの期間、費用を費やしている。2008 年版組込みソフトウェア産業実態調査[1]によると、ソフトウェア開発全体に占めるソフトウェアテスト工数の割合は、30.5%にも達している。

一方で、実機を使用してテストを行う必要のある組み込みソフトウェア開発の場合は、実機をタイムリーに必要台数分準備できるとは限らず、テストで十分な品質を確保することも難しい状況である。ソフトウェア品質ガイドブック[2]にあるように、上流工程で作りこまれた不具合を下流工程で検出し改修するまでには多くの時間とコストがかかることも考慮すると、下流工程で不具合を取り除いて品質を高めるのではなく、上流工程で品質を作りこむことが本質的な施策であることは間違いない。例え、組み込みソフトウェア開発特有の問題があり、上流工程での品質作り込みが難しいとしても、このままで良いのだろうか。もしかしたら、我々組み込みソフトウェア開発技術者は、これを理由に上流工程での品質作りこみを怠っているのではないかと考えた。

そこで、現状を打破し、少しでも上流工程での品質を向上したいとの思いから、上流工程での品質作りこみを研究目標として選定した。

3. 目標

上流工程での品質作り込みを目的として、SLCP-JCF2007 の共通フレーム体系 2007[3]で定義されている"1.6.4 ソフトウェア要件定義"及び、"1.6.5 ソフトウェア方式設計"のアクティビティにおいて作成されるドキュメント(以下、要求仕様書と称する)の品質を向上させる施策を検討する。これにより、要求仕様書に起因する不具合の発生をゼロにすることをねらう。

具体的には、以下の手順で研究を行った。

(1) 現状把握
要求仕様書の品質が原因となっている不具合の調査。

(2) 要因分析
要求仕様書に起因する不具合はなぜ発生してしまうのかを分析。
要求仕様書のどこに問題があるのかを分析。

(3) 施策検討
要求仕様書の品質を低下させている原因として、「仕様不明瞭」、「仕様記載ミス」、「仕様記載漏れ」を取り上げ、施策を検討。

(4) 期待効果と今後の課題
施策により、期待できる効果と今後の課題を考察。

4. 現状把握

まず、我々の開発現場で発生している不具合がどの段階で作り込まれてしまっているか、いくつかのプロジェクトに対して調査した。その結果、図1に示すように、プロジェクトにより差はあるものの、約 40%が要求仕様を作成する工程であった。

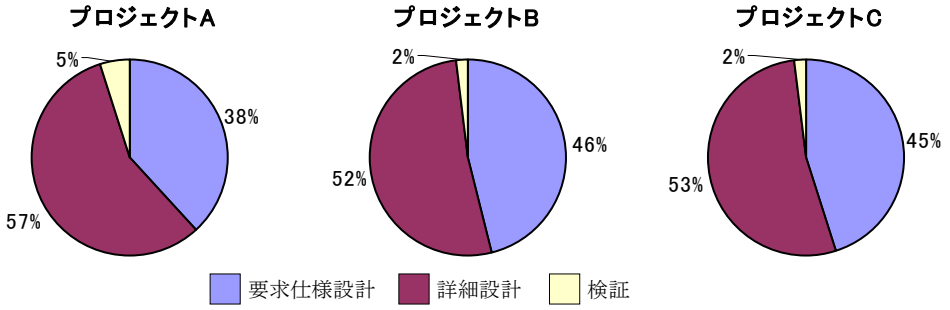

図1　開発現場で発生している不具合の作り込み工程別割合

5. 要因分析

我々のソフトウェア開発現場では、要求仕様書を作成する工程で作り込まれた不具合が約 40%あることが分かった。そこで、その不具合がなぜ作り込まれたのかについて分析を行った。その結果を、付録1の図4に示す。

要因を大別すると、「個人」、「組織」、「プロセス」、「環境」に分けられるが、「プロセス」や「環境」のフレームワークとしては、CMMI や ISO/IEC-15504(SPICE)などが存在しており、既に確立されている。本研究では、要求仕様書の品質に直接的な影響を与える「個人」に着目した。

まず、「個人」は要求仕様書を「書くことができるのか？」、「書けないのか？」 あるいは、「書く意思はあるのか？」という観点を検討した。「書くことができるか？」、「書けないか？」は、「個人」のスキルに依存する。一方、「書く意思があるか？」は、要求仕様書を誰のために書き、書くことの重要性を感じているかという意識に関わる。 先に述べたように、組み込みソフトウェアでは、ハードウェアと結合しなければ分からない不透明な仕様を上流工程で排除しきれないという特徴がある。これが固定概念となってしまい、知らぬ間に、書かなくても問題がないと納得してしまっているのではないだろうか。本来のソフトウェア品質向上の根本施策、そしてあるべき姿は、上流工程にて要求仕様書の品質を向上させることであるが、設計者にその意識が欠けていることが問題と考える。

次に、発生した不具合のうち要求仕様書に起因するものを取り出し、要求仕様書がどのように悪く、その結果、どのような問題を引き起こしているのかを調査した。その結果を表1に示す。

表1　要求仕様書に起因する不具合要因と下流工程（詳細設計・テスト設計）にて発生する問題

要因	詳細設計時の問題	テスト設計時の問題
仕様不明瞭	仕様不明瞭（何通りにも解釈できる、冗長な文章により読み手が理解不能となる）により、解釈ミスをする	テスト設計時にテスト項目を抽出できない
仕様記載ミス	要求仕様書間の不整合（矛盾）により、誤った実装をする	テスト工程になってから問題が発覚する
仕様記載漏れ	要求仕様書の記載漏れにより、機能実装漏れが発生する	テスト設計時にテスト項目にならない

要因は大きく次の3つに分類できる。

(1)　仕様不明瞭　　:要求仕様書の記載が不明瞭で解釈ミスを起こしている

(2)　仕様記載ミス　　:要求仕様書の記載ミス(矛盾)により、誤った実装をしてしまっている

(3)　仕様記載漏れ　　:要求仕様書の記載漏れにより、機能実装漏れが発生している

上記3つが引き起こす問題は、詳細設計時に留まらずテスト設計時にまで影響を及ぼす。 例えば、「仕様不明瞭」の場合、何通りにも解釈できる、冗長な文章により読み手が理解不能となる、テスト設計時にテスト項目を抽出できない、と言った、目には見えにくい根の深い問題が多く潜む。「仕様記載ミス」の場合は、要求仕様書間の不整合(矛盾)が引き起こされ、これに気付かずに誤った実装を行ってしまうリスクを含む。このようなことは、一般にテスト工程になってから問題が発覚する。そして、「仕様記載漏れ」の場合は、機能自体が実装されず、テスト設計時のテスト項目にもならないため、不具合が検出されるまでにさらに長い時間を要する。

本研究では、要求仕様書の品質を低下させているこれら3つの要因に対し、開発現場ですぐにでも実行できる施策を検討した。

6. 要求仕様書の品質を高めるための施策

下流工程にて不具合を誘発する要求仕様書の品質問題、「仕様不明瞭」、「仕様記載ミス」、「仕様記載漏れ」の3つについて、それぞれ施策を検討した。

6.1. 要求仕様書とテスト設計書の同時作成による仕様不明瞭の抑止(施策①)

仕様不明瞭であると、何通りにも解釈できる記述や、冗長な文章により読み手が理解不能となり、「解釈ミスによる実装」や、「テスト項目を抽出できない」問題を誘発する。更に、これは市場導入後の不具合発生にまでに発展する。

仕様不明瞭の防止策として、底上げを図り一定品質を目指すガイドラインの作成、レビューの実施、人材育成などが挙げられる。ガイドラインやレビューは、そもそも品質が低いドキュメントに対しては大きな効果は期待できない。一方で、文章や設計スキルの向上などの人材育成は、重要ではあるが即効性に欠ける。

そこで、開発期間、費用に最も影響を与えるテスト工程に着目し、更に、即効性があり、現状の工数とさほど変わることなく実施可能な施策について考えた。それが、要求仕様書とテスト設計書を同時に設計する方法である。 要求仕様書とテスト設計書を同時に作成することのメリットは、設計者がテスト者の立場で記述し、また、設計者が誰のために書くのかを明確にできる点である。要求仕様書とテスト設計書の作成を同時期にすることで、記載漏れを防止することができる。例え要求仕様書の記述内容が多少分かりにくい場合でも、テストケースという形で明確に記述されることで、要求仕様を明文化したドキュメントになる。この方策を開発現場で実現するには、これを支援するツールを導入するのが効果的である[4]。今回我々が作成したツールは、テスト設計書から要求仕様書にフィードバックできる機能を持つ。プロセス上の改善として、設計・検証フェーズから「要求仕様書」へのフィードバックは必須である(図2)。

本施策をあるプロジェクトに適用した結果、「仕様不明瞭」を改善し要求仕様書の記述や、テストの網羅性を向上させ、要求仕様書に起因する下流工程における不具合を最小限にすることができた。

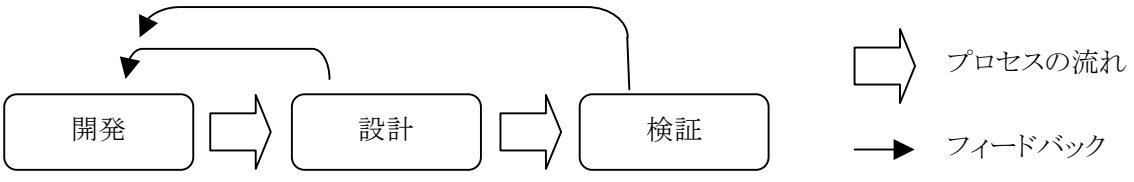

図2　プロセス中のフィードバックアプローチ

6.2. 仕様書間の不整合を検出しやすくするための要求仕様書への関連の埋め込み(施策②)

要求仕様書間に不整合があると、統合テストや、システムテストになってからはじめて問題が発覚する場合が多く、改修に多くのコストがかかる。また、仕様変更の際、影響範囲を正しく把握できないため、修正漏れなどを発生させやすい。要求仕様書間で仕様が不整合となる原因の一つとして、各仕様書間において内容の関連を示す工夫がなされていないことがある。このため、仕様間の関連を正確に理解しないまま設計を進めてしまう、あるいは、理解の正しさの確認を怠ってしまう。内容の関連を正しく把握できていない限り、レビューでも不整合を発見することはできない。

上記問題を解決する施策として各仕様書間における内容の関連を、各仕様書に直接ハイパーリンクの形で記述する (図3)。

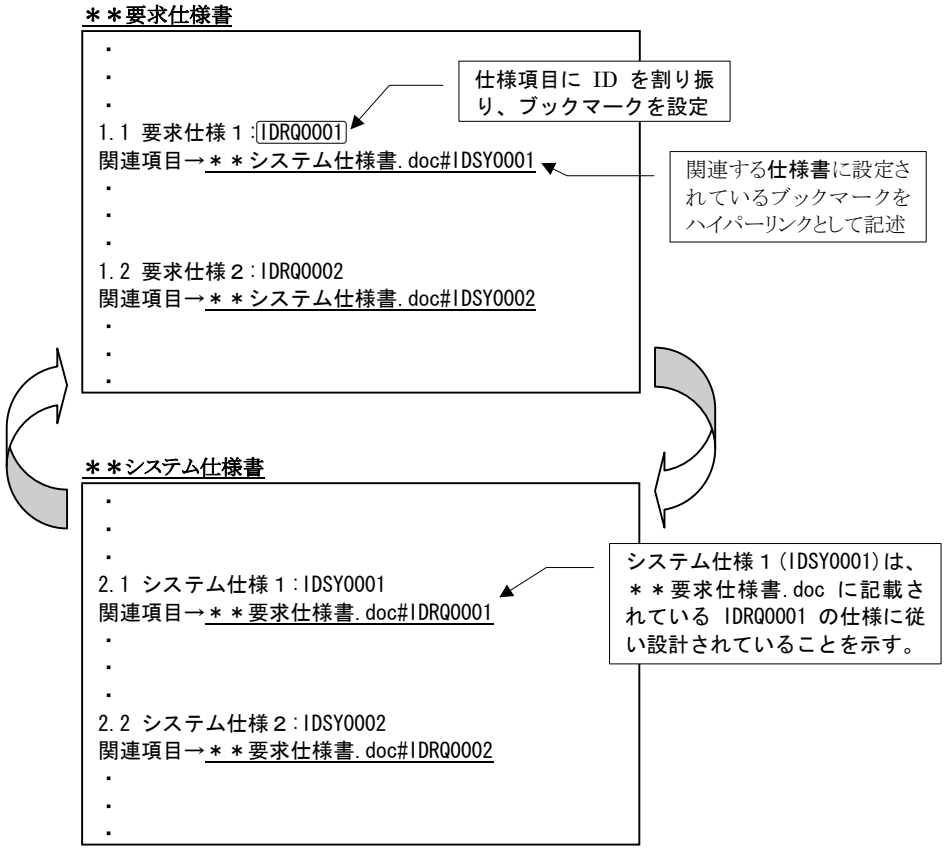

図3 仕様書間の関連記載方法

本方式は、Microsoft Word のブックマーク、ハイパーリンクの機能を利用して、ドキュメント内に関連を埋め込んでいる。このため、ドキュメントを参照しながら、それぞれのドキュメント間を自由に行き来ができる点がメリットである。Microsoft Word のマクロ機能を利用してツールを作成することにより、トレーサビリティ・マトリクスなどを自動生成することも可能であり、レビュー時に全体の整合性を確認することもできる。また、仕様書改版に伴い、リンク切れや反映忘れなどが発生する懸念があるが、これに関してもマクロ機能を利用したツールを作成することにより整合性の確認を行うことも可能であり、メンテナンス性についても優れる。

6.3. 記載漏れを防ぐための記載項目チェックリストの作成(施策③)

記載漏れを防げないのは、要求仕様書に記載すべき項目が開発現場では明確になっておらず、また、適切な確認も行われていないことが原因の一つであると分析した。

記載項目が明確になっていない理由としては、機能に重点を置いて要求仕様を作成することが習慣化しており、非機能項目については、あまり重要視されていないことが挙げられる。

そこで、非機能項目を含む記載項目チェックリストを作成し、設計時およびレビュー時に利用することを考えた。チェックリストは、IEEE1220[5]で規定されている、要求分析の手順に従って、システム設計で実施しなければならない項目を、「境界の明確化」と「成果物自体の明確化」の 2 つに分け検討した。

(1) 境界の明確化チェックリスト

組み込みソフトウェア開発における要求仕様の境界を明確にするという視点で、IEEE1220[5]からチェック項目を抜粋したチェックリストを作成した。

IEEE1220 の定義では、境界の明確化の方法は、「関係する要求洗い出し」、「対象システム範囲明確化」、「使用方法の明確」に大別して定義されている。この項目に従って詳細化したチェック項目とした。作成したチェックリストを付録 2 の表 3 に示す。

(2) 成果物自体のチェックリスト

組み込みソフトウェアにおける要求仕様の成果物を明確にするという視点で、IEEE1220 からチェック項目を抜粋したチェックリストを作成した。

IEEE1220 の定義では、「要求仕様の成果物の明確化」の方法は「機能性能明確化」「検証性識別」「要求記述明確化」に大別して定義されている。この項目に従って詳細化したチェック項目とした。作成したチェックリストを付録 2 の表 4 に示す。

要求仕様書の作成時及び、レビュー時にこのチェックリストを用いて確認を行うことにより、必要な記載項目が明確になるため、記載漏れを減少させる効果が期待できる。

7. 本研究の考察と今後の課題

組み込みソフトウェア特有の問題として、ハードウェアと結合しなければ分からない不透明な仕様を上流工程で排除しきれないという課題はあるが、組み込みソフトウェアは上流工程での品質作りこみができないというのは、理由にならない。 設計者が上流工程にて要求仕様書の品質を向上させることが、ソフトウェア品質向上の根本施策であるという信念を持ち続けないと、下流工程で発生する要求仕様書に起因する不具合を「ゼロ」に近づけることや、市場導入後の不具合を撲滅することはできない。

本研究で取り組んだ3つの施策を IEEE830[6]の要求仕様の特性に関連付け、要求仕様書の品質を高められたか否かの評価を試みた。表 2 に、その結果を示す。3つの施策全体で要求仕様書の特性の各項目をカバーしているため、下流工程での要求仕様書が起因する不具合をゼロにすることができそうである。

今回、 3 つの施策を提案したが、上流工程における要求仕様書の品質を向上させることは容易な取り組みではない。システムの規模が大きくなり、システム全体を把握できる人が居なくなったことや、アウトソーシングにより開発者のスキルが低下したことも強く関係している。開発現場では、業務分散により各個人の責任管轄が曖昧になり、その結果、問題意識が欠落していると感じる。上流工程に携わるものは下流工程まで責任を持ち、自分たちのミスによる問題を重く受けとめ、現

状の悪しき状況を打開するため改善活動を継続すべきである。そして、組織的に運営することで、この現状を打開できると考える。

本研究では、「仕様不明瞭」、「仕様記載ミス」、「仕様記載漏れ」を防止するために、開発現場ですぐにでも実行できる施策を述べた。今後は、複数の開発プロジェクトに適用し、実績を積み上げ、より効果的な適用ノウハウが蓄積されていく仕組みを構築する必要がある。何より、上流工程での要求仕様書の品質を向上させることが設計、検証工程でのあと戻りを軽減でき、結果的に開発日程短縮及び、下流工程における工数削減に繋がるという強い信念を持って施策に取り組むことが重要である。

表2 要求仕様の特性(IEEE830[6])と施策の関係

要求仕様の特性	施策①	施策②	施策③
完全性	－	○	○
正確性	○	－	－
一貫性	－	○	－
検証容易性	○	－	－
修正容易性	○	－	－
追跡性・参照容易性	○	○	－

参考文献

[1] 独立行政法人 情報処理推進機構 ソフトウェア・エンジニアリング・センター、「2008 年版 組込みソフトウェア産業実態調査 報告書 －プロジェクト責任者向け調査－」

[2] 森口編、ソフトウェア品質管理ガイドブック、日本規格協会、1990

[3] 独立行政法人 情報処理推進機構 ソフトウェア・エンジニアリング・センター編、共通フレーム 2007、オーム社、2007

[4] 独立行政法人 情報処理推進機構 ソフトウェア・エンジニアリング・センター編、プロセス改善ナビゲーションガイド～ベストプラクティス編～、オーム社、2008

[5] IEEE1220:IEEE Standard for Application and Management of the Systems Engineering Process

[6] Institute of Electrical and Electronics Engineers,Inc. 著, IEEE std 830-1998 IEEE Recommended Practice for Software Requirement Specifications

付録1　現状の問題点の分析結果

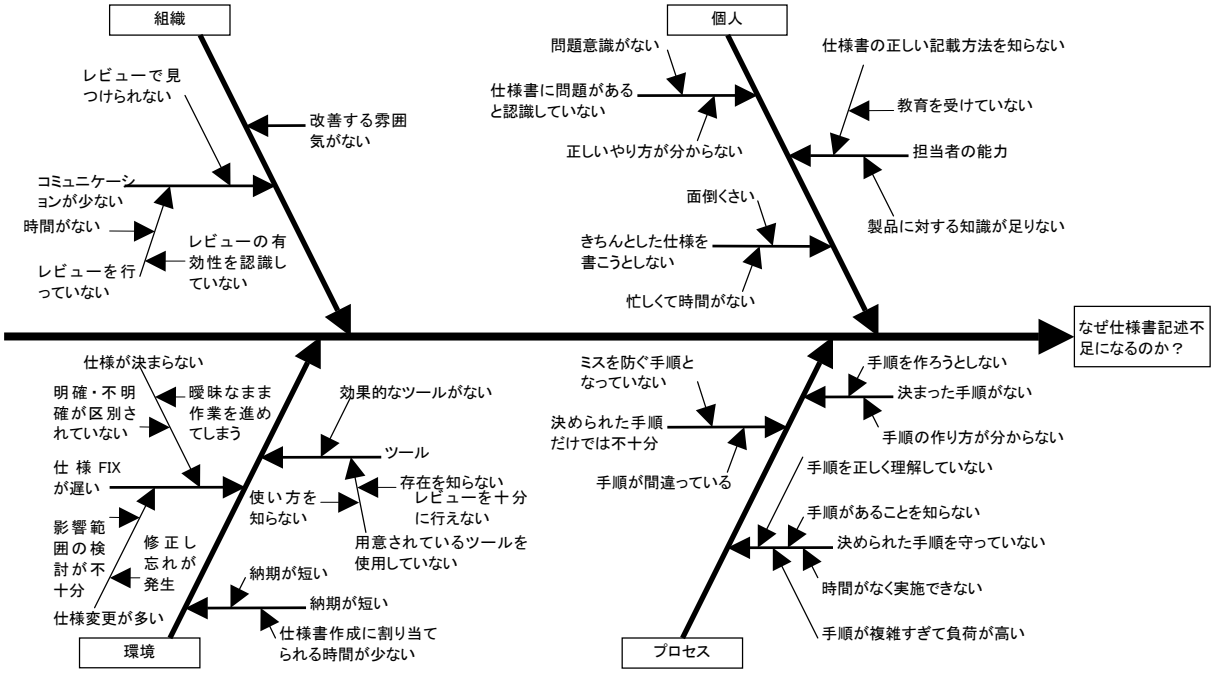

図4 仕様書に起因する不具合発生要因の連関図

付録2　記載項目チェックリスト

(1)　境界の明確化チェックリスト

表3 機能の明確化チェックリスト

手順	確認項目	確認結果
関係する要求洗出し	顧客の期待の定義ができているか？	
	要件設計の段階では、顧客の期待は「指示」として記述できているか？	
	書かれた「指示」は「なにを」「いかに」が整理されているか？	
対象システム範囲明確化	システム境界の定義ができているか？	
	システム境界の定義ができているか？	
	顧客の期待からなる「指示」を「システム境界」が網羅しているか？	
	インタフェースの定義ができているか？	
	データ関係、マンマシン、他システムのインタフェースに関する機能に関して「どんなデータを」「いかに関係するか」が整理されているか？	
	利用環境の定義ができているか	
	性能、不具合と、問題発生時の影響リスクなどが定義できているか？	
	外部制約の定義ができているか？	
	公式規格、技術基準、国際規格、人的要求、競合製品などの外部制約を定義できているか？	
	効果指標の定義ができているか？	
	性能、安全、操作性、可溶性、信頼性、保守性などの効果を計る指標が定義できているか？	
	プロジェクトの設計管理ができているか？	
	設計方法、ツールの適用等　技術面での管理方法は明確になっているか？	
	ライフサイクル、リソースの配置など　プロジェクト運営面での管理方法は明確になっているか？	
使用方法明確化	運用シナリオの定義ができているか？	
	他システムとの関係、マンマシンインターフェイス等を考慮した運用シナリオが定義できているか？	

(2)　成果物自体のチェックリスト

表4 成果物自体の明確化チェックリスト

手順	確認項目	確認結果
機能性能明確化・検証性識別	機能要求の定義ができているか？	
	サブ機能の定義、機能実現の手順、データの流れや制御の流れ、排他制御の流れなどは定義できているか？	
	機能要求の検証方法を検討しているか？	
	性能要求の定義ができているか？	
	性能要求をどのように実現するかが記載されているか？	
	コスト、スケジュール等を考慮した性能になっていか？	
	性能要求の検証方法を検討しているか？	
	運用モードの定義ができているか？	
	システム訓練のための運用や模擬運転の運用などを検討できているか？	
	運用の検証方法を検討しているか？	
要求記述明確化	ヒューマンファクターの定義ができているか？	
	スペース、視界、手の届く範囲、利便性などの人的要素を考慮した設計になっているか？	