Vol.2011-SE-172 No.5
Vol.2011-EMB-21 No.5
2011/5/18

開発手戻り回避のためのソフトウェアアーキテクチャ妥当性検証活動

鈴木尚志†

要旨：組込みソフトウェア製品開発において「手戻り」の原因となる低再現性障害の実態調査・分析を行った。
その結果、これら障害の原因の多くが、開発初期に実行されるアーキテクチャの動的構造定義の不備であることが判明した。
そこで、開発初期のアーキテクチャ設計時に、これらの障害原因を摘出するためのモデル検査ツール SPIN を用いたアーキテクチャレベル動的構造の妥当性検証活動を計画、実開発で試行したので、その結果を示す。

Software Architecture Validation Activities for Redo Costly Development Avoidance

HISASHI SUZUKI†

Summary: Actual embedded software development projects were analyzed for poor-reproducible defects that cause costly redo development. The results showed that the root cause of many of them were incomplete definitions of the dynamic structural architecture within the first stage of development.
Then trying to apply the SPIN model checking for architecture level dynamic structure validity verification for real production at the architectural design when developing, the results from the activities are shown.

1. はじめに

組込みソフトウェア開発プロジェクトにおいて、予期せぬコストを発生させる大きな理由がいわゆる「手戻り」だといわれている[1][2]。従来、産業界では、ソフトウェア開発のV字プロセスで知られているように、上流工程の成果物の妥当性の検証は、上流工程でのレビュー及び、実装後の下流工程での統合テストにて実施されてきた。そのため、常に実装後の下流工程に入ってから、上流工程での誤りが発見されることになるため、その修正のための、膨大な「手戻り」時間が発生していた。

それゆえ、近年、産業界においては、組み込みソフトウェアの大規模化、複雑化に対応するため、モデル駆動型開発の適用[3]や、開発フェーズ戦略としてのイテレーティブ・インクリメンタル（反復漸増型）開発の適用など「手戻り」を削減し、それに伴うコストを発生させないための様々な施策が実行されている。

一方、組込みソフトウェア製品開発において、大きな「手戻り」の原因とされているものの一つに、低再現性障害がある。障害発生の再現性が高ければ、原因究明、障害修正は比較的容易だが、再現性が低かったり、再現の頻度が一定でない場合などは、その原因を究明するのは非常に困難である。そして、一旦障害を修正しても、その修正確認作業も困難なものとなる。また、これらの問題は、他障害のための対応作業の副作用によって現象が再現が困難となってしまい、再現不能により解決と見なされ、そのまま出荷の後、フィールドで大きな障害が発生してしまったと言う事例も多い。

そこで本論文では、同じ組込みソフトウェアドメインの複数製品開発プロジェクトから得た障害報告データを元に、低再現性障害に関する統計的・定性的分析を行った。

結果、これら障害の原因の多くが、アーキテクチャ設計レベルの動的構造の妥当性の不備にあることが判明した。

一方、近年、数理的な技法である形式的検証技術の実用化が急速に進み、これら技法のソフトウェア開発への様々な適用が報告されてきている[4][5]。このうち特にSPIN などのモデル検査は上記のような動的構造妥当性の検査に適していると考えられる。

そこで、本論文では、上述したような開発コストの増大の原因である障害の早期除去を目的とした、モデル検査ツール SPIN を用いたソフトウェアアーキテクチャ動的構造の妥当性の検証活動を提案し、これを実開発プロジェクトに対し、本活動を試みた結果についても紹介する。

2. 低再現性障害に関する調査

本章では、開発プロジェクトに大きな「手戻り」コストを要求する低再現性障害

† 日本アイ・ビー・エム(株)
IBM Japan Ltd.

Vol.2011-SE-172 No.5
Vol.2011-EMB-21 No.5
2011/5/18

について、実製品開発時の障害データをもとにして行った調査について述べる。

2.1 調査概要

今回使用したデータは、1997 年より 2005 年までに行われたプリンター、および、デジタル複合機 5 製品の組み込みソフトウェア開発プロジェクトにおける障害データである。これらのデータは、同一の障害管理ツールを用いて管理されていた。

これら 5 つのソフトウェアは、その全てが、機器に組み込まれ、異なるメーカーから異なる製品として出荷されている。

対象ソースコードは、全てが C 言語で記述されたものであって、それぞれが、およそ 200 万ステップ規模であった。また、開発は、6 週間を反復開発期間の１単位とした反復的開発で行われた。

分析の対象とした５プロジェクトの障害報告総数は、7488 件であった。各障害報告には、障害報告者による現象、障害分析者による障害の重要度づけを含む障害の詳細分析、そして、開発者による障害の原因と対策が含まれているため、全件の内容確認作業によって分析が可能であった。また、このうち 15 件が、障害登録ツールへの２重登録や空登録などのツール誤操作であったため、今回の有効標本数はこれらを除く 7473 件とした。

全障害の修正に要する期間（TAT）の平均は 34.2 日であり、また、１反復単位以下で解決した障害は、6051 件（81%）であった。反復型開発において、ある反復期間中に検出された障害は、基本的に次の反復期間で対応することになる。今回得られた TAT の調査より、多くの障害修正が、予定通り１反復単位中に修正が終了している事を示している。

2.2 低再現性障害の性質について

低再現性障害は、障害の原因調査の開始と障害修正確認作業を困難にする、そこで低再現性障害について調査を試みた結果、全障害中の 194 件の障害が、完全な障害再現性がない障害として、障害報告者、あるいは障害分析者によって報告されており、これらの TAT は、平均 51.5 日であった。すなわち、低再現性の障害解決には、高再現性障害に比べ、２週間程度多くの時間がかかっていたことが判明した。

そして、これら障害に対する発見時の第一次状況報告の多くが、システム結合後テストで発見され、それぞれの障害を再現する直接的なトリガーや、再現頻度が不明であり、調査が必要であると報告している。即ち、これは単体テストや、定義された単一のテストケースによって障害を発見すること自体が困難であることを意味していると共に、ソースコードレベルで障害が修正されたとしても、再現状況を構築することが困難であるため、その修正確認にかかるコストも通常より大きくなることをも意味する。

また、これら障害の記録をたどると、これらの低再現性障害修正の副作用により、製品パフォーマンスの低下や、その他インターミッテントな新規障害の発生を引き起こすことがあり、いわゆる「バグのモグラたたき」の引き金となることも多いことも観察された。

2.3 低再現性障害の分析

低再現性障害の原因を、以下のように定性的に分類し、TAT を調査した。

１．特定の機能の同時実行による機能の相互干渉
２．想定外の運用・運用
３．メモリリークなどの実装不全
４．その他、機材不良など

表 1　低再現性障害の原因と TAT

原因	発生件数（出現率）	平均 TAT
機能相互干渉	149 (76.8%)	56.0 日
実装不全	24 (12.4%)	40.1 日
想定外運用・操作	12 (6.2%)	36.7 日
その他	9 (4.6%)	27.1 日

ここで、最も多く発生していた機能相互干渉に分類された障害は、タスク構造、状態遷移、例外処理など、いわゆるアーキテクチャ動的構造の妥当性の不備に起因する障害である。以下、これらの機能相互干渉を引き起こすアーキテクチャ障害（以下 Dfi 障害群とする）に着目する。

2.4 障害分析者による Dfi 障害群の性質の分類

これらのプロジェクトにおいて、障害分析者は、現象と経験により、各障害の発見時に現象より障害を分類し、それぞれに重大なものから順に１から４の重大度を付与している。対象開発プロジェクトにおいては優先度は明示的に設定されていなかったが、開発プロセス定義上は、障害の重要度１は、障害解決までは、他の検査工程を停止すべきクリティカルな問題である、といった指針が定義されており、実際には重要度を作業優先度としても用いていた。表 2 に、障害分析者が障害に対し重要度をどのように割り当てているかの傾向を示す。

全障害を見た際には、開発速度に大きな影響を与える副作用を持つ重要度１、２とされた障害の割合は合わせて 21.2%であるにも関わらず、Dfi 群では 73.1%であった。これより、障害分析者は、Dfi 群に属するような障害の発生時に、現象面からみて開

Vol.2011-SE-172 No.5
Vol.2011-EMB-21 No.5
2011/5/18

発工程へのインパクトが重大なものであると予測･判断する傾向があることがわかる。
また、表 3 には、Dfi 群とそれ以外の障害群の TAT の比較を提示する。

表 2　障害分析者による障害重大度の割当て

	全障害	Dfi 群
重大度１	112 (1.5%)	17 (11.4%)
重大度２	1472 (19.7%)	92 (61.75)
重大度３	5119 (68.5%)	34 (22.8%)
重大度４	770 (10.3%)	6 (4.0%)
計	7473	149

表 3　TAT について

	全障害平均	Dfi 群障害平均	Dfi 以外の障害平均
TAT(日)	34.2	56.0	33.7

前述したように全障害の修正に要する期間（TAT）の平均は 34.2 日であった。そして、Dfi 群とそれ以外を比較すると、Dfi 群の TAT の平均は 56.0 日、それ以外の障害群では平均 33.7 日であり、これら Dfi 群とそれ以外の障害群の TAT の間には統計的に有意な差が認められた。（U検定 $p<0.05$）。このことより、Dfi 群を修正する作業とそれ以外の障害群を修正する作業は、本質的に異なる作業であることが推測される。

また、単純な概算では、１プロジェクトあたりの Dfi 群の平均障害対応工数は、およそ 1670 人日であった。

2.5 考察

Dfi 群の TAT の平均 56.0 日は、今回の製品群の反復開発の一回の単位である 6 週間を大きく上回る。一方、それ以外の障害群の TAT の平均 33.7 日は、１反復単位以下である。効率的な反復開発においては、一般的に、ある反復フェーズのテストによって発見された障害は、そのフェーズ内で障害を特定した上、次フェーズ計画にその修正計画を盛り込み、次フェーズにおいて修正作業とテストによる検証を行う。すなわち、本例においては、Dfi 群の障害は、修正には反復開発期間２回以上を要するような、解決困難な障害であった。

一般的に、早期の障害発見活動は、開発スケジュールを 60%まで短縮できるポテンシャルを持っていると言われている[6]。また、今回確認はできていないが、Dfi 群障害は、アーキテクチャ設計レベル障害であり、これら障害が原因となる別障害の発生の存在も推測される。そのため、Dfi 障害の早期解決は、大きな手戻り削減効果があると推測される。

これまで、一般的には、Dfi 障害を検出するようなソフトウェアアーキテクチャレベルの動的構造の妥当性検証の多くは、関係者による設計書レビュー、あるいは使い捨てプロトタイプのシミュレーションによって行われていた。レビューによる検証は、極めて経験的で、労働集約的な作業であるため、人為的な過ちがおこりやすいという問題が、そして、シミュレーションによる検証は網羅性に欠けたり、製品化時におけるプロトタイプとの不一致が発生したりする問題などが存在する。そして、いずれにしても、その検証の妥当性はソフトウェアのコーディングが終了した後のシステムテストが実行されるまで確認されることはなく、そこで初めて発見される動的構造の不具合は、最悪、多くの実装に影響を与えるようなアーキテクチャレベルの修正を必要とするような大きな「手戻り」を発生させることによって、開発効率に大きな影響を与える可能性があった。

従って、アーキテクチャ設計時に、Dfi 障害群の発生を検出可能な、労働集約的ではない、アーキテクチャ動的構造妥当性を検証する活動が、大きな手戻りを防止するためには重要であると考えられる。

3. SPIN によるアーキテクチャ検証の方法

本章では、前述したような、大きな手戻りを発生させる可能性が高い低再現性障害の発見をより早期に行うための、アーキテクチャ動的構造妥当性検証を目的とする初期開発活動を提示する。これは初期の開発成果物であるアーキテクチャモデルや文書レベルのソフトウェアアーキテクチャ設計を Promela 言語を用い、非決定性のチャネル通信オートマトンとしてモデル化し、モデル検査ツール SPIN を用い、その動的構造の妥当性を検証するものである。

3.1 検証内容

本検証活動では、ソフトウェアアーキテクチャの動的構造に関わる性質のうち、以下の 3 点の妥当性の検証を行う。

1. システム内部イベント駆動機構の妥当性
2. システム外部トリガーイベント対応の妥当性
3. イベント列の順序妥当性

システム内部イベント駆動機構の妥当性検証では、対象ソフトウェアアーキテクチャにおけるオブジェクトのコラボレーション、即ち、各オブジェクトが、システム内

Vol.2011-SE-172 No.5
Vol.2011-EMB-21 No.5
2011/5/18

部イベントを入力とし、処理や次のシステム内部イベント送出を含む制御の実行の連鎖が停滞することなく実行されることを目的とした検証である。ここでは、コラボレーション中に､デッドロックが発生することなく､コラボレーションが終了すること、そして、終了時には、そのシステムが期待どおりの終了状態に到達していることを検査する。この検査は、形式的検証技術における安全性、即ち、ある性質が常に成り立ち、想定外の問題は起こらないというシステム性質の網羅検査であり、モデル化されれば、SPIN によって自動検証可能である。

システム外部トリガーイベント対応の妥当性検証では、システムのモデル化のみならず、その実行される環境もモデル化し、環境から発生するシステムへの外部トリガーに対し、システムが対応可能であることを検証する。組み込みシステムは、システム外部からの割り込みや、ハードウェアからの割り込みなど、環境からの非周期的な刺激を受け、予測不能で複雑な動作タイミングで動作する可能性がある。これらの理由で、組み込みシステム開発ではユニットテストをパスした部品を、ただ組み合わせても動作するとは限らないということがよく知られている。また、これらに対応するためのシステムテストも、一般の業務系システムに比べて、テストデータの生成に自然法則や､ハードウェアなどの特殊な技術を要するため､多くの場合､実開発時には、ロングランテストという開発効率に大きな負担をかける工程を持たざるを得ない。モデル検査では、環境を対象システムと共にモデル化-モデル検査することによって、環境を含んだ網羅的な検証が可能である。即ち、例えば、無限の割り込みが発生する状況を、環境としてモデル化することにより、上流工程でこれらの工程をカバーする網羅検査も可能となる。この検査も、SPIN で自動検証が可能である。

イベント列の順序妥当性検証は、終了状態に到達するための、システム内部イベントの順序性の検証であり、動的構造に記述されたイベント（メッセージ）が期待された順序性を維持していることを検査する。この性質は、時間に関係する性質であり、網羅検査を行うためには、システム、環境のモデル化に加え、性質のモデル化を LTL（線形時相論理）式を用いて行わねばならない。

3.2 モデル検査の実行ステップ

本検証活動では、作業のステップを以下のように定義した。

1.「Promela プロセスの決定」
2.「プロセス間コミュニケーションの定義」
3.「プロセス内振る舞いの定義」
4.「検証すべき性質の確認」

ここで、1.2.3.は検証準備のプロセスと位置づけた。

1.「Promela プロセスの決定」は、検査対象を、SPIN が検査可能なチャネル通信オートマトンとして Promela を用いてモデル化するために必要な、検査対象の要素をどのようにモデルの基本要素であるプロセスに対応させるかを決定するステップである。ただし、モデル化は、抽象化であり、検査したい性質をモデル化の過程で捨象しないように注意が必要である。即ち、ソフトウェアアーキテクチャ記述の意図を捨象しないようにしながら、モデルを構築する必要がある。

2.「プロセス間コミュニケーションの定義」では、基本要素であるプロセスが決定された後、それらの間のコミュニケーションを定義し、Promela でモデル化する。これら２つのステップにより、検証モデルの静的な構造が決定する。

3.「プロセス内振る舞いの定義」は、上記したモデルの静的構造の上に動的構造をモデル化するステップである。具体的には、「Promela プロセスの決定」で得られた、モデル内の基本実行単位であるプロセス内に振る舞いを定義するステップである。

上記の３つの準備ステップ終了後に､「検証すべき性質の確認」において、検査を行う。検証の方策としては、大別すると、Promela の振る舞いそのものに対する網羅検査により､「望ましくない例＝反例」が発見可能なように Promela の記述を工夫する方法と、時相論理式（LTL）を用い、論理的に「成立してはならない条件」を記述した上で「望ましくない例＝反例」を発見させる方法がある。

4. SPIN によるアーキテクチャ検証結果

本章では、前記のアーキテクチャレベル動的構造の妥当性検証活動を実開発プロジェクトへ適用した知見を示す。

4.1 SPIN を用いたソフトウェアアーキテクチャ検証の方法

本研究では、3.1 検証内容で示した検査を目的としているため、基本的にはコラボレーション図のオブジェクトとして記述された、分析モデルを基本要素としている。

4.1.1 対象ソフトウェアアーキテクチャ

今回作業を適用したのはデジタル複合機制御ソフトウェアのためのソフトウェアアーキテクチャである。作業は、実際の製品開発と並行して行った、同程度のソフトウェア規模と考えられる先行開発製品においては、アセンブラ、C、C++によりコーディングされ、およそ 300 万 LOC、開発の最盛期には、同時に最大 250 人ほどのソフトウェアエンジニアが開発に携わった。

本ソフトウェアアーキテクチャは、開発容易性向上を主な目的として、既存ソフトウェアアーキテクチャを参考の上、新規に設計途中のものであり、ソフトウェアアーキテクチャの設計記述には全面的に UML を用いていた。また、本ソフトウェアアー

Vol.2011-SE-172 No.5
Vol.2011-EMB-21 No.5
2011/5/18

キテクチャは、基本的にイベント駆動方式をとっており、構造的には、レイヤパターン、パイプ＆フィルタパターンなどをアーキテクチャパターンとして採用している。

このソフトウェアアーキテクチャ記述において、その動的構造は、主に図 1 に示すような、UML1.4 に準拠したコラボレーション図を用いて記述されている。また、この動的構造は、本検証活動の実行時には、既にアーキテクトや、開発関連部門のレビューによってあらかじめ妥当性の確認作業が終了していた。

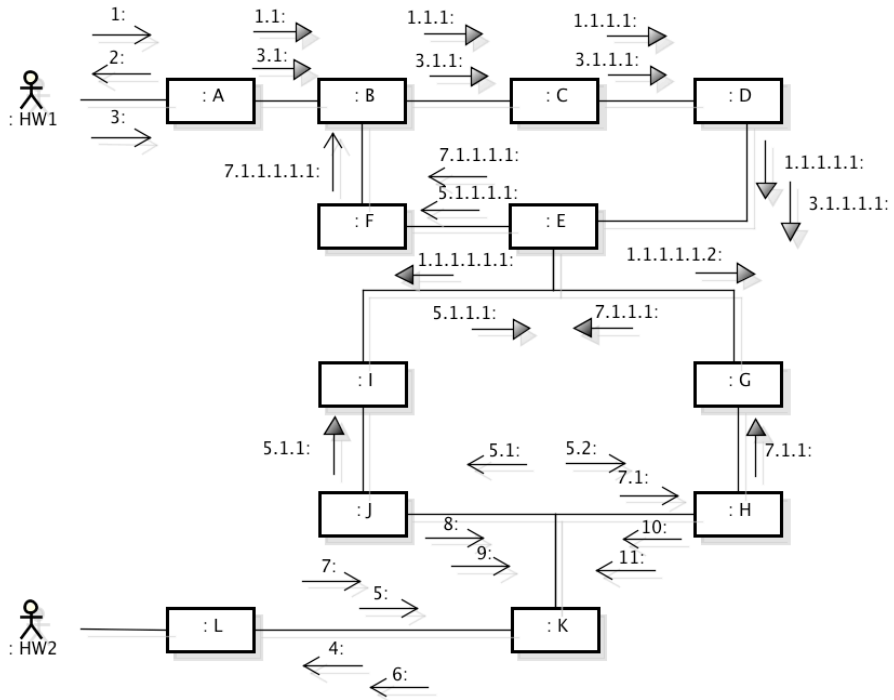

図 1 アーキテクチャ記述文書に記載された動的構造の例（名前などの詳細は省略）

4.1.2 モデル検査の準備ステップ

ここでは、3.2 モデル検査の実行ステップで述べた 1.2.3.の各ステップに関する具体例を挙げる。

1.「Promela プロセスの決定」では、基本的にはコラボレーション図のオブジェクトのうち､非同期メッセージ交換を行っているオブジェクトを基本プロセス要素とした。同期メッセージ通信を行っているオブジェクト同士は、同じ基本プロセス要素群として扱った。

2.「プロセス間コミュニケーションの定義」では、1.で定義した基本プロセス要素と見なすことのできるオブジェクト間リンクと、メッセージをコミュニケーションの定義とし、モデル化している。本ステップの詳細な作業フローを図 2 に示す。また、ここでのモデル化の際には、コミュニケーションの同期性について厳密に定義しなければならないため、モデル化の指針を作成した。

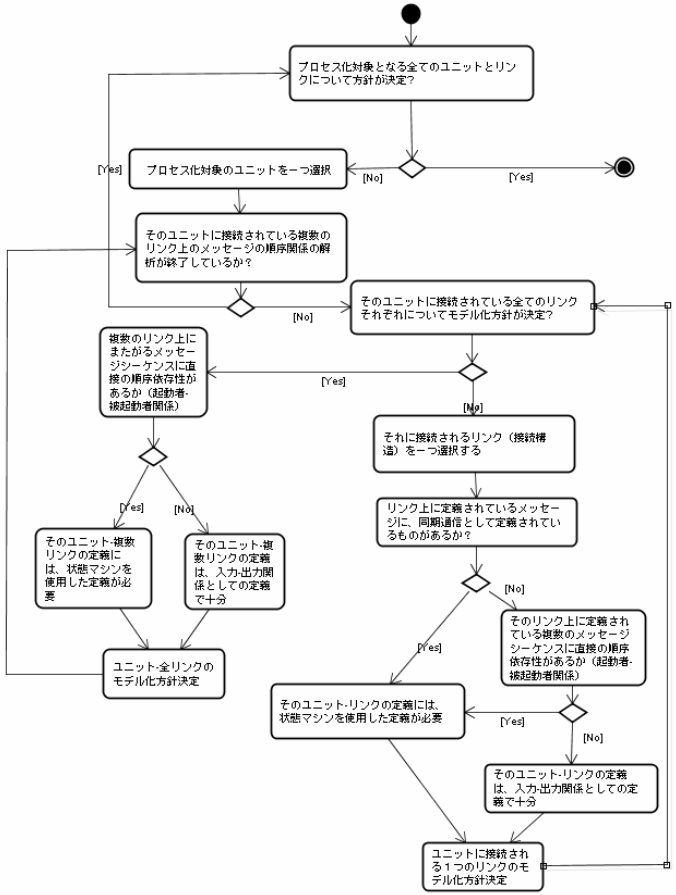

図 2　プロセス間コミュニケーションの定義

Vol.2011-SE-172 No.5
Vol.2011-EMB-21 No.5
2011/5/18

1. 2. の２つのステップにより、検証モデルの静的な構造が決定する。

続く 3.「プロセス内振る舞いの定義」において、コラボレーション図に記載されたメッセージシーケンス番号や、ノートに記された設計制約事項を Promela プロセスの中の振る舞いとして記述した。

4.2 SPIN によるアーキテクチャ検証活動の実開発適用とその結果について

4.2.1 得られた反例の扱いについて

本検証活動は、SPIN によるモデル検査から得られた反例より、「アーキテクチャ設計上の指摘項目」を抽出するということを目的として実行された。これは、今回の事例においては、モデル検査から得られた反例を実開発に適用するためには反例の意味する事を解釈するという行為が必要であったからである。即ち、今回得られた反例の発生が一意に、ソフトウェアアーキテクチャ設計上に、記述の問題や障害が存在していることを示しているとはいえないということである。

例えば、もし、これらの反例の発生理由が、予め、アーキテクチャルディシジョンとして「ソフトウェアアーキテクチャレベルでは未定義とし、詳細設計で解決すること」という判断に基づくものであれば、それはアーキテクトの想定内であるため、「ソフトウェアアーキテクチャレベルでの問題ではない」と判断される可能性が高い。

また、モデル検査は網羅検査を行うため、概念レベルでのモデル検査は、現実の製品では発生し得ない状態をも検査することが可能である。例えば、本活動中にも、人間であるユーザーが 1 秒間に 100 万回ボタンを押すといったような現実にはあり得ない検査を行ってしまったことがあった。このような場合、その反例が製品としての要求に対する反例であるかどうか、という判断も必要となる。

これらの理由により、今回の事例においては、モデル検査によって得られた「望ましくない性質」の発生可能性は、ソフトウェアアーキテクチャの障害としてではなく、アーキテクチャ設計上の指摘項目として、アーキテクトに報告されるものとした。この意味で、今回のモデル検査は、開発プロジェクト工程上はオプショナルなレビュー工程として扱われたといえる。

4.2.2 活動結果概要

本検証活動では、3 ヶ月間、3 人の作業で、52 個のモデルを構築し、36 種類の動的構造を検査した。そして、52 の検査のうち、30 の検査でアーキテクチャ設計上の指摘項目が得られた。

うち重要なものとして、デッドロックの可能性が 16 件、記述の不足・不備が 6 件、イベントの追い抜きが 3 件指摘されている。

52 個中 16 個のモデルは、同名のソフトウェアアーキテクチャ動的構造に対し、2 回目以上の繰り返し検査を行った。即ち、1 回目のモデル検査によりアーキテクチャ設計上の指摘項目が得られたもののうち、アーキテクトによって、ソフトウェアアーキテクチャレベルで明確に解が必要であるとの判断された動的構造の指摘事項に関しては、アーキテクトによって不足情報の追加や、動的構造に対する修正や洗練作業が行われ、再検査を実行した。

4.2.3 本活動の実開発適用の効果について

アーキテクトが、本モデル検査によるアーキテクチャ設計上の指摘項目を再検討することによって、主要なユースケースを実現する既存のコラボレーション設計に 1 件の明確な誤りがあることが判明した。即ち、タスクスイッチのタイミングにより、デッドロックおよび、イベントの追い抜きを引き起こす可能性を持つイベントパスが、主要ユースケースを実現するコラボレーションに存在していることが判明した。このユースケースは、システムに要求される最も基本的なフィーチャに関わるようなユースケースであった。それゆえ、他の検査により得られたアーキテクチャ設計上の指摘項目の多くは、排他処理などの実装技術により回避可能であるとアーキテクトによって判断された一方で、上記の問題は、実装レベルの手直しでは回避不能であり、ソフトウェアアーキテクチャレベルで、分析オブジェクト間の関連構造の修正が必要と判断され、開発初期にこの部分のアーキテクチャの修正がなされた。

又、特定の数個のオブジェクトについては、特にアーキテクチャ上の指摘が多かった。これらオブジェクトは、様々な主要な機能コラボレーションの交点であり、いわゆる機能の交通整理をするような制御オブジェクトであった。アーキテクトの判断としては、これらは、単一の製品開発という視点からは、設計レベルで個々の指摘への対応を行うという判断もあり得るが、今後の保守なども考慮した開発効率の観点より、ソフトウェアアーキテクチャレベルで、このオブジェクトの状態遷移を厳密に再定義する必要があるというものであった。

これらの項目は、ソフトウェアアーキテクチャ設計時の担当者レビューによって見落とされていた項目であるため、本検証活動なしでは、これらの項目が原因となるような問題は、実装後のシステムテストまで発覚しなかったであろう事が推測される。

5. まとめと考察

産業界における組み込みソフトウェア開発プロジェクトでは、品質、コスト、納期という３つの条件すべてを満たさなければプロジェクトの成功とは言わない。そして、このうち、特にプロジェクトマネージメントの観点からコストと納期の側面で悪影響を与えると言われているものが「手戻り」である。

Vol.2011-SE-172 No.5
Vol.2011-EMB-21 No.5
2011/5/18

本論文では、その原因の一つである低再現性障害の多くの原因が、開発初期に定義すべきアーキテクチャ設計における動的構造の不備であることを示した。

そして、これまで、多くの産業界において、上記のようなソフトウェア開発上流工程成果物の妥当性は上流工程での労働集約的なレビュー、プロトタイプにてのシミュレーション及び、下流工程での統合テストなどで検証されてきたため、この不備に起因する膨大な「手戻り」が発生してきたと考えられる。

そこで、これらの障害を、可能な限り早期に除去する事を目的として、モデル検査ツールである SPIN を用いた、ソフトウェアアーキテクチャ動的構造の妥当性検証活動を提案し、実際の開発に対して試行を試みた。その結果、多くのアーキテクチャ設計上の指摘項目を得ることが出来た。そして、これらの中には、レビューにより見落とされていた重要なソフトウェアアーキテクチャ上の回避不能な障害も含まれるとともに、誤った実装を行えば、クリティカルな障害を引き起こす可能性があることを示唆するような指摘点も存在しており、SPIN の実用性について確認できた。

この事実は、ソフトウェア開発へのモデル検査技術の現実的な適用可能性を意味しているとともに、組み込みソフトウェアの大規模化、複雑化に伴う開発困難性を減少し、開発効率を改善するための、上流工程成果物の検証作業の重要性も示唆していると考えている。

ただし、このような新規活動を産業界で適用するためには、明確なコスト対効果を提示する必要があると考える。本事例では、コストに関する単純な概算では、1 プロジェクトあたりの Dfi 群の障害対応工数は、およそ 1670 人日であり、Dfi 群を、検出、除去可能とする、SPIN によるアーキテクチャ検証の工数は、270 人日であったため、1 プロジェクト当たり、少なくとも、およそ 1400 人日(47 人月)の工数が削減可能であると予測される。しかし、Dfi 障害群はアーキテクチャレベルの開発に極めて大きな影響力ある障害で、他障害の原因となる可能性も極めて高く、チーム開発の側面から見れば、その効率に極めて大きな影響を与えていると想定される。本論文では、Dfi 障害群対応が開発プロセス上、開発工程を停止させたり、停止させたりする可能性や、相当量の開発工程遂行にかかわる管理オーバーヘッドなども考慮されていないため、十分なコスト対効果を提示はできていない。

今後、これら活動の実施のためには、本活動の自動化も視野に入れた効率化なども含んだコスト対効果の明確化も重要な課題と考えている。

謝辞

北陸先端科学技術大学院大学の片山卓也教授には、形式検証についてご教授を頂くとともに、本論文をまとめるうえで、全幅のご支援をしていただきました。
また、青木利晃特任准教授には、SPIN のご紹介とご教授を,落水浩一郎教授には、本論文をまとめるにあたって、様々なご示唆をいただきました。

日本アイ・ビー・エム株式会社グローバル・エンジニアリング・ソリューションの組み込みソフトウェアソリューション浜谷千波次長には、本検証活動の実製品開発への適用、および本論文の発表をご許可いただきました。
山下雄二、北誉志彦、浅田千織の諸氏は、実行チームとして本検証活動の実行への御尽力をいただきました。
そして、私どもが開発業務をさせていただいたそれぞれの組み込み機器メーカー様のご協力頂いた皆様に，謹んで感謝の意を表します.

参考文献

1) B. W. Boehm , P. N. Papaccio, Understanding and Controlling Software Costs, IEEE Transactions on Software Engineering, Vol.14, No.10, p.1462-1477 (1988).
2) Grady, Robert B.: An Economic Release Decision Model: Insights into Software Project Management; In Proceedings of the Applications of Software Measurement Conference, pp.227-239 (1999).
3) Jerry Krasner : The Economics of Embedded Development, Testing, Deployment and Support , Embedded Market Forecasters, www.embeddedforecast.com, (2009)
http://www.eetimes.com/electrical-engineers/education-training/tech-papers/4137722/The-Economics-of-Embedded-Development-Testing-Deployment-and-Support
4) 中島 震, 玉井 哲雄 : EJB コンポーネントソフトウェアアーキテクチャの SPIN による振舞い解析, コンピュータ・ソフトウェア, Vol.19, No.2, pp.2-18 (March 2002).
5) Paola Inverardi, “Automated Check of Architectural Models Consistency Using SPIN” Automated Software Engineering archive, Proceedings of the 16th IEEE international conference on Automated software engineering, pp346 (2001).
6) Gilb, Tom. Principles of Software Engineering Management, Addison-Wesley, Wokingham, England(1988).