# Use CASEを利用したソフトウェアフォールトに対する SS-FTAの提案

平山雅之*1,2，岸本卓也*3，水野修*2，菊野亨*2

*1 (株)東芝　研究開発センター　システム技術ラボラトリ
phone：044-549-2403
e-mail: masayuki.hirayama@toshiba.co.jp
*2 大阪大学　大学院基礎工学研究科　情報数理系専攻
*3 (株)東芝　家電機器社　研究開発部

**あらまし**　ソフトウェアシステムへの依存度が高まるにつれ，高い水準の信頼性がそれらのソフトウェアに求められるようになってきている．ソフトウェアの信頼性は主にテストの繰り返しによって確保されるので，そこで利用されたテスト項目によって保証される信頼性のレベルは大きく異なってくる．このため信頼性の観点から最適なテスト項目を求めることがソフトウェア開発現場におけるテストフェーズでの重要な課題の一つとなっている．本論文ではテスト項目の作成のための新しい方法を提案する．提案法ではまずテスト対象となるソフトウェアの振舞いをユースケースで記述する．次にそのデビエーション分析を行い，ソフトウェアのフォールト木を作成する．このフォールト木を利用してテスト項目の集合を作成する．提案法と従来法で作成されたテスト項目を比較評価した結果，従来のテスト項目に対する重複や抜けを提案法では効率的に検出・指摘できることを確認した．

**キーワード**　ユースケース，テストケース設計，FTA

# Generating Test Items by Applying Software Fault Tree Analysis

Masayuki Hirayama*1,2, Takuya Kishimoto*3
Osamu Mizuno*2, Tohru Kikuno*2

*1 R&D Center System Engineering Lab., TOSHIBA Corporation
*2 Department of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University
*3 R&D Group. Home Appliance Company. TOSHIBA Corporation

**Abstract**　Recently software systems are extensively used in our social life and then high reliability is required to be attained in such software systems. Generally, software testing is a key approach toward high reliability and thus the resultant reliability deeply depends on the quality and quantity of test items used in software testing.

In this paper, we propose a new method for generating test items. The proposed method consists of the following steps: (1) Describe software behavior using Use-case notation, (2) Apply deviation analysis to the description, (3) Construct software fault tree using analysis result, and (4) Generate test items based on software fault tree. As a result of experimental evaluation, it is shown that the proposed method can detect both the overlapping and the lack of test items obtained by conventional method.

**Keywords**　Use CASE, Test Case Design, Fault Tree Analysis

## 1　まえがき

近年，ソフトウェアシステムへの依存度が急速に高まり，これに伴いより高い信頼性がソフトウェアに求められるようになっている．ソフトウェアの信頼性は一連の開発作業の中のテストフェーズでの作業によって確保される．このためより高い信頼性レベルを保証しようとする場合，対象とするソフトウェアに関してどのような項目についてテストを行うかが重要な課題になる．ソフトウェアに対するテスト項目を定める作業は，”テストケース設計”として認識される．このテストケース設計に関してソフトウェアの信頼性を考慮した手法は必ずしも十分に整備されているとは言い難い．

ソフトウェアのテスト項目は，一般的にそのソフトウェアで実現が求められている機能仕様や動作仕様などをもとに作成される．従来の研究の多くはフォーマルな形式で表現されたソフトウェア仕様をもとに網羅的なテストケースを生成していた[1]．これらの方法は制約条件が厳しく必ずしもあらゆるタイプのソフトウェアに当てはまるものではなかった．

本研究では自然言語で記述されたソフトウェア仕様を対象に，デビエーション分析技術や FTA(Fault Tree Analysis)のソフトウェアへの適用を試みる．具体的にはこれらの技術を融合した新しいソフトウェアテストケース設計方法論の整備を目指している．

本報告ではこのテストケース設計方法論のうちで特にソフトウェア FTA の実施方法に焦点をあてた提案をする．提案する方法では自然言語で記述されたソフトウェア仕様書にもとづき，信頼性の観点からソフトウェアの代表的な動きを UML(Unified Modeling Language)[2]で提案されているユースケース記述を拡張した記法で表現する．次にこのユースケース記述に対してデビエーション分析の技術を適用して異常動作の要因分析を行う．この要因分析に基づいてソフトウェアに対するフォールト木を作成する方式を提案する．最後に，ソフトウェアフォールト木の分析を行ってソフトウェアのテスト項目を導く．本報告では，実際の冷蔵庫制御ソフトウェアの一部を題材に用いて，上記の方法論を検討した結果を示すとともに，ソフトウェアフォールト木から導き出されたソフトウェアのテスト項目の評価結果についても紹介する．

## 2. 従来のソフトウェアテストの問題点

従来のソフトウェアテストに関しては次に示すような問題が考えられる．

**(1) 不具合の存在確率の違い**

ソフトウェアの不具合は一般に対象ソフトウェアの各部分において均一には存在しない．即ち，ソフトウェアの部分によって不具合が混入しやすいところと，しにくいところがある．例えば，新人の作成した部分や新規開発した部分には不具合が混入しやすく，ベテランが作成したり過去のソフトウェアを流用した部分には不具合が少ない．また個々の不具合を考えた場合，その不具合によるフィールドへの影響が大きいものと小さいものがある．これは対象ソフトウェアが実現する”機能”間にも重要なものと，付帯的なものなど様々な重要度が設定できることを意味している．現状のソフトウェアのテストを考えた場合，こうした機能の重要度や発生不具合の影響度，あるいは不具合の混入率などを体系的にとらえてテストケースを作成することはほとんどない．このため本来は不具合が余り混入していない箇所を，又は製品ソフトウェアにとってさして重要でない箇所を一所懸命にテストするといった的外れなテストが行われる場合も少なくない．

**(2) 網羅率に偏重したテスト**[3]

従来のソフトウェアテストでは，対象とするソフトウェアの論理的動作や機能を極力網羅することが至上命題として与えられ，網羅率をあげる方式が重視されてきた．しかし，近年のソフトウェア規模の増大に伴い，ソフトウェアが実現する全ての動作を網羅してテストケースを作成することは難しくなりつつある．また仮にこうしたテストケースの全てを作成できたとしても，限られたテスト期間内にこれらの膨大なテストケースの全てをテストすることは実質上も不可能に近い．網羅率に依存したテストでは最悪の場合，システムの信頼性にとって最も重要であるテスト項目が作成されなかったり，時間切れでテストされなかったり，という深刻な事態を引き起こしかねない．

**(3) 不具合経験の体系的な利用**

通常の製品ソフトウェア開発では過去に類似した開発を踏まえたものが多い．この場合，過去の類似開発で発生した不具合を参考にして，テストケースの作成などを行う必要がある．しかし実際の開発現場ではこうした過去の不具合を体系的に分析し，次回以降のテストケースに反映するための手法が整備されていない．そのため必ずしも十分にこうした過去の不具合の経験がテストケースに反映されていない．なお不具合の分析手法として FTA(Fault Tree Analysis)が代表的な手法として知られている．これまで FTA は主にハードウェアの故障分析に利用されてきており，FTA をソフトウェアに適用するためには分析手順などの見直しと整備が必要となる．

**(4) テストケース作成作業**

ソフトウェアのテストケースは，一般にソフトウェア動作仕様をもとに作成される．通常，自然言語で記述されたソフトウェア動作仕様をテストケース作成担当者が理解し，テストケースを逐次的に生成する．そのためこの作業には多大な労力を要する．さらに，これらのテストケース作成の作業を行う担当者のスキルによって，作成されるテストケースの質にばらつきが生ずるといった問題がある．また，こうして作成されたテストケースについては抜けや重複が発生することも少なくない．

## 3. テストケース設計手法の概要

ここでは 2 節で述べた課題の解決を目標としたテストケース設計手法について紹介する．

### 3.1 全体の流れ

提案するソフトウェアテストケース設計手法(S-TEMRA: *Selective Testing Method based on Reliability Analysis*) の全体の流れを図 1 に示す．S-TEMRA は図 1 に示すように全体で５つのステップから構成される．

Step-1: ソフトウェアシステムの動作仕様の理解
Step-2: ソフトウェア動作のユースケース記述
Step-3: デビエーション分析
Step-4: システム＆ソフトウェア故障分析(SS-FTA)
Step-5: テスト項目の抽出

### 3.2 各ステップでの処理

**Step-1: ソフトウェアシステムの動作仕様の理解**

ここではテスト対象となるソフトウェアシステムがどのような構成で，どのような動作を行うかについて概要理解を図る．そのための道具として，ソフトウェアの機能構成を可視化するソフトウェアブロック図，および，ソフトウェアが影響を及ぼしたり受けたりする周辺のハードウェアに関するブロック図を利用する．ソフトウェアブロック図に関しては一般的なデータフロー図などで代用しても構わない．

**Step-2: ソフトウェア動作のユースケース記述**

対象ソフトウェアが実現する機能の代表的な動作や信頼性上重要となる動作をユースケース記述を利用して明確にする．ユースケース記述はオブジェクト指向開発方法論 UML [2]の中で提案されたソフトウェア動作の記述表現形式の一つである．ここではこのユースケース記述の表現形として，後述する拡張アクティビティチャートを利用する．

**Step-3 :デビエーション分析**

ユースケース記述で明確にされたソフトウェアの基本動作に対し，異常な状況下での動作をデビエーションとして抽出する．更に，これらの基本動作から逸脱して異常をきたす動作も抽出する．デビエーションは安全性工学[4]の領域で利用されている概念で，”逸脱”を意味する．ここではデビエーション抽出の手法として N. Leveson らが提唱している「ガイドワード」[5]を利用する．

**Step-4: システム＆ソフトウェア故障分析(SS-FTA)**

デビエーション分析の結果を利用し，ソフトウェアに対するフォールト木 (SS-FTA : System and Software FTA) を作成する．SS-FTA では Step-2, 3 での分析結果に含まれる用語を利用して，対象ソフトウェアにとって望ましくない事象が起きるケースをソフトウェアの内部処理と関連付けて分析する．ソフトウェアフォールト木のルート(最上位)事象としては当該ソフトウェアにとっての望ましくない事象を選ぶ．実際には過去の類似開発において発生した不具合の中からフィールドへの影響度などを考慮して決定する．

**Step-5: テスト項目の抽出**

SS-FTA の結果を利用して，ソフトウェアフォールト木のリーフに記載された要因を抽出し，この要因をチェックするテスト項目を作成する．こうすることによって，対象ソフトウェアにとって望ましくない事象を引き起こす全ての要因に対するテストが可能となる．

S-TEMRA は上記に示した 5 つの段階を経て最終的にソフトウェア信頼性を考慮したテスト項目を生成する．なお Step-2, 3, 4 の分析が担当者の技量に依存することを少なくするために，分析過程で利用した用語を統一し，用語辞書の形で管理することも合わせて採用する．

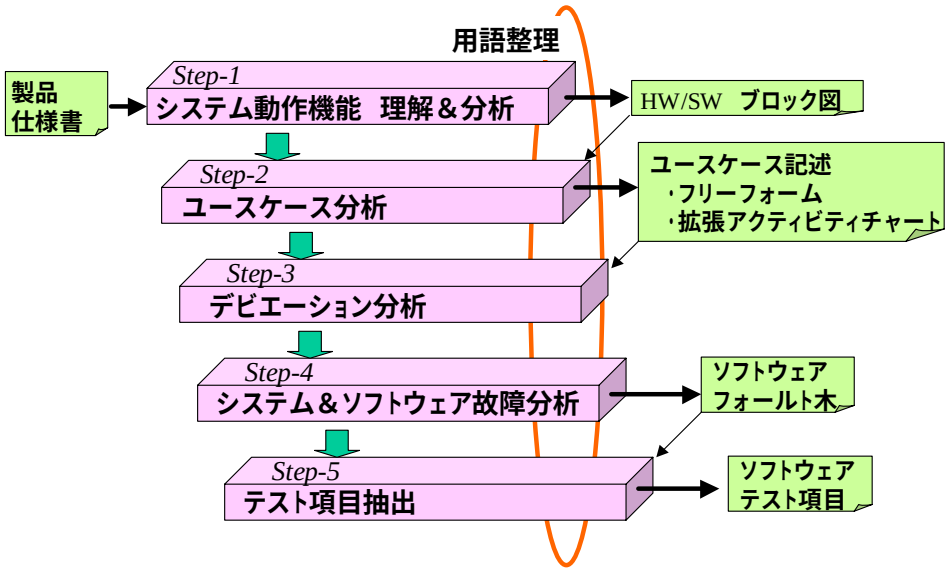



図１　　全体の流れ

## ４． テストケース設計手法

ここでは冷蔵庫制御用ソフトウェアを例に，ユースケース記述，用語整理，デビエーション分析，ソフトウェアフォールト木作成の各段階について詳細に説明する．

### 4.1 ユースケースの分析

冷蔵庫制御用ソフトウェアの製氷機動作に関するユースケース記述を図 2, 3 に示す．ユースケース記述では，図 2 のような自然言語によるフリーフォームのユースケース記述を作成した後に，図 3 に示す拡張アクティビティチャートを利用して記述する．

**(a) フリーフォームによるユースケース記述**

フリーフォームによるユースケース記述では唯一の記述制約として，機能動作の前提条件，サブ機能名，機能動作の詳細記述という枠組みを設けている．例えば，製氷動作では「製氷皿に水，氷が入っていない」という前提条件のもとで，大きな動作の流れとして「給水動作」「製氷動作」「離氷動作」「製氷装置水平戻し動作」が順次実行されることが分かる．また，「給水動作」では<給水，X 秒実施>などの個別動作が[給水タイマ，X 秒設定], [給水モータ，正回転]などの操作によって順次行われ，一つの動作として完結していく．

機能動作 ：製氷機能

| 前提条件 | 継続動作のなかで（前回サイクルにおいて離氷動作完了後）<br>製氷皿に水、氷が入っていない状態で |
|---|---|

| サブ機能 | 機能動作の詳細（振舞い） |
|---|---|
| 給水動作 | 振舞い-11 製氷皿 [給水タイマ,X1秒設定] [給水モータ,正回転]<br>振舞い-12 給水後 [給水タイマ,X2秒設定] [給水モータ,逆回転] |
| 製氷動作 | 振舞い-21 給水完了後 [製氷動作,開始する]<br>振舞い-22 [皿温度 -T以下] [製氷動作,完了] [離氷動作開始] |
| 離氷動作 | 振舞い-31 [製氷皿,Z2度回転] [貯水検知動作、開始]<br>振舞い-32 [貯水室,満杯でない] [離氷動作,継続]<br>振舞い-33 [製氷皿,正回転Z2度] |
| 製氷装置水平戻し動作 | 振舞い-41 [皿タイマ,X3 秒設定] [製氷皿水平戻し動作,開始] [皿回転モータ、停止]<br>[製氷皿水平位置出し動作,開始] |

図２　ユースケース記述（フリーフォーム）

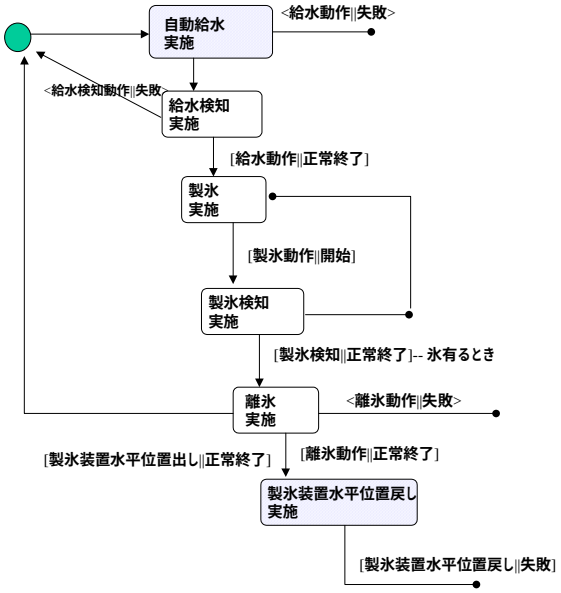
**(b) 拡張アクティビティチャートによるユースケース記述**

次に，フリーフォームのユースケース記述を参考に拡張アクティビティチャートを作成する．

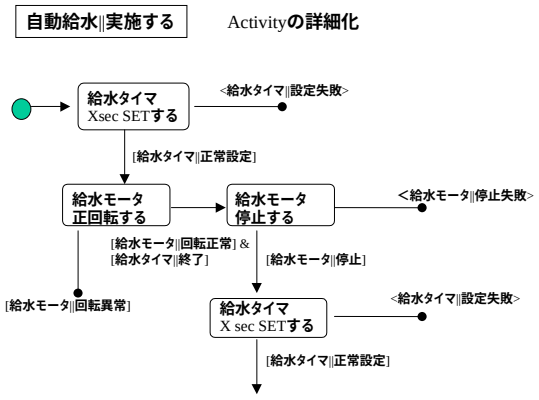
アクティビティチャートは UML でその記述様式が定められているが，S-TEMRA では信頼性要求分析のしやすさなどを考慮して以下の拡張を加えた．

- 個々のアクティビティは，例えば「自動給水を実施する」というように「XX を YY(する)」という形式で整える．
- アクティビティのトリガーについてはソフトウェアの制御対象オブジェクトとそれがとり得る状態に着目し，[給水モータ|回転正常]などのように記述する．
- 拡張アクティビティチャートではソフトウェアの基本動作(アクティビティ)の流れを矢印→で記述し，これに対して，個々のアクティビティに関する異常動作を矢印 ——● で記述する．こうすることによって，ソフトウェア異常動作を明示する．

図 3(a)は冷蔵庫制御ソフトウェアの基本動作を拡張アクティビティチャートで表記したものである．基本アクティビティ「自動給水，実施（する）」「製氷，実施（する）」などが順次実行されることが分かる．また，図 3(b)はこの「自動給水，実施（する）」をさらに詳細に展開記述したものである．ここでは「給水タイマ，X 秒 SET する」「給水モータ，正回転する」などの下位のアクティビティを利用して順次詳細化を進めている．




(a) 全体記述




(b) 詳細記述

図３　ユースケース記述 (拡張アクティビティチャート)

### 4.2 用語の共通化

ソフトウェア故障分析では分析担当者が異なった場合に分析の粒度が異なる可能性がある．このため SS-FTA ではソフトウェアのユースケース分析，デビエーション分析，フォールト木作成の各段階で利用される用語に制約を設け，共通化を図る．こうすることによって分析担当者による粒度の違いを最小限に押さえている．

用語は[オブジェクト||とり得る状態]という基本的な枠組みで整理する．用語の共通化を進めるため，まず，オブジェクトの洗い出しをおこない，次いでこれらのオブジェクト毎のとり得る状態の洗い出しを行う．

**(a) オブジェクトの洗い出し**

”オブジェクト”としては制御用ソフトウェアの制御対象となりうる次のようなものを考える．

**ファンクション・オブジェクト**

ソフトウェアが提供，実現する機能そのもの．例えば，「製氷動作」「離氷動作」などを指す．

**ハードウェアデバイス・オブジェクト**

ソフトウェアが制御する対象としての実態（物）があるもの．例えば，「コンプレッサー」「給水モータ」などがこれに当たる．なお，内部制御で利用されるソフトウェアタイマなども例外的にこのカテゴリのオブジェクトとして扱う．

**データ・オブジェクト**

ソフトウェアが扱ったり必要とする主に外部のデータ．例えば，「外気温」「庫内温度」など．

**ファームウェア・オブジェクト**

制御ソフトウェアの実装上の都合で必要とされる内部の変数やフラグなど．

**(b) オブジェクトのとり得る状態の抽出**

次に，抽出された個々のオブジェクトのとり得る状態の洗い出しを行う．例えば，ファンクション・オブジェクトの 1 つである[給水動作]の場合，とり得る状態として<開始する/正常終了する/失敗する>といった３つの状態を抽出できる．

これらのオブジェクトやそのとり得る状態の抽出は，ユースケース分析，デビエーション分析，フォールト木作成の各段階で追加，修正を繰り返しながら行い，最終的には分析担当者間で共通に利用できる用語辞書の形で整備していく．

## 4.3 デビエーション分析

デビエーション分析ではソフトウェアの基本動作から逸脱したことによる異常動作の分析を主として行う．分析に当たっては特に外部とのインタフェース，ユーザ操作の２つの視点に着目する．

外部とのインタフェースでは入出力データに関する値，データ入出力タイミングなどがシステムの異常動作につながる要因と考えられる．一方，ユーザ操作はソフトウェアにとっては最終的には入力データの誤りとなるので，ユーザ操作ミスなどについて体系的に分析することは異常動作を考える上で有効である．

これら 2 つの視点に注目し，ソフトウェアに対する異常動作を導き出すことを目的として「ガイドワード」[4]を利用する．「ガイドワード」は図 4 に示すような，システムの正常動作仕様に対する異常を導き出すためのキーワードである．

図 4 はガイドワードの一例を示したものである．これ以外のガイドワードについても次に示すようなソフトウェアの動作状況を念頭において用意する．

・周辺ソフトウェアの故障
・周辺ハードウェアの故障
・ソフトウェア環境の異常(OS，ミドルウェア，ライブラリなどの誤動作や不具合)
・周辺外部環境の異常

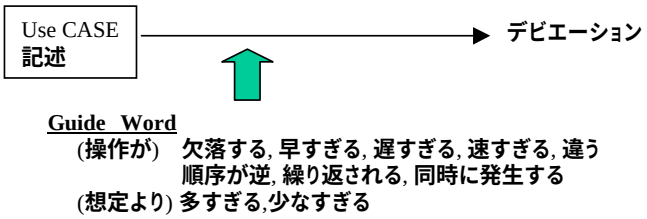



図 4　　ガイドワード

例えば，冷蔵庫製氷機能を考えた場合には，ソフトウェアの異常動作を引き起こす要因として次のものが考えられる．製氷機能の周辺にあたる冷凍サイクル制御におけるコンプレッサー制御ソフトウェアの異常やコンプレッサー自身のハードウェア異常，あるいは利用しているリアルタイム OS のバグや，製氷機外部の周辺温度の異常高温などである．

これらの要因の抽出にはガイドワードを利用する．先に作成したユースケース記述(テスト対象ソフトウェアの主たる動作)に対して，ソフトウェアの動作を順に机上トレースしながら，このガイドワードを適用して正常動作状態からの逸脱(デビエーション)を抽出していく．

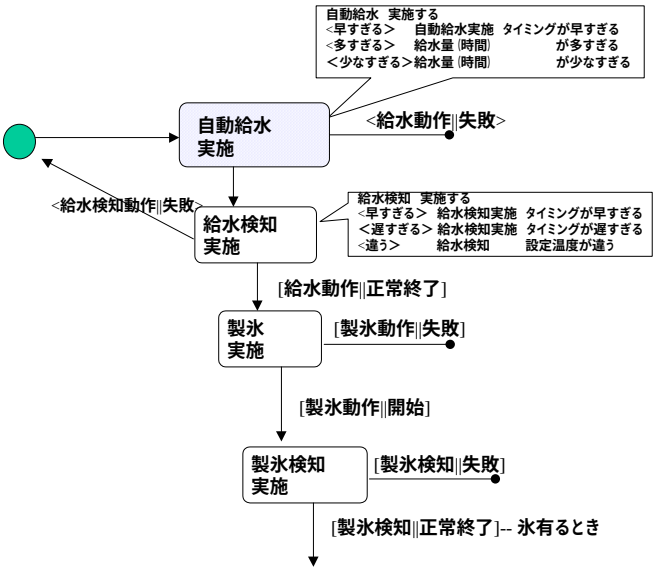



図 5　デビエーション分析

図 3 に示したユースケース記述(拡張アクティビティチャート)に対して，このガイドワードを当てはめてデビエーションを抽出したものを図 5 に示す．例えば，アクティビティ[自動給水を実施する]に対しては，<早すぎる>というガイドワードの適用により，『自動給水実施　タイミングが　早すぎる』というデビエーションが導かれる．同様に，<多すぎる>というガイドワードの適用により『給水量(時間)　が　多すぎる』を，<少なすぎる>というガイドワードの適用により『給水量(時間)が　少なすぎる』をそれぞれ導くことができる．

### 4.4 ソフトウェアフォールト木の作成

抽出されたソフトウェアデビエーションとソフトウェアフォールトの因果関係をつけることによりソフトウェアフォールト木を作成する．ソフトウェアフォールト木は最上位のルート事象に当該ソフトウェアにとっての不具合事象をとり，この事象が発生するケースについてソフトウェアデビエーションとの対応付けをとりながら逐次展開していく．

不具合事象の展開は「機能不具合の展開」と「ソフトウェア不具合への展開」の 2 つのステップに分けて行う．

#### (1) 機能不具合の展開フェーズ

通常，ソフトウェアフォールト木のルート事象はシステム（ソフトウェア製品）が提供するサービスの観察可能な不具合事象を取り上げる．「機能不具合の展開フェーズ」では，取り上げた不具合事象とこれを引き起こすソフトウェアの機能上の不具合を対応させる作業をおこなう．実際には，ユースケース記述で示したソフトウェアの機能動作の流れをトレースして，ルート事象の不具合の原因となりうる機能動作の不具合事象を抽出し，細分化していく．

例えば，冷蔵庫制御ソフトウェアに関する製氷機能に関しては，一般ユーザが認識する望ましくない不具合事象として「氷ができない」という不具合事象が考えられる．これを引き起こす不具合事象として，「水が製氷装置にたまらない」「製氷中に水こぼし」などが考えられる．ここで「水が製氷装置にたまらない」という事象を考えると，その原因として「製氷装置が正常にセットされない」「給水自体が失敗した」が考えられる．このうち，「製氷装置が正常にセットされない」については，「製氷装置のセット/リセットを検知できない」場合と「製氷装置水平位置戻し動作不正」場合の 2 つに分けられる．後者は先のユースケースで記述した「製氷装置水平位置戻し動作」に対応していることが分かる．更にこの「製氷装置水平位置戻し動作が不正」というのは，先のデビエーション要求分析の結果を参考に，「製氷機が水平状態を行き過ぎる」「製氷機が水平状態に達しない」「製氷機の回転が止まらない」「製氷機が回転しない」などの原因に分けて考えることができる．

#### (2) ソフトウェア不具合への展開

「ソフトウェア不具合への展開」フェーズではソフトウェア機能の不正をさらにソフトウェア実装上の不正のレベルにまで詳細にブレークダウンする．

例えば，先の「製氷機水平戻し動作が不正」という事象に関しては「製氷機が水平状態を行き過ぎる/達しない」「製氷機回転が止まらない/回転しない」などの機能的な不正がソフトウェアとしてどのような誤りをした場合に発生するかを分析していく．ここで「製氷機が水平状態を行き過ぎる/達しない」という事象が発生するのは，製氷機の水平状態を監視しているセンサー信号に関連し，

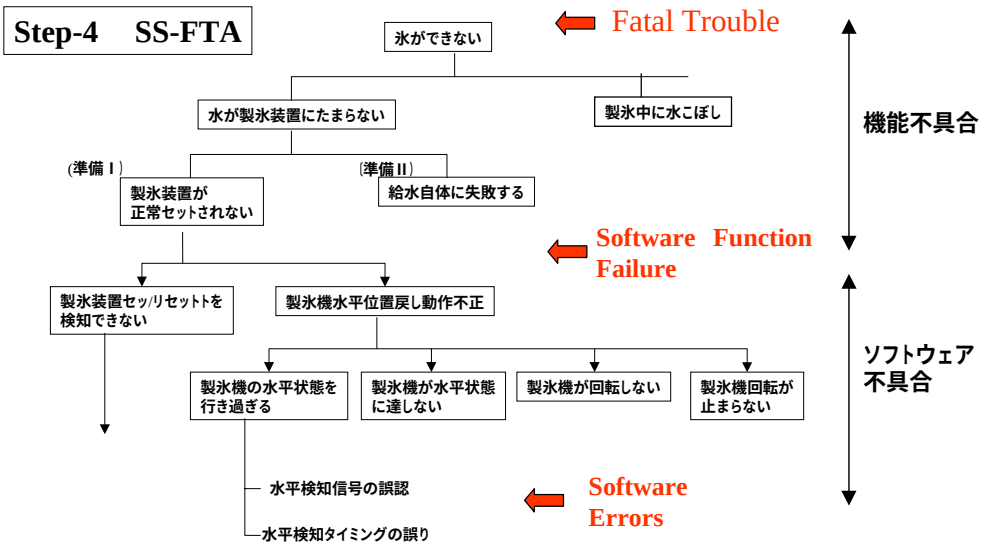



図 6　　ソフトウェアフォールト木

① 水平検知信号を誤認した.
② 水平検知のタイミングがおかしい.
などがその要因として考えられる.
実際のソフトウェアでは水平検知のタイミングはソフトウェア側からのリクエストで指定し，検知信号を受ける形となっている．そのため①については信号データの読み間違い（アドレス誤りなど），②については水平検知を指示する箇所の誤りなどが原因となる．このようにして，ソフトウェアの機能動作の不正を引き起こす要因を分析し，ソフトウェア実装上の不正という最もプリミティブな要因まで展開していって，これらをソフトウェアフォールト木の形で整理していく.

### 4.5 ソフトウェアテスト項目の抽出

作成したソフトウェアフォールト木を利用してテスト項目の抽出を行う．既に説明したようにフォールト木のリーフには，最上位のルート事象に記載された不具合事象を引き起こすソフトウェア実装面での不正が記載される．テスト項目としてはこのリーフのアイテムを検出可能なものを考える．例えば，前述の①製氷装置水平位置検知信号に関するテストケースとして「水平検知信号は正常値が設定されているか」というテスト項目が作成される．実際のテストではソフトウェア実行時に信号データをモニタし，実際の装置動作に合っていることを確認する

このようにして作成したソフトウェアフォールト木をもとに，最上位のルート事象に関するテスト項目を作成することが可能となる.

## 5. 適用実験

### 5.1 対象ソフトウェア

前節までに提案したテスト項目設計手法を冷蔵庫制御ソフトウェアの一部に適用した例について述べる．本実験では製氷機能の一部に対し提案手法を適用してテスト項目を作成した．この生成されたテスト項目と，従来のチェックアイテムを用いて作成されたテスト項目との比較を行う.

今回の試行ではソフトウェアフォールト木のルート事象として「氷ができない」という事象を設定した．そしてこれを展開した機能不具合である「製氷装置初期位置設定動作」に関してさらに詳細に展開したソフトウェアフォールト木を作成する．このフォールト木に基づいてソフトウェアのテスト項目の作成を行う.

### 5.2 適用結果

このソフトウェアに提案法を適用した結果，次の 2 つの効果を確認することができた.

(1) より具体的なテスト項目の作成

「製氷装置初期位置設定動作」に関して，従来のテスト項目では『離氷動作終了後の製氷装置水平復帰時の水平位置検出を確認』という項目があり，水平検知信号確認によりこの確認をしていた．これに対し，今回の方法では「ソフトウェア側での製氷装置水平位置検知のタイミングは適切か」「製氷装置水平位置検知センサーの値は正常値か」といったソフトウェア実装により近いレベルでのテスト項目が作成された.つまり，ここで作成されたテスト項目はより具体的なものとなっている.

(2) 従来法で発見できなかった不具合の検出

提案法で作成したテスト項目を品質保証段階での製品チェックで指摘された不具合項目と対比させてみた．その結果，「製氷装置水平位置検出」に関する不具合に関して，「水平位置センサーの検出タイミング誤りに起因する不具合」が品質保証部門の検査で指摘されていることが確認できた．つまり，この不具合は設計部門でのテストにおいては「離氷動作終了後の製氷装置水平復帰時の水平位置検出を確認」というテスト項目では検出できず，後の工程である品質保証部門での検査において検出されたものであった．今回の方法で抽出された新しい 2 つのテスト項目「ソフトウェア側での製氷装置水平位置検知タイミングは適切か」「製氷装置水平位置検知センサーの値は正常値か」をデバッグ段階で適用すれば，これらの不具合を開発部門内で事前に検出することが期待される.

## 6. 評価

ここでは，提案法の実ソフトウェアへの試行実験を通して得られた成果について考察する.

### 6.1 効果に関する議論

今回の適用実験の結果，次のような事項を確認した.

**(1) ソフトウェア動作の形式的記述**

従来の自然言語によるソフトウェア動作仕様記述をもとにテスト仕様を作成する場合には，ソフトウェアの動作の流れなどを的確に把握することが難しかった．今回，拡張アクティビティチャートを利用したユースケース記述を導入したことでこれらの課題が解決でき，ソフトウェア動作の流れに沿ったテスト仕様の作成が可能となった.

**(2) ガイドワードを利用した信頼性要求分析**

ユースケース記述に対してソフトウェア信頼性の観点からのガイドワードを適用することにより，ソフトウェアの基本動作に対する異常について抜けなく分析することが可能となった.

**(3) SS-FTA によるソフトウェアフォールト木の作成**

ユースケース記述，ガイドワードなどを利用して段階的にソフトウェアフォールト木を作成するアプローチを採用したことにより，ソフトウェアフォールト木が従来より作りやすく，かつ均一なものが作成できるようになった.

### 6.2 残された課題点

S-TEMRA では過去の不具合例を参考にしてソフトウェア

フォールト木を作成することで，ソフトウェアのテスト項目を作成し，最終的にはこれらに優先度付けをすることを狙っている．今回の検討により，その基本的な部分であるソフトウェアフォールト木を作成し，テストケースの抽出を行うステップについては明確になったと考えている．

次の課題として，ソフトウェアフォールト木の最上位にどのような不具合事象を置くかが重要になる．ソフトウェアの不具合について考えると，不具合の発生頻度およびその不具合が発生した場合に与える影響について考慮することが必要となる．例えば，原子力プラントなどの制御では冷却水給水循環制御機能に不具合が発生した場合，その発生頻度自身は低くても不具合の結果引き起こされる影響は計り知れないものとなる．このような不具合の検出に関するテスト項目の優先度は高くする必要がある．一方でワープロソフトにおける特定文字の変換ミスなどは発生頻度は比較的高いものの不具合の影響という面ではそれほど大きな影響を及ぼすとは考えにくく，テスト項目としての優先度は低くしても差し支えないと考えられる

また，2 節でも示したように，不具合の存在確率についても考慮する必要がある．即ち，より複雑な機能あるいは開発者のスキルが不足している箇所などについては，テスト項目の優先度を高くするといったことを検討する必要がある．

## 7． まとめ

本稿ではユースケース記述を利用し，ソフトウェアフォールトに対するフォールト木を作成し，これを利用してテスト項目の抽出と選択を行うテスト項目設計手法 S-TEMRA について報告した．テスト項目設計手法 S-TEMRA についてはテスト項目設計の全体の流れを提案し，

(1) ソフトウェア動作を拡張アクティビティチャートによって表現し，これに対してガイドワードを利用した信頼性分析を行う手法
(2) ユースケース記述に対するデビエーション分析結果をもとに，ソフトウェアのフォールト木を作成する手法
(3) 上記分析の過程での用語を用語辞書の形で整理していくことで，作成するフォールト木の記述レベルの均一化を実現する手法

などを明らかにした．

また，この手法を実際の冷蔵庫ソフトウェアの一部に適用した実験では，この手法によって作成されたテスト項目と従来のテスト項目を比較し，本方式による方が従来方式に比べより実装に近く詳細なテスト項目を作成できることを確認した．

今後の研究課題としては，テスト項目の優先度付けや選択の観点からどのようなソフトウェアフォールトに着目するか，また，テスト重要度や構造面からソフトウェアの構成部分に，どのように優先度を与えていくかといった点の検討を進めていきたい．

**参考文献**


[1] 萩原他，”マイコン組込み SW の開発を支援する CASE ツール：TestCASE”,情報処理学会第５７回全国大会,pp. l-248-l-249 (1998).

[2] M. Fowler and K. Scott: ”UML DISTILLED: Applying the Standard Object Modeling Language” (羽生田栄一,”UML モデリングのエッセンス”) アジソン・ウェスレイ・パブリッシャーズ (1998).

[3] Z. Xinjun and Y. Tashiro: “An Approach to Automated Program Testing and Debugging”, Proc. of APSEC’99, pp.582-589 (1999).

[4] N.G. Leveson: Safeware: System Safety and Computers, Addison-Wesley (1995).

[5] J.D. Reese et al.: “Software Deviation Analysis”, Proc. of ICSE97, pp.250-260 (1997).

[6] T. Fukaya, M. Hirayama and Y. Mihara: “Software Specification Verification using FTA”, Proc. of FTCS-24, pp.131-133 (1994).

[7] K. Tamura, J. Okayasu and M. Hirayama: “A Software Testing Method based on Hazard Analysis and Planning”, Proc. of ISSRE-98, pp. 103-110 (1998).

[8] T. Tsuchiya et. al.: “Derivation of Safety Requirements for Safety Analysis of Object-Oriented Design Documents”, Proc. of COMPSAC’97, pp.252-255 (1997).

[9] Y. Kim and C.R. Carlson: “Scenario Based Integration Testing for Object-Oriented Software Development”, Proc of 8th ATS, pp.283-288 (1999).

[10] K.-C. Tai and H.K. Su: “Theory of Fault-based Predicate Testing for Computer Programs”, IEEE Trans. on SE, Vol.22, no.8, pp. 552-562 (1996).