Vol.2012-SE-175 No.14
2012/3/15

FMEA を援用した例外処理の仕様定義と設計の方法論

中西 恒夫[†1] 久住 憲嗣[†1] 福田 晃[†1]

例外処理はしばしば不十分に仕様が定義され，ゆえに属人的かつ場当たり的な設計や実装がなされがちである。本論文では，Failure Modes and Effect Analysis，いわゆる FMEA を援用した，例外処理の仕様定義と設計の方法論を提案する。提案方法論は，1) システムがクラスに分割済みであること，2) クラスのメンバ関数の正常処理の仕様が記述済みであること，3) C++，Java 等の言語に見られる `try-catch-finally` 例外処理構文を用いることを前提としている。提案方法論は，正常処理の各実行ステップの仕様記述に対して故障モードを導出し，それら故障モードに対して FMEA を適用し，その結果得られた故障対策に共通性／相違性解析を実施し，例外処理の仕様を定義するとともに，例外処理構文で用いる例外クラスを設計する。

Specifying and Designing Exception Handling with using FMEA

Tsuneo Nakanishi,[†1] Kenji Hisazumi[†1]
and Akira Fukuda[†1]

It often happens in development sites that exceptional operations are insufficiently specified and thus designed and implemented in an ad-hoc, individual dependent manner. This paper proposes a methodology to specify and design exception handling with using failure modes and effects analysis (or FMEA). The methodology assumes that i) the system has been decomposed into components, ii) normal (namely, not exceptional) specifications of components are given, and iii) the exception handling structure in programming languages such as C++ and Java is used. The methodology identifies failure modes of each normal operation step, applies FMEA to them, performs commonality and variability analysis of failure mode countermeasures, describes exceptional specifications, and design exception classes for the exception handling structure.

1. はじめに

高い信頼性，安全性の要求されるシステムでは，定められたサービスを提供できない場合，すなわちシステムに何かしらの故障（failure）が生じる場合においても妥当な振舞いをすることが求められる。故障が生じる前においてはその防止，故障が生じた後には検知，影響の解消・緩和・補償を行い，システムがもたらす損害を最小化しなければならない。

大規模，複雑なソフトウェアシステムにおいて，こうした異常系を妥当かつ正しく実現するには，正常系同様，開発プロセスが重要な役割を担う。異常系についても要求分析がなされ，仕様が定められ，設計，実装，テストされなければならない。正常系と異常系を分けて考えるか否かは別として，故障が生じた際のシステムの振舞いは多かれ少なかれ要求仕様を定義している段階で記述されているはずである。しかしながら，要求工程の段階では，システムの分割は定まっておらず，モジュールレベルでの故障について論じることはできない。工程が下ってシステムの詳細化が進み，開発上の意思決定が重なるにつれ，要求段階では想定していなかった故障の態様，すなわち故障モードが生じてくる。こうした後工程で生じる故障モードを洗い出し，対策を検討したうえで仕様に追加していかなければならないが，システムがモジュールレベルまで分割されていると，体系的な分析がなされることなく，設計者任せとされることが少なくない。結果，異常系の設計にムラやムダが生じ，少なからぬ手戻りが発生したり，システムの拡張・変更の際には異常系の設計に大きな変更を強いられたりしがちである。おおよそ $\frac{2}{3}$ のシステムの故障は，システムの $\frac{2}{3}$ 超を占める異常系に作り込まれた設計上の欠陥に因るものと言われている[1)]。

こうした問題は設計工程のみならず実装工程でも生じる。C++，Java 等のプログラミング言語を用いてシステムを実装する場合，異常系の実現にしばしば例外処理構文（`try-catch-finally` 構文）が用いられる。例外処理構文は，一定の指針に基づいて効果的に用いれば，異常系の可読性や再利用性の向上に大きく寄与する。しかしながら，異常系の設計がおざなりにされ実装者任せとなっていると，場当たり的に例外処理構文が用いられ，処理すべき例外処理の無視，例外の自身の呼出元への無責任な転送，過剰な数の例外クラスの定義といった結末に陥りやすい。

上述の問題を解決すべく，本稿ではシステムの信頼性，安全性の分析手法である Failure

†1 九州大学大学院システム情報科学研究院
Faculty of Information Science and Electrical Engineering, Kyushu University

Modes and Effects Analysis（FMEA）[2] を用いた，モジュールレベルの異常系の仕様定義，ならびに C++，Java 等の `try-catch-finally` 例外処理構文で用いられる例外クラス設計の方法論を論じる。本稿 2 節では FMEA の概要，3 節では `try-catch-finally` 例外処理構文の概要を述べる。4 節では FMEA を用いた例外処理の仕様定義と設計の方法論を提案する。5 節では関連研究を紹介する。最後に 6 節で本稿を総括する。

2. FMEA: Failure Modes and Effects Analysis

本節では，提案手法で用いる Failure Modes and Effects Analysis（FMEA）[2] の概要について述べる。FMEA は一般性の高いの分析手法であり，数十年にわたって，航空，宇宙，自動車，原子力といった分野においてセーフティクリティカルシステムを開発，運用するのに用いられてきた。同様の目的を有する方法論としてはほかに，Fault Tree Analysis（FTA），Hazard and Operability Analysis（HAZOP）等が知られ，これらはしばしば FMEA と併用されている。

FMEA では，システムのさまざまなステークホルダを招集し，システムの構成要素ごとに故障モードを洗い出し，各故障モードが当該要素，当該要素を含むサブシステム，さらにはシステム全体にどのような影響を及ぼすか分析し，その予防や影響の解消・緩和・補償を図る対策を検討する。さらに，システム構成要素の各故障モードについて，その影響のみならず，その発生頻度や影響の重大度を評価し，これらの評価に基づいて影響解消／補償対策に優先順位をつけることも行われる。影響解消／補償対策の優先順位づけを伴う FMEA を，特に Failure Modes, Effects, and Criticality Analysis（FMECA）と呼ぶことがある。優先順位づけにはさまざまな方法があり，それぞれ一長一短である。文献 2) で簡潔な解説がされているので参照されたい。

基本的に FMEA はシステムの「構成要素」の故障モードを分析の起点とするボトムアップ型の分析手法であり，システムが何かしらの下位構造に分割されるまでは，システムに対して FMEA を適用することはできない。しかし，システムの構成要素への分割が完全になった状態で FMEA を適用し，信頼性，安全性を向上すべくシステムの設計を変更することは，大規模，あるいは複雑なシステムでは非現実的である。システムの設計改変によって，せっかく行った詳細設計の結果が無駄になったり，あるいは性能等の非機能要件を満足できなくなり，多大な手戻りコストが生じるためである。そのため，FMEA はその 50 年以上の歴史において，単なる部品単位の分析手法から，システム分割の粒度にあわせて，より上流の工程から段階的に適用できるようなプロセス指向の分析方法論へと進化してきた。具体的には，システムをサブシステムに分割した後にサブシステム単位の FMEA を実施しサブシステムレベルの改変を行い，その後，サブシステムをコンポーネントに分割した後に，サブシステム単位の FMEA の結果を再利用しつつコンポーネント単位の FMEA を実施しコンポーネントレベルの改変を行うようになった。

FMEA の実施には多くの時間を要する。しかし，故障モードがシステムに与える影響は故障モードによってまったく異なるわけではなく，相当の重複が見られる。そのため，FMEA ではシステムに同一の影響を与える（ひとつ以上の）故障モード群を Fault Equivalent Class[3] として扱うことが行われる。Fault Equivalent Class の概念により，重複作業を減らし，FMEA の規模を抑えることが可能となる。ボーイング 777 型機の客室管理システムの事例では，12,401 個の故障モードを 1,759 個の Fault Equivalent Class にまとめ，FMEA の規模を抑えられたことが報告されている[3]。異なる抽象度で実施した FMEA で定義された故障モードであっても，システムに与える影響が同じならば，同じ Fault Equivalent Class に含められる。したがって，Fault Equivalent Class の概念によって，上流工程で実施した FMEA の結果を下流工程で部分的に再利用することも可能となる。

3. 例外処理構文

本節では，提案手法で前提とする，C++，Java 等のプログラミング言語における例外処理構文について述べる。

これらのプログラミング言語における例外処理構文は下記のような文法となっている。

```
try {
  // 正常系の処理
  ...
}
catch (例外クラス 仮インスタンス) {
  // 指定の例外クラスが try 節内で throw されたときの例外処理
  ...
}
finally {
  // catch 節の実行の有無にかかわらず最後に必ず実行される処理
  ...
}
```

try 節には正常系が，catch 節には異常系が実装される。finally 節には，例外処理実施の有無に関係なく実行しなければならない，リソース解放等の後始末の処理が実装される。各 try 節には異なる例外に対処するひとつ以上の catch 節が続く。finally 節は省略可能である。try，catch，finally の各節は他の try-catch-finally 節をネストすることができる。

try 節，あるいはその呼出先の関数において何かしらの理由により処理の継続が不可能になった場合，以下に示す throw 文が実行される。

```
throw 例外クラスの実インスタンス;
```

throw 文の実行により try 節の実行が中断される。実インスタンス（actualExInstance）と互換の例外クラス（ExClass）を仮インスタンス formalExInstance とする catch 節が，この throw 文による例外を処理する。実行中の try 節に続く catch 節の中から該当する catch 節が探索される。見つからない場合は，当該 try 節をネストする，直近上位の try 節に続く catch 節の中から該当する catch 節が探索される。このネストされた try 節を逆にたどる探索は該当する catch 節が見つかるまで再帰的に繰り返される。さらに，該当する catch 節を実行中の関数内で見つけられなかった場合は，当該関数を呼び出した callee 中の，当該関数のアクティブな呼出点を含む try 節に続く catch 節の中から，該当する catch 節が探索される。この関数呼出を逆にたどる探索は，関数内の最外側の try 節に続く catch 節の中から該当する catch 節が見つけられなかった場合に再帰的に繰り返される。適切な catch 節が発見，実行された後は，実行された catch 節を有する例外処理構文のすぐ後に制御が移され，中断された try 節や関数用のスタックフレームが解放される。例外処理構文から制御が離れるときは，その例外処理構文の finally 節が，catch 節の実行とは関係なく常に実行される。

throw 文で指定した例外クラスの実インスタンスは，catch 節で指定した例外クラスの仮インスタンスとして参照できる。そのため，例外クラスは生じた例外の識別のみならず，例外処理の実行に必要なデータを引き渡すのにも利用できる。

4. FMEA を用いた例外処理の仕様定義と設計

本節では，FMEA を用いた例外処理の仕様定義と設計の方法論を提案する。システムのクラスへの分割，クラスの正常処理の設計はすでに完了されているものとする。

以下，グラフィカルユーザ端末のログイン画面フォームを例に，提案方法論を工程ごとに順を追って解説する。端末を使用するユーザは，図 1 に示すようなログイン画面フォームのテキストフィールドにユーザ名とパスワードを入力し，ログインボタンをクリックする。このときクラス LoginForm のメンバ関数 Login() が呼び出される。Login() は入力されたユーザ名とパスワードが正しいか，ユーザデータベースに問い合わせるべく，クラス UserDB のメンバ関数 Auth(username, passwd) を呼び出す。入力されたユーザ名とパスワードが正しければ，Login() はログイン画面フォームを閉じ，呼出元にログインが成功したことを通知する。正しくなければ，Login() はログイン画面フォームを閉じずにユーザがユーザ名とパスワードを再度入力するのを待つか，あるいはログイン画面フォームを閉じて呼出元にログインが失敗したことを通知する。

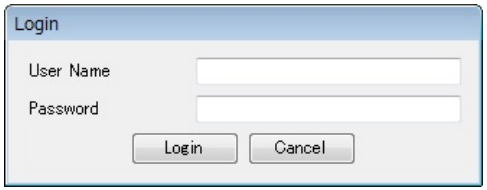

図 1 ログイン画面フォーム

4.1 正常処理仕様の記述

最初に，クラス設計で確定したクラスの各メンバ関数について正常処理仕様を記述する。正常処理仕様記述は，当該メンバ関数がクラス仕様書で定められるサービスを要求されてから成功裏に終えるまでの，メンバ関数内部の処理を時系列で分割記述したものである。各ステップの処理粒度は揃え，ひとつのステップに複数の処理を含めてはならない。また，各ステップは「〜する」と動詞終止形で記述する。条件処理や繰返し処理を含めても構わない。正常処理仕様記述は，メンバ関数の正常処理のみを擬似コードで記述したものと考えてもよい。

表 2，表 3 はそれぞれ UserDB.Auth(username, passwd)，LoginForm.Login() の FMEA 表である。正常処理ステップ欄にはこれらの関数の正常処理仕様が記述されている。

4.2 故障モードの導出

第二に，メンバ関数の正常処理仕様に「〜する」の動詞形で記述される各ステップに対して，文献 4) に提案する故障モード導出手法を応用し，当該ステップを成功裏に完遂できない状況を洗い出す。同故障モード導出手法では，モノ，モノの間の関係，振舞いの諸属性値の異常を故障と捉え，これらの属性値に *no*，*less*，*more*，*other than* の 4 つの HAZOP ガ

イドワードを適用して故障モードを導出している。振舞いについては，始期，終期，期間，回数／頻度，間隔，順序の一般的属性とさらに各振舞いに固有の属性について，それらの値の異常を考える。

`UserDB.Auth(username, passwd)` の正常処理ステップ 2「ユーザ名で指定されるユーザのパスワードをユーザ DB から読み出す。」の場合，振舞いとしては「読み出す」，モノとしては「ユーザ名」,「パスワード」,「ユーザ DB」がある。表 1 では，振舞い「読み出す」に関してその一般的属性と固有属性「成否」「レコード数」の各属性値に，モノ「ユーザ」の属性値「人数」，モノ「パスワード」の属性値「個数」「文字数」の各属性値に対して HAZOP ガイドワードを適用している。これらはすべて故障モードの候補である。これらの中から実際に害を及ぼし得るものを故障モードとして採用し，表現を適切に調整し，FMEA 表の故障モード／原因欄に列挙する。

導出された故障モードが抽象的過ぎる場合は，故障モードに対して FTA 等を実施してその原因を調べ，故障原因にあわせた対策が分析できるようにする。表 2 では，`UserDB.Auth(username, passwd)` の正常処理ステップ 2 において「ユーザ DB の読出しに失敗する。」の原因として,「ユーザ DB がロックされている。」と「その他の理由」が分析されている。

他の関数を呼び出している場合，その呼出点における故障モードには，callee の FMEA 表の呼出元への通知欄（後述）を転写する。たとえば，表 3 の `LoginForm.Login()` の正常処理ステップ 3 の故障モードは，表 2 の `UserDB.Auth(username, passwd)` の呼出元への通知欄を転写したものとなっている。

4.3 各故障モードに対する分析の実施

第三に，各故障モードについて，当該故障モードをどこでどのように検知するのか，当該故障モードがどのような影響を与えるのか，その影響に対する対策，当該メンバ関数の呼出元に対する通知事項を分析し，FMEA 表に記述する。提案方法論の FMEA 表には，正常処理，故障モード／原因欄に加え，検知，影響，対策，呼出元への通知欄が設けられる。

検知：当該故障モードをメンバ関数内で検知できる場合はその方法を簡潔に記述する。

また，当該故障モードがメンバ関数から呼び出した他の関数で検知される場合は，どのような手段でその関数から当該メンバ関数に通知されるのかを記述する。これについては後で述べる呼出元への通知欄を参照されたい。

影響：当該故障モードがシステムに与える影響を記述する。ここに記述する影響は，2 節で述べた Fault Equivalent Class ごとにグルーピングし，通し番号である Fault Identifier Number（FIN）を与えていく。すなわち，上流工程や現工程で作成した FMEA 表に既述の影響については，同じことを二度書くかわりに FIN で参照する。表 3，表 2 の影響欄では，初出の影響にはローマ数字で書かれた FIN と影響を，既出の影響には FIN のみを書いている。

対策：当該故障モードへの当該メンバ関数における対策を記述する。対策は故障が生じる前の予防策と生じた後の解消・緩和・補償策に大別され，後者の一部が例外処理として実行される。さらに対策は製品リリース前に採られる開発時対策とリリース後に採られる実行時対策とにも大別される。

呼出元への通知：当該故障モードを検知した際の当該メンバ関数の呼出元への通知手段と通知事項を記述する。故障モードの通知は必須ではない。情報隠蔽を徹底し，クラス間の結合を疎にするためにも，故障モードの通知を必要以上に行うべきではない。故障モードの通知が必要となるのは以下の場合である。

- このメンバ関数の責務の範囲内では故障モードの処理ができない場合。
- このメンバ関数の呼出元において故障モードの影響を解消，緩和，補償できる場合。
- このメンバ関数の呼出元によって故障モードに対する対策が変わり得る場合。

故障モードの発生をメンバ関数の呼出元に通知するには，関数の戻り値や参照パラメータ，例外クラス，グローバル変数，規定あるいは事前登録された関数の呼出し等の手段が考えられる。通知には，例外処理中で使用されるデータや例外処理の振舞を変える制御情報が含められる。この欄の記述は caller の FMEA 表の検知欄に再度現れ得ることに注意されたい。

その他：その他，必要に応じて，個々の故障モードに対して対策の優先順位をつけるべく，故障モードの致命度，すなわち故障モードの発生頻度や影響の重大度を査定する。

4.4 故障モード対策の共通性／相違性分析

第四に，故障モード欄に記述された実行時対策に対し，共通性／相違性分析を実施し，実行時対策相違性モデルを記述する。実行時対策相違性モデルは実行時対策間の共通性／相違性をフィーチャ単位で記述したものである。

ソフトウェアプロダクトライン[6] では，フィーチャを用いてプロダクトラインの製品間の共通性／相違性を記述する，フィーチャモデル[5] が広く用いられている。しかし，フィーチャモデルはソフトウェア構成時のフィーチャの選択性を記述する静的なモデルであるのに対し，実行時対策相違性モデルはソフトウェア実行時のフィーチャの選択性を記述する動的なモデルである。実行時対策相違性モデルは，ソフトウェア実行時のフィーチャの選択性を記述できるよう，フィーチャモデルを拡張したものである。実行時対策相違性モデ

表 1 「読み出す」（上段）／「ユーザ」（中断）／「パスワード」（下段）へのガイドワード適用

属性	型	適用なし	適用あり			
		ϕ	*no*	*less*	*more*	*other than*
始期	時刻	適時読出しを始める	読出しが始まらない	早く読出しを始める	遅く読出しを始める	／
終期	時刻	適時読出しを終える	読出しが終わらない	早く読出しを終える	遅く読出しを終える	／
…	…	…	…	…	…	…
成否	真偽値	成功裏に読み出す	読出しに失敗する	×	×	×
レコード数	個数	適数読み出す	全く読み出さない	少なく読み出す	多く読み出す	／
人数	個数	適当な数	なし	少ない	多い	／
個数	個数	適当な数	なし	少ない	多い	／
長さ	文字数	適当な長さ	空文字列	短い	長い	／
…	…	…	…	…	…	…

ルにおけるソフトウェア構成時のフィーチャ選択性の表現は，フィーチャモデルとおおよそ同じく，装飾なしの節点は必須（mandatory）フィーチャ，白丸つきの節点はオプショナル（optional）フィーチャ，親フィーチャとの枝が実線の円弧で括られる一群の節点は択一的（alternative）フィーチャを，実線の細い枝は全体・部分の関係，実線の太い枝は実装関係を表す。但し，汎化・特化関係はオリジナルのフィーチャモデルのように点線の枝ではなく，子フィーチャから親フィーチャに向かう白抜き矢印つきの枝で表すものとする。さらに，備えられていれば必ず実行されるフィーチャは親フィーチャとの枝を実線で，そうでないフィーチャは親フィーチャとの枝を点線にするものとする。また，実行時に択一的に選択される一群のフィーチャは親フィーチャとの枝を点線の円弧で括るものとする。

図 2 に表 3 に基づいてモデリングした実行時対策相違性モデルを示す。この例では，ソフトウェア構成時の相違性はなく，ソフトウェア実行時の相違性のみが現れている。この例外処理では，エラーメッセージ表示は必ず行われるが，具体的に表示されるエラーメッセージは「ユーザ名未入力」,「ユーザ DB のロック」等から択一されることがわかる。ユーザフィールドのクリアは必ず実行されるとは限らないが，実行されれば User Name フィールドは必ずクリアされる一方，Password フィールドはクリアされたりされなかったりすることがわかる。

4.5 例外処理仕様の記述

第五に，メンバ関数の例外処理仕様を記述すると同時に，必要に応じて正常処理仕様の追加や変更を行う。

実行時対策相違性モデルのフィーチャを，正常処理として `try` 節内で処理される正常フィーチャと，例外処理として `catch` 節内で処理される例外フィーチャに分ける。実行の際に一連の正常処理を中断しなければならないようなフィーチャが例外処理フィーチャとなる。その他のフィーチャについては正常フィーチャにも例外フィーチャにもなり得る。フィーチャをクラス分けした後，例外フィーチャに関する例外処理仕様を正常処理仕様と同じ書式で記述する。さらに，正常フィーチャに分類された実行時対策相違性モデル上のフィーチャについては，正常処理仕様の記述を追加または変更する。

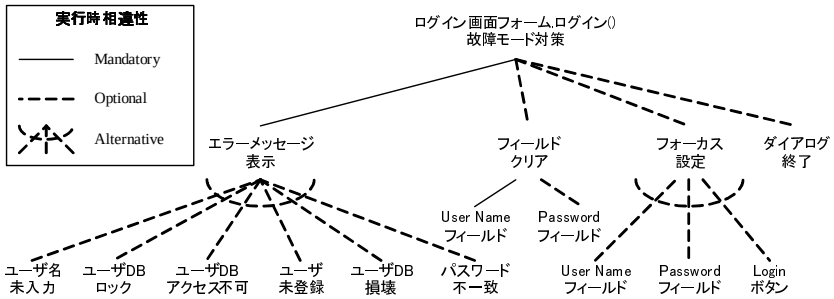

図 2 実行時対策相違性モデル

4.6 例外クラスの設計

最後に，実行時対策相違性モデルを参照して例外クラスを設計する。クラスの各メンバ関数について構築された相違性モデルには，故障モード対策となる実行時の振舞い，すなわち例外処理と，例外処理で使用されるデータを表現するフィーチャが含まれている。また，

相違性モデルは，例外処理においてどのフィーチャが条件的に実行されるのかを示す，制御情報も表していることに注意されたい。

互いに強く関連し，多くの故障モードを共有しているようなメンバ関数については，それらの実行時対策相違性を例外クラス設計前に統合し，例外クラスの総数を抑える。統合後の実行時対策相違性モデルごとに例外クラスを定義する。例外処理の実行時の振舞いを切り替える制御情報を意味するフィーチャは，例外クラスのフラグ属性に帰着される。例外処理で使用されるデータを表すフィーチャは，例外クラスの属性に帰着される。これらの属性の参照や変換は例外クラスのメンバ関数として定義される。

図 2 の実行時対策相違性モデルに従って定義された例外クラスを以下に示す。

```
class ExLoginForm : public Exception
{
public:
  enum LoginFormError {
    ERR_EMPTY_USER_NAME,
    ERR_USERDB_LOCKED,
    ...
  } login_form_error;
  enum ClearingFields {
    UserNameField = 0x01,
    PasswordField = 0x02,
  } clearing_fields;
  enum FocusingUIObject {
    UserNameField,
    PasswordField,
    LoginButton
  } focusing_UI_object;
  bool terminating;
  string getErrorMessage();
  ...
};
```

5. 関 連 研 究

FMEA あるいは HAZOP をソフトウェアにおいて適用する事例は，過去 10 数年の間に相当数報告されている[4),7)–17)]。

HAZOP は想定される故障を網羅するうえで有効な方法論である。本研究で用いている著者文献[4)] の故障モード導出手法は，振舞いとその対象たるモノの二観点で HAZOP ガイドワードを適用するが，これは河野の先行研究[16)] の延長上にあるアプローチとなっている。ソフトウェアでの HAZOP 適用事例は，UML を対象とする Hansen らの研究[12)] や状態遷移図を対象とする金らの研究[17)] があるが，いずれも本研究の対象とする内部設計レベルよりは抽象度の高い抽象度の記述を対象とするものである。

本稿では，故障モードの抽象度が高い場合は FTA 等を実施して故障原因を探り，故障原因にあわせた対策が分析できるようにした。故障モードから故障原因を探るアプローチは，Lutz ら[7)] や Goddard[8)] の研究，ならびに著者研究[4)] によっても行われている。夏目らは故障原因も含めて故障モードと見なし，異なる抽象度での故障モードの観点リストを与えている[15)]。著者らは故障モードと故障原因を分ける立場だが，観点リストそのものは本稿提案手法でも活用できるものである。一方，Maxion らは，例外要因を演算，ハードウェア，入出力，ライブラリ，データ，戻り値，外部環境，Null ポインタ／メモリの問題に分類する CHILDREN 観点に基づく魚骨図の観点リストを示している[18)]。

6. ま と め

本稿では，FMEA を援用した例外処理の仕様定義と設計の方法論を提案した。

提案方法論は，各メンバ関数の正常処理仕様記述の各処理ステップについて，著者研究による HAZOP ベースの故障モード導出方法論[4)] で故障モードを導出，FMEA を実施する。FMEA で検討された対策に対して，共通性／相違性分析を実施し，実行時対策相違性モデルを記述する。実行時対策相違性モデルは，ソフトウェアプロダクトラインで用いられるフィーチャモデルを，製品間の相違性のみならず，製品の実行時の振舞いの相違性をも表現するように拡張したものである。同モデルを用いて，例外処理の実行時の振舞いの違い，例外処理で必要とされるデータとその違い，さらにはそれらの製品間での違いを整理し，C++ や Java 等の `try-catch-finally` 例外処理構文で使用される例外クラスを設計する。

提案方法論は新規開発のみならず，既存システムの例外処理関連コードのリファクタリングにも応用できるものと思われる。今後の課題として提案方法論の実アプリケーションへの大規模適用が挙げられる。

参 考 文 献

1) F. Cristian, “Exception Handling and Tolerance of Software Faults,” *Software Fault Tolerance*, M.R. Lyu, ed., Chapter 4, John Wiley & Sons, 1995.

表 2 メンバ関数 UserDB.Auth(username, passwd) 内部の FMEA 分析

番号	正常処理ステップ	故障モード／原因	検知	影響	対策	caller への通知
1	ユーザ DB に接続できることを確認する。	ユーザ DB に接続できていない。	ユーザ DB アクセス用のハンドルが NULL。	I: ユーザの認証ができない。	例外発生。	ユーザ DB に接続できていないこと
2	ユーザ名で指定されるユーザのパスワードをユーザ DB から読み出す。	ユーザ DB の読出しに失敗する。／ユーザ DB がロックされている。	DB ライブラリの戻り値とエラーコード	I	例外発生。	ユーザ DB がロックされていること
		ユーザ DB の読出しに失敗する。／その他の理由	DB ライブラリの戻り値とエラーコード	I	例外発生。	ユーザ DB を何かしらの理由で読み出せないこと
		該当ユーザが見つからない。	マッチレコード数が 0。	I	例外発生。	該当ユーザが見つからないこと＋該当ユーザ名
		該当ユーザが 2 人以上見つかる。	マッチレコード数が 2 以上。	II: ユーザ DB が論理的に壊れている。ユーザ DB に継続してアクセスすると，ユーザ DB の部分的な回復ができないぐらい壊れるかも知れない。	例外発生。	ユーザが重複登録されていること＋該当ユーザ名
3	パスワードが一致しているか確認する。	パスワードが一致していない。	パスワード不一致。	I	ユーザを認証せず，関数を終了する。	戻り値：false
4	認証を完了する。			（ユーザを認証が成功裏に終了する。）	ユーザを認証し，関数を終了する。	戻り値：true

2) J.B. Bowles, “The New SAE FMECA Standard,” *Proc. Annual Reliability and Maintainability Symp. (RAMS) 1998*, pp.48–53, Jan. 1998.
3) C.S. Spangler, “Equivalence Relations within the Failure Mode and Effect Analysis,” *Proc. Annual Reliability and Maintainability Symp. (RAMS) 1999*, pp.352–357, Jan. 1999.
4) 中西 恒夫, 久住 憲嗣, 福田 晃, 「ソフトウェア FMEA の一手法とプロダクトライン開発におけるその利用」, 信学技報, 2012 年 3 月（発表予定）.
5) K.C. Kang, J. Lee, and P. Donohoe, “Feature-Oriented Product Line Engineering,” *IEEE Software*, Vol.9, No.4, pp.58–65, July/August 2002.
6) P. Clements and L. Northrop, *Software Product Lines: Practice and Patterns, Addison-Wesley*, 2001.
7) R.R. Lutz and R.M. Woodhouse, “Experience Report: Contributions of SFMEA to Requirements Analysis,” *Proc. 2nd Int. Conf. on Requirements Engineering (ICRE ’96)*, pp.44–51, Apr. 1996.
8) P.L. Goddard, “Software FMEA Techniques,” *Proc. Annual Reliability and Maintainability Symp. (RAMS) 2000*, pp.118–123, Jan. 2000.
9) J.B. Bowles and C. Wan, “Software Failure Modes and Effects Analysis for a Small Embedded Control System,” *Proc. Annual Reliability and Maintainability Symp. (RAMS) 2001*, pp.1–6, Jan. 2001.
10) D. Nguyen, “Failure Modes and Effects Analysis for Software Reliability,” *Proc. Annual Reliability and Maintainability Symp. (RAMS) 2001*, pp.219–222, Jan. 2001.
11) N. Ozarin, “Failure Modes and Effects Analysis during Design of Computer Software,” *Proc. Annual Reliability and Maintainability Symp. (RAMS) 2004*, pp.201–206, Jan. 2004.
12) K.M. Hansen, L. Wells, and T. Maier, “HAZOP Analysis of UML-Based Software Architecture Descriptions of Safety-Critical Systems,” *Proc. 2nd Nordic Workshop on the Unified Modeling Language (NWUML) 2004*, pp.59–78, Aug. 2004.
13) 山科 隆伸, 森崎 修司, 飯田 元, 松本 健一, 「保守開発型ソフトウェアを対象としたソフトウェア FMEA の実証的評価」, ソフトウェア品質シンポジウム 2008 発表報文集, pp.157–164, 2008 年 9 月.
14) A. A. Shenvi, “Software FMEA: A Learning Experience,” *Proc. India Software Engineering Conf. (ISEC) 2011*, pp.111–114, Feb. 2011.
15) 夏目 珠規子, 村山 知寛, 小島 昌一, 「ソフトウェア開発における FMEA の適用可能性検討」, 第 41 回信頼性・保全性シンポジウム予稿集, pp.359–364, 2011 年 7 月.
16) 河野 哲也, 「ソフトウェア要求仕様における HAZOP を応用したリスク項目設計法」,

表 3 LoginForm.Login() 内部の FMEA 分析

番号	正常処理ステップ	故障モード／原因	検知	影響	対策	caller への通知
1	User Name フィールドのユーザ名を取得する。	ユーザ名が入力されていない。	ユーザ名の長さが 0。	I	**実行時対策**：i) ユーザ名を入力するようにメッセージを表示する；ii) User Name，Password 両フィールドをクリアする；iii) User Name フィールドにフォーカスをあてる；iv) ダイアログを続行する。	なし
		入力されたユーザ名に不正な文字が使用されている。	ユーザ名に規定外の文字が使用されている。	III: ユーザ DB に問い合わせた時にログイン不可と判断される。	（ここでの対策は不要。）	なし
	ユーザ名の長さが規定超。	入力されたユーザ名が長すぎる。	自身	III	**開発時対策**：User Name フィールドに入力できる最大文字列長をユーザ名の最大長に設定する。	なし
2	Password フィールドに入力されているパスワードを取得する。	パスワードが入力されていない。	パスワードの長さが 0。	O: 問題なし。（パスワード設定していないユーザが考えられるため。）	（対策は不要。）	なし
		入力されたパスワードに不正な文字が使用されている。	パスワードに規定外の文字が使用されている。	III	（ここでの対策は不要。）	なし
	パスワードの長さが規定超。	入力されたパスワードが長すぎる。	自身	III	**開発時対策**：Password フィールドに入力できる最大文字列長をパスワードの最大長に設定する。	なし
3	ユーザ DB にログイン可能か問い合わせる。	ユーザ DB に接続できていない。	ユーザ DB. 認証 () からの例外	I	**実行時対策**：i) ユーザ DB にアクセス不可能である旨メッセージを表示する；ii) ダイアログを終了する。	なし
		ユーザ DB がロックされている。	ユーザ DB. 認証 () からの例外	I	**実行時対策**：i) ユーザ DB がロックされている旨メッセージを表示する；ii) Login ボタンにフォーカスをあてる；iii) ダイアログを続行する。（しばらく後にロックが解除されるかもしれないため。）	なし
		ユーザ DB を何かしらの理由で読み出せない。	ユーザ DB. 認証 () からの例外	I	**実行時対策**：i) ユーザ DB にアクセス不可能である旨メッセージを表示する；ii) ダイアログを終了する。	戻り値：false
		該当ユーザが見つからない。	ユーザ DB. 認証 () からの例外	I	**実行時対策**：i) ユーザが登録されていない旨メッセージを表示する；ii) User Name，Password 両フィールドをクリアする；iii) User Name フィールドにフォーカスをあてる；iv) ダイアログを続行する。	なし
		ユーザが重複登録されている。	ユーザ DB. 認証 () からの例外	II	**実行時対策**：i) ユーザ DB が論理的に壊れている旨メッセージを表示する；ii) ダイアログを終了する。	戻り値：false
		パスワードが一致していない。	ユーザ DB. ユーザ認証 () からの戻り値	I	**実行時対策**：i) パスワードが一致していない旨メッセージを表示する；ii) Password フィールドをクリアする；iii) Password フィールドにフォーカスをあてる；iv) ダイアログを続行する。	なし
4	フォームを終了する。			（ユーザのログインが成功裏に終了する。）		戻り値：true

ソフトウェアテストシンポジウム 2012（JaSST '12 Tokyo）予稿集, pp.37–42, 2012 年 1 月.
17) 金 周慧, 松原 豊, 高田 広章,「状態遷移図に着目した安全分析手法」, 信学論, Vol.J95-A, No.2, pp.198–209, 2012 年 2 月.
18) R.A. Maxion and R.T. Olszewski, "Eliminating Exception Handling Errors with Dependability Cases: A Comparative, Empirical Study," *IEEE Trans. on Software Engineering*, Vol.26, No.9, Sep. 2000.