ソフトウェアテストシンポジウム 2003
JaSST '03: Japan Symposium on Software Testing 2003

不具合の分析とフィードバックによるテストの設計

西 康晴†

†株式会社エス・キュー・シー コンサルティング部 〒215-0012 神奈川県川崎市麻生区上麻生 1-7-14
E-mail: †nsh@mtd.biglobe.ne.jp

あらまし テストの工数を発散させずに不具合を検出するには，機能やデータなどの組み合わせによるバグを網羅せずに検出する必要がある．本研究では，バグが偏在することに着目し，過去に見逃したバグの知識をクラス階層として抽象化し構造化することで，テスト設計をフィードバックし改善する手法を提案する．また CAD ソフトウェアに対する仕様レビューに本手法を適用して効果を確認する．

キーワード ソフトウェアテスト，フィードバック，組み合わせ，ナレッジマネジメント

Test design strategy by analysis and feedback of bugs

Yasuharu NISHI †

†Consulting Division, SQC Inc. 1-7-14 Kami-asao, Asao-ku, Kawasaki-shi, Kanagawa, 215-0012 Japan
E-mail: †nsh@mtd.biglobe.ne.jp

Abstract Divergence of effort is an essential problem of software testing. It is necessary to avoid combinations of functions and/or data. This paper proposes a test strategy by self-improvement focusing on clustering of bugs. It is a feedback strategy with abstraction and structuring knowledge of bugs once missed as class hierarchy. We also present an example whether to detect combinatorial bugs in specification review of CAD software.

Keyword Software testing, Feedback, Combination, Knowledge Management

1. まえがき

1.1. Myers の呪縛

Myers[1]は，多くのテスト技法を提案したと同時に，ブラックボックステストに関して「一般に，プログラムのすべてのエラーをみつけることは，非現実的でもあり，しばしば不可能でもある.」という原則を提示した．偉大なる Myers の原則を読んで，勉強熱心だが気の早いソフトウェア技術者は「ならばテストの質を向上してもムダなのではないか」という呪縛でがんじがらめにされてしまった．テストの質を向上することに取り組まず，設計や分析の技術だけに注力したり，プロセス改善に取り組んだのである．これらはテストの質の向上が伴わないため，結果として不具合の低減に結びつかない取り組みが発生してしまい，ソフトウェア工学的活動全般に対する猜疑心を生むこととなった．これを“Myers の呪縛”と呼ぶこととする．

始末の悪いことに，Myers の呪縛によってムダと感じてしまうためテストに力を入れなくなり，それによってテストの精度が下がり，ますますムダに感じられるという悪循環も生まれてしまった．もちろん Myers の呪縛に囚われずテストの質を向上させた組織もあるが，Myers の呪縛によりテストの質を向上させようという気概が失われてしまった組織も少なくない．

Myers の呪縛は本質的に，ブラックボックステストですべての不具合を検出するためには，テスト項目数が発散してしまうことを指している．したがって我々は Myers の呪縛から解き放たれるために，なぜテスト項目数が発散するのかを明らかにし，テスト項目数の発散を抑えながら不具合をより多く検出する方法を研究する必要がある．

1.2. 組み合わせによるテスト工数の発散

ブラックボックステストでテスト項目数が発散する原因は，大きく 2 つに分けられる．

1 つは，数値範囲のテスト項目は無限に存在するため発散してしまうという問題である．1 から 10 の範囲に存在する浮動小数点の数を考えてみると，直感的にすぐ理解できるだろう．本質的には実数が無限に存在する，という公理と同じである．もちろんデジタルなので無限ではないが，近似的に無限と考えてよい．数値範囲に文字や制御コードなど無効な入力を含めると，テスト項目数はさらに発散してしまう．

この問題に対する解は，境界値分析である．境界値分析を用いると，連続な数値範囲のテスト項目は範囲の両端の境界の内外である 4 項目となるため，テスト項目は発散しない．また，連続であることを保証するには，テスト以前の工程でデザインレビューやコードインスペクションを十分行えばよい．

もう 1 つが，テストの組み合わせである．純粋なブラックボックステストでテスト設計を行おうとすると，機能やパラメータの組み合わせが必要となり，テスト項目数は発散してしまう．組み合わせの発散を抑えるには，直交配列表を始めとするラテン方格を用いる方法[2]および，数学的にはほぼ等価な All-pair 法[3]を用いることになる．とはいえ，これらの方法は多次元の組み合わせの網羅（直交配列表では 3 次元以上）を保証するものではない．また，組み合わせの次元が増大すると，用いるラテン方格が大きくなり，テスト設計が困難となるため現実的では無くなってしまう．

1.3. 本論文の構成

そこで本研究では，ブラックボックステストにおいてテスト項目数の発散を抑えながら不具合をより多く検出するための，不具合分析およびテスト設計の手法を提案する.本手法では不具合のメカニズムに着目し，オブジェクト指向におけるクラス階層に類似した抽象化構造を用いることにより，組み合わせバグに関する知識のフィードバックを行い，テスト設計を改善していく．

2 章では，テスト項目数の発散に関する考察について述べ, 発散を抑制するための予想を示す. 3 章では，組み合わせの発散を抑制するようなテスト設計手順と，バグ見逃しのフィードバックのための知識構造について示す．4 章では，仕様レビューに対する適用例を示す．5 章では，まとめを行う．

2. 組み合わせによるテスト項目数の発散に関する考察

2.1. 組み合わせによるテスト項目数の急激な発散

テスト組織が成熟していくと，テストの際に機能やパラメータの組み合わせを考慮するようになり，テスト項目数が図 1 のように指数関数的に発散していくと筆者は考える．テスト対象に含まれているバグを全て検出するためには，実質的に無限のテスト項目数が必要となる．これが Myers の呪縛である．

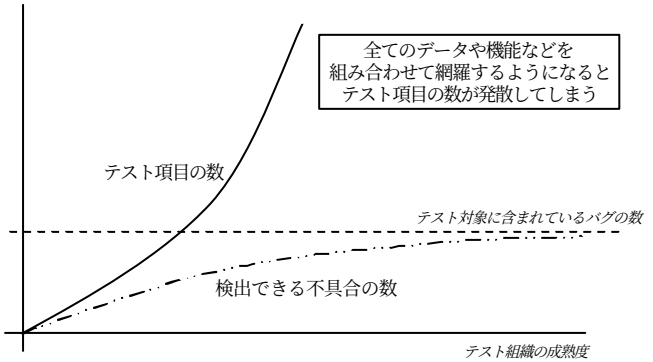


図 1　組み合わせによるテスト項目数の急激な発散

2.2. 組み合わせの回避によるテスト項目数の穏やかな発散

品質の高い組織では，組み合わせによるテスト項目数の発散を回避するために，デザインレビューやコードインスペクションといった上流でのレビューを行うことで，機能やパラメータの依存性を最小限に留めようとする．依存性を極力排除することで，組み合わせなければ見つけられないバグを作り込む可能性を低めるのである．

組み合わせを考慮せずにテストを行うと，図 2 のようにテスト組織が成熟していくにしたがってテスト項目数も収束していくと筆者は考える．とはいえ組み合わせバグの可能性はゼロにならないため，全てのバグを検出することはできない．

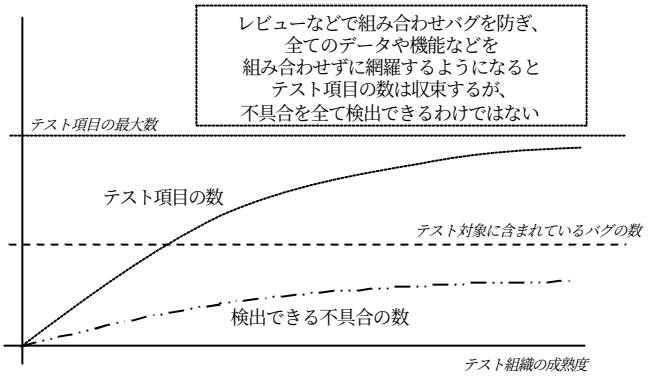


図 2　組み合わせの回避によるテスト項目数の収束

そのため現実的には，上流でのレビューを実施しても一部の組み合わせテストを行う必要がある．したがって図 3 のようにテスト項目数はその分増加し，テスト項目数は穏やかに発散してしまうと筆者は考える．

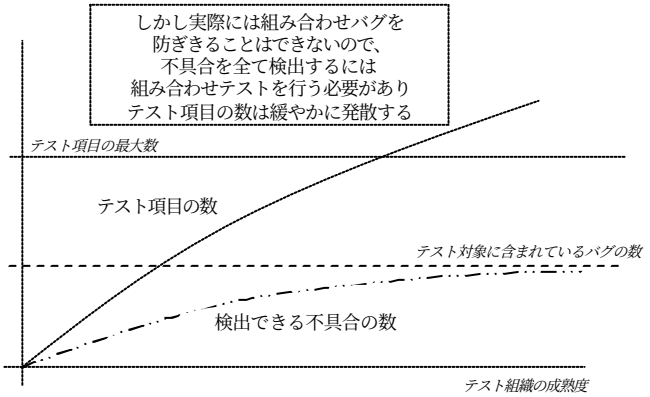


図 3　組み合わせの回避による穏やかな発散

2.3. 不具合分析とフィードバックによるテスト項目数の発散の抑制

一般に，バグには傾向があることが知られている．Davis[4]は「バグは偏在する」という経験則を提示している．Kaner[5]は実際的なテストの戦略について「バグの出方を深く理解したうえで，バグが出そうなテストを行うべきである」と述べている．すなわちバグの多そうな部分をあらかじめ理解しておけば，効率的にテストを行うことができる．

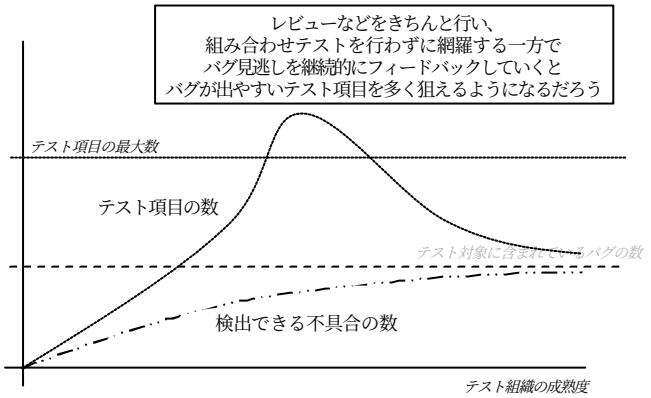


図 4　バグ見逃しのフィードバックによる発散の抑制

この経験則を組み合わせバグに適用すると，興味深い予想が成り立つ．組み合わせバグには傾向があり，その傾向をあらかじめ理解してテスト設計にフィードバックすると，効率的にテストを行うことが可能となり，図 4 のようにテスト項目数の発散を抑制できるという予想である．テスト組織が成熟していくと，不具合を検出できる組み合わせテスト項目をより多く設計することができるので，不必要な組み合わせテストの項目を省いていくことが可能になると考えられる．

3. フィードバックテストと不具合分析

3.1. フィードバックによるテスト設計

2.3 で示した予想に基づいて組み合わせバグの傾向を把握するためには，バグの見逃しを収集し分析した上で，テスト設計にフィードバックする必要がある．こうした組み合わせの発散を抑制するようなフィードバックによるテスト設計は，図 5 のような 4 つの手順から構成されるサイクルで実施する

まず，単機能網羅テストのように，様々な観点での組み合わせを行わずに，機能やデータなどを単純に網羅するようなテストを設計する（Design）．また設計したテスト項目を実施する．

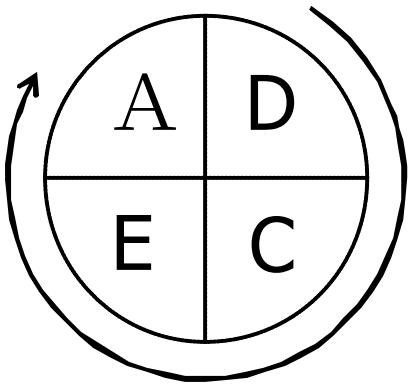
次に，バグの見逃しを収集し分析する（Collect）．第 1 のステップで適切に網羅されているのであれば，バグの見逃しは，何を網羅するかという観点が抜け落ちているか，網羅した項目の組み合わせが抜け落ちているか，のどちらかとなる．前者の抜け落ちは，テスト組織の成熟度が向上すればほとんど考えなくてよい．したがってこのフェーズでは，実質的に組み合わせバグの見逃しの収集と分析を行うことになる．

そして，バグの見逃しを分析し，組み合わせが爆発しないようにテスト設計を行うためのポイントを抽出（Extract）していく．

最後に，抽出したポイントをテスト設計のために追加する（Add）．

そして再度，追加したポイントを加えて改善されたテストを設計する（Design）．

このサイクルをきちんと回すためには，抽出したポイント，すなわちバグ見逃しの原因の知識を適切に抽象化し構造化した上で蓄積し，再利用する必要がある．バグ見逃しのナレッジマネジメントと呼んでもよい．このサイクルによって組み合わせバグの傾向が把握でき，組み合わせの発散を抑制するテスト設計が可能になる．



- テストを設計する（Design）
- バグ見逃しを収集する（Collect）
- バグ見逃しが起きやすいポイントを抽出する（Extract）
- 抽出したポイントをテスト設計に追加する（Add）

図 5　フィードバックテストのサイクル

3.2. バグ見逃しのフィードバックのための知識構造

バグ見逃しに限らず，知識を再利用すべく抽象化や構造化を行うのは容易ではない．一言に抽象化と言ってもさまざまだからだ．そこで本研究では適切な抽象化のガイドとして，オブジェクト指向分析におけるクラス階層の構造を用いることとした.クラス階層には，クラス，インスタンス，継承という抽象化のためのモデルが用意されているため，現場で不具合を再利用する必要のある技術者，すなわち知識工学の専門家ではないテスト技術者でも比較的容易に抽象化を進めることができる．したがって，バグ見逃しの抽象化と構造化は，組み合わせバグの見逃しについてのクラス階層ツリーを構築することと同義となる．フィードバックを行う際には，クラス化された組み合わせバグのパターンを照らし合わせることで，新たな場所で発生する似たような組み合わせバグを検出していく．

一般的に知識ツリーを作成するアプローチには，大きく分けてトップダウンとボトムアップの２通りがある．知識ツリーで表現したい分野のアーキテクチャが分かっている状態で，個々の知識の関係を明らかにしたい場合は，トップダウンが適している．しかしバグ見逃しのように，全体の知識構造のアーキテクチャそのものを試行錯誤しながら知識ツリーを構築する際には，ボトムアップが適している．したがってバグ見逃しのインスタンス，すなわち組み合わせバグの事例からバグのクラスを抽出したり，バグのクラスから子クラスを導出することを繰り返して進めることとなる．

またクラス構造を作成する前には，テストで見逃された組み合わせバグを記述し直す必要がある．通常の不具合の記述は，不具合そのものの再現や修正が目的であり，再利用を前提としていないため，抽象化を行いにくい記述となっているからである．再記述の際には，３つのポイントを明らかにしておく必要がある．

１つは，どのようにしてバグが発生したか，という記述である．バグが発生する状況や条件と言ってもよいだろう．これうは，テストを見逃しやすい兆候を抽出するために必要である．

２つ目と３つ目は，期待結果と実施結果である．バグが発生するメカニズムと言ってもよい．これは，なぜそのバグを作り込んでしまったのかという兆候を抽出するために必要である．バグを作り込んでしまったメカニズムを抽出することで，バグが多い部分を明らかにし,テストを見逃しやすい兆候を抽出することで，テスト見逃しを減らしていく．

3.3. バグ見逃しのクラス階層を構築する手順

組み合わせバグの見逃しのクラス階層を構築する際には，5 つの手順が必要である．

まずバグを収集し，抽象化し，同じメカニズムで発生するバグを具体化する．また全く同じではないが類似のメカニズムを推測し，そのメカニズムで発生するバグを具体化する.

図 6 にバグ見逃しのクラス階層を，図 7 にバグ見逃しのクラス階層を構築する手順を示す.

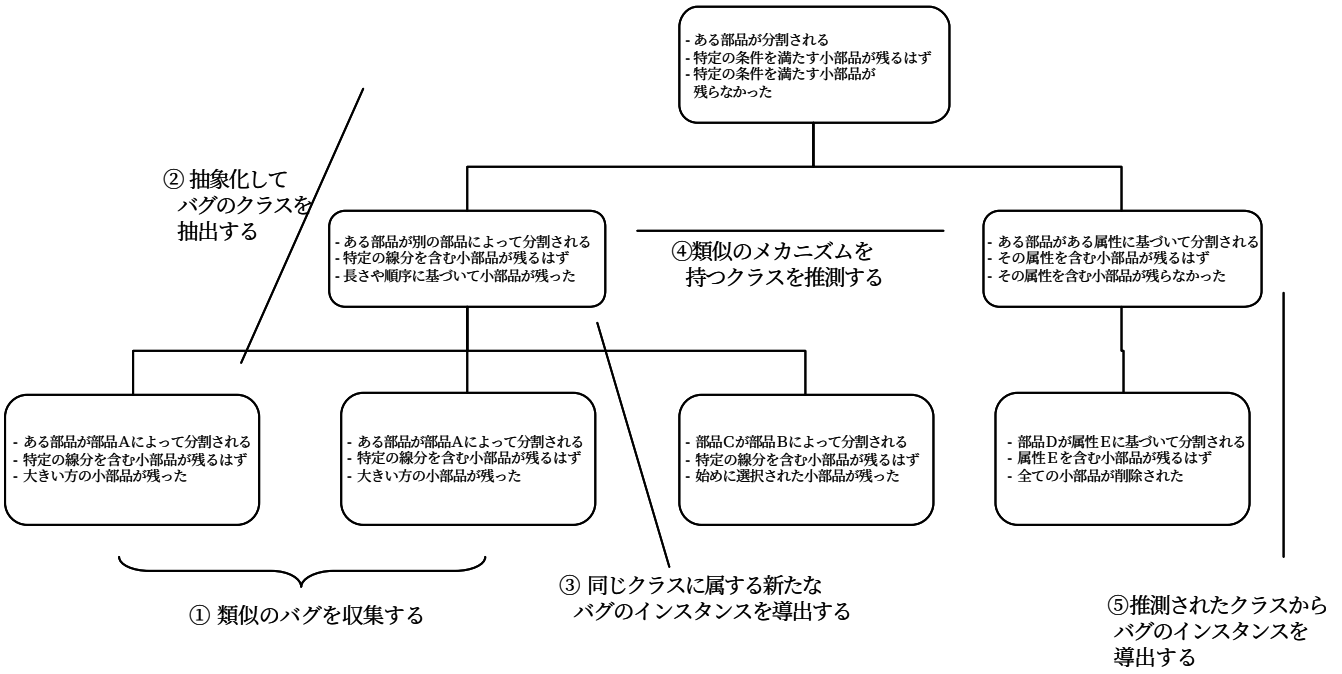


図 6　バグ見逃しのクラス階層

3.4. 仕様レビューやデザインレビューへの適用

① 類似のバグを収集する
- バグの事例をインスタンスとして収集する
- バグの記述を書き直す
- 似たようなバグに分類する

② 抽象化してバグのクラスを抽出する
- 似たようなバグとして分類されたバグを，本質的に共通な部分と異なる部分に区別する
- 共通な部分を残し異なる部分の記述を代名詞に置き換えることで，バグのクラスの記述を作成する

③ 同じクラスに属する新たなバグのインスタンスを導出する
- クラスの記述に当てはまるテスト対象の機能やデータ構造などを探す
- 当てはまったクラスの示すメカニズムや兆候を持つバグを具体化する
- 当てはまった機能やデータ構造などと，具体化したバグから，テスト項目を設計する

※当てはまる機能やデータ構造などが多ければ多いほど，バグを見つけられる可能性は低下する

④ 類似のメカニズムを持つクラスを推測する
- 親クラスを抽出し，その親クラスから新たに可能な子クラスを論理的に推測する

⑤ 推測されたクラスからバグのインスタンスを導出する
- クラスの記述に当てはまるテスト対象の機能やデータ構造などを探す
- 当てはまったクラスの示すメカニズムや兆候を持つバグを具体化する
- 当てはまった機能やデータ構造などと，具体化したバグから，テスト項目を設計する

図 7　バグ見逃しのクラス階層を構築する手順

テストとレビューの違いは，不具合を検出するためにテストないしレビューの対象を動作させるかどうかである．そのため，レビューは静的テストとも呼ばれる．不具合を検出するためにテスト項目やレビューでの指摘事項を設計するという点では同じである．

したがって，3.1 に示したテスト設計サイクルや，3.2 や 3.3 に示したクラス階層の構築手順は，レビューにおける指摘事項の設計に適用しても有益である．

3.5. 過度の抽象化の回避

3.1 に示したテスト設計サイクルや，3.2 や 3.3 に示したクラス階層の構築手順をきちんと実施するためには，抽出したポイント，すなわちバグ見逃しの原因の知識を適切に抽象化する必要がある．

気を付けなくてはならないのは，過度の抽象化を回避することである．過度の抽象化を行うと，テスト見逃しの原因は工数不足であるとか，スキル不足であると分析してしまう.「そもそも論」になって議論が発散してしまうのだ．例えば，そもそも無理なプロジェクトなのだ，そもそもスキルのある協力会社を雇っておけばよかったのだ，そもそも教育やトレーニングが充実してないのが問題なのだ，そもそもソフトウェア産業の構造的問題なのだ,いやそもそも不況が悪いのだ,など結論づけてしまう．しかしこうした議論は言い訳にすぎないため，続けていてもバグ見逃しを防ぐために必要なポイントを適切に抽出することはできない

「そもそも論」は，本質的に過度の抽象化の産物である．過度の抽象化を行うと，妥当な分析を行った納得感を得られる一方で，実際に知識を具体化して再利用する際には，本来は関係しないはずの情報まで具体化されてしまう．知識ツリーの「性能」の１つである "S/N 比"，すなわち再利用しようとする際に必要な情報と関係のない情報との比が低下してしまうのだ．S/N 比の高い知識ツリーを構築するためには，適切に抑えた抽象化を行う必要がある．

重要な点は，バグ見逃しの原因を分析し抽象化することで同様のバグ見逃しの再発を防ぐ，という目的を十分理解しながら抽象化を進めることである．そのためには，必要な組み合わせテスト項目を見逃してしまった原因そのものが，何度も繰り返し発生してしまった理由を深く考察することが必要である．バグ見逃しは単なるうっかりミスではない．見逃すには見逃すだけの合理的な原因があるのだ．原因が繰り返されるからこそ，見逃しも繰り返されてしまう．したがってバグ見逃しを防ぐためには，バグ見逃しが発生する原因を理解し，その原因に着目してテスト設計を改良すべきなのである．決してバグ見逃しを個人の問題に押し付けたり，社風や組織構造，産業構造の問題にすり替えてはいけない．

言い替えると，バグ見逃しは，人間の認知プロセスが持つ弱点が具現化したものであり，誰にでも同じように起こりうる．そのため，見逃しやすい組み合わせテスト項目をレビューし改善していき，組織全体での再発を防ぐという意志が必要なのである．

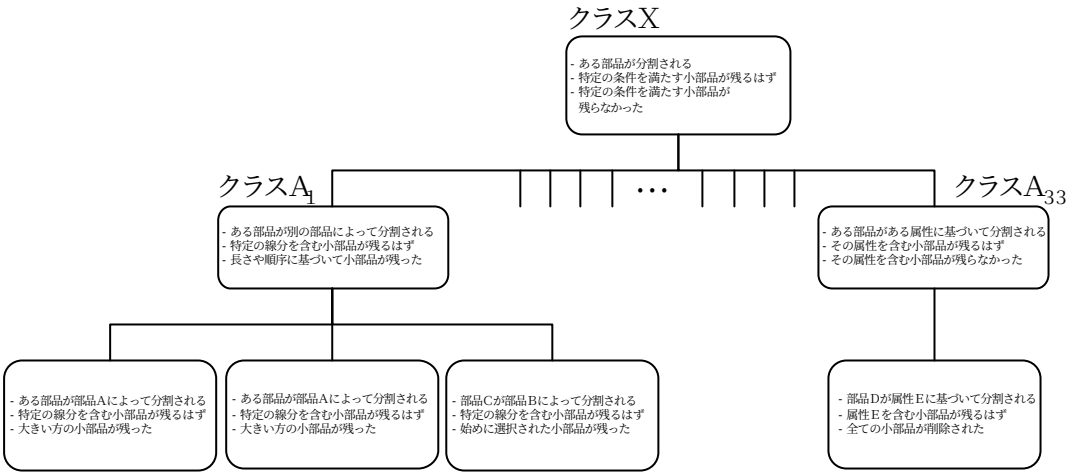


図 8 CAD ソフトウェアにおける仕様レビューに関するバグ見逃しのクラス階層

4. 適用例

4.1. 仕様レビューに対する適用例

本章では, CAD ソフトウェアに対する仕様レビューに本手法を適用した例を示す．3.4 で述べたように，本質的に本手法はテスト設計であってもレビュー（静的テスト)での指摘事項の設計であっても適用できる．

対象とする組織では，複数のデータオブジェクト（部品）同士の組み合わせによる不具合を仕様レビューで見逃していた．一方データオブジェクトの種類は全体で約 20 万件と膨大であり, 仕様レビューで 2 つの組み合わせを全て網羅するような指摘事項を設計するのは非現実的であった．そこで，以前に仕様レビューで見逃した 3 件の不具合から知識ツリーを構築することで，組み合わせを網羅せずに不具合を検出する可能性の高い指摘事項を設計することを目的とした．

まず 3 件のバグ見逃しをインスタンスとして 1 つのクラス A_1 を抽出し，さらにその親クラス X を抽出することで, バグ見逃しの簡単なクラス階層を構築した．そしてインスタンスから抽出されたクラス A_1 とは異なる 32 個のクラス A_n をクラス X の子クラスとして論理的に導出した．構築したクラス階層を図 8 に示す．

次に本手法の効果を評価するために，構築したクラス階層から設計できる指摘事項が不具合を検出できるかどうかを検討した．ここでは，過去の仕様レビューで見逃された不具合が, 32 個のクラスから導出されるインスタンスとしてのバグを基に設計した指摘事項で検出しうるかを精査した．

その結果，4 件の不具合が該当し，7 件の潜在的不具合を指摘可能な項目の設計を行うことができた．対象とする組織が類似の仕様バグを作り込みやすいという経験則が成立するならば，本手法は効果があると考えられる．

5. むすび

テストの工数を発散させずに不具合を検出するには，機能やデータなどの組み合わせによるバグを網羅せずに検出する必要がある．本研究では，バグが偏在することに着目し，過去に見逃したバグの知識をクラス階層として抽象化し構造化することで，テスト設計をフィードバックし改善する手法を提案した．また CAD ソフトウェアに対する仕様レビューに本手法を適用して効果を確認した．今後の課題としては，多くのバグを構造化し汎用的な知識ツリーを作成することが挙げられる．

謝　辞

本研究を進めるにあたり，貴重なご助言を頂きました旭化成株式会社住宅情報システム部の加藤雅樹氏，山田尚睦氏を始め皆様に感謝いたします．

文　献

[1] G. J. Myers, The Art of Software Testing, John Wiley & Sons，New-York，1979，ソフトウェア・テストの技法，近代科学社，東京，1980.

[2] David M. Cohen，Siddhartha R. Dalal，Michael L. Fredman，Gardner C. Patton，The AETG System: An Approach to Testing Based on Combinatorial Design，IEEE Trans. Software Engineering，vol.23，no.7，pp.437-444，July 1997.

[3] C. Kaner，J. Bach，B. Pettichord，Lessons Learned in Software Testing, John Wiley & Sons, New-York，2002.

[4] A. Davis，201 Principles of Software Development，McGraw-Hill，New-York，1995，ソフトウェア開発 201 の鉄則，日経 BP 社，東京，1996.

[5] C. Kaner，J. Falk，H. Q. Nguyen，Testing Computer Software，ITP，Boston，1993，基本から学ぶソフトウェアテスト，日経 BP 社，東京，2001.