CB-001

仕様書からのテストケース設計プロセスの定義とテストケース生成支援ツール
Process for Generating Test Cases from System Specifications in Natural Language and Support Tools

久代 紀之† 青山 裕介† 村上 響一†
Noriyuki kushiro Yusuke Aoyama Kyoichi Murakami

1. はじめに

システム製品の市場での品質確保には、開発最終工程であるシステム試験が重要な役割を果たす。システム試験のテストケースは、一般的に自然言語で記載されたシステム仕様書から、機能を実行する際の条件（充足すべき前提や制約）と、動作（アクションや機能実行後の状態）を抽出することで得られる。

しかし、システム仕様書には、しばしば、曖昧な記述や記述漏れ・誤りが含まれている。テスト設計を進める中で、システム設計者、プログラム設計者を交えたレビューを実施し、不明点・誤りを逐次確認・修正しながら進めていっているのが現実である。この手戻りにより、テストケース設計には、それほど複雑でないシステム製品でも、500～1000 時間程度の膨大な工数が必要となっていた。

また、システム仕様記述の曖昧性や誤りに気づくためには、テスト技術者に、テスト技術のみならず製品知識が要求される。このため、テスト設計により導出されるテストケースの品質は、テスト設計者個々の力量に大きく左右され、新人の育成が難しいという課題があった。

本研究では、テストケース設計作業を、自然言語で記載された機能記述から、機能が実行される際の条件とその機能の実行結果の変換作業として捉え、

1. 熟練のテスト設計者が、暗黙的に行っている上記抽出プロセスをヒアリングと観察に基づき定義
2. 変換作業のうち、機能記述の単文化や単文化された機能記述間の論理的な関係の同定などルール化しやすい定型作業をアルゴリズムにより自動化
3. 上記変換による出力結果を人に分かりやすい形式で提示することで、元になるシステム仕様書自体に含まれているエラーの検出を容易化

することで、これら２つの課題の解決を図ることを試行する。

本論文では、現場でのヒアリング結果に基づき再定義したテストケース設計プロセスと、左記設計プロセス作業を支援するために開発したテストケース生成支援ツール群に関し説明する。さらにこれら設計プロセスと支援ツール群を、システム仕様書“話題沸騰ポット（第３版）” [1]のテストケース設計に適用し、設計プロセスの妥当性と支援ツール群の有用性に関して評価した結果について述べる。

2. テストケース設計プロセスの記述

テストケース設計プロセスを記述するため、企業に所属するテスト設計技術者に、現状のテストケース設計をヒアリング調査した。その結果、おおむね複数企業において、下記のような手順で実施していることが判明した。これら調査結果をもとに、図 1 のようなテストケース設計プロセスを定義した。

† 九州工業大学 情報工学部 知能情報工学科, KIT

システム試験には、システム仕様に記載された機能を満足するかを確認する機能試験と ISO9126 で規定された品質特性を満足するかどうかを確認する品質試験がある。両試験ケースの設計ともに、下記のステップで実施される。

1. システム仕様書の章立てに現れる機能名称をテストケースの大項目として抽出する。
2. システム仕様書記述を熟読し、仕様書に記載された文・表の記述を逐一ラインマーカで塗りつぶしながら、条件・動作文の関係を手作業で抽出する。
3. 条件文をテスト条件、動作文をアクション・状態として、テストケース（小項目）を抽出する。
4. 抽出した小項目に対し、その条件文・動作文に記載された値域から、試験パラメータを決定し、テストケースとして具体化する。
5. 試験手順を考慮し、テストケース（小項目）をクラスタリングし、中項目をとしてまとめる。
6. 上記テスト設計作業中に、発生した疑問点（欠落や誤りへの気づき）については、システム仕様書作成者に確認を行い、逐次修正・補完を行う。

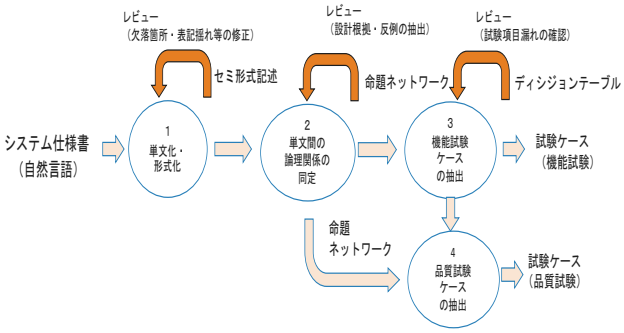


図 1　テストケース設計プロセス

3. レビューの容易化のためのアウトプット設計

前述のように、テスト設計の入力となるシステム仕様書には、誤記を初め、多くの表記揺れ、記述漏れなどのエラーが含まれている。テストケース設計プロセスでは、これらシステム仕様書が内包するエラーを正しながら、テストケースに変換していく必要がある。

一方、単純な誤記を除いては、これら検出・修正には、製品・ドメイン知識を必要とするため、アルゴリズムによる検出は、困難である。技術者間のレビューにより、これらを検出・修正していく必要がある。

このため、図 1 に示す設計の各ステップの出力結果（アウトプット）を、各ステップのレビュー時に検出すべきエラー対象にフォーカスし、レビュー時にエラーが指摘しやすくなるように設計した。以下に、設計した図 1 の各ステップのアウトプットに関して説明する。

3.1 セミ形式記述（単文化・形式化ステップ）

自然言語で記載された仕様書の文には、単文、複文、重文など種々の形式の文が混在している。また、日本語で記載された仕様書では、主語が省略された文も多い。

テスト設計者は、これら自然言語表現がなされた冗長な記述からテストケースに必要な情報のみを抽出し、さらに、欠落部分を補完しつつ作業を進めていく必要がある。これらの作業を容易にするため、本研究では、図 2 に示すようにすべての文を単文化し、これをセミ形式記述 [2]に変換した形式で出力する。セミ形式記述された単文を、命題プリミティブと呼ぶ。図 2 において、欠落部分を“??”で表示することで、レビュー時に欠落部分の補完を促す(図 2 の例では、“ユーザ”が補完される)。

[セミ形式記述]
関係語（主体、対象、[制約, {“,” 制約}]

[例]
原文：設定温度を１００度に設定する。
セミ形式記述：設定する（??、設定温度、１００度）

原文：ユーザが、設定温度を１００度に設定し、蓋をロックする。
ポットは、100度に沸騰する。
セミ形式記述：設定する（ユーザ、設定温度、100度）＆ロックする（ユーザ、蓋）⇒沸騰する（ポット、100度）

図 2　セミ形式記述

また、仕様書原文に表現された命題プリミティブ間の論理関係については、表 1 に示す記号 [3]を用いて記述する。

表 1 命題プリミティブ間の論理関係の記述

論理関係	記号	意味
AND	∧	かつ
OR	∨	または
IMPLY	⇒	ならば
NOT	!	でない
ONE	One	ただ一つ
EXCLUSIVE	E	少なくとも一つ
INCLUSICE	I	多くても一つ
REQUIRE	Req	前提として必須

セミ形式記述の記述文法を EBNF 記法で記述したものを図 3 に示す。

```
/* Written in Extended Backus-Naur Format (EBNF) */
statement = expression "." ;
expression = "(", expression, ")",
  [ "<", constraint, { ",", constraint }, ">" ] |
  expression, "&", expression |
  expression, "|", expression |
  "!", expression |
  clause ;
clause = verb, "(", subject, ",", object,
  { ",", i_constraint }, ")",
  [ "<", p_constraint, { ",", p_constraint }, ">" ] ;
p_constraint =
  [ "!" ], "O", "<", group_name, ">" /* One */ |
  [ "!" ], "E", "<", group_name, ">" /* Exclusive */ |
  [ "!" ], "I", "<", group_name, ">" /* Inclusive */ |
  [ "!" ], "R", "<", direction, ",",
    group_name, ">"                  /* Require */ |
  [ "!" ], "M", "<", direction, ",",
    group_name, ">"                  /* Mask */ ;
direction = "s" | "d"; /* Source or Destination */
group_name = ident;
verb = ident;
subject = ident;
object = ident;
i_constraint = ident;
ident = /* Japanese and English Letters */;
```

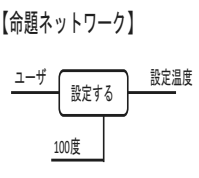
図 3　セミ形式記述文法

3.2 命題ネットワーク（単文間論理関係同定ステップ）

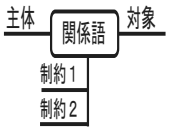
命題プリミティブ間の不適切な論理関係や論理関係の記述漏れの発見を促すため、命題プリミティブ間の論理関係の可視化（命題ネットワークと呼ぶ）を行う。図 2 に示したセミ形式記述を命題ネットワークで可視化した例を図 4 に示す。

【セミ形式記述】
設定する（ユーザ、設定温度、100度）

【命題ネットワーク】



凡例



【セミ形式記述】
設定する（ユーザ、設定温度、100度）＆ロックする（ユーザ、蓋）⇒沸騰する（ポット、100度）

【命題ネットワーク】

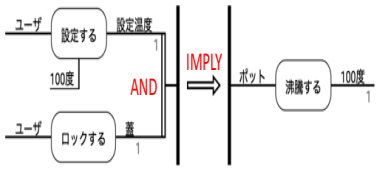


図 4 命題ネットワーク

3.3 ディシジョンテーブル（機能試験ケース抽出）

命題ネットワーク上で命題プリミティブ間の論理関係をレビューした後に、左記論理関係の妥当性を検証するために必要となるテストケースをデシジョンテーブル [4] 形式で提示する。命題ネットワークの前件部をディシジョンテーブルの条件部に、後件部をアクション部に記載することで、ディシジョンテーブルが生成される。表 2 は、図 4 に示した命題ネットワークをディシジョンテーブルに変換したものである。ここでは、

① 命題：設定する（ユーザ、設定温度、100 度）が偽の場合
② 命題：設計する（ユーザ、設定温度、100 度）が真∧ロックする（ユーザ、蓋）が偽の場合
③ 命題：設定する（ユーザ、設定温度、100 度）が真∧ロックする（ユーザ、蓋）が真の場合

の３種のテストケースが生成され、テスト設計者は、このディシジョンテーブルの出力から、実施するテストケースを具体化する。

表 2 ディシジョンテーブル

	試験ケース	0	1	2
条件	設定する(ユーザ,設定温度,100度)	0	1	1
	ロックする(ユーザ,蓋)	-	0	1
動作	沸騰する(ポット,100度)	-	-	1

4. 支援ツールの開発

本研究では、図 1 に示したテストケース設計プロセスの各ステップでの変換作業を支援するツール群を開発した。以下に開発したツール群に関し、概要を説明する。

4.1 自然言語文の単文化・セミ形式化と論理関係導出

自然言語で記載されたシステム仕様書から、命題プリミティブと記述された命題プリミティブ間の論理関係を導出するプロセスを図 5 に記載する。日本語記述の仕様書では、構文解析器 Juman/KNP [4]を用い構文解析した後、この構文解析結果の用言・体言の区分、係受け関係、格種類（ガ格、ヲ格等）、論理関係(AND, OR 等）を抽出する。
さらに、これら構文解析の情報をもとに、自作ツール(Python にて実装)を用い、自然言語文から命題ネットワークを抽出し、命題ネットワーク間の論理関係を同定する。

図 5 セミ形式記述・論理関係の抽出プロセス

上記変換のアルゴリズムを以下に示す。

- 文節が用言（動詞、判定詞、形容詞）の場合は、セミ形式記述の関係語とする。
- 文節が体言の場合は、表 3 の規則に則り、主体、対象、制約に割り当てる。
- 係り受け情報から、表 4 の規則に則り、命題ネットワーク間の関係を決定する。

表 3 セミ形式記述の割り当て決定ルール

種別	構文解析結果の Feature
主体	< 解析格 : ガ >, < ハ : True> かつ < 主題表現 : True>
対象	< 解析格 : ヲ >, < 係 : ヲ格 >
制約	< 係 : ニ格 >, < 係 : ヘ格 >, < 係 : カラ格 >, < 係 : ヨリ格 >, < 係 : デ格 >, < 係 : マデ格 >, < 係 : ノ格 >, < 係 : ト格 >

表 4 命題ネットワーク間の論理関係決定ルール

関係	構文解析結果の Feature
And	< 並列タイプ : AND>, ID : ～て (用言)>, <ID : ～で (判)>, <ID : ～ても >, <ID : ～ながら >, <ID : ～て > かつ < 用言 : 動 >
Or	< 並列タイプ : OR>, <ID : ～たり >
Imply	< 外の関係 : True> かつ「場合」,「時」,「際」,「限り」、< 外の関係 : True> かつ <ID : ～ため >, < 外の関係 : True> かつ < 時間 : True> かつ < 相対名詞 : True>,< 係 : 連用 > かつ <ID : ～ば >, <ID : ～たら >, <ID : ～ので >, <ID : ～が >, <ID : ～ように >, < ト : True>

4.2 命題ネットワークを用いたセミ形式間の論理関係の可視化

命題プリミティブは、図 6 に示すアルゴリズムを実装した自作ツール（JavaScript で実装）を用い、命題ネットワークに変換される。命題ネットワークでは、図 4 の形式で表現した命題プリミティブを葉として、論理演算子・命題間論理制約関係をもとに空間上に配置することで、その論理関係を可視化する。

Definition:
T: a set of node types;
$T = \{Atomic, And, Or, Imply\}$
n: a node; $n = (t, A)$
A: the arguments of a node;

$$A = \begin{cases} (s, o, r, C) & (t = Atomic) \\ \text{a set of } N & (t \neq Atomic) \end{cases}$$

s: subject
o: object
r: relation
C: a set of constraints

Input:
n_l: a node; $n_l = (t_l, A_l)$
n_r: a node; $n_r = (t_r, A_r)$
Output:
N_m: a set of merged nodes

```
1:  procedure MERGE(n_l, n_r)
2:      A_new ← {}
3:      merged ← false
4:      if t_l = t_r = Atomic then
5:          C_l ← C of A_l
6:          C_r ← C of A_r
7:          if (A_l − C_l) = (A_r − C_r) then
8:              return {(t_l, (A_l − C_l) ∪ (C_l ∪ C_r))}
9:          end if
10:         return {n_l, n_r}                          ▷ not merged
11:     else if t_l = Atomic ∨ t_r = Atomic then
12:         define t_k (t_k ∈ {t_l, t_r}, t_k ≠ Atomic)
13:         for all n ∈ A_k do
14:             A_merged ← Merge(n, n_k)
15:             A_new ← A_new ∪ A_merged
16:             merged ← merged ∨ |A_merged| = 1
17:         end for
18:         if merged then
19:             return {(t_k, A_new)}
20:         end if
21:         return {n_l, n_r}                          ▷ not merged
22:     end if
23:     if t_l = t_r then                              ▷ t_l or t_r is not Atomic
24:         if ∃n ∈ A_r such that n ∈ A_l then
25:             return (t_l, A_l ∪ A_r)
26:         end if
27:         return {n_l, n_r}                          ▷ not merged
28:     end if
29:     for all n ∈ A_l do
30:         A_merged ← Merge(n, n_r)
31:         A_new ← A_new ∪ A_merged)
32:         merged ← merged ∨ |A_merged| = 1
33:     end for
34:     if merged then
35:         return {(t_l, A_new)}
36:     end if
37:     return {n_l, n_r}                              ▷ not merged

38: end procedure
```

図 6 命題ネットワークアルゴリズム

4.3 フレームワークを用いたレビュー支援

システム仕様書には、一般的に正常なユースケース（基本系列）と異常なユースケース（代替系列）が記述されている。節 3.2 で説明した命題ネットワークによる可視化により、命題プリミティブ間の論理的妥当性の確認を支援することができる。

一方、品質特性試験項目の抽出には、上記基本系列・代替系列に明示的には記載されていない前提・制約が充足されなくなる異常ケースの検討が必要とされる。明示されていない前提・制約の抽出は、システム仕様記述から抽出された命題ネットワークをレビューするだけでは、充分ではない。

本研究では、この支援のために、蓋然性の高い議論のフレームワークである Toulmin's model [5]を導入し、これを命題ネットワークの設計根拠・反例抽出レビューのためのフレームワークとして活用する。

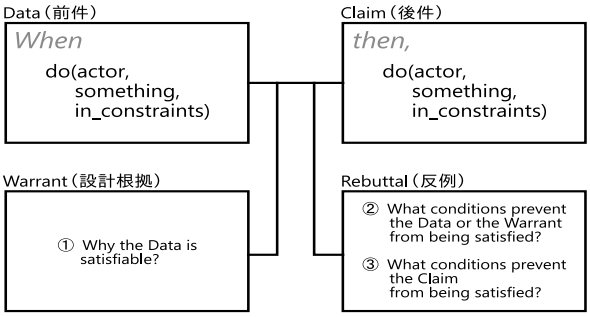


図 7 レビューフレームワークとしての Toulmin's model

Toulmin's model を用いたレビュー（図 7）においては、仕様書変換で抽出された前件部、後件部に対し、明示的に仕様書に記載されることが少ない設計根拠と、“前件⇒後件”、“設計根拠”が成立しなくなる反例を、提示したレビューフレームワークにより、半ば強制的に補完させることで、記述された仕様書記述の背後にある前提や制約事項の抽出を支援する。

4.4 ディシジョンテーブルの生成

以上のプロセスを経て修正・補完されたセミ形式記述から、デシジョンテーブル形式 [6]のテストケースを生成する。前述のように、セミ形式記述では、仕様書文章が論理式として表現されている。Imply の形式で記載された機能仕様において、前件で記載された論理式を入力条件とし、後件で記載された論理式をアクションとすることでテストケースを得ることができる。

論理的に可能な入力条件の組合せを得るために、PyEDA [7]にて、充足可能性問題を解くことで、可能な入力条件を求める。この入力条件の組合せと、アクションを表形式で埋めた表に自作ツールで整形する。また、膨大になりがちな入力条件の組合せに対処するため、Don't care という、真偽どちらの値をとっても論理式全体の真偽に影響しないことを意味する割り当て値を用いて、デシジョンテーブルとして生成される試験ケース数を縮小することができるようにした。

5. 評価

本研究で定義したテストケース設計プロセスの妥当性と開発したツールの有用性を話題沸騰ポット（第３版）のテストケース設計に適用し、評価した。

5.1 評価実験の概要

評価実験の概要として、被験者プロファイル、実験手順および、本研究で開発した支援ツールを用いて、話題沸騰ポットの仕様を変換した結果について以下に示す。

5.1.1 被験者プロファイル

評価実験は、異なる製品ドメインを担当する２チームを構成し、同一の題材および同一の実験プロセスで実施した。

評価実験に参加した被験者のプロファイルを表 5 に記載する。チーム１ （4 名）は、主に産業・民生用の AV 機器関連のテスト設計技術者、チーム２ (7 名)は、組込機器の制御ソフト開発者・テスト設計者で構成されている。それぞれのチームは、実務の際のレビューメンバー構成を想定し、経験豊富な技術者と比較的経験年数の浅い技術者が混在するように構成した。また、製品知識の多寡が、評価実験に影響しないように、両チームともに専門外の製品となる電気ポットの仕様書を評価対象として選定した。

表 5 評価実験被験者プロファイル

チーム	被験者	業務経験	職務
1	A	19	プロセス改善担当
	B	31	検証管理担当
	C	5	テスト設計担当
	D	3	テスト設計担当
2	E	15	制御ソフト開発担当
	F	3	SW 品質改善担当
	G	7	制御ソフト開発担当
	H	3	制御ソフト開発担当
	I	3.5	通信制御ソフト開発担当
	J	14	SW 品質管理担当
	K	0.5	組込ソフト開発担当

5.1.2 実験手順

評価実験は、以下のプロセスで実施し、プロセス配分時間を（ ）内に示す。実験時間は、テスト内容の説明から実施終了まで約２時間とした。

【実験プロセス】

1. 実験概要の説明（15 分）
2. 対象とするシステム仕様書の説明（話題沸騰ポット第３版のシステム仕様記述）（15 分）
3. ツールによりセミ形式記述化した仕様書の欠落部分の補完と仕様書のレビュー（20 分）
4. ツールにより命題ネットワーク記述した欠落部分（記述漏れ・論理的関係）の補完とレビュー(20 分)
5. Toulmin's model を用いた前提・制約事項の抽出（15 分）
6. ツールによる生成したディシジョンテーブル上のテストケースの冗長部分の確認とレビュー（20 分）
7. プロセス・ツールに関するアンケートの実施（20 分）

5.1.3 実験の評価対象とした仕様記述

テストケース設計の対象とする話題沸騰ポットの仕様記述（自然言語）は、以下の通りである。

【評価対象とした仕様記述】

仕様０:システム全体は、以下の動作仕様を満足する。
仕様１：第 n 水位センサが on で、かつ満水センサが off の場合、ポットは、温度制御ができる。
仕様２：それ以外なら、沸騰ボタン・ヒータは動作しない。
仕様３：蓋が開けられると、ヒータは停止する。
仕様４：ヒータが動作していない時は、沸騰ランプは消灯し、保温ランプは消灯する。
仕様５：保温モードに設定し、かつ 100℃でなかった場合は、一度沸騰させて、設定温度に保つ。
仕様６：ユーザが、ボタンを押すと、ポットは、ブザーを１回鳴らします。

仕様７：沸騰ボタンを２つの制約下（（１）に記載の条件）で押された時は、ブザーを鳴らさない。
仕様８：ユーザの設定したタイマがタイムアウトし、かつ沸騰状態終了時には、ブザーを３回鳴らす。

5.2 ツールによる変換結果

本研究で開発した変換ツールによる節 5.1.3 に示した仕様記述のアウトプット結果を以下に示す。

5.2.1 セミ形式記述

節 5.1.3 に記載した自然言語で記載された仕様を開発したツールでセミ形式記述に変換した結果を図 8 に示す。図中 ?? 部分が欠落部分を表す。

```
仕様0  満足する (システム, 以下の動作仕様).
仕様1  である (第ｎ水位センサ,on) & である (満水センサ,off)-> 出来る (温度制御,??).
仕様2  である (??, それ以外)->!動作する (沸騰ボタン・ヒーター,??).
仕様3  開ける (??, 蓋)-> 停止する (ヒーター,??).
仕様4  !動作する (ヒーター,??)-> 消灯する (沸騰ランプ,??) & 消灯する (保温ランプ,??).
仕様5  設定する (??,??, 保温モードニ) & である (??,100[℃])-> 一度沸騰する (??,??) & 保つ (??,??, 設定温
       度ニ).
仕様6  押す (ユーザー, ボタン)-> 1 回鳴らす (ポット, ブザー).
仕様7  押す (??, 沸騰ボタン,  2つの制約時)->!鳴らす (??, ブザー).
仕様8  タイムアウトする (ユーザの設定したタイマー,??,  沸騰状態終了時ニ) -> 3 回鳴らす (??, ブザー).
```

図 8 ツールによるセミ形式記述への変換

5.2.2 命題ネットワーク

セミ形式記述の欠落部分をレビューで補完後に、開発ツールで命題ネットワークに変換した結果を図 9 に示す。

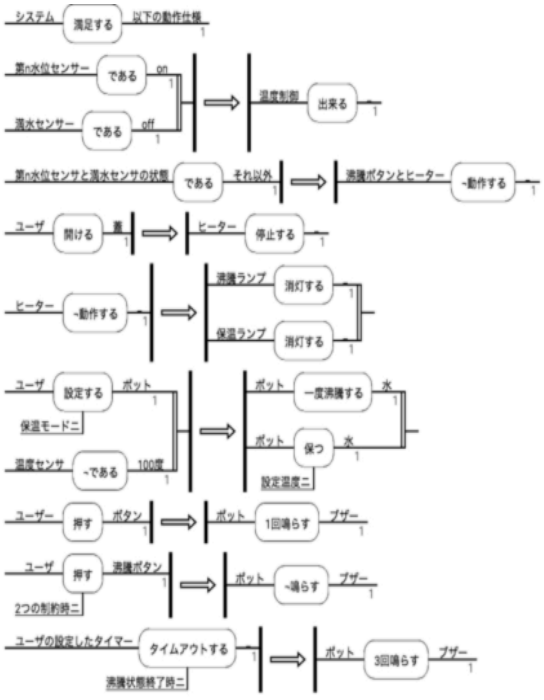


図 9 ツールによる命題ネットワークへの変換

5.2.3 ディシジョンテーブルの生成

図 9 の命題ネットワークの論理的な欠落箇所を Toulmin's model に基づくレビューにより補完したものを、開発ツールを用いデシジョンテーブルに変換したものを図 10 に示す。

		0	1	2
条件	である(満水センサ,off)	1	0	1
条件	である(第n水位センサ,on)	1	-	0
動作	出来る(温度制御,)	1	-	-

		0	1	
条件	である(第n水位センサと満水センサの状態,それ以外)	1	0	
動作	動作する(沸騰ボタンとヒーター,)	0	-	

		0	1	
条件	開ける(ユーザ,蓋)	1	0	
動作	停止する(ヒーター,)	1	-	

		0	1	
条件	動作する(ヒーター,)	0	1	
動作	消灯する(保温ランプ,)	1	-	
動作	消灯する(沸騰ランプ,)	1	-	

		0	1	2
条件	である(温度センサ,100度)	0	-	1
条件	設定する(ユーザ,ポット,保温モードニ)	1	0	1
動作	一度沸騰する(ポット,水)	1	-	-
動作	保つ(ポット,水,設定温度ニ)	1	-	-

		0	1	
条件	押す(ユーザー,ボタン)	1	0	
動作	1回鳴らす(ユーザー,ブザー)	1	-	

		0	1	
条件	押す(ユーザ,沸騰ボタン,2つの制約時ニ)	1	0	
動作	鳴らす(ポット,ブザー)	0	-	

		0	1	
条件	タイムアウトする(ユーザの設定したタイマー,,沸騰状態終了時ニ)	1	0	
動作	3回鳴らす(ポット,ブザー)	1	-	

図 10 ツールによるディシジョンテーブルへの変換

5.3 評価結果

本研究では、テストケース設計プロセスの妥当性、ツールのアウトプットの分かりやすさ、各アウトプットを対象としたレビューの質の改善に関し、評価した。

5.3.1 テストケース設計プロセスの妥当性評価

本研究で定義したテストケース設計プロセスに沿って、テストケースの生成ができたことから、テスト設計プロセスは、妥当であると言える。また、実験終了後にアンケート調査から、下記ポジティブな意見の収集できた。

1. 仕様書から試験ケースまで変換する一連の流れは、理解しやすく、ステップ毎に修正可能である点が良い。
2. セミ形式化・命題ネットワーク記述により、動作仕様書の補完が容易になった。補完により、テストケースの網羅性を高めることができる。
3. 形式的な変換の自動化とアウトプット毎のレビューにより手戻りを削減し、テスト設計時間の大幅な時間短縮が期待できる。
4. 文章を図形のように可視化することで、構造が理解しやすく、レビューがやりやすい。
5. 文書で書かれた仕様書では、何が抜けているかなどに気がつかなかったが、セミ形式記述や命題ネットワークで表現されることで、何かがおかしいということに気づくことができた。

一方、改善を要する点として、

- 各ステップで発見されたエラーを元のシステム仕様書に反映するステップやツールが必要
- 仕様記述毎にテストケースが抽出されるが、テストケースの実施手順の検討には、同じ前件部あるいは後件部を持つテストケース毎にマージし、このまとまり毎に中項目のテスト項目を検討するステップが必要

などの意見が抽出された。

5.3.2 ツールのアウトプットの分かりやすさ

ツールのアウトプットの分かりやすさに関しては、表 6 に示すように、おおむね高い評価を得られた。

表 6 ツールのわかりやすさの評価

アウトプット	全体	チーム 1	チーム 2
セミ形式記述	3.82 ± 0.71	4.50 ± 0.50	3.43 ± 0.49
命題ネットワーク	4.18 ± 0.71	3.75 ± 0.43	4.43 ± 0.49
ディシジョンテーブル	3.36 ± 0.64	3.00 ± 0.00	3.57 ± 0.73

チーム 1 においては、セミ形式記述のわかりやすさの評価が高く、チーム 2 においては、命題ネットワークの評価が高い。これは、チーム 1 は、通常業務で、自然言語で記載された仕様書を主体にテストケース設計作業を行っているのに対し、チーム 2 は、状態遷移表や状態遷移図を対象にテスト設計を行っていることが影響している可能性がある。つまり、自然言語の仕様書に重点を置くチームでは、セミ形式記述からの抜け・誤記の修正への貢献を高く評価したのに対し、チーム 2 は、相対的に自然言語の仕様記述の重要性が低く、状態遷移が論理として可視化される命題ネットワークに高評価を与えたものと考えられる。一方で、両チームともに通常業務で使用しているデシジョンテーブルのわかりやすさの評価が、他のアウトプットに比較して低い。これは、条件・動作文に記載された命題の真偽（特に、偽の場合の）意味的な解釈が難しいことが要因と想定される。

5.3.3 レビューの質の改善

熟練のテスト設計者と比較的経験の浅い技術者が、ツールの各アウトプットに対してレビュー時に指摘できた事項に関し、その差異を分析した。

セミ形式記述上での欠落部分の補完作業におけるテスト設計の業務経験年数と補完した欠落部分の正答数の関係を図 11 に示す。また、命題ネットワーク記述上での論理誤り、論理の抜けを抽出した際のレビュー時の指摘項目として有効と判断される指摘数と業務経験年数との関係を図 12 に示す。さらに、Toulmin's model を用いたレビューにおいて指摘できた設計根拠、反例数と業務経験年数の関係を図 13 に示す。

本研究では、

- 参加者が、ほぼ同程度の製品知識を保有すると考えられる電気ポットのシステム仕様を対象としたこと、
- 従来テスト設計者の頭の中で暗黙的に行われていたシステム仕様からの条件・動作の抽出過程とその作業を支援ツールにより自動化したこと
- 各ステップのレビューにおける重要視点を設定し、左記視点でのエラー検出がしやすいようなアウトプットの設計を行ったこと

で、技術者の既得のドメイン知識以外の要因をできるだけ均一化した。

この結果、プロセス・ツールの適用により、システム仕様書の欠落部分の補完、仕様書記述の論理的な漏れ・誤りの抽出に関しては、熟練技術者と経験の浅い技術者間の差異を無くすことができた。

経験の浅い技術者は、ツールのアウトプットをベースに妥当性を判断することで、より精度の高い補完や漏れ・誤りの抽出ができることが観察された（図 11, 図 12）。一方、熟練技術者は、セミ形式記述、命題ネットワークを対象とするレビュー時においても、自然言語によるシステム仕様記述を頻繁に参照し検討することが観察され、自然言語で記載された仕様書の曖昧さを引き摺り、欠落部分の補完、論理的な誤り・漏れの抽出に失敗してしまうことが観察された（図 11）。

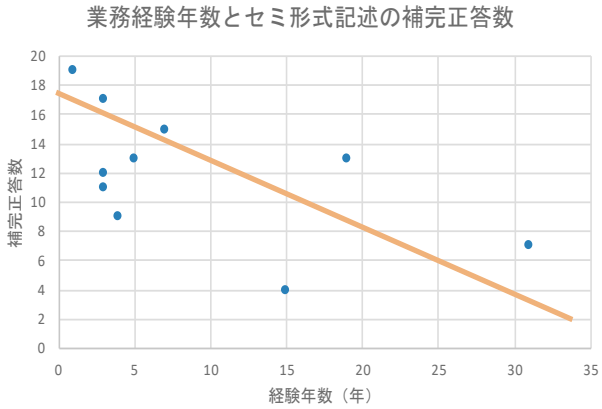


図 11 業務経験年数とセミ形式記述における補完正当数

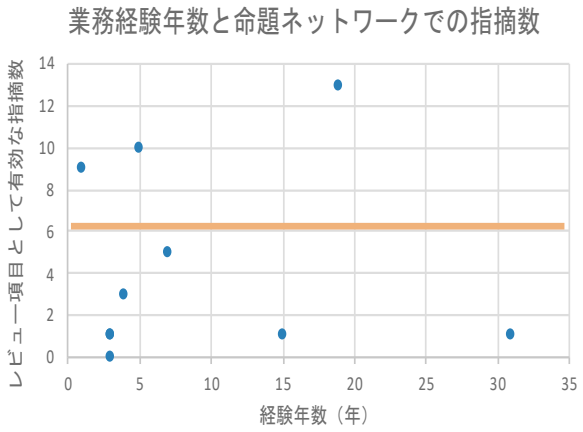


図 12 業務経験年数と命題ネットワークでの有効指摘数

一方で、Toulmin's model 上での設計根拠や反例を抽出するレビューにおいては、熟練者のテスト設計に関する技術

および製品知識が有効に働き、重要なレビュー項目の抽出ができることが観察された（図 13）。ただし、３～５年程度の業務経験年数が低い技術者においても、熟練技術者と同等の指摘ができていることから、個々の技術者の科学・工学的な知識の多寡が、レビューでの指摘に影響することが想定される。

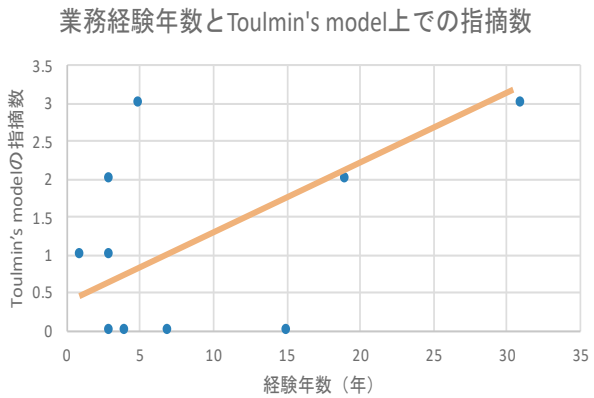


図 13 業務経験年数と Toulmin’s model 上での指摘数

5.3.4 プロセス・ツールへの改善要望

本評価実験を通じて得られたプロセス・ツールへの改善要望を以下に挙げる。

【プロセスのアウトプットに関する要望事項】

- セミ形式記述、命題ネットワークに時相論理の表現の追加（仕様 5：保温モードに設定し、かつ 100℃でなかった場合は、一度沸騰させてから、設定温度に保つ のような記述では、時間的経過を扱える記述方式が必要）
- ディシジョンテーブルの可読性の向上機能の追加（項目間の関係の表示、状態・動作の分離表示）

【支援ツールに関する要望事項】

- セミ形式記述・命題ネットワーク記述で、補完・修正した項目を編集する機能
- 上記追加した項目の元仕様書への反映機能の追加
- 命題ネットワーク上で Toulmin’s model を用いたレビューを支援する機能
- ネットワーク上で、任意の命題プリミティブを選択することでのネットワーク再構成の機能（関連する前件・後件の検索を支援する機能）

6. おわりに

本研究では、テストケース設計作業を、自然言語で記載された機能記述から、機能が実行される際の条件とその機能の実行結果の抽出作業として捉え、

1. 熟練のテスト設計者が、暗黙的に行っている上記抽出プロセスをヒアリングと観察に基づき定義
2. 変換作業のうち、機能記述の単文化や単文化された機能記述間の論理的な関係の同定などルール化しやすい定型作業をアルゴリズムにより自動化
3. 上記変換による出力結果を人に分かりやすい形式で提示することで、元になるシステム仕様書自体に含まれているエラーの検出を容易化

することで、テストケース設計の課題の解決を図ることを試行した。

実際の現場での設計手順に基づき再定義したテストケース設計プロセスと、左記プロセスの作業を支援するツール群を開発し、これらプロセスと支援ツールを、システム仕様書“話題沸騰ポット（第３版）”のテストケース設計に適用し、プロセスの妥当性とツール群の有用性に関して評価した。その結果、以下の事項を確認した。

定義したプロセスの妥当性については、定義したプロセスによりテストケースの抽出が可能であること、プロセス各ステップのアウトプットのレビューにより、漏れ・誤り等の仕様書のエラーを抽出することが可能であることを確認した。

支援ツールのアウトプットの有用性の有用性については、アウトプットのわかりやすさについてのアンケート結果から良好な評価が得られた。また、システム仕様書の欠落部分の補完、仕様書記述の論理的な漏れ・誤りの抽出に関しては、ツールのアウトプットを対象としたレビューにおいて、熟練技術者と経験の浅い技術者間の差異をなくすことが可能であることを確認した。

一方で、現状の支援ツールには、節 5.3.1、節 5.3.4 に示した課題も指摘されており、今後、これらの課題を改善すべくツール完成度を上げるとともに、企業における実際のテスト設計への適用を進めていくことで、テストケース設計ツールとしての実用性を高めていく。

謝辞

本研究は，JSPS 科研費 16K00100， 16H01836，JST CREST の支援を受けて実施したものである。また、評価実験に協力いただいたパーソルＡＶＣテクノロジー株式会社殿、三菱電機株式会社殿に、深く感謝の意を表します。

参考文献

[1] 組込みソフトウェア管理者・技術者育成研究会 (SES- SAME), “話題沸騰ポット第 3 版”.http://www.sessame.
[2] C.Rolland and C.B.Achour, “Guiding the construction of textual use case specifications”, Data & Knowledge Engineering, Vol. 25, Issues 1-2, pp.125-160, ELSEVIR, (1998)
[3] J.Mayer:”ソフトウェアテストの技法 第 2 版”，近代科学社，(2006)
[4] D.Kawahara and S.Kurohashi: “A Fully-Lexicalized Probabilistic Model for Japanese Syntactic and Case Structure Analysis”, Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL2006), pp.176-183, (2006)
[5] Kirschner, Paul A., Simon J.Buckingham-Shum, and Chad S. Carr, eds. Visualizing argumentation: Software tools for collaborative and educational sense-making. Springer Science & Business Media, (2012)
[6] B.Beizer, “Software testing techniques,” pp.269-276, Dreamtech Press, second edition, 2003. 小野 間彰 and 山浦恒央 (訳). ソフトウェアテスト技法, 日経 BP 出版 センター, (1994)
[7] C.Drake: Python library for electronic design automation (PyEDA), https://media.readthedocs. org/pdf/pyeda/v0.28.0/pyeda.pdf, 2011. Last accessed: May 23, (2018).