Vol.2022-SE-211 No.4
2022/7/28

仕様書⇔テストケース双方向自動生成による仕様書の再利用支援

若松 大雅[1,a)]　久代 紀之[1,b)]

概要：システム仕様書には, 一般的に漏れや誤りといった不具合が含まれる．テスト設計時には, 仕様書各文を精読しテストケースを抽出する必要があるため, これら不具合がテスト設計時に発見されることも多い．一方, 後工程で発見された不具合は, 元仕様書に反映されないことが多く, 仕様書再利用時の大きな課題となっている．また, 現状の多くの仕様が表計算ソフト等を用いたアプリケーション依存の形式で管理されており, 再利用時の検索・バージョン管理が困難であるといった課題もあった．本研究では上記課題の解決のため, 自然言語・表・UML 図から構成される仕様書をマークアップ言語により記述すると共に, これらを形式記述に自動変換し, 形式変換した仕様書を論理的に充足するテストケースを自動生成するツールを開発した．更に, 形式記述の修正を, マークアップ言語で記述した仕様書に自動反映させるツールを開発した．これらのツールの実現により, 仕様書作成と同時にテストケースを通じて仕様書記述の妥当性を確認することが可能となる．また, 仕様書全体の高品質化とその維持を担保すると共に, 検索性・管理性の向上により, 仕様書の再利用性を向上することを試行した.

キーワード：形式記述変換, テストケース自動生成

Support for reuse of specifications through specification ⇔ test case bidirectional automatic generation

1. はじめに

システム開発において，ソフトウェアテストは非常に重要な工程である．このテスト工程においてテスト設計者は仕様書各文を精読し，手作業で条件とそれに対する期待値を抽出することでテストケースを設計する．しかし，一般的に仕様書には漏れや誤り等の不具合が含まれており，これらの不具合がテスト工程で発見されることが多い．漏れや誤りの要因として，自然言語記述による曖昧性や設計者個々が有するドメイン知識（設計根拠）への依存が考えられるが [1]，現状では仕様書を作成する段階でこの不具合の排除は困難である．また，開発途中で発見された不具合は元仕様書に反映されていないケースが多い．更にシステム仕様書やテストケースは表計算ソフト等のアプリケーション依存の形式で管理されており，再利用時の検索・バージョン管理が非常に困難である．このようにシステム仕様書及びテストケースの再利用において多くの障壁が存在する [2], [3], [4]．上記を解決するために，自然言語・表・図をからなる仕様書をテキストベースで記述することを提案する．本研究では AsciiDoc [5] と呼ばれるマークアップ記述を用いる．AsciiDoc は自然言語や表はもちろん，PlantUML [6] のプラグインを用いることで UML 図もテキストベースで記述可能である．筆者らは図や表を含む仕様書の形式記述とテストケースとしてデシジョンテーブルを自動生成するシステムを開発しており，更に先行研究 [9], [10] では自動生成したデシジョンテーブルから実行可能なテストプログラムを自動生成し，これを自動実行する実行環境を提案している．本研究ではこれらのツールを統合し，マークアップ形式で記述された仕様書を形式記述，テストケースへと自動変換し，更にそこに含まれる漏れや誤りの修正を仕様書・形式記述・テストケースの全てのドキュメントに同期するシステムを開発した．これにより，仕様書作成と同時にテストケースを確認することができ，

[1] 九州工業大学 大学院情報工学研究院 情報創成工学研究科
Kyushu Institute of Technology
[a)] wakamatsu.taiga418@mail.kyutech.jp
[b)] kushiro@csn.kyutech.ac.jp

仕様書記述の妥当性を評価することが可能となる．また，仕様書全体の高品質化とその維持を担保すると共に，検索性・管理性の向上により仕様書の再利用性の向上を図った．近年はテスト駆動開発に基づく開発体制の確立が，システム開発の効率化・高品質化の視点から強く望まれており，システム仕様書からのテストケース自動生成技術にも注目が寄せられているが [7]，本研究はテストファースト [8] に加え，本ツールを用いたテストケースファーストを提案することで開発体制の強化を狙う．以下，2 章にて仕様書の形式化・テストケースの自動生成について述べ，3 章にて本研究で開発したツールについて述べる．4 章にてツールを用いた評価実験について述べ，5 章でその考察を述べる．6 章にて締めくくる．

2. 形式記述とテストケースの自動生成

2.1 形式記述の自動生成

仕様書に含まれる自然言語・表・UML 図に対し，本研究ではセミ形式と呼ばれる形式記述に変換する．変換するツールとして先行研究 [9], [10] で開発されたツール (semi-formalizer) を適用する．セミ形式とは「関係語 (主体語, 対象語, 制約語)」で構成される命題プリミティブと，それを命題論理の 4 つの論理演算子 `Not:「!」,And:「&」,Or:「|」,Imply:「→」` で相互接続した形式で表現する．一例を図 1 に示す．

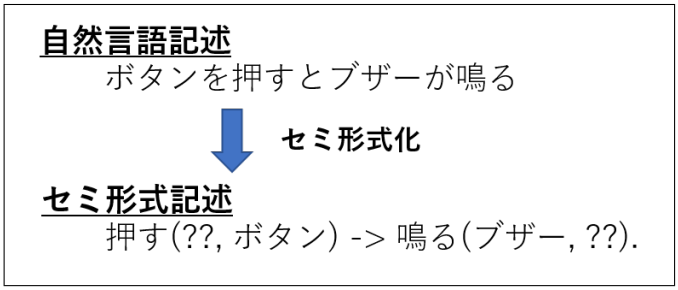

図 1 セミ形式記述の例

“ボタンを押すとブザーが鳴る” は，“ボタンを押す” という動作条件と，“ブザーが鳴る” という確認事項から構成されている．動作条件，確認事項はそれぞれ命題プリミティブへと変換され，その 2 つが包含関係 (→) で接続されている．セミ形式中の`??` は欠落箇所を表し，図 1 の例では “ボタンを押す” に対する主体，“ブザーを鳴らす” に対する補語が欠落していることを示している．semiformalizer におけるセミ形式記述変換は以下のルールで記述されたアルゴリズムにより実現される．

(1) 文節, 関係語とその他要素の同定：用言を関係語とし, 文節と関係語, その他の要素に分ける．形態素解析器 Juman++で未定義となる文字を置換 (例：半角カタカナを全角カナに置換) し, 文の終了スタイルを統一する (例：「…すること」を「…する」のように, 用言で文を打ち切るスタイルに統一する).
(2) 文節の接続：文節間に係り受け関係がある場合, 文節を接続し, 句とする．
(3) 句を命題プリミティブに接続：形態素解析器 Juman, 及び日本語構文解析器 KNP から得られる解析格を基に, 主体, 対象, 制約に当てはめる．ガ格の句を主体, ヲ格の句を対象, それ以外の修飾句を制約として扱う．
(4) 命題と命題を接続：AND：「&」,OR：「|」などの条件文の同定を行う．これは次のルールで記述されている．
- 末尾の文節が接続助詞「か」,「または」の時は OR：「|」とする．
- 末尾の文節が接続助詞「かつ」の時は OR：「|」とする．
- 末尾の文節が条件系の用言の時,Imply：「→」とする．
- 上記のいずれでもないときはデフォルト値として AND：「&」が選択される．

以上の手順により自然言語で記述された仕様書各文をセミ形式記述へ変換する．

2.2 テストケースの自動生成

2.1 節で記述したセミ形式記述では，仕様の動作条件・確認項目を命題プリミティブとし，論理演算子でつないだ論理式として表現しているため，命題プリミティブの真理値割当を求めることでデシジョンテーブルへの変換が可能になる．このデシジョンテーブルを用い充足問題を解くことにより，論理関係の組合せを網羅したテストケースとすることができる．本研究ではテストケースを生成するツールとして， 先行研究で開発された `Splosat` と呼ぶツールを用いた (図 6)． `Splosat` では，`PyEDA` [11] というツールを用いて命題論理式の真理値割当を行う．セミ形式では「動作条件 → 確認項目」と表現されるため，前件の命題プリミティブをデシジョンテーブルの条件欄に，後件の命題プリミティブをデシジョンテーブルの動作欄に埋めることで自動的にデシジョンテーブルを生成する．また，各命題プリミティブが真偽のどちらの値をとっても論理式の真理値に変化がない際には，`Don't Care`(デシジョンテーブル内の「-」) に表記を縮約することで，テーブルサイズが膨大になることを防いでいる．図 6 の例では，“押す (`??`, ボタン)” が動作条件，“鳴る (ブザー,`??`)” が確認項目となる．“動作条件 → 確認項目” とすることでこのようなデシジョンテーブルを生成することが可能であり，これをテストケースとすることができる．

3. ツールの概要

3.1 ツールのユースケース

仕様-テストケース間トレースツールの構成を図 3 に示

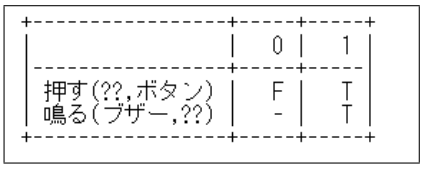

図 2 デシジョンテーブル作成例

す.2 章で記述した Semiformalizer と Splosat を Python で提供される subprocess モジュールを用いて実行することで機能を実現する．本ツールは Visual Studio Code を用いて実行することを想定しており，ツールを更に有効に使うために，AsciiDoc のプレビュー，plantUML のプレビューなどの各種拡張機能のインストールを推奨する．そうすることで仕様書中の表や UML 図が視覚的に調整されるようになり，レビュー性が格段に向上する．

3.2 フォワードトレース

自然言語や表・UML 図からセミ形式を生成する機能をフォワードトレースと呼ぶ．本ツールは自然言語に加え，表・UML シーケンス図・ステートマシン図に対応しており，記述方法の分類は表 1 のように行われる．

表 1 記述方法の分類

記述方法	分類方法
自然言語	図と表の記述方法に当てはまらない場合
表	以下の形式に当てはまる場合 . 表題 |=== | 条件 | 期待値 | 条件 A| 期待値 B |===
図	以下の形式に当てはまる場合 ・シーケンス図 [plantuml] ---- 'sequence diagram 本文 ---- ・ステートマシン図 [plantuml] ---- 'statemachine diagram 本文 ----

尚，表 2 のマークアップ形式に対応しており，それ以外の記述方法には対応しておらず，必要に応じて今後適宜ルールを拡大する必要がある．

自然言語は 2.1 に記載したルールで変換を行い，表・UML 図のセミ形式の変換は以下のルールを適用する．

(1) 表

AsciiDoc 形式で入力される記述に対し，表 3 のルールを適用する．

表 2 対応しているマークアップ形式の制御文字

制御文字	変換時の処理
改行 (文末に「 +」)	「 +」による改行が含まれる文は，「 +」がの取り除かれた状態でセミ形式に変換される．
コメントアウト (//)	コメントアウト部分はセミ形式に変換されず，文頭に「##」を付けて出力される．
箇条書き (文頭に「. 」「* 」「= 」)	箇条書き部分はセミ形式に変換されず，文頭に「#」を付けて出力される．
キャプション (文頭に「.」)	キャプション部分はセミ形式に変換されず，文頭に「#」を付けて出力される．
オプション (「[」「]」でくくられた文)	オプション部分はセミ形式に変換されず，文頭に「#」を付けて出力される．

表 3 表の変換ルール

変換要素	変換時の処理
|=== | 条件 | 期待値 | 条件 A| 期待値 B |===	is(条件, _, 条件 A) -> is(期待値, _, 期待値 B).
セル結合 (横)	セミ形式への変換時，セル結合は解除される．
空のセル (ヘッダ以外)	空のセルがある場合は真上のセルの内容が入力された状態でセミ形式へ変換を行う (ただし，真上がヘッダの場合は空のセルが入力される)．

(2) UML シーケンス図

AsciiDoc 上で plantUML 形式で入力される記述に対し，表 4 のルールを適用する．

表 4 シーケンス図の変換ルール

変換要素	変換時の処理
A -> B : Request	Request(A, B).
A --> B : Response	以前の文に「A -> B: Request」がある場合「Request(A, B) -> Response(B, A).」に変換される．
コメントアウト (文頭に「'」)	コメントアウト部分はセミ形式に変換されず，文頭に「#」を付けて出力される．

(3) UML ステートマシン図

AsciiDoc 上で plantUML 形式で入力される記述に対し，表 5 のルールを適用する．

表 5 ステートマシン図の変換ルール

変換要素	変換時の処理
state state1	is(_, _, state1).
[*] -> state1	is(_, _, [*]) -> is(_, _, state1).
state1 -> state2	is(_, _, state1) -> is(_, _, state2).
state1 -> state2 : Trigger1	is(_, _, state1) & is“Trigger”(??, _, Trigger1) -> is(_, _, state2).
state1 -> state2 : [Guard である]	is(_, _, state1) & is“Guard”(??, _, Guard1) -> is(_, _, state2).
state1 -> state2 : / Effect1	is(_, _, state1) -> is“Effect”(??, _, Effect1) & (_, _, state2).
state1 -> [*]	is(_, _, state1) -> is(_, _, [*]).
コメントアウト (文頭に「'」)	コメントアウト部分はセミ形式に変換されず，文頭に「#」を付けて出力される．

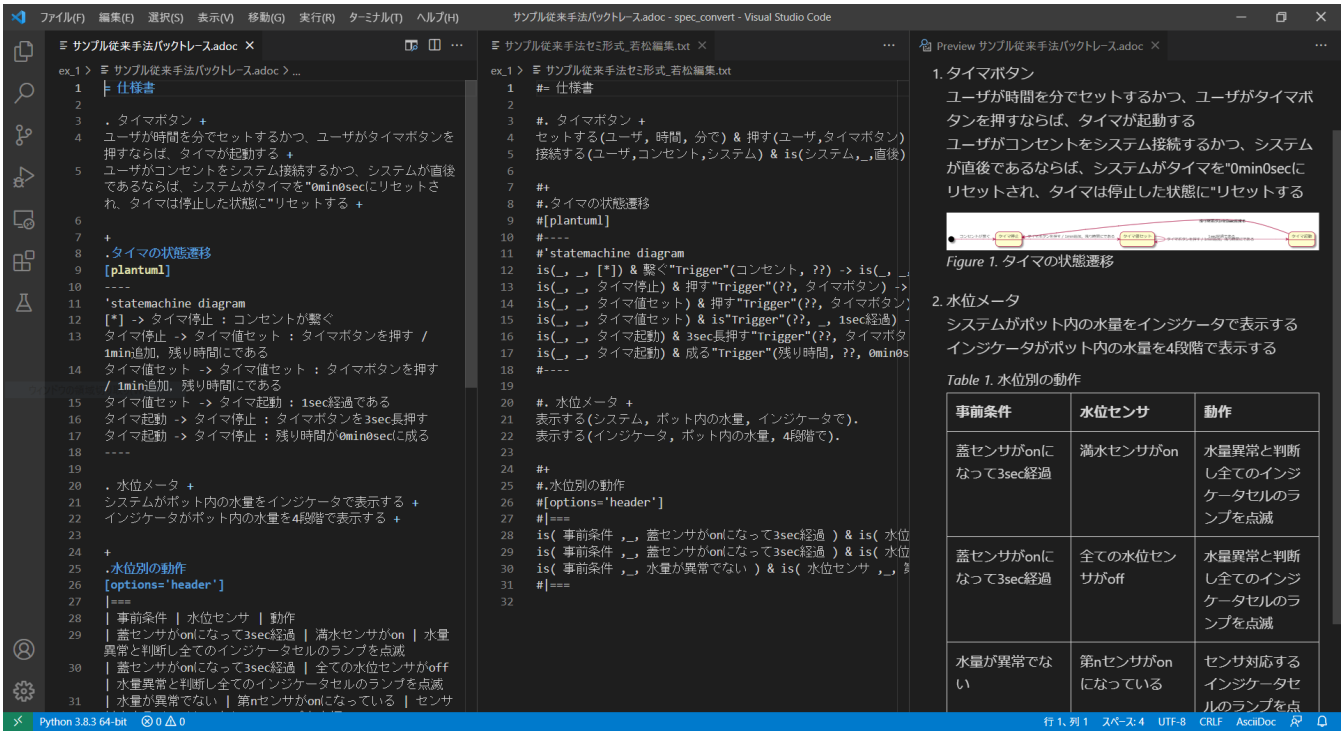

図 3 ツール構成

3.3 バックトレース

3.2 節に記載したルールで変換されたセミ形式を AsciiDoc 形式の仕様書へと変換する機能をバックトレースと呼ぶ.

(1) 自然言語

セミ形式で表現された自然言語に対し，表 6 のルールを適用する.

表 6 自然言語の逆変換ルール

変換要素	変換時の処理
セミ形式	「動作 (主体, 対象, 制約).」の場合,「‘‘主体’’ が ‘‘対象’’ を ‘‘制約’’‘‘動作’’」 と変換される. また, 「→」 は「ならば」, 「&」 は「かつ」, 「|」 は「または」に変換する.
コメントアウト (##)	コメントアウト部分は変換されず，文頭に「//」を付けて出力される.
制御文字 (#)	制御文字部分は変換されず，文頭の「#」を外して出力される.

(2) 表

セミ形式で表現された表記述に対し，表 7 のルールを適用する.

(3) UML シーケンス図

セミ形式で表現された UML シーケンス図記述に対し，表 8 のルールを適用する.

(4) UML ステートマシン図

セミ形式で表現された UML ステートマシン図記述に対し，表 9 のルールを適用する.

表 7 表の逆変換ルール

変換要素	変換時の処理
is(条件, _, 条件 A) -> is(期待値, _, 期待値 B).	|=== | 条件 | 期待値 | 条件 A| 期待値 B |===
コメントアウト (##)	コメントアウト部分は変換されず，文頭に「//」を付けて出力される.
制御文字 (#)	制御文字部分は変換されず，文頭の「#」を外して出力される.

表 8 シーケンス図の逆変換ルール

変換要素	変換時の処理
Request(A,B).	A から B へのメッセージとしてシーケンス図に変換される.
Response(B,A).	以前の文に「Request(A,B).」がある場合は，その文に対するリプライメッセージとしてシーケンス図に変換される.

4. 評価実験

4.1 実験概要

ツール評価のため，以下の 2 つの評価実験を実施した.

(1) 評価実験 1

学生 2 名と実務者 2 名による操作性評価及びテストケースを正しく導出できるかの確認試験

(2) 評価実験 2

実務者 3 名による支援ツールとしての有用性試験

評価実験 1 の結果，ツールを用いることで仕様書記述内容が妥当なテストケースに自動変換されること，操作性の課

表 9 ステートマシン図の逆変換ルール

変換要素	変換時の処理
is(_, _,state).	1 つの状態として plantuml に変換される.
is(_, _, [*]) -> is(_, _, state).	開始ノードから state への状態遷移としてステートマシン図に変換される.
is(_, _,state1) -> is(_, _,state2).	state1 から state2 への状態遷移としてステートマシン図に変換される.
is(_, _, state1) & is“Trigger”(??, _, Trigger1) -> is(_, _, state2).	Trigger1 をトリガとする state1 から state2 への状態遷移としてステートマシン図に変換される.
is(_, _, state1) & is“Guard”(??, _, Guard1) -> is(_, _, state2).	Guard1 をガードとする state1 から state2 への状態遷移としてステートマシン図に変換される.
is(_, _, state1) -> is“Effect”(??, _, Effect1) & is(_, _, state2).	Effect1 をエフェクトとする state1 から state2 への状態遷移としてステートマシン図に変換される.
is(_, _, state) -> is(_, _, [*]).	state から終了ノードへの状態遷移として plantuml に変換される.

題 (ツールでの形式変換処理時間など) を確認し，これらを改善したのちに評価実験 2 を実施した．評価実験 2 は，現職のエンジニアの方にツールを利用してテストケース設計作業を実施していただくことで行った．今回は実務者 3 名にツールを適用し，作業時間の変化・設計したテストケースの比較により評価を行った．入力する仕様書は “SysMLによる組込みシステムモデリング” [12] から一部抜粋・加筆したものを用いた．実験は Zoom を用いたオンライン形式で行い，各作業者ごとにブレイクアウトルームに分かれ，単独で作業を行った．筆者らは実験を観察し，被験者の疑問点に直ちに対応できる環境を用意した．実験は以下の手順で実施した．

(1) 実験概要の説明
(2) 従来手法の実施
(3) 提案手法の実施
(4) アンケートの実施

4.1.1 従来手法

従来手法では，Word,Excel を用いた作業を実施した．Word ファイルで用意された仕様書から，Excel ファイルで用意されたテンプレートに従ってテストケースを設計する作業を実施した．テストケースのテンプレートはデシジョンテーブルをベースに設計したものを用いた．

作業時間・作業項目については特に指定せず，被験者の任意の作業を行っていただいた．

4.1.2 提案手法

提案手法では，本研究にて開発したツールを用いた作業を実施した．AsciiDoc 形式で用意された仕様書からツールを用いて形式記述へと変換し，`Splosat` によりデシジョンテーブルを生成する作業を実施した．

4.2 実験結果

従来手法・提案手法それぞれ被験者一名の結果を代表して記載する．被験者が実験で行う作業を「読む」，「書く」，「考える」という 3 つのカテゴリに割り当てた．手法間での作業時間の変化を観察することで提案手法の効果について考察する．従来手法・提案手法における設計時間を図 4，手法毎に設計者作業及び PC 作業に分解したものを図 5 に示す．

総合的な時間としては提案手法のほうが増加しているが，設計者が行った作業とコンピュータにより作業した時間を比較した結果，設計者が行った作業時間が減少し，コンピュータに依存した作業が増加していることがわかる．また，テストケース設計の本質的な作業に費やす時間が増加していることがわかる．具体的には，「機能のテストすべき内容が抽出されているか確認する」，「テスト項目が網羅されているか確認する」作業である．その他，従来手法では「テストすべき項目を精査する」や「追加・変更部分を抽出する」部分を実験者が行っていたが，ツールによりテストケースを自動生成することによりその時間が削減され，提案手法ではコンピュータによりその作業が行われていることがわかる．この時間は実験者がツールを実行し，フォワードトレース・バックトレースを実行した際にかかった時間を計上している．また，実験者が設計したテストケースのデシジョンテーブル数及びケース数は表 10 のようになった．

表 10 設計されたデシジョンテーブルのテーブル・ケース数

	従来手法		提案手法	
	テーブル	ケース	テーブル	ケース
被験者 A	8	8	21	35
被験者 B	5	14	20	45
被験者 C	6	49	19	45

提案手法では，テーブル数・ケース数ともに従来手法に比べ増加する結果となった．図 6 からもわかるように，従来手法では比較的規模の大きなテストケースが少数設計され，提案手法では細かく分解されたテストケースが複数設計されていることがわかる．図 6 では従来手法・提案手法の双方で設計されたテストケースを抜粋しているが，それぞれテストとしては同じ内容のもの箇所を抜粋している．従来手法では 1 テーブルに対し多くのケースが抽出されているが，提案手法では細かく分けられたテーブルが複数設計され，確認する内容が明確に理解しやすくなっていることがわかる．また，それぞれの編集後の仕様書に複数の手法で形態素解析を行うことで情報量の可視化を行った．

従来手法では動詞などが増加し，総検出数では約 20%ほど増加している．一方で提案手法では従来手法に比較し検出結果に大幅な変化は認められなかった．

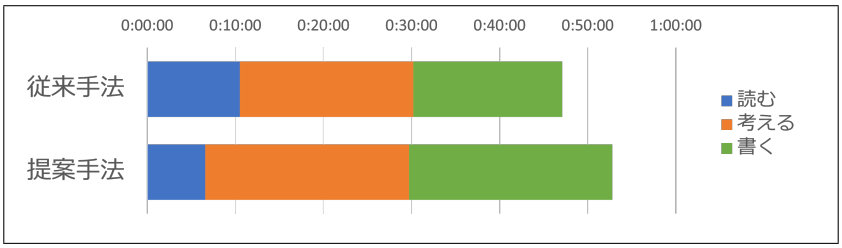

図 4 時間計測結果

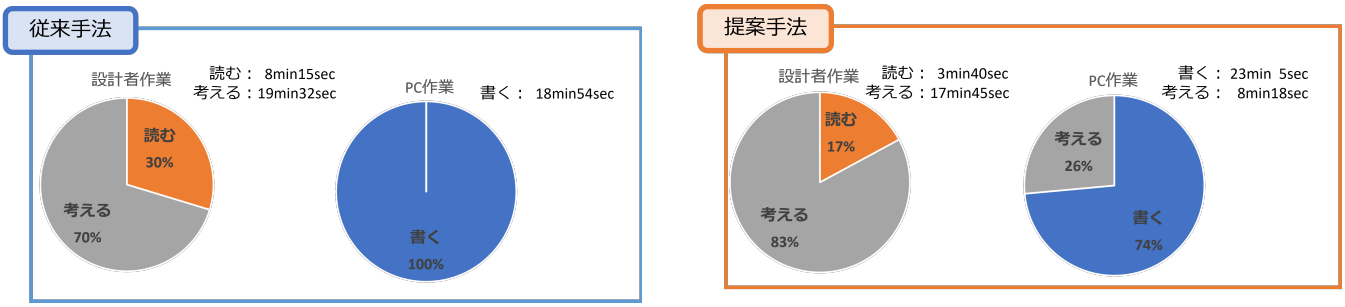

図 5 作業別割合

表 11 形態素解析の結果

	従来手法		提案手法		提案手法セミ形式	
	編集前	編集後	編集前	編集後	編集前	編集後
形容詞	4	9	2	1	1	1
副詞	2	3	1	1	1	1
名詞	58	62	48	50	48	51
動詞	13	15	11	11	11	10
検出単語数	105	120	82	82	80	82

5. 考察

(1) 時間に関する考察作業時間において，作業の総時間としては増加したが，テスト設計者が行う作業が減少し，コンピュータにおいて行う作業が増加している．また，テストケース設計のより本質的な作業をテスト設計者が行い，それ以外の作業をコンピュータに代替できた結果となった．図 5 に示すように，従来手法では設計者により行われていた「考える」作業の一部が提案手法の PC 作業に変化している．提案手法における PC 作業の「考える」時間は，従来手法における以下のような作業が計上されている．

- 仕様書からの「前件 → 後件 」の抽出
- デシジョンテーブルの真理値設計
- 編集による追加・編集項目の抽出

以上のような本来設計者が考える必要のない機械的な作業がツールの適用により自動化されている．今後ツールアルゴリズムや利便性の向上により，提案手法における「考える」カテゴリの時間はさらに短縮を見込むことができる．

「考える」時間が提案手法においては一部機械化できているにも関わらず，提案手法の設計者における「考える」時間は従来手法と同等に確保されている．ツールによる設計の本質的な作業以外が機械化できたことにより，テストケース設計作業に本来必要な「テスト網羅」や「漏れ・誤りの確認」といった作業に集中することができていることを表していると考える．

また，従来手法では 8 分弱必要だった「読む」カテゴリについては 3 分強程度の時間が計上されている．この時間は仕様書・テストケースのレビュー時間等を計測している．セミ形式の??による情報不足箇所の明示や論理記号による仕様の直感的な理解，また仕様書と対応したテストケース記述により，レビューの際の認知負荷の低減が主な要因だと考える．また，実験中の各作業の数を蓄積したところ，従来手法では 47 作業ほどあったものの，提案手法では 24 作業と約半数に減少している．認知負荷の低減により仕様書・テストケース間の対応を直感的に取ることができるため，逐次確認することなく作業できるようになっている．

「書く」時間は従来手法に比べ提案手法では増加している．提案手法ではツールの自動化によりデシジョンテーブルの条件・動作・真理値を記述する必要がないため，その時間が短縮されているが，テストケース設計後の編集箇所の元仕様書への反映に非常に時間がかかる結果となった．今回ツールにバックトレースという機能を実装していたが，被験者によっては表層的な言葉や文書としての体裁にこだわってしまう傾向にあり，今後のツール改善の余地ある結果となった．

(2) 設計されたテストケースに関する考察

設計したデシジョンテーブル数とテストケース数には大きな違いがみられた．従来手法では比較的規模の大きなテストケースが少数設計される傾向にあり，提案手法では細かく分解されたテストケースが複数設計さ

ケース番号		0	1	2	3	4	5	6
動作	硬貨を投入	T	T	T	T	T	T	F
条件	投入された硬貨は偽貨	F	F	F	F	-	F	-
条件	投入された硬貨は1円硬貨	F	F	F	F	-	F	-
条件	投入された硬貨は5円硬貨	F	F	F	F	-	F	-
条件	投入された硬貨は500円硬貨	T	F	F	F	-	-	-
条件	投入された硬貨は100円硬貨	F	T	F	F	-	-	-
条件	投入された硬貨は50円硬貨	F	F	T	F	-	-	-
条件	投入された硬貨は10円硬貨	F	F	F	T	-	-	-
条件	硬貨の有効判断終了	T	T	T	T	F	T	-
条件	カウント終了	T	T	T	T	-	F	-
動作	500円釣銭チューブに保存される	T	F	F	F	F	F	F
動作	100円釣銭チューブに保存される	F	T	F	F	F	F	F
動作	50円釣銭チューブに保存される	F	F	T	F	F	F	F
動作	10円釣銭チューブに保存される	F	F	F	T	F	F	F
動作	現物エスクロに保存される	F	F	F	F	T	T	F

	0	1
投入する(ユーザ,硬貨)	F	T
保存する(システム,投入された硬貨,現物エスクロ機構に)	-	T

	0	1
is(投入された硬貨,_,500円)	F	T
is(保存場所,_,500円釣銭チューブ)	-	T

	0	1
is(投入された硬貨,_,100円)	F	T
is(保存場所,_,100円釣銭チューブ)	-	T

	0	1
is(投入された硬貨,_,50円)	F	T
is(保存場所,_,50円釣銭チューブ)	-	T

	0	1
is(投入された硬貨,_,10円)	F	T
is(保存場所,_,10円釣銭チューブ)	-	T

左 従来手法　　**右** 提案手法

図 6　設計されたテストケースの一部抜粋

れる傾向にあることがわかった．従来手法において被験者は仕様書を熟読し，そこからテストケースを設計するが，仕様書中の表や UML 図等の情報が整理されている箇所をベースにテストケースを設計していた．一方，それより前に書かれている平文の説明などは設計中の補足情報として扱い，テストケースに現れる部分は少なかった．結果，テスト設計者の知識・常識に依存して整理された規模の大きなテストケースが設計されたと考える．

実際に筆者が以前実施した評価実験の中で，被験者が設計したテストケースに以下のようなものが見られた（実験に用いた仕様書は話題沸騰ポッド [13] から抜粋）．

ケース番号		6	7	8
条件	蓋が開けられている	T	T	T
条件	沸騰ボタンを押す	-	T	T
条件	沸騰状態である	-	T	F
動作	ヒータをoffする	T	T	T
動作	沸騰状態になる	-	T	F

図 7　被験者が設計したテストケース一部抜粋

図 7 の保温状態と沸騰状態の真理値において，保温状態の真理値 `False` が沸騰状態の真理値 `True` であるという暗黙的な読み替えを行っている．これは保温状態の否定は必ず沸騰状態であるというテスト設計者の暗黙的な読み替えにより発生していると考えられる．実際には沸騰状態, 保温状態の他にも取りうる状態があるのでこれは誤りであるといえる．他にも,`ON` 状態のテストケースの真理値 `False` が `OFF` 状態のテストケースの真理値の `True` であるといった読み替えが発生していた．仕様書には `True` の内容は仕様書中に記述されているが，その否定である `False` の記述，つまりある状態以外のとりうる状態については曖昧もしくは未定義になっていることも多く，これらはテスト設計者の仕様書の理解, 読み込みの充実度によってテストケースへの抽出結果にバラつきがでるものだと考えられる．

一方で提案手法では仕様書の各文を形式記述に変換し，テストケースを自動生成するため，設計者の意図しない知識に依存した情報の省略等を防ぐことができ，設計者のみならずテストを実行するものにとってもレビューのしやすいテストケースになったと考えることができる．

(3) テストケースに含まれる情報量に関する考察

情報量に関しては従来手法で増加・提案手法で変化がない数値となっているが，従来手法では不足している仕様の追加，提案手法においては表記揺れしていた用語の統一・重複した仕様のテスト項目の統合によるものであり，仕様の漏れ・誤りに起因するものではなかった．

(4) 従来手法と提案手法の差

傾向として，従来手法においては設計者の知識に依存したテストケースが設計され，各被験者でレビュー視点が異なり，追加されたテストケースも内容にバラつきがあるが，提案手法では設計者各自の知識に依存しないテストケースが設計されている．従来手法におけるテスト項目の抽出は機能毎・入力するパラメータ毎等，テストの抽出観点がバラバラであることに対し，提案手法では仕様書に応じた構造を持ったテストケースが設計者間で大きな差がなく設計されていた．

従来手法と提案手法の差について，以下の考察を行った．

- 従来手法
  機能ごとにテストケースが設計される
- 提案手法
  構造化されたテストケースが設計される

従来手法について，被験者の設計の様子から，表や UML 図で表された機能を中心にテストケース設計を行い，その他の情報は補足として活用していた．そのためテスト設計

者は文頭から仕様書を読み始め，前提や説明など様々な情報を整理した後，最終的に図・表でまとめられた情報のまとまりをテストケースとして設計している．具体的には，同じ仕様書を用いて実験を行った表 10 に示す被験者 B の従来手法と被験者 A・C の提案手法のテーブル数に現れている．被験者 A・C の提案手法ではテーブル数はそれぞれ 21，19 となっている．それに対し被験者 A の従来手法はテーブル数 5 となっている．このテストケースを見ると，提案手法のツールが図・表からテストケースを生成している箇所が，従来手法の被験者が抽出したケースとほぼ一致している．提案手法はそれに比べ，前提やその他の説明もテストケースとして現れているが，従来手法ではその情報は欠損している．人間の認知では最後に見たものの記憶が強く残ることもあり，仕様書の冒頭から精読し，説明文や事前知識を理解した後，情報の整理された図・表をそのままテストケースとして抽出したと考えられる．テスト設計者には仕様書に記述された説明文や事前知識は設計根拠として整理された状態だが，テストケースにはそれが欠損，もしくは整理された状態でしか記述されない．結果として，テスト設計者が設計の中で何らかの意図を持って設計したものがテストケースになっているが，その意図が大きなケースの中の一部に埋もれてしまい，またテスト設計者に依存した形式で整理されているためレビュー・テスト実行の際にその意図に気付ける可能性は限りなく低い．

一方提案手法では仕様書の最序盤に書かれていた前提や機能の概要，説明文などもテストケースとして自動抽出される．これにより，仕様書記述中の機能概要等をテストケースのキャプションとして理解することができ，以降のテストケースの内容をより理解しやすくなっていると考えられる．

この差は設計者のメンタルモデルの差異により生まれるものだと考える．本ツールの適用により，仕様書・テストケースに対し，ツールから生成されたものを編集することになる．整理された仕様書から自動生成されたテストケースは，メンタルモデルの介在が最小限となるため，必然的にその構造が引き継がれた構造のテストケースとなる．そのためツールを適用した際には設計者毎に異なる知識・メンタルモデルに依存することなく安定したテストケース設計が可能となる．

更に，本ツールを用いることで仕様書とテストケースが対応していることで，レビュー性が格段に向上していると考える．ツールでの編集中，被験者は仕様書中に出現する用語の表記揺れを修正しようとする傾向にあり，これにより異なるデシジョンテーブルにまたがって出現するテストケースに関しても用語が統一されていることでそれぞれのトレーサビリティに確保にもつながっていると考える．テスト実施者はテスト設計者が知識依存で設計したテストケースを瞬時に理解することは困難であり，本ツールを適用することによりレビュー性・理解性が向上することでテストケース漏れや試験不備などの課題の解消も期待できると考える．

6. おわりに

本稿では，テストケース設計の自動化及び修正支援についてアプローチした．本ツールを用いることにより，設計者はテストケース設計のより本質的な作業に注力でき，その他の作業をコンピュータに代替することで作業の効率化を実現した．また，ツールを用いることにより成果物の知識体系化，仕様書・テストケースの再利用性の向上も確認できた．今後はツールの操作性の向上及び評価実験を実施することで，システム開発現場でより有効的に活用できるツールへと精緻化したいと考える．

参考文献

[1] 山田茂, 冨高功介. ソフトウェア信頼性に影響を及ぼす品質工学的アプローチに基づく人的要因分析と信頼性予測. REAJ 誌,Vol.27,No.7,2005
[2] 中西恒夫, 久住憲嗣, 福田晃. 派生開発からプロダクトライン開発への漸次的移行プロセス XDDP4SPL におけるコア資産管理手法. 情報学会論文誌, Vol.2013-SLDM-160 No.9, Vol.2013-EMB-28 No.9, 2013
[3] M.R.Girardi, B.Ibrahim. A Software Reuse System Based on Natural Language Specifications. Computing and Information, 1993. Proceedings ICCI '93., Fifth International Conference.
[4] Carlos Paredes, José Luiz Fiadeiro. Reuse of Requirements and Specifications: A Formal Framework. ACM SIGSOFT Software Engineering Notes 20(SI):263-266, August 1995
[5] AsciiDoc. https://asciidoc.org/
[6] PlantUML. https://plantuml.com/ja/
[7] 増田聡, 松尾谷徹, 津田和彦. テストケース作成自動化のための意味役割付与方法. コンピュータソフトウェア, Vol. 34, No. 2, pp. 16–27, 2017.
[8] 坂本一憲 , 土肥拓生. テストカバレッジを用いた新しいテスト駆動開発プロセスの提案. 情報学会論文誌, Vol.2015-SE-187 No.34, 2015
[9] Yusuke Aoyama, Noriyuki Kushiro, and Takeru Kuroiwa. Test case generation algorithms and tools for specifications in natural language. In 2020 IEEE International Conference on Consumer Electronics (ICCE). IEEE, jan 2020.
[10] 青山裕介, 黒岩丈瑠, 久代紀之. テストケース生成のためのシステム仕様書の論理記述変換アルゴリズム. 情報処理学会論文誌, Vol. 61, No. 3, pp. 521–534, mar 2020.
[11] 黒岩丈瑠, 青山裕介, 久代紀之. エミュレーション技術の活用による IoT システムのテスト効率化. 電気学会論文誌, Vol. 140, No. 1, pp. 113–121, 2020.
[12] 株式会社テクノロジックアート：SysML による組込みシステムモデリング. 株式会社技術評論社 (2011).
[13] 組込みソフトウェア管理者・技術者育成研究会（SESSAME）. 組込みシステム教育教材　話題沸騰ポット GOMA-1015 型 要求仕様書