組込みシステム開発のための
ドキュメント・ソースコード分析支援

山本　椋太

組込みシステム開発のためのドキュメント・ソースコード分析支援

要旨

近年，組込みシステムの大規模化・複雑化が進行しており，開発効率の向上が求められている．また，組込みシステムのライフサイクルは短いため，短納期化の要求があり，開発の効率化に対する要求がある．開発の効率化の手法として，手戻りの抑制や，ソフトウェアの再利用があげられる．これらの開発の効率化のためには，ドキュメントの低品質化や再利用が困難なソフトウェアが問題となる．そこで，これらの問題によってについて述べる．

まず，ドキュメントの低品質化について述べる．開発者がそれらの文書を自然言語によって記述することで，曖昧表現や誤り表現を生じる可能性がある．自然言語によって記述された文書に存在する曖昧表現や誤り表現は，システム開発の手戻りの要因となることがあり，修正されることが望ましい．すなわち，要求仕様書はシステム開発の最序盤で作成される文書であり，この時点での曖昧さや誤りは後の工程に対して悪影響を及ぼし，手戻りの原因となりうる．また，組込みシステムの要求仕様書は，エンタープライズシステムの要求仕様書と比べて，詳細な情報が書かれていることが多く，状態遷移記述や設計情報などが存在することが多い．この状態遷移記述とは，直接ソースコードと対応するかは不明だが，広範な視点で対象システムやサブシステムの状態を表している．エンタープライズシステムの要求仕様書中の誤り・曖昧表現の抽出や調査の研究は存在するが，組込みシステムの要求仕様書における，誤り・曖昧表現の種別や出現数が明らかではない．以上より，本論文では組込みシステムの要求仕様書について着目する．実際に，要求仕様書から誤り・曖昧表現を抽出するためには，抽出するべき誤り・曖昧表現が明らかになっている必要がある．

次いで，ソフトウェアの再利用について，開発現場では，ソースコードのレガシー化が生じていることがある．ここでレガシー化したソースコード（レガシーコード）とは，開発者が不在となり，かつ開発関連文書の保守が不十分であるような状況となったとき，内容を理解することが困難となったソースコードである．レガシーコードを用いて開発する開発者は，ソースコードの内容の詳細を把握していないが，入出力情報やテストケースなどは残っていることを仮定しており，それらからレガシーコードをブラックボックスとして扱うことはできる．そのため，レガシーコードを用いる場合には，アルゴリズムを変更するのではなく，分岐を

増やすことで対応することがある．その結果として，ソースコードの複雑度が向上することとなり，さらに保守性の低下につながる可能性がある．

以上より，本研究では組込みシステム開発の効率化に向け，次の 4 つの研究を行う．(**研究 I**) 要求仕様書中の誤り・曖昧表現の調査，および (**研究 II**) 要求仕様書を対象とした状態遷移記述抽出ツールの定量的評価，(**研究 III**) ソースコードとの対応が取りやすいモデルである細粒度状態遷移表（Micro State Transition Table, MSTT）の提案，(**研究 IV**) C 言語ソースコードから MSTT の抽出を支援するツールの提案，以上の 4 つの研究を実施した．

研究 I と **研究 II** は，組込みシステムに関する要求仕様書の分析である．まず**研究 I** として，要求仕様書の誤り・曖昧表現の実証的調査である．要求仕様書中に存在する誤り・曖昧表現を含む問題を定義し，組込みシステムについての実際の要求仕様書中に存在する問題の分析を試みた．組込みシステムの要求仕様書は，エンタープライズシステムの要求仕様書と比較して，クライアントもドメイン知識を有していることが多く，設計情報が含まれていることが多い．そのため，後述する**研究 II** に関連して，記述対象システムの状態遷移に関する記述が存在することもある．調査の結果，ドメイン知識への依存が弱い問題については，ドメインの専門家でなくとも検出することができた．そして，最も検出数が多い「構文の曖昧表現と一貫性の問題」を 6 つに細分化し，細分化された問題の検出手法として，精度を向上する手法と，再現率を向上する手法の 2 つを検討した．これらの手法を実装して評価した結果，構文上の曖昧表現の検出数が最多であった．また，手法の適用結果から，字句・構文解析によって検出可能な問題は，高い精度・再現率で検出可能だとわかった．

研究 II は，先行研究において開発された自然言語で記述された要求仕様書から状態遷移表を抽出支援するツールの機能の内，節の分類機能の評価を行い，その結果から誤り・曖昧表現の検討を行った．このツールは，インタラクティブな操作に基づき，ツールの入力制約を満たした要求仕様書から，状態遷移表の抽出を支援する．入力した要求仕様書は，**研究 I** でも用いた組込みシステムの要求仕様書であり，上述の通り，エンタープライズシステムと比べて詳細な記載がある事が多い．そのため，状態遷移に関する記述も存在しており，先行研究におけるツールは，組込みシステムの要求仕様書が持つこの特徴に注目している．状態遷移表を抽出する過程において，このツールは独自定義の解析木を抽出し，抽出された解析木を用いて要求仕様書中の各文中の節に対して，**条件節**，**処理節**および**定義節**のいずれかの分類を与える．これらの分類精度を調査し，このツールの分類能力を確認

するとともに，分類できない節が存在しないかを確認することで，誤り・曖昧表現の検出につながらないかを確認した．その結果，分類精度は，F 値で評価した場合に全体として 0.9 を超える結果となった．しかし，インタラクティブな操作において，処理節か定義節かを分類することが困難な節が存在することがわかった．解釈を機械的に決定することができないために，インタラクティブな操作を必要とするため，インタラクティブな操作を必要とする箇所は曖昧表現である可能性があると考えた．

研究 III では，ソースコードとの対応が取りやすいモデルである MSTT の提案を行う．本研究では，MSTT の定義を行い，ケーススタディを実施する．本研究の関連研究にあるように関数呼び出しをイベントや状態値として扱うのではなく，MSTT は，条件分岐文をイベントや状態値として扱う．MSTT は，組込みシステムの状態遷移表を表現することを目的としており，組込みシステム特有の非同期なイベントが出現することを考慮し，MSTT の上の行から下の行に向かってイベントを評価するように表現している．また，関連研究では状態分割ポイントや状態変数を指定することで，組込みシステムのソースコードから状態遷移モデルを作成可能な手法があるが，状態分割ポイントを指定するためには，ソースコード全体を読み，状態分割ポイントを判断する必要がある．イベントや状態値判定が，従来の状態遷移表と同様に一意なもののみとするためには，状態分割ポイントを指定するか，または元のソースコードを書き換える必要がある．これに対して，本研究における MSTT は，ソースコードを変更せず，かつ状態分割ポイントを判断する必要がない．従来の関数呼び出し単位の状態遷移表と比べて大規模になることが多いが，ソースコードとの対応はわかりやすく，詳細な処理の記述を可能としている．表の規模は大規模なものになるが，ソースコード上の 1 つのパスの条件をまとめてイベントを独立させた表と比べて 1 つのイベントの条件式の項数は低減しうる．ケーススタディでは名古屋大学の学生 4 名が，ソースコードから MSTT の抽出を試みた．このとき，ソースコードから MSTT を抽出するプロセスや MSTT そのものから，ソースコードへの理解を深めることができた．しかし，手作業による MSTT の抽出は難度が高くかつ時間がかかる．すべての参加者は，手作業において 1 度では正確に MSTT を抽出できず，抽出結果を修正する必要があった．これらの結果から，MSTT の自動抽出または抽出支援を行うツールが必要であると考えた．

そこで，**研究 IV** では，MSTT の抽出支援ツール RExSTM の開発に取り組んだ．RExSTM は，C 言語ソースコードを入力としてソースコード解析を行い，状態遷

移表抽出を支援するツールである．ユーザは，RExSTM を用いた MSTT の抽出過程において，ソースコード中の状態を表す変数（状態変数）をツールに与える必要がある．状態変数候補の抽出は自動化されているが，MSTT として有効な表現となるような変数は，ユーザが選択することとしている．RExSTM を用いた MSTT の抽出と，手作業による作成時間を比較すると，ツールの有無において有意差があると認められ，RExSTM による抽出支援の有効性があると結論づけた．

以上 4 つの研究により，組込みシステム開発を効率化するためのソースコードや要求仕様書の解析に基づく技術的負債を発見する支援，および識別を行った．

目 次

図 目 次

表 目 次

第1章　序論

1.1　研究の背景

近年の組込みシステム開発においては，あらゆるものがネットワークに接続される IoT（Internet of Things）が注目されている．IoT 端末からデータを収集してクラウドコンピュータ上で処理するなどの開発が進行している [1]．IoT の普及に伴い，IoT 端末の数は，2020 年には 300 億台になるとも予想されている [1]．IoT 端末には，自動車や，監視カメラ，FA（Factory Automation）などの組込みシステムも含まれており，多種多様な組込みシステムが含まれる．また，IoT 活用のためには，必然的にネットワークを利用することになるため，従来，ネットワークには接続されていなかった組込みシステムは，規模が増大する可能性があり，機能増加に伴う複雑化も進行しうる．

車載制御においても，パワートレインアプリケーション（以降，パワトレアプリと呼ぶ）などの高機能化に伴い，マルチコア環境での開発が行われるなど，大規模化・複雑化が進行している [2]．加えて，様々なパワトレアプリ用のマイコンが存在しており，車種によって使用するマイコンを使い分けることがある．そのため，利用できるコア数やメモリ構成がパワトレアプリごとに異なる場合がある．コア数やメモリ構成が変化することで，処理の割り当てなどの構成が変化するため，実装や設計の難度が上昇する．パワトレアプリは非常に大規模であるため，異なるマルチコアアーキテクチャ向けにフルスクラッチで書くことは現実的ではない．

組込みシステム開発において，上記のような問題があり，かつ，組込みシステムは現実世界に作用することが多いため，高いディペンダビリティが要求されるものもある [3, 4]．これらの状況下において，組込みシステムの大規模化が進行しているが，従来より，組込みシステムのライフサイクルは短く，短納期化の要求がある [3]．また，組込みシステム開発においては，エンタープライズシステムの開発と同様，成果物としてドキュメントやソースコードが存在する．これらの品質が低い場合，開発効率の低下につながる場合が多い．短納期化の現状に対応するためには，ソフトウェアの再利用や開発の手戻りの抑制が要求される．

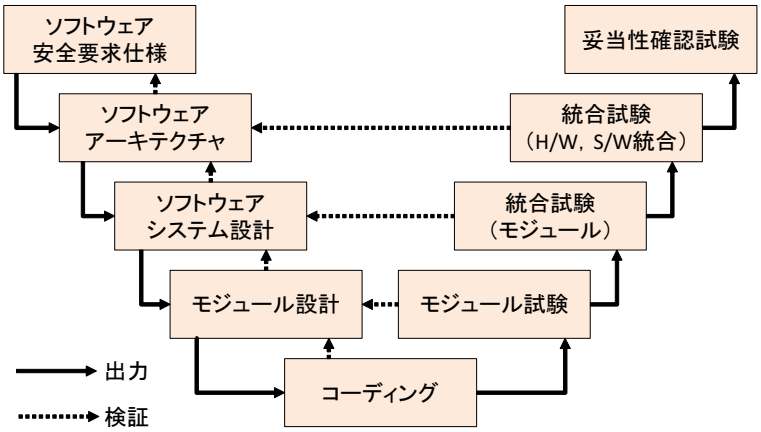

図 1.1: IEC 61508 に基づくソフトウェアライフサイクル

また，機能安全が要求される組込みシステム開発に対しては，図 1.1 に示すようなソフトウェアライフサイクルが定義されている [5, 6]．本論文では，このうち，ソフトウェア安全要求仕様を策定する際の要求仕様書の低品質化と，再利用にあたってのコーディングからモジュール設計への検証プロセスを支援する目的でのレガシーコードのモデル化を検討する．

以上を解決するために，本論文では以下の 2 つの研究課題に注目する．

- ドキュメントの低品質化
- ソースコードのレガシー化

まず，ドキュメントの低品質化について説明する．開発時には，要求仕様書や設計書などのドキュメントが作成される．これらのドキュメントは通常自然言語で書かれるため，属人性や，曖昧さを持つことが多い．加えて，大規模なシステムである場合には，1 人でシステム全体のドキュメントを作成せず，分担してドキュメントを作成した上で，最後に結合を行うことが考えられる．この時点で，メンバ間で用語の統一などが取られていれば問題は低減されている可能性があるが，専門用語の揺らぎなどの，文章の一貫性がない状況（inconsistency）や，個人のスキルによって，ドメイン知識の理解度が異なる場合に，説明を省略する状況（imcompleteness），および,「大きい」「小さい」など，定量的ではない表現などが用いられている状況（ambiguiousness）が考えられる．これらの問題は，必ずしもその後の開発プロセスに悪影響を与えると断定できないが，上記の結果，システム開発において漏れや誤りを生じ，手戻りにつながる場合がある．他に，保守活動が非効率的になることも考えられる．例えば,「組み込み」と「組込み」などの語句の揺らぎが存在し，

その語句の周辺についてドキュメントを通して修正が必要な場合を考える．修正作業にあっては，例えば，開発者はエディタやワードプロセッサの文字列検索機能を用いて該当箇所を探索するが，語句の揺らぎによって，文字列検索機能を用いても見落としを生じる可能性がある．以上のように，ソースコードのレガシー化や，ドキュメントの低品質化は効率的な開発を阻害する要因となりうる．

特に，組込みシステムでは第 2 章においても述べる通り，エンタープライズシステムの要求仕様書と比べて，クライアントもドメインの専門家である場合が多く，設計情報や状態遷移記述など，より詳細な記述が多い．このような，ドメイン知識が含まれており，設計情報も含む組込みシステムの要求仕様書において，文書の品質を向上する手法を検討するためには，誤り・曖昧表現の問題の分類が必要である．

また，要求仕様書は大きいもので 2,000 ページ程度になることもあり [7]，開発者が自然言語の文書をすべて読み，その内容を理解することは，高コストかつ高難度である．自然言語文書を読みやすくするべく，先行研究において，要求仕様書から状態遷移モデルを抽出支援するツールが開発された [8, 9]．このツールは，対象の要求仕様書を組込みシステムの要求仕様書としており，上述の通り，状態遷移記述が要求仕様書中にあることを想定している．また，このツールは，入力された要求仕様書を節単位に分断し，節の分類を行って節に属性を付与する．このとき，ツールが節の分類を一意に行えない場合，ユーザインタラクションに基づいて節を分類する．その後，状態遷移記述を抽出し，状態遷移表を生成する．しかし，このツールは入力に対する制約が多い．たとえば,「箇条書きに接続詞が入ってはならない」などの制約があり，ある程度修正された，誤り・曖昧表現が低減された要求仕様書が対象となる．このように，誤り・曖昧表現の抽出およびその修正は，効率的な開発や効率化手法の適用のために，重要である．

中村らによって開発された要求仕様書から状態遷移モデルを抽出支援するツールは，経験則に基づく入力制約が定義されているのみであり [8, 9]，ツールの定量的な評価がなされていない．現状の経験則に基づく入力制約で，どの程度正しく節を分類できているか調査することは，入力の制約の不足を認識できる可能性がある．入力の制約は，誤り・曖昧表現の修正項目に繋がる可能性があり，状態遷移表抽出にあっての中間情報である節の分類の誤りを発見できれば，入力に新たに付与するべき制約が発見される可能性がある．

次に，ソースコードのレガシー化について説明する．ある開発者が担当の組込みシステムの開発を進めており，そのシステムが完成した時点で，現開発者が担当

ではなくなった場合，新しい担当者は，ソースコードを引き継ぐこととなる．このとき，もしソースコードに関するドキュメントと，ソースコードの間で一貫性を保つことができておらず，元の担当者との対話が困難である状況の場合，新しい担当者はソースコードの理解が困難となる場合がある．対象システムの保守にあっては，ソースコードを正しく理解することができていなければ，ソースコードの修正が困難となり，入出力の形式を変更するのみの対応をするなどをすることで，修正ごとにソースコードの構造がさらに複雑化する可能性がある．また，再利用の際にも同様のことが起こる可能性が存在する．

加えて，組込みシステムにおけるレガシー化について説明する．組込みシステム開発では，実機を用いたテストが必要であり [10]，テストにかかるコストが大きい問題がある．そのため，すでにテストを通過したソースコードのアルゴリズムを変更せずに，分岐の追加などの修正で対応する場合が多い [11]．また，既存製品と類似の製品開発においては，既存製品のソースコードの部分的な変更によって再利用をすることが多い [12]．

以上のように，ソースコードのレガシー化によって，ソースコードをブラックボックスとして扱い，その結果として再利用や保守が困難となる場合がある．

1.2　論文の概要

本論文では，組込みシステムの効率的な開発のため，要求腫瘍書中の技術的負債の抽出や，技術的負債となったレガシーコードをモデル化し，可読性の向上を試みることが目的である．組込みシステムのソフトウェア開発における成果物は要求仕様書を始めとしたドキュメントや，ソースコードである．これらの成果物は，保守や再利用にあたって参照されるため，これらの成果物に技術的負債が存在することは，保守や再利用においての障害となると考えられる．そこで，本論文では，ドキュメントとして開発の最序盤で作成される要求仕様書と，ソースコードを解析対象とした．

各章の内容に関して詳細は後述するが，大別すると上述の解析について，2 章および 3 章は要求仕様書の解析，4 章および 5 章はレガシーコードの解析について述べる．これらの解析によって，組込みシステム開発における成果物の主要なものについて調査やモデル化を行った．要求仕様書に注目した理由として，要求仕様書は開発の最序盤で作成され，技術的負債が存在する場合に後工程への影響が大きい．また，組込みシステムの要求仕様書であるため，設計に近い情報まで記述

されているが，設計書に比べれば抽象度は高く，設計書に比べて一般的な問題の識別が可能だと考えた．

そこで，本論文では，前節で述べた研究課題を解決する必要があり，および上述の成果物をモデル化する必要があると認識している．そのため，本論文は，大きく以下の 4 つの研究からなる．

研究 I 要求仕様書中の誤り・曖昧表現の調査

研究 II 要求仕様書から状態遷移モデルを抽出支援するツールの節分類精度の調査

研究 III ソースコードとの対応が取りやすいモデルである細粒度状態遷移表（Micro State Transition Table, MSTT）の提案

研究 IV C 言語ソースコードから MSTT の抽出を支援するツールの提案

研究 I および**研究 II** は，組込みシステムに関する要求仕様書の分析である．要求仕様書の分析においては，2 つのアプローチによって分析を試みた．まず**研究 I** として，要求仕様書の誤り・曖昧表現の実証的調査である．上述した組込みシステムの要求仕様書の特徴を持つ要求仕様書の場合，要求仕様書には誤り・曖昧表現が存在するかを，今後，研究者がドキュメントの低品質化を防ぐ手法を検討するために明らかにする必要がある．そこで，要求仕様書中に存在する誤り・曖昧表現を含む問題を定義し，実際の要求仕様書中に存在する問題の分析を試みた．その結果，ドメイン知識への依存が弱い問題については，ドメインの専門家でなくとも検出することができた．そして，最も検出数が多い「構文の曖昧表現と一貫性の問題」を 6 つに細分化し，細分化された問題の検出手法として，精度を向上する手法と，再現率を向上する手法の 2 つを検討した．これらの手法を実装して評価した結果，構文上の曖昧の検出数が最多であった．また，手法の適用結果から，字句・構文解析によって検出可能な問題は，高い精度・再現率で検出可能だとわかった．

研究 II として，先行研究において開発された自然言語で記述された要求仕様書から状態遷移記述を抽出支援するツールの機能の内，節の分類機能の評価を行い，その結果から誤り・曖昧表現の検討を行った．上述の通り，中村らによって開発された要求仕様書から状態遷移モデルを抽出支援するツールは，経験則に基づく入力制約が定義されているのみであり [8, 9]，ツールの定量的な評価がなされていない．この入力制約は，文書の誤り・曖昧表現を除去する制約を含んでいる．そのため，現状の入力制約およびツールの能力によって分類できない節を調査することは，入力制約の不足である可能性があり，不足である入力制約は文書の誤り・曖昧

表現の識別に繋がる可能性がある．このツールは，インタラクティブな操作に基づき，ツールの入力制約を満たした要求仕様書から，状態遷移表の抽出を支援する．その過程において，このツールは独自定義の解析木を抽出し，抽出された解析木を用いて要求仕様書中の各文中の節に対して，**条件節**，**処理節**および**定義節**のいずれかの分類を与える．これらの分類精度を調査し，このツールの分類能力を確認するとともに，分類できない節が存在しないかを確認することで，誤り・曖昧表現の検出につながらないかを確認した．その結果，分類精度は，F 値で評価した場合に全体として 0.9 を超える結果となった．しかし，インタラクティブな操作において，処理節か定義節かを分類することが困難な節が存在することがわかった．解釈を機械的に決定することができないために，インタラクティブな操作を必要とするため，インタラクティブな操作を必要とする箇所は曖昧表現である可能性があると考えた．

研究 III および **研究 IV** は，上述した組込みシステム開発におけるレガシーコードの問題を低減するべく，レガシー化したソースコード（レガシーコード）からリバースエンジニアリングによって MSTT を抽出することで，その振る舞いを可視化するものである．レガシーコードが生じた場合，分岐の追加などの部分的な変更によって，構造が複雑化している可能性がある．そのため，これらの構造を可視化する狙いがある．**研究 III** では，MSTT の定義および MSTT を用いたケーススタディについて論じている．MSTT では，Walkinshaw[13] らの関数単位によって表されたイベント・状態値判定の粒度ではなく，条件分岐文にもとづいてイベント・状態値判定を行う．また，ケーススタディにより，MSTT を抽出する過程または MSTT からソースコードを理解することができるとわかったが，手作業によってソースコードから MSTT を作成することは，困難かつ時間がかかることもわかった．

MSTT を作成する難度を低減し，かつ MSTT を短時間で作成可能とするため，**研究 IV** では，MSTT の抽出支援ツール（RExSTM，以降支援ツールと呼ぶ）を開発した．RExSTM は半自動のツールであり，ソースコード中の状態を表す変数（状態変数）の指定のみ，ユーザが行う．状態変数候補は RExSTM が自動抽出できるが，その中からソースコードの状態を表している変数をユーザはただ 1 つ選択する．これによって，短時間かつ，容易に MSTT を抽出する支援を可能とできる．

1.3　論文の構成

本論文の構成を以下に示す．まず，第 2 章では，**研究 I** について述べ，組込みシステムの要求仕様書中の誤り・曖昧表現の調査について説明する．第 3 章では，**研究 II** について述べ，自然言語で記述された要求仕様書から状態遷移表を抽出支援するツールの機能の内，節の分類機能の評価を行い，その結果から誤り・曖昧表現の検討を行った結果についても述べる．第 4 章では，**研究 III** について述べ，ソースコードとの対応が取りやすいモデルである細粒度状態遷移表（Micro State Transition Table, MSTT）の提案と MSTT を用いたケーススタディについて述べる．第 5 章では，**研究 IV** について述べ，C 言語ソースコードから MSTT の抽出を支援するツールの提案と，そのツールを用いたケーススタディについて述べる．第 6 章では，本論文のまとめと今後の課題について説明する．

第2章　要求仕様書中の誤り・曖昧表現の実証的調査

2.1　要求仕様書に存在する技術的負債の背景と研究目的

システム開発においては，様々な文書が自然言語によって作成される [14, 15]．例えば，要求仕様書，システム設計書およびテスト設計書などがあげられる．特に要求仕様書は，ステークホルダが形式言語の専門家ではないことが多いため，自然言語によって作成されることが多い [14]．自然言語によって記述することで，認識の相違や，目視によって発見することが困難な抜けや漏れが生じる可能性がある．この問題を軽減するべく，ソフトウェア開発の上流工程において，状態遷移モデルの利用が検討されている [16]．しかし，要求仕様書は大きいもので 2,000 ページ程度になることもあり [7]，開発者が自然言語の文書をすべて読み，状態遷移モデルの作成に必要な要素をすべて正しく抽出することは，非常に高コストかつ高難度である．

加えて，要求仕様書はシステム開発の中でも上流プロセスで使用されるため，要求仕様書中に誤り表現や，読み手が解釈を誤る可能性のある曖昧表現記述が存在しうる．誤り・曖昧表現によって，手戻りの工数が増大することが考えられる．読み手が書き手の意図を正しく解釈できる要求仕様書は，ソフトウェア開発の手戻り抑制のために重要だが，システム開発者が要求仕様書などの技術文書を自然言語によって作成することで，文書中に誤り表現や曖昧表現を混入する可能性がある [15, 17]．誤り表現や曖昧表現の混入を防ぐために手作業による要求仕様書の確認を実施することは可能だが，作業コストの増大や指摘漏れのリスク発生が考えられる [14]．誤り表現や曖昧表現がなく，書き手の意図をステークホルダに正しく伝えるための品質として，正当性，非曖昧性，完全性など，9 つの項目が定義されている [18]．

誤り・曖昧表現について，ドメインや問題の知識に依存せずに一部の省略，不整合および曖昧な文を自動抽出できる可能性があると述べた研究がある [19]．文献

[19] では，要求仕様に対する検証（verification）と 妥当性の確認（validation）について述べている．しかし，これらの作業の結果に対して，要求仕様の品質が誤り・曖昧表現が悪影響を与えることがある．たとえば，要求仕様の品質属性が表す特性として，完全性，一貫性，および非曖昧性が挙げられる [19]．文献 [19] では，誤り・曖昧表現の自動抽出についても述べられており，ドメインへの依存が弱ければ自動抽出できる可能性があると述べられている．ドメインに強依存である場合の誤り・曖昧表現の抽出は，ドメイン毎に固有の抽出手法を定義する必要があり，普遍的な自動化手法を定義することが困難である [20]．対して，ドメインとの依存が弱い問題の抽出は，一般的な言語上の問題や要求工学における知見を用いて自動化手法を定義しやすい．そのため，ドメイン知識への依存度が低い問題であれば自動抽出手法を定義しやすい．これらの問題を抽出することは，特定ドメインや特定の問題に対して知識がない場合でも可能な場合があると述べている [19]．

我々は，文献 [19] に基づき，誤り・曖昧表現の自動抽出によって，検証や妥当性確認を支援できると考えている．検証および妥当性の確認作業は，上述した要求仕様の品質属性が表す特性は，誤り・曖昧表現によって低下しうる [19]．そのため，曖昧・誤り表現の自動抽出はこれらの特性の向上となりうる．本論文では，文献 [19] で述べられている，完全性，一貫性を損なう表現を**誤り表現**，および曖昧な文を**曖昧表現**と呼ぶ．ドメイン知識がなくとも自動抽出手法を定義できる可能性が高いことから，本論文では誤り・曖昧表現に注目する．

実際の要求仕様書中の表現上の問題の種別と頻度を実証的かつ定量的に調査した研究は少ない．被験者が検出可能な要求仕様書中の欠陥数を実証的に調査した研究が存在する [21]．この研究の中で，アドホックな方法，チェックリストベースの方法，および，提案手法であるシナリオベースの方法に基づいて，被験者が要求仕様書を読み，欠陥を指摘した．アドホックな方法は問題の分類を示し，チェックリストベースの方法は，アドホックな方法で用いた項目を詳細化したものを被験者に示す．また，シナリオベースの手法は，チェックリストベースの方法における項目よりも詳細であり，実際に被験者が行うべき手順も被験者に示される．提案手法では，確認する観点だけではなく，ある欠陥を発見するための手順も意識されている．文献 [21] の結論として，チェックリストベースの方法はアドホックな方法よりも効果的だと言えず，シナリオベースの方法はその他 2 つの方法よりも欠陥の抽出率が向上した．このことから，確認の項目をより詳細に定義すると，欠陥の抽出率が向上することがわかった．

文献 [21] を踏まえ，確認の項目をより詳細に定義している提案手法が欠陥の抽

出率が高い．すなわち，詳細な確認の項目により高品質な要求仕様書を作成できると考えられる．しかし，確認の項目を詳細に定義するためには，要求仕様書中の問題を明らかにする必要がある．このことから，実際に組込みシステムの要求仕様書を調査して現状を把握することによって，以下が可能となる．

- 実務者がレビューを行う際の指針とすることができる．
- 今後研究者が，ツールに基づく要求仕様書の修正支援を考案する上での指針とすることができる．

しかし，チェックリストベースの方法ではアドホックな方法と比べて不十分だと文献 [21] において述べられている．そのため，より効果的なシナリオベースの手法に基づくならば，問題の項目を明らかにするだけではなく，問題抽出のための作業手順の定義が必要である．作業手順を明らかにするためには，問題の種別を確認した上で各問題種別の抽出手法も検討する必要がある．

また，この問題を軽減するべく，自然言語によって記述された組込みシステムの要求仕様書から，後述するが，組込みシステムの要求仕様書は，エンタープライズシステムの要求仕様書と比較して，より具体的な記述を持つことが多い [22]．この理由は，エンタープライズシステムの開発と比べて多くの場合にハードウェア仕様の概要が決定しており，ソフトウェア開発のみでは最終工程まで実施できないことがある [23]．さらに組込みシステムは，アクチュエータによって物理世界への影響を与えることが多い [22]．

本研究では，日本語で記述され，また実際に企業で使用されている仕様書を用いて，要求仕様書中の曖昧・誤り表現の調査，および自動検出可能な問題の調査を行う．

自動検出手法を検討することにより，開発者による問題の見落としを低減し，客観的に実施可能な手法を定義できる可能性がある．対象とする要求仕様書は，レビューが加えられた比較的高品質なものである．本論文では，この要求仕様書中の問題の出現頻度を調査し，より高品質な要求仕様書の作成に向けて注意するべき問題の種別を実証的に明らかにする．ここで，要求仕様書の曖昧・誤り表現には分類があると考えられる [24, 25]．実際に要求仕様書中の問題分類作業を行い，その後自動抽出手法の検討を行った．本研究における目的 1 を達成するべく，以下のリサーチクエスチョン (**RQ**) を定義した．

RQ4-1 どの分類の問題が最も高頻度で存在するか．

RQ4-2 どの問題の分類を自動検出可能か.

上記 2 つの **RQ** について調査するべく，予備実験とツールを用いた実験を行った．予備実験では，この要求仕様書を用いて，作業者が要求仕様書中の問題だと判断した箇所を検出した．問題の手作業による検出にあっては，作業者が問題の種別を定義しながら実施した．作業者は，要求仕様書中の問題を 6 つの問題に分類した（図 2.2 参照）．作業者による予備実験の結果を集計して種別ごとに検出数まとめた結果，**問題 2** の出現頻度が最大であったため，本論文では**問題 2** について議論する．また，著者によって，**問題 2** の細分類化を行った（図 2.3）．本論文では，この**問題 2** の細分類に基づいて自動検出手法を検討・実装し，その結果から今後の自動検出のための課題について議論する．

本章では，2.2 で組込みシステムの要求仕様書とエンタープライズシステムの要求仕様書の特徴について論じ，2.3 で組込みシステムの要求仕様書中に存在する問題の分類について述べる．2.4 では予備実験として，組込みシステムの要求仕様書中に存在する問題を，手作業によって抽出および分類し，2.5 では予備実験の結果から，**問題 2** に焦点を当てて，再度組込みシステムの要求仕様書中に存在する問題を，手作業によって抽出および分類し，分類手法を検討して実装した．2.6 にて考察を述べた後，2.7 において関連研究を述べ，2.8 にて本論文をまとめる．

2.2　組込みシステムの要求仕様書

本研究では，企業において実際に使用され，作成元によってレビューによる修正がすでに加えられた組込みシステムの要求仕様書を使用する．本章では，組込みシステムの要求仕様書の特徴と，本論文で対象とする要求仕様書について述べる．

2.2.1　組込みシステムの要求仕様書の特徴

組込みシステムの要求仕様書は，エンタープライズシステムの要求仕様書と比較して，より具体的な記述が多くの場合存在する [22]．この理由として，エンタープライズシステムの開発と比べて多くの場合に使用するハードウェア仕様の概要が決定しており，ソフトウェア開発のみで最終工程まで実施できないことがある [23]．また，組込みシステム開発ではハードウェアを意識するが，エンタープライズシステムではデータに注意を向けることが多い [22]．さらに組込みシステムは，アクチュエータによって物理世界への影響を与えることが多い [22]．そのため，組

込みシステム開発におけるクライアントは，発注時点で開発対象の仕様を考えるためにハードウェアの仕様を理解している必要となり，クライアントも開発対象の要素技術に知見があることが多い．多くの組込みシステムには厳しいリソース制約が課せられており，例えば動作温度条件や低消費電力化などの要求や，低コスト化の要求がある場合には，メモリやプロセッサなどのハードウェアリソースの制約が要求されることもある [3, 26, 2, 27]．加えて，より厳しいタイミングや信頼性への要求が存在する組込みシステムもあり，例えば，エンジン制御システムが挙げられる [2, 17]．また，組込みシステム開発では，システムレベルの文書中にソフトウェアの要求定義や，ハードウェアとソフトウェア間のインタフェースの定義も記述されることがある [28]．そのため，独自の分析が必要となる．

本論文で組込みシステムの要求仕様書に限定した理由を述べるために，組込みシステムの要求仕様書とエンタープライズシステムの要求仕様書の違いを述べる．文献 [29] より，下記の文書に記載されている項目を調査した．文献 [29] の表 9 には，エンタープライズシステムに関する要求仕様書などのソフトウェア開発文書が 24 件示されており，アクセス可能な文書は 8 件である．さらに基本設計書を除外し，今回アクセスしたエンタープライズシステムの要求文書は，以下 7 件である．

d1 京都市立病院総合情報システム仕様書（京都市立病院）

d2 京都府立新総合資料館 (仮称) 統合情報システム 図書系システム機能要求仕様書 （京都府）

d3 新図書館情報システム構築等委託業務仕様書（高知県と高知市）

d4 『栄養管理システムソフトウェア一式』仕様書（国立障害者リハビリテーションセンター）

d5 佐賀中部広域連合地域包括支援センターシステム要件定義書（佐賀中部広域連合）

d6 国有財産総合情報管理システムに係る設計・開発及び移行業務一式仕様書 (案)（財務省理財局管理課）

d7 松江市上下水道局水道施設管理マッピングシステム構築業務要求仕様書（松江市上下水道局）

これらの文書中に記載してある項目を表 2.1 にまとめる．これらの項目は，目次から読み取ることができる項目のうち，代表的なものを著者が検出したものである．また，本論文で対象とした組込みシステムに関する要求仕様書は，表 2.2 の項目からなる．

表 2.1: エンタープライズシステムの要求文書がもつ項目

項目	記述内容	d1	d2	d3	d4	d5	d6	d7
導入背景	システムの必要性	○		○			○	○
ドメインの背景/概要	組織の状況	○	○	○	○		○	○
対象システムの機能要求	必要な機能が列挙	○	○	○	○	○	○	○
ハードウェア要求	利用可能な計算機の台数や性能	○	○		○	○	○	
ネットワーク要求	利用可能なルータなどの情報	○			○	○	○	
非機能要求	セキュリティや耐障害性など	○	○	○	○	○	○	○
業務実施/運用保守要求	人員構成や保守に向けての要求	○	○	○	○	○	○	○
納品/納入	納品日や納入先の指定	○		○	○	○	○	○
法務関係	関連法規に関する説明			○			○	

表 2.2: 対象とした組込みシステムの要求仕様書が持つ項目

項目	記述内容
機能要求	状態定義，ガード条件，イベント，処理，遷移に基づく記述階層構造になっている．
非機能要求	使用性および保守性に関する記述

エンタープライズシステムの要求仕様書と本論文にて対象とした要求仕様書は顧客の立場が異なっている．既述の通りであるが，組込みシステム開発の顧客もドメイン知識を持っている可能性が高く，本論文にて対象とした車載組込みシステムにおいても，顧客も自動車制御に対する知識を所持している可能性が高い．つまり，組込みシステム開発においては，顧客も開発者もエンジニアであることが多い．一方で，エンタープライズシステムにおいては，顧客が行政や医療施設であるなど，情報システムの専門家ではない担当者が含まれている状況で発注を行うケースがあり，発注者は情報システムの開発経験が少ないまたは無いと考える．

次いで，表 2.1 および表 2.2 にあるように，エンタープライズシステムの要求仕様書と本論文にて対象とした要求仕様書は内容が大きく異なる．エンタープライズシステムに関する要求仕様書は，業務を遂行するにあたっての人的もしくは物的に関する制約，およびドメイン知識や法務関係に関する記述の割合が多くを占めている．一方で，今回対象とした組込みシステムに関する要求仕様書は，人的・物的制約に関する制約，およびドメイン知識や法務関係に関する記述が少ない，または存在していない．その代わり，組込みシステムに関する要求仕様書には，機能要求および非機能要求，また，その仕様が必要な理由や階層化による詳細な要求の説明，および状態遷移に関わる記述など，極めて設計情報に近い内容が記述されている．

以上の差異から，今回の調査や自動検出が，エンタープライズシステムとどのように異なるかを述べる．まず，今回対象とした組込みシステムの要求仕様書には，ドメイン知識の説明がない．エンタープライズシステムの要求仕様書には，ドメインの背景などについても記述があり，ドメイン知識が無いか少ない読み手であっても，組込みシステムの要求仕様書と比べて意味レベルの指摘が容易となる可能性が高いと考える．

そして，エンタープライズシステムの要求仕様書は，基本的にはシステムを構成する個々の機能とそれらを動作させる環境についての記述が多く，状態遷移や画面遷移に関わる記述は存在しないか，相対的に少ない．しかし，今回対象とし

表 2.3: 要求仕様書の概要と詳細な問題の検出結果

		項目数	問題 1	問題 2	問題 3	問題 4	問題 5	問題 6
機能要求		121	322	561	22	98	332	92
非機能要求	使用性	5	16	3	0	0	6	3
	保守性	3	4	1	0	0	4	3
合計		129	342	565	22	98	342	98

た組込みシステムの要求仕様書には，単一の機能とそれらの状態遷移，および詳細なガード条件やイベントが多く含まれている．そのため，組込みシステムの要求仕様書は要求間の結合度が高く，より細粒度の記述が多く存在する．

以上のことから，今回対象とした要求仕様書中には，表 2.1 に示したエンタープライズシステムの要求仕様書と比べてより細粒度かつ難度が高い記載がある．そのため，構文や語句レベルを超えて，意味の解釈を必要とする調査を実施するためには，より専門性の高い要求や，具体的な振る舞い上の制約の妥当性を読み手は解釈する必要がある．

エンタープライズシステムの要求仕様書を対象とした調査や手法の適用に言及するためには，実際に調査を実施し，手法を適用した結果が必要だと考える．そのため，本論文では，エンタープライズシステムに調査を実施した場合や，手法を適用した場合については言及しないこととする．

2.2.2　本論文で対象とする要求仕様書

本論文で利用する要求仕様書は，実際に企業で用いられている車載組込みシステム向けの要求仕様書である．この要求仕様書は，129 項目からなり，すでにその企業でレビューが実施された，比較的品質の高いものである．この要求仕様書は，階層構造をとっており，深くなるほど要求が詳細化される．すでに，“機能要求” と “非機能要求：使用性”，“非機能要求：保守性” に項目が分かれている．機能要求については，分類が困難であるため，これ以上分割はしていない．

レビュー済みであることによって，以下の影響があると考えられる．

- 要求仕様書は，レビュー済みであっても完全に曖昧表現や誤り表現を除去できているかは不明である．

- レビュー済みの要求仕様書の方が，自然言語処理ツールによって適切な結果を得られやすい．

本論文では，手法の適用を含めた開発プロセスとして図 2.1 を想定している．対象とする要求仕様書は，書き手もしくはステークホルダによってレビューされており，その結果を受けて辞書を作成するものとしている．作成された辞書を用いて手法を適用し，その結果を受けて要求仕様書の修正を行う．

本手法を実装したツールの利用者は，手作業では検出できない問題の検出を目指していることを想定している．そのため，人間によるレビューの結果を上書きするものとして本手法は利用される．

レビュー済みの要求仕様書を用いる必要性については，辞書の作成および自然言語処理ツールの性能に依存する．辞書は，一般的に利用可能なものとその要求仕様書のドメインでのみ利用可能なものに分けられる．前者については，事前定義されていると想定されるため，問題ないが，後者については，書き手が必ず検出できるとは限らず，ステークホルダによって検出される可能性がある．そのため，本論文で辞書を必要とするような問題の自動検出のためには，レビュー済みの要求仕様書を対象とすることを想定する．

また，レビュー済みである必要性として，自然言語処理ツールを利用することが挙げられる．自然言語処理ツールによって対象の文章を解析するためには，利用する自然言語処理ツールが要求する形式や表現に修正する必要がある．そのため，文書を執筆しながら問題検出を行うことや，未レビューの要求仕様書を利用することについては，今回想定していない．

2.3　要求仕様書中の問題の分類

Berry は，英語で記述された計算機中心のシステムに対する要求仕様書中の曖昧さを定性的に分類した [25]．彼は,「曖昧さ (ambiguity)」について論じており，言語学における曖昧さは，語彙的 (lexical)，構文的 (syntactic)，意味的 (semantic)，および語用論的 (pragmatic) の 4 つに区別されると述べている．加えて，彼の経験に基づく問題として，非決定性 (indeterminacy) および言語上の誤り (language error) が存在するとも述べている．これら 6 つの区分は，実際に文章中に出現する場合に排他的ではないと述べている．そのため，定性的な定義は明らかにされているが，定量的にどの程度これらの曖昧表現が存在するかが明らかではない．

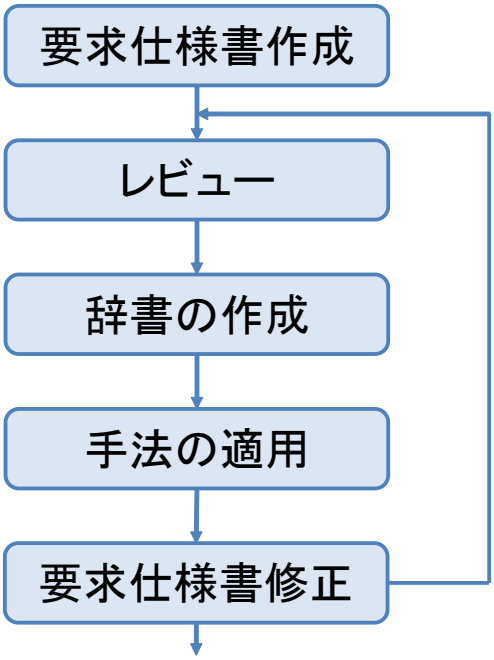

図 2.1: 想定している要求仕様書の作成フロー

そこで，本研究においては，Berry による分類を参考として，作業者による実験を実施する前に，問題の分類を行った．問題の分類にあたって，要求仕様書の構成要素を定義する．まず，字 chr は 1 文字のことを指し，本論文においてはこれ以上分解することができない．語 wrd は長さが 1 以上の chr の列である．句 phr は長さが 1 以上の wrd の列である．文 snt は長さが 1 以上の phr または wrd の列である．加えて，snt に対する構文木 syn は，$syn = SyntaxTree(snt)$ から得られる．述語 $SyntaxTree$ は，引数で与えられた文に対して構文解析を行い，構文木を抽出するものである．最後に，意味 sem は，$sem = Semantics(snt_1, snt_2, ..., snt_i)$ から得られる．述語 $Semantics$ は，引数で与えられた 1 つ以上の文に対して意味解析を行い，文章間の文脈を含めた意味を検出するものである．

ここで，本論文では要求仕様書中の問題の分類を，表 2.4 のように定義する．また，本論文では，誤り表現は，完全性や一貫性を損なう表現を指す．これらが要求仕様書に出現した箇所のことを誤り表現と呼ぶこととする．表 2.4 の分類から，**問題 1** については，曖昧語句や未定義に関する分類であり，表中に示すとおり，すべての問題に対して意味 sem が関係している．例えば，**問題 1** では，何が曖昧語句なのかはその組織や記述対象に依存する上，その文や文脈によっても変化するため，意味が必要となる．**2.4** における予備実験では，実験対象の要求仕様書中においては，**問題 1** から**問題 6** が存在していた．**問題 1** から**問題 6** を，上述の誤り表現

および曖昧表現に分類すると，**問題 1** と**問題 2** は曖昧表現，**問題 3**，**問題 5** および**問題 6** は誤り表現（不完全）．**問題 4** は誤り表現（不整合）かつ曖昧表現である．また，これらの**問題 1** から**問題 6** はドメイン知識への依存が弱く，**問題 7** から**問題 9** はドメイン知識への依存が強い．

1 章で述べたとおり，ドメインへの依存が弱ければ自動抽出できる可能性がある．ドメインに強依存である場合の誤り・曖昧表現の抽出は，ドメイン毎に固有の抽出手法を定義する必要があり，普遍的な自動化手法を定義することが困難である．誤り・曖昧表現は，ドメイン知識がなくとも自動抽出手法を定義できる可能性が高いことから，本論文では誤り・曖昧表現に注目する．上述の通り，**問題 1** から**問題 6** は，ドメインの専門家でなくとも抽出可能性の高い問題である．しかし，**問題 7** から**問題 9** について，ドメインの専門家ではないことによる，以下の困難さがある．

- **問題 7** は，対象が必要な操作なのかの判断が困難である．
- **問題 8** は，文間の意味的矛盾の解釈が必要であり，明らかな矛盾でない限りにおいてはドメイン知識が必要である．
- **問題 9** は，**問題 8** と同様である．

本論文では，問題の抽出を行った作業者が車載組込みシステムの専門家ではなく，実際に**問題 7** から**問題 9** が抽出されなかったことからも，ドメインの専門家でなくとも抽出の可能性が高いと考えられる曖昧・表現の抽出を対象とする．すなわち，**問題 1** から**問題 6** の問題を対象とすることとしている．

方法の欠如（**問題 6**）を取り上げ，目的語の欠如（**問題 7**）を取り上げない理由は，予備実験において作業者が**問題 7** を抽出できなかったことにある．これらの分類は作業者には伝えられておらず，事前に曖昧・誤り表現の定義を作業者に伝えていない．そのため，先入観を持たずに文書を読んでいる．このことから，要求仕様書には**問題 7** に該当する誤りが存在しないのか，**問題 7** を作業者が見つけ出せていないのか，判断することが困難である．予備実験の考察にて述べるが，**問題 6** は，ある動作に対する言葉の定義が不足していることによる問題が多く，ドメイン知識がなくとも抽出できている．しかし，**問題 7** の目的語の欠如は，動作を適用する対象が不明となり，文書の品質を低下させるものである．そのため，**問題 7** の抽出を試みたいと考えるが，作業者が抽出できなかったため，本論文では対象から除外した．

表 2.4: 要求仕様書の問題分類

問題の分類	式
問題 1: 曖昧な意味の語句	$sem \wedge (wrd \vee phr \vee snt)$
問題 2: 語句の一貫性・構文上の曖昧表現	$(wrd \vee phr) \wedge syn \vee sem$
問題 3: 誤字・脱字	$\begin{cases} sem \wedge (wrd \vee phr \vee snt) \\ chr \vee wrd \end{cases}$
問題 4: 疑問	sem
問題 5: 主語の欠如	$sem \wedge syn$
問題 6: 方法の欠如	$sem \wedge (wrd \vee phr \vee snt)$
問題 7: 目的語の欠如	$sem \wedge (syn_1, syn_2, ..., syn_i)$
問題 8: 文間の整合性	sem
問題 9: 妥当性	sem

2.4　予備実験

予備実験では，要求仕様書の曖昧表現や誤り表現を調査するために，作業者が曖昧表現や誤り表現を検出する．その後，作業者は，検出した内容に対して，曖昧表現や誤り表現を分類する．予備実験の目的は，自動検出手法を定義できる可能性が高い問題を検出することである．予備実験において使用した要求仕様書は，2.2.2 において説明したものを使用する．また，作業者は実験時，名古屋大学大学院情報学研究科博士前期課程の学生であった．作業者は，予備実験実施時点で要求仕様書から状態遷移モデルを抽出する研究に従事しており，自然言語処理ツールの JUMAN および KNP を利用した経験がある．

以下に，予備実験の手順を示す．

P1 作業者は，要求仕様書を読み，作業者自身が誤り表現や曖昧表現だと判断した部分を検出する．

P2 検出時には，作業者自身で誤り表現や曖昧表現の分類を定義する．

P3 検出した誤り表現や曖昧表現を分類する．

P4 検出結果を著者が確認し，作業者および確認者を交え，問題の分類の妥当性についての議論を行う．

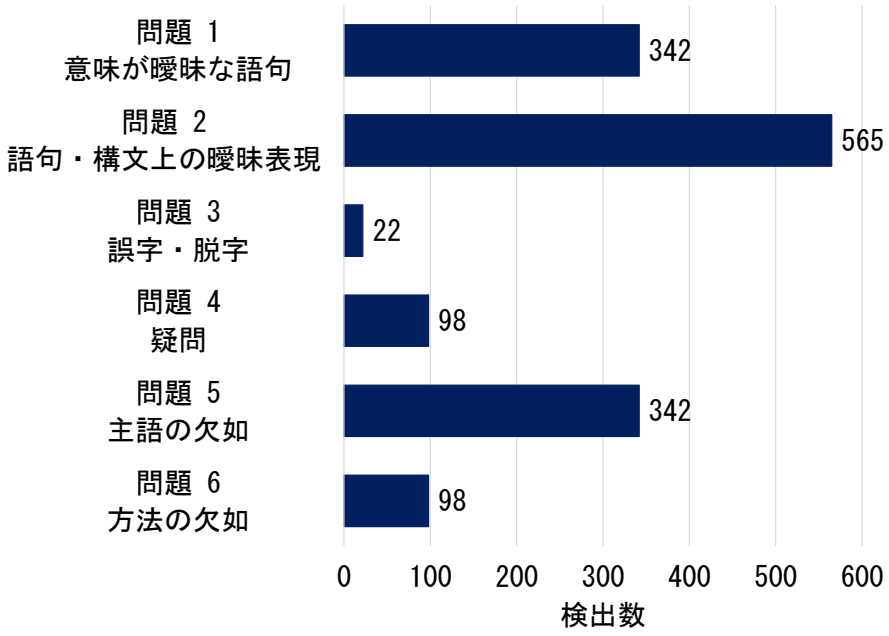

図 2.2: 対象とした要求仕様書における各問題の検出数

加えて，検出結果を著者が確認し，作業者および確認者を交え，問題の分類の妥当性についての議論を行う際の流れは，以下の通りである．

T1 作業者が検出した問題の箇所を，著者が要求仕様書を読んで確認する．

T2 ある問題について，著者が，確認中により適切な分類が存在すると判断した場合，著者は作業者に問題の適切な分類を提案する．

T3 問題の分類を意識しながら，再度著者は要求仕様書を読んで漏れを確認し，漏れが存在する場合，著者は作業者に漏れと思われる箇所を提案する．

T4 提案内容について著者，作業者との間で議論を行った結果，**T2** および **T3** の提案が作業者にとっても受け入れられるものならば，作業者は検出結果を修正する．

問題の検出にあっては，作業者が手作業によって検出しており，ワープロソフトにおける検索機能や着色機能を利用しているものの，自作スクリプトやその他の関連ツールを利用していない．また，確認作業にあっては，確認者は初回の問題分類の確認と最終的な問題分類を確認した．

実験手順の妥当性を向上するため，**P4** における議論の結果，著者，作業者，確認者間で合意を取ることができなかった場合，**P4** までの結果を踏まえて，再度 **P1**

を実施する．これらの手順は，5 回実施され，5 回目の時点においては，著者，作業者，および確認者間で検出箇所の誤りや不足が存在しないことについての合意をとった．本論文において用いた要求仕様書は企業から提供を受いるために公開できない．そのため，作業者を増やすことが困難であり，妥当性を高めるためにこれ以上の作業者を増やすことは困難であった．

以上の内容を予備実験として実施した．作業者によって，**2.3** に示した通りの問題のうち，**問題 1** から**問題 6** の計 6 分類が得られた．6 つの問題それぞれの検出数は， 図 2.2 に示す通りである．

以降では，各問題についての作業者による分類と検出の結果およびそれらの考察を述べる．以降の例については実際の要求仕様書から抜粋したものではなく，同様の問題を持つ文章に置き換えたものである．

2.4.1　問題 1（曖昧な意味の語句）

［結　　果］本論文では，曖昧な意味の語句を以下 2 つに分類する．

問題 1-A 曖昧語句

問題 1-B 未定義語句

まず，曖昧語句とは，定量的に扱うための情報が存在しない，または，書き手や読み手の解釈によって程度が変化しうる語句を指す．例えば,「大きい」,「丁度いい」および「十分な速度」などが挙げられる．次いで，未定義語句とは，ある文中に存在する語句のうち，定義が存在しない記述対象のシステム特有の語句を指す．例えば，文章中に「管理システム」という語が存在するとき，その定義がどこにもなければ,「管理システム」は何を管理するのか，入出力は何かなど，情報が不足している．そのため，要求仕様書を読むために必要な知識がより多く必要となり，ステークホルダ間の意思疎通の妨げになる可能性がある．

［考　　察］今回対象にした要求仕様書はレビューを加えられた比較的高品質のものだが，**問題 1** については，300 箇所を超える問題が指摘されており，6 分類の問題の中では，**問題 5** と並んで 2 番目に多い．

2.4.2　問題 2（語句・構文上の曖昧表現）

［結　　果］この問題では，名詞・動詞に関するもの，助詞に関するもの，および係り受け表現に関するものの大きく 3 種類が存在する．後に示す**問題 2** の細

分類（図 2.3）においては，名詞・動詞に関するものが**問題 2-A** および**問題 2-B**，助詞に関するものが**問題 2-C**，**問題 2-D** および**問題 2-E**，係り受け表現に関するものが**問題 2-F** に対応する．名詞・動詞に関するものとして，語句の揺らぎ (**問題 2-A**) と受動態（**問題 2-B**）がある．語句の揺らぎとは，同一の事物だが，異なる表記で表現される場合に該当する．例えば,「パソコン」と「パーソナルコンピュータ」や,「ON」と「オン」などである．受動態については，ある主語に対して受動態と能動態とが混在している場合の表現を指す．助詞に関するものは，助詞の使用方法の揺らぎ（**問題 2-C**），節末尾の揺らぎ（**問題 2-D**）および格助詞の連続（**問題 2-E**）がある．助詞の使用方法の揺らぎは，同様のことを記述したいと思われるが，明らかに助詞の統一が取れていない問題である．例えば,「ユーザに連絡する」と「ユーザへ連絡する」などである．節末尾の揺らぎは，節末尾の助詞の使用方法が揺らいでいる問題である．意味は同じだが，接続詞や副詞的名詞の直後に書かれる助詞が異なるものを指す．例えば,「場合は、」,「場合に、」および「場合、」などがある．同一格助詞の連続は,「が... が」などのように同一格助詞が連続している問題を指す．このとき，同一格助詞の間に接続詞，副詞的名詞，接続詞を含むもの，および鉤括弧と二重鉤括弧で括られている文を除く．加えて，副助詞である「は」の連続も問題であることとする．副助詞「は」は主語の直後に来ることがあり，連続すると主語が曖昧になることが理由である．最後に，係り受け表現に関する問題では，並列語句の切れ目が不明確（**問題 2-F**）である．並列語句の直後に書かれる動詞がどの語句に係るかが不明確となる箇所を指す．または，並列語句が連体修飾の役割を保つ場合がある．例えば,「制御システムによって DC モータや空気圧縮、点灯の制御を行う」について,「制御を行う」のは「点灯」だけなのか，それとも「DC モータ」および「空気圧縮」も含むのかを，読み手が判断することは困難な場合がある．

［考　　察］**問題 2** については，500 箇所超の問題が指摘されており，6 分類の問題の中では，最も多く検出されている．これらについては，記述上，確かに読み手の解釈が困難な可能性はあるが，読み手が解釈できるが構文的に誤りであるものも存在した．

2.4.3　問題 3（誤字・脱字）

［結　　果］**問題 3** は，誤字・脱字である．まず，明らかに連続するはずのない語句の連続があげられる．例えば，格助詞「が」が 2 回連続して「がが」など

のように書かれている場合，明らかな誤り表現である．また，誤字・脱字とも表現の揺らぎともとれるものがある．例えば,「なっているのか」と「なっているか」が，文章中にそれぞれ出現した場合が挙げられる．最後に，格助詞の誤用が挙げられる．例えば,「OFF する」など,「OFF にする」のように助詞を除いて表現したものである．

［考　　察］**問題 3** については，20 箇所程度の問題が指摘されており，6 分類の問題の中では，最も少ない出現回数である．

2.4.4　問題 4（疑問）

［結　　果］**問題 4** は，疑問である．これは，作業者が理解できなかった箇所を指す．まず，参照するべき資料名は記載されているが，内部資料か外部資料かが不明で，かつ入手先も不明のため，資料を参照できない問題がある．次いで，**問題 2** にも関連するが，語句の揺らぎか自信を持って判断できない場合がある．例えば,「制御システム」と「制御 ECU (Electronic Control Unit)」は同じなのかなどが挙げられる．第 3 に冗長表現が挙げられる．明らかに自明な言葉を詳しく説明しすぎており，かえって誤解を生じたことが挙げられる．他に，説明が長いだけではなく，助詞に「が」ではなく「も」を用いることで意味に含みを持たせているとも解釈可能な文が存在する．また，定義に対する疑問がある．明らかに記述されるべき制約や事前条件などが不足していることがある．例えば,「電車はドアを閉じてから発車する」などの制約が欠如している場合である．最後に，作業者のドメイン知識の不足により理解できない語句があった．

［考　　察］**問題 4** については，100 箇所程度の問題が指摘されている．この問題については，他の問題にも依存する可能性がある．例えば，主語が存在しないために文章を解釈できないなど，他の問題との複合的な原因の可能性がある．

2.4.5　問題 5（主語の欠如）

［結　　果］**問題 5** は，主語の欠如である．主語が存在しない文には曖昧表現だと判断した．

［考　　察］**問題 5** については，350 箇所程度の問題が指摘されている．**問題 1** と同程度であり，2 番目に多く検出されている．欠如した主語の多くは要求仕様

書の記述対象だと思われるものだが，中には別の主語が補完されるべき箇所も存在した．

2.4.6　問題 6（方法の欠如）

［結　　果］**問題 6** は，方法の欠如である．まず，ある動作についてデータの授受の方法が不明であるという問題がある．具体的には，通信規格や送受信データの形式が明記されていない問題である．または，制限事項の不足が挙げられる．例えば，デッドラインや周期が示されていないなど，非機能要求が不足しているため，要求される性能の程度が不明である．最後に，要求仕様書中の数式における述語定義が曖昧である問題が挙げられる．

［考　　察］**問題 6** については，100 箇所程度の問題が指摘されている．これは，**問題 3** に次いで出現数が少ない問題であった．

2.4.7　予備実験のまとめ

予備実験の結果から，6 分類の問題が作業者によって検出された．以上の予備実験の結果から，以下のことがわかった．

- **問題 2** を除く問題は，文脈やドメイン知識に対する依存が強い．
- **問題 2** は，検出数が最大である．

問題 5 も構文解析によって検出できる可能性はあるが，主語が無くとも伝わる文の場合には，主語を補完するべきかどうかはドメイン知識や文脈に依存する．そのため，**問題 2** を自動検出できれば要求仕様書に存在する問題をより低減できると考えられる．

問題 2 は，既述の通り検出数が最多である．本論文では，すべての問題を自動検出することを目的とはしておらず，最も検出数の多い問題の自動検出を試みることにより，**問題 2** 以外の自動検出可能性の調査を目的としている．検出数が多い問題を選択することにより，より多様な問題細分類が可能となり，それによってより多面的な考察が可能になると考えたため，**問題 2** の曖昧表現や誤り表現を対象とした調査に注目することとした．

以降，本論文では**問題 2** の自動検出手法を検討することにより，要求仕様書に存在する問題の自動検出手法の検討を行う．

2.5　問題の自動検出

予備実験の結果から，6 分類の問題が作業者によって検出された．本論文では，これらの分類のうち，**問題 2** について自動検出する手法を実装した．そこで，図 2.3 にあるように，**問題 2** の内容を細分類化した．

本章では，**問題 2** の細分類の自動検出手法を検討するため，以下の手順にて実験を実施した．

S1 作業者は，再度，要求仕様書を読み，**問題 2** に該当すると判断される箇所を検出する．このとき，予備実験の結果を受けず，検出した誤り表現や曖昧表現を分類する．

S2 検出結果を著者が確認し，著者と作業者で問題の分類の妥当性についての議論を行う．

S3 **S1** と **S2** を，著者による指摘がある間反復する．

S4 著者による各問題に対する自動検出手法の検討・実装する．

S5 自動検出結果を作業者に提示する．

S6 自動検出結果について著者と作業者が議論する．

S7 議論の結果を踏まえ，必要なら作業者が再度要求仕様書を確認，検出結果を修正する．

S1 から **S2** は **P1** から **P4** と比べて，分類を作業者が定義しないことと，分類する問題が異なる点を除くと同様である．また，妥当性を向上するために **S1** と **S2** は 2 回実施され，2 回目の時点において，著者および作業者間で検出箇所の誤り表現や曖昧表現の不足が存在しないことについて合意をとった．このとき，確認手順は，2.4 の **T1** から **T4** に基づく．

S4 では，各細分類に対する検出手法を検討する．手法については，日本語形態素解析システムとして JUMAN[1]を使用している．また，JUMAN と連携可能な日本語構文・格・照応解析システム KNP[2]によって係り受け解析 [30][31] や照応解析 [32][33] を行い，さらに検出能力を高めることを検討している．詳細は **2.5.1** にて

[1]http://nlp.ist.i.kyoto-u.ac.jp/index.php?JUMAN
[2]http://nlp.ist.i.kyoto-u.ac.jp/index.php?KNP

述べる．**S5** から **S7** では，自動検出結果について議論を行い，その結果に応じて再度手作業による週出結果の修正を実施した．議論の結果は **2.5.9** において述べるが，対象にした要求仕様書は 129 項目以上からなり，手作業による検出作業においては見落としが生じる可能性がある．そこで，提案手法による検出結果を作業者が確認した上で，作業者が手作業によって再度検出作業を行った．

2.5.1　細分類に対する自動検出方法

以下，**問題 2** の細分類毎に自動検出手法を説明する．以下の各手法においては，保守的 (conservative) な手法と，積極的 (liberal) な手法の 2 つを提案する．各手法の詳細なアルゴリズムは**付録**にて説明する．また，手法を実装した際のソースコードを WEB 上にて公開した[3]．

保守的な手法は，適合率 (*precision*) が向上することを目指し，積極的な手法は，再現率 (*recall*) が向上することを目指している．また，F 値は *precision* と *recall* の調和平均とする．正解集合を作業者が検出した曖昧・誤り表現とし，その個数を正解数とするとき，*precision* および *recall* は，それぞれ以下の式で定義される．

$$precision = \frac{\text{正解集合に含まれる自動検出結果の数}}{\text{自動検出による検出数}} \tag{2.1}$$

$$recall = \frac{\text{正解集合に含まれる自動検出結果の数}}{\text{正解数}} \tag{2.2}$$

しかし，作業者の見落としやツールによる誤検出の可能性が存在することを考慮し，ツールによる検出数と手作業による検出数に差がある場合，再度問題の検出作業を実施することとした．このとき，著者および作業者で議論を行い，問題に検出精度のさらなる向上を行った．

また，**問題 2-A**，**問題 2-C** および**問題 2-D** においては，辞書の作成が必要となる．辞書は，図 2.1 のようなプロセスで辞書を作成することを想定している．辞書は，一般的に利用可能なものとその要求仕様書のドメインでのみ利用可能なものに分けられる．

具体的な問題の種別を用いて説明する．**問題 2-A**，**問題 2-C** および**問題 2-D** について辞書を作成する必要がある．しかし，**問題 2-A** はドメイン固有の用語が必要となることがあり，**問題 2-C**・**問題 2-D** は一般的な日本語としての揺らぎであ

[3] http://ertl.jp/~muku/ieice_201905.html

るという違いがある．そのため，後者については事前に辞書を定義しておくことが可能となる．

問題 2-A については，書き手が必ず検出できるとは限らず，ステークホルダによって検出される可能性がある．そのため，本論文では，書き手によるレビューおよびその他のステークホルダによるレビューの結果として，手法適用の最初にドメイン固有の辞書が作成される．

2.5 章の実験において，すべての辞書情報を作業者による作業が 1 度終わるたびにアップデートした．特に，1 回目では，語句の揺らぎという分類を意識し始めた段階であり，揺らぎ語句を完全には検出できていない可能性がある．最終的な作業者による作業の結果の辞書情報を，自動検出の際に利用している．

問題 2-A に対しては，作業者による辞書情報をそのまま積極的な手法で用いており，そのうち略称の関係にあるものを残して保守的な手法で用いている．

問題 2-C および**問題 2-D** については，**問題 2-A** とは異なり，一般的な文法上の問題であるため，該当する形態素を容易に絞り込むことが可能である．そのため，作業者による辞書情報をベースとして，パターンを増加させて辞書を作成致した．**問題 2-C** および**問題 2-D** の辞書については，本論文で示した積極的な手法と保守的な方法の 2 つしか考えられないわけではなく，辞書を変更することで，その組織のポリシーにあった制約に組み替えることも可能である．

2.5.2　問題 2-A（語句の揺らぎ） に対する検出手法

問題 2-A については，保守的な手法と積極的な手法で，検出手法は同等である．検出手法を説明する．本手法では，事前に揺らいでいる語句の組を辞書として用意し，辞書中に定義されている語句を全て検出する．

積極的な手法は，辞書として用意される揺らぎ語句の組の数が保守的な手法よりも多い．保守的な手法は，積極的な手法における辞書の項目の内，略称の関係となるもののみを残している．略称の関係とは，辞書のある組において，文字列長が長いものに対して，1 文字以上の削除のみを行うことで，文字列長が短いものと一致する組の関係である．

2.5.3　問題 2-B（受動態） に対する検出手法

まず，保守的な方法について述べる．保守的な方法は，受動態表現すべてを検出する．このとき，KNP を用いて “¡態:受動態¿” と分類されたものを受動態であるとする．

そして，積極的な方法について述べる．積極的な方法は，保守的な方法に加えて，ある名詞に対して受動態でも能動態でも表現している場合の能動態表現も，揺らぎであることとして検出している．

2.5.4　問題 2-C（名詞に係る助詞の不統一） に対する検出手法

問題 2-C は，保守的な手法と積極的な手法で，同様の検出手法をとる．共通の検出手法について説明する．文章中に存在する，名詞と助詞の組を全て検出する．事前に同様の意味を持つ助詞の組の辞書を定義し，もし,「私は」と「私が」などのように，同一名詞かつ異なる助詞の組み合わせがあった場合，辞書に基づいて同様の意味だと判断し，助詞の揺らぎとしている．

積極的な手法では，同一の意味を持つ助詞の組が保守的な手法よりも多い．

2.5.5　問題 2-D（節末尾の助詞の不統一） に対する検出手法

問題 2-D は，保守的な手法と積極的な手法で，同様の検出手法をとる．共通の検出手法を説明する．文章中に存在する副詞的名詞 （場合，時，際など）と，その直後に存在する助詞の組み合わせを検出する．このとき，助詞が存在せず読点が出現する場合は，副詞的名詞と読点の組として検出する．このとき，混同されていても問題ない助詞は，辞書として定義される．

積極的な手法では，辞書の項目数が保守的な手法よりも少ない．

2.5.6　問題 2-E（同一格助詞の連続） に対する検出手法

問題 2-E に対する手法については，保守的な手法と積極的な手法がほぼ同一である．

下記の構文を考える．

⟨ 格助詞 A⟩ ... ⟨ 格助詞 A⟩

形態素解析の結果，上記のような構文がひとつの文の中で現れた場合，格助詞の不正な連続パターンとする．このとき，上記の構文であっても問題がないと判断することがある．その基準は以下のとおりである．

- 始端の格助詞の直後に動詞 (する，して等) が存在する場合
- 同一格助詞の間に読点が存在する場合
- 同一格助詞の間に副詞的名詞が存在する場合
- 二重鉤括弧『』の中に存在する格助動が構文の始端/終端に選ばれた場合

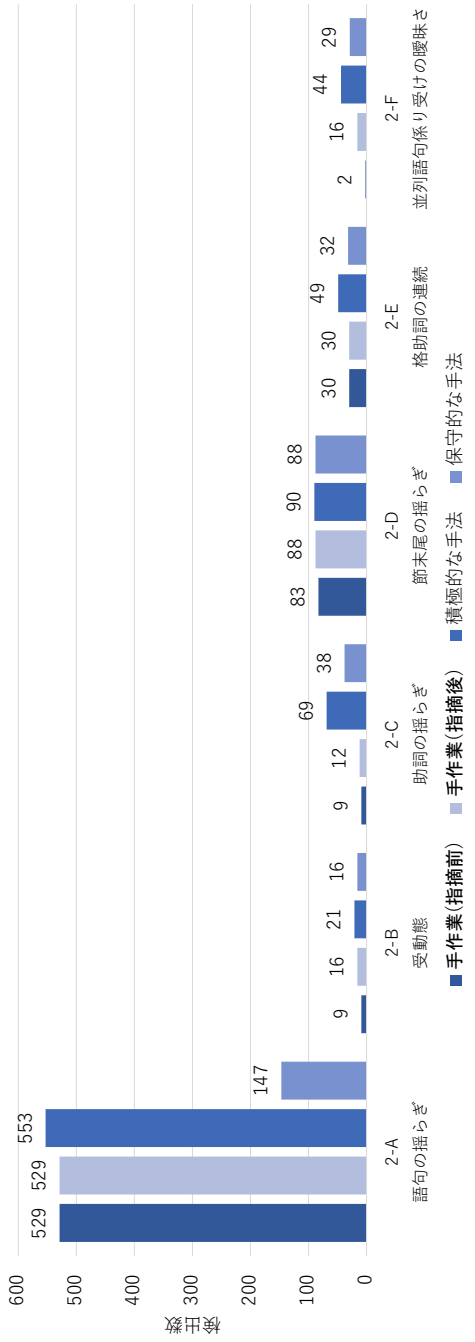

図 2.3: 作業者とツールによる問題の検出

表 2.5: **問題 2** の細分類に対する検出結果の precision と recall

	指摘前						指摘後					
	積極的な手法			保守的な手法			積極的な手法			保守的な手法		
	precision	*recall*	F	*precision*	*recall*	F	*precision*	*recall*	F	*precision*	*recall*	F
2-A	0.95	0.99	0.97	1.00	0.28	0.43	0.95	0.99	0.97	1.00	0.28	0.43
2-B	0.43	1.00	0.60	0.56	1.00	0.72	0.76	1.00	0.86	1.00	1.00	1.00
2-C	0.13	1.00	0.23	0.24	1.00	0.38	0.17	1.00	0.30	0.32	1.00	0.48
2-D	0.92	1.00	0.96	0.94	1.00	0.97	0.98	1.00	0.99	1.00	1.00	1.00
2-E	0.59	0.97	0.73	0.91	0.97	0.94	0.59	0.97	0.73	0.91	0.97	0.94
2-F	0.05	1.00	0.09	0.07	1.00	0.13	0.32	0.88	0.47	0.41	0.75	0.53

これら 4 つの基準について説明する．まず，始端の格助詞の直後に動詞 (する，して等) が存在する場合は，格助詞が連続しても文意を解釈することができる．いま，「A により B して C になる。」という文を考える．このとき，格助詞「に」が重複しているがその後の動詞「して」によって，最初の述語の時点で対応しうる主語がただ 1 つに確定するため，問題は無い．次に，同一格助詞の間に読点が存在する場合は，読点で節が分かれていることが多いことから，例外とした．そして，同一格助詞の間に副詞的名詞が存在する場合について，副詞的名詞である「時」，「際」および「場合」など，次の節につなげるための語句が間に存在する場合は，例外とした．二重鉤括弧『』の中に存在する格助動が構文の始端/終端に選ばれた場合について，二重鉤括弧『』の中は固有名詞であることが多いため，その中に存在する格助詞は適用対象外とするためである．

積極的な手法と保守的な手法の差分は，連続を許す助詞の存在の有無である．積極的手法では，全ての助詞の連続を許容しないが，保守的な手法においては，格助詞「の」の連続を許容している．

2.5.7　問題 2-F（並列語句の係り受けの曖昧表現） に対する検出手法

ここでいう並列語句とは「A，B，C および D」における A から D のように，等価に扱われるべき名詞または名詞句を指す．

積極的な方法について述べる．積極的な方法は，KNP を用いて並列表現を検出する．このとき，動詞に対する並列表現は語句レベルの並列表現ではないと判断し，除外している．

次いで，保守的な手法について述べる．保守的な手法は，この内，並列語句が連体の役割を持つか，もしくは並列語句の直前に連体の役割を持つ語句があるものに限定している．これを実現するために，KNP の解析情報において，“連体修飾” を含むものを，連体の役割を持つ語句とした．

2.5.8　各検出手法のアルゴリズム

ここでは，**問題 2** の細分類に対する自動検出手法のアルゴリズムを説明する．

以下の手法では，共通して使用される述語が存在する．これらの述語について以下に示す．

述語 SSearch　第 1 引数に与えた文字列中に，第 2 引数に与えた文字列の組が含まれている場合，その位置を返す．ただし，第 1 引数に，第 2 引数に与えた文字列が複数存在する場合もあるため位置の組が返される．

述語 Juman　第 1 引数に与えた文字列に対して Juman を実行し，第 2 引数にその結果を要素として追加して返す．

述語 Knp　第 1 引数に与えた文字列に対して'knp -tab' を実行し，第 2 引数にその結果を要素として追加して返す．

述語 ConcatNoun　第 1 引数に，Juman による解析結果を与える．解析結果について，名詞また未定義語が連続している場合，それらを結合して 1 語と扱うように解析結果を変更して，変更結果を返す．未定義語は，本論文においては名詞として扱っても問題のないものだけであった．なお，結合時に見出し語は結合されるが，それ以外の解析結果は，2 つ目の形態素の解析結果を用いる．

述語 GetW　Juman による 1 つの形態素の解析結果を与え，その見出し語を返す．

述語 HCheck　第 1 引数に与えたある形態素の Juman の解析結果が，第 2 引数に与えた品詞である場合 $True$ を返す．

述語 HSCheck　第 1 引数に与えたある形態素の Juman の解析結果が，第 2 引数に与えた品詞細分類である場合 $True$ を返す．

問題 2-A

Algorithm1 に，**問題 2-A** の自動検出手法を示す．**Algorithm1** の入力においては，**S** は要求仕様書全体であり，s_i は要求仕様書の項目とする．また，**d** は辞書であり，$\langle w_{i0}, w_{i1} \rangle$ は揺らぎ語句の組である．**d** は事前にユーザによって用意されている．ただし，**d** に属する任意の，$\langle w_{i0}, w_{i1} \rangle$ について，必ず第 1 要素である w_{i0} のほうが，w_{i1} よりも文字列長が短いこととする．この制約は必須ではないが，辞書としての可読性を考慮したものである．

Algorithm1 中の述語について説明する．述語 Omit は，第 1 引数と第 2 引数がいずれも空ではないとき，第 3 引数の文字列の組が，略称の関係である場合には，非省略語の検出箇所を，省略語の検出箇所から除去している．

積極的な手法と保守的な方法では，**d** のサイズが異なる．**Algorithm1** の返り値 **R** は，揺らぎ語句が存在するファイル名とファイルごとの検出数である．

問題 2-B

Algorithm2 および **Algorithm3** に，**問題 2-B** の自動検出手法を示す．これらは，いずれも引数として，要求仕様書全体 **S** を与える．ただし，s_i は要求仕様書の項目とする．

Algorithm2 および **Algorithm3** 中の述語について説明する．述語 PassiveCheck は，第 1 引数に KNP による解析結果を与え，もし，受動態が検出された場合に $True$ を返し，受動態が存在しなければ $False$ を返す．述語 GetNoun は，第 1 引数に KNP による解析結果を与え，主語を返す．最後に，述語 PassiveExpPos は，第 1 引数に KNP による解析結果の列を与え，受動態表現の位置を返す．

保守的な手法では，受動態の箇所だけを全て検出しているが，積極的な手法は，それに加えて受動態と能動態の主語が混同して使用されている場合も検出している．

問題 2-C

Algorithm4 に，**問題 2-C** の自動検出手法を示す．**Algorithm4** の入力においては，**S** は要求仕様書全体であり，s_i は要求仕様書の項目とする．また，**d** は，揺らぎと判断される助詞の組である．

Algorithm4 中の述語について説明する．述語 NounJoshiCheck は，第 1 引数に，Juman による 1 つの文の解析結果を与え，第 2 引数に，その文中の形態素のインデックスを与える．もし，第 2 引数で指したインデックスの形態素が名詞または未定義語であり，かつ副詞的名詞ではないことと，そのインデックスに 1 を加算した形態素が助詞ならば，$True$ を返す．述語 DeleteExpression は，第 1 引数に，これまでの検出結果を与え，第 2 引数に，辞書を与える．第 1 引数のうち辞書に存在しない表現を削除して返す．

積極的な手法と保守的な方法では，**d** のサイズが異なる．

問題 2-D

Algorithm5 に，**問題 2-D** の自動検出手法を示す．**Algorithm5** の入力においては，**S** は要求仕様書全体であり，s_i は要求仕様書の項目とする．また，**d** は辞書

であり，要素は，揺らぎと判断しない表現の組からなる．

Algorithm5 中の述語について説明する．述語 GetPhr は，第 1 引数に Juman の解析結果を与え，第 2 引数に文，第 3 引数に形態素のインデックスを与える．規定の表現になるまで見出し語を結合し続ける．具体的には，⟨ 名詞 , 助詞 , 読点 ⟩ または，⟨ 名詞 , 読点 ⟩ の組となる表現を返す．

積極的な手法と保守的な方法では，**d** のサイズが異なる．積極的な方法では，**d** のサイズが零である．

問題 2-E

Algorithm6 に，**問題 2-E** の自動検出手法を示す．**Algorithm6** の入力においては，**S** は要求仕様書全体であり，s_i は要求仕様書の項目とする．**d** は連続でも問題ないとユーザが判断した助詞の組である．

Algorithm6 中の述語について説明する．述語 Check は，第 1 引数に Juman の解析結果を与え，第 2 引数，第 3 引数に参照する形態素のインデックス，第 4 引数に辞書を与える．返り値は 2 つあり，1 つ目については，2.5.6 にて定義した問題のない構文の基準を満たしており，かつ，辞書 **d** に含まれない助詞の連続が認められる場合に $True$ を返し，そうでなければ．$False$ を返す．2 つ目の引数は，2 つ目以降の連続した格助詞の位置の組を返す．

積極的な手法と保守的な方法では，**d** のサイズが異なる．積極的な方法では，**d** のサイズが零である．

問題 2-F

Algorithm7 および **Algorithm8** に，**問題 2-F** の自動検出手法を示す．**Algorithm7** および **Algorithm8** の入力においては，**S** は要求仕様書全体であり，s_i は要求仕様書の項目とする．

Algorithm7 および **Algorithm8** 中の述語について説明する．述語 NounMod は，第 1 引数に KNP の解析結果を与え，被連体修飾語を検出する．述語 PreNounMod は，第 1 引数に KNP の解析結果を与え，連体修飾語を検出する．述語 ExtractPara は，第 1 引数に KNP の解析結果を与え，並列表現に当たる文節を検出する．述語 CutVerb は，第 1 引数に KNP の解析結果，第 2 引数に並列表現の文節を与え，動詞を含む分節かどうかを判定し，文節を含んでいれば空集合を返す．

積極的な手法と保守的な方法を比べると，保守的な手法では，動詞を含む並列表現を除去している．

2.5.9　自動検出の結果と議論

上記の手法を適用するためにツールを実装した．著者および作業者による議論を行う前後の検出結果を，図 2.3 にまとめる．図 2.3 においては，作業者がツールの結果を確認せずに行った手作業による検出作業のうち，5 回の反復における最後の検出結果を指摘前として示している．また，指摘後とは，作業者がツールの出力結果を確認し，作業者が著者と議論を行った後に再度手作業による検出作業を行った結果である．**問題 2-B**，**問題 2-C**，**問題 2-D** および**問題 2-F** の指摘前後の数に差が存在する．これらについては，後述する議論の結果生じたものである．

また，*precision*，*recall* および F 値を表 2.5 に示す．表 2.5 における指摘前・指摘後も図 2.3 と同様の意味である．なお，表 2.5 の数値の導出のための詳細なデータは，WEB 上にて公開した[4]．

手作業およびツールの検出結果を用いて，著者と作業者間で**問題 2** の各細分類の検出結果に関する議論を行ったため，その結果を述べる．議論の内容は，手作業と自動の各検出結果との差分と，問題が与えうる影響についてである．

まず，**問題 2-A** の議論を説明する．次の例は，語句の揺らぎとの複合的な問題である．

> 加速度センサの出力値から、加減速度を決定する。
> センサの値が正の場合は加速度、負の場合は減速度を表す。

この例は「加減速度」と「加速度、減速度」の定義の文であり，他の複数の文において「加減速度」も「加速度、減速度」も使用されるため，作業者は「加減速度」と「加速度、減速度」の組を語句の揺らぎと判断している．しかし，この定義によれば，センサの出力値が零の場合の定義が存在しないこととなる．加速度は，一般に実数値を取る単位である．これは略称語句と非略称語句の揺らぎではあるが，単に語句の揺らぎという問題だけではなく，曖昧な定義ともいえる．本来，予備実験の時点で，作業者が**問題 4** に分類するべき問題であるが，語句の揺らぎを議論することで検出された．加えて，この議論の際に「システム加減速度」なる語句が出現することがわかった．議論の結果，この語句は「加減速度」との揺らぎ

[4] http://ertl.jp/~muku/ieice_201905.html

ではないこととしたが，確実な結論を出すためにはドメイン知識が必要であった．これらのことから，表現に対する疑問を生じることもあるため，略称語句と非略称語句の揺らぎを検出することは，要求仕様書の品質向上に繋がる可能性があると結論づけた．

問題 2-B の議論について説明する．受動態表現は，主語を省略することが能動態表現と比べて容易であるために主語の欠如を生じ，**問題 5** との複合的な問題になることが多くなると予想される．そのため，**問題 2-B** が修正されることで，**問題 5** を軽減することにもつながる可能性がある．**問題 5** は主語の欠如であり，受動態は主語の省略を助長する可能性がある．そのため，受動態表現はすべて問題であると著者と作業者の間で結論づけた．よって，すべての受動態表現を問題としている**問題 2-B** の保守的な手法による検出結果を，全て問題であることとした．

問題 2-C の議論について説明する．この問題については，その助詞でなければ文として成立しない表現以外は統一されるべきだと結論づけた．例えば，助詞「に」と助詞「へ」は統一可能な場合とそうではない場合がある．例文を用いて説明する．

1) 私は水族館に行く
2) 車はガソリンによって動いている

例文 1 では,「に」を「へ」と置換しても問題なく読むことができるが，例文 2 では,「に」を「へ」と置換することができない．つまり，例文 1 の様な場合に,「に」と「へ」が文書中で統一されていなければ，曖昧表現として検出するべきだが，例文 2 の様な場合は曖昧表現として検出するべきではない．

問題 2-D の議論について説明する．この問題については，表現の差によって，軽微な意味の差があるのではないかと結論づけた．例えば,「場合は、」,「場合に、」および「場合、」の差について検討をした．

- **「場合は、」**は，条件を強調しており，また，他にも場合分けがあることを強く示唆したい．
- **「場合に、」**は，特に強調したいことはなく，単にその条件を示したい．
- **「場合、」**は，上記どちらとも取ることができる．

しかし，文章解釈としてはあくまで些末な問題であると言える．また，ツールは「場合」だけではなく,「時」も抽出しており，これは手作業では抽出できていなかっ

た．議論の結果,「時」も問題であることとしたため，検出数が変化した．

問題 2-E の議論について説明する．格助詞および副助詞「は」の連続が問題になる場合について議論を行った．基本的には，主述の関係を取ることが困難になるため，問題であると結論づけた．しかし，例外として,「の」,「と」および「や」は問題ない場合も考えられる．これらの例外とすることが可能な助詞も，過度に連続して用いると可読性は低下するため，これらについては，どの程度連続しても可読性に問題を生じないかを定義する必要がある．

最後に，**問題 2-F** の議論について説明する．この問題については，並列語句に係る修飾語が括り出されていると，係り受けが不明確になる場合があると結論づけた．係り受けが不明確になる場合とそうではない場合を例文を用いて説明する．

友人のラーメンとうどんを食べた。

これの例文はいずれも並列語句に係る修飾語が括り出されている場合である．例文では,「友人の」の係り先が不明確である.「ラーメン」だけが「友人」の所有物であるのか,「うどん」も「友人」の所有物であるのかが曖昧である.「友人」は「ラーメン」と「うどん」をいずれも所有していたのか,「ラーメン」だけを所有していたのかは，前後の文やドメイン知識に基づいて，読み手が並列語句に対する係り受けを解釈するることとなる．そのため，冗長になる場合もあるが,「友人のラーメンと自分で購入したうどんを食べた」であれば，誤解なく読むことができる．そのため，並列語句に対する係り受けは曖昧にならないように記述されていることが望ましい．

2.6　考察

2.6.1　RQ1 に対する考察

図 2.2 をみると，最も出現数が多い問題は**問題 2** となった．**問題 2** の中でも，図 2.3 をみると，語句の揺らぎが最も多い結果となった．対して，図 2.2 をみると，最も出現数が少ない問題は**問題 3** となった．**問題 2** は，**2.4.2** の考察でも述べた通り，読み手が解釈可能だが構文上の誤り表現や曖昧表現が存在する問題である．また，書き手にとっては文意を解釈可能な問題である可能性が高い．これに対して**問題 3** は，**2.4.2** の考察でも述べた通り，文章校正時に，書き手が比較的容易に認識が可

能な問題である．例えば,「をを」の連続などは構文上の問題ではなく，読み進めていく段階でドメイン知識によらず検出可能な問題である．しかし，格を誤るような誤字脱字は，ドメイン知識が不足しているならば，問題を検出することは困難である．以上から，**RQ1** に対する結論は以下の通りである．

・書き手による認識が比較的容易な **問題 3**，**4**，**6** は検出数が少ない．
・書き手による認識が比較的困難な **問題 1**，**2**，**5** は検出数が多い．

2.6.2　RQ2 に対する考察

問題 1 から **9** を考えたとき，意味解析が必要な問題は自動検出手法が非常に高度になると考えられる．そのため，本論文では意味解析を用いなくとも検出可能性のある**問題 2** に焦点を当てて検討を行った．

問題 2 の結果を考える．**問題 2-A** の検出結果について，積極的な手法と保守的な手法を比較すると，*precision* は 保守的な手法のほうが高く，*recall* は 積極的な手法のほうが高い結果となった．積極的な手法では,「温度」と「体温」のような組も対象としており,「炉内温度」のように単純に体温との揺らぎではない語句も検出しているために，過検出している．そのため，比較的 *precision* が低い．保守的な手法では，略称語句のみ対象としているため *recall* が低い．また，**問題 2-A** では，辞書を用意しているため F 値が高いが，語句の揺らぎだという認識がなければ，辞書を用意できない．そのため，保守的な手法では，同一の意味を持つ別表現だということが比較的明確である略称語句のみ辞書に登録して実施した．非略称語句に対しても検出する場合には，例えば「温度」と「体温」のような語句が揺らぎとして存在した場合，これらが上位語と下位語の関係だと書き手が意図しているならば，揺らぎと断定できない場合もある．これを判断するためには，ドメイン知識や辞書から語句の関係が必要とある．このように，**問題 2-A** においては語句間の関係性を用いることでドメイン知識への依存を低減できる可能性がある．

問題 2-B の検出結果について，指摘前後で手作業による結果が異なるが，指摘後の検出結果をベースに考察する．以降，すべて指摘後の検出結果をベースに考察する．積極的な手法と保守的な手法を比較すると，*precision* は 保守的な手法のほうが高く 1.00 となり，*recall* は 同一の結果となった．積極的な手法は，受動態表

現の目的語と能動態表現の主語で揺れている場合に，能動態表現も検出している．そのため，作業者が問題だと検出しなかった箇所も検出しており，*precision* は 保守的な手法のほうが高くなった．JUMAN や KNP を利用した手法により，*recall* を 1.00 とすることができ，*precision* も 1.00 とできたため，自然言語処理ツールの能力にも依存するが，自動検出が容易な問題と言える．

問題 2-C の検出結果について，積極的な手法と保守的な手法を比較すると，保守的な方法の方が高い *precision* となり，いずれも *recall* は 1.00 となった．これは，積極的な手法の方が，検出対象とする辞書の項目数が多いためである．*precision* が 1.00 にならない原因は，議論において述べた，その助詞でなければ成り立たない場合の検出に手法が対応していないことが原因である．**問題 2-C** の *precision* を向上するためには，読み手が文意を解釈し損なうことのない助詞の使用方法を明確に定義した上で，そのルールベースで検出手法を検討する必要がある．

問題 2-D の検出結果について，積極的な手法と保守的な手法を比較すると，*precision* は 保守的な手法のほうが高く，*recall* は 同一の結果となった．積極的な手法で *precision* が 1.00 にならない原因は，積極的な手法では，異なる意味の助詞表現であっても検出しているためである．今回は存在しなかったが,「場合も、」のような表現が存在するとき,「も」によって，他の場合もあるのだろうということが読み取れる．誤字の可能性も存在するが，もし，その「他の場合」についてどこにも明記がなければ，それは条件の不足となり，文間の整合性の問題である **問題 8** にも影響する可能性がある．そのため，節末尾に存在する助詞を注視して，統一されていない箇所のニュアンスを考えることで，解釈を誤ることのないように注意することができる．本論文で対象にした要求仕様書では，**問題 2-D** が与える影響は存在しないが，他の要求仕様書ではこの問題を生じる可能性がある．

問題 2-E の検出結果について，積極的な手法と保守的な手法を比較すると，*precision* は 保守的な手法のほうが高く，*recall* は 同一の結果となった．*precision* が 1.00 にならない原因は，作業者の見落としと考えられる．**問題 2-E** では，連続している格助詞にもよるが，格助詞の連続は多くの場合で，正しい主述の解釈や可読性が問題となる.「が」や「を」などが連続していると,「が」の場合は主語,「を」の場合は目的語が曖昧になる．例えば，以下の文を考える．

・誤）回路がモータが回転することを通知する。
・正）モータが回転することを，回路が通知する。

誤例では,「が」が連続している．この例では,「回路が回転する」という主述の関係を意図していないことは，一般に，回路が回転するものではないということから類推できる．そのため，この例で回転するのはモータであり，モータは情報を通知できないため,「回路が通知する」という主述関係だと類推できる．しかし，回路とモータのように，深いドメイン知識がなくとも理解できる語句であれば良いが，そうでなければ，この文だけから主述関係を理解することが困難な可能性がある．正例では，1 つの節に主述関係を 1 つずつに書き換えたものである．正例のようにすれば，主述関係を疑義なく解釈できる．対して，連続していても軽微な問題である場合もある．例えば,「の」の連続である.「の」が 3 回以上連続すると，可読性は著しく低下するが，2 回程度の連続ではあまり問題にならない．しかし，可読性を損なう可能性があるため，修正するべき問題である．

問題 2-F について，積極的な手法と保守的な手法を比較すると，*precision* は 保守的な手法のほうが高く，*recall* は 同一の結果となった．*precision* が 1.00 にならない原因は，KNP における係り受け解析において，本来並列語句になるものが並列語句として扱われていないためである．**問題 2-F** は，ドメイン知識によっては曖昧表現があっても正しく読むことができるケースがあり，そのような曖昧表現を除外するための方法を検討する必要がある．そのため，同一クラスに属する語句を分類することや，語句間の関係性に注目することで，*precision* が向上する可能性はある．すなわち，語句間の関係性を用いた手法が必要になる．実際，加藤らはシソーラスを利用することで，ドメイン知識の不足を補って要求を獲得する方法を提案している [34]．また，文献 [34] では，要求獲得を支援するソフトウェア機能に関するドメイン知識として，4 つの特性を定義している．

特性 1 用語の識別と用語間の関係

特性 2 機能の識別

特性 3 機能間の関係

特性 4 機能に付随する制約

すなわち，これらが存在しなければ，要求獲得が困難になるということであり，これらの特性を持つ情報は要求の定義にあっても重要であると言える．そのため，これらの情報が解析できていなければ，意味解析が必要な問題の検出が非常に困難となる．

RQ2 のまとめは以下の通りである．

・辞書を使用するか字句・構文レベルの解析によって検出可能な
　問題 2-B と **問題 2-E** は，解析が比較的容易である．
・読み手が文意を解釈可能な場合の多い**問題 2-C**，**2-D** は，文意を解釈
　不可能な場合のルールの定義が必要である．
・今後，語句間の関係性に基づいて解析するべき**問題 2-A**，**2-F** に
　ついても検討を進める必要がある．

2.6.3　全体の考察

RQ1 において，検出数の多い問題は，比較的致命的にはなり難い問題である．検出数が特に多い**問題 1**，**問題 2**，**問題 5** は，ドメイン知識がある前提であれば，書かなくても文意が読み手に伝わる可能性が高い問題または曖昧になっていても文意が読み手に伝わる可能性が高い問題であるといえる．ただし，**問題 5** では，全ての文を対象として主語が存在しない場合に検出されるため，検出数が多い可能性もある．対して，検出数が少ない**問題 3**，**問題 4**，**問題 6** は，存在すると理解ができなくなるような重大なものが多いと考えられる．ただし，**問題 6** では，方法定義が不要な語句も多く存在しているため，確認対象の箇所が少ない．そのため，検出数が少ない可能性もある．

RQ1 と **RQ2** の結論から，初回作成時には重大な問題に関しては強く意識されているためあまり検出されず，軽微な問題である**問題 2** が多く検出されることがわかった．また，字句・構文の解析によって検出可能な問題であれば自動検出の可能性が高いことがわかった．また，分類した問題の内，自動検出可能性が比較的高いと考えられる**問題 2** について，その細分類を検討し，6 つの細分類に対してそれぞれ自動検出手法を提案した．自動検出手法は，保守的な手法と積極的な手法の 2 つを提案した．これらによって，高い *recall* を得られているが，*precision* の向上のためには，語句間の関係性の検出方法とその利用を検討する必要がある．

本論文の目的は，曖昧表現や誤り表現の種別，および自動検出可能な問題を調査することによって注意するべき問題の種別を明らかにし，より高品質な要求仕様書を作成可能かを調査することである．要求仕様書中には，書き手が気付くこと

の容易な影響の大きい問題だけではなく，書き手が気付くことの困難な影響の小さい問題が存在することがわかった．**RQ1** に対する考察で述べたが，**問題 1**，**問題 2**，**問題 5** の検出数が比較的多い理由は，読み手に対して意図が伝わるだろうと，書き手が見落とした問題であるためだと考えられる．

書き手が気付くことが容易な問題であっても，問題は存在している．そのため，これらを検出するための手法は必要である．しかし，そのためには書き手と同程度のドメイン知識の定義が必要となるため，手法の検討は困難であると考えられる．

書き手が気付くことが困難な問題は，これまで述べた通り，字句・構文レベルの問題も存在している．これらの問題は検出数が比較的多いため，これらの問題を検出するための手法は必要である．また，ほとんどの場合，保守的な手法を採用すると，9 割程度の再現率によってこれらの問題を検出可能だと示した．しかし，精度を向上するためには，ドメイン知識の定義や，語句間の関係性による解析方法の検討が必要である．加えて，これらの問題を低減するためには，書き手が問題を認識する必要がある．これらの問題についての検出手順を定義して要求仕様書を作成・確認することが重要である．要求仕様書の品質向上のためには，**2.1** でも述べたとおり，シナリオベースの方法を用いることで要求仕様書の品質向上を見込める [21]．このとき，本論文にて示した手順や技術を用いることができる．

組込みシステムの要求仕様書であることによる影響を考察する．今回の作業者である作業者は，今回対象とした要求仕様書のドメインの専門家ではない．そのため，作業にあっても，最初からドメイン知識を必要とする誤り表現を検出することは困難であると作業者は考えていたため，結果としても，字句・構文レベルの誤り表現や曖昧表現が多く検出され，本来，意味も必要となりうる誤字脱字は明らかなものだけしか検出できておらず，文間の意味の矛盾については検出できないという結果となったと考えている．

また，**問題 4**，**問題 6** も組込みシステムの要求仕様書であるからこそ検出数の多かった問題と考えている．**問題 4** については，ドメインの専門家ではないことから判断できなかった疑問が生じている．具体的には，先述した「制御システム」と「制御 ECU (Electronic Control Unit)」が同一なのかどうかなど，ドメインの専門家でないと判断が困難な疑問が生じている．**問題 4** の存在は，今回の調査において作業者が発見することができていないが，**問題 8**，**問題 9** に発展する可能性がある．これらの疑問から，論理的な矛盾を生じる可能性はあり，表 2.2 に示したとおり，状態遷移などの振る舞いに関する情報を要求仕様書中に多く含んでいることからも論理的な矛盾を生じやすくなっていると考える．しかし，**問題 8** や**問題 9** に

該当する問題を示すためには，ドメインの専門家が熟読した上で問題点を示す必要がある．**問題 6** については，方法に関するより詳細な記述が不足している問題である．組込みシステムでは，時間制約に関する要求がエンタープライズシステムと比べて，多く存在している．また，通信規格によってハードウェア構成が変化することもあり，より詳細な記述が要求仕様書であっても求められる．これらの記述が不足していると判断されており，**問題 6** が多く検出されたと考えている．

また，**問題 2** の中でも組込みシステムの要求仕様書であるからこそ多く生じた誤り表現や曖昧表現として，語句の揺らぎが挙げられる．表 2.2 で示した通り，ドメインの背景の記述が今回対象とした要求仕様書には存在していない．そのため，用語の使途がエンタープライズシステムの要求仕様書と比べて不明確になったと考えている．

データの形式などの記述がより詳細に書かれているために抜け落ちてはならない主語が エンタープライズシステムの要求仕様書に比べて多く存在したと考えている．

ここで，影響の少ない問題を検出する意義は，上述の通り，どの問題が記述者にとって重大となるかは判断できないことと，表現の揺らぎによって文書の保守に影響を与える表現を抽出できる可能があることである．たとえば，文書保守のために文字列検索などによって，文書全体の特定の文や語句を修正することがある場合には，**問題 2-A** の語句の揺らぎは後の技術的負債となる可能性がある．他にも，**問題 2-B** の受動態は主語を省略して書かれることが多いため，特定主語に対する変更がある場合に，見落としを生じる可能性がある．このように，文書執筆時点での文書の内容理解にあっては影響は小さいかと考えているが，ソフトウェアのライフサイクル全体として見た場合には影響が大きくなる可能性がある．企業でも，実際にキーワード検索を用いて文書作成後の変更箇所の抽出を試みた例がある [35]．キーワード検索で抽出するような手法の場合は，文字列が僅かでも異なっていると，問題のある箇所または抽出したい箇所を抽出できない可能性がある．そのため，文書執筆時には軽微な問題であっても，修正されていることが重要と考えている．

次いで，文献 [36] では要求インタビューにおける曖昧さのモデルについて検討を行っている．文献 [36] によると,「無害な曖昧表現」はコミュニケーション中に存在しており，修正しなくとも読み手は正しく曖昧表現を除去して解釈できる場合があると述べられている．しかし，文献 [36] においては，特に自動検出において，同時に「軽微」と「有害」の境界を定義することも難しいと述べている．その

ため，今回，著者らが要求仕様書を読んだ上では影響が小さい問題だが，他者にとっては重大な問題である可能性がある．

また，軽微な問題の検出結果を要求仕様書の記述者に示すことは，負担になることも事実かと思われる．しかし，本論文の立場として，実際に検出可能かどうかを検討することを目的としており，記述者の負担を考えるものではない．既述の通り，重大・軽微の境界には一定の基準が存在しない．そのため，仮に，本手法をシステムとして記述者が利用することを考えて実装する場合，記述者がどのような問題の種別を確認したいかは，ツールとして実装される際にオプション等で表示する問題種別をユーザが選択できればよいと考えている．すなわち，問題となりうる点はすべて検出しておき，それを表示するかどうかは記述者が決定できれば，記述者の負担にはならないと考えている．

本論文では，以下のように結論付ける．

・影響が大きいが検出数は少ない**問題 3**，**4**，**6** と，影響が小さいが
　検出数が多い**問題 1**，**2**，**5** が存在する．
・検出数が少なくとも，影響は大きいため，修正はされるべきである．
・検出数が多い問題は，書き手は文意を伝達可能と判断しても，
　読み手は誤り表現や曖昧表現と判断した箇所が多く存在する．

影響の大きい問題と影響の小さい問題，双方に対して異なるアプローチが重要である．

2.6.4　妥当性への脅威

本論文においては，著者，作業者および確認者らによって問題を検出している．著者らは対象とした要求仕様書のステークホルダではないため，要求仕様書の内容に対するドメイン知識が不充分な可能性がある．この背景から，本論文における問題は，記述されているシステムに対するドメイン知識の少ない開発者が，要求仕様書を読んだ際に生じる問題である．そのため，ドメイン知識が豊富な開発者がこの要求仕様書を確認する場合には，彼らにとっては問題ではない項目や，文脈から問題であると思う箇所を生じる場合がある．加えて，実験実施時点において，作業者は自然言語処理技術を用いた研究に従事している背景がある．そのため，構文上の誤り表現や係り受けの曖昧表現をより多く検出している可能性があ

る．作業者は，JUMAN や KNP を用いたツール開発にも従事しているため，日本語文法上への理解が比較的高く，自然言語処理ツールの振る舞いに対する理解が高い可能性がある．

次いで，題材とした要求仕様書には，実務者によるレビューを実施したものを使用している．そのため，誤字脱字等の認識が比較的容易な問題は，既に修正されている可能性がある．しかし，レビューを実施したにも関わらず多くの問題が検出される現状がある．レビューが施されていない要求仕様書中に存在する問題の出現頻度も，調査が必要である．

本実験の結果が示すように，作業を反復するなど，妥当性に影響を及ぼさないよう留意しているが，実際には検出漏れを生じている可能性はある．また，著者らは，ドメインの専門家ではないため，ドメインの専門家がレビューに加わると結果が変わる可能性がある．以上のように，確認作業の実施者が異なると，結果は変化する可能性がある．

車載組込みシステムにおいては，外的妥当性について，本論文では分析対象は 1 つだけではあるが，500 項目からなる規模の大きな仕様書であり，複数人で作成，レビューされているため，組込みシステムかつ同様の構造をとっている要求仕様書であれば，同様に結果を得られると考える．今回対象とした要求仕様書は構造化されており，段階的に詳細化された要求仕様書を対象としているため，内容が整理されている．

交差検証については，本論文では必要ないと判断している．本論文では対象とした要求仕様書全体から辞書を作成している．また，作業者が問題だと提示した箇所について，それを著者らで議論した結果，問題だと判断されれば，それがすべて正解集合となるため，交差検証を行う必要はないと考えている．

機械学習において交差検証を実施する場合，大量のデータがある前提のもとに学習用データとテスト用データに分割される．学習用データとテスト用データに分割する理由は，学習用データを過学習するリスクを避けるためである．

一方で本論文では，辞書の作成も手法に含めており，手法は，適用対象となる要求仕様書に特化して辞書を作成することとしている．そのため，過学習するリスクを考慮する必要はない．しかし，仮に，要求仕様書の何かしらの方法によって，学習部分（辞書の作成部分）と，適用部分（辞書の精度の確認部分）に分割されている場合，交差検証は必要だと考えている．この分割によって辞書の作成が実施される場合とは，要求仕様書が非常に大規模で，1 人で目を通すことが困難な場合だと考えている．本論文で対象とした要求仕様書は 500 項目からなる比較的規模

の大きい要求仕様書ではあるが，作業者はすべて読んでいる．部分的に抽出して辞書を作成していないため，交差検証の必要性は無いと考える．また，**問題 2-C** や**問題 2-D** のような，ドメインに対する依存性の低い辞書に対しては，他の要求仕様書を用意することができれば，交差検証を行うことが望ましいと考えている．

2.7　関連研究

要求事項レビューは，要求事項の妥当性確認のためによく用いられる方法である [18]．この標準では，要求事項レビューを実施するのが要求事項文書の検証及び妥当性確認の両方のおそらく最も普通の手段であり，レビュー者のグループを任命し，簡潔な説明をし，エラー，誤った前提条件，明瞭性の欠如，検証可能性問題，及び実際上の常識からのズレを見つけることを目的としている．

位野木らは，要求仕様の品質特性のうち，一貫性に注目して要求仕様の一貫性検証知識の形式知化，要求仕様の一貫性検証の支援ツールの開発およびその評価を行った [37, 14]．評価にあっては，ウェブサイトリニューアル事業といったエンタープライズシステムの要求仕様書を用いている．彼女らは，実際に企業から検証知識をインタビューによって抽出して共通部分，可変部分に仕分け，検証ルールと辞書として形式知化した．また，これらの検証ルールと辞書を組み込んだ要求仕様の一貫性検証支援ツールを開発し，このツールは，要求仕様書内で言及されている「アクター」「データ」「画面」「振る舞い」の記述が，要求仕様書内の記述と整合しているかを検証することができる．彼女らが提案したツールは，要求仕様書の一貫性として用語の不一致に着目し，省略表現，修飾表現および表記揺れに着目している．他にも，NG ワードの抽出やその自動補正の機能を持つ．

林らは，IEEE 830 の品質特性に基づく日本語要求仕様書の問題検出ツールである reqchecker を提案している [38]．この論文では，組込みシステムではない情報システムの要求仕様書を対象として予備実験を実施し，良好な結果を得ている．本論文では実際に企業で使用されている組込みシステムの要求仕様書を対象としており，より設計情報に近い詳細な記述が存在する．また，企業でレビューが実施された要求仕様書であり，比較的品質が高いものである．そのため，実際に現場で生じうる要求仕様書の品質低下の分析のためには適した題材である．

竹内らはソフトウェア開発における文書成果物の分析技術とその活用について調査を行った [39]．彼らは，要求仕様書だけではなく，ソフトウェア開発で作成される文書に対する分析技術を調査した．また，彼らはテキスト分析技術を用いた

開発関連文書の品質分析も行った [40]．組込みシステムのソフトウェア開発で作成された要求仕様書，アーキテクチャ文書，設計文書を対象として，自然言語処理を用いて曖昧表現の抽出を試みている．抽出結果から，オントロジーを用いずに曖昧表現の発見に貢献できた．この文献 [40] においても，本論文と同様，ガ格（主語）が省略されている問題が多く見つかっており，文献 [40] では，文全体のうちおよそ 50 %程度存在した．加えて，受動態表現が，トピックとなる語句を見つける際の妨げになる可能性についても言及している．

Gleich らは，曖昧表現を自動的に検出するための方法について検討を行った [41]．彼らの動機は，曖昧表現の検出の自動化をすること，ツールによって検出された曖昧表現が文書の問題であることの証拠を示すこと，および検出された曖昧表現の原因を説明することで要求分析者を教育することである．彼らのツールは，分析者が手作業で解析した場合と比べて 4 倍の曖昧表現を検出しており，かつ高い精度で曖昧表現を検出した．彼らは，実用的なツール開発を目指しており，高速かつ高い移植性を持つ，ウェブベースのツールである．彼らのツールは，grep ライクな技術を用いており，自然言語処理には依存していない．

次いで，Hasegawa らは，日本語の文書から概念グラフ（Conceptual Graph）の抽出を試みた [42]．彼らは，開発コストの低減のため，最終的には文書から高品質な要求モデルの抽出を目的としている．そのために，様々なモデルに変換可能な中間表現（概念グラフ）の自動抽出を，彼らは，形態素解析や依存性解析といった自然言語処理技術を用いた手法を提案した．彼らは，ヒューリスティックに基づく方法ではなく，単語の出現頻度や類似度などの数値に基づく方法を提案している．単語の出現頻度は語を重要度でソートするために用いられているが，出現頻度だけでは不十分であり，オントロジーなどを利用する必要性にも言及している．

関連して，Kof は，構文木の抽出，および字句および構文レベルの解析によってオントロジーの要求文書からの抽出を試みている [43]．彼は，要求文書を読む上での誤解は，要求文書中の語とその関係性を抽出，検証することが重要だと述べている．他にも，Kof は，要求文書から MSC (Message Sequence Chart) を計算言語学（computational linguistic）に基づいて抽出した [44]．

他にも，自然文書中の誤りの傾向を利用することで，高精度な誤り表現の訂正を目指した研究が行われている．Mizumoto らは，相互添削型言語学習 SNS である Lang-8 の添削履歴を用いて，誤り訂正を行う技術を提案している [45]．笠原らは，日本語学習支援を目的とした誤り訂正に関する研究の一環として，格助詞の誤り訂正技術の開発を行った [46]．この技術では，日本語学習者の誤り傾向を利用して

格助詞の訂正を行っている．要求仕様書において，頻出する誤り傾向をモデル化することができれば，誤り訂正の精度向上を実現できる可能性がある．英語で記述された文書を対象としても，同様の研究が行われている．例えば，Rozovskayaらは英語の非母語話者が記述した文書の前置詞訂正を行っている．彼らの研究においても，英語の非母語話者の誤り傾向を利用している [47].

今枝ら [48] は日本語学習者による格助詞の誤りに注目して誤りの検出と訂正を試み,「をを」などの同じ助詞の連続や，本研究でも取り扱った「を... を」のパターンや，隣接してはいけない助詞の連続を検出することに成功している．本論文では，現在ルールベースによる処理を行っており，格フレーム照合解析は今後の課題である．

本論文で提案した手法と関連する手法として，CEC 社による商用ツール ClearDoc がある[5]．ClearDoc は，仕様策定時点での抜けや漏れによって，その後のソフトウェア開発プロセスにおける不具合や手戻りの発生を抑制することを目的としている．ClearDoc は,「あいまい度指数」で文書を定量的に評価し，ドキュメントの品質管理を可能としたツールである．ClearDoc で，読み手が一意に解釈できない文ほど「あいまい度指数」は高くなる．ClearDoc は，検証時のテスト項目の元となることを考慮に入れて記載する項目を検討することで，必要な要件を盛り込んだ仕様書を作成することを目指している．ClearDoc を用いて， 以下のフローに基づくドキュメントの検証を実施できる．

1. あいまい度計測により，冗長な文を修正対象文として全文から抽出
2. 検証時にテスト項目として利用可能な仕様項目と，それ以外の項目を分類する機能を搭載
3. テスト項目作成時に追加検討されることが多い準正常系，異常想定項目を分類する機能を搭載
4. 検討されていない可能性の高い準正常系項目として利用できる条件節を抽出する機能を搭載

このように，本論文における提案手法とは異なり，テストプロセスを意識したツールである．しかし，公開されている情報では,「あいまい度」の具体的な評価が存在しておらず．その妥当性が不明である．

[5] http://www.proveq.jp/verification/document/cleardoc/

これに対して，本論文は，定量的調査が主題であり，手法を実装したツールの開発が主題ではない．手法を実際に適用することで，文章中の曖昧表現や誤り表現を自動的に抽出するためにはどのような手段をとるべきか，または，人間による曖昧表現や誤り表現の抽出支援のためには，どのような手段をとるべきかを考察するために，部分的に手法を実装している．そのため，どの程度手作業およびツールによる抽出が可能かを実証的に調査することが目的である．

以上より，ClearDoc と提案手法の比較を行うためには，ClearDoc についても本論文で利用した要求仕様書を与え，その実行結果を得る必要がある．ClearDoc は有償の商用ツールであることから，本論文においてはその有効性を調査していない．そのため，有効性については今後，調査を行う必要がある．

2.8　おわりに

本論文では，実際に企業で使用されている組込みシステムのコンポーネントの要求仕様書を用いて曖昧表現や誤り表現の分類・調査を行った．

曖昧表現や誤り表現の分類・調査においては，2 つの **RQ** を設定し，書き手にとって検出が容易な問題と，書き手にとって検出が困難な問題が存在することがわかった．また，自動検出にあっては，字句・構文解析によって検出可能であれば自動検出の可能性が高いことがわかった．

問題の種別の内，自動検出可能性が比較的高いと考えられる**問題 2** について，その細分類を検討し，6 つの細分類に対してそれぞれ自動検出手法を提案した．自動検出手法は，保守的な手法と積極的な手法の 2 つを提案した．保守的な手法では *precision* を向上するような手法を検討し，積極的な手法では *recall* を向上するような手法を検討した．その結果，保守的な手法では，*precision* が最低で 0.32，最高で 1.00，積極的な手法では最低で 0.32，最高で 0.98 となった．*precision* が低い結果となった分類は，揺らぎかどうかの意味的な解釈が必要となり，辞書やパターンマッチだけでは抽出が困難であった．辞書やパターンマッチ，自然言語処理ツールによって完全に抽出可能な問題は高い精度で抽出できた．*recall* については，保守的な手法では，最低で 0.28，最高で 1.00，積極的な手法では最低で 0.88，最高で 1.00 となった．*recall* が低い結果となった分類は，自動抽出を容易にするため，略称語句のみに辞書の登録語を絞ることや，*precision* と同様，揺らぎかどうかの意味的な解釈が必要であった．総じて，自動検出手法を実装したツールの結果は，高い *recall* を得られているが，*precision* は向上の余地がある．*recall* についても，

辞書に基づく手法は高い数値を示すが，辞書を使用しない場合に数値が減少することも見受けられた．

今後の課題として，$recall$ を向上するため，語句間の関係性の利用を検討する必要があること，および，他の要求仕様書についても定義した問題分類に合わせて問題の分類が可能か確認することが挙げられる．

Algorithm 1 問題 **2-A** の検出手法

Require:

$\mathbf{S} = \langle s_0, s_1, ..., s_{n-1} \rangle$

$\mathbf{d} = (\langle w_{00}, w_{01} \rangle, \langle w_{10}, w_{11} \rangle, ... \langle w_{(m-1)0}, w_{(m-1)1} \rangle)$

$\forall_x (x \in \mathbb{N} \wedge 0 < x < m \wedge (|w_{x0}| < |w_{x1}|)$

Ensure:

$\mathbf{R}$

```
1: function P2-A(S, d)
2:     R ← φ
3:     for i = 0 to n do
4:         for j = 0 to m do
5:             str1 ← SSEARCH(S[i], d[j][0])
6:             str2 ← SSEARCH(S[i], d[j][1])
7:             pos1, pos2 ← OMIT(str1, str2, d[j])
8:             if pos1 <> NULL then
9:                 R ← R ∪ ⟨d[j][0], i, pos1⟩
10:            end if
11:            if pos2 <> NULL then
12:                R ← R ∪ ⟨d[j][1], i, pos2⟩
13:            end if
14:        end for
15:    end for
16:    return R
17: end function
```

Algorithm 2 問題 **2-B** (保守的手法) の検出手法

Require:

$\mathbf{S} = \langle s_0, s_1, ..., s_{n-1} \rangle$

Ensure:

R

function P2-B-CONSERVATIVE(**S**)
  cnt $\leftarrow 0$
  J $\leftarrow$ JUMAN(**S**, **J**)
  K $\leftarrow$ KNP(**J**, **K**)
  for $i = 0$ to $|\mathbf{K}|$ **do**
    if PASSIVECHECK(**K**[i]) $= True$ **then**
      R $\leftarrow$ **R** $\cup$ PASSIVEEXPPOS(**K**[i])
    end if
  end for
  return R
end function

Algorithm 3 問題 **2-B** (積極的手法) の検出手法

Require:

$\mathbf{S} = \langle s_0, s_1, ..., s_{n-1} \rangle$

Ensure:

R

```
1: function P2-B-LIBERAL(S)
2:     cnt ← 0
3:     J ← JUMAN(S, J)
4:     K ← KNP(J, K)
5:     for i = 0 to |K| do
6:         if PASSIVECHECK(K[i]) = True then
7:             Noun ← Noun ∪ GETNOUN(K[i])
8:         end if
9:     end for
10:    for i = 0 to |S| do
11:        for j = 0 to |Noun| do
12:            R ← R ∪ SSEARCH(S[i], Noun[j])
13:        end for
14:    end for
15:    return R
16: end function
```

Algorithm 4 問題 **2-C** の検出手法

Require:

$\mathbf{S} = \langle s_0, s_1, ..., s_{n-1} \rangle$

$\mathbf{d} = (\langle \mathbf{p_{00}}, \mathbf{p_{01}} \rangle, \langle \mathbf{p_{10}}, \mathbf{p_{11}} \rangle, ..., \langle \mathbf{p_{(m-1)0}}, \mathbf{p_{(m-1)1}} \rangle)$

Ensure: $\mathbf{R}$

function P2-C($\mathbf{S}, \mathbf{D}$)
    cnt $\leftarrow 0$
    $\mathbf{J} \leftarrow$ JUMAN($\mathbf{S}$, $\mathbf{J}$)
    $\mathbf{J'} \leftarrow$ CONCATNOUN($\mathbf{J}$)
    for $i = 0$ to n **do**
        for $j = 0$ to $|\mathbf{J'[i]}| - 1$ **do**
            if NOUNJOSHICHECK($\mathbf{J'}[i], j) = True$ **then**
                $t \leftarrow$ GETW($\mathbf{J'}[i][j]$)+ GETW($\mathbf{J}[i][j+1]$)
                $\mathbf{M} \leftarrow \mathbf{M} \cup \langle t, i \rangle$
            end if
        end for
    end for
    $\mathbf{R} \leftarrow \mathbf{M} \setminus$ DELETEEXPRESSION($\mathbf{M}, \mathbf{d}$)
end function

Algorithm 5 問題 **2-D** の検出手法

Require:

$\mathbf{S} = \langle s_0, s_1, ..., s_{n-1} \rangle$

$\mathbf{d} = (p_0, p_1, ..., p_{m-1})$

Ensure:

$\mathbf{R}$

function P2-D($\mathbf{S}, \mathbf{d}$)
  $\mathbf{cnt} \leftarrow 0$
  $\mathbf{J} \leftarrow$ JUMAN($\mathbf{S}[i], \mathbf{J}$)
  $\mathbf{J}' \leftarrow$ CONCATNOUN($\mathbf{J}$)
  for $i = 0$ to n **do**
    for $j = 0$ to $|\mathbf{J}'[\mathbf{i}]| - 1$ **do**
      $nn \leftarrow$ HCHECK($\mathbf{J}'[i][j]$,"名詞")
      $cn \leftarrow$ HSCHECK($\mathbf{J}'[i][j]$,"副詞的名詞")
      $w \leftarrow$ GETW($\mathbf{J}'[i][j+1]$)
      if $nn = True \wedge cn = True \wedge w \notin \mathbf{d}$ **then**
        $iscj \leftarrow$ HCHECK($\mathbf{J}[i][j+1]$,"助詞")
        if $iscj = True \vee w =$ "、" **then**
          $\mathbf{M} \leftarrow \mathbf{M} \cup$ GETPHR($\mathbf{J}', i, j$)
        end if
      end if
    end for
  end for
  for $i = 0$ to n **do**
    for $j = 0$ to $|\mathbf{M}|$ **do**
      $cnt \leftarrow$ SSEARCH($\mathbf{S}[i], \mathbf{M}[j]$)
      $\mathbf{R} \leftarrow \mathbf{R} \cup \langle i, j, cnt \rangle$
    end for
  end for
  return R
end function

Algorithm 6 問題 **2-E** の検出手法

Require:

$\mathbf{S} = \langle s_0, s_1, ..., s_{n-1} \rangle$

$\mathbf{d} = (p_0, p_1, ..., p_{m-1})$

Ensure: **R**

function P2-E($\mathbf{S}, \mathbf{d}$)
  cnt $\leftarrow 0$
  $\mathbf{J} \leftarrow$ JUMAN($\mathbf{S}[i], \mathbf{J}$)
  for $i = 0$ to n **do**
    for $j = 0$ to $|\mathbf{J}[\mathbf{i}]| - 1$ **do**
      $cj \leftarrow$ HCHECK($\mathbf{J}[i][j]$,"助詞")
      $k \leftarrow$ HSCHECK($\mathbf{J}[i][j]$,"格助詞")
      $f \leftarrow$ HSCHECK($\mathbf{J}[i][j]$,"副助詞")
      $w \leftarrow$ GETW($\mathbf{J}[i][j]$))
      if $cj \wedge (k \vee f \wedge w =$ "は") $= True$ **then**
        $c, pos \leftarrow$ CHECK($\mathbf{J}, i, j, \mathbf{d}$)
        if $c = True$ **then**
          $\mathbf{R} \leftarrow \langle w, \quad (i, j) \cup pos \rangle$
        end if
      end if
    end for
  end for
  return R
end function

Algorithm 7 問題 **2-F**（保守的手法）の検出手法

Require:

$\mathbf{S} = \langle s_0, s_1, ..., s_{n-1} \rangle$

Ensure: **R**

function P2-F($\mathbf{S}$)
  cnt $\leftarrow 0$
  $\mathbf{J} \leftarrow$ JUMAN($\mathbf{S}[i]$)
  $\mathbf{K} \leftarrow$ KNP($\mathbf{J}[i]$)
  $\mathbf{NM}, \mathbf{PNM}, \mathbf{PARA} \leftarrow \phi$
  for $i = 0$ to n **do**
    for $j = 0$ to $|\mathbf{J}[i]| - 1$ **do**
      $\mathbf{NM} \leftarrow \mathbf{NM} \cup$ NOUNMOD($\mathbf{K}[i][j]$)
      $\mathbf{PNM} \leftarrow \mathbf{PNM} \cup$ PRENOUNMOD($\mathbf{K}[i][j]$)
      $p \leftarrow \mathbf{PARA} \cup$ EXTRACTPARA($\mathbf{K}[i][j]$)
      $\mathbf{PARA} \leftarrow \mathbf{PARA} \cup$ CUTVERB($\mathbf{K}[i][j], p$)
      $\mathbf{R} \leftarrow (\mathbf{NM} \cup \mathbf{PNM}) \cap \mathbf{PARA}$
    end for
  end for
  return R
end function

Algorithm 8 問題 2-F（積極的手法）の検出手法

Require:

$\mathbf{S} = \langle s_0, s_1, ..., s_{n-1} \rangle$

Ensure: R

```
1: function P2-F(S)
2:     cnt ← 0
3:     J ← JUMAN(S[i])
4:     K ← KNP(J[i])
5:     for i = 0 to n do
6:         for j = 0 to |J[i]| − 1 do
7:             p ← PARA ∪ EXTRACTPARA(K[i][j])
8:             PARA ← PARA ∪ CUTVERB(K[i][j], p)
9:             R ← PARA
10:        end for
11:    end for
12:    return R
13: end function
```

第3章　要求仕様書を対象とした状態遷移記述抽出ツールの定量的評価

3.1　要求仕様書に存在する技術的負債の背景と研究目的

2章にて述べたとおり，大規模なもので2000ページを超える規模からなる要求仕様書が存在している．要求仕様書の内容を理解し，仕様を整理するためには，モデリングを行うことで文書の抽象度を向上することができる．しかし，手作業によって要求仕様書から正確にモデルを抽出することは，非常に困難であり，かつ時間がかかる．

この問題を軽減するべく，自然言語によって記述された組込みシステムの要求仕様書から，状態遷移モデルを抽出する支援手法の検討を行った研究がある（以降，先行研究と呼ぶ）[8, 9]．2章にて述べたとおり，組込みシステムの開発では，エンタープライズシステムの開発と比べて多くの場合にハードウェア仕様の概要が決定しており，ソフトウェア開発のみでは最終工程まで実施できないことがある[23]．さらに組込みシステムは，アクチュエータによって物理世界への影響を与えることが多い [22]．そのため，要求仕様書の段階から，状態遷移記述を含んでいる可能性がある．先行研究の手法では，自然言語によって記述された組込みシステムの要求仕様書から独自の解析木を導出して節を抽出し，定義節，条件節および処理節のいずれかに分類する．分類された節を利用して，状態遷移記述の抽出を試みる．これらの手法を実装したツール（以下，解析ツールと呼ぶ）が存在しており，特定の使用条件下においては状態遷移モデルを抽出可能であることがケーススタディによって示されている [9]．先行研究における貢献の1つが節の分類であり，解析ツールは，ユーザによるインタラクティブな操作に基づいて，分類された節から状態遷移モデルにおける状態，イベント，処理および遷移を抽出する．そのため，節の分類精度は，状態遷移記述の抽出精度に影響を与える．たとえば，

定義節から状態遷移モデルで表現したい状態が持つ変域を抽出し，条件節からはイベントやガード，処理節からは処理や遷移を抽出できる．しかし，節の分類精度について十分な評価が行われておらず，先行研究における提案手法が，どの程度節を正しく分類可能か不明である．この節分類は，処理・条件・定義に該当する箇所の抽出を可能とするため，状態遷移表の抽出以外にも応用が考えられる．たとえば，形式仕様記述の抽出支援や，テストケース生成支援などが考えられる．また，これらの処理・条件・定義を正しく抽出することで，今後，意味解析を含む問題検出に寄与する可能性もある．

しかし，先行研究におけるツールにおける節分類の性能は調査されていない．そこで本研究では，日本語で記述され，また実際に企業で使用されている仕様書を用いて，先行研究にて開発された状態遷移表抽出支援ツールの節分類精度の調査を行うことを研究目的とする．

このため，以下のリサーチクエスチョン（**RQ**）を設定した．

RQ5-1 節の分類精度はどの程度か

RQ5-2 先行研究の提案手法では，分類できない節の種別が存在するか

上記 2 つの **RQ** について調査するべく，解析ツールによって解析可能な自然言語の文を調査した．

本章では，

3.2　先行研究によって提案された手法

3.2.1　手法の概要

中村らによる先行研究では，実務者によってレビューによる修正が加えられた組込みシステムの要求仕様書を対象とし，状態遷移モデルの抽出支援手法を提案している [8, 9]．先行研究において提案された手法を実装したツール（以下，解析ツール）のフローを図 3.1 に示す．本論文では，節の分類に焦点を当てて調査するため，節の分類までの手法の流れを概説する．詳細な説明は，文献 [8, 9] において述べられている．

解析ツールは，ユーザによるインタラクティブな操作によって状態遷移モデルの抽出を行う．はじめに節の抽出・分類を行うために，要求仕様書から解析木を抽出する．解析木の例を図 3.2 に示す．解析木は，節の抽出を目的としており，単純な

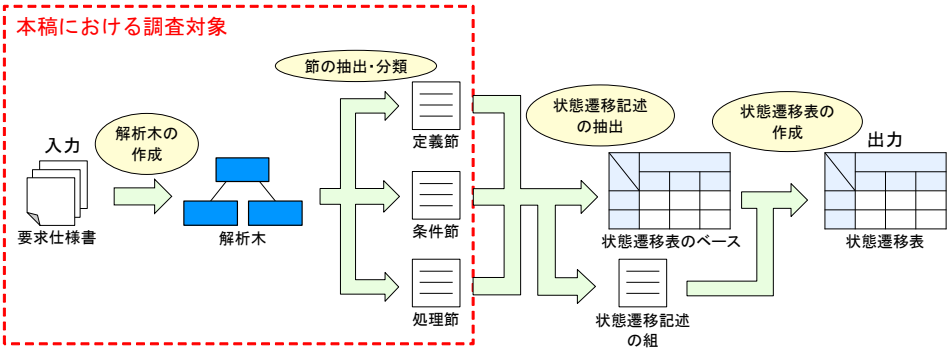

図 3.1: 先行研究にて開発された解析ツールのフロー

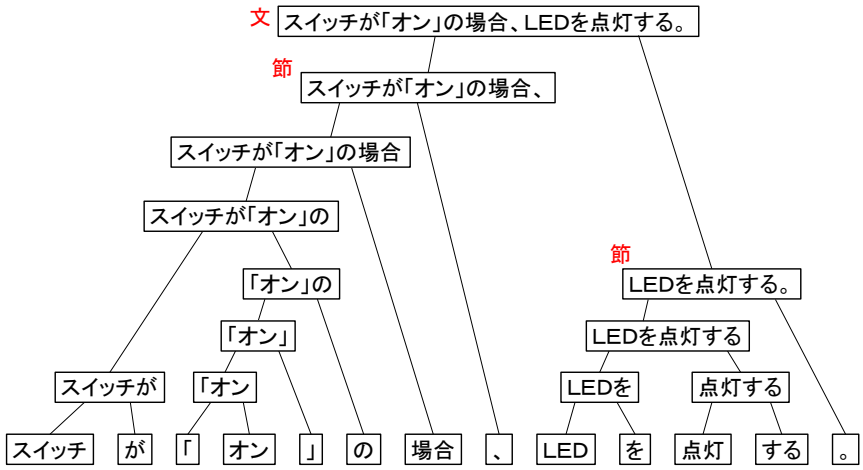

図 3.2: 解析木の例

構文木とは異なる構造をとっており，節が出現するようになっている．解析木は，日本語形態素解析システムの JUMAN [1] および，日本語構文・格・照応解析システムの KNP [2] を利用して抽出される．ここで，節とは，それ単体でも文として成り立つ形態素または語句の列であり，1 つ以上の節から文は構成される．また，解析木の葉ノードの粒度は形態素である．この解析木は，JUMAN の解析によって得られる品詞情報や品詞細分類の情報だけでは抽出することが困難であるため，KNP の解析によって得られる fearure 情報による文節への属性付与 [49, 50] を利用して，解析ツールの導出を行う．

次いで，解析木から節を抽出・分類する．解析ツールでは，要求仕様書の各文が以下の 3 つの節に分類される．

[1] http://nlp.ist.i.kyoto-u.ac.jp/index.php?JUMAN
[2] http://nlp.ist.i.kyoto-u.ac.jp/index.php?KNP

- 定義節: 状態を定義している節
- 条件節: 条件を表す節
- 処理節: 条件が満たされた時に実行される節

節の分類規則については，文献 [8, 9] において詳細に説明しているため，ここでは，各節種別の例文を示す．

定義節は，例えば以下のような文である．

スイッチは、「オン」と「オフ」の値を取る。

この文は，特に条件などなく，また振る舞いを表しておらず,「スイッチ」が取りうる値の範囲が定義されている．提案手法ではこのように状態を定義している節を定義節と呼称している．定義節は，仮に節としてさらに分解できそうな場合でも，文全体を 1 つの節として扱う．

条件節および処理節は，例えば以下のような文に存在する節である．

スイッチが「オン」の場合、LED を点灯する。

下線部が条件節，非下線部が処理節である．条件節は，それに伴う処理節が実行されるための条件が記述されている節である．そのため，条件節および処理節はそれ単独で文中に存在することはなく，必ず，1 つ以上の条件節に対して 1 つ以上の処理節が存在する．条件節は，1 つにまとめられるとは限らず，1 つの条件で 1 つの節となる．

以上は文に対する節の抽出・分類の手法であるが，要求仕様書中には箇条書き表現が存在する可能性がある．先行研究における手法では，箇条書き表現の同定のためにインデントの存在を確認している．もし，インデントが存在した場合，その行は箇条書きであると扱われる．箇条書き表現は，必ずしも文になるとは限らず，単に名詞句の列挙であることも考えられる．しかし，直前に存在している文を親の文としたとき，箇条書きは親の文から参照されているはずである．そのため，先行研究における手法では，要求仕様書が以下の表現であることを制約としている．

- 親の文には,「以下の」という表現が必ず存在していること．
- 親の文中の「以下の」が箇条書きを参照していること．

- 箇条書きの行頭にインデントが挿入されていること．

箇条書きは，親の文中の「以下の」が属する節を継承する．たとえば，以下の文を考える．

> 以下の条件がすべて成立しているとき、LED を点灯する。
> ・人感センサが周囲に人間がいると判断していない。
> ・システム異常が検出されていない。

この例では,「以下の」は親の文中の条件節に存在しているため，各箇条書きは条件節に紐付けられる．

以上によって抽出された節を用いて，解析ツールは状態遷移モデルの抽出を試みるが，本論文では状態遷移モデルの抽出は調査対象外であるため，説明を割愛する．

解析ツールに対してケーススタディが実施されており，6 名の被験者に対して，手作業，および解析ツールの支援により状態遷移モデルを抽出させる実験である [9]．選択する状態名（状態遷移モデルで記述したい状態を持つオブジェクト）を実験実施者が被験者に指定しているが，このケーススタディによって，解析ツールは要求仕様書から状態遷移モデルを抽出する支援が可能だと示されている．このケーススタディでは，手作業による状態遷移モデルの作成と比較して解析ツールによる支援を受けて状態遷移モデルを作成するほうが，要する時間が短いことを示している．このことから，特定条件下では状態遷移モデルを抽出支援できることは明らかとなったが，解析ツールの節分類精度などは評価が不充分であるため，調査が必要である．

3.2.2　解析ツールの限界

解析ツールに入力される要求仕様書は，以下の制約を満たしている必要がある．

- 数式を含む文にマーキングする．
- 箇条書きの接続詞を除去し，項目を分割する．
- 接続詞の位置を明確にする．
- 条件を表す記述において,「〜し」という表現を除去する．

- 箇条書き中の処理と遷移の記述を分離する．
- 順序関係を表す矢印を除去し同等の日本語表現に変更する．
- 単一項目の箇条書きを削除し，箇条書きを参照している文中に含める．
- 代名詞をそれが指す語句に置換する．
- 用語・表現を統一，明らかであるような論理的不整合を除去する．
- 主語抜けや目的語抜けなどの曖昧さを除去する．
- ある条件に対する動作で，遷移でない場合は状態名を含めないようにする．
- 誤字・脱字を除去する．

また，解析ツールは，節の分類にあたって，節種別を 1 つに決定できない場合が存在しているため，ユーザに対してインタラクティブな操作を要求している．すなわち，解析が困難である場合には，ユーザが節の分類を行うことがある．たとえば，解析ツールは以下のような問い合わせをユーザに提示する．

> 【起動したときのモードの初期値は「待機」であることとする。】
> に存在する【起動したときの】は処理に伴う条件ですか？

これに対して,「はい」または「いいえ」で回答を行う．もし「はい」と回答した場合，この文は条件節と処理節の組と判断され,「いいえ」と回答した場合，この文は定義節と判断される．

3.3　実験

本論文では，2.1 にて述べた，2 つの **RQ** を調査するべく，以下の 2 つの実験を実施した．

実験 1　節の分類の precision および recall の調査

実験 2　箇条書きの分類の precision および recall の調査

本実験においては，先行研究にて用いられた，修正済みの車載システムの要求仕様書を使用した．修正内容については，3.3.1 にて述べる．実験実施者は著者であり，あらかじめ著者が節の分類を手作業によって実施した．

ここで，**実験 1** の実施結果から，箇条書きの文は対象とせず解析を実行しているとわかった．その理由は，箇条書きの解析は，箇条書きを参照している親となる文中の,「以下の」をキーとしており,「以下の」が含まれている節の種別を継承するためである．そのため，別途**実験 2** を設け，箇条書きの各項目が正しい節種別となるかを確認した．

3.3.1　適用対象

本論文では，実際に企業にて使用された車載システムの要求仕様書を使用する．しかし，解析ツールに適用するためには要求仕様書を修正する必要がある．具体的には，3.2.2 の箇条書きにて示した制約を，要求仕様書が満たしている必要がある．3.2.2 の制約を満足するため，名古屋大学大学院情報学研究科の学生 1 名により，下記の修正が加えられている．

また，ツール開発者によって，制約が満たされたか，および一貫した修正が加えられたかが確認されており，もし確認結果が不充分である場合には，修正作業実施者である学生は指摘箇所について再度修正を行う．これを繰り返す．この結果，文単位に分割されたテキストファイルの数は 243 となった．

3.3.2　実験手順

実験 1 は，以下の手順で実施された．

手順 1 先行研究にて開発された解析ツールを実行する．

手順 2 解析ツールが分類できない節は，条件節であるかを著者が判定し，解析ツールに入力する．

手順 3 解析ツールが生成した解析木の情報を確認し，著者による節の分類と比較する．

実験 2 は，上記**手順 3** から引き続き実施される．

手順 4 箇条書きの文を識別する．

手順 5 箇条書きの親の文における「以下の」を含む節種別を，箇条書きの文に機械的に割り当てる．

手順 6 手順 5 の結果を解析ツールによる分析結果とし，著者による節の分類と比較する．

実験 1 および**実験 2** における比較では，適合率 ($precision$) および再現率 ($recall$)，F 値 (F) を算出する．また，F 値は $precision$ と $recall$ の調和平均とする．正解集合を著者が分類した節の分類集合 $\mathbf{A}$ とし，解析ツールが分類した節の集合を $\mathbf{E}$ とする．解析ツールが分類した節の分類のうち正解集合に属する個数を正解数とするとき，$precision$ および $recall$，F 値は，それぞれ以下の式で定義される．

$$precision = \frac{|\mathbf{A} \cap \mathbf{E}|}{|\mathbf{E}|} \tag{3.1}$$

$$recall = \frac{|\mathbf{A} \cap \mathbf{E}|}{|\mathbf{A}|} \tag{3.2}$$

$$F = \frac{2 * precision * recall}{precision + recall} \tag{3.3}$$

3.3.3　実験結果

実験 1 の結果を表 3.1 に示す．表 3.1 中の X は不定値を表し，括弧の外は著者によるインタラクティブな操作を含む抽出結果であり，括弧の中は解析ツールが自動抽出のみによって抽出した結果である．**実験 1** の結果について説明する．表 3.1 では，著者による節の分類の数 ($|\mathbf{A}|$) および，解析ツールによる節の分類の数 ($|\mathbf{E}|$)，解析ツールによる節の分類の内，著者による分類と一致している数 ($|\mathbf{A} \cap \mathbf{E}|$) を計測し，それらから導出された各節の $precision$ および $recall$，F 値を示している．括弧の外，および括弧の内の数値は，解析ツールによる完全な自動抽出か，著者によるインタラクティブな操作が含まれているかをそれぞれ示している．インタラクティブな操作とは，解析ツールが著者に対して節の分類ができないことを通知し，著者は，条件節と処理節の組み合わせの文であるか，それとも「制約の節」であるかを判断し，解析ツールに判断結果を入力する．今回，該当する文は 3 文あり，それらはすべて制約の内容であると判断した．現状，解析ツールは条件・処理・定義の 3 つの節のみにしか分類できないため，これらの制約の節は解析ツールによって定義節に分類される．

表 3.1: **実験 1** の結果

対象	$\|\mathbf{A}\|$	$\|\mathbf{E}\|$	$\|\mathbf{A} \cap \mathbf{E}\|$	*precision*	*recall*	*F*
条件節	112	113(113)	112(112)	0.991(0.991)	1.000(1.000)	0.996(0.996)
処理節	85	86(86)	85(85)	0.988(0.988)	1.000(1.000)	0.994(0.994)
定義節	33	32(29)	32(29)	1.000(1.000)	0.967(0.879)	0.985(0.935)
未分類	0	4	0	0	X	X
合計	230	235(232)	229(226)	0.974(0.974)	0.996(0.983)	0.985(0.978)

表 3.2: **実験 2** の結果

対象	$\|\mathbf{A}\|$	$\|\mathbf{E}\|$	$\|\mathbf{A} \cap \mathbf{E}\|$	*precision*	*recall*	*F*
条件節	195	191	191	1.000	0.979	0.990
処理節	7	3	3	1.000	0.429	0.600
定義節	0	0	0	X	X	X
合計	202	194	194	1.000	0.960	0.980

また，本実験においては分類不明な節が存在していた．解析木の情報を持つファイル中には，特定の形態素列が節であるという情報と，その節の種別が明記されている．未分類とは，節の種別が当該ファイルに記載されていない節である．未分類は 4 つ存在しており，それらはすべて箇条書きの文であり，かつ条件節と処理節の組み合わせからなる節であった．

次いで，**実験 2** について述べる．結果を表 3.2 に示す．表 3.2 中の X は，不定値を表す．表 3.2 においても，著者による節の分類の数 ($|\mathbf{A}|$) および，解析ツールによる節の分類の数 ($|\mathbf{E}|$)，解析ツールによる節の分類の内，著者による分類と一致している数 ($|\mathbf{A} \cap \mathbf{E}|$) を計測し，それらから導出された各節の *precision* および *recall*，*F* 値を示している．**実験 2** の結果は，インタラクティブな操作に対する依存はない．本実験においては，数式も箇条書きと分類されるが，解析ツールの制約上，数式は解析対象外であるため，今回の結果に含めていない．

3.4　考察

3.4.1　RQ1: 節の分類精度

実験 1 の結果から，箇条書きを除く文については，総じて非常に高い F 値を得

ている．特に，インタラクティブな操作を含む場合では，F 値は 0.98 を超えている．また，自動抽出のみによる結果でも，F 値で見れば 0.9 を超えており，良い結果を得ることができていると言える．しかし，この結果から自動抽出だけでは，軽微ではあるが分類精度が低下するということがわかる．自動抽出によって判断できないものは，条件節と処理節の連続からなる文に見えるが,「制約の文」かどうか判断することが困難なものである．すなわち，条件節に伴う文が，振る舞いではなく性質や制約を表している場合があり，条件節に伴う文が振る舞いを表していればそれは処理節であり，そうでなければ定義節と分類されるべきである．この分類を行う手法が決定できれば，精度をさらに向上できる可能性がある．

次に，**実験 2** の結果から，全体としての F 値は 0.9 を超えているが，処理節の精度が低い結果となった．これは，**実験 1** で未分類となった箇条書きの文が解析されていないことによる．これら 4 つの文は，すべて条件節と処理節の組である．そのため，手作業による抽出結果の方が条件節と処理節が各 4 つずつ多くなった．箇条書きの中に複数の節がある場合の分類規則が不足であるため，これは手法が改良される必要がある．それ以外の箇条書きは，今回対象とした要求仕様書中において，すべて箇条書きを参照する節中に「以下の」が含まれていたため，間違いなく抽出することができていた．

文および節の分類精度は，手法によって定義されていない構文が存在し得た場合，精度が低下する恐れがある．しかし，修正を加えてはいるが，243 文からなる要求仕様書に対して高い精度で節の分類ができているため，解析ツールの分類能力は非常に高いと判断する．

また，未分類となった節は，すべて実際には箇条書きであった．

以上から，**RQ1** について，以下のようにまとめる．

- **自動抽出であっても 0.9 を超える F 値によって節種別を正しく分類できる．**
- **箇条書きについては，特定のキーワードを正しい位置に含んでいることで分類できる．**
- **箇条書き中に複数の節がある場合の解析方法の検討は，今後の課題である．**

3.4.2　RQ2: 節の分類種別

まず，**RQ1** でも述べたとおり，**実験 1** の結果における未分類は節の定義が不足していたわけではなく，手法としての今後の課題である．そのため，**実験 1**，**実験**

2 において，節種別の不足はない．

実験 1 にて分類のためにインタラクティブな操作を必要とした節について考察する．先行研究における手法では，条件節，処理節および定義節の 3 種の節を定義している．条件節と処理節は基本的には組であり，定義節はそれだけで存在する．**実験 2** によって示された通り，条件節や処理節も単独で存在する可能性はあり，それは，箇条書きの各項目である．条件節は,「条件」であるため，条件に伴う動作が必ず必要であるため，箇条書きの項目である場合を除けば，必ず後ろに従属する節が必要となる．3.4.1 においても述べたが，条件節の後ろに来る節が処理節ではないことも考えられる．実際，**実験 1** においては，著者は条件節と処理節の組であるとも考えることが可能な文の最初の節を「条件節ではない」と判断している．この結果，条件節となりうる箇所も含め，解析ツールによって文全体が定義節であると分類された．この結果から，箇条書きに関しては節の分類が十分だと判断している．

次に，ユーザインタラクションが必要な節について検討する．現状，副詞的名詞 (「とき」,「際」,「場合」など) と接続助詞（「ので」,「けれど」など）の連続が節末尾に存在した場合に，条件節と処理節の組か判断ができず，ユーザによるインタラクティブな操作が必要である．例文としては，以下の場合が挙げられる．

> モジュールを起動したときのモータの初期値は「休止状態」とする。

上記の例で,「ときの」は JUMAN により副詞的名詞と接続助詞の連続とみなされる．これは，

> モジュールを起動したとき、モータの初期値を「休止状態」にする。

と書き換えることができ，書き換えると条件と処理の組に見える．また，条件節と処理節の組と分類されるが，処理節が制約の文とも見える例を考える．たとえば，以下の文である．

> センサ値が規定値を超過した場合は，動作を停止しなければならない．

この文は，条件に伴う処理とも，条件に伴う制約とも解釈することが可能である．しかし,「しなければならない」と書いてあるため,「センサの値が規定値を超えた

場合に，動作が停止していない」ことがないと確認をする必要があるという制約であり，システムとしての不変条件であるという解釈となる．しかし，実際にはこれは機能として実装される必要がある可能性もあり，もしそうであるならば，条件と処理の組であることがより明確になる書き方とすることが望ましいと考える．すなわち，以下のような書き方が望ましいと考える．

> センサ値が規定値を超過したとき，動作を停止する．

このように，書き方によって条件節に伴う処理，および条件節に伴う制約は相互に書き換え可能な場合がある．しかし，最終的に状態遷移モデルを抽出することを考えるならば，これらの表現の差は重要なものであると考える．つまり，制約であれば既述の通り，状態遷移モデルに記載されない要素となるため，処理として扱われなければ，最終的にはイベントや処理・遷移の抜けとなる可能性がある．しかし，特定の場合だけに処理として実装されるのではなく，常に成り立たなければならない不変条件であるならば，それはイベントとして表現することは困難であり，処理ではなく制約として書かれているべきである．このように，書き手が,「処理」を意識して仕様書に記載したのか,「制約」を意識して記載したのかは，解析ツールが目的としている状態遷移モデルの抽出に対して影響を与える可能性があり，また，機能として実装するかどうかの判断が困難になることがあると考える．

ここまで，処理節と制約の文の比較を述べたが，定義節と制約の文の比較についても考察する．定義節は,「状態を定義している節」と定義している．すなわち，状態遷移モデルを抽出する際には，要求仕様書中に存在する開発者が状態名として書き表したいオブジェクトが持つ値を定義している文となる．対して，制約の文は禁止状態を示していることが多く，型定義または変域の定義とは趣旨が異なり，定義節によって表現したい目的と一致していないと解釈することもできる．

現状，**実験 1** において分類できていない文は存在していないが，上記の問題があるため，定義節から制約節を分離することが望ましいと考えている．

以上から，**RQ2** を以下のようにまとめる．

- **現状，分類できていない節は存在しないが，定義節は細分化されるべきである．**
- **定義節とは性質が異なる「制約節」として新たに定義されるべきである．**
- **処理節と制約節は意識して明確に区別されている必要がある．**

- 状態遷移モデル中に現れないことから，本来条件節とも言える箇所を制約節は包含して持つ．

3.4.3　全体の考察

RQ1 の結論から，解析ツールは今回対象とした要求仕様書については，非常に高い精度で節の分類を行うことができる．また，**RQ2** の結論から，今回対象とした要求仕様書については，制約節を種別として追加すると，すべての文を節に分類することが可能となった．

3.4.2 においても述べたが，制約節として記述されているか，処理節として記述されているかは，書き手の意思を反映している可能性がある．そのため，制約として既述されているだけなのか，それとも処理としてイベントに応じて実行されるべきなのかは，節の種別を明確に表す書かれ方によって決定される．このことから，解析ツールによる対応の可否だけではなく，より高い品質の要求仕様書を作成する上でも，制約か処理かを明確にすることは重要であると言える．

精度の高い節の分類は，解析ツールが最終的に目的としている状態遷移モデルの抽出だけではなく，様々な応用が考えられる．たとえば，形式手法の記述支援に繋がる可能性がある．処理節と条件節は，そのまま形式モデルにおける対応する表現として記述でき，具体的な定義や制約が型宣言や不変条件，事前条件，事後条件，時相論理式による検査式など，要求仕様書を形式モデルに書き換える際に注視する箇所を明らかにできる可能性がある．大森らによって，自然言語からキーワードを抽出することで形式モデルの作成支援を行うツールが開発されており [15]，語句の粒度による支援が可能である．節の分類は，より大きな粒度の文の情報が得られるため，既述の通り，さらなるモデル作成支援につながる可能性がある．

また，ソフトウェアテストにおける，テストケース作成支援にもつながる．条件が明らかになるため，入力パターンを作成することが容易となり，それに対する出力（振る舞い）も対応する処理節を確認すればよい．また，非機能要求が制約節として書かれていれば，機能要求以外の要求を抽出することも容易になる可能性がある．

そのため，解析ツールにおける節の分類の評価を，以下のようにまとめる．

- 内容を確認した結果,「定義節」高い精度で節を分類することができる．
- 新たに制約節を定義することで，節の分類としては現状十分である．

- 状態遷移モデル以外の応用の可能性がある．

3.4.4　妥当性への脅威

本論文における妥当性への脅威を説明する．まず，実験で対象とした要求仕様書が，解析ツールの制約を満たすように書き換えられたものである．また，実験で対象とした要求仕様書は，箇条書きを参照する際，すべて「以下の」という表現を持っていたが，箇条書きを参照する場合には「次の」や「下に示す」など，他の表現も存在しうる．このように，ツールで解析可能となるよう対象の文書を修正しているため，もし，想定されていない記述が存在した場合には精度が低下する可能性がある．

また，実験で用いた要求仕様書は 1 つだけであるため，異なる要求仕様書を用いると結果が異なる可能性がある．

最後に，読み手が処理と制約の差において解釈を誤る場合があるか明らかではない．

3.5　関連研究

自然言語によって記述された文書から様々な情報を抽出する研究が，多数存在する．村上らは，自然言語で記述されたシステム仕様書から，試験ケースを作成するまでのプロセスを提案している [49]．彼らは，セミ形式記述を定義し，文章をセミ形式記述に変換するアルゴリズムを定義し，条件・動作同定機能を実装したツールを開発した．条件，動作は，いずれも 8 割以上の *precision* および *recall* によって抽出されている．他に，要求仕様書からドメイン固有言語（DSL）を抽出支援する手法がある [51]．彼らは，自然言語とそれに対応する DSL，および DSL の文法を機械学習し，自然言語から DSL を生成するフレームワークを提案している．また，自然言語のユーザマニュアルからソフトウェアの機能に関連する情報を抽出支援する手法が提案されている [52]．

自然言語で記述された文書の誤りを抽出する研究も存在する．辞書を用いて書き誤る可能性が高い日本語助詞の検出・修正を行う研究が存在する [53, 48]．

また，4 章および 5 章で述べたとおりであるが，C 言語ソースコードから細粒度状態遷移表（MSTT）を抽出する支援ツールの開発を進めてきた [54]．本章における解析ツールによって生成された状態遷移モデルと，ソースコードから状態遷移

表を抽出するツールの生成結果を比較することで，状態遷移モデルによって表現される範囲に限定して，要求仕様書とソースコードの整合性を確認できる可能性がある．MSTT と要求仕様書から作成した状態遷移モデルは，粒度が異なる可能性は存在する．つまり，詳細設計に相当する情報から得られた状態遷移モデルと MSTT は対応関係が求められるが，要求仕様書は詳細設計よりも抽象度が高いため，MSTT とは一致しないことが考えられる．しかし，広範な視点で見た場合に，機能が漏れているといったことを見つけることは可能であると考える．

Hattori らは，状態遷移表を用いた要求仕様の欠陥分析手法を提案している [55]．まず，彼らの用いている状態遷移表は，本章で想定している状態遷移表と比較して，行に現在のアクタの状態，列に状態遷移後のアクタの状態を記載しており，セルにはそのときの動作を示している．彼らの表現では，「現在のアクタの状態」の階層が深くはなるが，状態がアクタごとに階層化されている点で可読性が高い．

本章の支援ツールが出力する状態遷移表は，行にイベント，列に状態を持ち，セル内で次状態遷移を定義している．そのため，イベントが階層化されていない点で，確認したいアクタに関するイベントをユーザが確認することが困難となる．しかし，本章の支援ツールが出力する状態遷移表は，ある状態名に注目した場合の表現となっており，1 つの状態名に注視して漏れを探索する場合には有用だと考える．また，本章の支援ツールは，イベントを節の形式で表示し，状態を名詞または名詞句の形式で表示する．そのため，本章の支援ツールにおいて Hattori らの表現形式を採用した場合，行および列見出しが大きくなるため，可読性が損なわれる懸念がある．また，Hattori らは行および列の見出し全てを本章における状態値と等価な名詞や名詞句によって記述しているが，本章の支援ツールでは，イベントを節の形式で記述している．このことにより，要求仕様書中の文意を損なわずに表現することができ，比較的要求仕様書中の情報の欠損を低減できると考える．

以上から，本章における提案手法の欠陥検出能力と比較して，状態の組み合わせを網羅的に確認することで検出可能な要求仕様の欠陥については，Hattori らの表現および手法のほうが優れていると考え，1 つの状態に注視し，要求仕様書の文意を解釈しながら確認することで検出可能な要求仕様書中の欠陥については，本章の表現および手法のほうが優れていると考える．

3.6　おわりに

本論文では，実際に企業で使用されている組込みシステムのコンポーネントの要求仕様書を用い，先行研究において提案された手法を実装したツールに対して，節の分類精度の調査を行った．

先行研究において提案された手法を実装したツールにおける節の分類精度の調査については，今回対象とした要求仕様書では高い精度で節を抽出・分類できることがわかった．しかし，現在の節種別にてすべての文を分類できているが，定義節はさらに細分化され，制約節を新たに定義する必要があるとわかった．節の分類は，状態遷移モデル抽出だけではなく，その他のモデルや，テスト支援，要求の整理などにも活用できる可能性があり，様々な応用が可能であると考えている．今後の課題は，他の要求仕様書を与えたときの性能を調査すること，および分類精度の向上をすることである．

第4章　細粒度状態遷移表

4.1　組込みシステム開発の背景と研究目的

近年，ソフトウェアシステムのソフトウェアの大規模化や複雑化に伴い．開発プロセスの効率化やソースコード再利用性の向上に対する強い要求がある [56, 57, 58]．それに対して，高品質・低コスト・短納期というソフトウェア開発の要求がある [3]．そのため，組込みシステム開発の現場ではソフトウェアの再利用性や保守性の向上の要求がある．しかし，ソースコードのレガシー化によって，再利用や保守におけるコストが増大している現状がある [59, 60, 61]．

現在の開発現場においては，レガシーコードの設計書が存在しない場合や文書が最新版のコードに対応しきれていない場合が存在しうる [61, 62]．このような状態でレガシーコードを扱うことは，ソフトウェアの品質低下につながるか，もしくはソースコードの変更を行うことが非常に困難になる [63, 61, 64, 65, 66]．加えて，ソフトウェアの再利用手法を適用することが困難となる [64, 65, 67]．また，組込みシステムはリアルタイム制御システムであることが多いため，状態遷移の構成要素が多く存在しており [68, 63, 66, 69, 70, 71, 72, 73]，これらを網羅的に理解することは困難である [63, 66]．

しかし，開発担当者が変更になった場合や，保守が不十分であることによりソースコードの文書を利用することが困難になる場合が存在する．このように，開発者もおらず，利用が困難な文書のみが残ることによって，レガシーコードが生じる．

そのため，ソースコードのレガシー化によって，再利用や保守におけるコストが増大している現状がある．現在の開発現場においては，設計書が存在しないレガシーコードも存在しうる [74]．このような状態でレガシーコードを扱うことは，ソフトウェアの品質低下につながる．また，[74] によれば，レガシーシステムの存在によって，その刷新に多大なコストを要すると言われている．特に組込みシステムの分野では，レガシーコードは，保守性が低いソースコードであるとみなされる．すなわち，開発者は大まかにソースコードを把握することはできるが，詳細を理解することができないソースコードがレガシーコードである．

一般的に，組込みシステムはイベント，状態，処理，遷移といった状態遷移の要素を持つ．そのため，状態遷移表を用いて表現可能である場合が多い．状態遷移表とは，イベント・状態・動作・遷移からなる普遍的なモデルである．状態遷移表は，システムがある状態のときにイベントが発生すると，どのような動作が生じ，どのような状態に遷移するのかを表す．状態遷移表でシステムの振る舞いを表現することのメリットは，イベントと状態の組み合わせを網羅的に俯瞰することができる点である．その結果，ある状態において事象が発生したときの動作についての欠落を発見することができる．

本章では，組込みシステム技術協会（Japan Embedded Systems Technology Association, 以降，JASA と呼称する）[1]との産学連携によって得られた産業界の要求に基づいて，細粒度状態遷移（Micro State Transition Table, MSTT ）を定義する．組込みシステム技術協会には組込みシステム開発に携わる多くの実務者が参加しており，彼らからは以下のような要求が存在した．

- 静的解析手法に基づいて，システムレベルではなくモジュールレベルの状態遷移モデルをソースコードから抽出したい．
- 上記の技術を用いて，モデルを表現する共通の形式が必要である．

MSTT は，Walkinshaw[13] らによる関数単位によるイベントではなく，条件分岐文にもとづきイベントや状態値判定を表現する．MSTT の表現形式として状態遷移図ではなく状態遷移表を選択した理由は，すべてのイベントと状態の組み合わせを俯瞰することができるためである．そのため，状態遷移図に比べて考慮するべき項目の欠如に気づくことが容易である．そこで本論文では，組込みシステムのソースコードから，MSTT を手動によって抽出する方法と課題について述べる．

4.2　細粒度状態遷移表（Micro State Transition Table, MSTT）

本節では，本論文において提案する Micro State Transition Table (MSTT) について，ソースコードとの対応および MSTT の読み方について説明する．MSTT が優位である点についても本章で説明する．MSTT は，関数単位の解析である Walkinshaw らの手法 [13, 68] に比べて，条件分岐文単位の解析によって得られる．この粒度の

[1] http://www.jasa.or.jp

細かい視点によって，より詳細な状態遷移表を作成することができる．また，一般的な状態遷移表と比べると，イベントの実行順序を確認することができる．

4.2.1　ソースコードと MSTT の対応

例を図 4.1 に示す．まず，ソースコードと MSTT の対応について説明する．図 4.1 の右表において MSTT の例を示しており，MSTT は列に状態，行にイベントを取る．列については状態変数に指定した変数（以後，単に状態変数と呼ぶ）の取りうる値を図中から抽出している．今後，状態変数に指定した変数の取りうる値のことを状態値と呼ぶ．状態変数とは，システムの状態を表すような変数のことであり，本論文では以下の定義をすべて満たす変数であることとする:

1. 状態変数は，ソースコード中のいずれかの条件分岐文中で用いられている．
2. 状態変数は，その変数を含む条件分岐文が評価される条件分岐のスコープ内で更新される．
3. 状態変数は，有限値である．

これに加えて，ソースコード中でシステムの状態を表す意味を持つ変数である．状態に関連して，MSTT が持つ短所として，状態値が多すぎる場合に MSTT の規模が大きくなり，可読性を損なう可能性がある．図 4.1 では，これらの定義を満たす変数は *state* のみである．

次いで，行には，状態変数を除く変数についての条件分岐文をイベントとして表記する．図 4.1 には *None* というイベントが存在する．*None* は，条件によらず実行されるという意味である．

4.2.2　MSTT における表現

通常の状態遷移表は，一意なイベントと状態の組み合わせから，その時実行される処理と，次状態遷移を表形式で表現している．そして，あるセルの処理および遷移が実行されると，そのイテレーションにおいての処理はこれ以上存在しない．遷移した状態に基づき，またイベントが発生すると状態およびイベントに基づく次のイテレーションの処理が実行される．

しかし，MSTT は，状態とイベントの組み合わせが必ずしも一意ではなく，表の最初の列から最後の列までが 1 イテレーションで実行される．図 4.1 を用いて説

```
void task(void){
  int t = in1, s = in2;
  if(t == 1){
    S++;
    if( S < X1){
      if(state = S1){
        out = S;
      }else if(state == S2){
        out = 0;
        state = S3;
      }else{
        S++;
        state = S1;
      }
    }
  }
  return 0;
}
```

condition		state		
		S1	S2	ELSE
None		t = in1; s = in2;	t = in1; s = in2;	t = in1; s = in2;
t==1	None	S++;	S++;	S++;
	S<X1	out = S;	out = 0; state = S3;	S++; state = S1
None		return 0;	return 0;	return 0;

図 4.1: MSTT の例

明する．まず，図 4.1 右表の冒頭で，イベント $None$ が生じたとき，$state$ がどの状態にあるかを考える．イベント $None$ は，if(True)（条件に非依存）という意味である．この例では $state$ がどの値かに依らず，変数 t および s の初期化が実行される．次のイベントは，$t == 1$ かつ $None$ である．このときも $state$ の値に依らず s のインクリメントが実行される．ここで，以降は $state = S2$ であることとして説明を進める．そして，次のイベントである $t == 1$ かつ $s < X1$ によって，状態変数 $state$ に依存する処理が実行される．いま，$state$ は $S2$ であることを仮定しているため，$out = 0$ および $state = S3$ が実行される．最後のイベント $None$ も状態変数の値によらず同一の処理であるが，$state$ の値が S3 に遷移したため，ELSE の列に記載されている $return\ 0$ が実行される．このように，MSTT は，表の上から下に向かって継続的に読んでいく表であり，その途中で列の移動が生じうる．

MSTT が，従来手法 [13, 68] と比べて効果的な点について，説明する．そのあと，既存の状態遷移表と MSTT の比較について述べる．

まず，従来手法 [13, 68] と比べて，MSTT は粒度が小さい．このことによるメリットについて考察する．従来手法 [13, 68] は関数をイベントの単位としているのに対し，MSTT は条件分岐をイベントの単位としているため粒度が小さい．粒度が小さいことは，粒度が大きいことに比べてより詳細なモデルとなっていることを意味している．そのため，開発者は，ある抽象的なオペレーションが実際にどのように実装されているかを詳細に確認することができ，潜在的なバグを探索することが比較的容易になる．また，関数名の抽象化等による先入観の介在を低減できる．関数名をイベントとすることにより，関数名が妥当ではない場合に，先入観の介

在による解釈の誤りが生じる可能性がある．もしくは，関数名がその振る舞いを正しく表現できていない場合や曖昧な名前となっている場合には，開発者が状態遷移表を理解できない，または理解を誤る可能性がある．例えば，‘*communicate*’という関数がある場合に，その関数が内部通信なのか外部通信なのかわからない．これによって，開発者の先入観や経験則などにより，システムの性質から内部・外部のどちらかを類推することによって誤解が生じ得ることがあり，他にも送受信をどちらも含むのか，それとも送信だけ，受信だけを行うのかも関数名から理解できない問題があり，結果的にソースコードを読まないと理解をすることができない場合があると考えられる．以上より，関数名だけを抽出することによっては，コード理解の妨げになる可能性がある．換言すると，開発者は，より詳細なモデルを正確に得ることで，ソースコードの詳細を理解できる．MSTT の主たる使用目的は，レガシーコードの理解であり，ここまで述べたとおり誤解を生じる表現はこの目的に適さない．しかし，詳細化を行うことで抽象度が減少するため，ソースコード全体の概略を把握することは困難になる．このため，抽象的なモデルと組み合わせることで可読性を向上できる可能性はある．

次に，状態遷移表は状態遷移図よりも網羅的な表現をすることができる．従来手法 [13, 68] は状態遷移表ではなく状態遷移図を使用している．状態遷移図は，状態遷移表と等価であるが，見え方は大きく異なる．正常系であっても異常系であっても，考慮された組み合わせが表示されているモデルが状態遷移図であり，存在しうるイベントと状態の組み合わせを全て構造化して表記したものが状態遷移表である．状態遷移図であっても，考慮されていない状態間のエッジは表示されないが，イベントと状態の組み合わせが構造的に表現されていない．これに対して状態遷移表では，考慮されていないイベントと状態の組み合わせで生じる処理や遷移は，空欄となるため，考慮されていないイベントと状態の組み合わせを把握することが比較的容易となる．そのため，状態遷移表を使用することで，開発者は状態遷移図では気づくことのできないイベントと状態の組み合わせを発見できる可能性がある．また，他にも状態遷移表は状態遷移図と比べると長所があり，開発者は，ある特定のイベントと状態の組み合わせを探索する場合に，状態遷移図よりも状態遷移表の方が有利である．状態遷移表では見出しの行および列を確認すれば良いのに対し，状態遷移図は所定の状態を表すノードを探し，そのノードを視点とする遷移を確認して，ラベルとして表記されているイベントを確認する必要があり，特に規模の大きい状態遷移図では，ノードを探すだけであってもコストがかかる．

次いで，従来の状態遷移表と MSTT を比較する．従来の状態遷移表ではなく

MSTT を用いることによって，開発者は，イベントの評価順序を容易に把握することができる．既述のとおりであるが，イベントには状態値判定を含まないため，この順序関係について状態値判定は含まれない．例えば，図 4.1 で言えば，開発者は，はじめに $None$ が評価され，その後 $t == 1$ が評価されることを理解することができる．従来の状態遷移表は必ずイベントに対する組み合わせは一意に存在するが，イベントの評価順序を表現することはできない．イベントの評価順序がシステム理解上必要なケースもあると考えられるため，これは MSTT の長所である．さらに，イベントに対する組み合わせは一意であるとは限らないことによるメリットが存在する．それは，冗長なイベントの存在を発見することができる可能性があることである．同一の MSTT の定義に基づくイベントが複数回生じることは，コーディング上考えられる．しかし，同一のイベントを複数記述している場合，イベントを整理してまとめることができる場合がある．すなわち，MSTT を用いることによって，表中に複数存在するイベントは，処理の最初および最後（事前処理と事後処理）を除けば冗長である可能性があると，開発者は目視で容易に確認することができる．

また，ソースコード中の表現によっては，従来の状態遷移表では表現しきれない場合がある．文献 [75] では，本論文と同様，C 言語ソースコードを対象として状態遷移モデルの抽出を試みている．文献 [75] では，状態遷移抽出のために，補足情報をソースコードに追記している．この補足情報には，解析範囲や状態変数の指定を行うものである．また，状態分割ポイントの指定を行っている．状態分割ポイントは，イベントの明示的な記述である．この状態分割ポイントによて，解析範囲を決定し，状態遷移モデルを記号実行によって抽出している．すなわち，リスト 4.1 に示すようなソースコードの場合，一旦分割ポイントを挿入しなければ，条件分岐文の粒度でイベントを考える場合に，イベントの完全な独立は難しい．このことによって，逐次評価される表記をとっている MSTT でなければ表記することは，非常に困難である．

4.2.3　MSTT の長所

リスト 4.1 を用いて，従来の状態遷移表では表現が困難な場合を検討する．まず，従来の状態遷移表では，状態変数を $state$ とした場合，表 4.1 や表 4.2 のようになる．しかし，従来の状態遷移表では，リスト 4.1 を表現することができていない．表 4.1 における問題点は，‘$proc\_and\_tans\_3\_1$();’ および，‘$proc\_and\_tans\_3\_2$();’ である．ま

リスト 4.1: 従来の状態遷移表では抽出が困難な例

```
1  #include "process.h"
2  // E1,E2,S1,S2: const value
3  int state, event;
4  int cyctsk(){
5    if (event == E1){
6      if (state == S1){
7        proc_1_1();
8        state = S2;
9      }
10     else if (state == S2) proc_and_tans_1_2();
11   }
12   else if (event == E2){
13     if (state == S1) proc_and_tans_2_1();
14     else if (state == S2) proc_and_tans_2_2();
15   }
16   if (event == E1){
17     if (state == S1) proc_and_tans_3_1();
18     else if (state == S2) proc_and_tans_3_2();
19   }
20 }
```

ず，‘$proc\_and\_tans\_3\_1();$’が非表示となった．これは，初回の$event == E1$の判定の時点で，状態変数$state$が$S2$に書き換わっている．よって，2度目の$event == E1$の判定においては，必ず状態$S2$を実行するため，‘$proc\_and\_tans\_3\_1();$’は実行されえず，‘$proc\_and\_tans\_3\_2();$’に書き換わっている．また，状態変数$state == S1$のとき，かつ$event == E1$のとき，‘$proc\_and\_tans\_1\_2();$’と‘$proc\_and\_tans\_3\_2();$’が実行されている．これは，リスト 4.1 中の 10 行目の関数‘$proc\_and\_tans\_1\_2();$’の時点で変数$event$が変更されないという前提がある．しかし，変数$event$は大域変数であるため，関数‘$proc\_and\_tans\_1\_2();$’中で値が書き換わる可能性があり，関数‘$proc\_and\_tans\_1\_2();$’の次に関数‘$proc\_and\_tans\_3\_2();$’が実行されると断定で

表 4.1: リスト 4.1 から作成した従来の状態遷移表（1）

	state	
	S1	S2

event == E1	proc_1_1();
	state = S2;
	proc_and_tans_3_2();
		proc_and_tans_1_2();
		proc_and_tans_3_2();

event == E2	proc_and_tans_2_1();	proc_and_tans_2_2();

表 4.2: リスト 4.1 から作成した従来の状態遷移表（2）

	state	
	S1	S2

event == E1 && dummy == 0	proc_1_1();
	state = S2;
	dummy = 1;
		proc_and_tans_1_2();
		dummy = 1;

event == E2 && dummy == 0	proc_and_tans_2_1(); dummy = 1;	proc_and_tans_2_2(); dummy = 1;
!(event == E1) && !(event == E2) && dummy == 1	dummy = 1;	dummy = 1;
event == E1 && dummy == 1	proc_and_tans_3_1();	proc_and_tans_3_2();

きない．関数‘*proc_and_tans*_1_2();’を分析すれば，次に関数‘*proc_and_tans*_3_2();’が実行されるか断定可能である可能性はあるが，現在は組込みシステムへの適用を考えている．このため，割込み処理などの要因で非同期に大域変数の更新が生じる可能性がある．以上から，表 4.1 のように，イベントが一意になるように，すなわち同名のイベントが出現しないように，イベントをまとめることでは表現できない可能性がある．

表 4.1 における問題点を解決するべく，表 4.2 では，ダミー変数 *dummy* を導入することで，イベントを一意にしつつ，順序関係を保持できるようにした．ダミー変数 *dummy* を導入する際，$if - elseif$ 構造の最後で *dummy* の更新を行うだけではなく，$if - elseif$ 構造に対する *else* を追加して，*dummy* が必ず更新されるようにしている．しかし，実際にはソースコードに存在しない変数を追加することで順序関係を保持しているため，ソースコードとの対応を取ることが困難とな

表 4.3: リスト 4.1 から作成した MSTT

	state	
	S1	S2
event == E1	proc_1_1(); state = S2;	proc_and_tans_1_2();
event == E2	proc_and_tans_2_1();	proc_and_tans_2_2();
event == E1	proc_and_tans_3_1();	proc_and_tans_3_2();

※ 下線は状態遷移を示す.

る．加えて，条件式が複雑となり，可読性も低下しうる．

以上のことから，従来の状態遷移表では，組込みシステムのソースコードをモデリングするためには抽象度が高すぎる，または順序関係を保持しようとした場合に実際にはコードに存在しない記述を追加する必要が生じる．そのため，従来の状態遷移表ではソースコードと対応をとることができない．

ここで，MSTT による表現を検討する．リスト 4.1 から，状態変数を *state* として MSTT を作成すると，表 4.3 のようになる．

MSTT による表現では，イベントの重複は生じうるため，$event == E1$ が 2 つ生じている．しかし，これは条件式の評価順序を示しており，MSTT が順序情報を保持できることを示している．順序情報保持できているため，非同期処理が途中で実行された場合であっても，1 つのセル内の処理は継続して実行されるため，セル内の処理を実行し終えた時点での状態値判定やイベントの変化を検討すればよい．そのため，非同期処理が途中で実行された場合であっても読みすすめることができる．加えて，新たに変数やイベントの追加は必要なく，ソースコードとの対応関係もわかりやすい．また，遷移が生じた場合には，注視するべき列の移動が生じるが，この例では下線を引くことによって遷移を識別しやすいようにしており，列を移動するべき箇所として明示されている．以上により，MSTT による表現では，従来の状態遷移表では表現することが困難なソースコードを表現することができている．

4.3　ケーススタディ

本節では，実施したケーススタディについて述べる．ケーススタディを以下の 2 つを目的として，2 つ実施した．

ケーススタディ1 MSTT を用いてソースコードの理解ができるかを確認する．

ケーススタディ2 MSTT を用いて組込みシステムのソースコードを表現したときのイベントの行数や複雑さを確認する．

ケーススタディ1 は，被験者 4 名が MSTT を作成し，理解度の調査のため設問に回答した．ケーススタディ2 は，ケーススタディ1 で用いたソースコードを対象として，著者が一般的な状態遷移表と MSTT を作成し，イベント数やイベントの複雑さについて比較を行った．

4.3.1　ケーススタディ1

2 つの適用対象ソースコードに対して，MSTT を用いるケーススタディを行った．被験者の内 3 名は名古屋大学大学院情報学研究科情報システム学専攻所属の博士前期課程 1 年生であり，残りの 1 名は同大学工学部電気電子・情報工学科情報工学コース所属の 4 年生である．これらの被験者を 2 名ずつ 2 グループに分ける．被験者の概要を表 4.4 に示す．彼らは，2 つの MSTT をソースコードから作成し，作成した MSTT を確認しながら実験実施者が用意した対象ソースコードについての設問に回答する．

本ケーススタディで用いたソースコードは以下の通りである．

ソースコード1 自動車制御システムの一部を表すベアメタル（OS 無し）のソースコード（110 LoC）

ソースコード2 電子レンジの制御システムの一部を表す RTOS 向けのアプリケーションを記述したソースコード（190 LoC）

実験の前に，フェーズ 0 として，5 章にて説明する C 言語から MSTT を抽出支援するツール（以降，支援ツールと呼ぶ）を使用する場合，および使用しない場合それぞれの状態遷移表作成方法のチュートリアルを行う．フェーズ 0 でのチュートリアルの内容は，以下の通りである．

- 条件処理表の導出手順
- 状態変数の定義と状態変数を選択する目的
- 条件処理表から状態遷移表への変換手順

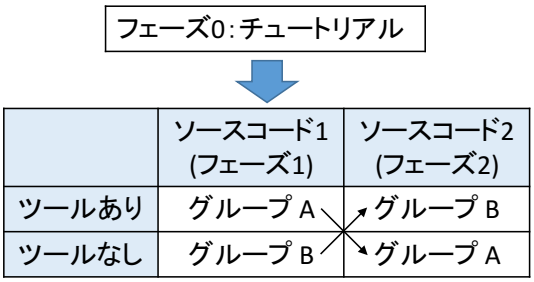

図 4.2: 実験手順

上記について，支援ツールを用いた場合と，手作業による場合のそれぞれについて，説明した．また，このとき，実験で用いるソースコードと同程度の規模のソースコードを実際に用いて，上記のチュートリアルを説明だけではなく作業を含めて実施した．チュートリアルはおよそ 40 分程度を必要とした．

その後の図 4.2 に示す 2 つのフェーズを実施する．そのため，5 章にて述べる支援ツールを用いた場合を先に実施した被験者と手作業を先に実施した被験者とが存在する．また，ソースコードに関しても，ソースコード 1 を支援ツール，ソースコード 2 を手作業で実施した学生と，ソースコード 2 を支援ツール，ソースコード 1 を手作業で実施した学生とが存在する．各フェーズにおいては，以下 3 つの手順を実施する．

【手順 1】 状態変数候補の抽出

【手順 2】 妥当と考えられる状態遷移表の作成

【手順 3】 対象ソースコードに関する理解度調査 (表 4.6 に示す設問を用意)

手順 1 および手順 2 においては，状態変数候補の選択および，作成された MSTT について，実験実施者によって確認を行っている．確認時点で，すべての指摘可能な問題を被験者に示し，被験者はこれを修正する．これを，実験実施者による指摘がなくなるまで繰り返す．実験実施者はあらかじめ正解を用意しており，手順 1 の結果については，完全に正解と一致しているか，手順 2 の結果については，同等の MSTT が作成されているかを基準として指摘している．

図 4.3 は，状態変数を md としたときのソースコード 2 を表す MSTT である．開発者は，MSTT から，「状態が 0 のときに，stp_b が押下されると，周期ハンドラが起動する」ことを読み取ることができる．また，表 4.6 に，MSTT 抽出後の設問とその回答の結果を，表 4.7 に修正回数および，作成に必要とした時間を示す．表

表 4.4: 被験者の C 言語使用経験

	C 言語の使用経験			備考
	研究	講義・演習	合計	
A1	2 年	4 年	5 年	
A2	なし	3 年	3 年	
B1	なし	3 年	3 年	学部生
B2	1 年	4 年	4 年	

表 4.5: 対象ソースコードの概要

	ソースコード 1	ソースコード 2
LOC	110	190
関数の数	3 個	7 個
条件分岐文	22 文	19 文
処理文	32 文	53 文
変数の数	8 個	10 個
状態変数候補の数	2 個	5 個

4.6 において，表記 ** は正答を，表記 * は部分的に正解である，または部分的に欠落がある回答を，表記 F は誤答を示している．

もし被験者が状態，イベントおよびそれらの組み合わせを理解していれば，q_1 to q_4 に回答することができる．表 4.6 においては，ほぼすべての被験者が正答している．このことから，開発者は MSTT を用いて状態遷移に関わる要素を抽出できている．

次いで，もし被験者が状態，ソースコードの振る舞いや抽象化を理解できていれば，q_5 および q_6 に回答することができる．q_5 の正答は，ソースコードの振る舞いが正しく反映されて書かれており，誤答は，全く存在することのない振る舞いを書いている場合である．q_6 の正答は，ソースコード 1 については自動車や自動車制御システムなどに類する回答，ソースコード 2 については電子レンジとしている．

これらの設問に回答する際，ユーザは 5 章にて説明する，支援ツールを用いている場合がある．グループ A は，ソースコード 1 において支援ツールを用いており，グループ B は，ソースコード 2 において支援ツールを用いている．支援ツールによる抽出は，手作業による MSTT の作成と比べ，ソースコードを読む時間が

		md	
		2	![2]
void cyc_1sec()	if(tmp >= 40)	ss = 0; md = 0;	N/A
	else if(tmp >= 80)	N/A	ss = 0; md = 0;

		md	
		0	![0]

void opr()	if(attm_b == 1 && ss	md = 1;
		sta_cyc(C1);
		turn_tlb();
			N/A

	else if(attm_b == 1 &	md = 1;
		sta_cyc(C1);
		turn_tlb();
			N/A

	else if(kt_b == 1)	md = 2;
		temp = gettmp();
		sta_cyc(C1);
		turn_tlb();
			N/A

	else if(stp_b == 1)	N/A	md = 0;
			stp_cyc(C1);
			stp_tlb();

図 4.3: ソースコードから生成した MSTT

短くなる．すなわち，一方のソースコードにおいてはソースコードを読む時間が長く，他方のソースコードにおいてはソースコードを読む時間が短くなる．読む時間が長くなる，または短くなるソースコードは，グループ A とグループ B で異なる．この状況で，設問に対する正答率は，全体的に高い結果となっている．そのため，必ずしも MSTT の導出過程がソースコード理解を支援しているわけではなく，MSTT 自身もソースコードの理解支援につながっている可能性がある．

これらの結果から，MSTT を導出する過程，または MSTT によってソースコードの理解につなげることができた．

最後に，妥当性への脅威について説明する．まず，被験者のうち 3 名が実験実施時点において博士前期課程在学している学生であり，残りの 1 名が工学部 4 年生であることによる，組込みシステムに対する理解度の差が存在することである．しかし，工学部所属の学生は，この実験実施時点で記号実行に基づく状態遷移表抽出支援の研究に従事しており，組込みシステム向けの C 言語から状態遷移表を抽出することについて予備知識がある．博士前期課程在学中の 3 名の学生については，モデルベース開発による組込みシステム開発の研究に従事している学生が 2 名と，組込みシステムの要求仕様書の分析の研究に受持している学生が 1 名である．いずれの学生も，組込みシステムに関する基礎知識がある．これらのことから，4

名全員が組込みシステムに関する知識を持っているため，大きな程度の差は無いものと考える．

次いで，対象としたソースコードが 100 行から 200 行程度の，比較的規模の小さいものであることである．そのため，MSTT を用いなくともソースコードの理解をすることができた可能性がある．しかし，5 章において述べるが，支援ツールを用いる場合には，ソースコードを読む時間は 5 分以下である場合もある．被験者の能力次第ではあるが，5 分でソースコードの分岐や処理をすべてを覚えることは困難であると考えている．加えて，設問に回答する際，被験者はソースコードをみておらず，MSTT のみを見て回答している．そのため，MSTT を用いてソースコード理解ができた可能性は高いと考えているが，確実にこれを論じるためには，さらに評価を行う必要がある．加えて，正答率が高いことから，ソースコードを MSTT のみで理解できたと断言できないとしても，理解支援につながっていることは言えると考えている．

4.3.2　ケーススタディ 2

ケーススタディ 2 は，ケーススタディ 1 で用いたソースコードを対象として，著者が一般的な状態遷移表と MSTT を作成し，イベント数やイベントの複雑さについて比較を行った．対象としたソースコードの情報は表 4.5 のとおりである．

このとき，一般的な状態遷移表は 4.1 にもとづき，dummy などは用意していない．ただし，今回対象としたソースコードは if - if の構造を持たず，if - else if - else の構造に基づくものである．そのため，else if の条件式が評価される前に，1 つ以上の条件式が評価されている．これらの事前に評価されている条件式が偽であることを表中に表現することとした．

表 4.8 にケーススタディ 2 の結果について，イベント数，およびイベントの複雑さを示す．イベントの複雑さは，イベント中に存在する，論理積（&&）および論理和（||）の数とする．文献 [76] においては，連言節の数を減少することによって，条件式を簡単化すると述べている．本稿では，選言標準形などの形式に標準化していないため，論理積および論理和の数をもって複雑さとする．

以上の結果から，イベントの数そのものは変化がないが，イベントの複雑さは減少することがわかった．ただし，今回は if - if の構造を持たないソースコードを対象としており，MSTT では，このような構造を持つソースコードを対象にするとイベントの数が減少するか，複雑さが減少することが考えられる．

妥当性への脅威として，試行しているソースコードの数が少ないことがある．そのため，

4.4　関連研究

Wasim ら [63] は，コンコリックな手法に基づく状態機械モデルの抽出を試みた．彼らの研究と本研究の動機は非常に近く，開発者に対するコード理解の支援を行うことである．彼らの研究においては，状態変数の選択を開発者が行う．コードの規模や複雑度が大きいことによってモデルの可読性が著しく損なわれることから，状態変数が用いられている．

Walkinshaw ら [13] は，ソースコードを使用して記号実行を行い，その結果から有限状態機械を生成している．この研究では，状態はあるデータと制御イベントの組み合わせによって定義される．また，状態遷移はシステムの機能を実行した場合，すなわち関数呼び出しを行ったときに生じる．さらに，Walkinshaw ら [68] は，トレースログを使用して動的解析による状態遷移図の抽出を試みた．彼らの提案手法では，イベントや状態は関数呼び出し単位であり，比較的巨視的なものであった．これらの巨視的な状態遷移図は，レガシーコードの全体像を把握する上では非常に有用な手法である．しかし，レガシーコードをブラックボックスとして扱わないようにするための手法としては，詳細なコードの振る舞いを知ることができない．加えて，動的解析による抽出であるため，実行頻度が低い処理を見落とす可能性がある．

状態遷移モデルの生成は，そのほとんどが実行時情報に基づくものである [68, 77, 78, 79]．本研究で提案している MSTT はレガシーコードを対象とした，静的情報のみに基づく導出手順である．また，レガシーコードにはテストコードやテストケースなどテストに関する成果物や文書が古いもしくは欠如していることが考えられる [74]．そのため，本研究で対象としているレガシーコードに対して，実行時情報を必要とする手法を適用することは現実的ではないため，MSTT を用いることの優位性が存在する．

4.5　今後の課題

MSTT は，現状では考慮できていない C 言語の表現が存在する．まず，全ての条件分岐文における条件式の表記に対応していない．条件式が関数である場合や，

論理和表現の場合などの場合の表記法を定義する必要がある．

組込みシステム，中でも車載システムにおいては，数多くの ECU (Electronic Control Unit) が使用されており，メニーコア環境を構築している場合も存在する．もしくは，近年では 1 つの ECU 上に仮想機械を構築し，仮想機械を切り替えることで車載アプリケーションを切り替える場合もある [80]．これらのシステムに対して MSTT を用いて表現できるか，検討する必要がある．また，今回の例では示していないが，割込みハンドラなどの非同期な処理は，ハンドラ名をイベントとして，表現することが可能である．

最後に，可読性の向上が挙げられる．MSTT は粒度の小さい状態遷移表であるため，表の規模が，イベントを関数単位とするような状態遷移表と比べて大きくなりがちである欠点を持つ．

4.6　おわりに

本章では，条件分岐文単位でイベントや状態を表現する状態遷移表である MSTT を提案した．MSTT を利用することで，従来の状態遷移表では，表現することが困難なソースコードや，煩雑になるような表現を避け，順序情報を保持したまま状態遷移モデルに表現できる．また，今回ケーススタディで用いたソースコードを MSTT によって表現すると，イベントの数は減少しなかったが，イベントの複雑さを低減することはできた．具体的には，一般的な状態遷移表中のイベントと比べて，MSTT のイベント全体の論理演算子（&&，||）の数が 2 割程度に減少した．

今後の課題は，表現上の限界について議論を深めることと，ケーススタディにおける対象ソースコードの追加や，被験者の増員，MSTT の理解性に関するさらなる調査である．

表 4.6: MSTT を用いたソースコードに含まれる状態遷移や内容に関する説明

	Source code 1	*	**	F
q_1	変数 spd が増加する場合の状態とイベントの組み合わせを答えよ.	3	1	0
q_1	変数 spd が減少する場合の状態とイベントの組み合わせを答えよ.	4		0
q_3	状態変数がゼロである場合，処理を含むすべてのイベントを答えよ．.	3	0	1
q_4	イベント accl == ON が生じた時，すべての処理を含む状態を答えよ.	2	1	1
q_5	このプログラムの振る舞いの概要を述べよ.	2	1	1
q_6	このプログラムは日常生活で使用する製品を表している．それは何か.	4	0	0

	Source code 2	**	*	F
q_1	いつ，周期ハンドラ 1 は起動するか.	2	1	1
q_2	いつ，周期ハンドラ 1 は修了するか.	4	0	0
q_3	状態変数が 2 のときに，処理を含む全てのイベントを答えよ.	3	1	0
q_4	イベント stp_b == 1 が発生したとき，処理を含む状態を全て答えよ.	4	0	0
q_5	このプログラムの振る舞いを述べよ.	3	0	1
q_6	このプログラムは日常生活で使用する製品を表している．それは何か.	3	0	1

表 4.7: 手作業による実験結果

被験者	修正回数	**実験手順 1**	**実験手順 2**	合計
A1	1 回	701 秒	4,750 秒	5,451 秒
A2	2 回	650 秒	5,808 秒	6,458 秒
B1	4 回	2,118 秒	3,010 秒	5,128 秒
B2	4 回	1,924 秒	5,379 秒	7,303 秒

※グループ A の 2 名はフェーズ 2 において支援ツールを使用せず，グループ B の 2 名はフェーズ 1 において支援ツールを使用しなかった．

表 4.8: 対象ソースコードの概要

	ソースコード 1		ソースコード 2	
	従来の状態遷移表	MSTT	従来の状態遷移表	MSTT
イベント数	3	3	6	6
イベントの複雑さ	6	1	30	4

第5章　細粒度状態遷移表抽出支援ツール

5.1　細粒度状態遷移表抽出支援ツールを開発する動機

4 章で述べた通り，細粒度状態遷移表（MSTT）を手作業によって作成することは，難度が高く，多大な時間を要する．そのため，自動抽出または抽出支援を行うツールによって，MSTT の作成を支援したい．

組込みシステム技術協会では状態遷移設計研究ワーキンググループの活動の一環として，組込みソフトウェアのソースコードから手作業で MSTT を抽出するための手順を提案してきた [12]．しかし，開発現場のソースコードに手作業で適用するためには，多大な時間を要するという課題があった．

本論文では，組込みソフトウェアから MSTT を抽出する作業を支援するツールを実装することで，MSTT の抽出にかかる時間を短縮した．また，MSTT 抽出支援ツール（以降，支援ツールと呼ぶ）の効果を適用実験を通して確認した．

以降，**5.2** 章ではツールとして実装する MSTT 抽出手法の概要を説明する．**5.3** 章では MSTT 抽出手法のツール化について述べる．**5.4** 章では支援ツールを使用する場合および使用しない場合それぞれの MSTT 抽出実験を行った結果を示す．**5.5** 章では **5.4** 章における結果の考察や，支援ツールの限界について述べる．**5.6** 章では本研究の関連研究について述べ，**5.7** 章で本論文をまとめる．

5.2　細粒度状態遷移表抽出手法

本章では，C 言語ソースコードを入力とした細粒度状態遷移表抽出手法（以降，抽出手法と呼ぶ）を説明する．状態遷移表の形式については，4 章で説明した細粒度状態遷移表の定義を用いる [12]．

抽出手法における入力，その中間生成物，および MSTT を図 5.1 に示す．抽出手法の手順は，以下の通りである．

```
#define S1 1
#define S2 2
#define S3 3
unsinged char state, cnt, out;
void task(void){
  int t = in1, s = in2;
  if(t == 1){
    cnt++;
    if(state == S1){
      out = cnt;
    }else if(state == S2){
      out = 0;
      state = S3;
    }else{
      if(out == 0)
        state = S1;
    }
  }
}
```

(a) 入力ソースコード

条件			処理
無条件実行			t = in1; s = in2;
t==1	無条件実行		cnt++;
	state==S1		out = cnt;
	state==S2		out = 0; state = S3;
	else	out==0	state = S1;

(b) 条件処理表の例

条件	state		
	S1	S2	ELSE
無条件実行	t = in1; s = in2;	t = in1; s = in2;	t = in1; s = in2;
t==1	cnt++;	cnt++;	cnt++;
	out = cnt;	out = cnt;	out = cnt;
	out = 0; state = S3;	out = 0; state = S3;	out = 0; state = S3;
	state = S1;	state = S1;	if(out == 0) state = S1;

(c) MSTT への変換

イベント	state		
	S1	S2	ELSE
無条件実行	t = in1; s = in2;	t = in1; s = in2;	t = in1; s = in2;
t==1	cnt++;	cnt++;	cnt++;
	out = cnt;	out = 0; state = S3;	if(out == 0) state = S1;

(d) MSTT の例

図 5.1: 入力ソースコードに対する中間生成物と作成された MSTT

(手順 1) ユーザは支援ツールにソースコードを入力し，支援ツールによって条件処理表を作成する．

(手順 2) 支援ツールは，ソースコードから状態変数の定義を満たした変数を抽出し，ユーザに提示する．

(手順 3) ユーザは**手順 2** において示された変数の一覧と，条件処理表から MSTT において状態として使用する変数（状態変数）を選択する．

(手順 4) 支援ツールは，ユーザによって選択された状態変数に対する MSTT を抽出する．

上記の手順について，**5.2.1** 節以降にて説明する．

5.2.1　条件処理表の作成

まず，条件処理表の概要を説明する．条件処理表とは，ある条件において実行される処理を関数ごとにまとめた表である．ソースコード中の，ある条件分岐文とそれに対応する処理を，条件分岐文の階層構造にしたがって表現した条件と処理の対応表である．

本論文においては，MSTT を抽出するための中間状態として，支援ツールが条件処理表を作成する．図 5.1(a) を入力としたときの条件処理表の作成例を図 5.1(b) に示す．条件処理表は，図 5.1(b) のように条件部と処理部に分割される．

条件部は，ソースコード内の関数および条件分岐文 (if，elseif，else および switch case) から作成される．条件部について，ソースコードから条件処理表に変換するための変換手順を説明する．まず，if および elseif については，その条件式を条件部に記述する．この例題においては else の場合は，表に else と記述することとする．switch については，switch(var) において指定された変数 var と，case num: のようにラベルで指定された値 num を用いて，表に var == num のように記述する．ラベルが default の場合については，else と記述する．処理部は，ある条件部に対応する処理を記述する．

5.2.2　状態変数の選択

支援ツールによって条件処理表を作成した後，支援ツールは条件処理表から状態変数候補を抽出し，ユーザが全ての状態変数候補の中から MSTT で表現したい状態変数を選択する．抽出手法では，状態変数を 1 つだけ選択する場合を考える．

支援ツールは，この状態変数の選択を支援するべく状態変数として選択可能な変数の集合である状態変数候補をリストアップする．状態変数候補の定義は，以下のとおりである．

- 分岐条件に使用されている．
- 取りうる値は有限個である．
- その変数をもつ条件式の内部のいずれかで更新される．

ここでいう有限個とは，高々10 種類までとする．10 種類に制限している理由は，後のツール実装において Excel VBA を使用しており，Excel VBA を用いて表を描画するために時間がかかるため，上限を定めている．状態変数候補の中からただ 1 つ選ばれた変数が状態変数となる．たとえば，図 5.1(b) の条件処理表における状態変数候補は，変数 state である．変数 state は，条件式に含まれており，かつそのスコープ内で値が更新されているため，state を状態変数として選択することができる．

ここで，図 5.1(d) について，処理文の列 **P**，条件分岐文の列 **C**，遷移の列 **T**，動作の列 **B** およびイベントの列 **E**，状態変数 $v = state$ とすると，

$\mathbf{P} = [\text{“}t = in1;\text{”}, \text{“}s = in2;\text{”}, \text{“}cnt{+}{+};\text{”}, \text{“}out = cnt;\text{”}, \text{“}out = 0;\text{”}, \text{“}state = S3;\text{”}, \text{“}state = S1;\text{”}]$

$\mathbf{C} = [\$, \text{“}t == 1\text{”}, \$, \text{“}state == S1\text{”}, \text{“}state == S2\text{”}, \text{“}else\text{”}, \text{“}out == 0\text{”}]$

$\mathbf{T} = [\text{“}state = S3;\text{”}, \text{“}if(out == 0)state = S1;\text{”}]$

$\mathbf{B} = [\text{“}t = in1;\text{”}, \text{“}s = in2;\text{”}, \text{“}cnt{+}{+};\text{”}, \text{“}out = cnt;\text{”}, \text{“}out = 0;\text{”}]$

$\mathbf{E} = [\$, \text{“}t == 1\text{”}]$

このように表すことができる．なお，$ は “無条件実行” を表し，if($true$) と等価なものである．**C** では，ある条件分岐文の直前または直後に実行される処理文に対して，表に変換する際の深さの調整のために追加される．**E** においては，条件によらず実行されるイベントを表している．

B に添字をつけて，$\mathbf{B}_{(state==s1,t==1)}$ とした場合，$\mathbf{B}_{(state==s1,t==1)} = [\text{“}S{+}{+};\text{”}, \text{“}out = S;\text{”}]$ のように，対応する動作を表している．遷移列 **T** も同様である．

5.2.3　細粒度状態遷移表の抽出

選択された状態変数にもとづき，支援ツールは MSTT を抽出する．MSTT の抽出は，条件処理表の作成から行われる．

まず，条件処理表の処理部に対する処理を説明する．処理部において，状態変数が取る値を条件部から抽出し，その状態の数だけ条件処理表の処理部の列を複製する (図 5.1(c))．変数 state は，S1，S2 および else の 3 つの値を取りうるため，同じ処理部を 3 列作成し，それぞれの値の場合に実行されうる処理に対応させてい

く．すると，図 5.1(c) においては，状態とイベントの対の中に入力ソースコードと一致しない処理の実行や遷移が生じる．すなわち，状態列の複製によって，処理が存在し得ない状態とイベントの組み合わせであるにも関わらず，処理が存在することが問題として生じる．このような理由で不要な箇所が図 5.1(c) 中において，黒く塗り潰されている．不必要な箇所を削除し，表を整形すると，図 5.1(d) のような MSTT を得ることができる．

しかし，図 5.1(d) では存在しないが，抽出手法においては同様の値を示す状態列が複数存在する場合がある．すなわち，特にまとめることをしなければ，同様の状態値を持つ列が複数作成されることとなる．表の可読性向上のためには，これらが統合される必要があるため，等しい状態値を持つ列を統合する．加えて，状態値が完全に同一では無くとも統合可能な場合がある．統合可能な場合について，図 5.2 に示すとおり，列の統合を行うことができる．まず，完全に状態値が同じ場合，統合することができる（図 5.2 中の (1)）．次いで，最右の ELSE は，状態値 S2 のときと中央の ELSE をあわせた範囲であるため，最右の ELSE 列を，状態値 S2 のときと中央の ELSE に書き込むことができる．このように，列単位で見ると状態値が完全に同一ではないが，複数列をとったときに値の範囲が一致する場合は，統合可能である場合がある．また，処理や遷移がソースコードの記述として完全に同一な列を統合し，状態値を列見出しに複数表示することで可読性を向上できる．加えて，処理や遷移がソースコードの記述として完全に同一な列について，状態値が包含関係である場合がある．例えば，ある具体値が else に包含される場合である．このとき，else の列に具体値の列を統合することで解決が可能となる．また，else の範囲がわかりやすいように集合表現に表記を変えることで可読性を向上することおよび，複数の else が表中に現れた時であってもその値域がわかりやすくなるよう，ツールの実装時に工夫している．

5.3　支援ツール

本支援ツールの実装方針について説明する．実装方針は，以下の 3 つである．

方針 1 支援ツールは， Microsoft Office Excel (以降，Excel と呼ぶ) のマクロ有効 Excel ブックをインタフェースとして実装されている．

方針 2 実体は，Python スクリプトによってコード解析を行い，表の描画とインタフェースを VBA で実装した．

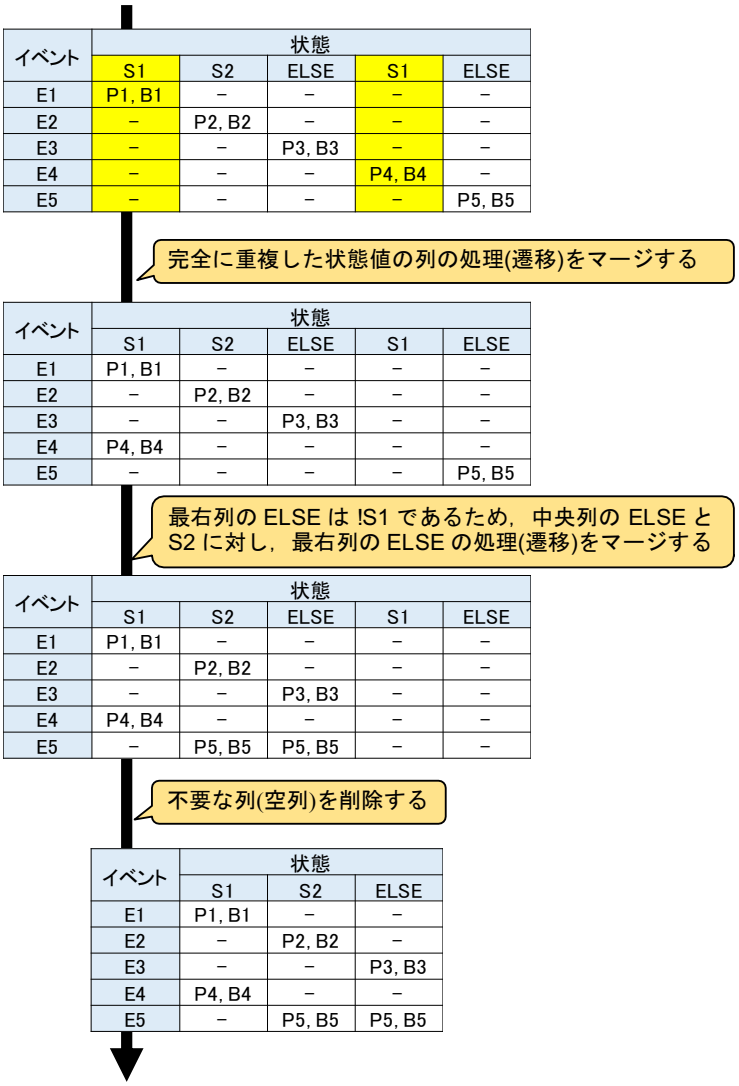

図 5.2: 列の統合

方針 1 について説明する．組込みソフトウェア開発の現場において Excel が広く採用されているため，開発現場には Excel がインストールされていることが多い．このことから，新たにソフトウェアをインストールためのコストを低減でき，かつ，Excel の ルックアンドフィールにユーザとして想定している開発者が慣れていると考えられる．

方針 2 について説明する．ソースコード解析機能などはすべて Python スクリプトによって実装されており，これを Excel VBA (以降，VBA と呼ぶ) のアプリケーションから呼び出している．これらの Python スクリプトはすべて py2exe によって EXE ファイルにコンパイルされており，これを呼び出している．表の描画

や UI は VBA によって実装されている．このような実装を取った理由を説明する．VBA は Python と比較しても実行速度が遅く，ユーザにとってストレスになると考えため，ツールのすべてを VBA で実装するのではなく，ツールの実行時間の多くを占めると考えられるソースコード解析部分を Python によって実装した．また，Python はライブラリが充実しており，言語ユーザ数も多いため，VBA と比較して今後の拡張性を担保することが可能であると考えている．それでも一部を VBA によって実装した理由は，Excel に対するバージョン依存の問題が少ないと考えられ，表の描画などについては一番親和性が高いと考えたためである．

5.3.1　支援ツールの利用手順

支援ツールの概観を図 5.3 に示す．

支援ツールは，Excel のタブ上およびフォームアプリケーション上から操作をすることができる．Excel のタブ上から操作可能にした理由は，可能な限り通常の Excel 操作から逸脱せず，ユーザが自然に利用できるようにすることを意識したためである．フォームアプリケーションを採用した理由は，状態変数の選択にあたってユーザに支援機能を提供するためである．

本支援ツールの利用手順は，以下の 3 つのステップからなる．

ステップ 1 条件処理表作成ボタンをクリックすると，以下が実行される（図 5.3 中の (A)）．

- ソースコードの整形
- 条件処理表の作成

ステップ 2 状態変数選択フォームボタン（図 5.3 中の (B)）をクリックすると対話型 UI が起動（図 5.3 中の (C)）し，ユーザが状態変数の選択を行う．

ステップ 3 対話型 UI 上の状態遷移表作成ボタン（図 5.3 中の (D)）をクリックすると，MSTT の作成が実行される．

支援ツールの利用手順について説明する．まず，ユーザは解析対象のソースコードをツールが指定するディレクトリに配置する．このとき，C ソースファイル毎に解析が実施される．

その後，支援ツールを起動し，**ステップ 1** により，条件処理表の作成を開始する．条件処理表は MSTT 抽出のための中間状態であるが，状態変数を作成するた

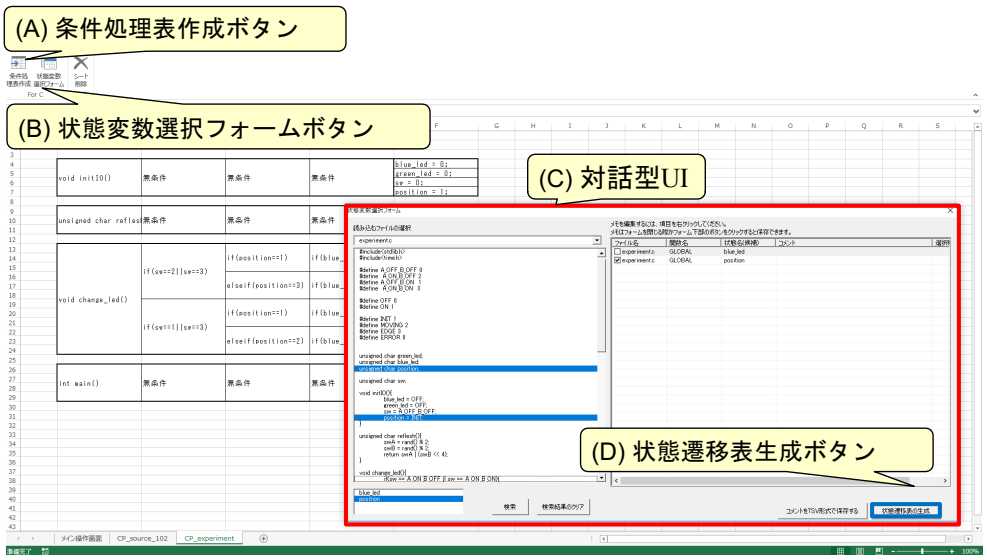

図 5.3: 支援ツールの概観

めの参考にもなる．条件式の組み合わせと対応する処理を俯瞰することができ，状態変数候補の変数がどの程度使われているかや，どのような条件式に依存しているかなどを簡単に確認することができる．

次いで，ユーザは**ステップ 2** において対話型 UI を起動する．対話型 UI に搭載されている機能については，5.3.2 にて説明する．対話型 UI には，ユーザが状態変数を選択するための支援を目的とした機能が実装されており，それらの補助機能や条件処理表を参照しながら状態変数の選択を行う．状態変数の選択の作業は手作業によって実施される場合，非常にコストの掛かる作業であり，作業に慣れていない作業者の場合，正確に状態変数候補を抽出できないこともある．手作業によって実施する場合，まず，ソースコード中に存在する全ての変数について状態変数候補となる条件を満たしているか評価する必要がある．その上で，状態変数候補 1 つ 1 つについて状態変数として選択されるべきかを考慮する必要がある．ある状態変数候補の変数が，ソースコード中のどこでどのように使用されているか，ソースコードを何度も読み返して確認する必要があり，非常に時間のかかる作業となる．この対話型 UI を利用すると，状態変数候補を自動的に表示し，かつどのように使用されているかを容易に確認することができる．

最後に，ユーザは **ステップ 3** において MSTT を作成する．すると，指定された状態変数についての MSTT が抽出される．状態変数の指定を変更して，MSTT を作り直すことも容易であり，状態変数を変化させて MSTT を抽出することで目的に合致した MSTT を生成する事が可能である．

5.3.2　支援ツールの実装

支援ツールの実装について説明する．MSTT 抽出までの各機能を以下に示す．以下の括弧内は，5.3.1 において示したステップとの対応を示している．

機能 1 ソースコードの整形 (**ステップ 1**)

機能 2 条件処理表の作成 (**ステップ 1**)

機能 3 状態変数候補の抽出と選択 (**ステップ 1**, **ステップ 2**)

機能 4 MSTT の作成 (**ステップ 3**)

上記の通り，支援ツールは 4 つの機能によって C 言語ソースコードから MSTT を抽出する．これらの 4 機能は，支援ツールの UI である Excel のマクロ有効 Excel ブック上から呼び出されることにより実行される．

始めに，**機能 1** について説明する．ここでは，GCC を利用してマクロ展開およびコメントの削除を実施する．そのため，入力するソースコードは GCC によってコンパイル可能である必要がある．GCC は C 言語コンパイラであるため，マクロ展開処理を安定して実行できると考え選択した．次に，自作の Python スクリプトによって，インデントの統一を行う．インデントの統一時，同時に中括弧の位置の統一など，ソースコードの表現を統一する．最後に，自作の Python スクリプトによって，関数や条件分岐文に対してタグ付けを行う．関数には “@@@FUNC@@@” という文字列を付与するなど，条件分岐文に対してもタグの名前を適宜変更しながら同様の方法によってタグ付けを行う．今後の拡張性を考慮し，既存のツールを利用せずに自ら実装した．

次いで，**機能 2** について説明する．まず，ソースコードの整形時に実施したタグ付けに基づき，条件分岐文や処理文の関係性を解析し，それらの依存関係を構文木として表現する．その後，表形式で表現することを意識して，構文木の深さを統一するべく，ダミーの条件分岐文として，無条件分岐文である “無条件処理” という条件を追加する．そして，Excel で表現することが容易になるよう，TSV (Tab Separated Value) 形式による表構造情報ファイルを作成する．ここまでを Python スクリプトによって実行する．表構造情報ファイルに TSV 形式を選択した理由は，C 言語中ではコンマが利用されるため，CSV 形式の場合に表記を工夫しなくてはならず，その処理を TSV 形式の場合に省略できることである．この表構造情報ファイルに基づき，VBA を利用して，Excel シート上に条件処理表を展開する．

そして，**機能 3** について説明する．状態変数候補の抽出は，条件処理表作成時に実施しておく．状態変数候補であるための条件を満たしている変数を自作の Python スクリプトによって抽出しておく．この処理は，条件処理表を作成する際，自動的に実施される．ここで，対話型 UI を起動する．対話型 UI には，ユーザが状態変数を選択する際，その支援を行うべく次の機能を実装した．

- ソースコードプレビュー機能 (図 5.4 中の (a))
- 状態変数検索機能 (図 5.4 中の (b))
- 状態変数選択機能 (図 5.4 中の (c))

ソースコードプレビュー機能は，既に作成された条件処理表と併せて，状態変数を選択する際の参考情報として必要な対象ソースコードを表示する機能である．ソースコードプレビュー機能に対して，指定した状態変数候補がどこで使用されているかをハイライトすることが可能な状態変数検索機能がある．この機能は，正規表現を利用して，指定した変数を含む行をハイライトすることができる．最後に，状態変数選択機能を用いて，ユーザがどの状態変数候補を状態変数とするかの選択を行う．現状では複数の状態変数を考えず，単一の状態変数のみであることとしているため，ただ 1 つの状態変数候補に対してチェックボックスにチェックを入れる必要がある．対話型 UI は，すべて VBA によって実装されたものである．ソースコードプレビュー機能を実装した理由は既述の通り状態変数候補がどのように使用されているかを確認するためであるが，実際の利用時には条件処理表と併せて確認できるようにしたい．そこで，フォーム実行中でも Excel シートを操作できるように設定し，ソースコードと条件処理表を同時に確認しながら状態変数候補の使途を確認できるように設計した．また，状態変数選択機能の中には，メモ機能を実装している．現在示しているソースコードでは状態変数候補が 10 個に満たないが，状態変数候補の数が多くなると各変数の用途などを記憶することが困難になる．そこで，ユーザが自由に状態変数についてコメントをメモすることができるよう，メモ機能を実装した．

最後に，**機能 4** について説明する．ここでは，条件処理表作成時に抽出された構文木と，ユーザによって指定された状態変数を用いて MSTT を抽出する．まず，Python スクリプトによって，構文木中の条件式の内状態変数を含むものを抽出し，これを状態として値を抽出する．次いで，状態と判断されたノードの親ノードにあたる条件文をイベント，子ノードにあたる条件文を処理として抽出する．ま

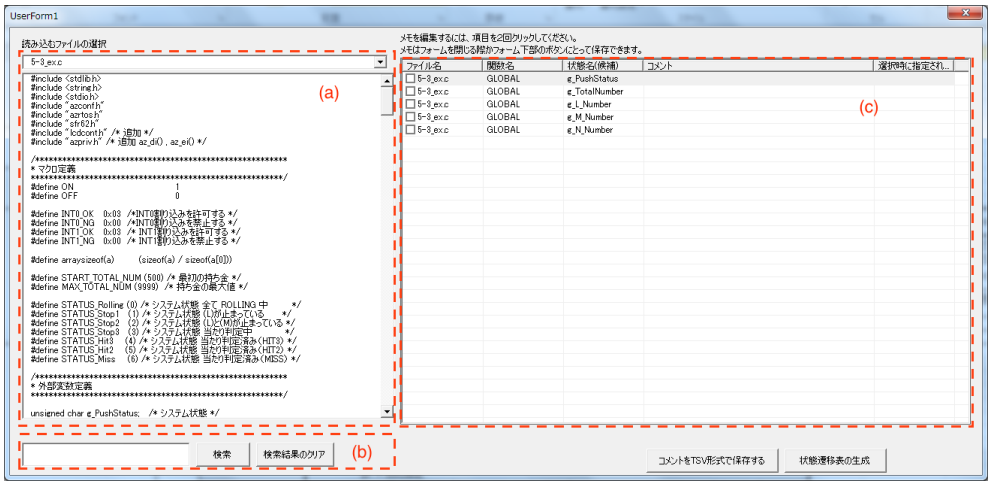

図 5.4: 状態変数ユーザインタフェース (対話型 UI)

		md	
		2	![2]
void cyc_1sec()	if(tmp >= 40)	ss = 0; md = 0;	N/A
	else if(tmp >= 80)	N/A	ss = 0; md = 0;

		md	
		0	![0]

void opr()	if(attm_b == 1 && ss	md = 1;
		sta_cyc(C1);
		turn_tlb();
			N/A

	else if(attm_b == 1 &	md = 1;
		sta_cyc(C1);
		turn_tlb();
			N/A

	else if(kt_b == 1)	md = 2;
		temp = gettmp();
		sta_cyc(C1);
		turn_tlb();
			N/A

	else if(stp_b == 1)	N/A	md = 0;
			stp_cyc(C1);
			stp_tlb();

図 5.5: 支援ツールの出力例

た，状態変数を左辺に含む処理を遷移の文として抽出する．これらの情報を用いて，MSTT の必要な場所に必要な要素を配置するような表構造情報ファイルを TSV ファイルとして生成する．最後に，VBA によって Excel シート上に MSTT を展開し，MSTT の抽出が完了する．

5.3.3　支援ツールの出力例

支援ツールの出力例を図 5.5 に示す．入力したソースコードは **5.4** 章において述べる適用実験のソースコード 2 である．状態変数を条件分岐文に含まない関数は表示していない．

5.4　適用実験

本研究においては，以下のリサーチクエスチョン (**RQ**) について検討する．

RQ1 支援ツールを利用することで，時間的コストの削減に寄与するかどうか．

RQ2 状態変数候補の抽出を正確に行うことができるかどうか．

RQ3 MSTT を正確に作成することができるかどうか．

以降，実験内容について説明しその結果を示す．

5.4.1　実験内容

本章では，手作業および支援ツールの適用実験と，支援ツールを使用することによる効果についての実験について説明する．2 つの適用対象ソースコードに対して，4 名の被験者によって，手作業および支援ツールによる適用実験を行った．被験者の内 3 名は名古屋大学大学院情報学研究科情報システム学専攻所属の博士前期課程 1 年生であり，残りの 1 名は同大学工学部電気電子・情報工学科情報工学コース所属の 4 年生である．これらの被験者を 2 名ずつ 2 グループに分ける．

具体的な実験内容を説明する．実験においては，支援ツールを使用する場合と使用しない場合の 2 フェーズがある．被験者は，グループごとに図 5.6 のような手順に従って実験に取り組む．実験の前に，フェーズ 0 としてツールを使用する場合および使用しない場合それぞれの MSTT 作成方法のチュートリアルを行う．

その後の各フェーズにおいては，以下 2 つの手順を実施する．

【実験手順 1】 状態変数候補の抽出

【実験手順 2】 妥当と考えられる MSTT の作成

まず，支援ツールを使用する場合の手順について説明する．はじめに**実験手順 1** について，5.3.1 に示す**ステップ 1** を実施して条件処理表を作成し，対話型 UI を起動後，解析対象ソースコードを指定した時点で状態変数候補が表示される．次に，**実験手順 2** では，5.3.1 における**ステップ 2** および**ステップ 3** を行い，MSTT を作成する．

これらの実験手順において，被験者の思考が介在する箇所について，被験者は理由を記述することとした．具体的には，状態変数の選択理由と，作成された MSTT が正しいと考える理由である．

次に，支援ツールを使用しない場合の実験手順について説明する．はじめに**実験手順 1** について，対象ソースコードをエディタから開く．次に，条件式で使われている変数を列挙する．そして，列挙された各変数について，取りうる値が有限の変数を列挙する．さらに，列挙された変数について，その変数を含む条件式内において値が更新されている変数を列挙する．このとき列挙された変数が状態変数候補である．

次に，**実験手順 2** の流れを説明する．まず，列挙された状態変数候補の取りうる値を変数ごとに列挙する．状態変数候補の取りうる値やソースコードを参考に，状態変数候補からシステムの状態としてもっともらしい変数を 1 つ状態変数として選択する．そして，状態変数のとりうる値を列として作成し（状態列の作成），状態変数を含まない条件式のうち，状態遷移判定の条件式を含むが，状態遷移判定の条件式よりもネストが浅いものをイベント行として作成する（イベント行の作成）．また，状態列とイベント行に対応するよう処理を記入する．ただし，状態遷移判定の条件式の中にある条件式を処理とみなす．通常，処理とは，条件式以外の記述を指す（処理セルの作成）．最後に，状態遷移を表す処理に下線を引く（遷移部分の表記上の区別）．

これらの実験手順において，使用するエディタは被験者が普段使用しているものとし，エディタにおける検索機能の使用を禁止した．支援ツールを使用する場合と同様，被験者の思考が介在する可能性がある箇所については，被験者は理由を記述することとした．

各フェーズにおいて，各被験者は**実験手順 1** および**実験手順 2** の生成物である適用対象のソースコードから抽出した状態変数候補および MSTT を実験実施者である著者に提出する．提出した各表に誤りがある場合，実験実施者が誤りを指摘し被験者が修正する．この修正作業は，被験者による各提出物が実験実施者が予め作成した解答と，レイアウトの不一致および表記は異なるが同じ意味を表していると考えられる単語や式を除き，完全に一致するまで繰り返される．一度に伝える誤りの数は，提出時点で実験実施者が発見したすべての修正箇所数である．

今回の実験において適用対象とするソースコードは 2 つある．ソースコード 1 は，自動車の動作の一部を表すプログラムであり，自動車の走行状態を示す変数が存在する．ソースコード 2 は，電子レンジの動作の一部を表すプログラムであり，現在の電子レンジの動作状態を示す変数が存在する．なお，ソースコード 2 については，リアルタイム OS 上で動作するアプリケーションプログラムを意識しており，関数呼び出しは存在しないが，周期ハンドラと 1 つのタスクからなるプログラム

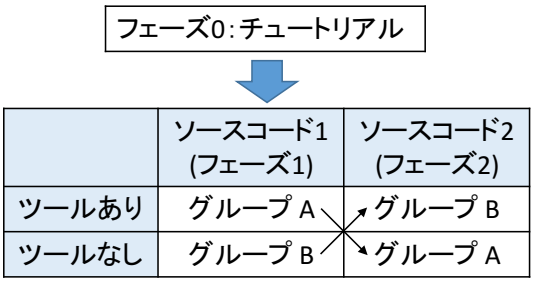

図 5.6: 実験手順（再掲）

である．また，ソースコード 1 およびソースコード 2 について，LOC はそれぞれ 110，190 であり，状態変数候補の数はそれぞれ 2 個，5 個である．

ソースコード 2 について，支援ツールが出力した MSTT を図 5.5 に示す．今回，プログラム中に else が存在しており，その else が取りうる値の範囲を明確にするため，else を!{0} のように記述した．この MSTT から，例えば，状態変数 md が 0 であり，かつ変数 stp_btn 以外の変数に依存したイベントが生じたとき，どの状態であっても sta_cyc によって周期ハンドラが起動される．

5.4.2 実験結果

4 名の被験者が状態変数候補抽出に要した時間，MSTT 抽出に要した時間およびそれらの合計と，4 名の被験者が指摘を受けてから MSTT の修正を行うサイクルを何度繰り返したかを，支援ツールを使用した場合について表 5.1 に，支援ツールを使用していない場合について表 5.2 にそれぞれ示す．

まず，支援ツールを使用した場合について説明する．MSTT 抽出作業において，支援ツールを使用した場合は修正作業が生じなかった．修正作業とは，あくまで実験実施者に MSTT を提出し，修正を行った回数を指している．加えて，同じ支援ツールを使用していることから自明であるが，抽出された状態変数候補は被験者間で一致している．加えて，選択した変数が同一のため，修正することなく同一の MSTT が抽出された．ここで，MSTT の抽出作業を観察したところ，選択可能な状態変数候補について，全ての変数について MSTT を作成し，それらすべてを見比べながら提出する MSTT の選択を行った被験者や，ソースコードを見ながら，状態変数候補の変数が取りうる値を確認して，その範囲や数を確認して状態変数の選択を行ってから MSTT を提出する被験者が存在した．

次に，支援ツールを使用していない場合について，実験実施者が指摘した項目

表 5.1: 支援ツールを使用した場合の実験結果

被験者	修正回数	**実験手順 1**	**実験手順 2**	合計
A1	0 回	114 秒	1,017 秒	1,131 秒
A2	0 回	124 秒	566 秒	690 秒
B1	0 回	206 秒	380 秒	586 秒
B2	0 回	241 秒	985 秒	1,226 秒

※グループ A の 2 名はフェーズ 1 において支援ツールを使用し，グループ B の 2 名はフェーズ 2 において支援ツールを使用した．

表 5.2: 支援ツールを使用していない場合の実験結果（再掲）

被験者	修正回数	**実験手順 1**	**実験手順 2**	合計
A1	1 回	701 秒	4,750 秒	5,451 秒
A2	2 回	650 秒	5,808 秒	6,458 秒
B1	4 回	2,118 秒	3,010 秒	5,128 秒
B2	4 回	1,924 秒	5,379 秒	7,303 秒

※グループ A の 2 名はフェーズ 2 において支援ツールを使用せず，グループ B の 2 名はフェーズ 1 において支援ツールを使用しなかった．

を説明する．表 5.2 中の，支援ツールを使用しない場合における修正回数については，すべて**実験手順 2** における繰り返し作業である．**実験手順 1** において，状態変数候補の抽出が仮に間違っていたとしても，抽出された変数の中に本実験において正解と定める状態変数が含まれていれば指摘しないこととした．実際に，手作業においては状態変数候補であるための 3 つの定義をすべて満たした変数を正しく抽出することができた被験者はいなかった．すべての被験者において，3 つの定義を満たしている変数が不足している結果となり，必ず見落としがあった．しかし，今回，被験者 B1 を除き，状態を表している変数を必ず 1 度で抽出することができていた．MSTT 作成時の修正作業について，実験実施者が指摘した項目を以下に示す．

- ソースコードと MSTT の対応がとれていない．
- 本来，MSTT に表記するべき関数が書かれていない．
- 状態変数が全く出現しない箇所についても MSTT を作成している．

5.5　考察

5.5.1　リサーチクエスチョンに対する考察

まず，**RQ1** について検討する．支援ツールを使用した場合および使用しない場合の時間に関して，被験者 4 人分の結果を用いて，t 検定を実施した．その結果，p 値は 0.0000593 となったため，有意水準 5%において有意差がある．そのため，支援ツールを使用することによる時間的コストの削減に関する効果があることが確認された．

次いで，**RQ2** について検討する．実験の過程から，支援ツールを使用していない場合，被験者は状態変数候補を正しく抽出することができなかった．そして，支援ツールにおいて抽出された状態変数候補は，必ず状態変数候補であるための 3 つの条件を満たしたものであった．そのため，支援ツールは正しい状態変数候補の抽出のために有用であることが言える．

そして，**RQ3** について検討する．MSTT の実験実施者による修正回数を見ると，支援ツールを使用していない場合は最低でも 1 回，正しい MSTT を抽出するための修正作業を実施している．そのため，支援ツールを使用することにより，ソースコードと MSTT の対応や関数の不足といった，見落としなく MSTT を正しく作成できている．しかし，支援ツールを使用している場合，支援ツールには表現上の問題がある．それは，else の範囲を同定できないことである．

そのため，else の範囲に誤解がないよう，支援ツールでは else が存在する場合，同じ状態値でも状態値列を統合せず複数の列を表示する．具体例を用いて説明する．状態変数が取りうる値を V とし，$V = \{0, 1, 2, 3, 4\}$ であると仮定すると，if (state == 0){;} に対する else は $\{1, 2, 3, 4\}$ である．また，if (state == 0){;} else if (state == 1){;} に対する else は $\{2, 3, 4\}$ である．このように，同じ else であっても，その値域が異なる場合が存在する．そのため，ラベル else の左の列に存在する値を確認することで，else の範囲を示すこととした．この問題は手法の限界から生じる表現上の問題である．

5.5.2　手法の正しさ

本章では，MSTT を生成する手法を実装・提案している．手法の正しさについて，以下，3 つの観点に基づいて考察する．

観点 1 条件処理表の作成

観点 2 状態変数候補の抽出および状態変数の指定

観点 3 MSTT の作成

条件処理表の作成

条件処理表は，MSTT 生成の抽出のための中間出力であるが，条件処理表を用いることで，ソースコードの構造が整理することができる．この条件処理表は条件分岐文とそのネストに基づいて構造化されればよく，本ツールの実装にあっては，正しく条件処理表を抽出できている．しかし，現状実装されていない構文は存在しており，5.5.3 にて述べる通り，ループ文を条件分岐文とはみておらず，処理と判断される．ループ文は，手作業においても MSTT 上での表現方法を定義しておらず，5.5.3 にて述べる通り，変換そのものは可能である．しかし，ループ文を最終的にイベントとして扱うことを変換ルールとした場合，MSTT の読み方である,「上から下に読む」というルールの一貫性が損なわれる．そのため，現状は MSTT の読み方を一貫させるべく，ループ文を条件としないこととしている．

以上から，限界としてループ文などの記法があるが，単純な分岐構造に基づく記述であれば条件処理表を生成することができており，手法として正しいと考える．

状態変数候補の抽出および状態変数の指定

状態変数候補の抽出については，条件文で使われており，かつそのスコープ内で更新されている変数であるものを全て抽出し，提示している．状態変数の定義のうち，上記 2 つを満たしている変数は現状のテストコードにおいては正しく抽出できている．

しかし，その変数が取りうる値の数か有限個かどうかについて，厳密に数えておらず，MSTT の生成の段階での列数によって判断をしている．そのため，実質，この段階では状態変数の取りうる値が有限個かどうかについては確認をしていない．この実装については，Excel の列数の制約であり，手法としては仮に 32bit 整数型の 1 つ 1 つの値について評価されたソースコードであっても表に表現することは可能である．そのため，実装としては警告をあげることにとどめている．

以上から，有限の値であるかどうかについて，厳密な確認をこの段階では実施していない．Excel の表現能力であり，イベントのネストの深さにも依存する．そ

のため，有限である値の数をこの段階において厳密に定義できない．この制約を除けば，条件分岐文で使われており，そのスコープ内で更新されている変数を正しく見つけることはできているため，手法として正しいと考える．

MSTT の生成

条件処理表において条件だと判定されている文については，すべてイベントまたは状態値として抽出できており，対応する処理や遷移も正しく抽出できている．

しかし，現在のツールでは，5.5.3 にて述べる通り，else の範囲を同定することが困難である．そのため，状態変数に関する分岐文のうち同一のものをまとめず，すべて列に表示することとしている．

また，今後議論が必要である表現は，比較演算子の内，論理積（&&）および論理和（||）である．特に論理和については，C 言語において論理和で複数の条件式が接続されている場合，最初に出現した条件式が真であれば，それ以降の項は評価されない．そのため，仮に else の範囲が同定できたとしても，状態値をまとめることが困難となる．

以上から，else の値の範囲を同定できないために，同一の状態値をまとめることができていないが，条件処理表において条件だと判定されている文については，すべてイベントまたは状態値として抽出できており，対応する処理や遷移も正しく抽出できているため，手法として正しいと考える．

5.5.3　ツールの限界

抽出手法の目的は，組込みソフトウェアにおける状態変数に対する MSTT の出力である．現状では変数に着目しており，リアルタイム OS 上で動作するアプリケーションにおける，セマフォをはじめとしたオブジェクトの状態遷移に対する MSTT を対象外としている．

5.5.1 においても説明したとおり，現状の提案手法では else の値の範囲を同定することができず，それによって表現上の問題が生じる．

現状の抽出手法では，while 文と for 文の解析を行っていない．しかし，for や whlie の条件式を条件分岐文として扱い，そのスコープ内の処理を処理文として扱うことで，if，else などと同様の変換が可能である．

次に，支援ツールにおける限界について考察する．現在の支援ツールが対応しているソースコードは，今回の適用実験において用いたソースコードのように，簡

単な構造のものだけである．そのため，対応していない構造が存在する．具体的には，goto 文や関数ポインタなどには対応していない．また，今回考慮していない C 言語における変数値の更新方法として，ポインタ渡しがある．

現状では，単一の状態変数に対する MSTT の抽出を行っている．そのため，複数の状態変数を選択した場合の MSTT の抽出は，今後の課題である．状態列を入れ子構造にすることで解決できるが，ソースコードの解析が複雑になると予想される．

また，関数のインライン展開についても，検討をするべきである．現状の支援ツールは，関数のインライン展開に対応していない．そもそも，関数のインライン展開を行うべきケースとそうではないケースがあると考えられるため，自動展開ではなく，ユーザが指定できるような仕組みの検討を進める必要がある．

5.6　関連研究

組込みシステムがネットワークに接続されることが一般化され，組込みシステムの複雑化・大規模化が問題となってきている [3, 16]．複数の組込み機器を通信ネットワークで接続したサービスを実現するためには，それぞれの組込み機器が持つ膨大な数の状態と様々な処理を理解しながら開発を行う必要がある [16]．このような複雑なサービスの開発において，状態とそれに対応する処理を確認しやすい状態遷移表が用いられている [16, 81]．例えば，松崎はネットワークへの接続を考えられていない従来型家電機器を用いたホームネットワーク構築において，機器の動作を状態遷移表を用いて定義し，赤外線信号等を入力として扱うことで，機器の内部状態を取得している [81]．また，高橋と津田が提案する医薬品製造設備のための制御ソフトウェア開発手法において，シーケンス制御部分は状態遷移表を用いて記述される [82]．

中島らは状態遷移表を用いて記述された要求仕様に基づいて，組込みソフトウェアの自動試験を行う方法を提案している [83, 84]．MSTT はソースコードから作成されているのに対し，中島らによる状態遷移表は要求仕様を表現するために作成される点が異なる．しかし，対象として組込みシステムを想定している点は同一であり，文献 [83] において述べられている量産型組込みソフトウェア [84] では，反応的な要求機能を持つと述べられている．反応的なシステムとは，通常は入力を待ち，入力に対して内部状態の変更と出力を決定するシステムであり，状態ベースの仕様記述法は，反応的なシステムに対する要求機能の記述に向いているとも

述べられている．この点から，組込みシステムのソースコードも状態遷移設計に基づいていることが多いと考えている．

状態遷移図の自動抽出を目的とした手法がいくつか提案されているが，そのほとんどは実行時情報を基に生成する手法である [77, 79, 78]．本研究で提案している手法はレガシーコードを対象としており，レガシーコードにはテストコードやテストケースなどテストに関する成果物や文書が古いもしくは欠如していることがほとんどである．そのため，本研究で対象としているレガシーコードに対して，実行時情報を必要とする手法を適用することは現実的ではない．Walkinshaw らは，抽象実行を用いてソースコードから状態遷移図を抽出する手法を提案している [13]．彼らの手法が抽出する状態遷移図は，メソッド呼出し文や例外処理の発生による遷移のみを扱う粒度の大きいものである．この研究では，関数呼び出しと例外などを状態遷移点として扱っている．このとき，記号実行 [85] により記号実行木を抽出し，上記の状態遷移点と記号実行における状態を見つけ，これらを対応付ける．このとき，状態遷移点に関係のある記号実行木中の処理を使用する．そして，各状態について状態遷移のための条件を，記号実行木内の抽出した状態遷移点から導出する．Walkinshaw らの手法では関数を状態遷移点としており，システム全体の状態遷移図を抽出している [85] が，本研究では状態遷移点として関数中の条件分岐を扱っており，1 つの関数に着目して粒度の小さい状態遷移表を抽出している．

さらに，Walkinshaw らは，実行トレースから状態遷移表を自動的に生成するツールを提案している [68]．動的に状態遷移図を抽出しているため，実行されなかったパスは考慮されない．機械学習を用いて状態遷移の条件を導出しているため，実際のトレースに対応していない値が閾値となる場合がある．また，Walkinshaw らは，拡張有限状態機械（EFSM, Extended Finite State Machine）の推論を試みた [86]．従来の EFSM では，モデルが演繹的ではなく，どのような流れで計算されるかを予測できなかったため，その拡張として，実行中の変数の計算の流れをモデル化した [86]．この抽出にあっては，実行トレースを利用した動的実行に基づくものであり，手法として遺伝的プログラミングを用いている．

また，オブジェクト指向言語によって記述されたソースコードに対してリバースエンジニアリングを行い，状態遷移モデルを抽出する手法がある [28, 87, 88]．[28] では，具体的な状態値と抽象的な状態値の導出について述べられており，具体的な状態値とは条件文の式であり，それを抽象化した値が抽象的な状態値である．状態遷移モデルでは，抽象的な状態値を使用しており，この抽象的な状態値を使用するために以下の 3 つが定義されている必要がある．

- 抽象的な状態値のドメイン
- 具体的な状態値から抽象的な状態値への写像
- 与えられたプログラム中のすべてのプリミティブな命令の抽象的な意味

本論文にて提案している支援ツールでは，これらの制約が定義されていなくとも状態遷移表を抽出することができる．[87, 88] では，Java のバイトコードや C++ のソースコードから記号実行を利用して状態遷移モデルの抽出を試みた．記号実行の利用については，状態やイベント数の削減を可能とする可能性があるため，今後利用を検討する必要があると考えているが，記号実行のツールに依存して入力ソースコードの制約が追加される場合を留意する必要がある．

Said らは，組込みシステム向けのレガシーソフトウェアから状態機械を抽出する手法を検討している [76]．組込みシステムに適用するべく，実務者がどのような基準で状態変数を選ぶかを実験的に明らかにした．彼らは状態機械を抽出する手法を提案したが，抽出された状態機械は規模が大きく煩雑なものとなった．彼らは，この問題を軽減するべく遷移し得ない遷移条件の削減や遷移条件の簡単化を行った [89]．

5.7　おわりに

本論文では，組込みソフトウェアを対象とした細粒度状態遷移表（MSTT）の抽出手法を提案した．MSTT 抽出手法を実装した支援ツールによって，組込みソフトウェアから MSTT を抽出する適用実験を実施した結果，MSTT を抽出することができた．加えて，手作業における MSTT の抽出に比べて，支援ツールを使用することで MSTT 抽出時間を短縮でき，MSTT の抽出に要した時間は，支援ツールを使用する場合と使用しない場合において有意水準 5%において有意差があることを確認した．また，支援ツールを使用すると，より正確に MSTT を抽出することができることを確認した．正確さについては，実験において被験者が正しく表を作成できていなければ，実験実施者である著者は修正指示を与えていた．この修正回数が，ツールを用いた場合には 0 回であるのに対して，ツール不使用の場合には平均で 2.75 回の修正を要し，最低でも 1 回の修正を必要とした．そのため，より正確に MSTT を抽出することができていると判断した．

本ツールの限界として，4 章で述べた通り，ループ構造などの表現方法が定義されていない構文からは MSTT を抽出できないこと，ポインタ解析ができないこと，

および複数の状態変数を選択した場合の MSTT を抽出できないことである．また，今回の実験では実験に用いたすべてのソースコードで MSTT を抽出できたが，条件分岐文の深さや複雑さが大きい場合に，MSTT を抽出できないソースコードが存在する可能性がある．

今後の課題として，支援ツールによって MSTT を抽出可能なソースコードの構造を増やし，状態変数を複数選択した場合の MSTT を出力可能にするなどの機能追加を含む，さらなる支援ツールの機能追加が挙げられる．

第6章　結論

6.1　まとめ

本論文では，組込みシステム開発の効率化に向け，実際に企業で使用されている組込みシステムのコンポーネントの要求仕様書を用いて曖昧表現や誤り表現の分類・調査を行い（**研究 I**），自然言語で記述された要求仕様書から状態遷移表を抽出支援するツールの節分類精度について調査を行った（**研究 II**）．また，細粒度状態遷移表の定義（**研究 III**）およびその作成に向けたツールの提案（**研究 IV**）を行った．

研究**研究 I** と**研究 II**，研究**研究 III** と**研究 IV** は密接に関連しあっている．**研究 I** によって前処理された文書に対して**研究 II** で用いた支援ツールを用い，また，支援ツールの結果をフィードバックして**研究 I** の**問題 1** などへの抽出支援に繋げられる可能性がある．次いで，**研究 III** によって定義された形式を作成支援するツール開発が**研究 IV** である．

最後に，研究**研究 I**，**研究 II** および研究**研究 III**，**研究 IV** の関連について述べる．組込みシステムの要求仕様書が設計に近い，比較的詳細な情報が記述されているものであるとするならば，直接ではなくとも，**研究 II** の支援ツールによって作成された状態遷移モデルと，**研究 IV** で作成された MSTT の 2 つを比較できる．粒度のギャップはあるが，ソースコードが仕様を満たしているかどうか確認することができる可能性はあり，これを確認することは今後の課題である．

研究 I について，組込みシステムの要求仕様書の曖昧・誤り表現の分類・調査においては，書き手にとって検出が容易な問題と，書き手にとって検出が困難な問題が存在することがわかった．また，自動検出にあっては，字句・構文解析によって検出可能であれば自動検出の可能性が高いことがわかった．検出することができた問題を分類し，その出現数をまとめた．全体を通して予備実験では 1,467 件の問題が検出された．問題の種別の内，語句・構文上の曖昧表現は 565 件存在しており最大の割合を占めていた．また，語句・構文上の曖昧表現は字句・構文レベルの解析による自動検出可能性が他の問題種別と比べて高いと考える．この問題の細分

類を検討し，6 つの細分類に対してそれぞれ自動検出手法を提案した．自動検出手法は，保守的な手法と積極的な手法の 2 つを提案し，保守的な手法では *precision* を向上するような手法を検討し，積極的な手法では *recall* を向上するような手法を検討した．その結果，保守的な手法では，*precision* が最低で 0.32，最高で 1.00，積極的な手法では最低で 0.32，最高で 0.98 となった．*precision* が低い結果となった分類は，揺らぎかどうかの意味的な解釈が必要となり，辞書やパターンマッチだけでは抽出が困難であった．辞書やパターンマッチ，自然言語処理ツールによって完全に抽出可能な問題は高い精度で抽出できた．*recall* については，保守的な手法では，最低で 0.28，最高で 1.00，積極的な手法では最低で 0.88，最高で 1.00 となった．*recall* が低い結果となった分類は，自動抽出を容易にするため，略称語句のみに辞書の登録語を絞ることや，*precision* と同様，揺らぎかどうかの意味的な解釈が必要であった．総じて，自動検出手法を実装したツールの結果は，高い *recall* を得られているが，*precision* は向上の余地がある．*recall* についても，辞書に基づく手法は高い数値を示すが，辞書を使用しない場合に数値が減少することも見受けられた．

研究 II について，実際に企業で使用されている組込みシステムのコンポーネントの要求仕様書を用い，先行研究において提案された手法を実装したツールに対して，節の分類精度の調査を行った．その結果，今回対象とした要求仕様書では高い精度で節を抽出・分類できることがわかった．具体的には，F 値が 0.9 を超えているが，*precision* は 1 となっていない．*precision* を低下させる 4 つの節は，後述する現状の支援ツールが分類できていない節であることがわかった．現在の節種別にてすべての文を分類可能だと考えているが，定義節はさらに細分化され，制約節を新たに定義する必要があるとわかった．節の分類は，状態遷移モデル抽出だけではなく，その他のモデルや，テスト支援，要求の整理などにも活用できる可能性があり，様々な応用が可能であると考えている．他のモデルにも応用できる可能性がある根拠として，このツールは，条件，処理，定義を半自動で抽出することができ，これらを構成要素として持つモデルであれば状態遷移モデルでなくとも抽出できる．また，この支援ツールの限界として，箇条書きの解析がある．現状明らかである限界は，箇条書きの項目中に，節が 2 つ以上存在する場合に解析できないことである．他に，支援ツールによる解析では，箇条書きを参照する際に「以下の」をキーワードとして参照している．特定の用語が対象の要求仕様書に存在しなければ，正しく解析を行うことが困難になると考えられる．そのため，次節において述べるとおりであるが，さらに多くの要求仕様書を入力し，他にも対

応できていない構文や解析ルール外の記述が存在しないかを調査する必要がある．

研究 III について，条件分岐文単位でイベントや状態を表現する状態遷移表である MSTT を提案した．MSTT を利用することで，従来の状態遷移表では，表現することが困難なソースコードや，煩雑になるような表現を避け，順序情報を保持したまま状態遷移モデルに表現できる．MSTT を用いてソースコードを理解したかどうかについて調査を行ったところ，完全に正答した割合は約 77%であり，部分的に正解を含めると約 86%であった．また，MSTT のイベント全体の論理演算子（&&，||）の数が 2 割程度に減少した．

研究 IV について，組込みソフトウェアを対象とした細粒度状態遷移表（MSTT）の抽出手法を提案した．MSTT 抽出手法を実装した支援ツールによって，組込みソフトウェアから MSTT を抽出する適用実験を実施した結果，MSTT を抽出することができた．加えて，手作業における MSTT の抽出に比べて，支援ツールを使用することで MSTT 抽出時間を短縮でき，MSTT の抽出に要した時間は，支援ツールを使用する場合と使用しない場合において有意水準 5%で有意差があることを確認した．また，支援ツールを使用すると，より正確に MSTT を抽出することができることを確認した．正確さについては，実験において被験者が正しく表を作成できていなければ，実験実施者である著者は修正指示を与えていた．この修正回数が，ツールを用いた場合には 0 回であるのに対して，ツール不使用の場合には平均で 2.75 回の修正を要し，最低でも 1 回の修正を必要とした．そのため，より正確に MSTT を抽出することができていると判断した．本ツールの限界として，4 章で述べた通り，ループ構造などの表現方法が定義されていない構文からは MSTT を抽出できないこと，ポインタ解析ができないこと，および複数の状態変数を選択した場合の MSTT を抽出できないことである．また，今回の実験では実験に用いたすべてのソースコードで MSTT を抽出できたが，条件分岐文の深さや複雑さが大きい場合に，MSTT を抽出できないソースコードが存在する可能性がある．

以上の研究によって，図 6.1 中の赤字および赤線箇所を改善した，または改善するための問題を抽出した．次いで，**研究 I** および**研究 II** は，ソフトウェア要求仕様の作成プロセスおよびその成果物の品質を低下させる問題についての定量的な調査および，その自動検出手法の検討を行った．**研究 I** では，要求仕様書中の曖昧・誤り表現の出現数を調査し，ソフトウェア要求仕様の作成プロセスの改善およびその成果物の品質を低下させる問題の検出を手作業で行い，検出数を調査した．また，今後このプロセスを改善するために，語句・構文上の曖昧表現についての自動検出を試み，上記のような結果をることができた．語句・構文上の曖昧表現につ

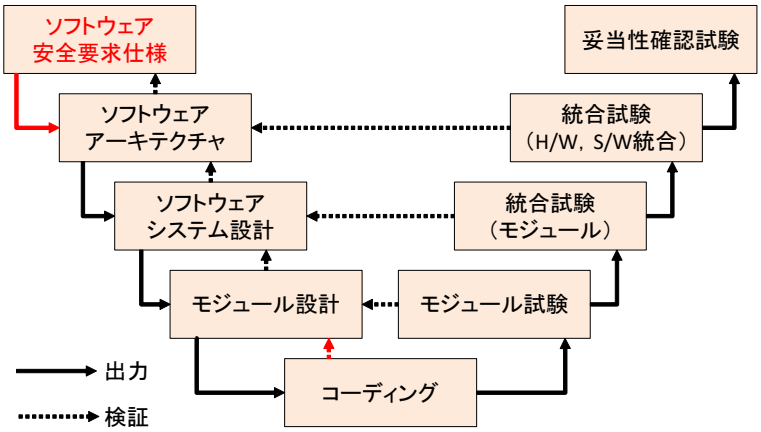

図 6.1: 本論文で改善したプロセス

いては，問題を検出することができており，これらの問題がユーザによって改善されれば，要求仕様書の品質は向上すると考えている．**研究 II** は，本質的には，要求仕様書から振る舞いのモデルを得る支援を行うツールである．そのため，要求仕様からソフトウェアアーキテクチャより後工程のプロセスを支援することにもつながる．この支援ツールの節分類の精度を調査し，これが要求仕様中の誤り表現の抽出に繋がると考えている．そのため，本論文においては．**研究 II** もソフトウェア要求仕様の作成プロセスおよびその成果物の品質を低下させる問題の改善や低減に寄与したものと考えている．**研究 III** および**研究 IV** は，再利用時のコーディングからモジュール設計を改善した．レガシー化が進行し，設計情報が欠如したまたは保守の中で文書との対応を取ることが困難となったソースコードの可読性を向上するため，MSTT を定義し，MSTT を抽出支援するツールを開発した．

6.2　今後の課題

研究 I について，今後の課題として，*recall* を向上するため，語句間の関係性の利用を検討する必要があること，および，他の要求仕様書についても定義した問題分類に合わせて問題の分類が可能か確認することが挙げられる．今後，手法をツール化することが考えられる．もしツール化するならば，*precision* を向上したいか，*recall* を向上したいかは，手法を用いるユーザによって変化することであるため，将来的にツール化する場合には，これら 2 つの手法をユーザが選択できるようにするべきであると考える．

研究 II について，今後の課題は，他の要求仕様書を与えたときの分類精度を調

査すること，および分類精度の向上をすることである．また，評価尺度について，F 値による評価のみ行っているが，今後，この節分類が組込みシステムをモデリングする上で妥当化などの評価を行う必要がある．他にも，状態遷移記述抽出支援ツールを応用して，さらなる要求仕様書解析手法の検討を進めることで，要求仕様書の理解のための支援手法や要求仕様書の検証手法に応用可能だと考えている．

研究 III について，今後の課題は，表現上の限界について議論を深めることと，ケーススタディにおける対象ソースコードの追加や，被験者の増員，MSTT の理解性に関するさらなる調査である．

研究 IV について，今後の課題として，状態遷移記述抽出支援ツールによって MSTT を抽出可能なソースコードの構造を増やし，状態変数を複数選択した場合の MSTT を出力可能にするなどの機能追加を含む，さらなる状態遷移記述抽出支援ツールの機能追加が挙げられる．

謝辞

本研究を実施するにあたり，多くの方々にご指導，ご協力，ご支援を賜りました．以下に謝意と御名前を記します．本当にありがとうございました．

名古屋大学 大学院情報学研究科 情報システム学専攻 高田 広章 教授には，本研究を実施するために，的確かつ親身なご助言を賜りました．また，博士前期課程においては異なる研究室の所属であったにも関わらず，enPiT を通じて組込みシステムの研究に携わる機会を提供いただきました．また，様々な研究会，ワークショップ，シンポジウム，展示会に参加する機会をいただき，産学問わず多くの方々と議論することができました．博士後期課程の 3 年間，充実した研究生活を送ることができましたのも，日々の先生のご指導，お人柄によるものと確信しております．心より御礼申し上げます．

名古屋大学 大学院情報学研究科 情報システム学専攻 山本 修一郎 教授には，本研究を実施するために，貴重なご意見を賜りました．また，博士前期課程においては山本研究室でご指導いただき，博士後期課程に進学するにあたり，高田研究室に移動する旨をご相談した際,「やりたいことをやったほうがいい」と，暖かく送り出していただきました．大学院から名古屋大学に進学し，右も左もわからない私に対して，熱心にご指導いただいたおかげで，博士論文を執筆するに至ったと確信しております．心より感謝いたします．

名古屋大学 大学院情報学研究科 組込みシステム研究センター 吉田 則裕 准教授には，研究だけではなく，学生生活なども含め常日頃からご支援賜りました．また，本論文の修正につきましても多くのご指摘・ご指導をいただきました．博士前期課程 1 年の頃より，enPiT での状態遷移表抽出支援ツールの開発について継続的にご指導いただき，本論文を執筆することができました．自然言語処理による要求仕様書解析についても，幾度も議論していただき，現在の論文を書き上げることができました．心より御礼申し上げます．

名古屋大学 大学院情報学研究科 本田 晋也 准教授には，研究だけではなく，学生生活なども含め常日頃からご支援賜りました．DNN の高位合成による設計に関する研究をはじめ，ハードウェアに関する研究の機会をいただきました．共同研

究を通して，技術補佐員として雇用していただき，金銭面の心配なく研究活動に集中することができました．また，この内容では，日本電気（株） 若林 一敏 氏，中村 寿彦 氏，酒井 完 氏，および，兵庫県立大学 中本幸一 教授からも，ご指導を賜りました．感謝いたします．

名古屋大学 大学院情報学研究科 枝廣 正人 教授 および石原 亨 教授，松原 豊 准教授，増田 豊 助教には，研究室ゼミやグループミーティングを通じ，研究についての様々なご指摘をいただきました．また，様々な相談に乗っていただき，ご助言を多数頂戴いたしました．感謝申し上げます．

細粒度状態遷移表の定義および支援ツール開発において，5 年に渡り議論していただきました，組込みシステム技術協会 状態遷移設計研究 WG の皆様に感謝を申し上げます．特に，私が参加した当時の主査 竹田 彰彦氏 および，現主査の青木 奈央氏には，支援ツールの仕様についての議論や，マネジメントをしていただき，本論文の 4 章および 5 章を作成するにあたり，多大なる貢献をいただきました．また，同 WG の（株）エリック・アンド・アンディ 大塚 悦生 氏，日本電気通信システム（株）難波 秀之 氏,（株）メタテクノ 小林 良 氏,（株）ニッキ 稲生 州俊 氏には，支援ツールに関する具体的な問題点の指摘や表現などについてのコメントを頂き，ツール化を推進することができました．感謝申し上げます．

2 章・3 章に関連して，要求仕様書の曖昧・誤り表現に関する議論に参加いただいた,（株）エクスモーションの玉木 淳治 氏に感謝申し上げます．企業の方の視点における要求仕様書に対する問題意識は，本論文を書き上げる上で参考となりました．感謝申し上げます．

日頃より研究生活についての御支援や研究についての議論をして頂きました名古屋大学大学院情報学研究科情報システム学専攻の高田・松原研究室および枝廣・本田研究室の皆様に厚く御礼申し上げます．中でも，小川 真彩高 博士，中村 成 氏，岡本 卓也 氏，清水 貴裕 氏，都築 夏樹 氏，稲石 日奈子 氏には，多大なるご助言，ご協力をいただきました．感謝いたします．また，本論文における実験の被験者をしていただいた方に感謝いたします．本当にありがとうございました．

最後に，長期に渡る研究生活を支えてくれた家族に対し心から感謝致します．

参考文献

[1] 鈴木 一哉, 森本 昌治, and 岩井 孝法. IoT 技術の最新動向. **電子情報通信学会通信ソサイエティマガジン**, 12(1):12–20, 2018.

[2] 小川 真彩高, 本田 晋也, and 高田 広章. 車載制御システム向けマルチコアプログラミングフレームワーク. **情報処理学会論文誌**, 58(2):507–520, 2017.

[3] 高田 広章. 組込みシステム開発技術の現状と展望. **情報処理学会論文誌**, 42(4):930–938, 2001.

[4] Peter Liggesmeyer and Mario Trapp. Trends in embedded software engineering. *IEEE software*, 26(3):19–25, 2009.

[5] International Electrotechnical Commission. IS/IEC 61508-3: Functional safety of electrical/electronic/programmable electronic safety-related systems, part 3: Software requirements, 1998.

[6] 組込みシステム技術協会 機能安全委員会 製品安全ワーキンググループ. 平成 20 年度 組込みシステムにおける機能安全に関する調査研究組込み系技術者のための安全設計入門, 2008. `http://www.jasa.or.jp/TOP/download/technical/H20_SafetyReport.pdf`.

[7] Ekaterina Boutkova and Frank Houdek. Semi-automatic identification of features in requirement specifications. In *Proc. of RE 2011*, pages 313–318, 2011.

[8] 中村 成, 山本 椋太, 吉田 則裕, and 高田広章. 組込みシステムを対象とした要求仕様書からの状態遷移記述の抽出. **情報処理学会研究報告**, 2018-SE-198(5):1–8, 2018.

[9] 中村 成. 組込みシステムの要求仕様書を対象とした状態遷移表作成支援. 名古屋大学大学院情報学研究科 情報システム学専攻 修士論文, https://sites.google.com/site/yoshidaatnu/naru.pdf, 2019.

[10] 平山雅之. 組み込みソフトウェア開発技術: 1. 組み込みソフト開発の現状. **情報処理**, 45(7):677–681, 2004.

[11] 鵜飼 敬幸. Toppers/ssp への組込みコンポーネントシステム適用における設計情報の可視化と抽象化. 第 9 回クリティカルソフトウェアワークショップ (WOCS2), 2011. https://www.ipa.go.jp/files/000005299.pdf.

[12] 竹田 彰彦. 状態遷移設計研究会活動紹介「状態遷移表のリバースモデリングへの適用」. 組込みシステム総合技術展 JASA 技術本部セミナー, 2013. http://www.jasa.or.jp/TOP/download/technical/et2013_design_reversemodeling.pdf.

[13] Neil Walkinshaw, Kirill Bogdanov, Shaukat Ali, and Mike Holcombe. Automated discovery of state transitions and their functions in source code. *Software Testing, Verification and Reliability*, 18(2):99–121, 2008.

[14] 位野木 万里 and 近藤 公久. 省略と修飾パターンを用いた用語不一致検証による要求仕様の一貫性検証支援ツールの実現と適用評価. **コンピュータ ソフトウェア**, 35(3):3_109–3_127, 2018.

[15] 大森 洋一 and 荒木 啓二郎. 自然言語による仕様記述の形式モデルへの変換を利用した品質向上に向けて. **情報処理学会論文誌プログラミング** *(PRO)*, 3(5):18–28, 2010.

[16] 渡辺 政彦. 状態遷移ベースのソフトウェア開発環境の現状と動向. **計測と制御**, 41(2):117–121, 2002.

[17] Wenbin Li. *Consistency Checking of Natural Language Temporal Requirements using Answer-Set Programming*. PhD thesis, College of Engineering at the University of Kentucky, 2015. https://uknowledge.uky.edu/cs_etds/34.

[18] International Organization for Standardization
. ISO/IEC/IEEE 29148:1998 - systems and software engineering - life cycle processes - requirements engineering, 2011.

[19] Artem Katasonov and Markku Sakkinen. Requirements quality control: a unifying framework. *Requirements Engineering*, 11(1):42–57, 2006.

[20] 青山 幹雄 and 中根 拓也. ReqQA：ソフトウェア要求仕様書品質解析ツールの提案と評価. **情報処理学会論文誌**, 57(2):694–706, feb 2016.

[21] Adam A Porter, Lawrence G Votta, and Victor R Basili. Comparing detection methods for software requirements inspections: A replicated experiment. *IEEE Transactions on software Engineering*, 21(6):563–575, 1995.

[22] 石田 裕三. ソフトウェア再利用の新しい波-広がりを見せるプロダクトライン型ソフトウェア開発-: 6. エンタープライズ・システムにおけるソフトウェアプロダクトラインの適用. **情報処理**, 50(4):303–310, 2009.

[23] 小高 文博 and 佐藤 建吉. 組込みシステム開発のリスクマネジメント. **情報科学技術フォーラム講演論文集**, 10(1):455–458, 2011.

[24] 関 洋太朗, 林 晋平, and 佐伯 元司. ユースケース記述中の不吉な臭いの体系化と検出. In **情報処理学会研究報告**, volume 2019-SE-201, pages 1–8, 2019.

[25] Daniel M Berry. Ambiguity in natural language requirements documents. In *Monterey Workshop*, pages 1–7. Springer, 2007.

[26] 岡本 周之. **組込みソフトウェア開発における工数削減および期間短縮に関する研究**. PhD thesis, 大阪大学大学院情報科学研究科, 2016.

[27] 新 吉高, 鯨井 俊宏, 土井 敬司, 深谷 直彦, 原 隆浩, 西尾 章治郎, 萩尾 一仁, 御手洗 秀一, 石野 明, and 竹田 正幸. 組込みシステム向け HMI ツールのモデルベース開発について. *DBSJ Letters*, 6(2):1–4, 2007.

[28] Paolo Tonella and Alessandra Potrich. *Reverse engineering of object oriented code*. Springer, 2005.

[29] 増田 聡, 松尾谷 徹, and 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**, 34(2):16–27, 2017.

[30] Daisuke Kawahara and Sadao Kurohashi. A fully-lexicalized probabilistic model for japanese syntactic and case structure analysis. In *Proc. the main*

conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, pages 176–183, 2006.

[31] 河原 大輔 and 黒橋 禎夫. 自動構築した大規模格フレームに基づく構文・格解析の統合的確率モデル. **自然言語処理**, 14(4):67–81, 2007.

[32] 笹野 遼平 and 黒橋 禎夫. 大規模格フレームを用いた識別モデルに基づく日本語ゼロ照応解析. **情報処理学会論文誌**, 52(12):3328–3337, 2011.

[33] Ryohei Sasano and Sadao Kurohashi. A discriminative approach to japanese zero anaphora resolution with large-scale lexicalized case frames. In *Proc. IJCNLP*, pages 758–766, 2011.

[34] 加藤 潤三, 佐伯 元司, 大西 淳, 海谷 治彦, and 山本 修一郎. シソーラスを利用した要求獲得方法 (THEOREE). **情報処理学会論文誌**, 50(12):3001–3017, 2009.

[35] 高橋 加寿子, 塚本 良太, and 磯田 誠. 変更要求に対するシステム設計書修正箇所抽出法の検証. **情報処理学会研究報告ソフトウェア工学** *(SE)*, 2018(5):1–6, 2018.

[36] Alessio Ferrari, Paola Spoletini, and Stefania Gnesi. Ambiguity and tacit knowledge in requirements elicitation interviews. *Requirements Engineering*, 21(3):333–355, 2016.

[37] 位野木 万里 and 近藤 公久. 要求仕様の一貫性検証支援ツールの提案と適用評価. *SEC journal*, 13(1):16–23, 2017.

[38] 林 晋平, 有賀 顕, and 佐伯 元司. reqchecker: IEEE 830 の品質特性に基づく日本語要求仕様書の問題点検出ツール. **電子情報通信学会論文誌** *D*, 101(1):57–67, 2018.

[39] 竹内 広宜, 中村 大賀, and 荻野 紫穂. ソフトウェア開発における文書成果物の分析技術とその活用. **コンピュータソフトウェア**, 30(1):53–64, 2013.

[40] 竹内 広宜, 荻野 紫穂, 中田 武男, 坂本 佳史, and 福岡 直明. テキスト分析技術を用いた開発関連文書の品質分析. **組込みシステムシンポジウム** *2009*, pages 93–100, 2009.

[41] Benedikt Gleich, Oliver Creighton, and Leonid Kof. Ambiguity detection: Towards a tool explaining ambiguity sources. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 218–232. Springer, 2010.

[42] Ryo Hasegawa, Motohiro Kitamura, Haruhiko Kaiya, and Motoshi Saeki. Extracting conceptual graphs from japanese documents for software requirements modeling. In *Proceedings of the Sixth Asia-Pacific Conference on Conceptual Modeling-Volume 96*, pages 87–96, 2009.

[43] Leonid Kof. Natural language processing: Mature enough for requirements documents analysis? In *International Conference on Application of Natural Language to Information Systems*, pages 91–102. Springer, 2005.

[44] Leonid Kof. Scenarios: Identifying missing objects and actions by means of computational linguistics. In *RE 2007*, pages 121–130. IEEE, 2007.

[45] Tomoya Mizumoto, Mamoru Komachi, Masaaki Nagata, and Yuji Matsumoto. Mining revision log of language learning sns for automated japanese error correction of second language learners. In *Proc. of IJCNLP*, pages 147–155, 2011.

[46] 笠原 誠司, 藤野 拓也, 小町 守, 永田 昌明, and 松本 裕治. 日本語学習者の誤り傾向を反映した格助詞訂正. **言語処理学会第** *18* **回年次大会**, pages 14–17, 2012.

[47] Alla Rozovskaya and Dan Roth. Generating confusion sets for context-sensitive error correction. In *Proc. of the 2010 conference on empirical methods in natural language processing*, pages 961–970, 2010.

[48] 今枝 恒治, 河合 敦夫, 石川 裕司, 永田 亮, and 桝井 文人. 日本語学習者の作文における格助詞の誤り検出と訂正. **情報処理学会研究報告コンピュータと教育** *(CE)*, 2003(13 (2002-CE-068)):39–46, 2003.

[49] 村上 響一, 青山 裕介, 村上 神龍, 久代 紀之, 牧 茂, 田畑 一政, 神代 勉, and 中村潤. 自然言語仕様書からの試験ケース生成のための条件・動作の同定手法. **情報処理学会研究報告**, 2018-SE-198(7):1–7, 2018.

[50] 黒橋 禎夫. 日本語構文解析システム KNP version 4.1 使用説明書. http://nlp.ist.i.kyoto-u.ac.jp/?KNP, 2013.

[51] Aditya Desai, Sumit Gulwani, Vineet Hingorani, Nidhi Jain, Amey Karkare, Mark Marron, and Subhajit Roy. Program synthesis using natural language. *Proceedings of the 38th International Conference on Software Engineering, ICSE ' 16*, pages 345–356, 2018.

[52] Thomas Quirchmayr, Barbara Paech, Roland Kohl, Hannes Karey, and Gunar Kasdepke. Semi-automatic rule-based domain terminology and software feature-relevant information extraction from natural language user manuals. *Empirical Software Engineering*, 23(6):3630–3683, 2018.

[53] 南保 亮太, 乙武 北斗, and 荒木 健治. 文節内の特徴を用いた日本語助詞誤りの自動検出・校正. **情報処理学会研究報告**, 2007-NL-181(94):107–112, 2007.

[54] 山本 椋太, 吉田 則裕, 青木 奈央, and 高田 広章. 組込みソフトウェアを対象とした状態遷移表抽出支援ツール. **電子情報通信学会論文誌** *D*, J102-D(3):151–162, 2019.

[55] Noboru Hattori, Shuichiro Yamamoto, Tsuneo Ajisaka, and Tsuyoshi Kitani. Proposal for requirement validation criteria and method based on actor interaction. *IEICE TRANSACTIONS on Information and Systems*, 93(4):679–692, 2010.

[56] Berndt Bellay and Harald Gall. An evaluation of reverse engineering tool capabilities. *Journal of Software Maintenance*, 10(5):305–331, 1998.

[57] Lucian Cojocar, Jonas Zaddach, Roel Verdult, Herbert Bos, Aurélien Francillon, and Davide Balzarotti. PIE: parser identification in embedded systems. In *Proc. ACSAC*, pages 251–260, 2015.

[58] Benjamin Hummel and Thomas Kinnen. Incremental software quality analysis for embedded systems. In *embedded world Conference 2015*, 2015.

[59] Yan Liu and Ted Wong. Component architecture and modeling for microkernel-based embedded system development. In *Proc. ASWEC 2008*, pages 190–199, 2008.

[60] Bas Graaf, Marco Lormans, and Hans Toetenel. Embedded software engineering: The state of the practice. *IEEE software*, 20(6):61–69, 2003.

[61] Wilhelm Kirchmayr, Michael Moser, Ludwig Nocke, Josef Pichler, and Rudolf Tober. Integration of static and dynamic code analysis for understanding legacy source code. In *Proc. of ICSME*, pages 543–552, 2016.

[62] S.K. Mishra, Dharmender Singh Kushwaha, and Arun Kumar Misra. Creating reusable software component from object-oriented legacy system through reverse engineering. *Journal of object technology*, 8(5):133–152, 2009.

[63] Wasim Said, Jochen Quante, and Rainer Koschke. Towards interactive mining of understandable state machine models from embedded software. In *MODELSWARD*, pages 117–128, 2018.

[64] Ioannis Stamelos, Lefteris Angelis, Apostolos Oikonomou, and Georgios L Bleris. Code quality analysis in open source software development. *Information Systems Journal*, 12(1):43–60, 2002.

[65] Jan Hannemann and Gregor Kiczales. Overcoming the prevalent decomposition in legacy code. Workshop on Advanced Separation of Concerns, 2001. `http://www.sdml.cs.kent.edu/library/Hannemann01.pdf`.

[66] Wasim Said, Jochen Quante, and Rainer Koschke. Reflexion models for state machine extraction and verification. Proc. of ICSME, 2018.

[67] Ira D Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Anna, and Lorraine Bier. Clone detection using abstract syntax trees. In *Proc. of ICSM, 1998*, pages 368–377, 1998.

[68] Neil Walkinshaw, Ramsay Taylor, and John Derrick. Inferring extended finite state machine models from software executions. *Empirical Software Engineering*, 21(3):811–853, 2016.

[69] 情報処理推進機構 ソフトウェア・エンジニアリング・センター. 組込みソフトウェア開発における品質向上の勧め［設計モデリング編］. アイティメディア, 2006.

[70] David Harel, Hagi Lachover, Amnon Naamad, Amir Pnueli, Michal Politi, Rivi Sherman, Aharon Shtull-Trauring, and Mark Trakhtenbrot. Statemate: A working environment for the development of complex reactive systems. *IEEE Transactions on software engineering*, 16(4):403–414, 1990.

[71] Michel D Ingham, Robert D Rasmussen, Matthew B Bennett, and Alex C Moncada. Engineering complex embedded systems with state analysis and the mission data system. *Journal of Aerospace Computing, Information, and Communication*, 2(12):507–536, 2005.

[72] Daniel Dvorak. NASA study on flight software complexity. AIAA Aerospace Conference and AIAA "Unmanned Unlimited" Conference, 2009. `https://arc.aiaa.org/doi/10.2514/6.2009-1882`.

[73] Klaus Grimm. Software technology in an automotive company-major challenges. In *Proc. of ICSE*, pages 498–503, 2003.

[74] M Srinivas, G Ramakrishna, K Rajasekhara Rao, and E Suresh Babu. Analysis of legacy system in software application development: A comparative survey. *International Journal of Electrical and Computer Engineering*, 6(1):292, 2016.

[75] 川勝 則孝. モデルベース開発に向けたソースコードからの状態遷移モデル抽出技術. **東芝レビュー**, 69(8), 2014.

[76] Wasim Said, Jochen Quante, and Rainer Koschke. On state machine mining from embedded control software. In *Proc. of ICSME*, pages 138–148. IEEE, 2018.

[77] A. W. Biermann and J. A. Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE Trans. Comput.*, C-21(6):592–597, 1972.

[78] Tao Xie and David Notkin. Automatic extraction of object-oriented observer abstractions from unit-test executions. In *Proc. ICFEM*, pages 290–305, 2004.

[79] Davide Lorenzoli, Leonardo Mariani, and Mauro Pezzè. Inferring state-based behavior models. In *Proc. WODA*, pages 25–32, 2006.

[80] 本田 晋也 and 山本 椋太. 車載制御システム向け次世代プロセッサの仮想化支援機能を用いたハイパーバイザー. **情報処理学会技術報告**, 2019-EMB-52(16):1–6, 2019.

[81] 松崎 寛之. レガシーデバイスを用いたホームネットワーク構築における機器の状態取得及び管理に関する研究. Master's thesis, 北陸先端科学技術大学院大学 情報科学研究科, 2005.

[82] 高橋 正和 and 津田 和彦. 医薬品製造設備における制御ソフトウェアの効率的コンピュータ・バリデーション. **情報処理学会論文誌**, 45(12):2869–2879, 2004.

[83] 中島 毅, 別所 雄三, 山中 弘, and 広田 和洋. 状態遷移モデルで記述された要求仕様に基づく組込みソフトウェアの自動試験法. **電子情報通信学会論文誌** *D*, 84(6):682–692, 2001.

[84] 中島 毅, 別所 雄三, 山中 弘, and 広田 和洋. シングルチップマイコン用 S/W 開発における問題点と一解決法. **情報処理学会研究報告**, 1997(74):17–24, 1997.

[85] Sarfraz Khurshid, Corina S Păsăreanu, and Willem Visser. Generalized symbolic execution for model checking and testing. In *Proc. TACAS*, pages 553–568. Springer, 2003.

[86] Neil Walkinshaw and Mathew Hall. Inferring computational state machine models from program executions. In *Proc. of ICSME*, pages 122–132. IEEE, 2016.

[87] Tamal Sen and Rajib Mall. Extracting finite state representation of java programs. *Software & Systems Modeling*, 15(2):497–511, 2016.

[88] David Kung, Nimish Suchak, Jerry Gao, Pei Hsia, Yasufumi Toyoshima, and Chris Chen. On object state testing. In *Proc. COMPSAC 94*, pages 222–227, 1994.

[89] Wasim Said, Jochen Quante, and Rainer Koschke. Towards understandable guards of extracted state machines from embedded software. In *Proc. of SANER*, pages 264–274. IEEE, 2019.

研究業績

学術論文

- 山本 椋太，吉田 則裕，青木 奈央，高田広章， 組込みソフトウェアを対象とした状態遷移表抽出支援ツール, 電子情報通信学会論文誌 D, Vol.J102-D, No.3, pp.151-162, Mar 2019.
- 山本 椋太，中村 成，吉田 則裕，高田広章，組込みシステムの要求仕様書に対する誤り・曖昧表現の実証的調査, 電子情報通信学会論文誌 D，Vol.J103-D, No.5，May 2019（採録決定）.

国際会議論文（査読あり）

- Ryota Yamamoto, Norihiro Yoshida, Hiroaki Takada, Towards Static Recovery of Micro State Transitions from Legacy Embedded Code, Proceedings of the 1st ACM SIGSOFT International Workshop on Automated Specification Inference (WASPI 2018), pp.1-4, Nov. 2018

国内研究会・シンポジウム発表論文（査読あり）

- 山本 椋太, 中村 成, 吉田 則裕, 高田 広章: ”組込みシステムの要求仕様書に対する修正候補の定量的調査”, 日本ソフトウェア科学会, 第 26 回ソフトウェア工学の基礎ワークショップ (FOSE 2019), pp.57-62, Nov. 2019

国内研究会・シンポジウム発表論文（査読なし）

- 山本 椋太, 吉田 則裕, 高田 広章: ”組込みシステムの要求仕様書に対する修正候補の定量的調査”, 電子情報通信学会技術研究報告, Vol.117, No.246, SS2017-29/DC2017-28, pp.49-54, Oct. 2017
- 山本 椋太, 吉田 則裕, 青木 奈央, 高田 広章: ”組込みソフトウェアを対象とした状態遷移表の抽出と分析支援の検討”, 電子情報通信学会技術研究報告, Vol.117, No.136, SS2017-16/KBSE2017-16, pp.133-138, Jul. 2017

- 山本 椋太, 吉田 則裕, 竹田 彰彦, 舘 伸幸, 高田 広章: ”組込みソフトウェアを対象とした状態遷移表抽出手法”, 電子情報通信学会技術研究報告, Vol.116. No.127, SS2016-3/KBSE2016-9, pp.13-18, Jul. 2016