ソフトウェア開発実験に基づく構造化分析／設計手法の評価

岡 敦子† 山木 修一郎† 磯田 定宏†

構造化分析／設計手法（以下，SA 手法）をベースとする CASE ツールが多く開発されている．その効果として，プログラム品質が向上する，プログラムの生産性が向上する，テスト・デバッグ工数が減少するなどの結果が期待されている．また，一方では習熟に時間がかかるという点も指摘されている．しかし，いずれも定性的な予測であり，具体的・定量的な報告はごくわずかしかない．本論文では，これらの予測を定量的に検証するために，できるかぎり同一条件の下で SA 手法と従来手法とを比較分析するソフトウェア開発実験を行い，以下を明らかにした．(1)SA 手法を用いて開発したプログラムの受入れテスト時の品質は従来手法を用いて開発した場合の約2倍に向上する．(2)SA 手法を用いて開発したほうがテスト・デバッグ工数が少ない．(3)SA 手法では約 30 時間の講習を受けさせた場合，2回目の適用から仕様品質に関して習熟効果がある．

An Experimental Evaluation of Structured Analysis and Design Methodology

Atsuko Oka,† Shuichiro Yamamoto† and Sadahiro Isoda†

CASE tools based on Structured Analysis and Design (SA/SD) methodology are said to bring about an increase in program quality and productivity, and a reduction of efforts for testing and debugging, although a long learning time is required before reaping these benefits. However, these are mostly qualitative assumptions. This paper describes software development experiments conducted to quantitatively compare SA/SD methodology with the traditional methodology under conditions which are as similar as possible. The following results are obtained: (1) Programs developed with SA/SD methodology have twice as high quality on the acceptance test as those developed using the traditional methodology. (2) SA/SD methodology reduces the effort required in testing and debugging. (3) The quality of specifications in the functional review increases at the second application of SA/SD methodology when the software engineers had 30 hour training, demonstrating an earlier learning effect than usually acknowledged.

1. はじめに

構造化分析／設計手法（以下，SA 手法）[1)-3)] をベースとする CASE ツールにより，プログラム品質が向上する，テスト・デバッグ工数が減少するなどの結果が期待されている．また，一方では SA 手法の習熟に時間がかかるという点が指摘されている．しかし，いずれも定性的な予測であり，具体的，定量的な報告はごくわずかしかない．山本らは，CASE ツールの導入により，結合試験工程での平均誤り発生率が約 25% 減少し，プログラム品質が向上した例を報告している[4)]．Lempp らは，SA 手法を約 20 件の開発プロジェクトに適用した結果，従来手法と比較して，開発時の生産性が 9% 向上したと報告している[5)]．しかし，これらの報告はいずれも従来の標準値との比較であり，同一の要求仕様について従来手法と SA 手法とでプログラム開発を行った結果に基づいていない．中条らは，大規模ソフトウェア開発プロジェクトの一部，約 52 KLOC に対して SA 手法と従来手法をそれぞれ6サブシステムずつに適用し，手法が開発時の誤りに与える影響を分析した．この結果，単体／結合テストでは，SA 手法を用いて開発したソフトウェアは従来手法と比較して，単位規模当たりのエラー数が約 1/2 になるという結果を得た[6)]．しかし，同一の要求仕様について従来手法と SA 手法とでプログラム開発を行った結果の比較ではなく，異なる機能を持ったプログラム同士を比較しているため厳密な比較にはなっていない．Nosek らは，情報科学専攻の大学生を被験者とし，SA 手法を含む3種の設計法で作られた機能仕様書に対する理解度の比較実験を行った．この

† NTT ソフトウェア研究所
NTT Software Laboratories

結果，設計法の間には有意な差がないという結果を得た[7]．しかし，この研究では開発作業自体を評価対象としていない，被験者がソフトウェア開発の実務経験がない，被験者が設計手法を理解していないという問題点がある．

以上述べたように，これまでに行われた研究では，まだ設計法に対する定量的な評価が不十分である．そこで，本論文ではできるかぎり同一条件のもとで SA 手法と従来手法とを比較する実験を実施し，仕様およびプログラムの品質／開発工数／生産性に関して両手法を比較分析する．

2. 実験方法

2.1 実験条件

一般に2つの設計手法を比較評価するためには，実際のソフトウェア開発とできるだけ同一状況で，かつ，2つの設計法の間で条件ができるだけ同等になるように実験を制御する必要がある．このためには，次の条件が必要である．

(1)作成規模は数 KLOC 以上ある．
(2)被験者は実験で用いるプログラミング言語を用いたソフトウェア開発の実務経験がある．
(3)被験者はその設計手法を理解している．
(4)同一問題について比較する．

実験期間が限られていたため対象プログラム規模を2〜3 KLOC 程度におさえた（条件(1)）．また，集めることができた被験者はいずれもC言語によるソフトウェア開発経験を有していた（条件(2)）．しかし，SA 手法に精通している被験者を多く集めることが困難であったため，次善の策として被験者に SA 手法の講習会を受けさせてから実験を行うことにより，条件(3)に近づけた．

なお，品質の指標としては，あらかじめ用意したチェック項目のうち，誤りが発見された項目の割合（誤り発生率）を用いた．

また，本論文では統計的に有意な差があるかどうかを調べるために，t 検定を実施した．

2.2 実験の目的

本実験では以下に示す3仮説を検証する．

[仮説1] SA 手法を用いて開発した方がソフトウェアの品質が従来手法と比較して高い．

[仮説2] SA 手法を用いて開発した方がソフトウェアの生産性が従来手法と比較して高い．

[仮説3] SA 手法を用いて開発した方がテスト・デバッグ工数が従来手法と比較して少ない．

また，被験者が SA 手法の初心者であることを利用して SA 手法の習熟性についての知見を得る．

2.3 被験者

C言語によるプログラミングならびにソフトウェア開発の実務経験者（1年〜10 年，12 名）を被験者とした．1チームを2名の被験者から構成し，SA 手法チーム（S1, S2, S3, S4），従来手法2チーム（J1, J2）を構成した．被験者に対し，実験前にC言語のプログラミング能力テストおよび設計レビュ能力テストを実施した．その結果，SA 手法チームと従来手法チームとの間に有意な差はなかった（有意水準 5%）．また，経験年数についても同様だった．

2.4 実験スケジュール

本実験は開発実験と分析実験から構成される．実験の流れを**図1**に示す．

2.5 実験対象プログラム

実験対象プログラムは，同程度の難易度を持つ小規模（C言語，約 2.5 KLOC）のトランザクションシステム，技術会議管理システム（P1），図書室管理システム（P2），医薬品在庫管理システム（P3），部品の在庫管理システム（P4）である．

手法による分析作業の違いを測定するために，実際のソフトウェア開発と同様，あいまいさを含んだ問題文を用意した．開発実験では P1 と P2 のみを対象とし，日本語で記述された問題文に基づき，分析・機能設計／詳細設計／コーディング／テスト・デバッグの各工程を経て，UNIX 上でコンパイル／実行可能なプログラムを開発した．

開発実験終了後，引続き分析実験を実施した．手法に対する習熟効果を調べるために分析・機能設計作業を2回続けて行った．

各チームへのプログラムの割り当てを**表1**に示す．

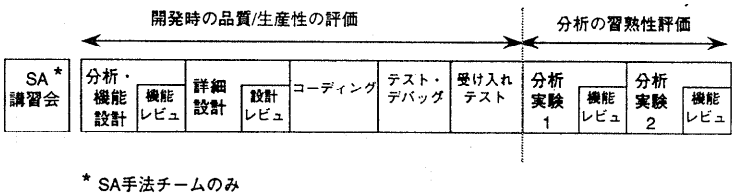

図1 実験スケジュール
Fig. 1 Experiments schedule.

表 1 開発チームと開発対象システムとの関係
Table 1 Problem assignments for development and analysis experiments.

チーム名	開発実験	分析実験 1	分析実験 2
S1	P1	P2	P3
S2	P1	P2	P4
S3	P2	P1	P3
S4	P2	P4	P3
J1	P1	P3	P2
J2	P2	P1	P3

2.6 使用ツール・作成ドキュメント

SA 手法チームは，統合化 CASE SoftDA[8),9)] の構造化分析／設計サブシステム SoftDA/SA を用いて，SA 手法に基づくドキュメントを作成した．機能設計での作成ドキュメントは，データフロー図，プロセス仕様書，データ構造図である．また，詳細設計での作成ドキュメントは，モジュールの呼び出し関係とモジュール間の受渡しパラメータを記述するモジュール構造図およびモジュール仕様書である．プロセス仕様書とモジュール仕様書については特に定まった図式を用いず，日本語文章を用いて記述した．

一方，従来手法を用いた開発では，パソコン上のワープロを用いて日本語文章形式のドキュメントを作成した．機能設計での作成ドキュメントは，インタフェース仕様書，機能仕様書，ファイル仕様書である．インタフェース仕様書は，システムと外部との間の入出力データから構成される．機能仕様書の記述項目は，機能名，機能概要，入力データ，出力データである．ファイル仕様書の記述項目はファイル名，ファイル概要，ファイルの構成要素である．詳細設計では手書きのモジュール構造図を記述した．これは，モジュールの呼び出し関係のみを記述し，受渡しパラメータを記述しない点が SA 手法と異なる．また，モジュール仕様には手書きの PAD または HIPO を用いた．したがって，従来手法を用いた場合の作成ドキュメントは，記述形式は異なるが，記述内容は SA 手法グループの作成ドキュメントとほぼ同じである．

コーディングでは，両手法ともに UNIX 上のテキストエディタを用いた．テスト・デバッグでは，両手法ともに，SoftDA/TCA[12)] を用いて，モジュールごとの C1 テストカバー率（全分岐網羅率）を測定した．SA 手法チームと従来手法チームの作成ドキュメントを**表 2** に示す．

2.7 開発実験の進め方

本実験の実施者（以下，コーディネータ）は実験全体を指揮することに加え，通常のソフトウェア開発の場合の発注者の役割を果たした．すなわち，問題文を事前に分析し，プログラムとして実現すべき機能および制約条件を決定し，これを作業文書（以下，基準仕様）として記録した．さらに基準仕様の内容を確認するためのチェック項目を作成した．ついで，コーディネータは，実験の過程で被験者からの仕様に関する問い合わせに基準仕様に従って答えた．ただし，その過程でコーディネータが基準仕様およびチェック項目を詳細化する場合もあった．コーディネータは同様にレビュないしテストを実施した．なお，基準仕様は被験者には開示しなかった．

表 2 SA 手法と従来手法のドキュメント
Table 2 Documents of SA/SD methodology and those of the traditional methodology.

工　程	SA 手法	従来手法
機能設計	データフロー図 プロセス仕様書	インタフェース仕様書 機能仕様書
	データ構造図	ファイル設計書
詳細設計	モジュール構造図	モジュール構造図
	モジュール仕様書	モジュール仕様書
コーディング	ソースコード	ソースコード
テスト・デバッグ	テスト仕様書	テスト仕様書

チェック項目は（1）**機能に関する項目**，（2）**データに関する項目**，（3）**制約条件に関する項目**の 3 種類から構成される．

（1）機能に関する項目は，必要な機能項目が実現されているかどうかを確認する項目である．

（2）データに関する項目は，必要なファイルやファイル中のデータ項目が管理されているかを確認する項目である．データに種別がある場合に種別が考慮されているかどうか，データの型があらかじめ決まっている場合に正しく設計されているかどうかを確認する項目も含まれる．

（3）制約条件に関する項目は，登録処理における二重登録をチェックするなど，複数のデータや処理に関する制約条件が実現されているかどうかを確認する項目である．その他，（1），（2）のいずれにも該当しない項目は，この種別に分類した．したがって，各チェック項目はそれぞれ独立であった．

各チームは，以下の手順に従って対象システムを開発した．

［手順 1］ SA 手法の学習

SA 手法を用いる 4 チーム（S1, S2, S3, S4）は，

SA 手法と SoftDA/SA について 3 日間の講習（約 30 時間）を受講した．

[手順 2] 分析・機能設計

各チームの被験者は，開発対象システムに関する問題文を分析することにより，対象システムの外部入出力，機能および各機能の入出力，さらにファイルや入出力帳票についてデータ構造の明確化を行い，それぞれの手法に基づくドキュメントを作成した．

実際のソフトウェア開発ではソフトウェアの発注者が必要な機能をすべて提示するとは限らない．このため，開発者は不足している機能を追加する必要がある．また，発注者が提示した要求はあいまいさを含んでいるため，開発者は分析過程で問題文中の矛盾する内容や改善すべき項目，不明な項目を発見することがある．その都度，開発者は発注者に問い合わせを行い内容を照会する．

本実験では，コーディネータが発注者の役割を果たした．すなわち，被験者は質問票を作成し，コーディネータに質問内容を照会した．コーディネータは基準仕様に基づき，質問票に回答した．

分析終了後，被験者はチーム内でドキュメントを読み合わせ，分析内容をチェックし，誤りや抜けを修正した．

機能レビュではコーディネータがあらかじめ作成した 30 件のチェック項目について，コーディネータが被験者に質問し，基準仕様との一致性を確認した．機能に抜けや誤りがある場合，コーディネータが被験者に指摘し，被験者は指摘された項目を修正した．その後，被験者は修正ドキュメントに基づき作成規模を見積もった．なお，基準仕様に含まれない機能が機能仕様に含まれていた場合（超過機能），機能レビュではチェックの対象にしなかった．しかし，超過機能は見積り規模の増大として現れた．見積り規模が著しく開発可能規模を越える場合，後工程に影響を及ぼすため，開発可能規模におさまるように，コーディネータは超過機能の一部を削除するよう被験者に指示した．

[手順 3] 詳細設計

詳細設計では，被験者はモジュール構造，モジュール間の入出力データおよび各モジュールの仕様を設計した．なお，SA 手法チームでは SoftDA/SA のモジュール構造図自動生成機能を用いてデータフロー図からモジュール構造図を自動生成した．自動生成されたモジュール構造図は，初期化・終了処理，例外処理が不足していたり，冗長な部分が存在するため，被験者はモジュールの詳細化・統合などの整形を行った[10),11)]．

一通り設計が終了した後，被験者はチーム内レビュを実施し，その結果を反映した．さらに，モジュール構造およびモジュール仕様をチェックするために，コーディネータがあらかじめ作成した約 10 件のチェック項目に従って，コーディネータと被験者は設計レビュを実施した．レビュ後，被験者はコーディネータが指摘した設計誤り，抜けを修正した．

[手順 4] コーディング

被験者は，詳細設計工程で作成したモジュール構造図とモジュール仕様書に基づき，プログラムコードを作成した．

[手順 5] テスト・デバッグ

開発期間が限られていたため，コーディングした全プログラムに対してテスト・デバッグを実施できなかったので，コーディネータがシステムに不可欠な機能にテスト対象範囲を限定し，被験者にテスト・デバッグを実施させた．テスト対象規模は約 2〜3 KLOC であった．被験者はモジュール仕様書に基づきテストデータを作成し，検収条件として構造テスト・機能テストを含めて C1 テストカバー率が 80% に達するまでテストおよびデバッグを行った．このとき，テスト項目は 150〜200 件/KLOC 程度であった．

[手順 6] 受け入れテスト

コーディネータはデバッグが完了したプログラムに対して，テスト範囲を被験者が行ったテスト対象機能（[手順 5]）に限定し受け入れテストを実施した．この受け入れテストのチェック項目は約 50 件であった．

2.8 分析実験の進め方

手法による分析作業経験の習熟効果の違いを評価するため，開発実験後に分析・機能設計を 2 回実施した．これを分析実験と呼ぶ．開発実験時の機能設計工程を含めると，分析・機能設計は計 3 回となる．

3. 開発実験結果

3.1 開発データ

開発工数を**表 3** に示す．なお，SA 手法での機能数は各レベルのデータフロー図の中の詳細化されていないプロセスの数である．一方，従来手法での機能数は機能仕様書中に機能項目として記述した機能の数である．したがって，SA 手法での機能の方が細かいため，従来手法と比べて機能数が多い．

また，SA 手法チームでの SA 手法およびツールに

表 3 開発データ
Table 3 Characteristics of developed programs.

項目	SA 手法				従来手法	
	S1	S2	S3	S4	J1	J2
(1) 全工数 (人時)	706	701	874	696	716	585
(2) 分析・機能設計で抽出した機能数(個)	12	24	26	28	13	9
(3) 開発規模 (KLOC)	6.3	2.6	2.6	4.2	2.5	2.9
(4) 検査対象規模 (KLOC)	3.5	2.3	2.5	1.6	2.4	2.0
(5) 検査対象機能数(個)	7	13	23	12	8	7
(6) テスト・デバッグ工程でのテスト項目数(件)	162	405	407	215	459	288
(7) テスト・デバッグ工程での単位規模当たりのバグ数 (件/KLOC)	10.9	7.0	6.8	6.9	8.3	22.0
(8) 分析・機能設計からテスト・デバッグまでの生産性 (KLOC/人年)	10.8	7.2	6.2	5.0	6.8	9.1

ついての学習工数は，3日間の講習を含めて1人当たり約30時間であった．

3.2 品質分析

機能レビュ時の誤り発生率および受け入れテスト時の誤り発生率を分析した (2.1 節参照)．

[観察1] 受け入れテストでのプログラム品質は，SA 手法の方が従来手法より高い (有意水準 5%)．

SA 手法の受け入れテストでの平均誤り発生率 (19%) は，従来手法の平均誤り発生率 (36%) の約 1/2 であり，両者の間には統計的に有意な差が認められた．

[観察2] 機能レビュ時の仕様品質は，SA 手法の方が従来手法より高い (有意水準 10%)．

機能レビュ時の誤り発生率を分析したところ，SA 手法の平均誤り発生率 (53%) が従来手法の平均誤り発生率 (76%) より低かった．両者の間には，有意水準は 10% と低いが，統計的に有意な差が認められた．

次に，手法による影響をより細かく分析するため，機能レビュ時の誤り発生率をテスト種別に基づいて分析した．

[観察3] SA 手法による分析結果には，従来手法による分析と比較して機能に関する誤りが少ない (有意水準 5%)．

機能レビュ時における機能に関する平均誤り発生率については，SA 手法 (31%) の方が従来手法 (56%) より低く，統計的に有意な差が認められた．SA 手法では，日本語文章形式ではなくデータフロー図等の図形を用いて機能仕様書を形式的に記述するため，仕様書の厳密性が高くなり，かつ，チーム内レビュでの検証作業が容易になったためと考えられる．

[観察4] 機能レビュ時の誤り発生率と受け入れテストでの誤り発生率には正の相関がある (**図2**)．

受け入れテスト時の誤り発生率と機能レビュ時での誤り発生率の間の関係を分析した結果，両者の間の相関係数は 0.86 であり，強い正の相関があることが明らかになった．したがって，品質の高い分析・機能設計を行うと，最終的なプログラムの製品品質が高くなることが確認できた．

3.3 工数分析

各チームごとの各工程にかかった工数の割合を**図3**に示す．なお，問題によるプログラム規模の違いを吸収するために，検査対象プログラム規模を用いて正規化している．

[観察5] 各工程ごとの工数の割合に関しては手法間に有意な差がない (有意水準 5%)．

一般に，SA 手法では綿密な要求分析を行うため機能設計工程での工数割合が増加するが，一方，バグが減少することによりテスト・デバッグ工数が減少すると考えられていた ([仮説3])．しかし，分析・機能設計にかかる工数およびテスト・デバッグ工数には手法間で顕著な差が認められなかった (図3)．

[観察6] 分析・機能設計からテスト・デバッグまでの生産性について手法間に有意な差がない (有意水準 5%)．

検査対象規模に基づき生産性を比較したところ，両手法の間には有意な差がなかった(表3)．しかし，受

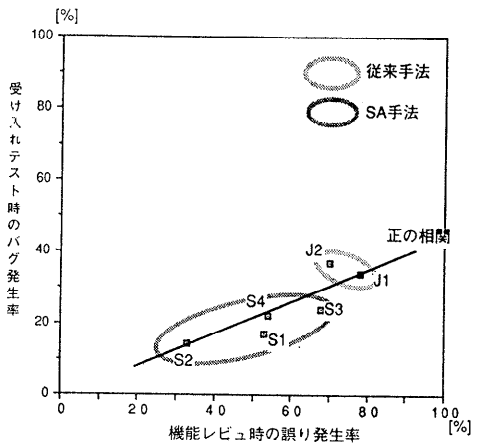

図 2 仕様品質とプログラム品質の相関
Fig. 2 Correlation between error rates at functional review and acceptance test.

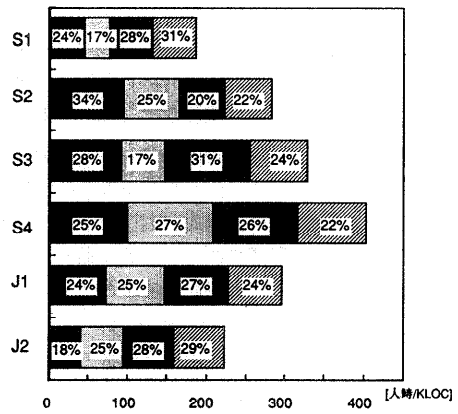

図 3 開発工数
Fig. 3 Efforts for each software development phase.

け入れテストで発生したバグ数については手法間に有意な差があるので，受け入れテスト時のバグ吸収工数を含めれば，SA 手法の方が従来手法よりも総工数が少なくなり，生産性は高くなると想定される．したがって，[仮説2]は成立すると予想されるが，厳密には今後の検証を待つ必要がある．

4. 分析実験結果

開発実験同様，工数および品質の観点から結果を分析した．

4.1 分析工数

開発実験と同様に，各分析対象ごとの作成規模の違いを吸収するため，機能仕様書の機能数で正規化した．ただし，SA 手法での機能数は各レベルのデータフロー図の中の詳細化されていないプロセスの数であり，従来手法と比べて細かい．したがって，SA 手法の方が抽出機能数が多くなるため，1機能当たりの分析工数は従来手法より少ない．各チームの分析工数の変化を図 4-1，図 4-2 に示す．

[観察7] 分析経験と分析工数の間には負の相関がある．

SA 手法，従来手法ともに，3回目の分析実験での1機能当たりの分析工数は1回目より少ない（有意水準 10%）．これは，両手法とも類似するトランザクションシステムを繰り返し分析することにより，必要な機能およびデータを抽出しやすくなったためと考えられる．さらに，SA 手法チームについては手法に慣れたことも影響していると考えられる．

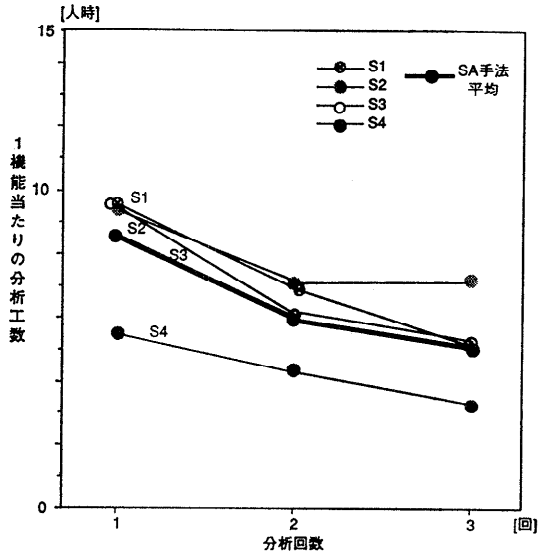

図 4-1 SA 手法を用いた場合の分析工数の変化
Fig. 4-1 Learning effect on efforts for analysis and preliminary design with SA/SD methodology.

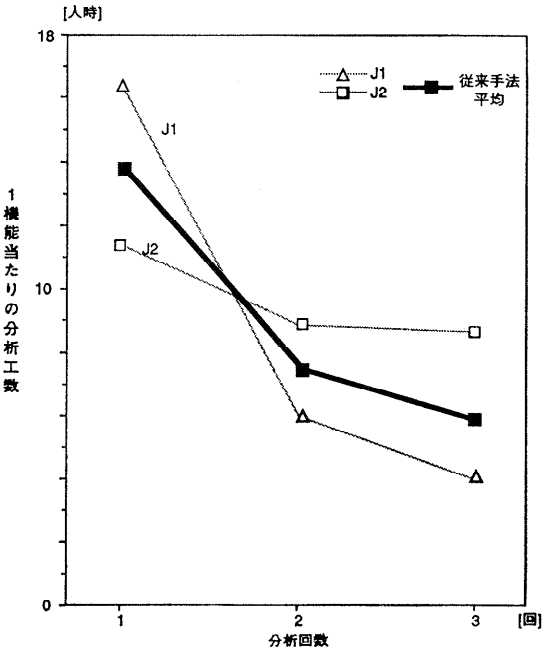

図 4-2 従来手法を用いた場合の分析工数の変化
Fig. 4-2 Learning effect on efforts for analysis and preliminary design with the traditional methodology.

4.2 分析品質

分析回数と分析品質との間の相関について分析した（表4）．

表 4　分析作業品質の習熟効果の検定結果
Table 4　Reliability test of learning effect at function analysis/design.

テスト分類	SA 手法	従来手法
(1)　機能に関する項目	◎	○
(2)　データに関する項目	◎	○
(3)　制約条件に関する項目	×	×
全　体	◎	×

[凡例]　◎：有意水準5%で3回目の方が1回目と比べて仕様品質が高い，○：有意水準 10% で3回目の方が1回目と比べて仕様品質が高い，×：有意な差がない．

[観察 8] SA 手法では，2回目の適用から習熟効果がある（**図 5**）．

図5からわかるように，SA 手法による分析では回数を重ねるほど誤り発生率が減少している．平均機能レビュ誤り発生率は1回目から2回目にかけて約 26% 減少し (53%→39%)，有意水準は 10% と低かったが両者の間には有意な差が認められた．したがって，2回目の品質は1回目の品質より高い．また，1回目から3回目にかけては約 45% 減少し (53%→29%)，両者の間には有意な差が認められた（有意水準 5%）．

一方，従来手法では1回目と3回目の平均機能レビュ誤り発生率がほぼ同じであった．特に，チーム J2 では2回目で誤り発生率が減少したが，3回目に再び誤り発生率が高くなった（図5）．

このように SA 手法では品質が次第に向上したのに対して，従来手法では一定しない結果になった．2つの設計法の間で問題分野に対する習熟性は同等なことから，実験結果の差は，設計手法に対する慣れの違いに起因すると考えて良い．すなわち，SA 手法では約 30 時間の講習会を受けさせた場合，2回目の適用から手法に対する習熟効果が得られ，品質が向上した．

次に，SA 手法の場合の習熟による影響をより細かく分析するため，1回目と3回目の機能レビュ誤り発生率についてテスト種別に基づいて分析した．機能に関する誤りは 64% 減少し (36%→13%)，データに関する誤りも 76% 減少した (62.5%→15%)．いずれの場合も，1回目と3回目の間には有意な差が認められた（有意水準 5%）．制約条件に関する平均誤り発生率は，1回目から3回目にかけて約 40% 減少した (67%→41%)．しかし，両者の間に有意な差はなかった．これは，機能とデータについてはデータフロー図およびデータ構造図という適切な図形表現があるが，制約条件についてはそのようなものがないことが影響していると推定される．

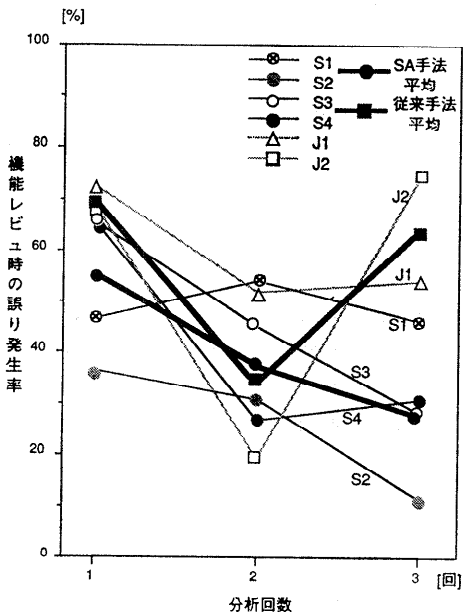

図 5　分析品質の学習効果
Fig. 5　Learning effect on error rate at functional review.

5. 考　　察

本実験では SA 手法の受け入れテストでのプログラム品質が従来手法の約2倍という結果を得た．一方，中条らは開発で発生するエラー数を測定し，SA 手法を用いた開発で検出するエラー数が従来手法の約 1/2 になるという結果を得ている[6]．2つの実験は測定時点が異なるが，SA 手法による品質向上効果がほぼ2倍という同じ値になったことは興味深い．

従来手法と比べ，SA 手法は習得に時間がかかるため，導入の初期では効果はあまり期待できないと考えられている[14]．しかし，本実験では約 30 時間の講習会を受けさせた場合，SA 手法の2回目の適用から品質が向上した．したがって，導入の初期であっても効果を期待することができる．

ヒューストン大学で CASE および SA 手法の適用経験の中で行ったアンケート調査によると，被験者は SA 手法の習得のためには，約1カ月かかると考えている[15]．これに対して本実験では，この約 1/5 の 30 時間程度訓練しただけで SA 手法を適用した．したがって，約1カ月程度の学習工数をかけると，より大

きな SA 手法の適用効果を得る可能性がある.

6. おわりに

本論文では，構造化分析／設計手法と従来手法を用いてソフトウェア開発実験を実施した場合，SA 手法を用いて開発したプログラム品質が従来手法を用いて開発した場合の約2倍に向上する（[仮説1]の実証），SA 手法を用いて開発した方がテスト・デバッグ工数が少ない（[仮説3]の実証），という結果を得た．なお，今回の実験では両手法の間で生産性の差は確認できなかった．これは受け入れテストで検出されたバグの吸収工数を計測しなかったことによる．ただし，受け入れテストで発生したバグ数は SA 手法の方が少ないので，このバグの吸収工数までを含めれば SA 手法の方が生産性が高くなると想定される．しかし，厳密には今後の検証を待つ必要がある（[仮説2]の保留）．また，SA 手法では約30時間の講習を受けた場合，分析品質に関し2回目の適用から習熟効果があるという結果を得た．

本実験では，小規模プログラムを小人数で開発する場合について SA 手法と従来手法を比較分析した．したがって，今回の結果を大規模ソフトウェア開発にも適用できるかどうかについては，今後の課題である．

参考文献

1) Yourdon, E.: *Modern Structured Analysis*, Prentice-Hall Inc. (1989).
2) Yourdon, E. and Constantine, L.L.: *Structured Design: Fundamentals of a Discipline of a Computer Program and System Design*, Prentice-Hall Inc. (1978).
3) DeMarco, T.: *Structured Analysis and System Specification*, Prentice-Hall Inc. (1978).
4) 山本修一郎，国立 勉：構造化分析・設計手法に基づくソフトウェア開発支援システム SoftDA の適用経験，*NTT R&D*, Vol. 40, No. 11, pp. 1413-1422 (1991).
5) Lempp, P. et al.: What Productivity Increases to Expect from a CASE Environment: Results of a User Servey, *ACM 27th Annual Technical Symposium*, pp. 1-7 (1988).
6) Nakajo, T.: A Case History Analysis of Software Error Cause-Effect Relationships, *IEEE Trans. Softw. Eng.*, Vol. 17, No. 8, pp. 830-837 (1991).
7) Nosek, J. et al.: User Validation of Information System Requirements: Some Empirical Results, *IEEE Trans. Softw. Eng.*, Vol. 14, No. 9, pp. 1372-1375 (1988).
8) 磯田定宏，黒木宏明：統合化 CASE システム SoftDA の機能—上流と下流の統合化—，コンピュータソフトウェア，Vol. 10, No. 2, pp. 1-12 (1993).
9) 山本修一郎，磯田定宏：ソフトウェアの構造化分析／設計支援システムを開発，NTT 技術ジャーナル，8月号，pp. 62-65 (1988).
10) Yamamoto, S. and Isoda, S.: A Transformation Algorithm for Generating Natural Structure Charts from Dataflow Diagrams, *Joint Conference on Software Engineering '92*, pp. 190-197 (1992).
11) 西永誠司，山本修一郎，磯田定宏：モジュール構造設計支援機能の提案，情報処理学会研究会報告，91-SE-85, pp. 9-16 (1991).
12) 下村隆夫，近藤彰一，高橋健司，国立 勉：ソフトウェアのテスト品質を把握し，作業を効率化，NTT 技術ジャーナル，2月号，pp. 47-48 (1991).
13) Bernstein, B.: *Software Psychology—Human Factors in Computer and Information Systems*, Wintherop Publishers, Inc. (1980).
14) Jones, T.C.: CASE's Missing Elements, *IEEE Spectrum*, Vol. 29, No. 6, pp. 38-41 (1992).
15) Loh, M. and Nelson, R.R.: Reaping CASE Harvests, *Datamation*, Vol. 35, No. 13, pp. 31-34 (1989).
16) 高橋健司，岡 敦子，山本修一郎，磯田定宏：構造化分析／設計手法の一評価，情報処理学会研究会報告，92-SE-10, pp. 57-64 (1992).
17) Basili, V.R. et al.: Experimentation in Software Engineering, *IEEE Trans. Softw. Eng.*, Vol. 12, No. 7, pp. 733-743 (1986).

(平成5年2月25日受付)
(平成5年9月 8 日採録)

岡 敦子（正会員）

1963年生．1986年慶應義塾大学管理工学科卒業．1988年同大学大学院管理工学専攻修士課程修了．同年日本電信電話株式会社入社．以来，ソフトウェア開発支援環境，CASE システムの研究開発に従事．現在，ソフトウェア研究所研究主任．

山本修一郎（正会員）

1954年生．1977年名古屋工業大学情報工学科卒業．1979年名古屋大学大学院情報工学専攻修士課程修了．同年日本電信電話公社入社．以来，ソフトウェア開発支援環境の研究開発に従事．現在，ソフトウェア研究所主幹研究員．電子情報通信学会，人工知能学会，IEEE 各会員．

磯田 定宏（正会員）

1946年生．1969年東京大学理学部物理学科卒業．1971年同大学理系大学院物理学専攻修士課程修了．1971年日本電信電話公社（現NTT）に入社．管理システム研究室長，ソフトウェア基礎技術研究部長などを経て，現在，NTTソフトウェア研究所主席研究員．1978～79年米国イリノイ大学計算機科学科客員研究員．ソフトウェア工学全般，特に CASE，設計法，再利用，メトリックス，プロジェクト管理，ヒューマンファクタ等に興味を持っている．1993年より情報処理学会ソフトウェア工学研究会主査．COMPSAC '91，ICSE '93，ICCI '93，ICRE '94 等のプログラム委員．電子情報通信学会，ソフトウェア科学会各会員，IEEEシニアーメンバー．