実践的ソフトウェア開発実習によるソフトウェア工学教育

松浦 佐江子†

Java や UML をはじめとするオブジェクト指向によるソフトウェア開発技術は多くの IT 企業から注目され，ソフトウェア開発技術者の育成が大学に求められている．産業界の求めるソフトウェア開発技術者を育成するためには，講義や簡単な演習のみではなく，ソフトウェア工学の知識を活用して様々な場面における問題解決能力を養うための実践的なソフトウェア開発を経験する必要がある．このような PBL（Project Based Learning）は学生が講義で得た知識を活用しながら主体的に問題解決を行う力を養う効果があることから，近年注目されている．しかし，大学学部教育において実践的なソフトウェア工学教育を行うためには，前提知識となる科目の教育，適切な課題設定，評価方法も含めた学生自身が遂行可能な授業設計とその支援環境の構築といった問題を解決する必要がある．我々は 2002 年度より，学部 3 年生を対象に後期の授業として，オブジェクト指向開発を用いたグループワークによるソフトウェア開発実習を実施している．本実習は半期をかけてある程度複雑なシステムを 8 人から 10 人程度のグループで開発するソフトウェア開発の全工程を体験する PBL である．本稿では，我々の実践的なソフトウェア工学教育を行ううえでの問題に対する取り組みを紹介し，5 年間の適用結果に基づいてその有効性について議論する．

Software Engineering Education Based on Practical Software Development Experiments

Saeko Matsuura†

Software development technology, especially both Java and UML started to attract wide interest of many IT companies. Moreover, it has been widely acknowledged that classes designed by utilizing PBL (Project-Based Learning) are effective in enhancing the problem-solving ability of university students. In PBL-based classes, students try to apply their knowledge to solve the problems by themselves; therefore, such classes are effective in improving problem-solving and communication abilities of students in software development. To conduct practical software engineering experiments, a plan of the experiment needs to be devised on four perspectives (education of prerequisite knowledge, selection of appropriate topics, a class design including assessment methods for grading students and computer supported environments). Since 2002, we have been planning and conducting group-work-based software development experiments as an approach to PBL. The aim of this class is to master software development and project management technologies based on Object Oriented Development. With a group of 8 or 10 students, a moderately complicated software system can be developed over half a semester. This experiment aims at educating undergraduate students to develop the faculty to become able software engineers. This paper describes the design of our practical software engineering education based on the experiment.

1. はじめに

ソフトウェア開発技術者の育成は大学に課せられた大きな義務である．経済産業省が策定している IT スキル標準や組み込みスキル標準等産業界から人材に要求されるスキルを獲得するための教育プログラムを展開する必要がある．我々は 2002 年度より，学部 3 年生を対象に後期の授業として，グループワークによるソフトウェア開発実習を実施している[1),2)]．本授業はオブジェクト指向開発に基づきソフトウェア開発技術とプロジェクト・マネージメント技術を学習することを目的とし，グループで半期をかけて要求分析から実装・テストまでのソフトウェア開発の全工程を体験する PBL（Project Based Learning）である．本稿では，実践的なソフトウェア開発実習を実施するうえでの問題点を議論し，我々の設計した実習の内容およびその実施結果について報告する．2 章では大学学部教育においてソフトウェア工学教育のための実践的なソフトウェア開発実習を行ううえでの問題点を議論する．

† 芝浦工業大学
Shibaura Institute of Technologies

3 章では本実習の目標を説明し，目標達成の要因をプロジェクトの課題・プロセス・メンバの観点から設定する．4 章では本実習の授業設計をこれらの観点から議論し，過去 5 年間の改善の状況を説明する．5 章では目標に照らして，本実習の有効性および教育効果について議論する．

2. ソフトウェア工学教育の問題点

ソフトウェア開発はグループで実施されるプロジェクトである．近年，大学教育においてもプロジェクトベースの教育の必要性が認識され，様々な授業が実施されている[3)〜5),12)〜15)]．特に文献 3)，12)，14) の研究では複雑な課題に対し，講義で得た知識を活用してソフトウェア開発の全工程を経験し，グループワークの問題点とそれに対する取り組みを体験できることがこうした実践的ソフトウェア工学教育の効果であると述べている．

我々も産業界が求めるソフトウェア開発技術者の育成には，実践的な演習を通してソフトウェア工学知識を修得する必要があると考え，2002 年度より実習を実施している．確かに体験することの意義は大きいが，開発されたソフトウェアは要求を満たすものか，体験により講義で学んだソフトウェア開発に関する知識が深まったか，一部の学生が作業をするのではなく，すべての学生が高いモチベーションで課題に取り組んだかという点がソフトウェア開発技術者を育成するうえでは重要であると考える．これらの事項に対して良好な結果を残せるかは，学生の前提知識の教育カリキュラム・履修学生数・時間と教育時期・指導者の構成・支援ツールの有無といった条件に依存する．

すなわち，大学の学部教育において実践的なソフトウェア開発実習を実施するためには次のような問題をふまえて実習授業の設計を行う必要がある．

(1) 学部 3 年後期までにある程度の規模のソフトウェア開発を行える前提知識を効率良く教授する必要がある．

(2) 学生が自ら課題を設定する場合，面白いが実現が困難であるか，非常に簡単な課題設定になる場合が多く，失敗あるいは達成感の少ないことがある．開発対象の社会的な有益性が感じられ，かつ実現可能な，ある程度複雑な課題を用意する必要がある．

(3) プロジェクトベースの教育は学生が講義で得た知識を活用しながら主体的に問題解決を行う力を養う効果があるが，グループで行動するため，授業としての個人評価が難しくなっている．そこで，学生の個々の作業理解や作業への参加意欲，貢献度を判断できる材料を教員が持つことを可能とする授業設計が必要となる．

(4) 履修学生の人数に依存するが，大学の授業であることから履修学生の人数は 50 から 100 人程度となる．複数のグループを対象とするため，グループごとにその作業および活動を支援する支援ツールおよび教員と TA（Teaching Assistant）によるサポート体制を十分準備する必要がある．

3. ソフトウェア開発実習の目標

本章では我々のソフトウェア工学教育の目標と目標達成のためのソフトウェア開発実習における要因について議論する．

3.1 ソフトウェア工学教育の目標

ソフトウェア開発の最終的な目標は要求を満たすソフトウェアを完成する，すなわち高品質な「プロダクト」を作成することであるが，この目標を達成するにはいつ，何をどのように行えばよいかという完成までの「プロセス」を工夫し，「グループメンバ（人）」がどのように協力して効率良く作業を行えるのかを考えながら，「プロジェクト（= 実習）」を進めることが重要である．我々はソフトウェア開発実習プロジェクトをプロダクト（成果物），プロセス（実行の仕方），グループメンバ（実行する人）のコラボレーションの 3 つの要素で構成されるととらえている．ソフトウェアの品質に関しては ISO/IEC9126：ソフトウェア品質特性 6 項目（副品質特性が 21 項目）が定められている．本学科におけるソフトウェア工学教育ではこれらの品質特性のうち，次の 3 つの観点から高品質なプロダクトを開発する能力を育成することを目標としている．

– ユーザの明示的および暗示的な必要性を満たす一連の機能を実現しているかを示す「機能性」（特に，明示された仕事に対する機能が適切であるかを示す「合目的性」と正しい結果および効果をもたらすことを示す「正確性」）
– ユーザがソフトウェアを使用するのに必要な労力や使用結果，使い勝手に関する「使用性」
– ソフトウェアに対して明示された改訂を行うために必要な労力に関する「保守性」

また，近年，エンタープライズ系だけでなく，組み込み系のシステム開発にもオブジェクト指向開発への期待が高まっている．本ソフトウェア開発実習においてもオブジェクト指向による開発技術の習得を目標とする．特にオブジェクト指向の特性である上流工程から下流工程までの機能要求のトレーサビリティ[16)] に

表 1 目標達成の要因
Table 1 Factors for the goal.

分類	因子	内容	関係
課題要因	分野	ソフトウェア開発の主な分野であるエンタープライズ系と組み込み系のアプリケーションを対象とする.	4.2
	リアリティ	現実のユーザが存在する，または，現実的な利用価値があるか否かにより「高」「中」「低」とする.	4.2
	要求の難易度	要求は A4 用紙 2 から 3 枚程度で定義する．機能性・使用性の観点からの難しさを「高」「中」「低」とする.	4.2
	技術の難易度	自力で学習しなければならない未習得技術. エンタープライズ系では Web アプリケーションの 3 層アーキテクチャを実現するための JSP/Servlet 等の技術とデータベース技術が必要であり，組み込み系ではシミュレータ用の GUI 技術または実機のファームウェア技術が必要である．言語は Java を用いるので，これらの技術も Java 関連技術を用いる.	4.1
プロセス要因	定義	各フェーズの時間的配分・フェーズの繰返しの有無・作業ごとの分担の有無・実質的な作業期間	4.1 4.4
	指針	プロセスの例題の有無・タスクの定義（カテゴリと内容）・タスクの客観的指標となるメトリクスの定義・インスペクションの実施とその指標	4.1 4.4
	作業モデル	作業計画・作業報告の内容と期日・コミュニケーション範囲と手段・データ共有	4.5
人的要因	前提知識	オブジェクト指向開発（UML・Java）・テスト技法・プロジェクト管理に対する習熟度	4.1
	グループ構成	グループ数・メンバ数・決定方法	4.3
	サポート体制	指導者数と役割・作業モデルおよび指針に対する支援ツール	4.6
	評価	プロジェクトの達成目標に対するグループと個人の評価方法	4.7

着目し，指導を行う.

3.2 目標達成の要因

ソフトウェア開発実習では，前述のソフトウェア工学教育の目標に従い，下記の 2 つの項目をプロジェクト目標とする.

A) プロダクトの品質（機能性・使用性・保守性）の作り込み
B) 履修者の知識（開発技術 + コミュニケーション能力）の向上

品質を作り込むためには開発に関する知識を持ち（前提知識），いつ，何をすればよいのか（プロセスの定義と指針）を知り，計画的に作業を行うことが必要である．さらにプロジェクトの制約（期間，人員等）の範囲で良い結果を残せるかには知識と方法だけではなく実行する人間の意欲が大きく関わってくるため，学生の課題取り組みへの意欲を向上させる因子が重要である．目標達成の要因を**表 1** のように分類し，4 章において，各因子に対する本実習における設定内容を説明する．表 1 の「関係」に各因子に関わる項目を示す．そして 5 章において，目標達成に対し，これらの因子がどのように関わっているかを考察し，本実習の設計の有効性を議論する.

4. ソフトウェア開発実習の授業設計

4.1 前提知識の教育カリキュラム

実践的なソフトウェア開発実習を行うためには最低限以下の項目について学習している必要がある.

– プログラミング基礎
– オブジェクト指向開発技術による分析・設計・プログラミング（Java，UML）
– 開発プロセス（要求分析・システム分析・システム設計・実装・テスト）とレビュー技術
– コミュニケーション

3 年後期に行うソフトウェア開発実習の前提として，上記の観点から下記のように授業を計画，実施している．本実習の履修生は 1 年次後期と 2 年次前期に，C 言語の講義・演習各 2 コマを，2 年次後期に Java 言語の授業ならびに演習各 1 コマを履修する．また，数名を除いた学生がオブジェクト指向開発ならびに UML（Unified Modeling Language）に関する講義 1 コマを 3 年次前期に履修し，小規模な例題に関する要求分析・システム分析・システム設計を行う．ただし，設計書に基づいた実装とテスト計画ならびにテストの実施については未経験である．これらの中から PBL の前提となる特徴的な教育方法について下記に示す.

表 2 年度別課題要因
Table 2 Factors for the subjects by the year.

年度	分野	リアリティ	要求の難易度	技術の難易度	講習・調査等
2002	A-1	中	中	JSP/Servlet+DB	なし
	B-1	中	中	GUI	なし
2003	A-1	中	中	JSP/Servlet+DB	サーバの提供
	B-1	中	中	GUI	自動販売機の内部見学
2004	A-2	高	高	JSP/Servlet+DB	職員へのインタビュー・サーバの提供
	B-1'	中	中	GUI	自動販売機の内部見学
2005	A-2	高	高	JSP/Servlet+DB	職員へのインタビュー・サーバの提供
	B-1	中	中	GUI	自動販売機の内部見学+メーカからの説明
	B-2	低	低	LEGO MINDSTORMS	ファームウェア（leJOS）の講習
2006	A-2	高	高	JSP/Servlet+DB	職員へのインタビュー・サーバの提供
	B-1	中	中	GUI	自動販売機の内部見学+メーカからの説明
	B-2'	中	中	LEGO MINDSTORMS	ファームウェア（leJOS）の講習

- 2 年次後期の Java プログラミング演習

本演習では，課題プログラムを自動テストするテストプログラムを用いて，教員が学生 1 人 1 人に直接質問しながらプログラムを評価する審査方式を取り入れている[6]，これにより，プログラムをテストすることを教示し，プログラムの説明能力の訓練を行っている．

- 3 年次前期ソフトウェア設計論における演習

本授業では UML による要求分析からシステム設計・実装までの講義を行い，演習として小規模な例題による分析・設計を行う．ここで，他の学生が行った分析・設計ドキュメントを評価することにより，レビュー技術の学習を行っている．

- 3 年次前期の応用プログラミング実習（授業時間は 8 コマ分）

本実習では 1 つの課題に対して，インタフェースを標準入出力から GUI へ変更する作業をテストケースの定義とテスト実行を含めて段階的に学習することにより，仕様変更におけるプログラムの変更容易性やテストケースの妥当性について学習する．テスト実行の際に 2 人で 1 組となり，テストケースとプログラムを交換して，プログラムの理解やコミュニケーション能力を養う．カリキュラムの都合上，2002 年度は本授業の代わりに，オブジェクト指向言語ならびに Java によるプログラミング演習を実施した．また 2003 年度は本実習を半期すべて（28 コマ分）実施している．

- 学科の共通科目におけるグループ学習

本学の総合科目「創る」やシステム工学演習では与えられたテーマや学習課題に対してグループで問題解決を行うグループ学習を実施している．これらの授業を通して，学生はグループ学習におけるコミュニケーションを経験している．

4.2 課題の設定

ソフトウェア開発の対象には大きく分けてエンタープライズ系と組み込み系のアプリケーションがある．実習を始めるにあたり，2 つの分野から，半期（3 カ月強）に複数人で開発が可能であり，ある程度の複雑度を持つ課題を下記のように設定した．年度別の課題設定の相違は**表 2** に示すとおりである．

エンタープライズ系課題

A-1：販売管理システム（情報処理学会オブジェクト指向シンポジウムでのモデリングの共通課題）

A-2：芝浦工業大学大宮校舎会議室予約システム（本学において手作業で行われている会議室の予約管理をシステム化するため，現状の手続きとシステムへの要件を定義した．A-1 に比べて，システム化の要件もユーザから獲得し，定義する必要がある）

組み込み系課題

B-1：自動販売機のロジックとシミュレータ（情報処理学会オブジェクト指向シンポジウムでのモデリングの共通課題にシミュレータの要件を追加した）

B-1'：B-1 と同じ要求であるが，ロジックとシミュレータを別々のグループで担当し，1 つのシステムを共同開発することとした．

B-2：LEGO MINDSTORMS のロボットを使用した「荷物の自動搬送システム」（経済産業省，平成 17 年度産学協同実践的 IT 教育促進事業「産学協同実践的 IT 教育基盤強化事業」に採択された「組み込みソフトウェア教育プログラム開発・実証」のテーマである．ここでは実機を用いて行う組み込みシステムの開発の現実的な問題点を解決する必要がある）

B-2'：B-2 とはハードウェアおよび搬送路の制約，要求されるサービス等のシステム要件が異なる．

「販売管理システム」は企業の販売管理方法に関する知識が乏しいため，学生には理解が難しい点があった．そこで，2004 年度からはより身近な課題として「芝浦工業大学会議室予約システム」を課題とした．この課題では実際に本学で行っている業務をシステム化することが目標であり，職員・教員・学生がシステムのユーザとなる．特に業務を遂行する職員に直接インタビューを行うことで，実習中に顧客の要求確認も行えるようにした．一方，自動販売機は日常的に利用している機械であり，学生はこれが何をするものであるかは熟知している．さらに，実際の内部の仕組みをメーカの方から説明を受けることができるように計画した．本実習ではこのように，身近でかつ社会的な有益性が感じられる課題を設定した．

4.3 グループ構成

受講生は 2002 年度が 121 人，2003 年度が 66 人，2004 年度が 84 人，2005 年度が 69 人，2006 年度が 55 人である．学生が各自課題の中から 1 つを選択する．その後 2002 年度はくじ引きにより，課題 A-1 を 15〜17 人，課題 B-1 を 11〜12 人の 9 つのグループに分けた．2003 年度からは前期に実施した実習授業の成績によって 9〜10 人のグループに分けた．2003 年度は 7 グループ，2004 年度は 9 グループである．2005 年度は課題 A-2 が 6〜7 人で 4 グループ，課題 B-1 が 7〜8 人で 3 グループ，課題 B-2 が 5〜6 人で 3 グループとした．2006 年度は課題 A-2 が 8 人で 4 グループ，課題 B-1 が 8 人で 2 グループ，課題 B-2' が 7 人で 1 グループである．各グループにはリーダ 1 人，サブリーダを 1 から 2 人おくこととした．その他の役割は各グループで決定した．ただし，2002 年度はメンバ数が多いため，分析段階ではサブグループで作業することとした．

4.4 開発プロセスの定義と指針

学生は前提知識の修得により，ソフトウェア開発の流れ（フェーズ）の概要は知っているが，小規模なプログラムしか開発したことがないため，具体的な作業に基づいて 3 カ月強の開発の見通しを立てることができない．そこで，実習の目標を達成するためには，開発プロセスを明示する必要がある．ここでの開発プロセスとはどのような作業を，何を目的として，いつ，どのような順序で行い，何を成果とすればよいのかを示すものである．

4.4.1 フェーズと成果物

本実習ではオブジェクト指向開発（UML，Java）をソフトウェア開発方法として用いる．開発プロセスは要求分析，システム分析，システム設計，実装，テストの 5 つのフェーズに分ける．各フェーズの目的ならびに成果物は**表 3** のとおりである．

表 3 フェーズの目的と成果物
Table 3 Goals and products for each phase.

フェーズ／目的	内容
要求分析 与えられた要求を分析し，ユーザの求める機能，システムの形態，性能等を明らかする．	• 要求定義（自然言語） • ワークフロー（アクティビティ図*） • 用語集 • ユースケース図* • ユースケース記述 • 概念モデル • ユーザインタフェース・イメージ図
システム分析 分析された要求を満たすために必要なシステムの構成要素と構成要素の役割を明確にする．	• ユースケース図* • ユースケース記述 • クラス図* • シーケンス図* • ステートチャート* • [オブジェクト図*] • [コラボレーション図*]
システム設計 分析されたシステムをプログラミング言語で実装するために必要な仕組みを定義する．	• クラス図* • シーケンス図* • パッケージ図* • 統合テスト仕様 • 単体テスト仕様 • テスト計画書
実装 設計で作成されたシステムの構成をプログラミング言語を使って定義し，実行可能なモジュールを作成する．	• ソースコード一覧 • コード • Java API ドキュメント • インストールガイド • 操作説明書
テスト 実装されたプログラムが要求された機能や性能を満たしているかを検査する．開発の方向とは逆の方向に，段階的に確認を行う．	• テストクラスコード一覧 • テストクラスコード • [パッケージ図*] • テストケース一覧 • テスト結果一覧

[]はオプション．*は UML に従う．

各授業時間は 2 コマ（計 180 分）で，授業回数は 14 回（2004 年度より 15 回）である．第 1 回にグループ編成およびガイダンスを，最終回に発表会（説明とデモンストレーション）を行った．また，2003 年度以降は授業終了後に最終検査（教員が用意した統合テストケースについてテストする）を実施した．実質的な作業期間は 2002，2003 年度が 14 週，2004 年度が 15 週であり，2005，2006 年度は冬休みを含めて 17 週であった．

各フェーズは通常 3 週間程度とし，各フェーズの終

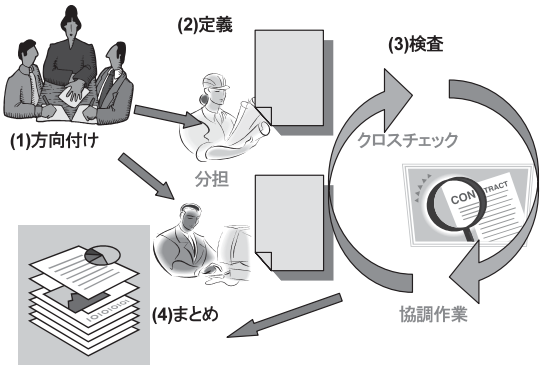

図 1 タスクのカテゴリ
Fig. 1 Category of tasks.

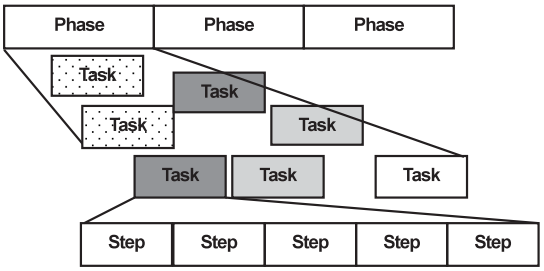

図 2 フェーズの構成
Fig. 2 Composition of a phase.

了時に表 3 の成果物を納品する．2006 年度は開発目標を 1 期と 2 期に分け，イテレーションを実施した．

4.4.2 タスクとタスクのカテゴリ

各フェーズの目標は表 1 の成果物を完成することである．次にそれぞれのフェーズで行うべき作業のガイドラインを定義する．ガイドラインでは，以下のように開発プロセスの目標を詳細化し，各フェーズにおける最終目標を達成するために行うべき作業「タスク」を規定した．

タスクは**図 1** に示すようにフェーズのはじめに行う作業（方向付け），プロダクトを記述する作業（定義），定義されたプロダクトに含まれる間違いや矛盾を発見する作業（検査），修正された定義に基づいて成果物を作成する作業（まとめ）の各カテゴリに分類される．そしてフェーズは**図 2** のように各カテゴリに属するタスクの列で構成される．「検査」としては各仕様書のインスペクション[7),8)] とソースコードのテストを義務づけた．

付録 1 に本実習におけるタスク一覧を示す．たとえば，システム分析フェーズでは，システムの構成要素と構成要素の役割を明確にするために，以下のタスクを適宜組み合わせて作業を行う．「方向付け」として要求分析フェーズで定義したユースケース記述・概念モデルを用いてオブジェクトの抽出を行う．「定義」として各ユースケースのシーケンス図の記述，オブジェクトのステートチャート図の記述を行いながら，オブジェクトの属性や操作を抽出しクラス図を定義する．「検査」ではシーケンス図・ステートチャート図・クラス図のインスペクションを行う．「まとめ」によって表 1 の成果物を作成する．下線部が各カテゴリのタスクである．フェーズは「方向付け」⇒「定義」⇔「検査」⇒「まとめ」といった構成になる．ここで ⇔ は繰返しを示している．また，オブジェクト指向の特徴として前段のフェーズの成果物も修正され，ともに進化していく．各成果物における整合性だけでなく，複数の図間の整合性を保証する作業が「検査」である．

各タスクにはその対象となる成果物の特性を測るためのメトリクスを導入する．客観的な計測結果をグループ全体で共有して，作業状況の把握や問題点の発見に利用する．さらに，タスクは学生自身が定める具体的な作業から構成される．これをステップと呼ぶ（図 2 参照）．高品質なプロダクトを作成するためにはプロセスを工夫することが必要である．タスクとして規定された作業の目的を理解し，適切なステップによりタスクを実践することが求められる．

4.4.3 インスペクション

ソフトウェア開発において経験を積まないと分かりにくいことの 1 つは，定義したプロダクトの良し悪しを判断することである．モデル記述言語で 1 つ 1 つのプロダクトを記述することはできるが，プロダクトの完成度や，プロダクト間の整合性がとれていないために，工程における大きな後戻りが発生することになり，このことがソフトウェア開発の現場における工程遅延を生む原因となっている．インスペクションやレビューはこのような問題を解決する 1 つの手段である．タスクの概念は 2004 年度より実習に導入されたが，「検査」のタスクとしてインスペクションを導入した．

インスペクションの対象は各フェーズで「定義」されるプロダクトである．インスペクションは個別のプロダクトおよびその構成要素を対象とした閉じた検査と，シーケンス図やステートチャート図を用いたクラス図の検査のように別のプロダクトを利用した検査がある．本実習ではインスペクションを学生間で行う．「定義」を行った学生が「依頼者」として複数人の「評価者」に「検査」を依頼する．教員はインスペクションのガイドライン（インスペクションの種類ごとに検査すべき項目とその内容および修正方針）を与える．

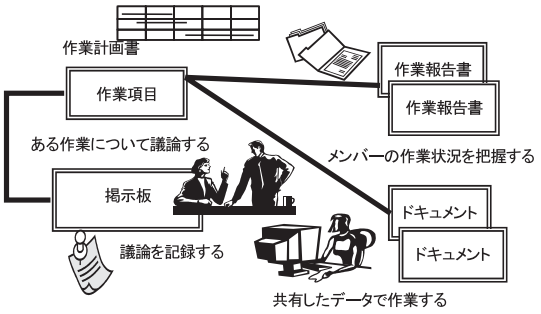

図 3 作業モデル
Fig. 3 Work model.

たとえば，**付録 2** はシステム分析フェーズのインスペクションガイドラインである．

4.5 作業モデル

開発プロセスの定義に従い，**図 3** の作業モデルを想定する．作業は作業計画によって運営される．作業計画は前述のタスクから作られる具体的な作業項目で構成され，作業項目ごとに該当作業に関する議論およびデータの共有が行われる．グループは週 1 回以上作業計画を更新し，各学生は作業項目ごとに週に 1 回以上の作業報告をする義務がある．

4.6 支援ツールとサポート体制

プロジェクトの目標を達成するためには 2 章で述べた問題 (4) に対する取り組みが必須である．学生の作業と教員の指導を支援するグループワーク支援システムおよび教員と TA の役割について説明する．

4.6.1 グープワーク支援システム

我々は 4.4 節および 4.5 節で述べた開発プロセスおよび作業モデルに基づき，グループメンバの作業を支援するグループワーク支援システムを開発し，実習において運用している[1)]．本システムでは作業内容の報告・各種連絡・議論および質問の場を提供することにより，非同期型グループウェアにおけるアウェアネス（協調作業者の作業状況，作業履歴等の認識）[10)] を実現する．学生はこの場を通じて，授業時間以外にも情報を共有し，他の学生の作業状況を知ることができる．また，授業時間以外の作業を支援するために，学内外を問わず，自由な時間にアクセスできる環境として本システムを実現した．

本システムでは担当者および作業期間を定めた作業項目に基づく定期的な作業報告，会議の議事録，掲示板による議論，仕様書の書庫の 4 つの場を管理する以下の機能を Web アプリケーションとして提供している．

① 個人情報管理（パスワード・連絡先）
② 作業計画書の作成・更新・参照

図 4 作業報告書
Fig. 4 Work report.

③ 作業報告書の作成・更新・参照（**図 4** 参照）
④ ファイルのアップロード・ダウンロード
⑤ 掲示板の利用による連絡・議論
⑥ 議事録の利用による決定事項の確認
⑦ インスペクション報告
⑧ 教員・TA および管理者からの連絡
⑨ 教員・TA および管理者への質問・連絡

本システムは実習に参加するすべてのグループにグループごとの場を提供している．学生は自グループの場にのみ参加できるが，教員および TA はすべてのグループの場を閲覧することができる．これにより，教員および TA はつねに全グループの作業状況を把握することが可能である．教員および TA は ⑧ において履修者全員・特定のグループ・特定の個人を選択して連絡を行うことができる．これにより，全体へのアドバイス，グループへのアドバイスを適切に行うことが可能である．また学生も ⑨ において個人としての質問あるいはグループとしての質問を区別できることから，情報の共有を状況に応じてコントロールすることができる．

4.6.2 教員と TA の役割

本実習は教員 2 人（2006 年度は 1 人），TA 2 人（2004 年度は 4 人，2005 年度は 6 人，2006 年度は

3 人）で担当した．教員は本実習を遂行するのに必要な資料（開発プロセスの説明，システムの説明，サンプルプログラム等）を準備し，授業時間に簡単な全体説明を行う．また，グループワーク支援システムを通じて作業状況を把握し，システムを通じて随時コメントを行う．授業時間には各グループを訪問し，進捗に応じてコメントおよび質問を受け付ける．

本実習は 2002 年度より実施したため，学部 3 年で本実習を経験した大学院生が 2004 年度より TA として実習に参加している．2004 年度より TA をグループのアドバイザ（あくまで質問に答える役目であり，プロジェクトマネージャではない）として 1 つのグループを担当させた．アドバイザはグループの状態だけでなく，成果物についても責任を持ってアドバイスする必要がある．この役割の目的は学生にとって教員より身近な相談相手をおくことと TA の能力の養成である．

4.7 評価モデル

3 章で述べたとおり，ソフトウェア開発実習プロジェクトをプロダクト（成果物），プロセス（実行の仕方），グループメンバ（実行する人）のコラボレーションの 3 つの要素で構成されるととらえ，これらの達成目標に対してグループおよび個人の観点から評価を定義し，重みづけを行い総合的にプロジェクトの評価を定義した[2]．**表 4** に示すとおり，プロジェクト評価モデルは定義・達成目標・評価データ・評価基準・評価対象（個人またはグループ）の各要素により定義する．

ここで「最終レポート」は実習の終了時に学生が本実習の総括として提出するものであり，質問項目は以下のとおりである．なお，() 内は 2005 年度の各項目の質問数である．

- 技術の理解度について（10）
- ソフトウェア開発に対する意識について（8）
- 各フェーズの実習内容について（18）
- グループワークについて（13）
- 実習の内容設定について（8）
- オブジェクト指向開発について（3）
- 実習に対する取り組みの自己評価（10）

「ログ」とは 4.6 節で述べた支援ツールのログデータを指す．また，理解度確認テストでは自グループの提出ドキュメント内のシーケンス図の読み方とクラス図との整合性の確認方法を個別にテストした．

本実習は今年度で 5 年目であり，これまで述べてきた各要因に関して毎年実施内容の改善を行ってきた．各フェーズの成果物および発表会はすべての年度で評価したが，その他本実習で用いた評価データを年度別に**表 5** に示す．

表 4 評価モデルの構成要素
Table 4 Elements of project evaluation model.

プロダクト	
定義	各種ドキュメントおよびシステム
達成目標	機能性・使用性・保守性の観点からプロダクトの完成度を高め，整合性を保持できる．UML およびシステムに関して十分理解できる．
評価データ	教員のプロダクトに対するレビュー・プロダクトの測定・統合テストケースによる最終検査・理解度確認テスト
評価基準	開発方法に基づく評価 ex. 文献 9)・テストの合否
プロセス	
定義	プロダクトを作成するための作業
達成目標	機能性・使用性・保守性の高いプロダクトを作成するための作業を理解できる・実践できる
評価データ	作業報告のレビュー・作業ログのレビュー・最終レポートの評価
評価基準	理解度・実践における達成度
コラボレーション	
定義	プロダクトをプロセスに従い作成する過程（作業モデル）における個人の行動
達成目標	グループの一員として作業に参加し，協調して，メンバとしての責任を果たす
評価データ	作業計画，議論のログのレビュー・議論，作業報告，ドキュメント投稿の回数・学生の相互評価（貢献度）
評価基準	ログの測定値における達成度・役割の実践度

5. 考　　察

本章では本研究で提案した授業設計に基づく評価モデルについて議論した後，3 章で述べた下記の 2 つの目標の達成度について要因との関連の観点から本実習の有効性と問題点を最終レポートならびに成果物の評価に基づいて述べる．

A) プロダクトの品質（機能性・使用性・保守性）の作り込み
B) 履修者の知識（開発技術 + コミュニケーション能力）の向上

5.1 関連研究との比較

最終レポートにおける質問「実習を行って，自分にとって得るものはあったと思うか？」に対して，**図 5** の結果が得られた．5 年間を通じて約 91%から 95%の学生が「得るものがあった」と答えている．ここで縦

表 5 評価データ
Table 5 Evaluation data.

年度	支援ツールログ			確認テスト（個人）	最終検査（グループ）	最終検査（個人）	最終レポート質問数（貢献したメンバへの投票を含む）
	作業計画・作業報告（問題点・考察）・掲示板・ファイル共有	作業報告（メトリクス・ステップ）	インスペクション				
2002	×	×	×	×	×	×	137
2003	○	×	×	×	○	×	99
2004	○	○	×	×	○	×	74
2005	○	○	○	○	○	○	72
2006	○	○	○	○	○	○	56

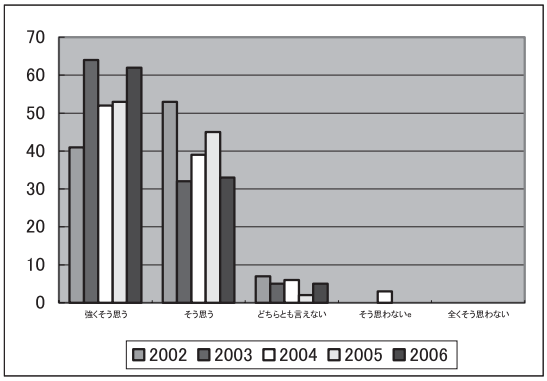

図 5 実習に対する満足度
Fig. 5 A feeling of satisfaction of students.

軸は回答者のパーセンテージを表している．本実習においても，関連研究 3)〜5)，12)〜15) と同様に，体験することの意義は確認された．

一方，これらの関連研究では 2 章の (3) で述べた評価に関する報告がほとんどない．グループワークではグループの成果に対する評価は成果物やプレゼンテーション等で可能であるが，個人のグループワークへの貢献度や理解度の評価についてはどの関連研究でも明らかではない．文献 4) ではソフトウェア開発プロジェクトとは異なり，データ構造とアルゴリズムの教育にグループ学習を適用している．このような場合には最終段階でテストを実施して個人の理解度を評価することができるが，複雑なソフトウェア開発では開発に時間がとられるため同時に長時間の試験時間をとることが難しい．本実習における個人の確認テストや最終テストは教員が質問文とその評価基準を定義し，TA と分担して実施することで，短時間でテストを行うことができた．本研究では実習の授業設計により評価モデルを明らかにし，これにより個人評価をふまえた成績評価を行うことが可能になった．

Hayes ら[11] の研究ではグループ作業におけるグループおよび個人の評価に必要な 5 つの項目について述べている．本評価モデルでは，これらの項目はすべて含んでいる（詳細は文献 2) を参照のこと）ことから，グループ作業における個人の評価方法としては妥当であると考える．

本実習では UML ツール（Jude）を利用した．文献 3)，4) でも既存のケースツールを導入しているが，習熟するのに時間を要し，使いこなせない場合がある．また，あまり高度な機能は初心者には難しい．グループワークの支援システムは文献 5)，15) と同様に独自に開発した．本実習で用いたグループワーク支援システムは 2002 年度に本実習を経験した学生が卒業研究で開発したものである．毎年，このツールを研究課題とする学生が改善を行っており，本実習には不可欠なものである．しかし，継続的な改善および運営は学生だけで行うには限界があり，大学内でのソフトウェア開発支援体制が必要であると考える．

5.2 プロダクトの品質の評価

5.2.1 評価モデルの観点

表 6 は 2002 年度から 2006 年度のプロダクトの成績分布である（点数は 100 点満点である）．「1〜10」が各グループの点数であり，「平均」は年度ごとのグループの平均点である．グループの点数は，各フェーズのプロダクトの記述度合いや最終検査において，当初予定した機能を正しく提供しているかという観点から評価を行った結果である．本実習の目標である「使用性」ならびに「保守性」の観点からは評価できる状態にはなっていなかった．これらについては，高すぎる目標であった．

一方，「成績平均」は学生の取得平均点である．成績分布ではグループワーク支援システムを導入して，作業ログを収集した結果の成績の方が点数のばらつきが生じていた．さらにプロダクトの個人評価を導入した 2005 年度では理解度の低い学生が明確になっている．

表 6 成績分布
Table 6 List of marks for the experiment.

年度	平均	1	2	3	4	5	6	7	8	9	10	不合格者数	UML未履修者数	成績平均	比率
2002	73	80	78	78	75	75	72	71	66	62	−	5	4	73.39	4:3
2003	81.6	93	88	85	82	81	72	70	−	−	−	5	4	70.64	3:4
2004	75.9	96	88	81	78	75	74	72	68	51	−	7	7	70.33	1:2
2005	64.7	78	77	72	69	66	63	62	56	52	52	14	13	62.8	1:4
2006	81.7	91	84	84	83	82	81	67	−	−	−	0	0	79.1	1:4

実際に 2005 年度に不合格になった学生は，確認と最終の個人評価が平均点（61.5 と 62.5）以下であり，他のメンバの投票による貢献度加点もない．一方，不合格者が 0 人である 2006 年度における同様の確認と最終テストの結果の平均点は 84.3 点と 80.9 点であった．表 6 に示すとおり，各年度の不合格者の大部分が UML に関する講義を未履修であった．UML に関する知識がないため，作業についていけず履修を放棄した学生もいた．また，不合格者のいたグループはグループ点も低く，未履修者のいることがグループ内での作業の足を引っ張っていたことがうかがえる．

2003 年度の平均点が高いのは，4.1 節で述べたとおり，本実習の履修者すべてが受講した前提となる 3 年前期の応用プログラミング実習の時間が他の年度に比べて 3 倍強あったことが理由として考えられる．この実習では「オセロゲーム」を題材に GUI を含めたオブジェクト指向の特徴を生かした Java プログラミングを十分な時間をかけて学習することができた．一方，2006 年度は UML に関する講義の未履修者がいなかったことが点数の高い理由として考えられる．これらのことから，前提知識（要素技術の教育およびプロセス要因におけるプロセスの指針に対する理解）の反復学習は重要であり，今後も改善を考えていきたい．

なお，ここで「比率」はグループ得点と個人得点の比率である．2003 年度と 2006 年度はプロダクトの平均点がほぼ同じであるが，学生の平均点は 2003 年度の方が低い．また比率から見ると，2003 年度はグループの点数で評価が上がった学生が多いが，2006 年度は個人評価の比率が高いにもかかわらず成績の平均が高いのは個人の努力も十分にあったことが推測される．このような学生の成績は教員および TA によるグループメンバの授業時の作業観察結果とも合致している．

5.2.2 課題設定の観点

最終レポートの質問「課題は実習の題材として適切であったか？」に対しては**図 6** の結果となった．プロセスの指針が明確になった 2004 年度以降の方が，「大変難しい」と感じる学生が幾分減少している．

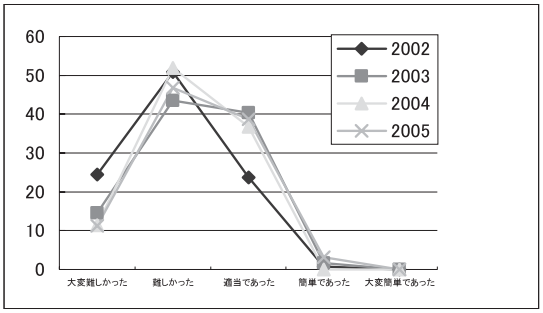

図 6 課題の難易度
Fig. 6 Difficulties of the subjects.

これまでの実習における成果物の内容に関しては以下の問題点があると考える．課題 A-1，A-2 および課題 B-1 はソフトウェアのみで実現するため，モデルの切り分けが分かりにくい．たとえば，課題 A-1，A-2 は Web アプリケーションであるため，学生の関心は JSP，Servlet による Web ベース GUI とデータベース接続に向き，本質的に重要な販売管理や会議室の予約ロジックを十分に分析できないことが多かった．課題 B-1 ではシミュレータは課題 B-2 のような実機ではなくソフトウェアであることから，自動販売機を構成するハードウェア部品をクラスとしてモデルを構築することはできるが，シミュレータとその制御ロジックを切り分けて分析することができていなかった．2004 年度には，要求分析を共通に行い，システム分析からシミュレータと制御ロジックを異なるグループにより開発させることを試みたが，ハードウェア部品のクラス群を中心として制御とシミュレータのモデルができるのではなく 1 つのクラスを介して情報をやりとりするモデルになっていた．1 つの課題を複数のグループの共同開発によって開発することは，結局互いのシステムの刷り合わせに時間をとられ，柔軟なモデルを作成することができず，品質の向上には効果がなかった．2005 年度からはドメインチャート（**図 7** は課題 A-2 のドメインチャート）をはじめに提示することで，実装技術に引きずられないモデリング，組み込みであることを明示できるモデリングを行うことを示唆した．

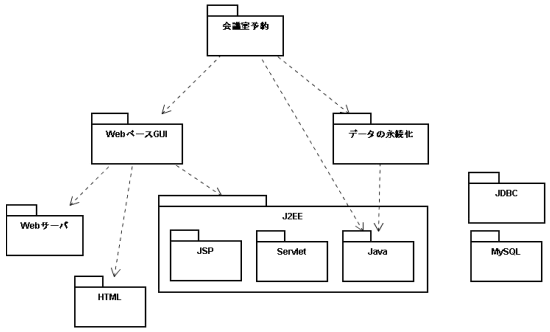

図 7 ドメインチャート
Fig. 7 Domain chart.

一部を除いてはモデルに改善の様子が見られた．

2005 年度の課題 B-1（LEGO MINDSTORMS のロボットを使用した「荷物の自動搬送システム」）は前述のとおり「組み込みソフトウェア教育プログラム開発・実証」のテーマで採択されたものである．2002 年度から課題 B-1 により組み込みソフトウェア教育の試みを実施してきたが，実機を使用することはなかった．そこで，この課題により，組み込みソフトウェア開発教育の改善を行うこととした．しかし，課題設定が他の課題 A-2，B-1 に比べ単純であったことも影響して，学生はハードウェアや走行路を固定されたものととらえて，ソフトウェアのみでシステムへの要求を満たそうとしたため，本来の組み込みソフトウェアのモデルとしては柔軟性がなく不十分であった（実際に分析・設計モデルを十分構築しなくてもプログラムができてしまうため，モデルは形式的な提出物でしかなかった）．そこで，追加の仕様変更を行うことにしたが，1 つのグループは走行路に工夫を行い，うまい解決策を見つけることができたが，プログラムの大幅な変更を余儀なくされた．2006 年度は課題 B-1 の内容をハードウェア・走行路・ソフトウェアの協調的なモデル化が可能なものに改善して実施し，他の課題と同程度の複雑度を持つ課題とすることができ，分析・設計モデルも十分検討された．

5.2.3 開発プロセスの観点

本項では学生が開発プロセスをよく理解して実践することによりプロダクトの品質が向上したかについて考察する．最終レポートにおいて，開発プロセスに対する理解について下記の 4 つの質問を行った．以下は 2005 年度および 2006 年度の回答結果である．

(1) 開発プロセスにおいては，フェーズ・フェーズの段階を表すカテゴリ・フェーズを構成するタスク，タスクを構成するステップ，タスクにおけるメトリクス，インスペクション項目がある．これらの概念を理解して作業を行っていたか？

十分理解していた（3%，5.4%），ある程度理解していた（58%，59.1%），どちらともいえない（22%，31.9%），あまり理解していない（17%，1.8%），まったく理解していない（0%，1.8%）

(2) 理解できなかった概念はどれか？（複数選択可）

a. フェーズ（3%，1.2%）
b. フェーズの段階を表すカテゴリ（13%，8.5%）
c. フェーズを構成するタスク（8%，7.3%）
d. タスクを構成するステップ（14%，17.1%）
e. タスクにおけるメトリクス（46%，47.6%）
f. インスペクション項目（9%，9.8%）
g. なし（9%，8.5%）

(3) 実習終了後，これらの概念に対する理解，すなわち，ソフトウェア開発プロセスに関する理解（何をいつ，何のために行うのか）は深まったか？

強くそう思う（8%，23.2%），そう思う（83%，69.8%），どちらともいえない（8%，9.6%），そう思わない（0%，3.6%）・まったくそう思わない（0%，0%）

(4) ソフトウェア開発プロセスに関する理解が深まったと思う人（上記の質問で「強くそう思う」「そう思う」と回答した人）はその理由は何か？（複数選択可）

a. 各フェーズにおけるタスクが規定されていたから（16%，16.7%）
b. タスクのメトリクスが規定されていたから（8%，5.2%）
c. タスクがカテゴライズされていたから（10%，10.3%）
d. インスペクションを行ったから（16%，20.1%）
e. 3 カ月の開発を経験したから（31%，26.4%）
f. 作業計画書はタスクに基づいて作成する必要があったから（11%，9.8%）
g. 作業報告書はメトリクスの値を報告する必要があったから（2%，3.4%）
h. 作業報告書では作業ステップを報告する必要があったから（6%，17.5%）

質問 (4) の回答にもあるように，3 カ月の開発経験はソフトウェア開発への理解を深めたと考えられる．タスクの導入やインスペクションの実施も理解を深める一因となったと考える．また，開発プロセスの定義に関してもメトリクス以外はある程度理解ができ，これらの概念に対する意識が出てきたように見受けられるが，タスクのメトリクスに関しては指導が不十分であった．

タスクの導入の効果

作業状況から判断して，個人あるいは数人でコー

ディングするのではなく，実装・テスト段階での分担作業および助け合いが実施できたグループの割合は，2002 年度 1/9（グループ数比）に対し，2003 年度は 4/7，2004 年度 8/9，2005 年度は 7/10，2006 年度は 6/7 であった．タスクの概念を導入した 2004 年度以降の方が，良い結果が得られていると考える．これはタスクの導入により，作業計画を漠然とではなく，作業すべき内容と順序・担当者を考慮して考えられるようになったために，メンバのプロジェクトに対する自覚ができてきたのではないかと思われる．

インスペクションの導入の効果

2004 年度からはインスペクションという作業を義務付け，各フェーズ終了時にはドキュメントの確認方法をグループごとにコメントした．上記の回答にも見られるようにインスペクションを意識して行っている学生も見られたが，ガイドラインに従った妥当な指摘が必ずしも行われていない．これは，UML に関する知識不足に起因する．インスペクション支援ツールで蓄積されたデータ（2005 年度指摘数 312，2006 年度 355）のうち，妥当な指摘の割合は 2005 年度が 44%，2006 年度が 66%であった．2005 年度はシステム分析フェーズ終了時に行った整合性に関する理解度確認テストにおいて，100 点満点で 80 点以上が 32%，60 から 80 点が 26%，60 点未満が 42%であった．一方，2006 年度は 80 点以上が 67%，60 から 80 点が 31%，60 点未満が 2%であった．結果に大きな差があるが，1 つの原因はグループ総数（2005 年度は 10 グループ，2006 年度は 7 グループ）にあると考えられる．7 グループの場合には 2 コマの授業時間内に直接対話しながらコメントをすることが可能であったが，10 グループの場合には時間内には実施できなかった．その場合には，次の時間に回すか，掲示板によるコメントとなり，フォローが十分でなかったことも適切なインスペクションを行えず，プロダクトの品質に影響を与えたと考える．一例ではあるが，UML に関して比較的深い知識を持った学生がインスペクションにおける「評価者」の役割を専門に引き受けていたグループがあった．確かに指摘内容は妥当なものが多かったが，指摘された側も十分に理解できないと結果には反映されない．

イテレーションの導入の効果

2006 年度はたとえば課題 A-2 では第 1 期開発として「CUI + ファイルベースの会議室予約システム」を開発し，第 2 期開発として「Web ベース GUI +（DB | ファイルベース + 排他制御）の会議室予約システム」を開発するという計画で実習を行った．最終レポートの質問「本実験を第 1 期開発と第 2 期開発に分けて行った結果であてはまるものを複数回答せよ.」に対しては以下の回答があった．

a. 対象システムに対する理解が深まった（22.1%）
b. 作業に対する理解が深まった（21.3%）
c. 作業計画が立てやすくなった（16.9%）
d. 作業時間を短縮できた（2.2%）
e. システムの品質が向上した（14.7%）
f. 何もメリットはなかった（4.4%）
g. 作業が 2 度手間になり分かりにくかった（15.4%）
h. システムの品質が低下した（2.9%）

このように，繰り返し開発を行うことは，理解を深めるために役立っているようである．実際には，第 1 期の開発がテストにより十分でなかったため，完全に第 2 期の目標を達成できないグループがほとんどであった．しかし，第 1 期の開発で，会議室の予約に関するロジックを十分開発できたため，機能的には良いシステムとなったといえる．問題が複雑である場合には，どこを優先すべきかを考慮して，段階的な開発を指導する必要があると考える．

開発プロセスの定義および指針においてタスク・インスペクション・イテレーションはプロダクトの品質向上を意識させるという観点からは有効であった．しかし，これらの概念の理解は十分ではない．

5.2.4 グループ構成の観点

最終レポートの質問「グループの人数は適当であったと思うか？」に対する回答は**表 7** のとおりである．

本実習は選択必修科目であることから，受講生の人数を制限することができない．年度ごとに人数の変動があるが，これまでの経過からグループとして活動できる人数は 8 人程度までであると考えられる．実際には 5 から 6 人くらいが適切であるかもしれないが，グループ数が増大するために各グループに均等に教員からのフィードバックを与えることが難しい．また，2002 年度はグループの人数が多かったこともあり，最終的にグループとして活動できたグループは 1 グループのみであった．2003 年度以降は前期の別の実習の成績によってグループ編成を行ったが，5.2.3 項で述

表 7 グループ人数
Table 7 Validity of the number of group members.

年度	yes	no	メンバー数
2002	75%	25%	11–17
2003	87.1%	12.9%	9–10
2004	71.3%	28.7%	8–11
2005	71.2%	28.8%	5–9
2006	89.1%	10.9%	7–8

表 8 オブジェクト指向技術の理解度
Table 8 The Degree of understanding for method and languages.

年度	オブジェクト指向開発方法					モデリング言語(UML)					プログラミング言語(Java)					ライブラリ等(Swing, AWT, JSP, JDBC)				
	a	*b*	*c*	*d*	*e*	*a*	*b*	*c*	*d*	*e*	*a*	*b*	*c*	*d*	*e*	*a*	*b*	*c*	*d*	*e*
2002(前)	10	55	26	18	8	11	62	20	16	8	7	40	28	34	8	1	17	19	40	33
2002(後)	19	71	27	2	0	26	74	15	3	0	15	55	29	18	2	6	36	36	27	11
2003(前)	1	36	16	9	0	0	30	18	14	0	5	32	18	7	0	0	21	17	20	4
2003(後)	14	44	4	0	0	12	45	5	0	0	13	44	5	0	0	2	41	18	0	1
2004(前)	3	22	27	24	4	2	34	26	24	4	4	23	21	29	3	1	7	12	42	18
2004(後)	13	52	11	2	2	16	50	13	1	0	9	54	15	1	1	3	35	33	7	2
2005(前)	0	30	8	20	3	0	19	15	7	3	3	16	16	20	5	0	10	15	21	13
2005(後)	12	41	5	2	0	8	45	7	0	0	9	31	17	7	0	3	26	22	9	0
2006(前)	2	25	13	16	0	1	26	15	13	1	3	33	11	9	0	−	−	−	−	−
2006(後)	12	39	5	0	0	14	38	4	0	0	14	36	6	0	0	−	−	−	−	−

べたとおり，これによりまったく機能しないグループはほとんどなかったと考える．表 7 の回答からは学生自身はグループ人数に対して「適切である」と考えているように思われる．「適切ではない」という理由には「知識の不足から実際に作業できない人がいるため，少数でよい」といった意見があり，ここでも前提知識の影響が出ている．

教員からのプロダクトに関するフィードバックは重要であり，毎時間直接学生の反応を見ながらコメントできることが問題を早期に解決するためには望ましい．そこで，グループの人数は 10 人以上でなければ，グループ数によって構成を決めた方がよい．

5.2.5 グループワーク支援システムの観点

グループワーク支援システムの 1 グループあたりの使用状況平均数（掲示板投稿数・ファイルアップロード数・作業報告数）は，2003 年度が（122・135・183），2004 年度が（349・287・440），2005 年度が（99・398・188），2006 年度が（184・403・326）となっている．グループ単位で見ると使用頻度にはかなりの差が出ており，グループの作業の進め方には違いがあるが，コミュニケーションツールとしては十分機能していると考える．本システムがなかった 2002 年度は 5.2.3 項で述べたとおり，個人あるいは数人でコーディングするのではなく，実装・テスト段階での分担作業および助け合いが実施できたグループは自力でサーバを構築し，掲示板を利用した 1 グループのみであった．その他のグループは直接話し合える数人でのみ開発するという結果になった．しかし，支援システムを通じての連絡では「連絡を確認しない人がいる」といった問題も生じ，結局は個々の学生の参加意欲を高めなければならない．

5.3 知識の向上に関する評価

本実習により，オブジェクト指向開発技術に関する知識やコミュニケーション能力が向上したかは客観的には測ることはできない．開発プロセスに関する理解については 5.2.3 項で述べたとおりである．

最終レポートにおいて，オブジェクト指向開発方法・モデリング言語（UML）・プログラミング言語（Java）・ライブラリ等について，実験前（実験後）にどの程度理解していたか？（理解しているか？）という質問を行った．**表 8** はこの回答結果である．ここで，回答は以下の 5 段階である．

a. 十分理解していた（いる）
b. ある程度理解していた（いる）
c. どちらともいえない
d. あまり理解していなかった（いない）
e. まったく理解していなかった（いない）

この回答からは表 8 において太字で示したように，学生は実験前に比べて実験後は理解が深まったという意識を持っている．しかし，ライブラリ等授業では十分な指導がなく自力で学習する必要があった技術に関しては，全員が実際に使用することもなかったため，終了後にも理解していない学生が多いと思われる．

最終検査は 2003 年度から実施しているが，2003, 2004 年度はグループを対象として，個別の検査は行わなかった．グループを対象とする場合には，主にリーダが説明するか，実際にプログラムを作成した学生が質問に答えることが多かった．グループによってはすべてのメンバが適宜答えることもあり，コミュニケーションがうまく行われていることが分かった．2005 年度からは，確認テストと同様に TA と分担して，教員が決定した質問を個別に行った．学生のシステムに対

する理解の程度がよく分かったが，教員がすべての学生の理解度を直接把握することができないという問題もある．TA から詳細な報告は受けるが，TA 自身の評価者としての能力もさらに養う必要がある．

知識の向上には学生の学習意欲が大きく関わる．確認テストのような個人の評価を導入することで，学生の実習への参加意欲の高まりが見受けられた．

5.4 オブジェクト指向開発の教育

最終レポートの質問「オブジェクト指向開発はソフトウェア開発に役立ったと思うか?」に対しては 89%から 97%の学生が「役立った」と回答している．「役立った」理由としては，分析・設計・実装のつながりによりシステムに対する理解が深まったことや，作業分担が行えたこと等をあげている．「役立たなかった」と回答した理由は，「前提としている授業を履修していなかったため，役立てるに至らなかった」というものであった．

また 3 年前期の応用プログラミング実習ではインタフェースを規定した仕様変更が容易なモデルに基づくプログラミングの訓練を行っているが，このときのやり方を今回の実習でも適用できると考え，実践した学生がいた．オブジェクト指向でのモデリングは仕様変更や再利用によってその利点が理解できることから，実習でのヒントとなる授業を設計する必要がある．本実習をソフトウェア工学教育のターゲットとして，1 年次からの授業全体に開発プロセスやオブジェクト指向技術のポイントを取り入れた授業設計を行うことが望ましい．本実習の結果をふまえて検討を行っている．

6. ま と め

2002 年度から改善を行いながら，ソフトウェア開発実習を行ってきた．ソフトウェア工学の理論を理解するためには，実践が不可欠である．しかし，本稿で述べたように，実践には 1 年次からのカリキュラムを含めた入念な授業設計，支援環境と TA の育成を含めたサポート体制の強化が必要である．本実習で使用しているグループワーク支援システムは本実習の履修生が考案し，実習における支援を実現するために毎年 2 人の学生が卒業研究として取り組んでいる．また，実習の経験を就職活動に活かす学生も多いと聞いていること等から，1 回目の経験としては本実習の試みは成功していると考える．しかし，技術的には未熟であり，本実習を行った学生からは「もう 1 回やれば，もっとうまくできる」という感想をよく聞くことからも，プロダクトの品質の向上のためにはさらなる改善が必要である．以下に，5 年間の実習で得られた結果をまとめ，今後の改善について述べる．

課題要因に関して

- 課題は複雑度が低いと分析・設計を怠ってプログラムだけが完成し，変更要求への対応が難しくなる．課題要因の「リアリティ」ならびに「要求の難易度」は「中」以上にすべきである．ただし，自力で学習しなければならない技術がある場合に，学生が本質的なロジックをきちんと分析せずに，その学習に時間をとられすぎる場合がある．この場合には，2006 年度に実施したように開発プロセスに段階を設け，第 1 期にはロジックの確立を目標に開発を行わせるといったプロセスの「定義」や「指針」で対処するのが良い．

プロセス要因に関して

- 1 つの課題を 2 つのグループで共同開発することは複雑なプロジェクトの初心者には難しい．結局は 2 倍のメンバで開発することになり，効果的ではない．
- タスクの導入は作業すべきことを明確にすることで，効果があった．しかし，オブジェクト指向の学習には繰返しが必要である．たとえば，オブジェクト指向の特性である上流工程から下流工程までの機能要求のトレーサビリティを学習するためには，先行する授業において，インスペクション指針にあるような整合性に関する知識を習得させることを「前提知識」として繰り返し教育すべきである．トレーサビリティを理解しているメンバが多くなれば，妥当な指摘を行うことができ，インスペクションの効果はあがると考えられる．

人的要因に関して

- 最終レポートの質問「グループワークでは何が難しかったか」に対して，多くの学生が，「理解していないメンバの影響で作業が進まない」ことを問題としていたことからも，人的要因が全体に与える影響は大きい．2006 年度は実習を独立した 2 つの授業に分け，教員が分担し，前提条件となる授業を履修しているか否かにより履修学生を振り分けたため，結果としてはプロダクトの完成度が高かった．また，これまでに比べて均等にコメントする時間がとれるようにもなり，学生への教員からのフィードバックが充実したことも影響していると考えられる．
- グループ数は授業時間内に全グループに対して直接コメントできる数が望ましい．履修者数にも依存するが，7 グループが限度であると考える．
- 本実習を経験した学生を TA として採用し，アドバイザの役割を与えた．このことは反復学習という意味でも TA にとっての良い学習の場となっている．2 年目にはさらに積極的なアドバイスを行っていることが

見てとれる．人にもよるが，受講している学生にとっても気軽に相談できる存在になっている．

- 個人評価を明確にしたことで学生の参加意欲が向上し，グループワークへの良い影響が見られた．

技術力の向上には学生の学習意欲の向上と実習による達成感を得ることが重要である．以上の結果をふまえ，今後は以下の改善を行いたいと考える．

技術的な未熟さを補うために，前提知識のカリキュラムにプロセス要因の「指針」に関する反復的学習を取り入れる．また，現在前提授業における学習をサポートする支援ツールを検討中である．これらにより，全体の知識レベルの向上を図る．一方，可能であれば，前提科目を必修とすべきである．前提科目を履修していない学生が実習を履修する場合には，同じグループに配属しないといった配慮が必要である．これらにより，前提知識をそろえ「習っていない」という学生をなくし，個々の学生の学習意欲を向上させたいと考える．

謝辞 本実習の履修生である芝浦工業大学システム工学部電子情報システム学科の皆様と TA の皆様およびグループワーク支援システムの開発に携わったソフトウェア工学研究室のメンバ諸氏に感謝いたします．

参 考 文 献

1) Matsuura, S.: Practical Software Engineering Education based on Software Development Group Experiments, *E-Learn 2005 — Proc. World Conference on E-Learning in Corporate, Government, Healthcare and Higher Education*, pp.1701–1708 (2005).
2) Matsuura, S.: An Evaluation Method of Project Based Learning on Software Development Experiment, SIGCSE2006, *Proc. 37th ACM Technical Symposium on Computer Science Education*, pp.163–167 (2006).
3) Alfonso, M.I. and Mora, F.: Learning Software Engineering with Group Work, *Proc. 16th Conference on Software Engineering Education and Training* (2003).
4) Davenport, D.: Experience Using a Project-Based Approach in an Introductory Programming Course, *IEEE Trans. Education*, Vol.43, No.4, pp.443–448 (2000).
5) 櫨山淳雄，中野秋子：ソフトウェア設計・開発グループ演習教育のためのコミュニケーション支援システム，情報処理学会論文誌，Vol.42, No.11, pp.2550–2561 (2001).
6) Matsuura, S., Atsuta, S. and Fujiwara, T.: Programming Exercise Evaluation Method using Code Review for Improvement of Programming Skill, *Proc. 8th IASTED International Conference on Computers and Advanced Technology in Education* (*CATE 2005*), pp.89–94 (2005).
7) Fagan, M.E.: Design and Code Inspections to Reduce Errors in Program Development, *IBM Systems Journal*, Vol.15, No.3, pp.182–211 (1976).
8) Votta, L.G.: Does Every Inspection Need a Meeting?, *ACM SIGSOFT Software Engineering Notes*, Vol.18, No.5, pp.107–114 (1993).
9) Travassos, G.H., Shull, F., Fredericks, M. and Basili, V.R.: Detecting defects in Object-Oriented Designs: Using Reading Techniques to Increase Software Quality, *Proc. OOPSLA*, pp.47–56 (1999).
10) Dourish, P. and Bellotti, V.: Awareness and Coordination in Shared Workspaces, *Proc. ACM CSCW'92*, ACM, pp.107–114 (1992).
11) Hayes, J.H., Lethbridge, T.C. and Port, D.: Evaluating Individual Contribution Toward Group Software Engineering Projects, *Proc. 25th ICSE*, pp.622–627 (2003).
12) Berndtddon, M.: Anaklyzing Course Configurations for Teaching Object-Oriented Modeling and Design, *IEEE Trans. Education*, Vol.48, No.2 (2005).
13) Cheston, G.A. and Tremblay, J.P.: Integrating Software Engineering in Introductory Computing Courses, *IEEE Software*, Vol.19, No.5 (2002).
14) Demuth, B., et al.: Experience in Early and Late Software Engineering Project Courses, *15th Conference on Software engineering Education and Training* (2002).
15) Chiken, K. and Hazeyama, A.: Awareness Support in Group-based Software Engineering Education System, *10th Asia-Pacific Engineering Conference*, IEEE CS (2003).
16) Braude, E.J.: *Software Engineering: An Object-Oriented Perspective*, Wiley (2000).

付録1 タスク一覧

※各タスクの内容に関しては別途資料を配布している.

	要求分析フェーズ	システム分析フェーズ
方向付け	• ワークフローの作成 • シナリオの作成 • 用語集の作成 • B:自動販売機の調査 • C:LEGO MINDSTORMS, leJos の調査 • 開発ツールの決定 • ドキュメント規約の設定	• オブジェクトの抽出 • クラスの責務の検討
定義	• 要求定義(自然言語・ワークフロー) • 非機能要求の定義 • ユースケース図の作成 • ユースケース記述の作成(対象ユースケース) • 概念モデルの作成 • A:B:ユーザインタフェース・イメージの作成 • A:職員へのインタビュー • C:搬送路の定義	• シーケンス図の作成(対象ユースケース) • ステートチャート図の作成(対象オブジェクト) • クラス図の作成 • ユースケース図の修正 • ユースケース記述の修正(対象ユースケース) • [オブジェクト図の作成] • [コラボレーション図の作成] • C:搬送路・ハードウェア構成の定義
検査	• ユースケース図のインスペクション • ユースケース記述のインスペクション(対象ユースケース)	• シーケンス図のインスペクション(対象ユースケース) • ステートチャート図のインスペクション(対象オブジェクト) • クラス図のインスペクション
まとめ	• 要求仕様書の作成	• システム仕様書の作成

	システム設計フェーズ	実装フェーズ	テストフェーズ
方向付け	• 設計目標の設定 • A:Web アプリケーション構築の調査 • B:GUI ライブラリの調査	• コーディング標準の設定	• テスト計画の確認
定義	• クラス図の作成 • シーケンス図の作成(対象ユースケース) • パッケージ図の作成	• コーディング	• 統合テストケースの定義(対象ユースケース|対象ステートチャート) • 単体テストケースの定義(対象パッケージ)
	• 統合テスト仕様の定義(対象ユースケース|対象ステートチャート) • 単体テスト仕様の定義(対象パッケージ)		
検査	• クラス図のインスペクション		• 統合テストの実行(対象テスト番号) • 単体テストの実行(対象テスト番号)
	• 統合テスト仕様のインスペクション(対象ユースケース|対象ステートチャート) • 単体テスト仕様のインスペクション(対象パッケージ)		
まとめ	• システム設計書の作成	• ソースコード一覧の作成 • Java API ドキュメントの作成	• テストクラスの作成 • テスト結果報告書の作成
	• テスト仕様書の作成 • テスト計画書の作成	• 操作マニュアルの作成	

付録 2　インスペクションガイドライン（システム分析フェーズ）

インスペクションの種類	インスペクション項目の内容と修正方針
シーケンス図のインスペクション（対象ユースケース）	• 形式の確認： UML 記法に従っているかを確認する． ⇒ UML 記法に従って修正する． • シーケンスの十分性： すべてのユースケースに対して，その基本フロー・代替フロー・例外フローのすべてに対するシーケンスを定義できているか，各フローに定義されているステップをもれなく定義しているかを確認する． ⇒ オブジェクト・メッセージを追加する，または削除する，または名前を変更する． • シーケンスの整合性： １つのユースケースに対するシーケンス図として整合性が取れているか（基本・代替・例外フロー間での矛盾はないか）を確認する． ⇒ オブジェクト・メッセージを追加する，または削除する，または名前を変更する． • シーケンスの妥当性： 定義したシーケンスがそのユースケースを行うのに十分な操作とそれを行うオブジェクトから構成されているかを確認する． ⇒ オブジェクト・メッセージを追加する，または削除する，または名前を変更する．
ステートチャート図のインスペクション（対象オブジェクト）	• 形式の確認： UML 記法に従っているかを確認する． ⇒ UML 記法に従って修正する． • 状態の妥当性： 状態をもつと考えられるオブジェクトのすべての状態を列挙できているか，システムにおいて意味のない状態はないかを確認する． ⇒ 状態を追加する，または削除する，または名前を変更する． • 状態遷移の妥当性： すべての状態に対して，各状態におけるアクティビティは適切か 状態が遷移するイベントは適切か・遷移に付随する操作であるアクションは適切か・遷移における条件であるガードは適切かを確認する． ⇒ アクティビティ・アクション・イベント・ガードを追加する，または削除する，または名前を変更する．
クラス図のインスペクション	• 形式の確認： UML 記法に従っているかを確認する． ⇒ UML 記法に従って修正する． • クラス名の妥当性： ユースケース分析で理解したシステムの概要と対比してクラスの名前が開発するシステムを説明するのに妥当であり，十分であるかを確認する． ⇒ クラスの名前を追加・削除・変更する． • シーケンス図による確認： すべてのシーケンス図と対比し，クラスの操作が正しいクラスに属しているかを確認する． ⇒ クラスの操作の修正． • ステートチャート図による確認： すべてのステートチャート図と対比し，クラスの操作が正しいクラスに属しているかを確認する． ⇒ クラスの操作の修正． • クラスの責務の確認（名前）：

	クラスの属性および操作の名前がクラスの名前と対比して妥当であるかを確認する． ⇒ クラス・属性・操作の名前を変更する． • クラスの責務の確認（役割分担）： クラスに属する属性や操作の数が妥当であるか（多すぎて，複数の責務をもっている．少なすぎてクラスとして存在する意味がない）を確認する． ⇒ 責務を分割する．責務を統合する．クラス・属性・操作の名前を変更する． • クラス間の関係の妥当性： クラス図においてクラス間の静的な関係（ロール・多重度）がクラスの意味を考えたうえで妥当であるかを確認する． ⇒ 関係を修正する． • 概念モデルとの整合性： クラス図と概念モデルの整合性はとれているかを確認する． ⇒ クラス・属性を変更する．

(平成 18 年 11 月 30 日受付)
(平成 19 年 5 月 9 日採録)

松浦佐江子（正会員）

1985 年津田塾大学大学院理学研究科数学専攻博士課程単位取得退学．博士（情報科学）早稲田大学．(株）管理工学研究所研究員，津田塾大学学芸学部情報数理科学科非常勤講師を経て，2002 年より芝浦工業大学システム工学部電子情報システム学科助教授，2006 年より同大学同学部教授．ソフトウェア開発環境および設計方法論，オブジェクト指向開発技術に関する研究に従事．1993 年情報処理学会研究賞受賞．2004 年 FIT 論文賞受賞．日本ソフトウェア科学会，人工知能学会，IEEE 各会員．