講義紹介：「実践的シミュレーションソフトウェア開発演習」

居駒幹夫

東京大学大学院工学系研究科非常勤講師

１．はじめに

2009 年より情報基盤センターで機械系の大学院生を対象に実施している科目「実践的シミュレーションソフトウェア開発演習」を紹介する。

本科目の目標は、信頼性、保守性、移植性に優れた大規模シミュレーションソフトウェアを開発するスキルを受講者に身に付けさせることである。この目標を達成するため、複数人の受講生がチームを組み、情報基盤センターのスーパーコンピュータや教育用のマシン環境を活用してシミュレーションソフトウェアの開発演習を行う。情報処理系の学科では、ソフトウェア開発プロジェクトを学生主体で実行させ、体験的にソフトウェア工学を学ぶ PBL（Project Based Learning）という教育方法が普及しつつある。本科目は、この考えをシミュレーションソフトウェアの分野に適用し、さらに、大学側からは流体力学、分子動力学の専門家、企業側からソフトウェア工学の専門家が講師として参加し、講師間でもコラボレーションをしながら受講生を指導するという野心的な試みである。

本稿では、本科目の内容紹介と、「企業側からの非常勤講師」という立場でみた課題認識、教育の実施状況、4 年間の経験を経て現在感じていることを述べる。筆者は、企業に籍を置くソフトウェア工学の専門家であり、その視点に偏った記述、大学側から見ると当たり前のこと又は見当外れのことも少なからずあると思うが御容赦お願いしたい。

２．企業側から見た科目の背景、課題認識

産業におけるシミュレーションの重要性は近年特に増している。ハードウェア製品の開発の場合、スーパーコンピュータの処理能力増大によって、これまでは実験室で大掛かりな実験装置を使って試験するしか手段がなかったものが、コンピュータ上での高精度シミュレーションが可能になってきた。昨今では、自然現象をシミュレートするだけでなく、これまで知られていなかった新たな知見を得る手段としてもシミュレーションが活用されている。また、従来、数時間から数日かかって結果が出ていたプログラムが、最近のスーパーコンピュータでチューニングすることにより、同じプログラムで数分、数秒で終わるような時代になっている。「量が質に転化する」という格言通り、抜本的な性能向上により実験室での作業が主でシミュレーションが従であった過去から、徐々に、主従が逆転しつつある現状であろう。

一方、筆者の専門であるソフトウェア工学とは、「ソフトウェアの開発、運用、保守に体系的、専門別、量化可能なアプローチを応用、すなわち工学をソフトウェアに応用」[1]する学問である。現代、あらゆる分野でソフトウェアが活用されており、また、その規模は増大する一方である。さらに、社会の基幹システム全体の信頼性が、ソフトウェアへの依存度を高めており、ソフトウェア工学の重要性も増している。定義から分かるように、ソフトウェア工学とはプログラムの書き方レベルだけの話ではない。書店で多く見かけるプログラミング系の書籍から得られる

[1] IEEE ソフトウェア工学用語集 1990 年版より

スキルだけで、大規模ソフトウェア － 複数の開発者が数ヶ月から数年をかけて初期開発し１０年以上保守できるようなソフトウェア － の開発は不可能である。例えば、超高層ビルを設計、施工するのと同様に、大規模ソフトウェアの開発においても、管理すべき膨大なモノ、時間、コストなどを適度な粒度に分割し、最適制御し、最終的に目的に合ったものとして完成させるための知識、方法が必要である。そういった、”Software Industrial Engineering”ともいうべき分野のソフトウェア工学の知識が今後さらに重要になってくる。

このような背景が正しいとしても、機械系の学生に対してシミュレーションソフトの開発演習を行う必要性は自明ではない。この点に関して筆者が課題だと考える以下の二点について吟味してみたい。

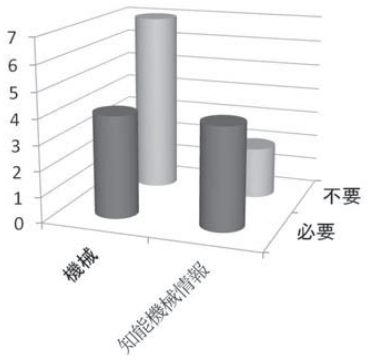
課題１　機械系の大学院生にシミュレーション系のソフトウェアを題材にソフトウェア工学的な知識を教える価値があるのか。学生は必要としているのか。

課題２　現在良く使われているシミュレーション系のソフトウェアは、ソフトウェア工学の立場から見て問題があるのか

課題１：機械系の学生に対するソフトウェア工学の必要性

大学の工学系から製造業の製品開発や研究に従事している卒業生の多くは、シミュレーションに関連する業務に就いている。ただ、シミュレーション関連と一言で言っても、汎用的なシミュレーションパッケージをエンドユーザの立場で使うレベルの業務も多いのは事実である。しかし、実験装置の場合でも、汎用的な実験装置で製品開発/研究が可能な場合もあれば、ある研究、ある開発用に実験装置を特注/自作しなければいけないようなものもある。同様に、新規性の高い研究、世界をリードする製品で使われるシミュレーションには、その用途に沿った大規模ソフトウェアの開発が必要な場合も少なくない。さらにいえば、そうして開発したソフトウェアを他用途にも活用可能なように汎用パッケージ化することが、その組織、企業の優位性を確保するためにも重要になってきている。

情報処理系の学科では大規模ソフトウェアを書くための知識スキルを修得するための教育は、行われているのに対してシミュレーションに関連する業務に就く工学系、特に機械系の学生はそのような訓練を積んでいない。しかし、現実には機械系を卒業した多くの学生は、ソフトウェア工学系の知識を必要とする業務に就いている。第１図は、ここ数年で東京大学の機械系を卒業し､筆者の勤務先に入社した 17 名のうち､どの程度がソフトウェア工学のスキルが必要な職場に配属されているかを示したグラフである。人数ベースで調査してみると、シミュレーションと言う観点で言うと、17 名全員が関わっている。また、ソフトウェア工学のスキルが入社後にも役に立つような学生が 17 名中 8 名、47%いることが分かった。ここで、分母の 17 名という数字は、入社した学生数である。実は企業側でも技術系の新卒採用を行う際



第１図　ソフトウェア工学的な知識を必要とする機械系卒業者の数

に、情報処理に関するスキルを見る[2]。機械系の学生が情報系の事業所に就職活動を行ったが，情報系の素養が不足していることが一因で不採用になっているケースもあると聞いている。

課題２：ソフトウェア工学からみたシミュレーション系ソフトウェアの評価

インターネットで調べてみると、シミュレーションの分野でも、商用パッケージだけでなく、オープンソースの応用ソフトウェア、ライブラリも広く流通している。また、そのほとんどが海外で開発されたものである。記述言語は、長い間、科学技術計算のデファクト言語であったFORTRAN も健在であるが、最近のものはC 言語、C++言語のものも少なくない。規模的にみると、汎用的なオープンソースのソフトウェアの多くは数万行から数 10 万行というかなり大規模なソフトウェアである。

ソースコードが参照可能なシミュレーションソフトウェアに対して、その大きなレベルでの構造や、小さなレベルでのソースコードの質、中長期的にソフトウェアを保守するための情報の有無などを精査すると、大きな課題に気付く。感覚的な評価ではあるが、全般的に、長期間にわたってソフトウェアを保守、拡張していくための性質（ソフトウェア工学の用語で「保守性：Maintainability」）が、他の分野のソフトウェアに比べて劣っているように見える。さらに、海外製のソフトウェアと比較して、国内で開発されているソフトウェアに課題がある。

定量的に評価可能なソースコードレベルで、各種ソフトウェアの保守性を比較してみた。ソフトウェア工学の世界では MacCabe のサイクロマティック数[3]がソースコードの保守性をあらわすメトリクスとして良く使われる。条件分岐もループも無いプログラムのサイクロマティック数は１で、プログラムが複雑になるとこの数は増加する。米国国家標準局(NIST)の基準[4]では、サイクロマティック数１０以下であることが求められているが、一般には、１０以下であれば簡潔なコードだといえ、１５以下であれば保守性の面で大きな問題が無いとみなされている[5]。サイクロマティック数だけ良ければ良いプログラムとは言い切れない。しかし、経験的にサイクロマティック数が異常に大きいプログラムは、どのようなメトリクスで計測しても悪いプログラムであることが多い。

表１に各種ソフトウェアのソースコードのサイクロマティック数の測定結果を示す。#1〜#4 は、いずれも数 10 万行〜数百万行のソースを数 100 から数千のソースファイルに持つ大規模ソフトウェアである。#5 は本演習の過程で同僚の非常勤講師、ソフトウェア開発の専門家である高橋英男氏が作成した約 2000 行のソフトウェアである。

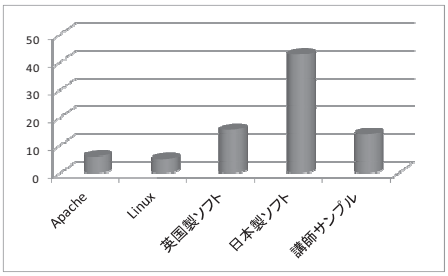
表１　複雑度を計測したソフトウェア

#	説明	記述言語
1	Web サーバのデファクトスタンダードである Apache	主に C 言語
2	OS の Linux®カーネル	主に C 言語
3	英国企業が開発した流体力学解析用ソフト(オープンソース)	C++言語
4	日本製の某シミュレーションソフトウェアのパッケージ	C++言語 Java 言語
5	ソフト工学専門家が書いた#4 と同種ソフトウェア	C++

Linux は，Linus Torvalds 氏の日本およびその他の国における登録商標または商標です。
Apache は，Apache Software Foundation が提供するオープンソースソフトウェア（フリーソフトウェア)です。

[2] 例えば筆者の勤務先の場合、どの学科の学生でもプログラミング経験を書く欄がある
[3] Wikipedia での説明は下記 URL 参照
http://ja.wikipedia.org/wiki/%E5%BE%AA%E7%92%B0%E7%9A%84%E8%A4%87%E9%9B%91%E5%BA%A6
[4] http://hissa.nist.gov/HHRFdata/Artifacts/ITLdoc/235/title.htm
[5] http://en.wikipedia.org/wiki/Cyclomatic_complexity#Limiting_complexity_during_development

これらの大規模ソフトウェアの一つ一つの関数（またはメソッド）のサイクロマティック数の平均を第２図にグラフ化した。このグラフから明らかなように、日本製のシミュレーションパッケージの複雑度が群を抜いて複雑度が高い。また、科学技術計算用のソフトウェアは、世界中に広まっているようなソフトウェアでも OS や Web サーバといった汎用ソフトウェアに比べて高いように見える。この理由として、科学技術計算向けだからなのか、それとも、そのソフトウェアの開発者のスキルの問題なのかはこのグラフからだけでは明らかではない。表１#5 のソフトウェアは、#4 と同じ分野のソフトウェアを模範解答として開発した。そのソフトウェアのサイクロマティック数は、第２図のとおり、ほぼ英国製のシミュレーションソフトウェアパッケージに等しくなった。



第２図：サイクロマティック数の計測結果

これらの結果により、シミュレーションソフトウェアの場合、企業の業務に使われるような汎用的なソフトウェアパッケージに比べて、必然的にループ処理などが多くなり複雑度は高くなる傾向がある。しかし、現状の日本製のシミュレーションパッケージの中には、科学技術計算ソフトの本来の性質による複雑度の差を上回るような、かなり保守困難なものがあるということがいえる。

これらの課題の解析により、シミュレーションソフトウェアを題材に、機械系の大学院生にソフトウェア工学を教えることは、学生の中長期的なキャリアに必要なスキルを身に付けさせるとともに、特に日本の科学技術計算用ソフトウェアの開発力の強化に寄与できるという仮説を建てた。

３.「実践的シミュレーションソフトウェア開発演習」の概要

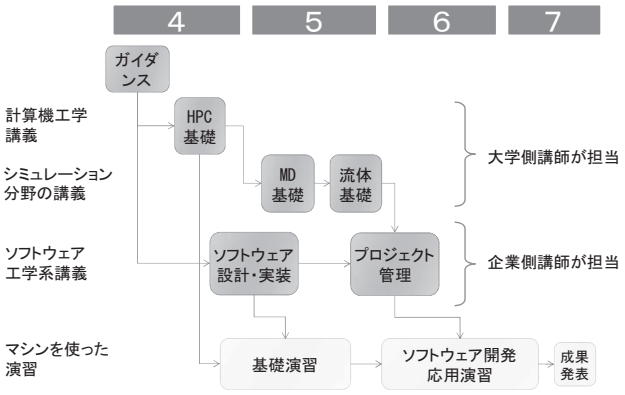
本章では、2009 年度から開始した「実践的シミュレーションソフトウェア開発演習」の概要として、スケジュール、演習環境を紹介する。

概略スケジュール

本科目の概略スケジュールを第３図に示す。

講義の日数は、年度によって多少増減はあるが、一日２コマ（3 時間）で１３日～１５日である。前半は、ガイダンス、座学の講義、基礎演習を７日。後半の５日～７日が情報基盤センターのマシン環境を活用した応用演習を行い、最後に成果発表を行う。

前半の座学の講義は、大きく三分



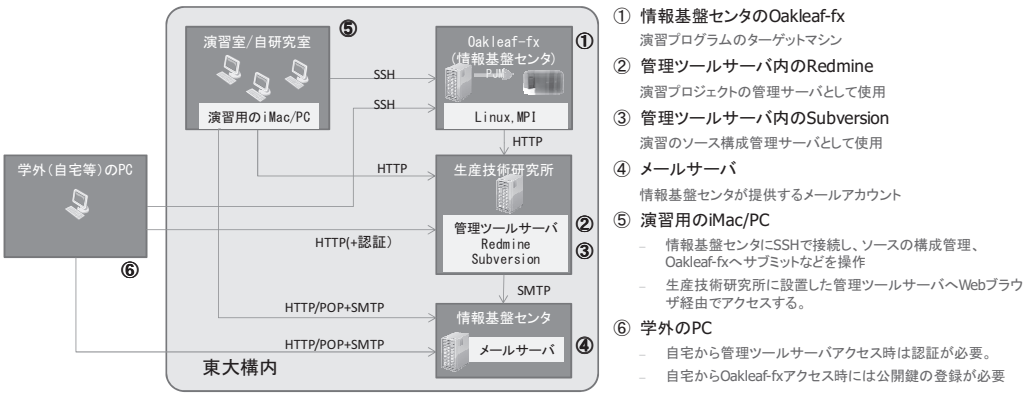
第３図：実践的シミュレーションソフトウェア開発演習の概略スケジュール

野、ハイパフォーマンス計算機の話、シミュレーションをする対象分野である流体力学と分子動力学の話、それに、ソフトウェア工学の話に分けられる。講義の分担は、計算機工学、シミュレーション分野の講義は大学側の講師が担当し、ソフトウェア工学系の講義は、企業側からの講師が担当するように決めた。前半部分でも、基礎演習として簡単なプログラムを各個人で書かせプログラミングレベルの指導を行っている。

後半は、講義で得た知識を元に、学生がチームを組んでスーパーコンピュータで並行動作するシミュレーションソフトウェアを開発する。一チーム２～３名で、チームごとに流体力学、分子動力学のどちらかのシミュレーションソフトウェアを開発するかを選択する。開発言語はC 言語または C++言語で、規模は、2000～3000 行である。学生が主体になって、そのソフトウェア開発プロジェクトの計画、設計、プログラミング、テストを行い、大学側、企業側の講師が共同で受講生を指導している。最終日に、成果発表会としてチームとしての成果物のプレゼンテーションと個人のプレゼンテーションを行っている。

演習環境

ソフトウェア開発演習という科目の性質上､演習の時間外にもマシンを使用する場合が多い。また、複数の受講者が共同してソフトウェア開発し、指導する講師も大学、企業に分散しているという本科目の特徴を踏まえ、演習時間外でも受講生や講師がコラボレーション可能な演習環境を構築した。本科目で使用している環境を第４図に示す。



第４図：実践的シミュレーションソフトウェア開発演習の演習環境

この中で、特徴的な構成は、生産技術研究所に設置した管理ツールサーバと、その中で運用しているプロジェクト管理ツールの Redmine(図中②)および、成果物の構成管理ツールであるSubversion(図中③)である。これらのツールは、講義中でも講義外であっても、情報基盤センターの iMac だけからでなく、研究室や自宅からもアクセスすることが可能である。例えば、Redmine を使用してプログラム開発に関する作業を同じチームの他の受講生に依頼したり、見つけたプログラムのバグ情報をチーム内で共有したりする。また、Subversion を使用して自分のコーディング､テストをしたソースコードをプロジェクトメンバで共有できる｡こういった、複数メンバによるソフトウェア開発演習を円滑に実施できる環境を整えた。

４．教育の実施

「実践的シミュレーションソフトウェア開発演習」は 2009 年から 2012 年まで 4 年間実施済みである。受講対象者は、2009 年冬学期は、M2、ポスドク、2010 年以降は M1 を対象に夏学期に実施している。2009 年は、試験的な年度だったため、本章では、M1 の学生を対象にし始めた 2010 年から 3 年間のデータを使い教育の実施状況を報告する。

・受講者数、プロジェクト数。目標達成プロジェクト数

表 2　受講者、プロジェクト数の推移

	受講者数	プロジェクト数	目標達成プロジェクト
2010	10	4	0
2011	8	3	3
2012	5	2	1

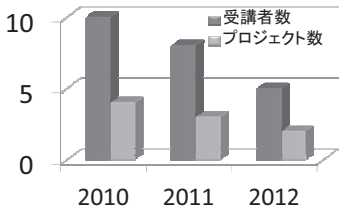


図５　受講者、プロジェクト数の推移

続いて、毎年、演習の最終日に取っているアンケートの主要項目の推移を以下に示す。

・演習の成果物に対する評価（高５ ―――― 低１）

表 3　演習成果物に対する評価の推移

	5	4	3	2	1
2010	0	1	1	7	1
2011	4	1	2	1	0
2012	0	1	3	0	1

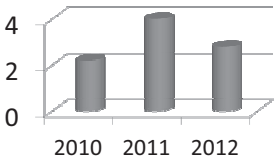


図６　演習成果物に対する評価の推移

・来年の M1 学生に受講を勧めたいか？

表 4　科目に対する満足度の推移

	ぜひ勧めたい	勧めたい	止めはしない	やめさせる
2010	2	2	6	0
2011	1	1	5	1
2012	1	2	2	0

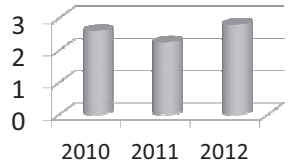


図７　科目に対する満足度の推移

・本演習で習得した知識、スキルは今後のプログラム開発で役に立つか

表 5　スキル獲得の観点での満足度の推移

	大変役に立つ	役に立つ	役に立つ場合もある	ほとんど役に立たない
2010	7	2	1	0
2011	4	4	0	0
2012	5	0	0	0

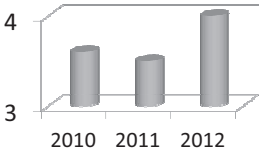


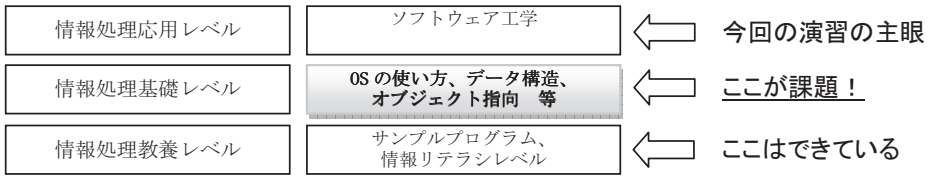
図８　スキル獲得の観点での満足度の推移

各年度での結果に大きな差がある理由は、受講生側の問題ではなく、講師側の各年度の試行

錯誤の結果である。以下、各年度でのソフトウェア工学関連で発生した課題と対策を述べる。

2010 年の演習

2010 年の演習は、受講する学生がソフトウェア工学の前提知識である情報処理の基礎レベルの知識はすでに理解していることを前提に講義を進めた。機械系の M1 の学生は、B1、B2 で学習する教養レベルの情報処理知識（数 10 行レベルのプログラミング経験、OS とは何か、Web とは何かレベル）は身につけている。しかし、ソフトウェア工学の前提になるような情報処理の教養レベルより高い知識、具体的には、OS の使い方、データ構造、オブジェクト指向設計や、それらの知識を使った高水準言語での実装といった情報処理分野の基礎的なスキルがほとんど身についていない(第９図)。



第９図：　現状の機械系 M1 学生の知識の課題

この結果、C 言語でいうと、「構造体の概念が分からない」「C 言語で可変長の多次元配列を作ることができない」といったレベルで、受講生がつまずいてしまった。また、そのレベルのプログラム開発経験が無いので、例えばソフトウェア開発の開発項目を列挙せよ、と言っても、全く思いつかない状態だった。結局、この年の 4 チームはスーパーコンピュータでのシミュレーションというゴールまで達することができずに終わってしまった。

2011 年の演習

前年の反省を踏まえ、この年は情報処理系の学科であれば学部で修得するはずの基礎レベルの講義を増やした。大震災の影響で講義日数が減少したこともあり、結果として前年度よりも多いソフトウェア工学の知識を短い講義の中に詰め込むことになった。この年のソフトウェア工学系で教えた知識項目を表６に示す。

表６　2011 年度で教えたソフトウェア工学系の知識項目

分類	知識項目
設計技法、記法	科学技術計算におけるデータモデル、
	アルゴリズム設計、性能確保方法の基礎
	設計の各種記法（UML のクラス図、PAD、シーケンス図等）
	ソフトウェアレビュー、テスト技法の基礎
プログラミング技術	OS の資源を意識したプログラミング
	コンパイラを意識したプログラミング
	各種ライブラリを活用したプログラミング
	C/C++の実装規約の考え方
小規模開発向けの プロジェクト管理	プロジェクト管理の基礎
	スクラムによる反復的な開発方法
	チケットを活用した軽量な品質管理、スケジュール管理
	プロジェクト管理ツール(Redmine)、構成管理ツール(SubVersion)、Makefile の活用方法

おそらく、情報処理系の学科の通常の科目であればこの科目の２～３倍の時間を費やして教えるような知識を、短時間に詰め込んだうえに、これら以外にも、HPC やシミュレーション分野、シミュレーション分野に特有なプログラム的な知識も必要で、講義、演習ともに密度の濃い内容になった。結果として、参加した 3 チーム全部がスーパーコンピュータでの演習ゴールに到達という成果が得られたが、科目全体に対する満足度は低かった。アンケートでは、「講義のレベルを下げて欲しい」「他の授業のことも考えてくれ」というコメントを多くいただいた。

2012 年の演習

2012 年の大きなトピックは、情報基盤センターに導入された新しいスーパーコンピュータである Oakleaf-FX を利用させていただいたことである。世界 18 位の能力を持つマシンの利用は、受講者のモチベーションアップにも役に立った。この年は、前年度のスパルタ教育の影響もあってか受講生が減少してしまったが、過去 2 年と比べて、プログラミングの素養の高い受講者が集まった。また、前年度よりも講義日を増やし、内容もこれまでの演習で受講生がつまずく部分を中心に拡充、演習に不要な部分はカットした。この結果、2 チーム中、1 チームがゴール到達。ゴールに達せなかったチームも含め、その成果物（ソースコード、ドキュメント）は過去の年よりもレベルが高かった。また、科目全体に対する受講生の満足度も高かった。従って、応用演習で行うレベルの開発プロジェクトの遂行能力を受講生に与えることは成功したといえる。

ただ、本科目のもともとの目的である「複数の開発者が数ヶ月から数年をかけて初期開発し１０年以上保守できるようなソフトウェア」が開発できるようになったかと問われれば疑問である。演習ゴールに到達するための知識という観点だけでなく、本来、受講生に修得してもらいたかった知識の一部（実は、筆者自身が担当している管理系の項目）も、カットしてしまったのは反省事項である。

５　今後の課題

４年間の試行錯誤の結果、現状の受講者のスキルのマッチした講義、演習を行えるようになってきたと評価している。しかし、2 章で述べた今後の産業界でのシミュレーション位置づけ、その中でのソフトウェア工学の重要性を考えた場合、多くの課題が残っている。これらの課題の中には、特定の科目の中で検討、対策が可能なものもあるが、もう少し広い立場で検討が必要なものも多い。本章では、その中から、一つずつ取り上げて問題提起をしてみたい。

5.1　数値計算向けのソフトウェア工学

第一の課題として取り上げたいのは、シミュレーションソフトウェアまたはもっと広い範囲で数値計算向けのソフトウェア工学である。

ソフトウェア工学という学問自体は、ソフトウェアの種類/分野を限定したものではない。IEEE が出しているソフトウェア工学の知識をまとめた SWEBOK[1]を精査してみると、分野に共通な部分と、分野に依存する部分はある。もともと、コンピュータというシステムは、数値計算用に発明されたものであり、ソフトウェアという観点でも、数値計算系のソフトウェアのほうが他の分野のソフトウェアよりも歴史がある。しかし、これまでのソフトウェア工学の多くの知識は、企業の大規模業務プログラムの開発を対象に発展してきた。また最近では組込み系

のソフトウェアの開発量が増え、組込み系ソフトウェア開発向けのソフトウェア工学の知識も蓄積されつつある。歴史的にみれば、数値計算系ソフトウェアのほうがエンタプライズ系や組込み系のソフトウェアに比べて兄貴格のはずであるが、数値系ソフトウェア向けのソフトウェア工学というものは、寡聞にして耳にしたことが無い。

もちろん、従来のソフトウェア工学的な知識がそのまま数値計算やシミュレーション向けのソフトウェアに適用可能なのであれば、全く問題は無い。しかし、この４年間の経験を踏まえて、ソフトウェア工学の多くの分野で、数値計算系のソフトウェアを開発するための工学的な知識というのは厳然として有ると筆者は考えている。本節では、プロジェクト管理レベル、ソフトウェアの設計レベル、コーディングレベルでの数値計算で特有だとかんがえるソフトウェア工学の例を一つだけ挙げてみる。

プロジェクト管理レベル：

エンタプライズ系の業務ソフトウェアの多くは、そのソフトウェアを使う人は情報処理を知らない顧客組織内の一般従業員であり、一方そのソフトウェアを開発する人は情報処理専門家ではあるが、実際開発するソフトウェアが使われる業務を知らないという場合が多い。その帰結として、そのようなソフトウェアを開発する場合にどのようなことが実現できなければならないのかといった要求項目（requirements）を正確に定義し、開発の過程でそれらの要求事項が保たれていることをソフトウェア開発側で保証する必要がある。このような背景で、SWEBOKでは、ソフトウェア工学の１０の大分類の一つとして「ソフトウェア要求」が含まれている。もちろん、シミュレーションソフトウェアに対する要求(必要条件)は存在する。実現したい自然現象がシミュレート可能という機能の要求や、特定のマシンで、数分で計算終了するという性能の要求はほとんど全てのシミュレーションソフトウェアに当てはまるであろう。しかし、エンタプライズ系のソフトウェア開発で行うような「要求の分類と概念モデル作成」「要求のトレーサビリティの管理」といった項目を厳格に実施する意味があるかどうかは筆者は疑問に思う。

ソフトウェア設計レベル：

一般に大規模ソフトウェアを開発するときには、ソースコードよりも抽象的なレベルで、ソフトウェアの設計を表記するモデルが使われる。現在、ソフトウェア業界で主流のモデルは、UML（Unified Modeling Language）[6]という記法（実際には、複数種類の図）である。このUMLと、ソフトウェアの設計ノウハウをパターン化したデザインパターンによって、ソフトウェアの設計レベルでプログラミング言語とは独立に、ソフトウェアの設計を(完璧にではないが)評価し、改善できるようになっている。

一方、シミュレーションソフトウェアの場合、究極的なモデルは、数式であろう。数式がそのまま､プログラムになるようなアプローチも古くからあり､小規模のソフトウェアであれば、そもそもプログラミングが不要なものもあるだろう。しかし、シミュレーションが必要な現実の多くの課題は、数式をモデルにするアプローチだけでは不足で、コンピュータの能力を最大限に発揮するための大きなレベルの設計が必要になる。

6

http://ja.wikipedia.org/wiki/%E7%B5%B1%E4%B8%80%E3%83%A2%E3%83%87%E3%83%AA%E3%83%B3%E3%82%B0%E8%A8%80%E8%AA%9E

そのとき、ソフトウェア工学の UML やデザインパターンが有効なもの、有効な場面はある。しかし、隔靴掻痒の思いをする場面もある。例えば、多次元の行列の処理や領域をセグメントした場合の境界処理、配列の各要素単位の処理のように、シミュレーションソフトウェアでは良く登場し、また、いろいろノウハウがありそうな場面の表現や、ノウハウのパターン化を考えてみた場合、現在の UML が提供している図だけでは不足しているのである。

本件に関しては、ヒントはある。情報基盤センターの中島先生の講義資料[7]を読むと、数値計算系でよく現れるような課題に対するプログラム処理が、専門家にもプログラム初心者の学生でも良く分かるように図示され、さらに対応するプログラムまで書かれている。大規模シミュレーションプログラムの開発時にも、同様な図があれば、その構造や挙動が分かりやすくなるはずである。ただ、中島先生の図は、一つ一つのプログラムごとにソフトウェア開発者が書く図としては少し負荷が高い。すなわち、現在の「中島図」を UML レベルの抽象的かつシンプルな図にすることにより、シミュレーションソフトウェアの設計時の設計記法として実用的になる可能性が十分にある。さらに、そのような記法が普及すれば、それを使ったシミュレーションソフトウェア向けのより良い設計のパターンが容易に記述でき、設計ノウハウの蓄積も可能になることが期待できると考える。

コーディングレベル：

従来のソフトウェア工学でいうところの多くのコーディングルールは、シミュレーションソフトウェアでも有効である。コメントは適時必要であるし、数百行もある関数/メソッドは厳禁である。しかし、コーディングのレベルでも違いは存在する。

エンタプライズ系でも、大規模プログラムで良く実行される部分と、ほとんど実行されない部分に分かれる。しかし、シミュレーションソフトウェアみたいに、ダイナミックな全実行ステップの 99.9%はこの数 10 ステップというほどの明確な違いは無い。また、性能クリティカル部分のコーディングについては、通常のコーディングルールとは別のルールが必要なことは、数値計算ソフトウェアを書いたことのある方であれば全て御存知であろう。

一方、従来のコーディングルールでは厳禁とされているが、シミュレーションソフトウェアであれば良いのではないかという例もある。一般にソフトウェア工学では、複数の文を一つの行に記述することは禁止されている場合が多い。ただ、数値計算で各座標を表す変数に数値を代入するような場合、第 10 図左側のように、文ごとに分けて書くのが良いかは、個人的には疑問である。ひと塊の処理であれば、第 10 図右側のように一文に書いて良いと考える。

x=0;
y=0;
z=0;
	x=0; y=0; z=0;

第 10 図　コーディングレベルの課題

5.2　学部における情報処理教育の充実

4 章で述べた、機械系の修士学生の情報処理の基礎レベルのスキル不足は、一つの科目だけでは解決できる問題ではない。もし、多くの機械系の研究分野で情報処理的なスキルが必要、又は、社会が機械系の大学卒に情報処理的なスキルを求めている場合、学部のレベルで OS、デ

[7] http://nkl.cc.u-tokyo.ac.jp/09w/CW02/2009-10-14-CW02.pdf

ータ構造、それらの知識を使ったプログラミングといった情報処理分野の基礎的なスキルを学生に身に付けさせるべきだと筆者は考える。

情報処理学会では、情報系の学科向けに、「情報専門学科におけるカリキュラム標準 J07」[8]を示している。また、このカリキュラムをベースに、情報専門の学科ではないが情報関連の知識を必要としている学科に対してカスタマイズして適用させる「J07 の展開 － 情報専門学科以外への展開」[9]という資料が公開されている。この資料では、組込みソフト系の学科を想定していると思われる技術開発系副専攻と、経営工学系の学科を想定している経営管理系副専攻の二分野についてそれぞれ、大学で教えるべき情報処理系の知識項目を８エリア（160 時間分の講義）に纏めている。筆者と同じ課題認識を持たれている方は是非参考文献[2]に挙げたカリキュラムを参照していただきたい。ただ、そこで示された知識項目の中には、全ての機械系の学生が知っておくべきものではないものも含まれるし、必要なものが抜けている可能性もある。

今回報告した、シミュレーションソフトウェア開発演習用に必要な基礎知識という、極めて身勝手な選択理由であるが、表７に示したような知識項目に対応した講義が学部に有ると良いのに、という希望を筆者は持っている。

表７　機械科の学部レベルで教えていただきたい
情報処理系のカリキュラム(記号は J07 の分類)

必須	選択
CPR. プログラミングの基礎	CDS. ディジタル信号処理
CAL. アルゴリズム	CHL. ヒューマンインタフェース
CDL. ディジタル論理	CAD. 人工知能とデータベース
CCA. コンピュータのアーキテクチャと構成	CNW. コンピュータネットワーク
COS. オペレーティングシステム	CN. 計算科学と数値計算

６．おわりに

本稿で述べた、「実践的シミュレーションソフトウェア開発演習」は、東京大学生産技術研究所革新的シミュレーション研究センターの加藤千幸センター長、佐藤文俊教授をはじめとした多くの先生方と、企業側も同僚である高橋英男非常勤講師で成り立っている科目である。今後も、継続、発展させることにより、今後のシミュレーションソフトウェア分野を牽引するような人材を育て、ひいては、産業の活性化も目指していく所存である。

参　考　文　献

[1] IEEE：松本吉弘訳『ソフトウェアエンジニアリング基礎知識体系―SWEBOK』, オーム社, 2003
[2] 情報処理学会情報処理教育委員会『情報専門学科におけるカリキュラム標準 J07』, http://www.ipsj.or.jp/12kyoiku/J07/J0720090407.html（2012 年 10 月 25 日参照）
[3] 中島研吾『有限要素法』の講義資料, http://nkl.cc.u-tokyo.ac.jp/12s/

[8] http://www.ipsj.or.jp/12kyoiku/J07/J0720090407.html
[9] http://www.ipa.go.jp/jinzai/sangaku/pdf/07/siryo6-1.pdf