⑥ ソフトウェア開発教育における共通問題

権藤克彦（東京工業大学）

ソフトウェア開発教育の共通問題の必要性

ソフトウェア開発教育の共通問題は，ソフトウェア開発教育の技術や方法論の有効性を比較検討するためのベンチマークである．以下で述べる通り，これはソフトウェア工学の共通問題と多くの点で類似しており，その類似性を比較検討することは，ソフトウェア工学の共通問題をより深く理解するために有用と考える．筆者がかつて行った教育関係の研究（教育用 OS[10]，教育用コンパイラ[11]）などの経験に基づいて，ソフトウェア開発教育の共通問題の解説を試みる．

ソフトウェア工学と同様に，ソフトウェア開発の教育効果の測定は難しい．教育の場合，筆記試験等である程度，客観的に教育効果を測定できる場合もあるが，残念ながらソフトウェア開発教育ではそれは難しい．ソフトウェア開発能力を筆記試験だけで測定することは難しいからである．たとえば，オブジェクト指向プログラミングを教えた後で，カプセル化，情報隠蔽，継承などの**機能**を尋ねる筆記試験を作ることは容易だが，これらの機能を上手に組み合わせて，保守性などのさまざまな観点で高品質なソフトウェアを開発できる**能力**を測定することは難しい．

またそれ以前の問題として，ソフトウェア開発教育の共通問題がないため，仮に教育効果を測定できたとしても，別々の教育効果を測定しているという問題がある．このため，ソフトウェア工学の場合と同様に，ソフトウェア開発教育においても共通問題は重要となる．

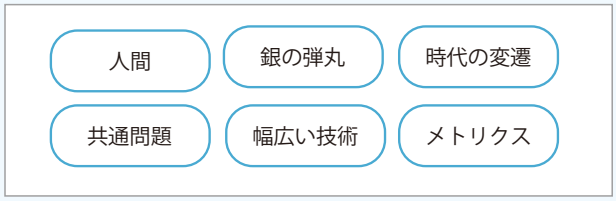

図 -1　ソフトウェア開発教育とソフトウェア工学は似ている

ソフトウェア工学の共通問題との類似性

❖ ソフトウェア開発教育とソフトウェア工学の類似性

ソフトウェア開発教育とソフトウェア工学は次の点で類似している（**図 -1**）．

- 評価には人間が必要となる（人的要因がある）
- 良いメトリクスがない
- 普遍的な技術や決定的な技術（銀の弾丸）が存在しない
- 技術の対象が多岐にわたり，時代とともに変遷する
- （実際に使える）共通問題や例題集が少ない

最後の「共通問題や例題集が少ない」について，少し詳細に述べる．たとえば，文献 1) はプログラミング入門教育に関するサーベイ論文であり，カリキュラム，教授法（pedagogy），プログラミング言語の選択，教育用ツール，という 4 つのカテゴリに分類して論文を紹介している．しかし，「共通問題や例題集を与える」という観点の論文は紹介されていない．

教育手法を比較する場合，何を教育するかを固定する共通問題がなければ，複数の教育手法を適切に比較することはできない．たとえば，プログラミング言語の選択の場合，プログラミング課題を共通問題として固定する必要がある．さもなければ，各プログラミング言語で解きやすい例題を多く集めてしまう結果となりやすい．すべてを網羅的に調べたわけではな

いが，筆者が調べた限りでは，プログラミング入門以外のソフトウェア開発教育でも，実践的な共通問題は存在しなかった．

また，例題集も同様である．たとえば，モジュール化やリファクタリングという，比較的教えやすく見えるソフトウェア開発技術に対する実践的な例題も，筆者が調べた限り，存在しなかった．書籍などの例は人工的かつ小さすぎる例題であり，例題としては適切ではない．小さすぎる例題は，たとえばボディが数行程度の手続きを複数に分割するといったものであり，これではモジュール化やリファクタリングの恩恵やトレードオフを教育したり，学生に考えさせることが難しい．

❖「小は大を兼ねる（べきである）」

上で述べた，「小さすぎるとダメ」という考え方はソフトウェア工学における「Programming-in-the-small と Programming-in-the-large では必要となる技術や方法論がまったく異なる」という議論と類似している．しかし，興味深いのはプログラミング開発教育の研究では，「小は大を兼ねる（べきである）」という考え方もかなり多く見られることである．

たとえば，情報工学科でオペレーティングシステム（OS）やコンパイラを実践的に教えるために，Linux や GCC のソースコードをそのまま使うことはその巨大さゆえにかなりの困難が伴う．そのため，本質を保存したままサイズを小さくした（と主張する），教育用 OS や教育用コンパイラが数多く提案され実装されてきた．たとえば，教育用 OS は数千行から数万行規模であり Linux などの実用的な OS と比較すると，（学部生が十分に読みこなせるかどうかは別として）そのサイズはかなり小さい．

実用的な OS では，最適化による高速な実行，エラー処理による頑健性，セキュリティ，移植性などの非機能的要件や多機能化（例：多くのデバイスサポート）のために，ソースコードは巨大になりがちである．しかし，教育用 OS ではこれらを思い切って簡素化し，OS 教育の本質（たとえば，プロセススイッチやページングの仕組み）に焦点をあてることができると主張している[10]．

❖ 評価の温度差

ソフトウェア開発教育とソフトウェア工学では，どちらも評価が難しいという点は同じであるが，やや温度差があるように筆者は感じている．たとえば，プログラミング開発教育の代表的な国際会議 SIGCSE[4] では，評価方法に関して，

- 被験者数は数十人規模から
- 対照実験は必ずしも行わなくてもよい
- 評価は自己評価アンケートのみのことも多い

という傾向がある．

対照実験については，理想的には被験者を実験群と比較群に分けて，提案手法の優位性を示す実験をすることが望ましいが，次の 2 点で難しい．

- 1 つ目は倫理的な問題である．通常，学生の成績は就職や進学（留学）などに使用されるため，実際の教育現場で実験を行う場合，同じ授業料を払っている学生に（実験群と比較群に分けたために生じる）異なる内容の教育をすることは教員にとって大きな心理的障壁となる．
- 2 つ目は客観的な尺度がない問題である．知識を問う問題であれば，従来の筆記試験を課して，その得点の統計的有意差を調べれば十分である．しかし，たとえば，OS の**実践的な**知識を教えるために教育用 OS を用いる場合，通常の教科書[5] のみの座学と比較して，教育用 OS の使用が有効であることは，おそらく筆記試験では測定できないだろう．そもそも，この種の実践的な知識を効果的に測定する方法は知られていない．このため，実験群と比較群を比較する良い方法がなく，評価は自己評価アンケートだけになりやすい．

ソフトウェア開発教育の共通問題への要件

ソフトウェア開発教育も，ソフトウェア工学と同様の問題を抱えていて，評価が難しく，共通問題の作成も難しい．本章ではソフトウェア開発環境の共通問題が満たすべき主な性質を述べる．これはやはりソフトウェア工学の共通問題と多くの共通部分があり，共通問題作成の困難さの説明ともなる．

- 学生への動機付け
- コンパクトさ・明解性
- 中立性・包括性・リアリティ・分解能
- 利用性・再利用性・普遍性
- 追試可能性・測定可能性
- 人的要因・社会学的側面への配慮

共通問題は学生の興味をひき（学生への動機付け），限られた講義時間に収まることが望ましい（コンパクトさ・明解性）．しかしながら，抽象化や単純化をしすぎると，特定の方法論に特化されすぎたり（中立性），問題内容の網羅性が低かったり（包括性），ソフトウェア開発の現実的な問題に欠けたり（リアリティ），点差がつかなくなったり（分解能）してしまう．

共通問題はさまざまな意味で使えなければ意味がないし（利用性），共通問題の開発コスト削減のために再利用できるべきである（再利用性）．また，なるべく時代や文化の違いを超えた本質な内容を含むべきである（普遍性）．さらにベンチマーク（ものさし）として，異なる教育手法をある程度は比較できることが望ましい（追試可能性，測定可能性）．

普遍性は言い換えると「当たり前のことが重要」とも言える．ソフトウェア工学やソフトウェア開発教育の知見の多くは，当たり前のことであるがその達成は自明ではないことが多い．たとえば「モジュール化すべきである」という考え方はソフトウェア工学の黎明期から常識であるが，多くのドメインにおいて，高品質なモジュール化の達成方法は現在でも自明ではない．アスペクト指向プログラミングやアプリケーションフレームワークの研究や実装も進んでいるが，必ずしも十分ではない[8]．その意味で，共通問題は**枯れた当たり前の問題**を扱うべきである．逆に，通常のソフトウェアとは異なり，後方互換性を保つ必要はないので，この点では共通問題の改善は楽である．

最後の人的要因・社会学的側面への配慮は，特にPBL（課題解決型学習，problem-based learning，または project-based learning）などの実践的な開発演習で必要となる．チームワーク，コミュニケーション，リスク管理，スケジュール管理，要件開発などは技術的な側面だけでなく，人的要因や社会学的側面が大いにかかわるからである．もちろん，スーパーハッカーの存在などを仮定するような属人性は排除すべきであるが，だからといって人的要因・社会学的側面の全排除は避けるべきである．

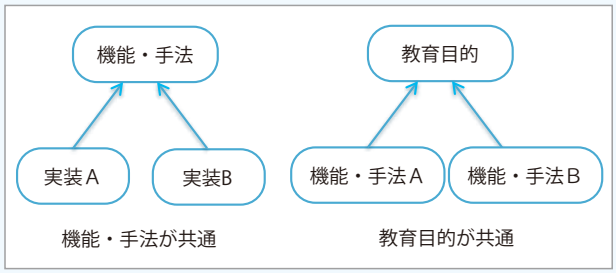

図 -2　2 種類の技術的な共通問題

以下では 3 種類の共通問題について議論するが，もちろん本章で述べた要件をすべて満たせるものは提供できない．ソフトウェア開発教育の共通問題作成は本質的に難しい．

技術的なソフトウェア開発教育の共通問題

本章では比較的扱いやすい技術的な共通問題をまず述べる．ここで技術的な共通問題とは，人的要因をあまり含まない共通問題であり，教育支援ツールや教育方法論などを比較したり評価するものとする．たとえば，教育用 OS や教育用コンパイラはこの技術的な共通問題で比較評価する典型的な教育支援ツールである．技術的な共通問題はさらに次の 2 つに分類できる（**図 -2**）．

- 機能・手法が共通
- 教育目的が共通

教育用 OS を例にすると，「教育用 OS は実機で動作すべきである」という考え方と，「教育用 OS は仮想機械やユーザ空間での動作でも構わない」という考え方に大別できる．前者の典型は Minix や GeekOS[6] であり，後者の典型は Nachos[7] である．ここで，たとえば「教育用 OS は実機で動作すべきである」という考え方を固定して，それぞれの教育用 OS の実装を比較するために，共通問題を作成するのが「機能・手法が共通」である．一方，「教育用 OS を用いて実践的に教育する」という教育目的を固定して，「実機で

動作する教育用 OS」や「仮想機械やユーザ空間で動作する教育用 OS」など異なる機能を持つ教育用 OS を比較するために，共通問題を作成するのが「教育目的が共通」である．

以下でそれぞれ述べる．

❖ 機能・手法が共通

「機能・手法が共通」の場合，共通問題を作ることは（ほかに比べると）比較的容易である．たとえば，実機で動作する教育用 OS の仕様を次のように定めれば，この仕様を満たす異なる実装の教育用 OS を比較しやすくなる．

- POSIX API のうち，次のシステムコールをサポートすること：`open`, `fork`, …
- 実機として PC/AT 互換機 [9] で動作すること．
- 内部実装として以下を満たすこと：「プロセスはタイマー割り込みでプリエンプションすること」，「各プロセスのメモリ空間は仮想メモリで独立させること」，…

ただし，理想的には次の 2 点もきちんと定める必要があり，共通問題作成に難しい点は残っている．

- 教育対象に仮定するスキル
- 何に重点をおくか（何を本質と捉えるか）

また，実際の教育用 OS の実装は非常に数多く，それらの教育効果の比較は十分にはできていないのが現状である．これはビジュアルプログラミングで非常に多くのユーザを獲得し成功している Scratch[3] のような，定番となる教育用 OS がより多く登場すれば解決する問題かもしれない．

❖ 教育目的が共通

「機能・手法が共通」に比べて「教育目的が共通」の場合は共通問題の作成はより難しくなる．それはたとえば「教育用 OS を用いて実践的に教育する」という教育目的を具体的に書き下すことが難しいことに加えて，上でも述べた次の 2 点において

- 教育対象に仮定するスキル
- 何に重点をおくか（何を本質と捉えるか）

合意を得ることが難しいからである．たとえば教育用 OS の例では「自分の学生には実機で動作する教育用 OS を使える」「いや複雑すぎるから使えない」という点で相違しており，対象とする教育レベルの合意は難しい．無理に合意を図ろうとするとボリューム過多の共通問題になってしまう．

人的要因・社会学的なソフトウェア開発教育の共通問題

いわゆる PBL は，ソフトウェア開発教育の中でも実践的で効果的な教育方法として注目され，ソフトウェア科学やソフトウェア開発教育の分野でもさかんに研究 [2]・実践 [12] されている．

PBL とは文献 2) の定義によれば，「学生自身に自らの学習に対して責任を取る権限を与えることを模索しながら，学生を学習プロセスの焦点とする教育的アプローチ．従来のアプローチは，学生ではなく，知識の伝達者である教師が学習プロセスを駆動する点が異なる」．

ソフトウェア開発教育における PBL は典型的にチーム開発を伴うので，人的要因や社会学的側面を必然的に伴う．この章ではソフトウェア開発教育の PBL の共通問題化を議論する．

enPiT プロジェクト [12] は，最先端の情報技術を実践的に活用できる人材育成を目指した全国的な教育プロジェクトであり，各大学ごとに特色のある PBL 教育を試みている．たとえば，筆者が所属する東京工業大学では，enPiT プロジェクトの一部として，以下の開発を行う PBL 教育を行っている．

- 1 チームは 6 ～ 7 名．開発期間は約 8 カ月．
- 開発するソフトウェアは「大学講義を支援するアプリ」「大学生活を豊かにするアプリ」などであり，意図的に要件開発の余地を大きく残している．
- 開発するソフトウェアは複数台の Android 端末とクラウド上のサーバから構成されること（異なる立場のステークホルダを創出）．
- 全チームに同じ課題を与えて，成果発表会で投票を行い，最優秀チームを選ぶ（競争とマイルストーン）．
- なるべく学生の自主性に任せるが，デスマーチになら

ないように上流工程ではある程度，教師が介入する．

- 学生の負荷を慎重にコントロールして，過負荷にしない．たとえば，安易な国際化（英語による PBL）はしない．

この教育事例は共通問題となり得るだろうか？　ありきたりな答えだがイエスでもありノーでもある．ノーとなる要素には以下がある．

- 学生に対するインセンティブ
- 開発にどこまで教師が介入するか
- 評価方法

学生に対するインセンティブは PBL が成功する鍵となるが，どのようなインセンティブを与えればよいかは自明ではない．多くの学生は「就職活動に有利になる」と考えてくれるが漠然としがちであるため，身近なインセンティブ，たとえば「最新の開発機材が貸与される」「海外で発表できる」「純粋に開発が面白い」などを与えた方がよい．

開発への教師の適切な介入も自明ではない．あえてデスマーチを演出するために開発を攪乱させるというレベルから，教師が開発に逐一介入するというレベルまで，その範囲は広い．

3 番目の評価方法には 2 つの側面がある．

- まず PBL 学習において，学生をどう評価すればいいかはしばしば議論となる．チーム開発では一部の学生に開発負荷が偏ることが多いため，その偏りを加味した成績をつける必要があるからである．その一方で技術的な貢献は少ないものの，チームの雰囲気を良くする触媒としての役割を担う学生もいるため，この問題解決は自明ではない．現場の声を聞くと「ある程度は諦めて，えいやで評点をつけるしかない」「学生同士に互いに評点をつけさせる」などの声が多いようである．
- もう 1 つの問題はこの種のプロジェクト自体をどう適切に評価すべきかという問題である．実践的，かつ人的要因・社会学的側面を含むソフトウェア開発教育を適切に評価するメトリクスが存在しない以上，適切な評価は事実上不可能である．しかしながら，政治的な理由（説明責任）等から，PROG テスト[13] などで評価とせざるを得ない現状がある．

以上の問題はあるが，成功事例と失敗事例を多く集めて，共通問題を形成することはこの分野の発展のために重要である．

参考文献

1) Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M. and Paterson, J. : A Survey of Literature on the Teaching of Introductory Programming. SIGCSE Bull. 39, 4, pp.204-223 (Dec. 2007).
2) O'Grady, M. J. : Practical Problem-Based Learning in Computing Education. Trans. Comput. Educ. 12, 3, Article 10, 16 pages (July 2012).
3) Maloney, J., Resnick, M., Rusk, N., Silverman, B. and Eastmond, E. : The Scratch Programming Language and Environment. Trans. Comput. Educ. 10, 4, Article 16, 15 pages (Nov. 2010).
4) SIGCSE, The ACM Special Interest Group on Computer Science Education, http://www.sigcse.org
5) Tanenbaum, A. S. : Modern Operating Systems, 3rd ed., Pearson-Prentice Hall, ISBN-13: 978-0138134594 (2008).
6) Hovemeyer, D., Hollingsworth, J. and Bhattacharjee, B. : Running on the Bare Metal with GeekOS, 35th ACM SIGCSE, pp.315-319 (2004).
7) Christopher, W. A., Procter S. J. and Anderson, T. E. : The Nachos Instructional Operating System, Proc. USENIX Winter, pp.481-488 (1993).
8) 満田成紀，福安直樹：ウェブアプリケーションフレームワーク利用に潜む課題〜実プロジェクトデータを題材に〜，コンピュータソフトウェア, Vol.27, No.3, pp.2-12 (2010).
9) PC Open Architecture Developer's Group，OADG テクニカル・リファレンス DOS/V BIOS インターフェース技術解説編第 2 版 (1994).
10) 権藤克彦, 大場　勝 : 中レベル抽象・薄い中間層・追跡性の実践によるコンパクトな教育用オペレーティングシステム udos の設計と実装，電子情報通信学会論文誌，J90-D[5], pp.1194-1208 (2007).
11) 権藤克彦，福安直樹，荒堀喜貴：ネイティブアセンブリコードを出力する教育用コンパイラ（XCC）と，水平スライスが可能な可視化ツール（Mieru-Compiler），電子情報通信学会論文誌，Vol.J95-D, No.5, pp.1225-1241 (2012).
12) enPiT：分野・地域を越えた実践的情報教育協働ネットワーク，http://www.enpit.jp
13) PROG テスト：教育を通じたジェネリックスキルの成長を支援するプログラム，http://www.kawai-juku.ac.jp/education-research/prog-outline/

（2013 年 6 月 4 日受付）

権藤克彦　gondow@cs.titech.ac.jp

1994 年東京工業大学理工学研究科情報工学専攻博士課程修了．同年同大情報理工学研究科情報工学専攻助手．講師・准教授等を経て，2011 年より東京工業大学学術国際センター教授．博士（工学）．ソフトウェア開発環境・システムプログラミングに興味を持つ．著書「例解 UNIX プログラミング教室」「Java によるプログラミング入門」．ACM，日本ソフトウェア科学会，電子情報通信学会各会員．