XPテストファーストと教育法・学習法

Test-first method and Study

小林　健一郎
Ken-ichiro KOBAYASHI

（平成30年10月２日受理）

要旨

学習をソフトウエア開発と考え、ソフトウエア開発技法のひとつであるエクストリームプログラミングを教育法や学習法に適用する。特に、テストファーストを機能させる方法を考察する。

KeyWords：教育法、学習法、ソフトウエア開発技法、XP

１．序論

学習は、脳内における知識・知恵の構成であり、知識・知恵はソフトウエアであるので、学習や教育にはソフトウエア開発技法が適用できると考えられる。[１]では、特にエクストリームプログラミング（XPと略記）の技法[２][３]の適用を考えた。

XPは「多くのプログラマに良いと思われている開発方法を極端にして実行する」というもので、いくつかプラクティスとよばれる「実行すべきもの」で構成される。著者は、そのうち、次のものの教育法・学習法への適用を考えた。

その際、学習者を開発者に対応させる。学習者は発注者の要求に応じて「自己の能力」を開発していくのである。発注者は、教員の場合と学習者本人の場合の2種類がある。

・計画ゲーム

中目標（リリースとよぶ）とさらに小さな目標（ストーリーとよぶ）を作る。

さらに、各ストーリーについて、「それを実現するためにやるべきこと」（タスクとよぶ）を決める。

通常のXPでは、ストーリー（実現するべき機能）が発注者から与えられ、それらを発注者と開発者がまとめてリリース（実現すべき機能をまとめたもの）にする。これは「何をすべきか」の全体像が確定していない場合に極めて有効だと思う。しかし、教育・学習に適用する場合、「学ぶべきこと」ははじめに決まっていることが多い。そこで、本小論では、リリースを「ある程度まとまった中目標」、ストーリーを「それ以上分解できない（分解しない）小目標」などとし、「リリース→ストーリー」の順で考えることにした。もちろん、「学ぶべきこと」がはっきり決まっていない場合は、通常のXPの手順にすればよ

い。

なお、ストーリーは、「二次方程式が解けるようになる」など、なるべく成功・達成時のイメージが持てるようにすることが重要である。

タスクは、ここでは「各ストーリーを実現するためにやるべきこと」とした。（ソフトウエア開発ではもう少しイメージがあるが、学習法の対応ではこうするのが自然だと思う。）「教科書の○ページを読む・理解する」「章末問題を解く」など、具体性が重視され、「方程式が解けるようになる」といった成功・達成イメージはなくてもよいとする。

なお、ストーリーのタスクへの分解等はXPではリリース計画・イテレーション計画などとよばれる。本小論ではそれらも含めて「計画ゲーム」という１項目にまとめることにした。工程をあまり分解しない方がよいと考えるようになったためである。

・テスト戦略

はじめに小テストを多数作り、それらを順次クリアしていくことで、学習を進める。

・継続した結合

「新たに獲得した知識」と「これまでに獲得していた知識」を、毎学習時に統合していく。

XPの哲学的側面で重視される事は「発注者（教師など）と受注者（学習者）の責任分担をはっきりさせる」「極力無駄を省く（無駄なことにとらわれない）」「気持ちよく労働（学習）する」だと思う。これらは、本小論でも重要であるが、特に「極力無駄を省く（無駄なことにとらわれない）」を重視している。

実践としては、上記のように、計画ゲーム、テスト戦略、継続した結合を取り上げて考察する。XPの特徴的プラクティスとしてペアプログラミングがあるがここでは扱わない（[１]を参照）。

[１]では、テスト戦略には困難があることを指摘した。XP式の学習法・教育法では、タスクの完成を確認するタスクテスト、ストーリーの完成を確認するストーリーテスト、リリースの完成を確認するリリーステストが必要となる（本節末のテスト名に関する注を参照）。

リリーステストは「一定の範囲（単元など）のテスト」であり、ストーリーテストは「小テスト」のようなものとなる。これを教師が提示することは難しくない。

タスクテストは、タスク終了毎に「その達成をピンポイント的にチェックするテスト」となる。これは本来学習者自身が用意するものであるが、初心者である学習者が「自分用のテスト」を作ることは簡単でない。そこで、教師が用意してもよいと考えるが、ピンポイント的なものを多数用意するのは簡単ではない。これが上記の「テスト戦略の困難」である。そこで、複数教員の協力によるデータベースの作成などを提案した。これらについての著者の考えは変わっていない。

しかし、本小論では、タスクテストを学習者が自分で作成していく方法（XP本来の方法）を提案する。この方法は、以下で論じるように「継続した結合は学習者の内面で行う

ことであり、その手順を示すことが難しい」というもうひとつの困難を、部分的に解決すると思われる。

ところで、「学習者に（タスクテストであれ何であれ）テストを作らせる」とは、「学習者に自分だけの問題集を作らせる」ことである。これは目新しいものではない。（XPに真に新しいことはない。）ただし、XPの理念に沿った形での実現を考えていきたい。

「学習者に自分だけの問題集を作らせる」という教育法は、少なくとも著者が小学生の頃（1960年代）からあり、著者自身何度か指導された。たとえば、「問題集やテストでできなかった問題をノートに写し、続けてその解答を清書する。そのノートをあとで復習する」などというものだった。この方法がうまくいく場合もあるだろう。しかし、ここで提案している方法は、理念的にも実践的にも異なるものであると思う。

著者が受けた指導、また、見聞した指導では、「丁寧にノートを作ること」が重視されていた。問題をきれいに書き写したり、解答を清書すること自体を学習と考え、それにより学力が向上すると主張する教育者が多かったようにも思う。確かに、「重要な情報をゆっくり書き写し熟考すること」には、大きな効果があるだろう。しかし、それをずっと続けられる学習者は少ない。次第に惰性となり、「考えずに書き写す作業」になることもあるだろう。そうなると、学習効果の薄いただの苦行になる。

「関心・意欲・態度」が重視される最近の初等教育では、一定数の生徒たちが実にカラフルな美しいノートを作る。彼らは優等生であり、きれいなノートを作成し、内申点重視の「良い学校」に進学していく。しかし、著者には「ノートを作ること」に意識が行き過ぎて、勉強になっていないように見える。数学の問題を解くことに頭を使うより、数学の公式や模範解答を見栄え良く表記することに頭を使っているのではないだろうか。それは、数学の勉強ではない。

繰り返しになるが、「重要な情報をゆっくり書き写し熟考すること」自体に反対しているのではない。「丁寧なノート」が無用かどうかを議論しているのでもない。それを継続でき効果を感じられる学習者には大きな利得があるだろうと思う。しかし、ここでは、XP的な「簡単に作れるメモ集」のようなノート（プロジェクトノートとよぶ）の作成を提案する。

次節では、具体例を見ながらこれを説明する。

テストの名称に関する注：

リリーステスト・ストーリーテスト・タスクテストという名称は自然であり、おそらく多くのXP支持者にも使われていると想像するが正式な用語ではない。特に、学習法・教育法への適用に限れば、著者の造語である。本小論では、[1]でストーリーテストとよんだものを分割してタスクテストにすることにした（1つのストーリーテスト　→　複数のタスクテスト）。[1]では「1ストーリーが終了後に、そのストーリーの全タスクの達成度をまとめてテストすること」を想定していたが、本小論の方法により、各タスク終了後に個別に達成度を見ることができるようになったと考えたからである。なお、[1]で書いたタスクテストは「遂行したか自分に問うこと」などで、これもタスクテストの一部として残した。

2．XP式学習法の実際

本節では、著者の提案する学習法を、実際の適用を見ながら説明したい。発注者＝受注者＝学習者＝著者である。

本小論の主題ではないが、著者は、教育法・学習法を主張する教育者は、まず自分に適用すべきだと考える。「このようにしたところ、生徒がいきいきしてきました」などと報告をする前に、自分が経験してどうだったかを見るべきだろう。そこで扱う内容はもちろん「本人にとって学習」でなければならない。大学教員が中学の数学を勉強し直してみせて「わかりやすかった。理解が進んだ」などということに意味があるだろうか。さらに、少なくとも引退するまで（あるいは、意見を撤回するまで）はずっとその学習法をし続け、実証し続けるべきだろう。もちろん、意見を撤回したならそう宣言すべきでもある。著者はそのつもりである。

ここでは、中岡宏行著「圏論の技法」[4]を取り上げる。以下では、この本と著者の状況を説明しながら進めるが、それは「具体例」であるからで、具体例から他の学習者へ適用できる一般論を抽出することを考えたい。

「圏論の技法」のまえがきには、数学科の学部3年次で扱う環や加群の操作についての基礎的な知識のみが必要で、学部生・院生向けの圏論の入門書とある。著者は、数学科の出身ではないが、たぶんその程度の知識はあり、さらに圏論について基礎的な理解もあるが、この本が扱う三角圏や導来圏などはほとんど知らない。「今読んで丁度よい（挫折するかもしれない）」というレベルの本だろうと思う。

この本に対し、プロジェクトノートを作って学習していく。

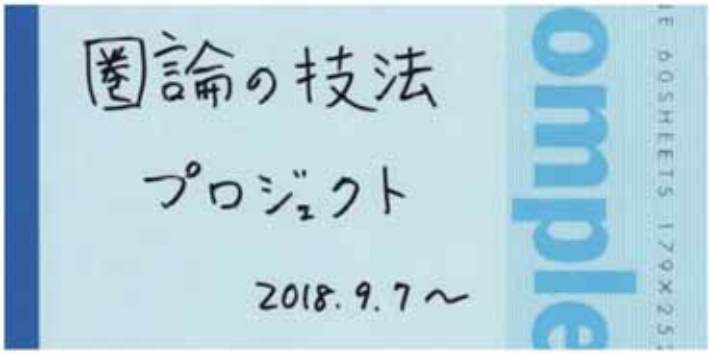


図1「圏論の技法」プロジェクトノート

まず計画ゲームで、リリース・ストーリー・タスクを作るのだが、「具体的に何を学ぶのか・学んでどうなるのか」は、学習者である著者にはほとんどわからない。これが通常の学習者の状態だろう。したがって、目次を中心に、本文をパラパラ見ながら、考えるしかない。

まえがきには、次のような図がある（当然、目次の章立ても同じである）。

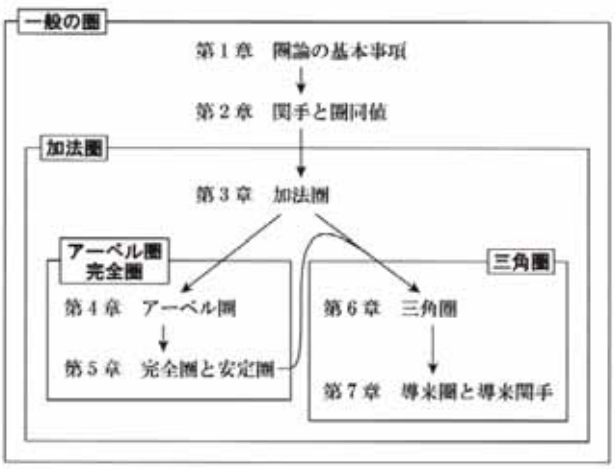


図２　圏論の技法・まえがきより

この図は、中岡先生ご自身が考えたリリースのタイトル（各章のタイトル＝リリースのタイトル）だろう。学習者としては、ほとんど何の実感もわかないが、これに従うしかない。そこで、リリースは章立てで行うことにする。ただし、可能な限り「イメージを持てるもの」にしたい。つまり、リリース１は章のタイトル通り「圏論の基本事項」でも良いが、可能ならよりイメージを持てるものにしたい。

再びまえがきを見ると、「第１章では、圏と関手についての基本事項を記してあります。対象や射といった用語を導入し、数学のさまざまな分野に現れる概念が圏や関手を用いてどのように統一的に記述できるようになるかを見ます。…」とある。これを参考に、リリース１は次のようにまとめることができるだろう。

Release 1 数学のさまざまな分野に現れる概念を、圏や関手を用いて統一的に記述できる（ようになる）。

そんなことが本当にできるのかわからないが、教科書をまとめることで、目標がはっきりしたと思う。

学習者は知らないことを学んでいるのであり、完璧なリリースをはじめから作ることは期待できない。できないことに時間をかけても無駄である。ここで作られるリリースは仮設的なものであり、あとで変更されてもかまわないし、実際、理解の深まりにしたがって積極的に変更していくことを推奨したい（本節末に詳述する）。

リリース２（第２章に相当）以降は、リリース１の完遂後に考えることにする。

次にリリース１をストーリーに分割する。目次を見ると、第１章「圏論の基本事項」の

中の節は「１.１　圏と関手、１.２　普遍性、　１.３双対概念　１.４　圏の構成　１.５　圏の定義再論」とある。ストーリーも節に従って、次のように書いてみる。

Release 1
　数学のさまざまな分野に現れる概念を、圏や関手を用いて統一的に記述できる。

Story 1
　圏と関手の定義や例を覚える。

Story 2
　普遍性を理解する。

Story 3
　双対概念を用いて議論を簡明化できる。

Story 4
　与えられた圏から新たに圏を構成できる。

Story 5
　圏の定義を理解する。

著者が書いたストーリー２には、内容がほぼない。このようなものは避け、なるべくイメージを持てるものにしたいのだった。しかし、これ以上のものが書けないのだから、しかたがない。さらに言えば、ストーリー１も、内容がありそうでそれほどない。他のストーリーも同様である。初学者が書いているのだから、この程度になるだろう。

ストーリーもはじめから完璧なものを作ることは期待されないのである。ここで作られるストーリーは仮設的なものであり、あとで変更されてもかまわないし、実際、理解の深まりにしたがって積極的に変更していくことを推奨する。

ここまでのところ、「ゆっくり考えながらやっている」と思われるかもしれないが、それは説明の都合上のことである。結果をノートに書き出すまで、ざっと本を見て、少し考えながら、ほとんど手を止めずに行っていることは述べておきたい。次にストーリーのタスクへの分解でも同じである。それは、「時間をかけても意味のない事には時間をかけない」からである。時間をかけて意味があるのなら、計画ゲームにも時間をかけることは言うまでもない。

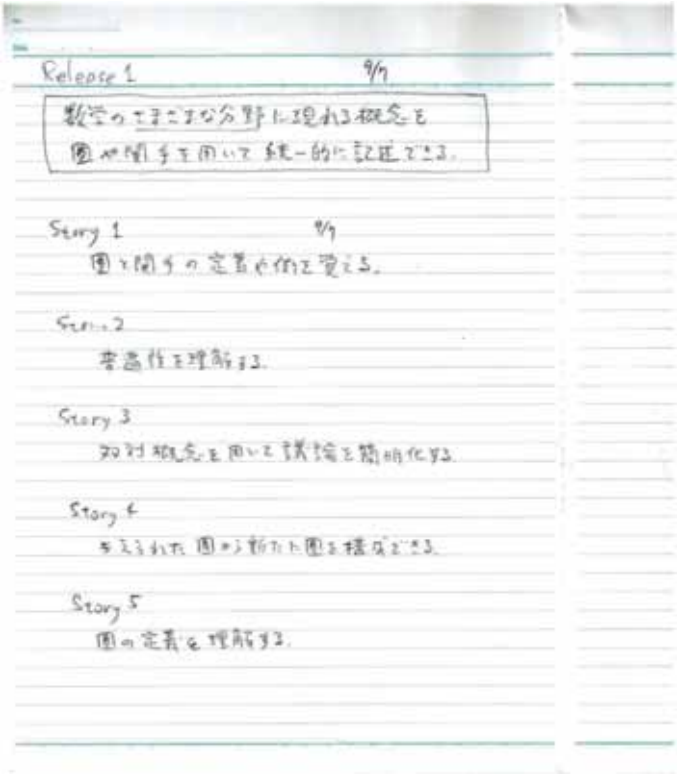


図3　リリース1

タスクは、イメージはなくとも、「何をするか」を明確なものにしなければならない。

実は、著者は、数学書（一般に難しい本）を読む際に、ほぼ必ず「ざっと見る」「重要単語に印をつける」「全体の構成を考える」を順に行う。「ざっと見る」は、著者の「ぼんやりでも全体を見ておいた方がその後読みやすい」という個人的経験から実行していて、学生諸君にも強く勧めている。しかし、人それぞれだろうから、「自分には合わない」「時間の無駄」と考える人には勧めない。次の「重要単語に印をつける」も同様である。

その内容はあとで説明するが、まず形式の話をしたい。すなわち、上述したように、どのストーリーでも著者の最初のタスク3つは同じなのである。したがって、これは「必ずやるもの」として「プロジェクトノートにはわざわざ書き出さない」という流儀でも良いと思う。著者は、略記した形で書き出すことにした。その方が「何かをしている感」があるからだ。

以上から、プロジェクトノートには次のようにタスクを書き込んだ。

Release 1
　数学のさまざまな分野に現れる概念を、圏や関手を用いて統一的に記述できるようになる。

Story 1
　圏と関手の定義を覚え、その記述ができるようになる。

以下のタスクで、「全範囲」とは「p 1 〜p14」のことである。

Task 1
　全範囲をざっと見る。

Task 2
　全範囲にある「重要そうな単語」に鉛筆で印をつける。

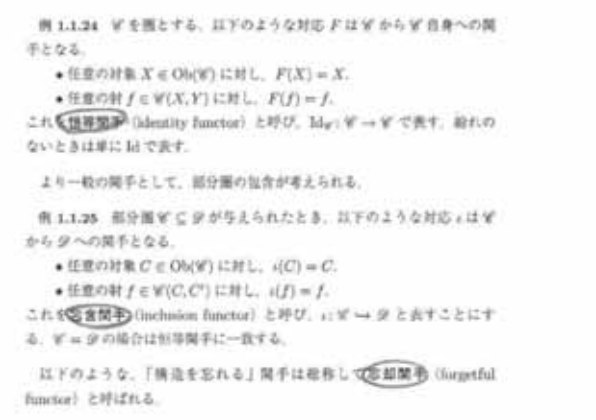
Task 3
　全範囲を見直し、構成を考える。

続く…

　タスク 1 を実行する場合、タスクテストはどうなるだろうか。これは、「心の中だけでの確認」だけで済ます。タスクは学習者が自分で決めるものでその達成も学習者自身が確認するのが原則である。本節では「何を内心で確認するか」も明記することにするが、これは本小論の読者のためであって、実際に明記するという意味ではない。

　タスク 2 は、本をもう一度ざっと見ながら、重要そうな単語に印をつける作業である。世の中に「数学がわからない」という人は多い。そういう人も、また指導者も「理解すること」に注意を集中することが多い。しかし、著者の見るところ、理解できないのは、基礎事項を覚えていないからであることが多い。たとえば、「傾き」の定義を覚えていない人が微分を理解できるとは思えない。著者は「覚えるためにはターゲットを絞る必要がある」と考える。そのための第一歩が単語の印つけである。ただし、この段階で初心者である学習者（今の場合は著者だが、そうでない場合も）が正しく重要単語に印をつけることは期待できない。したがって、あとで消せる筆記用具を使い、実際不要となれば消すことにする。

　これはやってみると、教材により「すべての太字にマルを付けるだけ」となったり、逆に「難しくて印をつけられない」となることも多い。そのため、時間の無駄と考える人もいるかもしれない。したがって、やりたくない人は省略してもよいと思う。実行する場合、そのタスクテストは「本に印がついているかどうかざっと見る」のみである。



例 1.1.24 $\mathscr{C}$ を圏とする．以下のような対応 F は $\mathscr{C}$ から $\mathscr{C}$ 自身への関手となる．

- 任意の対象 $X \in \mathrm{Ob}(\mathscr{C})$ に対し，$F(X) = X$．
- 任意の射 $f \in \mathscr{C}(X, Y)$ に対し，$F(f) = f$．

これを恒等関手 (identity functor) と呼び，$\mathrm{Id}_{\mathscr{C}} : \mathscr{C} \to \mathscr{C}$ で表す．紛れのないときは単に Id で表す．

より一般の関手として，部分圏の包含が考えられる．

例 1.1.25 部分圏 $\mathscr{C} \subseteq \mathscr{D}$ が与えられたとき，以下のような対応 ι は $\mathscr{C}$ から $\mathscr{D}$ への関手となる．

- 任意の対象 $C \in \mathrm{Ob}(\mathscr{C})$ に対し，$\iota(C) = C$．
- 任意の射 $f \in \mathscr{C}(C, C')$ に対し，$\iota(f) = f$．

これを包含関手 (inclusion functor) と呼び，$\iota : \mathscr{C} \hookrightarrow \mathscr{D}$ と表すことにする．$\mathscr{C} = \mathscr{D}$ の場合は恒等関手に一致する．

以下のような，「構造を忘れる」関手は総称して忘却関手 (forgetful functor) と呼ばれる．

図4 単語に印を付けたところ

タスク3はストーリー1の構成（ここでは「圏論の技法」の1.1節の内容）を考え直すことである。1.1節は次のように見える。()の中には、著者が考えたことを示した。

1.1.1 圏
　　定義1.1.2　（圏の定義。長い。続けて単語の定義がある。）
→ 例　（たくさんある。）
→ 命題1.1.9　（証明短い。）
→ 定義1.1.13　（同型、、、などと、重要そうな言葉がある。）
→ 命題1.1.15　（同型の話らしい。）
→ 例　（そんなにない。）
→ 定義1.1.19　（部分圏の定義？　圏があるなら、その部分もあるだろう。）
→ 定義1.1.20　（「同型で閉じている」？）

1.1.2 関手
　　定義1.1.22　（関手の定義。長い。）
→ 命題1.1.23　（同型の話らしい。）
→ 例　（かなりたくさんある。）
→ 定義1.1.33　（関手の合成。）

著者は、実際にこのように書き出さないが、頭の中では実行する。これがその後の理解につながるからだ。

ここまで考えると、「定義1.1.2と定義1.1.22が中心になっている」ように見える。それは、「初心者の勘違い」かもしれないが、そのようにあたりをつけていく方法が良いと考えるのである。

タスク3のテストは、たとえば、「この範囲の重要そうな定義・定理・例は何か（どの辺にあるか）のチェック」だろう。著者の解答（正解とは限らない）は、「定義1.1.2

と定義１.１.22が重要そう。命題は３つくらいあった。例はたくさんあった」などとなるだろう。

タスク３が終わると、タスク４以降が書けるようになる。「タスク４以降を書くこと」をタスクテストの１つにしてもよいと思う。

タスク１、２をしない場合でも、タスク３（全体の構成を見る）は必要であると思う。しかし、これもタスクと数えなくてもよい。その場合は、著者がタスク４と書くものをタスク１として書いていくことになるだろう。

図５　タスク４以降も書き込んだもの（かなりラフだと思う）

上のノートではほとんどDef.やEx.の番号しか書いていないが、「これらを読み理解し、重要箇所は覚える」という意味である。これはいつも同じなので（Def.1.1.2以外では）書かなかった。

さらに、タスク４以降ではタスク実行時に

　重要単語をノートのタスク近辺に書き写す

ことにする。これが、本小論の最重要ポイントである。「近辺」とはどこでも良いが、著

者は、左ページにタスクを書き、右ページにそこにかかわる単語を書き写すことにした。また、細かいことだが、タスクテスト終了時には、タスクのすぐ右にチェックマークを入れている。

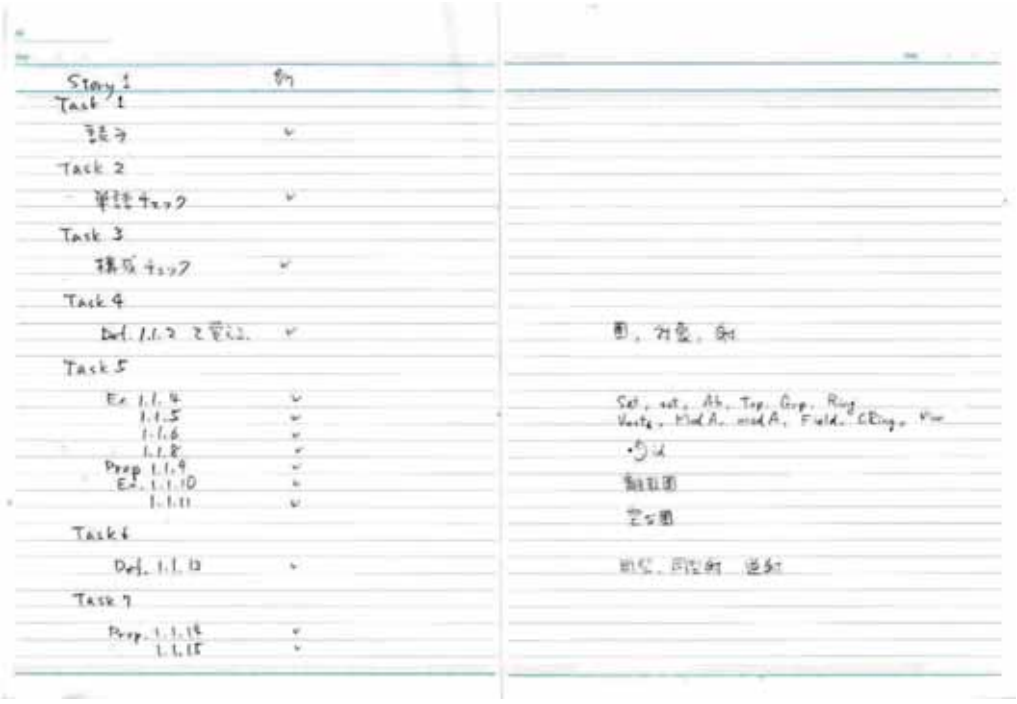


図6　タスクが進んでいく様子

タスクテストは、

　この単語を見ながら内容を述べる

とするのである。

たとえば、タスク4のタスクテストなら、「圏、対象、射」を見ながら、「圏は対象と射からなり、それらは次の公理を満たす。すなわち、ある射のコドメインと別の射のドメインが一致するなら合成できる。また、すべての対象に恒等射とよばれる射が対応し…」などと続けていく。これを言えればテスト合格（XPではテスト成功という）とするのである。

単語は、ある程度理解したと思った時に書き込むので、今度は勘違いの可能性は低いはずである。細かいことだが「それではテストファースト（テストを先に書く）ではないのでは？」と思われるかもしれない。しかし、「単語を見ながら内容を述べることができるようになる」がタスクの終了を意味し、その終了の前に単語を書くのだからテストファーストである。

「タスク4は『Def.1.1.2を覚える』だったのだから、何も見ずに言えるようにならなけ

ればタスク終了と言えない」と思われるかもしれない。著者は、その「覚える」の意味を「右側のヒントを見て言えればよい」としたわけである。もし、本当に「何も見ないで言えるべき」と考えるなら、右側の書き込みを「圏、対象、射」ではなく「圏」のみにすればよいわけである。

この方法に関して、いくつかの論点を質問・回答の形でまとめる。一般に「万人に適用できる学習法」はなく、ケースバイケースであるだろうと思う。しかし、これは著者が強く勧める方法である。

問：単語を並べておくより、問題をきちんと書いた方がよいのではないか。
答：初心者がそれをやろうとすると時間と労力がかかり効果が薄いだろう。
（教師が前もって作っておけるならそれも一つの方法だと思う。）
したがって、単語のみがよいと考える。

問：何も（単語も）書き出しておかなくてよいのではないか。
答：それでタスクテストが実行できれば書かない方がよい。
しかし、著書の場合実行できなかった。
多くの人も同様ではないかと思い、この方法を提案した。

問：タスクテストの解答はテスト実行時に書き出さないのか。
答：基本は、内心のチェックで良い。
なお、次項目も参照のこと。

問：模範解答をはじめに書いておかないのか。
答：「書くと覚える・理解が進む」と本人が思う場合は書いても良い。
しかし、それはテストではなく、タスクの１つ追加である。
学習者が自分で行うタスクテストには、模範解答の書き出しはしないことを強く勧める。問題の書き出し同様、労力が大きすぎて効果が少ないと考えるからであ　　る。
また、多くの人に「解答を書くと安心してしまい、そこで学習が終わりになる」という傾向があると思う。解答を書かないことで、何度も繰り返しやすくなると思う。この繰り返しが、「知識の統合」につながると考える。

問：タスクテストの正解ができたかとうかはどう判断するのか。
本人の自覚しかない。
ただし、最終的にはリリーステストがある。リリーステストは発注者が出すもので、発注者が教師の場合、そこで「通常のテスト」を行うことになる。学習者は、いずれにしても通常のテストを受けることを想定しながら学習しなければならない。

・タスクテストの正解がわからなかったらどうするか。
タスクをやり直す。
実は、タスクの内容を書き出した理由は、このやり直しをし易くするためである。

タスクをやり直すことは失敗ではなく、通常の作業である。
そして、タスクが足りないと感じたなら、書き足し、実行すればよい。

ストーリーテスト・リリーステストはどうするのか。今の場合、発注者＝著者であり、著者が自分で決めてよい。著者のストーリーテストは「該当範囲のまとめを圏論初心者に話し、理解させる」とし、実行している。さらにそれらをまとめることをリリーステストとしている。

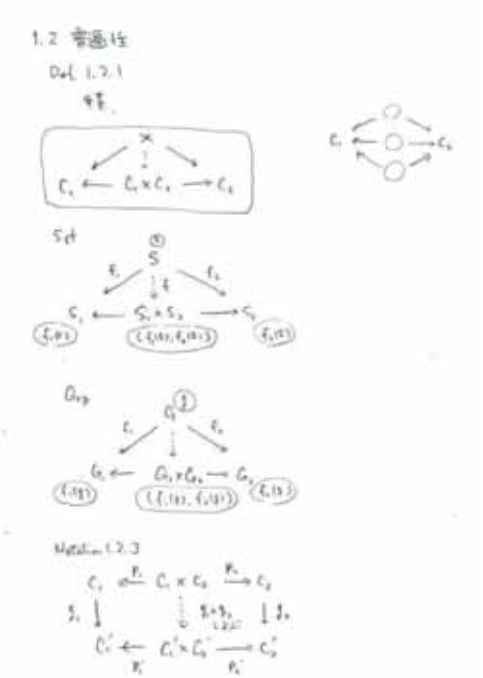


図7　説明用に書いたまとめ（リリーステストの一部となる）

「圏論の技法」の学習は、本小論執筆時に終わっていないが、順調に進んでいる。すでに終わったものは、上野健爾著「代数幾何入門」、A.Hatcher著「Algebraic Topology chap. 1 〜 3 」などいくつかあり、すべて満足できる結果になっている。

最後に、リリース・ストーリー・タスクの書き直しについて述べる。タスクは、「たぶんこのくらいがひとかたまりだろう」という予想に基づく「仕事の単位」である。もちろん、その通りに実行できるとは限らない。したがって、タスクは「できないから」という理由で頻繁に変更してよい。
ストーリーやリリースも変更してよい。しかし、なるべくなら「できないから」という理由で頻繁に変更しないのがよいだろうと思う。それは計画の破綻を意味するからだ。しかし、「最初に考えたリリースは初学者の誤解に基づくもので意味がなかった。よって変

更する」という変更は、「良い変更」である。それは理解が進んだことを意味するのだから。実際、それが学習だろう。そのような変更は、プロジェクトノートに書き込んでいくことになる。たとえば、著者の場合、ストーリー２ははじめ「普遍性を理解する」だったが、書き足して「積・余積を例にして、普遍性を理解する」と改めた。書き足しやすいように余白の多いノートを作っているのである。

３．XP式学習法の講義への応用

前節では発注者＝受注者、すなわち、自習の例を示した。著者は大学生の学習に自習は欠かせないと考えるので、この方法の学生諸君の自習への適用を考えたい。しかし、本節では大学講義への適用を考察する。著者は、C＋＋を教える「プログラミング基礎・基礎演習」を担当しており、前年度より［１］で示したXP型の教育を取り入れ、成果を上げていると考える。「成果」の客観的指標はないが、学生諸君の理解が向上していると感じるのである。また、授業満足度も「プログラミング基礎　100.0%　プログラミング基礎演習　95.8%」となっているので悪い結果ではないだろう。

プログラミング基礎は講義であり、発注者＝教員＝著者、受注者＝学生となる。講義であるので教員である著者がリリース・ストーリーを以下のように与えた。Rがリリースで、Sがストーリーである。

R1　画面に文字列を出力するプログラムが書ける。
S1　コンパイラが使える。
S2　プログラムの基本形を書くことができる。
S3　画面に文字列を出力できる。

R2　あいさつし、計算するプログラムが書ける。
S1　int変数やstring変数が作れる。
S2　キーボードから整数値や文字列を受け取ることができる。
S3　名前と整数を受け取り、あいさつし、受け取った整数を
　　10倍して表示するプログラムが書ける。

R3　クイズプログラムが書ける。
S1　if文が使える。
S2　for文が使える。
S3　クイズを出題するプログラムが書ける。

R4　占いプログラムが書ける。
S1　乱数を表示するプログラムが書ける。
S2　乱数によって表示する内容を変えるプログラムが書ける。
S3　占いプログラムが書ける。

R5　モンスターと遊ぶプログラムが書ける。
S1　基本的なクラスを書きそのオブジェクトを使える。
S2　モンスターのクラスが書ける。
S3　モンスタークラスを使ったプログラムが書ける。

R6　アドベンチャーゲームが書ける。
S1　アドベンチャーゲームの構成を作ることができる。
S2　複数のクラスを組み合わせることができる。
S3　アドベンチャーゲームが書ける。

R7　ファイルからデータを読み込むアドベンチャーゲームが書ける。
S1　データをファイルから読み込むプログラムが書ける。
S2　vectorが使える。
S3　ファイルからデータを読み込むアドベンチャーゲームが書ける。

R8　セーブ機能付きのアドベンチャーゲームが書ける。
S1　データをファイルに書き込むプログラムが書ける。
S2　書き込んだデータを読み込むプログラムが書ける。
S3　セーブ機能付きのアドベンチャーゲームが書ける。

R9　ポリモーフィズムを利用できる。
S1　クラスの派生ができる。
S2　仮想関数が定義できる。
S3　ポリモーフィズムを利用できる。

R10 C＋＋でプログラムが書ける。
S1　これまで学んだことを組み合わせられる。
S2　自由にプログラムを構想し書くことができる。

ただし、これは[１]で紹介したものを少し直したものである。主な変更点は、「R3 クイズプログラムが書ける」に対し、最後のストーリーとして同じ内容の「S3 クイズを出題するプログラムが書ける」を付け加えたことなどである。この方が目標を明確にできると考えたからである。

初年度は、ほぼ毎回講義でこの表を見せ、「何ができるようになったか。これから何をするか」を考えさせるという方式にした。「毎回」ではなく「ほぼ毎回」であった理由は、「あまり進んでいないときに同じ表の同じところを見せられても愉快ではないだろう」と考えた場合などにスキップしたのである。このようなスキップは、実践的な意味では良かったと思う。しかし、それによって学生に与えるはずだった「復習予習のチャンス」を勝手にスキップしたとも言える。

この問題の回避の意味も含めて、今年度からは、これらを印刷し全講義に携帯すること

を義務付けることにした。そして、タスクの内容や「タスクテストとなる単語」を余白に書き込ませるようにしたのである。その結果、前節で著者が作ったノートのようなもの（これもプロジェクトノートとよぶ）が完成することになる。

図８　プログラミング基礎のプロジェクトノートの使用

図８の印刷部分は著者が与えたリリースとストーリーで、手書き部分が学生が自由に書き込んだ部分である。また、ここではタスクは書かないことにした。手書き部分はこの論文の用語ではタスクテストになるが、学生には「来週の自分がいろいろ思い出すためのヒントを短く書くように（ぎっしりは書かないように）」と指示している。そして、これらのページを毎講義のはじめに見させ、「見て意味がわからなければ、より詳しく書いたノートなどを見るように」と言っている。

ただし、前節のノートより詰まっている。たとえば、前節では１つのストーリーのタスクが何ページにもわたることが多かったが、ここでは１つのリリースのすべてのストーリー

が１ページにある。これは「講義に合わせる」「それほど多くのタスクはない」などの理由からである。

プロジェクトノートを作らせ、活用させるための講義の手順は次のようになる。これは１つのリリースについての説明である。

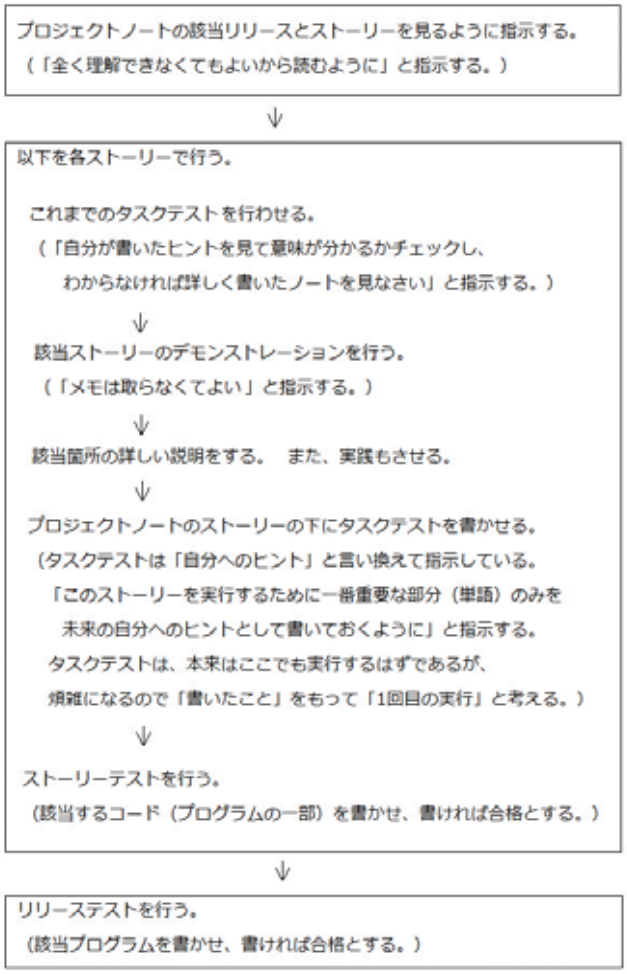


図９　１つのリリースでの手順

このリリースを繰り返していくわけである。

「タスクテストの書き出しと実行をちゃんとしたか」を教員がチェックするのは難しい。本小論では「やる気がない学生にやる気を出させる方法」は議論していないが、やる気があっても著者の意図を正しく理解せず異なることを行う学生はいるだろう。これは、プロジェクトノートを見ればある程度わかると思われる。しかし、ノートチェックをして点数

化するようなことはしない。それをすれば、「魂のこもっていない見せかけのノート」になる可能性が高いだろう。今年度は、教室を巡回する際に声をかける以上のチェックはしない予定である。

この講義自体途中であるが、現状ではうまく機能していると考える。

4．バリエーション

本小論で提案したプロジェクトノートを使うことの意義は次のようにまとめられる。

- 詳細な学習がはじまる前に全体を概観できる。
- 自分のなすべきことを書き出し、把握できる。
- ノートを見ることで「安心」できる。
  （学習者が膨大な教材の中で「自分は何もわかっていない」などと不安にならずに済む。）
- 学習開始時と終了時に短時間の復習ができる。
  （これは、XPで言うところの継続した統合である。）
- リリース終了後、「自分が獲得した能力」を具体的に述べることができる。

もちろん、この方法が適しない場合もある。たとえば、学習教材を頭から読んでいき、それでストレスなく理解でき要点も覚えていけるなら、わざわざプロジェクトノートを作らなくてもよいだろう。また、どう読んでもほとんど理解できない教材なら別の方法が必要になると思う。

一方、上記の利点をいかせるような状況では、その利点がいかせるようにバリエーションを考え、カスタマイズすべきである。

たとえば、著者の自習の経験では、「もともとなじみがあり、知っていることが5割前後の教材」にこの方法が使いやすい。したがって、講義に適用する場合、まず、学生諸君を「もともとなじみがあり、知っていることが5割前後」にするような前段階が必要であると考えた。図9のデモンストレーションがそれにあたる。

いずれにしても、この方法で重視されるのは「小さいテストをすばやく作り続け、繰り返し解き続ける」ということである。

「学習とは頭の中に知識・知恵を構成していくことであり、その知識・知恵はソフトウエアと考えられる」が本小論の出発点であった。ただし、コンピュータ・ソフトウエアは何もしなければ自動的に維持されるのに対し、頭の中の知識・知恵は何もしなければ簡単に消え去ってしまう。コンピュータ・ソフトウエアよりメインテナンスにコストがかかるのである。

しかし、「よりコストがかかる」というだけで、コンピュータ・ソフトウエアも頭の中の知識・知恵も、メインテナンスが必要ということにかわりはなない。（コンピュータ・ソフトウエアは人間が手を加えることで「崩壊」していく。）そのため、XPでは**テストは必ず残し**、頻繁に実行し、テストに失敗した場合はソフト開発の作業をし直すのである。

実はXPでは、ストーリーやタスクにはノートを使わず、カードに書き込む。そして、

一般的には、終了時にカードは破棄するとされる。

著者はXPを学習法・教育法に適用するにあたり、ストーリーもタスクもノートに残すことにした。その理由は、形（記録）を残していった方が、この方法に不慣れな利用者にはわかりやすく、復習（タスクのやり直し）もしやすいと考えたからである。

XPでカードを使い、達成後に破棄するのは、「立派な書類を作ることで無駄な時間を取らない」という意思表示でもある。著者はこの理念を支持するもので、プロジェクトノートにも無駄な時間を取られない（取らせない）ようにしたいと考える。一方、カードを破り捨てるのには「達成感を持たせ、モチベーションを維持する儀式」という側面もあると思う。プロジェクトノートを使う場合は、それほどドラスティックではないが「タスクの右横にチェックをつける」という形で行っているのである。

それでも、カードを使う方法もあり得るだろう。実際、著者はカード式で学習を行うこともある。

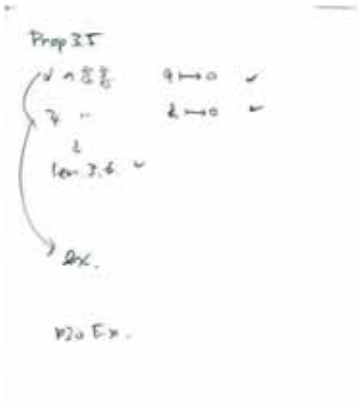


図10. タスクカードの例

この場合、タスクカードは適宜作成し達成後には破棄するが、タスクテストはノートを作って書き残し、頻繁に実行している。著者がカードを好むのは「終わりが見えない場合」である。プロジェクト（正しくは、継続的なプロジェクト群）があまり長く続くと、その間に状況（自分の能力、必要な知識、職業など）が変わってしまい、過去のタスクの記録自体はほとんど役に立たなくなるからである。

本小論の方法ではタスクテストとして重要単語を利用した。それは重要単語が情報のかたまりを代表するからである。しかし、課目によっては情報のかたまりを代表するものが単語という形にまとまらないかもしれない。たとえば、中学数学を考えてみると、単語より公式や例題が「情報のかたまりを代表する」かもしれない。しかし、公式や例題のすべてをタスクテストとしてノートに書き写すことは推奨できない。理由はこれまでに述べたように労力が大きすぎると思われるからである。あるいは、教科書の該当ページ数のみを記しておくのも本小論の方法論ではない。それではイメージができず、すばやいテストもできないからである。この場合、著者は「公式や例題の核心部分を想起できる短いメモ」

を作り出しノートに書くべきだと思う。また、英語学習では実際の英単語より構文などが重要になるかもしれない。その場合は、構文の名称などを利用すればよいと思う。いずれにしても、ケースバイケースである。

本小論の方法は、自習にも授業にも適用可能であることを主張した。しかし、正確に言えば、「著書自身の自習」と「学生の受ける授業」である。教育者としては「学生の自習」への適用こそがもっとも望ましい。そのためには、「授業でこの方法を用い、それから学生諸君に適用を促す」という方法がベストのように思うが、その「研究」はこれからである。

謝辞

ご協力いただいた静岡産業大学情報学部の先生方、また、学生諸君に感謝します。

参考文献

［1］ 小林健一郎『エクストリームプログラミングと教育法・学習法』静岡産業大学情報学部紀要第20号（2018）p119 - p136
［2］ K. Beck『XPエクストリーム・プログラミング入門』ピアソン・エデュケーション
［3］ K. Beck, M Fowler 『XPエクストリーム・プログラミング実行計画』 ピアソン・エデュケーション
［4］ 中岡宏行『圏論の技法』日本評論社