エクストリームプログラミングと教育法・学習法

Extreme Programming and Teaching/Learning Methods

小林　健一郎
Ken-ichiro KOBAYASHI

（平成29年10月 2 日受理）

要旨

ソフトウエアの開発には種々の技法が考えられている。本小論では「学習」をソフトウェア開発と考え、それらの技法のひとつであるエクストリーム・プログラミングの適用を考察する。その際、学習者を開発者に対応させるが、発注者は、教師の場合と学習者本人場合の 2 種類がある。ここでは、主に発注者が教師である場合を想定するが、それは読み替えによって、「発注者が自分」にも適用できるはずである。また、ここで扱う学習は、主に「（教師にアシストされた）学習者の自習」になる。

1．序論

ソフトウェア（以下、略してソフト。本論文ではプログラムと同義とする）の開発は、一般に、顧客・ユーザが開発者に発注して行われる。しかし、ソフト開発には大きな困難[1]があり、しばしば失敗してきた。失敗とは、次のようなものである。

- 納期に間に合わない。
- バグ（プログラム上のミス）がたくさん出る。
- 発注者の意図を実現していない。

これらは、

- 発注者の意図が開発者にうまく伝わらない。
- 発注者の意図が頻繁に変わる。
- 開発者が自分のミスに気が付かない。

などを原因とする同根の問題でもある。（納期に間に合わないのは、問題が時間内に解決できない場合に起こる。問題が発覚する前に納品してしまえば、バグや要求の実現不足になる。）

その背景には「巨大なソフトを人間（発注者も開発者も）が把握するのは難しい」ということがある。そこで、巨大なソフトは「部品」に分割して開発し、組み上げるという方法がとられる。しかし、分割しても全体を把握する必要はあり、やはりそれは難しいので

ある。

この問題に対処するため、「ソフトウェアの開発手順」（開発プロセスという）も研究されてきた。開発プロセスは絶対的な解決策ではなく、一般に賛否があるが、一定の成果をおさめてきたと考えられる。

本小論の目標は「教育法・学習法の開発」である。ただし、主は学習法であり、そのサポートとしての教育法を考える。学習とは、人間の脳の中に知識を組み込んでいくものであるので、広い意味でソフト開発であると考えられる。したがって、ソフト開発の技法が学習法に適用できると考えるのである。

そもそも人間の活動の多くは精神的なものであり、精神的な活動の「失敗の少ない効率的な方法」はいろいろな分野で研究されている。それらの多くは、いわゆる「人間の心理」に関するものとなるであろうし、当然、共通するものが多いと思う。したがって、異なる分野の研究（ここでは、ソフト開発という分野の研究）を利用することは有意義な事だと考えるのである。

たとえば、最古の開発プロセスはウォーターフォール型とよばれるものである。これは、ソフト開発の手順を最初に（長い時間をかけて）決定し、その後はスケジュールに従って開発していくというものである。その特徴は、「後戻りせず、計画に従う」である。これは、「学習計画をたて、後戻りせずに学習を進めていく方法」と同じである。

一方、ウォーターフォール型には、「問題が発生した時の対処が難しい」「問題を発生させないために、相当期間の準備がいる」などの困難がある。それに対して現れたのは、スパイラル型などとよばれる開発プロセスである。これは、ソフトウェアを本格的に開発する前に、少しずつ試作品を作り、発注者と意見を合わせていくという方法である。この方法も有用であるが、開発がはじまるとそこからウォーターフォール型と同様になり、したがって、その欠点を（その局面では）受け継ぐことになる。

その後、エクストリーム・プログラミング型（XPと略記する）とよばれる開発プロセスが提唱された[2][3]。詳しくは後で述べるが、「多くのプログラマに良いと思われていた開発方法を極端にして実行する」というもので、ひとつひとつの方法はXPより前に実行されているものである。XPの批判者の中には「特に目新しくない」という人もいる。しかし、そのような「特に目新しくないもの」の中から「組み合わせることでより有用になるもの」を選び出し、徹底して実行することに、著者は大きな可能性を感じ研究を続けている。

上で「ウォーターフォール型は計画を立てて進める学習方法」と同じであると書いたが、スパイラル型やXP型に相当する教育法・学習法は何だろうか。本小論では、XP型の教育法・学習法への適用を考える。

本小論で、教育・学習をソフトの開発プロセスになぞらえる場合、学習者を開発者に対応させる。開発者は発注者の求めに応じて「自分の能力」を開発していくのである。発注者は、教師の場合と学習者本人場合の2種類がある。前者は教師が「こうなってほしい」と指示する場合であり、後者は、学習者自身が「こうなりたい」と考える場合である。したがって、発注者が教師の場合は教育法、発注者が本人の場合は学習法となるだろう。著

者は、一般的な高校生以下の学習者には教師が発注者であり、大学生以上には自分が発注者となると考える。ここでは、主に発注者が教師である場合を想定するが、それは読み替えによって、「発注者が自分」にも適用できるはずである。また、ここで扱う学習は、主に「(教師にアシストされた) 学習者の自習」になる。

ただし、ソフト開発の場合、「受益者＝発注者」であるのに対し、教育・学習の場合、受益者＝学習者であるので「受益者＝開発者」となる。この事実は、特に重大な問題にはならないと考えるが、指摘はしておきたい。

XPに関しては次節以降で説明するが、ひとつコメントをしておきたい。上で述べたようにコミュニケーションの問題がソフト開発では重要になる。それは、単純に「言葉が足りない」ということもある。しかし、一方で、発注者（お金を払う側）と開発者（お金をもらう側）の力の差からくる「発注者のルール無視」が問題になることも多い。これは発注者が「悪い人」という意味ではない。単に、力を持った側が「自分の気持ちは無条件に通じているだろう」、「通じていなくてはならない」と思い込むことで起こる問題である。これはしばしば教育現場でも見られることではないだろうか。XPはこの問題を「責任と権利を明確にすること」で解決しようとする。本小論で、発注者・開発者という言葉を多用するが、それはそういう理由による。実際の教室で発注者・開発者という言葉を使うことを想定しているわけではない。

2．XPと教育法・学習法

XPは、基本的に「責任と権利を明確にし、コミュニケーションを密にして、それによってミスを減らし、開発時のミスのダメージを最小限に抑えよう。情熱や信念のようなものに頼らず、合理的に無理なくやっていこう」というものである。もちろん、他のプロセス同様、「全体をうまく分割すること」も考える。

XPには、いくつかのプラクティス（実行すべきこと）がある。それが具体的なXPのメソッドであるので、本節では「XPエクストリーム・プログラミング入門[2]」より引用（若干表記を変えている）し、それぞれに簡単な説明をつける。これは、著者が「一般的に認められていると考える説明」である。また、続く［適用］には、教育法・学習法への応用・適用を簡単に述べるが、「著者の視点からの説明」も含める。こちらは一般的に認められている保証はない。

なおコードとは、「プログラム全体やその一部」のことである。

⑴　計画ゲーム

ソフトの機能をストーリーの集合として分割し、その開発を計画する。

ストーリーは「コンピュータ画面で商品を確認する」などで、発注者の言葉で書く。

［適用］

ストーリーは、「そのソフトで何がしたいの？」に対する素朴な答になる。

ストーリーは、開発者によってタスク（手順、するべきこと）に分解される。

教育法・学習法に当てはめると、ストーリーは「勉強の結果、生徒がどうなればよいか」

に相当する。ただし、「どうなればよいか」とは、「宇宙飛行士になる」とか「自分で物事を考える力を持った人になる」のような大目標ではなく、「２次方程式が解ける人になる」のような小目標である。

⑵　リリース計画

リリースとは「完成」という意味である。

XPでは、「機能（ストーリー）をしぼって、その部分だけを完成させる」を繰り返し、その１つ１つをリリースとよぶ。リリースの間の時間（開発時間）は短く取る。

短期で繰り返すリリースにより、発注者に使用してもらい、問題があればすぐに直すことができる。

［適用］

「短期間でのリリース」の主目的は、上記のように「発注者からのすばやいフィードバック」であるが、それ以外に「開発者が持続的に達成感を得る」という効果が大きい。３年かけてソフトをリリースするより、３カ月に１度の（小さな）リリースを12回繰り返す方が、精神的に楽だろう。３年かけて作ったソフトを発注者が「考えていたものと違うから、賠償してほしい」と言う可能性もあるのだから、なおさらである。当然、これは計画ゲームと深く結びついている。

教育法・学習法に当てはめると「こまめに達成感を得る（得させる）計画」に相当する。

⑶　メタファ（暗喩）

ソフトの全体的な機能を厳密にではなく、「こういうものだ」という形で記述する。

［適用］

ストーリーは個々の機能に対応するが、メタファは全体の機能に対応する。

教育法・学習法で言えば「2次方程式を解ける人」「江戸時代を理解している人」のイメージを明確化することに相当すると思う。モチベーションを維持したり、学習の方向性を失わないために重要だと思うが、本論文では論じない。

⑷　単純な設計

ソフトウェアはプログラミング言語で書かれるが、その前に、構造を決めておく。それを設計という。それをできる限り単純にする。

［適用］

本小論の視点とは違うが、たとえば、授業を「作成すべきソフト」とすると、「年間授業計画」が設計に相当する。ただし、本小論での「作成すべきソフト」は学習者の（学習者個人の）学力である。

⑸　テスト戦略

テストをまず作り、それからコードを書く。

テストを自動化する。

［適用］

おそらく、XPのプラクティスとして最もよく知られ、多くの人に有意義だと思われて

いるもので、XPの反対者の中にも「これだけは採用する」という人もいる。

一般に、プログラマは自分が書いたコードのテストをしなければならない。ここで言うテストは、「ソフト全体の動作テスト」ではなく、「プログラマが受け持ったごく小さな“ソフトの一部（コード）”のテスト」のことである。これを単体テストという。当然、ソフト全体が正しく動くためにも、単体テストは重要である。しかし、多くのプログラマは、単体テストを好まない。「自分が書いたコードが正しいかどうか」のチェックであるから、「たぶん、大丈夫なはず」として避ける傾向があると言われている。この心理は学習者が自己診断テストを避けるものに似ていると思う。

そこで、提案されているのは、「コードを書く前にテストをする」ということである。コードを書く前にテストをするので、最初は確実に「失敗」という結果がでる。その上で、そのテストを「成功」させるべくコードを書くのである。

そうすると、「コードの完成＝テストの成功」となるので、心理的にやりやすいわけである。これをテストファーストという。

学習で当てはめれば、「勉強する前にテストをする」というものである。実際には、テストを見て「できないよね。まだ勉強していないのだから当然だよね」と確認することになる。

「テストの自動化」とは、「テストを簡単に（クリック１つとか）実行できるようにしておく」という意味である。これを学習に当てはめるには、たとえば、コンピュータアシストが考えられるがここでは論じない。

⑹ ペアプログラミング

プログラムは必ずペアで書く。

［適用］

テストファーストと並んでよく知られたものだが、反対者も多い。XPの支持者は、「２人のプログラマが独立に８時間コードを書くより、２人が一緒に８時間コードを書く方がより多くのコードが書ける」と主張する。普通に考えれば、独立に書いた方が2倍書けるはずであるが、「プログラム中に発生するバグを1人ではなかなか直せず、かえって時間がかかってしまう」ということである。

著者にももっともらしいと思える反論は、「プログラミングは個人芸であってペアではできない」「プログラマのレベルが違う場合うまくいかないだろう」といったものである。

著者はXPの支持者だが、ここでこの議論自体はしない。教育法・学習法への応用は４節で述べる。

⑺ コードの共同所有

一般的には、コードを書いたプログラマがそのコードに責任を持つ。

したがって、「他人がそのコードを勝手に手直しする」などということはない。しかし、XPでは、それを肯定する。テストが自動化されていれば、他人のコードを直しても、問題がないと考えるのである。

［適用］

これもXPの反対者が嫌うものだと思う。

教育法・学習法への適用として、「共同ノート」などがあるかもしれない。しかし、ノートは頭の外にあるものであり、本論文が対象とする「ソフト」は頭の中にある。したがって、これは普通の意味では適用できず、論じない。

⑻　継続した結合

個々のプログラマが書くものはソフトの一部である。それらを最終的にまとめあげて一つのソフトにするわけである。そのまとめあげを結合という。
XPでは、毎日（しかも、可能なら何度でも）書いたコードを結合してチェック（結合テストという）すべきとしている。
［適用］
一般に、「1つ1つ正しく動作するコードでも、結合すると正しく動作しない」ということが起こる。コード間に矛盾がある場合である。そこで、このプラクティスがある。これは、開発をしていない人から見れば、むしろ当然に思えると思う。しかし、現実には結合チェックを後回しにすることも多い。理由はしばしば「時間がない」などであるが、心理的には、単体テストをしない理由と同じだと思う。
教育・学習では、「今日得た知識とこれまでに得た知識の統合」に相当すると思う。

⑼　1週間40時間労働

読んで字のごとしである。
［適用］
教育・学習にあてはめれば、「オーバーワークさせない」ということであり、当然なので論じない。

⑽　オンサイトの発注者

発注者は開発者と同じ建物にいて、開発者の質問に応える。
［適用］
これは「発注者に開発者の会社にずっといてもらう」ということであり、XPの反対者が大いに嫌うものである。彼らが嫌う理由の代表的なものは、それが有害とか無駄ということではなく、「現実的に無理」ということである。XPの支持者でも、「これは無理」という人は多いと思う。
教育・学習に当てはめれば、たとえば、「教師が生徒の家にずっといる」というようなものである。ネットの発達によって疑似的には可能かもしれないが、本論文では論じない。（できれば有用だとは思う。）

⑾　コーディング既約

プログラミングのスタイルを統一する。
［適用］
教育・学習への適用は考えられないので、本論文では論じない。

⑿　リファクタリング

プログラムを恒常的に改良し続けること。

［適用］

しばしば、「プログラムを“改良”すると、実際には改悪になっている」ということが起こる。そのため、「なるべくいじるな」という考え方がある。しかし、XPでは、「テスト戦略によりそうはならない。したがって、改良を続けよう」と考える。

教育・学習で言えば、「自分の知識を常に向上させ、磨き続ける」ということになる。極めて重要なことで、これこそ「学習の要」だと考えるが、「自覚を促す」以上のことは言えないので、ここでは論じない。ただ、教育者であるならば、常に学習者に「自覚を促す」必要はあると思う。

本論文で論じるのは、計画ゲーム、リリース計画、テスト戦略、継続した結合である。また、ペアプログラミングは別途論じる。

3．具体例による説明

ここでは、中学の数学と歴史を例に説明したい。ただし、ここで論じているのは形式であって、本当の「数学や歴史の学習法」ではない。

3－1　計画ゲーム

学習者が学習すべき内容は広大である。したがって、分割して学習すべきであろう。その際、「どう分割するか」「それをどう学習していくか」という問題が発生する。それが計画ゲームである。

学習は、学習者に（学習の前後で）変化をもたらす。そこで学習の結果「学習者がどうなるべきか」をストーリーとよぶ。これは、XPのストーリーに直接的に対応する。XPでは、ストーリーは発注者と開発者の共同作業で作られる。教育・学習としては、大学生以上の場合、共同作業とする（ゼミなどの場合）か、本人のみで決める（通常の授業に対応する自習）ことになると思う。しかし、高校生以下の場合、教師が（発注者として）指定するのが自然だろう。もちろん、状況によっては、「話し合う」というオプションも可能だと思うが、ここでは単純に「発注者（本人でも教師でも）が決定する」とする。

実のところ、高校生以下の学習に限るなら、基本的な分割は済んでいると考えられる。それは、教科書や参考書の章立て・節立てである。また、参考書では、「ここは理解する／理解しなくてよい」「ここは覚える／覚えなくてよい」と指示されていることもある。

以上より、ストーリーは次のようなテンプレートに乗るだろう。

○○を理解する。
○○を覚える。
○○ができるようになる。

他に、「○○になじむ」とか「○○を鑑賞する」のようなものも工夫すれば可能だと思う。ただし、XPでは実現度を測定する。すなわち、テスト可能なものでなければならない。そのため、最もやりやすいのは「○○を覚える」「○○ができるようになる」などだろうと思う。（教育工学でも「○○を理解する」は曖昧なため目標とすることを嫌うとのことである。謝辞を参照。）

中学の数学なら、

・二次方程式の解の公式を覚える。
・二次方程式が解けるようになる。
・面積の問題が解けるようになる。
（ここでの面積の問題とは、「二次方程式を使って解く面積の問題」である。）

などとなるだろう。

また、中学の日本史なら、

・江戸幕府の支配のしくみを覚える。
・鎖国を完成させていく年号を覚える。

などとなるだろう。

これは、小学校や中学校の講義初めに示される「この単元の目当て」のようなものであり、特別新しい提案ではない。事実、XPのすべてのプラクティスは奇異なものではなく、「すでに知られているもの」である。ただ、このようなプラクティスを総合していこうということなのである。

次のようなものは、ストーリーとしては排除すべきだと考える。

（「あらゆる教育法・学習法、あるいは、あらゆる局面で排除すべき」という意味ではなく、「XPの適用としては排除すべき」という意味である。これは以下でも同様である。）

・曖昧なもの。範囲が膨大なもの。
例：基本問題がすべて解けるようになる。
例：日本史を理解する。
・達成のイメージがしにくいもの。
例：教科書の10ページから35ページを理解する。

ストーリーは、「プロジェクト型授業」の例に挙げられる「外国人旅行者の誘致方法を考える」「駅前商店街の活性化の方法を考える」などではなく、あくまでも「学習者の学力をコツコツ上げていくストーリー」である。

また、以下の注意がある。

・ストーリーは、なるべくまとめて提示すべきである。

・学習者が未熟である場合、教科書等の参照個所を示すべきである。
・テスト可能なストーリーでなくてはならない。
・複数の目的や「隠れた意図」を持たせてはいけない。

XPでは、最優先されるストーリー群が実現された時のことを、「最初のリリース」とよぶ。そして、「次に優先されるストーリー群が実現された時」を「次のリリース」、、、と続けていく。教育・学習で考える場合でも「関連するストーリー群が1つ実現できた」をリリースと考えるのが適当だろう。ただし、用語法として、「１つのリリースで実現すべきストーリー群」も簡単に「１つのリリース」とよぶ。これは日本語として自然だと思う。

上の数学の例で言えば、たとえば、次のようにできる。

リリース１：二次方程式の基本を身につける。
ストーリー１　二次方程式の解の公式を覚える。
ストーリー２　二次方程式が解けるようになる。

こうした場合、先に示した「面積の問題が解けるようになる」は、次のリリースに入れることになる。

発注者はこのようなリリースの最低１つ分を同時に提示しなければならない。開発者（学習者）は、「自分が何をしているのか」を知らなければならないからである。また、リリースが複数ある場合は、それらの優先順位も示すものとする。

実は、XPでは、ブレインストーミングのようなやり方で、発注者と開発が共同でストーリー群を書き出し、それをまとめてリリースにしていく。これは次節で見るリリース計画の一部（最初の部分）である。たとえば歴史の学習で、著者には「古代メソポタミアを先に学ぶか、古代中国を先にするか」に自然な順番は見えないので、話し合いで決めてもよいと思う。しかし、リリースの最終的決定権は発注者にある。教育でも同様であろう。したがって、途中「生徒と先生が話し合う」などのオプションをいれたとしても、最終的にはストーリーとリリースは発注者が決めるものとする。

３－２．リリース計画

最初のリリースが決まれば、それに含まれるストーリー１つ１つを見て、それぞれの実現のための手順を考える。この手順１つ１つをタスクとよぶ。タスクは開発者が決めることで、発注者が指示してはいけない。そして、開発者は自分が決めたタスクを発注者に報告する義務もない。ソフト開発失敗の原因の１つは、発注者と開発者の役割分担の曖昧さからくる混乱である。XPでは、役割分担を明確化するのである。

教育法の中には、学習者に「どう勉強したか」を報告させたり、さらには、それによって成績評価をつけるものもあるが、本小論のメソッドでは、それをしない。仮に「どう勉強したか」を問題にしたいなら、「勉強法を考え出す」をリリース（あるいは、ストーリー）とし、その実現を要求すればよい。ストーリーが「解の公式を覚える」なら、それ以上の意味を持たせてはいけないのである。

なお、報告の義務がない以上「タスクを紙に書く」などの義務もない。しかし、学習者には「走り書きでよい（清書しない方がXP的）から、一度書き出して検討すること」「タスクが１つ終了するごとにチェックしていくこと」を強く勧めたい。（教育の場合、発注者、開発者以外に、相談を受ける支援者がほしいと感じる。実はソフト開発でも同じで、それをメンターという。上記の「勧める」はメンターの仕事である。）

著者は教師の側にいるので、タスクを決めてはいけないのだが、説明のために例を挙げる。もちろん、学習者にも例示する必要があるだろう。

「ストーリー：二次方程式の解の公式を覚える」を実現するためのタスク群（例１）
タスク１　教科書○ページの導出をよく読む。
タスク２　教科書○ページにある公式を紙に書く。
タスク３　問題集○ページにある基本問題を解く。

「教科書○ページ」のような指定はストーリーでは避けるべきだが、タスクではかまわない。例1にタスク1がある理由は、学習者が「理解していないものは覚えられない（覚えにくい）」という発想の持ち主だからだろう。ストーリーはあくまで「覚える」であり、それ以外のことをそこでしてはいけない。しかし、覚えるために理解する必要があるのなら、タスクの中に入れてよいわけである。
一方、「理解しなくてもいい」と思う学習者なら

「ストーリー：二次方程式の解の公式を覚える」を実現するためのタスク群（例２）
タスク1　教科書○ページにある公式を紙に５回書く。
タスク2　問題集○ページにある基本問題を解く。

とするかもしれない。どういうやり方であれ、このストーリーが実現できれば、発注者（教師）は満足しなければならない。公式を覚えてきた生徒に「覚えていても、導出ができなければだめだ」などというのは反則である。
もし、不満があるなら、不満が出ないように、ストーリーを書くべきであった。ストーリーに「もうひとつの意図」や「隠れた目的」を持たせてはいけないのである。ただし、XPでは、発注者が無謬であることを仮定しないので、ストーリーを変更したり、追加することもできる。たとえば、「解の公式を導出できるようになる」をストーリーとして追加することも可能である。しかし、それは「そのリリースのあとに」である。
もちろん、タスクをより具体的に書く学習者もいるだろう。

「ストーリー：二次方程式の解の公式を覚える」を実現するためのタスク群（例３）
タスク１　$ax^2 + bx + c = 0$ を平方完成し、その手順を覚える。
タスク２　タスク１の結果から解の公式を導出し、結果を覚える。
タスク３　教科書の該当する基本例題を解く。

まったくの初心者がこのように書けるかどうかはわからない。もしかすると、教科書から言葉を拾っただけかもしれない。どうであっても、ストーリーが実現できればよいのである。

歴史の例では、たとえば、次のようなものが考えられる。

「ストーリー：・江戸幕府の支配のしくみを覚える」を実現するためのタスク群（例）
タスク１　教科書○ページの図をノートに写す。
タスク２　大老、老中、若年寄、大目付、目付、三奉行の役割を覚える。
タスク３　郡代、代官、所司代、城代を覚える。

もちろん、「上記のタスク１を最後に持ってくる」「まとめノートを書く」などのバリエーションもあるだろうが、開発者がどう決めてもよいのである。

開発者（学習者）にとってもっとも重要な作業はタスクだろう。したがって、５節の具体例でもう少し論じる。

３－３．テスト戦略

ソフト開発では、ソフトの完成までに多くのテストを行う。XPで言えば、タスクごと、ストーリーごと、リリースごとにテストができる。さらに、XPのテスト戦略は、はじめにテストを作り、そのテストに合格したとき、それを「完成」とよぶ。著者はこれが最も合理的なテスト戦略だと思う。（「テストの結果が悪くても開発者の人柄や熱意が良いソフト」で動く飛行機や手術機器を利用したい人がいるだろうか。）

XPの原則に従って教育・学習にあてはめると、タスクの完成を確認するテスト、ストーリーの完成を確認するテスト、リリースの完成を確認するテストが考えられる。これらを、本小論ではタスクテスト、ストーリーテスト、リリーステストとよぶ。（一般的なXPでは、タスクテストとストーリーテストは区別せず、単に「コードのテスト」というだけであると思う。）

発注者（普通は教師）はリリースを提示するときに、同時にリリーステストを提示しなければならない。当然、提示した段階では、開発者（学習者）はそのテストをクリアできない。「できないから見ないでよい」ではなく、はじめによく見せて、「今これができないのは当然である」「これができれば合格である」「他は望まない」ということを確認しておくことが重要である。開発者の人柄や努力とはまったく関係なく、テストに受かればそのリリースは終了である。当然、リリーステストをあとで変更するのはルール違反であり、極力避けるべきである。（変更する場合は、発注者が自分のミスであることを認め、変更に伴う処置を考えなければならない。）

一方、タスクテストは開発者（学習者）が自分で作る（考える）。タスクを考えた時に、「どうなったらそのタスクは終了か」を自分で決めるのである。これは発注者に無関係なことである。タスクを実行する過程で、自分でタスクテストを追加したり、変更することは大いにあり得るだろう。ただし、タスクテストを追加したり変更するのは、発注者ではなく、開発者である。

ストーリーテストは、一般に、開発者が主導して作るものである。開発者が大学生以上ならそれでよいと思う。しかし、高校生以下ではそれは難しいかもしれない。その場合は、発注者が示してよいと思う。

すると、発注者（普通は教師）は、実にたくさんのテストを用意しなければならないということになる。これは強調しておきたいが「本当にたくさん」である。本論文のメソッドに従うなら、教師は「教える人」というより「テストを出す人」である。（もう少し好意的に書くと「テストを通して教える人」である。これらのテストは選別するためのものではなく、成長を促すためのものだから。）たとえば、「ストーリーテストは行わずに、あとでまとめてテストする」では、XPの根本的主張に反していることは指摘しておきたい。

テストの用意方法は本小論の主題ではないが、いずれにしてもテストが用意できないのであればこの方法は実現できないので、著者の意見を述べておく。しかし、それを強く主張するものではない。

いずれの場合もリリースやストーリーに厳密に対応した問題である必要がある。まず、利用できるのは市販の問題集だと思う。リリーステストに使えるのは、一般的な教科書の章末問題にあるような問題だろうから、対応する問題集も数多くあると思う。一方、ストーリーテストになるとかなり範囲を限定するテストなので、問題の収集はあまり簡単ではないだろう。小テスト的な問題集や一問一答式の問題集が利用できるかもしれない。もし、本小論のメソッドに賛同する教員が多数いれば、手分けして作ってデータベース化してもよいと思う。

タスクテストは、開発者が考えるものだが、「終了したことを自分でチェックするように」という指示は最初に出しておかなければならない。（学年等状況によっては、テストという言葉は避けてもよいかもしれない。）儀式的なチェック（「目をつぶって思い出してみる」等）であっても実行するよう指示すべきである。小さな達成感を積み上げることがXPの一つの方針であるからだ。

慣れるまでは発注者が例を示してもよいだろう。あるいは、「２次方程式の解の公式を覚えるためのタスクは何？」「じゃ、そのタスクが成功したかどうかはどうやって確かめるの？」と聞いてみるのもよいかもしれない。ただし、その会話が成績評価につながっていないことを保証し、それを確信させなければならない。「紙に書く」を推奨してもよいが、「それを提出させて教師がチェックする」「父母に印をもらって確認する」はXPの趣旨に合わない。タスクの結果できあがるのは、「頭の中の知識」である。そのために使った紙やノートは副産物にすぎない。だから、副産物に注目しすぎてはいけないのである。このような副産物を残しておくかどうかはXPの支持者でも意見がわかれる。ウォーターフォール型などでは、副産物も成果物（将来の参考とするため）として残しておくのが主流だと思う。しかし、変化を受け入れることを重視するXPでは、「副産物を残すかどうか」は「思い出の品は残すのが良いか、捨てるべきか」の選択であり、「思い出の問題」でしかない。教育・学習でも、同様に判断すればよいと思う。著者個人は、達成感の持続のために残す傾向にあるが、実際に役立つことはまれと感じる。

3－4．継続した結合

ソフトは「部品」にわけて開発し、それを組み上げて「完成品」とする。その組み上げを結合という。「正しく動く部品」を結合したものが動作しないことはよくある。そのため、結合後に結合テストを行う。XPの主張は、「結合と結合テストをコードを書くごとに実行する」というものである。それは、事実上、1日に何回も実行するということになる。

教育・学習では、これは、「新しく獲得した知識とこれまでの知識を結合する」ということである。人間はこれを意識せずとも実行するものかもしれない。しかし、漏れを防ぐためにも、意識して行うべきである。実際、著者の印象では、「勉強のできる子」は、この結合を不断に続けている学習者だと思う。

たとえば、「2次方程式の解の公式」を知識として獲得した場合、それまでの知識（因数分解や平方完成など）と矛盾がないかをチェックすべきである。実際、「矛盾」はないはずだが、「腑に落ちない」ということは起こり得る。それをよく考えて決着させれば、「結合が終了した」ということになる。「江戸幕府の支配のしくみ」を理解したなら、その前に学習した「江戸幕府の成立」とどう結合するかチェックすべきである。

発注者は、このような結合を指示しておかなければならない。例を示して説明もできるだろう。しかし、知識の結合テストを提示することは、残念ながら難しい。（もちろん、それを意図するテストを作り出すか見つけ出すことは可能であり、できれば実行すべきだろうが。）教師に時間とエネルギーがあるなら、「口頭試問」のようなものが結合テストに使えると思う。「前に学習した○○と今学習した△△とはどういう関係になるのかな」と聞くのである。意欲ある学習者がそろっていれば、「討論」という形で実行できるかもしれない。

4．ペアプログラミングの適用について

「ペアプログラミングはプログラミングをペアで行う」というものである。これは、二人でひとつのコンピュータ画面を見ながら、プログラムを書いていくものである。その際、一人がキーボードを持ち（ドライバとよぶ）、もう一人はキーボードを持たない（ナビゲータとよぶ）。プログラムはドライバが書き、ナビゲータはドライバがアイデアに困ったときや自信がない場合には「相談役」になるが、そうでないときは「チェック係」に徹する。これは、新人とベテランを組ませて行うOJT（実務を通した職業教育）ではなく、通常の業務として行う。プログラミング上の最大の問題は、「ミスをしても自分では気が付かない」であるが、ペアを組むことで、それが減らせると考えるのである。

これには、「実際にペアにしたら効率が上がった」という報告もあるが、反対論もあり、その議論自体はしない。ただ、これをプログラミングの教育法として取り上げる人もあったことは指摘しておきたい。（もちろん、ここでは教育が目的である。）著者もその一人である。たとえば、「１つのプログラム課題を２人でやらせる」を実行したことがある。しかし、その「実験」は失敗だった。学生２人ともに責任感が足りない場合、あるいは片方に足りない場合、残念ながら進捗はよくなかったのである。その状況を見て、即座に中止した。

ペアプログラミングは、普通は１人でやるものを２人でやるのだから、共同行動の訓練が必要である。その訓練自体にそれなりの時間がかかり、その間は結果も出ない。また、そのような状況をコントロールするには教師の側にも経験が必用だと思う。したがって、ペアプログラミングを「ちょっとやってみる」のは難しい。

しかし、教育法としてのペアプログラミング、あるいは、単にペア学習には、成功すれば一定の利点があると考える。それは、

・同等の学習者を見ることで自分を客観視できる。
・教えることで知識が身につく。

ということである。したがって、学習者に強い動機付け（やる気）を与えられれば、その困難を軽減でき、良い結果がでるかもしれない。著者は、新しい工夫として、次のように行った。

・「課題」を「テスト」と言い換えた。
・ドライバを受験者、ナビゲータを採点者と言い換え、採点者に採点させた。
・採点は次の基準とした。
　採点者は受験者が課題のプログラムを完成できるかチェックし、
　補助なしで完成できれば20点を与える。
　もし完成できなければ、補助をして良い。
　補助されて完成できた場合は10点を与える。
　補助されても完成できない場合は０点とする。

これを５回繰り返した（それで100点満点になる）。ただし、「補助されても完成できない学習者」は実際にはいないように人員を案配した。

点数がざっくりしているのは、採点より補助が重要だからでもある。また、採点の仕方によって感情的な問題を発生させないためでもある。

学習者は以前にペアプログラミングをさせた学生ではなく（10年以上の時間差がある）、新規の学生である。「課題をテストと言い換える」などは、小手先の工夫にすぎない。また、本来は「補助者」とよぶべき相手を「採点者」とよぶのも正しくはない。しかし、それでも「最初の試み」として、著者の感触では、「うまくいった」と思う。これを正しく実現できるよう、現在も検討しながら進めているところである。

また、論文執筆時には担当していなかったため実現できていないが、数学などでペアプログラミングの応用ができないか、考察している。ただし、対象は市販の教育指南書にあるような「頭を使う面白い問題」ではなく、公式をそのまま適用するような単純な計算問題である。それらを速く確実に実行できることがさらなる数学力の構築のために必須であり、しかも、単純な問題の方が、ペアプログラミングのやり方に沿うと考えるからである。「教師が計算している手元を凝視していて、誤りを即座に指摘する」ということを嫌う学習者は多いと思う。また、教師が全学習者を見ていることも難しい。一方、計算のチェッ

クなら同級生でもできるし、やり方によれば教師に見られるより嫌がらないのではないだろうか。

もちろん、「生徒同士の採点」なら特に目新しいものではないが、ここで強調していることは、単なる採点ではなく、「計算途中にアドバイスを随時実行する」ということである。「ドライバが書き写しや計算間違いをしたことに気が付けば、ナビゲータは即座に指摘する」という仕組みを作るのである。

これによる利点は、「教師の労力の節約」ではなく、

・計算方法の共通化
・自分の方法・能力の客観視

が行われるということである。

近年の教育では個性や自己責任が強調されるように思う。しかし、単純な計算の方法にそれほどの個性は必要ないと思う。それより、教師が示した標準的な計算方法を早く間違いなく実行できるようになることの方が、生徒の幸福につながると著者は考えるのである。そのためにも、計算問題のペア学習は有用ではないだろうか。

（もちろん、「先生が示した方法以外は絶対だめだ」という圧政的な雰囲気を出すか「今は、先生が言った方法でやってみようね」と提案型にするかどうかは、その教師の個性に任されると思う。また、生徒が考案した計算方法が本当に標準のものより良い場合は、そのときに議論すれば良いだろう。）

さらに、実証していない予想に過ぎないが、著者は、生徒が「自分を客観視するもうひとりの自分」を創出する助けにもなるのではないかと思っている。

5．適用例

著者は担当している講義「プログラミング基礎」（C++によるプログラミング・初心者クラス）にて本小論の方法を行った。そこで提示したストーリーは以下のものである。

リリース１：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。

リリース２：あいさつし、計算するプログラムが書ける。
S４　int変数やstring変数が作れる。
S５　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。
S６　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。

リリース３：クイズが書ける。

S７　if文が使える。
S８　for文が使える。
S９　クイズを出題するコードが書ける。

リリース４：占いプログラムが書ける。
S10 乱数を表示するプログラムが書ける。
S11 乱数によって表示する内容を変えるコードが書ける。

リリース５：モンスターと遊ぶプログラムが書ける。
S12 基本的なクラスが書きそのオブジェクトを生成し使える。
S13 オブジェクトのデータを初期化できる。
S14 モンスターのクラスが書ける。

リリース６：アドベンチャーゲームが書ける。
S15 複数のクラスを組み合わせることができる。

リリース７：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。
S16 データをファイルから読み込むプログラムが書ける。
S17 vectorが使える。
S18 データの読み込みを使うプログラムが書ける。

リリース８：セーブ機能付きのアドベンチャーゲームが書ける。
S19 データをファイルに書き込むプログラムが書ける。
S20 書き込んだデータを読み込むプログラムが書ける。

リリース９：ポリモーフィズムを利用できる。
S21 クラスの派生ができる。
S22 仮想関数が定義できる。

リリース10：C++でプログラムが書ける。
S23 これまで学んだことを組み合わせられる。
S24 自由にプログラムを構想できる。

著者は、これらに関する教え込みの講義を実施し、そのあとで課題として出題（発注）した。たとえば、リリース２のS4の結果できあがるのはプログラムではなく、「int変数が作れる自分」であり、他も同様である。

実際には、すべてのリリースに成功したわけではない（授業時間が足りなくなり、後半は教え込みの時間が増えた）が、概ね成功したと考えている。

タスクは学生（受注者）に考えさせるが、そのために必要な知識は上記の教え込み授業

で与えている。そして、タスクに対して必ずタスクテストを決めさせ、実行させる。その際には、「どうテストすればよいのですか」という質問が出る。

たとえば、S4の場合、タスク自体が「（教わったことを改めて）自分で考え覚える」しかないと思う。（より具体的には「教科書の○ページを読み直す」などとなるだろう。）このように、ストーリー1つにタスク1ということはしばしば起こる。また、このタスクテストは、おそらく、「思い浮かべる」「書いてみる」くらいしかないだろうと思う。「“思い浮かべる”でもよいので、事後に必ず実行するように」と促すのが学習支援者として教師の仕事となる。

一方、S6「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」になると、より複雑である。繰り返すが、これも今「開発しているもの」はプログラムではなく、「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自分」である。学生によっては、「変数の使い方を思い出す」「入力方法を確認する」「プログラムの基本構造を書き出し確認する」などというタスクを考えるかもしれない。もちろん、このようなタスク例を参考としては示す。自分でこのような複数のタスクを切り出せる学生なら、それなりのタスクテストを考え出すこともできるだろう。しかし、単に「考える」などとする学生もいるだろう。その場合のタスクテストは単純に「書けるかどうか、やってみる」となるかもしれない。それでもかまわないのである。ただ、「テストを怠らないように」ということは注意し続けなければならないだろう。（そういう学生には、たぶん、「教科書を読む」というタスクが最初にあることを指摘してもよいだろう。）

プログラミングを学習する際のストーリーテストも、「実際にできたと自覚する」以上のテストは作りにくいし、それでよいだろうと思う。（数学、理科、社会のような分野なら、前述のように、ピンポイント的な「一問一答の問題群」が有用かもしれない。学習者が高校生以下の場合は、教師が「この問題集から自分で選びなさい」と指示しておいてもよいと思う。）

発注者が実際に望むのはリリースのみである。そのリリーステストは発注者が作り、発注者の指示のもとに行う。S4～S6はリリース2をなしていて、私が用意したリリーステストは、「こちらで用意したプログラム（10倍を100倍にするなどのマイナーチェンジ）を書いて見せてくれ」であった。実際には、学生同士で一斉に相互チェックさせた。（前節のペアプログラミングに似ているが、チェック役を隣の人に頼むテストであって、ペア学習ではない。）

これらは、「教科書を読む」「復習する」「考える」「覚える」を勧め、課題をさせているだけとも言える。すなわち、やっていること自体は、特段目新しい教育法・学習法でもない。著者が本小論で強調したいことは、学習者に「実行すべきこと」をなるべく明確に自覚させること（そのために、教師もやるべきことを明確化する）、短期間で達成感を感じさせつつ、それらを実行させるということである。

また、学習者が覚えるべきことを覚え、身に付けるべきことを身に付けたあとには、スムーズにプロジェクト解決型の学習に移行できると考えるのである。

6．ストーリーの再構築・新規構築

教育者の中には、学習したことを振り返って書く「振り返りノート」を書かせている人がいる。大変有意義で有用なことだと思う。しかし、その手間（教育者にも学習者にも）は小さくない。また、残念ながら、想定したものと違う方向に進んでしまうこともある。（学習者が毎日「数学を勉強した。難しかった」と書くだけのノートなど。）

その代替案ではないが、ストーリーを１つ終わらせるたびに、そのストーリーを言ってみさせることは、振り返りノートと同様の意味において有用だと思う。これはノートよりはるかに「軽い」という欠点があるが、「軽い」こと自体は長所ともなると思う。

ストーリーは、発注者が与えた短いもので、受注者は悩まずにそれを復唱するだけでよい。むしろ、はじめは忠実に復唱させるべきだろう。それにより、学習者が間違った方向に行く可能性を減らせるからだ。

そのような用途があるからこそ、ストーリーは短く、かつ、わかりやすいものがよいと思う。学習が進めば、「複雑なこと」もこなさなくてはならない。しかし、それも「短いストーリー」で表現される。ただその内容が、これまでこなしてきた短いストーリーの組み合わせになっているのである。著者は、このような訓練（復唱など）を一定期間した後は、学習者自身が自分でストーリーを組み替えたり、新規構築できるようになると期待する。これが著者が考える教育のゴールである。

謝辞

静岡産業大学情報学部の先生方に感謝致します。
高橋等教授には、教育工学からのご指摘もいただきました。ありがとうございます。

参考文献

[1]　Jr. Frederick P. Brooks 『人月の神話』　丸善出版　2010年（新装版）
[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　2000年
[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　2001年