ソフトウェア開発発注者育成のための形式手法を取り入れたプログラミング教育

伊藤 栄一郎
山梨学院大学
e-itoh@ygu.ac.jp

小田 朋宏
株式会社 SRA
tomohiro@sra.co.jp

荒木 啓二郎
九州大学
araki@ait.kyushu-u.ac.jp

要旨

品質の高い仕様記述は開発における手戻りを減らし最終成果物であるソフトウェアシステムの品質を高めるために重要な工程である．仕様記述は開発を計画するドメイン専門家とその実現を担当する開発者の双方が記述やレビューを通して関与すべき工程である．したがって，品質の高い仕様記述を得るためにはドメイン専門家と開発者の双方において仕様記述に関する知識と技能を高める必要がある．我々は，品質の高い仕様記述を得るための手法として形式仕様記述に着目し，将来企業においてソフトウェアシステムの発注者側として開発に携わることが予想される大学生へのプログラミング導入教育において，形式仕様のエッセンスであるプログラムの正当性，手続きの事前条件と事後条件について講義を行い，ビジュアルプログラミング環境上での演習およびペーパーテストを実施した．本論文では，講義の内容と結果，および講義向けに開発したビジュアルプログラミング環境への拡張機能を説明し，議論する．

1. はじめに

ソフトウェア開発において仕様記述をはじめとする上流工程は開発の経済性や成果物の品質を高める上で重要な工程である．専門教育においてもソフトウェア工学などを通して多くの大学においてソフトウェア開発のライフサイクルや分析，仕様記述，設計に関する技術教育がカリキュラムに組み込まれている．

近年では，PBL (Problem Based Learning) を通して，実社会で求められているソフトウェアの開発を経験することで技術だけでなくシステム開発者としての基本的な姿勢を学ぶ訓練が行われている [1]．それらの多くはソフトウェア開発者の視点に立ったものである．PBL においても学生は主に開発者として模擬プロジェクトに参加する．しかしながら，ソフトウェア開発における仕様記述工程は開発者のみが行う工程ではない．システムが解決すべき問題に対してシステムの仕様が妥当であることを確認するためには，ドメイン専門家が仕様記述を理解し適切なフィードバックを行うことが必要である [2]．したがって，ソフトウェアの仕様記述に関する基本的な知識および技術は，開発者のみならずドメイン専門家も習得することが望ましい．

仕様記述の品質を高める技術として形式仕様がある [3]．形式仕様とは，構文および意味論が厳密に定義されている記法によって，解釈に曖昧性のない仕様を記述する技術である．モデル規範型の仕様記述言語では手続きや関数の事前条件および事後条件を表明することで，手続きや関数が満たすべき仕様を高い抽象度で記述することができる．経済性や成果物の品質の観点から多くの成功事例があるが [4, 5, 6]，ソフトウェア開発全体においてはまだ適用は一部に止まっている．Hall は [7] において，形式手法に関する誤解の 6 番目に「形式手法はユーザに受け入れられない」という誤解を挙げて，ソフトウェアシステムの受託開発におけるユーザ企業側に受け入れられないという誤解から形式手法の普及が遅れていると指摘している．この誤解は現在においても改善されていない．

我々は，システム発注者のための形式仕様の基本的な技術として，事前条件および事後条件の表明に着目した．表明は，プログラミングの入門段階から導入することができ，適切な課題設定によって必ずしも高度な数学知識に依らなくても記述することが可能である．本研究では，小規模なプログラミングの演習で事前条件および事後条件を扱うことによって，形式仕様のエッセンスを学ぶこ

とができるのではないかという仮説を立てた．プログラミング導入教育を目的とした講義において，形式仕様の基本的な要素である事前条件および事後条件の表明についての知識をレクチャーし，演習として簡単な問題をプログラミング入門向けのビジュアルプログラミング環境上で表明を行う課題を与え，ペーパーテストを実施した．

第 2 節では演習のために開発した表明付きビジュアルプログラミング環境 Assertch について紹介する．第 3 節ではプログラミング導入教育で実施したレクチャーや演習，試験を説明し，その結果を示す．第 4 節では実施した結果について考察を行い，最後に第 5 節ではまとめと今後の課題を示す．

2. 表明付きビジュアルプログラミング環境 Assertch

プログラミング教育を目的として Scratch[8] や Viscuit[9] を始めとするビジュアルプログラミング環境が開発され，小学生から大学生を対象に適用されている．ビジュアルプログラミング環境は，構文を覚えるなどの学習者の負担を最小限に止めながら，プログラムを構築することのエッセンスを実践を通して学ぶことを可能にしている．

我々は，プログラミング導入教育の中でプログラムの正当性や表明の概念を導入するために，Scratch に類似したブロック型のビジュアルプログラミング環境である Phratch に表明ブロック等を追加する拡張機能として，Assertch を開発した [10]．Phratch は オープンソースとして公開されており[1]，2017 年 3 月現在も活発に開発が続けられている．

図 1 に Phratch 上で簡単なプログラムを作成した画面例を示す．画面は大きく左側，中央，右側に 3 分割されている．画面左にはプログラムを構成するための「ブロック」と呼ばれる部品がカタログとして分類されて提供されている．ユーザがカタログ上部で分類を選択すると，カタログ下部にその分類に含まれるブロックが縦方向に並べられて提供されている．ユーザはブロックをカタログ上から画面中央のスクリプトエリアにドラッグ＆ドロップによって配置する．図 1 の例では，スクリプトエリアには，いわゆる FizzBuzz 問題のプログラム（1 から 100 までの数を数え上げ，もし 15 の倍数であれば FizzBuzz あるいは 5 の倍数であれば Buzz，3 の倍数であれば Fizz，いずれでもなければその数を表示する）がブロックを使って組み立てられている．画面右側の上部では，各変数の値や，タートルグラフィックスやユーザとの対話を行うエージェントとしてロボットが表示されている．

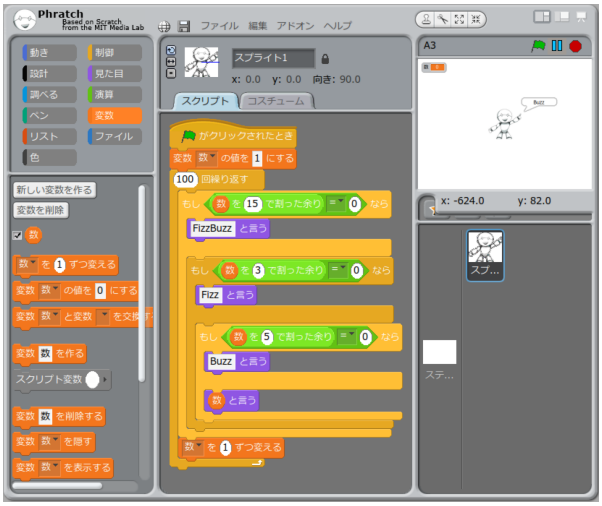

図 1. Phratch 環境の画面例

Assertch は，Phratch に表明を始めとする仕様記述に有用なブロックを追加する拡張モジュールである．現在 Phratch の標準アドオンとして提供されており，AddOn メニューから Assertch を選択することでネットワーク経由で自動的にインストールすることができる．図 2 に，表明ブロックを収めた設計カタログを示す．

「を確認するブロック」は，一般のプログラミング言語における assert 文に相当し，引数として与えられたブロックを評価実行し，true であれば実行を継続し，false であれば表明エラーとして停止するブロックである．「事前条件ブロック」は，内側にブロックを抱え込む構造を持つブロックで，内側のブロックに対する事前条件を表明するブロックである．引数として与えられたブロックを評価実行し，true であれば内部に抱えたブロックを実行し，false であれば表明エラーとして停止する．「事後条件ブロック」は，内側のブロックに対する事後条件を表明するブロックである．内部に抱えたブロックを実行した後で，引数として与えられたブロックを評価実行し，true であればプログラムの実行を継続し，false であれば表明エラーとして停止する．事前条件と事後条件の両方を表明する「事前条件事後条件ブロック」も提供されている．いずれの表明ブロックも，表明エラーが発生した

[1] http://www.phratch.com/

図 2. Assertch で拡張された表明ブロック

ブロックの色を黒色から赤色に変化させることで，どの表明が守られなかったかをユーザに示す．エラーとなった表明ブロックの色は，「表明エラーをリセットするブロック」によってリセットすることができる．

これらの表明ブロックを利用することで，プログラミング学習者であるユーザは仕様記述とプログラムの関係について以下のことを学ぶことができる．

- 試行錯誤の結果たまたま動くプログラムができるのではなく，プログラムの目的と前提条件を明確にした上で，ブロックを合理的に選択しプログラムを組み立てること
- プログラムに誤りがあった時に，各部分で期待した条件に照らし合わせて，どのブロックに問題があるかを特定すること
- プログラムを作るだけでなく，意図した通りの動作が行われるかをテストすること
- プログラムを拡張する時に，元のプログラムの設計意図に適合した改変を行うこと

これらの技術を学ぶことによって，仕様記述がプログラムの開発においてどのような役割を担っているかを体得するための環境を提供することが，Assertch の開発目的である．

3. プログラミング導入教育への適用

現在，多くの大学の様々な課程においてプログラミング導入教育が行われている．山梨学院大学経営情報学部は，経営学と情報科学をまたがる専門的な知識と技能の習得によって，新しい時代を担う情報技術者・情報管理者を育成することを目標の 1 つにしている．ソフトウェアシステムのユーザ企業側においてシステム開発に携わる人材を育成することが期待されていることから，プログラミング教育においては開発者としての視点だけでなく発注者として求められる知識や技能を学習することが，プログラミング導入教育を行う講義において求められる．

平成 28 年度後期に「情報処理論」として以下の内容の講義を実施した．

- アルゴリズムとプログラミング言語
- グラフアルゴリズム

- プログラムの正当性

うち，アルゴリズムとプログラミング言語は Phratch，プログラムの正当性は Assertch を利用して講義および演習を行った．

プログラムの正当性の演習では，二次方程式の解の 1 つを求めるプログラムを表明ブロックを使って作成し，レポートとして提出する課題を出した．受講者に対して，二次方程式 $ax^2+bx+c=0$ の解法として，$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$ の公式を示した上で，この式が多くの前提条件，すなわち事前条件を持っていることを説明し，その事前条件や事後条件を洗い出しながらプログラムを構成するように指示した．

形式仕様記述工程では，ドメイン知識や実現する機能の特性から様々な事前条件や事後条件を発見し，それらを適切な機能単位に記述する作業が行われる．これを簡単なプログラムの作成で模擬的に体験させるために，二次方程式の解法を課題として選んだ．二次方程式の解の 1 つを求めるプログラムを構成するためには，これらの事前条件および事後条件を発見し，適切に表現し，それらの事前条件および事後条件を満たすようにブロック群を合理的に構成しなければならない．

プログラム全体の事後条件は $ax^2+bx+c=0$ だが，$a=0$ の場合には分母が 0 となるので，上記公式は単純に適用できない．そこで，$a=0$ を事前条件とした副プログラムを構成する必要があるが，その場合には $a=0$ により事後条件は $bx+c=0$ となり，一次方程式の解法を使うことができる．さらに，一次方程式の解法において，$a=b=0$ の場合には事後条件が $c=0$ となるため，$c=0$ ならば任意の数が解となり，$c\neq 0$ ならば解なしとなる．また，$a\neq 0$ の場合には，判別式によって実数解があるかどうかを判定しなければならない．

実習において，受講者は以上のような事前条件および事後条件の抽出を行い，表明ブロックによってそれらを表現し，表明ブロックに内包されるブロック群を合理的に構成することで課題を遂行する必要がある．図 3 に Assertch での正答例を示す．

学期の最後に定期試験としてペーパーテストを実施した．ペーパーテストでは，アルゴリズム，プログラミングおよび正当性それぞれについての基本的な用語の理解，プログラムやアルゴリズムの実行に関する穴埋め，そして，事前条件や事後条件がプログラム開発の委託や受託にどう関係するのかを自由形式で記述する問題を出題した．

3.1. 結果

授業では全受講者 79 人のうち最終試験を受験した 72 人を評価の対象とした．そのうち 61 人が演習後のレポートを提出し，9 人は課題において表明ブロックを使いこなしていた．27 人は表明は適切でなかったが求められたプログラムを完成させた．

演習の結果とペーパーテストの結果の関係を表 1 に，演習の結果と表明に関する設問の正解/不正解による分布を表 2 に示す．演習において適切な表明を付けてプログラムを完成させた受講者は，表明がプログラムの委託および受託それぞれにおいてどのような役割を果たすかという設問に対する正答率は 0.67 であり，その他の設問，すなわちプログラミング言語およびアルゴリズムに関する設問に対する正答率は 0.77 であった．表明に不足はあったがプログラムを完成させた受講者はそれぞれ正答率が 0.63 および 0.77 と，適切な表明を付けた場合と大きな差はなかった．プログラムが未完成だった受講者は，表明と委託/受託に関する設問では正答率が 0.53 と，プログラムを完成させた受講生グループと比較して低い正答率にとどまった．その他の設問においては正答率に大きな差はなかった．レポート未提出の場合には，いずれの設問についても低い正答率になった．結果として，表明の意義に関する理解は，演習において表明そのものを適切に使いこなしていたかに関わらず，プログラムを完成させたかどうかで差がついた．

4. 議論

表明ブロックを用いた演習結果とペーパーテストでの表明に関する設問での正答率の関係について，プログラムが完成したか否かが差をつける因子であることがわかった．考えられる説明としては，表明ブロックの利用を意識して課題に取り組んだことによって，たとえレポートとして提出したプログラムでの表明が不完全であっても，表明という概念について適切な理解を得ることに繋がったと考えられる．

受講生の一部が既に一定のプログラミング技術に習熟していたことの影響も考えられるが，表 2 から，アルゴリズムやプログラミング言語に関する設問では得点が低いが表明の設問には正答できている受講者や，逆に表

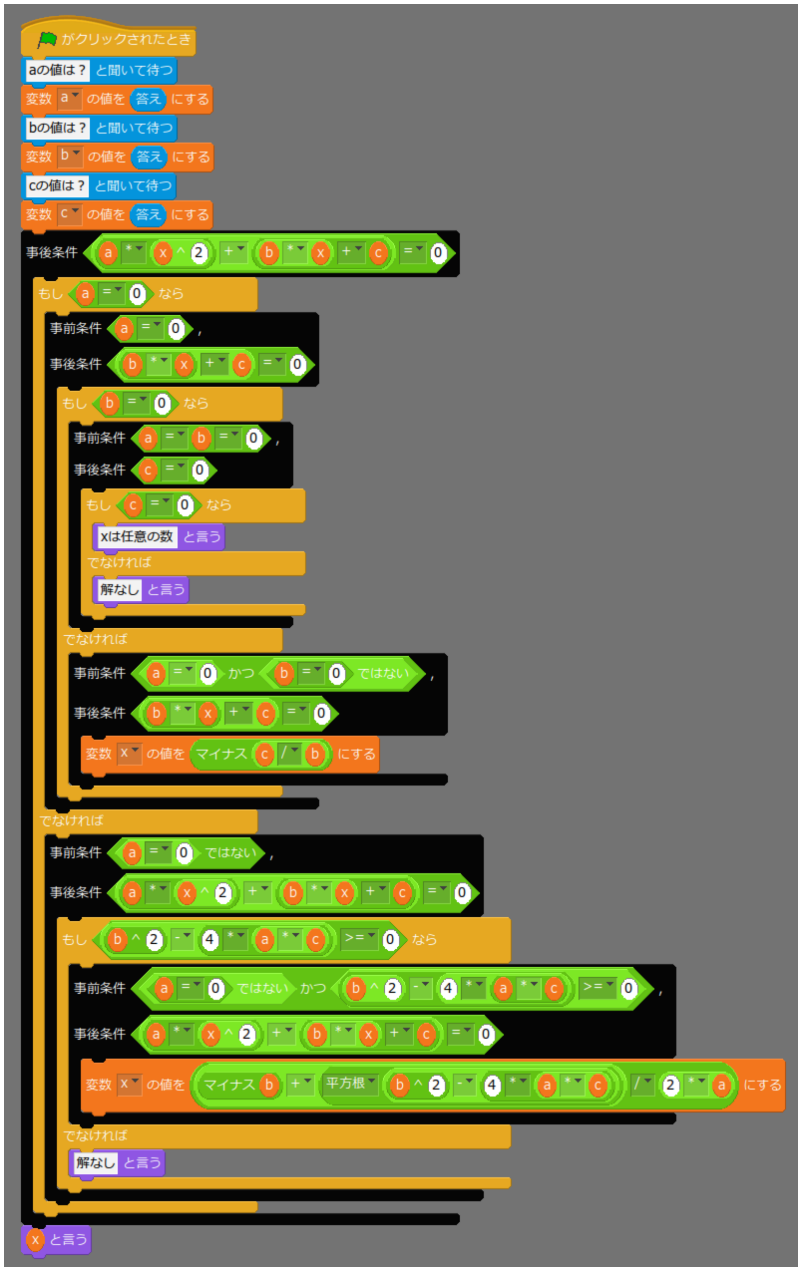

図 3. Assertch での二次方程式解法の正答例

表 1. 演習結果とペーパーテストの正答率

演習結果	「表明と委託/受託」 正答率	その他の設問の 正答率
表明付きでプログラム完成	0.67	0.77
表明なしでプログラム完成	0.63	0.77
プログラム未完成	0.41	0.71
レポート未提出	0.33	0.57

表 2. ペーパーテストにおける表明以外の設問の得点と表明設問の正答の人数分布

表明以外の得点	表明設問 正解	表明設問 不正解
-31	9	4
32-35	9	6
36-39	11	15
40-	6	12

明の設問には正答できなかったがアルゴリズムやプログラミング言語の設問では得点が高い受講生が少なくないことが示されており，プログラミングの知識や技能は表明や仕様記述に関する理解とは必ずしも直結していないと考えられる．すなわち，将来ソフトウェアシステムのユーザ企業において発注者としての役割を担う技術者を育成する上で，アルゴリズムやプログラミング言語の知識や技能だけで適性を判断するのではなく，表明に関する知識や技能によって適切な仕様記述を行うための教育が必要である可能性を示唆している．

演習に用いたプログラミング環境については，多くの改良の余地があることがわかった．特に，演習において表明に記述する式が複雑になり，受講生の時間や労力の多くが式をブロックで組み立てることに費やされてしまっていた．ブロック式のビジュアルプログラミング環境であることの欠点であり，条件式をテキスト形式で入力可能なブロックを導入するなどして解決する必要がある．

5. おわりに

ソフトウェアシステムを開発するための技術者の不足が叫ばれて久しい．情報システム開発の要求およびその複雑さは，これからも増えていくことが予想されている．そこで求められる生産性と品質を獲得するためには，品質の高い仕様を記述することができる技術者が必要である．従来のソフトウェア開発を受託する技術者の育成により開発者の数を増やすだけでなく，仕様を策定するユーザ企業側においても，品質の高い仕様を記述する知識と技能を持った技術者を確保することが求められる．

本研究では，将来ユーザ企業またはソフトウェア開発企業において技術者として開発に携わることが期待される受講者に，プログラミング教育向けのビジュアルプログラミング環境に表明機能を追加することで，正しいプログラムについて学ぶ機会を提供することを試みた．演習の中で，プログラムが期待した動作をしなかった時に，プログラムの誤りなのか仕様の誤りなのかを考え修正する受講生の姿を観察した．

「情報処理論」にプログラムの正当性および表明に関する講義や演習を取り入れて 2 期目が経過した．今後もプログラミング環境の改善を行いつつ，教材や演習内容の改善に務める．

6. 謝辞

本研究を遂行するにあたって Jannik Laval 氏，阿部和広氏，中小路久美代氏，山本恭裕氏より多くの助言を受けた．ここに謝意を記す．有益なご指摘を頂いた査読者の方々に感謝する．本研究の一部は, JSPS 科研費 (26330099) の助成を受けたものである．

参考文献

[1] 井上明, 金田重郎. “実システム開発を通じた社会連携型 PBL の提案と評価.” **情報処理学会論文誌**, Vol. 49, No. 2, pp. 930-943, 2008.

[2] Oda, T., Yamomoto. Y., Nakakoji, K., Araki, K., and Larsen, P. G. “VDM Animation for a Wider Range of Stakeholders”. In Proceedings of the 13th Overture Workshop, pp 18-32, 2015.

[3] Fitzgerald. J. and Larsen, P.G. “Modelling Systems – Practical Tools and Techniques in Software Development, Cambridge University Press, 1998.

[4] Fitzgerald. J, Larsen, P. G., and Sahara, S. “VDM-Tools: Advances in Support for Formal Modeling in VDM”, *ACM Sigplan Notices*, Vol. 43, No. 2, pp. 3-11, 2008.

[5] Kurita, T., and Nakatsugawa, Y. “The Application of VDM++ to the Development of Firmware for a Smart Card IC Chip, *Intl. Journal of Software and Informatics*, Vol. 3, No. 2-3, pp. 343-355, 2009.

[6] IPA/SEC. “厳密な仕様記述における形式手法成功事例調査報告書”, 独立行政法人情報処理推進機構 技

術本部 ソフトウェア・エンジニアリング・センター, 2013.

[7] Hall, A. “Seven myths of formal methods.” *IEEE software*, Vol. 7, No. 5, pp. 11-19, 1990.

[8] Maloney, J., et al. “The scratch programming language and environment.” *ACM Transactions on Computing Education (TOCE)*, Vol. 10, No. 4: 16, 2010.

[9] Harada, Y., and Potter, R. “Fuzzy Rewriting.” *End User Development*, Springer Netherlands, pp. 251-267, 2006.

[10] Oda, T., Araki, K., and Larsen, P. G. “ViennaTalk and assertch: building lightweight formal methods environments on pharo 4.” in Proceedings of the 11th edition of the International Workshop on Smalltalk Technologies (IWST16), pp. 4:1-4:7, 2016.