ソフトウェア工学 62－6
（1988. 9. 28）

ソフトウェア工学の教育と実践

新谷　勝利
日本アイ・ビー・エム（株）ソフトウェア開発センター

要約

主として製品としてのソフトウェアを開発しているIBMのソフトウェア開発部門で実施されているソフトウェア工学教育とその実践について報告する。全部門共通の教育として〝ソフトウェア工学演習〟という2週間のクラスが提供されている。教室で学んだことをスムースに実際のプロジェクトに適用するために、プロジェクトへの参加を通して得たものをベースに、筆者は新4段階設計法を開発した。これはD. O'NeillとJ. Rymerの4段階設計法をベースにして、前述クラスで説明する設計方法論、ツール、および当社ソフトウェア開発センターの開発工程を統合したものである。本稿ではこれらの背景を述べるとともに、実践アプローチを紹介する。更に、今後への展望も述べる。

Software Engineering Education and its Implementation

Katsutoshi Shintani
Tokyo Programming Center, IBM Japan, Ltd.
1-14 Nisshin-cho, Kawasaki-ku, Kawasaki-shi 210 Japan

Abstract

This paper explains a 2-week software engineering education known as Software Engineering Workshop tought throughout IBM programming centers responsible for software products to be marketted. And also explained are ways to implement what has been taught in real projects. The author, based on his involvement in pilot projects, has developed the new 4-Level Design Methód. This metod is, based on D. O'Neill & J. Rymer's 4-Level Design Method, to integrate design methodologies taught in a class, tools, and the development process at Tokyo Programming Center, IBM Japan. This paper also explains the background of this new method and briefs possible future extention.

はじめに

B.W. Boehmの最近の報告 1)に依れば、生産性は次のように定義されている。

$$\text{生産性} = \frac{\text{プロセスからの産出量}}{\text{プロセスへの労働投入量}}$$

上式によれば、生産性は以下によって向上される。

- プロセスへの労働投入量を同じにして、産出量を増加させる。
- プロセスからの産出量を同じにして、労働投入量を減少させる。
- 産出量を増加させるとともに、労働投入量を減少させる。

産出量の増加は一般的には効率を上げるという観点から述べることができ、それには仕様およびコードの再利用、アプリケーション・ジェネレーターの使用等が対象になるものと考えられる。
労働投入量の減少は一般的には開発工程の品質を上げるという観点から述べることができ、それには前工程における不正確あるいは曖昧な記述による後工程における再作業を無くす、チームにおけるミス・コミュニケーションを発生させない等が対象になるものと考えらえる。

日本アイ・ビー・エム株式会社においては、仕様およびコードの再利用が可能となる設計手法をとりながら正しさが検証できるとともに正確な仕様書と設計書が作成できる記述法の教育が実施されている。視点は、主として開発工程の品質を上げるということができるであろう。開発工程におけるワークロードの配布については、 Boehmの上記報告によれば、開発作業の内、オペレーショナルな作業は 80%であり、その内訳は要求定義に 5%、設計に 27%、コーディングおよびユニット・テストに 16%、テストに 20%投入されている。更にそのうち 30%が再作業である。よって、開発の初期段階である要求定義および設計の工程に対する方法論に立ち入ることは当を得ていると考えられる。

本稿においては、どのような設計アプローチを設計者にソフトウェア工学の教育を通して取得してもらっているかを述べる。要求定義については、「今後への展望」において簡単にふれる。

ソフトウェア工学教育とその適用

当社は 1960年代後半から Systems Reserch Institute において Constantine等によるソフトウェア工学の教育を実施しており、実践的なレポート 2 も多く刊行している。 1980年初頭から全社的なレベルでソフトウェア開発部門に対して、ソフトウェア工学の 1970年代以降の成果を取り込み 1990年代以降を見越した組織化、教育、実践が行なわれており、その状況は 1985年に IBM Systems Journalの特集号 4)として発表された。

参考文献 3)において説明されている当社米国政府および民間大規模システム開発担当部門の成果 を全社的なものとするために 1981年に Software Engineering Instituteが設立された。世界中の当社プログラミング・センター員に同一内容の教育をレベルを保ちながら実施するために、２週間のクラスを担当するインストラクターの育成教育が巾広いソフトウェア工学の学習も含め１年近くかけて実施されてきた。現在、日本における２人を含め 50人以上の教育担当有資格者が各国にいる。

これらインストラクターは単に教育するだけでなく、実際のプロジェクトにコンサルタントとして参画すると共に経験を他のプロジェクトあるいは教育に生かす活動をしている。本稿はそのような活動を通して得られた経験をベースにしている。ソフトウェア工学の実際プロジェクトへの適用は、その方法論をサポートする新しい開発プロセスと新しいツール・セットとが組み合わされて実施されている。新しい開発プロセスとツール・セットについては、参考文献 4)において説明されている。参考文献 4)において、 M.B. CarpenterとH.K. Hallmanが当社におけるソフトウェア工学教育の入門コースであるソフトウェア工学演習、（以降 SEWとよぶ）、について説明している。当センターにおいては 1984年末よりSEWの教育が実施され、現在、ほぼ全員が受講を終了している。当クラスの目的は、今までのゼロ・ディフェクトを目指す方法論がインスペクションとテストであったものを、更に規律のある設計をソフトウェア工学の成果を取り込むことにより実施できるようにすることである。この目的を達成するためにSEWにおいては、次の項目が練習問題、ケース・スタディ、ツールの実習等を通して説明される。

- 抽象データ型によるコンポーネント記述
- 機能モデルによるプロシージャーの抽象化記述
- 段階的詳細化
- 種々の検証法の適用
- 設計用言語によるシステム、コンポーネント、プロシージャー、データの記述

たとえ練習問題、ケース・スタディ、ツールの実習を２週間に渡ってクラスで実施したとしても、それだけでは充分ではなく、インストラクターによる実際プロジェクトにおけるサポートが必須である。特にいかにしてコンポーネントを抽象データ型で記述するかは、オブジェクトはいかにして識別されるかとあいまって、実践を通した訓練が必要である。

当センターにおける学習曲線として次のものが報告されている。

- クラス前の前提学習　１週間
- クラス出席　２週間
- クラス後の復習　１週間
- 実際に自分のプロジェクトの中から適当なものを選び適用　１週間

数百万ラインにのぼる大規模プロジェクトに適用した米国の例では、数百人のプロジェクト要員の平均値として次の学習曲線が報告されている。

- プロシージャー部分への設計用言語の適用
  １～２ケ月
- 抽象データ型の適用
  ３～６ケ月
- システム全体の設計用言語による記述
  ６ケ月以上

上述報告において、段階的詳細化への適用は、書き方に対する慣れの問題であり、最も思考を必要とするのは、データと機能の分割を伴う抽象データ型の適用であるとしている。学習曲線に対する個人差、グループ・ダイナミックスに関する配慮を管理者は充分しなければならないことになる。

当センターにおける実際プロジェクト適用時には、復習、学習曲線を考え、次の点に留意している。

- クラス出席とプロジェクト開始の期間がメンバー間にバラツキがありうるので、復習と適用時の考慮事項、例を説明するプロジェクト単位のワークショップを３日間実施する。
- プロジェクトの初期に、インストラクターはプロジェクト側に机を得、常に彼等の側にいる。
- 設計用言語、方法論について３分以上自分のみで悩まないという３分間ルールを適用し、インストラクターがサポートする。
- 形式化を設計段階に応じて進められるようにする。

最後の項目のための方法論として前述二報告の二番目のプロジェクトにおいて４段階設計法 5)というものが、考案された。

次にこの４段階設計法について説明する。

４段階設計法

４段階設計法は各レベルを次の様に定義している。

- レベル１

  ユーザーとの契約事項を記述する。一般的に外部仕様と称されているものである。ステート・マシン図に依る記述を行い、抽象データ型によるデータと機能の初期分割と要求事項の結合が目的とされる。

- レベル２

  分割された機能単位を１つの抽象データ型ととらえ、コンポーネントの記述をする。当レベルにおいては、コンポーネント内の機能の抽象化記述が主目的で、いわゆる、 WHAT記述に徹することが求められる。 WHAT記述における主要項目は、詳細な入力、出力、記憶データの記述と機能の論理演算記述である。

- レベル３

  コンポーネント内の各機能に対して、構造化プログラミングの段階的詳細化に基づく記述が行われる。当該開発部門では、設計レベルの機能部品化と再利用が推進されており、当レベルの記述はハードウェア、ソフトウェアに依存しないロジックの展開とするよう要求されている。

- レベル４

レベル３の設計書を対象システム用に変換する。システムの諸要求、例えばパフォーマンス、メモリー・サイズ、ファイル構成等、に関する考慮は各レベルにおいても考慮されるが、主として、このレベルで解決される。

前述二番目報告例の数百万ステップの大規模プロジェクトにおいて、次のデータが報告されている。当プロジェクトは単に規模が大きいだけでなく、極めて複雑なシステムである。全米４ケ所のプログラム開発センターで、当社と協力会社２社とによって数年に渡って開発されたものである。比較の対象になったデータは当該開発部門の新手法を使用しないで開発されたものの平均値である。

- ソースコード 1,000行当りバグ　50%減少
- 月当り出荷量　25%向上
- 人月当り生産性　15%向上

実践アプローチ

筆者は、 1985年に４段階設計法開発者の一人であるJohn Rymer氏と SEWをどのように実践すべきかについて議論した結果、 SEWと当センターの設計工程および使用している設計用言語に合致するように修正作業を進めた。修正における基本的な考え方としては、従来の機能を中心とした段階的詳細化をデータ抽象型を中心とした段階的詳細化にすることである。本文では、当修正をしたものを新４段階設計法と称し後述する。 1986年の１年間パイロット・プロジェクトに適用を試み 1987年には更に別のプロジェクトにも適用した。当センター向けに修正するに当り、前述のソフトウェア工学教育の実際面の適用ということを最重点に置きツールの積極的な活用に留意した。これら２プロジェクトはパイロットであり、生産性の向上ということは必ずしも得られなかったが、品質の向上に関しては、成果は著しいものが得られた。最初のプロジェクトでは、当該製品のベース・モデルに依る予測値の半分以下のディフェクト率になり、２番目のプロジェクトにおいては、機能テスト以降、エラーは統計上無視できる程度にしか発生しなかった。

当センターにおける設計工程は参考文献 4)のプロセスに準拠して定義されており、次の如くである。

ソフトウェア開発センターの設計工程

- 製品レベル設計
  要求定義工程の出力をベースにし、次の２設計書を作成する。
  - 初期プログラム機能仕様書
    要求定義工程の出力を機能別に分割すると共にどのような機能を使用者に提供するかを概説する。
  - システム構造図
    分割された機能間の関係を階層構造図としてまとめたものである。当図は、設計が進むにつれて更新される。多くの場合、当図は製品レベル設計においては１段階、多くても２段階である。

- コンポーネント・レベル設計
  製品レベル設計の出力から出発し、システム構造図の更新に加えて、次の２設計書を作成する。
  - 最終プログラム機能仕様書
    製品の最終外部仕様書であり、当仕様書がユーザー・マニュアルのベースとなる。仕様変更はソフトウェア開発にはつきものであり、設計変更要求書は最終プログラム機能仕様書に対して出されるものである。
  - プログラム・ロジック仕様書
    各コンポーネント、モジュール、内部機能、インターフェース等を記述するものである。多くの場合、自然語で記述されてきた。

- モジュール・レベル設計
  コンポーネント・レベル設計の出力から出発し、各モジュールについて最終プログラム機能仕様書を満足するように内部ロジックを記述する。特別新しい設計書は作成されず、システム構造図とプログラム・ロジック仕様書の更新が行なわれる。多くの場合、擬似コードで記述されてきた。

前述工程は枠組みが定義されているものであり、いわばプログラム開発におけるプログラム機能仕様書の如きもので、筆者は４段階設計法を修正するに当たり、上記枠組みは可能なかぎり温存した。

新４段階設計法

- レベル０

  データ・フロー・ダイヤグラムあるいは任意の図表現に依りシステムの全体像を画く。通常この作業はチーム・リーダーか彼を含め限定されたプロジェクト・メンバーのディスカッションによって行なわれる。全体のイメージがある程度かたまったところで、全メンバーに対して説明され、全員が全体のイメージを共通なものとして持つことが要求される。

  このレベルは当センターの設計工程の内、製品レベル設計の初期プログラム機能仕様書を書く部分に相当する。

  設計者は、通常、対象を見る時、まず個々のケースについて考える。次いで一般化する。個々のケースというものは、名詞で表現される。この名詞はオブジェクトと言われるものである。よって、システムはオブジェクトの集合として先ず把握される。オブジェクト指向設計法が適用できる可能性がここにある。一般化するというのは、いくつかのオブジェクトをグループ化して、共通的なカタマリ、すなわち、型、としてまとめることである。よって、コンポーネントとしてまとめる時に型としての認識になるものと考えられる。抽象データ型の考え方がこの時点で使用できるのは、これに依る。この１つが次のレベル１の出発点になる。

- レベル１

  このレベルは基本的に４段階設計法のレベル１に相当する。抽象データ型は図表現としてはステート・マシン図が使用され、対応する設計用言語で記述することにより機械処理が可能となる。この段階で留意しなければならないのは次の点である。

  - 設計用言語による機械処理に主点は置かず、自然語を用いてもよいから、形式化された記述の出発点になるようにする。
  - 名前の付け方について明確な定義をし、ディクショナリーの準備をする。設計段階で使用する名前とインプリメンテーションで使用する名前を別のものにする愚はおかしてはならない。システムによる制約は、多くの場合、コンパイラーのマクロ機能でバイパスできるが、もし、使用コンパイラーにその機能がない場合は、この時点で方法を考えておく必要がある。

  このレベルは当センターの設計工程の内、製品レベル設計のシステム構造図を作る部分に相当する。

  当レベルでコンポーネントは殆ど定義される。よって、次レベルは各コンポーネントに関する設計に移る。

- レベル２

  レベル１において、設計用言語の形式化された記述の出発点が設計されたので、内部ロジックに可能な限り踏み込まないで、コンポーネント内の機能を次のレベルでインターフェースのみ守っていれば独立して設計作業が進められるように記述する。レベル１で書かれた自然語あるいはそれに近い記述を、設計用言語を用いながら段階的詳細化を図る。このレベルでは、最終的なモジュールがその機能、インターフェース記述と共に認識されなければならない。レベル０からレベル２まで、単純にトップ・ダウン設計ができるわけではなく、必らず前工程とすりあわせをし、必要な修正を前工程の設計書にしておくことにより統一性のある設計が可能になる。このことは他のレベルでもいえることであるが、特に当レベルまでで重要である。

  このレベルは当センターの設計工程の内、コンポーネント・レベル設計に相当する。

  当レベル以降は各モジュールに分かれて設計されるので、チーム全体での共通理解が必要となる。最終プログラム機能仕様書により外部仕様書をかため、プログラム・ロジック仕様書で次工程以降を各メンバーが独立して作業できるようにモジュール名、各モジュールが果たすべき機能、インターフェースがかためられなければならない。

- レベル３とレベル４を合体

  ソフトウェア製品をつくる場合、対象のシステム環境は予め決められているので、４段階設計

法のレベル３とレベル４は、一般的に、コンパインされるか、レベル３をスキップした形でレベル４が実施される。このレベルの主要な方法は構造化プログラミングであり、特に、段階的詳細化である。擬似コードの代わりに設計用言語による機械処理可能な形式化された記述がなされる。ディクショナリーは、自動的に維持される。機械処理により、シンタックス・チェックのみならず、セマンティックス・チェックがなされる。このレベルの留意点は、設計用言語に依るコーディングにしないことである。コーディングとユニット・テストをインプリメンテーションと称しているが、これはモジュール設計の次の段階である。モジュール設計は、あくまでも設計であって、インプリメンテーションではない。

このレベルは当センターの設計工程の内、モジュール・レベル設計に相当する。

このレベルの記述はツールに依ってソース・コード生成に使用される。設計用言語の記述にも依るが、 80%以上のソース・コードは生成されている。

従来型と新４段階設計法の対比

従来型	新４段階設計法
・製品レベル設計 - 初期プログラム機能仕様書 - システム構造図	・レベル０ - システム全体像の把握 - システムの外部と内部を区別 ・レベル１ - 内部設計の観点からシステム全体像を複数の抽象データ型で記述
・コンポーネント・レベル設計 - 最終プログラム機能仕様書 - システム構造図 - プログラム・ロジック仕様書	・レベル２ - 外部要件を最終的に確定する - 内部構造を最終的に確定する （プロシージャー以外のbody）
・モジュール・レベル設計 - システム構造図 - プログラム・ロジック仕様書	・レベル３と４合体 - 各プロシージャーのbodyを段階的に詳細化

注）本来ならば、設計文書類も新しいアプローチによりその内容等が変更されるのが望ましいが、社内標準あるいは他のプロジェクトとのからみもあり全面的に改訂されるに至っていない。抽象データ型の段階的抽象化をサポートする新しい設計文書類が新４段階設計法を採用したプロジェクトではプログラム・ロジック仕様書の代わりに使用された。

今後への展望

Boehmの Cocomoモデル以降の研究でも明らかにされているようにソフトウェア開発における諸問題の解決には、人の能力をどのように引き出し活用するかが重要であると考える。最近の傾向であるソフトウェアの大きさと複雑さの増加は、一人の人間のみでソフトウェアを作ることを不可能にしているし、人間が作るソフトウェアであれば、プロジェクト・メンバー全員のレベルを上げることを主眼としなければならない。ディフェクトは一番弱いところに現れるからである。 Boehmの品質および生産性に関する小文 6)は極めて示唆に富んでいる。ハードウェアの製造工程を類似なものとして、ソフトウェア開発においてその開発工程を重視したとしても、ソフトウェア開発においては単純に既存の部品を組み立てればよいというものではなく、自動化を目的としてツールを種々導入したとしても、ツールに合うように固定された形をもっているわけでもない。更にソフトウェア開発担当者はモダン・タイムズのチャップリン以上に自らを埋没させることには強い抵抗を示す。だからといって、勝手に自分の思うようにすることでは統一性のあるシステムはできないであろ

うし、それ以上に開発コストと同額かそれ以上といわれる保守コストは更に増加するであろう。ソフトウェア工学をベースにする規律あるソフトウェア開発が要求されるゆえんである。

SEWの教育を通して寄せられた多くのコメントの中から特筆すべきものとしては次のようなものがある。

- 考え方は良くわかったし、そのメリットも納得できた。しかし、実際に適用するには初期学習曲線が気になる。現実のプロジェクトでは、コスト的にもスケジュール的にもそのような余裕がない。
- 既存の開発プロセスの中に新しい考え方を取り込むには多くの試行錯誤が必要であろう。自分にはその余裕がない。
- リリース・アップ製品を担当しており、既存の製品は新しい手法で開発されたものではなく、適用が困難である。

もちろん、自分のプロジェクトに取り込みたいとコメントする人は多いわけで、適用を広めてゆくには、上記３大阻害点を取り除くことが重要となる。最初のコメントは、考えようによっては、阻害点というより推進力になるとものと考える。何故なら、多くのプロジェクトはテストに余裕を持たせていることが多いので、いくつかのパイロットあるいは他の事例を通して、テスト時間のセーブ分が初期学習曲線より大きいこと、テストにおけるコストは設計時のコストより大きいことを実際に見せることである程度心配を柔らげることができるであろう。２番目のコメントが筆者をして新４段階設計法を開発させた理由である。実際の適用例を説明している最近のクラスでは、この面に関するコメントは殆ど出てきていない。第３のコメントに関しては、今後のパイロットで考慮しなければならない点である。最初のパイロットはリリース・アップに相当するものであったが、他プロジェクトにおいては、＂自分のプロジェクトは他とは異なる＂というメンタリティもあり、今後、種々のリリース・アップ・パターンで試行する必要があろう。

今までSEW手法を適用してきた１０にのぼるプロジェクトに２回目の適用が含まれていることもあり、次のプロジェクトにおいても適用すると管理者およびリーダーが言っていることは今後の適用に明かるい展望を与えてくれている。

人がからむが故に、前述の米国例のプロジェクト・マネージャーは次のように言っている。

＂ＣＲＡＷＬ－ＷＡＬＫ－ＲＵＮ＂

次の事項のバランスを取りながら、長期的視野に立ち、常にソフトウェア開発において高品質と高生産性を追及してゆく必要があるであろう。

- 教育とそのフォロー
- プロセスの改善
- ツールの導入

更に、要求定義からインプリメンテーションまで連続したプロセス、（最近シームレス・ソフトウェア開発と言われているようであるが）、を考えるならば、要求定義からレベル０への移行において、オブジェクトを認識できることが必須と考えられる。筆者はこの点に関し、最近のMills等の報告 7)8)に注目している。

謝辞

新４段階設計法を実際のプロジェクトに適用するに当りその機会を与えて下さった当センターの大藤担当、舟橋担当そして当時プロジェクト・リーダーであった大嶋担当、更に関係プロジェクト・メンバーのみなさんに感謝します。１年に渡る筆者のプロジェクト参加中に修正され現行のものになったのである。どんな新しい方法論も使用され修正されてこそ実践的なものになると考える。

参考文献

1. B.W. Boehn, Improving Software Productivity, IEEE Computer, V.20, N.9, pp.43-57
2. IBM Systems Journal,V.15,N.3,1976,G321-0046
   - M.E. Fagan, Design and Code Inspetions to reduce errors in program development
   - G.J. Myers, Composite design facilities of six programming languages
   - L.A. Belady and M.M. Lehman, A model of large program developpent
3. IBM Systems Journal, V.19, N.4, 1980, G321-0063
   Software Developpent特集号
   - The management of software engineering

  - H.D. Mills, Part I: Principles of software engineering
  - D. O'Neill, Part II: Software engineering program
  - R.C. Linger, Part III: Software design practices
  - M. Dyer, Part IV: Software development practices
  - R.E. Quinnan, Part V: Software engineering management practices
- D.J. Mishelevich and D. Van Slyke, Application development system: The software architecture of the IBM Health Care Support/DL/I -Patient Care System
- S. Katz, L.I. Risman, and M. Rodeh, A system for constructing linear programming models
- R.D. Gordon, The Modular Application Customizing System
- L.A. Belady, C.J.Evangelist, and L.R. Power, GREENPRINT: A graphic representation of structured programs

4. IBM Systems Journal, V.24, N.2, 1985, G321-0080
   - W.S. Humphrey, The IBM large-systems software development process: objectives and direction
   - R.A. Radice, N.K. Roth, A.C. O'Hara,Jr, and W.A. Ciarfella, A programming process architecture
   - R.A. Radice, J.T. Harding, P.E. Munnis, and R.W. Phillps, A programming process study
   - G.F. Hoffnagle and W.E. Beregi, Automating the software development process
   - M.B. Carpenter and H.K. Hallman, Quality emphasis at IBM's Software Engineering Institute
   - R.G. Mays, L.S. Orzech, N.A. Ciarfella, and R.W. Phillips, PDM: A requirements methodology for software system enhancements
   - C.L. Jones, A process-integrated approach to defect prevention
   - M.J. Flaherty, Programming process productivity measurement system for System/370
5. D. O'Neill and J. Rymer, Software engineering and Ada* in design, IBM, FSD, 1986年改訂版
   *米国政府 Ada JPOトレードマーク
6. B. Boehm, Industrial software metrics top 10 list, IEEE Software, Sept.'87, pp.84-85
7. H.D. Mills, R.C. Linger, A.R. Hevner, Box structured information systems, IBM Systems Journal, Vol.26, No.4, 1987, pp.395-413
8. H.D. Mills, R.C. Linger, A. Hevner, Principles of Information Systems Analysis and Design, Academic Press, Inc, New York (1986)

(付)ステート・マシーン図

抽象データ型を記述するものとしてＨＩＰＯ図を拡張したステート・マシーン図が開発されている。