社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2009－SE－163 (21)
2009／3／19

測定可能な個人プロセスを対象とした形式手法導入に関する提案

小材健 * 日下部茂 † 大森洋一 † 荒木啓二郎 †
* 九州大学大学院システム情報科学府
† 九州大学大学院システム情報科学研究院

概説

形式手法導入の成功事例報告は既にあるものの，その普及にはまだ次のような課題があると指摘されている：導入効果の可視化・定量化や導入コストの定量化，開発方法論の確立，導入に向けての人材育成等の課題である．本論文ではこのような課題に対処するために，測定可能な個人プロセスのデータを欠陥の型に着目して分析を行い，効果的に形式手法を導入してプロセス改善をする方法の提案と事例報告を行う．提案した方法で形式手法を導入した結果，生産性の低下を引き起こさずに，着目した欠陥の型を上流工程で発見・除去することができた．

Introducing formal methods into measurable personal software development process

Takeshi Kozai * , Shigeru Kusakabe †, Yoichi Omori†, Keijiro Araki †,
*Grad.School of Information Science and Electrical Engineering, Kyushu University
†Dept. of Computer Science and Communication Engineering, Kyushu University

Abstract

There are several reports of system development in which they could effectively use formal methods. However, there are several obstacles in introducing formal methods into actual software development such as visualization or quantification of the effectiveness and cost of introducing formal methods, establishment of development methodology, and training before and during introduction. We propose a method of effectively introducing formal methods into measurable personal software development process by analyzing process data on defect type. According to the result of our case study, we conclude that we can find and remove the defects we focused in upstream process, without degrading the productivity.

1 はじめに

ソフトウェアの利用範囲が広がり，重要な社会基盤を支えるシステムとしても利用されるようになっている．そのため，ソフトウェアに不具合が生じた際，社会に与える影響は大きい．一方でソフトウェアの大規模化・複雑化が進み，従来のテスティングを主体としたソフトウェアの検査ではソフトウェアの不具合をリリース前に発見することが難しくなってきている．

ソフトウェアの信頼性や安全性をテスティングのみに依存せず，いかに保証するかということはソフトウェアを開発する上で重要な課題である．この課題を解決するための技術として形式手法（Formal Methods）が注目を集めている．形式手法とは，開発対象システムを数学的基盤に基づいた形式仕様記述言語で記述することにより，システムを厳密に設計し，システムが正しく動作することを検証する技術の総称である．これによりシステムの仕様記述のあいまいさを排除するとともにシステムが満たすべき性質の厳密な検証が可能になると期待されている．

形式手法を実際のソフトウェア開発に適用し成果を上げた事例としてはオランダ・チェス社の例 [1] やフェリカネットワークス株式会社の例 [2] などがある．一方で，形式手法はその有用性が言われながらも定量的な効果があいまいであり具体的な導入方法も確立されているとはいえないため，高い信頼性が求められるインフラシステムや組み込みシステム以外での導入はあまり進んでいない．その理由として導入効果の可視化・定量化や導入コストの定量化，開発方法論の確立，導入に向けての人材育成といった課題がある [3]．

形式手法のより広い普及を進めるためには実際のプロセスにおける具体的な導入方法を示すとともに，効果を定量的に評価していくことが必要である．

本論文では計測可能な個人プロセスに対して形式手法の一つである VDM を導入する場合を取り扱う．具体的には，VDM で用いられる形式仕様記述言語である VDM++[4][5] を用いた設計記述を個人の開発プロセスに導入する方法を提案する．事例研究を通して提案した導入方法のプロセス改善効果を評価し，VDM の導入における効果と導入コストを明らかにする．

2 実施方法

2.1 プロセスデータの計測方法

本論文ではプロセスを計測するために Personal Software Process[1]（以下 PSP と表記する）[6] を利用した．PSP は個人向けのプロセス改善支援プロセスである．PSP で利用するプロセスの計測と分析のための作業手順スクリプト，データ記入フォームおよび PSP を習得するための演習課題キット，サポートツールが CMU/SEI[2]により提供されている．本論文でもデータの収集・管理を CMU/SEI が提供するサポートツールにより行った．

PSP のスクリプト中で要求される作業内容は図 1 に示すように段階的に追加され，プロセス測定から計測，品質管理までをカバーしている．本論文で行った事例においては PSP0 および PSP0.1 で導入されるプロセスの規律と計測が重要であるのでプロセスのデータとして収集するデータについてのみここで説明をおこなう．

プロセスデータとして計測するデータはフェーズごとの作業時間ログおよび欠陥ログである．作業時間ログについては各フェーズの作業時間と作業の中断時間を記録する．欠陥ログについては欠陥の型および欠陥が混入したフェーズ，欠陥が除去されたフェーズ，修正にかかった時間，欠陥の説明を記録する．

[1]Personal Software Process および PSP は Canegie Mellon University のサービスマークです

[2]http://www.sei.cmu.edu/tsp/

[3]DLDR および CR フェーズは PSP スクリプト 2.0 以降から導入される

表 2: PSP 演習課題の概要

課題番号	課題内容	使用スクリプト
1	LinkedList	PSP 0
2	プログラムの規模（LOC）カウンタ	PSP 0.1
3	PROBE 見積もり値の計算	PSP 1.0
4	相対プログラム規模の計算	PSP 1.1
5	数値積分	PSP 2.0
6	確率密度関数 P(x) の積分値が p になる x の値の計算	PSP 2.1

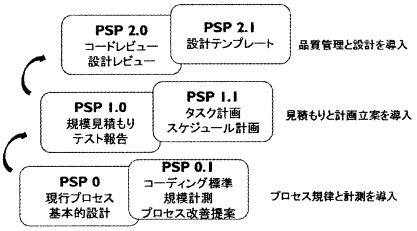

図 1: PSP の段階とその内容の構成

表 1: PSP の作業内容[3]

	フェーズ名	作業内容
上流	PLAN	計画立案
	DLD	設計
	DLDR	設計レビュー
⋮	CODE	コーディング
	CR	コードレビュー
	UT	単体テスト
下流	PM	事後分析

2.2 実施概要

本論文では個人プロセスのデータを収集するにあたり，PSP のスキル習得用演習キットの課題を測定の対象として採用した．各課題と作成するプログラムの対応を表 2 に示した．測定を行ったのは第一著者本人のプロセスデータのみである．開発環境には Eclipse を用い，プログラミング言語には Java を使用した．以下に VDM 導入前後のプロセスデータを測定した手順を示す．

1. 演習キットの課題 1 から課題 4 を実施し，VDM 導入前のプロセスデータを測定する．
2. 測定したプロセスデータを分析し，VDM の導入方法を提案する．
3. 演習キットの課題 5 と課題 6 を 2 で提案した VDM の導入方法の下で実施し，プロセスデータを測定する．
4. VDM を導入する前後でプロセスデータを比較・分析し，提案した VDM の導入方法の考察を行う．

3 既存プロセスデータの分析

3.1 既存プロセスのデータ

課題 1 から課題 4 のプロセスデータについて，表 3 の作業時間ログおよび表 5 の欠陥ログを得た．

3.2 欠陥データの分析方法

VDM++による設計記述導入前の既存プロセスのデータの分析手順を示す．

1. 混入された欠陥のうち修正に時間がかかっている，もしくは件数の多い欠陥型に着目する．
2. 着目した欠陥型がどのフェーズで主に混入しているかを調べる．

表 3: VDM 導入前におけるフェーズあたりの作業時間の割合 (%)

課題番号	PLAN	DLD	CODE	UT	PM	開発時間 (分)	規模 (LOC)
1	27.6	7.6	35.9	24.1	4.8	270	111
2	18.6	9.0	54.9	11.3	6.2	812	243
3	25.3	14.0	29.2	24.9	6.6	535	225
4	20.6	14.6	39.6	13.7	11.5	321	154
Total	22.0	11.1	42.6	17.3	7.0	1938	733

表 4: VDM 導入後におけるフェーズあたりの作業時間の割合 (%)

課題番号	PLAN	DLD	DLDR	CODE	CR	UT	PM	開発時間 (分)	規模 (LOC)
5	18.4	30.3	8.8	16.1	6.2	16.8	3.3	333	88
6	20.3	18.5	20.0	15.8	7.9	11.3	6.0	265	119
Total	19.2	25.1	13.8	16.0	7.0	14.4	4.5	598	207

表 5: 欠陥型別の欠陥件数と修正時間

欠陥型	課題 1 から課題 4			課題 5 から課題 6		
	件数	欠陥密度	平均修正時間 (分)	件数	欠陥密度	平均修正時間 (分)
文書化	5	6.82	8.2	1	4.83	2.0
文法	2	2.73	2.0	4	19.32	5.5
インタフェース	12	16.4	7.2	0	0.00	0.0
チェック	1	1.36	2.0	1	4.83	2.0
データ	2	2.73	3.5	0	0.00	0.0
機能	16	21.8	7.4	5	24.15	5.8

3. 着目した欠陥型を原因ごとに分類し，主な混入原因を特定する．

4. 手順1から3の結果をもとにVDM++による設計記述で混入を防ぎたい欠陥を決定する．

上記の手順にしたがい，表6を得た．小分類の内容については次の通りである．

インタフェース欠陥

(A) 操作・手続きに関する欠陥
例： ループや操作順序に関するもの

(B) 条件に関する欠陥
例： 想定されていなかった条件が存在した

(C) 入力ミスに関する欠陥
例： 論理演算子の入力間違い

機能欠陥

(A) 操作・手続きに関する欠陥
例： ループや操作順序に関するもの

(B) 条件に関する欠陥
例： 想定されていなかった条件が存在した

(C) 入力ミスに関する欠陥
例： 論理演算子の入力間違い

(D) その他
例： API の仕様を誤解していた

今回は平均修正時間の大きい次の3つの欠陥に着目してVDM++の利用方法を検討することとする．

- DLDで混入しCODEで除去されるインタフェース欠陥の（B）
- CODEで混入しUTで除去される機能欠陥の（A）
- CODEで混入しUTで除去される機能欠陥の（C）

表 6: 欠陥データの分析結果

大分類	小分類	発生件数	総修正時間	平均修正時間
DLDで混入しCODEで除去されるインタフェース欠陥	(A)	3件	5分	1.6分
	(B)	4件	63分	15.8分
	(C)	0件	-	-
CODEで混入しUTで除去される機能欠陥	(A)	6件	62分	10.3分
	(B)	1件	3分	3分
	(C)	4件	27分	6.8分
	(D)	0件	-	-

3.3 VDMの導入方法の提案

3.2節の結果に基づき下記の2つの導入方法を提案する．

1. VDM++で記述した設計の静的な性質のみを検査する方法
2. 仕様アニメーションにより妥当性検証まで行う方法

導入方法1ではDLDで混入しCODEで除去されるインタフェース欠陥の（B）の混入を防ぐことを目指す．混入の原因はメソッドの責任があいまいになっていることであったので設計フェーズにおいてVDM++によりインタフェースの設計記述を行い，各メソッドの責任を明確にする．記述の手順としてはまず，すべてのクラスおよび操作・関数のシグネチャの概略を記述する．次に不変条件・事前条件・事後条件を記述する．さらに，各メソッド間に依存関係がある場合には陽な記述を行い依存関係をすべて記述する．ここでメソッド間の依存関係がある場合とは，あるメソッドの実行結果が他のメソッドの振る舞いに依存している場合をいう．その後，VDM支援ツールであるVDM++ Toolbox[4]を用いて構文チェックと型チェックを行う．エラーがあった場合には修正をする．以上を設計フェーズでおこなう．

導入方法2ではCODEで混入しUTで除去される機能欠陥の（A）および（C）の混入を防ぐことを目指す．ここで防ごうとしている欠陥は，条件を記述し間違えた場合や条件を理解し間違えていた場合にレビューでは発見が難しい．そこで，導入方法1に加えて複雑な条件判断を含む状態操作を行うようなメソッドについてはDLDフェーズにおいて陽に設計を定義する．DLDRフェーズにおいてツールを用いた設計の仕様アニメーションにより妥当性検証をして欠陥があれば除去する．

課題5では導入方法1を用い，課題6では導入方法2を用いてプロセスデータの測定を行う．

4 VDMの導入後のプロセスデータ

3.3節で提案したVDMの導入方法により実施した課題5と課題6のプロセスデータをのうち表4に作業時間ログを，表5に欠陥ログを示す．

5 考察

5.1 VDMの導入効果について

欠陥の混入の仕方と時間配分の変化についての考察をおこなう．3.2節で対象として選んだ欠陥の種類はインタフェース型と機能型の一部であった．

[4]VDM information web site: http://www.vdmtools.jp/

表5よりVDM++記述による設計を導入した課題5と課題6ではインタフェース型の欠陥が混入していないことがわかる．今回提案したVDMの導入方法はインタフェース型欠陥の混入を防ぐことに効果があったといえる．一方，機能型の欠陥については欠陥密度が導入前後で変化していない．しかし，機能型欠陥の除去フェーズに着目すると，VDMによる設計の導入前は機能型欠陥の87.5%(17件中14件)がUTフェーズで除去されているが，導入後にUTフェーズで除去された機能型欠陥は20%(5件中1件，他の4件はDLDRフェーズで除去されている)であった．設計を検証可能な方法で記述することにより上流での欠陥除去ができるようになったためであると考える．

上流で除去される欠陥が増加したことは図2より，VDM++による設計記述導入後に欠陥除去率[5]が上昇していることからもわかる．CODEフェーズでの作業時間割合が減少し，DLDとDLDRフェーズでの作業割合が増加していることから，上流のフェーズに時間をかけたことでUTフェーズまでに発見される欠陥が増え欠陥除去率が向上したためであると考える．

開発規模（kLOC）あたりの作り込み欠陥数はVDM++導入前後でそれぞれ51.8，53.14であり，規模あたりの作り込み欠陥の数自体はほとんど変化していない．この原因はインタフェース型欠陥の減少と同等の文法型欠陥が混入したためであることが表5よりわかる．文法型の欠陥は設計方法等とは独立して発生する欠陥であり開発者の言語に対する習熟度に依存する欠陥であるといえる．したがって，欠陥密度の変化が見られなかったことは今回提案したVDMの導入方法の有効性を否定することにはならないと考える．

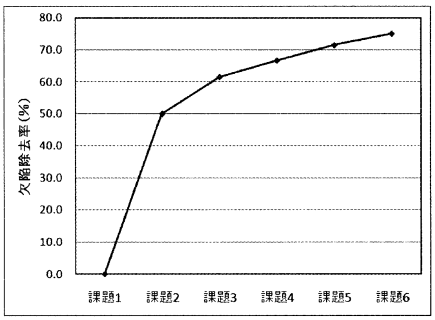

図 2: 欠陥除去率

5.2 導入コストについて

演習課題1から6の追加・修正規模と開発時間の関係をプロットした図を図3に示す．VDM++による設計記述の導入前後で生産性に変化は見られない．表3，表4からわかるように，CODEフェーズ以降にかかった時間の減少とDLDおよびDLDRフェーズにかかった時間の増加がおおよそ等しいからである．

今回の事例ではVDM++の記述の良さ（設計と実装が適切に分離できているか）の点は考慮していない．VDM++による設計を簡潔に記述する方法が確立できればDLDおよびDLDRフェーズの時間を短縮でき生産性の向上が可能になると考える．

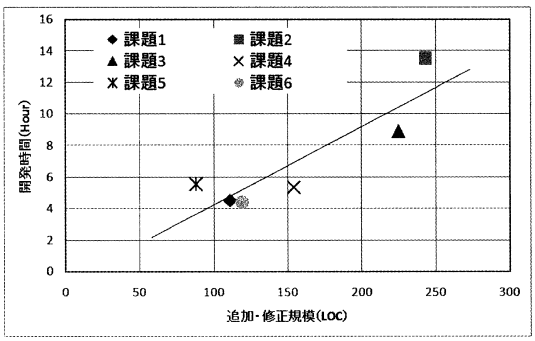

図 3: 追加・修正規模と開発時間の関係

[5]UT直前の時点までに除去された欠陥の割合

6 終わりに

6.1 まとめ

本論文ではPSPを用いて計測した第一著者の開発データを対象に，欠陥型に着目した個人プロセスのデータの分析にもとづくVDM++による設計記述の導入方法の提案と効果の考察をおこなった．

VDM++の設計記述のみを行った場合でもインタフェース型欠陥や機能型欠陥の作り込みを減らせること，全体の生産性はVDM++導入前と変化しないことがわかった．生産性が低下しなかったのはVDM++を用いた設計を導入したとしても設計フェーズにかかる時間の増加をコーディングフェーズにかかる時間の減少で取り戻せたためである．

6.2 今後の課題

本論文で扱った事例の範囲ではVDM++を用いた設計記述の導入前後で生産性の変化はみられなかったが，その詳細な分析は行っていない．今後，VDM++による設計記述の導入は個人の開発プロセスにおいて生産性の向上をもたらさないのか，記述の仕方によっては生産性を向上させることができるのかを検討していく必要があると考える．また，今回測定の対象とした演習キットで作成したプログラムは単純なものであったが，複雑な処理や状態遷移を含むプログラムでも本事例と同様の結果が得られるかについても実際に測定を行い検証を行う必要がある．

参考文献

[1] Marcel Verhoef, M. W., Manuel van den Berg: *Formal Specification of an Auctioning System Using VDM++ and UML, an Industrial Usage Report*, Chess Information Technology (1999).

[2] 栗田太郎, 太田豊一, 中津川泰正：携帯電話組み込み用“モバイルFeliCaICチップ”開発における形式仕様記述手法の適用とその効果, ソフトウェア・シンポジウム 2006, pp. 49–66 ソフトウェア技術協会 (2006).

[3] 情報処理推進機構：高信頼ソフトウェア構築技術に関する動向調査報告書 (2008).

[4] CSKシステムズ：VDM++言語マニュアル ver.1.2, CSKシステムズ (2008).

[5] 佐原伸：形式手法の技術講座, ソフト・リサーチ・センター (2008).

[6] Humphrey, W. S.: *PSP: A Self-improvement Process For Software Engineers*, Addison-Wesley Pub (2005).