技術紹介

ソフトウェアシステムの品質向上を目的とした取り組みの紹介 ーコーディング規約と静的解析の適用についてー

高野　武寿[*1]　久山　岳[*2]
Takano Takehisa　*Hisayama Takeshi*

制御、計測関連のソフトウェア開発業務において、実装フェイズのミス撲滅を最重点項目として、品質の向上に取り組んでいる。プログラム記述の決まりごとを規定したコーディング規約を設定し、これに従った実装を徹底するとともに、静的解析ツールを用いた解析の実施により、規約の遵守状況を審査することとした。本取り組みをモデル案件に試験導入し、一定の成果を得たことから、部門内に順次展開していく。

キーワード：実装品質、コーディング規約、静的解析

1. はじめに

当部門では、ソフトウェア開発を業務所掌としており、その中でも制御・計測分野を主要なターゲットとしている。

これらの分野のソフトウェアシステムは、制御装置や計測装置といったハードウェアと連携して機能するため、品質の維持・向上のためには、この分野に特有な取り組みが必要となる。

本稿では、当部門におけるソフトウェアの品質向上のための取り組みについて紹介する。

2. 品質向上への取り組みの背景

2.1 ソフトウェア品質に関する課題

ソフトウェアシステムの開発工程は一般的に、設計フェイズ、実装フェイズ、検査フェイズからなる。

電気、機械、建設といった分野でも、設計、実装（製造）、検査という概念は共通であるが、ソフトウェアシステムには、実装（製造）において以下に述べる特有の事情があり、品質の担保が難しい分野と言える。

機械、建設等の世界では、設計書、製作要領書等があれば、製作者が異なっても基本的には同等のものができあがる。このため検査の項目は、設計通りの物品が製造されていることを、設計値、性能値に着目して設定することになる。

しかしソフトウェアの世界では、上記の考え方はそのまま適用できない。

プログラマが各自の知見、スキルに従いプログラムを記述するため、同一の設計に対し、プログラマが異なれば百人百様のプログラムができ得る。ソフトウェア開発が、工業より料理や音楽に例えられる所以（ゆえん）である。このため、外部から見たソフトウェアの振る舞いだけでは、内部の構造の

*1：制御システム事業部　コンピュータ制御部　次長
*2：制御システム事業部　コンピュータ制御部

良否まで検証することは困難で、品質の担保が難しい世界であると言える。

ソフトウェアシステムでの開発プロセスは、設計を複数のレベルに分割し、各レベルに対応した検査を実施することで、品質を管理する手法が広く一般に用いられている。

このプロセスにおいて、設計フェイズは、システム要求分析、システム方式設計、ソフトウェア要求分析、ソフトウェア方式設計、ソフトウェア詳細設計という段階に分かれる。各段階で、要求から設計へと詳細化していくことで、開発するソフトウェアの仕様を細目に渡って明確化してゆく。これにより、実装フェイズでの誤認を招くような曖昧さを、極力減らしていこうという考え方である。

実装フェイズでは、詳細設計にて明確に示された事項に従い、プログラムを記述（実装）していく。

検査フェイズでは、単体テスト、ソフトウェア結合テスト、ソフトウェア総合テスト、システム結合テスト、システムテストという段階で、ソフトウェアの部分ごとからシステム全体へと、微視的な項目から巨視的な項目へと視点を変えて検査を行う。設計フェイズを逆にたどり、設計で示された事項が漏れなく実現されていることを確認するという考え方である。

設計から実装、実装から検査という流れは、実装を最下端としたＶ字型の概念図として示されることから、ソフトウェアシステム開発のＶ字モデル（**図１**）と呼ばれる。

ここで留意すべきは、検査により検証されるのは、ソフトウェアの内部的な構造や動作ではなく、外から見てわかる振舞いであるという点である。このため、内部構造に由来する潜在的不具合をいかに見逃さず、検査により品質を担保するかが課題となる。

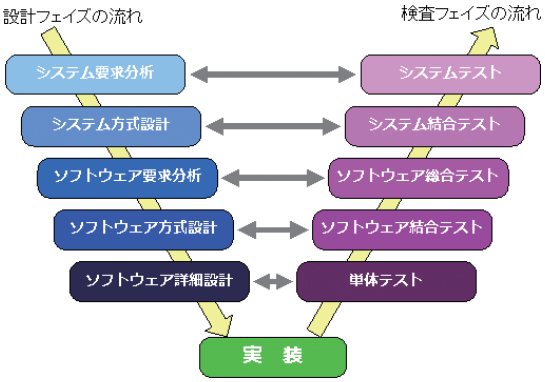

図１ ソフトウェアシステム開発のＶ字モデル

2.2 ソフトウェアシステム開発業務の特徴

当部門では、制御、計測関連のソフトウェア開発を、主たる事業対象としている。

制御の分野では、電気装置・機械装置が取り扱い対象である。また計測の分野では、各種の物理事象を数値化するための、センサ類が取り扱い対象となる。

ともに、ハードウェアを直接取り扱う必要が生じることから、それに適したプログラミング言語であるＣ言語と、その拡張形であるC++を使用することが多い。これらの言語は、プログラム記述の自由度が高いことから、プログラミングミスにより思わぬ誤動作を生じかねないという面も有している。

また、開発したソフトウェアシステム単独での動作とはならないことから、ハードウェアとのインターフェース部において、ハードウェアの動作状況との相関により想定外の挙動となって、不具合事象が現れることがある。

これらの問題は、設計フェイズよりも実装フェイズのミスに起因することが多い。このため、製品品質の向上を目的として、実装フェイズのミス防止を特に重要視している。

3. 実装品質向上の取り組み

当部門でも、ソフトウェアシステム開発分野の例に漏れず、V 字モデルによる開発プロセスを採用しているが、前述の事情により実装フェイズのミス撲滅を、最重点項目として挙げている。

このため、以下の取り組みを行っているので紹介する。

3.1 コーディング規約の制定

前述のようにソフトウェアの記述では、プログラマが異なれば百人百様となる面がある。そのようなばらつきを抑制するために設定するのが、プログラム記述の決まりごとを規定したコーディング規約である。

当部門では、ソフトウェアの品質低下に結び付き得る事項を、以下のように分類したうえで、各分類についてのコーディング規約を設定した。

各規約は、陥りやすい不適合の例と、本来あるべき適切な記述とを併記することで、読み手が自身の記述したプログラムの良し悪しを判断できる体裁とした。

(1) 文（ステートメント）に関する規約

ソフトウェアにおける文（ステートメント）とは、プログラムを構成する要素で、処理の手続きを表すものである。分岐処理、反復処理などの実行文があり、これらを用いてプログラムの処理の流れを記述する。各処理の条件判定等の記述を誤った場合、処理の流れが適切に実行されない。

コーディング規約では、文を取り扱う際に誤りやすい記述と、適切な記述とを示した。

(2) 宣言／初期化に関する規約

本項目は、主にソフトウェア内の変数の定義方法を規定するものである。

変数には、ソフトウェア内のメモリ配置の違いなどにより複数の宣言方法がある。目的にあった宣言方法を正しく選択しないと、誤動作を生じる場合がある。また、変数を正しく初期化しなかったことで値が不定となることや、誤った初期化操作により変数の内容を破壊してしまい、当該変数を用いた以後の演算処理の結果が、不正な値となる場合がある。

コーディング規約では、これらの問題を防止するために、変数の宣言や初期化に関わる実装方法のルールを定めた。

(3) データ型に関する規約

本項目は、主にソフトウェア内の変数のデータ型の取り扱いを規定するものである。

変数には、整数型や実数型、符号の有無、取り扱い可能な数値の範囲（データ長）の大小により、複数の種別がある。異なるデータ型同士で演算や代入を行った場合に、演算結果に思わぬ副作用が生じる場合がある。

コーディング規約では、異種のデータ型間での演算･代入にあたって､注意すべき事項を示しルール化した。規約の例を、**図 2** に示す。

(4) 演算子／演算式に関する規約

ソフトウェア内での数値演算には、四則演算以外に、論理演算・ビット演算という概念も加わる。各演算の種別間で、優先順位が定められているが、その誤認により演算処理にミスを生じる場合がある。

コーディング規約では、演算ミスの生じにくい演算式の記述方法や、第三者が読んだ時に意図を誤解しにくい記述方法を示した。規約の例を、**図 3** に示す。

(5) 配列／ポインタに関する規約

C 言語系の開発言語において、習得時に最も理解の難しい概念が、ポインタである。

ポインタとは、ソフトウェアをメモリ空間上に配置した場合の、メモリ上の位置（番地＝アドレス）を示すものである。主に、関数に対し数値を参照渡しする場合に用いる。特に経験年数の少な

いプログラマが、変数の内容値とポインタ値（配置のアドレス）とを取り違えるといったミスを生じやすいが、ポインタ値に対する演算処理など複雑な取り扱いをした場合には、ベテランであっても実装ミスをする可能性がある。

また、複数の変数の集合を取り扱うデータ構造である配列においても、ポインタの理解が必要であり、同様な実装ミスを生じる場合がある。

このため、コーディング規約では、メモリ上のデータ配置や、ポインタの取り扱いに関する規約を制定しミスの防止策とした。規約の例を、**図4**に示す。

(6) 関数定義に関する規約

ソフトウェアにおける関数とは、繰返し利用する一連の処理を、1つにまとめたものである。関数の名称、入力として与えるデータ、出力されるデータ、内部の処理などを定義すると、以後は関数の名称を呼び出せば、内部の処理を繰り返し実行することができる。

入力データと出力データには、それぞれデータ型の種別やポインタの取り扱いが関連する。このため、関数を取り扱う場合には、前述の **(3)**、**(5)** と同様の注意が必要である。

このため、コーディング規約において、関数を定義する際の記述方法について、呼び出しミスを生じにくい例を示した。

(7) 可読性に関するもの

ソフトウェアは、開発完了後の保守性に配慮し、読みやすい記述とする必要がある。

コーディング規約では、ソフトウェアの記述の意図や目的を、第三者が正しく理解できるよう、

5.10 演算に用いる変数と演算結果を代入する変数の型が異なる場合は，期待する演算精度の型へキャストしてから演算を行う。

変数の型によって値域や内部表現が異なり期待した演算結果にならない場合がある。期待する演算精度の型へ演算に用いる変数をキャストしてから演算を行う。

適合例	不適合例

int val1 = 314;
int val2 = 100;
double res = 0.0;
 	
res = (double)val1 / (double)val2;
	int val1 = 314;
	int val2 = 100;
	double res = 0.0;
	
	res = val1 / val2;
	res には 3.14 が入ることを期待したが 3 が入る

図2　データ型に関するコーディング規約の例

5.19 &&または||の右側のオペランドには，副作用があってはならない

&&や||演算子の右式は，左式の条件結果により実行されない場合がある。インクリメントなどの副作用のある式を右式に記述すると実行されない場合があるためそのような記述はしないようにする。

適合例	不適合例

tmp = *pVal;
pVal++;
tmp1 = *pVal;
if ((MIN < tmp) && (tmp1 < MAX)){
　・・・
}
	if ((MIN < *pVal) && (*(pVal++) < MAX)){
	　・・・
	}

図3　演算子に関するコーディング規約の例

5.20 ポインタ変数の間接参照とインクリメント・デクリメントを１つの式で行う場合は実行順序を( )で明示し，結果を代入してはならない。

間接参照とインクリ・デクリメントを１つの式で行うと，参照先をインクリ・デクリメントしているのかポインタをインクリ・デクリメントしているのか分かりづらくなる。括弧をつけて実行順序を明示したとしても結果を代入する文の場合，代入されてからインクリ・デクリメントが行われるのか分かりづらくなるので結果を代入してはならない。

適合例	不適合例

int data[] = {1,10,100};
int* pData = &data[0];
int tmp;
 	
tmp = *pData;
pData++;
 	
(*pData)++;
tmp = *pData;
	int data[] = {1,10,100};
	int* pData = &data[0];
	int tmp;
	
	tmp = *pData++; 代入を行ってからインクリメント
	tmp = *++pData;インクリメントしてから代入
	tmp = (*pData)++;代入してから参照先をインクリメント

図４　ポインタに関するコーディング規約の例

コメント等の説明文を適切に記述するためのルールを定めた。

3.2 静的解析による規約の遵守状況の審査

前節において、当部門で実装担当者に展開したコーディング規約を紹介した。

しかしながら、当部門の人員規模において（数十人、派遣社員を含む）、実装に当たっての決まりごとを示すのみでは、実装品質を担保する手段としては不十分であり、決まりごとの遵守を担保するための仕組みが必要と考えた。このため、静的解析の実施により規約の遵守状況を審査することとした。

静的解析とは、ソフトウェアの記述内容（ソースコード）について検証を行う手法である。ソフトウェアを実行しながら内部状況の解析を行う動的解析と、対を成すものである。動的解析が、ソフトウェアを実際に動作させ、その振る舞いを解析するのに対し、静的解析では記述内容の良否自体を解析する点が異なる。静的解析は、専用の解析ツールにより実施することが一般的で、解析対象であるソフトウェアのプログラム記述が、あらかじめ定めた解析ルールに合致していることを判定するものである。

コーディング規約の遵守状況を審査することを目的として、市販の静的解析ツールを導入した。導入にあたっては、さまざまな市販ツールの中から、解析ルールの定義の自由度が高い製品を選定した。

同製品の導入後に、以下の手順により解析ルールのチューニングを実施した。

(1) 当部門で制定したコーディング規約の全項目を、解析ツール内で使用可能な設定の形式に落とし込み、解析ルール化した。

(2) コーディング規約に意図的に違反したプログラムを作成し、解析ツールにより漏れなく検知できることを確認した。

(3) 過剰検知や検知漏れが発生した解析ルールについて、設定の見直しを行い適切な解析結果が得られるよう調整した。

この結果、単に市販のツールを用いるに留まらず、当部門のノウハウを注入したオリジナルの解析ルールを得ることができた。

3.3 コーディング規約および静的解析ツールの展開状況

前述のとおり、コーディング規約をルール化した静的解析ツールは、現在、モデル案件を設定し試験導入している段階である。

その過程で、潜在不具合となりうる不適切な実装が実際に発見されており、一定の成果を得ている。今後、試験導入に関わったメンバーを中心に、部門内に順次展開していく。

4. 今後の課題

C 言語および C++ については、前述の取り組みを順次展開する予定であるが、近年、ソフトウェアシステムの適用分野が拡大されるにつれ、開発環境と開発言語が多様化してきている。必ずしも、これまで主流であった C 言語や C++ のみを対象とした取り組みでは十分と言えなくなってきた。このため今後は、.Net Framework や Java といった開発言語についても、同様の取り組みを進める予定である。

各言語について特有の規約が必要となり、言語ごとに解析ツールも異なるので、それぞれについてノウハウを蓄積していく。

5. 結言

当部門において注力しているソフトウェアの品質向上の取り組みにおいて、実装の品質に着目し、コーディング規約を制定するとともに、その適用状況を担保するために静的解析ツールを導入した。試験導入により一定の成果を得たことから、今後は部門内に順次展開していく。

制御システム事業部
コンピュータ制御部
次長
髙野 武寿
TEL. 042-523-8315
FAX. 042-523-8320

制御システム事業部
コンピュータ制御部
久山　岳
TEL. 042-523-8315
FAX. 042-523-8320