第 10 章　ファンクション・ポイント

私個人の失敗から

ファンクション・ポイントの話は、私個人の失敗談から始めたい。

1980 年頃、当時私が勤めていた日興證券が「債券投資情報システム」と呼ぶ情報システムを開発することにした。私はその情報システムの規模を、100 万ステップと見積もった。

その情報システムの開発を受託することになる某 SI 企業の技術者は、その規模を 200 万ステップと見積もった。100 万ステップか 200 万ステップかで、私とその企業の営業担当者の間でしばらく激しい論争をして、最終的に私が勝って 100 万ステップでその SI 企業と契約し、開発作業に入った。

2 年後にその情報システムが完成し結果をチェックすると、開発したステップ数（ステートメント数）はおおよそ 200 万だった。つまり、SI 企業の技術者の見積もりが正しかったことが証明された。別の言い方をすれば、私の見積もりは大外れだった。その SI 企業はこの開発で、大赤字を出した。私はそれまで、情報システムの規模の見積もりにそれなりに自信を持っていた。しかしこの結果を受けて私は完全に自信を喪失し、それ以降私は情報システムの規模の見積もりを一切行わないことにした。

当時、日興證券の子会社の日興システムセンター（企業名はいずれも当時のもの）は、情報システムの開発にプログラム言語として PL／1 を使っていた。したがって私の見積もりは、PL／1 でその情報システムを開発する前提に立っていた。一方その SI 企業は、もっぱら COBOL を使用していた。したがってその企業の技術者の見積もりは、COBOL ベースのものだった。そして実際の開発も、COBOL を使ってなされた。つまり後になって分かったことだが、規模の数字の違いは使用するプログラム言語の表現力の違いに起因したものだった。

1980 年頃には、私の耳にファンクション・ポイントの話はまだ届いていなかった。したがって同じ情報システムの開発で、使用するプログラム言語が変わることによって開発する量が大きく変わることを、私は知らなかった。その後ケーパース・ジョーンズ（Capers Jones）の本[1]でファンクション・ポイントを知り、結果として私の見積もりもそう外れていた訳ではないことを知った。私はそれなりに自信を取り戻したけれど、時はすでに遅かった。長い間情報システムの規模の見積もりを避け続けていたため、自信は取り戻したけれど以前の能力が戻った訳ではなかった。

これが、私の失敗談である。

プログラムのステップ数

今でもまだ情報システムの開発で、規模の見積もりにプログラムのステップ数が使用されることがある。しかしソフトウェアの規模の単位にステップ数を使用することは、以下の理由で望ましくない「JON08」。

- 使用するプログラム言語によって、数値に大きな差が出る。
- ステップ数の数え方にいくつかの方法があって、統一されたルールがない。
- そのためステップ数を使って表現されたプロジェクトの生産性や品質のデータは、同じ言語を使用したものだけの間でも比較が難しい。

[1] 参考文献で挙げたケーパース・ジョーンズの今の本「JONES08」は第 3 版だが、その初版の翻訳本は 1993 年に発行されている。私は、その直後にその本を読んだ。

- 最近は複数の言語を使用するプロジェクトがあったり、Visual Basic のようにプルダウンメニュー方式などでプログラミングする言語が使われたりで、ステップ数の概念が希薄になってきている。
- プログラムのステップ数は、プロジェクトの中頃のプログラム構築の作業が終わらなければ確定できない。
- アセンブラのような低水準言語を使用することで、Java などの高水準言語と比較して高い生産性を達成したような結果を出す。（この詳細な話は、後述する。）
- 利用者や開発を委託した立場の人には、プログラムのステップ数は意味の無いものである。
- 要件定義書や各種設計書、利用者用の文書などはプログラムのステップ数とは直接関係しない。

ここで記述したようなことから生じる問題を避けるため、プログラムのステップ数に代えてファンクション・ポイントを使用することが好ましい。

ファンクション・ポイントとは何か

それでは、ファンクション・ポイントとは、何だろうか。

ISO と IEC、IEEE が合同で作成した用語集には、ファンクション・ポイントの説明として以下の 2 つが挙げられている[ISO10a]。

1. アプリケーション、又はプロジェクトのサイズを表す単位
2. アプリケーション・ソフトウェアの機能的なサイズを表す量

後述するように、今はファンクション・ポイントに多くの種類がある。しかし当初は 1 つの種類しかなく、それはビジネス・アプリケーションの機能の大きさを表すものだった。

つまりこれまで述べてきたステップ数による計測の弊害を避けるために 1979 年に、当時 IBM に勤めていた A. J. アルブレヒト（A. J. Albrecht）がファンクション・ポイントの構想を発表した[2]。

アルブレヒトは、ビジネス・アプリケーションの機能の大きさは次の 5 つで表すことができるとした。

- 入力
- 出力
- 照会
- 内部論理ファイル
- 外部インタフェースファイル

そしてこのそれぞれが 3 段階の複雑さを持ち、それによって機能の大きさが変わるとした。その複雑さによる機能の大きさの変化について、今の IFPUG 法の数値を図表 10-1 に示す。

つまりあるプログラムに注目すると、そのプログラムの前記 5 種類の要素のそれぞれの数とその複雑度を決定し、図表 10-1 で示す数値を合計することで、そのプログラムの機能量が求まる。これを「未調整ファンクション・ポイント」という。

[2] アルブレヒトは、1979 年の秋に開かれた IBM ユーザー団体である GUIDE と SHARE の合同シンポジウムで、ファンクション・ポイントについて発表した。

「未調整」という言葉が付いていることからも分かるように、ここで求めた数値を調整する方法がある。同じトランザクションの処理といっても、例えばオンラインの処理とバッチ方式での処理では難しさが異なることがある。集中処理方式の場合と分散処理の場合でも、やはり難しさは異なる。したがって図表 10-2 に示す 14 項目を調整のための項目として、それぞれについて今ファンクション・ポイントを求めようとしている情報システムについて 0 から 5 までの 6 段階で評価し、まずその評価点の合計値を出す。そしてその合計値を使って、ファンクション・ポイントの数値を補正するという段階で調整を行う[3]。

図表 10-1　複雑さの変動による機能の大きさの変化[UZAWA13]

	複雑度		
	低	中	高
入力	3	4	6
出力	4	6	7
照会	3	4	6
論理ファイル	7	10	15
外部インタフェースファイル	5	7	10

図表 10-2　調整のための項目[UZAWA13]

1	データ通信
2	分散データ処理
3	性能
4	高負荷構成
5	トランザクション率
6	オンラインデータ入力
7	利用者効率
8	オンライン更新
9	複雑な処理
10	再利用可能性
11	導入容易性
12	運用性
13	複数サイト
14	変更容易性

この調整係数の合計値（TDI）を次式に代入して、調整係数（VAF）を求める。

VAF =　（TDI ＊ 0.01）+ 0.65

つまりこの方式で調整した場合、調整後のファンクション・ポイントは未調整ファンクション・ポイントの 65%から 130%の範囲になる[4]。

---

[3] 未修正ファンクション・ポイントを求める時の複雑度の判定は、さほど難しくはない。しかし調整項目の評価は、かなり難しい。これに的確に対応しようとすると、後述する JFPUG から CPM（Counting Practice Manual）を入手し、併せて教育を受ける、などが必要である。

[4] ここで述べているファンクション・ポイントの計測方法は、IFPUG が発行している CPM に基づいているため、IFPUG 法と呼ばれている。

なおこの計算方式を用いると、要件定義が明確になった時点でファンクション・ポイントの数値を求めることができる。しかし一般に、ファンクション・ポイントの計測は容易ではない[5]。

ファンクション・ポイントに関わる団体

アルブレヒトの発表後すぐに、この計測法を広める団体が活動を始めた。この団体を International Function Point Users Group（IFPUG）と呼ぶ。この団体は会員制の非営利の組織で、ファンクション・ポイント法の計測ルールを決めたり、その計測のための訓練を実施したり、ソフトウェア計測技術の普及や支援を行っている[JIS10a]。

日本にはその日本支部があり、「日本ファンクションポイントユーザー会（Japan Function Point Users Group：JFPUG）」[6]という。JFPUG は米国の IFPUG が決めた計測のルールを記載したマニュアル（Counting Practice Manual：CPM）を翻訳して販売したり、計測方法の講習や訓練を実施したりしている。

ファンクション・ポイントとステップ数

ケーパース・ジョーンズはその著書の中で、いくつかのプログラム言語についてファンクション・ポイント当たりのステップ数を提示している。その一部を図表 10-3 に示す。

図表 10-3　1 ファンクション・ポイント当たりのステップ数[JON08]

世代	言語	ステップ数／FP		
		低	中央値	高
1	Basic Aemebly	200	320	450
	Macro Asembly	150	213	300
	C	60	128	170
	Basic	70	128	165
2	Fortran	75	107	160
	ALGOL	68	107	165
	COBOL	65	107	150
	PASCAL	50	91	125
3	PL/1	65	80	95
	Ada83	60	71	80
	Lisp	25	64	80
	C++	30	53	125
	Ada9x	28	29	110
	Visual Basic	20	32	37
	APL	10	32	45
	SMALLTALK	15	21	40
	SQL	7	12	15
	Spread Sheet	3	6	9

IFPUG 法以外にいくつかのファンクション・ポイントの計測法がある。その中に逆算法というものがあり、実際に記述したプログラムのステップ数から図表 10-3 のような表を使用し

[5] この調整は必ずしも容易ではないので、今でも未調整のままでの数値がよく用いられる。
[6] JFPUG のホームページの URL は、http://www.jfpug.gr.jp/ である。

てファンクション・ポイントに変換するもので、ある意味で簡便にファンクション・ポイントの数値を求めることができる。しかしケーパース・ジョーンズ（Capers Jones）は、この方法は誤差が大きいので注意するようにと書いている[JON08]。確かに、COBOL を例にとると 1 ファンクション・ポイント当たりのステップ数は最低で 65、最高で 150 と、この間に 2 倍以上に開きがある。

図表 10－4　主なファンクション・ポイント計測法（[JON08]、[UZAWA13]）

計測法	概要	ISO規格	JIS規格
IFPUG法	Alan J. Albrechtの手法を引き継いでいる。米国や日本など、広く整怪獣に普及している。	ISO/IEC 20926:2009	JIS X 0142:2010
COSMIC法	ソフトウェアの計測に関する国際的な研究グループCOSMICが考案した手法。	ISO/IEC 19761:2011	JIS X 0143:2013
Mark-Ⅱ法	1983年に英国のソフトウェア研究家Charles Symonsが発表した手法。	ISO/IEC 20968:2002	
NESMA法	オランダのソフトウェア協会NESMAが考案した手法。	ISO/IEC 24570:2005	
SPR法	1985年に米国のSPR社が発表した手法。		
ファンクションポイントライト	David Consulting GroupのDavid Herronが開発したもの。		
フルファンクションポイント	1997年頃に、ケベック大学のAlain Abranが開発した手法。		
パターン・マッチング	ファンクション・ポイントの数値が分かっている既存の情報システムなどと比較することで、簡便に数値を求める。		
ストーリー・ポイント	アジャイル方式の開発で、ユーザー要求を展開した「ストーリー」で機能の規模を表現する。		
オブジェクトポイント	FPの概念をオブジェクト指向の概念と混合したもの。		
未調整ファンクションポイント	IFPUG法で、調整作業を行わないもの。		
ユースケースポイント	IFPUG法の基本要素をベースに、オブジェクト指向のいくつかの要素を付加したもの。		
WEBオブジェクトポイント	Webベースのアプリケーション向けの機能規模測定法。		
逆算法	ソースプログラムのステップ数からファンクション・ポイントを逆算する方法。		

その他のファンクション・ポイント計測法

IFPUG 法については、計測法の概要を示した。この方法は今ではそれに限定されている訳ではないが、ビジネス・アプリケーションの計測からスタートしたのでそれに向いたものを計測法のパラメータで使用している。

しかし世の中のソフトウェアは、今やビジネス・アプリケーションだけとは限らない。組込みソフトがあり、制御系のソフトもある。日本はゲームソフトの分野で、たいへん健闘している。開発方法にも構造化技法やデータ中心アプローチの他に、オブジェクト指向技法もある[7]。これらを受けて、ファンクション・ポイントの計測法も今は IFPUG 法に加えて、いくつかの

[7] 各種の開発方法論については、第 6 部で記述する。

ものが存在している。その中には、ISO が計測法を国際標準にしているものもある。

図表 10-4 で IFPUG 法を含めて、主なファンクション・ポイント計測法をまとめた。

ステップ数を使用することで生じる矛盾の例

ケーパース・ジョーンズの著書に、アセンブラのような低水準言語を用いると生産性が高く、Java のような高水準言語を用いると生産性が低く出るという矛盾が発生することについての詳細は説明がある[JON08]。ここで、それを紹介したい。

仮に、ファンクション・ポイントによる全体の規模が 35 である情報システムを、アセンブラと Java を使って開発するとする。アセンブラには 7,500 ステップが必要であり、Java は 2,500 ステップですむ。この概要を、図表 10-5 に示す。

要件定義や設計、テスト、ユーザー用の資料の作成などは使用する言語に関係なく一定（3 人月）である。コーディングはアセンブラでは 3 人月、Java では 1 人月とすると、合計の工数はアセンブラの場合 6 人月、Java の場合 4 人月になる。これでそれぞれの場合のステップ数（アセンブラは 7,500、Java は 2,500）を割ると、人月当たりのコード行数はアセンブラの場合 1.250 ステップ／人月、Java の場合 625 ステップ／人月となり、アセンブラを使用した方が生産性が高いような数値が出る。

図表 10-5　ステップ数が不適切な例[JON08]

	アセンブラ	Java
情報システムのコード行数	7,500	2,500
コーティング以外の工数(人月)	3	3
コーディング工数(人月)	3	1
総プロジェクト工数(人月)	6	4
人月当たりのコード行数	1,250	625

これを、ファンクション・ポイントを使って計算すると、当然 Java の方の生産性が高いという結果になる（図表 10-6 参照）。

図表 10-6　ファンクション・ポイントの場合[JON08]

	アセンブラ	Java
情報システムのファンクション・ポイント	35	35
コーティング以外の工数(人月)	3	3
コーディング工数(人月)	3	1
総プロジェクト工数(人月)	6	4
人月当たりのファンクション・ポイント	5.83	8.75

固定費の大きな製造プロセスでは、生産数量が減少すると単位当たりのコストは上昇する。これは、産業革命直後から分かっていたことである[JON08]。これが、情報システム開発で現れた例である。

これも、ファンクション・ポイントを使う方がステップ数を使うより好ましいことの 1 つの例である。

キィワード

ファンクション・ポイント、ステップ数、未調整ファンクション・ポイント、IFPUG、IFPUG 法、JFPUG、CPM

略語

IFPUG：International Function Point Users Group
JFPUG：Japan Function Point Users Group
CPM：Counting Practice Manual

人名

ケーパース・ジョーンズ（Capers Jones）、A. J. アルブレヒト（A. J. Albrecht）

規格

IFPUG CPM

参考文献とリンク先

[ISO10a] ISO/IEC/IEEE, “System and software engineering – Vocabulary－ISO/IEC/IEEE 24765:2010(E),” ISO/IEC, 2010-12-15.

[JIS10a] 日本工業標準調査会審議、「ソフトウェア技術－機能規模測定－IFPUG 機能規模測定法（IFPUG4.1 版未調整ファンクションポイント）計測マニュアル」、日本規格協会、平成 22 年.

[JON08] Capers Jones 著、富野壽、小坂恭一監訳、「ソフトウェア開発の定量化手法 生産性と品質の向上を目指して 第 3 版」、構造計画研究所、2010 年.
この本の原書は、以下の通りである。
Capers Jones, “Applied Software Measurement: Global Analysis of Productivity and Quality Third Edition,” McGraw-Hill, 2008.

[UZAWA13] 鵜澤　仁著、「実践！実例で学ぶファンクションポイント法」、（財）経済調査会、平成 25 年.

（2014 年（平成 26 年）5 月 2 日　新規作成）