SEC BOOKS

ESMR-2

組込みソフトウェア向け
プロジェクトマネジメントガイド
[定量データ活用編]
～定量データ活用による組織の開発･管理力向上～

独立行政法人 **情報処理推進機構**
技術本部 ソフトウェア高信頼化センター

目次

第1章

第2章

第3章

第4章

第5章

第1章

1. はじめに

組込みソフトウェアの大規模化・複雑化が言われて久しくなりますが、その傾向はますます進んでいます。単に一つの製品の内部の話に留まらず、製品の複合化にともなって開発スタイルが従来日本の得意技と言われた「摺合せ開発」から「組合せ開発」に変わらざるを得なくなってきています。さらに、ＩｏＴ時代における組込みシステムは、M2M に代表されるセンサネットワークであったり、構成要素が独自に更新を繰り返し成長していくようなシステムであったり、また独自に学習を繰り返して進化していくようなシステムであったりします。システムのテストの概念を根底から変えなければならないかもしれません。

このようなシステムは、当然高い信頼性と安全性を備えていなければなりません。では、このようなシステムを実現する組込みソフトウェアの開発はどうでしょう。勿論従来の技術では対応できない部分は、必然的に新しい技術を導入することになるでしょう。一方開発や管理の方法はどうでしょうか。従来通りの人に依存した方法で大丈夫でしょうか。品質・コスト・納期は、適切に見積もれるでしょうか。見積もり通りに開発できるでしょうか。開発の状況がどうなっているのか見えているのでしょうか。五里霧中でどうしていいか解らず立ちすくんでしまうことは無いと言えるでしょうか。

組込みソフトウェアの開発管理にも、工学的な管理手法の採用が必須になりつつあるのではないでしょうか。

さて、工学的な管理をするにはどうすればよいでしょうか。ケルビン卿の言葉に「測れないものは、予測したり管理したりすることができない」というのがあります。工学的な管理を行うためには、ソフトウェア開発状況、開発成果物の良否が見えることが大前提です。そのためには、開発状況、開発成果物（規模、性質、良否など）を判断するための定量的指標が必要です。そして、指標の値を決定する要素の定量データを測定・収集しなければなりません。

本書では、工学的な管理を実践するための指標と定量データ、さらに定量データをタイムリーかつ低コストで収集する方法について提案しています。組込みソフトウェアの開発・管理に携わる方々に有用な情報として参考にして頂けると幸いです。

最後にご多忙の中、定量データ活用で使用する用語や読み易い文章への見直し等、多くのコメントをいただきました、東海大学　理学部　古山恒夫　教授、東洋大学　経営学部　経営学科長　野中　誠　教授に謝意を表します。

2015 年 11 月

独立行政法人　情報処理推進機構（IPA）
技術本部　ソフトウェア高信頼化センター（SEC）
システムグループ
三原 幸博、松田 充弘

第2章

2. 経営者の行動に役立つ定量データの活用

ソフトウェア開発に関わる企業の経営者は、損益の予実管理に対する経営判断や競争力を維持するための開発力改善の判断を的確に行うために、ソフトウェア開発を定量的に把握する必要がある。また昨今では、お客様に対する品質説明責任が問われており、説明にはソフトウェア開発を定量的に管理していることが不可欠である。2章では、ソフトウェア開発の定量データを活用することがこれらの場面で、どのように効果的なのかその一例を説明する。

2.1 お客様に対する品質説明責任と定量管理

経営者はステークホルダーや市場に対して当該組込み製品・組込みシステムの説明責任を果たさなくてはならない。そのためには、論拠となる開発基準（開発プロセス）や品質基準のエビデンスとなる製品の品質データ（定量データ）を記録しなければならない。

（1）開発基準（開発プロセス）

開発プロセスは、SPICEなどの国際標準、CMMIなどの業界標準や会社固有のものが様々存在する。開発が国内だけで行われ、販売先の市場も国内だけであれば会社固有の開発プロセスを採用していても、特段問題にはならない。しかし、販売先が海外（グローバル）、開発がオフショアを含めグローバルである場合、品質の説明責任を果たすという意味から、国際標準に準拠していることが望ましい。

開発プロセスも広い意味では、品質目標を達成する手段の一つとして品質基準に含まれるが、大きな纏まりであることと製品品質との関係が間接的であることから、ここでは、品質基準と分けて考える。

定量管理では、個々のプロセスにおける基準の遵守割合、個々のプロセスにかけるリソースの多少、個々のプロセスで生成される成果物の種類と量をもって品質を推定する。

（2）品質基準（製品品質）

製品品質を直接的に左右する要素に係る基準には、レビュー / テストの網羅度、投入リソース、テスト終了基準、検出バグの多少などがある。殆どが定量データであり品質データとして記録し品質保証の根拠となる。組込み製品・組込みシステムの重要度に応じた対応が求められる。

2.2 損益の予実管理

定量データを収集、整理することにより、開発中のプロジェクトの生産性、組織としての生産性を把握することが出来る。これにより見積もりの適正化につながる。

また、開発中プロジェクトの管理指標（コスト、品質、進捗）の予実とリスクを可視化できることからタイムリーに対策を講じることで計画（値）からの乖離（悪化）を防ぐことが可能になる。

また、開発中製品の製品としての最終損益（生涯損益）の予測も可能になることから、製品戦略、販売戦略、技術戦略など総合的な戦略の見直しが可能になる。

本節では、プロジェクトを担当する各部門の品質、生産性の比較によって対策対象部門を見つける管理方法の例、製品の障害損益を推測する管理方法の例を紹介する。

図表 2-1 は、特定の開発プロジェクトの生産性と品質の詳細を定期的にチェックするための資料で、開発中のどの機能（サブシステム）或いは、どの開発担当部門の品質、生産性に問題があるか判断できる。表(A)、表(B) はそれぞれ、サブシステム 1 とサブシステム 2 のソースコード行数（SLOC 数）と検出バグ数を各サブ機能毎に一覧で示している。表(C) は、担当部門毎の開発費を示し、表(A)、表(B) の SLOC 数と検出バグ数を担当部門に集計することにより、担当部門毎のバグ密度（プログラム 1000 行当たりの検出バグ数)、生産性（プロ

図表 2-1　開発プロジェクトの生産性・品質詳細の四半期報告の例

表（A）

サブシステム 1（SS1）					
サブ機能	① SLOC 数（KSLOC）	②検出バグ数（累計）			担当部門
			深刻度大	その他	
AAA	29	118	79	39	自社
BBB	297	56	50	6	自社
CCC	157				外注
DDD	16				外注
EEE	134	21	11	10	自社
FFF	1	2	2		外注
GGG	54	3	3		自社
HHH	18	1	1		外注
III	645	283	175	108	現法
JJJ	28				自社
KKK	1				外注
LLL	102				外注
MMM		64	42	22	自社
NNN					自社
OOO		2		2	外注
PPP					自社
XXX	-	118	67	51	XX 社より購入
YYY		4	4		YY 社より購入
ZZZ		1	1		ZZ 社より提供
xxx		2	2		xx 社より購入
yyy		11	9	2	yy 社より購入
合計	1,482	686	446	240	

表（C）

マクロ分析

【開発ｽﾀｰﾄ～15/1Q】

	③ 開発費（百万円）	1KSLOC当り バグ密度 ②/①	1KSLOC当り 生産性 ③/①
SS1（自社）	641	0.34	1.18
SS1（外注）	-	0.04	-
SS1（現法）	752	0.27	1.02
SS 2（現法）		0.99	
SS 2（自社）	852	1.46	2.72

➡ 生産性低い（但し、難易度高い）

グラム 1000 行あたりの開発費用）を導出している。この例では、赤枠で囲った SS2 を担当している自社部門の品質（バグ密度:高いと品質が低い）と生産性（この例の計算式では大きいと生産性が低い）が他の部門と比べて低いことが分かる。そこで、他の部分と開発の難易度を比較するなど要因を調べた上で、対策を立てることができる。

表（B）

サブシステム 2（SS2）					
サブ機能	① SLOC 数（KSLOC）	②検出バグ数（累計）			担当部門
			深刻度大	その他	
aaa	9	124	77	47	自社
bbb	45				自社
ccc	2	2	1	1	自社
ddd	13	37	19	18	自社
eee	7	22	19	3	現法
fff	17	187	102	85	自社
ggg	13	10	5	5	自社
hhh	7	27	10	17	自社
iii	28	14	9	5	自社
jjj	69	99	50	49	自社
kkk	18	4	1	3	自社
lll	1	9	3	6	現法
mmm	70				現法
nnn	1				自社
ooo	5				現法
ppp	1				現法
qqq	2				自社
rrr	2	0			自社
sss	1				現法
ttt	0				現法
uuu	14				現法
vvv	1	0			自社
www	5				自社
aaaaa	2				現法
bbbbb	2				現法
ccccc	6				現法
ddddd	5	43	30	13	現法
eeeee	36				現法
fffff	1				現法
ggggg	0				現法
hhhhh	4	51	32	19	現法
iiiii	8	26	25	1	現法
jjjjj	3	0			自社
kkkkk	1				自社
lllll	3				現法
ZZZZZ		26	21	5	ZZ 社より提供
aa	-	156	63	93	α α社より購入
bb		160	60	100	β β社より購入
cc		26	20	6	γ γ社より購入
合計	405	1,023	547	476	

図表 2-2 は、ある組織の全開発プロジェクトの品質・生産性を一覧で示した月次報告の例である。部門あるいは全社で開発中のプロジェクト毎の品質をバグ密度で、生産性を人月当たりの開発プログラム行数で示している。更に、それぞれの内訳を詳細に示している。

プロジェクトが予定通りに進捗しているか否かのアラームは、一見して判別出来るようにフェイスチャートで示している。これにより問題のあるプロジェクトに絞って状況と対策、見通しと結果をフォローすればよい。

図表 2-2　全開発プロジェクトの品質・生産性月次報告の例

Y 月度月報

	アラーム		指標		品質								
					検出バグ数								
					今月（x年y月）抽出								残件数前月増減
					累計				残件数				
プロジェクト名	検出バグ密度（検出バグ数/KSLOC）	生産性（KSLOC / 人月）	検出バグ密度（① / ③）	生産性（③/②）	重大	重要	その他	①合計	重大	重要	その他	合計	
AAAAA	☹	☹	241.77	0.03	1	27	185	213	0	0	14	14	0
BBBBB	☹	☺	4.22	1.48	4	48	447	499	0	1	62	63	-37
CCCCC	☺	☹	0.70	0.44	8	20	135	163	0	0	22	22	0
DDDDD	☹	☺	3.96	20.54	42	124	796	962	0	3	57	60	-1
EEEEE	☹	☺	6.08	29.52	28	115	1,098	1,241	0	0	283	283	-23
FFFFF	☹	😐	6.84	0.86	3	19	216	238	0	0	16	16	-1
GGGGG	☹	☺	6.74	1.55	30	32	735	797	2	5	128	135	-9
HHHHH	☹	☹	31.44	0.06	0	1	25	26	0	1	25	26	23
IIIII	☹	☹	10.69	0.45	94	219	944	1,257	1	0	86	87	0
JJJJJ	☹	☺	6.87	1.68	1	4	66	71	0	0	1	1	0
KKKKK	😐	☺	1.57	4.55	15	79	255	349	0	0	0	0	0
LLLLL	☹	☹	36.62	0.12	0	6	46	52	0	0	2	2	0
MMMMM	☺	☺	0.72	20.20	1	6	38	45	0	0	5	5	0
NNNNN	☹	☺	6.56	1.45	14	20	444	478	2	0	70	72	-26
OOOOO	☹	☹	170.69	0.08	34	39	1,185	1,258	5	1	68	74	2
PPPPP	☹	☹	24.10	0.29	10	30	616	656	10	14	198	222	20

アラーム
【欠陥密度】
☺：青信号（1.0 未満）
😐：黄信号（1.0 ～ 3.0）
☹：赤信号（3.0 以上）

アラーム
【生産性】
☺：青信号（1.0 より大きい）
😐：黄信号（0.8 ～ 1.0）
☹：赤信号（0.8 以下）

生産性					
開発工数	開発規模（x年y月時点）				
② 累計工数 （人月） ～x年y月	母体含む SLOC 数 (KSLOC)	対再利用元製品			前月との差異 SLOC 数 (KSLOC)
		③ 新規 SLOC 数 (KSLOC)	再利用 SLOC 数 (KSLOC)	再利用率(%) 再利用 / 母体含む SLOC	
28.5 人月	1,264	1	1,263	99.9%	0
80.1 人月	1,035	118	917	88.6%	0
526.8 人月	1,145	234	911	79.6%	0
11.8 人月	1,161	243	919	79.1%	0
6.9 人月	1,088	204	884	81.3%	0
40.5 人月	1,109	35	1,075	96.9%	0
76.2 人月	1,107	118	989	89.3%	1
13.9 人月	1,161	1	1,160	99.9%	0
263.2 人月	1,164	118	1,047	89.9%	0
6.2 人月	1,163	10	1,153	99.1%	4
48.8 人月	1,092	222	870	79.7%	0
12.3 人月	1,092	1	1,091	99.9%	0
3.1 人月	1,184	63	1,121	94.7%	0
50.3 人月	1,072	73	999	93.2%	3
90.5 人月	1,027	7	1,019	99.9%	0
93.5 人月	1,096	27	1,069	97.5%	2

〈算出条件〉

SLOC 数　かぞえチャオ（かぞえチャオ！にて、**x年y月時点**での最新ソースコードから算出

母体含む SLOC 数　新規 + 修正 + 再利用 SLOC 数

新規 SLOC 数　新規 + 修正 SLOC 数

再利用 SLOC 数　再利用した SLOC 数

図表 2-3 は、特定の開発プロジェクトの開発費の予実算の内訳の推移を月次で報告した例である。この例では、担当部門（開発機能）毎の開発投資額と全体の投資額の予実を半期ごとに累計で示している。このデータと販売計画データを使えば製品の終身の損益を月次で計算（予測）することができ、タイムリーな経営判断につなげることができる。

図表 2-3　開発プロジェクトの開発費（予実）半期報告の例

AA プロジェクト開発費（サマリー）

2015 年 6 月 5 日
プロジェクト管理部

開発投資　予測（2015 年 5 月時点）

（百万円）

	担当部門	開発機能	2013 年度(実績) 上	下		2014 年度(実績) 上	下		2015 年度(予測) 上	下		2016 年度 上	下		2017年度 上
開発費	開発 1 部	AAA	0		0		153	153	97	6	104	9	9	18	6
		BBB	40	58	98	182	372	554	308	528	836	382	284	666	197
		CCC					183	183	238	211	448	202	175	377	196
			40	58	98	182	708	890	643	745	1,388	594	467	1,061	398
	開発 2 部	EEE				114	342	456	318	304	622	244	167	411	50
	開発 3 部	FFF							17	48	65	38	22	60	36
	回路		0	16	17	56	33	89	60	51	110	60	49	110	23
	意匠			3	3	12	10	22	17	24	41	20	16	37	2
	試験						23	23	30	51	81	50	49	99	61
	国内（小計）		41	77	118	364	1,116	1,480	1,084	1,222	2,307	1,007	771	1,778	571
	現法 1					142	319	461	663	668	1,331	616	301	918	383
	現法 2								48	50	98	92	69	161	
	開発費（①）		41	77	118	506	1,436	1,942	1,795	1,940	3,735	1,715	1,141	2,856	954
人件費	社内人件費（②）		18	36	54	72	142	214	156	156	312	113	89	202	65
	人員（単位：人）		5	10	-	20	41	-	44	44	-	35	30	-	25
開発投資（③=①+②）			59	113	172	578	1,577	2,155	1,951	2,096	4,047	1,828	1,230	3,058	1,018
開発投資累計			59	172	-	750	2,327	-	4,278	6,374	-	8,202	9,432	-	10,451

※合計値には丸め誤差の影響により、誤差が含まれている

2.3 競争力強化のための開発力改善

組込み製品、特に量産品は、それぞれの製品分野において日々競争に晒されている。競争力の源泉は、製品の付加価値であったり、信用あるいは販売のタイムリーさである。製品の機能の殆どがソフトウェアで実現されるようになってきており、付加価値競争力は勿論デザイン、アイデアもあるが、如何に早く実現（開発）できるかにかかっている。また信用は、信頼性の高さとその継続的提供である。

一度品質問題を起こして顧客の信頼を失うとブランドの喪失に留まらずビジネスの継続にまで影響が及ぶこともある。

これら競争力の源泉を強化するためには、開発力の継続的な維持・改善が必要となる。開発力は、組織文化、組織の管理力と技術者の能力の３つの要素に分けることが出来、それぞれの要素の到達度合いの組み合わせによって企業の開発力を判断することが出来る。

（1）組織文化

顧客満足を第一に考え、社員一人一人が当たり前のこととして実践している状態にしていること。そして、それが定量的に可視化されていること。例えば、明確な品質判断基準を持ち、それに従って定量的に品質が測定され、管理されていることである。

（2）組織の管理力

明確な品質判断基準を持ち、それに従った定量的品質測定を実現するには、開発の中で品質を随時定量的に測定出来る仕組みを持っていなければならない。いわゆる定量データの収集とそれの活用による組織の生産性、製品品質レベルの可視化（把握）と改善のための PDCA サイクルの実践能力である。

（3）技術者の能力

プロジェクト運営能力の如何が、プロジェクトの成否（品質、予算、納期）を決めることが多いことから、プロジェクトリーダの能力の判定と向上が企業活動を円滑に行う上で重要である。

プロジェクトのマクロなデータ（定量データ）を時系列で観測することで、プロジェクトリーダが適切な対応をタイムリーにとれているかを判断することが出来る。これを継続的に行うことによりプロジェクトリーダの能力の変化を

捉えることが出来る。

一方ソフトウェア開発においても技術者個人の能力が大きくプロジェクトの成否（品質、予算、納期）に関わってくる。個人の能力の差は一桁異なる場合があるとも言われている。従って、プロジェクトリーダの場合と同様に技術者のソフトウェア開発能力の判定と向上は重要である。プロジェクトのミクロなデータ（定量データ）を時系列で観測することで、個々の技術者の能力を判断することが出来る。これを継続的に行うことにより能力の変化を捉えることも出来る。

企業の競争力を維持していくためには、重要な要素である開発力の継続的な維持・改善が欠かせない。それには、開発プロジェクトの定量管理（定量データの収集と分析、及び、それに基づく管理、改善）を実践し続けることが欠かせない。

第3章

3. 定量データ収集と管理指標

2章では、経営者向けに定量データ活用の必要性やメリットを伝えたが、経営者は定量データ管理の推進に躊躇するケースが見受けられる。それは、経営者が定量データを収集する仕組みを構築することは手間とコストが掛かるという先入観をもっているためである。3章では、そのような経営者の懸念や否定的なイメージを払拭するために、比較的実行可能な仕組み構築方法およびソフトウェア開発の進捗・コスト・品質の状況を定量的に把握し比較できる指標と指標を導出する方法を紹介する。

3.1 収集すべきデータ項目

経営者の関心事は、現在進行中プロジェクトに対する進捗、コスト予測、品質状況であり、経営者による判断が必要な状況に陥っている場合はより速く察知しなければならない。また、自組織の将来のプロジェクトに対しては、生産性や信頼性を向上させて、競争力を維持することにある。そのためには、ハードウェアと違って見えないソフトウェアを色々な角度から可視化できるように次のデータを収集する。

・SLOC 規模とドキュメントボリューム
・工数
・品質データ
・プロファイルデータ
・工程工期、進捗管理データ

これらのデータは、生産性やバグ密度等の指標（モノサシ）を導出するための基礎データになる。

ここでのポイント

●収集が難しいデータを収集しようとはせず、最低限収集するデータを決める。
●規模や工数等、測定の仕方にバラつきがあるデータの精度を追及するこ

とは止める。定量管理の仕組みを構築することが目的なので、バラつきのあるデータは定量管理を収集する仕組みが構築できたあと、規模や工数の数え方の規定を標準化して精度を上げる。

3.1.1 SLOC 規模とドキュメントボリューム

ソフトウェアの規模は、工数やテスト項目数などと同様に、ソフトウェア開発の作業量を見積もる上で必ず収集しなければならないデータの 1 つである。

ソフトウェア規模を把握するためには、プログラムソースコードの行数 SLOC 規模（Source Line Of Codes; プログラム行数）を計測する。SLOC 規模は、開発に必要な工数やテスト項目数を見積もる際の基準となるデータである。

また、SLOC 規模は、生産性（単位工数あたりに開発できるプログラム行数）やバグ密度（ある単位のプログラム行数あたりに発見されるバグ件数）といったコスト予測や品質把握のための指標にも利用される。

新規・改造部分の SLOC 規模の変化量をモニターすることでコーディング作業の進捗を定量的にみることができる。ドキュメントボリュームの変化量をモニターすることで要求定義や設計の作業進捗をみることができる。ドキュメントボリュームの変化は、対象ファイルの更新日時、ページ数、文字数等を収集する。

コラム

規模を把握するデータには、SLOC 規模の他に FP（ファンクションポイント）規模があり、エンタープライズ分野で良く用いられる。エンタープライズシステムでは、ベンダ企業とユーザ企業の間でコスト見積もりの摺合せを行う場合に、機能毎に見積もることが可能な FP 規模を利用するほうが、SLOC 規模を用いるよりもユーザ企業の理解を得やすい。しかし、組込みソフトウェアでは、装置ドメインが異なると、機能の名称が同じでも機能の中身や異常系処理の深さが異なるため、FP 規模では作業規模を把握しづらい。定量データ活用の導入が進んでいない企業では、SLOC 規模を利用するほうが、作業規模を把握しやすい。FP 規模を活用したい場合は SLOC 規模による定量データ活用が定着したあと自組織に閉じた指標として試行してみれば良い。

3.1.2 工数

工数は要員の投入人数と期間で積算する作業量を表すもので、工数単価を掛け合わせることでコストを算出できる。プロジェクト実施期間の途中でのプロジェクトのトータルコストを予測したり、組織の生産性向上を目指すためには必ず収集しなければならない。作業者が１つのプロジェクトに専任している場合は、比較的容易に収集することができるが、複数プロジェクトを掛け持っている場合は、プロジェクトごとに工数を管理する仕組みが要る。標準的な開発プロセスが導入されていない組織では、工数を管理する仕組みすら無い場合がある。

3.1.3 品質データ

プロジェクトの最終成果物（ソフトウェア）の品質を定量的に把握出来るデータは、

・テストケース数
・検出バグ数

等、テスト結果のデータである。

ただし、テスト結果は開発後半になって初めて収集できるデータであるため、テスト結果から品質が悪いと判断し対策を講じた場合に、手遅れとなって納期遅延やコスト超過を引き起こすことになり兼ねない。そのため、テスト工程の以前の工程で、コーディング品質や設計品質等など出来るだけ上流の工程から品質データを収集する仕組みも構築して行くことが、重要である。

上流工程では品質データとして、

・レビュー指摘件数（仕様書レビュー、設計レビュー、コードレビュー）
・静的コードチェッカー指摘件数

等を収集する。

（１）テストケース数・着手数・完了数

テストには、単体テスト、結合テスト、総合テストがあり、それぞれ目的とテスト環境が異なるため、品質重視の考えに基づくなら工程を区切って実施するべきである。コストや納期の制約により効率重視の考えによって、一部の単体テストを省いて結合テストの中に単体テストを含めてしまったり、結合テストと総合テストを、工程を分けずに行ったりする場合も考えられる。テスト工程の決め方は、開発規模や上流工程での品質予測により、判断すれば良い。

いずれの場合でも、ソフトウェアもしくは機器やシステムとしての品質を定量的に表すためには、テストケースの内容を記録に残すとともに、実施（完了）したテストケース数を管理する必要がある。

（2）検出バグ数・解決数

実施したテストケースの内容が妥当なものか、また、テスト終盤で品質が確保されているかの判断のために、テスト実施時に検出したバグは、実施したテスト項目と関連付けて検出バグ数（検出バグ現象数）として管理することが必要である。

（3）レビュー指摘件数・解決数

上流工程でプロジェクトの品質を見るためのデータとして、要求仕様書レビュー記録、設計書レビュー記録、コードレビュー記録、テスト仕様書レビュー記録から、レビュー指摘件数、解決数を収集する。

（4）その他

・重複バグ数

1件のバグが原因で複数のバグ現象が発生することが良くある。バグ原因を分析することにより、検出バグ数のうちの重複バグ数（検出バグ現象数－検出バグ原因数）が分かる。重複バグ数が多い場合は、テスト効率が悪いと判断できるため、テストのやり方を見直す契機になる。

・静的コードチェッカー指摘件数

作成した全てのソースコードをレビューすることは、一般に作業効率の面で必ずしも推奨されていない。それを補うために静的コードチェッカーを掛けることは有効であり、チェッカーの指摘件数を、コードレビューを実施すべきかどうかの判断に用いることもできる。

3.1.4 プロファイルデータ

プロジェクトの特性、プロダクト（装置や機器）の特性を組織共通の様式で記録しておく。プロファイルデータは、他のプロジェクトデータと比較する場合や組織の指標作成に用いる。

3.1.5 工程工期、進捗管理データ

プロジェクトの進捗を把握するためには、プロジェクト計画時に進捗把握のベースとなる工程とその作業内容及び作業時期（開始、終了）をプロジェクト計画書の形で PMO（Project Management Office）部門とプロジェクトが共有していなければならない。工程毎の開始時期、終了時期は、プロジェクトと PMO 側が共有できる管理ツールや共有フォルダと共に決められた様式の進捗管理帳票を用いて管理する。

工程を決めるためには、作業内容の共通認識が必要であり、組織の標準的なソフトウェア開発プロセスが要る。全体の工期（納期）は、ソフトウェア開発プロジェクトの外部条件として与えられるが、各工程の工期は、関連部門と調整しながらプロジェクト内部で決める。

3.2 コストを掛けない定量データ収集方法

プロジェクトが管理している定量データの収集は、プロジェクトを支援するPMOがプロジェクトの邪魔をせず収集できるようにしなければならない。そのためにはプロジェクト側とPMOが協力して、どんな指標をモニターすべきなのかを明確にし、収集の仕組みを構築する。

定量データ収集の対象は、前節3.1に紹介した「収集すべきデータ」の実績値データや計画値データであるが、実績値の場合、SLOC規模の様に収集が容易な場合と、バグ数のように管理する仕組みが無ければ収集が困難な場合がある。計画値の収集は、プロジェクトリーダがプロジェクト計画書を作成し、経営者や品質管理部門と共有する仕組みが出来ていることが前提となる。

ここでのポイント

- ●コストを掛けないで収集できること。MS-Office等の既存ツールやオープンツールを活用する。
- ●収集したいときにタイムリーに収集できること。
- ●バッチファイルなどで環境に合ったツールを作り出来る限りデータ収集を自動的に行うこと。プロジェクト側に負担を掛けるとデータ収集は定着しない。
- ●プロジェクト側とPMO側が共有できる作業フォルダを整備すること。
- ●成果物の規模をPMOが収集できるように、ファイルの命名ルール・版数管理ルール・置き場所を決めておく。
- ●管理帳票類はファイルの命名ルール・置き場所・書式を決めておく。Excelで作成すれば、データ収集ユーティリティツール作成が容易にできる。標準書式はESPR（参考文献［1］）が参照できる。
- ●工数は勤怠管理システムから採れば良い。誤差は構わない。正確さの目安は、8割くらいでも良い。
- ●ビルドの頻度は、プロジェクトの終盤では、週次から日次とする。ビルド実施予定を決めておく。

◆収集の仕組みと環境作り

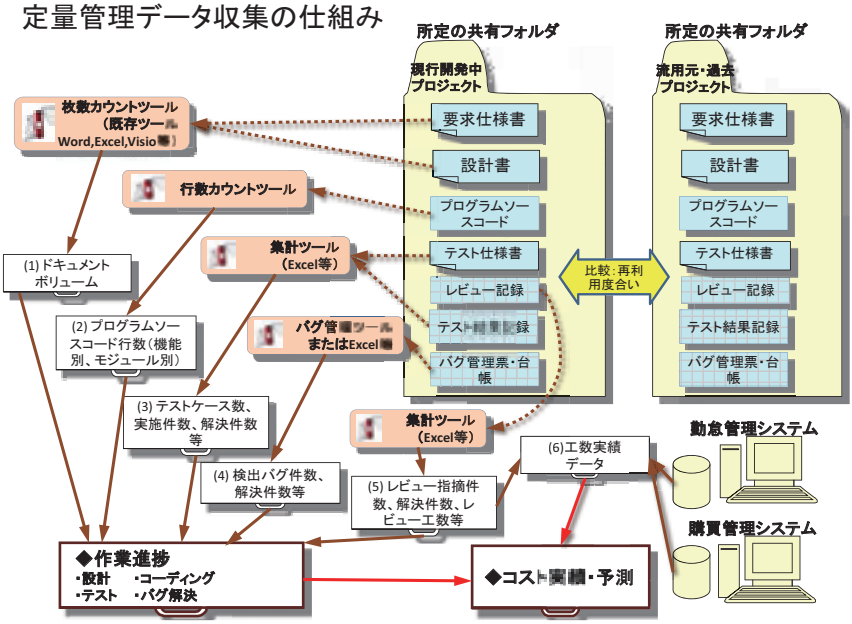
図表 3-1 に 3.1 項で紹介した「収集すべきデータ」とそれらのデータの収集先を示す。また、データの収集先からどのようにデータを収集するのかを図表 3-2「定量データ収集の仕組み」に示す。

図表 3-1　収集するデータと収集先

収集場所（収集先） / 収集するデータ	プロジェクト計画書	成果物				品質			勤怠管理情報	購買管理情報
		要求仕様書	設計書	プログラムソースコード	テストケース	レビュー記録	テスト結果記録	バグ管理票・台帳		
	置き場所：プロジェクト所定の共有フォルダ								社内システム	
(1) ドキュメントボリューム（枚数、項目数）		●	●							
(2) プログラムソースコード行数				●						
(3) テストケース数、着手数、完了数（単体テスト、結合テスト、総合テスト）					●		●			
(4) 検出バグ数、解決数、未解決数、重複バグ数（単体テスト、結合テスト、総合テスト）								●		
(5) レビュー指摘件数、解決件数、未解決件数（仕様書、設計書作成時）						●				
(6) 工数（全体、各工程、レビュー工数）						●			●	●
(7) プロファイル情報	●									

●収集先から入手するもの

図表 3-2　定量データ収集の仕組み



（1）ドキュメントボリューム

要求定義や設計作業の作業進捗の把握は、ドキュメントの枚数をカウントし、枚数変化をモニターする。

ソフトウェア開発におけるドキュメントは、ドキュメントの再利用や複数組織による分担作業を考慮して、一般には MS-Office 等業界の標準ツールによって作成することが多い。ドキュメントボリュームの変化量は、対象ファイルの更新日時、ページ数、文字数等からカウントするが、既存の MS-Office でファイルを開いてページ数や文字数を数えると開発部隊の邪魔をすることになるため、マイクロソフトが提供する“Dsofile.dll”等のユーティリティソフトの利用を勧める。Dsofile.dll は、対象ファイルを開くことなく、ページ数や更新日等のプロファイル情報を読むことが出来るが、VB 等による簡単なプログラミングを行ってツール化する必要がある。

〔参考〕Dsofile.dll の活用方法が分かるサイト：
http://www.microsoft.com/en-us/download/details.aspx?id=8422
https://technet.microsoft.com/ja-jp/scriptcenter/ff191274.aspx
http://macro-excel-vba.blogspot.jp/2015/04/dsofiledll.html

（2）プログラムソースコード行数

コーディング作業の進捗把握は、プログラムコード行数をカウントし、行数変化をモニターすることによって行う。

プログラムソースコード行数のカウントは、『かぞえチャオ！』等のフリーソフトやシェアウェアソフトと呼ばれているものが利用できる。

〔参考〕かぞえチャオ！の説明ページ：
http://homepage2.nifty.com/fortissimo/ft_manu.html

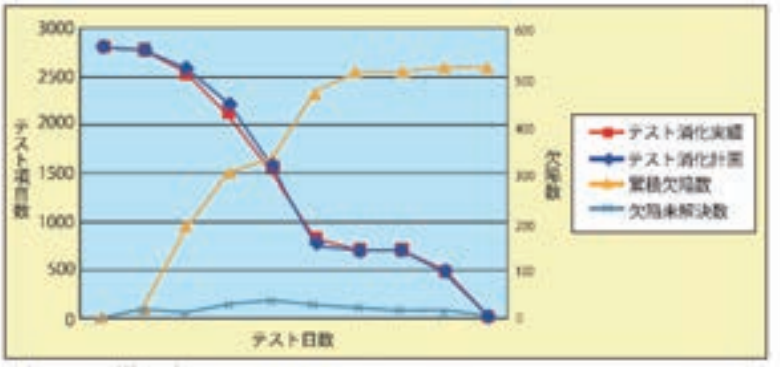
ここでのポイント

- ●コメント行は数えない
- ●プログラムソースコード行数を数えるツールを利用する
- ●プログラムの実行ファイルを作成するビルド作業フォルダは、プロジェクトと PMO が共有する。そうすることで、ソースコード行数のカウントは、PMO が行える。
- ●ソースコード行数を収集するユーティリティは、ビルドのフロー中に組み込めば自動収集できる。

（3）テストケース数、着手数、完了数等

テスト管理ツールが導入されている場合は、具備されている収集機能を利用すればよいが、Excelのような表計算ソフトウェアツールを利用している場合は、テスト担当者がテストに着手したり完了したりするタイミングで更新する管理表をモニターし、例えば、毎日決まった時刻に、消化項目数を集計する。テス

図表 3-3　テスト進捗及びバグ解決状況管理グラフ



ト進捗は、テスト期間中に見つかるバグ解決の進捗に連動するため、通常、図表 3-3 に示す「テスト進捗及びバグ解決状況管理グラフ」（参考文献［2］）を用いて管理する。

（4）検出バグ管理件数・解決件数

結合テストや総合テストで見つかるバグは、図表 3-4 に示すようなバグ管理票に登録し、解決までフォローする。バグ管理ツールが導入されている場合は、

図表 3-4　バグ管理票

バグ管理票

プロジェクト名					管理No		
状況	発見日		時間		発見者		対象機能名
	表題						
	［バグ内容／再現手順／発生率などを記載］						テスト項目ID
							ソフトウェアVer
							発見工程
	添付：□有 □無						
原因・影響	調査日		チーム		担当		主な原因
	原因						［原因分類用］
	添付：□有 □無						
	影響	［影響範囲／対応しない場合の影響など（必要に応じて記載）］					バグを作り込んだ工程
	添付：□有 □無						
対応内容	検討日		チーム		担当		見積工数
	完了日		チーム		担当		
	添付：□有 □無						
対応方針	対応する／しない		優先度		期限	備考	

※確認欄

対応承認	検証完了
／ ／	／ ／

図表 3-5　バグ登録台帳

バグ登録台帳

プロジェクト名		工程		更新日		管理No	
				更新者			

バグ管理票No	状況			原因・影響調査完了日	対応					検証完了日
	標題又は概要	対象機能名	発見日		チーム	優先度	期限	状態	完了日	

具備されている収集機能を利用すればよいが、ツール化されていないバグ管理票で管理する場合は、図表 3-5 に示すようなバグ登録台帳を Excel のような表計算ソフトウェアで作成し、記述様式を統一しておく。

バグ登録台帳には、登録日、重要度ランク、ステータス、解決日が記載されているので、そこから登録日から解決日までの TAT（Turn Around Time）と重要度ランクを収集する。

（5）レビュー指摘件数、解決件数、未解決件数

要求定義や設計の品質データは、レビュー記録帳票から収集する。仕様書レビュー、設計書レビュー、コードレビューによって指摘される問題は、レビュー記録票に登録し、解決までフォローする。管理ツールが導入されている場合は、具備されている収集機能を利用すればよいが、ツール化されていないレビュー記録票で管理する場合は、図表 3-6 に示すようなレビュー記録票を Excel のような表計算ソフトウェアで作成し、記述様式を統一しておく。

! ここでのポイント

- Excel で作成した帳票類は、PMO とプロジェクトが共有できるフォルダに置く。
- 帳票類は Excel で標準フォーマットを決めることで、PMO が必要な項目を取得するユーティリティツールを作ることができる。

図表 3-6　レビュー記録票

共同レビュー記録

プロジェクト名			管理No		
レビュー対象			部署		
日時	年　月　日(　)　:　～　:		確認	確認	作成
場所					
出席者 レビュー責任者					
出席者 レビューア					
出席者 被レビュー者			/ /	/ /	/ /

レビュー資料	

レビュー結果	□レビュー終了　□再レビュー
総括	

No	指摘事項	対策	担当 期限	修正確認
1			/ /	/ /

指摘事項 対策実施確認	レビュー責任者コメント	終了確認 / /

（6）工数実績データ

社内要員の工数実績は、勤怠管理システムの出力を利用し、社外要員の工数は、購買管理システムの出力を利用する。複数のプロジェクト業務に関わっている要員に対しては、各プロジェクトに関わった工数が分かるように管理する。また、工数データは工程に紐付ける。

個人毎の工数情報をプロジェクトの実施工程と紐付けし、勤怠管理システム

で管理するのが理想であるが、そのような仕組みを持たない場合は、精度を追及してはいけない。

また、プロジェクトメンバ全員が同じ工程の作業を行うことが理想であるが、実際のところ、詳細設計が計画通りの時期に終了してプログラムのコーディング作業に進めるメンバと進めないメンバが居るため、一律の管理は不可能である。プロジェクトリーダやマネージャは、毎週のプロジェクト会議でメンバにプロジェクト全体で管理する工程の切れ目を伝えるとともに、作業が遅延して次工程に移れないメンバには、工程を変えないように指示する。

レビュー工数実績は、所定のフォルダで管理し、記述様式を Excel で統一したレビュー記録から収集する。

ここでのポイント

- ●工数は要員毎に収集し、工程と紐付ける。
- ●プロジェクトに関係のない作業工数を区別できるようにする。区別できない場合は、おおよその割合を把握する。
- ●精度を追及してはいけない。

◆収集する定量データ詳細と収集方法

上記に概説した収集すべき定量データと収集の仕組みを図表 3-7 に整理する。収集するデータの詳細は、ESQR（参考文献［3］）に定義されている基礎指標、「組込みソフトウェア開発データ白書 2015」（参考文献［4］）を参照されたい。

収集方法の中で「自動」としているものは、収集頻度が高く、収集のためのユーティリティツールを作成して自動化すべきことを推奨している。また自動化推奨のレベルが高くないものは「半自動」、「手動」で表している。

収集のタイミングは、プロジェクトの各工程の実施期間中毎日、または週単位など、比較的頻繁に収集すること等を表している。

図表 3-7　収集する定量データ詳細と収集タイミング

収集場所（収集先）・方法・時期 ＼ 収集するデータ		プロジェクト計画書	成果物				品質			勤怠管理情報	購買管理情報
			要求仕様書	設計書	プログラムソースコード	テスト仕様書	レビュー記録	テスト結果記録	バグ管理票・台帳		
		置き場所：プロジェクト所定の共有フォルダ								社内システム	
(1) ドキュメントボリューム（枚数、項目数）※項目数は進捗管理用	要求仕様書ボリューム（※1）										
	設計書ボリューム（※1）		●	●		●					
	テスト仕様書ボリューム（※1）										
(2) プログラムソースコード行数、静的コードチェッカー指摘件数	ソースコード行数（新規・改造）（※1）										
	ソースコード行数（全体）（※1）										
	モジュール数（再利用）										
	モジュール数（全体）										
	ファイル行数（※1）				●						
	関数の行数（※1）										
	制御文数（※1）										
	コメント行数（※1）										
	静的コードチェッカ指摘数（コーディングルール逸脱数）（※1）										
(3) テストケース数、着手数、完了数（単体テスト、結合テスト、総合テスト）	テスト項目数（※1）										
	テスト実施（着手）項目数（※1）					●		●			
	テスト完了項目数（※1）										
(4) 検出バグ数、解決数、未解決数、重複バグ数（単体テスト、結合テスト、総合テスト）	結合テスト検出バグ数（※1）										
	結合テスト解決バグ数（※1）										
	総合テスト検出バグ数（※1）										
	総合テスト解決バグ数（※1）										
	バグ深刻度ランク比率								●		
	深刻度高ランクバグ数										
	深刻度高ランクバグ解決数										
	長期未解決バグ数										
	総合テスト平均解決日数										
	重複バグ数										
(5) レビュー指摘件数、解決件数、未解決件数（仕様書、設計書作成時）	仕様書レビュー指摘件数										
	仕様書レビュー解決件数										
	設計レビュー指摘件数										
	設計レビュー解決件数						●				
	コードレビュー指摘件数										
	コードレビュー解決件数										
	テストレビュー指摘件数										
	テストレビュー解決件数										
(6) 工数（全体、各工程、レビュー工数）	開発全工数（※1）										
	仕様作成工数（※1）										
	設計作成工数（※1）									●	●
	コード作成工数（※1）										
	テスト準備・確認工数（※1）										
	テスト工数（※1）										
	全レビュー工数（※1）										
	仕様レビュー工数（※1）										
	設計レビュー工数（※1）						●				
	コードレビュー工数（※1）										
	テストレビュー工数（※1）										
(7) プロファイル情報											

収集方法	収集インターバル（目安）			収集の仕組み
	開発初期	開発中期	開発後期	
自動	週次			〈仕組み①〉 ・File System ディレクトリ構造統一して管理 ・頁数カウントツール利用 〈仕組み②〉 ・構成管理ツールにて管理 ・頁数カウントツール利用
自動	週次			
自動	週次			
自動	—	週次	日次	・File System ディレクトリ構造統一して管理、または、構成管理ツールにて管理 ・行数カウントツール利用 ・静的コードチェッカ利用
自動	—	—	日次	・記述様式を統一（Excel） ・File System ディレクトリ構造統一して管理 ・情報読み取りツールを自作
自動	—	—	日次	〈仕組み①〉 ・記述様式を統一（Excel） ・File System ディレクトリ構造統一して管理 ・情報読み取りツールを自作 〈仕組み②〉 バグ管理ツール利用
手動または半自動	実施直後			・記述様式を統一（Excel） ・File System ディレクトリ構造統一して管理 ・情報読み取りツールを自作、または手動で読み取り
	週次			
	実施直後			
	週次			
		実施直後		
		週次		
			実施直後	
			日次	
半自動				・勤怠管理システム及び購買管理システムから入手
	月次			
	月次			
		月次		
		月次		
			月次	
手動または半自動				・記述様式を統一（Excel） ・File System ディレクトリ構造統一して管理 ・情報読み取りツールを自作、または手動で読み取り
	実施直後			
	実施直後			
		実施直後		
			実施直後	

（※1）は、ESQR で定義された基礎指標をそのまま利用、または一部拡張して利用。"不具合"という用語は"バグ"に置き換えている。測定方法または計算式は利用し易い形に調整している。

3.3 収集データから導出する管理指標

前節 3.2 に紹介した定量データ収集の仕組みを構築し、収集したデータから効果的に進捗や品質の管理指標を導出する。導出する指標は、

・プロジェクトマネジメント（プロジェクト管理）

・組織改善

のどちらか、または両方で活用する。

本節では、ソフトウェア開発工程の時系列に沿って、収集データから導出する管理指標を紹介する。活用目的に応じた活用方法は第 4 章、第 5 章に詳しく紹介する。

IPA/SEC では、組込みソフトウェア開発で利用する指標を標準的な利用方法とともに「組込みソフトウェア開発作り込みガイド（ESQR）（参考文献［3］）」で解説しており、本書ではそれを参照する。

◆要求定義工程の指標

注）＊を付した指標は ESQR に定義されている

管理指標	計算方法または計算式	プロジェクト管理	組織改善
R1: 要求仕様書ドキュメントの変化量	要求仕様書ドキュメント － 要求仕様書ドキュメント（前回）	○	
R2: 要求仕様書ボリューム率 *	要求仕様書ボリューム／ ソースコード行数	○	△
R3: 要求仕様書バランス*	各パートのページ数／ 全体ページ数の総和	○	△
R4: 仕様レビュー作業充当率 *	仕様レビュー工数／ 仕様書作成工数		◎
R5: 仕様レビュー作業実施率 *	仕様レビュー工数／ ソースコード行数	○	◎

要求定義工程のアウトプットは、要求仕様書であり、これらの指標を利用すると、作業進捗のモニター、要求仕様書のレビュー工数の目安判断、ドキュメント品質の間接的な評価が行える。

作業進捗は、プロジェクトマネージャが作業項目管理表に作業実績を各担当者に記入させて、それをモニターすることが理想である。一方で PMO が客観的に行うには、ドキュメントボリュームの変化量をモニターするのがよい。

◆設計工程の指標

注）＊を付した指標は ESQR に定義されている

管理指標	計算方法または計算式	プロジェクト管理	組織改善
D1: 設計書ドキュメントの変化量	設計書ドキュメント － 設計書ドキュメント（前回）	○	
D2: 設計書ボリューム率 *	設計書ボリューム ／ ソースコード行数	○	△
D3: 設計書バランス *	各パートのページ数 ／ 全体ページ数の総和	○	△
D4: 設計書レビュー作業充当率 *	設計書レビュー工数 ／ 設計書作成工数		◎
D5: 設計レビュー実施率 *	設計レビュー工数 ／ ソースコード行数	○	◎

設計工程のアウトプットは、設計書であり、利用できる指標は、要求定義工程の指標に準ずる。

◆実装工程の指標

注）＊を付した指標は ESQR に定義されている

管理指標	計算方法または計算式	プロジェクト管理	組織改善
C1: プログラム行数の変化量	ソースコード行数 － ソースコード行数（前回）	◎	
C2: コード再利用率	（ソースコード行数（全体）－ ソースコード行数（新規・改造）） ／ソースコード行数（全体）	◎	
C3: モジュール再利用率	改造せずにそのまま再利用する モジュール数 ／ 全体のソフトウェアモジュール数	◎	
C4: ファイル行数 *	（収集値）	◎	
C5: 関数の行数 *	（収集値）	◎	
C5: 制御文記述率 *	制御文数 ／ソースコード行数		
C7: コメント行記述率 *	コメント行数 ／ソースコード行数		
C8: 静的コードチェッカ指摘密度（コーディングルール逸脱率 *）	静的コードチェッカ指摘数 ／ ソースコード行数	◎	○
C9: コードレビュー作業充当率*	コードレビュー工数 ／ コード作成工数		◎
C10: コードレビュー作業実施率*	コードレビュー工数 ／ ソースコード行数		◎

実装工程のアウトプットは、プログラムソースコードであり、これらの指標を利用すると、作業進捗のモニター、再利用状況の把握、ソースコード品質の間接的な評価、ソースコードレビュー工数や品質の目安判断が行える。

作業進捗は、プロジェクトマネージャが作業項目管理表に作業実績を各担当者に記入させて、それをモニターすることが理想である。一方で PMO が客観的に行うには、ソースコード行数の変化をモニターする。

モジュールやソースコードの再利用状況は、進捗が計画値と乖離している場合に、テスト計画を見直すための指標として使う。

ソースコードの複雑度やコーディングルールの順守状況等のコード品質を把握できれば、レビューを実施するかどうかの判断ができる。

◆テスト工程（計画フェーズ）の指標　注）＊を付した指標は ESQR に定義されている

管理指標	計算方法または計算式	プロジェクト管理	組織改善
T1: テスト仕様書ボリュームの変化量	テスト仕様書ボリューム － テスト仕様書ボリューム（前回）	○	
T2: テスト仕様書ボリューム率 *	テスト仕様書ボリューム ／ ソースコード行数		△
T3: テスト仕様書バランス *	各パートのページ数 ／ 全体ページ数の総和	○	△
T4: テスト作業充当率 *	テスト工数 ／ 開発全工数		◎
T5: テスト作業実施率 *	テスト工数 ／ ソースコード行数		◎
T6: テスト密度 *	テスト項目数 ／ ソースコード行数		◎
T7: テストレビュー作業充当率 *	テストレビュー工数 ／ テスト準備工数		◎
T8: テストレビュー作業実施率 *	テストレビュー工数 ／ ソースコード行数		◎

テスト計画フェーズのアウトプットは、テスト仕様書（テストケース）であり、これらの指標を利用すると、作業進捗のモニター、テスト仕様書のドキュメント品質（テスト計画の妥当性にも通じる）の間接的な評価、テスト仕様書レビュー工数や品質の目安判断が行える。

作業進捗は、プロジェクトマネージャが作業項目管理表に作業実績を各担当者に記入させて、それをモニターすることが理想である。一方で PMO が客観的に行うには、テスト仕様書ボリュームの変化をモニターする。

◆テスト工程（実施フェーズ前半）の指標　注）＊を付した指標はESQRに定義されている

管理指標	計算方法または計算式	プロジェクト管理	組織改善
T9: 結合テスト項目消化率	消化項目数 ／ 全体項目数	◎	
T10: 結合テスト検出バグ数の変化量	結合テスト検出バグ数 － 結合テスト検出バグ数（前回）	◎	
T11: 結合テスト解決バグ数の変化量	結合テスト解決バグ数 － 結合テスト解決バグ数	◎	
T12: 結合テスト未解決バグ数の変化量	結合テスト未解決バグ数 － 結合テスト未解決バグ数	◎	
T13: 結合テストバグ密度	結合テスト検出バグ数 ／ ソースコード行数	◎	◎
T14: 結合テストバグ検出率	結合テスト検出バグ数 ／ テスト実施（着手）項目数	◎	◎

テスト実施フェーズは、単体テストや結合テストのようなテスト前半と総合テストやシステムテストのようなテスト後半とではテストの観点が異なり、テスト後半ほどプロダクトの品質を見る指標を重要視する傾向にある。

テスト前半では、これらの指標を利用して、テスト作業の進捗モニター、テスト作業の品質評価が行える。

◆テスト工程（実施フェーズ後半）の指標　注）＊を付した指標は ESQR に定義されている

管理指標	計算方法または計算式	プロジェクト管理	組織改善
T15: 総合テスト項目消化率	消化項目数 ／ 全体項目数	◎	
T16: 総合テスト検出バグ数の変化量	総合テスト検出バグ数 － 総合テスト検出バグ数	◎	
T17: 総合テスト解決バグ数の変化量	総合テスト解決バグ数 － 総合テスト解決バグ数	◎	
T18: 総合テスト未解決バグ数の変化量	総合テスト未解決バグ数 － 総合テスト未解決バグ数	◎	
T19: 総合テストバグ密度	総合テスト検出バグ数 ／ ソースコード行数	◎	◎
T20: 総合テストバグ検出率	総合テスト検出バグ数 ／ テスト実施（着手）項目数	◎	◎
T21: バグ深刻度ランク比率	（収集値）	○	○
T22: 深刻度高ランクバグ数の変化量	深刻度高ランクバグ数 － 深刻度高ランクバグ数（前回）	○	
T23: 深刻度高ランクバグ未解決数の変化量	深刻度高ランクバグ未解決数 － 深刻度高ランクバグ未解決数（前回）	○	
T24: 深刻度高ランクバグ比率	深刻度高ランクバグ数 ／ 総合テスト検出バグ数	○	◎
T25: 重複バグ数の変化量	重複バグ数 － 重複バグ数（前回）	○	
T26: 重複バグ比率	重複バグ数 ／ 総合テスト検出バグ数	○	
T27: 長期未解決バグ数の変化量	長期未解決バグ数 － 長期未解決バグ数（前回）	○	
T28: 総合テストバグ平均解決日数	（収集値）	◎	○
T29: 総合テストバグ平均解決日数の変化量	総合テストバグ平均解決日数 － 総合テストバグ平均解決日数（前回）	◎	
T30: 不具合収束率 *	最終 10% のテスト期間のバグ発見率 ／ 当初 90% のテスト期間のバグ発見率	◎	
T31: 不具合修正率 *	解決バグ数 ／ 検出バグ数	◎	

テスト後半ではこれらの指標を用いて、テスト作業の進捗、テスト作業の品質（効率）、ソフトウェアの品質を総合的に見なければならない。

テスト前半の指標に追加された指標は、プロジェクトまたはソフトウェアプロダクトに、深刻なリスクが潜在していないかどうかを見るための指標と品質がどの程度確保されているかを見る指標である。

◆プロジェクト終了後の指標　　注）＊を付した指標は ESQR に定義されている

管理指標	計算方法または計算式	プロジェクト管理	組織改善
P1:SLOC 生産性	ソースコード行数（新規）／開発全工数	◎	◎
P2: レビュー作業充当率 *	全レビュー工数 ／ 開発全工数		○
P3: レビュー作業実施率 *	全レビュー工数 ／ ソースコード行数		○

組込みソフトウェア開発では生産性を表す指標として、SLOC 生産性を用いる。SLOC 生産性は、１人の技術者が単位期間（一般的には１ヶ月）にどれだけのソフトウェアを開発することが可能かを表す指標で、開発量は、ソースコードの行数（SLOC：Source Line Of Code）で表す。ソースコード行数の範囲は、一般に新規作成部分と母体を改造した部分を対象とし、母体のうちそのまま再利用する部分は含めない。改良開発や派生開発の場合は、母体から不要な部分を削除するための調査工数が必要であるため、ソースコード行数の範囲に削除した行数を加える考え方を取る場合もある。開発作業の範囲は、組織の作業形態に応じて決める。例えば、外部から与えられたシステム要求仕様書を元にソフトウェア要求仕様書の作成から作業を開始する場合は、ソフトウェア要求定義からソフトウェア総合テスト完了までを SLOC 生産性の作業範囲とする。「組込みソフトウェア開発データ白書 2015」では、ソースコード行数の範囲を新規作成部分＋改造部分＋削除部分と定義しており、作業範囲は、開発５工程（ソフトウェアアーキテクチャ設計から、ソフトウェア詳細設計、実装及び単体テスト、ソフトウェア結合テスト、ソフトウェア総合テストまで）を範囲としている。

以上、ソフトウェア開発工程の時系列に沿って、収集データから導出する管理指標を紹介した。図表 3-8 に導出する管理指標一覧と、導出元の収集データとの関係を示す。

図表 3-8 には、指標活用の用途：

・進捗のモニター

・作業量や品質の目安

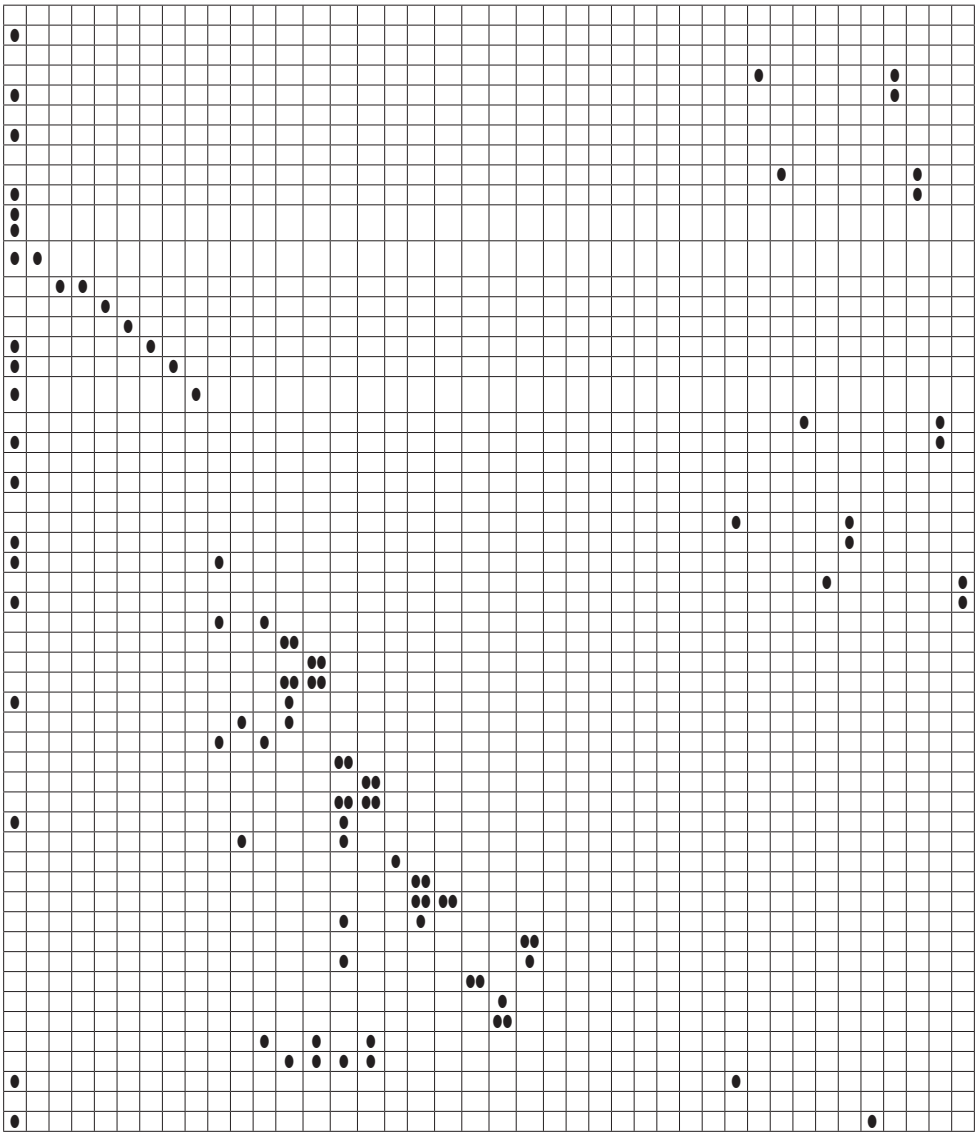
・他プロジェクトと比較するために測定結果を評価するもの

も示している。

図表 3-8　収集データから導出する管理指標

注）＊を付した指標は ESQR に定義されている

分類（時系列）	管理指標	計算方法または計算式	単位	用途	プロジェクト管理	組織改善	(I)ドキュメントボリューム 要求仕様書ボリューム*	設計書ボリューム	テスト仕様書ボリューム*
要求定義	R1: 要求仕様書ドキュメントの変化量	要求仕様書ドキュメント−要求仕様書ドキュメント(前回)	頁数	モニタ	○		●●		
	R2: 要求仕様書ボリューム率*	要求仕様書ボリューム/ソースコード行数	頁数/KLOC	目安・結果評価	○	△	●		
	R3: 要求仕様書バランス*	各パートのページ数/全体ページ数の総和	配分比	目安・結果評価	○	△	●		
	R4: 仕様レビュー作業充当率*	仕様レビュー工数/仕様書作成工数	%	目安・結果評価		◎			
	R5: 仕様レビュー作業実施率*	仕様レビュー工数/ソースコード行数	人時/KLOC	目安・結果評価	○	◎			
設計	D1: 設計書ドキュメントの変化量	設計書ドキュメント−設計書ドキュメント(前回)	頁数	モニタ	○			●●	
	D2: 設計書ボリューム率*	設計書ボリューム/ソースコード行数	頁数/KLOC	目安・結果評価	○	△		●	
	D3: 設計書バランス*	各パートのページ数/全体ページ数の総和	配分比	目安・結果評価	○	△		●	
	D4: 設計書レビュー作業充当率*	設計書レビュー工数/設計書作成工数	%	目安・結果評価		◎			
	D5: 設計レビュー実施率*	設計レビュー工数/ソースコード行数	人時/KLOC	目安・結果評価	○	◎			
コーディング	C1: プログラム行数の変化量	ソースコード行数−ソースコード行数(前回)	KLOC	モニタ	◎				
	C2: コード再利用率	(ソースコード行数全体)−ソースコード行数(新規・改造))/ソースコード行数全体)	%	モニタ	◎				
	C3: モジュール再利用率	改造せずにそのまま再利用するモジュール数/全体のソフトウェアモジュール数	%	モニタ	◎				
	C4: ファイル行数*	(収集値)	KLOC	モニタ	◎				
	C5: 関数の行数*	(収集値)	LOC	モニタ	◎				
	C5: 制御文記述率*	制御文数/ソースコード行数	%	モニタ					
	C7: コメント行記述率*	コメント行数/ソースコード行数	%	モニタ					
	C8: 静的コードチェッカ指摘密度（コーディングルール逸脱率*）	静的コードチェッカ指摘数/ソースコード行数	箇所/KLOC	目安・モニタ	◎	○			
	C9: コードレビュー作業充当率*	コードレビュー工数/コード作成工数	%	目安・結果評価		◎			
	C10: コードレビュー作業実施率*	コードレビュー工数/ソースコード行数	人時/KLOC	目安・結果評価		◎			
テスト準備	T1: テスト仕様書ボリュームの変化量	テスト仕様書ボリューム−テスト仕様書ボリューム(前回)	頁数	モニタ	○				●●
	T2: テスト仕様書ボリューム率*	テスト仕様書ボリューム/ソースコード行数	Page/KLOC	目安・結果評価		△			●
	T3: テスト仕様書バランス*	各パートのページ数/全体ページ数の総和	配分比	目安・結果評価	○	△			●
	T4: テスト作業充当率*	テスト工数/開発全工数	%	目安・結果評価		◎			
	T5: テスト作業実施率*	テスト工数/ソースコード行数	人時/KLOC	目安・結果評価		◎			
	T6: テスト密度*	テスト項目数/ソースコード行数	項目/KLOC	目安・結果評価		◎			
	T7: テストレビュー作業充当率*	テストレビュー工数/テスト準備工数	%	目安・結果評価		◎			
	T8: テストレビュー作業実施率*	テストレビュー工数/ソースコード行数	人時/KLOC	目安・結果評価		◎			
テスト前半	T9: 結合テスト項目消化率	消化項目数/全体項目数	%	モニタ	◎				
	T10: 結合テスト検出バグ数の変化量	結合テスト検出バグ数−結合テスト検出バグ数(前回)	件数	モニタ	◎				
	T11: 結合テスト解決バグ数の変化量	結合テスト解決バグ数−結合テスト解決バグ数	件数	モニタ	◎				
	T12: 結合テスト未解決バグ数の変化量	結合テスト未解決バグ数−結合テスト未解決バグ数	件数	モニタ	◎				
	T13: 結合テストバグ密度	結合テスト検出バグ数/ソースコード行数	件数/KLOC	目安・モニタ	◎	◎			
	T14: 結合テストバグ検出率	結合テスト検出バグ数/テスト実施着手項目数	件数/項目数	目安・モニタ	◎	◎			
テスト後半	T15: 総合テスト項目消化率	消化項目数/全体項目数	%	モニタ	◎				
	T16: 総合テスト検出バグ数の変化量	総合テスト検出バグ数−総合テスト検出バグ数	件数	モニタ	◎				
	T17: 総合テスト解決バグ数の変化量	総合テスト解決バグ数−総合テスト解決バグ数	件数	モニタ	◎				
	T18: 総合テスト未解決バグ数の変化量	総合テスト未解決バグ数−総合テスト未解決バグ数	件数	モニタ	◎				
	T19: 総合テストバグ密度	総合テスト検出バグ数/ソースコード行数	件数/KLOC	目安・モニタ・結果評価	◎	◎			
	T20: 総合テストバグ検出率	総合テスト検出バグ数/テスト実施着手)項目数	件数/項目数	目安・モニタ・結果評価	◎	◎			
	T21: バグ深刻度ランク比率	(収集値)	比率	モニタ	○	○			
	T22: 深刻度高ランクバグ数の変化量	深刻度高ランクバグ数−深刻度高ランクバグ数(前回)	件数	モニタ	○				
	T23: 深刻度高ランクバグ未解決数の変化量	深刻度高ランクバグ未解決数−深刻度高ランクバグ未解決数(前回)	件数	モニタ	○				
	T24: 深刻度高ランクバグ比率	深刻度高ランクバグ数/総合テスト検出バグ数	%	目安・モニタ・結果評価	○	◎			
	T25: 重複バグ数の変化量	重複バグ数−重複バグ数(前回)	件数	モニタ	○				
	T26: 重複バグ比率	重複バグ数/総合テスト検出バグ数	%	モニタ	○				
	T27: 長期未解決バグ数の変化量	長期未解決バグ数−長期未解決バグ数(前回)	件数	モニタ	○				
	T28: 総合テストバグ平均解決日数	(収集値)	日	目安・モニタ・結果評価	◎	○			
	T29: 総合テストバグ平均解決日数の変化 量	総合テストバグ平均解決日数−総合テストバグ平均解決日数(前回)	日	モニタ	◎				
	T30: 不具合収束率*	最終10%のテスト期間のバグ発見率/当初90%のテスト期間のバグ発見率	比率	モニタ	◎				
	T31: 不具合修正率*	解決バグ数/検出バグ数	%	モニタ	◎				
	P1: SLOC生産性	ソースコード行数(新規)/開発全工数	KLOC/人月	目安・モニタ・結果評価	◎	◎			
	P2: レビュー作業充当率*	全レビュー工数/開発全工数	%	目安・結果評価		○			
	P3: レビュー作業実施率*	全レビュー工数/ソースコード行数	人時/KLOC	目安・結果評価		○			


(2)プログラムソースコード行数、静的コードチェッカー指摘件数
(3)テストケース数、実施数、完了数
(4)検出バグ現象数、解決数、検出バグ原因数
(5)レビュー指摘件数、解決件数（仕様書、設計書作成時）
(6)工数（全体、各工程、レビュー工数）
ソースコード行数（新規・改造）*
ソースコード行数（全体）*
モジュール数（再利用）
モジュール数（全体）
ファイル行数*
関数の行数*
制御文数*
コメント行数*
静的コードチェック指摘数
テスト項目数*
テスト実施（着手）項目数
テスト完了項目数*
結合テスト検出バグ数*
結合テスト解決バグ数*
総合テスト検出バグ数*
総合テスト解決バグ数*
バグ深刻度ランクバグ比率
深刻度高ランクバグ解決数
深刻度高ランクバグ解決数
長期未解決バグ数
総合テスト平均解決日数
重複バグ数
仕様書レビュー指摘件数
仕様書レビュー解決件数
設計レビュー指摘件数
設計レビュー解決件数
コードレビュー指摘件数
コードレビュー解決件数
テストレビュー指摘件数
テストレビュー解決件数
開発全工数*
仕様作成工数
設計作成工数*
コード作成工数*
テスト準備・確認工数*
テスト工数*
全レビュー工数*
仕様レビュー工数*
設計レビュー工数*
コードレビュー工数*
テストレビュー工数*

第4章

4. プロジェクトマネジメントにおける定量管理

プロジェクトリーダは、定量データを活用して進行しているプロジェクトの状況を常にモニターし、計画通りの進捗（コスト、品質、納期）が見込めないと判断した時点で、WBS や要員アサインの変更等、迅速なプロジェクトコントロールを行う。また、PMO（Project Management Office）は定量データを活用してプロジェクトの進行状況を定期的に経営者に報告する。経営者は、常時マクロ指標を監視し、経営判断が必要か否かを迅速に判断して、必要な場合はプロジェクトに関係する部門に対して指示を出す。

プロジェクトマネジメントにおける定量管理が提唱される以前は、プロジェクトの運行はベテラン社員の独自の勘など属人的な要素に頼る部分が大きかった。定量データと体系だったプロジェクトマネジメントの手法を使用することで作業の標準化が可能になり、プロジェクトの成果が高まることが期待されている。

本章では、プロジェクトの現場で日々の管理を定量的に行う方法を示し、定量管理を行うことでリーダの管理力が強くなることを示す。

4.1 プロジェクトマネジメントの目的と役割

プロジェクトマネジメントの目的は、プロジェクトを成功させること、つまり「プロジェクトを予定した期限内に、予算金額内で、期待レベルの技術成果の下、割り当て資源を有効活用して、関係者が満足する状態で完了させること」ということが出来る。

プロジェクトマネジメントでは計画（Plan）、実行（Do）、チェック（Check）、是正（Act）という管理サイクル（PDCA サイクル）が常に稼動している必要がある。

ここでは、定量データ及び定量データの分析から得られた指標データを用いることにより効果が得られる作業に絞って説明する。

4.1.1 計画作成 (Plan)

計画段階では、組織が継続して定量データを分析することによって得られた指標データを用いて以下の項目を策定する。

① 必要なリソースの見積

組織の現状の生産性の指標値を用いて開発に必要なリソースを見積もる。（開発規模の見積もりを別途行ったうえで、生産性を用いる）

② プロジェクトメンバへの作業の割り振り

①の見積もり結果とプロジェクトメンバの能力を勘案して作業を割り振る。

③ 管理指標（進捗、品質、コスト、生産性）の達成目標と管理限界値の設定

当該プロジェクトの管理指標の目標値と管理限界値（上限値、下限値）を組織として積み重ねた過去の定量データをベースにプロジェクト毎にカスタマイズし、改善目標も含めて決定する。

以下に、管理限界値を設定する時の注意点を示す。

- 新規開発と改造は分けて管理する

  改造の結合テストのテスト密度のレンジが新規開発と比較して高くなる場合がある。これは新規開発と改造では同一の開発規模であっても改造の方がデグレードを考慮してテスト範囲を広くするためで、改造量が多い場合は開発規模に対するテスト密度を相対的に高く設定したほうが良い。

- テスト密度を下げる時は限度を持たせる

  計画時点でテスト密度を下げると工数が削減できるので、開発委託先が経験を積んでくるとテスト密度を下げたくなるがことがある。しかし、コストの削減効果はあるものの品質確保とのトレードオフがあるので、一定の限度を設定する。

- 総合試験のテスト密度は、前工程迄の品質状況や難易度を勘案し、総合的に決定する。

- バグ密度はよほどの理由がない限り、目標値を下げない。既存システム自身のバグは別管理とする。

  開発対象が既存システムの拡張である場合、テスト工程で既存システムの欠陥（潜在バグ）を発見することがある。このような場合、当該開発の管理限界値とは別に、管理限界値を別途定めた方がよい。

- 管理限界値は、一般的に目標値の ± 20%以下に設定しているところが多い。

4.1.2 管理指標のモニター（Check）

設計からシステムテストまでの工程ごとに設定された管理指標の現状値が目標値の管理限界値の範囲内に収まっているか否か（計画とのずれを把握）をモニターする。図表 4-1 は、3.2 節で紹介した各管理指標とモニターする工程を示している。すべての指標をモニター対象にすることに負荷を感じる場合は、必須管理指標（◎印）、任意管理指標（○印）のように優先順位を付けることを推奨する。

結果指標は、対応する工程（⇒印）では目安として活用し、プロジェクト終了後にモニターし、組織目標値の改善に利用する。

また、プロジェクト全体を通して、開発中の製品の最終損益（生涯損益）表をモニターするために製品の総コストを把握する。そのために過去に投資したＲ＆Ｄ費用を含めた総コストを把握し、最終的なコスト見込みを把握する。

4.1.3 プロジェクトコントロール（Act）

前項 4.1.2 で設定された管理指標の現状値が目標値の管理限界値の範囲内に収まっているか否か（計画とのずれを把握）をモニターした結果、管理限界値を超えたズレが確認された場合に、原因を確認した上で作業追加、リソース追加、リソース再配置、仕様変更（削除）、体制変更等の対策を実行し再度 PDCA を回す。

対策の内容によっては、経営層への説明と了解を取り付ける。

図表 4-1　管理指標

管理指標＼工程	要求定義	設計	コーディング及び単体テスト	結合テスト	総合テスト	結果指標
R1: 要求仕様書ドキュメントの変化量	○					
R2: 要求仕様書ボリューム率 *	⇒					○
R3: 要求仕様書バランス *	⇒					○
R4: 仕様レビュー作業充当率 *	○					
R5: 仕様レビュー作業実施率 *	⇒					○
D1: 設計書ドキュメントの変化量	○					
D2: 設計書ボリューム率 *		⇒				○
D3: 設計書バランス *		⇒				○
D4: 設計書レビュー作業充当率 *		○				
D5: 設計レビュー実施率 *		⇒				○
C1: プログラム行数の変化量				◎		
C2: コード再利用率			◎			○
C3: モジュール再利用率			◎			○
C4: ファイル行数 *				○		
C5: 関数の行数 *				○		
C5: 制御文記述率 *				◎		
C7: コメント行記述率 *				○		
C8: 静的コードチェッカ指摘密度				◎		
C9: コードレビュー作業充当率 *			○			
C10: コードレビュー作業実施率 *				◎		○
T1: テスト仕様書ボリュームの変化量				⇒		○
T2: テスト仕様書ボリューム率 *				⇒		○
T3: テスト仕様書バランス *				⇒		○
T4: テスト作業充当率 *				○		
T5: テスト作業実施率 *				○		
T6: テスト密度 *				◎	◎	
T7: テストレビュー作業充当率 *				○		
T8: テストレビュー作業実施率 *				⇒		○
T9: 結合テスト項目消化率				◎		
T10: 結合テスト検出バグ数の変化量				◎		
T11: 結合テスト解決バグ数の変化量				◎		
T12: 結合テスト未解決バグ数の変化量				◎		
T13: 結合テストバグ密度				◎		
T14: 結合テストバグ検出率				◎		○
T15: 総合テスト項目消化率					◎	
T16: 総合テスト検出バグ数の変化量					◎	
T17: 総合テスト解決バグ数の変化量					◎	
T18: 総合テスト未解決バグ数の変化量					◎	
T19: 総合テストバグ密度					◎	○
T20: 総合テストバグ検出率					◎	○
T21: バグ深刻度ランク比率					○	
T22: 深刻度高ランクバグ数の変化量					○	
T23: 深刻度高ランクバグ未解決数の変化量					○	
T24: 深刻度高ランクバグ比率					○	
T25: 重複バグ数の変化量					○	
T26: 重複バグ比率					○	
T27: 長期未解決バグ数の変化量					○	
T28: 総合テストバグ平均解決日数					◎	○
T29: 総合テストバグ平均解決日数の変化量					◎	
T30: 不具合収束率 *					◎	
T31: 不具合修正率 *					◎	
P1:SLOC 生産性	⇒	⇒		◎		
P2: レビュー作業充当率 *						○
P3: レビュー作業実施率 *						○

4.2 プロジェクトリーダの業務

プロジェクトリーダは、定量データを用いて進行しているプロジェクトの状況を常にモニターし、計画通りの進捗が見込めないと判断した時点で、要員の再配置、再スケジューリングなどにより、適切かつ迅速にプロジェクトをコントロールする。ここでは、プロジェクトリーダが定量データを活用して、どのような指標をどのような書式でプロジェクトをモニターし、どのようなコントロールを行うかを紹介する。

4.2.1 モニターとコントロール

Q（品質）、C（予算）、D（納期）とリスクの４つの観点から関係する指標をモニターすることにより目標値との差異が分かる。差異が管理限界値を超えている作業に対してアクション（対策）を起こすことでプロジェクトを計画に近づけるようにコントロールする。４つの観点について関係する指標とその指標から何が判断（推測）出来るか、またどのようなアクションが取りうるかについて解説する。

（1）納期（進捗）

上流工程（要求定義や設計）での納期遅延が何か月起こりそうかといった定量的な判断をすることは難しいが、下流工程（コーディング作業やテスト）では、幾つかの指標をモニターすることで定量的に納期遅延の警告を発することができる。ここでは、指標を適用する視点として、「機能の作り込み」と「バグ修正」を取り上げ、活用可能な指標とその指標から判断できる効果を説明する。

◆納期遅延傾向をモニターする視点：(a) 機能の作り込み

指　標	効　果
1. ソースコード量の変化	未着手・未完了判断が可能
【指標のモニター状況と判断】 i　再利用をしない場合で、ソースコード量が０の時、開始予定日に拘わらず未着手 ii　再利用をしない場合で、ソースコード量が変動しているとき、終了予定（報告）に拘わらず完了する見込みがない iii　再利用をする場合で、ソースコード量がもとのコード量のままの時、開始予定に拘わらず未着手 iv　再利用をする場合で、ソースコード量が変動している時、終了予定（報告）に拘わらず完了する見込みがない	

◆納期遅延傾向をモニターする視点：(b) バグの修正

指　標	効　果
1. 検出バグ数の変化量	収束可能性判断が可能
【指標のモニター状況と判断】 i　検出バグ数の変化量より増加率がゼロに近づいている場合、収束すると判断できる〈図表 4-3(a)〉 ii　増加率がプラスのままでなかなか収斂していかない場合、想定以上のバグがあるため大幅な遅延が予想できる。最悪は実現できないリスクもある〈図表 4-3(b)〉 iii　増加率が低い状態が続いている場合、テスト（評価）が遅れているか、テスト仕様が不十分で再度テスト設計とテストが必要か、機能の作り込みが遅れているためにテストが遅れている（できていない）可能性がある〈図表 4-3(c)〉	

指　標	効　果
2. 未解決バグ数の変化量	収束可能性判断が可能
【指標のモニター状況と判断】 i　未解決バグ数の変化量が増加のままで減少に転じない場合、想定以上のバグがあるか、バグの内容が難しくて修正が進まないため大幅に遅延する可能性がある〈図表 4-4(a)〉 ii　一定の範囲に留まったままで減らない場合、原因特定が困難なバグか、修正してもその影響でバグが出続けている（もぐら叩き）状態で修正不可能なソフトウェアである可能性が高い〈図表 4-4(b)〉 iii　減少傾向が続いている場合、グラフの延伸が未解決バグ０に到達する時期に終了することが予測できる〈図表 4-4(c)〉	

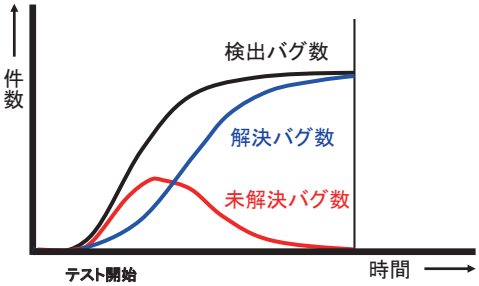
指　標	効　果
3. 解決バグ数の変化量	収束時期の予測可能
【指標のモニター状況と判断】 i　検出バグ数の増加率がほぼゼロ状態にきている場合、（未解決バグ数／解決バグ数の 1 日あたりの平均件数）が残り必要日数となる〈図表 4-5(a)〉 ii　検出バグ数の増加率がプラスの場合、（検出バグ数の期間平均－解決バグ数の期間平均）が、プラスであれば終了予測困難、マイナスであれば（未解決バグ数／解決バグ数の 1 日あたりの平均件数）が残り必要日数となる（超概算）〈図表 4-5(b)〉	

指　標	効　果
4. 納期（予定）遅延数の変化量	収束時期の予測可能
【指標のモニター状況と判断】 i　納期遅延数の増加率がプラスの場合、収束時期の遅延が拡大している ii　納期遅延数の増加率がマイナスの場合、収束時期は（納期遅延数）／（検出バグ数の期間平均－解決バグ数の期間平均）の遅延となる	

指　標	効　果
5. 平均解決日数とその変化量	収束時期の予測可能
【指標のモニター状況と判断】 i　平均解決日数が縮小している場合、（未解決バグ数／平均解決日数）日が必要解決日数となる ii　平均解決日数が増加している場合、遅延が拡大している〈図表 4-6、図表 4-7〉 iii　平均解決日数が上限値を上回っている場合、解決困難なバグが増加している可能性がある	

図表 4-2 ～図表 4-5 は、バグ件数による進捗管理の考え方を示したものである。(1) はバグ曲線の基本構成要素を示したものである。検出バグ数、解決バグ数、未解決バグ数の累計をテスト開始からの時間経過とともに示している。(2) は、様々なバグ曲線の形状から判断のポイントと判断内容を示している。①は、検出バグ数の変化から判断出来るパターンを、②は、未解決バグ数の変化から判断出来るパターンを、③は、解決バグ数の変化ら判断出来るパターンを示している。

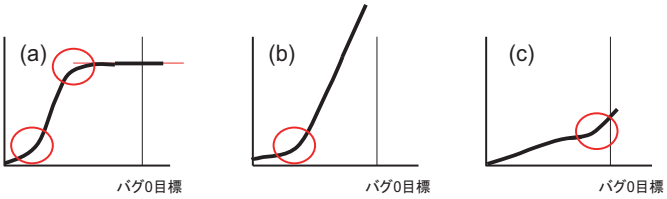
(1) バグ曲線の構成



図表 4-2　バグ曲線による進捗判断の例（バグ曲線の構成）

(2) 各指標の視点

① 検出バグ数の変化



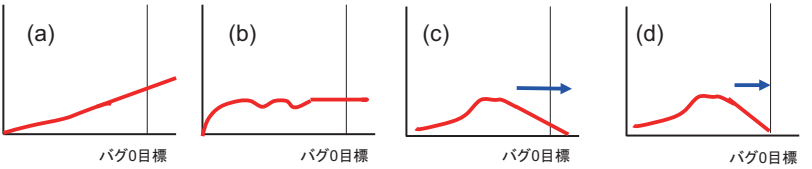
ⅰ　増加率がゼロに近づいている。
（テスト継続していても増加なし）
⇒バグ数は、収束と判断可能

ⅱ　増加率がプラスのまま。
⇒大幅な遅延の可能性
⇒実現できない

ⅲ　増加率が低い状態。
⇒機能の作り込み遅れ
⇒テスト（評価）が遅れ
⇒テスト仕様が不十分
⇒品質が良い？？？

図表 4-3　バグ曲線による進捗判断の例（検出バグ数の変化）

② 未解決バグ数の変化



ⅰ 増加のままで減少に転じない。
⇒想定以上の不具合
⇒バグの修正が難しくて進まない
⇒大幅な遅延の可能性

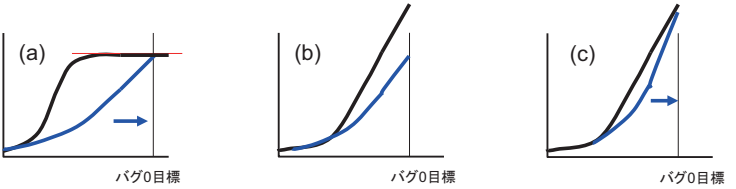
ⅱ 一定の範囲に留まっている。
⇒原因特定が困難なバグ
⇒修正の影響でバグが出続ける
（もぐら叩き）

ⅲ 減少傾向が続いている。
⇒グラフの延伸が未解決バグ0に到達
⇒未解決バグは収束するが、予定遅れ

ⅳ 減少傾向が続いている。
⇒グラフの延伸が未解決バグ0に到達
⇒未解決バグは、収束

図表 4-4　バグ曲線による進捗判断の例（未解決バグ数の変化）

③ 解決バグ数の変化



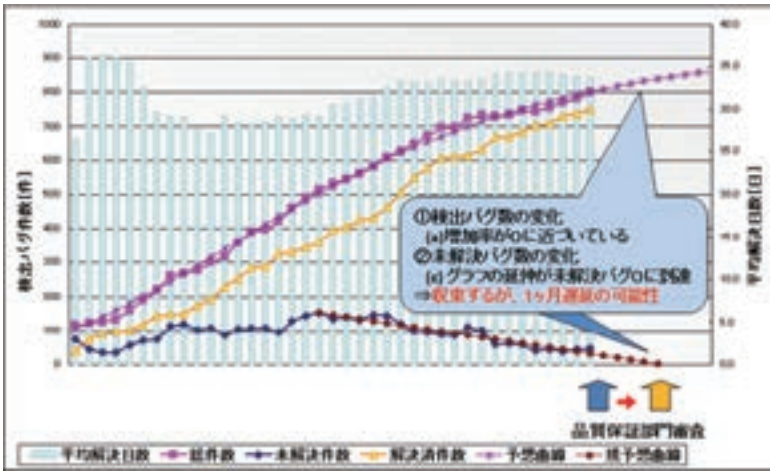
ⅰ 検出バグ数の増加率ほぼゼロ
⇒「未解決バグ数／解決バグ数1日平均」
が残り必要日数

ⅱ-1 検出バグ数の増加率プラス
⇒「検出バグ数の期間平均－
解決バグ数の期間平均」が、
プラスであれば終了予測困難

ⅱ-2 検出バグ数の増加率プラス
⇒「検出バグ数の期間平均－
解決バグ数の期間平均」が、
マイナスであればグラフ延長上で不具
合0となるが、検出バグ数の増加の可
能性あるため、開発は継続。

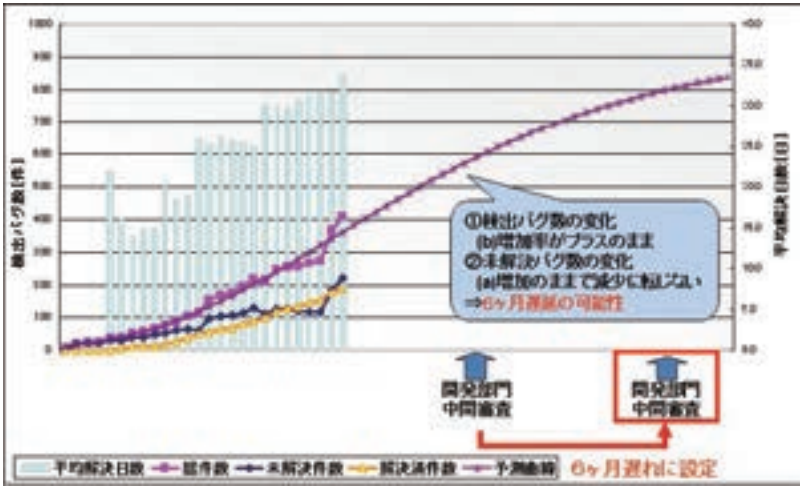
図表 4-5　バグ曲線による進捗判断の例（解決バグ数の変化）

図表 4-6 は、システムテスト工程におけるバグ管理のためにバグ曲線と平均解決日数を重ねた例である。バグ曲線の収束点、未解決バグ数の予測から品質保証部門審査予定日には間に合わないことと 1 ヶ月程度の遅れが推測できる。



図表 4-6　バグ管理表の例 1

図表 4-7 も図表 4-6 と同様の管理表であるが、この例では、全く収束する気配がなく、このままでは計画値に対して半年の遅れが推定される。早急な対策が必要である。



図表 4-7　バグ管理表の例 2

以上の様に、幾つかの指標を活用して定量的に納期遅延を判断した場合、納期遅延を回避するために何らかのアクションをとる必要がある。そのような状況の下、ソフトウェア開発現場で取られているアクション（対策）とリスクを列挙する。

◆納期遅延を回避するアクション

アクション		リスク
1. リソース追加	弱点部分（チーム、WBS）に投入	費用の追加を伴う
2. リソース入替え	・担当者に負荷がかかりすぎている場合は、担当パートの一部を他の担当者に移す ・担当者のスキルが低い場合は、スキルの高い別担当者と入れ替える	一時的にスローダウンする
3. リソース強化	エキスパート投入し、支援・指導に当たらせる指導に	費用の追加が発生する場合がある
4. WBS 削減	機能を落とす	顧客又は商品企画部門と交渉が必要である
	標準プロセスの一部を省き、作業をショートカットする	品質低下による納期遅延・費用追加リスクがある
5. WBS 入替え	WBS の順序変更（含先送り）を行う	全体としてスケジュール不整合の発生リスクがある
		後々品質低下による納期遅延・費用追加リスクがある
6. リスケジュール	マイルストーンを変更する	全体としてスケジュール不整合の発生リスクがある
		結局遅延するリスクがある
7. 作り直し	別リソースを投入して当該部分（モジュール、コンポーネント）を再開発する	追加費用を伴う
		全体としてスケジュール不整合の発生リスクがある
		継続する場合との納期遅延・追加費用リスクの比較必要で、作り直しの判断時期を誤ることが多い

(2) 予算

上流工程（要求定義や設計）でコストの増加見込みを定量的に判断することは難しいが、下流工程（コーディング作業やテスト）下流工程（コーディング作業やテスト）では、幾つかの指標をモニターすることで定量的にコスト増加の警告を発することができる。ここでは、指標を適用する視点として、「機能の作り込み」と「バグ修正」を取り上げ、活用可能な指標とその指標から判断できる効果を説明する。

◆コスト増加傾向をモニターする視点：(a) 機能の作り込み

指　標	効　果
1. モジュール再利用率	対予算差異の判断が可能
【指標のモニター状況と判断】 i　モジュールの再利用率（全く変更しないモジュール）が、想定より高い場合、開発量が少なくなりその分の開発とテスト、バグ修正のための工数が減る ii　モジュールの再利用率（全く変更しないモジュール）が、想定より低い場合、開発量が多くなりその分の開発とテスト、バグ修正のための工数が増える	

◆コスト増加傾向をモニターする視点：(b) バグの修正

指　標	効　果
1. 検出バグ数とその変化量	超過発生の予測が可能
【指標のモニター状況と判断】 i　バグ予想曲線で算出されたバグ数が想定（上限値）以上の場合、バグ数が超える分だけの工数超過が予測できる（概算 5 ～ 20 万円／件　或いは、平均解決日数分の工数／件で算出できる）＜図表 4-6、4-7 参考＞ ii　増加率が低い状態が続いている場合で、 ・テスト（評価）が遅れている時、遅れ分の評価工数が超過 ・テスト仕様が不十分で再度テスト設計とテストが必要な時、追加テスト設計・テスト実施工数が超過する ・機能の作り込みが遅れているためにテストが遅れている（できていない）時、機能の作り込みとテスト手待ち工数が超過する	

指　標	効　果
2. 未解決バグ数の変化量	超過費用予測が可能
【指標のモニター状況と判断】 i　一定の範囲に留まったままで減らない場合で ・原因特定が困難なバグの時、解決のための専門チーム設置費用が追加でかかる ・修正してもその影響でバグが出続けている（もぐら叩き状態で修正不可能なソフトウェアである可能性が高い）時、該当モジュール（コンポーネント）の再開発費用が追加でかかる	

指　標	効　果
3. 解決バグ数の変化量	超過費用予測が可能
【指標のモニター状況と判断】 i　検出バグ数の増加率がほぼゼロ状態にきている場合、（未解決バグ数／解決バグ数の１日あたりの平均件数）が残り必要日数で予定を超過していれば、超過期間の PJ 維持費用が予算を超過する ii　検出バグ数の増加率がプラスの場合で、（検出バグ数の期間平均－解決バグ数の期間平均）が、マイナスであれば（未解決バグ数／解決バグ数の１日あたりの平均件数）が残り必要日数（超概算）であり、予定を超過していれば、超過期間の PJ 維持費用分だけ予算が超過する。プラスなら PJ の終了は予測困難となる	

指　標	効　果
4. 納期（予定）遅延数の変化	超過費用予測が可能
【指標のモニター状況と判断】 i　納期遅延数の増加率がマイナスの場合、（納期遅延数）／（検出バグ数の期間平均－解決バグ数の期間平均）の遅延期間分だけ PJ を維持する超過費用が発生する	

以上の様に、幾つかの指標を活用して定量的にコスト超過を判断した場合、コスト超過を回避するために何らかのアクションをとる必要がある。そのような状況の下、ソフトウェア開発現場で取られているアクション（対策）とリスクを列挙する。

◆コスト増加傾向を回避するアクション

アクション		リスク
1. リソース入替え	・担当者に負荷がかかりすぎている場合は、担当パートの一部を他の担当者に移す ・担当者のスキルが低い場合は、スキルの高い別担当者と入れ替える	一時的にスローダウンする
2. リソース強化	エキスパート投入し、支援・指導に当たらせる	費用の追加が発生するリスクがあるため対策しない場合の超過費用との比較が必要
3. WBS 削減	機能を落とす	対顧客交渉必要
	標準プロセスの一部を省き、作業をショートカットする	品質低下による対策費用追加のリスクがある
4. WBS 入替え	WBS の順序変更（含先送り）を行う	全体としてスケジュール不整合の発生リスクがある
		後々品質低下による納期遅延・費用追加リスクがある
5. 作り直し	別リソースを投入して当該部分（モジュール、コンポーネント）を再開発する	追加費用を伴う
		全体としてスケジュール不整合の発生リスクがある
		継続する場合との納期遅延・追加費用リスクの比較必要で、作り直しの判断時期を誤ることが多い

前出の図表 4-6 の例では、１ヶ月程度のスケジュールの遅延が推測できることから、追加費用の発生が見込まれる。追加費用は、内製の場合は１ヶ月分の労務費、外部への業務委託の場合は一ヶ月分の外注費または未解決バグ件数 x（概算５～ 20 万円／件　或いは、平均解決日数分の工数／件）となる。

図表 4-7 の例では、半年と大幅なスケジュール遅延が推測出来ることから追加発生費用も大きく膨らむ可能性がある。この例の場合は、機能の縮小も視野に入れた対策が必要となる。

(3) 品質

上流工程（要求定義や設計）でソフトウェアの品質低下を定量的に判断することは難しいが、下流工程（コーディング作業やテスト）では、幾つかの指標をモニターすることで定量的に品質低下の警告を発することができる。ここでは、指標を適用する視点として、「機能の作り込み」と「バグ修正」を取り上げ、活用可能な指標とその指標から判断できる効果を説明する。

◆品質低下をモニターする視点：(a) 機能の作り込み

指　標	効　果
1. モジュール再利用率	品質向上／低下の可能性の判断が可能
【指標のモニター状況と判断】 ⅰ　想定値より高い場合は、その分のバグ数とバグ密度の改善が期待できる。逆の場合は品質の低下が予想される	

◆品質低下をモニターする視点：(b) ソースコード

指　標	効　果
1. 静的コードチェッカ重要（危険、複雑）指摘数と指摘密度	品質低下の可能性の判断が可能
【指標のモニター状況と判断】 ⅰ　指摘密度が高い場合、未解決バグの率も高くなる傾向が強い。設計仕様にも誤りがある可能性が高くなる。	

◆品質低下をモニターする視点：(c) バグの修正

指　標	効　果
1. 検出バグ数とその変化量	品質低下の可能性とその程度の判断が可能
【指標のモニター状況と判断】 ⅰ　実績データをもとに算出したバグ予想曲線から最終検出バグ数の予測が可能。予測との対比で改善／悪化の判断が可能。 （テスト進捗、テスト仕様の出来不出来、組込み遅れ等の影響の考慮も必要である）	

指　標	効　果
2. 未解決バグ数の変化量	品質低下の可能性とその程度の判断が可能
【指標のモニター状況と判断】 ⅰ　一定の範囲に留まったままで減らない場合で ・原因特定や対策が難しいバグの時、難度の高い潜在（市場への流出の可能性）バグが多く品質が低いことが予想できる。 ・修正してもその影響でバグが出続けている（もぐら叩き状態）時、修正不可能な品質である可能性が高い。	

指　標	効　果
3. バグ密度	品質の定量的把握と残存リスクの予測が可能

【指標のモニター状況と判断】

i　検出バグ数の増加率がほぼゼロになっている場合、最終バグ密度と見做せる。最終バグ密度が 1.5［件 /KSLOC］以下は良好、3［件 /KSLOC］以上は潜在バグが多い可能性が大。

ii　検出バグ数の増加率がプラスの場合、テストの進捗と共にバグ密度は上昇（品質は悪化）していく。

iii　バグ密度が下限値を下回っている場合、テストが不十分の可能性がある。

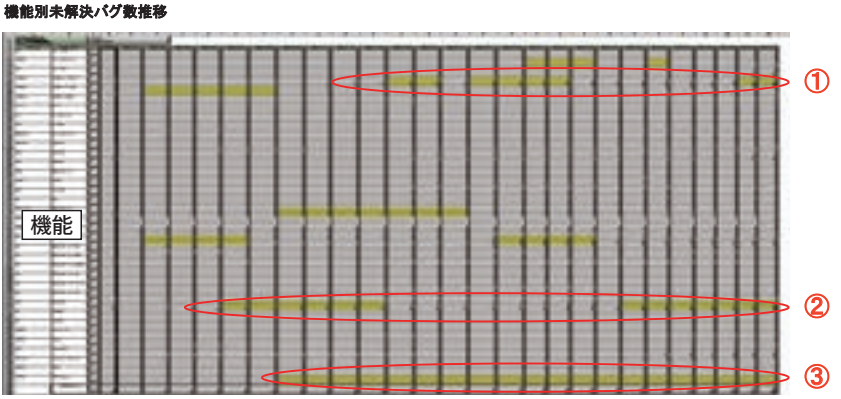
iv　バグ密度が上限値を上回っている場合、モグラ叩き状態に陥っている可能性がある。

指　標	効　果
4. バグ深刻度ランク比率	品質の定量的把握と残存リスクの予測が可能

【指標のモニター状況と判断】

i　一般的に、S/A ランクバグの 5 ～ 10 倍の数の B/C ランクバグが存在する。発見された B/C の数が少ない（S/A 比率が大きい）場合、差分のバグが残存している可能性がある。
（バグ深刻度を深刻な順から S,A,B,C とした場合）

図表 4-9 は、システムテスト工程における機能別バグ状況を表したものである。週次に機能ごとの未解決バグを個々のバグの深刻度に応じた重みを乗じた数値の総和の大きさに応じて色分けして表したものである。（バグ状況の深刻度合を深刻な順から赤、黄、無色としている。この例では黄、無色のみ。）未解決バグが収束しないのは、特に、①、②、③のコンポーネントにおいて長期、または、断続的にバグが収束しきれないことが原因と判断できる。問題機能に対するリソース強化などの対策が必要である。



図表 4-9　機能別バグ管理表の例

図表 4-10 は、発見されたバグを週次で棒グラフにしたものに移動平均を重ねたものである。これは、今後のバグの発生を予測する Rayleigh 曲線を作成するための準備として作成したものである。この結果を使ってバグの発生予測（Rayleigh）曲線を作成したものが図表 4-11 である。棒グラフの青色は実績を赤色は予測を表している。バグ曲線と併せてシステムテストの終了を判断した時点での未解決バグの量の推定に使う。

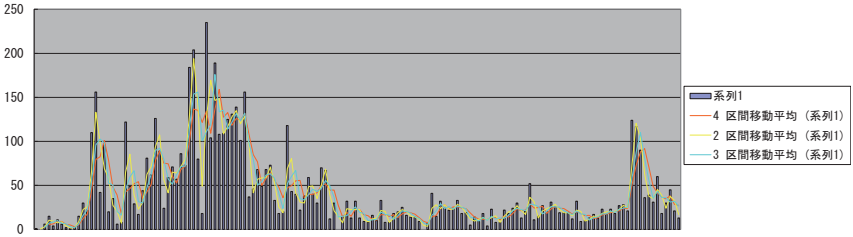
Rayleigh 曲線算出の方法

理論上累積発生件数　F(t)=K[1-exp-(1/2(tm**2))*(t**2))]

理論上発生件数　f(t)=K[(1/tm)**2*t*exp-(1/2(tm**2))*(t**2))]

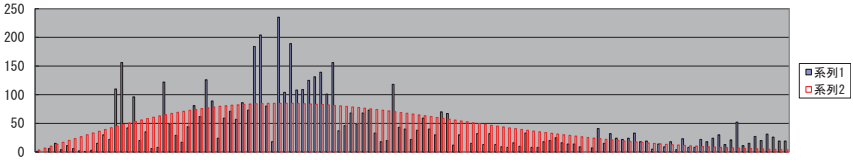
tm は、実際の発生件数が max になった時点(t:一般的には週)で、K は、F (t) に tm を代入して求めるが、簡易的に、tm までの累積が全体の 40%として計算できる。

（a）バグ発生表



図表 4-10　バグ発生予測表の例（バグ発生表）

（b）Rayleigh 曲線



図表 4-11　バグ発生予測表の例（Rayleigh 曲線）

以上の様に、幾つかの指標を活用して定量的に品質低下を判断した場合、品質低下を回避するために何らかのアクションをとる必要がある。そのような状況の下、ソフトウェア開発現場で取られているアクション（対策）とリスクを列挙する。

◆品質低下を回避するアクション

アクション		リスク
1. リソース追加	弱点部分（チーム、WBS）に投入	費用の追加を伴う
2. リソース入替え	担当パートの入替え／担当者の入替え	一時的にスローダウンする
3. リソース強化	エキスパート投入（支援・指導）	追加費用が発生する場合がある
4. WBS 追加	レビュー作業（プロセス）追加	追加費用が発生する場合がある
	評価（テスト）の追加（設計、実施）	追加費用が発生する
5. WBS 入替え	評価を中断して改修に集中	一時的に見かけ上品質改善されたように見えるが、評価部隊に手待ち発生（費用増加）のリスクがある
	改修を中断（限定）して評価に集中	改修による影響を抑制できるが、一時的に品質（進捗）が悪化しているように見える
6. WBS 削除	当該低品質機能をドロップ	客先（商品企画）の同意が必要
7. バグ棚卸し	長期未解決バグを削除（REJECT）	（別の現象として）再現するリスクがある
8. 作り直し	別リソースによる当該部分（モジュール、コンポーネント）の再開発	追加費用を伴い全体としてスケジュール不整合の発生リスクがある
		継続する場合との納期遅延・追加費用リスクの比較必要で、作り直しの判断時期を誤ることが多い

(4) その他のリスク

ソフトウェア開発の後半から終盤にかけて、バグの修正状況をモニターしているとプロジェクトが深刻な状況に陥っている状況が推察される場合がある。ここでは、バグの深刻度ランクやモジュール別の分析、重複バグの比率分析等に関する指標を活用して、それらの指標とその指標から判断できる効果を説明する。

◆その他の深刻なリスクをモニターする視点：(a) バグの修正

指　標	効　果
1. 深刻度高ランクの検出バグ数とその変化	品質・動作の（不）安定性判断、実現性の判断
【指標のモニター状況と判断】 ⅰ　増加率がプラスのままでなかなか収束しない場合、想定以上のバグがあるため大幅な遅延リスクがある。最悪は実現できないリスクもある。 ⅱ　増加率が低い状態が続いている場合、テスト仕様が不十分なためのテスト設計とテストのやり直しリスクか、機能の作り込みが遅れているための遅延リスクがある。	

指　標	効　果
2. 深刻度高ランクの未解決バグ数の変化	品質・動作の（不）安定性判断、実現性の判断
【指標のモニター状況と判断】 ⅰ　増加のままで減少に転じない場合、想定以上のバグがあるか、バグ解析の困難さによるスケジュール大幅遅延のリスクがある。 ⅱ　一定の範囲に留まったままで減らない場合、バグの解析が困難またはバグの修正が不可能なソフトのために実現できないリスクがある。	

指　標	効　果
3. 深刻度高ランク比率	潜在（未発見）バグ予測
【指標のモニター状況と判断】 ⅰ　一般的に、S/A ランクバグの 5 ～ 10 倍の数の B/C ランクバグが存在する。S/A 比率が大きいと B/C ランクバグ修正遅延のリスクがある。 ⅱ　S/A 比率が大きいと設計・仕様不整合による重要バグ流出リスクがある。（バグ深刻度を深刻な順から S,A,B,C とした場合）	

指　標	効　果
4. モジュール別検出バグ数とその変化量	対策必要チーム（人）の発見
【指標のモニター状況と判断】 ⅰ　バグが集中しているモジュールの影響で全体スケジュールが大幅遅延するリスクがある	

指　標	効　果
5. モジュール別未解決バグ数の変化量	対策必要チーム（人）の発見、実現性の判断
【指標のモニター状況と判断】 ⅰ　未解決バグが集中しているモジュールの影響で全体スケジュールが大幅遅延するリスクがある	

指　標	効　果
6. 個人別未解決バグ数の変化量	対策必要メンバの発見
【指標のモニター状況と判断】 ⅰ　ギブアップ或いは全体スケジュールが大幅に遅延するリスクがある	

指　標	効　果
7. 重複バグ数の比率とその変化量	テスト体制・方法課題発見
【指標のモニター状況と判断】 ⅰ　重複バグ数の比率に急激な増加傾向が見られる場合、大幅な解析のための無駄工数が発生するリスクがある。	

指　標	効　果
8. 長期未解決バグ数の変化量	不具合棚卸し時期の判断
【指標のモニター状況と判断】 ⅰ　発見時点とソフトウェアが大幅に変わっているため、再現しないものが滞留している可能性があり、解決できないリスクが高まる	

以上の様に、幾つかの指標を活用して深刻なリスクが潜在している可能性があると判断した場合、深刻なリスクを回避するために何らかのアクションをとる必要がある。そのような状況の下で、取るべきアクションと（対策）とアクションを取ることにより起こり得る二次リスクを列挙する。

◆その他の深刻なリスクを回避するアクション

アクション		リスク
1. 報告内容の精査	チームリーダに報告内容の見直しと聞き取りを指示し以下の対策を取る	ー
2. リソース追加	弱点部分（チーム、WBS) に投入	費用の追加を伴う
3. リソース入替え	担当パートの入替え／担当者の入替え	一時的にスローダウンする
4. リソース強化	エキスパート投入（支援・指導）	費用の追加が発生する場合がある
5. WBS 削減	機能削減	対顧客交渉が必要
6. WBS 入替え	評価を中断して改修に集中	一時的に見かけ上品質改善されたように見えるが評価部隊に手待ち発生（費用増加）リスクがある
	改修中断（限定）して評価に集中	改修による影響を抑制できるが、一時的に品質（進捗）が悪化しているように見える
7. バグ棚卸し	長期未解決バグを削除	別の現象として再現するリスクがある
8. 作り直し	別リソースによる当該部分（モジュール、コンポーネント）の再開発	追加費用を伴い全体としてスケジュール不整合の発生リスクもある
		継続する場合との納期遅延・追加費用リスクの比較必要で、作り直しの判断時期を誤ることが多い

4.2.2 報告

プロジェクトリーダは、定量データを活用して進行しているプロジェクトの状況を常にモニターし、計画からの逸脱を検知した時点で、迅速に対策を立案し、実行する。一方、経営者に常時マクロ指標を報告するとともに、経営判断が必要な場合はプロジェクトに関係する部門に対して必要な指示を出してもら

図表 4-12　進捗・コスト・品質・生産性管理表の例

yyyy 年度 mm 月末実績 ソフト開発状況

開発プロジェクト名	全般情報				コスト								
	開発地		量産開始日	進捗状況	対開発決定時〈ソフト開発総費用〉単位：百万円					対 yyyy 年度予算 単位：百万円			
	ハード	ソフト		進度	見積時情報		②支払額総計	分析結果（残情報）		③申請額評価部含む	④支払額yyyy年～m月	分析結果（残情報）	
					開発費用	①見積額		残金＝①－②	消化率＝②／①			残金＝③－④	消化率＝④／③
Project1	東京	東京	yy1/mm1	▲	852	852	557	295	65.4%	414.3	105	309	25%
Project2	東京	東京A社	yy2/mm2	◎	160	160	154	6	96.5%	99.9	7	93	7%
Project3	東京	東京B社	yy3/mm3	■	123	123	163	-40	132.9%	23.7	21	2	90%
Project4	東京	東京C社	yy4/mm4	◎	1,760	1,760	1,475	285	83.8%	838.8	203	636	24%
Project5	東京	東京A社	yy5/mm5	★	1,475	1,475	3,750	-2,275	254.2%	932.4	222	710	24%
Project6	東京	東京A社	yy6/mm6	★	110	110	895	-785	813.5%	41.7	-78	120	-187%
Project7	東京	東京B社C社	yy7/mm7	▲	1,996	1,996	1,824	172	91.4%	2188	244	1,944	11%
Project8	横浜	東京	yy8/mm8	▲	1,890	1,890	1,791	99	94.7%	694.1	123	571	18%

計画日程に対して … ◎：１週間以内遅れ、▲：１ヶ月以内遅れ、■：２ヶ月以内遅れ、★：２ヶ月を超える遅れ

う。ここでは、経営者への報告内容について例を上げて解説する。

図表 4-12 は、管理対象の開発中プロジェクトに関する進捗、コスト、品質、生産性の状況を示している。進捗は、記号で計画日程に対する遅れの程度を示しており、コストは、予実差で示している。品質は、発見されたバグ数（累計）と未解決バグ数（残件数）を深刻度別に示している。更に前回報告時からの進捗を記載している。

品質									生産性					
検出バグ数（yyy/mm/dd 時点）								残件数前月増減	工数	開発規模（yyyy 年 mm 月 dd 日時点）				前月との差異 SLOC数（KSLOC）
累計				残件数							再利用元製品			
S	A	その他	合計	S	A	その他	合計		累計工数（人月）～ yyyy/m/d	母体含む SLOC 数（KSLOC）	新規 SLOC 数（KSLOC）	再利用 SLOC数（KSLOC）	再利用率（%）再利用 / 母体含む	
0	84	105	189	0	82	104	186	-3	1,022人月	1,143	784	359	31%	72
292	542	1,215	2,049	13	55	297	365	-31	178人月	8,541	370	8171	96%	66
24	76	136	236	7	12	33	52	-15	172人月	1870	160	1710	91%	6
13	159	408	580	2	49	281	332	48	2,013人月	2,818	再利用元が多種の為不可能			3
827	1,652	2,352	4,831	458	1,126	3,138	4,722	-316	4,276人月	3,019	3,019	新規		234
48	975	2,136	3,159	1	96	383	480	1	959人月	818	787	31	3.7%	21
48	70	508	626	33	61	309	403	-69	2,122人月	2,546	2,546	新規		1
202	658	1,227	2,087	5	35	205	245	72	2,103人月	3,516	2,091	1,433	41%	2

この例では、生産性が極端に悪化しているプロジェクトの欄を赤で囲っており投入コストと品質も赤字で示している。管理限界(コスト、進捗)を超えたプロジェクトについては経営の視点から迅速に精査して対策を講ずるべきである。

ここで示した例は、計画データと定量データから自動的作成可能であり月次ベースでの管理が望まれる。

第5章

5. 指標活用による組織の開発力改善

組織の開発力改善の最たる目的は、生産性と品質の両方を向上させることにある。

組込み機器やシステムに新しい技術やデバイスが利用されるにつれて、新たに求められる機能が従来の機能に加わり、ソフトウェアの規模と複雑さが増す。技術的に難しくても小規模であれば、優秀な人材に依存することでものづくりは成功するであろうが、大規模化・複雑化したソフトウェア開発にはソフトウェア開発プロセスの導入と共に工学的なアプローチによる改善が必要になってくる。定量データ活用は、工学的なアプローチを行うためのひとつの有力な手段であり、競争力を維持してゆくためには定量データから抽出された指標を活用して、組織の開発力改善を推進する必要がある。

5章では、ソフトウェア開発に従事する組織において、組織の指標が無ければ改善することが難しい下記の課題：【1】、【2】、【3】について、指標を活用した対策例を紹介する。図表5-1は、各課題：【1】、【2】、【3】の中で比較的関心の高い項目とその項目を改善するために活用できる指標をマトリクスで対応させたものである。指標活用には、開発規模の見積もりデータや前提条件、現状を把握するための実績データも必要なため、併せて示す。

【1】見積もりの適正化
- 全体コストの見積もり
- テスト工数の見積もり

【2】品質（信頼性）の把握

【3】生産性の把握

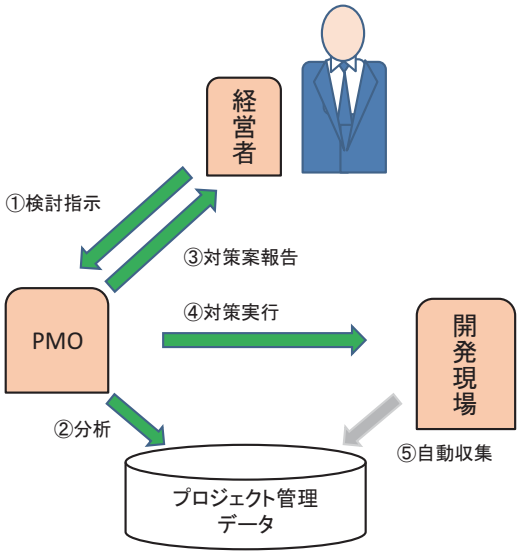
表 5-1　改善可能な作業項目と活用する指標

※ 3 章の図表 3-3 で紹介した管理指標には、指標 ID を示す。

改善可能な作業項目		活用する指標・データ	組織の基準となる指標※ P1 SLOC生産性［KSLOC／人月］	T13 T19 バグ密度［件数／KSLOC］	T6 テスト密度［項目数／KSLOC］	T14 T20 バグ検出率［バグ件数／テスト項目数］	T5 テスト作業実施率［人時／KSLOC］	テスト項目消化平均工数	バグ解決に要する平均工数	R5等 レビュー作業実施率［人時／KSLOC］	工程別工数比率	工程別工期比率	見積もりデータまたは前提条件等 SLOC規模（新規・改造）	確保可能な要員数	全体工期	工数単価	実績データ SLOC規模（新規・改造）	工数実績（全体、工程別）	テスト実施・消化項目数	テスト検出バグ件数	テスト解決バグ件数	テスト進捗実績
【1】見積もりの適正化																						
(1)	全体工数（コスト）	SLOC 規模／ SLOC 生産性（コスト＝工数×単価）	●										●			●						
(2)	工程別工数	工程別工数比率×全体工数（1）	(1)								●		(1)									
(3)	工程別工期	工程別工期比率×全体工期										●			●							
(4)	レビュー工数	SLOC 規模×レビュー作業実施率								●			●									
(5)	テスト工数	SLOC 規模×テスト作業実施率（または、工程別工数 (2)）					●						●									
(6)	テスト項目数	SLOC 規模×テスト密度			●								●									
(7)	テスト期間	計画に依存（または、工程別工期 (3)）										(3)			(3)							
(8)	検出バグ件数	SLOC 規模×バグ密度		●									●									
【2】品質（信頼性）の把握																						
(1)	バグ密度	バグ件数／SLOC 規模		●													●			●		
(2)	テスト密度	テスト項目数／SLOC 規模			●												●		●			
(3)	バグ検出率	バグ件数／テスト項目数				●														●		
【3】生産性の把握																						
(1)	全体の生産性	SLOC 規模／全体工数															●	●				
(2)	テストの生産性	テスト項目数／テスト工数																●	●			
(3)	テスト工数比率	テスト工数／全体工数																●				

◆改善プロセス

組込みソフトウェア開発企業が現状の業務を改善するためには、まずは組織全体から各開発プロジェクトの実態を見渡し、不効率な作業や中身の見えない作業を見つけることに努める。不効率な作業や中身の見えない作業を見つけるための手段は、各開発プロジェクトから収集した定量データの活用に他ならない。経営者は、組織の各開発プロジェクトを横断的に見ることができる品質関連部門や PMO に、現状の把握と分析を指示しなければならない。品質関連部門や PMO は、経営者からの改善指示を理解し、収集した定量データの現状分析を行う。分析結果より不効率な作業、中身の見えない作業を見つけ、どうすれば対策可能か、いくつかの対策案を経営者に示し、互いの認識を共通にした上で、対策の実行に移す。



図表 5-2　改善プロセス

5.1 見積もりの適正化

短期的もしくは中期的に見て、ソフトウェア開発の採算を取ることは勿論のこと、開発競争力を維持するためには、コスト見積り精度を上げて、見積りの範囲内でプロジェクトを終了しなければならない。3.3 節に示す生産性、規模当たりのテスト項目数等の指標を活用することで、見積もりを定量的に行うことができる。プロジェクトを開始する際に定量的に見積もった値は、工程やプロジェクト完了時の実績値との乖離状況を精査することで、次回に向けて見積もりに使う指標を適正化することができる。組織の標準指標を用いた見積もりを行う習慣が定着してくれば、計画値と実績値の分析を繰り返すことで、さらに精度の高い見積もりを行うことが可能になる。

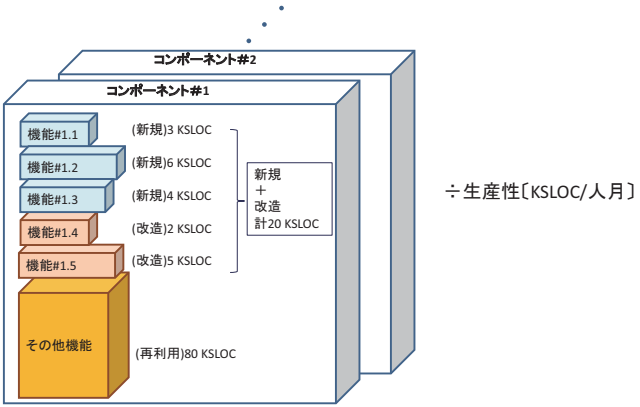
また、組織の指標を共有する仕組みが構築されると見積もり評価を組織的に行うことも可能になる。

5.1.1 全体コスト見積もり

ソフトウェア開発コストの大部分は工数が占める。小規模ソフトウェアの場合は、経験と勘による見積もりでもリスクは小さいかもしれないが、規模が大きくなると定量的な見積もりを行って精度を上げなければ、大きなリスクを抱えることに成り得る。

◆ここで利用する指標・データ

- SLOC 規模（経験と勘で見積もったもの）
- SLOC 生産性（指標）



図表 5-3　SLOC 規模と生産性による工数見積もり

プロジェクトの工数を定量的に見積もるためには、

① 開発規模を見積もること。

② プロジェクトの生産性を定めること。

が原則である。

図表 5-3 に示すように、新規にコーディングする部分と既存のソフトウェアを改造してコーディングする部分の SLOC 規模を生産性で割ることにより工数を算出する。

・工数＝ SLOC 規模÷生産性

見積もりの精度を上げるためには、新規に開発する部分と改造する部分の SLOC 規模をコンポーネント毎、機能毎に見積もり積み上げて行く。

SLOC 規模の見積もりは経験と勘に依るが、過去のプロジェクトの類似ソフトウェアモジュールの SLOC を参考に機能毎に見積もり値を積み上げて行くことで定量化できる。類似する過去の資産が無い場合は、条件を仮定する等、見積もりの根拠を出来る限り考慮しながら予測する。尚、見積もりの根拠は記録しておき、見積もった値を修正する際の基準にする。

生産性も過去の類似プロジェクトデータを分析することで、目安を決めることができる（5.3 節「生産性の把握」参照）。生産性は、実際に分析してみると分かることであるが、アプリケーションとドライバや作業者のスキルレベルに依って違いがあるため、いくつかのパターンで決めたり、係数を掛けたりする。

ただし、3.3 節で触れたとおり、改良（派生）開発の場合の SLOC 規模は、改造せずにそのまま再利用する母体ソフトウェアの規模が含まれないため、過去

のデータから算出した生産性指標に、そのまま再利用する母体ソフトウェアに関わる作業（影響範囲の調査、総合テスト）がどの程度考慮されているか否か注意する。

ここでのポイント

- ●要求仕様から勘と経験で工数を見積もるのではなく、個々の要求事項からSLOC規模を見積もり更に組織の実力値指標としての生産性を適用させて見積もる習慣が定着してくれば、プロジェクト毎に計画値と実績値を分析し、実力値指標としての生産性の導出式を精査して行くことにより、精度の高い見積もりを行うことが可能になる。
- ●計画時に見積もった規模が妥当であるかどうかは、コーディングを始めるにつれて、徐々に分かる。工数は規模に比例するため、規模が増加傾向にあることを早期に見つけることが出来れば、担当者が能力以上の作業量を負担する危険性に対策を打つことが出来る。

5.1.2 テスト工数の見積もり

テストに必要な工数は、コーディング作業までに作り込まれた品質に依存し、想定外に多くのバグが発生すると、バグ対処にかかる工数やバグ対処に起因するデグレーションを回避するための追加テストが増加して、見積りから大きく増加することがある。また、テストの網羅性を上げれば上げるほど、品質を担保することが出来るが、通る可能性が極めて低いルートやルートの組合せのために、どこまでテストに工数を割くべきかの判断は難しい。

ここでは、テスト開始前の品質レベルとテスト網羅性の二つの観点に基づくテスト工数の見積もり例を紹介する。

◆ここで利用する指標・データ

- ・テスト作業実施率
- ・工程別工数比率
- ・テスト密度（SLOC規模当たりのテスト項目数）
- ・バグ密度（SLOC規模当たりのバグ検出件数）
- ・テスト項目消化平均工数〔人時／項目〕
- ・バグ解決に要する平均工数〔人時／件〕

ここでのポイント

●ソフトウェア結合テストや総合テストの工数を見積もる際には、当然のことながら、テストの作業内容を明確にする。

図表 5-4　ソフトウェア結合テスト作業の内訳

ソフトウェア結合テスト作業の内訳（例）		
	テスト準備	①テスト仕様書の作成（テスト項目とテスト内容や手順を含む） ②テスト環境の作成
	テスト実施	③テスト項目に対応したテストデータの作成 ④テストの実施
	バグ対処	①原因調査 ②プログラム修正 ③確認テスト

（1）SLOC 規模当たりのテスト工数見積もり

単位 SLOC 規模あたりに必要なテスト工数の指標「テスト作業実施率〔人時／KSLOC〕」を活用したテスト工数見積もりの手順例を示す。この例では、テスト工程を結合テスト・総合テストの二つの工程に分けられたテスト工程のうち、結合テスト工程のテスト工数を見積もっている。また、それぞれのテスト工程でバグの定量データを蓄積しているものとする。

〔**手順①**〕過去の類似プロジェクトのデータから、

・SLOC 規模
・結合テスト工数実績
・結合テストのバグ検出状況

を抽出する。

〔**手順②**〕抽出したデータから、単位 SLOC 規模あたりの結合テスト工数実績：

・結合テスト作業実施率〔人時／KSLOC〕

を算出する。

〔**手順③**〕「結合テスト作業実施率」は、コーディング作業までに作り込まれた品質に依存するため、抽出した「結合テストのバグ検出状況」を分析し、テスト開始前の品質レベル（例：良、中、低）を推測し、手順②で算出した「結合テスト作業実施率」を品質レベルで分類する。

・品質レベル良の「結合テスト作業実施率」

・品質レベル中の「結合テスト作業実施率」
・品質レベル低の「結合テスト作業実施率」

〔**手順④**〕当該プロジェクトの品質レベルを予測する。

〔**手順⑤**〕手順③、④の結果から、当該プロジェクトに適用する「結合テスト作業実施率」を決め、当該プロジェクトのSLOC規模に掛けあわせて結合テスト工数を見積もる。

総合テストの場合も同様に見積もるが、テスト対象に前項5.1.1で述べたように再利用する母体を含めるため、新規と改造部分のSLOC規模を基準にした総合テスト工数の見積もりは、再利用率や母体への影響範囲を考慮して精査する。

テスト工数の見積もりは、全体工数を見積もった後に、各工程に配分する工数の工程比率（工程別工数比率）を用いて算出する方法も一般に利用されている。見積もる工数が全体か部分かの違いだけでありSLOC規模と工数実績から算出する点では同じである。

(2) テスト密度・バグ予測からの見積もり

前項(1)のSLOC規模を基準としたテスト工数が妥当かどうかの裏付けをとるためには、

・テスト密度（SLOC規模当たりのテスト項目数）
・バグ密度（SLOC規模当たりのバグ検出件数）

からテスト工数を算出して比べる。

テスト工数は、テスト項目数や発生したバグの件数により増減するため、過去の類似プロジェクトデータから、

・テスト項目消化平均工数〔人時／項目〕
・バグ解決平均工数〔人時／件〕

を算出しておく。

テスト密度を高く設定することで、ある程度網羅性は向上するであろうが、テストケースを作成する際の想定力が豊かでなければ網羅性はあるレベル以上上がらない。どこまでテスト密度を高くすれば良いのかの判断は難しいが、まずは、過去データを分析して、出荷前のバグ収束状況や出荷後のバグ発生状況とテスト密度の関係を調べることである。

当該プロジェクトのテスト項目数と検出バグ数は、下記の手順で導出する。

〔**手順①**〕過去の類似プロジェクトのデータから、
・SLOC 規模（新規・改造）
・結合テスト実施項目数
・結合テスト検出バグ数
を抽出する。

〔**手順②**〕抽出したデータから、単位 SLOC 規模あたりの結合テスト実施項目数:
・結合テスト密度〔項目数／ KSLOC〕
を算出する。

〔**手順③**〕手順②の結果から当該プロジェクトの「結合テスト密度」を決定し、SLOC 規模に掛けあわせて
・結合テスト項目数
を決める。

〔**手順④**〕次に、抽出したデータから単位 SLOC 規模当たりの結合テスト検出バグ件数（解決した件数）:
・結合テストバグ密度〔件数／ KSLOC〕
を算出する。

〔**手順⑤**〕結合テストのバグ密度は、コーディング作業までに作り込まれた品質に依存するため、抽出した「結合テストのバグ検出状況」を分析し、テスト開始前の品質レベル（例：良、中、低）を推測し、手順③で算出した「結合テストバグ密度」を品質レベルで分類する。
・品質レベル良の「結合テストバグ密度」
・品質レベル中の「結合テストバグ密度」
・品質レベル低の「結合テストバグ密度」

〔**手順⑥**〕当該プロジェクトの品質レベルを予測する。

〔**手順⑦**〕手順⑤、⑥の結果から、当該プロジェクトに適用する「結合テストバグ密度」を決め、当該プロジェクトの SLOC 規模に掛けあわせて
・結合テスト検出バグ数
を予測する。

以上により、結合テスト項目数と結合テスト検出バグ件数を見積もることができれば、「テスト項目消化平均工数〔人時／項目〕」と「バグ解決平均工数〔人時／件〕」により結合テスト工数を見積もることができる。

ソフトウェア総合テストの工数を定量的に見積もる方法も同様に行えば良い。

ここでのポイント

「総合テストのテスト密度の考え方」

●結合テストは、モジュール間のインタフェースをテストする目的（※１）のため、改造したり新たにコーディングしたプログラムを対象とする考えが一般的である。そのため、テスト密度（SLOC 規模あたり実施すべきテスト項目数）の設定の仕方にバラつきは少ない。一方、総合テストは、ソフトウェア全体のテストを行うことが目的のため、テストの対象には、改造せずにそのまま再利用した母体が含まれる。テスト密度の妥当性を考えた場合、改造部分・新規部分が全体ソフトウェアに分散して配置されている場合と一箇所に局所的に配置されている場合では、テストケースの作り方や考えが異なる。

※１）参考文献［5］：ESTR、Page10：1.1.1 項「テストの目的と位置付けの明確化」に単体テスト、結合テスト、総合テストの目的と位置付けが説明されている。

5.2 品質（信頼性）向上

組織内で開発するソフトウェアの信頼性を向上させるには、組織が開発したソフトウェアの品質（信頼性）を定量的に把握し、その上で打つ対策を考える。

（1）品質（信頼性）の把握

◆手順

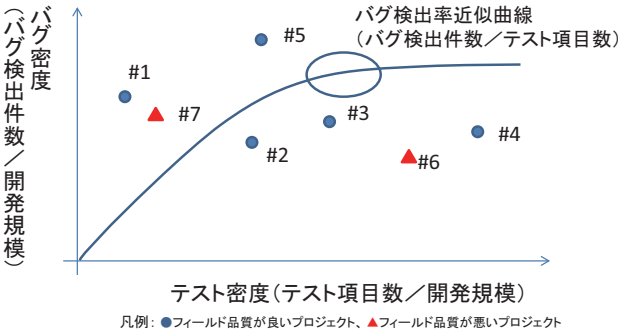
①組織に蓄積された過去のプロジェクトデータから、出荷後にフィールドで発生した問題の件数、影響度等を分析し、フィールド品質が良いプロジェクトと悪いプロジェクトに分類してみる（図表 5-5）。

図表 5-5　出荷後 1 年間のフィールド発生バグ状況の例

	影響度 A のバグ	影響度 B のバグ	合計	評価
プロジェクト＃ 1	0 件	2 件	2 件	●良い
プロジェクト＃ 2	0 件	1 件	1 件	●良い
プロジェクト＃ 3	0 件	2 件	2 件	●良い
プロジェクト＃ 4	0 件	3 件	3 件	●良い
プロジェクト＃ 5	0 件	1 件	1 件	●良い
プロジェクト＃ 6	2 件	3 件	5 件	▲悪い
プロジェクト＃ 7	2 件	4 件	6 件	▲悪い

②その評価結果をもとに、出荷前テスト（総合テスト）の品質を次の指標を用いて比較してみる（図表 5-6）。

・総合テストバグ密度（バグ検出件数／開発規模 KSLOC）
・総合テスト密度（総合テスト項目数／開発規模 KSLOC）
・総合テストバグ検出率（バグ検出件数／テスト項目数）



図表 5-6　出荷前テスト（総合テスト）の品質状況の把握

③図表 5-6 にプロットした●フィールド品質が良いプロジェクトのテスト密度とバグ密度からバグ検出率（バグ件数／テスト項目数）の近似曲線を描いてみる。近似曲線上の点線の円で囲んだ部分は、テスト密度を増加させてもバグ検出密度が頭打ちになるところであり、組織の標準指標としてテスト密度の目安値とすることができる。

④次に図表 5-6 にプロットした▲フィールド品質が悪いプロジェクトのテスト密度とバグ密度を分析してみる。

▲ #6：テスト密度が高い割にバグ密度が低い。

⇒ テスト項目を多く消化しているが潜在しているバグが検出できていないと判断できる。もっとバグにヒットする様にテスト内容を見直す必要がある〈対策 1〉。コスト面においても無駄なテスト工数を消費している可能性がある。

▲ #7：テスト密度が低くバグ密度が高い。

⇒ 潜在しているバグを検出するために必要なテスト項目を消化していないのに、バグが多く検出されていると判断できる。もっとテスト項目数を増やしてバグ検出が収束することを確認する必要がある〈対策 2〉。

(2) 品質（信頼性）向上への対策

〈対策 1〉バグにヒットするようなテスト内容の作成

⇒テスト内容が悪い原因、真の要因を把握して対策する。

【想定原因の例】

①テスト仕様書やテストケースが作成されていない。

②テスト仕様書やテストケースは作成されていてもレビューされていない。

③テスト仕様書やテストケースを作成するための元資料、つまり要求仕様書や設計書が作成されていない。

〈対策 2〉適正なテスト項目数の実施

⇒テスト項目数が不足する原因、真の要因を把握して対策する。

【想定原因の例】

①組織の標準指標として、テスト密度の目安がない。

②出荷前テスト（総合テスト）を開始する時点で、開始できるレベルの品質が作り込まれておらず、テスト項目の消化よりも、前工程のテスト（結合テスト）で解決すべきバグの対処に時間を割かれた。

5.3 生産性向上

前節 5.2 で言及しているように、バグの対策は、後の工程に行くほどコストが掛かると言われている。コストが掛かる要因は、

・上流工程で解決すべき問題は、コーディング後には、複数個所に分散して存在することがあり、複数個所のソースコードに修正が入る。
・テスト工程でバグが見つかった場合、修正した箇所以外に他に影響を及ぼしていないかどうかのテストが必要になり、同じテストをくりかえすため、テスト工数が重複する。

である。

要求定義や設計の上流工程の作業でバグが混入する問題は、仕様書や設計書のレビューを実施し、混入した工程で検出し解決することが一般に推奨されている。一方で、レビューを実施するために、暗黙的に関係者が共有している内容を仕様書や設計書に整理して記載する作業工数や、直接コーディングすれば良い内容を、他の人が理解できるようにわざわざ設計書に整理する作業は、かえって無駄な工数とみなされる考えもある。上流工程のレビューのためにそれなりの工数を費やしても、下流のテスト工程で検出してバグ対処する工数は、それほど減らないと考えるベテラン技術者も少なくない。

ドキュメント作成作業にある程度の工数が割かれても仕様書や設計書のレビューが生産性向上に効果的であることを理解してもらうためには、ソフトウェア開発作業の効率を定量的に示せる生産性を利用する。まずは、過去のプロジェクトデータから、生産性を把握し、個人のスキルの違いではなく、開発プロセスの違いにより生産性が良くなる事例を探してみる。

(1) 生産性の把握

◆手順

①組織に蓄積された過去のプロジェクトデータを分析し、出来るだけ条件を揃えて生産性を比較する。
②仕様書や設計書レビューの有無、テスト工程の切り方等、そのソフトウェアを開発したプロセスと生産性の関係を定量的に分析する。
③結合テスト、総合テスト（出荷前の最終テスト）それぞれのバグ密度、テスト密度と生産性の関係を調べる。
④潜在バグ件数が少なく、生産性も高いプロセスを標準化する。

ここでのポイント

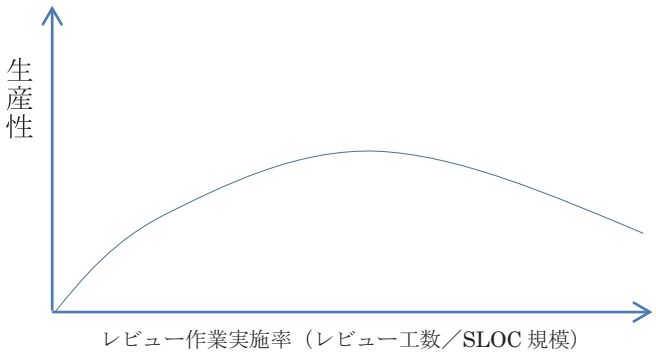
- ●新規開発と改良（派生）開発の生産性は区別する。
- ●開発規模の違いにより生産性も変わる。
- ●生産性の比較は、生産性を算出する工数の工程範囲を揃える。
- ●生産性は一般に新規と改造部分の SLOC 規模を基準にするため、母体ソフトウェアの再利用率に注意する。

(2) 設計レビューと生産性

NIST レポート（参考文献［6］）に見られるように、上流工程でバグを解決するためにかかる工数は、下流工程でバグを解決するためにかかる工数の 10 ～ 15 倍少なくて済むと言われる。下流工程でバグを見つける作業はテストであるが、上流工程でバグを見つける作業は設計レビューである。つまり、「設計レビュー作業実施率」や「レビュー作業充当率」を今より上げることで、生産性が向上するかをモニターしてみる。ただし、レビュー工数を掛けることは全体工数が増えるわけであるから、むやみにレビュー工数を掛ければ良いというわけではない。

◆手順

①終了したプロジェクトの設計レビュー作業実施率と生産性を調査する。
②図表 5-7 のようなグラフにプロットしてみる。



図表 5-7　レビュー作業実施率と生産性

以上

＊参考文献

[1] ESPR :Embedded system development Process Reference
（SECBOOKS：ESPR Ver.2.0：【改訂版】組込みソフトウェア向け　開発プロセスガイド、IPA/SEC、翔泳社、2007 年）
http://www.ipa.go.jp/sec/publish/tn07-005.html

[2] SECBOOKS：定量的品質予測のススメ
～ IT システム開発における品質予測の実践的アプローチ～、IPA/SEC、株式会社オーム社、2008 年
http://www.ipa.go.jp/sec/publish/tn08-004.html

[3] ESQR :Embedded system development Quality Reference
（SECBOOKS：ESQR　Ver.1.1：【改訂版】組込みソフトウェア開発向け品質作り込みガイド、IPA/SEC、2012 年）
http://www.ipa.go.jp/sec/publish/tn12-001.html

[4] SEC BOOKS：組込みソフトウェア開発データ白書 2015、IPA/SEC、2015 年

[5] ESTR :Embedded system development Testing Reference
（SECBOOKS：組込みソフトウェア開発における品質向上の勧め［テスト編～事例集～］、IPA/SEC、2012 年）
http://www.ipa.go.jp/sec/publish/tn12-004.html

[6] NIST Planning report 02-3,“The Economic Impacts of Inadequate Infrastructure for Software Testing”, May 2002.
http://www.nist.gov/director/planning/upload/report02-3.pdf

執筆者

三原　幸博　　IPA/SEC
松田　充弘　　IPA/SEC
田代　宣子　　IPA/SEC

監修

古山　恒夫　　東海大学　理学部　客員教授
野中　　誠　　東洋大学　経営学部　教授

組込みソフトウェア向け

プロジェクトマネジメントガイド［定量データ活用編］

～定量データ活用による組織の開発・管理力向上～

2015 年 11 月 18 日　1 版 1 刷発行

監修者　独立行政法人情報処理推進機構（IPA）
技術本部 ソフトウェア高信頼化センター（SEC）

発行人　松本　隆明

発行所　独立行政法人情報処理推進機構（IPA）
〒113-6591
東京都文京区本駒込二丁目 28 番 8 号
文京グリーンコート センターオフィス
URL http://www.ipa.go.jp/sec/index.html

ISBN978-4-905318-37-8
C3055 ¥463E

定価：本体463円＋税

IPA 独立行政法人 **情報処理推進機構**
技術本部 ソフトウェア高信頼化センター

SEC-TN-15-002