ソフトウェア工学　88－3
(1992. 11. 10)

ソフトウエア開発工程の定量的取り扱い

河野　善弥,　ホマユン　ファー　ベルーズ

埼玉大学　工学部　情報工学科

浦和市　下大久保　２５５

あらまし　本報告はソフトウエア開発に関して定量的／科学的／合理的な管理の為の基礎研究である．人の行なう作業の工数を適当な単位で正規化すると一定性を示す．　しかし，ソフトウエア開発作業に適用しようとすると大きなバラツキが生ずるので，巨視的な値以外は信頼できるものではなかった．この報告は「一定性」の成立根拠から出発し，殆どの場合のバラツキの原因を明らかにした．　作業手順，作業内容，正規化する単位，作業区分であるドキュメント，作業能力，ソフトウエアテストにおける不良率等の原因を示し，作業効率が学習による習熟効果に従う事を見いだしている．　これらの結果，開発作業の都度，計画と実績を対照して作業方式を改善するフイードバックにより作業効率が高まりバラツキも減ていく．　この過程が作業組織／手法の成熟過程である事を明らかにしている．

和文キーワード　ソフトウエアエンジニアリング，作業効率，習熟効果，インダストリアルエンジニアリング

Quantitative Basis for Software Development Process

Zenya Koono, Homayoun Far Behrouz

Department of Information and Computer Sciences,
Faculty of Engineering, Saitama University.

255 Shimo-okubo, Urawa, 338, Japan.

Abstract　This paper reports a basic study aiming at quantitative, rational and scientific control of software developments. Man-hours for a work shows a constancy when normalized by a suitable factor. In software developments, however, most such figures do not show enough constancy, except a gross average. This paper shows causes of such variations; such as work procedure, work process, normalizing factor, documentation, failure rate during testing, work ability and morale. It was found that learning effect appears during developments. The learning process with feedback from past experiences is the vital factor for improving the process and thus resulting in smaller variations.

英文 key words　Software Engineering, Work Efficiency, Learning Effect, Industrial Engineering

1．始めに

色々な現象を定量的に捕捉し，原因と結果の関係を明確にする事によって，科学的な進歩が始まる．ハードウエア製造については１９世紀末から２０世紀初頭にF. W. Taylor により作業時間効率の定量的研究が行なわれた．　この結果に基づき科学的な管理が目指され，Industrial Engineering （以下ＩＥと略する）が成立し，今日のハードウエア産業の合理的／科学的な管理の基礎築かれたといわれている．

ソフトウエア開発は定量的な取扱いが強く求められているが，定量化は未だに充分なされたとは言えない．　原因として色々の事が考えられるが，対象につ いて

＊再現性が乏しい　　＊繰り返し／再現が困難

＊関係要因が余りに多種多様

等の困難性があげらている．

所が現実には多くのソフトウエア事業が正常に営まれている．　産業界は「アカデミックに科学的とは言えない」が，経験によりソフトウエアを制御している．　実体が先行して，科学が遅れているといえようか，　実作業の経験を定量的に評価し整理すると体系化が出来る．　無意識に行なわれている産業界での経験を整理していく事が，次第に洗練され定式化に繋がると考え，現状まだ不十分ではあるが，ここに報告として纏めた.[1]

2．　作業工程の基本的性質

2．1　作業効率の一定性

「総作業工数をある単位で正規化すると一定になる」事が経験的に知られている．　（例えば，１台の自動車を洗うのに１時間要した．Ｎ台の自動車を洗う時間は幾らか？と問われれば，殆どの人はＮ時間と答える．）　この性質を「作業効率の一定性」，このような単位を「原単位」，（総作業工数／原単位またはその逆数）を「作業効率」，入力（作業前状態）から出力（作業後状態）への広義の変換作業を「作業工程」と定義する．

「作業効率」は，作業方法／環境などの条件，能力，更に努力等の影響を受ける．　ハードウエア製造作業は繰返し作業で再現／計測が容易であり，科学的な研究が早くに行われ，ＩＥにより合理的な製造管理基礎が築かれた．　「標準の作業方法／環境などの条件，能力，通常の努力での作業効率の平均値が一定になる」ので，これを標準時間として，能力評価，作業計画等に用いている．

[1] 本論文は従来散発的に発表してきたこの種の研究を纏めた参考文献[Koono 90c]を再整備した物である．

ソフトウエア（やハードウエア）の開発作業で，プログラム（ゲート）規模を与えると，ベテランは必要な工数を予測出来る．　それは各種条件が略同一なら，細部条件相違は巨視的に平均する事により消える事を知っている為である．　そこで，現在多くのソフトウエア企業は，「作業効率の一定性」を予算制度等に無意識に用いている．　本報告は，科学的／合理的な作業の定量管理の基礎として，作業効率の一定性，バラツキ原因などに重点をおいて報告する．[Koono 90c]

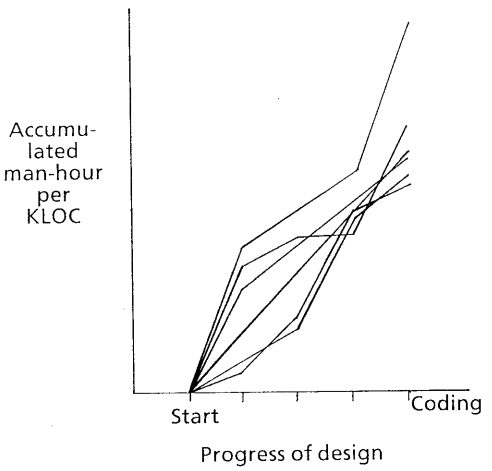

図２．１．１　累計作業効率 [Koono 90c]

図２．１．１[Koono 90c]は，昔アセンブラ言語での作業経験者集団が初めてコンパイラ言語を使う新規開発作業に従事したプロジェクト実績を示す．各チームの作業工数を開発作業結果のソースコード行数で正規化した．　累計していくと図の右端の区分（コード化の為の作業の終了時点）近くで，各累計作業効率が収束する．これは

＊各チームが同様な作業環境条件で作業したので　コード化する前の作業効率が略一定である事

を示す．　（作業環境条件が異なれば，作業効率は異なった値になる．　図中初期のプロットのバラツキは２．２．３で説明する．）

2．2　作業効率の変動

2．2．1　作業手順による作業効率の変動

参考として，ハードウエア製造作業での例を示す．図２．２．１[Koono 90a,b]はある部品製造（比較的単純な作業）工程につき，数名の作業者の作業を合理化（標準化）する前と後の作業効率の分布を示す．作業は比較的簡単なのにそれでも作業手順を標準化前には大きくバラツク．　標準化後は作業効率の平均値が減少（合理化結果）し，かつバラツキも減少

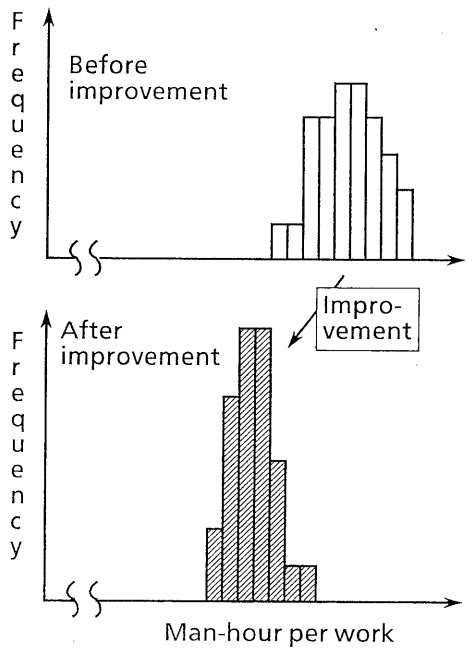

図２．２．１　作業効率分布の変化 [Koono 90a,b]

している．　これは，
＊作業手順が最も効率的な手順に統一された結果，
　バラツキの最大要因が消えた
事による．　この例は同一集団内なので変動は小さいが，作業条件が異なる集団間では平均値が異なり，各種集団混合で観測すると変動は大きくなる．

２．２．２　作業内容の相違による作業効率の変動

同一種製品を開発するＡ，Ｂの２集団について，開発の決定後，機能と構成の仕様を文書化完了する迄の上流の開発につき実績作業効率を調べた．　作業工数を製品規模で正規化しても効率は揃わず，改変作業量規模でも揃わない．　しかし作業結果の仕様書のＡ４換算枚数を原単位として正規化すると作業効率は集団毎に収束した．　集団毎に桁の位の系統差があり，集団内で最小／最大は数倍に達する．

集団の系統的な差の原因は次の通りであった．
Ａは新規製品開発，Ｂは現存製品の機能改変を行う製品開発を行なっており，集団毎に作業内容が相違していた．
Ａの作業内容＝
（新規開発の外部条件検討＋構成方式の研究）
＋（機能仕様書作成＋構成仕様書作成）
Ｂの作業内容＝
（変更開発の外部条件検討＋方式的変更洗出し）
＋（機能仕様書作成＋構成仕様書作成）
前の項の作業工数を除き，（機能仕様書作成＋構成仕様書作成）に限定して
＊同一作業内容にすると系統差は消失
した．　（系統差は消えてもバラツキは中心と最小，最大の比は数倍になる．）　作業効率は作業内容に依存する事は当然で，これは作業手順より上位の同種原因のバラツキといえる．

前記作業の後工程にあたる設計作業につき，高熟練者中心に社内作業する集団Ｃの作業効率を作業結果の仕様書のＡ４換算枚数を原単位として評価した結果は次のとおりであった．
＊この集団内で収束する
＊平均値は前者の１／３程度になる
＊相対（比）的バラツキは３倍程度になる
集団Ｃは出張は無く高熟練者中心なので，全決定が自集団内で速やかに行なわれ，また専念度が高かった．　（一般に間接作業者は専念率数十ないし数％といわれる．）　これらが上記結果をもたらした．

これらに共通して，作業結果の正規化文書量を原単位とすると良い結果が得られる．　人間の知的作業速度，作業結果を充分に記述するにはある程度決まった文書量が入用である事によるのであろうか．文書量による規定については　　既に菅野氏も指摘[菅野 79]している．

２．２．３　インタフェイスの不揃いによる変動

図２．２．１[Koono 90c]に於て，特に前位工程での累積作業効率はバラツキが大きい．　この理由は工程間インタフェイスの不揃いである．　各工程は文書完成で区切られている．　初期工程を早くに終了したチームは，総体的に少ない文書量しか作らず，文書枚数当たりの作業効率は一定であるから，見かけ上作業を早く終えた．　所が後続作業で難航しているので，前位作業結果は不十分であった事がわかる．　文書化が不十分な時代であったので設計上流での文書化項目，記述の深さの規定が不十分な為にこのバラツキが起こった．　以上の事から
＊工程間のインタフェイスのバラツキ
（文書の項目／記載の程度バラツキ）
実質作業内容が前後する
作業内容が多／少にバラツク
作業手順がバラツク
事が区間内作業のバラツキをおこし，結果として作業効率のバラツキをもたらす
事が理解できる．

２．２．４　個人の能力差による作業効率の変動

ソフトウエア（開発）での個人作業効率バラツキは大きいといわれているが，その実績例は見かけない．　個人別評価の常識傾向を標準者で正規化した概念図を図２．２．２に示す．

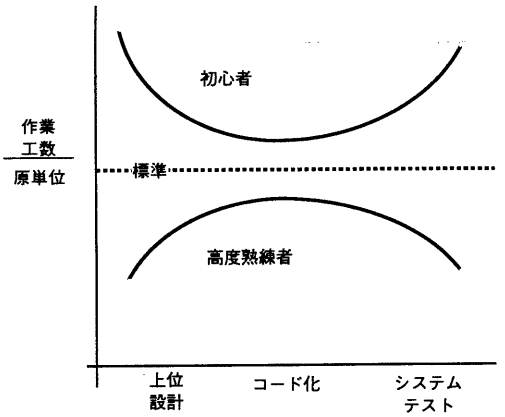

図２．２．２　個人能力別作業効率

熟練者は設計上流の作業で仕様，構成を早く決定し作業も手早い．　しかし初心者では「学習」，「調査」，「研究／検討」，「決定」の為に時間を取られ，以後の作業も遅く手戻りが多い．　上流ほどこの傾向は激しい．　最末期のシステムテストでも同様で，終に近づくほど差が顕著になる．　熟練者は現象の捕捉手段の知識が豊富で素早く判断し，現象を捕捉するとシステムの知識により適当な仮説を手早く立て原因を突き止める．　初心者は不十分な外部兆候から多数の仮説を立て多大の作業の後に不良原因が分かる．　中間工程のコード化では，必要知識の量は小さく，能力差はさほど大きくない．　図はこのような常識を図にした．　個人毎要素の影響は分かりにくいが，経験者比率等を取るとトレンドが明確になる事が知られている．[渡辺 82]

２．２．５　テスト作業での作業効率の変動

開発設計が終了後，作業結果を確認し正しい設計結果にする作業がある．　（正しい表現ではないが）一般にテストと言う．　テストの作業工数をプログラム規模で正規化した作業効率は，かなり大きなバラツキを示す．　テストとはいうが，その目的はテストをすることにより不良を検出／修正して正しい設計を確立する事であり，図２．２．３[Koono 90b]に示す作業を含んでいる．　大きなバラツキはこれらの作業構成要素を混在させる為に起る．

テストには，「テスト設計」，「テスト実行と照合」，（プログラムやテストデータに）不良が出た時には「不良解析と改変」の作業がある．　全く不良が無ければ前２者のみであり，不良があると後者も加わり一般に数回の（「テスト実行と照合」+「不良解析と改変」）が繰り返され，作業工数は大きくなる．　不良が少なくても，高品質を目指し高いテスト密度でテストすれば工数は増える．

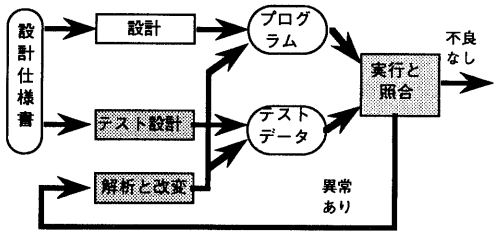

図２．２．３　テスト作業の構成

普通漠然と作業工数／プログラム規模を用いるが，この場合には構成要素毎に分けて次のように作業効率を定義するのが妥当であろう．

総合作業効率＝

｛テスト設計作業効率＋（実行と照合の作業効率）｝

ｘテスト項目数／プログラム規模

＋｛（実行と照合の作業効率）＋（解析と改変の作業効率）｝

ｘ不良数／プログラム規模

ここで

テスト設計作業効率＝作業時間／テスト項目数

実行と照合の作業効率

＝マシンテスト工数／テスト数

解析と改変の作業効率＝デバッグ工数／不良数．

これらの構成要素別の時間，回数から作業効率等を調べる事は研究目的で詳細な作業時間等の記録を採る場合には良いが，一般作業中に採取するのは煩わしい．　簡便にこれら要素別時間を求める例を図２．２．４[Koono 90b,c]に示す．

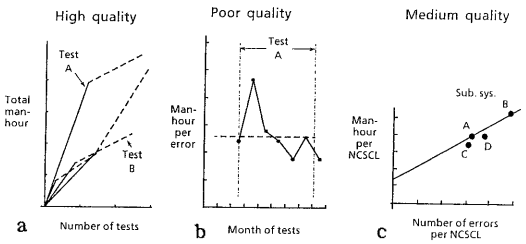

図２．２．４　テスト構成要素の分離

図ａは極めて品質が高くテストしても殆ど不良の出ない場合である．　累計テスト作業工数と累計テスト項目数の関係を表示して，勾配からテスト項目当たりの（「テスト設計」と「テストの実行と照合」）作業効率が近似的に求まる．　（テスト種別が変わると図の破線のように勾配は変わる．）

図ｂは極めて品質が悪い場合で，期間内の累計テスト作業工数と累計プログラム不良数の関係から（「テストの実行と照合」＋「不良解析と改変」）の合計近似作業効率が求まる．　（システムテストなど解析が困難になると作業効率は変わる．）

図cは中間的な場合で，横軸にプログラムの不良率を縦軸にプログラム規模当たりのテスト作業工数をとり，あるプロジェクト（同一作業基準／環境で作業した）複数のチームの成績を示す．同一作業基準／環境故，プロットは直線傾向線を示す．勾配はプログラム不良当たりの（「テスト実行と照合」＋「不良解析と改変」）の合計作業時間を示し，図の縦軸の切片は不良が無い時の「テスト設計」と「テストの実行と照合」の合計時間を示す．　テスト項目密度（テスト項目数／プログラム規模）で割れば（「テスト設計」と「テストの実行と照合」の合計）作業効率が求められる．

以上のようにテスト工程での作業効率のバラツキは説明でき，整理した要素別の各種指標は品質向上による作業効率向上，単位作業毎の作業効率を示し有用である．

2．2．6　集団のモラールによる変動

１９３０年台に有名なHawthorne Experimentにおいてハードウエア製造作業の作業効率にモラール（morale, やる気）が関係する事が知らた．　モラールによるバラツキは，単純な作業では約＋／－２０％程度の経験をもつ人が多い．

ソフトウエア開発でもモラールは大きな影響を与える．　短期的な例は設計工程の締切り日間近にチームメンバにハッパを掛けて（残業もあるが）驚異的スピードで期限に間に合せる事は第一線のリーダなら誰でも知っている．　長期的な場合には評価は難しい．　緊密なチームプレーによりとんでもないミスを事前防止できるとか，全員で智恵を出し困難を乗りきる等質的な例が多く，単純な効率面で測定しにくい．　モラールを向上させる方法[Koono 89a]もあるし，逆にモラールから製品の特性／作業効率を知る事も可能である．

2．3　作業工程の進歩による作業効率の向上

2．3．1　作業効率の習熟効果

人間には学習能力があるので，１回目より２回目....と作業効率が向上する．　向上は，始めは大きいが，次第に小さくなり，小さくなっても依然少しづつ進歩する．　（身近な例は各種の運動／ゲーム等の熟練である．）　これを「習熟効果」という．総経験回数Nを横軸に対数尺度で，作業効率Ｔ（Ｎ）を縦軸に対数尺度で表わす時，Ｔ（Ｎ）は

$$T(N) = K X^{-A}$$

但し，Ｋは第一サイクルの作業効率
Ｘは繰り返し数
Ａは習熟率から定まる定数

で表わされ，半対数用紙上で直線状になる．

習熟効果は，ＩＥの分野で「人」の「動作」を中心とするハードウエア製造作業の労働作業効率の進歩予測に応用された．　適用を拡大する内に，航空機全体の生産コスト等，複合的／大規模／複雑な対象についても適合する事が知られてきた．　最近では半導体生産コストの習熟効果が有名である．

2．3．2　開発作業における習熟効果

習熟効果は学習を基礎とするので，知的作業にも現われる．　図２．３．１[Anderson 80]は心理学での実測例で，２数の加算時間の習熟効果例を示しBlachburn [Blackburn 36]のデータをCrossmann [Crossman 59]がプロットしたものである．

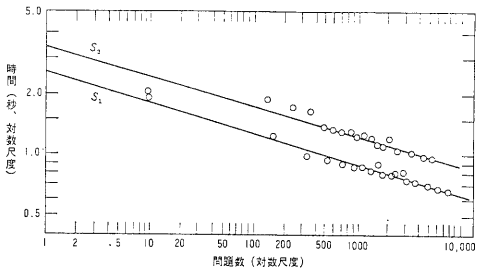

図２．３．１　２数加算の習熟曲線 [Anderson 80]

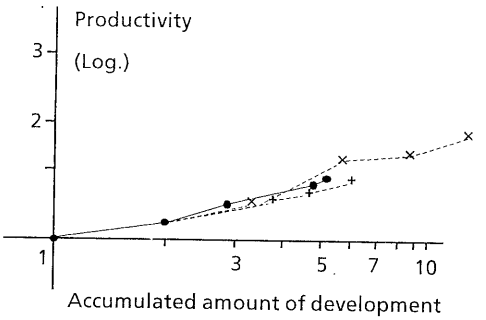

図２．３．２　ソフトウエア開発作業効率の習熟曲線 [Koono 90b]

ソフトウエア開発作業の作業効率[Koono 88, 90b]，のみならず品質[Koono 88]にも習熟効果が現われる．図２．３．２[Koono 90b]は各種のソフトウエア集団の作業効率の習熟曲線を示す．　図のように相互に略同勾配である．　（勾配の絶対値は「人」の単位的作業の習熟効果を示す図２．３．１と同程度である．）　習熟効果で向上するといっても，作業工程を進歩させず同一方法を繰返していては向上はない．進歩の初期は人の習熟でこの進歩は起こるが，ある段階からは系統的に経営資源を投入しないと進歩は止る．　とはいえ，人が作業工程の中心なので人間的な進歩と組み合わせなければまた停滞する．　これは図２．３．３[Koono 90b,c 原データはINPUT, Burns, Humphrey 89]に示すように，昔から組織論，進化論等で指摘されている．　（ここではバラツキを理解するに留め，進歩の手段と成熟自体については別の機会に譲る．）

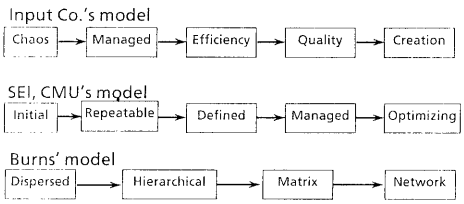

図２．３．３　各種の成熟モデル

３．　作業工程の構造的特性

３．１　作業工程の階層性

人間は全ての事を階層的に展開して理解／記憶／思考するから，作業を階層的に計画し，実行する習性がある．　ソフトウエア開発工程[Koono 87, 90a, 92]の中心部を例にとり図３．１．１に示す．（これは作業の構成（暗黙に作業結果の入出力結合関係）を示す．　作業は階層的に展開でき，展開を繰返すにつれて，作業内容は具体化し，詳細化する．　展開を続けると，最後には人間の思考や行動の単位要素に帰着する．　（これを推し進めれば人間の創造的過程も究明できる「河野 92]．）

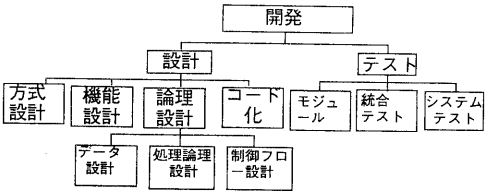

図３．１．１　開発の階層的展開

作業工程の階層性は，ソフトウエア[Koono 87, 90a]／ハードウエア[Koono 87, 90a]の開発や製造のみならず，一般のビジネスプロセス／間接事務作業一般に共通的に適用できる[Koono 90a]．　適用例として，「作業工程や組織は階層的に構成するべき」事が挙げられる．　この性質の為，階層的な構造化分析／構造化設計／構造化プログラミング／構造化テストとレビュー／構造化文書などが派生してくると理解できる．　図３．１．２[Koono 90a]にソフトウエアの作業工程を文書を中心に示す．

既に述べたように，バラツキ要因を追及していくと，どんどんと作業工程は階層的に細分化されていく．　研究の立場からは理解／分析が進むに過ぎないが，開発作業の管理者は作業の繰り返しの中でこのような進歩を行なっている．　すなはち，実績が予定／見込と食い違った場合には，その原因を究明して，例えば自分の管理／人の動かし方を反省／修正する．　これは個人レベルに留まるので，組織としては仕掛けにしていく．　具体的には

＊作業工程を階層的に細分化する
＊作業工程のインタフェイスである文書化の規定が（より詳細に）なされる

従来の１工程が名称区分とインタフェイスの両面で階層的に分割され，大きな観点で見て作業手順が前より詳細に規定されるが，必要なら更に

＊作業工程内での作業手順を決める．

このような改善の度に，作業のムラ／ムリが減り，漏れた要因が取り込まれ，各種の改善が人／仕掛け／設備の面で図られ，作業効率は向上していく．[Koono 89a,90a]　通常この外に期間，品質も向上を図る．　作業工程的な見方は，これは「標準化」という．「Koono 90a] この過程が２．３．２項の成熟の作業工程的な側面であるが，基本的に繰り返しとフイードバック[Koono 88'] が要件になる．

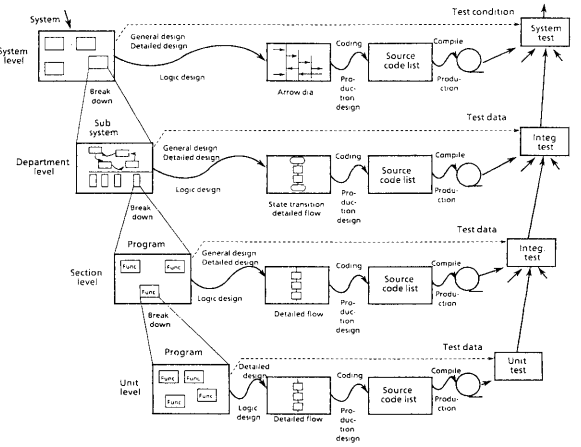

図３．１．２
網状の
ソフトウエア
開発の工程
[Koono 90a]

3．2　設計工程の共通構造

各設計工程は抽象化すると，図3．2．1に示す特性的構造を持つ．　設計の当初は何を如何に作るかといった「決定」中心であり，設計の末尾では例えばコーデイング／実装設計のようにある規約に基づく「変換」中心で，中間は両者の混合である．
これに先立ち「認識」があり，誤りを検知修正する「チェック」が付随する．　（これらは更に階層的に展開する事が出来る．）

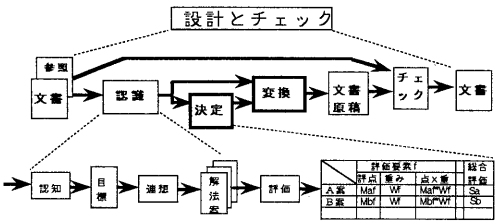

図3．2．1　設計の共通的構造 [Koono 88, 90c]

図3．2．1は作業効率の改善に何が影響するか具体的に理解する上で有用である．　作業をすべきかの条件が不備なら認識の確立は手間取るし，充分な関係知識を持たねば決定が遅延する，プログラム言語知識など基礎的技能が低ければ変換作業は能率的に進まない．

4．　検討

4．1　科学的／合理的な作業管理

現在のソフトウエアに必要な事は定量性に立脚した科学的／合理的な管理である．　現在のソフトウエア開発は人間を作業主体とし，それは人間の管理に外ならない面がある．　人間相手の困難性がいわれるが，かってのハードウエアも同様であった．

基本とする
　人の作業と生産対象の関係の定量的な把握
が行なわれ，ソフトウエア（開発）の各種の要因と結果の因果関係が把握できれば，ハードウエア製造作業におけると同様の科学的／合理的な管理を実現できる見込がある．　より複雑で，技能のみでなくて知識も関係するが，これらの相違を充分に認識すれば，ＩＥにおける科学的／合理的な管理手法はそのまま使える物が多い．　例えば
＊科学的な効率計測　　＊合理的な作業計画
＊確度の高い作業進捗管理　＊効率的な進捗統制
＊科学的な効率／品質管理と改善
などが挙げられる．　これらは何れも本報告の基本的性質から誘導できる見込である．

4．2　ソフトウエア生産性との関係

本報告は，ソフトウエア開発の為の経営資源（人を中心に論じているが計算機使用時間も同様に扱える）を定量的かつ合理的に，より高い確度と精度で管理する事を目的としており，生産性の定量化を意図してはいない．

ソフトウエア生産性とは，厳密には

$$\frac{\text{ソフトウエア製品の価値}}{\text{ソフトウエア開発コスト}}$$

である．　しかし製品の価値は計測が困難なので，ソフトウエア生産性として便宜上やむなく

$$\frac{\text{ソフトウエア規模}}{\text{ソフトウエア開発コスト}}$$

がよく用いられる．　この尺度は厳密でないにせよ，管理指標として手軽で有用である．

そこで，かような事を承知で用いる場合には，最終的な正規化がソフトウエア規模になるようにすればよい．

前記の所謂生産性と厳密な生産性の乖離の認識の為に，同一機能に対するソフトウエア規模のバラツキの例を示す．
DeMarcoは同一機能仕様を一般のプログラマに与え，出来上がったＥＤＰプログラムの規模の最小と最大では8倍に達する中心の片寄った規模分布例[DeMarco 89]を報告している．　筆者らは構造化設計の教育後の同一演習問題に対するプログラム規模分布を調べた所，方法を統制するので明確な中心の周囲にバラツク事が認められた．　方法で統制しても中心に対して最小は1／2倍，最大は2倍，最大と最小は4倍程度のバラツキが依然あった．[Koono 91]

これらのバラツキは，作業効率のバラツキより大きい．　ソフトウエア規模のバラツキを小さくして適度な規模にする事は極めて重要であり，それは製品の標準化，構造の標準化，作業手法の標準化，つまり作業工程の標準化に結び付いている事[Koono 90a]に注目を要する．

4．結び　（科学的な管理を目指して）

本報告は，ソフトウエア（開発）作業の科学的／合理的な管理の前提として，作業効率の問題を中心に如何に定量的に捕捉するかを報告した．

日本のソフトウエア産業の中核はハードウエア開発製造も行なう企業である．　ソフトウエア生産管理はＩＥの成果であるハードウエア開発／製造管理体系に準じて構築され，実質的な技術移転ができている．　またハードウエアで確立されたTotal Quality Controlがソフトウエアにも適用／定着してきて，世界トップレベルの産業水準に迫っていると考えられる．　産業界の指導者／経験者，Total Quality Controlの実践経験を整理した発表が待たれる．

これらの成果がアカデミアの場に出され，科学的／合理的な体系へ洗練される事が必要であろう．この分野は現在境界領域として放置されているが，Industrial Engineer，行動科学や認知科学の研究者が積極的に参入し共同で研究する必要があると思われる．　かくして，実体の影，作業結果のプログラム中心／ソフトウエアサイエンス中心のソフトウエアエンジニアリングから，作業の実体，人間中心のソフトウエアエンジニエリングへの発展もあろう．

5．謝辞

筆者の一人河野が１９６４年日立製作所戸塚工場へ配属以来，多くのIndustrial Engineerとハードウエア製造現場，ハードウエア／ソフトウエア開発のベテラン各位の教えを受けた．　多数の第一線の設計者各位には面倒な各種実績計測のご協力を頂いた．小生の研究を理解し育成された上長各位のご厚意も有難いことであった．　関係各位に厚くお礼申し上げます．

6．参考文献

[Anderson 80] Anderson, J. R. :Cognitive psychology and its applications, W. H. Freeman and Company, 1980.
邦訳　富田，増井，川崎，岸訳：認知心理学概論，誠信書房１９８２．

[Blackburn 36] Blackburn, J. M. : Acquisition of skill : An analysis of learning curves, IHRB Report, 1936.

[Crossman 59] Crossman, E. R. F. W. : A theory of the acquisition of speed-skill, Ergonomics, 1959, 2, 153-166.

[DeMarco 89] Demarco, T. et al, :Software development; State of the art vs. state of the practice, International Conference on Software Engineering, 1989.

[Humphrey 89] Humphrey, W., Kitson, D. and Kasse, T., :The state of software engineering practice: A preliminary report", International Conference on Softwar Engineering 1989.

[INPUT] INPUT comany, :A report from INPUT Company, 1980's.

[菅野 79] 菅野：ソフトウエアエンジニアリング，日科技連出版社，１９７９．

[Koono 87] Koono, Z., Ashihara, K. and Soga, M., :Structural Way of Thinking as Applied to Development, IEEE/IEICE Global Telecommunications Conference 1987.

[Koono 88] Koono, Z., Igawa, K. and Soga M., :Structural Way of Thinking as Applied to Improvement Process, IEEE Global Telecommunications Conference 1988.

[Koono 88'] Koono, Z.,Toiya, M., Matsuida, T. and Soga M., :In-Service quality improvement activity, IEEE Journal on Selected Areas in Communications, Vol. 6, No. 8, October 1988.

[Koono 89a] Koono, Z., Yonezu, Y., Iwata, Y. and Soga, M., : Experiences in applying SDL, the Fourth SDL Forum, 1989.

[Koono 90a] Koono, Z.and Soga, M., : Structural way of thinking as applied to quality assurance management, IEEE Journal on Selected Areas in Communications, Vol. 8, No. 2, pp. 291 - 300, Feburary 1990.

[Koono 90b] Koono, Z., Tsuji, H. and Soga, M., :Structural Way of Thinking as Applied to Productivity, IEEE International Conference on Communications 1990.

[Koono 90c] Koono, Z., Process aspect of developments, 1990 Joint Conference on Communications, Networks, Switching Systems and Satellite Communications, 1990.

[Koono 91] Koono, Z., : Structural way of thinking as applied to good design, (Part 1, Software size), IEEE Global Telecommunications Conference 1991.

[河野 92] 河野，馬場，薮内，内藤，重森，宮，：ソフトウエアクリエーション --系統的な取り組み--，信学技報，KBSE92-28(1992-09).

[Watanabe 82] 渡辺，緒方，：ソフトウエアの品質および生産性予測法の一事例について，第２回ソフトウエア生産における品質管理シンポジウム，１９８２．