Vol.2010-SE-170 No.8
2010/11/11

受注システムの開発方式

河野善彌† 陳慧††

本論文は，初めにソフトウエアは如何に開発すべきか検討する．自由競争市場の商品のソフトウエアと自社内用のソフトウエアの２場合がある．前者では，その良さで事業が左右されるから，基本的には外部に委ねがたい．用途に応じて的確なソフトウエアを作ることを詳細に論じた．

How to Develop a Tailored System

Zenya Koono† and Hui Chen††

At first, this paper discusses how a company should develop a software system. There are two cases, first one is to sell a system in a free but competing market, and another is a system for own internal use. In the former case, as it governs the success of the business; it should be developed by own best people. This paper details how the target should be aimed at.

1. はじめに

「要求工学」とはソフトウエア開発での要求仕様化プロセスを工学的に定式化する技術という．因果関係から云えば，対象である etwas が存在すれば，該対象が記述できる．対象以前に「要求を如何に定義するか」は仮想的課題で意味が乏しく，ソフトウエア工学では「何を作るのか」の確実さが最大の課題と考える．

企業がソフトウエアシステムを得るには，１）既成システムを購入する，２）自社開発する，３）依頼開発する，の３方策がある．１の購入の手順は既に存在する．２の自社開発の標準的な道筋は略確立されている．しかし３の依頼開発の方式は確立済みとは云い難い．歴史的に見ると，最初期のソフトウエア工学は「ソフトウエア化するに当たり，全要求をソフトウエア業者にぶっつけ，ソフトウエア業者に最善案を提案させる，と聞いた．発注者側で提案を評価して，最善の提案を選び，指導しながら希望するシステムを開発させた」とも聞いた．これに準拠して，雨霰と飛んでくる注文の数々を聞かされ，中心を狙って当てた成功例もある．しかし，必ずしも成功率は高くない．そこで，「外部に発注するなら，如何にするか」は確かに一つの課題である．

前記のように，

I. そもそも何を注文するのかが最大問題ではなかろうか？ 次には

II.発注側の立場と受注側の相矛盾する立場が二つある．

どのようにすれば，社会的にも，両当事者にも不都合がないシステムにできるかの問題がある．これはソフトウエア工学には深く関わるが，ソフトウエア工学側で今議論されていることで何の進展が見込めるのか，理解しがたい．

社会には多様な要求と厳しい諸制約条件があり，長い歴史のうちに標準形式が出来上がる．まず依頼側と受託側の要求条件の合致が確認されて詳細に入り，両者の納得する仕様書で契約する，のが常識あろう．James Martin は 1980 年代から情報系のトップダウン開発を主張しており,1990 年に出版した[1]に CASE ツールや自動生成システムを織込んだトップダウンな取組を詳説している．

しかし，システム化が一巡した近年，既存システムの変更や維持更新等の小型件数が増え,ボトムアップの傾向が強まった．アジャイル開発宣言は，「システムの開発は{システム技術者 - プログラマーが作る既存路線}を崩して,プログラマーがシステムを作る」宣言とも読める．要求工学も要件工学ではなくその後の詳細仕様を狙うとも見える．

この論文の立脚点は以下である．「システムを作る」機運は，トップダウンでもボトムアップでもよい. しかし, 企業～機関が開発を事業経費で実行するなら，権限と責任は組織の原理に従いトップダウンになる．具体的には，

発注担当幹部は会社にその投資をさせた責任を取る，従って

発注担当幹部は，その発注についての最高の権限を持つ．

それでは全体を技術的に纏めるには如何するか？ 最も明解に理解できる，自社開発の場合を初めに検討し，実際的な工程の例も説明する．その上で，受注開発を如何に行うかを考える．

† クリエーション プロジェクト
Creation Project, http://www.creationproj.org, koono@vesta.ocn.ne.jp
†† 国士舘大学，情報科学センター
Center for Information Science, Kokushikan University

Vol.2010-SE-170 No.8
2010/11/11

２．自主開発

２．１製品～システムの自社開発

本節は企業の自社の商品開発，即ち自社の収益を支配する重要な柱～商品を立上げる場合を考える．ここでは，市場での競争に勝つこと，即ち，主要競争社の最強力商品に優ることが第一の必須条件である．売価 - 原価 - 資本の関係を図 1[2]の粗い原理図で説明する．

図の縦軸の上部は製品売上など入ってくる金額を正方向に，下部は製品開発に投じる資金の総合計金額を負方向に示す．図の横軸は年を単位に取り，左方負の軸は開発着手から起算して原点 0 で開発を終えて販売に移行し，右方正の軸では原点 0 で販売開始した後の時間軸と単純化する．

図の水平軸 Y= -2 の時点で製品開発に着手した．以後の開発費の累計額が第３象限に表示され，総額は D であった．原点 0 で発売した後の販売状況は第一象限に表示される．右上に伸びる太破線 S は売上，同じく実線は原価 C で，粗利 P＝売上−原価である．開発費総額 D は各期毎の粗利 P で回収されて行き，D － P 曲線が水平軸と交差した以後から正味の利益が得られる．

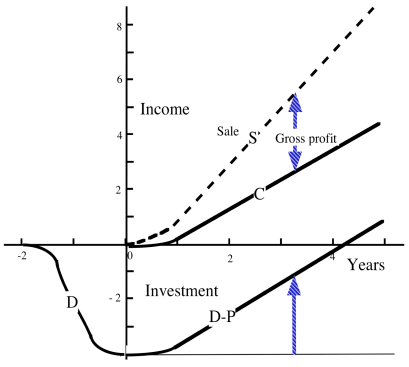

図 1 事業計画骨子図

他社に勝つには如何にあるべきなのか？図１により利益を増減大小する要因を調べる．簡単の為に，図の原価は全てハードウエア原価と簡単化する．

原価＝部品個数×部品単価

である．設計者を始め全てのハードウエア人は，

少しでも部品の数を減らし，
少しでも安い部品を使う

ことに目を血走らせる．これを成遂げる力が技術の高さであり利益の基本でもある．

ソフトウエアでも同様で

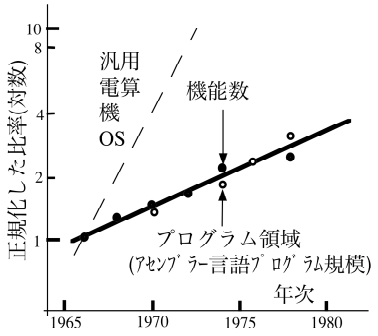

図２　規模の増殖性

開発費＝規模×生産性

であるから，ソフトウエア人も，

少しでも規模を減らし，
少しでも生産性を上げる

ことで，図の D-P を引下げる．

これは金科玉条である．これに加え，売上 S を増やすには，（規模は増えがちだが）

優れた機能を増す

ことが，前記に加えてソフトウエア人の重大な責任である．すなわち，設計者は

規模削減，生産性向上の重荷の他に製品拡販の重責を負う．

図２[3]を説明する．図の横軸は年次，縦軸は対数尺度で表示してあり，ソフトウエア規模は初年度の規模で正規化した．実線は計算機制御による電話交換システムで，現在の組込システムの元祖である．図の黒丸は交換システムの規模，白丸は交換システムのサービス（正確には feature 数）を示し，年々約 10%で増加する1．この増殖率は 1990 年代でも同率で，1990 年代には各社の母胎システムは Mega 行の規模になったが，1965 年からの増殖率の 17 倍と一致した．

図２は機能の黒丸と機能の白丸とが同一傾向線で並ぶから，母胎の規模が大きいと，年々のソフトウエア規模の増殖の為のソフトウエア規模拡張が大きな負担になる．上記２件を併せて，

ソフトウエア規模の縮小と生産性の向上はハードウエアの場合よりもより厳しくなければならない．

経営には profit center と云う概念がある．profit center とは，利益の総合的な全責任を持つ部署である．重責を負うから，profit center には全体に指示し統率する責任を持たせる．世界全体が大きくソフトウエア化の方向に進んでいることは皆承知している．この行着く所は下記である．

ソフトウエアを握るものは profit center であり，企業の利益の源泉

1　原資料は Bell Telephone Laboratory の発表した直線尺度の傾向図で，河野が定率増殖性を見出した．各種通信システムも同傾向で，増殖性は技術と社会の進歩による．細破線は同年次の汎用電算機 OS の規模の増殖性を示す．勾配が大きく違うが，当時は DB, Data Communication, 等の OS の大幅な機能追加の時期であった．ある 1 ジャンルの 10%/年の増殖と各種ジャンル（機能）追加により母胎規模が加わる事の混合特性と推定した．2000 年ころから携帯電話等のソフトウエア規模の急増加が問題になった．前記の混合特性と判った

2.2 社内システムの自社開発

自社の会計や経理，顧客管理等のシステムを自社で開発する場合を検討する．これらの目的は社内の間接[2]作業の合理化が多い．図 3[4]は直接及び間接作業例を IPO 図に倣って書いた．図 a は直接作業，図 c は間接作業である．図 b のように加工する機械を使えば，人の作業は機械で代替できる．同様にプログラムで計算機を動作させれば，図 c の人の間接作業もシステムで代替できる．

両者を比較すると，機械作業の方が，

・何時でも・高速でかつムラなく作業でき・誤り率は飛躍的に低下

など作業が合理化する．これは直接作業でも間接業務でも同じである．

図４はその効果を示す図で，直接作業の場合を説明するが，間接作業でも同じである．図の横軸は作業の経過日数，縦軸は稼働する人と省力機械/省力システムの合計金額である．同一作業量で人が作業する時，労務費用は直線的に増加する．これを右上方に伸びる直線（旧）で表す．次に，例えば 1/3 の作業を人から機械に移す．新直線の勾配は旧の場合の 2/3 に低下するが，機械を設備する費用がかさむ．図４では縦軸に設備投資した金額を切片で表した．設備投資額が小さい場合を茶色，約２倍金額に増した場合を緑色で示した．各切片の最上部から出発する費用増加カーブの勾配は，設備投資額が大きい（設置した機械数が多い）程，旧勾配より小さくなる．この新の線と旧の線との交差点迄は機械化/合理化した場合の方が経費支出合計はより大きいが，交差点を過ぎるとより小さくなるから，数年間の内には投資は回収[3]できる．

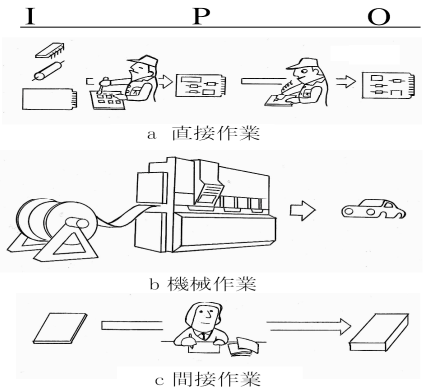

図 3　人と機械の作業

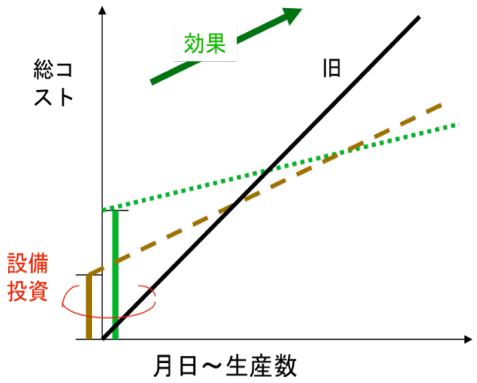

図 4　人の作業と自動化

2 製造作業など製造対象物に行う作業を直接作業，直接でない事務作業を間接作業と云う．

この場合でもソフトウエアは，最小限度でも年々の法制の変化や自組織内での不可避な増殖性を示すから，

ソフトウエア規模最小・ソフトウエア生産性最大

が前提的な原則に変わりない．新しいポイントは，製品ソフトウエアと同様であり，企業の利益を最大化する為に，どの業務をソフトウエア化するのか，その為に如何に組織を移行させるか，等に掛る．

プログラムをいじる時期ではない．全社を如何に合理化するか，

が課された使命である．製品部門が profit center なら，

社内システム部署は，（間接費中心の）cost center になることであり，社内各間接部署の様相を各種統計で把握できるから，現地に赴いて実態を掌握し，必要なら経営トップに速報を上げる．次回の改善投資の計画会議には，間接員作業実態の報告をして改善投資を提案する．（プログラム関係は各社の子会社に委ねれば良い．）

しかし，厳しさは製品ソフトウエアに比べるとより低く，幅のある対応が出来る．

歴史的には，自社内システムで合理化を図る創建期のパイオニアは各社ともに Punch Card System （PCS）を支えた先輩達であり，PCS→初期のコンピュータ＋アッセンブラ期コンピュータ→汎用コンピュータ＋COBOL と変遷の時期を力強く開拓されたのであった．今同様な変化の時期にある．原動力は社内の**システム部署**である．これら往時の諸先達の在り方を学んで欲しいものだる．

3.　経営/ソフトウエア/ハードウエアは全て同根

人は或る「対象」の実現を意図すると，次には実現方策を連想し，更に詳細化を繰返し続ける．例えば「牛丼を食べる」なら，次には，「１階に降り建物から出る」，「３丁目の角を曲がり○○屋に入る」，「牛丼を注文する」と階層展開し具体化する．これは親概念から子概念への階層的展開であり，また人の単位的な記

3 資金を投じた設備は年々損耗すると考え，設備の簿記上で減価して行き，この減価分の金額を減価償却費として積立て回収する．回収年限は税法で定められ，毎年定率あるいは定額を積立てる．社内規定として使用の度の回数あるいは使用時間にリンクさせた金額を使用料金として課する等を行なって回収を早める等をする

Vol.2010-SE-170 No.8
2010/11/11

憶動作でもあり，展開され詳細化される過程でもある．勿論，この種の人の実際の行動は，断片的であったり，各種が雑多に並行したりする．この階層展開はかなり顕著なので，色々な分野で早くに気付かれ，更に定式化されたりしている．以下はこの展開を単機能的にする等純化したものである．

図５[5]は３代表例を示す．軍事科学では「目的の階層性」[11]と呼ばれ，戦争計画の原理とされる．図ａのように国権の長（例：国王）は統合参謀総長に，戦争の最終目的を課する．かれは勝利する方策(戦略)を立案し，陸海空の３軍の司令官に司令する．各軍司令官は，課された作戦行動を（自分の目的と受止め）より具体的な実現策に展開し部下に指令する．これは企業での経営と同等であり，組織の構成はこの展開に合致することに注意されたい．図ａの展開連鎖は最後には１兵卒の単位的な動作迄展開される．最上位レベルは戦略に当たり，戦争の成否を握るから，高級将校には教育訓練する．下級者はそれぞれの軍の行動規範と併せて具体的な行動を訓練する．

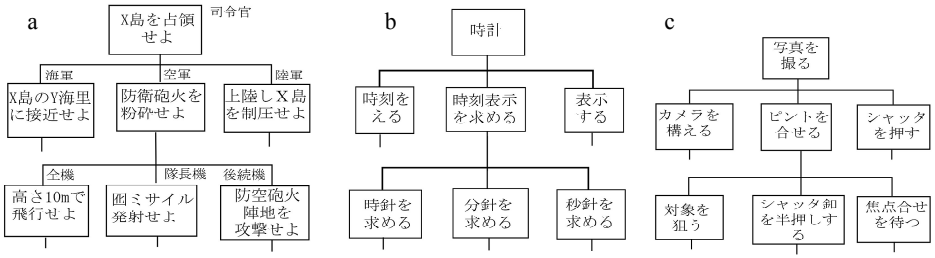

図５　各種の展開例

図ｂは前記階層展開連鎖の中位段階でのある機能展開例である．図ｂは所謂「構造化設計」の例でもあり，機能を階層的に展開する．図ｂの詳細に先立ち，図ｃを見る．図ｂも図ｃも同様な展開を続けることがわかる．これは身体的な作業[4]の例であり，ご覧になれば容易に理解できる．これは電気での「オームの法則」と同様な経験則であり，(筆者らが度々公開しているが，学会誌レベルでは証明が未了である.) IE の創始者 F. W. Taylor が経験則とも断らずに使ったことから，産業界では基本原則として使われており，約 100 年位未証明であった.(工学ではその効果により技術提案を評価する．理論証明は草創期には不可能だから不要．理学では無いから証明は不要.)

図ｂの構造化設計は実行が難しく，空虚な機能を作り易い等の欠陥があり，データフローを併用する DOA (Data Oriented Approach) として使われる．より厳密で正確で容易に出来る展開方式を考え図６[4]を得た．これを「拡張構造化設計」と名付けよう．これを用いて，展開の様相を調べる.(既報であり省略して記す.)図のデータフロー図は最強力な設計図面で，図 5b の機能部分を含む．第１段の仕様「時計」は第２段で入出力データを定義される．入力と出力の両データで統制したデータフローの機能箱でできる単位的データフローを標準単位とし，箱内には結果/外部/機能的に表記する．第３段への階層展開は Myers の STS 分割[6]によって入力と出力側の両最大抽象点が選ばれる．第３段は第２段の単位データフローを詳細化したデータフロー図で，単位データフロー図群で構成される．

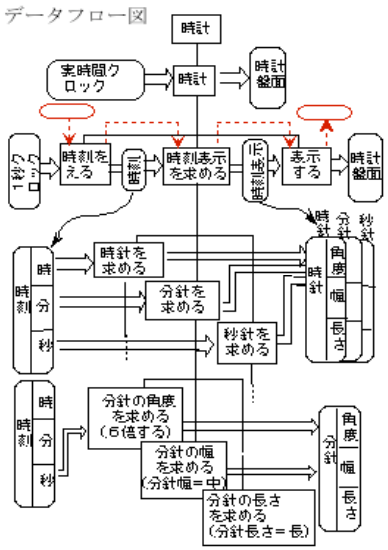

図６　時計プログラムの設計

第３段の３単位データフローの中央の「時刻表示を求める」は，第４段に下る前に，「時刻」や「時刻表示」を階層展開して子データ群を作り，「時刻表示を求める」を３並列な単位データフロー図群に階層展開する．これはジャクソンプログラム設計法[7]のパターンである．

次に第５段に移る．時針，分針，秒針の３種を幅と長さで区別して図のデータフローにする．充分に単位化したから，次には（6 倍にする等のように）対応するソースコードに書換えられる.(これはハードウエア論理回路図にも，書換えられる．これまでの記述はハードウエア論理でもある)

設計するには，(科学技術用語も含む) 自然言語の意味に着目した概念の階層的展開を繰返し詳細化する．最後に，(コードや論理回路と云った) 実現手段に置換する．これは，ヒトの知のレベルであるから経営段階ー設計段階ー身体的行動に共通する．

従来「自然言語は誤りが避けられないから，使用するに耐えない」と云われて来た．これは真っ赤な嘘であった．図 5a の最上部から図 5c の最下部迄の，概

4 ハードウエア製造作業では，19 世紀末～20 世紀の初期に F. W. Taylor や Gilbreth 夫妻が経験則として作業（工程）の階層展開と統合を利用して，定量的科学的合理的な工学，Industrial Engineering，(IE，経営工学)を創始した．以後，工程の階層展開～統合は，(電気系のオームの法則のように) 最も当たり前の関係式として自由自在に駆使され，これは名称も無い経験則であり，理論根拠も無いままで推移してきた．

念展開連鎖での詳細化の後に実現手段であるコードに到達する．従ってソースコードは色々の利用の道は多いが，要するに 1 実現手段に過ぎず，本質ではありえない．

ソフトウエア工学でプログラムの合成はできるが，ソフトウエアの自動生成は全て失敗している．筆者等は図 6 の単位データフロー図を親とし，子である単位データフロー群である展開結果の詳細データフローを得る自動設計方式を実現し，中国および米国特許を得ている[5]．現在研究は中断状態なので極力早くに再開したい．

図 6 の各展開は等比級数状の階層展開網状である．したがって経営レベル，設計レベルは身体的行動レベルと同特性を示す．図 7 [8]はソフトウエアについての実績の打点と傾向線群を示した図である．横軸に成果物数，縦軸に工数をとってある．

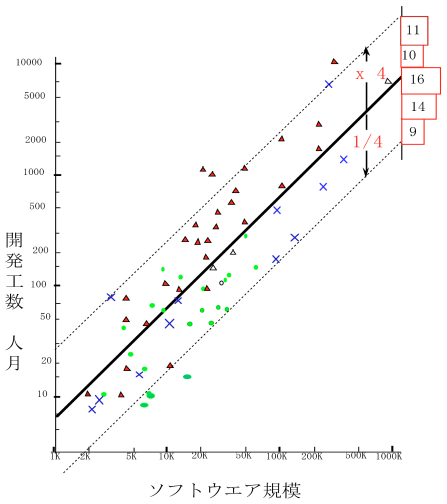

図 7　　規模対工数の特性

中央傾向線は直線状でソフトウエア規模∝工数を示している．貴方が 50 文字書く時間は 5 文字書く時間の 10 倍であることはご理解頂けよう．ソフトウエアの場合も，ハードウエア製造の場合でも全く同じ関係が成立つ．ハードウエアでは Industrial Engineering, IE 経営工学が，これら定量的関係を用いて，生産性を向上させ，品質を管理して，1980 年代の日本のハードウエア産業は世界トップレベルに躍り出た．ソフトウエアについても，基本的に同じである．全て合理的定量的科学的に行えば良い．

> 工学では，全て現状現物現場が基本で，全てを数/量で判断する．定量的な扱いを行わないなら，何がより良いのか評価判断できない．故に定量性を欠くなら，それは工学ではありえず，科学的でもありえない．

図に帯状の大きなバラツキがある．対数尺度で一定幅であり，右上の棒グラフが示すように釣鐘状であるから，これは数学的には対数正規分布と云う．最小の点は中央値の 1/4 から始まって垂直に上り，最大の点は中央の 4 倍に至る．この間に殆どが収まっている．対数正規分布は，数多くの影響因子があってバラバラな特性である時に表れる．故に経営/設計/身体的等の人の行う全行動に表れる．

> 図の帯状領域をある規模の垂直平面で切ると，その平面には帯状領域が正規分布状に写る，これはある規模での所要工数の分布であり，その下端と上端の比（生産性の比でもある）は 1: 4^2 = 1:16 に達する．言換えると，**生産性の最大最小比は 4^2 =16 倍に達する．**

ハードウエア製造でもナイーブな状態では同様にバラツキが表れるが，20 世紀初頭から半世紀余り 1960 年代には見掛けなくなった．ソフトウエアもプロセスを標準化し，次に自動化に早く進むべきである．

＊「ソフトウエアは他とは違います」と云う考えは，最終的には必ず破れる．（現在の所謂ソフトウエア工学の「謎」や定量性の欠如は既に纏めて報告した．
＊概念の階層展開連鎖は，機能面からは如何なる対象の設計についても当て嵌まる．但し，ある種の工学的設計は，設計上の拘束条件や機能設計以外の最適化が必要になる．その場合には，これらを並行させねばならない．

4．技術と研究

現在では，高い技術力を持つ企業が大きな利益を上げることはよく知られている．その技術力は研究により築かれ，製品開発に使用されて，効力を発揮する．企業の技術に対する姿勢は，研究開発比率（研究開発費/売上高）の高さに表れる．図 8 [2 原データは 9]は欧米での資料であるが，特殊事情で大きな比率を示す医療製薬関係に次ぐ立場は，ソフトウエア～コンピュータである．欧米のパッケージソフト会社は，Microsoft を始め極めて高い比率である．しかし，左下からの矢印は日本のソフトウエアベンダーの平均値で約 1%[6]に過ぎず異例に低い．（国際水準で比較して品質は最

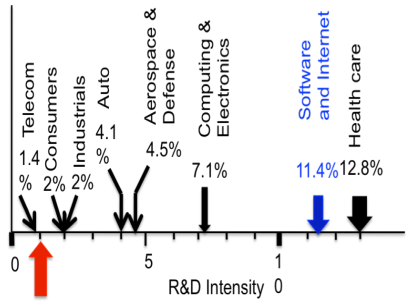

図 8 技術に対する姿勢

5 これは概念の階層展開連鎖を政府レベルで公認したと云える．

6 日本の代表的なソフトウエアベンダー約 600 社で構成される情報サービス産業協会（JISA）が公開している毎年の研究開発比率は 1994 年から 2009 年の長期にわたり 0.47～1.72%に過ぎない．1%はこれら

Vol.2010-SE-170 No.8
2010/11/11

高と云われるが,）この労働集約的な傾向は良いのであろうか？

ベンダの社員はソフトウエアの規模縮小の意識が無い．システムのレベルでは,
下記の実例[13]がある．

ベンダー的設計例： 一応の削減努力例 ： 削減研究結果 = 3.4 : 1 : 0.49

先の対数正規分布の生産性 $4^2 = 16$ ほどではないが，実力は $3^2 = 9$ 程度と思われる．**これでは商品事業は営めない．** 説明を求めると，「会社から『云われたらそのようにすぐに作れ』,『云われた仕様を検討して論理を整理して作る等はするな』といわれる」と云う．研究開発比率の低さと同根と思われる．

５．自社用システムの開発方式

図 9 新製品開発育成図[10]は「新 QC 七つ道具」をツールとする論文から引用した．ここでは商品の開発工程例として利用させて頂く．詳細は是非原論文を参照されたい．

図枠の左端は，R&D の初めからの工程区分を上から下へ並べた．図枠の最上部右端には，市場，次いで経営の最前線の常務取締役を頭とする常務会（実務面で経営レベルの意思決定をする）があり，企業レベルの決定は常務会の列の□箱中に○○決済と示されている．図枠の右方向には社内の実務組織に細区分されている．R&D の具体的な流れは矢印で示してある．

開発は３段階に区切られ，各段毎の設計審査で終了判定が下される．この設計審査は米国政府 NASA で行う方式に似て，区切り毎に QCD 等が設計審査（Design Review)され，確実さを確保する．

１．（図の最上部に示す）市場～顧客の情報と研究所で掴んだ技術情報に基づき，戦略を持った上で新製品開発の認可を出す．
２．戦略に基づく研究を行い，また市場を調査する．
３．研究結果に基づき，全体開発を行い，市場調査，販売計画を立案して商品化を決定し，量産とその品質を管理して，量産の流れを作る．

第３段階の流れ図から，品質関係に重点が置かれチェックが充実しており，次いで市場調査や販売計画がきめ細かに行われる．各種の問題を経験し対策を行ない，再発防止を明確にして積上げる．

第１段階の後に各具体作業が流れる．図では原理レベルなので，勝手ながら

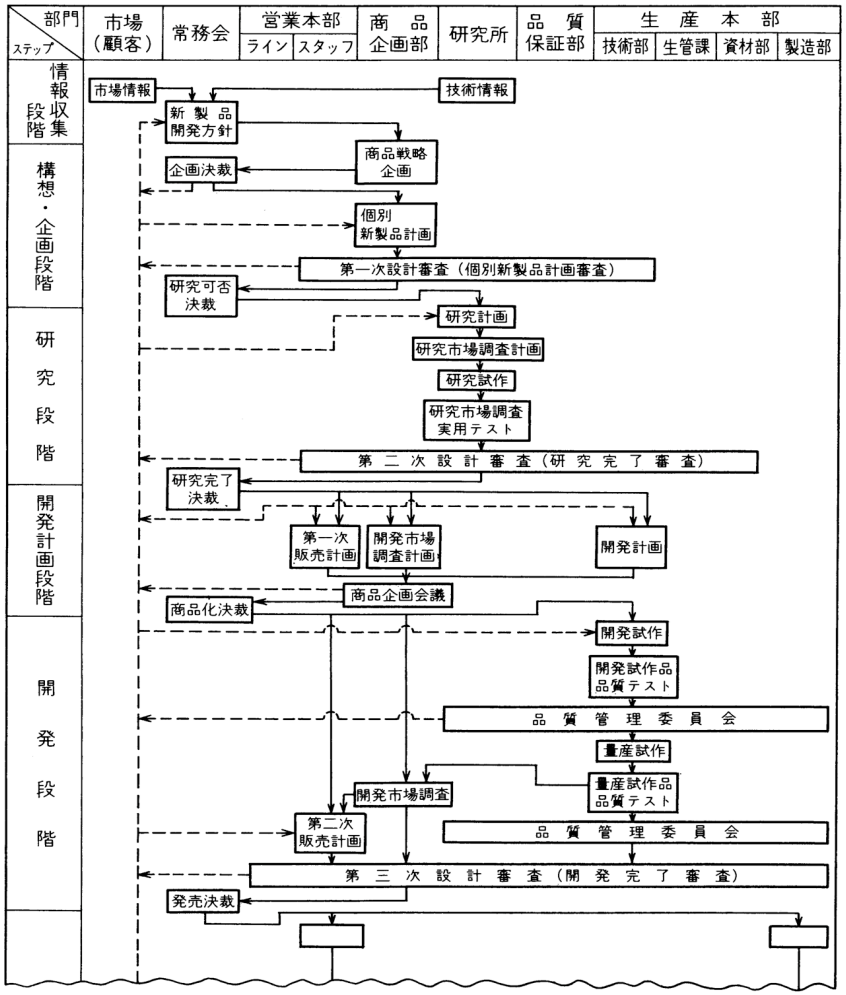

図９　新製品開発育成図（(株）サクラクレパスの例）

常識的な肉付けを試みる．第１段階の終りは成算ありで実質 GO だから，ここに

を代表する値とした．

至る迄の検討は数多い．ここに至る迄に，自社開発なら製品或は社内システムの開発～事業計画書（目論見）が承認された段階に当たり，また依頼開発なら，両社幹部で内示を伝える．製品なら設計は製品仕様書（目論見），時期毎製品価格，利益等にコミットし，営業部門は前記をサポートする時期別売上にコミットし，経理は時期別プロジェクト要員数や支出にコミットし，経営幹部は要因計画や支出計画にコミットする．社内システムなら，設備系なら経理がコミットした時期別支出と部分導入等に基づき，投資推進元は導入計画と効果（人員減や経費減）にコミットする．

製品は自由競争市場で競争に打勝って始めて満足に営業できる．競争に勝つには,如何にすべきか？図 9 の左上に商品戦略の企画と記されている.戦略とは，それが発動されると自分が自動的に勝つ／あるいは相手が立場的に負ける態勢になるよう計画を云う． 3 節で説明した意図的行動の展開連鎖の最上部は最終目的 :「○に勝つ」でもよいが，最終的な目的を表記する．この戦略を実現する手段群（2~4 項目で記す）を考えて各々を表記する．それは親概念と数個の子概念からなる戦略の立案である．次には，戦略を構成する実現手段/子概念を実現する方策を考える．これを続けて図 6 のように階層展開を繰返す．最後には,「ある作業の為に○，▽と□をする」のように具体化する．出来たら下から順に積上げて行き,最後には戦略発動のボタンを押せば良い,所迄準備して機の熟する時を待つ．

仮想的な戦略をお見せする．世界に先駆けて，自社で実用化中のソフトウエア自動設計システムの計画を１～２年繰上げ戦略武器とする．競争会社がこれから数年間掛けて機能拡張する程度の機能増加を圧倒できる多数の機能群をソフトウエア側で用意し，ハードウエアに関しては通常の 128bit MPU の quad system を使い，この市場の殆ど世界中の１年分に相当する数量を一挙に作るべく，あるEMS (電子機器の受託製造会社)に発注する．ハードウエアは量産規模が大きいからハード原価は格安になり，ソフトウエアは生産性が高いからコストアップにはならない．どの製品よりも圧倒的に機能は優れているから，競合社のどの製品よりも格安に売出し，全競争社を圧倒し，各社に売上を立てさせず，市場から撤退させる．これはノルマンデイ上陸作戦にも似た総力戦である．

開発戦略で競争会社群を落とした実例を紹介する．ある設計リーダは，チーム全員に猛勉強させ，徹底的に安い部品を少ない数の部品数でシステムを作り上げ，破格の安値で市場を押さえようと仕掛け，順調に滑り出した．競合社は現有機種では太刀打ちできない．各社が対抗機種を考えて開発に着手し，受注し始める等で引き返せない時点に，爆弾宣言を出した．この製品の能力は最初の公称値の２倍であったことを示し，更に安値で受注し始めた．以後これは市場で圧倒的な強さを誇ったと云う．図９の戦略企画とはこのような勝利を得るものである．戦略の失敗は戦術では補えない,と云い,失敗もリスクとして考えねばならない．

商品の世界では，市場調査から市場予測と政府その他の生産状況資料，各社の各機種毎の特徴集や売上金額～台数等，各社毎の特許や学会等の公開資料，これらを揃えて情勢予測などの技術を保有しているのが普通であり，企画や計画の出来るスタッフや経験深い設計経験者等が居るので，戦略/戦術/製品計画などを立案させる．それら蓄積技術が充分でないなら，その知識と能力の人々が持つ知恵を使う．大きな市場が出来ると見込まれ，前記の蓄積等が無い企業が幾つか居ると見られる場合には，コンサルタント（あるいは会社）が状況説明資料や計画案を PR しに来るとか，レポートを売りつけに来る．それらが無い場合には，その市場，業界と技術に明るいエキスパートに協力を依頼する．

製品開発は予見しがたい敵が居て戦闘的であるが，社内システム開発なら一応は情報が充分にある社内が考える場である．経営トップの目から見ると，例えば「利益を増す」目的に対し，事業部門や遊休設備の売却，現状製品の量産拡大，あるいは新規製品の開発等の多くの候補が考えられ，選択が要になる．

社内システムは，全社の事業を果たす人と設備や社内システムの状況を検討し定量的に評価した上で，効果駅と考えるシステム新設～増強を企画し計画して実行する．世の状況や推移を充分に見守るが，流行に左右されて良いものではない．投資先には多くの候補がある．

製造ライン等のハードウエア関係は，製造関係か生産技術，各研究担当等が常に稼働状況/成算状況/収益状況を見ており，設備投資関係も明るいので，関係部署に集合して事前検討の機会を作る等して設備投資関係の計画をする能力があることが普通と思われる．

間接作業関係は，歴史的には昔の EDP の時代のように，現場と設備（投資）に明るい部署を育成できていないことが大きな問題である．設備に準じる社内用ソフトウエアの投資の責任者は情報システム部門の長と思われるので，できれば情報システム部門が間接作業～部門全体を見て方向を定めて新しく投資すべき箇所と具体的な投資方法を進言できる様であって欲しい．会社によっては，経営企画室等の名称で事業全体を見て事業と共に成長する社内システム企画元を設け，あるいは実行に比重が傾くが CIO にスタッフを付けて，社内システムの推進を行いつつ，実態把握と次への企画や計画を進める．

実態が定量的に把握できれば，あるべき～ありたきレベルと実態を比較して，投資候補先を洗い出し，夫々の手当の投資と高価など掌握する．以後は経営トップの経営戦略に掛かる．何処に力をどのように入れるのか，の原資は如何にするのかである．これ以上は目的の階層性により，具体化を行いながら順次問題を潰して行く．これ以上は経営問題であるから検討外とする．し

社内システム構築上の問題は，狙いに対して確実な改善を実現することである．問題に近い所から，実態を詰める．間接員の効率向上であるなら，ワークフ

ロー類で現状を可視化できる[12]．それにより現状と改善後が把握でき，これらを元にプログラムレベルで機能を決められる．更に確実を期するには，ブロックレベルのシミュレーションや中心的なプログラムを先行試作して実時間消費などを実地確認して予想と比較照合などが出来る．この前後から CIO が活躍する場になる．

プロジェクトを確実に仕上げるには，関係者の意識を変え能力を向上させる．第一に教育すべきは，プロジェクトは厳重に管理されるべきもので，それを守り抜く強靭な人になることである．全てのコミットメントは几帳面に厳守する．それには，各人の予定に適当なマージンを持つ義務がある．このマージンとは各担当者なら，当該期間に平均的に生じる可能性のある異常事態に対し，自分の能力範囲～計画範囲内で異常が処置できる程度の余裕である．より上位の取纏は，同じ期間に（自分と動員できる程度の部下を含めて）受持ち範囲内で平均的に生じる可能性のある異常事態に，異常が自分の能力範囲内に留める処置ができる程度賄える程度のマージンを持つ．これは順次上に及んで行き，最後には工場長が傘下の各部長の協力で問題を食止める．出来ないなら社長の所に飛んで行き，対応をお願いする．

組織と権限／責任が階層的に展開し続け，日程計画等の全てが組織と対応する階層展開であれば，辻褄があう．但し，これは稀に起こる「まさか」であるから皆が協力できるので，トラブルが頻発するなら，対応できない．従って，軽度の階層から出発しながら，トラブルが起きる頻度を減らしていった「高成熟度組織」の話である．その域に達するまで，全員が努力しながらトラブル率を減らし，無事回復できる率を向上させる努力が要る．低成熟度から最高成熟度迄具体的な改善を積上げ人々を訓練して育成する，

６．むすび

以上から開発を外部依頼についての結論は以下の通りである．

1. 製品であるソフトウエア
   ・（通常のソフトウエアベンダーには依頼できず）当該品を既に生産している，あるいはそれと同等な充分に高い能力があるなら，開発を依頼できる．
   ・あえて外部に依頼するなら，システム部署を作り純プログラムレベル（コーデイング等）の業務で支援を受ける．

２．社内用ソフトウエア　　内容のクリテイカルさの違う各種の場合がある．前項と同様なシステム部署を作り，見極めを付けた上で，クリテイカルさに応じて外部作業にする．

３．実際のシステム作業の実体をご理解頂きたく，現場作業での理想を報告した．ご理解頂ければ幸いである．

参考文献

[1] [Martin90] James Martin，Information Engineering, Book 1-3, Prentice-Hall, Inc., 1990．訳本 インフォーメーション・エンジニアリング，第１−３巻，（株）凸版，1994.

[2] [Koono10] Z. Koono and H. Chen, Development of a Commercial Product Including Software, Proc. New Trends in Software Methodologies, Tools and Techniques (SoMeT 10), pp. 415-434, Sept. 2010.

[3] Koono, Z., Processor Systems in High Integration Age, Joint Conference of Four Electrical institutes 1979, No. 27-3, 1979.

[4] Z, Koono, K. Ashihara and M. Soga, Structural Way of Thinking as Applied to Development, IEEE/IEICE GLOBECOM 1987, pp. 26. 6. 1-26. 6. 6, 1987.

[5] 河野善彌，陳慧，ハッサン　アボールハッサニ，人の意図的行動の知の構造とそのはたらき，信学技報　KBSE2006-56 (207-01).

[6] G. J. Myers, Composite /Structured Design, Litton Industries, 1978．訳 国友ほか， ソフトウエアの複合／構造化設計，近代科学社，1979.

[7] M. A. Jackson, Principles of Program Design, Academic Press 1975．訳 鳥居，構造的プログラム設計の原理，日本コンピュータ協会，1980.

[8] Z. Koono, H. Chen and B.H. Far: Expert’s Knowledge Structure Explains Software Engineering, Joint Conference on Knowledge-Based Software Engineering (1996), 193-197, 1996.

[9] B. Jaruzelski and K. Dehoff, *Profits Down, Spending Steadily: The Global Innovation 1000*, pp. 6, preprint, strategy + business, issue 57, Winter, 2009.

[10] 岩橋優，新製品開発計画段階に於ける新QC七つ道具(N7)の活用（(株)サクラクレパスの事例），納谷嘉信編，研究開発とTQC，日本規格協会，1990.

[11] Carl Phillip Gottlieb von Clausewitz, Vom Kriege, 1832.

[12] 石橋博史,可視経営ー仕事が見えれば会社は変る，日経BP企画，2005.

[13] Z. Koono, T. Kondo, . Igari and K. Ohtsu, Structural Way of Thinking as Applied to Good Design (Part 1. Software Size), Proc of IEEE GLOBECOM 91, pp. 24.3.1-8, 1991.