1. ソフトウェア工学の40年

玉井哲雄 東京大学

1968年にドイツのGarmischで「ソフトウェア工学」をテーマとするNATOの会議が開かれてから，今年で40年を迎える．この機会に，ソフトウェア工学の40年の歴史をざっと振り返って，今後への参考にしようというのが，この小特集の趣旨であり，またこの記事のねらいでもある．

1968年

1968年は波乱の年であった．1月にチェコで「プラハの春」と呼ばれたドブチェクによる政治体制の自由化運動が始まったが，8月にはソ連につぶされた．4月にマーチン・ルーサー・キングが暗殺された．6月にロバート・ケネディが暗殺された．おりしもベトナム戦争のさなかで，米国ではコロンビア大学で戦争反対の学生が大学本部を占拠するなどの運動が拡大し，パリでは5月革命とも呼ばれる学生を中心とした反体制運動が過激化した．日本でも大学の旧体制解体を唱えて，東大では3月に安田講堂の占拠，7月に全共闘の結成があり，日大では5月に全共闘が組織された．東芝府中工場の3億円強奪事件が起こったのもこの年である．また明治維新は1868年であったから，奇しくも維新後100年という節目の年でもあった．

このような中で，10月にドイツのGarmischでSoftware Engineeringという名を初めて冠した会議が開かれたことなど，ニュースにもならなかったはずである．しかしソフトウェア工学の歴史はこのときから始まる．そして今年（2008年）はそれからちょうど40年．ソフトウェア工学も不惑の年を迎えたわけだ．

Garmischの会議はNATOの科学委員会が主催したもので，約50名の会議参加者はすべてヨーロッパと北米から呼び集められた．同じNATOの会議は翌1969年10月にもイタリアのローマで開かれ，やはり西欧諸国から約60名が参加した．

ありがたいことに，この両会議の後に出版された2つの会議録が，どちらの会議にも参加した米国のRobert M. McClureという人によってpdfファイルとして復刻され，それが両会議録の編集者の1人であるBrian Randellのwebページ上で公開されている[3), 5)]．現在の国際会議で通常出版される論文集は，事前に投稿され審査された論文を集めて編集したものだが，このNATO会議録は一部に論文形式で書かれた短い報告が載っているものの，主要部分は会議の録音テープを起こして編集されたもので，それだけに当時の熱気をそのまま伝える優れた読み物となっている．特に1968年の最初の会議は，会議としても成功し，編集もうまくいったとRandellが自賛するだけのことはある．

NATO会議の精神を受け継いで，ソフトウェア工学の国際会議（International Conference on Software Engineering，通称ICSE）が始まったのは1975年である．それ以来この分野で旗艦会議と位置づけられ，例年5月前後に開催されている．Garmischから40周年となる2008年は，あたかもよく誕生の地ドイツに戻り，5月中旬にライプツィヒで開かれた．その会議の初日，開会直後に「SEの40年」というセッションが企画され，Garmisch会議の主宰者であったPeter NaurとBrian Randellが招待されて，元気な姿を見せるとともに，パネル討論でも熱弁をふるった．

時代背景

IBMのSystem/360とそのオペレーティング・システムOS/360が発表されたのが，1964年である．System/360は初の汎用コンピュータと言われ，商業的にも成功して，その後の「メインフレーム」による情報システム開発と運用のためのコンピュータのモデルとなった．このSystem/360の登場とともに，応用システムのためのソフトウェア開発需要は飛躍的に増大した．

1960年代という大昔には，それほど大規模なソフトウェアは作られていなかったと我々は思いがちだが，実はそうでもない．60年代はおろか，50年代でもかなり大きなプログラムが作られていたとの証言が，F. Brooksによってなされている．Brooksの「銀の弾丸はない」というエッセイは大きな反響を呼び，多くの人がコメントを書いた．その中の1人D. Harelは，1950年代には100行から200行のプログラムを1人のプログラマが作っていただろう，という想像の上に立って議論を展開しているのに対し，Brooksは反論して次のような例を挙げている[2]．

「1955年までに50人年から100人年のビジネス・プログラムが作られていた．1956年までにGEは8万語以上の給与計算プログラムの運用を始めていた．1957年までに空軍のSAGEコンピュータは7万5千命令のプログラムで通信ベースの二重化耐故障実時間システムを30カ所で動かしていた．」

1968年時点では，System/360を使って多くの大規模システムが作られていたはずだが，この時代の大規模なソフトウェアを代表するのはSystem/360のオペレーティングシステムであるOS/360そのものである．この時期のOS/360の大きさは5百万ステップであるとNATO会議録に書かれている．このプログラム規模について，G. Holzmannは2002年に次のように述べている[4]．

「この時期（1968年）の大規模システムの1つであるIBMのOS/360は，アセンブラで5百万行のコードとされている．現代のC言語に換算するとおよそ100万行であろう．34年後の現在，典型的なOSの規模は1億行以下だろう．ということは大規模ソフトウェアの大きさは30年で2桁以下の増加しかしていないことになる．一方，そのコードを分析するコンピュータの処理能力は6桁増加している．」

筆者はこの講演を実際に聞いているのだが，ここで「典型的なOS」とは，Microsoftの Windowsを指している．Windowsは規模が公開されていないのが問題だが，おそらく4000万行から7000万行の間だろうとHolzmannは言っていた．またコンピュータの処理能力をコードの分析用途として論じているのは，Holzmannのモデル検査系SPINを始めとするツールを用いたプログラム解析が，このときの講演の主題だったからである．

NATOの1968年の会議録でも，OS/360は数多く言及されている．数えてみると15カ所ぐらいに登場する．それも応用システムを開発するベースとして触れらているよりも，OS/360そのものをソフトウェア工学の分析対象として議論している方が多い．たとえば，開発に5000人年を要したというデータが挙げられている．また，少し前のデータとして，1966年時点でのコンポーネント別（ここでいうコンポーネントはサブシステムというぐらいのレベルである）のモジュール数とステートメント数のデータが表として載せられている．さらに改訂版の公開間隔が90日で，平均いくつのエラー修正や変更があるという議論もある．またテスト時のエラー発見累積数の予測と実績を比較したグラフも掲載されている．こう見ると，OS/360についてはWindowsよりも多くのデータが公表されているようである．もちろん現在のオープンソースのような状況ではないが．

もっともWindowsが研究者によく悪口をたたかれるように，OS/360も大学からの参加者にあっては評判が悪いというところには，共通点がある．たとえばA. J. Perlis（Carnegie-Mellon大学）は「OS/360にはがっかりだ」と言っている．

OS/360はその後も引き続き，ソフトウェア工学の研究対象として取り上げられた．その中に後世に大きな影響を与えたものが少なくとも2つある．1つは，F. Brooksによって1975年に書かれた不朽の名著「人月の神話」である．その内容は，OS/360の開発をプロジェクトリーダとして取り仕切ったBrooksの経験に基づいている．この本が広く読まれたことは，1995年に20周年記念版が発刊されたことからも分かる[2]．もう1つの例は，BeladyとLehmanによる「ソフトウェア進化の法則」である．ソフトウェアが保守と機能拡張でどのように変化していくかをパターン化したこの「法則」は，OS/360の時系列的なデータ分析に基づいて「発見」された[1]．

いつの時代も，そのプログラミング状況を把握するのに都合のよい方法の1つは，その当時に作られたプログラミング言語を見ることである．FORTRAN，COBOL，LISP，ALGOLは1960年ぐらいまでに作られて，この時代はすでに日常的な言語となっていた（少なくともFORTRANとCOBOLは）．それに対し，より洗練され，ものによっては複雑化して重装備となった言語が作られたのがこのころである．年が名前の一部になっている言語に，Simula67，Algol68がある．これらと前後して登場したBASIC（1964），APL（1967），PL/I（1969）と並べれば，この時代の雰囲気が感じられるのではないだろうか（括弧内の年はいずれも，言語処理系が最初に出された年を指す）．

ソフトウェア工学の基礎概念

1968年のNATO会議が招集された背景として，現

代社会の中心的な活動にますます強く結びつきつつある情報システムの信頼性をどう確保するか，大規模ソフトウェア開発プロジェクトの納期を守り仕様を満たすことが難しい状況にどう対処したらよいか，ソフトウェア技術者の教育をどうすべきか，という問題が挙げられている．これらは現在でもまったくそのまま通用する問題意識であろう．そして会議の目的は，ソフトウェア工学上のさまざまな問題に光を当て，またそれらの問題を解決する可能性のある技術や方法を議論することであるとされた．

そういう意味ではこの会議の精神は，ソフトウェア工学の誕生を高らかに宣言した輝かしいものという感も与えるが，一方ではソフトウェアに期待されるものと実際の開発状況との乖離という暗い現実が重くのしかかっていたという面もある．後年この会議からもたらされたキーワードとして「ソフトウェア危機」という言葉がよく使われるようになったが，この会議録では危機（crisis）という語の使用個所は2カ所しかない．しかし，そのうちの1つが

「一部の参加者が「ソフトウェア危機」あるいは「ソフトウェア・ギャップ」とあえて呼んだことについて，多くの議論があった．」

というもので，これがその後に大きな影響を与えたことが想像される．もっとも，そのような見方はあまりに悲観的で現実を反映していないという発言も一方ではなされていたが．

用語という点でいえば，1968年のNATO会議の時点で，ソフトウェア工学の基礎概念を構成するような用語のほとんどがすでに登場していることが分かる．たとえば，要求，見積り，仕様，設計，テスト，保守，システム進化などはもちろんのこと，逐次改善（incremental improvement），品質保証，正当性の証明という概念や，ソフトウェアの性質としてのモジュール性，頑健性，拡張性，柔軟性などの用語も使われている．オブジェクト指向という言葉はむろん出てこないが，Simulaは処理系がすでにあっただけでなく，この会議で何度も言及されている．さらにコンポーネントはともかく，ミドルウェアという語もこのときからあったのには，実のところ驚いた．さらにびっくりしたのは，1970年代にT. DeMarcoとT. Listerが著書「ピープルウェア」で引用し，1990年代には設計パターンで再脚光を浴びた建築家のC. Alexanderが，このときすでに引用されていることである．それはNaurによるもので，

「ソフトウェアの設計者は，建築家や土木技術者，それも街や産業プラントのような大規模で異種混淆な建設物を設計する人々と同じような立場にある．したがって設計問題にいかに立ち向かうべきかについて，このような分野を参考にすることは自然である．そのような発想の元となる1つのよい例として，C. Alexanderの“Notes on the Synthesis of Form (Harvard Univ. Press,1964)”を挙げておきたい．」

というのである．

構造化の時代

NATOの会議でソフトウェア工学が必要だという問題意識が共有され，ソフトウェア工学を構成する基本概念もかなり出そろったと言っても，それにきちんとした肉付けが与えられたのは1970年代である．このときに推進力となった指導原理は「構造化」である．まず構造化プログラミングが提唱され，続いてソフトウェアプロセスを遡る形で，構造化設計，構造化分析が提案されて大きなインパクトを与えた．この時代，フランスの哲学界を席巻した1960年代の「構造主義」の余韻はまだ強く残っていたが，ソフトウェア工学における「構造化」と構造主義との直接的な関係は，あまりなさそうだ．

構造化プログラミングの考え方はまず，プログラムの制御構造の簡素化・明確化という形で打ち出された．その合言葉は「goto文はダメ」というものである．goto文の使用により，制御はあちこちに飛ぶ．それがプログラムの振舞いを分かりにくくし，エラーを起こしやすくする原因となる．ではgoto文の代わりに何を用いればよいか．その答えを理論的に与えたのが，あらゆる制御構造は連接と分岐と反復の組合せで記述できるというBöhmとJacopiniの「構造化定理」である．プログラムをこの3つの制御構造を組み合わせれば，すっきり書ける．分岐と反復は，それぞれの内部により小さい分岐や反復構造を入れ子にすることができる．入れ子構造は，プログラムを記述する際に字下げを活用することにより，視覚にも訴えるようにできる．

構造化プログラミングのもう1つの側面は，モジュールの抽象化である．プログラムのまとまった意味単位としてのモジュールという概念は，以前からサブルーチンあるいは手続きと呼ばれるプログラミング言語の構成要素として，提供されてきた．しかし1970年代の中頃に，D. Parnasによって提唱された抽象モジュールや，それに続いてAlphard, Clu, Euclidなどの新しいプログラミング言語に組み込まれて提案のあった抽象データ型は，モジュールという単位の抽象化のレベルを高めるとともに，その中により豊富な意味内容を盛り込みつつそれをカプセル化して使いやすくした，という点で画期的なものである．概念は60年代のSimulaで用意されていたことは確かであるが，その代数仕様による形式的意味論のような理論的研究は，この時代に花咲いた．また，

手続き処理を単位とする構造化はトップダウン型の階層構造になるが，抽象モジュールによる構造化はより柔軟性のある構造を形作るという特徴もある．

構造化はこのような理論，言語，手法の提案として推進されただけでなく，産業界における実践的な取り組みも盛んに行われた．たとえば米国 IBM では IPT という構造化プログラミング運動が進められた．日本では時期的には少し後になるが，フローチャートに代わる構造化図式が数多く提案された．たとえば日立の PAD，NEC の SPD，NTT の HCP などである．

構造化と直接関連するわけではないが，この時代に概念が出され，その後のソフトウェア開発に大きな影響を与えたものにソフトウェアのライフサイクル・モデルという考え方と，その代表としての落水（waterfall）型モデルがある．落水型モデルを提唱した元祖の論文は，W. Royce が書いて 1970 年に発表した「大規模ソフトウェアシステムの開発管理」（“Managing the Development of Large Software Systems”）であるというのが定説になっているが，この論文のどこにも，waterfall はおろかライフサイクルという語すら出てこない．Royce の先見性は，むしろ単純な落水型のモデルの持つ問題点を指摘しているところにある．

このように，1970 年代は「構造化」を合言葉に研究と実践が車の両輪のように働き，ソフトウェア工学を大きく発展させた時代ということができよう．

管理技術へのシフト

どのような技術分野も発展の過程で専門性への細分化が起こる．ソフトウェア工学も例外ではない．まずプログラミング言語やプログラミング方法論という分野はソフトウェア工学から離れた．もっともこれらの分野はソフトウェア工学誕生以前からあった老舗で，ソフトウェア工学から分離独立したというよりは，もとの居場所に戻ったというべきかもしれない．しかし，NATO の会議には Dijkstra，Naur，Perlis，Hoare，Gries，Wirth，Reynolds，Strachey などがいたのである．1980 年代になるとこのような面々は ICSE などには顔を出さなくなる．

代わって強調されたのが管理技術である．ソフトウェア工学の必要性を説くためにしきりに使われた枕言葉は，「大規模複雑化するソフトウェア」の開発に対処するということだったが，それにはプログラミング・レベルの技術では不足だという認識が広がった．大規模なシステムを長期間かけて大きな組織で開発するには，何よりも「管理」が必要だというわけである．管理の対象はさまざまである．プロジェクト管理，要員管理，予算管理，工程管理，品質管理，構成管理，計算機資源管理，といった言葉がたがいに重なりあいながら多様な管理対象を指し示している．

実務的な観点から管理面を強調することには意味があるが，その結果は一般的なプロジェクト管理との差異がはっきりせず，ソフトウェア工学としての特徴が薄れる傾向を招かざるをえない．経営学や産業心理学に近づくのもある程度必然性はあり，そこから意義のある成果も生じようが，ソフトウェア工学の中心部分で目覚ましいものが現れないと，1 つの科学技術分野としての活力が低下する．

実践に近いところでの成果の 1 つは，プロトタイピングという手法である．ソフトウェアの開発にとって最も困難な問題の 1 つは，ユーザの要求があいまいだったり不十分なことにある．それを解決するために，システムの動作をユーザが理解できるような簡単なプロトタイプを手早く作り，それをユーザに試してもらって要求を明確化し充実させるという方法である．これはソフトウェア開発のプロセスの新たなモデルとも言え，その後のさまざまなプロセスモデルの提案に影響を与えた．

やはり実践的なものの例に，査閲（inspection）あるいは見直し（review）と呼ばれる方法がある．これは仕様書やプログラムコードを人が読んで検査するという，それまでも広く行われていたやり方を，それを実施するチームの体制，その構成員の役割，記録の取り方，結果に基づく修正方法，などを形式化して手法として確立したものである．IBM における実施例が報告されて以降，多くの組織で意識的に実践されるようになった．

これらは着実な成果と言えるが，70 年代の構造化のような飛躍的な技術革新とは言い難い．一方，技術革新と言えば，1980 年代は時ならぬ AI ブームの時代でもあった．人工知能（AI）の研究はほぼコンピュータの誕生とともに始まっているが，80 年代は E. Feigenbaum の提唱による「知識工学」の掛け声とともに，実用化への期待が大いに高まった．日本では第五世代コンピュータ開発プロジェクトが 1982 年から 10 年計画で進められ，これが米国やヨーロッパにも類似のプロジェクトを発進させるきっかけとなった．

AI はソフトウェアとして実現されるからソフトウェア工学との関連が深いことは当然であるが，ソフトウェア開発という複雑な知的作業にも，知識工学の手法が使えるのではないかという着想から，多くの試みがなされた．その努力はソフトウェア工学用のエキスパートシステムというような形では結実しなかったが，問題領域の知識を収集し分析しモデル化するというアプローチは，ソフトウェアの要求分析のフェーズで活かされ，現在の要求工学にもつながっている．また，AI の推論に使わ

れる論理表現は，ソフトウェア開発における形式手法の発展に直接寄与した．さらにプログラムの自動生成を始めとするソフトウェア開発におけるさまざまな自動化ツールの作成・使用にもAIの影響が認められる．

ツールと言えばAIブームにやや遅れて，1980年代終わりから90年代前半にかけてCASE（Computer Aided Software Engineering）も脚光を浴びた．CASEという名前からはソフトウェア工学で使えるツール全体が対象となりそうだが，このときに特に注目されたのはデータフロー図などの描画機能と，図に現れるデータやプロセスなどの項目を管理するデータベースを組み合わせた，分析設計段階用のツールである．しかしこれもブームの宿命か，マスコミでもてはやされたのは一時期にすぎない．ただ，ブームは去っても，そのようなツールが製品化され使用されるという状況は，現在に至るまで着実に続いている．

これらの動きをまとめて言えば，活気のあった70年代と比べ，80年代のソフトウェア工学はどちらかといえば沈滞期にあったといって過言でなかろう．

オブジェクト指向

1990年代はオブジェクト指向の時代である．オブジェクト指向言語の元祖とみなされるSimulaは1960年代に作られ，オブジェクト指向の主要概念であるカプセル化，継承，多相性などはみな1970年代に確立されていた．しかし，オブジェクト指向という名称と考え方が広く世に知られるようになったのは，1980年にSmalltalk-80が公開されたことによる．

1970年代に構造化プログラミングの考え方から構造化設計，構造化分析が生まれたように，オブジェクト指向プログラミングからオブジェクト指向設計，オブジェクト指向分析が手法として生み出された．そのオブジェクト指向分析・設計が数種類の本の出版などによって大きな反響を呼んだのが，1990年前後である．ただ面白いことに，米国ではオブジェクト指向というテーマがソフトウェア工学系の会議で取り上げられるのに，数年の時間遅れがあった．これは筆者の見るところ米国の学会（この場合ACM）における政治力学が働いたせいではないかと思う．ACMの傘下にあるオブジェクト指向の会議としてはOOPSLA（Object-Oriented Programming, Systems, Languages, and Applications）が1986年に始まり，年々参加者が増大して大成功を収めた．これを主催しているのがACMのSIGPLANというプログラミング言語研究会である．一方，ICSEなどのソフトウェア工学の会議はSIGSOFTという研究会が主催しているが，オブジェクト指向についてはSIGPLANに機先を制せられ，一種の棲み分け原理でしばらく手を出さなかったのではなかろうか．しかし，日本やヨーロッパではそのような気兼ねがないから，それらの地域におけるローカルな会議では，早くからオブジェクト指向をソフトウェア工学のテーマとして取り上げていた．

90年代前半はオブジェクト指向方法論が乱立し百家争鳴の感もあったが，せめて記法だけでも統一しようということで出てきたのがUML（Universal Modeling Language）である．これはBooch法のG. BoochとOMTのJ. Rumbaughの合流したところにK. I. Jacobsonが後から参加して始められた方法論統一への動きがもとである．その後OMG（Object Management Group）という連合組織が引き取って記法の統一化に徹することとなり，1997年11月にUML1.1が出された．引き続き次々と改訂版が出され，2004年10月にはUML2.0が公表された．

先に述べたように，1980年代のソフトウェア工学は大規模複雑化するソフトウェアの開発に，いかに管理技術を駆使して立ち向かうか，という点に重点が置かれた．その間に世の中ではパソコンが製品化されて普及し，それがインターネットにつながって，小型化，開放化．分散化の波が滔々と押し寄せていた．ソフトウェア工学は一時期この動きに完全に立ち遅れ，昔ながらの「大規模複雑化」するシステム開発需要がもたらすソフトウェア危機というお題目を唱え続けて，小規模多品種でニーズの変動が激しいソフトウェアの開発にいかに対応するか，という点がなおざりになった．

オブジェクト指向技術はこの点でも有用だった．オブジェクトという概念は，分散化，部品化，再利用の単位としてもきわめて有効だったし，オブジェクトから構成される設計パターンやフレームワークは，インターネット時代に必要とされるソフトウェアを効率的に組み立てるのに役に立った．たとえば1990年代半ばに世に出たJavaは，最初からプラットフォーム非依存，Web上の応用システム開発，セキュリティという目標を意識して設計されたプログラミング言語として，時代の要請に応えるものであった．

総じて言えば，1990年代はこのようなオブジェクト指向技術をベースとして，ソフトウェア工学自身が再構築（re-engineering）された時期と見ることができる．1980年代がソフトウェア工学の沈滞期であったとすると，1990年代はソフトウェア工学が求心力を回復し始めた復興期と見ることができるだろう．

プロセスとプロダクト

オブジェクト指向の意義は大きかったが，しかし「構造化」という概念1つで1970年代をひとくくりにできた時代から見れば，1990年代はそれほど単純ではなく，1つの「指導原理」でソフトウェア工学全体の動きが説明できるというわけではない．別の視点を与える軸の一例として，プロセスとプロダクトの関係を取り上げてみよう．

1980年代の後半から1990年代前半にかけて，「ソフトウェアプロセス」がソフトウェア工学の一分野として改めて注目され，産業界でも研究界でもプロセスをテーマとする活動が隆盛となった．そのきっかけを与えたものが2つある．1つは1987年に米国カリフォルニア州モンタレーで開催されたICSEで，L. Osterweilが行った「ソフトウェアプロセスもソフトウェアである」という題の基調講演である．その趣旨は，ソフトウェア開発のプロセスも，たとえば

> 「エディタを用いてプログラムを作成し，コンパイラにかけてコンパイルエラーが出ればエディタに戻って修正し，コンパイラが通ればテストを実行し，テスト結果が想定通りであれば終了し，そうでなければふたたびエディタに戻って修正することを繰り返す．」

というような手続きとして書ける．このようなソフトウェアプロセスの記述自身もソフトウェアとみなすことができるから，その記述作業をプロセス・プログラミングと呼ぶことにする．そこで，そのためのプロセスモデルやプロセス記述言語を開発する必要がある．それによりプロセス記述の形式化がなされ，プロセス分析，プロセスの自動化，プロセス中心の開発環境構築が可能になるというものである．

もう1つは，プロセスの評価モデルとして，同じ1987年に最初に提唱されたカーネギーメロン大学ソフトウェア工学研究所によるCMM（Capability Maturity Model）である．これはソフトウェア開発組織が実施している開発プロセスを5段階で評価する，というものである．その背景が，米国国防省が導入するソフトウェアの調達先の信頼度を評価することにあったため，米国のソフトウェア企業，特に政府への納入を生業とする業者には大きなインパクトがあった．

この2つの動きは，同じプロセスという言葉を使いながら目標もアプローチも大きく異なる．しかし，1970年代の落水型ライフサイクル・モデルもプロセスモデルの一種であったことを考えれば，視点こそさまざまなものがあり得るが，プロセスはつねにソフトウェア工学の主要テーマの1つであることは間違いない．

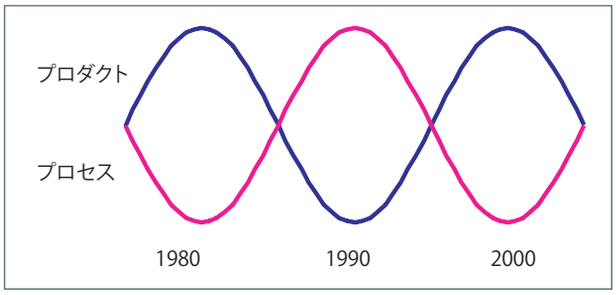

図-1　プロセスとプロダクトへの関心の交替

プロセス研究の活動は，1991年にソフトウェアプロセス国際会議が米国カリフォルニア州レドンドビーチで開催されたときにピークを迎えたが，その後急速にしぼんでしまった．1984年から続いていたソフトウェアプロセス国際ワークショップも，1994年を最後に休止した．もちろん，実践面での活動は地道に続けられたが，浮気な研究界は次の標的を探して移っていった．代わって登場したのがソフトウェア・アーキテクチャである．

アーキテクチャとはもちろん建築物や建築学を指すが，建築以外の工学分野では「基本設計思想」という意味合いで広く使われてきた．計算機分野でも基本命令セットをコンピュータ・アーキテクチャと呼んで以来，馴染みのある言葉である．ソフトウェアにアーキテクチャという概念が導入されたのもそれほど新しいことではないが，この時期プロセスへの関心の失速に伴ってアーキテクチャが浮上してきた．それを象徴するのがM. ShawとD. Garlanが共著で問うた「ソフトウェアアーキテクチャ」という本の出版で，1996年に出されて広く読まれた．

およそあらゆるエンジニアリングにとって，生産のプロセスとその成果であるプロダクトは双対の関係にある．そしてどちらかというとプロセスに関心が高まる時期と，プロダクトに関心が高まる時期とが交互に訪れる傾向が一般にあるようだ．ソフトウェア工学においても，プロセスがもてはやされた時期からアーキテクチャというプロダクトの構造に注目が移ったこの変遷は，そのような関心の交替という変動傾向から説明できるかもしれない．**図-1**はこの流れを模式化したものである．このような観測をより広げて，たとえばヘーゲル流の歴史観になぞらえることも可能かもしれないが，単なる言葉遊びに終わる恐れもある．

単純にこの図式を当てはめると，現在はプロセスへの関心が復活しつつあるときということになる．実際，XP（extreme programming）などで採用されている軽快なプロセスが注目されるなど，プロセス復活の兆しはある．しかし，共通部分と個別部分を持つ多様な製品群を一括して開発するための枠組みであるプロダクトラ

インの研究と実践が活気を呈しているという現象を見ると，プロダクト群を対象にしながらそれを並行して開発管理するというプロセスに着目しているという意味で，プロダクトも大事，プロセスも大事ということになるのかもしれない．これもヘーゲル流の止揚といえるだろうか．

日本のソフトウェア工学

情報処理学会のソフトウェア工学研究会は，情報処理学会の数ある研究会の中でも，最も歴史の長いものの1つである．ソフトウェア工学研究委員会として発足したのが1976年の4月．委員会としての1年の活動の後，1977年4月から現在の研究会の形になった．当時，研究会を始めるには，まず閉じた委員会として1～2年活動し，その結果を見て理事会で承認されるという手順であったとのことである．このような経緯については，1989年度から92年度まで主査を務められた原田賢一[☆1]（慶應義塾大学）が資料を保存されている．

すでに述べたように，1970年代は構造化の合言葉のもとにソフトウェア工学が盛り上がった時期であるが，その中でソフトウェア工学研究会は國井利泰（当時東京大学，現在法政大学）を主査として始められた．それ以降，登録会員数で500～600人の規模を保ち，4半世紀にわたって活発な活動を続けてきたのは，研究テーマのはやりすたりが激しい情報分野の中にあって，かなり特異なことといえるだろう．

ICSEが発足したのは1975年であるが，その第1回目はアメリカ標準局（NBS）の主催だったためにNational Conference on Software Engineeringと呼ばれて，ICSEが正式名称になるのは1976年からである．そのICSEが早くも1982年に東京で開催された．この頃は開催が1年半間隔だったから，1976年から数えると第5回目である．大野豊，榎本肇などが尽力された．日本で2度目のICSEが開催されたのは，1998年の京都である．このときは鳥居宏次，片山卓也などが中心であった．

ソフトウェア工学研究会では，アジア太平洋地域を対象とするソフトウェア工学の国際会議を起こした．まず1992年にJCSE（Joint Conference on Software Engineering）を日本と韓国を中心としてスタートさせ，1994年にそれを発展させて新たにAPSEC（Asia Pacific Software Engineering）という会議を創設した．APSECは当初，日本，韓国，オーストリア，香港，台湾，シンガポールという6つの国／地域の学術団体を創設グループとする国際会議として始められ，さらにインド，中国本土，タイ，マレーシアが参加している．第1回を東京で開催して以降，オーストラリア，韓国などを順に巡って，昨年（2007年）日本では3回目となる会議が名古屋で開かれている．APSECはヨーロッパにおけるESEC（European Software Engineering Conferecne）に対応するものと言えよう．

日本の産業界におけるソフトウェア開発体制としては，1980年代に「ソフトウェア工場」が世界的に注目された．またコンピュータメーカを中心に大量のソフトウェア開発を行っている企業では，社内にソフトウェア生産技術開発部門を組織することが一般化した．ただ，バブルのはじけた1990年代には，これらの生産技術部門の多くは縮小や廃止に追い込まれた．

2000年代に入って，産業界でもソフトウェア工学が徐々に力を取り戻してきた．企業の生産技術部門も一部復活した．産官学の体制作りの動きとしては，まず2000年4月に設置された国立情報学研究所を挙げることができる．そこでは，ソフトウェア工学が重要な柱の1つを占めている．また，2004年4月には情報処理推進機構（IPA）内にソフトウェア・エンジニアリング・センター（SEC）が創設された．そこには約40名の所員とセンター外の技術者や研究者が集まり，情報システム分野や組込みシステム分野でいくつかのプロジェクトを立てて実践的な研究を行っている．

日本の大学ではソフトウェア工学分野は比較的手薄である．ソフトウェア工学を中心とする学科やコースの数はきわめて少ない．情報科学科や情報工学科の中で，部分的に教えられたり研究されているのが実態である．教授団にソフトウェア工学を専門とする者の数もそれほど多くないが，企業の研究者が大学に移るという形での人材供給が増えつつある．

そして今

今世紀に入ってからのソフトウェア工学で目立つ動きは，プログラムや文書の分析技術の高度化であろう．プログラムの静的解析技術には長い歴史があるが，その手法はますます洗練化されてきている．さらに，モデル検査に代表されるようなモデルレベルの解析，プログラムの構造より振舞いの解析，という方向への進展も目覚ましい．UMLの普及によりUMLで記述されたモデルの解析も，適用範囲が大きく広がっている．

手法の進化はもちろんのこととして，このような分析技術の発達を助長する要因として，少なくとも2点を挙げることができるだろう．1つには，さまざまなオー

☆1 以下，日本人の名前に敬称をつけないのは失礼の感があるが，外国人と区別するのもおかしいので省略する．ご容赦願いたい．

プンソース・ソフトウェアが公開されていることである．これは解析対象とし得る膨大なデータを提供している．公開されているのは原始コードだけでなく，設計文書，メールのやりとり，バグ報告など多様である．これは定量的なデータ解析の適用を試みる研究者にとって，垂涎の宝庫といってよい．皮肉に言えば，学会論文や学位論文を量産するための恰好な材料ともいえる．

もう1つはハードウェアの性能の並はずれた向上である．たとえばちょっと前ではお手上げだったような大量のログデータの解析も，簡単にできるようになった．モデル解析の問題を充足可能性問題（SAT）に帰着させて，汎用的なSAT解法器にかけて解くなどという強引な方法が通用するようになったのも，アルゴリズムの工夫だけでなくやはりハードウェアの速さの賜物である．

このような背景で，経験的（empirical）ソフトウェア工学が強調されるようになり，定量的なデータによってソフトウェア工学のプロセスや手法を評価しようという動きが盛んになった．一方で，プログラムの精密な解析というアプローチは，ソフトウェア工学をプログラミング言語分野に再び接近させる契機ともなっている．最近のICSEなどの論文を見ると，プログラミング言語関係の会議で発表されてもおかしくないようなものが少なからずある．

さて，ソフトウェア工学誕生から40年経った現在を改めて見渡してみよう．40年前に言われた「現代社会の中心活動にソフトウェアの果たす役割が大きい」という事実は，現在，ますます確固としたものとなっているといえよう．Webサービスの形で提供される切符予約，商品販売，路線案内，地図検索，イベント情報提供，さらには，組込みシステムとしての携帯電話，デジタルカメラ，カーナビから冷蔵庫，炊飯器にいたるまで，身の回りにはソフトウェアを核とするシステムや製品が満ち溢れている．

しかしそのソフトウェアは目に見えない．その上に，コンピュータそのものが見えなくなっている．現代のクルマには数十のコンピュータが載っているが，クルマのドライバーにそれが見えないのはもちろん，クルマの整備士すらそれらを直接見ることはなく，コンピュータを組み込んだ部品ごと点検したり交換したりする．

そこでソフトウェア工学の果たす役割も，ますます増大する一方で，一般社会の目からは隠れてしまいがちである．ソフトウェアが新聞やTVに取り上げられるのは，空港のチェックイン・システムで障害が発生して8万人が影響を受けたとか，駅の自動改札機の障害で260万人の足が乱されたといった不祥事ばかりである．ソフトウェア工学のお陰でこれほど社会が助かっているというような話は，記事にならない．

しかし，1968年に提起されたソフトウェア工学の必要性が，過去のものとなったわけでは決してない．この40年の間にソフトウェア工学は着実な成果を上げてきてはいるが，たとえばハードウェアの生産性や性能の目覚しい進歩に比べるとそれほど目立たないのは，同じ土俵の上で数値的な比較がしにくいとはいえ，アピール不足の感も否めない．その意味ではソフトウェア工学の当面の課題の1つは，いかに社会にその意義を認知させるかにあるといえるだろう．

参考文献

1) Belady, L. A. and Lehman, M. M. : A Model of Large Program Development. *IBM Systems Journal*, 15 (3) : pp.225-252 (1976).
2) Brooks, F. P. J. : *The Mythical Man-Month : Essays on Software Engineering Anniversary Edition*. Addison-Wesley (1995).
3) Buxton, J. N. and Randell, B., editors. : *Software Engineering Techniques -Report on a Conference Sponsored by the NATO Science Committee*, Rome, Italy (Oct. 1969). http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1969.PDF
4) Holzmann, G. : The Logic of Bugs. In *ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-10)*, pp.81-87, Charleston, SC, USA (Nov. 2002).
5) Naur, P. and Randell, B., editors. : *Software Engineering -Report on a Conference Sponsored by the NATO Scinece Committee*, Garimisch, Germany (Oct. 1968). http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF

（平成20年5月12日受付）

玉井哲雄（正会員）
tamai@graco.c.u-tokyo.ac.jp

1948年生．1970年東京大学工学部計数工学科卒業．1972年同大学院工学系研究科計数工学専攻修士課程修了．同年（株）三菱総合研究所入社．1985年同社人工知能開発室室長．1989年筑波大学大学院経営システム科学専攻助教授．1994年東京大学教養学部教授，1996年同大学院総合文化研究科教授，現在に至る．工学博士．ソフトウェア要求技術，検証技術，モデル化技術，進化プロセスの分析，協調計算モデルの開発，等の研究およびそれらの技術の実際的な問題への適用に従事．著書に「ソフトウェア工学の基礎」（岩波書店，2004，大川出版賞受賞），「ソフトウェアのテスト技法」（共立出版，1988）など，訳書に「ソフトウェア要求と仕様 ―実践，原理，偏見の辞典」（新紀元社，2004）などがある．日本ソフトウェア科学会，日本オペレーショズリサーチ学会，人工知能学会，ACM，IEEE各会員．