2011年3月31日発行
第7巻第1号（通巻26号）
ISSN 1349-8622

SEC journal 24

Software Engineering Center

IPA
独立行政法人 情報処理推進機構
http：//www.ipa.go.jp/

SEC journal No. 24
独立行政法人 情報処理推進機構

「東日本大震災」のお見舞いを申し上げます

このたびの「東日本大震災」に被災された方々に、心からのお悔みとお見舞いを申し上げます。

日頃からIPA／SECの活動に対して様々なご協力をいただいてきた東北・関東など被災地域の団体・企業の皆様にお見舞いを申し上げましたところ、力強い返信をいただき、心を強くしたところです。まだまだ辛く苦しい日々が続くと思われますが、どうか一日も早い復旧をお祈りします。

このたびの震災では、地震･津波の想像を絶する自然の巨大な力に、人間の作り上げた人工物のもろさ、弱さを見せつけられた思いです。しかし私たちはそれに立ち尽くすのではなく、再興に向かって前進する必要があります。

SECとしては今後、被害を受け再構築が必要となったITシステムの再検討、再開発に向けた活動に対して、微力ではありますが、情報提供や助言などに協力していく覚悟です。関係団体などと協力して、この震災が情報システムに対してもたらした影響やそれに伴う課題をできる限り収集・分析し、対策を模索し、より安心できる情報システムの構築や現行システムへの補強策に関する情報を社会に発信していきたいと考えています。

この震災の経験を社会全体の財産として広く共有し、より安全な社会の再興への糧にすることが私たちの責務だと考えております。

SEC所長　松田 晃一

SEC journal No.24
2011年3月31日発行
第7巻第1号（通巻26号）
ISSN 1349-8622

巻　頭　言

地域の特色を活かした産業発展モデルを創造する

組込みシステム産業振興機構
理事長
宮原　秀夫

組込みシステムの重要性

近年、中国を中心とするアジア諸国は、これまでの生産拠点としての存在だけでなく、開発拠点へと変容をしはじめ、更には消費市場としても目覚ましい発展を遂げています。これら新興国が台頭する中、日本企業がグローバル競争に打ち勝っていくためには、情報家電とクラウドサービスの連携や電気自動車、スマートグリッドなどの環境エネルギー分野など、日本が得意とする分野において、新しいイノベーションの創出が不可欠であります。組込みシステムは、これらを実現するための鍵であり、日本の産業競争力の源となると考えています。

組込みシステム産業振興機構を設立

日本の技術力・競争力が世界を席巻し、持続的な成長を実現していくためには、地域が主体となりパワーを発揮し、特色を活かした新しい産業発展モデルを考え、国内外からの産業集積化を目指していかなくてはなりません。その先進的な仕組みの一つとして、「組込みシステム産業振興機構」（以下、振興機構）を2010年6月に設立しました。

我々は3年前、振興機構の前身である「組込みソフト産業推進会議」を設立し、会員の皆様の高い参画意欲のもと、組込みソフト産業の活性化に資する事業やサービスを企画・検討してまいりました。システムアーキテクトの育成を目的とした「組込み適塾」や組込みソフト開発の支援サービスなど、企業が単独では出来ない、また世の中に存在しないサービスを提供してきました。これらの経験から、組込みソフト産業を活性化するための進むべき方向性を導き出すことが出来ました。

振興機構では、これらの成果を、産業活性化につながる具体的な事業として、より深化させ直ちに実行に移し、更には、時代の変化に合わせて、組込みソフトウェアの領域拡大が見込まれる環境エネルギー、医療、FA制御、自動車などの分野に対応すると共に、ソフトウェアだけでなくハードウェアを含めた「組込みシステム」をターゲットにして活動を展開しています。我が国の組込みシステム産業の躍進には、世界を驚かす製品コンセプト・技術を創出出来る人材の輩出、企業のビジネス創造力が重要であり、振興機構はそのプラットフォームとして産業界を牽引したいと考えています。

これからは地方分権の時代であり、関西はその先導役として、産学官がしっかりと連携、協力することにより、時代に即した産業発展モデルを創造していきたいと考えています。

SECへの期待

SECが持つ、高品質な組込みソフトウェア開発手法や、組込みスキル標準ETSSなどの人材育成に関する取り組みは、組込みシステムの開発現場で広く活用出来るものであります。これらの活動を更に発展させ、日本のソフトウェア産業が世界をリードしていくために、今後も更なる成果を期待します。

ソフトウェアの信頼性に関する説明責任と品質監査のあり方について考える

中央大学総合政策学部 教授
平野 晋

SEC 所長
松田 晃一

ソフトウェアが自動車や家電などの製品に組み込まれ、ソフトウェアの役割が大きくなっている。それに伴い、ソフトウェアの信頼性及び安全性に対する関心が高まり、ソフトウェアの品質に関する説明責任やソフトウェアの信頼性を検証する仕組みが大きな話題となりつつある。そうしたことを考える上で、製造物責任が一つの参考となる。製造物責任の考え方やソフトウェアの品質監査のあり方について中央大学総合政策学部教授の平野 晋氏にお話を伺った。

松田：今回は、製造物責任法（PL 法）をご専門とされている平野先生との対談を通じて、ソフトウェアの品質やソフトウェアに関連して発生する事故と、そのソフトウェアを開発した側の責任などについて考えたいと思っています。現在ソフトウェアは PL 法の対象となってはいませんが、PL 法適用の事例は、今後のソフトウェアの問題を考える上で非常に参考になると思っています。情報システムにトラブルが生じると日本のマスコミは一斉に「駄目じゃないか」という論調で取り上げます。平野先生は米国の弁護士の資格をお持ちですが、情報システムのトラブルや事故に対する米国のマスコミの受け止め方はどのようなものでしょうか。

平野 晋（ひらの すすむ）
1984 年中央大学法学部卒。就職先メーカーから企業派遣で、PL 法の世界的権威 J. A. ヘンダーソン教授が在籍するコーネル大学ロースクールに留学し、法学修士号および米国弁護士（NY 州）資格取得。法律事務所および NTT グループ法務部門を経て、2000 年から NTT ドコモ法務室長。2004 年に母校の教授に迎えられ、2007 年博士号（総合政策・中央大学）取得。「ロボット PL」を研究しつつ、「インターネット法」にも取り組む。研究室 URL：http://www.fps.chuo-u.ac.jp/~cyberian

平野：米国では多くの交通事故が毎年、更には大きな列車事故が何年かに一度は起こりますが、ソフトウェアはそう大きな社会問題にはなりません。そういう意味で、信頼性や安全に対する認識に日米の違いがあるのかなと感じます。

松田：ソフトウェアは今、組込みソフトウェアとして様々な製品に入ってきています。ソフトウェアの役割が広がれば広がるほど、事故の影響は重大になり、ソフトウェアを提供する側の責任が大きくなると考えています。そのため、「このソフトウェアは安全です」という何かが必要になってきていると思います。

平野：たしかにソフトウェアはブラックボックス化していると言われます。では、製品の安全（すなわち欠陥か否か）はどう測ったらいいのか。製造物責任の考え方が一番進んでいる米国では、主に「危険効用基準」を用いています。英語の「risk utility test」です。「test」は「基準」を意味します。それと同じような概念で、経済学に近い言葉として「費用便益分析（cost benefit analysis：CBA）」があります。どちらも、完璧はあり得ないという観点から来ている考え方です。ただし、それだけが絶対的な尺度ではありません。例えば、消費者の期待を基準とするものとして、「消費者期待基準（consumer expectations test）」があります。これも、欠陥かどうかを測る要素と言われています。消費者の期待は商品ごとに違いがあります。装置は安全に機能することが当然、と多くの消費者は考えています。その期待が欠陥基準に大きな影響を与えるのです。例えば自動車であれば、基本的な機能は「走る」「曲がる」「止まる」ですね。ドライバーは「走る」「曲がる」「止まる」は、いつも通り確実に行われるものと考えて運転します。そして、いつもと違う動きになると、故障や誤作動（malfunction）、欠陥だという騒ぎになります。そのような考え方から推論すると、安全に動くことが当然という期待が高いものに対して、ソフトウェアもきちんと応えていないとまずいことになるでしょう。

松田：なるほど。

平野：しかし、場合によっては利用者の行動いかんで事故が起きることもあります。ローテクな話ですが、米国で 1992 年に、ファストフード店で熱いコーヒーをこぼして大やけどを負った人が訴訟を起こして、莫大な賠償評決額を勝ち取った悪名高い事件があります。熱い温度設定は学術用語でいう「設計欠陥」

に当たり得ます。レシピが設計であり、熱い温度設定のレシピが欠陥だという主張です。ところが、米国人にとってぬるいコーヒーはまずいものなのです。店としては、持ち帰って飲む時に冷めていないように、あえて温度設定を熱くしている。だから誤作動ではない。加えて、誰でもこぼすことはあるけれど普通はすぐに拭き取ります。そこで、すぐに拭き取らなかった利用者が悪いのではないか。つまりは誤作動ではなく、利用者側が回避可能な危険である。そしてこの危険は、冷静に考えればおいしさとトレードオフの関係にあります。コーヒーの温度を下げると、多くの消費者が望んでいるおいしさというベネフィットがなくなってしまうからです。しかしこの事件では、被害者が膨大な金額の賠償評決を勝ち取ってしまったので、「訴訟社会アメリカ」を象徴する事件として、世界的なニュースにもなりました。しかしこの種の PL 訴訟の判例傾向を私が調べてみたところ、こうしたケースは欠陥ではないという判例が圧倒的に大勢を占めていることが判明しました。しかし、消費者は、いつも期待しているものと違うと、欠陥だと思うものなのです。両極端の話をしましたが、組込みソフトウェアはどちらに近いのでしょうか。

松田：ソフトウェアもどちらだと言い切れないと思います。条件によっては、消費者の期待に沿わない誤作動をしたというケースもあるでしょう。消費者の使い方が悪いというケースもあるでしょう。ソフトウェアだからどちらだ、という議論にはならないと思うのです。昨年米国で、自動車のアンチロックブレーキシステム（ABS）が問題視されました。ABS はブレーキ時にタイヤが滑り始めたら、ブレーキを少し緩めてタイヤが路面をしっかりグリップするようにする機能です。昔はドライバーが自分でそれをしていました。そのことを知っている人にとっては、ABS の動作は欠陥でもなんでもない。しかし、知らない人にとっては、ブレーキを踏んでいるのに空回りしているように感じることもあるわけです。このようにユーザの経験によっても欠陥と認識するかどうかの違いがあり、ゼロかイチかで決められない気がします。

リスクとベネフィットを総合的に判断することが大切

平野：誤作動と言えるかどうか、グレーエリアの事象を判断するときに消費者の期待もその要素ですが、費用便益分析、危険効用基準が影響するように思います。つまりリスクとベネフィットとを考慮して総合的に判断するのです。ABS の場合、人間の反射神経には限界があるので、ドライバーに操作を任せるより機械に任せるほうが望ましいと考えて生み出されたシステムだと理解しています。しかし、一部のドライバーにとってはブレーキ感に違和感が生まれる。では ABS 機能をなくし、その分のコストを他の新規開発に回したとしても、結局リスクは減らないのではないか。米国では、何か問題があったときには、開発の際にそうした費用便益を正しく総合判断したのか否かも、多くの裁判で検討されています。

松田：費用便益を総合的に判断すべきだという先生のお話は、情報システムの信頼性をどこまで上げるべきかというテーマと関係します。日本人は徹底的に品質を追求して競争力を保ってきました。しかし、過剰品質になっているのではないか、それならばもっと別のところにお金をかけるべきだという考え方もあるのですが、なかなかコンセンサスが得られていません。

平野：それに付帯する話ですが、設計欠陥を考える基準として、米国のハンドという裁判官が1947年に考えた「B＜P×L（ビーピーエルと発音）」という有名な公式が参考になります。「B」は「Burden」で事故を回避するコスト負担、「P」は「Probability」で事故が起きる頻度、「L」が「Loss」で事故１件あたりの損失額を指しています。この公式の右辺の「P × L」はリスクを表しています。リスクを見るときに、損失額だけでなく事故が発生する頻度も考慮して測りましょうというのがこの公式の優れた点です。消費者は、事故を回避するためにはどんなにお金をかけても取り組むべきだと思うかもしれません。しかし、資源は有限だし、高いコストをかけるとそれが消費者の購入価格に跳ね返るので、結局消費者は高額な商品を望んではいないことが判明します。そういうことも考慮できるように作られた公式が「B＜P × L」なのです。普通、人は Probability の部分を計算に入れずに、Loss の部分を非常に大きく捉える誤謬（error）を犯す傾向があります。つまり、1件でも重大事故が起きたらそれを大きなLossと考え、それが１年間ではどのくらい起きる事故なのかをあまり考えないという誤りを犯すものなのです。

松田 晃一（まつだ こういち）
1970 年京都大学大学院修士課程修了後、日本電信電話公社入社。NTT ソフトウェア研究所ソフトウェア開発技術研究部長、株式会社国際電気通信基礎技術研究所（ATR）取締役企画部長、NTT コミュニケーション科学研究所 所長、NTT 先端技術総合研究所所長、NTT アドバンステクノロジ株式会社代表取締役常務、NTT-AT IP シェアリング株式会社代表取締役社長を歴任し、2008 年２月 IPA（独立行政法人 情報処理推進機構）IT 人材育成本部長に就任、2009 年１月より SEC（ソフトウェア・エンジニアリング・センター）所長。工学博士。

欠陥を 3 つに分類している米 PL 法

松田：一般の消費者は、商品の安全性について専門家の判断を大切にします。食品や建築物などたくさんの例があります。複雑なソフトウェアにも専門家の判断を求める時代も訪れそうです。同時にソフトウェアを作る側も、このソフトウェアは安全で信頼性の高いものだということをきちんと説明出来ることが重要だと考え始めています。消費者側と製造者側が同じような方向を向いてきているのです。そこで今 IPA/SEC では、中立的な第三者がソフトウェアについてのお墨付き的なものを与えられると消費者にとっても製造者にとってもいいのでは、と議論しているところです。ところで、PL 法は過失に関係なく製品に欠陥があれば責任を問うものだと理解してよいでしょうか。

平野：米国と日本の考え方は違っています。日本の PL 法は、欠陥があれば過失がなくても責任はあると説明されています。しかし、何十年も進んでいる米国の PL 法では、過失的な要素が不可欠であることが再認識されています。

松田：PL 法は欠陥を分類していると聞いていますが。

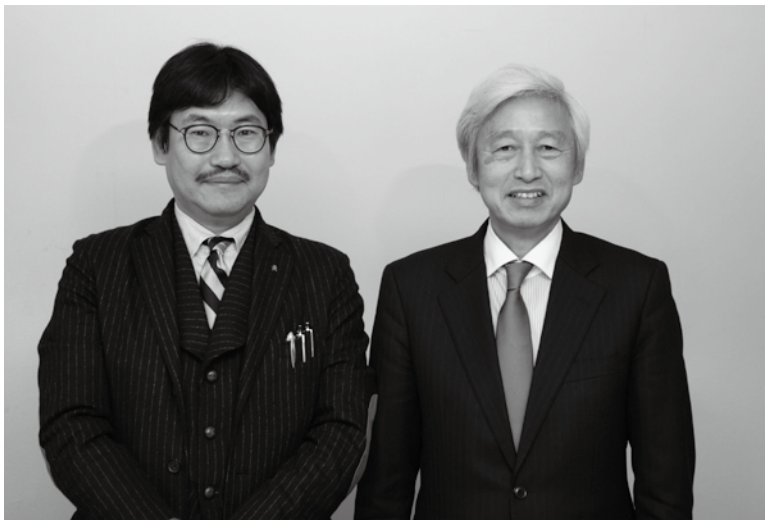
平野：米国を中心として、欠陥を 3 つに分類する学説が多くの支持を受けています。1 番目の類型は非常にプリミティブな形態である「製造上の欠陥」です。例えば、製品を 100 万個作ったときに検査をすり抜けて違うものが 1 つ出来てしまうという、誰が見ても欠陥と分かる古典的なものです。第 2 の類型は「設計欠陥」です。例えば、設計する際に助手席用のエアバッグを取り入れなかったという設計判断を欠陥とするものです。設計欠陥は、理論的には同じ図面で作った何百万台もの製品がすべて欠陥という扱いになるのでインパクトが大きいですね。この類型の訴訟の数も非常に増えています。3 番目の類型は 2 番目の派生で、「警告欠陥」です。例えば、持ち帰り用ホットコーヒーのカップの「熱いので気をつけてください」という記載が目立たなかったことでも警告欠陥を問われ得ます。この類型も訴えやすいので訴訟が多いものです。米国で一番支持を得ている考え方は、2 番目と 3 番目、設計と警告の問題は過失的に考えるという欠陥基準です。つまり、費用便益分析あるいは危険効用基準を中心とした欠陥判断を行う。1 番目の類型の古典的な製造欠陥だけは無過失責任で、費用便益を考えずに事故が起きたら問答無用で責任があるというのが米国の主流的な考え方です。

松田：欠陥には製造上の欠陥、設計上の欠陥、警告の欠陥があるということが分かりました。現在ソフトウェアは PL 法の対象となっていませんが、もしその考え方を当てはめるとするとどのようなものになるものでしょうか。

平野：有体物の場合、10 万個に 1 つの不具合を 100 万個に 1 つに減らそうとすると、膨大なコストがかかると言われています。しかしながら、前述したように製造上の欠陥は作った側が無過失でも責任を負うという判例が大勢です。たまたま欠陥品をつかんでしまったアンラッキーな人を救う倫理的な「分配的正義（distributive justice）」が可能である、という論拠がそこにはあります。事故が起きる頻度が非常に少ないので、1 人を救うコストを商品の価格に上乗せしても、みんなで支えきれるという現実的な理由もあります。もう一つの論拠は、メーカーは 10 万個に 1 個の不具合が発生することを知っているにもかかわらず製品を作り続けており、それで責任がないのは倫理的におかしい（unfair）という論拠です。さて、これをアナロジーでソフトウェアに適用するとどうなるか。二つの可能性があり得ると思われます。一つは、費用便益分析で考えるとして、ソフトでは、有体物製品の設計図面のような原案からそっくり同じものがコピーされるので、1 番目の類型の「製造上の欠陥」はあり得ない。しかし、設計上、10 億回に 1 回の割合で誤作動が発生する可能性があるとして、それを無くすようにすると膨大なコストがかかってしまい、消費者が買えるものではなくなる。そのため費用便益分析（すなわち設計欠陥）で考えるべきとして、有体物における製造上の欠陥の考え方をそのまま当てはめるのは無理ではないかという見方です。もう一つ、10 億回に 1 回誤作動が起きると分かっているのであれば、ソフトの製作者側に無過失責任を課して、常に被害者に賠償金を与えることとして、その賠償金額を広く浅く商品の価格に上乗せすれば危険あるいは損失をみんなで支え合うこと（これを「危険・損失の分散」と言う）が出来るという議論、つまり先の考えと異なり、1 番目の類型の「製造上の欠陥」を当てはめることが可能なのではないか。どちらを採用するか、判例が積み重ならないと見えてこないところがあります。

松田：ソフトウェアの場合、ハードで言うところの製造不良や経年変化による不具合は無く、バグがあれば、そのルートを通れば必ず不具合が起きます。つまり、設計段階で想定出来なかったためにバグが内在するのです。例を挙げると、ある銀行のシステムが止まって原因を調べてみたら、名寄せの機能にエラーがあり、600 万人の顧客の番号をしらみつぶしに調べたところ、不具合が起こる可能性があった組合せは結局その 1 件だけだと分かったのです。600 万分の一のケースを設計段階で想定してバグを無くすことは、ほぼ不可能と思います。ですからソフトウェアの欠陥を 3 分類で考えると、2 番目の類型の設計上の欠陥ということで費用便益とのバランスで判断が行われることが妥当だという気がします。

信頼されるソフトウェア品質監査のために



松田：ソフトウェアの場合でも、運用開始後10年以上も経ってからたまたまバグのあるルートを通り障害が起こることがあります。PL法には、「裁判時や事故発生時の技術水準から見てそういう事故を起こすのはおかしい」「いや、開発しているときには分からなかった」という議論があると聞いていますが、その点はどのようにお考えですか。

平野：法律的には、開発時・製作時の技術水準が基準になります。時代と共に技術は進歩しますが、10年後の基準で10年前の設計者の判断を裁くことはしません。

松田：そのためには、開発する側は、証拠として開発時点で実施した作業の書類を残しておくことが必要だということですね。

平野：それは当然、重要です。原告としては、「そういうことは知られていたはずだ」という証拠を出してきます。被告には、「これだけのことをしていました」という証拠があったほうがいい。もっとも、「分かっていたはずだ」という立証責任は原告にあるべきと一般に言われています。その逆に、「そういう知見は存在しなかった」という立証は非常に困難なもので、これを被告側に課すのは酷です。「悪魔の証明」という法律用語があるのですが、これは、不存在の立証は難しいという意味です。

松田：開発に際しては、その頃に分かり得る範囲で最高の技術で開発していることが証拠として残るように進めることが、開発側には大事なのですね。

平野：おっしゃる通りです。立証の元となるのは証拠であり証言です。証言は少し価値が劣ります。書面の証拠は証拠能力が高い。現実的な話をするとデジタルデータより手書きの日記が一番いいのです。改ざんが難しいですからね。

松田：第三者が監査するためには、開発する側が「どんな基準や標準に基づいてどのように開発を進めた」ということを証拠に基づいて客観的に示す必要があるということですね。

平野：裁判になると、権威のある基準を満たしているかに関して、証言や証拠の書面などが出て来ると、より信憑性が高くなるものです。

松田：そういう枠組みがうまく作れれば、ソフトウェアの品質監査を、社会的に信任が得られる形で実現出来るのではないかと考えています。

平野：米国では、陪審員や裁判官が欠陥の有無を判断する際に、証拠として提出された安全基準を高く評価するために必要な要素が幾つか挙げられています。1つ目は、安全基準自体が常に更新されていないといけません。古いままの基準を守っていても欠陥が無いとは言えないからです。2つ目は、安全基準は専門家が議論して作ったものであること。3つ目は公平性です。業界団体がお手盛り的に作ったものは駄目だけれども、消費者の代表や中立的な学者が入っているといったこと。おおざっぱに言ってこの3つを満たしていないと、その安全基準は高く評価してもらえません。

松田：まったくその通りだと思います。ところで、企業の財務監査も監査事務所が客観的なお墨付きを与えるようなものだと思いますが、米国では不正な財務報告を行ったエンロン事件が起こりました。報告にお墨付きを与えた監査事務所の責任はどうなるのでしょうか。

平野：監査事務所は、原則として責任を負わないような契約を企業と結んでいます。監査事務所が企業の監査を行うときに、企業から提供された情報はすべて正しいことを前提とするのです。

松田：ソフトウェアの品質監査的な仕組みが出来たとして、第三者が監査したソフトウェアで事故が起きた場合、監査機関も責任を問われる可能性があるのでしょうか。

平野：裁判の被告になる可能性は、原告に訴える自由・権利があるので避けられませんが、最終的な責任は契約次第で回避可能でしょう。通常、認証機関と依頼人であるメーカーとの間の契約では、万が一認証機関が訴えられた場合は依頼人側がすべての責任を負うことになっています。またそうでなければ、認証機関も認証業務を引き受けられなくなってしまうでしょう。

松田：本日は、たいへん参考になるお話をいろいろと伺うことが出来ました。ありがとうございました。これからも、IPA/SECでの検討に対して専門家としての見地でのご協力、ご支援をいただければ幸いです。

文：小林秀雄　写真：越昭三朗

検出漏れの無い割込み干渉検出システムの開発

株式会社豊田中央研究所
情報エレクトロニクス研究部
稲森　豊

アイシン精機株式会社
ソフトウェアセンター
山田　信幸

車載制御ソフトウェアでは割込み処理を多用するため、割込み干渉（割込み処理に起因するデータ競合）の未然防止対策は必須である。そこで我々はソフトウェア実装工程で作成されたCプログラムを対象に、割込み干渉が発生する箇所を漏れることなく検出するシステムを開発した。当システムは、第1ステップで同一の共有変数にアクセスする処理の組を判定箇所として網羅的に列挙し、第2ステップで抽象解釈やモデル検査等の手法を用いて、各判定箇所における割込み干渉の発生の有無を5種類の観点で判定する。なお本稿は、2011年1月18日に東京都･千代田区で開催された、第8回クリティカルソフトウェアワークショップ（WOCS2011）の一般講演をもとにしたものである。

1　はじめに

車載制御ソフトウェアには、高速応答性の確保等の理由により割込み処理を用いるが、使用には細心の注意を要する。とくに割込み干渉（割込み処理に起因するデータ競合、図1）はテストでの発見が困難なため、デザインレビューやコードインスペクション※1 等の多重的な保証手段を講じ、開発プロセス全体で高い信頼性を確保している。しかし、近年のソフトウェアの大規模化、複雑化の結果、保証に要する工数は増大しており、各工程の効率化が求められている。

プログラム作成後に実施するコードインスペクションの工程では割込み干渉の発生有無を複数人でチェックするが、制御の流れやアクセスメモリを調べる作業は難易度が高く、長時間を要する。そこで、割込み干渉の有無のチェックを大幅に自動化するために、表1の要件を満たす割込み干渉検出システムの開発を行った。解析対象は、単一プロセッサ上でOS無しに動作するCプログラムである。

ここでは、上記要件を満たすシステムを開発する上での考え方を中心に説明すると共に、開発システムの実務上の有用性を確認する。以降、第2節で割込み干渉について説明した上で、第3節で割込み干渉の検出に至るまでの解析の流れを概観し、第4節で個別の解析手法について説明する。第5節では開発したシステムを紹介すると共に、製品向けプログラムを対象に評価した結果を示す。

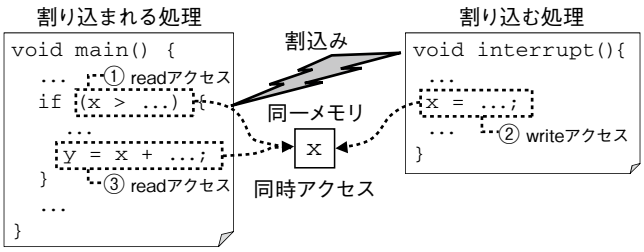


図1　割込み干渉の例

2　割込み干渉

割込み干渉は、メイン処理と割込み処理、もしくは、割込み処理同士のデータ競合であり（図1）、以下のa、bを共に満たす挙動である。

a：ある処理がメモリに2回アクセスする間に割込みが発生し、別の処理が同一メモリにアクセスする（図1で①、②、③の順の実行パスが存在する）

b：上記アクセスのうち少なくとも1つはwriteアクセスである

図1で、①と③の間で割込みが発生し、②でwriteアクセスが起こると、①と③で読み込むxの値が異なる。これが意図しない挙動であればプログラム修正が必要である。

以降、説明の簡略化のため、割り込まれる処理での第一アクセスを①、第二アクセスを③、割り込む処理のアクセスを②と呼ぶことにする。

3　割込み干渉検出までの解析の流れ

割込み干渉を漏れなく検出するために、以下のアプローチを

表1　割込み干渉検出システムの要件

要件	内容
割込み干渉を漏れなく検出する	割込み干渉の原因箇所をテスト工程以降に残さない
割込み干渉の誤検出を低減する	誤検出箇所（割込み干渉が発生しないのに誤って検出した箇所）の人手による再チェック工数を軽減する
人手の介在を最小限に留める	出来る限り自動化することによりチェック工数を削減する

採った。第１ステップで割込み干渉の可能性のある箇所を網羅的に列挙し、第２ステップで各箇所での割込み干渉の発生可能性を様々な観点から判定する。そして、すべての判定で割込み干渉の可能性が無いと判定されなかった箇所を「割込み干渉の可能性のある箇所」として検出する。

第１ステップで、割込み干渉の可能性のある箇所を網羅的に列挙するには、メイン処理と割込み処理がアクセスする大域変数を洗い出し、共通の大域変数にアクセスする処理の組を列挙する。そして、図２に例示するように、割込み干渉の判定箇所リストを自動生成する。

第２ステップで、各判定箇所を対象に、表２に示す５つの観点で判定する。検出漏れを防ぐため、各判定は「割込み干渉なし」か「割込み干渉の可能性あり」かを判定し、段階的に割込み干渉の可能性がある箇所を絞り込む。

このように、観点の異なる判定を多重的に配置することにより誤検出を少なくしている。また、解析時間の長い判定を後段に配置することにより全体の解析時間を短縮する工夫を行っている。第４節で各判定の詳細を説明する。

4 解析手法の詳細

4.1 アクセスパターン判定

各判定箇所について、第２節で示した割込み干渉の条件 b を満たしているかを判定する。すなわち、共有変数へのアクセスのうち１つも write アクセスがなければ、「割込み干渉なし」と判定出来る。

判定項目	共有変数	割り込まれる処理							割り込む処理					判定結果
		関数名	アクセス① 行番号	アクセス① 種類	アクセス③ 行番号	アクセス③ 種類	呼出し元 処理名	呼出し元 優先度	関数名	アクセス② 行番号	アクセス② 種類	呼出し元 処理名	呼出し元 優先度	干渉なし／干渉の可能性あり
1	x	fun1	110	read	113	read	main	0	sub1	15	write	int1	3	?
2	y	sub1	53	write	54	write	int1	3	sub2	22	read	int2	5	?
…	…	…	…		…		…	…	…	…		…	…	…

メイン処理main が呼び出すfunc1() の①110行目のreadアクセスと③113行目のreadアクセスの間に割込み処理int1 が呼び出すsub1() の②15行目のwriteアクセスが起こるか?

図2　判定箇所リスト

表2　判定の観点

判定内容	観点
アクセスパターン判定	アクセスの種類と回数の条件を満たすか
アクセス同時性判定	割り込まれる処理における２つのアクセス箇所（①、③）は同一の制御フロー上に存在し得るか
割込み禁止状態判定	割り込まれる処理の２つのアクセス箇所（①、③）は割込み禁止命令で保護されていないか
メモリ同一性判定	判定対象の２つの処理における共有変数へのアクセス（①、②、③）はビット単位で同一か
実行パス存在性判定	割込み干渉を発生させる実行パス（①、②、③の順に通過する実行パス）は存在し得るか

4.2 アクセス同時性判定

割り込まれる処理の２アクセス（①、③）が同一の実行パス上にあり得るか判定する。①と③が分岐文の then 節と else 節に分かれている場合等には同一パス上に無いため、割込み干渉の条件 a を満たさない。当判定のために、前処理で制御フローグラフを生成し、各文からの可到達文を求めておき、①から③への可到達性を判定する。

4.3 割込み禁止状態判定

割り込まれる処理の２アクセス（①、③）が割込み禁止命令で保護されているか判定する。すなわち、①から③の間継続して割込み禁止状態であれば、割込み干渉の条件 a を満たさず、「割込み干渉なし」と判定出来る。

この判定条件は以下の X、Y を共に満たすことである。

X：①の実行前に割込み禁止状態である

Y：①と③の間に割込み許可命令が無い

条件 X の判定には、①の直近の命令が割込み禁止命令か割込み許可命令かを調べればよいが、ループ文や関数呼出し文がある場合にアルゴリズムが煩雑となる（図 3）。

そこで、抽象解釈を用いる方法を考案した。抽象解釈[JSSST1995] は、変数の取り得る範囲等、プログラムの一側面を調べるために、プログラムを実際に実行させるのではなく、「抽象的」に実行する仕組みを提供する。そこで我々は抽象解釈を利用して、各文における割込み状態（禁止状態／許可状態）を算出するアルゴリズムを開発した（図 4）。

まず、割込み禁止命令以降は「禁止状態」、許可命令以降は「許可状態」である。また、分岐文の合流点の割込み状態算出用に OR 演算を用意する。例えば、「許可状態」と「禁止状態」の OR 演算結果は「許可状態 or 禁止状態」である。

抽象解釈を行う起点は各関数の先頭とし、関数毎に抽象解釈を行う。各処理の先頭を起点とした場合、複数の処理から呼び出される関数の扱いが難しいからである。各関数の先頭を起点とした場合に問題となるのは、先頭文の実行前の割込み状態が不明な点である。そこで、「関数呼出し元の割込み状態を継承する」という意味を持つ「継承状態」を設け、先頭文の実行前の割込み状態を「継承状態」とする。また、合流時の OR 演算も「継承状態」に対応した定義とする。ループ文内の割込み状態は、抽象解釈の繰り返し計算により正しく算出出来ることが保証されている [JSSST1995]。

そして、①の実行前の割込み状態が「継承状態」を含む場合（関数呼出し元の割込み状態に依存していることを示してい

脚注

※1　コードインスペクション：プログラム等の開発物を人の目で見て検証する作業のこと。

る)、あらかじめ用意しておいた関数コールグラフを用いて関数呼出し元を特定し、それらの実行前の割込み状態を調べ、図5に示す演算を行う。これにより、所望の割込み状態を算出出来る。

最後に、条件Yについては、①から可到達、かつ③へ可到達な割込み許可命令の有無を調べればよい。ただし、間に関数呼出し文がある場合は、関数呼出し先についても割込み許可命令の有無を調べる必要がある。

4.4 メモリ同一性判定

割込み干渉の条件aの「同一メモリ」について判定する。すなわち、①、②、③のアクセスメモリの同一性を判定する。その際、ビットフィールドを利用したビット変数の使用を想定し、ビットレベルで同一性を調べる。誌面の関係上詳細は省略するが、ビットレベルのメモリ情報を保持するメモリモデルを構築し、C言語の任意の式がメモリモデルのどの部分を指すかを算出する仕組みを開発した。

4.5 実行パス存在性判定

最後に、割込み干渉の条件aである「①、②、③の順の実行パスが存在するか否か」を直接判定する。当判定では、割込みタイミングや多重割込みの発生の仕方等、割込み処理の挙動をすべて調べ尽くすことにより、当該実行パスの存在性を判定しなければならない。

そこで、当判定にモデル検査器SPINを用いることにした。SPIN [中島 2009] は並行プロセスの検証に適したものである

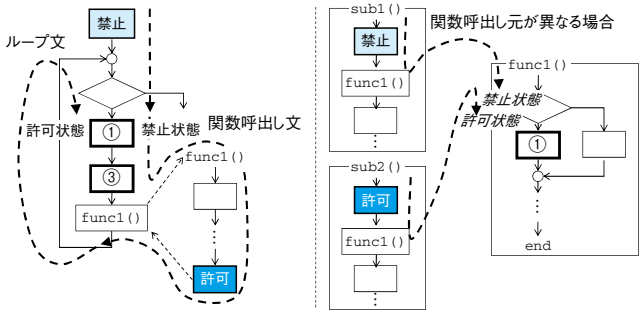


図3　割込み状態の解析が難しい例

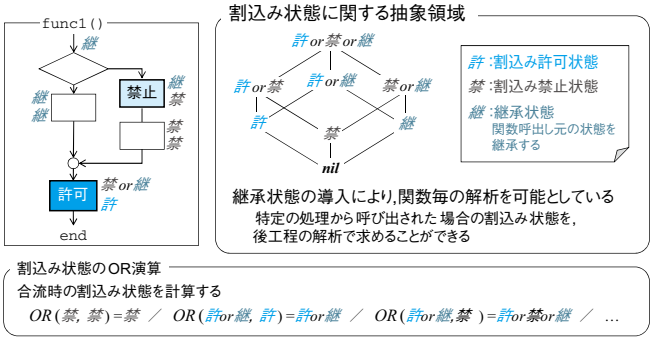


図4　抽象解釈を利用した割込み状態算出

が、モデル化の工夫により割込み処理の網羅的な挙動の検証にも利用出来る。ポイントは、SPINの並行プロセスを割込み処理として振舞わせるための動作条件の書き方である（図6)。

また多重割込みをモデル化するためにスタックを用意し、割込み発生時の実行状態をスタックに積み、割込み処理終了時に実行状態を復元する。各処理内の文はc_code命令を用いて文単位でプロセスモデルに埋め込む。これにより、各文の間で割込みが起こることを表現出来る。

ただし、すべての文をモデルに埋め込むとモデル検査は状態爆発※2を起こすため、判定に必要な文のみをCプログラムから抽出しモデルに埋め込む。その抽出にはプログラムスライシング技術※3 [下村 1995] を応用する。具体的には、アクセス箇所①、②、③について静的なバックワードスライシングを行うが、各処理は繰り返し実行されるため、ループ文に対する静的スライシングとほぼ同様のアルゴリズムとなる。今回開発したアルゴリズムと上記アルゴリズムとの違いはオーバー近似を行う点である。制御依存関係を辿る際に特定の条件でスライシングを打ち切ることにより、抽出コードを縮減出来る。ただし、実際に存在しないパスがモデル化され誤検出の原因となる（検出漏れは発生しない)。

最後に、①、②、③の順に実行パスが存在するか否かを検証するためのアサーション文※4を挿入し、SPINを実行することにより当判定が行える。

5 システムの開発と評価

我々はシステム開発の当初から実務適用を目指し、製品向けCプログラム1事例を代表的なターゲットとして設定した。代表事例はソースコード数万行で、メイン処理及び4つの割込み処理から構成される。開発目標として、検出漏れなし、自動判定率（当システムによる自動判定によって「割込み干渉なし」と判定された箇所数が判定箇所数全体に占める率）90%以上を設定した。

開発の初期段階から技術課題として挙がったのは、モデル検査の状態爆発である。第4節でプログラムスライシングについて述べたが、その他にも、無関係な割込み処理や割込み禁止命

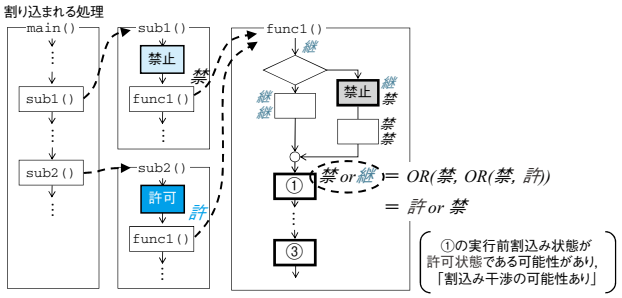


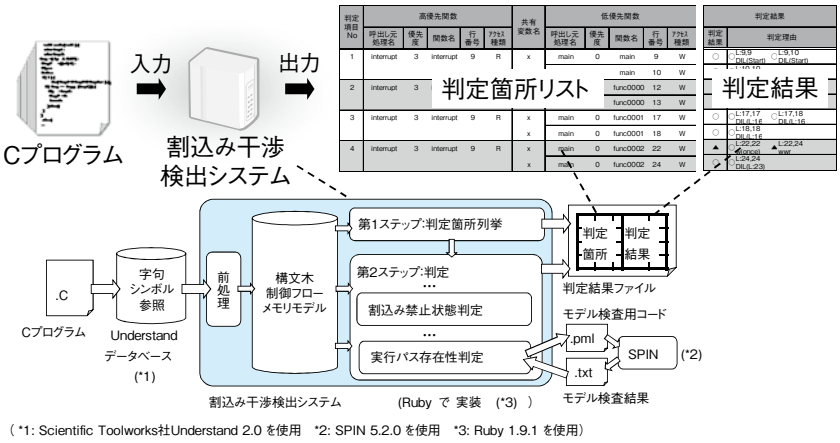
図5　関数コールグラフを利用した割込み状態の算出

令、許可命令の除去を行うアルゴリズム等を開発することにより、モデル検査用コードの行数を 20 ～ 50 分の 1 に縮減出来、状態爆発による SPIN の異常終了を全体の約 15% にまで抑えることが出来た。

また、開発中盤には解析時間の問題が出た。代表事例の解析に約 6 日かかり（CPU：Intel Xeon X5570 2.93GHz)、その原因を追究した結果、プログラムスライシングとモデル検査の計算が全体の 95% 以上を占めることが判明した。そこで、モデル検査をなるべく使わないように工夫した結果が、表 2 で述べた 5 つの観点による判定である。

開発システムの構成をに示す。前処理部で構文木や制御フローグラフ、メモリモデルを生成し、それらの情報を利用して各種判定を行っている。モデル検査を行う部分も、コードの生成、SPIN の起動、結果の解析までを、解析の本体部が自動で行う。

代表事例による評価結果は表 3 の通りである。



（*1: Scientific Toolworks社Understand 2.0 を使用　*2: SPIN 5.2.0 を使用　*3: Ruby 1.9.1 を使用）

図7　システム構成

表3　代表事例による評価結果

項目	評価結果
検出漏れ	なし（目標通り）
自動判定率	94.3%（目標値以上）
人手の介在	マイコン依存コードの修正のみ

従来すべての判定箇所を人手でチェックしていたのに対し、システム導入により人手でチェックする判定箇所数（「割込み干渉の可能性あり」と判定された箇所数）を大幅に削減出来ることの確認が出来、コードインスペクションの工数を大幅に削減出来る見通しが得られた。人手の介在については、アセンブリコードの修正、削除等、簡単な作業のみであり、利用者の負担を抑制出来ることを確認した。

6　最後に

本稿では、C プログラムを対象に割込み干渉の発生箇所を検出するシステムについて説明した。とくに、抽象解釈を用いた割込み状態の特定により、共有変数へのアクセス保護の有無を精度良く判定出来、無保護のアクセスに対しては、モデル検査を用いて割込み干渉となる実行パスの存在性を検証するところに特徴を持つ。数万行規模の C プログラムに対して適用可能である 1 事例ながら性能面での実用性を確認出来たため、現在は別の複数事例で評価中である。

車載ソフトウェアの大規模化に伴い、高信頼性を確保するための工数が増加の一途を辿る中、当システムは開発期間の短縮に貢献する見通しである。今後は視野をやや広げ、割込み干渉を含めたタイミングに起因する問題に対して設計手法と解析手法の両面から解決する枠組みについて研究していく予定である。

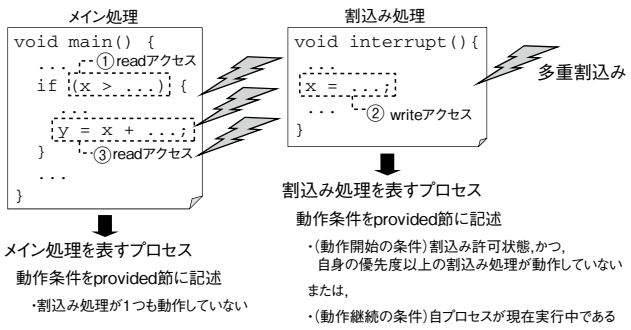


図6　並行プロセスによる割込み処理のモデル化

脚注

※2　状態爆発：検査対象（ここでは C プログラム）の取り得る状態の数が膨大となること。その結果、モデル検査はメモリ不足で異常終了する等、正しい応答が得られなくなる。

※3　プログラムスライシング技術：プログラムからある目的を果たすためのソースコードを抽出する技術。

※4　アサーション文：プログラムやモデルのある箇所で特定の性質を満たすべき場合に、その箇所に「assert (< 性質 >)」等の形式で挿入する文。モデル検査等の検証に使われる。

参考文献

[JSSST1995] 日本ソフトウェア科学会：抽象解釈とその応用：大会併設チュートリアル，日本ソフトウェア科学会，1995

[下村 1995] 下村隆夫：プログラムスライシング技術と応用，共立出版，1995

[中島 2009] 中島震：SPIN モデル検査：検証モデリング技法，近代科学社，2008

特定デザインパターンに基づく大規模基幹システムのオープン化技法

北川　陽一†

証券リテール基幹システムの再構築に際し,レガシーな既存基幹システムのオンライン処理システムをオープンシステムへ再構築した.通常のオブジェクト指向分析設計では膨大な期間・工数を要すると予想されていたが,今回の再構築のために開発された特定デザインパターンによる手法を適用することにより,期間・工数を大幅に短縮することが出来た.

また,我々が当手法を用いた再構築を完了してから既に5年以上が経過しているが,その後の度重なるエンハンスのための保守・開発においても,特定デザインパターンによる手法を適用することにより開発効率,品質が維持されている.ここでは,その手法とその効果について紹介する.

The methodology for restructuring a large-scale host system,into open system based on original design patterns

Yoichi Kitagawa†

We have restructured "a legacy large on-line system for securities retail business" from the host system to the open system. Although a usual object-oriented analysis and design seem to require an enormous term and cost, we established a methodology that can reduce both a term and cost drastically by applying an original design pattern developed for this restructuring. Five years or more have already passed since the completion of restruction. Although the system has been enhanced many times, the development efficiency and the quality level are retained high. Accordingly we think that this methodology can be expected adequately to be effective.

1　はじめに

システムを全面再構築する場合は，同時に業務アプリケーションソフトウェアロジックも全面的に刷新することが多い．そのためには新たに業務要件定義の実施から始めることになる．通常のオブジェクト指向開発技法を含めてこの業務要件定義では，通常は業務精通者から要件と業務ロジックをヒアリングする若しくは旧システムの設計書から要件を抽出し，ビジネスモデルを作成することでシステム化の範囲と要件を決定していく．旧システムにおいて設計書が完備されている場合は当該技法による開発が可能である．しかし設計書が不十分な場合は業務の担当者から要件をヒアリングし，ビジネスモデルを作成する必要がある．これを大規模基幹システムの再構築に適用すると，莫大な人数と時間が必要となり，一般企業であれば経営上許容し得ないコストと期間が求められることが予想される．

我々のプロジェクトでは，開発にあたって次の前提が置かれていた．

① 約２年半以内での開発完了．

② 旧オンラインアプリケーション※1における業務要件の全面把握と新システムに必要な業務要件の確定．

③ 新しいオープンシステム環境に合致するようにオンラインアプリケーションを刷新．

④ 保守生産性の向上と維持コストの低減．

しかも旧システムの設計書が最新のプログラムを反映していないという状況の中で，これらの前提を解決するための開発技法を考案し，実践した．

2　システム再構築プロジェクトの全体像

2.1　システム再構築プロジェクトの範囲

システム再構築プロジェクト（以下、当プロジェクト）の上位プロジェクトでは，バッチシステムや過去データを含んだデータベースの移行も含めた旧証券業務システム全体の再構築

† 日本証券テクノロジー株式会社, Nippon Securities Technology Co., Ltd.

を行っている．「オンラインシステム」と「バッチシステム」は，図１に示す通り，順次レガシーシステムからオープンシステムへ再構築を行った．そのうち，当プロジェクトは，図２に示す通り，オンラインシステムの再構築を対象にしている．

オープン化にあたり，オンラインシステムは最もその恩恵を受けるシステムであったため，最初の開発対象とした．

2.2　オンラインアプリケーション要件の概要

当プロジェクトでは，システム化業務のうち、次に述べる業務を新証券業務システム上で再構築し，「コンプライアンス」業務を新規に構築した．

・ 株式，債券，投信等の「営業店からの注文」を担う業務
・「取引所への注文」，「約定の受取り」を担う注文約定業務
・「顧客管理」，「銘柄管理」等のマスター管理業務
・「証券管理」，「経理処理」等の残高管理業務

トランザクション性能などの非機能要件については，システムの構成が新旧システムで全く異なり全面的な新規対応となった．またユーザインターフェース（UI）については，画面デザイン等は新旧システムで全く異なる新規構築となったが，入出力項目等は旧ホスト画面から入出力項目などが容易に把握可能であった．

なお非機能要件及びUIについては当論文の対象外としている．

脚注

※1　COBOL 言語換算で 300 万ステップ．ただしフレームワーク及びデータ定義部分は除いた業務ロジック部分のみ．

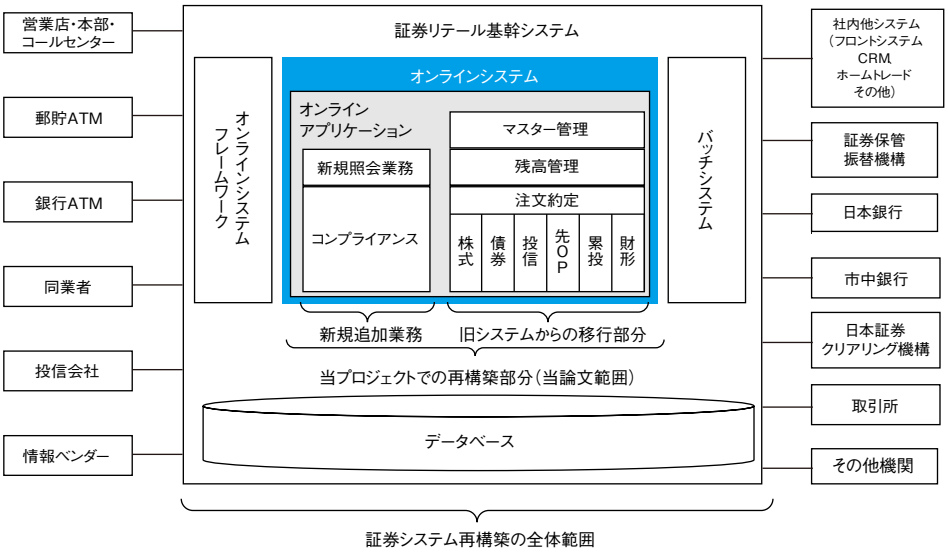


図1　証券システム再構築の範囲

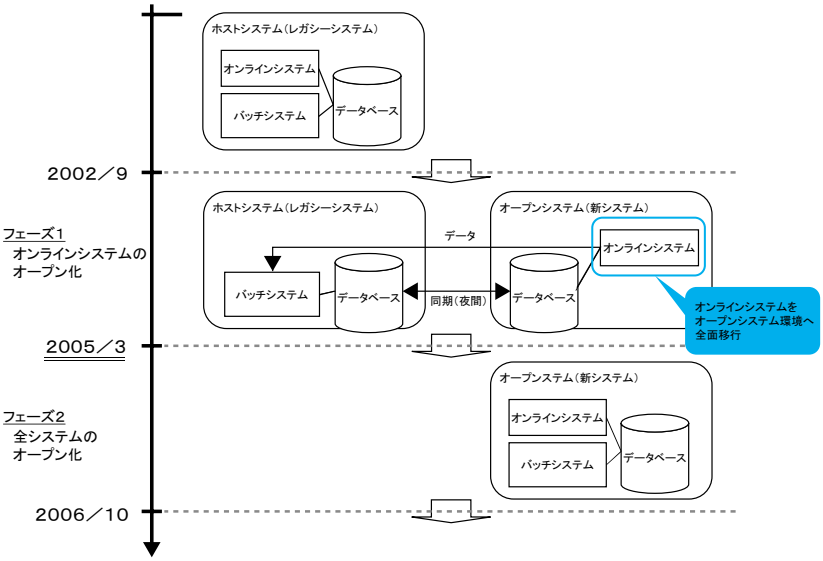


図2　オープン化のフェーズ分割

3 オンラインシステム全面再構築目標

3.1 オブジェクト指向開発

旧システムでは開発言語にCOBOL言語を用いていたが，オープンシステムへ再構築するに当たり，GUIの採用や基盤システムとの親和性確保の点を考慮し，オンライン処理の開発言語として全面的にJava言語を適用した．そしてJava言語の特徴を活用するため，分析・設計・製造の方法論にはオブジェクト指向開発方法論の適用を目指した．

3.2 再構築目標

(1) 保守担当者の立場に立った保守性の確保

証券業務システムでは，制度変更等に対応するために業務要件の追加や修正が頻繁に発生する．そのため基幹業務を担う業務アプリケーションは，頻繁に追加や改修が要求される．この保守要求に対し，実施が容易な構造となっていなければならない．

(2) 旧システム業務要件の把握と新システムとしての仕様の確定

旧証券業務システムから新証券業務システムへの移行時において，ユーザ要望による要件変更を除いては，システム側都合で業務フローの変更を強いることは避けなければならない．また，当証券業務システムは公的機関との連携も含めて多数の他システムとの連携が存在するため，旧来通りの仕様で他システム連携を実現することが必達の要件となる．

(3) プログラム設計構造の刷新

オープン化された新しいプラットフォーム環境に適合したプログラム構造にする．なお旧システムのプログラム構造は長年の保守の繰り返しによりロジック分岐の錯綜化が進んでおり，そのまま引き継ぐことは受容出来ない状態であった．

(4) 開発制約がある中での遂行

新システムの開発期間では，着手から終了まで約2年半であったが，その間，弊社の業務精通者の大半が旧システムの保守に従事し，新システム開発に携わることが出来る業務精通者は少数に限られていた[※2]．この参画出来る業務精通者に制限がある中で，開発を遂行させることが求められた．

4 システム再構築における問題

4.1 要求分析・要件定義に関する現実的な問題

当システムの規模を表1に示す．

これらのシステム規模に対して，要求分析・要件定義の把握を行うに際して，弊社では以下の状況であった．

① 業務要件が多様で複雑なため，要件のすべてを把握しているキーマンがシステム開発部門及び業務部門の双方にいない．

② 詳細な仕様まで含めた業務の機能要件の大部分は，実際にその機能を利用している業務の担当者でなければシステムの動作を説明出来ない．

③ 業務部門においては，業務を定義した文書の整備が完全ではないところがあった．また一部の担当者のみが利用している機能も多く，担当者と利用機能の関連付けの多くはあいまいであった．

④ システム開発側部門においては，度重なる制度変更に対して設計文書の改修が追従出来ておらず，設計書が最新となっていなかった．

4.2 オブジェクト指向開発における基本的なプロセス

オブジェクト指向開発で採用する方法論としては，OOSE Objectory法[※3]，OMT法[※4]，Booch法[※5]，UP法[※6]，RUP法[※7]等の方法が一般に広く知られており，実際の基幹業務システムの開発現場においては，これらの手法を組み合わせた方法で開発される場合が多いと考えられる．

この通常用いられるオブジェクト指向開発の基本的なプロセスの流れを図3に示す．

このプロセスの流れは新規開発でも再構築においても共通であり，通常オブジェクト指向開発技法で用いる開発プロセスとして一般に知られているものである．

4.3 通常オブジェクト指向開発の問題点

「実装・テスト」を除く上流工程側の各プロセスにおける代表的作業と現場詳細作業を表2に示す．

表2のプロセス通りに開発を進捗させるためには，業務部門の全社員からヒアリングを行って要件を洗い出し，1業務毎に1ビジネスモデルを完成させなければならない．各社員には日

表1 当オンラインアプリケーションシステムの規模

項目	値
画面数（画面遷移の都度表示される画面を1画面と数えた場合）	約6,000画面
ユーザが利用する業務数（入力開始から完了までのトランザクション単位に完結する業務数）	約2,400業務
取引所等からの受信電文を起点に，処理開始から完了までのトランザクション単位に完結する業務数	約300業務

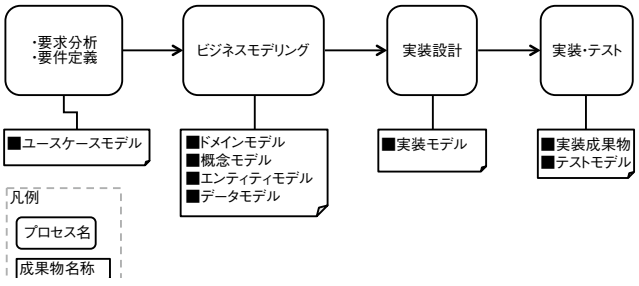


図3 通常オブジェクト指向開発における概要レベルプロセスの流れ

常の多忙な業務の合間を縫って時間を割いてもらう必要がある．

実際のプロジェクトではヒアリング及びビジネスモデル策定に要する期間の見積りは厳密には実施しなかったが，その見積りを行うと表3の通りとなる．

ここで，業務数を表1の2,400と300の和である2,700と置くと，全業務合計の工数は，OOD※8精通者工数が1,350人月，業務部門社員工数が1,105人月となる．当プロジェクトにおいて現実に通常の業務に支障をきたさず参画可能な全国の本支店の業務部門社員の人数は，期間中の平均で約20名が限度であった．その結果，必要な期間は上流工程プロセスだけで最低で55ヶ月となる．また一人一度はヒアリングを実施しなければならず，そうなると必要期間は更に延びることとなる。

4.4 通常オブジェクト指向開発におけるモデル化の限界

当証券業務システムでは，ユースケースモデルやドメインモデルを用いて要件を把握するにあたり，一つひとつの業務のモデル化自体はそれほど難しくは無い．しかし全社の様々な部署のおのおのの担当者でなければ把握出来ない機能が集まっており，これら多様な業務要件に対応したモデル化作業の実施には，前述の通り要件定義から基本設計まで最低でも5～6年以上を要し，実装からテストまでを含めると更に数年を要すると見積もれる．一方，当証券業務システムでは年間平均で50～60件以上の仕様変更要件が発生する．大規模開発の場合は仕様の凍結期間が必要になるが，その期間は経営の観点や，証券制度の変化の速さを考えるとせいぜい1年程度が限度である．要件確定に5～6年を要するのであれば，現状要件の把握が完了する前から変更要件の取り込みが必要になるという事態を招き，要件確定そのものが実現困難となる．またこのような長期間に及ぶ開発は，経営の観点からも受け入れられないものであった．

表2　各プロセスにおける代表的作業と現場詳細業

プロセス	代表的作業	現場詳細作業
・要求分析 ・要件定義	問題領域の捕捉	・担当者へのヒアリング ・設計図書調査
ビジネスモデリング	問題領域の視覚化または明文化	システム化対象ビジネスのプロセスやエンティティまたはオブジェクトの可視化
実装設計	実装モデル設計	実装オブジェクトの設計（相互作用図作成，実装クラス図作成等）

表3　当システムの業務要件把握に要する1業務当たりの見積工数

		・要求分析 ・要件定義	ビジネスモデリング	実装設計	合計
OOD精通者	工数	1人日	5人日	5人日	11人日
業務部門の全社員	工数	2人日	5人日	2人日	9人日
	想定作業	主にヒアリング	OOD精通者との共同作業	主にレビュー作業	

5 新技法の導入による解決策

通常オブジェクト指向開発における前記問題点を解決する目的で，以下の手順を取り入れた新しい技法を考案した．

5.1 旧システム業務要件の情報源

現場担当者へのヒアリングに代えて，旧システムのCOBOL言語アプリケーション（以下，旧COBOL）ソースコードを最大の情報源として活用する．ソースコードは開発側担当者が全数入手可能で，かつ記述されているコードは旧システムの業務要件をすべて内包している．もう一つ欠かせないものが，プログラムを起動するトリガーとなる画面や外部システムからの入力情報である．これらの情報も開発側ですべて管理されており，業務要件を理解するための重要な情報として活用出来る．

表4　業務の分類観点毎クラス定義表

No.	観点	クラス名称	定義単位
1	業務のグループ	Control	複数の一連の業務を集めたグループ．
2	業務の処理目的	Route	処理目的毎に1単位を定義する．Route毎にStage及びStageの実施順序を定義する．
3	抽象的に著される処理段階	Stage	業務処理目的（Route）を実現する処理段階毎に1単位を定義する．Stage毎に副Stage（SubStage）またはUnit及びそれらの実行順序を規定する．またUnitとRouteの依存関係を持たせ無いように分離する．
4	単位処理	Unit	業務処理目的（Route）には依存し無い独立した処理単位．DB参照／更新や外部システムとのデータ授受など，実際の機能を実装したもの．

脚注

※2　弊社には約100名の業務精通者がおり，そのうち約1～2割が新システム開発に従事した．
※3　OOSE Objectory法：ヤコブソン氏によって開発された，オブジェクト指向ソフトウェアによる情報システム構築の方法論．
※4　OMT法：ランボー氏などによって開発されたオブジェクトモデル化技法．
※5　Booch法：ブーチ氏によって開発されたオブジェクト指向開発方法論．
※6　UP法：OOSE Objectory法，OMT法，Booch法を統合して開発されたオブジェクト指向開発方法論．
※7　RUP法：UPをベースに開発されたラショナル社の開発方法論．
※8　OOD：Object Oriented Design，オブジェクト指向設計．

5.2 旧 COBOL プログラムの把握

旧 COBOL プログラムの全ソースコードを業務ロジック単位のモジュールに分解する．そして分解後のモジュール毎に担っている機能とモジュール間の関連を全数把握する．

続いて，各業務アプリケーションプログラムの処理を開始させるイベントを把握する．一般にシステムイベントと呼ばれるものであるが，当システムでは次に挙げる 3 種類のイベントがある．

① 画面入力

ユーザの画面操作によるもの．

② 外部システム入力

システム対システムの情報授受によるもの．

③ システム内部から発生する業務

ある業務処理が別の業務処理を呼び出す，または別の処理に引き継ぐ．

5.3 旧システムの業務要件の分類

旧 COBOL ソースコードに内包されている業務要件の個々の業務を次の 4 つの観点で階層的に分類する．

① 業務グループ．

② 業務目的．

③ 個別業務の処理順序．

④ 個別業務の業務処理ルール．

実作業では，旧 COBOL ソースコードを分解した前述の個々のモジュールを①～④のそれぞれの観点で分類する．

その分類方法は，実装と業務のそれぞれから「ボトムアップ※9観点」で業務を分類する方法である，と我々は自己評価している．なお，この分類作業は今回適用した技法の中核を為す．

これらの分類に用いる階層をクラスとして定義したものを表 4 に示す．

次に，これらのクラスの階層関係を表したモデルを図 4 に示す．Route が保持している Stage を TopStage，Stage が保持している Stage を SubStage と呼んでいる．

当技法では，このクラス図で表すモデルを当プロジェクト特定のデザインパターンと位置付け「RSU※10パターン」と呼んでいる．この RSU パターンを，当証券業務システムオンライ

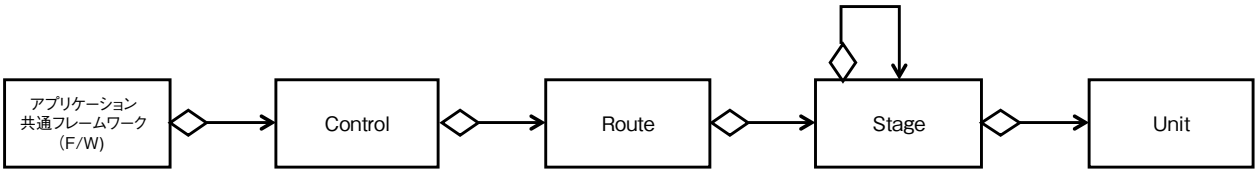


図4　業務の分類観点をクラスとして定義したクラス図

画面名	Control名	(和名)	Route名	TopStage名	SubStage名またはUnit名			
			△△△	×××リード	・・・			
				×××依頼	・・・			
			□□□	×××チェック	×××チェック1−1			
				×××リード	×××リード1−1			
					×××リード1−2			
				×××チェック2	×××チェック2−1	×××チェック2−1−1	×××判定−1	・・・
							×××判定−2	・・・
						×××チェック2−1−2	・・・	
						×××チェック2−1−3	・・・	
					×××チェック2−2	・・・		
					×××チェック2−3	・・・		
					×××チェック2−4	・・・		
					×××チェック2−5	・・・		
					×××チェック2−6	・・・		
				×××更新	・・・			
				×××チェック−2	・・・			
			▽▽▽	×××チェック	・・・			
				×××リード	・・・			
				×××チェック−2	・・・			
				×××更新	・・・			
				×××チェック−3	・・・			
				×××更新−2	・・・			
				×××更新−3	・・・			
				×××出力	・・・			
注文	xxxxxx	(xx注文)						

図5　RSUパターン構造の例

ンアプリケーションプログラム構造の共通デザインパターンとして規定した.

次に画面，Control，Route 等の関係を示した RSU パターン構造のサンプルを図 5 に示す.

また，当プロジェクトで再構築したオンラインシステム全体の構成を図 6 に示す. アプリケーションソフトウェアは F/W※11 の上に，RSU パターン構造クラスとして実装されている.

6 適用技法の特徴

当適用技法の最大の特徴は，業務の観点からシステム要件の定義，設計及び実装を行う「ボトムアップ型」の技法である点である．この前提の下，3 節で述べた再構築目標と照らし合わせて，当適用技法についてデザインパターン部分と開発方法論のそれぞれの特徴を一元的に集約して述べる.

6.1 Route 分類の容易性

図 7 に示す簡単な処理フロー図により，Route 分類された設計構造と非 Route 分類構造の違いを説明する.

RSU パターン設計構造側では，おのおのの処理が何れかの Route に分類されているため，設計構造が単純で処理順序の定義（Stage 分類）も Route 毎に行うことが出来る．一方，非 RSU パターン構造では業務処理 A 及び B はサブルーチン内で密結合となり，設計構造が複雑である．サブルーチンを単位処理として見ると非 RSU 構造の方が効率的に見えるが，サブルーチン内部の構造は複雑となり，業務目的と処理内容の対応付けがむしろ困難なものとなる.

旧オンラインシステムにおいては図 7 で示すような，処理が錯綜した構造を持ったサブルーチンが多く存在していた．しかしこのような複雑な業務処理であっても、RSU パターン構造とすることで単純に表すことが出来る．従って RSU パターンは多くの処理システムに対して共通デザインパターンとして適用出来る可能性がある.

6.2 旧システム業務要件の把握

(1) 分類法

旧 COBOL ソースコードから前述の通り分解したモジュールを CRSU※12 の観点で分類する．旧オンラインシステム業務の全業務において，「目的」は必ずいずれかの Route に分類され，「実現方法」は Stage 及び Unit で分類することが出来る．なお，当プロジェクトでは，これらの分類結果を得るために旧 COBOL ソースコードを基に業務ロジックを定義した単位※13 に CRC カード※14 を興し [LARMAN03]，RSU 観点で分類するという方法を用いた.

また各業務アプリケーションプログラムの処理開始イベントを全数把握することで先頭モジュールを把握し，Control 及び Route に相当する機能や業務単位を全数把握している.

なお，Route の単位について，我々は試行錯誤の結果，画面

脚注

※9 ボトムアップ：具体的な実装及びシステムオペレーションを基に抽象化を進める方式を指す．一方トップダウンとは，汎用的なフレームワークを具体化しながらシステムに適用する方法であると認識している.
※10 RSU：Route，Stage，Unit の一連の各クラスを示す.
※11 F/W：共通フレームワーク．画面や内部・外部システムとアプリケーションソフトウェアの間で制御とデータを仲介する役割を持つ [JP 2007-86828 A].
※12 CRSU：Control，Route，Stage，Unit の一連のクラスを示す.
※13 弊社では COBOL 言語ソースコード中の「モジュール」単位に CRC カードを切り出した.
※14 CRC カード：クラス定義の基礎となる情報（Class-Responsibility-Collaboration）をクラス単位に書き出したもの.

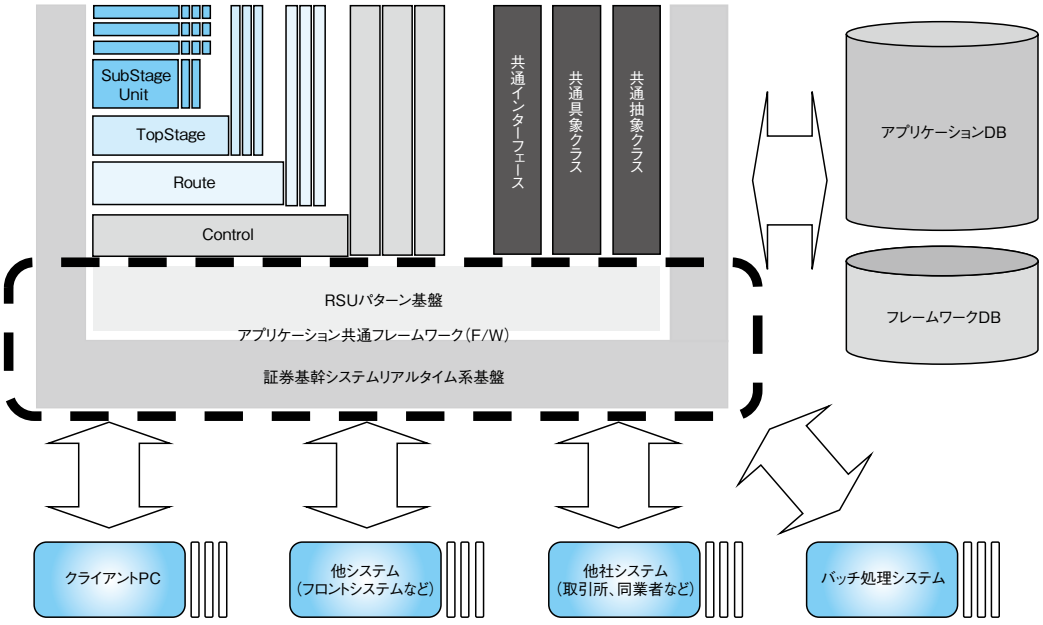


図6　オンラインシステム全体構成図

入力後に画面遷移を伴う 1 トランザクション処理単位として定義することが最適であると判断した．Route 分類の定義付けを中心に進めた後，次に Control 及び Stage の分類を定める．Control の単位は前章で「一連の業務グループ」と述べたが，そのグループの定義方法には定まった法則は無い．旧システムの画面構成や画面上に表示される処理メニュー等を参考に決めることになる．

(2) 実作業の順序

開発側業務精通者は、全業務のうち典型的な基本業務の分析・分類を優先させ、先行作業として実施し，それらの分析結果の確定後，Route 毎に Stage 構成と順序を規約書上に作成した上で図 8 及び表 5 に示すような基本パターンを定義する．規約書において典型基本業務の分類パターンを定めると，他の業務も規約で定められた Route 及び Stage のいずれかの分類と一致する場合が多いため，本体作業は規約中から適合するパターンの選択が中心となる．もし一致しない業務が出現した場合は規約書に当該業務のパターンを追加する．Control は具体的な入力イベントの観点で設定し，具体的な Route と関連付ける．

なお当システムでは，全業務の Route 構造を基に注文登録系，約定登録系，照会系等計 6 種類の基本 Route パターンが導き出されている．

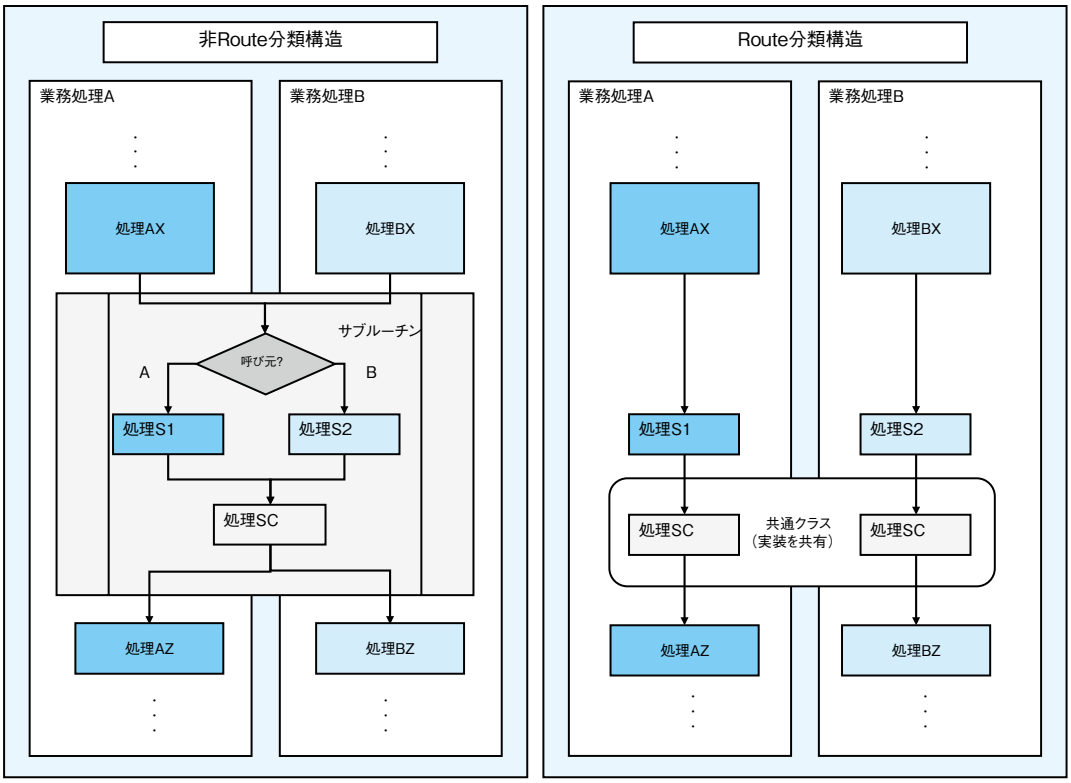


図7　RSU構造と非RSU構造の違い（ただしRoute分類のみの例）

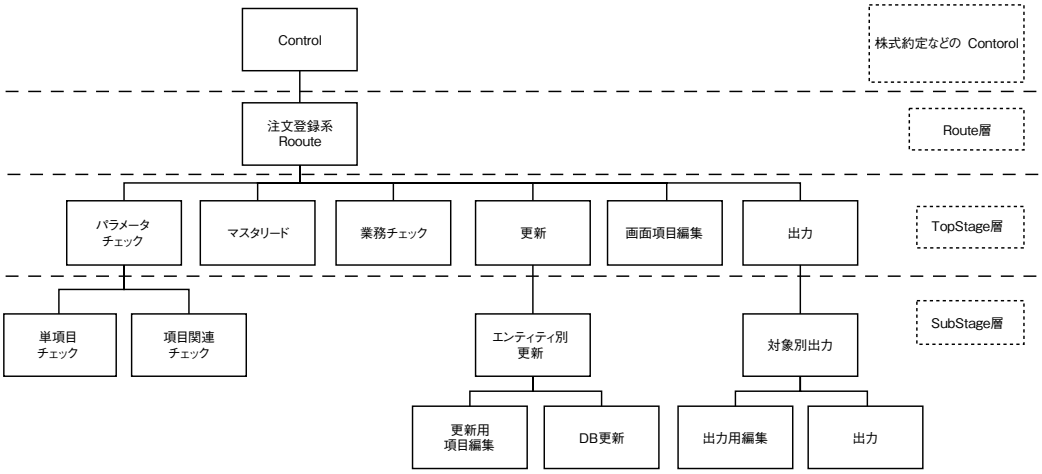


図8　基本パターン構造（注文登録系Routeの例）

(3) 業務ロジック移行トレーサビリティの確保

旧 COBOL プログラムロジックの移行トレーサビリティを確保するためには，旧システムの業務ロジックのモジュール単位に一意の番号を採番し，台帳並びに移行後 Java プログラムコードのコメントに記述し，その番号を引き継ぐ．そうすることでモジュール番号の一覧管理が可能となり，業務ロジックの移行トレーサビリティを記録し管理を行った．

(4) 項目辞書の活用

CRSU の要素と構成で一つの業務を表現するためには，それぞれのクラスにその分類上の役割と業務上の意味を明確に表現する名称を持たせる必要がある．そのため項目辞書によってオペレーション名称，商品名称等の業務に関わる名称の他，同音異義語，異音同義語の明確な分別及び典型的な名称を規定する．

クラスを分類する CRSU 観点は，クラスのステレオタイプと見なすことが出来，その役割を一目で判断出来る．その上，業務に由来する名称を付けることと項目辞書の活用により，業務精通者による可読性が格段に向上し，品質の向上に資することになる．

6.3 経営から許容される現実的なコスト，スケジュールの実現

(1) 開発期間の短縮

概要設計フェーズにおける旧システム業務要件の把握作業に際しては，当適用技法を用いることにより業務部門社員の参画が不要となり，開発側業務精通者の 10 ～ 20 名が先行分析調査を実施することで残りの本体作業はほぼ機械的作業とすることが出来る．

また概要設計においては，複雑な業務ルールの詳細を定義する必要はなく，図 8 に示した基本パターン構造を文書化した後，旧システムの移行対象の全業務に対して，具体的な Control，Route 及び Stage を順次定義する．ただし，成果物文書は UML クラス図である必要はなく※15，Route と Stage の名称と関連及び順序が記述された「表」で良い（図 5）．

これらの施策により概要設計フェーズの工数を通常オブジェクト指向開発における見積工数よりも大幅に削減し，期間の短縮を実現出来る．

概要設計フェーズで定義した Control，Route 及び Stage は，そのまま実装モデルに引き継がれる．基本設計では更に Unit レベルまでのクラス構造を定義し，詳細な業務要件を定義する．業務ルールは個別に定義し，具体的設計は Unit クラスに対して行う．

表5　TopStageの名称と役割の定義例

ステージ和名	ステージ英名	内　容	例
パラメータチェックステージ	ParameterCheck Stage	入力データの妥当性チェックを行う.	単項目チェック，相関（項目関連）チェック
マスタリードステージ	MasterRead Stage	参照すべきマスタ情報を読み込む.	銘柄マスタ読込み，顧客マスタ読込み
業務チェックステージ	GyoumuCheck Stage	業務上のチェックを行う.	業務チェック
更新ステージ	KousinStage	要求に対応する処理を行う.	DB に対する挿入・更新・削除
画面項目編集ステージ	GamenKoumoku HensyuStage	出力すべき画面項目データの編集を行う.	出力データ編集
出力ステージ	Syuturyoku Stage	伝票や他システムへの通知送信を行う.	営業店向け伝票，市場向け伝票取引所向け出力

(2) 下流工程での要件追加効率の向上

詳細設計工程以降で新たな Route，Stage または Unit が発見された場合やユーザから新規要件が追加された場合でも，その新規要件を CRSU 観点で分類・適用することで設計済み部分への影響範囲が明瞭になる．また RSU 層の各クラスは，同じ階層内でそれぞれ互いに独立しており，クラス追加時に他の設計済みのクラスに与える影響は限定的である．修正の場合も，削除・追加の手順を踏むことで追加の場合と同様になる．従って下流工程における業務要件追加及び修正に伴う影響調査及び設計・製造コストは，上流工程から要件を取り込む場合と比べて大きな差は出ない．構造が乱雑になり可読性が低下してしまった場合は，リファクタリング [FOWLER2000] を実施する．

(3) レビュー効率の向上

旧オンラインシステム業務を引き継ぐ部分の設計の大部分はシステム開発者のみの作業で実現出来るため，業務精通者の概要設計フェーズ後半以降の作業はレビューが中心となる．

設計成果物は設計情報が CRSU 観点で分類されているため，業務の目的と全体の処理構成・順序の表形式記述が可能で，更に各 Unit で定義する設計情報は機能毎に独立している個別ルールを記述する．これらの方法で記述することで，システム設計に関する知識を持たない者でも，業務知識さえあれば設計情報を容易に把握することが出来る．そしてレビュアに要求される設計書内容把握に要する負荷が小さいため，レビュー時間を短縮出来る上にレビューの質も高くなる．

6.4 保守担当者の立場に立った保守性

(1) RSU パターン構造の効果

保守要件を新規開発と全く同じルールを用いて CRSU 観点で分類することで，その要件を適用する Control 及び Route が明らかになり，影響する Stage 及び Unit を特定することが基本的に可能となる．要件がエンティティや項目データに関するものである場合は，既存 Stage 及び Unit に関して影響範囲を限定することが可能となる．

また RSU パターン構造自体は業務分野に非依存であるため，保守担当者が RSU パターン構造を習得していることが一定の保守設計品質を維持する上での十分条件となり，次に挙げる効

脚注

※15　当開発プロジェクトではクラス図を必須の設計成果物として求めていない．むしろ不要な成果物であると位置付けている．

果を期待出来る．

① 新オンラインシステムのアプリケーションプログラム構築が同一ルールに則ってCRSU観点で分類された階層構造になっているため，保守担当者が業務スキルの不足している業務分野を担当しても，プログラム構造の維持及び影響個所の特定や限定作業に対し一定以上の品質を確保することが出来る．

② 保守においても新規開発と同じCRSU観点で分類ルールを適用するため，保守作業を繰り返すことによる設計構造の劣化が起こり難い．

③ 保守担当者はパターン規約書を参照することで，RSUパターン構造を容易に理解及び実践することが出来る．

(2) 人的能力差の影響回避

保守要件をRoute，Stageで分類した結果は，原則はパターン規約書に定めたいずれかのパターンに一致する．例外が現れたときは有識者委員会で検討し，必要とされる新型パターンを追加する．同一システム内の全アプリケーションソフトウェアにRSUパターンを適用するため，開発者の個性やスキルの違いによって生じる設計構造の差を小さく限定することが出来る．

(3) RSUパターンの規定者

新オンラインシステムのアプリケーションプログラム構造に適用するRSUパターンを規定する作業を，保守開発担当者が実施することで保守生産性及び保守品質が向上する．新規開発のみ担当する設計者がRSUパターンの規定作業に加わることは可能な限り避けるべきであると考え，当プロジェクトの場合は，開発から参画し保守も担当する者でパターン規約書を作成した．

7 当適用技法導入の成果評価

7.1 新オンラインシステムのJavaコード規模

新オンラインシステムにおけるJava言語アプリケーションのクラス数等の規模を表6に示す．

当システムのJavaクラスは目的と機能の観点で大きく3種類に分けられる．全ステップ数の約半数を占めるデータオブジェクトクラスの内容は，そのほとんどが項目データ及びデータ入出力用メソッドの定義である．これらは大半がエンティティ設計書を元に機械的に生成される．業務要件を実現する中核の業務ロジックはStage及びUnitクラスに実装されている．そしてControlクラス，Javaインターフェース，抽象クラス等がその他のクラスとして作られている．

業務ロジックを担うStage及びUnitクラスのステップ数に関しては，旧COBOLプログラム約300万ステップに対し約280万ステップと少し下回る程度である．しかしJavadoc等のコメント行を2割程度含んでいる点や，Java言語に限ったステップの冗長性を加味すると事実上のコード量は実測値より少なく，既存ロジックの踏襲に加えてコードの軽量化を図ることも出来ていると考える．

表6　新オンラインシステムにおけるJava言語アプリケーション規模

	業務ロジックを担うStage，Unitクラス	データオブジェクトクラス	その他（抽象クラス，インターフェース，Controlなど）
クラス数（単位K）	28	15	12
メソッド数（単位K）	120	800	190
ステップ数（単位K）	2,800	4,400	1,400

7.2 目標ごとの成果評価

(1) 旧システム業務要件の新システム業務要件への取り込み

旧システム業務要件のうち新システムへの取り込みが要求されたものは、100%の取り込みを実現しており，社内外の他システムとの連携も完全な再現を実現している．

(2) 約2年半で開発を終了

当システムの着手から終了まで要した約2年半という期間の内訳は，今述べている適用開発方法論とデザインパターンの開発に約4ヶ月，それらの当システムの開発担当者並びに関係者への全面展開に約2週間，要件定義及び概要設計に約4ヶ月，設計と実装に約9ヶ月，そして残りの約12ヶ月がテストとなっている．この期間での開発はこれまで述べてきたように，当技法を適用して初めて達成出来たものであると我々自身は評価している．

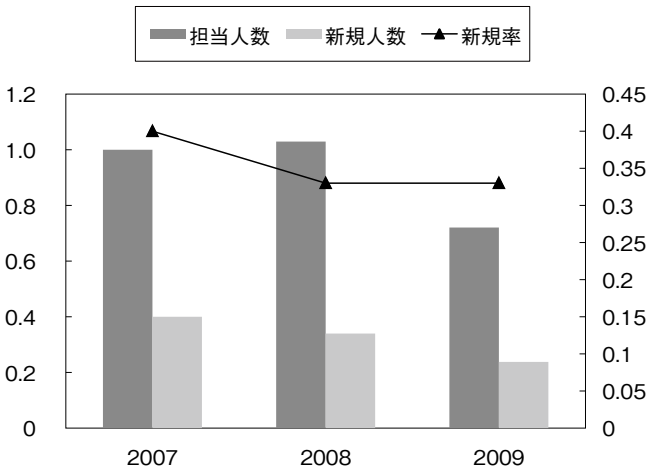


図9　オンライン保守担当者と新任担当者数の変化

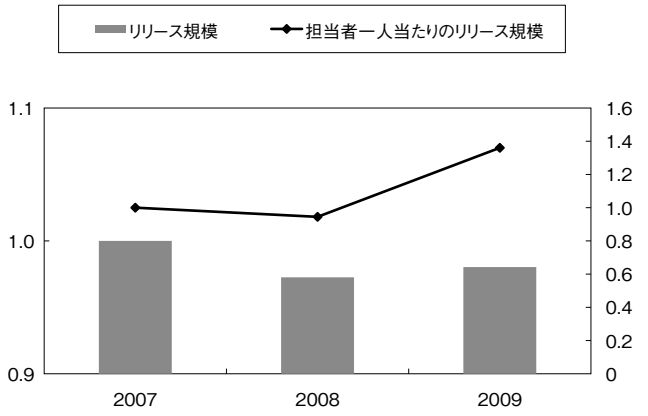


図10 保守開発リリース件数と担当者一人当たりのリリース規模の変化

(3) プログラム構造の刷新

旧 COBOL ソースコードのうち約 95% については RSU パターン適用に伴い全面的にロジックを再構築できたが，反省として，残り約 5% の旧 COBOL サブルーチンを CRSU 観点での分解・分類及び再構築が不十分なまま Unit として引き継いでしまった．主な原因は，一部の大規模サブルーチンに対して，難易度及び規模把握に不完全な部分があり，要員計画上の人員不足が発生したことによる．

旧 COBOL ソースコードに対して RSU パターンを適用して新規に設計・コーディングする作業は，当プロジェクトの後半においては設計作業手順が定型化されてきたため，多くのモジュールの移行が機械的作業で進められる状況であったが，それでも，もともとのプログラムの構造が複雑で理解の難易度が高いモジュールについては，相応の開発要員の配置が必要であったと改めて認識するに至った．

(4) 高い保守性の確保

RSU パターンの適用によりロジックの錯綜が非常に少なく，かつ保守要件から保守個所の特定が容易な構造とすることが出来た．保守性を示す指標として，図 9 に 2007 年度を 1 とした最近 3 年間の保守担当者数の変化を，図 10 に同 3 年間の一人当たりのリリース規模を示す．

ここ 3 年間では毎年の担当者のうち約 3 ～ 4 割が新任担当者である．保守担当者数は削減傾向にあるが，一人当たりのリリース規模について 2009 年は 2008 年の 1.4 倍に増加している．新任の保守担当者でも一人当たりのリリース規模増加に十分に対応出来ており，運用開始から 5 年以上経過した現在でも保守性の高い構造を維持出来ていると評価している．

8 まとめ

基幹業務システムオンライン処理の再構築に際し，当技法適用の最大の効果は以下の 2 点に絞られる．

① 設計書が不十分でも開発担当者だけでプログラムから業務要件を把握出来ること．

② 錯綜化したプログラム構造を刷新出来ること．

通常の技法では業務の発見と分析を行いモデル化によって要件を定義する．この際，汎用フレームワークを用いる場合も多い．しかし我々が適用した技法では，実際に稼動しているプログラムソースコードを分類することでシステム全体の要件を定義し，基本設計を行う．すなわち，プログラムソースコードの分類が，通常技法の要件定義から基本設計に相当していることになる．我々は前者をトップダウン方式，後者をボトムアップ方式と認識している．分類の観点は業務の「目的（Route）」及び処理の「段階（Stage）」を用いるところがポイントである．

謝辞

当開発方法論並びにデザインパターンの開発，及び適用に携わった方々に感謝する．

参考文献

[LARMAN2003] Larman，Craig：実践 UML 第 2 版 パターンによる統一プロセスガイド，依田光江訳，ピアソン・エデュケーション，2003

[FOWLER2000] Fowler，Martin：リファクタリング プログラミングの体質改善テクニック，児玉公信ほか訳，ピアソン・エデュケーション，2000

参考 特許情報

[JP 2007-86828 A] 特許番号 特許第 4487891 号

非機能要求グレード解説
～システム基盤の非機能要求の決め方～

SEC エンタプライズ系プロジェクト
研究員
柏木　雅之

ここでは、非機能要求を適切に決めることの難しさを説明し、システム基盤の非機能要求を段階的に漏れなく決める手法である非機能要求グレードについて解説する。また、非機能要求グレードをどのように使うかについても事例を通して紹介する。

1　非機能要求の難しさ

SEC では、ソフトウェアやシステム開発における「超上流」工程の重要性を訴えてきている [SEC BOOKS]。しかし、「超上流」工程の要求定義作業では、「非機能要求」の品質確保が難しいのが現実である。以下では、まず、どのような難しさがあるか、課題を明らかにする。

1.1　機能要求と非機能要求

要求は、業務のステークホルダ間での処理の流れや扱うデータ等に関する「機能要求」（業務要求とも呼ばれる）と、それをシステムで実現するための信頼性、性能、セキュリティ等の「非機能要求」から成る。「機能要求」は、利用者の業務に直結するため、利用者が決めることが出来る。一方、「非機能要求」は、稼働率や性能、あるいはセキュリティ等、システム構築の専門知識が必要であるため、利用者が決めるのが困難であったり、項目が漏れたりする。では、開発者が「非機能要求」を決めることが出来るかというと、体系的に整理されたものがないのと、業務からの要求へのレベル感が分からないため、過去の似たシステムを参考に決める等という手法を使うことが多く、その妥当性の検証が困難であるのが実情である。

1.2　システム基盤と業務アプリケーション

要求の対象という観点では、「非機能要求」は、主にハードウェア、OS、ミドルウェア等から成る「システム基盤」に対する要求である。一方、「機能要求」は、主に業務アプリケーションに対する要求から成る。しかし、機能要求と非機能要求の境界があいまいなため、定義があいまいになりがちである。

なお、「非機能要求」にも、業務アプリケーションに対する要求である「操作性（ユーザビリティ）」、「狭義のソフトウェアの品質」、「ソフトウェアの変更のしやすさ」等といったものもある。

1.3　どう書かれているか

一般に、RFP※1 や要件定義書に記載されている「要求」は、「機能要求」が中心であることが多い。「非機能要求」は、一個所にまとめられておらず、「機能要求」の記述の様々な場所に散りばめられている。そのため、場合によっては、矛盾や優先順位のあいまいさが見られる。

1.4　発生する問題

「非機能要求」の定義漏れや、利用者と開発者間の認識違いは、稼働直前や稼働後のトラブルの原因となる。これらのトラブルでは、機器の追加や変更を伴う設計変更によるコストの増加や稼働時期の遅延を伴うことが多い。場合によっては、個人情報の漏洩や長期間のシステム停止等の事業の継続性を危うくする重大なトラブルが発生することもある。

2　非機能要求グレードによる解決

「非機能要求グレード」は、企業内の情報システムのシステム基盤に対して、要件定義工程での非機能要求の上記課題を解決するために考案された「非機能要求」を決めるための手法である。国内の SI ベンダ 6 社からなる「システム基盤の発注者要求を見える化する非機能要求グレード検討会」※2 によって作成され、パブリックコメントの公募や経済産業省での実証実験 [METI HP] などを経て、2010 年 2 月に最終版が公開された。一層の普及を目指し、同年 4 月に SEC に移管された [非機能要求グレード HP]。

2.1　非機能要求グレードの分類体系

「非機能要求グレード」は、「非機能要求」を整理体系化したものであり、大きく 6 つの大項目がある（表 1）。更に、中項目、小項目、メトリクスと細分化されており、合計で 236 のメトリクス※3 から成る。

2.2　非機能要求グレードのコンテンツ

非機能要求グレードは次の 6 つのコンテンツから成る。

・利用ガイド（解説編）
　非機能要求グレードの作成の背景や適用範囲、各コンテンツの説明や用語集等から成る。
・利用ガイド（利用編）
　非機能要求グレードの利用方法を説明。
・項目一覧

非機能要求を分類整理し、メトリクスに対し最大6段階（レベル0～5）のレベルを設定し、それを解説。

・グレード表

3種類のモデルシステムと、それぞれに対し重要な92のメトリクスに対する推奨レベル（ベース値）とレベル調整の仕方を説明。

・樹系図

項目一覧を6つの大項目ごとに視覚的、階層的に示す。決定を優先すべき順に並んでいる。

・活用シート

項目一覧とグレード表をマージし、スプレッドシート形式で提供。

2.3 非機能要求グレードの利用手順

次に、非機能要求グレードの使い方を説明する。非機能要求を段階的に決める手順は、次の3ステップから成る（図1）。

① モデルシステムの選定

まず、グレード表のモデルシステムシートを使い、16個の特徴を比較して次の3つのモデルシステム※4から開発対象のシステムに最も似ているシステムを選択する。

Ⅰ．社会的影響度がほとんど無いシステム

Ⅱ．社会的影響度が限定されるシステム

Ⅲ．社会的影響が極めて大きいシステム

それぞれ、部門の小規模システム、企業の基幹系システム、社会インフラシステム等に相当する。

② 重要項目のレベル決定

重要項目とは、非機能要求を検討する上で品質やコストに大きな影響を与え得る項目のことであり、計92項目ある。グレード表は、重要項目だけを記載し、3つのモデルシステムそれぞれに対する推奨レベルを規定している。選択したモデルシステムの推奨レベルを確認し、目的のシステムと差があれば、調整する。決める順番は、樹系図を参考にする。

③ 重要項目以外のレベル決定

項目一覧を使い、重要項目以外のレベルを決定する。具体的には、項目一覧の重要項目以外のメトリクス毎に提示されている0レベルから最大5レベルまでの中から適切なレベルを決める。その際、運用時のコストやメトリクス間のトレードオフ等に留意する。

例えば、バックアップの自動化の要求のレベルを上げるとバックアップソフトが必要で導入時のコストは高くなるが、運用時は人手が不要になりコストが下がる。決められた予算で、要求されたオンライン性能とデータの暗号の両立が困難な場合、どちらを優先するか調整したり、予算増額を検討したりするケースもある。

3 非機能要求グレードの活用事例

非機能要求グレードは、前述のように、開発対象のシステムの非機能要求を系統的に効率良く漏れなく定義するための手法である。要求定義での本来の使用方法に加え、様々な活用方法がある。以下に、それらの事例を幾つか紹介する。

3.1 既存の社内システム診断

既存の社内システムのシステム基盤は、過去の経緯から統一

脚注

※1 RFP：Request For Proposal, 提案依頼書

※2 株式会社NTTデータ、富士通株式会社、日本電気株式会社、株式会社日立製作所、三菱電機インフォメーションシステムズ株式会社、沖電気工業株式会社の6社からなる。非機能要求グレードの完成を受けて、2010年3月末に解散。

※3 重複しているメトリクスがあるので種類としての数はこれよりも少ない。

※4 重要インフラ情報システム信頼性研究会報告書 [SEC HP] のType Ⅰ、Ⅱ、Ⅲに該当。

表1 非機能要求グレードの分類

要求の分類	説明	要求例
可用性	システムサービスを継続的に利用可能とするための要求	・運用スケジュール（稼働時間・停止予定等） ・障害、災害時における稼動目標
性能・拡張性	システムの性能、及び将来のシステム拡張に関する要求	・業務量及び今後の増加見積り ・システム化対象業務の処理傾向（ピーク時、通常時、縮退時など）
運用・保守性	システムの運用と保守のサービスに関する要求	・運用中に求められるシステム稼動レベル ・問題発生時の対応レベル
移行性	現行システム資産の移行に関する要求	・新システムへの移行期間及び移行方法 ・移行対象資産の種類及び移行量
セキュリティ	情報システムの安全性の確保に関する要求	・利用制限 ・不正アクセスの防止
システム環境・エコロジー	システムの設置環境やエコロジーに関する要求	・耐震／免震、重量／空間、温度／湿度、騒音等、システム環境に関する事項 ・CO_2 排出量や消費エネルギーに関する事項

されていないことが多い。そのため、現状がどうなっているか把握することが重要である。非機能要求グレードを使って、既存の社内システムに対する非機能要求項目のレベルが過大や過少となっていないかを調査する。これにより、IT リスクや過剰な設備等が把握出来る。

3.2 社内グレードの策定

非機能要求グレードは役に立つが、システム毎に 236 ものメトリクスを決めるのが大変という組織に対しては、非機能要求を次の 3 つの要求グループに分けて効率良く決めることを推奨する。

① すべてのシステムに共通な非機能要求グループ
② 対外向けシステム、基幹システム、部門システムなどの複数のランクからどれかを選ぶ要求グループ
③ システムごとに個別に決める要求グループ

最初のグループ分けと推奨値の決定に工数を要するが、一度決めれば、個別のシステムごとに決める項目を削減出来る。これは、ユーザ企業だけでなく、SI ベンダの提案パターンにも適用出来る。

3.3 RFI※5 や RFP 作成時の利用

RFI や RFP に非機能要求の項目一覧を添付することが出来る。前述のように、RFP などでは非機能要求は分散して記述されていることが多いので、一個所にまとめて記述することにより、非機能要求の漏れや矛盾を防ぐことが出来、結果的に情報システムの品質向上が期待出来る。あるいは、コストの割に要求のレベルが高過ぎる等の意見が出やすくなり、適切な要求レベルにすることが出来る。

4 課題と今後の予定

非機能要求グレードは、企業の情報システムのシステム基盤の企画や設計に従事する人向けに記述している。そのため、業務部門や CIO※6 等の経営層に理解出来る用語を使い、事業リスクやコストの観点からの訴求が出来ていない。IPA/SEC は、2010 年度にこれを改善するため活動を行った。

他にも、次のような要望が寄せられている。

・メトリクス数が 236 と多過ぎる
・モデルシステムの数が少ない
・業務アプリケーションの非機能要求に対応してほしい
・組込みシステムの非機能要求に対応してほしい

SEC としては、今後これらの課題に対し改善を検討していく予定である。

5 結び

非機能要求グレードは、決めることが難しい非機能要求を、段階的に誤解なく漏れなく決める手段を提供している。これを利用することによって、要求定義の品質向上が期待出来る。

非機能要求グレードの IPA/SEC からの公開から約 1 年経過し、様々な情報システムの開発現場での非機能要求グレードの利用事例が増えている。読者の方々にもぜひ活用していただき、利用した効果や要望等を IPA/SEC に寄せていただきたい。

最後に、非機能要求グレードを作成した、「システム基盤の発注者要求を見える化する非機能要求グレード検討会」の方々に感謝を述べて結びとしたい。

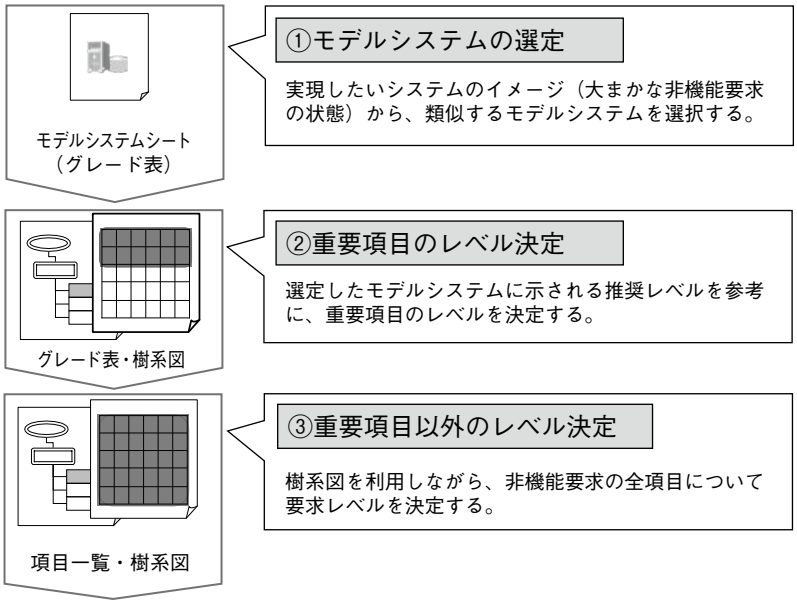


図1 非機能要求グレードを使った段階的な決定手順

脚注

※5 RFI：Request For Information，情報提供依頼書
※6 CIO：Chief Information Officer，最高情報責任者

参考文献

[METI HP] 非機能要求グレード評価委員会実証評価報告書，http://www.meti.go.jp/policy/it_policy/softseibi/grade.pdf
[SEC BOOKS] IPA/SEC 編：SEC BOOKS 経営者が参画する要求品質の確保 ～超上流から攻める IT 化の勘どころ～ 第 2 版，オーム社，2006
[SEC HP] 重要インフラ情報システム信頼性研究会報告書，http://sec.ipa.go.jp/reports/20090409.html
[非機能要求グレード HP]
日本語版：http://sec.ipa.go.jp/reports/20100416.html
英語版：http://www.ipa.go.jp/english/sec/reports/20101222.html

構造化日本語仕様書としてのVDM仕様

株式会社 CSK IT ソリューション社
産業システム事業本部　製造システム事業部　第一開発課
佐原　伸

形式手法は、証明やモデル検査等の高度な検証に威力を発揮するものではあるが、残念ながら使うのが難しいと考えている方も多い。最近筆者らは、形式手法に取り組み、形式仕様記述言語であるVDM++ [フィッツジェラルド2010]により記述した仕様を「構造化日本語仕様」と捉えた上で、形式手法の基礎的な技術を使用し、かつ過去に有用であることが実証されている回帰テスト等のソフトウェアエンジニアリング技術も活用したところ、大きな成果を得た。本稿では、その概要を紹介する。形式手法の可能性の一端に触れていただければ幸いである。

1 ソフトウェア開発プロジェクトの問題点

現在のソフトウェア開発プロジェクトを現場の視点から見ると、以下の大きな問題点がある。

(1) ソフトウェアエンジニアリング技術の欠如

基本的な問題は、ソフトウェアエンジニアリング技術が使われない結果、開発現場の技術者は大変な労力を無駄にしてしまっているということである。

その結果、ソフトウェアの開発という本質的な仕事ではなく、「混乱を収拾する」という二義的な仕事に労力の大半が割かれてしまっている。

(2) 構成管理・版管理の欠如

開発現場でコンサルティングをしていると、最初に直面する問題は「どのファイルが最新の仕様か？」が必ずしも明確ではないことである。これでは、仕様のレビューも正確には出来ない。

(3) 要求仕様の欠如

次の問題は、システムの要求仕様がほとんど存在しないことである。要件定義という名の文書が存在することが多いが、システムの計画・要求仕様・設計仕様・実装時の制約等、本来異なる工程で、順次記述し、検証していかなければならない事項が「すべて不完全な形」で記述されている。

結果として、次工程である設計工程においては、この要件定義の文書だけでは設計仕様を決めることが出来ないため、要求仕様作成・検証工程と同じ思考・議論が繰り返されることになる。

① 日本語仕様書の問題点

要求仕様に日本語を使うことで、不完全であいまいな仕様になり、大きな問題が発生する。

②不完全な仕様

日本語仕様では、容易に、不完全な仕様を作成することが出来る。

プログラムで言えば、構文チェックや型チェック、あるいはテスト実行によるチェック等をせず、すべてレビューによって検証するのであるから、ある程度以上のシステムでは、仕様には多くの欠陥が残ることになる。

とくに、仕様の修正が多いシステムでは、修正した部分の変更余波をレビューだけで担保することは不可能に近く、結果として、欠陥が後工程に非常に大きな悪影響を与える。

③あいまいな記述、要求辞書の欠如

日本語仕様に限らず、英語等の自然言語による仕様は、あいまいさを必ず含む。レビューによって、このあいまいさを完全に検出することはほぼ不可能である。

例えば日本語仕様書で使われている「同じ駅」という述語に、3つの異なる意味があったことがあった（後述する駅務システム）。

とくに問題なのは、システムに内在する用語のあいまいさである。仕様書に使われている顧客企業内用語について確認を求めると、顧客の側で議論が巻き起こることも多い。「え、それってそんな意味ですか。私は、こう思っていたのですが…」と始まり、延々と議論が続いて、時間が無駄に浪費されていってしまう。

これは、それらの用語を記述した要求辞書[※1]が存在しないことに原因がある。

2　形式仕様記述言語VDM++を使った解決法

筆者らは、上記の問題を解消するために、あいまいな日本語仕様に代わる「構造化日本語仕様」[佐原] として形式仕様記述言語 VDM++ を使用した。

表 1　形式仕様記述言語 VDM++ を使ったソフトウェア開発あるいは実験の成果

項目＼会社名	株式会社 CSK	フェリカネットワークス株式会社	独立行政法人 産業技術総合研究所 オムロン株式会社	備考
対象システム	証券業務パッケージ・ソフトウェア	おサイフケータイファームウェア	鉄道駅務システム	
適用工程	UseCase レベルの要求仕様記述と検証	API 外部仕様の記述と検証	既存システムの厳密仕様作成実験	
仕様規模	30,000 行	74,000 行	7,500 行	注釈を除く行数
回帰テストケース規模	30,000 行	66,000 行	800,000 行	注釈を除く行数
リリース後欠陥数	0 件	0 件	29 件（プログラムには欠陥無し）	
生産性（COCOMO 見積との比）	2.5 倍	2 倍		
開発期間（COCOMO 見積との比）	45%	85%		
開発チームの特徴	業務経験無し、形式手法知識有り、45 歳以上	業務経験有り、形式手法知識無し、30 歳以下	業務経験有り、形式手法知識有り、30 歳代	

2.1　成果

表 1 に、形式仕様記述言語 VDM++ を使ったソフトウェア開発あるいは実験の成果を示す。

ここで重要なことは、CSK やフェリカネットワークス株式会社におけるシステム開発 [KURITA2008] では、経験も背景も異なる技術者が、全く異なる種類のシステムで、いずれも高信頼性システムを低コストで開発していることである。

両システムとも仕様記述の方法は、ほぼ同じである。作成したライブラリーに多少の違いはあるものの、同じ考え方で階層化を行った仕様フレームワークで、要求仕様と API 外部仕様の記述が出来たのである。

独立行政法人 産業技術総合研究所とオムロン株式会社による鉄道駅務システムは、厳密仕様の作成・検証実験であるが、既にリリースされているシステムの仕様に 29 件の欠陥があることを検出した。このプロジェクトの技術者の経験や背景は前二者とは、全く異なる。

2.2　使用した技術

上記のプロジェクトで使用した技術は以下の通りであり、フェリカネットワークス株式会社とほぼ同じ技術を用いた。

（1）VDMTools による静的検証

VDMTools [VDMTOOLS HP] は、通常のプログラム言語の場合と同じく、構文検査や型検査を行うことが出来る。単純ミスの多くを検出すると共に、証明課題と呼ばれる条件式を、仕様を解釈して生成する機能がある。これらの機能によって、仕様中に存在するかなりの数の欠陥を静的に検出することが出来る。

① 構文検査

構文検査は、VDM++ の文法に違反しないかを、各クラス毎に検査する。

② 型検査

型検査は、クラスを含む型の不整合を、複数クラス間でチェックする。このチェックは、関数や操作のインターフェースのチェックも行うことになるので、仕様の実行テスト前に、かなりの量の欠陥を検出出来る。

③ 生成した証明課題のレビュー

ツールで生成した証明課題は、本来は証明に使うのであるが、開発現場の技術者が証明技術を使いこなすのは難しく、証明課題をレビューすることで代用した。

（2）VDMTools による動的検証

本来の VDM 手法では、作成した仕様を段階的に洗練（refine）しながら、証明付きでプログラムへ変換していく。VDMTools では、作成した仕様を同ツール上で実行し、テストすることを開発現場の技術者に奨励している。

そこで、我々は、回帰テストを用いて仕様を実行し検証した。

① 回帰テスト

VDM++ によるテストケースとそれを仕様実行した結果を比較する回帰テストを行い、変更余波が仕様の他の部分に波及していないかを検証しながら、動的検証を進めていった。こうすることで、ある部分の変更が他に影響しないことを保証出来る。

② コードカバレージ

VDMTools は、仕様実行の C0 レベル・コードカバレージ※2 を計測・表示出来るので、これを使用して回帰テストの質を向上していき、カバレージ 96%以上を達成した。

脚注

※1　要求辞書：昔の用語で言えば用語辞書になるが、リアルタイム構造化分析・設計手法時代に、名詞だけでなく述語も定義すべきという主張により要求仕様作成工程や設計工程の用語辞書に、要求辞書という単語を当てるようになった。

※2　命令レベルのコードカバレージなので、全組み合わせを実行したことを保証出来るわけではないが、実用上、かなり有効なことが分かっている。

(3) 手作業による証明

実装が難しいと判断した2関数についてのみ、筆者が、手作業によって段階的洗練による証明を行った。

(4) 構成管理・版管理

システム開発に使用した仕様、文書、プログラムは、すべて構成管理・版管理システムを使用して、いつ、誰が、なぜ修正したかを記録しつつ、管理していった。

(5) レビュー

レビューは、以下の2レベルで行った。

①ドメイン専門家による妥当性検査

回帰テストのテストケースとテスト結果について、システムの対象領域（ドメイン）の専門家にレビューを仰ぐことで、VDM++ の知識が無いドメイン専門家による妥当性検査を行った。

②プログラマによる検証

作成した VDM++ 仕様についても、同様にプログラマにレビューを仰ぐことで、回帰テストによる仕様検証を補完した。もちろん、このレビューでは、仕様の正しさだけでなく、仕様の読みやすさ、再利用性、保守性等の検査を行った。

3 構造化日本語仕様としてのVDM++仕様

VDMTools は、当初、日本語識別子が使えなかったが、構造化仕様としての書きやすさや読みやすさを強化するため、国際語化を行った。日本語仕様の良さを生かしながら、厳密性と検証の容易さを追求したわけである。

3.1 階層構造による仕様記述の役割分担

VDM++ 仕様の記述を容易にし、保守性や再利用性を向上させるため、仕様記述の階層化を行った。

(1) 要求辞書階層とアプリケーション階層の分離

表2は、一部を簡略化してはいるが、仕様の各階層と充足すべき対象を示したものである。本稿では、階層のすべてについて解説する余裕は無いが、「運賃を求める」システム※3のアプリケーション階層と要求辞書階層の一部を解説する。

表2　仕様の各階層と充足すべき対象

階層	充足する対象
テストケース	回帰テスト
シナリオ	事後条件（事後条件が複雑な場合）
アプリケーション（業務論理）	業務アプリケーション、業務論理
要求辞書	業務・ドメインの名詞と述語定義
ユーティリティ	共通機能

(2) アプリケーション仕様としての日本語 VDM++ 仕様

図1は、アプリケーション階層の仕様の一部である。

この階層の仕様は、通常のプログラミング知識があれば、1～2日程度の教育で履修出来る VDM++ の文法しか使用していないので、通常のアプリケーション仕様の記述力を持つ者であれば容易に記述出来る。

アプリケーション階層は、要求辞書階層の用語を使って仕様を記述するだけなので、運賃計算等のドメイン知識は、1階層下の要求辞書階層に書くからである。

例えば、図1の例は、要求辞書階層の仕様の用語「運賃」「運賃表」「距離に比例した運賃を得る」を使用して運賃を得るという単純なものである。

(3) 要求辞書としての日本語 VDM++ 仕様

図2は、上位のアプリケーション階層仕様から呼ばれた要求辞書階層の一つのクラスの仕様例である。「運賃」「運賃表」という名詞と、「距離に比例した運賃を得る」という述語が、アプリケーション階層で使われている。

ここでは、名詞の二つは型として定義され、述語は関数として定義されている。

「運賃表」型には、運賃表としての制約を記述した不変条件が記述されている。距離が少ない方から大きい方へ、隙間無く運賃が定義されていることが要求されているのである。日本語では、解釈があいまいになってしまうが、VDM++ ソースの方は、構文の説明を5分ほど受けることにより、10分程度読み込むことでかなり正確に制約条件を理解出来るようになる。

「距離に比例した運賃を得る」関数は、pre で表す事前条件と、post で表す事後条件を持ち、事前条件は運賃が唯一つに決まることを要求している。

この階層の仕様は、やや VDM 系統の仕様記述言語に特有な文法があり得ることや、VDM++ 仕様を実行して検証するための構造も出てくるので、4日程度の教育が必要で、VDM++ 中級技術者が必要になる。ただし、このレベルの技術者は、1システムにつき、1人から数人程度でよいので、養成はさほど難しくはない。

```
-- 仕様１：アプリケーション階層の仕様例
-- 運賃を得るユースケース・レベルの要求仕様である。
class 運賃を得る is subclass of 運賃表辞書

instance variables
public ｓ運賃表 : 運賃表 := [];
...

public 適用する : 路線網`距離 ==> 運賃
適用する（ａ距離） ==
        return 距離に比例した運賃を得る（ｓ運賃表，ａ距離）;

end 運賃を得る
```

図1　仕様1

```
-- 仕様２：要求辞書階層の仕様例
-- 運賃表の用語（名詞と述語）を定義する要求辞書である。
class 運賃表辞書

types
public 運賃 = nat;
public 行 ::
        f下限 : 路線網`距離
        f上限 : 路線網`距離
        f運賃 :- 運賃;

-- 運賃表は、下記の不変条件を満たさなければならない。
public 運賃表 = seq of 行
inv w運賃表 ==
        forall i, j in set inds w運賃表 &
                下限より上限が大きい(w運賃表(i).f下限, w運賃表(i).f上限) and
                j = i + 1 => 上限と次の行の下限は等しい(w運賃表(i).f上限, w運賃表(j).f下限);

functions
static public 距離に比例した運賃を得る : 運賃表 * 路線網`距離  -> 運賃
距離に比例した運賃を得る(a運賃表, a距離) ==
        let n = 運賃表の何番目かを得る(a運賃表, a距離) in
        a運賃表(n).f運賃
pre
        運賃表にただ一つ存在する(a運賃表, a距離)
post
        let n = 運賃表の何番目かを得る(a運賃表, a距離) in
        RESULT = a運賃表(n).f運賃;

static public 運賃表のある行に存在する : 行 * 路線網`距離 +> bool
運賃表のある行に存在する(a行, a距離) ==
        a行.f下限 <= a距離 and a距離 < a行.f上限;

static public 運賃表にただ一つ存在する : 運賃表 * 路線網`距離 -> bool
運賃表にただ一つ存在する(a運賃表, a距離) ==
        exists1 i in set inds a運賃表 & 運賃表のある行に存在する(a運賃表(i), a距離);

static public 運賃表の何番目かを得る : 運賃表 * 路線網`距離 -> nat1
運賃表の何番目かを得る(a運賃表, a距離) ==
        let i in set inds a運賃表 be st 運賃表のある行に存在する(a運賃表(i), a距離) in i;

static public 下限より上限が大きい : 運賃 * 運賃 -> bool
下限より上限が大きい(a下限, a上限) == a下限 < a上限;

static public 上限と次の行の下限は等しい : 運賃 * 運賃 -> bool
上限と次の行の下限は等しい(a下限, a上限) == a下限 = a上限;

end 運賃表辞書
```

図２　仕様２

4 VDM仕様適用の効果

形式手法適用プロジェクトの成功要因について、効果があったと思われる順に紹介する。

なお、効果①、②、③、⑤は、厳密な仕様記述を用いたことによるものであり、形式手法はそのような厳密な仕様記述言語を提供したという役割しか果たしていない。

効果①　厳密な仕様による単純ミスの激減

厳密な仕様記述により、静的チェックによって単純ミスが激減し、仕様作成者は仕様の意味に集中出来た。

効果②　あいまい性の排除による意思疎通の向上

脚注

※３　実仕様は差し障りがあるので、教育に使用している例題で説明する。

要求仕様ソースそのものが要求辞書となり、開発チームのコミュニケーション・ミスが激減した。

VDM++ による要求辞書は、クラス名や型名といった名詞、関数名や操作名といった述語を含み、従来の名詞のみについて記述した用語辞書より、網羅性が高い。

効果③　回帰テストによる妥当性検査

VDM++ 要求仕様モデルの検証は、回帰テストを作成して、主として妥当性検査を行った。この際、事後条件のチェックも行ったので、通常のプログラミング言語の回帰テストよりも、より質の高い妥当性検査を行うことが出来た。

VDMTools のコード・カバレージ機能を使用することにより、仕様のカバー率を確認することが出来た。また、当該プロジェクトは仕様変更が多発したものの、変更余波の発生を抑えることが出来、効率よく品質を上げていくことが可能となった。

いわば、アジャイル仕様開発に近いことが実現出来たものと言える。

効果④　不変条件・事前条件・事後条件による検証

回帰テストケースは、不変条件、事前条件を検証することにもなり、仕様内部に矛盾が無いことを検証することにも役立だった。

効果⑤　版管理による単純ミスの防止

仕様のソースは、版管理システム cvs を使用して、誰が、いつ、どういう理由で修正したかを記録しつつ、管理を行った。

このため、他のサブシステムでは発生していた、間違った版の仕様を使うといった、修正に関するミスの発生を防ぐことが出来た。

効果⑥　証明課題のレビューによる検証

証明課題のレビューによる検証は、幾つかの欠陥を発見することに役立った。

とくに役立ったのは、見落とした事前条件や事後条件に気がつく機会が多いことである。

効果⑦　証明

二つの関数だけ、VDM 本来の段階的洗練手法を使って、関数の事後条件を段階的洗練の変換規則に基づいて手作業で洗練していき、最終的な手続き的仕様を得ると共に、証明した。このうち一つの関数は、通常の方法では、欠陥無しに実装するのはかなり難しいとの実感を得た。

5 おわりに

5.1　形式手法はソフトウェア工学の最新手法

形式手法は、これまでの開発技術と全く異なる技術だと思われがちであるが、実際には、構造化技術やオブジェクト指向技術の延長線上にある最新技術である。従って、過去の技術をすべて捨てて、形式手法を採用する必要は無く、もちろん全工程に一度に導入する必要も無い。本稿で述べた手法は、ソフトウェア開発の現場で一番大きな問題を起こしていると思われる要求仕様作成・分析工程に「構造化日本語仕様」として VDM++ 仕様を導入し、従来の手法で効果的と思われた技術と共に、形式手法の一番基礎的な技術を使ったものである。

5.2　教育効果

形式手法ツールを使うことは、ソフトウェアの高品質化・生産性向上に役立つだけでなく、開発に携わった技術者の教育にも役立つ。筆者自身、形式手法適用後は、技術者としての能力が格段に向上したと自負している。

また、UML を教えた場合と VDM を教えた場合を比較すると、UML は 10 人中 9 人が落ちこぼれたが、VDM では落ちこぼれが発生しなかった※4。

5.3　まとめ

VDM++ を「構造化日本語仕様」と捉え、活用をしたところ、形式手法の初歩的技術を使うだけで品質と生産性に劇的な効果が生まれ、かつ、技術者の教育にもなった。

今後は、形式手法本来の証明やモデル検査といった検証技術を、開発現場のプロジェクトに導入する方法が課題だと考えている。

VDMTools については、組み合わせテスト、証明ツールとの連携といった研究が欧州で進んでおり、国連大学では、VDM 系統の言語 RAISE-SL で、モデル検査との組み合わせも研究されている [UNU HP]。

また、EU の DESTECS プロジェクトでは、VDM++ に非同期並行処理機能を追加した VDM-RT を作成し、プラントモデルを Simulink 相当のツールである 20-sim で作成し、ソフトウェア制御モデルを VDM-RT で作成することで、システム・アーキテクチャをシミュレーションで評価する試みが昨年から始まっている [DESTECS HP]。

脚注

※4　最初のプロジェクトの VDM 教育は英語で行ったので、2 人落ちこぼれたが、その後の日本語による教育では、落ちこぼれが発生していない。

参考文献

[DESTECS HP] http://www.destecs.org/

[KURITA2008] 2008Taro Kurita, Yasumasa Nakatsugawa:The Application of VDM to the Industrial Development of Firmware for a Smart Card IC Chip, International Journal of Software and Informatics vol. 3. Issue 2-3. June/September 2009, Institute of Software. Chinese Academy of Science(ISCAS).

[UNU HP] http://www.iist.unu.edu/
Lizeth Tapia and Chris George. Model Checking Concurrent RSL with CSPM and FDR2. Research Report 393, UNU-IIST, P.O.Box 3058, Macau, April 2008.

[VDMTOOL HP] http://www.vdmtools.jp/

[フィッツジェラルド 2010] ジョン・フィッツジェラルド, ピーター・ゴルム・ラーセン , 他 :VDM++ によるオブジェクト指向システムの高品質設計と検証 , 翔泳社 , 2010

[佐原] 佐原伸 , 形式手法の技術講座 ソフトウェアトラブルを予防する : Using the SOFL Method, ソフトリサーチセンター ,

オムロン株式会社
駅務機器のソフトウェアアーキテクチャの策定に非機能要求グレードを活用、非機能要求に関する顧客との認識の共通化を目指す

SEC journal 編集部

最近、業務機能に関する要求以外の非機能要求について、情報システムの発注者と受注者との間にギャップが存在すると、情報システム開発・運用にリスクが生じることが分かってきた。そのリスクを解消することを目的として、IPA/SECは2010年4月、非機能要求の見える化と確認の手段を実現する手法である「非機能要求グレード」[SEC HP] を発表した。オムロン株式会社は、非機能要求グレードの活用に取り組み、社内の設計者同士や顧客との間で非機能要求に関する認識を共通化する仕組みを構築している。その取り組みについて、同社公共ソリューション事業部*開発部開発2課主事の押山浩之氏に伺った。

非機能要求に関する投資対効果を顧客に説明することが課題に

オムロン株式会社（以下，オムロンと記述）において非機能要求グレードを活用しているのは、鉄道事業者向けシステムや道路交通向けシステム等、社会インフラを支えるシステムを開発・提供している公共ソリューション事業部*である。今回、同事業部では、鉄道事業者向けシステムのうち、券売機のソフトウェアアーキテクチャの策定に非機能要求グレードを活用した（図1）。非機能要求グレードはエンタプライズ系ソフトウェアを対象に作成されたツールであるが、同事業部の取り組みは、券売機等の組込みソフトウェアにも非機能要求の適用が可能であり、かつ効果をもたらすことを示したケースと言える。

鉄道事業者向けシステムでは、ICカード対応駅務端末の導入から約10年が経過し、次世代機器への移行も始まっている。次世代機器の開発においては、社会システムとして高い信頼性を担保するという従来の要件に加え、少子化による旅客収入の伸び悩みを背景として開発コストを低減することが重視されている。そのため、鉄道事業者に対してのシステムの提案時には、投資対効果について、機能要求・非機能要求の別なく従来以上の明確な説明が必要になってきている。「我々が非機能要求を明確に定義して、非機能要求の投資対効果を顧客に理解いただける表現で説明し、安心していただくことが大切になっている」と押山氏は話す。

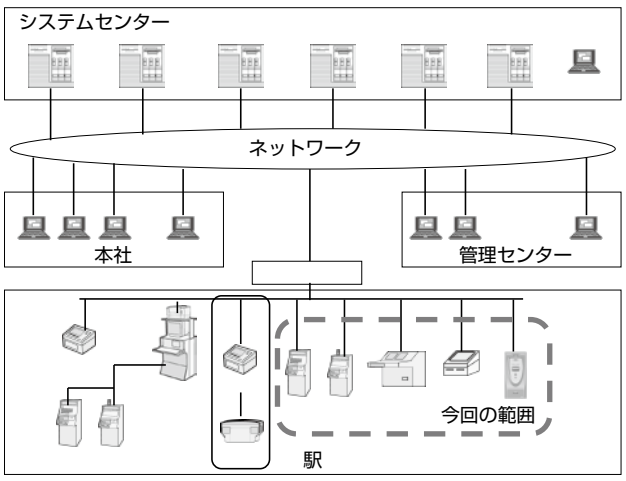


図1　鉄道事業者のシステム概略と今回の活用範囲

「活用シート」が迅速な利用を後押し

同事業部で非機能要求を定義する作業に取り組んだのは2008年頃のこと。当時は、非機能要求の定義に関する知見が少なく、ソフトウェアの品質標準であるJIS X 0129を参考としていた。しかし、非機能要求に関するJIS X 0129の記述内容はあいまいであり、またソフトウェア品質に限定されているので、他の指標も必要であった。そしてたどりついたのがIPA/SECの非機能要求グレードだった。押山氏は、「非機能要求グレードはユーザを巻き込んで作成されているので、これをもとにすれば顧客と話が出来る。券売機のソフトウェアアーキテクチャに適用してみよう」と考えた。ここでは、IPA/SECのスプレッドシートで作成された非機能要求グレード活用シートが迅速な利用に役立ったという（図2）。

IPA/SECの非機能要求グレードには3つのモデルシステムが提示されている。このうち鉄道関連のシステム自体は社会システムであるため「社会的影響が極めて大きいシステム」に、また、券売機のような端末の場合は、駅に設置する複数台での可用性を考えればよいため「社会的影響が限定されるシステム」にそれぞれ相当する。IPA/SECではモデルを選択してから非機能要求グレードを適用することを推奨しているが、同事業部は、まず非機能要求グレー

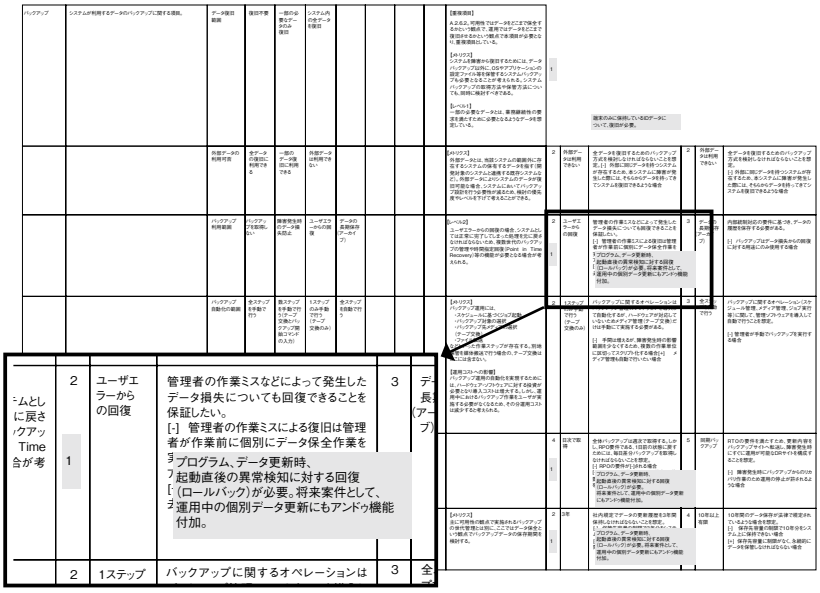


図２　IPA/SEC による非機能要求グレード活用シート
（オムロンでは一部改変して使用）

ドに示されているすべての項目に対して同社のシステムの特性を当てはめていくというアプローチとした。その理由は、非機能要求グレードがエンタプライズシステムを想定して作成されていることから、必ずしも券売機のような組込みシステムには合致しない可能性を考えたからだ。ところが実際には、「『社会的影響が限定されるシステム』というモデルケースに近い結果が得られた」と押山氏は言う。このことは、非機能要求グレードのモデルシステムは組込みシステムにも適用出来ることを示している。

こうして作成した要求項目及びレベルは、券売機のソフトウェアアーキテクチャに取り込むことになるが、その作業を進める上で助けとなったのは「非機能要求グレードの一覧表に、項目と他の規格との関係が明示されていたこと」と押山氏は話す。同事業部は上述のように、JIS X 0129 に基づいてアーキテクチャを作成しており、そのアーキテクチャに非機能要求グレードで確認した項目を適用する作業がスムーズに進んだのである。なお、「将来的に、非機能要求グレードのセキュリティに関する項目の定義とセキュリティ基準を作成している他の機関の定義との整合性が図られるようになると、非機能要求グレードの利便性がいっそう向上する」と同事業部は考えている。

非機能要求に関する社内の認識の共通化に大きな効果

このシステム開発において、非機能要求グレードを採用した効果は数多い。まず挙げるべきは、非機能要求に対する設計者の認識を共通に出来たことだ。今回の券売機の開発に関わったすべての設計者に対し、非機能要求に関する項目やレベルを明示したことにより、非機能要求に対する認識を統一することが出来たのだ。「非機能要求グレードは、上流の設計のフェーズで認識を合わせるのに効果がある」と押山氏は語る。また、マネジメント層等の社内ステークホルダと共通認識を持つことにも有効だという。IC カード乗車券で言えば、鉄道事業者間における IC カード乗車券の共通利用が広がる中、各鉄道事業者同士やシステムベンダ同士が非機能要求に関して共通認識を持つことが重要になっていると考えられる。今後は、そうしたステークホルダとの意識合わせにも効果を発揮すると期待する。

そして、非機能要求グレードで「いちばん興味深くかつ活用出来た」と押山氏が話すのは、運用・保守性に関する非機能要求の記述だ。同事業部は、非機能要求グレードで運用・保守性として取り上げられている項目を券売機のソフトウェアに適用したことによって、機器の稼働時間、ログデータの内容やサイズ等、保守に関する要件を数値で定義出来たという。運用フェーズにおいて配置しておくべきサポート要員の人数を考える指針ともなった。内部的なサポート体制を考える上でも有効と同事業部は見ている。

非機能要求グレードの活用に際して、同事業部は、要求項目の記述に同社の組織名や製品名といった固有名詞を採用するという工夫を行った。これは、記述内容の詳細を社内の設計者がすぐさま把握出来るようにする配慮だ。更にエンタプライズ系には無かった項目を追加して活用している。例えば券売機の設置環境を考慮し、システム環境・エコロジーの大項目に独自の小項目として「防塵」「直射日光」「絶縁」を追加、そのレベルを定めたことも同事業部の工夫と言える。品質向上やコスト削減を課題に、同事業部は今後、現在取り組んでいる端末から他の駅務端末へと非機能要求グレードの適用を広げ、順次、サーバー系ソフトウェアにも適用していく考えである。

参考文献

[SEC HP] IPA/SEC：非機能要求グレード～システム基盤における非機能要求の見える化ツール～，http:/sec.ipa.go.jp/reports/20100416.html

会社情報
商　号　オムロン株式会社
設　立　1948 年 5 月
本　社　（京都本社）京都府京都市下京区塩小路通堀川東入
（東京本社）東京都港区虎ノ門 3 丁目 4 番 10 号
資本金　641 億円
代表者　代表取締役社長　作田　久男
従業員数　5,133 人（2010 年 3 月 31 日）
事業内容　制御機器・FA システム事業、電子部品事業、車載電装部品事業、社会システム事業、健康医療機器・サービス事業、環境関連機器・ソリューション事業、パソコン周辺機器・産業用組込みボード事業

＊ 2011 年 4 月 1 日より「オムロンソーシアルソリューションズ株式会社」として、オムロン株式会社より分社。

最近の重要システムのトラブル事例に関する緊急レポート

ー新年早々の2大システムトラブル事例が示す対策の死角と教訓ー

SEC 統合系プロジェクト
プロジェクトリーダー
立石　譲二

1 新年早々発生したシステム絡みの２つのトラブル

うさぎ年の正月早々、2つのIT障害※1がマスコミでも大きく取り上げられ、話題になった。一つは1月5日の東京消防庁の119番通報配信システム、もう一つは1月17日のJR東日本の新幹線総合システム（通称COSMOS）でそれぞれ発生したトラブルである。いずれも重要インフラ※2と呼ばれるシステムで発生したもので、システムの不具合により、国民生活に直結するサービスが一時的に停止または低下する事態となった。以下が報道された範囲での障害の概要である。

【東京消防庁 119 番通報配信システムの IT 障害】

2011 年 1 月 5 日に発生した東京消防庁の 119 番通報配信システムの不具合によって、年明け早々約 4 時間半にわたって 119 番通報がつながりにくくなるという障害が発生した。同庁のシステムを運用していた現場職員が、通報処理システムと救急車等の運行状況を管理するホストコンピュータを結ぶ中継器の空きソケットを見つけ、ケーブルが外れていると勘違いして、誤接続したことが原因だった。

【JR 東日本新幹線総合システムの IT 障害】

2011 年 1 月 17 日早朝から発生した JR 東日本の新幹線運行管理システム COSMOS におけるシステム上の不具合で、東北、上越、秋田、山形及び長野の各新幹線が全線で一時ストップし、合計 15 本が運休する等 8 万人を超える乗客に影響が出た。本トラブルの引き金になったのは、一部の駅で雪によるポイント故障が発生し、COSMOS 上で計 24 本の列車に途中駅での停車指令が入力を指示したことだった。この停車指示により、その後の後続列車のダイヤに COSMOS 上の設計上限である毎分 600 件を超えるダイヤ変更が発生し、端末画面表示が一部表れなくなったため、システム上の問題の有無を確認する間、運行指令職員の判断で運行上の安全確保のため列車を一時的に全面停止したというものである。

ここまで読み進んだ読者の中には、「操作員の人為的ミスが原因だ。現場の緊張感が足りないからだ」、「こんな単純なミスがどうして起こるのか？」といった感想を抱く方もおられることだろう。トラブル発生企業に原因究明を急かす一方で、企業責任の追及と謝罪要求、バッシングの嵐が巻き起こり、テレビ会見が開かれ、カメラの前で組織の責任者が陳謝し、「再発防止に万全を期す」と締めくくることで収束を見る場合も多いと思う。

しかし現実には、類似のトラブルがまたどこかで必ず繰り返されてしまう。思うにこれは、現場の緊張感の欠如や人為的ミスが問題の本質ではないことを証明しているのではないか。どこかで私たちは問題の本質を見落としているのだ。間違った診断から正しい治療法が出て来るはずがない。

これら２つのトラブルは本来何の関係もなく、全く独立に起きたものだ。しかし、ちょっと見方を変えると、情報システムと運用現場との間に存在する「共通の死角」の存在を再認識させる出来事である。これを機会に、今後、同種のシステム障害を未然に防ぐためにも、私たちが見落としていた本質とは何か、言い換えれば、ITに起因するトラブルを根っこから解決していくために、システムを供給する側、利用する側が共通に理解しておくべき真の教訓は何かについて、読者の皆さんと一緒に考えてみたい。

2 重要インフラの信頼性 ーまずは正確な現状の認識からー

今や国民生活や経済活動になくてはならない社会インフラの多くは、ITによって支えられている。私たちの安全・安心に

脚注

※1　IT障害：重要インフラサービスにおいて発生する障害（サービスレベルを維持出来ない状態等）のうち、ITの機能不全により引き起こされるものをいう。（出典：「重要インフラの情報セキュリティ対策に係る第２次行動計画」，政府情報セキュリティ政策会議決定，2009年２月）

※2　重要インフラ：他に代替することが著しく困難なサービスを提供する事業が形成する国民生活及び社会経済活動の基盤であり、その機能が停止、低下又は利用不可能な状態に陥った場合に、我が国の国民生活又は社会経済活動に多大なる影響を及ぼす恐れが生じるものをいう。（出典：「重要インフラの情報セキュリティ対策に係る第２次行動計画」，政府情報セキュリティ政策会議決定，2009年２月）

IT が直結している形だ。このため、今日まで重要インフラに関わる IT の信頼性やセキュリティは、一般のシステムに比べて格段の対策がとられてきている。

図 1 は、社団法人 日本情報システム・ユーザー協会（JUAS）が毎年実施している「IT 動向調査」のうち、2009 年度に行った情報システムの信頼性実績に関する調査結果である。ご覧の通り、重要インフラ情報システムについては、一般の企業基幹システムに比べ、既に高い信頼性が確保されていることが分かる。米国の ATM システムの稼働率はおそらく 99%から高くて 99.9%程度 [ガートナーリサーチ] だが、図 1 を見ると日本の場合は確実に 99.999%以上の稼働率が確保されている。ATM が半日停止すると日本では社会問題になるが、米国では新聞にすら載らないという。なぜなら米国では大多数の国民が、99.9%そこそこの稼働率でも手数料なしで 24 時間・休日稼働する方が望ましいと考えており、一方、日本ではこんなにも高稼働率が保証された ATM に囲まれた生活を送りながら、国民の大多数は、信頼性やセキュリティは空気のようにただで提供されて当然だと考えている。

こうした安全性や信頼性に対する社会的価値観の違いを論じるのは別の機会に譲るとして、ここでは、重要インフラの信頼性の現状を正しく理解した上で、エンジニアリングの面から今後の有効な手立てを考える際のポイントを見ていくことにしたい。

3 エンジニアリングとしての重要システムの信頼性問題 ーソフトウェアや情報システムの特質ー

もともと機械や材料分野を起源とする信頼性は、摩耗や劣化といった経年変化に基づいて、図 2（1）のような信頼性曲線を前提に取り扱われることが多い。一方、ソフトウェアには摩耗や劣化は存在しないので、図 2（2）のように、条件が変化しない限り、利用時間の長短によらず信頼性は一定水準を維持出来ることになる。しかし、これらソフトウェアで構成される情報システムの信頼性となると、おそらく図 2（3）のような曲線をたどることになる。この場合の時間経過に伴い生じる信頼性の低下は、利用しているソフトウェアや情報システム自体の品質ではなく、設計時点で作り込まれた機能や利用上の前提条件と、時間の経過と共に変化した現状の利用状況との間にギャップ（乖離）が出来ることによって生じる。従って、長期

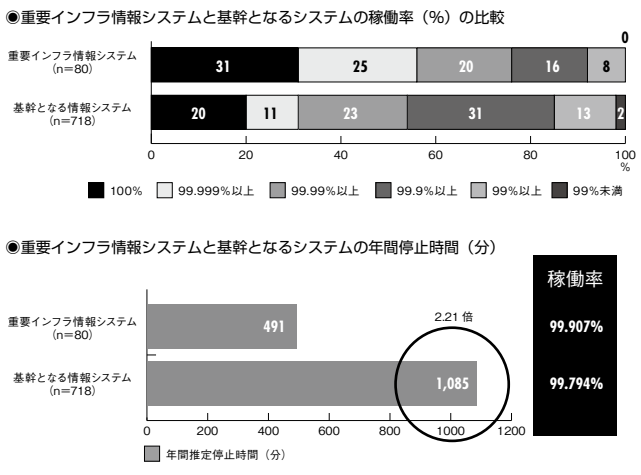


図 1　重要インフラ情報システムの信頼性の現状

的に見ると、現実の情報システムは改修や更新によってこの種のギャップを解消すべく、図 2（4）のようなシステムライフサイクルをたどることが多いと考えられる。

当たり前の話ではないか、と思われるだろうが、この点が結構見落とされがちであり、ソフトウェア・情報システムの信頼性を考える上での「第 1 の本質（ポイント）」になる。情報システムが正しく設計され、プログラムの欠陥が全く無かったとしても、その後、機能に対する要求が変化したり、運用の状況が変化したりすると、外部環境との適合性の低下によって、結果的に情報システムとしての信頼性は低下するのだ。事実、冒頭に紹介した 2 つの重大トラブルは、ソフトウェアのバグや欠陥によって起きたわけではなく、設計通り正しく機能していた中で起きたトラブルだ。

今日の情報システムを取り巻く環境変化の中で最も大きなものは、システム設計・開発者とシステム運用・利用部門との関係（立場）の変化である。システムの信頼性問題を考える際、長い間、情報システム及びその構成要素としてのソフトウェアの信頼性と利用者側の信頼性とを分けて考えてきたが、これでは一向にシステムの信頼性は向上せず、システム障害の発生も減少しなかった。しかし、最近は、利用者が操作ミスを起こすのは、その人間が悪いのではなく、ミスを誘発させるようなソフトウェアの機能設計やインターフェース設計の品質に問題があるからだと考え、従来のソフトウェアの品質概念を利用目的や利用形態との適合性に拡張しようとする動きが拡がってきた。IT 以外の分野に目を向ければ、これまで高い信頼性を求められてきた原子力発電や航空機の分野では、制御ルームやコックピットでの計器類や操作レバーの形状に至るまで、徹底してこの考え方が採用されている。

IT も人の命に直結する時代になった今、ソフトウェア品質を利用状況への適合性まで拡張しようという考え方が生まれてきたことはやや遅きに失した感があるが、自然な流れと言えよう。ソフトウェア品質特性の国際規格である ISO/IEC 9126 における「利用品質」の考え方（図 3）がそれであり、利用者に誤操作やミスを誘発させるのは利用者がたるんでいたのではなく、ソフトウェアが利用状況に適合していなかった、つまり、ソフトウェアの品質に問題があったとする考え方である。

この第 1 の本質を押さえると、利用品質に関する開発側と利用側の時間・空間的隔たりと、利用状況変化への感度という「第 2 の本質（死角）」が見えてくる。

4 重要システムの信頼性対策の意外な死角 ーゆっくり進む変化はたとえ重大なものであっても認知されない（非機能要求軽視への警鐘）ー

最近、いろいろな場面で「茹で蛙」の例えが引用されている。本来水温に敏感なはずの蛙だが、水に浸かった状態から非常にゆっくりと温度を上げていくと、温度変化に気づかずに蛙は茹だって死んでしまうという例えだ。この例えは、地球環境問題や社会構造変化に対して、ある種鈍感な日本人の国民性への警鐘を込めて様々な場面で引用されるケースが増えている。このように目には見えないゆっくりとした変化に鈍感なのは、私たちが日々利用する情報システムと運用者の関係についても当てはまる。

前述したように、摩耗や疲労といった経年劣化を伴う機械や材料と異なり、ソフトウェアの品質を考える際に重要となる「変化」には、「機能に対する要求の変化」と「利用（運用）形態の変化」の 2 つがある。このうち、前者の機能要求の変化につ

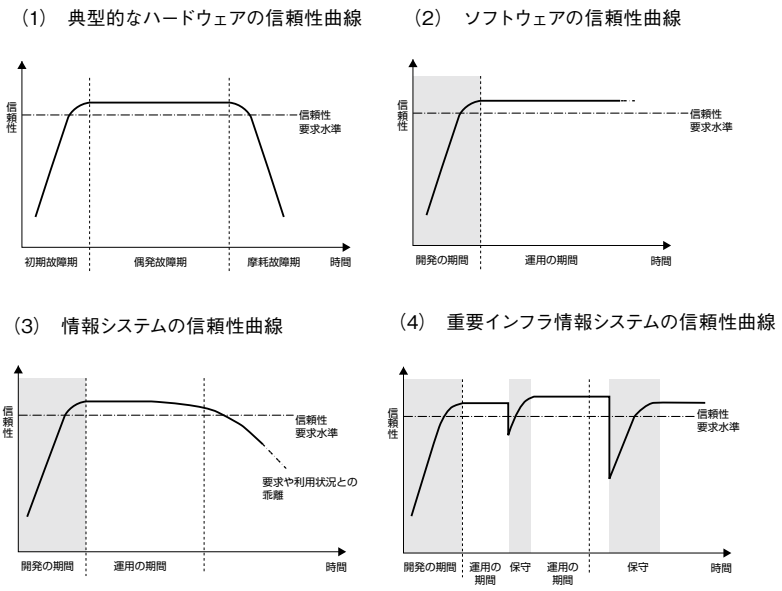


図2　信頼性に関する曲線

いては、日常の業務に直結する「機能要求」はシステムの開発側も利用側も常に意識する反面、処理速度、応答時間、記憶容量制限、セキュリティ等のいわゆる「非機能要求」と呼ばれる部分は、おかしな話ではあるが、これまで利用側が開発側任せにしていた部分が多かったために、往々にして利用部門では普段、正常稼働している状態では意識されないケースが多い。とくに、重要インフラのサービス供給を支えている基幹システムの場合は、システムの運用開始から 10 年以上の長期にわたって利用されているものも珍しくなく、ひとたび利用部門の引き継ぎ事項から「非機能要求」の内容が漏れてしまうと、それ以降、運用現場の誰一人そのことを知らない状況が生まれてしまう。

JR 東日本の新幹線総合システム（COSMOS）について言えば、運用開始は 1995 年であり、設計当初は東北、上越新幹線の運行管理をすればよかったものが、現在では長野、山形、秋田を加えた 5 新幹線の全業務をシステム管理することになっている。これは明らかにシステム運用状況が変化したことを意味し、運用開始当時は十分な余裕を考慮して設定された運行データの修正件数の上限 600 件という「非機能要求」事項が、現在の運用状況の中で妥当かどうかの検証が必要だったということになる。更に悪いことに、この上限違反時に起こる画面表示に関する制約事項が利用の現場に引き継がれていなかった。

重要情報の共有漏れという点に目を向けると、東京消防庁の

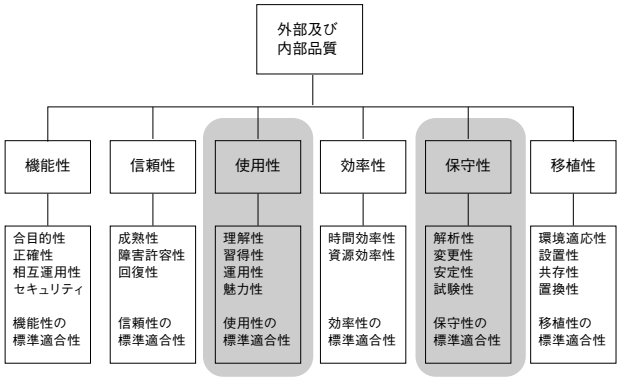


図3　ソフトウェアの品質特性 (ISO/IEC 9126) における「利用品質」

ケースについても、LAN ケーブルの中継装置で空きソケットとケーブルがあれば、初めてそれを見つけた消防庁の職員が思わずつなげてしまったというのは無理からぬ話だし、これを職員の注意力や緊張が足りないと言うのは酷というものだ。非機能要求に関連する情報共有上の問題を考えれば、これまでトラブルが起きなかったのは単に幸運だったに過ぎない。肝心なのは、システムの性能や利用上の制約に関する重要な情報が利用現場で適切に引き継がれていなかったこと、その結果として、運用開始当時の利用環境が時間の経過と共に変化しているにもかかわらず、現在の利用形態に即した改善や見直しが行われて来なかったことであり、決して職員の不注意や気の緩みがトラブルの原因ではない。

世の中で情報システムがまだ珍しかった頃には、情報システムの設計や開発の担当者は高度な専門家であり、それを操るのはコンピュータのことなど全く知らない現場の素人（エンドユーザ）という図式が長い間幅をきかせてきた。そのような情報システムでは、エンドユーザは厳格なルールや手順を守らなければ情報システムは言うことを聞いてくれなかったし、そのためにエンドユーザは時間をかけて利用者研修を受けなければならなかった。もし利用している間に何かトラブルが起きれば、それは多くの場合、システムを利用するユーザ側に手落ちがあったと考えられ続けてきた。

しかし、国民の誰もが情報システムに取り囲まれて生活している現在、システムトラブルの原因をいちいち利用者の落ち度の責にされたのではたまらない。情報システムの設計・開発側がもっと真摯に問題の本質に向き合わなければ、同じ過ちは必ず繰り返される。日々気がつかないほどゆっくりではあるが、着実に変化して来ている私たち社会と情報システムの関係の中で問題の本質を理解しなければ、冒頭の茹で蛙のように、座して死を待つことになりはしないだろうか。

今後の重要インフラの信頼性対策を危うくするもう一つの死角は、「恥の文化」といわれる日本人気質に由来する「知見共有の仕組み」の欠如だ。

5　重要システムの信頼性対策の意外な死角 ―似たような事例は実は過去にも起こっていたのに知見を共有する仕組みがどこにもない―

SEC では、2008 年に重要インフラ情報システム信頼性研究会を設置し、過去 5 年間に発生した重大システム障害事例について調査を行い、開発者、運用者及び専門家による討議を通じて、障害の要因分析と未然防止、障害発生時の被害拡大防止から再発防止に至る情報システムの信頼性向上対策に関する PDCA サイクルを回すための「管理フレーム」を取りまとめた[※3]。

この分析の過程で、今回障害が発生した新幹線総合システム（COSMOS）が 2008 年 12 月 29 日に起こしたシステム障害により、始発から新幹線が全面停止したという事例が取り上げられ、同研究会と関係者との議論を通じて表 1 のような対策が取りまとめられている。

障害原因の分析と再発防止策の検討の結果、以下のことが指摘されている。

・**抜本対策**　予防保全：「現場業務担当者とシステム担当者の間でデータ切り換え期限（デッドライン）の情報共有が無

かったか、あるいは長年の運用の中でこれが暗黙知となっており、両方の担当者に忘れられていた可能性がある。運用ルールの不徹底とデッドラインを過ぎた場合の対応方法のマニュアルとそれによる訓練及びその訓練の結果を活かす実践が不十分」

- **抜本対策**　重要部分：「運用スケジュールを含む運用ルールを関連部署間で共有しておき、例外事項が発生した場合の対応方法のマニュアルと、そのマニュアルに基づく訓練を十分に行っていく」

こうして見ると、過去に起きた様々なトラブルから得られた教訓や対策のポイントは、実は事業分野を超えて類似の障害発生を食い止めるのに役立ったはずである。一般のシステム障害では、どれほど大きなトラブルでも、航空や鉄道のような事故調査やその後の報告書公開といった仕組みが存在しない。規制当局による要請や行政処分がない限り、基本的にはトラブルを起こした当事者からの自主的な取組みや報告に任されている。その一方で、日本社会には産業界も含め身内の恥はあからさまにしたくないという「恥の文化」があり、客観的かつ冷静な知見の共有を妨げる要因になっている。もし、世界の航空機業界がこれと同様の企業文化に漬かっていたら、きっと今でも世界のどこかで主翼がもげたり、圧力隔壁が壊れたり、自動操縦装置が緊急時に解除されなかったりして、多くの人命が失われていたことだろう。重要インフラのIT障害防止の面でも、一日も早く、客観的な原因分析と再発防止に関する知見共有の仕組みを整えるべきだ。

6 おわりに

一概に決めつけることは危険だが、過去に起きたシステム障害の2大原因を挙げるとすれば、「システム機能の不適合」と「情報共有の不備」である。これらが実際の障害につながるのは、システム更新、運用方式の変更、新サービス稼働、列車ダイヤ改正等何らかの変曲点であることが多い。だとすれば、過去の変曲点で起きた苦い教訓を適切に共有し、それを着実に再発防止に生かしていくことが重要だ。更に言えば、情報システムを取り巻く環境が刻々と変化していることを考慮に入れ万全を期すためには、こうした変曲点にボロを出しやすい項目（システム容量、日付変更タイマー、性能上限／下限違反等）を日頃から重点的にリストアップし、異常が起こるか、起こった場合には、運用担当者はどのような心理状況の下でシステムを操作することになるのか、その状況下で運用現場がトラブルを回避する方法・手順は適切か、等を机上演習や訓練等で確かめ、必要ならば改善しておくということも場合によっては必要だろう。

その際、「訓練（Drill）」と「演習（Exercise）」は混同されやすいが、保安用語としては全く性質のことなるものである点に留意が必要だ。「訓練」とは、事前の答えが用意されたもの、すなわち決められた通りに事が運ぶよう練習すること。「演習」とは答えを見つけるための検証、すなわち、現状の手順、要員、設備が非常事態に対処可能か検証することである。これらを考えると、演習はともかく訓練すら実施せずに「再発防止に万全の措置を講じている」と言い切る企業責任者の神経とはいかなるものか。せめて、これら訓練・演習を通じたPDCAサイクルを回す仕組みを設けてから発言してもらいたいものである。

現状でも十分に高い日本の技術力を背景に、今後、新成長戦略に沿って官民連携し、インフラ輸出を進めていこうとしている。日本の持てる技術力を世界や地球的課題の解決に活用していくためには、ハード、ソフトの優秀さだけではなく、こうした表には見えない運用面での技術蓄積も合わせた信頼性への総合力をアピールしていかなければ、海外からの信頼は勝ち取れないのではないだろうか。

以上、最近のトラブル事例を題材に勝手な見解を述べたが、各方面での議論のきっかけになることを願うと共に、読者の皆様のご批判、ご助言をいただければ幸甚である。

脚注

※3　報告書案は本年1月末から広く一般に公開し、意見募集を行った。本稿をお読みいただいている時点では正式版のPDFがIPA/SECウェブサイトにて公開されているので、ぜひともご参照いただきたい。
「重要インフラ情報システムの信頼性向上の取り組みガイドブック ～情報システムの信頼性管理に必要な組織内の役割分担と活動の枠組み～」
http://sec.ipa.go.jp/

参考文献

[ガートナーリサーチ] Dataquest Insight: Unplanned Downtime Rising for Mission-Critical Applications (2008年9月分析, 10月3日発行), ガートナーコンサルティング分析

表1　2005年7月～2010年1月に発生した障害事例における対策の検討結果

事例内容（Web報道の要約）	障害概要（Web報道の要約）	検討メンバーが想定した主な原因	問題早期発見 ※問題発生時の原因特定	緊急対策	障害再発防止策		ダウン時間短縮対策
					抜本対策 予防保全	抜本対策 重要部分	
JR東日本の新幹線がシステム障害で始発から全面停止	JR東日本の新幹線がシステム障害で始発から全面停止、復旧は午前8時に延期。13万7,700人に影響。	前日のダイヤ乱れの影響で、運行システムCOSMOS（COmputerized Safety Maintenance and Operation systems of Shinkansen）内のデータの日付が不正な値になったため。 直接の原因は、列車データの入力が終わらないうちに午前5時にCOSMOSを立ち上げてしまい、それに気付いてデータ入力が終わった後それをCOSMOSに取り込もうとしたが、翌日のデータと認識されたことによる。			列車データを入れる担当（現場業務担当者）とCOSMOSの管理者（システム担当者）の間で、デッドライン（5:00）の情報共有がなかったと推定される。あるいは長年の運用の中でこれが暗黙知になっており、両方の担当者に忘れられていた可能性がある。さらに列車本数の増加と前日のダイヤの乱れなどで入力するべきデータ量が増加し、午前5時までに入力が終わらないデータ量になっていた可能性もある。 運用ルールの不徹底がある。デッドラインを過ぎた場合の対応方法のマニュアルと、それによる訓練、およびその訓練の結果を生かす実践が不十分。	運用スケジュールを含む運用ルールを関連部署間で共有しておき、例外事項が発生した場合の対応方法のマニュアルと、そのマニュアルに基づく訓練を十分に行っておく。	

形式手法の実践に対してよく尋ねられる質問とその回答
モバイルFeliCaの開発における形式仕様記述を通して

フェリカネットワークス株式会社
開発部２課　統括課長　博士（情報科学）
栗田　太郎

本稿は、モバイルFeliCaの開発において、形式手法の一つであるVDM※1を用いて、形式仕様記述を行った経験と知見に対するよくある質問の一部とその質問に対する回答を、気軽に読むことが出来るようにまとめた文章である。読者の形式手法や形式仕様記述、システム開発に関する検討、考察の助けとなれば幸いである。

はじめに

著者らはモバイル FeliCa [杉山 2007] の開発 [栗田 2010] において、形式手法 [荒木 2008] の１つである VDM [栗田 2009] を用いて、形式仕様記述 [荒木 2002] を行った [栗田 2007], [栗田 2008]。そして、国内外において広く著者らの経験や知見を文章や口頭を通じて発表してきた。

本稿は、これらの経験に基づき、著者らの発表に対するよくある質問の一部とその質問に対する回答を、読者が気軽に読むことが出来るようにまとめた文章である。既発表の論文では書かなかった、論文には直接的に書きづらかった著者の考えの一端を述べる。

なお、モバイル FeliCa の開発における形式仕様記述手法適用の詳細に関しては、上記の既発表の論文等を参照されたい。本稿は、単独で読むことが出来るように構成しているが、文章の位置付けから読者が既発表の経験論文や口頭での発表を読んだり、聞いたりした後に参照されることを想定している。

Q　形式手法には様々なアプローチがあると伺いましたが、導入に際してどのようにお考えになりましたか？　また、いずれも形式手法と言える、形式仕様記述とモデル検査の組み合わせについても伺いたいです。

A　私たちは、形式仕様記述言語 VDM++※2 と VDM Tools※3 を用いて、開発対象のシステム全体の機能仕様を表しました。また、仕様の記述とは別の機会に、仕様や設計の一部を状態遷移モデルとして表現するための言語 FSP※4 とツール LTSA※5、言語 PROMELA※6 とツール SPIN Model Checker※7 を用いてモデル検査を行いました。

誤解を恐れずに言えば、私たちの実践においては、これらの活動にとくに関係はありません。形式手法には様々なものがありますが、それらの連携にとらわれすぎず、自由に組み合わせて利用すれば良いと思います。

私たちの場合、形式仕様記述の目的は単純で明確です。プロジェクトのメンバが仕様を互いに誤解せずに表現、理解し合うために、仕様記述言語を用いて仕様を厳密に書き表したい、仕様をテストしたい、仕様を後工程に役立てたい、ただそれだけです。ただ厳密に書けば良いのです。

一方、モデル検査はその対象となるモデルを仕様記述言語で書き表し、その後に検査を行うという２段階を経ることになります。まず、検査の対象が明確である必要があるため、対象となる仕様や設計、プログラムの範囲と仕様を決定し、その仕様を厳密に論理式として書くという形式的な記述が必要になり、その後、仕様に対して仕様が満たすべき性質を式として表して検査することになります。このときの仕様や検査式の記述は、検査のための記述になりますので、コミュニケーションのための仕様記述よりも数学的に理解が難しかったり、プロジェクト全体での合意形成には向かなかったりするかもしれません。少なくとも私たちのプロジェクトではそうです。

それぞれの効果に対する感覚も異なります。仕様記述の場合、とくにシステム全体の仕様を表す場合は、厳密な仕様の記述により、あいまいな仕様に起因する課題の割合が減少すればその効果を実感することが出来ます。あいまいな仕様に苦しんでいる現場であれば、どこでもすぐに効果を感じることが出来るのではないでしょうか。

一方のモデル検査は、私たちの場合、検査の対象がシス

テムの一部分であるため、局所的に特定の目的で役に立っていると感じています。また、検査の結果、具体的な反例が見つからない、つまり何らかの問題が検出されない場合、(確認したい範囲において問題がなかったということは喜ばしいことではあるのですが) 確認出来ることがシステムのごく一部の、しかもある側面にすぎないため、効果の実感が宙に浮いてしまうかもしれません。ただ書くことで効果があるであろう形式仕様記述に対して、モデル検査はより対象と目的の明確化が重要になります。

Q 形式手法や形式仕様記述は難しいものなのでしょうか？

A そうですね。簡単ではない、つまり難しいのではないでしょうか。

ところで、例えば、プログラミングやテストは簡単なことなのでしょうか。そもそも、私たちが日常取り組んでいること、行っていることで、難しくないことなどあるのでしょうか。

その上で、あえて申し上げますと、形式手法や形式仕様記述が、他の手法や言語、ツール等と比較して特別に難しいことはないと思います。また、形式手法のすべてを学んだ後に、実践で用いること、何を書いたり、読んだりするべきなのかを最初からすべて見通すことはとても難しいので（というよりも形式手法に限らず、そもそも不可能であると思いますが)、はじめから無理に背伸びをすることもありません。

例えば、VDM++ 言語と VDMTools を用いる形式仕様記述の場合、これらは良くも悪くもプログラミングのための言語やツールの延長線上にあり、一般的なソフトウェア開発技術者であれば誰でも知っているであろう命題論理や集合、写像の基礎に加えて、述語論理（これも誰もが知っていることなのかもしれませんが）や VDM++ 言語の言語仕様の一部、VDMTools の簡単な使い方を学習すれば、基本的なモデリングを行うことが出来るようになります。また、例えばPROMELA 言語と SPIN Model Checker を用いたモデル検査の場合、有限状態遷移機械や時相論理、言語仕様の一部、ツールの使い方の基礎を学習すれば、基本的な検査が出来るようになります。

いずれにせよ、はじめからすべてを学習する必要はありません。仕様やプログラムを書きながら、少しずつ自らの記述を見直したり、開発環境の使い方に慣れたり、新しいことを取り入れたり、あるいは反対に基本的な記述や利用に絞ったりすることが出来れば良いのです。これは、実用的な開発に携わるすべてのプログラマが、例えばＣ／C++ 言語の仕様とライブラリと開発環境のすべてに熟達している訳ではなく、それでもとくに実際的な支障がある訳ではないことと同じです。

仕様記述をプログラミングと比較しますと、プログラムが動作する環境やシステム全体の都合等による制約が少ないため、記述時に考慮することが相対的に少なく、仕様策定に集中することが出来ます。この点に関しては、プログラミングよりも簡単である、ということが出来ます。

一方で、仕様を検討、記述する場合、まず、まだ何を開発するのか定まっていないため、と同時に、どのレベルの詳細度までを仕様として記述するのかを定める必要があるため、そして、そもそもドメインにおける課題や要求が複雑かつ不透明であろうがため、更には、要求や仕様に関してステークホルダと合意をしたり仕様の実現可能性について検証したりする必要があるため、簡単なことではありません（他の工程と比較して仕様策定が難しいと言っているのではありません)。このような要求の具体化や仕様策定の難しさの本質は、形式手法を用いるのか、あるいは用いないのかとは関係がないことです。しかし、形式仕様記述手法を用いることにより、仕様が整理されたり、議論が深まったり、その結果仕様が定まったり、ドメインやシステムに対する理解が深まったりする可能性が高まり、仕様があいまいであることや、自然言語の記述及び読解の能力の問題と仕様策定の根本的な困難性が重なり合い、仕様策定における課題を複雑化することを避け、仕様策定そのものに立ち向かっていくことが出来るかもしれません。

脚注

※１ VDM：Vienna Development Method。1970 年代の前半に IBM のウィーン研究所で、プログラミング言語 PL/I の仕様記述や、コンパイラの検証のために開発された手法。
※２ VDM++：ISO で標準化された汎用的な仕様記述言語 VDM-SL（Specification Language）に対して、主にオブジェクト指向の拡張を行った言語。
※３ VDMTools：VDM-SL や VDM++ 言語による仕様の記述や検証を支援するツール。
※４ FSP：Finite State Process。モデルをラベル付き遷移システム LTS（Labelled Transition System）として表現する仕様記述言語。
※５ LTSA：Labelled Transition System Analyser。FSP で記述したモデルの検査ツール。
※６ PROMELA：PROcess MEta LAnguage。モデルをオートマトンとして表現する仕様記述言語。
※７ SPIN Model Checker：PROMELA で記述したモデルの検査ツール。

Q 導入に際して、上司の方の反対はなかったのでしょうか？

A 上司の反対はありませんでした。私たちは上司に非常に恵まれていたと思います。

一方で、積極的な賛同も得られませんでした。しかし、私たちも含めて、形式手法についてよく分かっていなかったので（これは今もあまり変わりがありませんが）、理不尽に反対されないのであれば残念に思うことではないと思います。むしろ賛同されたら、上司がどこかで何かを聞きかじってきたのではないかと怪しむべきでしょう。これは半分冗談ですが、当時の上司が私に形式仕様記述手法を用いるように指示を出していたら、全力を尽くして抵抗していたかもしれません。

上司を含む開発の関係者との議論において重要なことは、もし新規の開発ではないのであれば、これまでの開発を振り返り、そしてこれからの開発に思いを巡らせ、開発の工程や組織の全体にどのような課題があり、これらを個別にあるいはまとめてどのように解決していくのかということについて、チームで、ステークホルダと議論していくことです。

例えば、あいまいな仕様や設計の不整合は課題のうちの一つに過ぎないのでしょう。ですから、形式仕様記述やモデル検査だけを取り上げて開発全体を議論することは出来ません。また、例えば、厳密な仕様の記述はテスト項目の増大という課題に対する方策の一つになるのかもしれませんし、モデル検査のためには設計書の記述方法を見直す必要があるのかもしれません。開発や組織の課題とその改善案は多岐にわたり、かつそれぞれが有機的に関係しているものなのです。

このような議論の中では、議論の対象となっているシステムやそのドメイン、プロジェクト、チームにおける開発の工程全体に対する認識と、仕様の位置付け及びその重要性が自然と浮かび上がってくるものです。その結果、仕様の記述に対して、現状のままでよいのか、あるいは何かを見直すのか、例えば形式仕様記述言語を用いて仕様を記述するのかどうかを決定すれば良いでしょう。

また、もう一つ重要なことは、形式仕様記述手法を用いるにせよ、あるいは用いないにせよ、あいまいな仕様を厳密に書き表そうとするのであれば、これまでの仕様記述に掛かる時間や人数を見直さなければならないということです。マネージャにとっては、ステークホルダとの関係の都合もあり、その記述のレベルはさておき、何を開発するのか、仕様は定まっているのかは非常に気がかりなことです。これまでよりも仕様の策定に戸惑っていると見なされないよう、上司やステークホルダと開発スケジュールや体制についてよく話し合っておくと良いでしょう。

それから、形式手法はあくまでもシステム開発の手法であり、発想や決定の方法論ではないため、これにより仕様や設計に対して、案の網羅的な検討と適切な選択を行うことがきるようになるわけではないことを合意しておきましょう。もちろん、厳密な仕様の記述により、開発ドメインに対する理解や、チームでの仕様に対する議論が深まり、仕様が具体化したり、洗練されたり、対案を思いついたりする可能性はあります。しかし、いくら議論を尽くしても、他にもっと良いアイディアがあるかどうかは、少なくともその検討時点では分かりません。何をしても、未来に考えつくかもしれないことを必ず発想することが出来るとは限らないのです。また、厳密な仕様の記述により、仕様の細部に対する明確な検討と管理が可能となり、結果として仕様変更の件数が増える可能性もあります。

いずれにせよ、仕様の変更に関する経緯や理由を考慮しながら、それが適切な変更なのか、あるいはそうではないのかを見極め、前者なのであれば、プロジェクトの内外において、仕様変更の数が多い、いつまでも仕様が決まっていない、なぜ最初からこの仕様になっていなかったのだなどと否定的に評価されないよう、プロジェクトの状況を適切に報告していく必要があります。

最後に、以上で述べたことは、形式手法の導入のみならず、一般に、プロジェクトやチームでの合意の形成にも通じることだと思います。

Q VDM++ 言語や VDMTools は長く安心して使うことが出来るものなのでしょうか？

A 難しい問題です。結論から申し上げますと、積極的に利用していくための障害は多くありませんが、例えば上司からの形式手法を適用せよとの依頼を断りたいのであれば、採用に踏み切らない理由をいくらでも挙げることが出来るでしょう。

VDM++ 言語の仕様や VDMTools の使い勝手や完成度が、形式手法に限らず、開発現場で用いる他の言語やツールと比較して大きな問題になることはないと思います。しかし一方で、C ／ C++ 言語の仕様や開発環境ほど成熟しているとは言えません（C ／ C++ 言語のコンパイラの欠陥にはいつも悩まされていますが）。また、利用者や開発環境の開発者の数が相対的に少ないため、依頼心の強い開発者が期待するサポートが得られないかもしれません。もちろんこれは、VDM++ 言語と VDMTools の現状に関す

ることであり、形式手法全体に、あるいは未来に対して言えることではありません。

形式手法に限らず、新しい言語やツールをチームで公式に採用することは難しいことです。私は、言語やツールの設計者や開発者、先行するユーザに会いに行くことをおすすめします。これによって、どのような思想や理念、熱意に基づいて設計や開発がなされているのか、他の言語やツールとの歴史的関係はどうなっているのか、将来の構想や見通しはどうなっているのかなどといったことが分かります。そしてはじめは学習者や利用者としてコミュニティに貢献していくことが出来たり、その後更なる協力関係を作ることが出来たり、さらには新しい手法を考案したりすることが出来るかもしれません。

また関連して、VDM++ 言語で書いた仕様書は、将来引き継ぎが出来るのかという質問を頂くことがあります。この質問に対するただ一つの回答はないのですが（この質問に限りませんが)、この課題は、VDM++ 言語や VDM Tools 固有の問題ではなく、ある文書を誰かが引き継ぐとはどういうことなのか、日本語やプログラミング言語の場合はどうかといったことについて考えることが重要なのではないか思います。というのも、ご質問くださる方々とお話しさせていただきますと、多くの方が、現在の開発において文書やプログラムの引き継ぎや再利用に苦しんでいらっしゃるようだからです。

Q 仕様を証明しないと意味がないのではないでしょうか？　またそもそも、証明をしないと形式手法を用いているとは言えないのではないでしょうか？

A 「意味がある」とはどのような意味でしょうか。
何度も繰り返していますように、私は、仕様を厳密に書くということを目的として具体的に活動することと、その結果記述した仕様書が存在することに意義があると考えています。これは現在、私たちが仕様記述言語を用いる理由です。もちろん、どのような時にもそのことだけに意義があるとは考えているわけではありません。同時に、誰もが仕様記述言語を用いて仕様を書くべきだとも考えていません。

もし、仕様や仕様の一部を証明したいのであれば、仕様を数理論理式として表現し、これを証明すれば良いと思います。これにより、仕様を証明したいという目的、更には仕様を証明することによって達成される、仕様を証明する真の目的が達成されるかもしれません。それにしても、大きな仕様をどのようにして証明出来る形式で表すのか、表した仕様は証明以外の用途、例えば仕様書として開発者が参照する用途でも用いることが出来るのか、仕様を変更したときに証明し直すのかなど、実システム開発の現場において仕様を記述し、これを証明するためには、解決しなければならないいくつかの課題があると思います。このことを言い換えますと、システムの中でも、さまざまな観点において熟考しなければならない、頻繁に変更する予定がない仕様の部分を、チームの中でも限られた特定の技能を持ったメンバが、プロジェクトやチームのメンバが参照する仕様書とは別に仕様の証明のために書き表し、これを証明することなら、すぐにでも可能なのではないかと思います。

このように、仕様を証明するためには、まず仕様を形式的に記述する必要があります。一般的な開発者が形式手法を用いる場合は、一足飛びに課題となっている仕様の証明を目指すのではなく、まずは仕様を形式的に記述することに挑戦し、開発対象を仕様として適切に表せるようになってから、システムの全体や部分を対象として、仕様の証明に取り組んでいけばよいのではないでしょうか。

「形式手法」という言葉の定義に関することであれば、私は、形式手法とはすなわち証明することであるという意見にはくみしません。しかし一方で、私たちが行っていること、仕様記述言語と呼ばれているものを用いて仕様を書き表したりすることが「形式手法」や「フォーマルメソッド」と呼ばれなくても、あるいは呼ぶことに値しなくてもまったく構いません（もちろんこれは、私が総称としての形式手法が指し示すものや証明に関心が無いということではありません)。なぜなら、私たちは「形式手法」を用いたり、研究したりすることではなく、厳密に仕様を記述する、ということを目的として活動しているからです。

Q プログラムの自動生成機能をお使いになりませんと、形式仕様記述の効果が半減してしまうのではないでしょうか？

A 基本的には、証明と同じ議論です。
私たちの場合、ツールが生成するプログラムは、開発の環境や対象には適さないものでした。また、自動生成されるコードは、開発者が読むことが出来るプログラムではなく、その拡張や品質の保証、サポートが出来ないと判断しました。しかし、自動生成したプログラムを用いて開発、運用されているシステムもありますから、そういったことが問題にならない場合もあるのだと思います。

プログラミング言語は、プログラムの記述のためにあり

ます。プログラムはコンピュータが具体的な処理を実行するための詳細な情報を、処理を実行するための様々な事情を考慮して記述するコードとデータであり、プログラムとして書かれた論理的な情報を起点として様々な役割を持つチームのメンバが開発を行うことは容易ではありません。仕様の論理から、記述のための記法への割り付けが素直に定義づけられた仕様記述言語を用いることにより、仕様を文書として自然に表現することが出来るのです。そしてこのときに、設計やプログラムの、仕様とは異なる具体性や事情を考慮し、仕様と設計、文書とプログラムの境界を見定めていくこと、場合によってはツールを用いて変換や具体化、抽象化していくことが重要になります。

Q 形式仕様記述により、どれくらいのコストを削減することが出来るのでしょうか？

A よく聞かれますが、答えることが非常に難しい質問です。

私たちは、形式仕様記述を行っているのではなく、ある時期にあるシステムを開発するために、あるプロジェクトにおいて、その時に集まったメンバと様々な手法や工夫や改善を重ね合わせて活動をしています。そして、手法や工夫や改善は全体として相互に補完し合いながら作用しています。

従って、ある手法単独の効果を直接的に述べることはとても難しいのです。また、ある手法の宣伝がしたいわけではありませんから、そのことを無理に申し上げる意義も見当たりません。

具体的には、例えば私たちのプロジェクトには、日本語で書いた仕様書と形式仕様記述言語で書いた仕様書がありますが、形式仕様記述言語による仕様書は日本語で書いた仕様書の品質向上に役立ちます。ですから、我々のプロジェクトにおいて、日本語で書いた仕様書だけでも十分だから形式仕様は不要であるということを後から言うことは出来ません（しかし、こういった議論はいつもあるものです）。そして、形式仕様が存在しなかったら、自然言語による仕様書やテストケースの品質も落ちるのかもしれません。しかし、極めて優秀な技術者により、頭の中で形式仕様記述と同様のこと、例えば論理の組み立てや確認を行いながら品質の高い仕様書やテストケースを作成することも出来るのかもしれません。形式手法の先生がお書きになった教科書に例題として載っている仕様に間違いがあったりもしますので（これはツールにかけるとすぐに分かるような欠陥なのですが）、そう簡単ではないのでしょうが。

一方で、もし現在の開発において仕様があいまいであることに起因する課題が山積しているのであれば、形式仕様記述手法の導入によって、これまでに発生した課題がどれくらい解決出来るのかを定量的に述べることが出来るかもしれません。あいまいな仕様にまつわるさまざまなバグや手戻りなどのトラブルと、トラブルの解決に掛かるコストや期間を計算することにより、どれくらいの無駄なコストを削減出来るのかを計算するのです。私たちは、苦しいながらも、形式手法の適用による具体的な効果について出来るだけ定量的に述べようと努力し、報告書をまとめています。機会があったらご覧ください。

Q 日本語で書いた文書の品質や、技術者の日本語の力が向上するとはどのようなことなのでしょうか？

A これには、いろいろな視点や議論があると思います。一つの考え方としては、通常、日本語を用いて仕様書、広くは文書を記述する場合、何を書くのか、どのように書くのか、対象を書き表すための文章の構成力や日本語の記述力はあるのか、想定する読者にその文書に対する読解力があるのかなどの様々なこと、課題について同時に考えていかなければなりません。しかし、自然言語と同時に形式仕様記述言語を用いる場合、何を書くのか、形式仕様ではどのような構成や表現になるのかが明確に出来る可能性があるため（もちろん日本語で書く場合、形式仕様とは構成や表現は全く同一では無いのでしょうが）、自然言語を用いて具体的にどのように表現するのか、ということについて集中的に考えることが出来るようになります。これは例にすぎませんが、日本語や仕様記述言語のいずれかを用いて文書を書くとしても、上達するまでは行為に対して同時に複数の目当てを持たず、今何をしようとしているのかをたえず考え続けることが大切だと思います。自戒を込めて申し上げますと、日本で普通に暮らしているからといって（普通とはどのようなことなのでしょうか）、日本語で文書を書くことが出来るとは限らないように（少なくとも私の身の回りでは）思います。

また、形式仕様記述言語の利用により非形式的な文書の品質も向上出来るのは、仕様の記述やテストなどにより、記述や検討、考察の不足が開発の初期段階で明らかになり、問題を解決するために議論を重ねるうちに、様々な抽象度で問題に対する理解が深まると共に、開発の目的や条件、制約を明確にしながら、構造や関係性の見通しが良く分か

りやすい文書構造の検討や記述を行うことが出来るようになるからです。そして、仕様に関するコミュニケーションのフィードバック先や修正の方法、回帰テストの方法が明確になることから、仕様の維持を継続する動機が生まれるからです。以上のような仕組みにより、あいまいな、更新されない、ひょっとしたら誰にも参照されることがない仕様書から遠く離れたところにプログラムやテストケースが存在している、という開発現場によくある課題から逃れられるのではないかと考えます。

更に、開発に関わるすべての仕様書や設計書を一つの形式仕様記述言語で表現することは難しいでしょうし、それぞれの表現の目的も異なりますから、これを目指すべきではないでしょう。私たちの場合、プロジェクトの外部の関係者が参照する機能概要仕様書や非機能仕様書は日本語で書いています。厳密に記述する形式的に書いた機能仕様書があることで、全体の基盤が盤石なものになると同時に、複数の言語を用いて記述する文書群の構成が安定したものになり、結果として日本語で書いた文書の位置付けや内容が明確になり、属人性を排したり、時間に耐えたりすることが出来るようになるのです。

Q 私のところではどうしたら良いでしょうか？

A それは私には分かりません。

しかし、まず間違いなく言えることは、それぞれの開発ドメインやプロジェクト、チームにおいて、開発の事情や対象、制約などが異なるということです。ですから、汎用的にこうすれば良いという万能の方法はありません。ソフトウェア開発における課題を一挙に解決出来る「銀の弾丸」はありません。また、他者の経験や知見は参考にはなっても、自分自身の具体的なことには当てはまりません。そして、自分ではない他者の言うことをそのまま鵜呑みにしてはなりません（これらは、先人に対して敬意を表さないということではありません）。ドメインやプロジェクト、チームのことは、自らの問題として考えていくしかないのです。

そして、新しい組織ではなかったら、あるいはまったく新しいシステムを開発するのではなかったら、従来の開発にどのような問題があるのか、それぞれの問題をどのような具体的な課題として分析出来るのか、それらの課題に対してどのような対応策が考えられるのか、最終的にどのような開発計画を立案するのかを考えると良いでしょう。もし、あいまいな仕様の記述が課題になっていないのであれば、どんなに精緻に、厳密に仕様を記述したとしても、とくに意義はないのかもしれません。

課題解決のためには、複数の手法や工夫を組み合わせて活用していかなければなりません。誰かに何かをさせられたり誰かに何かをさせたりするのではなく、先人や歴史に学びながら、基礎的な力を養成しながら身の丈に合わせて、一方で自身を少しずつ変えていきながら、何をどうしていくのかを考え続けながら、抽象と具体の間を行ったり来たりするより他にありません。また、専門家としての技術者は汎用的に、ドメインや事情により開発や記述の構造を開発出来る力とバランス感覚を磨いていくより他ありません。

記述と対話を行うことが出来る開かれたチームや環境を作ることが出来れば、様々な抽象度で厳密な仕様の記述と検証、維持が可能となる形式手法を用いて、本質的な議論と、効率良く品質が高いシステムの開発や、応用可能な技術力の向上を、チームで目指していくことが可能になるかもしれません。

謝辞

形式手法の開発現場への適用やその考察に当たり、九州大学の荒木啓二郎教授をはじめとする多くの方々にお世話になりました。厚く御礼申し上げます。また、これまでにご質問くださいました方々と、この発表の機会を与えてくださいましたIPAの新谷勝利氏に感謝いたします。

参考文献

[荒木 2002] 荒木啓二郎，張漢明：プログラム仕様記述論，オーム社，2002.
[荒木 2008] 荒木啓二郎：フォーマルメソッドの過去・現在・未来－適用の実践に向けて－，情報処理，Vol.49，No.5，pp.493-498，2008.
[栗田 2007] 栗田太郎：仕様書の記述スキルを鍛える－モバイル FeliCa 開発における形式仕様記述手法の導入事例，日経エレクトロニクス，pp.133-152，2007.
[栗田 2008] 栗田太郎：携帯電話組込み用モバイル FeliCa IC チップ開発における形式仕様記述手法の適用，情報処理，Vol.49，No.5，pp.506-513，2008.
[栗田 2009] 栗田太郎，荒木啓二郎：モデル規範型形式手法 VDM と仕様記述言語 VDM++ －高信頼性システムの開発に向けて－，信頼性，Vol.31，No.6，pp.394-403，2009.
[栗田 2010] 栗田太郎：モバイル FeliCa のソフトウェア開発における品質確保のための構造と実践－抽象度の制御やコミュニケーションの活性化に向けて，情報処理学会デジタルプラクティス，Vol.1，No.3，pp.148-157，2010
[杉山 2007] 杉山寛和，栗田太郎：携帯電話と FeliCa を融合したモバイル FeliCa 技術、情報処理，Vol.48，No.6，pp.561-566，2007.

CoBRA研究会

～説明力のある見積り方法の普及を目指して～

http://cobra.mri.co.jp/

代表幹事／株式会社三菱総合研究所　主席研究員
石谷　靖

CoBRA研究会は、“CoBRA法”の活用を考える自主的な集まりとして、2009年4月に発足した。CoBRA法とは、現場の“勘”を見える化してソフトウェア開発の工数の見積りモデルを構築する手法である。CoBRA法活用経験に基づいて議論し、見積りモデルの構築方法から見積りモデルを活用したプロセス改善等、CoBRA法のより良い活用方法を探求することを目的に活動している。

1 CoBRA 研究会

CoBRA 研究会は、2007 年度の IPA/SEC による CoBRA 法実証プロジェクトに参加したメンバを母体に、CoBRA 法を実際に活用して“惚れ込んだ”有志が集まっている（表 1）。

おのおのが所属する組織での CoBRA 法の効果的な活用を目指すと共に、ソフトウェア開発の現場で、より多くの人に活用していただきたいと考えて活動している。

2 CoBRA 法について

CoBRA 法は、Cost estimation, Benchmarking and Risk Assessment の略で、コブラと読む。少数の実績データと熟練者の経験・知識を利用して工数見積りモデルを構築する手法である。ソフトウェア開発プロジェクトや工数見積りの熟練者（経験豊富なプロジェクトマネージャ等）の協力の下に、その経験・知識を工数変動要因として定義・定量化し、透明性と説明性が高く、コストマネジメント可能な見積りモデルを構築する。プロジェクト関係者の経験・知識を集積・集約することから、組織能力の共有･向上へもつながる手法である。何よりも、熟練者が普段行っている見積りのノウハウをモデル化する手法であり、現場の受けが大変に良いものである。実績データも 6 個程度から見積りモデルの構築を行うことが出来、様々な組織での見積りモデル構築を可能としている [CoBRA HP]。

もともとは SEC の共同研究先であるドイツ・フラウンホーファ財団実験的ソフトウェア工学研究所（IESE）で 1997 年に提唱された手法で、SEC では 2004 年から IESE と共同研究を進めている。2010 年 3 月末には、CoBRA 法に基づく見積りモデル構築を支援するツールが公開された [SEC CoBRA HP]。SEC journal でも CoBRA 法がこれまで何度も紹介されている [SEC journal]。

表1　CoBRA研究会メンバ（組織）

独立行政法人 情報処理推進機構（オブザーバ）
株式会社アイネス
株式会社 NTT データセキスイシステムズ
沖電気工業株式会社
人事院 CIO 補佐官
T & D 情報システム株式会社
日新情報システム開発株式会社
株式会社日立製作所
株式会社三菱総合研究所
三菱電機株式会社

五十音順：以上 9 企業・団体（メンバ数 28 名）。2011 年 1 月 31 日現在

3 CoBRA 研究会の活動

2009 年の研究会発足時は、月 1 回の定例会を開催し、各社における CoBRA 法活用の経験と課題の共有を行った。2009 年 4 月のキックオフを最初に、2011 年 3 月までに 8 回開催している。1 年目が終わった 2009 年度の最後の回においては、2010 年の目標として「対外的な

発信」を設定した。

その一環として、SECとの協働を2010年度から積極的に進めている。SEC研究員の方の定例会議への参加（発足時から）はもちろんのこと、IPAの各種催しの機会にSECと連携し、CoBRA法の説明とSECが提供しているCoBRA法による見積り支援ツール[SEC CoBRA HP]のデモを行った。CoBRA法の知名度向上・普及に向けた対外発信の一つとして、ホームページも公開している[CoBRA HP]。

表2にCoBRA研究会の対外発信の実績を示す。図1は、2010年12月神奈川で開催された組込み総合技術展（ET2010）におけるCoBRA法のパネルでの説明風景である。

また、2010年度に入り、広くCoBRA法を知っていただく活動の一環として、ガイドブックを執筆した。2009年度からの2年間の議論や2007年度からの見積りモデル構築・活用経験に基づいた内容の濃い本と自負している。「CoBRA法入門」のタイトルで2011年4月に出版の予定である。表3に主な内容を紹介する。ベンダ企業のプロジェクトマネージャやPMOの方から、ユーザ企業の情報システム部門でソフトウェア開発の見積り評価に悩んでいる方まで広く参考にしていただきたい。

表2　CoBRA研究会の対外発信

対外的な活動	時期	主体
ホームページ	2009年 7月	CoBRA研究会
ソフトウェア開発環境展	2010年 5月	IPA/SEC
技術セミナー	2010年 7月	CoBRA研究会
IESEとのワークショップ	2010年10月	IPA/SEC
ET2010	2010年12月	IPA/SEC

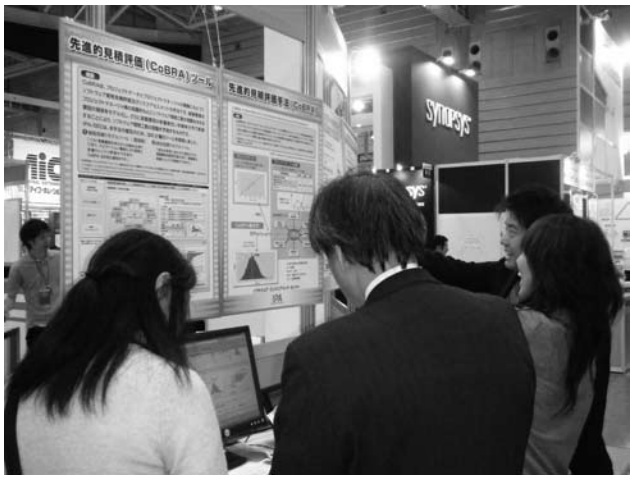


図1　ET 2010での説明風景（2010年12月）

4 今後の予定

引き続き、SECとの協働による技術普及を議論・実施していく予定である。CoBRA法は単に工数見積りに留まらない、プロジェクトマネジメントやプロセス改善への応用範囲の広い手法であるので、活用・応用方法を更に議論し、提示していきたい。

本研究会はまだ小さな集まりであるが、見積りモデル構築の経験・試行をされた方に参加いただき、より広い活動にしたいと考えている。ぜひお問い合わせいただきたい。

表3　「CoBRA法入門」の主な内容

1. 本書の読み方
2. やってみよう工数見積り
   （30分で工数見積りモデル）
3. CoBRA見積りモデルでできること
4. CoBRA法とは
5. CoBRA見積りモデルの構築手順詳細
6. CoBRA見積りモデルの保守
7. 構築・活用ベストプラクティス

2011年4月、オーム社より刊行予定。

参考文献

[CoBRA HP] http://cobra.mri.co.jp/index.html
CoBRA法の概要と詳細な解説（FAQ）があるので参照されたい。
[SEC journal] SEC journal, No.7, pp.10-21, 2006年 ; SEC journal, No.11, pp.26-31, 2007年 ; SEC journal, No.19, pp.377-379, 2009年
http://sec.ipa.go.jp/secjournal/index.html　からダウンロード可能（要登録）。
[SEC CoBRA HP] http://sec.ipa.go.jp/tool/cobra/
CoBRA法による見積り支援ツールを無料配布している（要登録）。

■問い合わせ先

・株式会社三菱総合研究所　CoBRA研究会 事務局
Fax：03-5157-2148
E-mail：cobra_info@mri.co.jp

組込みシステム産業振興機構

組込みシステム産業振興機構
主任研究員
舩戸　稔弘

http://www.kansai-kumikomi.net/

組込みシステム産業振興機構は、関西を組込みシステム産業の一大集積地とするための産学官協働プラットフォームとして、人材育成サービスをベースとした企業の競争力の向上と、開発支援サービスによる組込みシステム開発の品質向上と受発注機会の拡大を図るため、企業単独では取り組むことが難しい事業を第三者機関として効率的に集約したサービスの提供を実施している。また、講演会やセミナー等を通じた相互交流によるビジネス機会の提供等、組込みシステム産業の活性化に寄与していくことを目指している。

1　組込みシステム産業振興機構の設立

2007年からの3年間「組込みソフト産業推進会議」において、関西の情報家電を中心とするメーカや情報系企業、また最先端の研究を実施している大学・研究機関等産学官が集結し、組込みソフト産業の活性化に必要なサービスや機能を検討してきた。この中で積み上げてきた成果を深化・発展させ、発注側企業・受注側企業の双方に実効性がある具体的な事業として展開を行うと共に、今後、組込みソフトウェアの領域拡大が見込める環境・エネルギー、医療、FA制御、自動車等に検討分野を拡げ、ソフトウェアだけでなくハードウェアを含めた「組込みシステム」の産業活性化を図るため、2010年6月に「組込みシステム産業振興機構（理事長：宮原秀夫　独立行政法人 情報通信研究機構 理事長）」（以下、振興機構）を設立した。

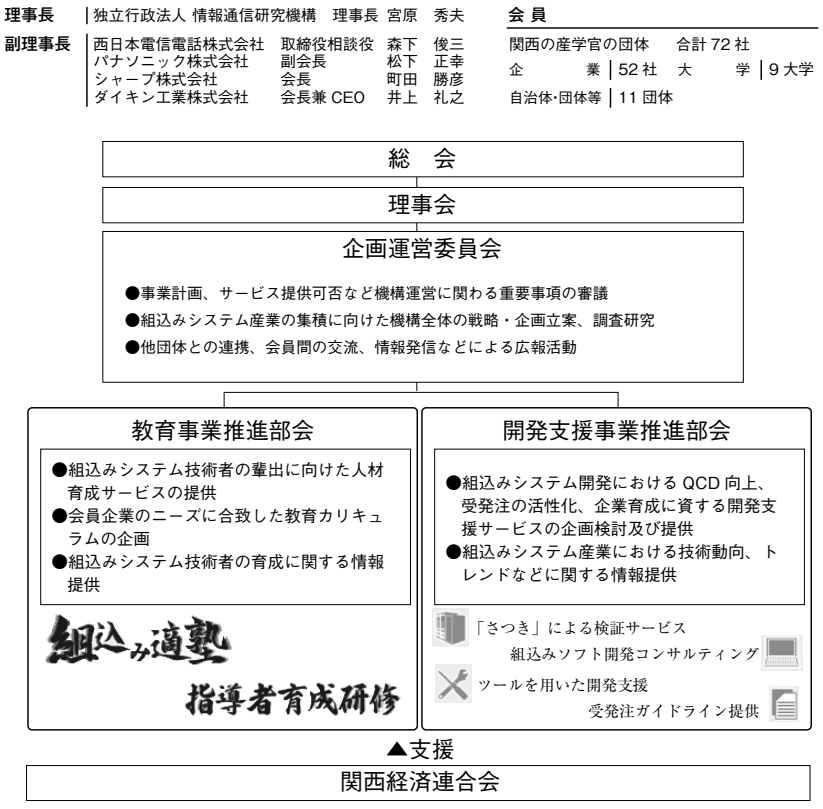


図1　組込みシステム産業振興機構・組織図（2010年6月7日現在）

2 振興機構の取り組み

(1) 活動概要

振興機構では、事業の柱を「教育」「開発支援」「企画・広報」の3つとし、その事業運営においては、委員会や部会、ワーキンググループ（以下、WG）を設置して、具体的なサービスの提供や新たなサービスの企画・検討を実施すると共に、交流サロンの開催等会員相互の交流の場を提供している。

(2) 委員会・各部会における具体的活動

① 企画運営委員会

企画運営委員会（委員長：伊東則昭　西日本電信電話株式会社 代表取締役副社長）では、振興機構の事業活動の「舵取り役」として、組込みシステム産業の集積に向けた振興機構全体の事業戦略の立案、調査研究、事業計画の策定やサービス提供可否等、機構運営に関わる重要事項の審議や検討を行っている。また、振興機構の活動に関連する他団体との連携及びセミナーや講演会を通じた会員交流の場を企画実施することで、振興機構の事業活動を推進している。

第1回会合（2010年7月16日開催）では、組込みシステム産業の目指すべき目標の検討や、振興機構が活動する分野・範囲について議論を行い、検討すべき課題を確認した。第2回会合（2010年11月30日開催）では、「組込みシステム産業の集積化」をテーマに、以下の3つの側面で議論を実施した。

・発注者側企業の既存ビジネスの発注先を関西に変更することによる受発注の活性化
・国内の他地域または海外からの受発注の活性化
・新たなビジネス、商品創造による新規受発注の創出

これらの議論により、受注側企業・発注側企業の関係におけるマッチングの仕組み、及び新たなビジネス創出の仕組みを整備することの重要性を確認し、本委員会において継続して検討していく予定となっている。

② 教育事業推進部会

組込みシステム産業分野の拡大に対応した人材輩出の拡大及び事業モデルの確立に向けて、産学官連携を更に推進し、「組込み適塾」による実践的知識・技術を備えた高度組込み技術者の育成や、「指導者育成研修」による社内育成担当者を介した効率的な初級・中級技術者の裾野拡大をテーマに取り組んでいる。

具体的な実施研修としては、システムアーキテクトを育成するためのプログラムである「組込み適塾シス

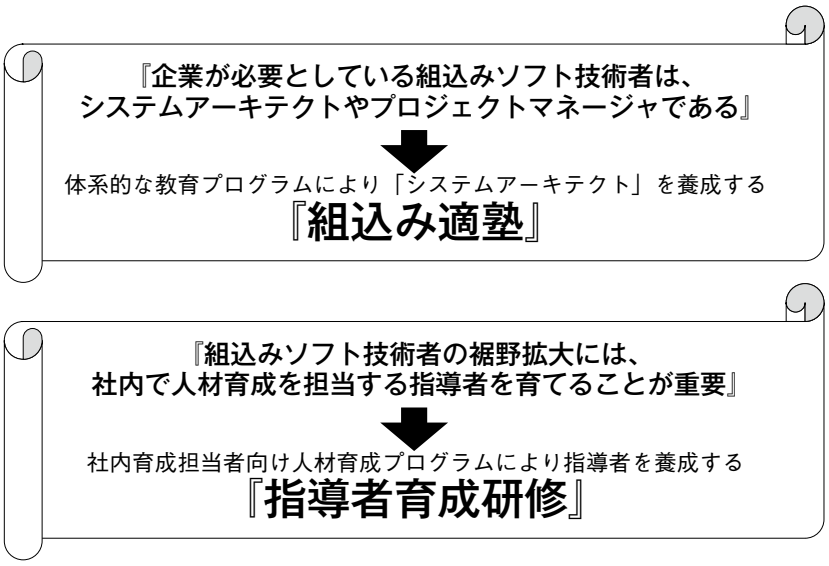


図2　人材育成サービスメニュー

テムアーキテクトコース」（塾長：今瀬真　大阪大学大学院情報科学研究科長）を、2010年6月より23日間の日程で開催した。このプログラムは、大阪大学や奈良先端科学技術大学院大学等の大学、パナソニック株式会社やシャープ株式会社等の企業と独立行政法人 産業技術総合研究所関西センターが連携し、それぞれの分野において最先端を行く講師陣を招聘することで、体系的かつ、より実践的なカリキュラムとなっている。関西の企業だけでなく、東海・中部地方からの長期出張で参加された受講生も合わせ23名の皆様が参加される等大変盛況であった。また、知識の習得だけでなく、知識を活用する力を育成するために「組込み適塾実践演習コース」を3コース（「実践的クラス設計演習（アンドロイド）（講師：中本幸一　兵庫県立大学大学院応用情報科学研究科 教授）」「実践的モデル検査（講師：関澤俊弦　大阪学院大学情報学部情報学科 講師）」「リバースエンジニアリング＆リファクタリング（講師：柳原圭雄　大阪市立大学大学院工学研究科 准教授）」）、8月から9月に開催した。

更に、初級・中級レベルの組込みソフト技術者の裾野を効率的に拡大させるため、社内育成担当者向け研修として、ソフトウェア技術者の業務プロセス改善手法の社内展開を目指す「パーソナルソフト開発作法指導者養成講座（講師:田中裕彦　パナソニック株式会社）を5月から6月に開催した。この講座については、企業からの要望が多かった、オンサイトコースを新設し、9月に開催した。

今後の活動としては、教育事業として研修コースの開催を行うだけでなく、産業育成としての観点で広く人材育成事業の検討を進めていく。

③ 開発支援事業推進部会

これまでの3年間で検討してきた、組込みソフト開発の品質向上及び受発注活性化につながる各種サービスについて、「有効性」「実現性」「継続性」等の観点より、下記の4つのサービスを開始した。

・「さつき」による検証サービス
・組込みソフト開発コンサルティング
・ツールを用いた開発支援
・受発注ガイドラインの提供

各サービスについては提供状況、会員からの要望等を基に改善し、サービスレベルの向上に向け検討を実施している。

図3　開発支援サービスメニュー

また、既に提供開始中の4サービス以外に検討すべき課題に取り組むため、開発支援事業推進部会では、5つのWGを設置し検討を進めている。

・開発サービス検討WG

技術支援に関するサービスを集約し、新しい開発支援ツールの導入検討、既存サービスの普及・拡大のための検討を行う。

・事業共同組合設立検討WG

事業共同組合により分野の異なる企業間の協力で新しい開発提案が出来る仕組みを検討する。

・企業マッチング検討WG

発注企業と受注企業、企業と人とのマッチングの仕組みを検討する。

・アジア諸国との連携共創WG

日本企業の海外進出だけではなく、海外組織・企業の誘致も含めた協業についても検討を行う。

・組込みソフト製品認証WG

製品認証事業に関するビジネススキームの構築と商用課題、技術課題、財務課題等の検討を行う。

これらのWG活動を通じて、その検討結果を具体的なサービスや事業として展開することで、企業の競争力向上と組込みシステムの受発注の支援を実施していく。

(3) その他の活動

振興機構では会員相互交流を目的に「組込みビジネス交流サロン」と「技術者向け交流サロン」の異なる2つのテーマで交流サロンを開催している。「組込みビジネス交流サロン」では、2010年10月に東大阪宇宙開発共同組合副理事長の吉田則之氏より「まいど1号人工衛星プロジェクト」の講演会を開催し、11月には米国在住のITジャーナリスト小池良次氏より「クラウド、ブロードバンドに注力する米国の情報通信業界」と題してご講演いただいた。また12月には、小惑星探査機「はやぶさ」のプロジェクトマネージャである川口淳一郎氏より「はやぶさ」プロジェクトの意義についてご講演いただいた。

「技術者向け交流サロン」では、11月に独立行政法人産業技術総合研究所関西センターの大崎人士氏及びメルコ・パワー・システムズ株式会社の早水公二氏より、クラスタシステムを用いた上流工程大規模テスト環境構築事例とモデル検査の適用事例についてご講演いただいた。2011年2月には組込みソフトウェア管理者・技術者育成研究会の島敏博氏より「モデル駆動開発を組み合わせたソフトウェアプロダクトライン開発入門」と題してご講演いただいた。

毎回多くの企業・団体から参加いただいており、今後も企業に役立つ、業界動向や最新の技術情報を提供していく予定である。

他団体との連携では、独立行政法人 情報処理推進機構（IPA）との連携協定による活動協力や、社団法人組込みシステム技術協会との連携協定の締結による会員サービスの相互利用を可能としている。また、組込みソフトウェア管理者･技術者育成研究会（SESSAME）や財団法人 関西文化学術研究都市推進機構 新産業創出交流センターの組込みソフト起業化推進事業と連携し、講演会や情報連携を行うことで会員相互の交流を図っている。

広報活動では、組込み総合技術展 関西（ET-west）への協賛をはじめ、組込み総合技術展（ET）や組込みシステムシンポジウム2010（ESS2010）等の展示会への出展を通じた情報発信と、講演会・無料セミナーの開催により振興機構の活動について情報発信を行っている。

3 今後の展開

現在、振興機構では、組込みシステム産業の活性化に向け「教育事業」と「開発支援事業」「企画･広報事業」の3つの事業を中心に活動を展開しているが、今後は、その活動範囲を広げ、さらなる産業の活性化・産業育成に結び付く産業発展モデルを検討したいと考えている。

■問い合わせ先
・組込みシステム産業振興機構　池田事務所
大阪府池田市緑丘1-8-31
独立行政法人 産業技術総合研究所関西センター
関西産学官連携研究棟　202号室
電話：072-751-8405

はずさない就職活動とは

IPA 顧問　学校法人・専門学校 HAL 東京 校長
鶴保 征城（つるほ　せいしろ）

テレビや新聞で報じられているように、リーマンショック後の就職難は半端ではない。2010 年春の大学卒業生は約 54 万人､そのうち就職したのは約 33 万人で、就職率は 61%となっている。大学院等への進学者を除くと、約 10 万人、大卒者の 2 割の若者が就職浪人を余儀なくされているとのことだ。

言うまでもないが、不安定な採用は社内の人事構成を歪めてしまうので、企業にとっては安定的な採用を継続することが望ましい。採用を極端に絞ると、若手社員がいつまでも雑用に追われるし、中堅の人材に仕事が集中して多くの知識やノウハウを抱え込んでしまう。景気が回復して、一転大量採用すると、今度はポストが足りなくなる。経営者はこのようなことは百も承知なのだが、昨今の業績悪化と先行き不安がこれを許さない状況にある。

大学生数の推移はどうなっているかというと、若者の人口減少を進学率の増加で補う形で、20 年間ぐらいほぼ一定数になっている。若者（19 ～ 22 歳）人口が 1993 年の 816 万人から 2009 年に 513 万人に減少する一方、28%程度であった進学率は 2009 年には 50%を超えている。表現は悪いが、「猫も杓子も大学へ」と言えるのではないだろうか。そこへ降って湧いたように就職氷河期が到来した。若者こそよい迷惑である。

では、どうすればよいのだろうか。その答えは三つあるのではないかと思う。

一つは、大企業から中小企業に目を向けることだ。日本では、企業数でいうと実に 99.7%、従業員数でも 70%が中小企業である。大企業の新規採用は多くても 10 万人を超えず、不況時では 4 万人程度になると言われている。大企業に集中して何十社も受験を繰り返すのは得策ではない。就職活動の手段がインターネットのナビサイトになってから、エントリーが大企業に集中するようになった。しかも、特徴のないエントリーシートを安易に送りつけるのだから、内定までに苦労するのも無理はない。大企業は安全な豪華客船のようで、中小企業は大海に翻弄される小船のように思うかもしれないが、今の時代はこのようなことはない。大きくても危ない会社、せっかく入社しても昇格も昇給もさせない会社は枚挙に暇がない。

二つ目は、産業構造の変化に対応すること、つまり第３次産業であるサービス産業やコンテンツ産業などに職を探すことだ。日本は、どちらかと言うと、ものづくりは実、サービスは虚、とする考え方があるが、時代の流れを見るとそうは言っていられないのである。

三つ目は、採用する側は欲しい人材の要件を、応募する側は自分自身のスキルをはっきりすることだ。ソーシャルネットワークをビジネスモデルとする会社に DeNA という会社があるが、ここの募集要項は実に明解である。事業内容、求める人物像、職種内容・特徴、必要な技術などが待遇とともに、数ページにわたって詳細に書かれている。これを見て、「時代の流れに乗っているから、御社を希望しました」などというあいまいな動機の応募者はいなくなるのではないかと思う。筆者はこれをスペック型人材採用と呼んでいるが、これが出来ない会社､これに応じられる人材を輩出出来ない大学は、これからの熾烈な競争から退出するしかない。

手前みそで恐縮だが、筆者が勤務している専門学校では上の 3 項目を着実に実施して､ほぼ 100%の就職（内定）率を実現している。未曾有の就職難は教育の問題とも考えられる。

上述した大学生数の推移を見ると、逆に若者の就業人口が激減していることが分かる。学生数を約 260 万人でほぼ一定とすると、職に就いている若者は、1993 年は 556 万人、2009 年は 253 万人だ。学生は原則として稼ぎがないから、GDP への寄与は極めて少ない。最近のベストセラーである「デフレの正体－経済は人口の波で動く」（角川書店，2010 年発行）で筆者の藻谷浩介氏は、生産人口（15 ～ 64 歳）の減少がデフレの主因である、と主張しているが､高学歴化も一つの要因と考えられる。

人口の減少、つまり稼ぎ手の減少が避けられないとすると、残された策は生産性の向上しかない。ご存知の方も多いと思うが、日本の日本の GDP の 80%近くを占める　サービス産業の生産性が意外に低いのだ。逆に言うと、ここに成長のチャンスが残されている。

Code Complete 第２版〈上〉〈下〉
—完全なプログラミングを目指して

Steve McConnell（著）、クイープ（訳）
ISBN：4-89100-455-X〈上〉
ISBN：4-89100-456-8〈下〉
日経 BP 社刊
B5 変型判・664 頁〈上〉、584 頁〈下〉
定価 各 6,405 円（税込）
2005 年 3 月刊

若手ソフトウェア技術者育成に有効な体系的入門書

近年、モデリングが重要視されている。システムの上流工程、とくに要求分析やアーキテクチャ設計がシステム開発の成否を決定する。しかし、ソフトウェアのコードの重要性は変わっていない。

本書はコード作成を中心としたプロセスにおける振る舞いが主対象であり、コード作成をキチンとできる若手の育成に有効な書籍である。

私が開発現場にいた頃、ソフトウェア開発の効率化のために自己啓発として取り組んでいる際に、本書（当時は第 1 版）と出会った。分厚く内容も濃く、具体的な話から精神論まで多岐にわたり、実務での活用と気づきの両面で嬉しい出会いであった。ソフトウェア工学講座の受講のきっかけとなり、体系的にソフトウェア工学を学ぶ重要性に気づかせてくれた。ソフトウェアを生業とする者のスタンスを学び、今のキャリアがあると感じている。

本書で示される具体的な内容は、開発現場で自分達のやってきたことの正当性を感じる反面、至らなさも確認できる。私は本書を参考に、自チームで利用する各種チェックシートを作った経験をもつ。その後、そのチェックシートは自分たちの経験や他の書籍などの情報を加え、多くのチームで共有・活用したことを思い出す。

上下巻で約 1,200 ページを超える、とても分厚い書籍である。しかし、若手のソフトウェア技術者に対して、無理にでも読ませておきたい書籍である。若手には、本書の内容を理解したうえで開発に従事してほしいと願う。（渡辺登）

ソフトウェア現場力ハンドブック

松本 吉弘　編
ISBN：978-4-274-50230-9
オーム社刊
B5 変型判・456 頁
定価 4,410 円（税込）
2009 年 4 月刊

ソフトウェア開発に必要な「現場力」とは

今や、ソフトウェアあるいはそれを一構成要素とするシステムの開発にあたっては、実に多方面にわたる知識が必要になってきている。この知識は、アカデミアで学習するだけでは十分には獲得出来ないことは明白であるし、特定の仕事環境の経験だけでも獲得出来ないこともまた明白である。

「現場力」という用語を編者は次のように定義している。「上流における企業ガバナンス、IT ガバナンス、IT 統制が理想的に行われたとしても、細部における統制上の矛盾に起因するしわ寄せは、必ず現場を襲ってくる。これに耐え得る力が『現場力』である」。

この「現場力」は、しかしながら、「作る側」にのみ具備されればよいというものではない。

「ソフトウェア現場力」という表題の本書は、「開発の現場」にとどまらず、「ソフトウェアの企画・運用」更には「ソフトウェアのビジネス」に及ぶ幅広いトピックスについて、33 人のそれぞれの分野の専門家が各トピックス数ページにわたって説明している。「序編 ソフトウェア現場力とは」は 4 つの章、「第 1 編 ソフトウェア現場力を生み出す基盤」は 11 の章、「第 2 編 現場力向上のための核心技術」は」6 つの章、「第 3 編 ソフトウェア現場力実践例」は 5 つの章、そして特別寄稿の計 27 章、456 ページの大作である。

全体を鳥瞰するためには、「第 1 編第 1 章 現場力を生かす基本事項」、25 ～ 63 ページをまずお読みになるようにお勧めする。

（新谷勝利）

編集後記

今回の突然の東日本大震災の被災者の方々に謹んでお見舞い申し上げます。また、このジャーナルも震災の影響を受けて発送が遅れましたことをお詫び申し上げます。

俳聖と呼ばれる松尾芭蕉は、俳句の世界では「芭蕉の前に芭蕉なく、芭蕉の後に芭蕉なし」と評されています。前人未踏の境地を開拓したと言われる松尾芭蕉が、俳聖と言われる所以を遺憾なく表現した言葉です。

さて、翻って我々のソフトウェア・エンジニアリング分野の閉塞感を打破するには何が必要でしょうか？　関係者の知恵をお借りして新しいソフトウェア・エンジニアリングを創造したいところです。

今回は、SECが進める事業のテーマの適用事例について技術解説、論文、成果事例紹介、アングルやご寄稿をいただいた記事の中で紹介しています。新たなマーケットを開拓されておられる方々のご参考になれば幸いです。なお、同封されているアンケートを通じてSECへのご意見をお待ち申し上げております。

（久保）

桜纏う京都本法寺

（撮影：金沢成恭）

SEC journal® 第7巻第1号（通巻26号）　2011年3月31日発行

編集兼発行人　〒113-6591　東京都文京区本駒込2-28-8　文京グリーンコート センターオフィス16階
独立行政法人 情報処理推進機構　ソフトウェア・エンジニアリング・センター　所長　松田　晃一
Tel.03-5978-7543　Fax.03-5978-7517
http：//sec.ipa.go.jp/
編集・制作　〒101-8460　東京都千代田区神田錦町3-1　株式会社オーム社　Tel 03-3233-0641

お知らせ

SEC journal 論文募集

IPA（独立行政法人 情報処理推進機構）
ソフトウェア・エンジニアリング・センターでは、
下記の内容で論文を募集します。

応募様式は、下記のURLをご覧ください。
http://sec.ipa.go.jp/secjournal/papers.html

論文テーマ

ソフトウェア開発現場のソフトウェア・エンジニアリングをメインテーマとした実証論文

- 開発現場への適用を目的とした手法・技法の詳細化・具体化などの実用化研究の成果に関する論文
- 開発現場での手法・技法・ツールなどの様々な実践経験とそれに基づく分析・考察、それから得られる知見に関する論文
- 開発経験とそれに基づく現場実態の調査・分析に基づく解決すべき課題の整理と解決に向けたアプローチの提案に関する論文

論文の評価基準

a. 実用性(実フィールドでの実用性)
b. 可読性(記述の読みやすさ)
c. 有効性(適用した際の効果)
d. 信頼性(実データに基づく評価・考察の適切さ)
e. 利用性(適用技術が一般化されており参考になるか)
f. 募集テーマとの関係

応募要項

投稿締切り

年4回、3ヵ月毎に締切り、締切り後に到着した論文は自動的に次号審査に繰り越されます。
（応募締切：1月・4月・7月・11月各月末日）
締切り後、査読結果は1ヶ月後に通知
詳細スケジュールについては、投稿者に別途ご連絡いたします。

提出先

独立行政法人　情報処理推進機構　ソフトウェア・エンジニアリング・センター内　SEC journal事務局
eメール：sec-ronbun@ipa.go.jp

その他

- 論文の著作権は著者に帰属しますが、採択された論文については SEC journalへの採録、ホームページへの格納と再配布、論文審査会での資料配布における実施権を許諾いただきます。
- 提出いただいた論文は返却いたしません。

論 文 賞

SEC journalでは、毎年SEC journal論文賞を発表しております（候補論文が少ない場合は、翌年の審議とする場合有り）。受賞対象は、SEC journal掲載論文他投稿をいただいた論文です（論文賞は最優秀賞、優秀賞、SEC所長賞からなり、それぞれ副賞賞金100万円、50万円、20万円）。

論 文 分 野

品質向上・高品質化技術
レビュー・インスペクション手法
コーディング作法
テスト／検証技術
要求獲得・分析技術、ユーザビリティ技術
見積り手法、モデリング手法
定量化・エンピリカル手法
開発プロセス技術
プロジェクト・マネジメント技術
設計手法・設計言語
支援ツール・開発環境
技術者スキル標準
キャリア開発
技術者教育、人材育成

SEC journal バックナンバーのご案内

詳しくはhttp://sec.ipa.go.jp/secjournal/をご覧ください。

ESxR特集号

No.19

No.20

No.21

No.22

No.23

SEC journal No.24
第7巻第1号（通巻26号）
2011年3月31日発行

IPA®
独立行政法人 **情報処理推進機構**

編集兼発行人
〒113-6591　東京都文京区本駒込2-28-8　文京グリーンコート　センターオフィス16階　Tel.03-5978-7543　Fax.03-5978-7517
独立行政法人 情報処理推進機構　ソフトウェア・エンジニアリング・センター
所長　松田　晃一
URL：http：//www.ipa.go.jp/

ISSN 1349-8622