組み込みソフトウェア開発の問題点の分析
── 技術者間のコミュニケーションに注目して ──

菅谷　克行

概　要

近年，様々な分野でソフトウェアに対する要求が高くなってきている。技術分野では顕著であり，それだけソフトウェアに対する依存度が増大していることを表している。特に，組み込みシステムに対する要求は拡大の一途を辿っており，その結果，組み込みソフトウェアが大規模・複雑化し，技術者間の情報・知識の共有を困難にしている。そのため，開発工程で様々な不具合が発生したり，場合によっては製品が市場に出てからもトラブルが発生したりしている。これらの課題に対して積極的に対処するためには，ソフトウェア開発に関する知識を効率的にマネージメントし，技術者間のコミュニケーションロスを減少させることが重要な点としてあげられる。

本研究では，問題点を具体化するために組み込みソフトウェア開発に携わる技術者にインタビューし，その分析結果から，ソフトウェアのライフサイクル全体を視野に入れた支援を考察することを目的とする。特に，技術者間のコミュニケーションに注目して分析・議論を展開する。

1　はじめに

近年，社会生活の様々な場面にソフトウェア技術が浸透してきている。特に，工業製品に代表される科学技術の応用分野では，ソフトウェア技術に対する要求は年々増大しており，その結果，組み込みソフトウェアに対する依存度が非常に増している。

例えば，自動車産業界は，まさにその顕著な例を示す分野の一つである。今や，自動車にはカーナビに代表される情報システムや，安全性や環境への配慮を実現するための電子制御システムなど，様々なソフトウェアが搭載されている。そのため，市場の様々なニーズに応えるための新しいモデルをタイムリに投入するためには，ソフトウェア開発が極めてクリティカルな要素となっている。特に，自動車用エンジン制御装置 (ECU：Engine Control Unit) に対する要求は拡大の一途を辿っており，その結果，ECUの動作を制御する組み込みソフトウェアが大規模・複雑化し，その開発に莫大なコストが必要となっている (Ohsuga et al. 2000)。そのため，組み込みソフトウェア開発のための支援環境（佐野2002）（二上2004）やモデル検証技術（中島2004）（水口他2005），更には自動コード生成（柿崎2002）なども活発に研究されており，最近では，自動検証・モデル検査の研究にチューリング賞 (The 2007 A. M. Turing Award 2007) が授与されている。

『人文コミュニケーション学科論集』6, pp.137-157.

通常，複数の技術者が協同で一つのソフトウェアの生産に携わっているが，ソフトウェアの大規模・複雑化は，技術者間の情報・知識の共有を困難にしている。その結果，開発工程で発生する不具合や問題点も多くなり，かつ，それら不具合箇所・問題点を発見することも容易ではなくなっている。場合によっては深刻な不具合が市場に出てから発見される場合もある。これらの課題に対して積極的に対処するためには，ソフトウェアに関連する情報・知識を如何にマネージメントするかという点が重要な視点の一つである。特に，複数の技術者間で発生するコミュニケーションロスを減少させることが第一歩であると考える。

そこで本研究では，問題点を具体化するために組み込みソフトウェアの開発者にインタビューし，その分析結果から，ソフトウェアのライフサイクル全体を視野に入れた支援を考察することを目的とする。特に，技術者である人間のコミュニケーションに注目して分析・議論を展開する。

本稿の構成は，以下のとおりである。本章で研究目的を明示したのち，2 章では組み込みソフトウェア開発の特徴を概観し研究背景・動機を述べる。3 章では本調査としてソフトウェア開発現場における問題点をインタビューデータによって分析・考察し，問題を解決するための支援方法・ツールを提案する。4 章は，本研究全体をまとめ，今後の課題を示して本稿をむすぶ。

2　組み込みソフトウェア開発の特徴

ソフトウェア技術が社会基盤に浸透した一番の要因は，組み込みシステムの普及によるものである。組み込みシステムとは，装置の中に組み込まれたコンピュータシステムのことを指し，いわゆるパソコンやサーバのような形をしていないコンピュータということができる。そのため，日常生活において「これはコンピュータが中に入っている」と意識することなく，コンピュータ制御によって高度な機能を実現している工業物は身の回りに多く存在している。2004年調査の段階で，組み込みソフトウェア産業に従事する技術者は15万人以上，その生産高は 2 兆円を上回る産業規模になっており，組み込みソフトウェア産業は現在の日本経済の一翼を担っている（平山2004）。

組み込みシステムは，大きく「情報系」と「制御系」に分けることができる。情報系とは，携帯電話や情報家電に代表される一般エンドユーザ向けのシステムである。情報系システムの特徴は，より汎用機能が必要とされることや省電力であることがあげられる。一方，制御系組み込みシステムはFA (Factory Automation) 機器や車載システムなど，リアルタイム制御や高信頼性が特徴である。その意味で，組み込みシステムは，システム・制御技術を製品に具現化した場合の一形状ということができる（中本2007）。そして，それら組み込みシステムを動かしているのが，組み込みソフトウェアである。

組み込みソフトウェアの特徴としては，マイクロコンピュータなどを利用し製品に組み込

まれて動作することから，(1) 周辺機器（ハードウェアデバイス）と連携して機能を実現する，(2) システムの外部状況をセンサーなどから受け取り，それに応じて処理内容が変化する（つまり実世界と相互作用する），(3) 処理時間やリソースに厳しい制約がある，(4) 製品に内蔵されるマイクロチップやＯＳは様々な種類が提供されている，(5) 使用環境や安全性に対する厳しい制約がある，などがあげられる。特に，ハードウェアデバイスと連携して機能を実現することから，ソフトウェア技術者とハードウェア技術者が協同で開発に取り組まなければならず，異分野技術者間のコミュニケーションが必要不可欠である。そして，これらの特徴の一つ一つが，組み込みソフトウェア開発過程に大きな影響を及ぼしており，その開発を困難なものにしている。それに加えて近年は，組み込みソフトウェアの開発規模が加速度的に増える一方で開発期間は短くなる傾向にある。その結果，現場では絶えず混乱が続くような開発も少なくない（平山2004）。

ソフトウェア開発に関する問題点や支援方法については，これまでも，ソフトウェア工学を中心に指摘・議論されている。中心的な議論はソフトウェア危機及びソフトウェアの部品化・再利用である。ソフトウェア危機とは，開発されるソフトウェアの規模が指数関数的に増加していることや，それに伴って開発プロセスを管理することが困難になること，更にはプログラマを含むソフトウェア技術者の慢性的な不足等の理由で，多くのソフトウェア開発が途中で頓挫してしまったり，成果物の管理メンテナンスができなくなってしまったりするような危機的事態のことを指す（高橋1969）。これらのソフトウェア危機を乗り越える一つの方法として，ソフトウェアの再利用性を高める手段が検討されてきた。その中の一つの成果がオブジェクト指向である。そして，オブジェクト指向を特に再利用性から支援するデザインパターンやUML (Unified Modeling Language) なども，広まりつつある。しかしながら，ソフトウェアの適用分野によってはオブジェクト指向によるモデル化が困難であったり，UMLやデザインパターンを実務で再利用できるレベルにまで使いこなすために一から学ぶことが困難であったり，などの理由から，ソフトウェアに携わる技術者の現実的な支援にはなっていないことも少なくない。例えば，工学分野では中心的な役割を担いつつある制御用組み込み型ソフトウェアでは，タイミング判定，割り込み処理が非常に多く，これらをUMLでモデル化し記述することは，非常に困難であるといわれている。また，制御要求の内容が高度になると，制御ソフトウェア自体が大規模・複雑化してしまうため，技術者一個人がソフトウェア全体を把握することも困難になってしまうことが指摘されている(Ohsuga et al. 2000)。

コンピュータ制御が社会に出る以前の機械制御ならば，実物を見て触って経験を積むという作業を何度も繰り返すことで，技術や知識を伝承することが可能であったが，ソフトウェアが制御の中心的存在になると，そのような手法では伝承することができない。その理由としては，まず，ソースコードを読んだだけで全体の制御機構を理解することは非常に困難であることがあげられる。特に，高度な制御を実現するためには莫大なソースコードを記述しなくてはならないため，その全体像を記述内容から判断することが不可能である。そして，

そもそもソースコードの記述には，適用分野間や技術者個人間で差異が大きく，他分野から異動してきた技術者や，他人の書いたコードを途中から突然任された技術者などは，多大な労苦を強いられることになる。結局，自分で一から作り直すことになる，という事態も少なくない。

また，ソフトウェアを設計する際の意思決定やその決定に利用された根拠・理由に関する情報，ノウハウと呼ばれるものは，大きな価値ある情報であるにも関わらず，これまでのソフトウェア開発体系では，うまく取り扱うことができていない。実際，これらの情報は設計者の頭の中に存在しているものであり，なかなか外には出ていない情報である。たしかに設計者の頭の中に存在する知識には，いわゆる暗黙知と呼ばれる，言葉には表すことができない個人的アナログ知 (Polanyi2003) も存在しているが，本研究では，必ずしも暗黙知のみが記述されていないのではなく，形式的に記述できる知識・情報であっても暗黙的になっているものが存在しているのではないかと考えている。ソフトウェア設計時における技術者同士の打ち合わせや議論の際には，言葉として発せられることはあるが，それらがノウハウとして再利用する方向を視野に入れた形で，再利用可能なドキュメント・データとして記録されていない。つまり，打ち合わせや議論時の会話には登場するものであるが，仕様書や設計書といった成果物には記述不要な情報という位置付けであるために，それらは記述されずに，技術者各個人の頭の中に留まったままの状態で，今日まで来ているのではないかと考えている。しかしながら，システムの大規模・複雑化と，それに伴うソフトウェア技術の影響力が社会的にも大きくなってきている現在では，ソフトウェアの設計・開発に関わる問題は，非常に重大な問題であり，早急に対策を施さなくてはならない問題である。現実に，ソフトウェアを原因とした様々な事故や不具合が発生し社会問題化している（銀行ATMのシステムや航空管制システムのダウンによって社会混乱が生じたことは記憶に新しい)。

このような現状に対して，何らかの支援策が必要である。それはソフトウェアのライフサイクル全体をカバーしたソフトウェア知識のマネージメントを，システム的側面と人的側面の両面から支援することである。ソフトウェア知識のマネージメントとは，単に管理することを意味するのではなく，知識の積極的な運用を可能にすることを意味する。具体的には，ソフトウェア知識をソフトウェア開発者から積極的に吸収・蓄積し，必要な場面で積極的に再利用を促すものであり，特に様々な技術者レベル間での知識の伝達・教育・伝承を考慮に入れた枠組みを指す。

本研究では，上述のようなソフトウェア知識を積極的にマネージメントすることを支援する枠組みを実現することを最終目標としている。本稿は，その第一歩として，ソフトウェア知識伝達・伝承時における不具合やトラブルを調査し，その原因を分析することにより問題点を明確にし，その支援方法・ツールを提案することを目的とする。

3 ソフトウェア開発現場における問題点の調査

本章では，現状において組み込みソフトウェア開発現場で問題になっていることを，インタビュー調査・分析した結果を報告する。特に，技術者間でコミュニケートされる情報・知識の伝達・伝承の場面に注目し，どのような工程で，どのような不具合・問題が発生しているのか，そして，それらを改善するためには如何なる支援が必要なのかを考察する。

3.1 仕様書によるコミュニケーション

現在，ソフトウェア開発の多くは，要求分析から設計・コーディング・テストといった一連の開発工程を，一人の技術者がすべて行っているということはない。それぞれの工程で，各専門技術者が担当し，かつ，大規模なソフトウェアに至っては多人数の技術者が担当工程・箇所を分担し，協同作業を通して一つの成果物を生成している。自動車制御に関するECUは，まさにその典型例であり，エンジン制御に関連する分野に絞ったとしても，エンジンの基本設計からソフトウェア設計・コード生成・適合テスト等，多くの異なった専門分野の技術者が協同で一つの成果物を作り上げている。

多くの技術者が一つの成果物に携わる場合，各技術者間での情報や知識の伝達が非常に重要な要素となる。一人の技術者が全工程を単独で進める場合には，誤解や誤謬など生じ得ないし，何らかの不具合や問題に遭遇しても，すべてその担当者の手の中・頭の中に存在するため，（問題解決が可能か否かは別として）当該担当者が原因を追求することは可能である。しかし，多数の技術者が携わっている場合は，このようにはいかない。各技術者が，同一の最終成果物を見ている（考えている）ことには違いないが，それぞれの専門分野によって，個人の興味・視点によって，全く違った側面を見ている場合がほとんどである。また，要求や機能をはじめとする仕様に関しては，仕様書を通じて誤解・誤謬なく伝達されることは非常に難しいことである。これは，専門技術・知識を次世代の技術者に伝承する場面においても同様のことが言える。また，同一の開発プロジェクトに携わっていなくても，類似したプロジェクトに携わっていたり，将来，成果物の一部を適用する必要性が生じた場合などにも，同様のことが言える。

一般に，ソフトウェア開発においては，知識の伝達には様々な仕様書や設計書が中心的な役割を果たしている。例えば，要求仕様書，実装仕様書，ソフトウェア設計書等がそれにあたる。これらは，ソフトウェア開発における各工程で生成され利用されるものであり，開発における上流工程から下流工程へと進むごとに，より詳細化され細分化されていく。そして，それらのドキュメントに載っている情報をもとに，ソフトウェアは設計・コーディングが進み完成して行く。しかし，ドキュメントに載っている情報のみでは理解できない場合や，誤解・誤謬が生じてしまうことが多々ある。そのために，数多くの打ち合わせ会議を開催し，

仕様内容やノウハウの確認から技術的すり合わせまでしているのが通常である。場合によってはソフトウェア開発工程全体において，これら会議や打ち合わせ時間の割合が非常に大きくなっていることもある。更には，誤解・誤謬が原因となって，納期が大幅に遅延してしまったり，開発プロジェクトそのものが中断してしまったりするような事態に陥ることもある。つまり，技術者間のコミュニケーションはソフトウェア開発の根幹であり，ソフトウェア開発の支援を考えた場合に避けて通れない必須要素である。

そこで，本研究では，これらの現状を把握するために，ソフトウェア開発に携わっている現場に対して調査を行った。調査対象としては，組み込み型の制御ソフトウェア開発を担当している技術者である。

3.2　調査方法

調査対象は，エンジン制御に関する組み込み型ソフトウェアの開発に従事している技術者12名であり，経験年数は 1 年〜15年であった。調査方法は，少人数グループによるインタビュー形式とした。グループの詳細およびインタビュー時間を表 1 に示す。

表1：インタビュー対象者（グループ）

担当業務内容	人数	インタビュー時間
エンジン制御標準部品（32bit）開発	2	2時間10分
エンジン制御（16bit）開発	4	2時間23分
プラットフォーム開発	4	1時間30分
ダイアグノシス開発	2	1時間30分

グループによって，携わっている内容が異なるために，不具合やトラブルの種類が異なる可能性があると考え，このようなグループに分けて，それぞれのグループごとにインタビューを行った。そのため，インタビューに参加しているのは日常顔を合わせている同じ部署のメンバー同士なので，初めて顔を合わすような馴染みの無い研究者が一人の技術者に対して質問するという形式に比べ，技術者同士が気軽に会話するような状態でインタビューに回答できるように配慮した。それによって，多種類かつ詳細な内容を，そして聞き手が予想もしないような内容まで聞き出すことができた。

各グループに対するインタビュー時間は 2 時間という予定で行ったが，技術者グループが全て言い尽くしたと感じるまでインタビューを続けた。各グループのインタビュー時間に若干の差はあるが，全体として，集中した質の高いインタビューを実現できた。

インタビューの質問内容は，現状の開発現場で，「プログラム開発における困難点・不具合事例と対処・支援方法」，「現在の技術者間コミュニケーションに不足している情報」という 2 つの観点について質問し，それに対する回答を技術者同士が自由に会話する形でコメント

してもらった。

インタビュー中の発話内容はすべて録音しておき，インタビュー終了後，すべて書き起こし発話記録データとした。この発話記録データを用いて，分析・考察した結果を以下で述べる。

3.3 調査結果と考察

インタビュー時に質問した 2 つの観点「プログラム開発における困難点・不具合事例と対処・支援方法」と「現在の技術者間コミュニケーションに不足している情報」ごとに発話記録データを分析した。その結果，「プログラム開発における困難点・不具合事例と対処・支援方法」については 9 つのカテゴリーに，「現在の技術者間コミュニケーションに不足している情報」については 3 つのカテゴリーに分類できた。以下，それぞれのカテゴリーごとに考察を述べる。

3.3.1 プログラム開発における困難点・不具合事例と対処・支援方法について

プログラム開発における困難点・不具合事例と対処・支援方法という観点からコメントしてもらった発話記録データを，コメントの要点をまとめ，分類した。その結果，「①文字（日本語）だけでは判りにくい」，「②不足情報が多い」，「③仕様書の記述レベルのばらつき」，「④仕様出しをする際のチェック機構の不備」，「⑤標準部品化による影響」，「⑥システム設計・実装時における困難」，「⑦ノウハウに関する困難」，「⑧静的なチェックだけでは不十分な点が多い」，「⑨その他」という 9 つのカテゴリーにまとめることができた。ここでは，カテゴリーごとに，主なコメントの要点を箇条書きで示し，考察を述べる。

①文字（日本語）だけでは判りにくい

・「誤解・誤謬が生じてしまう」
・「日本語の曖昧性が問題につながる」
・「最終的にどのような動きをするのか判断できないことがある」
・「日本語だと抜けが生じてしまう」

現状では，主に仕様書というドキュメントで必要な情報や知識を伝達することが通常となっている。また，ドキュメントも作業工程ごとに異種のものが用いられている。たとえば，要求仕様書，ソフトウェア設計書，テスト仕様書などがそれにあたる。そのため，各工程の技術者は，ほとんどの情報・知識を，そのドキュメントに書かれている文字から得る。その際に，誤解・誤謬が発生している。この点を本研究では，各工程により専門としている分野

が異なっているために，そこで用いられる用語や表現の違いが原因であると考えている。自然言語には，その性質上，曖昧性がある。実際，会話をしていても誤解が発生してしまうことは，我々は日常的に経験していることである。しかしながら，技術開発等の分野での誤解・誤謬は製品の不具合に直接繋がり，その結果，製造コストの面で大きな損失になるだけでなく，場合によっては，大事故や大惨事につながる可能性もある。つまり，技術開発においては，誤解・誤謬は，基本的にあってはならないものである。そのため，何らかの手段により，誤解・誤謬を防いだり減らしたりすることを考えなくてはならない。

対処・支援の方法として考えられるのは，仕様書を中心としたアプローチである。まず，仕様書を用いて情報・知識を正確に伝達するためには，どんな内容をどのような形式で記述するのがよいのか，しっかりと分析することが必要である。その分析には，認知科学的な手法や心理学・人間科学的な要素も十分考慮に入れるべきだと考える。その分析結果をもとに，仕様書の標準化を行い，それに基づいた仕様書テンプレート（仕様書記述支援エディタ等）を準備すれば，仕様書を記述するための支援になると考えている。つまり，仕様書を記述する側を支援することが，仕様書を理解する側をも支援することに繋がると考えている。また，自然言語以外での記述も必要である。例えば，タイミングチャートや状態遷移図等の各種チャート・図・数式等を用いることによって，自然言語の曖昧性を補完することが可能であると考えている。これまでも，様々なチャートは利用されてきているが，仕様書を記述する側が，チャートの必要性に気付かなかったり，場合によってはチャートを記述する手間をかけたくない等の理由で書かれないこともある。そのため，仕様書に記述する内容や項目によって，各種チャートが必要な部分に関しては必ず記述する標準化（規範）作りや，それらの作業を支援することが重要である。

また，この対処・支援を，もう少し大きな視点から考察すると，技術者同士のコミュニケーションや協同・協調作業を支援する枠組みであると言える。つまり，ここで指摘した課題は，コミュニケーションや協同・協調作業の中で発生する問題の一部であることを常に念頭において取り組まなくてはならない。

何らかのトラブルや不具合が発生した場合などの場合も，報告書というドキュメントの形で残されている。そのため，この点に関しても同様に支援することを考えていく必要があると考えている。

②不足情報が多い

・「処理順序が不明確」
・「優先度（仕様・仕様書）が不明」
・「ダイナミックレンジが記述されていない（推奨ソースからも判断できない）」
・「明記されないノウハウが存在する」

上で述べた自然言語の曖昧性と関連するが，不足情報が原因で誤解をしてしまうというケースもある。例えば，処理順序が明確でない，処理の優先度が不明，ダイナミックレンジが記述されてない，等の意見があった。これらの不足情報は，単に情報を入れ忘れたということもあるかもしれないが，それ以上に，各工程の技術者間の背景知識や専門領域の違いによって生じている問題であると考えられる。例えば，要求仕様を記述する側としてみれば，ソフトウェアが全体として満たす要求・機能を中心に考えて，それを記述することとなるが，その際，ソフトウェアの各モジュール単位の振る舞いや，それらモジュールで必要としているデータマップに対して考慮することは，ほとんどないであろう。しかし，ソフトウェアを実際にコーディングする側から見てみれば，モジュール間の値の受け渡しや各モジュールの振る舞いから，モジュール内の関数の優先度やそこで用いられる詳細な情報・マップのデータなど，いわゆる上流工程では考慮されない情報の方が，作業には必要である。それらの情報を正確に，かつ，過不足なく与えてもらえないと，開発に大きな負荷がかかるだけでなく，場合によっては開発工程全体をやり直すことが求められることもある。つまり，同じソフトウェア開発を目標としていても，各工程の技術者によって視点はばらばらであるために，各工程間で過不足なく情報・知識を提供・共有することは，容易なことではないのである。

これらの課題に対しても，先に述べた仕様記述の標準化および，その支援の枠組みが役に立つ。記述すべき内容を標準化することで，これまで注目・意識していなかったような情報も漏れなく記述することは，単に情報の不足を防ぐだけでなく，新しい発見も期待できる。従来，特に気にとめなかったような関連情報・データに，思わぬヒントが隠れていた，という意見はよく耳にするが，同様の効果が得られる可能性もあるということである。開発工程の上流から下流までシームレスに情報が受け渡されることにより，新しい切り口が見つかったり，同時に技術者間のコミュニケーション支援にも役立つと考えられるからである。また，一般にノウハウと呼ばれる，ドキュメント化（外化）されていない知識・情報も，標準化により記述（外化）される可能性がある（本稿で言うノウハウは，言語化や外化が不可能な暗黙知と呼ばれるものを指すのではなく，限られたドメインやソサイエティ内で暗黙的な了解として存在しているがドキュメント化されていない情報・知識を指す）。後でも述べるが，ノウハウは，技術開発や製造の分野では中心的な役割を果たしているにも関わらず，言語化（外化）することが容易でないこと等のために，なかなか整理されていない知識・情報と言う事ができる。そのため，各分野の各工程で，ノウハウを逐次記録し蓄積することは，非常に意義深いことである。そして，本研究の大きな目標である，知識マネージメントの実現への鍵になるものの一つである。

③仕様書の記述のばらつき

・「仕様書を書く人や部署によって完成度レベルが大きく違う」

・「結果オーライ的な記述の仕様書もある（例：この機能があれば何でも良い）」

・「仕様書の書き手に都合の良い書き方をされている」
・「仕様記述が統一されていない」
・「設定条件がばらばらに書かれていたり，書かれていなかったりする」

人間が仕様書を記述する以上，人間による記述のばらつきは仕方ないことである。しかし，技術開発に携わる場合には，それでは済まされない。特に，仕様書は，その性質上，必要不可欠な情報・知識を記述するものなので，完成度レベルは均一である必要がある。

しかしながら，インタビューの結果からは，仕様書レベルの差が担当部署や担当者によって大きいことが指摘されている。原因としては，いくつか考えられるが，大きなものとして，仕様書記述に関する標準化が完全になされていない，という点が指摘できる。記述するための規定（ガイドライン）が定められていることと考えられるが，その規定通りに記述しているかどうかのチェックはされているのだろうか。また，その規定は，記述する側と読解する側の双方から意見を取り入れて決められたものなのだろうか。規定の見直しを定期的に行い，時代による技術背景に適合するように改訂がなされているのだろうか。現状の仕様書がその役割を十分果たすことなく，単なるドキュメントの一つとして存在しているにすぎない部分があることは否定できない。特にソフトウェア開発では，ソフトウェアのプログラムコードのみが成果物として判断・評価される場合が多いために，仕様書が軽く見られ，その結果，仕様書の記述レベルのばらつきに繋がっていることと考えられる。

④仕様出しをする際のチェック機構の不備

・「誤記が多い（変数名，綴り）」
・「処理で使うデータがない」
・「ライブラリがない」
・「部品ナンバーの指定が間違っている」
・「セットで使わなくてはならない部品が明記されていなかったり，片方が間違っていたりする」
・「リリースされていない部品が指定されている」
・「タイプミスなのか，わざわざ変えて書いているのか判断できない」
・「似たような変数名の場合，新しい変数なのか誤記なのか判断できない」
・「ソースと仕様書の両方とも間違っていることがある」
・「推奨ソースよりも仕様書記述に関する不具合の方が圧倒的に多い」
・「同じECUに載る仕様書でも，ある部署からは32bit仕様書，他部署から16bit仕様書がくることがあって，どっちに合わせるのが良いのか考えるのも大変」

この問題は，組織に関わる問題ということができる。仕様書をはじめ様々なドキュメント

は，人間が書くために，どうしても誤記や内容の不備等，様々なミスが存在してしまう可能性に富んでいる。それらをカバーする一方法として，仕様を外へ出す前にチェックするということがある。実際，チェック機構は存在するものと思われるが，必ずしも有効に機能していない，という意見が多くあった。仕様書を読む立場として，誤記をはじめとするいくつかの仕様書の不備は，チェックがしっかりとなされていれば発見できるものだということである。実際にチェックはしたが発見できなかったためなのか，それとも，チェック機構が働いていなかったのかは不明である。

ここでの支援をするためには，仕様書のテンプレートを含めた仕様記述の標準化を押し進めることが，まずは必要である。その後，その記述ルールに従って，必要事項が記述されているのかをチェックする仕組みが必要になる。実際に，どのようにチェックを進めるのかは，仕様記述ルールに依存すると思われる。

⑤標準部品化による影響

・「部品の細分化により，一つの仕様書では機能全体が把握できない」
・「複数の条件がからみあっている場合，仕様書だけでは理解不能」
・「前後関係や処理タイミングの記述間違いは，仕様書だけではわからない」
・「どこで宣言すればよいのかは，仕様書とソースを見比べただけでは分からない」
・「仕様書上のタイミングは正しいが，定数等で狂ってしまうこともある。このような不具合を仕様書から見つけるのは難しい」
・「ドキュメント量の増大のため，使い勝手が悪く，理解することも困難」

近年のソフトウェアは，その多くが既存のソフトウェアに改良を加えて行くことで開発が進められている。その場合，ソフトウェアを細かい単位に分けて部品化することによって，再利用性を高めようという試みが，様々な場で試されている（MISRA-C研究会2006）。このような，ソフトウェアの標準部品化は，ソフトウェア知識の蓄積・再利用という観点からは非常に重要なことである。

しかし，ただ部品化しただけでは，非常に使いにくいものになってしまうこともあるということがインタビュー結果から判断できる。例えば，全体の機能の把握が困難であったり，複数条件の場合に内容を理解することが困難であったりするなど，仕様書を読んだだけでは把握できないことが多く発生している。また，仕様書等のドキュメントも，部品が多くなるに従って増大するため，使い勝手も悪くなることは容易に想像できる。

そこで，仕様内容を把握・理解するための支援や，各部品を開発・利用している技術者間のコミュニケーションを支援することが重要になると考えられる。ただし，それらを実現するためには，標準部品をしっかりと整理しておくべきである。インタビュー結果からは，まだ，整理も不十分であると思われるような意見もあった。例えば，「指示通りの標準部品を用

いたがまったく動かなかった」というような事態もあったという。これらの点からも，標準部品をソフトウェア知識として，しっかりと整理しながら（動作チェックもしながら）利用を進めなくてはならない。

⑥システム設計・実装時における困難

・「ノウハウを意識しながらの設計」
・「実装用ルールに従ったコーディング」
・「ルールの検索・把握」
・「ラベル付け（自動化できないか)」
・「特許の制約（警告して欲しい)」
・「法規の制約（警告して欲しい)」

⑦ノウハウに関する困難

・「エンジン開発段階におけるノウハウが仕様書に明示的に書かれていない」
・「標準部品や過去の部品の評価実績」
・「間接的・暗黙的に変更点が書かれている」
・「ある箇所を変更すると他の部分も変更しなくてはならないという場所をすべてつぶすことが大変。これらは必ずしも仕様書に書いてある情報ではない」
・「仕様変更はないが，ソースは変更しなくてはならない事態がある。ベテランは気付くかもしれない。プログラマのレベルによってソースに差が出てしまう仕様書は，仕様書として不十分」

⑥と⑦については，強く関連した内容であるので，ここでまとめて考察する。

システム設計・実装に関しては，ソフトウェア技術者（プログラマ）の経験や能力が大きく影響してくる面と言うこともできる。例えば，ノウハウと呼ばれるものは，その多くが経験知として使われているものが多い。そのノウハウを，システム設計や実装時に，無意識に考慮に入れることができる技術者もいるが，そうでない技術者もいる。このような状況を，技術者個人の勉強不足とか能力不足という言葉で済ましてしまうことも可能であるが，組織としてのノウハウや知識マネージメントという枠組みを考慮に入れた場合には，そのように済ませてしまうことはできない。どんな組織にも，当然，ベテランもいれば新人もいる。ある特定分野で長けていても，少しでも特定分野からずれてしまうと，まったく素人同然という状況もある。特に，システムの大規模・複雑化が進むことを考えると，今後は，各個人が把握できる領域は，システムの中の非常に限られた範囲になってしまうことも十分予想できる。このような場合において，異動等により，これまでの担当領域とは多少違う領域を担当するようになった場合，たとえ勤務年数が多いベテラン技術者であっても，上で述べた新人

や素人と同様の状況に陥ってしまうことがあるだろう。

つまり，設計や実装上に関する知識やノウハウを，何らかの形で記述しておく必要があるのではないか。そして，その記述を読んだ技術者が容易に理解できるような記述や，理解支援システムの一面として学習支援という機能も積極的に取り入れて行く必要がある。そのためにも，ノウハウと呼ばれている情報・知識を，なるべく技術者に負担をかけないように収集し，それらを再利用しやすいように整理することが必要である。また，ノウハウの妥当性や評価も十分検討すべきことである。例えば，「理由は不明だが，やってみたらうまくいったから，これはノウハウとしよう」ということは，少なからず耳にするが，このような類のノウハウは，非常に危険である。非常に限られた状況で偶然うまくいっただけなのか，それとも本質的に正しいのか，多数のエラーが複合して発生し不具合をキャンセルしてしまった結果として正しく動いているように見えるだけなのか，等いろいろと不安要素が広がっていってしまう。従って，ノウハウを共有して再利用を進めようと考えている場合には，一度，ノウハウと呼ばれている情報・知識を，しっかりと評価するべきである。そして，それらの再利用を支援してくれるようなシステムを構築するべきである。

このような観点から支援方法を考えると，ノウハウを記述するためのテンプレートや，通常の作業時に，あまり意識することなくノウハウを蓄積できるような作業環境を準備することが重要である。つまり，様々な情報・仕様・知識等を保持しつつ提供し，かつ，その環境内で開発作業の全工程を支援でき，作業中に発生したノウハウの記述も支援できるとよい。また，ノウハウを再利用する際の支援も重要である。そのためにも，ノウハウの検索機能や理解支援機能も強化しなくてはならない。

⑧静的なチェックだけでは不十分な点が多い

- 「標準部品を仕様通りに組み合わせてもうまくいかないことが多い（部品の不整合）」
- 「分野間をまたがっての不具合は発見することが困難」
- 「Cソースで閉じた範囲で不具合が出ることは少ないが，それらを組み合わせることで発生することが多い」
- 「ロジックの検証のみでは発見できない不具合がある（実際に部品同士を結合してみないと検証できない）」
- 「標準部品の整合性チェックに莫大な工数がかかっている」
- 「通常温度領域では問題ないが，低温領域では不具合が発生（実際に動かしてみてはじめて気付く。現実的に，マイナス20，30度は簡単にテストできない。）」
- 「静的なシミュレータだけでは気付かないことが多い（例：エンストなど）」
- 「ピンポイントで発生する不具合もある（例：10度の時だけ不具合発生）」
- 「不具合が発見されないと，間違いが継承されてしまう」
- 「エラーが起きれば解決するが，起きなければ分らない部分もたくさんある」

・「工場に送ってから不具合ということもある」

ソフトウェアには，潜在的にバグが存在する。そのため，デバッグを含むソフトウェアのテストは必要な作業である。特に，本研究で対象とした自動車用ECUなどの組み込み型ソフトウェアでは，ソフトウェアの範囲で完結しているものではなく，ハードウェア本体を含め各種センサーや環境要素等の影響を含めて考えなくてはならない。そのため，必ず実機に載せた適合作業を含め，動的テストが必要なのは当然である。しかし，ここでのインタビュー結果として述べていることは，そのようなレベルのものばかりではない。例えば，部品の不整合，ロジック検証のみでは発見できない不具合，がこれにあたる。

このような事態に対する解決策としては，まず，ランタイムチェックとして，シミュレータを用いて動かしてみて，その動作を見ることが考えられる。そのためには，シミュレータは言うまでもなく必要だが，シミュレータで用いる各種テストデータを準備しなくてはならない。テストデータの生成は，一般的に困難な作業だと言える。なるべく，ソフトウェアのすべてのパスを通り，かつ，通常では考えられない不測の事態でも致命的な不具合を避けることが可能とするソフトウェアの頑強性も考慮に入れて，テストデータを準備しなくてはならない。そのため，テストデータの生成を，少しでも補助できるような支援機能があるとよい。ただ，このような要求を満たすテストデータを生成支援することは容易なことではない。

しかし，このようなテストデータも，ソフトウェア開発に関する一つの知識でありノウハウである。従って，テストデータを蓄積・共有・再利用が可能な支援システムがあれば，ある程度の助けにはなるのではないかとも考えることはできる。

⑨その他

・「開発色の強いものは双方でパラレルに進めるしかない」
・「仕様確認書によるやりとりが必要」
・「仕様書を信用できない」
・「仕様書だけで仕様内容を伝えるのは，伝言ゲームに近い（不確実)」
・「後付けの仕様書では駄目」
・「最終的には，電話やチャット，電子メールによるやりとりが必要」
・「知識のギャップで苦労（プログラマ経験はあっても，分野が違うと大変)」
・「分野をまたがって起こる不具合は，発見も対処も難しい（全体を見渡せる人が必要)」
・「推奨Cソースが不十分のため信用できない」
・「悪循環（ずっと上流設計に携われない)」

複数の技術者が携わる仕事においては，コミュニケーションが重要な役割を果たす。そのため，協同作業におけるコミュニケーションを支援することが重要である。特に，様々な検

討や変更の履歴をしっかりと保持し，適宜，提示してくれるような支援は，同じ間違いを繰り返さないためや，懸案事項を忘却・凍結させないためには有効である。

また，技術者間の情報・知識の伝達は，技術知識の教示－学習という関係で表現可能であり，教師から学生への教示と同様の図式で表せる。つまり，本研究の目標の一部分は，技術者の学習過程を支援することと言い換えることができる。

学習支援を考えた場合，有効な支援を実現するためには，教師は学習者の理解状況やこれまでの学習履歴を把握し，その学習者に適した教示を与えなくてはならない。そのためには，学習者モデルを支援システム内部に持ち，その情報から支援システムが教示情報や教示方法を判断して，学習者に教示を行う機能が必要である。これらは，知的学習支援システム研究という方向からのアプローチであり，本支援システムを考える場合には，これらの要素も十分に考慮にいれるべき要素である。

また，技術者のモチベーションの低下を防ぐための対策も必要であるという意見もあった。単なるコーディング作業のみではモチベーションの維持が困難であるということである。自分が担当しているソフトウェアの意味や位置付けなどを，ある程度まで理解できるように情報を提示すれば，これらの面は改善できるという意見であった。

3.3.2 現在の技術者間コミュニケーションに不足している情報について

本節では，現在の技術者間コミュニケーションに不足している情報という観点で，インタビュー結果をまとめた。現状では，技術者間コミュニケーションの中心は仕様書で行われているため，現在の仕様書に不足している情報をまとめたと言い換えることもできる。

分析の結果，不足情報として「①心の部分」，「②表示方法」，「③ノウハウ」というカテゴリーに分類できた。それぞれのカテゴリーが抽象的ではあるが，その特徴をよく表している。各カテゴリーについて，主なコメントの要点と，考察を以下にまとめる。

①心の部分

・「制御意図がわからない」
・「動作（最終的な動作，途中の動作）が不明」
・「変数・定数の意味がわからない」
・「制御変数のダイナミックレンジが不足」
・「優先度・順序情報が不明」
  ▶関数のコール順
  ▶処理順序
  ▶仕様・仕様書の優先度

ここで「心の部分」と呼んでいる内容は，主に，ソフトウェアの制御意図や目的，変数・定数の意味，また，各種優先度を指している。これらの情報は，実際にソフトウェア開発の上流工程から携わっている場合には，当然，理解している内容である。しかしながら，本稿で何度も述べているように，ソフトウェアの大規模・複雑化のために，ソフトウェアの開発工程が細かく分類され，それぞれの工程で別々の技術者が担当するようになってしまっているために起きたことである。

では，このような「心の部分」は，プログラミング（コーディング）する際に，絶対に必要かと言えば，そうでもない。仕様書がしっかりとできていれば，その通りにコード化することは可能である。実際に，そのように割り切ってコーディングを行っている技術者もいた。しかし，このように進めていると，不具合が発生した場合に対処が困難であるという。プログラミングという作業には，当然，デバッグは必要な作業であるが，その際に，「心の部分」が必要になる。結局，上流工程の技術者に問い合わせたりして，意図を問い直す必要が生じる。

また，「心の部分」を一切考えずにコーディング作業を続けていると，モチベーションが非常に低下する，という意見もあった。これは，技術者個人の問題であるかもしれないが，意味を考えることなくコード化することは，たしかに困難な業務であるということは十分想像できる。特に，組み込み型ソフトウェアのようなものを対象としている場合には，その全体像を理解することなく進めてしまうと，場合によっては，非常に深刻な問題を抱える原因になってしまうかもしれない。

②表示方法

・「図表をつけて欲しい」
・「タイミングチャートをつけて欲しい」
・「フローチャートをつけて欲しい」
・「状態遷移図をつけて欲しい」
・「グラフィカルな動作表示があるとわかりやすい」

これは，前節の考察にもあるように，文字（日本語）だけでは判りにくい，ということである。状態遷移図，フローチャート，タイミングチャート，等の様々な表示が欲しいということであった。もちろん，これらのチャートが添付されている仕様書もあるので，現在の仕様書すべてが抱えている問題とは言えない。

ただし，技術者個人によって，要求している表示方法に違いがあることが，今回のインタビューによって判った。例えば，技術者Aは「状態遷移図があればよい」と言ったが，技術者Bは「状態遷移図よりもタイミングチャートが欲しい」と言った。担当している箇所にも依存するとは思うが，技術者に固有な理解しやすい表示方法というものもあるとも考えられ

る。そのため，様々な表示方法を備えた支援システムがあると，これらの要求に対処できると考えている。

③ノウハウ

・「開発・設計段階に検討した内容を知りたい」
・「外部との関連性や影響の情報が欲しい」
　▶他の仕様書や仕様内容とのリンク情報
　▶関連する影響範囲項目の一覧
　▶各標準部品間の整合・不整合情報
・「変更履歴情報が欲しい」
　▶変更理由・意図
　▶バグ情報・デバッグ情報
　▶不具合情報・不整合情報
　▶変更が及ぼす他箇所への影響
　▶関連項目の一覧
　▶差分情報（追加分・変更分のみ）
・「事例情報があると良い」
　▶成功事例
　▶不具合事例・失敗事例
　▶テスト実績情報
　▶デバッグ情報
　▶デバッグのための関連知識情報
　▶最終形（ソース）になるまでのプロセス情報
　▶評価・妥当性検討の際の判断理由

ノウハウに関しては，先にも述べたが，非常に重要な情報であることには間違いない。ものによっては，ほとんどノウハウの塊だと呼ばれることもある。しかし，それらの情報は，技術者個人の頭の中にのみ存在しており，なかなか外に出てこない。

例えば，設計・開発時に検討した内容などは，会議中には言葉となって発せられるが，それが記録されていないために，その会議に参加していない技術者には伝わらず，それが原因で不具合が発生するというケースは非常に多い。また，様々な事例情報も，様々な技術者が利用できるようになっていればよいが，実際は，単なる形式的な記録にすぎず，同じ失敗を何度も繰り返したり，同じ問題解決を複数の箇所が別々に取り組んでいたりして，後になってから，それに気付くということも非常に多い。

このような事態に対処するためには，ノウハウに関する整理を通常から進めていなければ

ならない。ただ，問題は，これらの整理に対して負荷をいかに減らすか，ということである。ノウハウが発生する状況とは，目の前の問題解決に全力を注いでいる状況であり，技術者の叡智が結集している場面といえる。そのような場面で，ノウハウを技術者の思考の負荷にならない程度で整理を促すことは容易ではない。そのため，何らかの支援策が必要である。

また，ノウハウを積極的に提示して，再利用を促すことも重要である。実際，技術者が現在抱えている問題が，過去に起きた事例と同じなのかどうかに気付かない場合がほとんどであるという。偶然，他の技術者と会話しているときに，その問題は以前あった問題と同じものであるという指摘を受けたり，不具合を報告した時に，同様に指摘を受けたりするということが多いとのことである。上で述べたとおり，ノウハウがしっかりと頭の外に出ていないということと，再利用される形で記録されていないということからも，これらの事態は起こるべきにして起きていることを表している。

3.4　問題点の整理と支援方法

前節で，組み込みソフトウェア開発現場における技術者の生の声をもとに問題点をまとめ，各項目に関する支援方法を考察した。本節では，前節で項目ごとに考察した内容を，全体として整理し，今後の改善・支援方法について提案する。

まず，前節の考察をもとに各開発工程における支援内容・ツールについて表 2 にまとめた。現状では，開発者・技術者の根性と努力では問題解決が困難になってきており，支援ツールへの要求が高まっていることが，現場からの生の声として確認できた。表 2 で示した支援内容・ツール全体を統合したものが，本研究が最終目標としている支援システムになると考えている。

また，技術者間コミュニケーションの中心は仕様書であるという現状を示したが，同時に，その仕様書周辺に問題が山積していることも明らかになった。特に，組み込みシステム開発の現場では，ハードウェア技術者とソフトウェア技術者のような異分野の技術者間コミュニケーションや，大規模ソフトウェア開発のために多人数ソフトウェア技術者間でのコミュニケーションが必要であり，そのコミュニケーションの中心的役割を担う仕様書は技術開発の根幹である。現状では，仕様書に不足している情報を補完するために，電話や会議での質疑応答・技術的なすり合わせが必要不可欠となっている。開発ソフトウェアがますます大規模になる将来においては，これら技術者間コミュニケーション，即ち，仕様書周辺の問題は，かなり深刻になるものと考えられる。つまり，仕様書周辺の問題を解決することが最優先に対処すべきことである。

前節で，仕様書に不足している情報として，「①心の部分」，「②表示方法」，「③ノウハウ」というカテゴリーを示した。特に，「①心の部分」と「③ノウハウ」は，それ自体が技術であると言ってもよいほど重要であるにもかかわらず，暗黙的な情報・知識として技術者の頭の

表2：各開発工程の支援内容

開発工程	必要な支援内容・ツール
設計・仕様書記述	**システム設計支援** ■　ラフシミュレーションツール ■　ノウハウ整理ツール **仕様書の標準化・記述支援** ■　仕様書テンプレート ■　使用内容チェックツール
仕様書理解	**仕様書の理解支援** ■　視覚化ツール ・　フローチャート，タイミングチャート ・　状態遷移図 ■　関連情報へのリンク ■　各種DB連携 ・　変数・定数DB ・　評価実績DB ・　部品の整合性DB ・　成功・不具合事例DB ・　履歴情報DB（バージョン）
実装（デバッグ含む）	**コーディング支援** ■　高機能エディタ ■　実装ルールチェック ■　特許情報チェック ■　法規情報チェック ■　履歴情報管理（バージョン管理） **テスト・デバッグ支援** ■　ランタイムチェックツール ■　テストデータ生成ツール ■　デバッグフォロー表 ■　静的・動的シミュレーションツール
その他	**知識マネージメント支援** ■　知的財産の共有・再利用支援ツール ■　学習支援ツール

中だけに存在し，ドキュメントに残りにくいことが確認できた。本稿では，これらの情報・知識（「①心の部分」や「③ノウハウ」）を暗黙的形式知と呼び，その重要性を主張するとともに，それらを外化することはできるはずであると提案する。一般に，「暗黙知（知っていても言語化できない個人的・経験的・身体的なアナログ知）」と「形式知（言語化できる明示的なデジタル知）」という２つの知で知識伝承・創造は議論されることが多い（野中1996）が，本調査で明らかになった仕様書の不足情報は，形式知（言語化できる情報・知識）であるが暗黙的になっているために知識の伝承・創造に支障をきたしているといえる。実際に，「①心の部分」や「③ノウハウ」は，技術者が不具合やトラブルに遭遇した時に他の技術者（設計者やベテラン技術者）から得て解決している。ただ，現状の開発体制では仕様書や報告書等

に記述されない（外化されない）ことが多く，結果的に暗黙的情報であったり，暗黙的知識になっていたりする。これら暗黙的形式知を，技術者の頭の中からいかに外化するか，ということが今後の技術継承・発展にとって非常に重要な要件となるであろう。その解決策の一つが，先に示した各種支援ツールの実現であると考えている。技術者の頭の中から，暗黙的形式知を技術者になるべく負担かけることなく外化させ，それらを他の技術者が適切に再利用できる仕組みがあれば，本調査でコメントされた不具合・トラブルの多くは解決されるのではないかと思われる。その仕組みを現実化させるための具体的方法は，今後の課題として検討をしなくてはならないが，早急に取り組むべき課題であると認識している。

4　おわりに

本稿では，近年，大規模・複雑化している組み込みソフトウェアの開発工程について，技術者へのインタビューによって，現実に発生している不具合や問題点を明確にし，その支援方法・内容を考察した結果を報告した。特に，技術者間のコミュニケーションに注目して分析した結果，現状では技術者間のコミュニケーションの中心は仕様書にあること，その仕様書周辺に問題が山積していることが明らかになった。

具体的には，技術者が開発現場で困っていること（不具合・トラブルの原因）として，「①文字（日本語）だけでは判りにくい」，「②不足情報が多い」，「③仕様書の記述レベルのばらつき」，「④仕様出しをする際のチェック機構の不備」，「⑤標準部品化による影響」，「⑥システム設計・実装時における困難」，「⑦ノウハウに関する困難」，「⑧静的なチェックだけでは不十分な点が多い」，「⑨その他」，の 9 つのカテゴリーにまとめ，各支援方法を提案した。

また，技術者間コミュニケーション（仕様書）に不足している情報として，「①心の部分」，「②表示方法」，「③ノウハウ」をカテゴリー化した。特に，「①心の部分」や「③ノウハウ」の重要性を確認することができた。それらは，通常の仕様書では欠落してしまう場合が多く，いわば暗黙的情報であったり，暗黙的な策になっていたりする。しかしながら，それらは決して言葉に表せない（形式的記述ができない）情報・知識ではなく，記述する機会が十分与えられない情報・知識であると考えた。これらを本研究では，暗黙的形式知と呼び，その重要性を確認し，暗黙的形式知の外化・再利用が，知識マネージメントを実現するための鍵であると提案した。

本稿で分析した結果をもとに，各工程における支援内容・システムの詳細を整理し，それらを統合することにより，開発工程全域およびソフトウェアのライフサイクルをカバーする知識マネージメント支援方法を具体化することが，今後の課題である。

今後も，組み込みシステムの社会への浸透が進むのは明らかである。ユビキタスコンピューティング（坂村2006）によって，日常生活の様々な場面で組み込みシステムがますます活躍する時代も遠くないことと思われる。組み込みソフトウェアへの要求・依存度はます

ます増加するであろう。このことからも，本稿で考察した支援システムの研究開発とその実現は，早急に取り組まなくてはならない課題であるといえる。

参考文献

二上貴夫 (2004)， 組み込みソフトウェア開発支援ツールの動向，情報処理，Vol.45, No.7, pp.704-708.

平山雅之 (2004)，組み込みソフトウェア開発の現状，情報処理，Vol.45, No.7, pp.677-681.

柿崎成章 (2002)，エンジン制御開発における自動コード生成技術の適用, 計測と制御, Vol.41, No.2, pp.147-150.

MISRA-C研究会 (2006)，組込み開発者におくるMISRA-C 2004, 日本規格協会.

中本幸一 (2007)，今後10年の組込みシステム，システム/制御/情報，Vol.51, No.9, pp.375-379.

中島震 (2004) , 組み込みソフトウェアのモデル検証技術入門, 情報処理，Vol.45, No.7, pp.690-693.

水口大知，渡邊宏 (2005)，組み込みソフトウェア開発におけるモデル検証の適用事例，コンピュータソフトウェア，Vol.22, No.1, pp.77-90.

野中郁次郎，竹内弘高 (1996)，知識創造企業，東洋経済新報社.

Ohsuga S., Chigusa S., and Sugaya K. (2000) ,“Model Based Program Specification and Program Generation -In a Case of Vehicle-Engine Control System Design-”, Advances in Artificial Intelligence, PRICAI 2000 Workshop Reader, Springer-Verlag, Berlin.

Polanyi, M. (2003) , 暗黙知の次元,（高橋勇夫 訳）, 筑摩書房（原書名：The Tacit Dimension）.

坂村健 (2006) , ユビキタスでつくる情報社会基盤, 東京大学出版会.

佐野範佳 (2002) , 制御系CADベースの組み込みソフトウェアの開発環境の現状と動向, 計測と制御, Vol.41, No.2, pp.129-133.

高橋英俊 (1969)，ソフトウェア危機，情報処理，Vol.10, No.6, pp.373-374.

The 2007 A. M. Turing Award (2007) , ACM Turing Award Honors Founders of Automatic Verification Technology, (http://www.acm.org/press-room/news-releases/turing-award-07/)