NISSAN TECHNICAL REVIEW

日産技報

2019 No.84

ソフトウェア
Software

NISSAN MOTOR CORPORATION

2019

日産技報

NISSAN TECHNICAL REVIEW

No. 84

日 産 技 報 第 84 号

目　　次

2019年3月　発行

Nissan Technical Review No. 84 (MARCH 2019)

CONTENTS

“ソフトウェア”は自動車技術者の新しいリテラシー

フェロー　豊増　俊一

37年前の私は「トランジスタ技術」誌を片手に、オペアンプ加算回路や絶対値回路を駆使して車両熱負荷を演算し、その結果を基に温調ドア開度やファン速度の制御を行っていました。しかし、このアナログ回路設計は排反事象や複合事象を解くには限界がありました。

まもなくクルマもZ80系マイコンの採用を開始しました。8ビットでメモリ容量は僅か32KBでしたが、割込み処理などの命令を理解することでそれまでに無い自由度の高い設計が得られました。その後、一気にディスクリート回路はマイコンとソフトウェアに置き換わりました。それは同時に、ソフトウェア・バグとの戦いの始まりでもありました。

ソフトウェアは自然の摂理や原理原則を母体とする自然科学とは一線を画します。エンジニアの発想と思考が設計の源泉であり、その可視化は極めて重要です。一方、最新のクルマはCコード換算で4千万行を超えるソフトウェアで機能しています。自動運転技術のカメラに内蔵されている認識チップは僅か9mm角ですが、その中に百万個のトランジスタが集積されています。OS（Operating System）もリアルタイムOSに加えて、LinuxやAndroidと様々です。従って、我々はソフトウェアの複雑性に支配されず、ソフトウェアを制する力のある開発組織へと転換する必要があります。「開発技術マネージメント」「開発プロセスやツール環境整備」「技術者コンピテンシー」の視点で、着実にソフトウェア技術を我々の手の内にすることが肝要です。

そのために我々は、まずシステムズエンジニアリングを活用して開発の衝となる要求分析・要求仕様の“見える化”に着手しました。次に電子アーキテクチャの統合･一本化を図り、各ECU（Electronic Control Unit）のソフトウェア機能割付けを再定義しました。さらにはモデルベース開発導入により、要求仕様をモデルとプログラムコードで伝達可能にすると共に、困難を極めるソフトウェア評価をテストシナリオ拡充で強化しています。一方で今後の自動車技術者には、Autosar OSやSoC（System-on-a-Chip）、マルチコアの使いこなし方など、より実践的なソフトウェア技術力と知識が求められます。

そこで、「“ソフトウェア”は自動車技術者の新しいリテラシー」と結論付け、自動車工学を専門とする技術者がソフトウェア技術を我が物にするために「ソフトウェアトレーニングセンタ」を開設しました。カリキュラムはタフで力量が試される構成です。マイコン／制御理論／ソフトウェア基礎を学び、モデリングツールMATLAB／Simulinkを使ってプログラムを設計し、コード生成／静的コード解析とHILS（Hardware-In-the-Loop-Simulation）評価に及ぶ全ての科目で“合格”が求められます。卒業生からは、自らの成長を自らが実感すると共に、学びの成果を担当開発業務へ積極的に適用していく旨の計画が示されています。今後、年間約100名が巣立つ計画で、彼らのエクスポーネンシャルな貢献と彼らソフトウェア人財ネットワークが創り出すシナジー効果が期待されます。

ソフトウェア工学のバイブルとも言われる書籍「人月の神話」に、“単純に人数を倍にしても開発期間は半減しない。人（人数）と月（時間）が交換可能という幻想は捨てなさい。”とあります。「お客様に共感される機能と価値」を「自信のある品質」で提供するために、我々はソフトウェア開発パワーを新たな競争力の源泉と位置付け、明確な戦略を持って継続的に高めていきます。

Software—New Literacy Requirement for Automotive Engineers

Shunichi Toyomasu
Fellow

Thirty-seven years ago I was holding a copy of the "Transistor Technology" magazine in one hand while making full use of an operational amplifier summing circuit and an absolute value circuit to calculate the thermal load of a vehicle cabin. The results were used to control the temperature control door opening and fan speed of the air-conditioner. However, analog circuit designs were limited with regard to solving mutually exclusive events and compound events.

Soon after that the Z80 microprocessor also began to be adopted on vehicles. It was an 8-bit microprocessor with a tiny memory capacity of 32 KB. Yet with an understanding of interrupt processing and other commands, designs with many more degrees of freedom than ever before could be obtained. Subsequently, discrete circuits were replaced overnight with microprocessors and software. Simultaneously, that also marked the beginning of the battle with software bugs.

Software is clearly different from the natural sciences that are founded on the laws and fundamental principles of nature. The sources of the design are the ideas and thinking of software engineers, and it is extremely important to visualize them. Meanwhile, the latest vehicles operate by means of onboard software consisting of over 40 million lines of source code when converted to a C code equivalent. The recognition chip embedded in the cameras used in an autonomous driving system is merely 9 mm by 9 mm, but one million transistors are integrated in that area. There are a variety of operating systems (OS) such as Linux and Android, in addition to real-time OS. Therefore, it is necessary for us to transform our development organization into one that has the capabilities to master software without being dominated by its complexity. It is essential for us to steadily gain a mastery of software technologies from the perspectives of managing technological development, putting in place a development environment with the necessary processes and tools, and strengthening engineers' competencies.

Toward that end, we began using systems engineering to visualize the required analyses and required specifications that are the essential foundations of development work. We then integrated and unified the electronic architecture and redefined the allocation of software functions to each electronic control unit (ECU). Furthermore, as a result of implementing model-based development, we made it possible to convey the required specifications in terms of models and program codes. We also strengthened our abilities for software evaluation, which has become exceeding complex, by expanding and improving our test scenarios. On the other hand, in the years ahead automotive engineers will need to have more practical software capabilities and knowledge, including the ability to skillfully use AUTOSAR OS, system-on-a-chip (SoC) and multicore technologies.

We concluded therefore that software represents a new literacy requirement for engineers specializing in automotive engineering and established the Software Training Center so that they can acquire a mastery of software technologies. The curriculum consists of rigorous courses that challenge the capabilities of the trainees. After learning the basics of microcontrollers, control theory and software, trainees design a software program using modeling tools like MATLAB and Simulink, generate the source code, and evaluate it using static code analysis and hardware-in-the-loop simulation (HILS). They are required to pass all these courses in the software development process. Graduates of the Training Center have said that they could truly feel the progress they made and that they plan to actively apply what they learned in the development work they are responsible for. We plan to graduate approximately 100 trainees annually in the coming years. It is expected that their exponential contributions and networking with software engineers will be effective in producing significant synergies.

The author of "The Mythical Man-Month," which is the Bible of software engineering, says that simply doubling the number of men won't halve development lead time and that one should discard the myth that men and months are interchangeable. We have positioned software development capabilities as a new source of competitiveness. We are continuing to improve our capabilities under a clearly defined strategy in order to supply functions and value that resonate with customers and possess quality in which we have confidence.

日産におけるソフトウェア40年の歩みと新たに必要となる人材

40 years of Automotive Software and New Requirements for Software Engineers at Nissan

ソフトウェア開発部　石　上　和　宏
Software Engineering Department　Kazuhiro Ishigami

1. はじめに ― 車載ソフトウェア40年の歩み ―

日産自動車が、マイクロコンピュータを搭載したエンジン制御ECU（Electronic Control Unit）を市販したのは、1979年のことである。430型セドリック向けL28E型エンジン用に、初めてソフトウェアが搭載された。以来、およそ40年が経過したが、依然として車載ソフトウェアは増大の一途をたどっており、今日では自動車にとって最も重要なテクノロジの一つとなっている。ここでは、まず、車載ソフトウェアが40年間にどのように変化し歩んできたのかを振り返ることとしたい。

1.1　ソフトウェアの拡大

最初のエンジン制御ECUに搭載されたソフトウェアのサイズは、およそ3000行程度であった。このソフトウェアが8KバイトのROM（Read-Only Memory）に、データも含めて実装された。エンジンや変速機、ブレーキなどへの電子制御の適用は、開発の自由度を大幅に上げることができ、また、細かな制御を行うことで、メカでは困難な性能間のトレードオフ問題を解決することができた。このため、ソフトウェアを用いた電子制御がクルマの各部位に次々と適用されていった。図1に日産自動車におけるソフトウェアを用いた電子制御の適用開始時期を示す。このように1990年代以降、急速にクルマの電子化が進み、ソフ

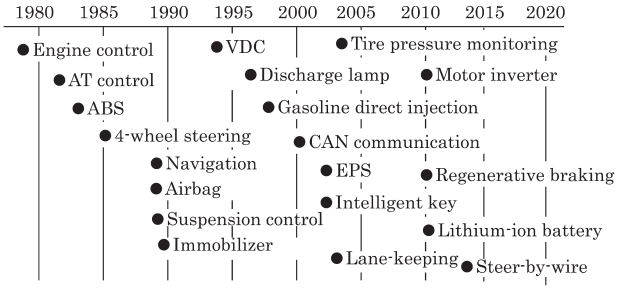


図-1　日産における主要カーエレクトロニクスの導入歴
Fig. 1　Implementation of major car electronics on Nissan vehicles

1. Introduction—40-year history of onboard software

Nissan first marketed an engine electronic control unit (ECU) incorporating a microcomputer in 1979. A software program was implemented for controlling the L28E engine mounted on the 430 series Cedric sedan. Approximately 40 years have passed since then and onboard software has continually been expanded during that time. Today, software is one of the most important technologies for vehicles. This article reviews how onboard software has changed over the past 40 years.

1.1　Expansion of onboard software

The size of the software code installed in the first electronic ECU was approximately 3,000 lines. This software, including the data, was implemented in an 8-kilobyte read-only memory (ROM). The application of electronic control to the engine, transmission, brakes and other vehicle systems greatly expanded the degrees of freedom for development work. In addition, fine-tuned electronic control made it possible to resolve trade-off

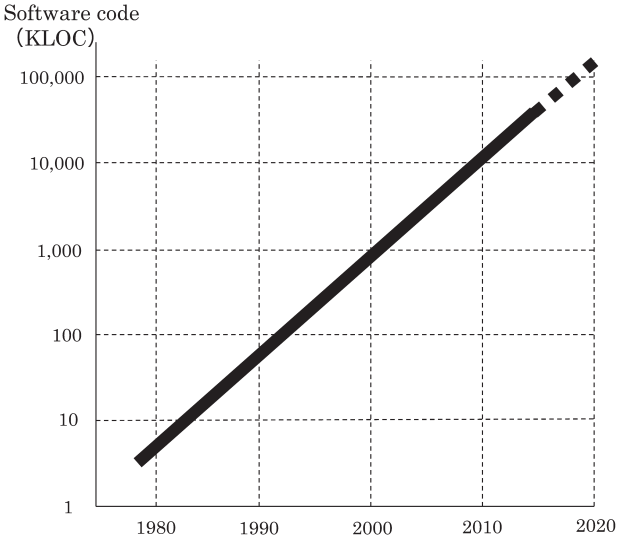


図-2　日産車の車載ソフトウェアの行数
Fig. 2　Embedded software size in kilo lines of code (KLOC) on Nissan vehicles

トウェアの搭載量が拡大していった。

具体的なソフトウェアの搭載量の推移を図2に示す。このように1979年におよそ3000行で始まったソフトウェア搭載量は、2013年には3000万行に達し、40年間で1万倍となった。しかし、ソフトウェアの開発手法の面では、いくつかの革新的技術はあるものの、依然として多くをソフトウェアエンジニアによる人力での開発に依存している。このソフトウェア開発に必要となる工数の大幅な増大が、自動車メーカにとって、品質面でもコスト面でも大きな負担となってきている。

1.2　車載ソフトウェアの技術的進化

ここまでソフトウェアの規模的増大について述べたが、以下では、技術的な変革について述べる。

1.2.1　モデルベース開発（MBD）

1990年代の中ごろ以降、車載ソフトウェアの開発手法は、独自の進化をたどる。1979年の車載開始時点では、アセンブリ言語が用いられた。その後1990年代に入り、徐々にC言語が採用されていった。ここまでは車載以外の組込みソフトウェアの進化と同じである。しかし、これ以降独自の進化の道をたどる。それは、MATLAB/Simulinkモデルを用いたMBD[1]である（図3）。特に走行系の車載ソフトウェアでは、制御設計をMATLAB上で行うことで効率的に制御ロジックを構築できる。MATLABを用いると、その場でシミュレーションを行うことができるので、シミュレーション結果を制御アルゴリズムにフィードバックすることで、早期に制御アルゴリズムを作成することができる。作られたアルゴリズムから、直接量産向けソースコードを生成できれば、更なる効率化が図られる。このような経緯から、車載の走行系ソフトウェアに対して、MATLABを用いたMBDが活用されるようになった。

1.2.2　AUTOSAR（Automotive Open System Architecture）

2000年代に入ると、MBDを支えるソフトウェアプラットフォームとして、AUTOSAR OS規格の策定が行われ

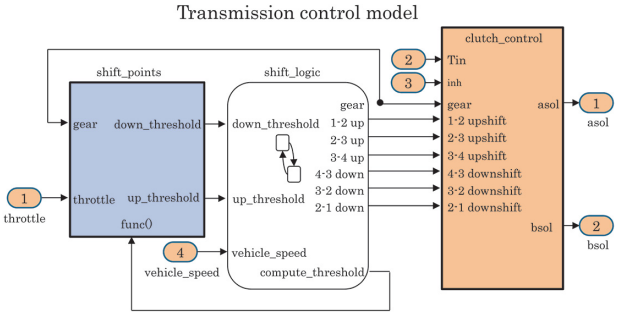


図-3　数値解析ソフトウェア MATLAB モデルの一例
Fig. 3　An example of a MATLAB model

problems between performance attributes that were difficult to resolve mechanically. For that reason, software-based electronic control has been applied to vehicle components one after another. Figure 1 shows the years when Nissan began to apply software-based electronic control to major vehicle components. As indicated in the figure, automotive electronics advanced rapidly from the 1990s onward and the volume of onboard software has expanded accordingly.

Figure 2 shows the specific increase in the volume of onboard software on Nissan vehicles. Software code that began with 3,000 lines in 1979 had increased to 30 million lines by 2013, an increase of 10,000 times in 40 years. However, software development still continues to depend on the human labor of large numbers of software engineers, though several innovative tools have been achieved in the area of software development methods. The drastic increase in man-hours needed for software development has become an enormous burden for vehicle manufacturers in terms of both quality and cost aspects.

1.2　Evolution of onboard software technologies

The increase in the scale of automotive software code was described in the preceding section. The following discussion will describe the technical changes that have occurred.

1.2.1　Model-based development (MBD)

Automotive software development methods have evolved along a unique path since the mid-1990s. Assembly languages were used in 1979 when Nissan started applying automotive software to vehicles. Subsequently, C language gradually came into use in the 1990s. Until that point, onboard software had evolved in the same way as other embedded software for non-automotive applications. However, it began to follow its own unique path of evolution from that point on, namely, MBD[1] using MATLAB/Simulink models(Fig. 3). For the software embedded in the driving systems of vehicles in particular, executing the control design with a MATLAB model facilitates efficient creation of the control logic. Using a MATLAB model allows a simulation to be run on the spot. The simulation results can then be fed back right away to the control algorithm, enabling early completion of the algorithm. Efficiency can be further improved if the source code intended for mass production can be generated directly from the resultant algorithm. That is how MBD using MATLAB models came be actively employed for developing the software embedded in the driving systems of vehicles.

1.2.2　Automotive Open System Architecture (AUTOSAR)

The AUTOSAR operating system standard was established in the early 2000s as a software platform supporting MBD. An application is designed using a MATLAB model and combined with AUTOSAR,[2] a software platform for supporting applications, to complete

た。アプリケーションをMATLABモデルで設計し、アプリケーションを支えるソフトウェアプラットフォームであるAUTOSAR[2]と組み合わせて、組込みソフトウェアとして完成させる。このAUTOSARは、自動車に必要なリアルタイム制御をサポートするため、AUTOSAR専用の特殊な仕組みと構造となっている。AUTOSARも自動車業界独自の進化の一つと呼ぶことができる。

2. 自動車業界の置かれている状況と新たに求められるソフトウェア人材

前述したように、車載ソフトウェアの搭載量の増大に伴い、これを開発し品質確認するための開発リソースも大幅に増加している。また、技術的にはMBDやAUTOSARといった自動車業界特有の組込みソフトウェア技術が導入されてきたため、自動車業界向けの専用ソフトウェアエンジニアが大量に必要となってきた。このような状況の中、現在自動車メーカ各社は、三つの新たなる領域の開発を進めている。それは、電動化、自動運転、コネクティドカーである。これらの実現にはソフトウェアが大きくかかわっている。それも、従来の組込みソフトウェア技術とは異なる領域のソフトウェア技術が求められている。このため、自動車業界では、これまでの自動車向け組込みソフトウェアエンジニアという人材に加えて、新たな技術領域のソフトウェア人材が求められるようになった。以下に、新たに求められる代表的なソフトウェア技術領域と人材を記述する。

2.1 インターネット関連のソフトウェア人材

コネクティドカーにインターネット技術が必要となるのは自明ともいえるが、それ以外の領域でも、インターネット技術が自動車向けに応用されてきている。一例として、欧米が中心となって策定された電動車両の充電制御用通信プロトコルであるISO/IEC15118（通称：コンボ）について説明する。ISO/IEC15118では、いくつかのIETF（Internet Engineering Task Force）の定めたRFC（技術規格）が参照されている。たとえば、下記のRFCをサポートする必要がある。

- RFC1981：パスMTUディスカバリ
- RFC5722：オーバーラッピングIPフラグメント
- RFC4443：ICMPv6

これは、これらのインターネット技術規格の知識無くしては、電動車両の充電器用ソフトウェアを開発することができないことを意味している。

このように、IETFはインターネットの技術的仕様を定義するために、膨大な数のRFCを制定している。これからの自動車向けのソフトウェア人材には、このようなインターネット技術の知識が必要となる。

the embedded software. AUTOSAR is configured with its own special dedicated system for supporting the real-time control required by vehicles. AUTOSAR can be regarded as one example of evolution unique to the automotive industry.

2. Current Situation of the Automotive Industry and New Requirements for Software Engineers

As described above, software development tools have also increased significantly in order to develop and confirm the quality of automotive software that has vastly expanded on vehicles in recent years. Because MBD, AUTOSAR and other embedded software technologies specific to the automotive industry have been adopted, there has been a need for much larger numbers of software engineers specialized in automotive industry applications. Amid these circumstances, all of the vehicle manufacturers are currently proceeding with development activities in three new fields. These are vehicle electrification, autonomous driving and vehicle connectivity. Software is playing a large role in turning these technologies into realities. Moreover, the software technologies required in these fields differ from those of conventional embedded software. Consequently, the automotive industry needs software engineers in these new technical fields, in addition to the software engineers who have been developing embedded software for vehicles to date. Typical examples of newly required software engineering fields and requirements for software engineers are described below.

2.1 Internet-related software engineers

It is obvious that Internet technologies are necessary for vehicle connectivity, but they are also being applied to vehicles in other areas as well. The communications protocol specified for controlling EV charging is explained here as one example. This protocol has been largely defined by Europe and the U.S. in the ISO/IEC 15118 (Combined Charging System) standard, which makes reference to several Requests for Comments (RFCs) issued by the Internet Engineering Task Force (IETF). For example, the following RFCs must be supported.

- RFC 1981: Path MTU Discovery
- RFC 5722: Handling of Overlapping IPv6 Fragments
- RFC 4443: Internet Control Message Protocol (ICMPv6)

This means that software for EV chargers cannot be developed from now on without knowledge of Internet technical standards.

As indicated here, the IETF has issued an enormous number of RFCs for the purpose of defining Internet technical specifications. It is expected that software engineers developing code for vehicles will understand these Internet technologies and have the ability to create software accordingly.

ターネット技術を理解し作り上げる能力が期待されている。

2.2 サイバーセキュリティ関連のソフトウェア人材

コネクティドカーはインターネット技術をベースに構築されているが、インターネットにはサイバーセキュリティがつきものである。特に、ホワイトハッカーによるセキュリティカンファレンスBlack Hat USA 2015でのJeep Cherokeeのハッキングに関する発表は、自動車業界に大きな衝撃を与えた。自動車でもインターネットと同様のサイバーセキュリティ対策が求められるようになった。つまり、暗号技術や暗号に基づくシステム設計技術が必要ということである。

これまで自動車メーカでは、社内ネットワーク向けや企業のウェブサイト向けには、もちろんエンジニアが存在していたが、自動車本体にサイバーセキュリティ技術を織り込むためのエンジニアという観点では、ほとんど保有してこなかった。しかし、本格的なサイバーセキュリティ対策は待った無しの状態にある。このように、新しい自動車向けのソフトウェア人材として、サイバーセキュリティエンジニアが強く求められている。

2.3 通信技術関連のソフトウェア人材

コネクティドカーのシステムは、通信路の一部に携帯電話による通信回線を用いている。携帯通信回線側の通信システムはなるべくインターネット通信仕様との親和性を上げる努力を行っているものの、依然として独自の仕様も存在している。また、国際標準化が推進されているが、各携帯電話会社によって世代更新の進め方や、機能のサポート状況が異なるケースが多く見られる。また、現在新たに第5世代移動通信システム（5G）の導入が進められているが、携帯通信技術は高度の通信技術に加えて、膨大なソフトウェアとセットで構成されている。そして、コネクティドカーのシステムは、このような携帯通信技術を前提として設計されている。このため、通信技術の理解が必須である。このように、今後の自動車向けのソフトウェア人材として、携帯通信をはじめとした通信技術を熟知したソフトウェアエンジニアが求められている。

2.4 Linux・Android関連のソフトウェア人材

車載インフォテインメントシステムは、近年独自の進化を続けており、2000枚を超える画面を制御し、膨大な機能を操作できることが求められる。このような製品は、他の業界を含めても類を見ないものとなっており、膨大なソフトウェアのかたまりとなっている。この車載インフォテインメントシステムのベースとなるソフトウェアプラット

2.2 Cybersecurity-related software engineers

Vehicle connectivity is built on the basis of Internet technologies, and cybersecurity is indispensable to the Internet. A presentation concerning the hacking of a Jeep Cherokee, which was given at the Black Hat USA 2015 security conference of white hackers, gave an especially large shock to the automotive industry. Vehicles are also required to have cybersecurity measures now equal to those of the Internet. In other words, cryptographic technology and system design methods based are cryptography have become necessary.

Heretofore, vehicle manufacturers have naturally had engineers who have been developing software for the company's internal network and website. However, they have had hardly any engineers involved in implementing cybersecurity technologies on vehicles themselves. Yet the situation today does not allow any waiting for full-fledged cybersecurity measures. Consequently, cybersecurity engineers are strongly needed as a new source of software engineers for vehicles.

2.3 Communications technology-related software engineers

Connected vehicle systems use the communications circuits of mobile phones as part of their communications channels. While efforts are being made to enhance the affinity between the communications systems of mobile phone communications circuits and Internet communications specifications, there are still some systems that have their own specifications. Moreover, while international standardization is being promoted, many cases are still seen where progress toward next-generation updates and the functional support status differ from one mobile phone company to another. Activities are also currently under way to launch new fifth-generation mobile communications systems (5G). Besides sophisticated communications technologies, mobile communications technologies consist of enormous volumes of software. Because connected vehicle systems are designed on the premise of such mobile communications technologies, a good understanding of them is indispensable. Accordingly, software engineers developing code for vehicles in the years ahead will have to be well-versed in communications technologies, especially mobile communications.

2.4 Linux/Android-related software engineers

Onboard infotainment systems have continued to follow their own path of evolution in recent years and are now required to control over 2,000 menu screens and to support the operation of an enormous number of functions. Such products are without parallel even in other industries and constitute a gigantic mass of software. In recent years, Linux or Android has come to be used as the software platform on which onboard infotainment systems are

フォームとしては、近年Linuxもしくは Androidが用いられるようになった。また、画面の美しさがクルマの魅力度を向上させるツールとして使われるようになっており、アニメーションと組み合わせて、美しく使いやすいHMI（Human Machine Interface）が求められてきている。これらの要求を実現するために、Linux及びAndroid関連の技術を使いこなせるソフトウェアエンジニアが求められている。

2.5 画像認識・音声認識関連のソフトウェア人材

画像認識は自動運転技術に、また音声認識はコネクティドカーに必須の技術である。現在は第3次人工知能（AI）ブームと言われているが、現在のAIブームの中で特に進化したとされるのが、これらの画像認識、音声認識技術である。両者にはベースに、ディープラーニング技術が用いられている。今後の自動車向けのソフトウェア人材としては、ディープラーニング技術を理解し、画像認識、音声認識技術をさらに改良していくことができるソフトウェアエンジニアが求められている。

3. ソフトウェア能力向上のための取り組み

日産自動車では、これまでに述べた新たなソフトウェア技術領域の能力向上のために、いくつかの方策を推進している。

(1) ソフトウェアエンジニアの確保
(2) ITベンダの資源の活用
(3) 社内のエンジニアのトレーニングによる能力アップ

これらを総合して、電動化、自動運転、コネクティドカーに対応できる能力を取得すべく活動を進めている。

これからも社会のニーズに対応し、自動車ビジネスの大変革に対応するため、これらのソフトウェア能力を獲得・拡大していきたい。

4. 参 考 文 献

1) 大畠明：自動車制御に対するモデルベース開発、自動制御連合講演会講演論文集、Vol. 50、p. 126（2007）.
2) 鈴村延保ほか：車載組み込み技術開発の欧州全体俯瞰と動向、情報処理学会研究報告EMB組込みシステム、Vol. 2009-EMB-14、No. 9、pp. 1-12（2009）.

based. In addition, beautiful screens are also being used as a tool for enhancing a vehicle's degree of product appeal. A beautiful-looking and easy-to-operate Human-Machine Interface (HMI) is demanded for use in combination with animation software. In order to satisfy these requirements, software engineers are needed who can skillfully use the technologies associated with Linux and Android.

2.5 Image/voice recognition-related software engineers

Image recognition is indispensable to autonomous driving technology and voice recognition is necessary for vehicle connectivity. A third-generation artificial intelligence (AI) boom is said to be under way at present. Amid this current AI boom, image and voice recognition technologies in particular have undergone a profound evolution. Both fields are based on the use of deep learning technology. Software engineers developing code for vehicles in the future will need to understand deep learning technology and be capable of further improving image and voice recognition technologies.

3. Activities for Enhancing Software Capabilities

At Nissan, we are currently proceeding with several measures noted below for enhancing our capabilities in the new software engineering fields described in the preceding sections.

(1) Recruiting software engineers
(2) Making effective use of the resources of IT vendors
(3) Conducting training to improve the abilities of in-house engineers

Comprehensive activities are being promoted in these areas to acquire the capabilities needed for dealing with electrification, autonomous driving and vehicle connectivity.

We want to continue to acquire and expand these software capabilities in order to thoroughly reform our automotive business and respond to social needs in the future

4. References

1) A. Ohata: Model-Based Development for Automotive Control, Proc. of the Japan Joint Automatic Control Conference, Vol. 50, p. 126 (2007).
2) N. Suzumura et al.: Overview and Trends of European Automotive Embedded Technology Approach, IPSJ SIG Technical Report, Vol. 2009-EMB-14, No. 9, pp. 1-12 (2009).

静的解析ツールの開発

Development of a Static Code Analysis Tool

長谷川　美和子*　　市　川　　智*
Miwako Hasegawa　　Satoshi Ichikawa

抄　　録　　クルマにソフトウェアが搭載されてから、すでに約40年が経過したが、いまだに車載ソフトウェアの規模や複雑度は増大している。また、不具合発生時にクルマの安全上致命的な影響を及ぼすシステムにもソフトウェアは大規模に採用されるようになっている。このような中、車載ソフトウェアにとって、高品質なソフトウェアを達成することは重要なテーマである。静的解析ツールはソースコードを解析することで、不具合の可能性のある部位を自動的に特定できるため、日産自動車では車載ソフトウェアの品質を確保する重要な手法と位置づけ、ツールの性能向上と実際の量産ソフトウェアへの適用を並行して行ってきている。ここでは、市販ツール活用の中で検出された問題点を紹介し、また我々がパートナと開発した静的解析ツールについて述べる。

Summary　　About 40 years have elapsed since software was first installed on vehicles. However, the size and complexity of onboard software are still increasing. In addition, software is being adopted on a large scale even in safety critical systems that can have a serious impact on the safety of vehicles when a problem occurs. Nissan regards static analysis as a very important technology for achieve high-quality software. The reason is that static analysis tools can automatically identify possible errors in the source code by analyzing the code. Therefore, Nissan has been improving the performance of such tools and also applying them to actual mass production software to check the quality of onboard software code. This article describes the limitations of commercial static analysis tools and presents a tool that Nissan developed with a partner.

Key words : *Computer Application, automotive electronics, embedded software, static analysis, AI, computer*

1. はじめに

1970年代後半に、ソフトウェアがクルマに搭載され始めた。当初は、アセンブラ言語が用いられたが、1990年代になると、次第にC言語に置き換えられていった。プログラム言語がC言語になると、ソースコードを自動的に解析し、不具合の可能性のある部分を特定する静的解析ツールが活用できるようになる。アセンブラ言語は、CPUコアごとに異なる言語とならざるを得ないが、C言語は、かなりの部分がCPUコアに依存せず標準化されているからである。1990年代には、いくつかの静的解析ツールが市販されるようになった。

一方、2000年代初めに、日産はソフトウェア品質で大きな問題に遭遇する。車載ソフトウェアが増大し、ソフトウェア品質がクルマの品質を左右するようになったためである。業界他社同様、日産はソフトウェア開発の多くをサプライヤに依存していた。このソフトウェア品質問題は、ほとんどがサプライヤの開発したソフトウェアに関する問

1. Introduction

Software began to be implemented on vehicles in the latter half of the 1970s. Assembly languages were initially used to write the software code, but they were gradually replaced by the C programing language in the 1990s. With C used as the programming language, it became possible to apply static code analysis tools for analyzing the source code automatically and identifying any places where there might be errors. Assembly languages invariably differed for each CPU core, but the C language was standardized for a fairly large portion of CPU cores, which facilitated the use of static analysis tools. Several such tools were put on the market in the 1990s.

At the beginning of the 2000s, Nissan encountered a huge problem with regard to software quality. That was because the expanded use of onboard software meant that software quality had a large impact on determining vehicle quality. Like other companies in the automotive industry, Nissan depended on suppliers to develop much of the company's onboard software. Nearly all of the software quality problems concerned the software developed by

* ソフトウェア開発部／Software Engineering Department

題であった。そこで、我々はソフトウェア技術者のチームによる“外注ソフトウェアの品質監査活動”を実施することを決めた。その中で、サプライヤが開発したソースコードから自動的に不具合の可能性のある部位を抽出するため、静的解析ツールを活用することとした。

2. 静的解析ツール開発の動機

前述の通り、2000年代初めに日産は、市場に出回っている三つの汎用静的解析ツールの活用を開始した。これらは、異なる観点で用いられた。

- QACは、MISRA-Cガイドラインへの適合をチェックする。
- I-magix 4Dは、ソースコードの構造や再帰呼出しの関数、タスク間共有変数をチェックする。
- Polyspaceは、ゼロ割、配列の領域外アクセス、オーバーフローなどのランタイムエラーを検出する。

三つの汎用ツールを用いて、外注ソフトウェアの品質査活動を行っていたが、汎用ツールの限界や改善すべき項目について認識するようになった。以下は、汎用ツールにおける主な制限内容である。

2.1 スケーラビリティ

インフォテインメントシステムは、静的解析を行うことが困難なシステムであった。日産は、以下のツールの制約に直面した。

- ファイルや関数を跨（また）いだ値の受け渡しの解析能力に限界があり、百万行を超える規模のソースコードではランタイムエラーの解析ができない。
- 一つのランタイムエラーを検出すると、その場で解析を停止してしまうため、検出された一つのエラー箇所を修正し再解析する必要がある。インフォテインメントシステムを解析するには数時間を要するが、エラーの数だけ解析を繰り返さなければならない。

2.2 自動車の組込みソフトウェア特有の機能

市販の汎用ツールは、多くの標準的なチェック機能を備えているが、自動車特有の機能は提供していない。たとえば、割込みによるデータの上書き、スリープ／ウェイクアップ処理やフラグOn／Off処理の操作ミスなど、車載ソフトウェア特有の問題を検出できない。

2.3 並列処理およびマルチコアシステム

並列処理システムやマルチコアシステムでは、並列に動作するソフトウェア間のデータの整合性やデッドロックなどによる不具合が発生する。汎用ツールはこれらの不具合を検出する機能を備えていない。

suppliers. Therefore, we decided to launch an activity conducted by a team of our software engineers for monitoring outsourced software quality. In this activity, it was decided to use static code analysis tools to automatically identify places where there might be errors in the source code developed by suppliers.

2. Motivation for Developing a Static Code Analysis Tool

As mentioned above, in the early 2000s Nissan began using three general-purpose static code analysis tools that were commonly available on the market. These tools were used from different perspectives.

- QAC was used for checking compliance with the Motor Industry Software Reliability Association (MISRA) C guidelines.
- Imagix 4D was used for checking the source code structure, recursive call function and inter-task shared variables.
- Polyspace was used to detect run-time errors such as division by zero, access outside the array bounds and overflow.

Outsourced software was inspected using these three general-purpose tools, but we came to realize that the tools had their limitations and that there were points needing improvement. The details of the principal limitations of these tools are described below.

2.1 Scalability

It was difficult to conduct a static analysis of an infotainment system. We encountered the following limitations of analysis tools.

- The ability to analyze the transfer of values extending across files or functions is limited. Large-scale source codes exceeding a million lines cannot be analyzed for run-time errors.
- When a run-time error is detected, the analysis is suspended at that point. The detected error must be fixed and that location re-analyzed. It takes several hours to analyze an infotainment system, and analyses must be repeated to the same extent as the number of errors discovered.

2.2 Functions specific to automotive embedded software

General-purpose tools on the market are equipped with many standard functions for checking software code, but they do not provide any functions specifically for vehicles. For example, they cannot detect problems specific to onboard software such as operational errors due to the overwriting of data caused by an interrupt, sleep/wakeup processes and flag On/Off handling.

2.3 Concurrent processing and multi-core systems

Concurrent processing systems and multi-core systems experience problems such as those caused by data inconsistency and deadlocks between software programs running concurrently. General-purpose analysis

2.4 ユーザビリティ

静的解析ツールは、解析のメカニズム上、ソースを１行変更しただけでも、ソース全体を再度チェックして、問題が発生する可能性のある行を指摘する。したがって、エンジニアは、指摘された不具合の可能性のある行を再度レビューしなくてはならない。しかし、現場のニーズとしては、今回の変更で影響のある行のみのレビューにとどめたい。なぜなら実際にレビューを行うためには、多大な工数がかかるからである。バージョン違いや、継続的インテグレーション（CI）での機能追加前後のレポート比較を可能にするレポート差分解析機能は、使い勝手や適用可能性に影響を及ぼす。市販ツールは、差分解析機能が提供されていない、もしくはあったとしても差分抽出が単なるコード差分のみで、サイドインパクトまで分析できていない。

以上のような問題を対策するため、日産はパートナと共同で、新しい静的解析ツールの開発を開始した。我々はこのツールを、Embedded Software Code Analysis Toolの略でESCATと呼ぶことにした。

3. ツール開発

日産はサプライヤから流出してくるソフトウェア不具合を分析し、静的解析が不具合検出に効果的である不具合を特定した。そして、その不具合を一般化し、不具合と同じソフトウェア構造やソースコード記述を抽出する機能をESCATに搭載した。

本章では、いくつかの実装された機能についてまとめ、紹介する。

3.1 スケーラビリティ

ESCATのスケーラビリティを確保するために、以下の手法を採用し、数百万行のソースコードを解析することを可能にした。

- パーティショニング・アルゴリズムを用いることで、大規模なシステムを、いくつかのサブシステムに分割することができる。これにより、分割されたサブシステムの複雑さは軽減し、サブシステムごとに解析を完了できる。そして、サブシステムの解析結果を統合することで、大規模で複雑なシステム全体の解析結果を取得することができる。[7)]
- ユーザによる解析精度の設定／変更を可能にすることで、システムの複雑度に合わせ最適な解析精度を選択できる。これにより、ツールのユーザはあるレベルの精度低下を起こさずに、より規模の大きい複雑なシステムを分析できる。

tools are not equipped with a function for detecting such problems.

2.4 Usability

Because of the analysis mechanism used, static analysis tools re-check an entire source code even for revision of just one line and indicate lines where a problem might occur. Consequently, engineers must review again any lines indicated as having a potential problem. However, the actual need in the workplace is to limit the review only to the lines that might be affected by that revision. The reason is that an enormous number of man-hours are needed to conduct an actual review. Functionality for analyzing differences between reports, making it possible to compare reports before and after the addition of a function owing to a difference in software versions or continuous integration (CI), has an effect on ease of use and applicability. Tools on the market do not provide a function for analyzing such differences, and even if they do, the extraction of differences is limited only to code differences. The tools cannot analyze side effects.

In order to address the foregoing problems, we initiated development of a new static code analysis tool together with a partner. The resultant tool is called the embedded software code analysis tool (ESCAT).

3. ESCAT Development

We analyzed the defects stemming from the software received from suppliers and identified defects that could be effectively detected by static analysis. Those defects were generalized and ESCAT was equipped with functions for detecting the same software structures and source code descriptions as those of the identified defects. This section describes several of the functions incorporated in ESCAT.

3.1 Scalability

The following methods were adopted to ensure the scalability of ESCAT, enabling the tool to analyze source code consisting of several million lines.

- A partitioning algorithm is used to divide a large-scale system into several subsystems. This reduces the complexity of the divided subsystems, enabling an analysis to be completed for each one. The analysis results for the subsystems are then integrated to obtain the analysis results for the overall large-scale, complex system.[7)]
- Analysis accuracy can be defined and modified by the user, making it possible to select the optimum analysis accuracy matching the level of system complexity. This enables ESCAT users to analyze larger scale and more complex systems without causing accuracy to decline to a certain level.

3.2 Functions specific to vehicles

Functions specifically for automotive embedded software were developed as key capabilities of ESCAT,

3.2 自動車特有の機能

ESCATの重要な機能となる、自動車組込みソフトウェア特有の機能を開発し、車載ソフトウェアの問題の検出を可能にした。

3.2.1 フラグOn／Off処理[5)]

車載ソフトウェアに限らないが、組込みシステムでは、共有リソースを使用する際に、リソースを一時的に占有するためのフラグが頻繁に使用される。しかし、設計不良のため、リソースを占有したまま解放せずに処理を終了するケースも見られる。特にインフォテインメントシステムのような巨大なソフトウェアでは、これらのフラグのセット（On）、リセット（Off）のペアが正しく設計されておらず、大きな品質問題となる。

この機能は、ソースコードから自動的にフラグを特定し、あらかじめ定義されたルール通りにフラグが用いられているかをチェックする。例えば、リソース占有を示すフラグについて、いかなる処理のパスを通った場合でも、セットとリセット（On／Off）のペアが成立することをチェックする。

3.2.2 スリープ／ウェイクアップ[1)9)]

従来の内燃機関を動力源とするクルマでは、バッテリ上がりを防止するため、キーオフ時の車載システムの消費電力を極力抑える必要がある。しかし、インテリジェントキーのように、お客様の接近を検知するため、キーオフ時も動作の必要なシステムがある。これらのシステムでは消費電力を抑えるため、マイコンのスリープ、ウェイクアップ制御が行われる。この制御は、マイコン内蔵のハードウェアによって実現される。スリープ／ウェイクアップで発生する問題は、ハードウェアの状態とソフトウェアの状態の不整合である。ハードウェアがスリープ状態にある時は、ソフトウェアはウェイクアップ命令が実施可能でなくてはならないし、ハードウェアがウェイクアップ状態にある時は、ソフトウェアはスリープ命令が実施可能でなくてはならない。しかし、ハードとソフトの状態不整合が起きると、スリープしたまま、もしくはウェイクアップしたままという不具合に至る。

ハードウェアの状態はレジスタで代表され、ソフトウェアの状態は変数で代表される。そこで、この機能は、これらの範囲の中で変更されたすべてのレジスタと変数をリストアップし、範囲の中で検出された不整合をレポートする。また、これらのレジスタを書き換える際の、割込みの許可禁止の状態についても、情報を提供する。

3.2.3 データ競合[3)]

この機能は、異なる優先順位のタスク間で、共有メモリに対しアクセスした場合のデータ競合状態を検出する。車載ソフトウェアの多くは、処理を複数のタスクに分け、切替えながらシステムを制御する。図1のサンプルコード1では、低優先度のタスク1と高優先度タスク2が実装され、

making it possible to detect problems in onboard software.

3.2.1 Flag On/Off handling[5)]

When embedded systems use shared resources, flags are frequently employed to indicate that a resource is temporarily occupied, although this is not limited to onboard software. However, cases are also seen where processing is concluded without releasing an occupied resource owing to a design defect. This becomes a large quality issue especially for mammoth software code like that of infotainment systems if the operations of setting (On) and resetting (Off) a pair of flags are not correctly designed.

This function checks whether flags are automatically identified by the source code and used in accordance with predefined rules. For example, for flags showing the occupied status of resources, it checks whether the setting and resetting (On/Off) of a pair of flags are validly done regardless of what processing path is followed.

3.2.2 Sleep/wakeup[1)9)]

Power consumption by onboard systems of vehicles powered by a conventional internal combustion engine must be minimized as much as possible while the ignition key is off to avoid causing a dead battery. However, there are onboard systems like the Intelligent Key entry system that must operate upon detecting the approach of the driver even when the ignition key is turned off.

To suppress consumption of electricity by such systems, the vehicle microcomputer performs sleep/wakeup control over them. This control is executed by hardware incorporated in the microcomputer. One problem that occurs concerning sleep/wakeup processes is an inconsistency between the hardware and software states. The software must be able to issue a wakeup command when the hardware is in the sleep state and a sleep command when the hardware is in the wakeup state. However, when a mismatch occurs between the hardware and software states, it leads to a defect where the hardware continues to remain in either the sleep or wakeup mode.

The hardware state is represented by a register and that of the software is represented by a variable. This function lists all the registers and variables changed within the range of these states and reports any inconsistencies detected within the range. It also provides information on whether interrupts are enabled or disabled when the data

```
1.  int shv;
2.  void Task1()     //Low Priority
3.  {
4.      int a, b = 10;
5.      shv = b;     /* access1 – write access */
6.      a = shv;     /* access3 – read access */
7.  }
8.  void Task2()     //High Priority
9.  {
10.     shv = 20;    /* access2 – write access */
11. }
```

図-1 サンプルコード 1
Fig. 1 Sample code 1

低優先度タスク実行中に高優先度タスクへの切り換えが発生する。5行目を実行した後に切り替えが発生した場合、タスク1で実行した変数shvへの代入は、タスク2により上書きされ、タスク2終了後にタスク1が6行目から再開されると、上書きされた不適切な値のshvを読み出すことになる。ESCATはこのような競合状態を検出し、上書き箇所をレポートする。

3.2.4 デッドロックの検出

この機能は、ソースコード内に、デッドロックを起こす可能性のある部分があるかどうかを検出する。デッドロックを起こすシナリオは、多くの状況で起こり得る。図2のサンプルコード2では、タスク1とタスク2が同じ変数shvにアクセスしている。変数shvに安全にアクセスするための排他制御として、タスク1はセマフォR1をロックする。タスク2も同様に排他制御が必要で、サンプル2では同じセマフォR1をロックしていることがわかる。今、タスク1が5行目を実行した後にタスク2への切り替えが発生したシナリオを考えてみる。処理は10行目に移り、タスク2がR1をロックしようとする。このセマフォはすでにタスク1によりロックされている。したがって、タスク2は待ち状態になり、タスク1がセマフォR1をリリースすることを待つ。一方、タスク1は、実行を再開するため、タスク2の処理終了を待っている。両方のタスクが待ち状態に入ってしまうため、システムはハング状態／デッドロックに至る。これに対し、ESCATは、システムがデッドロック状態に入る可能性のある場所を検知しレポートする。

ESCATは、排他制御としてRTOS（Real Time Operating System）ごとに異なるシステムコールを登録できるほか、割込みルーチンとの排他制御についても割込み禁止／許可命令を登録することができ、OSやマイコンに依存せず汎用的に競合状態を検出できる。さらに、マルチタスクスケジューリングで問題となるプライオリティシーリングにも対応可能である。

3.3 並列処理とマルチコアシステム

マルチタスクやマルチコアなど、並列処理を行うシステムでは、同期処理においてデッドロックや競合状態など、プログラミングでの不具合を発生させやすい。一方で、これらが起こった場合、通常の単体テストやシステムテストで、同期処理に伴う不具合を検出することは非常に困難である。複数のタスクが実行されるときに、タスク内でイベントが発生するタイミングは膨大な数のケースがあるからである。

たとえ不具合を起こすイベント割込みを特定することができたとしても、その時点でどのタスクがどのコードを実行しているかというスケジューリングは、すべて定義しきれないため、すべてのイベント割込みタイミングを網羅したテストケースを作ることは現実的に不可能である。

in the registers are rewritten.

3.2.3 Data race[3)]

This function detects a race condition in cases where two tasks having different orders of priority access a shared memory. Many onboard software programs divide a process into several tasks and control a system while switching among the tasks. In Fig. 1, sample code 1 implements a low priority task 1 and a high priority task 2. A switch to the high priority task occurs while executing the low priority task. If a switch occurs after processing the 5th line, substitution of the variable shv being executed in task 1 is overwritten by task 2, and after the completion of task 2, execution of task 1 is resumed from the 6th line. Consequently, an inappropriate value of the overwritten shv is read out. ESCAT detects such race condition and reports the places that are overwritten.

3.2.4 Deadlock detection

This function detects whether there are places in the source code that might cause deadlocks. A scenario causing a deadlock can occur in many different situations. In sample code 2 in Fig. 2, both task 1 and task 2 access the same shv variable. As an exclusion control to ensure safe access to the shv variable, task 1 locks semaphore R1. Similarly, task 2 also requires exclusion control. It is seen in sample code 2 that task 2 also locks the same semaphore R1. Here we will consider a scenario where a switch is made to task 2 after task 1 has been executed to the 5th line. The process proceeds to the 10th line and task 2 attempts to lock R1, but this semaphore is already locked by task 1. Consequently, task 2 enters the wait state and waits for task 1 to release R1. Meanwhile, task 1 is waiting for task 2 to be completed before it can resume execution. Because both tasks enter the wait state, the system reaches a hang state or a deadlock. To deal with such situations, ESCAT detects and reports places where the system might become deadlocked.

ESCAT can generically detect race condition without depending on the OS or the microcomputer. As an exclusion control, it allows a different system call to be registered for each real-time operating system (RTOS). In addition, it also allows interrupt disable/enable commands to be registered as an exclusion control for interrupt routines. Moreover, it can also handle priority ceiling

```
1.  int R1,shv;
2.  void Task1()  //Low Priority Task
3.  {
4.      lock(R1);
5.      shv = 10;  //Task2 pre-empts Task1 here
6.      release(R1);
7.  }
8.  void Task2()  //High Priority Task
9.  {
10.     lock(R1); //Deadlock instance as R1 is already locked by Task1
11.     shv = 20;
12.     release(R1);
13. }
```

図-2　サンプルコード2
Fig. 2　Sample code 2

そこで、これらの同期処理に起因する不具合を検出するため、前述のデータ競合やデッドロック検出機能を、マルチコアシステムでも対応できるようにし、ESCATに組み込んだ。

3.4 ユーザビリティ

3.4.1 レポート差分解析

大規模ソフトウェアシステムに対し、静的解析を行うと、通常大量のワーニングが検出される。そして、これらのワーニングはソフトウェア仕様書や設計書を基に手作業でレビューをしなくてはならない。レビューは、工数と時間がかかる作業であり、また、システムを良く理解している必要があるため、実施できるエンジニアも特定される場合が多い。このためソフトウェアを開発するチームにとって、非常に苦痛を伴う作業である。これらのワーニングの一部は、ソフト修正を行わなくても良いものもあり、誤検知と呼ばれる。誤検知が発生する理由はいろいろある。たとえば、外部入力や外部信号の値、データレンジに関する情報が与えられていないために、現実世界ではあり得ない外部入力や外部信号、そしてその組み合わせをコンピュータは起こり得ると考えること、複雑なデータ構造や演算に関わる静的解析技術の限界などが誤検知の要因となる。

一方、不具合対策や、要求変更でソフトウェアは改変される。このとき、静的解析を実施するためには、改変されていないソフトウェアの部分も含め、ソフトウェア全体を対象に再度静的解析を実施する必要がある。その理由は、改変されていない部分に存在する影響箇所を含めて解析しないと、正しい結果が得られないからである。一方で、コードの改変が影響を与えない部分については、すでに実施したワーニングに対するレビュー結果が有効である。市販の汎用ツールでは、コード改変時に影響範囲を特定して、影響がない部分のワーニングを除去する機能が十分ではなかった。そこで、我々はESCATで、この機能を開発した。

この機能は、現在のバージョンに対し行われた変更の影響分析を実施することで実現している。前のバージョンからの変更に影響されない部分で検出された誤検知を除去する。

3.4.2 手作業でのレビューの補助

静的解析ツールは通常、大量の不具合のワーニングを発生させる。そして、これらは手作業でレビューをする必要がある。このため、品質確認費用が増大する。この問題に対処するために、我々はいくつかの新しい手法を開発した。以下に二つの例を挙げる。

- ワーニングのグループ化：検出された多数のワーニングのうち、そのワーニングの原因を作っているコードが同じ関数や変数に起因している場合、それらのワーニングをグループ化し、グループの一つの代表例について手作

protocols that are an issue in multi-task scheduling.

3.3 Concurrent processing and multicore systems

Programming defects like deadlocks and race condition are apt to occur in synchronous processes in multitasking or multicore systems that execute concurrent processing. Yet when such defects are present, it is extremely difficult to detect their occurrence in synchronous processing either by unit testing or system testing. That is because there is an enormous number of cases concerning the timing for the occurrence of an event in a task when multiple tasks are being executed.

Even if an event or an interrupt that causes a problem can be identified, it is impossible to define the overall scheduling so as to know what task is executing what code at that time. Therefore, in practical terms, it is impossible to create test cases that would cover the timing of all events and interrupts.

Functions for detecting the data races and deadlocks described above have therefore been incorporated in ESCAT to detect problems occurring in synchronous processing in order to be able to deal with multicore systems.

3.4 Usability

3.4.1 Differential analysis report

When a static analysis is performed on a large-scale software system, a large number of warnings are usually detected. Such warnings must be reviewed manually on the basis of the software specifications and design documents. A review requires a lot of manpower and time, and there are many times when only specific engineers are capable of doing the work because it requires a thorough understanding of the system involved. For these reasons, reviews are an extremely troublesome activity for a software development team. Some of the warnings do not require revision of the software code and are referred to as false positives. There are many reasons why false positives occur. For example, one cause of a false positive is that the computer judges that the values of external inputs and signals or combinations of those values can occur, even though in the real world they are impossible. The reason for that is the computer has not been given any information regarding the values or the ranges of external inputs and signals. The limitations of static analysis tools regarding complex data structures and operations can be cited as another cause of false positives.

Meanwhile, the software may be modified due to requests for changes or to incorporate measures against defects. At such times, it is necessary to conduct a static analysis again of the entire software code, including parts that were not changed. The reason is that correct results cannot be obtained without analyzing places in the unchanged code that might be affected by the modifications. The results of the review already conducted for the previously detected warnings are effective for the places unaffected by the changes. General-purpose tools on the market have not

業で分析したら、この結果をそのグループの全てのワーニングに採用できるようにした。[8)]

- レビューに必要な情報の提供：レビューアがレビューを行う際、エラー箇所に至るパスや条件、関連するデータの情報などが必要になる。また、エラーの種類により必要な情報は異なる。エラーごとにレビューに必要な情報を自動的に集約提供する機能を実装し、レビューアの情報収集にかかる工数を低減させた。[12)]

4. 結　果

4.1 機能

図3は、ESCATに実装した機能のまとめである。標準的な静的解析機能も含め全31機能を実装した。[1)-11)]

4.2 性能

図4は、一つのソフトウェアの静的解析にかかる時間の計測結果である。ESCATは、百万行のソフトウェアを約1日で解析する。15万行のコードの解析で市販の汎用ツールと比較すると、約1／4の時間で解析ができる。

また、汎用ツールはコード規模が大きくなると解析速度が低下し現実的な時間内での解析ができなくなるが、ESCATは百万行レベルの大規模なコードでも解析が可能である。

Feature category	Number of features	Detailed feature list
Runtime error	8	・Zero division ・Array index out of bounds ・Overflow/Underflow ・Illegal dereference of pointer ・String buffer overflow ・Uninitialized variable ・Infinite loop ・Unreachable code
Function call check	2	・Recursive function ・Reentrant function
Variable read/write check	6	・Unused Assignment ・Read-only/Set-only/Unused variable ・Stale value usage ・Group variable ・Access inconsistency ・DFD reverse
Protocol check	4	・Flag handling ・Mismatched semaphore pair ・Module distance ・Inhibit system call in ISR
Worst-case analysis	2	・Maximum stack size ・Maximum steps
Race condition	6	・Shared variable ・Shared semaphore ・Semaphore locking period ・Semaphore deadlock ・Semaphore during tasklock period ・State machine livelock
Automotive specific function	2	・Sleep/Wakeup ・CC parameter
Differential report	1	
	Total 31	

図-3　ESCAT に実装した機能
Fig. 3　Features implemented in ESCAT

been sufficiently capable of identifying the range of influence of code changes and removing warnings in unaffected places. For that reason, such a function was developed for ESCAT. This function is accomplished by analyzing the influence of changes made in comparison with the present version of the code. It removes false positives detected in places unaffected by the changes made to the previous version.

3.4.2 Assistance for manual reviews

Static analysis tools usually generate large numbers of warnings of defects, which have to be reviewed manually. That work increases the cost of confirming software quality. We have developed several new methods of dealing with this issue. Two examples of them are explained below.

- Grouping of warnings: Among the many warnings detected, the codes causing some of them can be attributed to the same functions or variables. In those cases, such warnings are grouped and a representative example of the group is analyzed. The result can then be applied to all of the warnings in that group.[8)]
- Provision of information needed for a review: In order for a reviewer to conduct a review, it is necessary to have information on the conditions and path leading to the error location and the related data. The necessary information differs depending on the type of error involved. ESCAT is equipped with a function for automatically aggregating and providing the information necessary for reviewing each error. This function reduces the man-hours needed by a reviewer to gather the necessary information.[12)]

4. Results

4.1 Functions

Figure 3 summarizes the functions implemented in ESCAT. It features 31 functions, including standard static analysis capabilities. [1)-11)]

4.2 Performance

Figure 4 shows the measured results for the time needed to conduct a static analysis of one software code. ESCAT analyzed a software code consisting of one million lines in approximately one day. It can analyze a 150,000-line

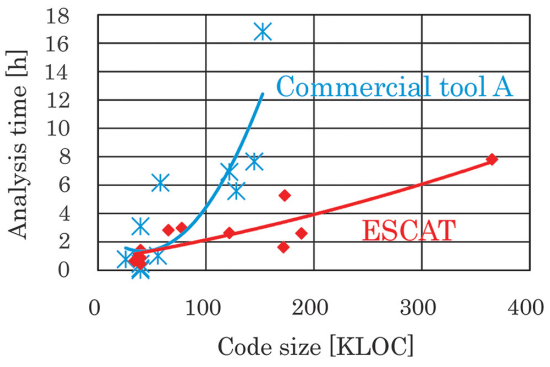


図-4　解析速度
Fig. 4　Analysis speed

5. 結　論

自動運転やコネクティドカーなど、クルマに求められる機能はますます拡大している。同時に、車載ソフトウェアの規模、複雑さも膨れ上がっており、ソフトウェアに起因する不具合も増加している。加えて、マルチコア、ハイパーバイザ、分散処理などの、ハードウェアとソフトウェアに関連するイノベーションは、更なる課題を引き起こしている。

ソフトウェアの不具合を流出させないためには、動的なテストだけでなく静的な解析が不可欠であるが、ソースコードの規模と複雑度の増大により、手作業でソースコードをチェックすることは不可能になっている。ソースコードを検証するための静的解析ツールの機能、及び性能の更なる改良は不可欠である。先進的な静的解析ツールは、その他の品質確認の方策と組み合わせて使用することで、市場不具合発生のリスクを大幅に減少させることができる。

6. 参 考 文 献

1) 長谷川美和子：ソフトウェア検査装置、ソフトウェア検査方法、ソフトウェア検査プログラム、特許第6037034号（2016.11.11）.
2) 長谷川美和子：データ更新漏れ検査装置、データ更新漏れ検査方法、データ更新漏れ検査プログラム、特許第5967225号（2016.07.15）.
3) 長谷川美和子：排他制御検査装置、排他制御検査方法、排他制御検査プログラム、特許第5979250号（2016.08.05）.
4) 市川智：ソフトウェア検査装置、ソフトウェア検査方法、ソフトウェア検査プログラム、特許第6004110号（2016.09.16）.
5) 市川智：フラグアクセス不具合検査装置、フラグアクセス不具合検査方法、フラグアクセス不具合検査プログラム、特許第5962779号（2016.07.08）.
6) 市川智：変数アクセス一貫性検査装置、変数アクセス一貫性検査方法、変数アクセス一貫性検査プログラム、特許第6015778号（2016.10.07）.
7) K. Shrawan: System and method for analysis of a large code base using partitioning. U.S. Patent 8, 612, 941（2013.12.17）.
8) M. B. Tukaram et al.: System and method to provide grouping of warnings generated during static analysis. U.S. Patent 9, 384, 017（2016.07.5）.
9) M. B. Tukaram et al.: Method and system for verifying sleep wakeup protocol by computing state transition paths. U.S. Patent 9, 141, 511（2015.09.22）.

code in about one-fourth the time needed by a general-purpose tool on the market. In addition, the analysis speed of the general-purpose tool declined as the scale of the code increased, making it impossible to complete an analysis within a realistic amount of time. ESCAT is capable of analyzing even large-scale codes on the order of one million lines.

5. Conclusion

The functionality required of vehicles is increasingly expanding as typified by autonomous driving and vehicle connectivity. At the same time, the scale and complexity of onboard software are growing rapidly, and troubles originating in software code are also increasing. In addition, innovations in hardware and software, such as multicore systems, hypervisor and distributed processing, are giving rise to new problems.

In order to prevent software defects from leaving the factory, static analysis is indispensable, in addition to dynamic testing. The increasing scale and complexity of source code have made it impossible to check the code manually. Further improvement of the functions and performance of static analysis tools is essential for the validation of source code. The use of advanced static analysis tools in combination with other quality assurance measures can markedly reduce the risk of problems occurring in real-world vehicle use.

6. References

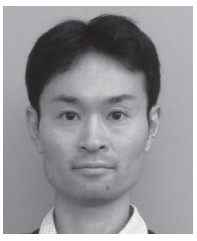
1) M. Hasegawa: Software inspection system, software inspection method, and software inspection program. Japan Patent 6037034 (2016.11.11).
2) M. Hasegawa: Inspection system of missing data update, inspection method of missing data update, and inspection program of missing data update. Japan Patent 5967225 (2016.07.15).
3) M. Hasegawa: Inspection system of exclusion control, inspection method of exclusion control, and inspection program of exclusion control. Japan Patent 5979250 (2016.08.05).
4) S. Ichikawa: Software inspection system, software inspection method, and software inspection program. Japan Patent 6004110 (2016.09.16).
5) S. Ichikawa: Inspection system of flag access defect, inspection method of flag access defect, and inspection program of flag access defect. Japan Patent 5962779 (2016.07.08).
6) S. Ichikawa: Inspection system of variable access inconsistency, inspection method of variable access inconsistency, and inspection program of variable access inconsistency. Japan Patent 6015778 (2016.10.07).
7) K. Shrawan: System and method for analysis of a large code base using partitioning. U.S. Patent 8, 612, 941 (2013.12.17).
8) M. B. Tukaram et al.: System and method to provide grouping of warnings generated during static analysis. U.S. Patent 9, 384, 017 (2016.07.5).

10）R.Venkatesh et al.: Resolving non progression of state machines. India Patent 1236/MUM/2013 (2013.03.28).
11）M. Ravi et al.: System and method for identifying source of run-time execution failure. U.S. Patent Application No. 14/037, 758.
12）M. B. Tukaram: System and method to facilitate a user interface enabled review of static analysis warnings. U.S. Patent 9, 201, 765 (2015.12.01).

9) M. B. Tukaram et al.: Method and system for verifying sleep wakeup protocol by computing state transition paths. U.S. Patent 9, 141, 511 (2015.09.22).
10) R.Venkatesh et al.: Resolving non progression of state machines. India Patent 1236/MUM/2013 (2013.03.28).
11) M. Ravi et al.: System and method for identifying source of run-time execution failure. U.S. Patent Application No. 14/037, 758.
12) M. B. Tukaram: System and method to facilitate a user interface enabled review of static analysis warnings. U.S. Patent 9, 201, 765 (2015.12.01).

■著者／ Author(s)■

長谷川　美和子
Miwako Hasegawa



市　川　　智
Satoshi Ichikawa

サービス指向アーキテクチャとクルマへの応用

Service-oriented Architecture and its Application to Vehicles

渡 辺 敏 之*
Toshiyuki Watanabe

抄　　録　サービス指向アーキテクチャは、2000年代前半に、IT業界で注目されるようになった大規模ソフトウェアシステムの構築手法である。特に、クライアントサーバモデルでコンピュータネットワークを構築する際に、有効であるとされている。一方、自動車向けの車載制御システムは、これまでなるべく個々のシステム内で制御を完結させることが多く、車内LAN（Local Area Network）で通信されるデータは、クルマの状態を示すデータの交換が主であり、サービスという概念は導入されてこなかった。しかし、クルマがオフボードサーバに接続されるコネクティドカーのシステムにおいては、車載の各システムがサービスを提供するというサービス指向アーキテクチャとの親和性が良い。ここでは、コネクティドカーにサービス指向アーキテクチャを適用した場合のケーススタディの一例について述べる。

Summary　Service-oriented architecture is a large-scale software development method that attracted attention in the IT industry in the 2000s. Service-oriented architecture is especially effective for developing a computer network system using a client-server model. In contrast, automotive embedded systems have mainly been completed within each system, and data exchanged in a vehicle network have principally concerned the vehicle status. So the idea of service was not considered. However, for a connected vehicle system, offboard servers are connected to communicate with the onboard vehicle system. Therefore, service-oriented architecture is well-suited to a connected vehicle system. This article describes an example of the application of service-oriented architecture to a connected vehicle system.

Key words : *Computer Application, automotive electronics, connected car, service-oriented architecture, computer*

1. は じ め に

サービス指向アーキテクチャ（SOA）とは、“サービス”という単位でソフトウェアを実装し、これを組み合わせてシステムを作り上げていくという考え方である。サービス指向アーキテクチャは、2000年代前半にIT業界で注目されるようになった大規模ソフトウェアシステムの構築手法である[1)]。一方、車載ソフトウェアは、カーナビゲーションなどを除き、クルマを使用するカスタマを直接意識して開発してきておらず、このため、自動車メーカ側も“サービス”を提供しているという考え方でソフトウェアを作ることは無かった。しかし、コネクティドカーにおいては、サービス指向アーキテクチャを意識した車載ソフトウェア構造を用意することで、インターフェースを標準化し、効率的で高品質のシステムが提供できるようになる。そこで、今回、既存の車載ソフトウェアにサービス指向アーキテクチャを導入した場合、どのような影響が起こるかを検討した。

1. Introduction

Service-oriented architecture (SOA) [1)] refers to the concept of implementing a software program for each service involved and combining them to create a large system. This approach to the development of large-scale software systems attracted attention in the IT industry in the first half of the 2000s. In contrast, automotive embedded software has not been developed heretofore with vehicle customers directly in mind, except for car navigation systems, among others. Accordingly, vehicle manufacturers have not developed onboard software from the perspective of being service providers. However, with regard to connected vehicles, the creation of onboard software structures based on an awareness of SOA results in a standardized interface, enabling high-quality systems to be provided efficiently.

In this study, we examined how existing onboard software might be influenced by introducing the concept of SOA.

* ソフトウェア開発部／Software Engineering Department

2. クルマにおける“サービス”

サービス指向アーキテクチャでは、まず“サービス”を定義しなくてはならない。コネクティドカーにおけるサービスとしては、一例として以下のようなサービスが考えられる（表1）。

(1) ドアを開ける

(2) 目的地まで行ってくれというカスタマの要求にこたえて、自動運転する

ソフトウェアとしては、かなりサイズが異なるが、カスタマにとっては、“ドアを開ける”ことも“自動運転する”ことも、それぞれサービスにあたる。このように、カスタマがクルマに要求する作業を整理して“サービス”として定義した。以下は、その一例である。

3. 標準化されたインターフェースの導入

サービス指向アーキテクチャでは、標準化されたインターフェースの導入が重要である。図1に標準化されたインターフェースの一例を示す。

コネクティドカーでは、カスタマ要求の発信元として、いくつかのデバイスが考えられる。従来の“車載のスイッチ操作”による要求発信をはじめ、携帯電話や音声認識エンジンからの要求発信、またオペレータによる要求発信などが考えられる。これに対し、要求を受けてサービス処理を行う側は“標準化されたインターフェース”を提供し、“標準化されたインターフェース”を用いて要求発信することを求める。

サービス処理側では、複数の要求が同時に発信されることも想定し、優先順位を決定する“ディサイダ”の導入が必要になると考えられる。

4. オフボード情報提供サービス

コネクティドカーのサービスの中には、インターネット上の情報など、オフボード情報をカスタマに提供するサービスがある。これらのサービスに、サービス指向アーキテクチャを適用した場合の構造を図2に示す。

表-1　サービスの事例
Table 1　Examples of services

	サービス内容	Service contents
1	トランクを開ける	Open the trunk
2	ランプを点灯する	Turn on the lights
3	ワイパを動作させる	Activate the wipers
4	窓を開ける	Open the window
5	室内を冷やす／暖める	Cool down/Warm up the cabin
6	ホーンを鳴らす	Beep the horn
7	シートを倒す	Recline the seat

2. Provision of Service in Vehicles

In considering SOA, it was first necessary to define service. The following are examples of conceivable services that might be provided in connected vehicles (Table 1).

(1) Opening the door

(2) Autonomous driving in response to a customer's request to be taken to a certain destination

From the customer's standpoint, both opening the door and autonomous driving correspond to services, though the size of the software involved would differ considerably. Accordingly, service was defined in general as an action that a customer requests a vehicle to perform. Examples of this are described below.

3. Implementation of a Standardized Interface

Implementation of a standardized interface is a key element of SOA. Figure 1 shows an example of a standardized interface.

Several devices can be considered for generating a customer's requests in a connected vehicle. These include requests made by operating conventional onboard switches, requests made via a mobile phone or a voice recognition engine, and requests made through an offboard operator, among other ways. A standardized interface is provided for receiving and processing such requests, and customers are asked to make requests through the standardized interface. In order to process service requests, it is assumed that a “decider” will have to be provided to determine the order of priority in cases where multiple requests are issued simultaneously.

4. Services for Providing Offboard Information

Provision of offboard information to customers such as information from the Internet is one of the services available in connected vehicles. Figure 2 shows an example of the configuration created when SOA is applied to such services.

Services are already being provided that deliver

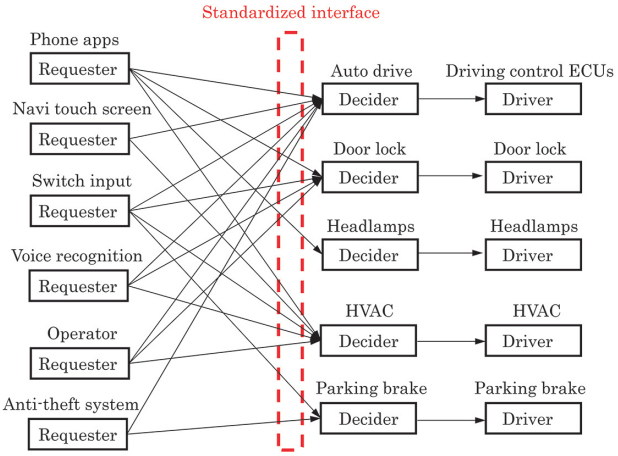


図-1　標準化されたインターフェースの一例
Fig. 1　An example of a standardized interface

自動車メーカのデータセンタやサードパーティのウェブサイトから情報を提供するサービスはすでに行われている。これに対し、要求の発信元は、ナビゲーションの画面操作以外にも、音声認識などがある。サービス指向アーキテクチャにより、標準化されたインターフェースを提供することで、統一的に要求の処理を行うことができる。

5. 新たなサービスの構築

サービス指向アーキテクチャに基づき、標準化されたインターフェースを導入することで、個々のサービスを駆動することができることは、これまでに述べた。これらのサービスを複数個組み合わせることで、更に高度なサービスを構築することができる。

一例として、カーシェアサービスを考えよう。カーシェアの場合、カスタマが予約した時間に応じて、携帯電話からの要求に基づき、ドアロックを解除し、カスタマがクルマを使用できる状態にすることが可能となる。また、必要な情報をカーナビゲーションの画面に表示することもできるし、場合によっては、ネット経由で料金の支払いも実現する。標準化されたインターフェースを用いて、個々の作業をリクエストすることで、カーシェアサービス全体を構築することができる（図3）。

6. お わ り に

コネクティドカーを実現する場合、クルマはIoT（Internet of Things）デバイスと考えることができる。IoTデバイスであるクルマ側に、標準化されたインターフェースを導入する。このようにすると、個々のインターフェースを持つサービスを組み合わせることで、高度なサービスを実現することができる。つまり、一旦クルマ側に標準化されたインターフェースを導入すれば、クルマを販売した後も、その標準化されたインターフェースを用いることで、新たなサービスを提供することが可能となる。これがサービスインターフェース標準化のメリットである。サービス指向アーキテクチャを推進することで、標準化さ

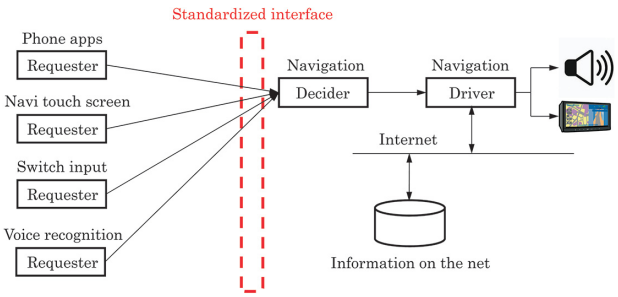


図-2　情報提供サービスへの適用の一例
Fig. 2　An example of applying SOA to information services

information from the data center of a vehicle manufacturer or from the website of a third-party supplier. Requests for such information can be issued through navigation system screen operations and also a voice recognition engine, among other ways. Applying SOA provides a standardized interface that enables integrated processing of requests.

5. Construction of New Services

The foregoing discussion has described how the implementation of a standardized interface based on SOA makes it possible to drive individual services. More advanced services can also be constructed by combining a number of these offerings.

Let us consider a car sharing service as one example. In the case of car sharing, a customer gains access to a vehicle by making a request via a mobile phone at the reserved time and has the door unlocked to enable use of the vehicle. Necessary information can be displayed on the car navigation screen, and it may also be possible to pay the charge via the Internet in some cases. A total car sharing service can be constructed by making requests for individual service operations through a standardized interface (Fig. 3).

6. Conclusion

When connected vehicles are made available, they can be considered Internet of Things (IoT) devices. As IoT devices, vehicles will be equipped with a standardized interface. As a result, more advanced services will be achieved by integrating individual services that have their own interface. Once vehicles are fitted with a standardized interface, new services can be created by using the standardized interface even after vehicles have been sold. This is one of the advantages of standardizing the service interface. The promotion of SOA in vehicles will lead to the provision of standardized connected services with a high degree of extendibility.

7.References

1) K. Oba et al.: The Status Quo and Challenges of Service-Oriented Architecture (SOA) Based Application Design,

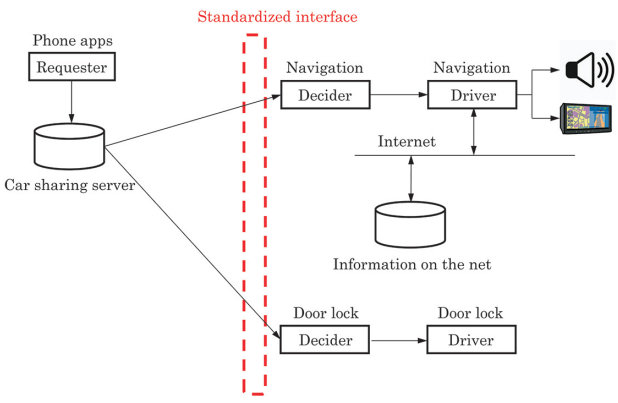


図-3　カーシェアへの適応の一例
Fig. 3　An example of SOA application to a car sharing service

れ拡張性の高いコネクティドサービスを提供していく。

7. 参 考 文 献

1）大場克哉ほか：サービス指向アーキテクチャ（SOA）に基づくアプリケーション設計の現状と課題、情報処理学会研究報告ソフトウェア工学（SE）、No. 75、Vol. 2005-SE-149、pp. 73-80（2005）.

IPS Japan, Technical Report on Software Engineering, No. 75, Vol. 2005-SE-149, pp. 73-80 (2005).

■著者／ Author(s)■

渡 辺 敏 之
Toshiyuki Watanabe

自動車におけるサイバーセキュリティ

Cybersecurity for Automotive Systems

宮 下 真 哲*
Masaaki Miyashita

抄　　録　自動車もインターネットに接続される時代となった。コネクティドカーの普及に伴い、自動車についても、サイバーセキュリティ対策が必要になってきている。サイバーセキュリティ技術には、変化し続けるハッカーの攻撃への対応と、日常的な監視が求められる。ここでは、一般的なサイバーセキュリティのフレームワークについて述べた後、日産自動車におけるサイバーセキュリティフレームワークの構築について述べる。

Summary　With the spread of connected vehicle technology, vehicles are also being connected to the Internet. This means that cybersecurity measures also need to be applied to vehicles. Because hacker attack methods are constantly changing, daily monitoring and prevention activities are required using cybersecurity technology. This article describes a general cybersecurity framework and also explains the construction of a cybersecurity framework at Nissan.

Key words : *Computer Application, automotive electronics, cyber security, IoT, computer, software*

1. は じ め に

近年、自動車がインターネットに接続されるようになり、ハッカーによるアタックが現実のものになりつつある。特に、2015年にラスベガスで開催されたBlack Hatカンファレンス[1]では、ホワイトハッカーによるJeepへの攻撃の事例が発表され、これに対し、リコール処置がとられる事態となった。今後、コネクティドカーが普及するにつれ、サイバーセキュリティ対策は必須の技術となっている。

このようなリスクへの対処を強化するために、米国においては、2013年2月12日にサイバーセキュリティに関する大統領令第13636号「Improving Critical Infrastructure Cybersecurity（重要インフラのサイバーセキュリティの向上）」[2]が発布された。この大統領令は、企業におけるサイバーセキュリティのリスク管理を支援し、業界標準およびベストプラクティスをまとめた自主参加型で、リスクベースアプローチに基づく「サイバーセキュリティフレームワーク」を策定することを要求している。

ここでは、標準的なサイバーセキュリティフレームワークについて述べ、そのうえで実際の日産における導入ケースについて解説する。

1. Introduction

With vehicles being connected to the Internet in recent years, attacks by hackers are starting to become a reality. Notably, an example of an attack on a Jeep by white hat hackers was presented at the Black Hat USA security conference[1] of white hat hackers in Las Vegas in 2015. That incident led to a product recall to fix the software bug. As connected vehicles become more prevalent in the future, cybersecurity measures will be an indispensable technology.

To strengthen measures against such risks, Executive Order 13636 (Improving Critical Infrastructure Cybersecurity)[2] was issued on February 12, 2013 concerning cybersecurity in the U.S. That Executive Order has provided support for cybersecurity risk management by private-sector companies and mandates the establishment of a cybersecurity framework on the basis of risk-based approaches with voluntary participation for the establishment of industry standards and promotion of best practices.

This article describes a general cybersecurity framework and then explains the efforts being made for the actual implementation of such measures at Nissan.

2. Cybersecurity Framework

Figure 1 shows an example of a general cybersecurity framework. Cybersecurity measures are

*ソフトウェア開発部／Software Engineering Department

2. サイバーセキュリティフレームワーク

図1に一般的なサイバーセキュリティフレームワークを示す。通常のサイバーセキュリティ対策では、(1) リスク評価→ (2) 多重レイヤ防御→ (3) 侵入の早期検出→ (4) 迅速な対応→ (5) ビジネス継続、の五つの活動を繰り返し実行する。

(1) 予測：リスク評価

コネクティドカーの場合はオンボード側、オフボード側の両方が対象になる。対象のシステム全体の中で、ハッカーの攻撃の侵入口となり得る部位をアタックサーフェスとして特定する。この侵入口からどのような攻撃を受けるか、その難易度と影響とを評価して対策の要否を決定する。

(2) 保護：多重レイヤ防御

リスク評価で抽出されたリスクに対する保護対策を検討する。セキュリティ対策に完璧はあり得ないので、最初の保護対策を突破された場合にも、すぐにシステムが致命的な状態に陥らないようにするために、多重レイヤの防御対策を施す。

(3) 検出：侵入の早期検出

前述した通り、セキュリティ対策に完璧なものはないので、侵入される恐れは常に存在する。侵入された場合、早期に侵入を検出することが肝要である。侵入検知ができず、多重レイヤ防御の各層が突破された場合、次の層の突破までハッカーは時間をかけて侵入を試みることができ、多重レイヤ防御の効果が大きく失われてしまう。したがって、侵入の早期検知機能を用意する。

(4) 応答：迅速な対応

侵入が検知されたら、迅速に侵入方法を特定し、脆弱性対策を打つ。

generally implemented through a repeated cycle of five activities: (1) risk assessment, (2) multi-layer protection, (3) early detection of intrusion, (4) quick response, and (5) business continuity.

(1) Prediction: risk assessment

In the case of connected vehicles, cybersecurity involves both onboard and offboard measures. Points of intrusion by a hacker attack in an overall targeted system are identified as an attack surface. An assessment is made of the types of attacks that might come through the points of intrusion, their relative difficulty and impact, and a decision is made concerning the necessity of security measures.

(2) Protection: multi-layer protection

Protection measures are examined against the risks identified in the risk assessment. Because no security measures provide perfect protection, multi-layer protection measures are applied. That prevents a system from immediately falling into a fatal condition in the event the first protection measures are breached.

(3) Detection: early detection of intrusion

As mentioned above, because there are no perfect security measures, there is a constant threat of intrusion. If an intrusion occurs, it is essential to detect it immediately. If not detected early and the individual layers of the multi-layer protection begin to be breached, the hacker can spend time trying to breach the next layer, so multi-layer protection losses much of its effectiveness. Therefore, a function for early detection is provided.

(4) Response: quick response

If an intrusion is detected, a method against it must be identified promptly and a countermeasure implemented against the vulnerability.

(5) Recovery: business continuity

Implementing a measure against the vulnerability will return the system to a normal state, enabling service to be provided.

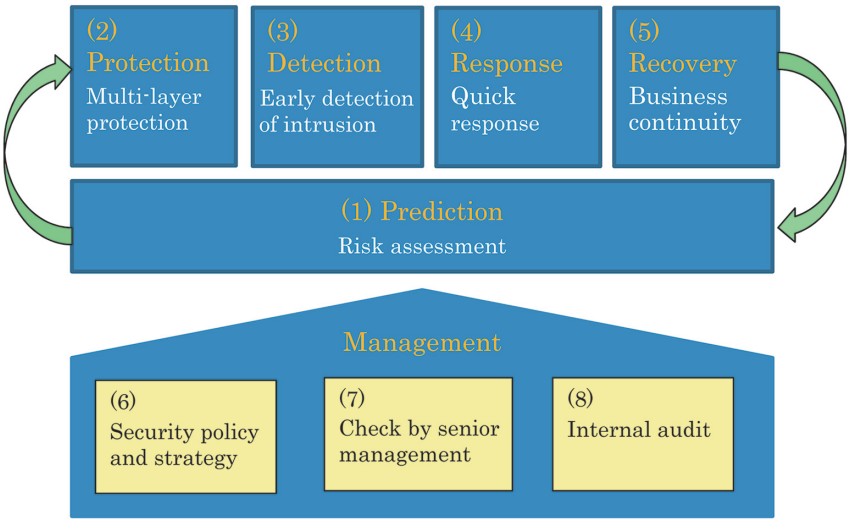


図-1　一般的なサイバーセキュリティフレームワーク
Fig. 1　General description of a cybersecurity framework

(5) 回復：ビジネス継続

脆弱性対策を打つことで、システムを正常な状態に回復し、サービスを提供できるようにする。

上記の（1）から（5）までの活動を繰り返し実行することで、サイバーセキュリティ対策を実現することができる。一方、マネジメント層では、セキュリティ対策のサイクルを回し続けられるように管理を行う。

(6) セキュリティのポリシと戦略

マネジメントを行う場合には、まずセキュリティのポリシと戦略を決定する必要がある。これに基づき、プロセスを定義運用し、セキュリティ対策を推進する。

(7) シニアマネジメントによるチェック

サイバーセキュリティ対策が適切に実施されるために、定期的にシニアマネジメントが実施状況をチェックする。

(8) 内部監査

セキュリティ対策が定められたプロセスにしたがって運用されていることを確かめるために、内部監査を定期的に実施する。

3. 日産におけるサイバーセキュリティフレームワークの実施状況

2章で一般的なサイバーセキュリティフレームワークについて述べたが、本章では、日産におけるサイバーセキュリティフレームワークの実施状況について具体的に述べる。

(1) 予測：リスク評価

クルマに対する攻撃を考えた場合、主には、車両外部からの無線通信がアタックサーフェスとなる。ITSシステ

Cybersecurity measures can be put in place by repeating this cycle of activities from (1) to (5). At the management level, meanwhile, the following are managed so that the cycle of security measures can be continued.

(6) Security policy and strategy

In order to manage the cycle, it is first necessary to decide a security policy and strategy. On that basis, the process is then defined and operated and security measures are promoted.

(7) Check by senior management

Senior management periodically checks the implementation status of cybersecurity measures to ensure that they are being carried out properly.

(8) Internal audit

Internal audits are conducted periodically to confirm that security measures are operating according to the established process.

3. Status of Cybersecurity Framework Implementation at Nissan

A general security framework was explained in Section 2. This section describes in detail the status of Nissan's implementation of a cybersecurity framework.

(1) Prediction: risk assessment

When considering hacker attacks against vehicles, the principal attack surface is via wireless communication from outside the vehicle. It is necessary to envision attacks coming through information devices, including an ITS system, telematics system, navigation system and so on. In addition, it is also necessary to envision attacks through paths connected to external networks indirectly. The following are some examples.

- Charging terminal connected to an electric vehicle (EV) charging station

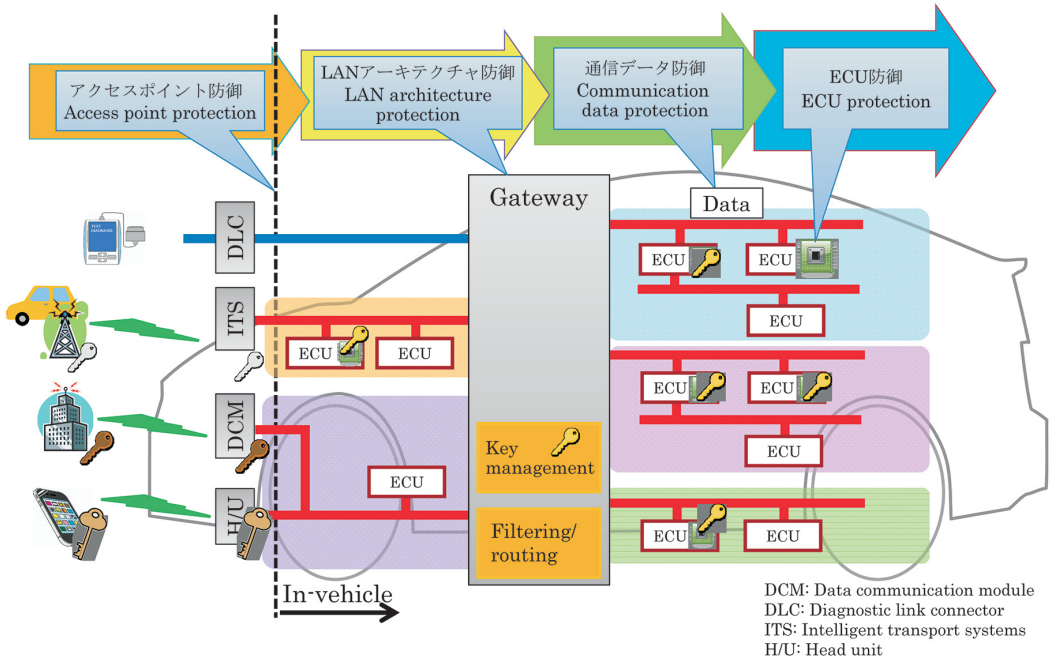


図-2　自動車における多重防御の適用例

Fig. 2　An example of multi-layer cybersecurity protection for vehicle systems

ム、テレマティクス、ナビゲーションなどの情報系機器を介したアタックを想定するべきである。これに加えて、間接的に外部ネットワークに接続される経路への攻撃も想定する必要がある。下記はその例である。

- 電気自動車（EV）用の充電ステーションと接続する充電端子
- スマートフォンやメモリデバイスを接続するUSBポート
- アフターセールスデバイスや整備業者がパソコンを接続する診断通信コネクタ

こうした経路では接続されるデバイスが、ハッカーの踏み台とされている可能性を考慮しなければならない。

(2) 保護：多重レイヤ防御

上記のリスク分析の結果として、情報系機器や診断通信ラインを介した攻撃がアタックサーフェスとなる。これに対し、多重レイヤでの防御対策が重要である。図2に適用例を示す。アタックサーフェスとなるアクセスポイントの各機器での防御が第1レイヤであるが、これに加えて、通信ゲートウェイを用いた防御、CAN（Controller Area Network）などの車内通信ラインにおける通信データに対する防御、そして、車両の走行を制御する各ECU（Electronic Control Unit）に対する防御の三つのレイヤを設けている。

(3) 検出：侵入の早期検出

保護が必要な通信データや各ECUのソフトウェアには、万が一改ざんが行われた場合にこれを検知し、通知することができるように、認証機能を用意しなくてはならない。たとえば、通信データにはメッセージ認証符号(MAC)を付加し、正しい相手からの通信でないことを検知した場合はこの通信データを廃棄し、また、改ざんが行われたことをセキュリティオペレーションセンタに通知する。このようにすることで、改ざんが行われたことを早期に検出できるようにしている。

(4) 応答：迅速な対応

改ざんが行われた場合には、情報がセキュリティオペレーションセンタに通知される。ここでは、アップロードされた情報に基づき、改ざんの原因や対策が検討、実施される。侵入経路によっては、サービスを中断し、車両をネットワークから切断する必要が発生する場合もある。

(5) 回復：ビジネス継続

セキュリティ対策が完了し、車両をネットワークに再接続して、通常のサービス体制に復帰できる状態になったら、システムを復旧させサービスを再開する。

(6) マネジメント

以上の（1）から（5）までの方策を繰り返し実行できるようにするため、セキュリティポリシの確立、セキュリティ戦略の策定を行い、これに基づき、日産では、現在も順次サイバーセキュリティ体制を構築している。

- USB port for connecting a smartphone or memory device
- After-sales service device or diagnostic communication connector for connecting a PC by a service technician

It is necessary to consider the possibility that devices connected to such paths might be a springboard for hackers.

(2) Protection: multi-layer protection

The results of the foregoing analysis indicate that attacks via information devices and diagnostic communication lines constitute an attack surface. Multi-layer protection measures are important against such attacks. Figure 2 shows examples of the protection applied. The first layer is the protection provided for various devices at access points that become the attack surface. In addition to that, protection is also provided at three layers: protection is provided using the communication gateway, protection is provided for communication data transmitted through an onboard communication line such as the controller area network (CAN), and protection is provided for each of the electronic control units (ECUs) that control the operation of a vehicle.

(3) Detection: early detection of intrusion

An authentication function must be provided so that if by some chance communication data or ECU software requiring protection is falsified, it can be detected and notified. For example, a message authentication code (MAC) can be added to communication data; if it is detected that a message is not from a proper party, the communication data are destroyed. An occurrence of falsification is reported to the cybersecurity operations center. These security measures enable early detection of any falsification.

(4) Response: quick response

The security operations center is notified in the event a falsification has occurred. Based on the uploaded data, the center investigates the cause of the falsification, examines a countermeasure and implements it. Depending on the path of the intrusion, it may also be necessary in some cases to discontinue service or to disconnect the vehicle from the network.

(5) Recovery: business continuity

After security measures have been implemented and the vehicle has been reconnected to the network so that the normal service system can be recovered, the system is restored and service is resumed.

(6) Management

Nissan has established a security policy and formulated a security strategy so that the cycle of activities from (1) to (5) above can be repeatedly carried out. Based on that policy and strategy, Nissan is currently taking further steps in turn to build a cybersecurity framework.

4. Conclusion

Cybersecurity measures are not temporary activities, but rather it is essential to promote such

4. おわりに

サイバーセキュリティ対策は、一過性の活動ではなく、継続的な活動の推進が重要であり、日産自動車では、これを踏まえて、今後も活動を推進していく。

5. 参考文献

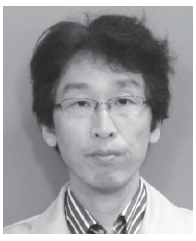
1）BlackHat 2015 : http://www.blackhat.com/us-15（参照日：2018年9月25日）.
2）米国国立標準技術研究所（独立行政法人情報処理推進機構）：重要インフラのサイバー・セキュリティを向上させるためのフレームワーク第1版、https://www.ipa.go.jp/ files/000038957.pdf（参照日：2018年9月25日）.

activities continuously. Based on this recognition, Nissan will continue to promote cybersecurity activities in the years ahead.

5. References

1) Black Hat 2015: http://www.blackhat.com/us-15/ (as of September 25, 2018).
2) National Institute of Standards and Technology, Framework for Improving Critical Infrastructure Cybersecurity, Version 1.0, National Institute of Standards and Technology, https://www.nist.gov/sites/default/files/documents/cyberframework/cybersecurity-framework-021214.pdf (as of September 25, 2018).

■著者／ Author(s)■



宮 下 真 哲
Masaaki Miyashita

SATソルバとその応用例

SAT Solver and Application Examples

井　野　淳　介*
Junsuke Ino

抄　　録　　SATソルバは、充足可能性問題（SAT）を解くための人工知能（AI）ツールである。充足可能性問題とは、いろいろな制約条件がある場合に、それらをすべて満たす解を見つける問題のことである。1990年代前半までは、コンピュータの性能面の制約やSATソルバ自体の性能の問題から、実用化は困難とされていたが、近年コンピュータとツール両者の性能向上が図られ、実用化されるようになった。一方、現在自動車メーカは数百の実験を実施している。コスト削減や開発期間短縮のため、実験車配車日程の最適化は非常に重要である。今回、実験車配車計画作成にSATソルバを応用するトライアルを行い、自動実験車配車日程作成の実用化の見通しを得た。

Summary　　A SAT solver is an artificial intelligence (AI) tool for solving the satisfiability problem (SAT). The satisfiability problem involves finding a solution to a given set of logical constraints. Until the first half of the 1990s, it was said that it was difficult to apply a SAT solver to real-world problems due to the limitations of CPU power or SAT solver performance. However, because both CPU power and SAT tool performance have been improved today, SAT solvers are now applicable to real-world problems. Vehicle manufacturers conduct hundreds of tests, so an optimized test vehicle schedule is very important for reducing costs and time to market. An attempt was made in this work to apply a SAT solver to the creation of a test vehicle schedule. The results showed the feasibility of creating a tool for automatically generating a schedule for allocating test vehicles.

Key words : *Computer Application, AI, satisfiability problem, SAT solver, computer*

1. はじめに

SATソルバは、充足可能性問題（Satisfiability problem、SAT）を解くための人工知能（Artificial Intelligence、AI）ツールである。SATとは、いろいろな制約条件がある場合に、それらをすべて満たす解を見つける問題のことである。SATソルバを用いる場合、制約条件を論理式に変換し、これをSATソルバに入力する必要がある。今回SATソルバを、実験車の配車計画を作成することに応用できないか検討した。以下に、その内容を紹介する。

2. SATとは

ブーリアン型とは、プログラミングにおいて「真（true）」と「偽（false）」の2種類の値をとるデータ型の一つである。ブーリアン型のSATソルバは、1990年代中期以降、飛躍的に改善が図られた。その動機としては、SATが自動車、航空、生物学などを含む多くの領域で、現実の問題に適用できるためである。SATは制約問題、組み合わせ問題、最適化問題を解くためのフレームワークとして広く用いられている。

1. Introduction

A SAT solver is an artificial intelligence (AI) tool for solving the satisfiability problem (SAT). SAT refers to the problem of finding a solution that satisfies all the various constraints involved. In order to use a SAT solver, the constraints must be converted to logical expressions and input to the solver. A study was undertaken to determine if a SAT solver could be applied to create a schedule for allocating test vehicles. This article describes the study details and results.

2. Overview of SAT

In programming languages, the Boolean data type refers to a data type that has two possible values denoted as true or false. Boolean SAT solvers have been dramatically improved since mid-1990s. The motivation for that was the broad applicability of SAT solvers to actual problems in many fields, including automobiles, aircraft and biological science, among others. SAT solvers are widely used as a framework for solving constraint problems, combinatorial problems and optimization problems.

Many actual problems can be modeled as a combination of logical expressions representing the constraints involved. For example, consider that a vehicle

*ソフトウェア開発部／Software Engineering Department

現実の問題の多くは、制約条件を表す論理式の組み合わせでモデル化することができる。たとえば、あるクルマがf_1、f_2、f_3、f_4、f_5の五つの機能（SATの用語では変数と呼ばれる）を持つ場合で、これらからカスタマは、さまざまな機能の組み合わせを選択できるとする。機能f_1はAT（Automatic Transmission）、機能f_2はMT（Manual Transmission）とする。これらの機能には制約条件があり、これは論理式で定義することができる。この場合、制約条件としては、“カスタマはATとMTのうち、いずれか一つを選択することができるが、両方を選択することはできない”ということが挙げられる。

この事例の制約条件を表す論理式は、$C = \{f_1\ or\ f_2,\ not\ (f_1\ and\ f_2)\}$である。ここで、この式の最初の項は、$f_1$と$f_2$のいずれかが選択されることを表しており、二つ目の項は、同時にf_1とf_2を選択することができないことを表している。$C = true$とできるf_1とf_2の値が、Cの解である。この例では、解とは、購入可能なクルマの機能の組み合わせである。つまり、$f_1 = true\ and\ f_2 = false$は解であるが、$f_1 = true\ and\ f_2 = true$は解ではない。

引き続き、残る機能f_3からf_5について、制約条件を論理式で表現する。たとえば、機能f_2が存在する場合、機能f_3または機能f_4が存在しなくてはならないこととする。これは$(f_2\ implies\ (f_3\ or\ f_4))$と表すことができる。また、機能$f_5$はオプション機能であるとする。この場合、$(f_5\ or\ (not\ f_5))$と表すことができる。これらの機能$f_1$から$f_5$までのすべての制約条件をまとめて、以下のように記述することができる。

- $\{f_1\ or\ f_2,\ not\ (f_1\ and\ f_2),\ (f_2\ implies\ (f_3\ or\ f_4)),\ (f_5\ or\ (not\ f_5))\}$

この論理式の解は、いずれも購入可能なクルマの機能の組み合わせであり、解の総数は、購入し得る機能の組み合わせの総数を意味する。そして、解が存在しない場合、制約条件は競合状態にあり、制約条件の定義に誤りがある可能性があることを指摘している。

3. SATソルバとは

SATソルバに制約条件の論理式を入力すると、SATソルバは自動的にその解を生成し、変数の値を回答する。SATは一般的に難しい問題であり、これを解くためのアルゴリズムは存在したが、その効率はあまり良くなかった。しかし今日では、いくつかの性能の良いSATソルバが開発されており、数百から数千の変数から構成される制約条件の論理式を解くことができる。このSAT技術の進歩により、問題を論理式で表現し、SATソルバを用いて解を得ることが現実に実用化されるようになった。たとえばZ3[1]やYices[2]などのライセンスフリーの性能の良いSATソルバが利用可能になっている。

has five functions denoted as f_1, f_2, f_3, f_4, and f_5 (which are called variables in SAT terminology) and that customers can select various combinations of these functions. We will let function f_1 denote an automatic transmission (AT) and function f_2 a manual transmission (MT). These functions are subject to constraints that can be defined in logical expressions. The following is an example of a constraint: “a customer can select either an AT or a MT, but cannot select both.”

The logical formula that expresses this constraint is: $C = \{f_1\ or\ f_2,\ not\ (f_1\ and\ f_2)\}$. The first term of this formula indicates that either f_1 or f_2 can be selected. The second term indicates that both f_1 and f_2 cannot be selected simultaneously. The values of f_1 and f_2 that produce $C = true$ represent solutions to C. In this example, solutions mean the combinations of vehicle functions that can be purchased. In other words, $f_1 = true\ and\ f_2 = false$ is a valid solution, but $f_1 = true\ and\ f_2 = true$ is not a solution.

Next, we will express the constraints of the remaining functions f_3 to f_5 in logical formulas. For example, if function f_2 exists, it is assumed that functions f_3 and f_4 must also exist. This can be expressed as $(f_2\ implies\ (f_3\ or\ f_4))$. Function f_5 is treated as an option. In this case, it can be expressed as $(f_5\ or\ (not\ f_5))$. All the constraints of functions f_1 to f_5 can be summarized and described as shown below.

- $\{f_1\ or\ f_2,\ not\ (f_1\ and\ f_2),\ (f_2\ implies\ (f_3\ or\ f_4)),\ (f_5\ or\ (not\ f_5))\}$

The solution to this logical formula in every case is the combination of vehicle functions that can be purchased. The total number of solutions represents the total number of purchasable combinations of functions. If a solution does not exist, it indicates that the constraints are in a state of conflict and that there may be an error in the constraint definitions.

3. Overview of SAT Solvers

When the logical formulas of the constraints are input to a SAT solver, it automatically generates a solution and responds with the values of the variables. In general, SAT is a difficult problem for which there were algorithms for solving it, but they were not very efficient. However, several high-performance SAT solvers have been developed nowadays that can solve logical formulas of constraints composed of several hundred to several thousand variables. This progress of SAT technology has resulted in the actual implementation of SAT solvers for obtaining solutions to problems expressed in logical formulas. For example, it is now possible to use high-performance SAT solvers like the Z3 theorem prover[1] and the Yices SMT solver[2] on a license-free basis.

4. Simple Application Example: Using a SAT Solver to Set a Meeting Time

This section describes an example of a SAT solver application as a simple tool for setting a meeting time.

4. 簡単な応用例：SATを用いた会議時間設定

ここでは、単純な会議設定ツールを用いて、SATソルバの応用例を紹介する。3人の人物がある日、以下の時刻{10, 11, 12, 13, 14, 15, 16, 17}から始まる1時間の会議を行いたい場合を考える。3人の人物は、変数p_1、p_2、p_3を用いて表現する。またここで、ブーリアン型の関数*isAvailable*(p_i, t)を考える。この関数は、時刻tから始まる1時間の間、人物p_iが会議可能の場合、*true*を返す。つまり、*isAvailable*(p_1, 10)は、人物p_1が10時から11時まで会議可能である場合に*true*を返す。

一方、今回の事例では、人物p_1はいつでも会議可能とする。また、人物p_2は11時、13時、15時、16時、17時に会議可能とする。そして人物p_3は10時、11時、13時、16時に会議可能とする。

これらの制約条件を論理式に変換し、SATソルバに入力することで解を求める。この会議可能な時間を見つけるという問題は、以下の論理式に変換することができる。

{(*isAvailable* (p_1,10) *and isAvailable* (p_2,10) *and isAvailable* (p_3,10))
or
(*isAvailable* (p_1,11) *and isAvailable* (p_2,11) *and isAvailable* (p_3,11))
or
……
(*isAvailable* (p_1,17) *and isAvailable* (p_2,17) *and isAvailable* (p_3,17))}

上記の論理式はより簡潔に、以下のように表すことができる。

{10 ≤ t *and* t ≤ 17 *and isAvailable*(p_1, t) *and*
isAvailable(p_2, t) *and isAvailable*(p_3, t)}

上記の論理式をSATソルバに入力すると、SATソルバはtの値を[10...17]の範囲で探索し、論理式を満足させる解を探す。この探索は、以下三つのいずれかの結論を出す。

- もし上記の論理式を満足させるtの値が存在する場合、SATソルバは、満足させる値の一つを回答する。この値は、論理式を満足させる任意のtの値であることに注意する必要がある。この値は、満足させる値の中で、最初、最後、最小、最大などであるかどうかはわからない。
- もし上記の論理式を満足させるtの値が存在しない場合、SATソルバは、“制約条件を満たす解は存在しない”というようなメッセージを回答する。
- もし上記の論理式がSATソルバにとって大きすぎる場合は、SATソルバは永遠に処理を続けるか、またはメモリ不足で実行を中止する。

上記の例では、三つの会議可能なタイムスロットがある。すなわち、11時、13時および16時である。SATソルバは、三つの会議可能なタイムスロットの内のいずれか一つを回答する。

Consider that three people want to hold a one-hour meeting on a certain day starting at {10, 11, 12, 13, 14, 15, 16, or 17:00}. The three people are represented by the variables p_1,p_2,p_3. We will also consider here the Boolean datatype function *isAvailable*(pi, t). This function returns *true* if person p_1 is available to attend the meeting for one hour starting at time t. In other words, *isAvailable*(p_1, 10) returns *true* if person p_1 can attend the meeting from 10:00 to 11:00.

On the other hand, let us assume in this example that person p_1 is available to attend the meeting at any time, that person p_2 is available at 11:00, 13:00, 15:00, 16:00 and 17:00, and that person p_3 is available at 10:00, 11:00, 13:00 and 16:00.

These constraints are converted to logical formulas and input into the SAT solver to find a solution. The problem of finding an available time for the meeting can be converted to the following logical formulas:

{(*isAvailable* (p_1,10) *and isAvailable* (p_2,10) *and isAvailable* (p_3,10))
or
(*isAvailable* (p_1,11) *and isAvailable* (p_2,11) *and isAvailable* (p_3,11))
or
……
(*isAvailable* (p_1,17) *and isAvailable* (p_2,17) *and isAvailable* (p_3,17))}

These logical formulas can be further simplified and expressed as:

{10 ≤ t *and* t ≤ 17 *and isAvailable*(p_1, t) *and*
isAvailable(p_2, t) *and isAvailable*(p_3, t)}

When this logical formula is input into the SAT solver, it searches for a value of t within the range of [10 ... 17] and seeks a solution that will solve the logical formula. This search produces one of the following three conclusions.

- If values of t exist that satisfy the logical formula above, the SAT solver returns one satisfactory value. It must be noted that this value is an arbitrary value of t that satisfies the logical formula. It is not known whether this value is the first, last, minimum or maximum, etc. value among those satisfying the formula.
- If a value of t satisfying the logical formula does not exist, the SAT solver returns a messaging that “a solution satisfying the constraints does not exist.”
- If the logical formula is too large for the SAT solver, the solver either continues processing forever or suspends processing because of insufficient memory.

In this example, three time slots are available for the meeting, namely, 11:00, 13:00 and 16:00. The SAT solver returns one of these three available meeting time slots.

5. Application to a Tool for Scheduling Test Vehicle Allocation

A study was made of the use of a SAT solver to develop a tool for automatically creating a schedule for

5. 実験車配車日程作成ツールへの応用

今回、SATソルバを用いて、実験車配車日程作成ツールを開発することを検討した。実験車の配車には、いくつかの制約条件がある。以下に、代表的な実験車に関する制約条件について説明する。

(1) 実験の前後関係指定

例：実験Aと実験Bには依存関係があり、実験Aが終了しないと実験Bを開始できない。

(2) 同一実験車指定

例：実験Cと実験Dは、同一実験車（001号車）を用いて実験しなくてはならない。

(3) 実験車占有

例:実験Eは、実験車（002号車）を占有して実験を行う。他の実験は、002号車を用いて実験することができない。

(4) 同一実験に対する設備制約

例：実験Fは、実験を行う設備は指定の実験設備(Lab#1)でなくてはならない。また、設備には同時に実施できる実験数に制約がある。

(5) 実験の優先順位

例：各実験項目に、優先順位を表す指数を割り付ける。優先順位の高い（指数の大きい）実験は、優先順位の低い実験より先に実施しなくてはならない。

(6) 輸送

例：実験車の国内輸送は、夜間に行うことで実験日程を損なうことなく実験できるが、国をまたがる実験車の輸送には、必要な輸送期間がかかる。

(7) 実験納期の種類

例：実験により、必要完了時期が異なる。実験Gはメタル手配期限までに完了しなくてはならないが、実験Hはトリム手配期限までに完了すればよい。

(8) 期間が特定される実験

例：寒地実験は、冬季の定められた期間のみでしか実験できない。

上記の制約条件を論理式で記述するために、実験Tについて、以下の表現を用いることとする。

- Veh(T)は、実験Tが行われる試作車を示す。
- Mod(Veh(T))は、実験Tに用いられる試作車両の仕様を示す。(車両バリエーション＝モデルの特定)
- Loc(T)は、実験Tが行われる実験場所･実験設備を示す。
- Days(T)は、実験Tの所要日数を示す。
- Pr(T)は、実験Tの優先順位を示す。
- PS(T)は、実験Tが専有実験の場合は1、専有実験でない場合は0とする。
- STR(T)とEND(T)は、実験Tの開始日と終了日を示す。

上記の制約条件は以下のように、論理式で表現するこ

allocating test vehicles. The allocation of test vehicles involves several constraints. Typical examples of constraints concerning test vehicles are explained below.

(1) Specification of the order of tests

Example: There is a dependency relationship between test A and test B. Test B cannot be started until test A has been completed.

(2) Specification of the same test vehicle

Example: Tests C and D must be conducted using the same test vehicle (vehicle No. 001).

(3) Test vehicle occupancy

Example: Test vehicle No. 002 is occupied for the conduct of Test E, so other tests cannot be conducting using this test vehicle.

(4) Facility constraints for the same test

Example: Test F must be conducted using a specified test facility (Lab #1). The number of tests that can be conducted at the same time with this facility is limited.

(5) Order of test priority

Example: An index showing the order of priority is assigned to every test item. Tests having a high priority (large index value) must be conducted before tests with a low priority.

(6) Transport

Example: Domestic transport of test vehicles is done at night so that tests can be conducted without affecting the test schedule. However, time is needed to transport test vehicles that are used at test facilities in other countries.

(7) Types of test deadlines

Example: The deadline for completing tests differs depending on the test. Test G must be completed by the deadline for procuring metals. It is all right if test H is completed by the deadline for procuring trim parts.

(8) Tests having a specified term

Example: Cold-weather tests can only be conducted during the interval specified for the winter season.

The following expressions are used concerning test T in order to describe the constraints above in logical formulas.

- Veh(T) indicates the prototype vehicle to be used in conducting test T.
- Mod(Veh(T)) indicates the specifications of the prototype vehicle to be used in conducting test T. (Vehicle variation specifies the model.)
- Loc(T) indicates the test site and test facility to be used in conducting test T.
- Days(T) indicates the number of days needed to conduct test T.
- Pr(T) indicates the priority of test T.
- PS(T) is denoted as 1 if test T is an exclusive test and as 0 if it is not.
- STR(T) and END(T) indicate the starting and ending dates.

とができる。一部の事例を示す。

- 言語表現：実験Aと実験Bには依存関係があり、実験Aが終了しないと実験Bを開始できない。

➢論理式：END(A)<STR(B)

- 言語表現：実験Cと実験Dは、同一実験車（001号車）を用いて実験しなくてはならない。

➢論理式：Veh(C)=Veh(D)=1

- 言語表現：実験Eは、実験車（002号車）を占有して実験を行う。他の実験は、002号車を用いて実験することができない。

➢論理式：PS(E)=1, Veh(E)=2

- 言語表現：実験Fは、実験を行う設備は指定の実験設備(Lab#1)でなくてはならない。

➢論理式：Loc(F)=Lab#1

また、“一般化された制約条件”は、以下のような論理式で表現できる。一部の事例を示す。

- “一般化された制約条件” − その実験が占有実験の場合、実験車はその実験のためだけに用いられる。

➢論理式：for all tests X, if PS(X) = 1, then for all tests Y, if (Y ≠ X), then Veh(Y) ≠ Veh(X)

- “一般化された制約条件” − 高い優先順位の実験は、他のより低い優先順位の実験が始まる前に実施されなくてはならない。

➢論理式：for all tests X, Y, if Pr(X) > Pr(Y), then END(X) < STR(Y)

以上すべての制約条件を論理式に変換し、SATソルバに入力して、解があるかを演算する。SATソルバは、これらのすべての論理式を満足させる変数値を探すことで、これらの制約条件を満足させる日程を得る。SATソルバは、一つでも与えられたすべての制約条件を満たすことができる場合は、実現可能な日程を回答する。制約条件を満足できない場合は、制約条件を満足させる日程が存在しないことを回答する。後者の場合、制約条件を緩和して、再度、制約条件を満足させる日程が無いかの検討を、SATソルバを用いて実施する。

6. 結　果

今回はSATソルバとして、Microsoft Researchが開発したZ3を用いた。そして、トライアルの結果、上記を含むすべての制約条件を論理式に変換し、SATソルバに供給することで、実験車の配車計画を自動的に作成できる見通しを得た。今後はこのツールを活用することで、更に車両仕様のバリエーション数と、必要な実験車台数を求めることで、仕様バリエーションと実験車台数のトレードオフ分析を行うことができるようになると考えられる。

Logical formulas of the constraints above can be expressed as shown in the partial examples below.

- Linguistic expression: There is a dependency relationship between test A and test B. Test B cannot be started until test A has been completed.

➢Logical formula: END(A)<STR(B)

- Linguistic expression: Tests C and D must be conducted using the same test vehicle (vehicle No. 001).

➢Logical formula: Veh(C)=Veh(D)=1

- Linguistic expression: Test vehicle No. 002 is occupied for the conduct of Test E, so other tests cannot be conducted using this test vehicle.

➢Logical formula: PS(E)=1, Veh(E)=2

- Linguistic expression: Test F must be conducted using a specified test facility (Lab #1).

➢Logical formula: Loc(F)=Lab#1

“Generalized constraints” can also be expressed in logical formulas as shown in the partial examples below.

- Generalized constraint: In the case of an exclusive test, the test vehicle will be used only for that test.

➢Logical formula: for all tests X, if PS(X) = 1, then for all tests Y, if (Y ≠ X), then Veh(Y) ≠ Veh(X)

- Generalized constraint: Tests having a higher priority must be conducted before other tests having a lower priority are started.

➢Logical formula: for all tests X, Y, if Pr(X) > Pr(Y), then END(X) < STR(Y)

All of the constraints above are converted to logical formulas and input into the SAT solver, which calculates whether a solution exists. The SAT solver searches for variable values that satisfy all the logical formulas. As a result, a schedule is obtained that satisfies all the constraints. The SAT solver returns an executable schedule if all the constraints can be satisfied, even if just one constraint is given. If the constraints cannot be satisfied, the SAT solver returns a message that a schedule satisfying the constraints does not exist. In the latter case, the constraints are eased and the SAT solver is used to examine once again whether a schedule satisfying the relaxed constraints exists.

6. Results

The Z3 theorem prover developed by Microsoft Research was used as the SAT solver in this study. All the constraints concerned, including those noted above, were converted to logical formulas and input into the SAT solver. The results of this trial showed the possibility of using the SAT solver to automatically generate a schedule for the allocation of test vehicles. It is planned to use this tool effectively in the future to find the number of test vehicles needed relative to the number of variations of the vehicle specifications. It is expected that this will enable us to analyze the trade-offs between the specification variations and the number of test vehicles needed.

7. おわりに

Z3をはじめとするSATソルバは、実用・応用可能な段階となってきている。一方、実社会にはSATが多く存在する。実験車の日程計画は、その一つであるが、これ以外にも、車両のワイヤハーネスのバリエーションとコストの問題など、多くの問題に適用可能である。業務の効率化を推進するため、SATソルバを各種の問題解決に応用していきたい。

8. 参考文献

1) The Z3 Theorem Prover, https://github.com/Z3Prover/z3（参照日：2018年7月15日）.
2) The Yices SMT Solver, http://yices.csl.sri.com/（参照日：2018年7月15日）.

7. Concluding Remarks

SAT solvers, notably the Z3 theorem prover, have reached the stage where they can be used in actual development work. Meanwhile, there are many SAT problems in the real world. Scheduling the allocation of test vehicles is just one example. SAT solvers are applicable to many other actual problems such as the issue of vehicle wiring harness variations and cost. It is planned to apply a SAT solver to resolve various types of problems in order to promote higher operational efficiency.

8. References

1) The Z3 Theorem Prover, https://github.com/ Z3Prover/z3 (as of July 15, 2018).
2) The Yices SMT Solver, http://yices.csl.sri.com/ (as of July 15, 2018).

■著者／ Author(s)■

井野淳介
Junsuke Ino

ソフトウェア品質改善のためのサプライヤ管理活動

Activities to Improve Software Quality through Supplier Management

有崎直樹* Naoki Arisaki
山本幸輝* Yukiteru Yamamoto

抄　録　従来、自動車メーカは、車載電子システムのECUをサプライヤに発注するに当たり、要求仕様書を提示し、ソフトウェアを含むECUを部品として調達する方式が一般的であった。このため、車載ソフトウェアの多くは、サプライヤにより開発されていた。一方、車載されるソフトウェアの規模が増大するにつれ、ソフトウェアに関する品質問題も増大した。自動車メーカはお客様に対し、クルマの品質を保証する立場にある。したがって、ソフトウェアに関する品質も保証しなくてはならない。この結果、サプライヤの開発するソフトウェアの品質管理が重要なテーマとなった。このような背景から、日産自動車では、2001年より専門のソフトウェア技術者によるサプライヤに対するソフトウェア品質管理活動を開始した。ここでは、その品質管理活動の内容を紹介する。

Summary　When placing an order with a supplier for an ECU for an automotive electronics system, a car manufacturer normally provides the required specifications and buys the ECU, including the software. Consequently, the majority of automotive software is developed by ECU suppliers. As the volume of onboard software has increased over the years, software issues have also increased. As a car manufacturer, we are responsible for guaranteeing the quality of our cars to our customers. Therefore, we have to guarantee the quality of the onboard software as well. Quality control of software developed by suppliers has become an important issue. Against this backdrop, Nissan started a team of skilled software engineers in 2001 to conduct software quality management activities for suppliers. This article describes the details of the quality control activities that we have carried out at Nissan.

Key words : *Computer Application, automotive electronics, software quality, embedded software, SQA, computer*

1. はじめに

日産自動車における“サプライヤに対するソフトウェア品質監査”は、2001年に始まった。カーエレクトロニクスの普及に伴いクルマに搭載されるECU（Electronic Control Unit）の数は増加し続け、1990年代末には十数個であったECU数は2015年には80個（ソフト行数35MLOC規模）に至った（図1）。ECUに搭載されるソフトウェアは、基本的にサプライヤで開発されていたが、その品質レベルはサプライヤごとに大きなばらつきがあった。一方、日産では各部品を担当するエンジニアが、電子回路もソフトウェアも含めて、品質確認を行っていた。しかし、ソフトウェアは目に見えないうえ、開発経験がないエンジニアでは、ソフトウェアの出来栄えが判断できず、実際に実物を実車で評価して不具合が顕在化し、大きな手戻りが発生する事態も発生した。

このような経緯から、サプライヤのソフトウェア品質監査を担当するソフトウェア技術を持つ専門チームを発足さ

1. Introduction

Nissan began conducting quality audits of software received from suppliers in 2001. Accompanying the penetration of car electronics, the number of onboard electronic control units (ECUs) has continually increased. The dozen or so onboard ECUs in use at the end of the 1990s rose to 80 in 2015, with the source code reaching 35 million lines of code (MLOC) (Fig. 1). The software implemented in ECUs was basically developed by suppliers,

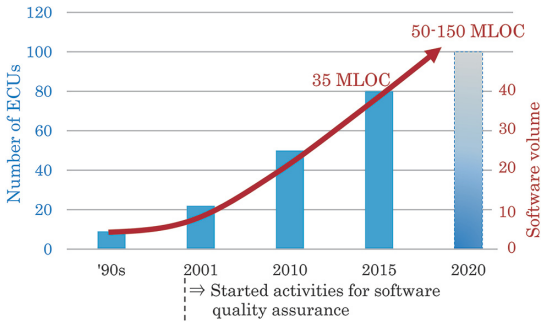


図-1　車載ECUトレンド
Fig. 1　Trend of automotive ECUs

*ソフトウェア開発部／Software Engineering Department

せることになった。

2. ソフトウェア品質監査の手法

ソフトウェアの品質監査を行うためには、監査の手法を定義する必要がある。監査手法は大きく“ソフトウェア技術”と、“ソフトウェア開発プロセス”に分けられる。前者に関しては、品質の良いソフトウェアを実現するために、どのような技術を用いて開発しているかを確認する。一方、後者では、品質の良いソフトウェアを継続的に開発するために、必要となる手順が定められているか、また定められた手順通りに開発が行われているかを確認する。

2.1 ソフトウェア開発プロセスの監査内容の定義

2001年当時、米国ではCMM（Capability Maturity Model）の開発が一旦終了し、CMMI（Capability Maturity Model Integration）（図2）に移行するタイミングであった。また、欧州ではSPICE（Software Process Improvement and Capability Determination）と呼ばれるISO規格（ISO15504）の作成が行われていたタイミングでもあった。そこで、日産では主にCMMで定義された内容を参考にし、SPICE制定の動きも考慮しつつ、車載ソフトウェアで重要な項目を選定し、監査項目として定義した。重要な項目は、CMMのレベルにとらわれずに選択した。例えば、“真の原因究明と再発防止”は典型的な項目である。CMMでは、“再発防止”はレベル5として定義されている。しかし、“真の原因究明と再発防止”は、従来サプライヤに強く要望していた内容であり、監査項目としてなくてはならない項目である。当時CMMレベル5はインドの一部の企業のみが取得した状況で、日本で取得した企業は皆無であった。しかし、必要な項目であれば当然監査内容として要求するべきであり、監査項目として選定した。

but quality levels varied greatly from one supplier to another. Engineers at Nissan responsible for individual parts worked to confirm the quality of both the electronic circuits and software. However, because software is not visible to the eye, engineers without any experience in developing software could not judge the performance of the software received. When the software was actually evaluated in test vehicles, defects were revealed that also led to situations where large rework was needed.

That was the background behind the start of a team specialized in software engineering that was made responsible for conducting quality audits of software from suppliers.

2. Methods of Software Quality Audits

In order to conduct software quality audits, it was first necessary to define the audit methods. Audit methods can be broadly divided into those concerning software technologies and those concerning software development processes. The former methods confirm what technologies were used to develop the software in order to obtain high levels of software quality. The latter methods examine whether procedures have been defined that are needed to continuously develop high-quality software and whether development activities are actually carried out according to the established procedures.

2.1 Definition of the details of software development process audits

Development of the Capability Maturity Model (CMM) was completed in the U.S. around that time in 2001 and a transition was being made to the Capability Maturity Model Integration (CMMI) (Fig. 2). In Europe, at that time the ISO 15504 international standard, which is also known as Software Process Improvement and Capability Determination (SPICE), was being created. Accordingly, Nissan selected key items concerning onboard software and defined them as audit items, mainly in reference to the details defined in the CMM and also

<table>
<tr><th colspan="2" rowspan="2">Organizational maturity levels</th><th colspan="3">Practice capability levels</th><th colspan="4">Process area</th></tr>
<tr><th>実施
Execute</th><th>管理
Manage</th><th>定義
Define</th><th>Engineering</th><th>Process management</th><th>Project management</th><th>Support</th></tr>
<tr><td>1</td><td>Initial</td><td>能力度レベル1
Ability level 1</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td></tr>
<tr><td rowspan="4">2</td><td rowspan="4">Managed</td><td rowspan="4">能力度レベル2（管理された）
Ability level 2 (Managed)</td><td rowspan="4" colspan="2"></td><td rowspan="4">–</td><td rowspan="4">–</td><td>要件管理（REQM）
Requirements management</td><td>測定と分析（MA）
Measurement and analysis</td></tr>
<tr><td>プロジェクト計画策定（PP）
Project planning</td><td>プロセスと成果物の品質保証（PPQA）
Process and product quality assurance</td></tr>
<tr><td>プロジェクトの監視と制御（PMC）
Project monitoring and control</td><td>構成管理（CM）
Configuration management</td></tr>
<tr><td>供給者合意管理（SAM）
Supplier agreement management</td><td>–</td></tr>
<tr><td rowspan="6">3</td><td rowspan="6">Defined</td><td rowspan="6" colspan="3">能力度レベル3（定義された）
Ability level 3 (Defined)</td><td>要件開発（RD）
Requirements development</td><td>組織プロセス定義（OPD）
Organizational process definition</td><td>統合プロジェクト管理（IPM）
Integrated project management</td><td>決定分析と解決（DAR）
Decision analysis and resolution</td></tr>
<tr><td>技術解（TS）
Technical solution</td><td>組織プロセス重視（OPF）
Organizational process focus</td><td>リスク管理（RSKM）
Risk management</td><td rowspan="4">–</td></tr>
<tr><td>成果物統合（PI）
Product integration</td><td>組織トレーニング（OT）
Organizational training</td><td rowspan="3">–</td></tr>
<tr><td>妥当性確認（VAL）
Validation</td><td rowspan="3">–</td></tr>
<tr><td>検証（VER）
Verification</td></tr>
<tr><td rowspan="2"></td></tr>
<tr><td>4</td><td>Quantitatively managed</td><td colspan="3">能力度レベル3（定義された）
Ability level 3 (Defined)</td><td>–</td><td>組織プロセス実績（OPP）
Organizational process Performance</td><td>定量的プロジェクト管理（QPM）
Quantitative project management</td><td>–</td></tr>
<tr><td>5</td><td>Optimizing</td><td colspan="3">能力度レベル3（定義された）
Ability level 3 (Defined)</td><td>–</td><td>組織実績管理（OPM）
Organizational performance management</td><td>–</td><td>原因分析と解決（CAR）
Causal analysis and resolution</td></tr>
</table>

エンジニアリング系プロセスはレベル3で定義される
The engineering process is defined as Level 3

再発防止はレベル5で定義される
Recurrence prevention is defined at level 5

図-2 CMMI成熟度レベル別プロセス領域
Fig. 2 Process areas by CMMI maturity level

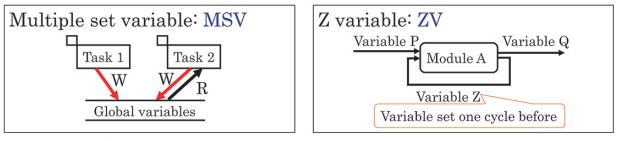
一方、要求獲得は、もう一つの重要な監査項目であった。自動車メーカの要求仕様定義の詳細さのレベルは、システムにより異なる。サプライヤの能力を尊重し、必要な項目のみ定義しているシステムもあれば、ほぼプログラムに変換できるレベルの仕様を定義しているシステムもある。仕様として定義されているものは、当然サプライヤに理解してもらい、正確に実装してもらう必要があるが、一方で、仕様として定義されていない部分についても、自動車メーカとしてはサプライヤの詳細設計内容を理解し、自社の意図と外れた内容になっていないかを確認する必要がある。当時サプライヤの中には、“文書に記述されている内容さえ守れば、それ以外は自由に定義してよい”と考えるサプライヤもあった。そこで、監査項目には、“要求仕様の内容の合意”、“要求仕様の不明確な点についての合意”に加え、“要求仕様に記述されていない内容が日産の意図と一致していることの合意”についても、確認するようにした。

2.2　ソフトウェア技術の監査内容の定義

車載ソフトウェアは、いわゆる“組込みソフトウェア”と呼ばれる種類のソフトウェアであり、リアルタイム設計が重要な項目である。当然、シングルタスクで設計することは困難であるため、マルチタスク構造が選択されるが、複数個に分割されたタスクの間での共有変数に対する書き込みの問題（図3 (a)）や、実行順序の問題（図3 (b)）などが正しく設計・実装されていないと、ソフトウェアは正しく動作しない。これらのソフトウェアのアーキテクチャ設計内容について、監査項目を定義した。

3. 監査の実施

監査は基本的に、サプライヤの開発拠点に訪問して行った。なるべく1日で監査が終了するように、監査項目を選定した。監査では、サプライヤ側から対象となるソフトウェアの概要、開発プロセスの概要を説明いただき、そのうえで手順に従って監査を行った。監査は、サプライヤの開発拠点で行われるため、開発内容を示す文書類は、その場で提示されるのが普通である。実際に、必要な作業



(a) タスク間の共有変数
各タスクの書き込みタイミングによる、変数の上書きが無いことを検証する
(a) Variables shared between tasks
To verify that there is no overwriting of variables data when writing data from all tasks.

(b) 実行順序の検証
モジュールAは1周期前の変数Zの値を参照するが、問題が無いことを検証する
(b) Verification of execution order
To verify that there are no issues when Module-A references the value of Z defined in previous schedule cycle.

図-3　アーキテクチャ設計監査項目の例
Fig. 3　Examples of assessment points in architecture design

taking into account the moves toward the establishment of SPICE. The key items were selected without sticking to the CMM maturity levels. For example, identifying the true cause of a problem and preventing recurrence is a typical item. In the CMM, recurrence prevention is defined at level 5. However, identifying the true cause and preventing recurrence is an item that we have traditionally strongly required of suppliers and is something that must be included in the audit items. At that time, only some companies in India had attained CMM level 5; there were no companies in Japan that had reached that level. However, if an item is necessary, it should naturally be required as an audit item, and so this item was selected for quality audits.

Requirements elicitation is still another key audit item. The level of detail defined in specifications required by a vehicle manufacturer differs depending on the system. There are some systems for which only the requisite items are defined out of respect for a supplier's capabilities. There are other systems for which the specifications are defined almost to the extent that they can be converted into a program. Things that are defined as specifications must be clearly understood by the supplier and be implemented correctly. On the other hand, for things that are not defined as specifications, the vehicle manufacturer must understand the contents of the supplier's detailed design and confirm that nothing deviates from the former's intention. At that time in 2001, there were some suppliers who thought that they only had to observe what was written in the specification documents and that other things could be defined as they wished. Therefore, it was decided to include the following among the audit items for confirmation: supplier agrees with the contents of the required specifications; supplier agrees with the ambiguous points in the required specifications; in addition, supplier agrees that anything not described in the required specifications is in accord with Nissan's intention.

2.2　Definition of audit details concerning software technologies

Onboard software is a type of software referred to as embedded software for which real-time design is a critical aspect. Naturally, a multi-task structure is selected because it is difficult to execute a single-task design. The software must not have any issues concerning the overwriting of common variables divided among several tasks (Fig. 3(a)) or the order of execution (Fig. 3(b)). Such things must be correctly designed and implemented, otherwise the software cannot operate properly. Audit items were thus defined regarding such details of the software architecture design.

3. Implementation of Audits

Audits were basically conducted by going to a supplier's software development center and making evaluations. Audit items were selected so that audits could be completed in one day as much as possible. An audit began with the supplier giving an overview of the software concerned and of the development process. On that basis, the audit was conducted according to the specified procedure.

が行われているか、その内容は適切であるかを確認した。

サプライヤは、製品ごとに異なる開発方式を取っているのが普通であり、製品ごとに別々に実施した。また、同じ製品でも、拠点ごとに異なる開発方式を取っている場合もある。このため、同じ製品でも開発拠点が異なれば、別々に監査を行った。

また、ナビゲーションなどの一部のソフトウェアでは、二次サプライヤに開発の一部を委託している。このため、二次サプライヤについても、監査を行った。

4. 不具合流出率の測定

自動車メーカがサプライヤに期待しているのは、自社への不具合流出数がゼロとなることである。一方、サプライヤが自力で測定できるのは、ソフトウェア開発規模（ソースコード行数など）と、サプライヤ内部の検出不具合数である。サプライヤとしては、自身の定義するプロセスで必要な作業を行い、不具合が最小となるように努力したうえで、ソフトウェアを自動車メーカに納入したはずである。しかし、それにもかかわらず、ソフトウェアの不具合がゼロであるかはわからない。自動車メーカは納入されたソフトウェアを用いて、システムテストおよび車両テストを行う。この中で残念ながら不具合が検出される場合がある。そこで、監査チームがサプライヤからのソフトウェア開発規模の情報と、実際の自社の流出不具合数より、自社へのソフトウェア不具合流出率を算出することとした。プロセス監査および技術監査の内容が良好で、かつ不具合流出率の小さいサプライヤは、やるべきことをやっており、結果として高品質を確保できていると判断することとした。

5. サプライヤ選定前監査の導入

上記の内容で、監査活動を行った。その結果、以下の点が明確になった。

良いサプライヤでは、プロセス適用も技術検討も適切に行われており、結果指標である不具合流出率も低い。一方、あまり良好と言えないサプライヤは、これらが不十分であり、不具合流出率は高い。

しかし、自動車メーカとしては、ソフトウェア品質があまり良好でないサプライヤのソフトウェアであっても、最終製品としては不具合を除去して、良好な品質の製品としてクルマを出荷する義務がある。この結果、自動車メーカ側が資源を投入し、不具合対策を行わなくてはならない。つまり、サプライヤ選定前にソフトウェア監査を実施しなければ、根本問題は除去できないことが明確となった。2001年当時、サプライヤ選定前にサプライヤ候補の技術内容を確認する作業は行われていたが、主に製品サンプルや、製品開発全体の体制、生産工場の品質プロセスな

Because an audit was conducted at a supplier's software development center, documents explaining the development details were usually presented at that time. It was confirmed whether the necessary work was actually being done and whether the details were appropriate or not.

Suppliers generally adopt a different development approach for each product, so audits were conducted separately for individual products. Even for the same product, there were times when different development approaches were taken depending on the development center. In such cases, separate audits were conducted for the same product if there were different development centers involved.

Moreover, for some software like that for navigation systems, part of the development work may be entrusted to secondary suppliers. Consequently, audits of secondary suppliers were also conducted.

4. Measurement of Defect Outflow Rate

One thing that a vehicle manufacturer expects of suppliers is a zero outflow rate of defects to the manufacturer. Meanwhile, what suppliers can measure by themselves is the scale of the software (number of source code lines, etc.) being developed and the number of defects a supplier detects in-house. It is assumed that suppliers make every effort to minimize defects by carrying out the necessary activities in processes they themselves have defined before delivering their software to a vehicle manufacturer. However, despite their efforts, no one knows for sure if the software is defect-free. The software delivered to the vehicle manufacturer is used to conduct system tests and in-vehicle tests. Unfortunately, there are times when defects are detected in such tests. Accordingly, the audit team decided to calculate the rate of software defect outflow to the vehicle manufacturer based on information received from the supplier concerning the scale of the developed software and the actual number of defects passed on to the manufacturer. Further, suppliers showing good results in process and engineering audits and also having a low defect outflow rate were judged as having done what they were supposed to do and that high quality was attained as a result.

5. Implementation of Pre-supplier Selection Audits

As a result of carrying out audit activities according to the procedures explained above, the following points became clear.

Good suppliers were both applying processes and conducting technical investigations appropriately. As an index of the results, they showed a low rate of defect outflow. In contrast, suppliers that were not so good had high rates of defect outflow because they were not doing these things sufficiently.

However, a vehicle manufacturer is obligated to ship its vehicles as products of excellent quality by removing any defects from the final product even for the software of suppliers whose software quality is not so

どについての確認であり、目に見えないソフトウェアについては、確認ができていなかった。

そこで、ソフトウェアについても、サプライヤ選定前監査を導入することとした。サプライヤ選定前監査も、上記の監査と同様にサプライヤの開発拠点に訪問して実施し、監査項目も可能な限り同じ指標で行った。監査結果が良好でないサプライヤについては、選定不可とするか、大幅な改善実施という条件付きで選定可とした。

6. 監査活動の成果

これらの監査活動を通じて、サプライヤの“ソフトウェア技術”と、“ソフトウェア開発プロセス”の能力をモニタリングし、サプライヤの改善を推進・サポートを行ってきた。その結果、サプライヤの開発能力は大幅に向上し、サプライヤの不具合流出率も2001年当時に比べて1／40まで低減することができた。

7. お わ り に

ソフトウェア開発プロセスの国際標準は2001年以降、逐次アップデートされてきている。これに対応して、我々の監査内容も改良を加えている。たとば、Automotive SPICE[1)] が自動車業界として整備されてきており、Automotive SPICEのプロセス要求を追加している。今後も世の中のベストプラクティスを導入することで、お客様に良い品質のソフトウェア、およびクルマを提供できるように、継続的に努力を行っていきたい。

8. 参 考 文 献

1）Automotive SPICEホームページ：http://www.automotivespice.com/（参照日：2018年9月26日）.

good. Consequently, the vehicle manufacturer has to invest resources in order to implement measures against software defects. In short, it became evident that this problem could not be fundamentally resolved without conducting software audits before selecting suppliers. At that time in 2001, the technical capabilities of supplier candidates were confirmed before suppliers were selected. Such confirmation work was mainly done on the basis of product samples and checks of a company's overall product development system and quality assurance processes at its manufacturing plants, among other things. However, such confirmation was not possible for software that is not visible to the eye.

Therefore, it was decided to implement pre-supplier selection audits for software as well. Such audits are conducted in the same manner as those explained above by going to the supplier's software development center where the evaluations are done and by using the same audit items and evaluation indices as much as possible. Supplier candidates showing poor results are either not selected or are selected on the condition that they make major improvements.

6. Results of Audit Activities

These audit activities have enabled us to monitor the software technologies and software development processes of suppliers as well as to encourage and support their improvement efforts. As a result, suppliers' software development capabilities have been substantially improved and their rate of defect outflow has been reduced to 1/40 of the level in 2001.

7. Conclusion

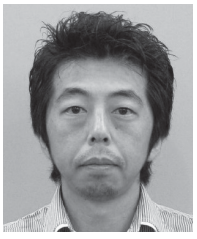
International standards for software development processes have been successively updated since 2001, and we have also made improvements to our audit details accordingly. For example, Automotive SPICE[1)] has been implemented in the automotive industry, and we have added process requirements based on it. We will continue our efforts to provide customers with excellent quality software and vehicles through the implementation of best practices around the world in the years ahead.

8. References

1) Automotive SPICE website: http://www.automotivespice.com/ (as of September 26, 2018).

■著者／ Author(s)■

有 崎 直 樹
Naoki Arisaki



山 本 幸 輝
Yukiteru Yamamoto

車載ソフトウェア開発における継続的インテグレーション（CI）の導入

Application of Continuous Integration to Automotive Software Development

伊 藤 義 信*
Yoshinobu Ito

抄　　録　近年、自動運転やコネクティド技術の台頭により、車載ソフトウェアの規模は増大の一途をたどっている。またシステムの多様化も相まって、ソフトウェアの開発活動は益々高度なものになりつつある。このような情勢は同時に、ソフトウェア管理や検証の複雑化をもたらし、人的ミスを誘発する。それを解決することが、ソフトウェアの品質向上や開発期間の短縮、ひいてはコスト削減に直結する重要な課題であると考えられる。本稿では、継続的インテグレーション（CI）を使った問題解決への取り組みと、その車載ソフトウェア開発への適用事例について紹介する。

Summary　Over the past few years, with the advent of autonomous driving and connected car technology, the scale of automotive software has been increasing steadily. This, coupled with diversification of systems, is making software development activities more and more complex. This situation complicates software management and validation, which in turn can cause human error. Solving this problem is viewed as being crucial because it is directly related to quality improvement and reduction of software development manpower and cost. This article describes ways to solve this problem using continuous integration and presents examples of its application to automotive software development.

Key words : *Computer Application, software development, software validation, continuous integration, version control system, git, cloud, agile*

1. は じ め に

私たち日産自動車が「ニッサンインテリジェントモビリティ」の取り組みを推進しているように、昨今自動車の電動化や知能化が加速している。それに伴い、自動車に搭載されるソフトウェアの規模は飛躍的に増大している。具体的には、2000年に百万行であった車載ソフトウェアのソースコードの行数は2018年現在、1億行に至っていると言われており、2025年には6億行に到達すると予測されている[1)]。また、ソースコードは度重なる機能追加やバグ修正を経てリリースされるが、その全ての成長過程をバージョンという概念を用いて管理する必要がある[2)]。このように、膨大なソフトウェアを、世代を含めて管理していくことが、昨今の自動車業界に求められている。

次に、システム観点に目を移してみる。例えば、「自動運転技術」と一言で語られることが多いシステムは、様々な機能の集合で成り立っている。縦方向の制御を司るIntelligent Emergency BrakeやIntelligent Cruise Control、横方向のLane Departure Preventionなどである。これら機能の組み合わせは、車種や仕向け地によって

1. Introduction

Vehicle electrification and incorporation of intelligence in vehicles have been accelerating in recent years, as typified by our efforts to promote Nissan Intelligent Mobility. These trends have dramatically increased the scale of automotive embedded software. Specifically, it is reported that the number of lines of onboard source code, which was one million lines in 2000, reached 100 million lines in 2018. It is forecast to reach 600 million lines by 2025.[1)] Source code is released after repeated addition of functions and debugging, and it is necessary to manage the entire growth process based on the concept of version.[2)] Accordingly, the automotive industry in recent years has had to manage this enormous volume of software, including successive generations.

Let us now shift our focus to onboard systems. For example, systems that are often talked about in simple terms as autonomous driving technologies are actually based on an aggregation of many different features. These include Intelligent Emergency Braking and Intelligent Cruise Control that help to manage a vehicle's longitudinal movement and Lane Departure Prevention that assists in controlling lateral movement. The combinations of these

*ソフトウェア開発部／Software Engineering Department

異なる場合がある。図1に示す様に、ある仕向け地には機能A、B、Cを組み合わせてシステムを構築し、別の仕向け地には機能A、D、Eを組み合わせている。そのため、各機能を齟齬（そご）なくインテグレーションし、システムとして正常に動作させることが極めて重要である。

これらは、ソフトウェア管理や検証の複雑化をもたらし、人的ミスを誘発する。それを解決することが、ソフトウェアの品質向上や開発期間の短縮、ひいてはコスト削減に直結する重要な課題であると考えられる。

2. 継続的インテグレーション（CI）とは

この「ソフトウェアの開発活動におけるソフトウェア管理や検証の複雑化」という問題提起に対して、以下の二つの課題を設定する。

- 膨大なソフトウェアを、世代を含めて管理できること
- 各機能を齟齬（そご）なくインテグレーションし、システムとして正常動作する品質を常に保つこと

これら課題におけるソリューションとして、我々はCIの導入を進めている。CIとは、広義には、刻一刻と変化する顧客の要求を素早くソフトウェアにフィードバックしつつも、常に正しく動作する品質を保つための仕組みやその思想を意味し、狭義には、ビルドやテスト、インスペクションなどを自動的かつ継続的に実行していくことを意味する。詳細なCIの定義や技術はウェブや参考書に譲るとし、以降、本稿では車載ソフトウェア開発におけるCIに焦点を絞る。一つ注釈を加えるとすれば、管理する対象にC/C++言語などで記述されたソースコードだけではなく、いわゆるモデルベース開発におけるモデルも含まれているということである。いずれにせよ本質は変わらない。

2.1 バージョン管理システム

まず「顧客の要求を素早くソフトウェアに取り組む」という点について掘り下げる。自動車に対する顧客の要求は時代と共に変化していく。それに伴ってソフトウェアも繰り返し機能を追加していくことが求められる。例え

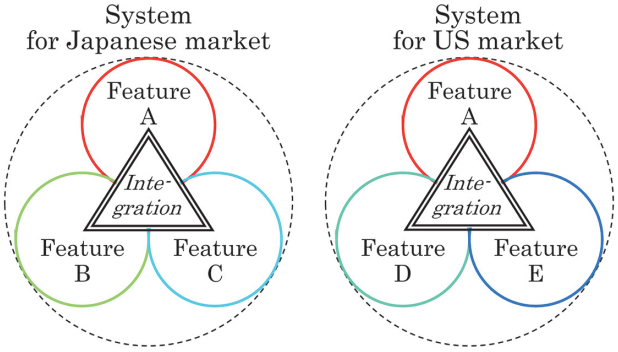


図-1　システム構成の例
Fig. 1　Examples of system structure

features sometimes differ depending on the vehicle model and the destination market. As shown in Fig. 1, features A, B and C are combined to create a system for a certain market, while features A, D and E are combined for a different market. Consequently, it is imperative to integrate the features without any variance and to ensure that the entire resultant system operates properly.

This complicates the work of managing and validating the software involved, which can give rise to human error. Resolving this critical issue can contribute directly to improving software quality and shortening development lead time, and, in turn, to reducing costs.

2. Overview of Continuous Integration (CI)

The following two issues are involved in this problem of the growing complexity of software management and validation in software development activities.

- Management of voluminous software, include successive generations
- Maintaining quality at all times when integrating functions without any variance so as to ensure the system operates properly

We are proceeding with the application of CI as a solution to these issues. In a broad sense, CI refers to the methods, practices and philosophy for maintaining quality to ensure proper system operation while promptly feeding back customers' constantly changing requirements to the software. In a narrow sense, CI refers to the automated and continuous execution of activities to build, test and inspect software. Detailed explanations of the definition and techniques of CI can be found on the Internet and in reference books. The following sections of this article will focus on CI in automotive software development. One note is added here that the object of management is not just the source code written in C, C++ or some other languages, but rather it also includes the models used in what is called model-based development. In any case, the essence is the same.

2.1 Version control system

We will first delve into the issue of promptly incorporating customers' requirements into software. Customers' needs and wants regarding vehicles change with the passing of time. Accordingly, it is necessary to repeatedly add functions to the software to reflect such changes. For example, cruise control that keeps the vehicle speed at a certain desired level without operating the accelerator pedal has gradually evolved into Intelligent Cruise Control with the addition of a function for following a preceding vehicle. When a need arises to add a new function to the source code or model in this way, the following approach is generally taken. First, the original model is reproduced and the replicated model is modified. This growth process is illustrated in Fig. 2. It is seen that when the original model is modified, the trunk (main line) representing the growth process of the model branches. The reason for the branching of the trunk is to avoid

ば、アクセルペダルを踏み続けることなく、ある一定の速度を維持するCruise Controlは、前方の車両を追従するIntelligent Cruise Controlへと次第に進化してきた。このように、新機能をソースコードやモデルに追加するという行為が発生する場合、一般的に次のような手法を取る。まず、元のモデルを複製する。そして複製したモデルに対して変更を加えていく。これは図2の様に表現できる。複製したモデルに対して変更を加えた時点で、モデルの成長過程を表現する幹が分岐（ブランチと呼ぶ）しているのが見て取れる。幹を分岐させる理由は、仮に新しい機能の実装に不備があった場合でも、元の機能へ影響を及ぼさないようにするためである。そうすることで、新機能の開発を既存機能と独立させて進めることができる。これは分散開発にもつながる話であり、ブランチの戦略はいくつか議論されている[3]が、ここでは深入りしない。分岐させたモデルに必要な機能を追加した後、最終的には元のモデルへ統合（マージと呼ぶ）することになる。この様にモデルは、ブランチやマージが繰り返されながら成長していく。その場合であっても、全ての幹の変更履歴を追従できるようにする必要がある。これはバージョン管理システムで実現される。このシステムのおかげで、新機能をソフトウェアにいくら取り込んでも、幹の構成やバージョンなどの管理が保たれたまま開発を進めることが可能となる。後述するが、CIを実現するシステムはバージョン管理システムと連携して動作するように構築される。よってCIを導入することが、一つ目の課題解決につながるのである。

ここに挙げた例はあくまで概念を説明するためのものであり、実開発で本過程が採られたかどうかは定かではない。また、モデルを複製して幹を分岐させると書いたが、内部処理として実際にモデルが複製されるか、差分管理されるかはバージョン管理システムによる、という但し書きを付けて加えておく。

2.2　自動的かつ継続的作業

次に「自動的」および「継続的」について掘り下げる。車載ソフトウェアの開発は、図3のような工程を採る。こ

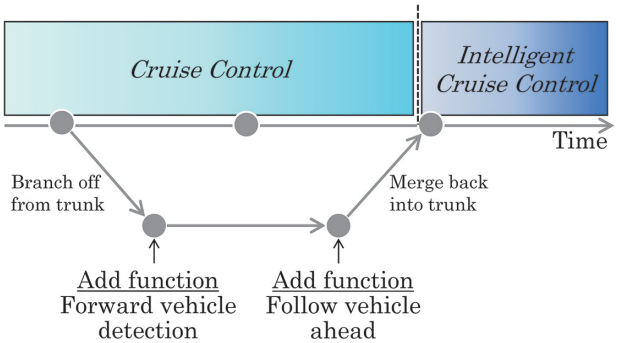


図-2　モデルの成長過程
Fig. 2　Growth process of a model

influencing the original function in the event that there is a problem in the implementation of the new function. Branching makes it possible to develop the new function independently of the existing function. This leads to the topic of distributed development. Several strategies have been discussed for branching,[3] but we will not go into the details here. After adding the necessary function to the branched model, it is ultimately merged with the original model. In this way, a model grows through a process of repeated branching and merging. In every case, it is necessary to be able to trace the entire modification history of the trunk. This is accomplished by the version control system. This system enables development to proceed while maintaining management of the trunk configuration, versions and other details, regardless of how many new functions are incorporated in the software. As will be explained later, the system that facilitates CI is constructed such that it operates in tandem with the version control system. Therefore, the application of CI leads to a solution to the first issue mentioned earlier.

The example cited here is merely for the purpose of explaining the concept. It is not definite whether this process is adopted in actual development work. It was mentioned above that the model is reproduced and branched, but a proviso is added here that the internal processing depends on the version control system as to whether the model is actually reproduced or differences are managed.

2.2　Automated and continuous operations

Next, let us consider more closely the terms automated and continuous. The development of automotive software follows the process outlined in Fig. 3. Among these operations, the following tests are conducted, for example, in the validation phase.

(1) Unit tests, including coverage analysis
(2) Static analysis for checking modeling guidelines
(3) Auto code generation
(4) Static analysis for checking coding standards
(5) Integration tests, including coverage analysis

In cases where multiple tests are combined and conducted as indicated here, automating this series of operations is a rational approach. The reason is that it can be expected to produce various benefits such as preventing

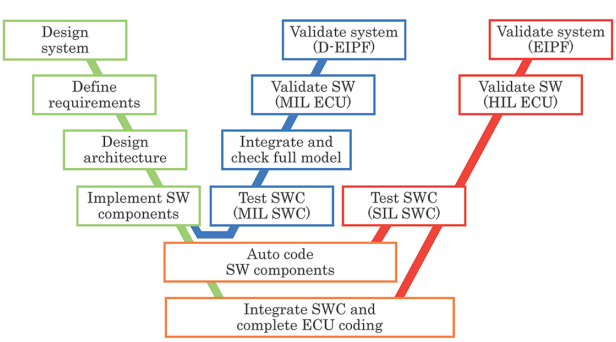


図-3　車載ソフトウェアの開発工程
Fig. 3　Development process of automotive software

の内、検証フェーズでは、例えば以下の試験を実施している。

(1) 単体試験（カバレッジ分析含む）
(2) 静的解析（モデリングガイドラインチェック）
(3) オートコードジェネレーション
(4) 静的解析（コーディング規約チェック）
(5) 結合試験（カバレッジ分析含む）

このように、複合的な試験の組み合わせを実施する場合、それら一連の作業を自動化するのが合理的である。何故なら、作業漏れの抑止や生産性の向上といった効果が期待できるからである。また自動であるが故の副次的な効果ももたらす。それは、本工程を回す頻度を上げたとしても、それが作業負荷に必ずしも直結するわけではないということである。そのため、毎日もしくはモデルを修正する度に、一連の作業を実施することも可能になる。つまり、本工程を自動化することで、それを負荷なく継続的に実施できる。これが二つ目の課題の解決につながることになる。

一般的に、CIを実現するシステムはバージョン管理システムと連携して動作するように構築される。簡単な流れは次の通りである。まず、更新されたモデルがバージョン管理システムにアップロードされると、その情報がCIサーバに通知される。CIサーバはそれをトリガとして、あらかじめ準備していた一連の検証作業の自動化スクリプトを、更新されたモデルに対して実施する。そして、検証に失敗した場合は、その箇所やエラー内容などがエンジニアへ直ちに通知され、修正を促す（図4）。この仕組みによって、修正したモデルは都度インテグレーションされ、検証に掛けられることになり、常に正常動作する品質が保たれるのである。

ここまでを整理すると、CIとは、バージョン管理システムを併用し、検証を含む開発工程の一部作業を自動化し、それを継続的に実施できるようにしたシステムである、と捉えることができる。これがCIの全貌である。

3. 期待する効果

では、CIを導入することで、どういった効果が見込めるのであろうか。一つ目は、作業を漏れなく正確に実施できる点が挙げられる。作業が複数にまたがればまたがるほど、作業漏れが発生しやすく、またそれ自体が属人化しやすい傾向にある。自動化する作業をスクリプトなどに記述することで操作が明文化され、実施者によらない正確な作業が可能となる。

二つ目は、バグを開発工程の早い段階で検出できるという点である。単体試験と異なり、結合試験は各機能の実装が完了した後に実施されることが多い。しかし、一般的に工程が後になるにつれてモデルは肥大化していくため、

tasks from being overlooked and enhancing productivity. Automating operations also brings secondary benefits. For example, even if the frequency of carrying out the process is increased, it is not necessarily directly related to the workload. Therefore, it is possible to perform the series of operations every day or every time a model is modified. In other words, automating this process means it can be done continuously without increasing the workload. This leads to a solution to the second issue mentioned earlier.

A system that implements CI is generally configured so that it operates in tandem with the version control system. The flow can be explained simply as follows. First, when an updated model is uploaded to the version control system, the CI server is notified of that information. That triggers the CI server to execute on the updated model an automated script for a series of validation exercises prepared in advance. In case the validation ends in failure, the engineers involved are notified directly of the failure locations and error details, prompting them to fix the problems (Fig. 4). In this system, every time a model is modified, it is integrated and subjected to validation, thus constantly maintaining quality for ensuring proper operation of the model.

In summarizing the foregoing explanation, CI can be understood to be a system that enables some development processes, including validation, to be automated and performed continuously in tandem with the version control system. The foregoing has been an overall picture of CI.

3. Expected Benefits

What benefits can be expected as a result of applying CI? The first one is that work can be carried out correctly without missing anything. The more an operation extends over multiple tasks, the easier it is for something to be left out and that tendency is apt to depend on the individuals involved. Indicating the tasks to be automated in a script makes them clear so that the work can be done properly regardless of who does it.

A second benefit is that bugs can be detected at an early stage in the development process. Unlike unit tests, integrated tests are usually conducted after each functional feature has been implemented. However, as processes move downstream, the model is greatly enlarged, making

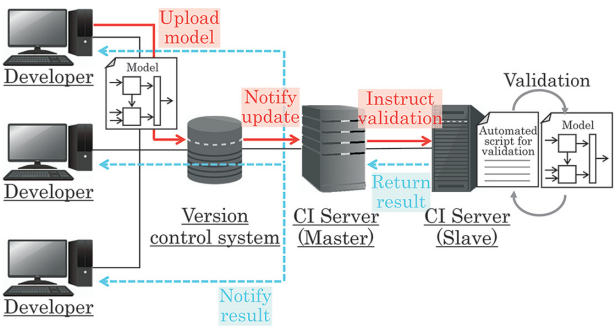


図-4 CIサーバの構成
Fig. 4 CI server structure

その分だけバグを発見する難易度が増す。もしエンジニアがモデルを修正する度に結合試験を実施できるようになれば、バグを早期に発見できる可能性が高まり、手戻りの削減や生産性の向上、ならびに品質の向上が期待できる（図5）。

三つ目は、インテグレーションを短いサイクルで実施するところにヒントがある。それは、機能が完全に実装されていない段階であっても作動させて確認することで、顧客の要求と実際の挙動とのかい離を早期に確認できるという点である。ギャップがあった場合でも、それが拡大する前に軌道修正を図ることができ、また急な要求変更にも柔軟に対応しやすいと言える。これらはアジャイル[4]と呼ばれる開発手法で一部語られているが、今回深くは触れない。余談ではあるが、CIはアジャイルの一つのプラクティスとして確立されたものである。

4. 車載ソフトウェア開発への適用

さて、ここからは、我々がCIをどのように車載ソフトウェア開発へ適用させようとしているか、とりわけ前述の二つの課題を念頭に置きながら説明したい。

4.1 開発機能のソフトウェア管理

まずは「ソフトウェアの管理」についてである。これからある機能を開発する場合、当該機能は特定の車種だけではなく、複数の車種にも共通して適用できるようにすることが望ましい。例えば、自動運転技術は今後様々な車種に搭載されるが、それを車種ごとに別々に開発するのは非生産的かつ非効率である。そこで我々は、一つの機能を一つのリポジトリで管理し、各車種が有するリポジトリがそのリポジトリを参照するような構成でソフトウェアを管理している（図6）。リポジトリとは、ファイルやディレクトリを記録する場所であり、それらの変更、ブランチやマージの履歴も記録している一つの集合体である。チューニングに関するパラメータなど車種固有の値を除き、機能ロジックの大半の部分はこのように共通管理ができる。そう

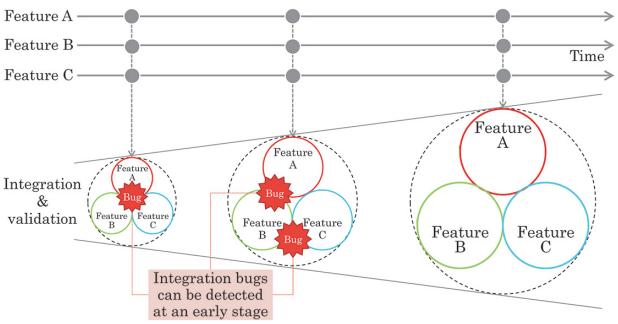


図-5　CIの期待する効果
Fig. 5　Expected benefits of CI

it that much more difficult to detect bugs. If an integrated test could be conducted every time an engineer modifies the model, it would increase the possibility of detecting bugs earlier. That could be expected to reduce return work, improve productivity and also enhance quality (Fig. 5).

A third benefit is that there would be hints for carrying out integration in a shorter cycle. In other words, it would be possible to confirm at an early stage whether there is any discrepancy between customers' requirements and the actual system behavior. That could be done by running the system even at a stage where all the functional features have not been implemented yet. If a gap is detected, it could be fixed and the system put on the right path before the discrepancy gets any larger. It would also allow flexibility for coping with sudden changes in customers' requirements. This is sometimes talked about as agile software development,[4] but we will not discuss it in detail here. As a small digression, CI was established as one practice of agile development.

4. Application to Automotive Software Development

This section will explain how we are endeavoring to apply CI to the development of automotive software, especially keeping in mind the two issues mentioned earlier.

4.1 Software management as a development function

First, let us consider software management. In developing a functional feature, it is desirable to make it applicable to several models in common, rather than limiting its use to just certain specific vehicle models. For example, autonomous driving technologies will be implemented on various vehicle models in the coming

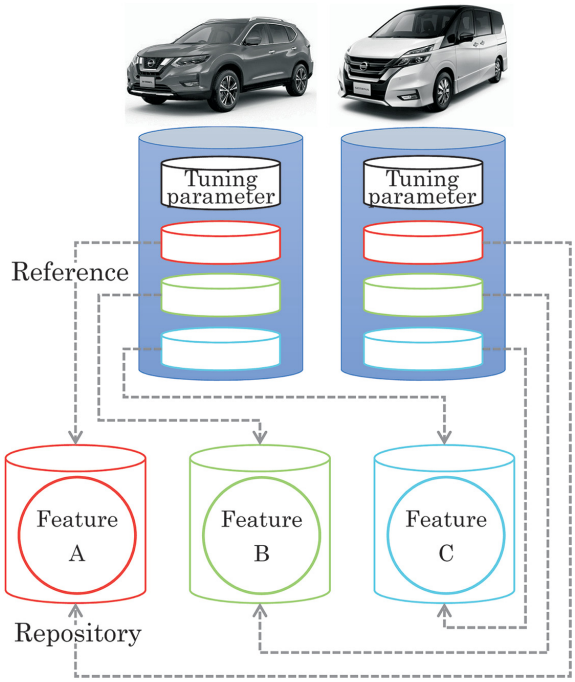


図-6　リポジトリ構成
Fig. 6　Repository structure

することで、新たに機能の追加や修正が生じた場合でも、それを管理しているリポジトリだけを更新すればよく、複数の場所に更新を施すといった重複管理であるが故の漏れを生みそうな作業を避けられる。

一方で留意しなければならない点もある。それは更新した機能が、それ単体で正しく動作するからといって、その機能を参照している全ての車種に必ずしも適合するとは限らないということである。そのため、機能単体の検証に加えて、それを参照している各車種へ組込んでの検証が不可欠であり、そういった世話をCIシステムでサポートしてあげる必要がある。これはスーパーマージと独自に名付けた仕組みで管理している。機能を管理するリポジトリにおいて、分岐させて開発を進めた内容を元のモデルへ統合するマージとは別に、そのリポジトリを参照するリポジトリ、つまり車種が有するリポジトリへマージすることがスーパーマージである（図7）。そして、マージとスーパーマージを実行する前段にクオリティゲートと呼ぶ関所を設け、実際にマージしてよい品質になっているかどうかを検証することで、インテグレーション後の品質を担保している。

4.2　開発機能の自動車への適用

では、「どのような検証フローを経て、開発した機能を各車種へ適用しているのか」について、図8を用いて説明する。まず、バージョン管理システムにアップロードされたモデルを第一のクオリティゲートであるMIL（Model In the Loop）検証に掛ける。ここでは、モデルに対して単体試験やモデリングガイドラインチェックなどの静的解析を実施し、最終的にC言語のソースコードへと変換する。ここまで不具合なく到達した場合に、そのモデルを元のモデルへマージする。その後、スーパーマージつまり車種固有のリポジトリへマージするための検証としてSIL（Software In the Loop）検証を掛ける。ここでは、変換したソースコードに対してコーディング規約チェックなどの静的解析や、元のモデルとの出力の差分を確認するBack to back試験を実施する。そして、AUTOSAR

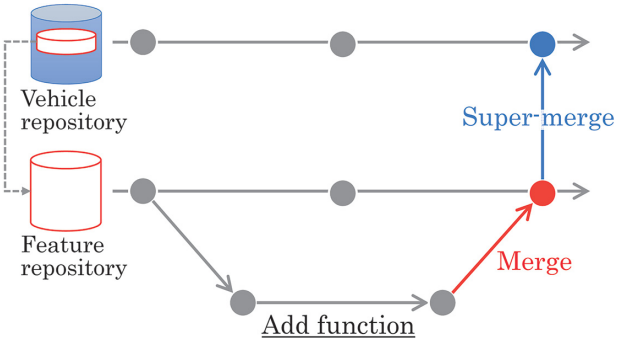


図-7　スーパーマージの概念
Fig. 7　Concept of super-merging

years. Developing them separately for each individual model would be unproductive and inefficient. Therefore, we manage software within a framework where one feature is managed in one repository that can be referenced by the repository of each vehicle model (Fig. 6). A repository is where files and directories are recorded. It is a collection that records histories of the modifications, branching and merging they have undergone. The greater part of functional logic can be managed in common in this way, excluding values specific to certain vehicle models such as tuning parameters. In this way, when a new feature is added or a modification is made, only the repository where it is managed needs to be updated. This avoids duplicative management in which multiple places must be updated and thus avoids work where tasks tend to be missed.

On the other hand, there are also points requiring attention. For example, just because a feature operates properly at the unit level does not necessarily mean it will fit all vehicle models that reference the feature. Consequently, in addition to validating the feature in unit testing, it is essential to validate it when it is embedded in each vehicle model that references the feature. It is necessary to support that activity by means of the CI system. This is managed by a practice we ourselves call “super-merging”. This is different from merging the changes developed along a branch back into the base feature repository alone. Super-merging refers to merging the changes into the repositories of all the vehicle models that reference the feature repository (Fig. 7). Quality gates are set up as checkpoints at the stage before merging and super-merging are executed. The quality gates validate whether the quality of the feature is actually suitable for merging, which guarantees quality after integration.

4.2　Application of developed features to production vehicles

Figure 8 is used here to explain the flow of the validation process that newly developed features go through before being applied to production vehicles. A model uploaded to the version control system first undergoes model-in-the-loop (MIL) validation, which is the first quality gate. Here, the model is subjected to unit tests and static analysis such as a modeling guideline check. Finally, it is converted to source code in the C language. If

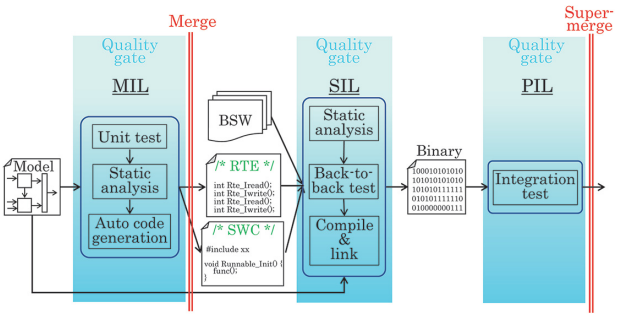


図-8　マージとスーパーマージのフロー
Fig. 8　Flow of merging and super-merging

（Automotive Open System Architecture）の構成要素であるRTE（Run Time Environment）やBSW（Basic Software）をコンパイル、リンクしバイナリファイルを作成する。このバイナリファイルをPIL（Processor In the Loop）検証に掛け、結合試験を実施する。これらのクオリティゲートを無事に通過して初めてスーパーマージを実施する。このCIで実施される一連の作業の流れをCIパイプラインと呼ぶが、このCIパイプラインを上手く設計することで、日々エンジニアがアップロードするモデルが各々の車種で正常動作することを担保している。誤解のないように断っておくが、CIは定型作業を自動実行する仕組みの一つに過ぎず、テスト項目やパイプラインの設計、実装は人間が実施するという点を忘れてはならない。

5. おわりに

ここまで、ソフトウェアの開発活動におけるソフトウェア管理や検証の複雑化と、その解決に向けたCIの取り組みについて紹介してきた。最後に将来への展望を語って締めくくりたい。

自動車業界においては新鮮であろうこのCIの技術は、ソフトウェア業界ではCI/CDへと発展してきた。CDは継続的デリバリのことであり、CIを拡張した手法で、ビルドやテストだけではなく、ソフトウェアのリリースまで含めて自動化することを意味する。自動車業界に当てはめると、新たな機能の追加や修正がインターネットを通じて市場に出回っている自動車へ自動的かつ継続的に配信、適用されることと言い換えられる。もちろんその実現には様々な障壁が立ちはだかるのは言うまでもない。しかし、コネクティドカーの発展により生まれる新たなビジネスへの柔軟な対応や、AI（人工知能）を用いた自動運転技術の搭載などに目を向けると、そのような機能が自動車に求められるのは必然である。またそれは、Adaptive AUTOSAR、クラウドや仮想化などの技術革新なくして実現は困難であろう。この様に、ソフトウェアの塊と化す自動車をこれら要素技術と組み合わせ、どう発展させていくか、時代の潮流がエンジニアに突き付けた命題である。

最後に、ここで挙げた例や説明には、簡略化のため事実と異なる内容、未確定の内容を一部含んでいる点にご留意願いたい。

6. 参考文献

1）経済産業省：自動車新時代戦略会議（第1回）配布資料、http://www.meti.go.jp/committee/kenkyukai/seizou/jidousha_shinjidai/001_haifu.html（参照日：2018年10月2日）.

2）ISO26262-8：Road vehicles –Functional safety – Part 8:

the model reaches this point without any problems being detected, it is then merged with the original model. It then undergoes software-in-the-loop (SIL) validation before being merged with the model-specific repository, i.e., super-merging. Here, the converted source code is subjected to static analysis such as a coding standards check and a back-to-back test to confirm any differences with the output of the original model. After that, a binary file is created by compiling and linking the run time environment (RTE) and basic software (BSW), which are constituent elements of the Automotive Open System Architecture (AUTOSAR). The binary file is then subjected to processor-in-the-loop (PIL) validation and an integration test. Super-merging is done only after successfully passing through these quality gates. The flow of the series of tasks executed in this CI process is referred to as the CI pipeline. A well-designed CI pipeline guarantees that the models engineers upload on a daily basis will operate properly in each vehicle model. To make sure there is no misunderstanding, it will be noted that CI is nothing more than one system for automatically executing routine tasks. It must not be forgotten that human engineers are the ones who design and implement the test items and CI pipeline.

5. Conclusion

The preceding sections have described how software management and validation have become more complex in software development activities and how CI is being applied to resolve such complexities. Finally, we will conclude with a discussion of the outlook for the future.

CI technology, which is a fresh approach in the automotive industry, has already evolved to CI/CD in the software industry, where CD stands for continuous delivery. It is a method of extending CI and refers to automation of the release of software in addition to build and test activities. In the context of the automotive industry, it can be rephrased as automatically and continuously streaming and applying to vehicles already on the market new or modified functions via the Internet. Naturally, it goes without saying that various hurdles are standing in the way of it becoming a reality. However, it is inevitable that such a capability will be required of vehicles when we consider the flexible accommodation of new businesses created by the deployment of connected vehicles or the implementation of autonomous driving technologies based on artificial intelligence (AI). Moreover, it will be difficult to achieve that capability without the use of technological innovation represented by Adaptive AUTOSAR, cloud computing and virtualization of hardware. As indicated here, current trends have thrust upon engineers the proposition of how to integrate, for future development, such key technologies with vehicles that are being transformed into a mass of software.

Finally, readers are kindly asked to note that, for the sake of simplification, the examples and explanations given in this article may include details different from actual facts or things that are still not definite.

Supporting processes、https://www.iso.org/standard/51364.html（参照日：2018年10月2日）.
3）A successful Git branching model Vincent Driessen、https://nvie.com/posts/a-successful-git-branching-model/（参照日：2018年10月2日）.
4）Manifesto for Agile Software Development、http://agilemanifesto.org/iso/en/manifesto.html（参照日：2018年10月2日）.

6. References

1) METI: Strategy Meeting for the New Era of Automobiles Handout at The first meeting, http://www.meti.go.jp/committee/kenkyukai/seizou/jidousha_shinjidai/001_haifu.html (as of October 2, 2018).
2) ISO26262-8: Road vehicles –Functional safety – Part 8: Supporting processes, https://www.iso.org/standard/51364.html (as of October 2, 2018).
3) A successful Git branching model Vincent Driessen, https://nvie.com/posts/a-successful-git-branching-model/ (as of October 2, 2018).
4) Manifesto for Agile Software Development, http://agilemanifesto.org/iso/en/manifesto.html (as of October 2, 2018).

■著者／ Author(s)■

伊　藤　義　信
Yoshinobu Ito

ソフトウェアトレーニングセンタ

Software Training Center

山　本　幸　輝*　Yukiteru Yamamoto
長谷川　美和子*　Miwako Hasegawa
小　野　裕　二*　Yuji Ono

抄　　録　現在、自動車業界は、百年に一度の大変革期にあると言われる。そして、この変革のキーとなる技術は、ソフトウェア技術である。したがって、自動車メーカにとり、ソフトウェア技術力の確保は急務となっている。これに対し2017年12月、日産自動車はエンジニアのソフトウェアに関するスキルアップを目的に、ソフトウェアトレーニングセンタを開設した。ここでは、同センタの教育内容、教育環境について述べ、また受講生の反響もふくめて紹介する。

Summary　It is said that the automotive industry is currently in a major transformation period that occurs once every 100 years. The key technology behind this transformation is software. Therefore, acquiring software development capabilities is an urgent task for every car manufacturer. Considering this situation, Nissan opened the Software Training Center to train engineers who have no software engineering skills. This article describe the training courses and training environment at the Software Training Center and also feedback received from the trainees.

Key words : *Computer Application, automotive electronics, software, training, model-base development, computer*

1. は　じ　め　に

現在、自動車業界は百年に一度の大変革の時期にあると言われる。これは電動化、自動運転、コネクティドカーに代表される新技術が導入されるためであり、これに伴い、自動車のビジネス形態も変革が起こり、カーシェアに代表されるビジネスが普及していくと言われている。

そして、これらの変革を支える基盤技術の一つがソフトウェア技術である。ソフトウェア技術無しでは、電動化、自動運転、コネクティドカーを実現することはできず、ソフトウェア技術の重要性が日々増大している。

このため日産自動車では、ソフトウェアトレーニングセンタを開設し、車載ソフトウェアを開発する能力を持つエンジニアの育成を図っている。本稿では、ソフトウェアトレーニングセンタの概要を説明する。

2. 教育対象となるエンジニア

日産のソフトウェアトレーニングセンタの教育対象は、基本的にソフトウェアに関する知識の少ないエンジニアとすることとした。現在、自動車に必要とされる技術は、急速にメカからソフトにシフトしている。これに対応するた

1. Introduction

The automotive industry is said to be undergoing a profound transformation now that occurs once in 100 years. This is happening because of the introduction of various new technologies as typified by electrification, autonomous driving and vehicle connectivity, among others. As a result, the nature of the automotive business is being revolutionized and new businesses such as car sharing are expected to become more prevalent in the coming years.

One of the fundamental technologies driving this transformation is software. Without software, electrification, autonomous driving and vehicle connectivity would not be possible. The importance of software is continually increasing every day.

For that reason, Nissan established the Software Training Center to nurture engineers capable of developing onboard software. This article presents an overview of the Software Training Center.

2. Engineers Targeted for Training

The training courses at Nissan's Software Training Center are basically targeted at engineers who have little knowledge of software. At present, the technologies required by vehicles are shifting rapidly from mechanical disciplines to software. In order to cope with this change, the training courses can be taken without any prior knowledge of software.

* ソフトウェア開発部／Software Engineering Department

め、ソフトウェアの予備知識の少ないメカニカルエンジニアであっても、ソフトウェア技術力をつけることができるようにするため、予備知識無しで受講できる教育内容とした。

3. 教育内容と目標レベル

走行系ECU（Electronic Control Unit）向けのソフトウェアを、MATLABを用いたモデルベース開発手法を用いて開発することを、教育内容として定義した。また、受講生がそれぞれ自動運転の一部の機能について、実際に仕様検討からモデル開発、単体テスト、システムテストを行い、動くソフトウェアを完成できるようになることを目標レベルとした。

4. 具体的な教育内容

ソフトウェアを扱えるエンジニアを育成するに当たり、全体を三つのステップに分けた。

• ステップ1：ソフトウェア基礎　～ソフトウェアの動作原理の理解～

ソフトウェアの前提知識の無い者にとって、ソフトウェアとは何かを理解することが最初のステップとなる。マイコンやCPU（Central Processing Unit）の構成を学び、ソフトウェアがCPU上で動作する仕組みを理解することで、ソフトウェア仕様の開発や不具合解析の基礎力を育成する。加えて自動車向けソフトに不可欠なCAN（Controller Area Network）、Sleep/Wake-up、WatchDogTimerなどの機能、ISO26262、Automotive SPICEなどの自動車業界の規格・標準について学ぶ。具体的な教育内容を表1に示す。

• ステップ2：モデルベース開発（MBD）～MBDツールの習熟とプロトタイプモデルの作成～

次のステップは、動くソフトウェアを作るスキルの習得である。MBD[1)]では、システムで実現したい動き（ソフ

表-1　ステップ１：ソフトウエアの基礎
Table 1　Step 1: Software basics

Software basics	Automotive embedded software basics
・Software overview	・In-vehicle communication
・Software development process	・Diagnostic overview
・Operating system	・ECU sleep/wake-up
Microcontroller basics	・Fail-safe
・Microcontroller basics	・Watchdog timer
・Flowchart	・Memory management
・CPU	・AUTOSAR overview
・Operator and addressing	Automotive software development process
・I/O port	・Software development model: V-cycle
・A/D conversion	・Software design
・PWM timer	・Software testing
・Exception handling	Automotive software standards
C programing language	・Software standards overview
・Basic C grammar	・Automotive SPICE overview
・Array and pointer	・ISO 26262 overview
・Structure and union	・MISRA overview
・Compiler	・MAAB overview

This policy enables even mechanical engineers who have limited knowledge of software to acquire skills in software engineering.

3. Training Content and Target Level

The training content is defined as to enable trainees to develop a software program for an electronic control unit (ECU) for controlling a vehicle system using the model-based development approach and MATLAB. Each trainee is given the responsibility for a certain function of an autonomous driving system. The target level is to be able to complete a working software program, beginning from an investigation of the actual specifications through model development, unit testing and system testing.

4. Specific Training Content

The nurturing of engineers capable of handling software is divided into three overall steps.

• Step 1: Software basics—Acquiring an understanding of the principles of how software programs operate

The first step for engineers who have no prerequisite knowledge of software is to understand what software is. Here, they learn the structure of a microcontroller and a central processing unit (CPU) and gain an understanding of how software runs on a CPU. In this way, they acquire the fundamental capabilities for developing software specifications and analyzing problems in software programs. In addition, they also study the functions indispensable to automotive software such as the Controller Area Network (CAN), sleep/wake-up, watchdog timer and others as well as the specifications and standards of the automotive industry, including ISO 26262, Automotive SPICE and others. The specific details of the training courses are shown in Table 1.

• Step 2: Model-based development (MBD)—Mastery of MBD tools and creation of a prototype model

The next step is to acquire the skills for creating a working software program. In the MBD[1)] approach, the desired system functionality (software specifications) is described in a model and the operation of the model is verified (specifications validation). Here, trainees learn the related technologies, including control theory such as proportional-integral-differential (PID) control and feedback control, how to operate MATLAB, Simulink, Stateflow and other modeling tools, and rapid prototyping techniques for evaluating the created model on an actual vehicle. The specific details of the training courses are shown in Table 2.

• Step 3: Software development for mass production—Acquiring an understanding of the mass production process and actual practice

In developing software for implementation on a production vehicle, it is not enough just to create a model and conduct simulations and evaluations. It is also necessary to have the techniques for maintaining quality in terms of source code readability, reusability and reproducibility. Here, trainees acquire the skills needed for developing

トウェア仕様）をモデルで記述し、動作確認（仕様検証）する。これに関連する技術として、PID（Proportional Integral Differential）制御、フィードバック制御などの制御理論と、MATLAB、Simulink、Stateflowなどのモデリングツールの操作、および作ったモデルを実機上で評価するためのラピッドプロトタイピングの手法について学ぶ。具体的な教育内容を表2に示す。

• ステップ3：量産ソフトウェア開発 ～量産プロセスの理解と実施～

実プロジェクトに搭載されるソフトウェアを開発するには、モデルを作成しシミュレーションや評価するだけでなく、可読性、再利用性、再現性といった品質を維持する技術が不可欠である。ここでは、ルノーと日産で定義したアライアンスモデリングガイドラインやアライアンスプロセスを理解し、量産ソフトウェア開発に必要なスキルを習得する。モデル開発、モデルテスト、自動コード生成、コード検証、システムテストのそれぞれの工程を基準に従って実施するスキルを身に着ける。具体的な教育内容を表3に示す。

5. 教　育　環　境

ソフトウェアトレーニングセンタは、神奈川県厚木市に

表-2　ステップ２：モデルベース開発
Table 2　Step 2: Model-based development

Model-based development overview ・MBD overview ・MBD process and toolsets	Stateflow ・Basic operation ・Flow chart ・State chart ・Other functions
Matlab ・What is MATLAB? ・Data input and editing ・Data operations ・Data types and arrays ・Graph drawing ・Functional programming	MicroAutoBox ・What is MicroAutoBox? ・Hardware specifications ・Harness creation ・RTI block set ・Flash memory write/clear ・RS232C serial communication ・CAN communication ・Bypass rapid prototyping
Simulink ・Basic operation ・Modeling and simulation ・Modeling equations ・Linear system modeling ・Discrete system modeling	ControlDesk ・ControlDesk overview ・Basic operation ・RTI block set ・Rapid prototyping

表-3　ステップ３：量産ソフトウエア開発
Table 3　Step 3: Software development for mass production

Alliance software development process ・Automotive SPICE and O52 alliance process ・O52 generic information ・O52 design activity introduction ・O52 validation activity introduction	Auto code generation ・Auto code generation overview ・Code generation by embedded coder
Alliance modeling rules and guidelines ・Scope ・References ・Terms & definitions ・Symbols & abbreviations ・Alliance rules	Static code analysis ・Static code analysis overview ・Static code analysis tools ・Standards checks using static analysis ・Code check by TCS ECA ・Code check by Polyspace
Model development for mass production ・Modeling for code generation ・Model verification process ・Test case generation by Reactis ・Model verification by Simulink V&V ・Simulink diagnosis report ・Model check report	HILS (Hardware-in-the-loop simulation) ・HILS introduction ・HILS setup and walk-through ・Testing procedure in HILS

software for mass production and an understanding of the Alliance modeling guidelines and Alliance processes defined by Renault and Nissan. They obtain the skills for carrying out model development and testing, automatic code generation and validation, and system testing according to the criteria of each process. The specific details of the training courses are shown in Table 3.

5. Training Environment

The Software Training Center is located in the Seminar House adjacent to the Nissan Advanced Technology Center in the city of Atsugi in Kanagawa Prefecture. Training courses are conducted in a large lecture hall and in two laboratory rooms for carrying out software evaluations using test equipment.

In the large lecture hall, each trainee is issued a personal computer (PC) and licenses to use software development tools. Trainees learn the intended skills by actually operating the PCs themselves while watching the operations performed by an instructor (Fig. 1).

The laboratories have a test environment where prototype models can be evaluated using radio-controlled model cars (Fig. 2). A test environment is available that is identical to the environment used in an actual development project to evaluate an ECU, for example, for an advanced driver assistance system (ADAS) (Fig. 3).

In Step 2, trainees create prototype models for low-speed driving, following a preceding vehicle and emergency braking and evaluate them by running the software on the radio-controlled model cars. In Step 3, they create models for hands-off steering wheel detection processing and lateral vehicle control processing, generate the source code, and integrate it with an actual development project model to run a hardware-in-the-loop simulation (HILS) using an actual ECU.

6. Feedback from Trainees

Through the three steps explained above, trainees learn basic principles, acquire skills in using development tools, create a prototype model, and develop and evaluate a software program for an ECU. One distinct feature is that through hands-on training the trainees experience all the processes in software development during the course of completing their own software program by trial and error.

The following favorable evaluations were received from trainees who completed the entire training program.

• I learned the importance of the specification document quality and of feasible specifications development.

• I learned the points requiring attention when pinpointing the root causes of problems and incorporating fixes in the model.

• I can now imagine how software operates, so I will be able to discuss things with suppliers on a deeper level.

• I am now able to estimate the manpower needed to develop or revise software. Overall, 98% of the engineers who undertook this training responded positively that they now have the confidence to develop software or that their anxieties about it have been dispelled.

ある日産先進技術発センタに隣接するセミナーハウス内に設立され、教室用の大ホール、および実機評価ラボ二部屋を設けている。

大ホールでは、一人一台のパソコン（PC）と各開発ツールライセンスが与えられ、講師の操作を見ながら実際に自らPCで手を動かしながら学習する（図1）。

実機評価ラボには、ラジコンカーでのプロトタイプモデル評価環境（図2）、例えば先進運転支援システム（ADAS）のECU評価環境として実プロジェクトで使用している環境と同じものを用意している（図3）。

ステップ2では、低速走行、前車追従、緊急ブレーキのプロトタイプモデルを作成し、ラジコンカー上での走行評価を行う。ステップ3では、ハンズオフ検知処理、横方向制御処理のモデル作成、コード生成を行い、実プロジェクトのモデルと統合し、実ECUでのHILS（Hardware In the Loop Simulation）テストを行う。

6. 受講生の反響

三つのステップを通して、原理を学び、ツール操作を習得し、プロトタイプモデルの作成、ECU向けソフトウェアの開発・評価を行う。トライ＆エラーで自作のソフトウェアを完成させていくことで、ソフトウェア開発の全工程を実体験として学ぶことを一つの特徴としている。

全過程を修了した受講生からは「仕様書の質、作り込みの重要さを学べた」「不具合原因の勘所や作り込み時に気を付けるべき観点を学べた」「ソフトウェアの動きがイメージできるようになったので、サプライヤとより深いレベルで議論できる能力がついた」「ソフト開発・変更にかかる労力が見積もれるようになった」など好評価を得ている。全体では98%が本トレーニングによりソフトウェア開発の自信がついた／不安がなくなったという肯定回答であった。

7. ま と め

これまでに第1期、2期が修了した。基礎知識や担当業務に差はあるものの、本センタでの教育を通じて一定レベルの基礎能力を身に着けることができている。個々のシステム特有の制御やアルゴリズムについては、本センタでは扱っていない。しかしながら、コアとなる技術はシステム間で大きく変わることは無く、そこを理解しておくことで、新しい技術への対応ができると考える。本センタで培ったコア能力を生かし、今後さらに進む自動車の変革、それに伴うソフトウェア技術の進化に対応していけることを期待し、今後、計画的にソフトウェアエンジニアの育成を推進していく。

7. Conclusion

The first and second training sessions have been completed. While there were differences in basic knowledge and work responsibilities, the trainees acquired a certain level of fundamental skills through the training courses they received at the Center. The Software Training Center does not treat the control procedures or algorithms specific to individual systems. However, the core technologies do not differ greatly between systems. If the trainees understand that core, they will be able to deal with new technologies. It is hoped that, by taking advantage of the knowledge and skills they acquired at the Center, they will be able to cope with the ongoing evolution of onboard software engineering accompanying the further transformation of vehicles in the coming years. We will continue to promote well-planned nurturing of software engineers at the Center in the future.

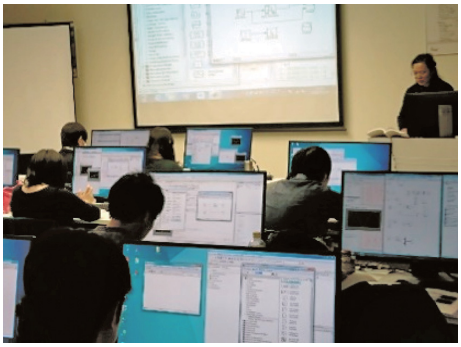


図-1　MATLAB 操作教育
Fig. 1　Lecture on MATLAB operations

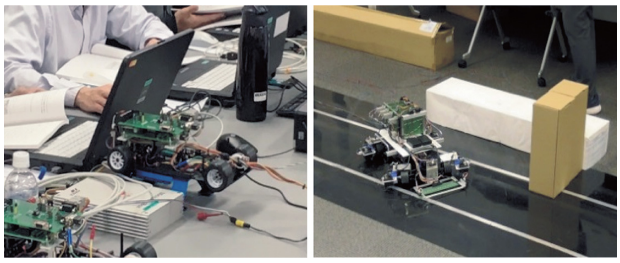


図-2　プロトタイプモデルのテスト環境
Fig. 2　Prototype model test environment

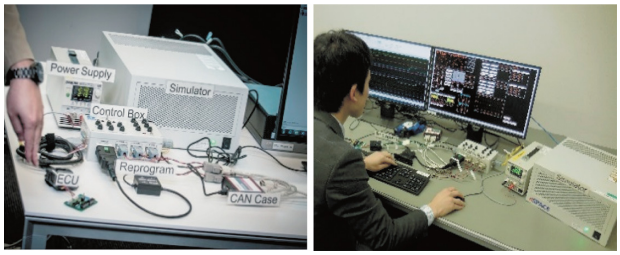


図-3　ECU テスト環境（HILS）
Fig. 3　ECU test environment (HILS)

8. 参 考 文 献

1）山田元美：組込みシステムにおけるモデルベース開発（MBD）技術者のスキル基準、JMAAB、SEC journal 、第13号、pp. 48-51（2008）.

8. References

1) M. Yamada: Engineers' skill standards for model-based development (MBD) in embedded systems, JMAAB, SEC Journal, Vol. 13, pp. 48-51 (2008).

■著者／ Author(s)■

山 本 幸 輝
Yukiteru Yamamoto

長谷川 美和子
Miwako Hasegawa

小 野 裕 二
Yuji Ono

社外技術賞受賞一覧表

1. 技術賞

〈2017年11月～2018年10月〉

※主要な技術賞、論文賞、貢献･功労賞を対象に掲載しております。
※所属は受賞時の所属、（ ）は研究開発当時の部署。
※敬称略。

受賞年月	賞名	受賞技術	受賞者
2017.11	平成29年度神奈川県技能者等表彰 〔神奈川県〕	優秀技能者	パワートレイン品質技術部　中川　俊 実験試作部　中司 隆一 パワートレイン実験部　塚本 恵治 車体技術部　阿部 重春 横浜工場　望月 慎一 実験試作部　櫻井 大二朗 塗装樹脂技術部　佐々木 公 新車生産準備技術センター　小林 正輝
		青年優秀技能者	車体技術部　尾曲 隆一 横浜工場　稲守 大樹 横浜工場　桒田 夕香里 実験試作部　朝久 雄太 パワートレイン実験部　坂本 元貴 追浜工場　田中 基臣 新車生産準備技術センター　池田 信也 実験試作部　三浦 善道 追浜工場　御原 義久
2017.11	平成29年度栃木県知事表彰 〔栃木県〕	卓越した技能者	栃木工場　蓮　義則
2017.11	平成29年度栃木県職業能力開発協会長表彰 〔栃木県〕	卓越した技能者	栃木工場　山田　正
		職業訓練功労者	栃木工場　佐藤　靖
2017.11	平成29年度福岡県優秀技能者表彰 〔福岡県〕	卓越した技能者	日産自動車九州　瀬川 忠臣
2017.11	平成29年度福岡県勤労者知事表彰 〔福岡県〕	製造業	日産自動車九州　谷本 新吾
2017.11	平成29年度厚生労働大臣表彰 卓越した技能者（現代の名工） 〔厚生労働省〕	第1部門 鋳込工	栃木工場　出頭 光好
		第2部門 数値制御金属工作機械工	横浜工場　中村 豊作
2018.3	2017年秋季大会学術講演会 優秀講演発表賞 〔公益社団法人自動車技術会〕	直噴ガソリンエンジンのPN低減技術の研究（第1報）― Tip-soot の発生メカニズムとそのキーパラメータ ―	パワートレイン・EV先行技術開発部　今岡 佳宏
2018.4	IEEE IAS Electrical Machines Committee Prize Paper Awards for Papers Presented at ECCE 2017 Second Prize 〔IEES IAS〕	Principle of Variable Leakage Flux IPMSM Using Arc-Shaped Magnet Considering Variable Motor Parameter Characteristics Depending on Load Current	EVシステム研究所　加藤　崇 EVシステム研究所　松浦　透 EVシステム研究所　佐々木 健介 EVシステム研究所　谷本　勉

〈2017年11月～2018年10月〉

受賞年月	賞　　名	受　賞　技　術	受　賞　者
2018.6	**品質工学賞発表賞** **銀賞** 〔公益財団法人精密測定技術振興財団〕	製造技術での品質向上の取組み 第1報～第3報	車両生産技術統括部　西野 眞司 車両生産技術統括部　會場 達夫 車両生産技術統括部　近藤 智昭 成 形 技 術 部　高橋 正也 成 形 技 術 部　合田 知男
2018.6	**2018年度日本溶射学会賞** **技術賞** 〔一般社団法人日本溶射学会〕	自動車エンジン用溶射ボアコーティングの開発と量産化	パワートレイン技術企画部　松山 秀信
2018.10	**2018 年春季大会学術講演会** **優秀講演発表賞** 〔公益社団法人自動車技術会〕	軸受面圧平準化効果を狙った可変圧縮比エンジン VC-T 用マルチリンク部品の構造	エンジン＆ドライブトレイン技術開発部　田辺 孝司
2018.3	**平成29年度日本機械学会奨励賞** **（技術）** 〔一般社団法人日本機械学会〕	油圧回路の不安定振動の解析と安定設計の開発	パワートレイン・EV先行技術開発部　山藤 勝彦
2018.6	**平成29年度論文顕彰** 〔公益財団法人油空圧機器技術振興財団〕	電磁比例弁内のスプールに作用するクーロン摩擦力に起因した不安定振動の解析と安定化させるための設計法	パワートレイン・EV先行技術開発部　山藤 勝彦 東 海 大 学　山本 建 電 気 通 信 大 学　澤田 賢治

2. 製品ほか受賞

〈2017年11月～2018年10月〉

※主要な製品賞を対象に掲載しております。

受賞年月	受賞車（製品）、その他	受　賞　名	主　催
2017.11	日産自動車株式会社テクニカルセンター	平成29年度神奈川県技能者等表彰 ・技能検定推進優良事業所・団体	神奈川県
2017.11	Nissan NP300 Frontier	Car of The Year Brazil ・Best Pickup 2018	（ブラジル）「Autoesporte Magazine」誌
2017.11	Nissan LEAF	CES 2018 Innovation Awards ・Vehicle Intelligence and Self-Driving Technology category	（米）Consumer Technology Association
2017.12	日産リーフ搭載技術	2017-2018日本自動車殿堂イヤー賞 ・カーテクノロジーオブザイヤー	特定非営利活動法人日本自動車殿堂
2017.12	3.0L Turbocharged DOHC V-6 (Infiniti Q50)	2018 Wards 10 Best Engines	（米）「WardsAuto」誌
2018.1	Nissan LEAF N-Connecta	What Car? Car of the Year 2018 ・Best Electric Car	（英）「What Car?」誌
2018.2	CO_2排出量削減に寄与するe-POWER技術	平成29年度省エネ大賞 製品・ビジネスモデル部門 省エネルギーセンター会長賞	一般財団法人 省エネルギーセンター
2018.3	Nissan LEAF	2018 World Green Car	2018 New York International Auto Show
2018.4	Micra	FirstCar Awards 2018 ・Car of the Year	（英）「FirstCar」誌
2018.6	Nissan Energy Electric Vehicle Power Solutions, France	FT/IFC Transformational Business Awards 2018 ・Excellence in Climate Solutions category	（英）The Financial Times (FT) and the International Finance Corporation (IFC)
2018.6	事業者間の連携の取組による受賞（株式会社Ｊ-オイルミルズ・日産自動車株式会社）	平成30年度ヨコハマ温暖化対策賞	横浜市
2018.7	Qashqai	Auto Express 30th Anniversary Awards ・Car of the Past 30 Years	（英）「Auto Express」誌、「evo」誌、「Octane」誌
2018.10	キューブ	2018年度グッドデザイン賞 ・ロングライフデザイン賞	公益財団法人日本デザイン振興会

IEEE Industry Applications Society, Industrial Power Conversion Systems Department, Electric Machines Committee, 2018 Second Paper Award

モータパラメータの負荷電流依存性を考慮した円弧磁石型可変漏れ磁束モータの基礎特性

Principle of Variable Leakage Flux IPMSM Using Arc-Shaped Magnet Considering Variable Motor Parameter Characteristics Depending on Load Current

加藤　崇* Takashi Kato　松浦　透* Toru Matsuura　佐々木　健介* Kensuke Sasaki　谷本　勉* Tsutomu Tanimoto

1. はじめに

電動車の主駆動用モータでは、小型・高効率化の観点から埋込磁石同期モータ（以下、IPMSM）が主に用いられている。また近年の更なる電費向上や航続距離拡大の要求に対しては、モータの高効率領域を拡大するための可変技術が盛んに研究されている。特性可変モータの有望案の一つとして、我々の研究グループでは可変漏れ磁束型モータ（以下、VLF-IPMSM）を提案している。この方式は回転子の磁気回路形状の変更のみで可変特性が得られるため、他の可変方式のようなアクチュエータや半導体スイッチなどの追加要素が不要であることが特徴の一つである。

このVLF-IPMSMでは一般的なIPMSMと異なり、磁石磁束が電流負荷に応じて受動的に変化するため、電機子磁束と磁石磁束との干渉や非線形現象である磁気飽和を考慮した設計がより重要となる。

本論文では、一般に困難とされるd軸上の磁石磁束成分と電機子磁束成分を正確に分離する近似解法を提案し、磁石磁束鎖交数Ψ_aを電流の関数$\Psi_a(i_d, i_q)$として導出することで、可変漏れ特性による効率向上のメカニズムを明らかにした。また同手法を適用して原理検証機の試作・評価を行い、可変漏れ磁束特性による高効率領域の拡大が可能であることをシミュレーション及び実験により示した。

2. 概　要

一般にIPMSMのトルクは、電流ベクトルi_{dq}と磁束ベクトルλ_{dq}の外積として表される。VLF-IPMSMではdq軸磁束が電流の関数となるためトルクは(1)式のように表される。磁束ベクトルのd軸成分λ_dは(2)式のように磁石磁束Ψ_aと電機子磁束$L_d i_d$の項からなるが、実験・解析で取得できるのは合計磁束量であるλ_dのみであり、原理的に磁石磁束項のみを抽出することはできない。

1. Introduction

Interior permanent magnet synchronous motors (IPMSMs) are the principal drive motors used on electric vehicles (EVs) because of their compact size and high efficiency. Variable technologies have been actively researched in recent years for expanding the high-efficiency operating region of the motor in response to demands for further improvement of electric energy consumption and extension of the driving range. Our research group has proposed a variable leakage flux IPMSM (VLF-IPMSM) as one promising type of variable characteristic motor. One major feature of this motor is that it does not need additional elements such as an actuator or semiconductor switches like those required by other variable motor systems because variable characteristics can be obtained by changing only the magnetic circuit configuration of the rotor.

This VLF-IPMSM differs from ordinary IPMSMs in that magnetic flux varies passively according to the current load. This makes it more important to execute the motor design by taking into account interference between armature magnetic flux and magnet magnetic flux as well as nonlinear magnetic saturation.

This article proposes an approximate solution method for accurately separating the magnetic flux component of the magnet and the armature magnetic flux component on the d-axis, which is generally thought to be extremely difficult. The mechanism by which the variable flux leakage characteristic improves efficiency is made clear by deriving the magnetic flux linkage $\Psi_a(i_d, i_q)$ as a function of the d-q axis current. This method was applied to construct and evaluate a proof-of-principle prototype motor. It was shown by simulation and experiments that the variable leakage flux characteristic makes it possible to expand the high efficiency operating region of the motor.

2. Overview

The torque of an IPMSM is generally expressed as the cross product of the current vector i_{dq} and the magnetic flux vector λ_{dq}. For the VLF-IPMSM, torque is expressed as shown in Eq. (1) because the d-q axis magnetic flux is

*EVシステム研究所／EV System Laboratory

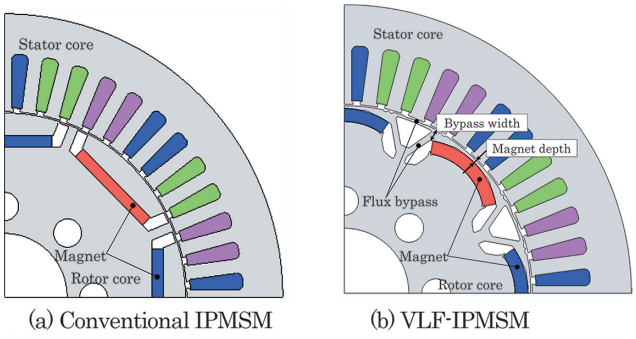
$$Tr = P\,[\lambda_{dq}(i_d, i_q) \times i_{dq}] \quad \cdots (1)$$

$$\lambda_d = \Psi_a(i_d, i_q) + L_d(i_d, i_q)\, i_d \quad \cdots (2)$$

そこで本論文では、d軸電流i_dが微小量Δi_dだけ変化した時のΨ_a項の変化量が$L_d\, i_d$項の変化量に対して微小であることを利用し、(3)式を用いて$\Psi_a(i_d, i_q)$を近似的に解析する手法を提案した。

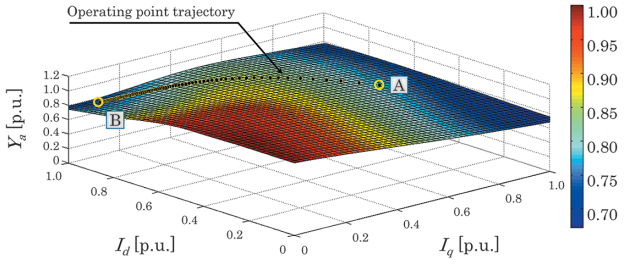
$$\begin{bmatrix} \Psi_a'(i_d, i_q) \\ L_d'(i_d, i_q) \end{bmatrix} = \frac{-1}{\Delta i_d} \begin{bmatrix} i_d - (i_d + \Delta i_d) \\ -1 \qquad 1 \end{bmatrix} \begin{bmatrix} \lambda_d'(i_d, i_q) \\ \lambda_d(i_d, i_q) \end{bmatrix} \quad \cdots (3)$$

図1に示すモータモデル（従来型IPMSM、及びVLF-IPMSM）について、提案手法を用いて電流負荷に対する磁石磁束$\Psi_a(i_d, i_q)$の特性を解析した結果を図2に示す。また同図上には最大出力ライン上で駆動した場合の動作点軌跡を併せてプロットしている。まず図2(a)の従来型IPMSMでは、電流0の無負荷状態で磁石磁束Ψ_aが最大値をとり、最大トルク条件Aでは磁気飽和によりΨ_aが減少していることが分かる。次に図2(b)に示すVLF-IPMSMでは、無負荷状態では漏れ磁束特性によりΨ_aが最小値をとり、電流

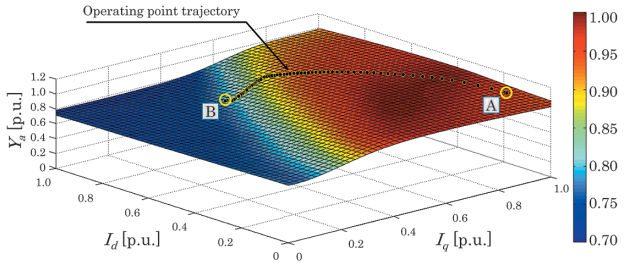


(a) Conventional IPMSM　(b) VLF-IPMSM

図-1　概略図の比較
Fig. 1　Schematic drawings for comparison



(a) Conventional IPMSM



(b) VLF-IPMSM

図-2　$\Psi_a(i_d, i_q)$ マップの比較
Fig. 2　Comparison of $\Psi_a(i_d, i_q)$ variation maps

a function of the current. The d-axis component λ_d of the magnetic flux vector is composed of the terms of the magnetic flux Ψ_a and the armature flux $L_d\, i_d$ as shown in Eq. (2). However, only the total amount of magnetic flux λ_d can be obtained experimentally or by simulation. In principle, it is not possible to extract only the magnetic flux term.

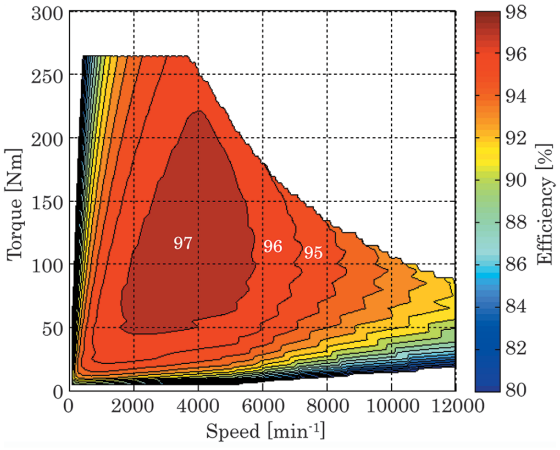
$$Tr = P\,[\lambda_{dq}(i_d, i_q) \times i_{dq}] \quad \cdots (1)$$

$$\lambda_d = \Psi_a(i_d, i_q) + L_d(i_d, i_q)\, i_d \quad \cdots (2)$$

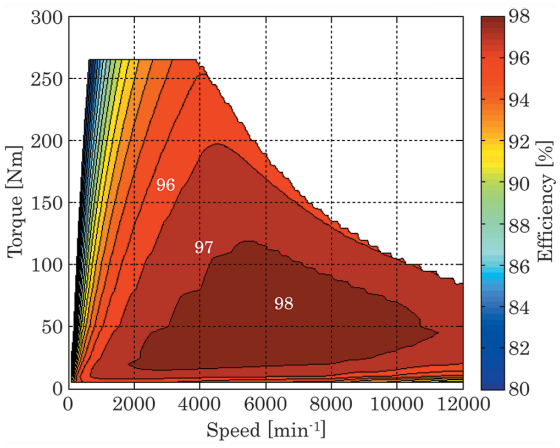
Therefore, this article proposes a method of approximately simulating $\Psi_a(i_d, i_q)$ by using Eq. (3), in which the amount of change in the Ψ_a term relative to the amount of change in the $L_d\, i_d$ term is infinitesimal when the d-axis current i_d is changed only by a minuscule amount Δi_d.

$$\begin{bmatrix} \Psi_a'(i_d, i_q) \\ L_d'(i_d, i_q) \end{bmatrix} = \frac{-1}{\Delta i_d} \begin{bmatrix} i_d - (i_d + \Delta i_d) \\ -1 \qquad 1 \end{bmatrix} \begin{bmatrix} \lambda_d'(i_d, i_q) \\ \lambda_d(i_d, i_q) \end{bmatrix} \quad \cdots (3)$$

The proposed method was used to simulate the magnetic flux $\Psi_a(i_d, i_q)$ characteristics relative to the current load for the conventional IPMSM and VLF- IPMSM models shown in Fig. 1. The results are shown in Fig. 2. The operating point trajectory of the motors when operated at the maximum output line are also plotted in the figure.



(a)　Conventional IPMSM



(b) VLF-IPMSM

図-3　効率マップの比較
Fig. 3　Comparison of efficiency maps

負荷の増加に伴って最大トルク条件Aでは磁石磁束Ψ_aが概ね最大値を得られていることが確認できる。また高速条件Bでは、d軸電流による漏れ磁束の促進によりΨ_aが減少し、結果として少ない弱め界磁量で駆動できていることが動作点軌跡から分かる。結果、高速域での弱め界磁電流抑制による銅損の低減、磁束可変による鉄損の低減の効果が相まって、従来型IPMSMに対して高効率範囲を高速域まで大幅に拡大することができた（図3）。

3. おわりに

提案手法により、車両駆動用モータとして最適な可変磁束特性の設計・検証が可能となった。この可変磁束の技術は、特に低～中負荷、高速域における電費向上に有効であることから、本技術が今後の電動車の更なる魅力・性能向上の一助となることを期待する。

The results for the conventional IPMSM in Fig. 2(a) show that magnetic flux Ψ_a is maximum under a no-load condition of zero current and that it decreases due to magnetic saturation under maximum torque condition A. It is observed for the VLF-IPMSM in Fig. 2(b) that Ψ_a is minimum due to the leakage flux characteristic under a no-load condition and that approximately the maximum value of magnetic flux Ψ_a is obtained under maximum torque condition A accompanying the increase in the current load. Under high-speed condition B, Ψ_a decreases due to the promotion of leakage flux by the d-axis current and the operating point trajectory indicates that as a result the VLF-IPMSM can operate under a small amount of field weakening. As a result, reduction of the copper loss due to suppression of the field-weakening current in the high-speed region combines with the iron loss reduction due to variable flux to markedly expand the high-efficiency region of the VLF-IPMSM to the high-speed range compared with that seen for the conventional IPMSM (Fig. 3).

3. Conclusion

The proposed method made it possible to design and validate the optimum variable magnetic flux characteristic for an EV drive motor. This variable magnetic flux technology is effective for improving electric energy consumption especially under low to medium loads and in the high-speed range. Accordingly, it is expected to further improve the attractiveness and performance of EVs in the coming years.

※IEEE IAS Industrial Power Conversion Systems Department, Electric Machines Committee, Prize Paper Awardsは、ECCE（Energy Conversion Congress & Expo.）に投稿された論文の中から、上位3件に贈られる。

The Electric Machines Committee in the IEEE-IAS Industrial Power Conversion Systems Department presents Prize Paper Awards to the top three technical papers among those submitted in this technical field to the Energy Conversion Congress and Exposition..

編　集　後　記

今回の特集は「ソフトウェア」です。日産技報でソフトウェアが特集されるのは、今回が初めてとなります。日産車に始めてソフトウェアが搭載されたのは1979年で、以後およそ40年が経過しましたが、いまだに車載ソフトウェアの規模拡大はとどまるところを知りません。この増大し複雑化するソフトウェアを効率的に開発し、一方で品質を確保し、日程通りにお客様に商品を届けていくため、日産自動車では、多くの技術開発を行い、プロセスを整備してきました。今回は、これらの技術やプロセスに焦点を当てて、特集を編成しました。また、内容的にも、ソフトウェアエンジニアだけでなく、一般の自動車エンジニアにも理解できるように、なるべく平易な内容となるように心がけました。さらに、サイバーセキュリティや人工知能など、先端的な技術トピックスも織り込むようにしました。

今後、自動車に占めるソフトウェアの役割は、ますます増大し、ソフトウェアが自動車の死命を制するとも言われています。今回の特集により、自動車にとってのソフトウェア技術を俯瞰（ふかん）することができ、今後の技術開発の参考となることを祈念いたします。

― ソフトウェア開発部・石 上 和 宏 ―

日産技報第84号

発行　2019年3月
発行・編集人　日産技報編集委員会
発行所　日産自動車株式会社 総合研究所 研究企画部
神奈川県厚木市森の里青山1番1号
〒243-0123
印刷所　相互印刷株式会社
東京都江東区森下3－13－5

Editorial Postscript

The special feature in this issue focuses on software. This is the first time the Nissan Technical Review has devoted a special feature to software. Approximately 40 years have passed since we first adopted software on Nissan vehicles in 1979. The ongoing expansion of the scale of onboard software still knows no end. At Nissan, we have undertaken enormous R&D work and put in place related processes in order to develop efficiently onboard software that is continually increasing in quantity and complexity, confirm its quality, and deliver products to customers on schedule. The articles organized in this special feature focus on these software technologies and processes. Careful attention has been taken to make the contents as plain as possible so that they will be readily understandable not only to software engineers but also to automotive engineers in general. At the same time, topics related to cutting-edge technologies such as cybersecurity and artificial intelligence have also been included.

The role performed by software in vehicles will continue to increase in importance in the years ahead, and it is said that software will govern the future lifeblood of vehicles. It is hoped that the comprehensive review of automotive software provided in this special feature will be a useful reference for future R&D activities.

Kazuhiro Ishigami
Software Engineering Department

Nissan Technical Review 84

March, 2019
Publisher Nissan Technical Review
(Editor) Editorial Committee
Distributor Research Planning Department
Nissan Reseach Center
NISSAN MOTOR CO., LTD.
1-1, Morinosatoaoyama, Atsugi-shi
Kanagawa, 243-0123, Japan

表紙コンセプト／ Cover Design Concept

ソフトウェア開発を行うオフィスは、パソコンが並んでいるありふれたオフィスであることが多い。しかし、そのパソコンの画面上では、クルマの制御アルゴリズムが表現され、そのアルゴリズムに誤りがないかのレビュー、評価計画の検討、実際の品質確認が実行されます。一連の開発作業がパソコン上で見える形で表現されて、これに基づき、エンジニアは開発を行います。今回の表紙デザインは、見えないソフトウェアを見えるようにすることで、ソフトウェアエンジニアのアイデアが、パソコン上で具現化されていくことをイメージしました。百年に一度の変革を実現するため、今後もパソコン上に表現されるソフトウェアを介して、クルマのイノベーションを推進していきます。

Many offices engaged in developing automotive software are ordinary workplaces with row upon row of personal computers. However, shown on the computer screens are vehicle control algorithms that engineers are reviewing for errors, studying quality evaluation plans or checking the actual quality. This series of tasks involved in software development is expressed in a visible form on the computer screens, and engineers carry out the development work on that basis. By presenting invisible software in a manner than be seen, the cover design of this issue represents the process in which the ideas of software engineers are embodied in concrete forms on their computer screens. We will continue to promote vehicle innovations through the medium of the software displayed on our computer screens in order to accomplish revolutionary changes that occur once in a hundred years.

石上　和宏
Kazuhiro Ishigami
ソフトウェア開発部
Software Engineering Department

ISSN 0385-9266

日産自動車株式会社　日産技報編集委員会