安全なソフトウェアシステムを実現するための新たなアプローチ

マサチューセッツ工科大学 教授
ナンシー・レブソン

×

SEC 所長
松本　隆明

IPA は 1 月 21 日、システムやソフトウェアの安全性に関する特別セミナーを開催した。登壇者の一人は、その分野の世界的な第一人者であり、「Safeware」や「Engineering a Safer World」などの著作で知られる、マサチューセッツ工科大学のナンシー・レブソン教授である。ソフトウェアに起因する障害やその分析、原因を導き出す方法といった安全性向上に向けた様々な対策を、先進的な手法と適用事例を含めて話していただいた。システムが複雑化している今、旧来の分析手法だけでは十分ではない。新しく、かつ安全なシステムを実現するためのアプローチについて、セミナーに先駆けてレブソン教授に伺った。

松本：本日はソフトウェアシステムの安全性について、世界でも第一人者のレブソン先生からお話を伺いたいと思います。安全性とは何か、システムの安全性を確保するにはどうしたらよいかを考察していきます。

レブソン：わかりました。まずは、このような機会をいただいたことを大変うれしく思っております。

ナンシー・レブソン

米国マサチューセッツ工科大学　航空宇宙工学部門及び工学システム部門教授。全米工学アカデミー会員。
ソフトウェア安全という新分野の創設者であり、現在も世界の第一人者。
これまでに 200 件以上の論文を執筆し、その研究結果や手法は、航空宇宙、交通運輸、科学プラント、原子力、医療機器など、安全に係わる多種多様な産業で応用されている。
代表的な著書に「セーフウェア―安全・安心なシステムとソフトウェアを目指して―」があり、最近の著書「Engineering a Safer World」(MIT Press) は 2012 年に出版された。

松本：こちらこそお越しいただき、ありがとうございます。では初めに、先生のご著書「Engineering a Safer World」の内容からお伺いします。本の中では、これまでのエンジニアリングの安全性に関する仮説と逆説がいくつか紹介されていますね。

安全性と信頼性の混同があるという仮説があります。この大きな違いというのはどこにあるとお思いですか。

レブソン：まず前提として、安全性や信頼性に関するこれまでのエンジニアリングが手法として確立されて以降、世界は変わりました。そもそも、FTA※1 や HAZOP※2、ETA※3 といったテクニックや確率論的なリスク分析が発明されたのは、コンピューターが広く普及する前です。その後、様々な場面でコンピューターが活用されるようになり、システムはより拡大・複雑化しました。これによって、ソフトウェアやコンピューターはそれまでのハードウェアとは異なり、エンジニアリングの方法を変えていったのです。

システムがシンプルだったときには、電子機械的な面で使用前に様々なテストを実施し、ほとんどの設計エラーを排除することが可能でした。しかし、コンピューターではあらゆるテストをし尽くすことは不可能です。どうしてもシステムの中には、設計のエラーが残ってしまいます。いま安全性に関する問題のほとんどは、そのエラーが原因になっているのです。安全性の問題は必ずしも、コンポーネントの故障に拠りません。

もうひとつの問題は、システムの複雑化によって、人間に拠る手動制御が難しくなったということです。システムを理解していない人間の起こすエラーは、ほとんど排除できない設計になっています。それにも関わらず、事故が起きたときには人間に責任があるといわれる。現在では、コンピューターが使われるようになる前に開発された安全・分析の古い技術は、もはや成り立っていないということです。

【脚注】

※1　FTA：フォールトツリー解析。ハザードの原因となりうる個々の障害の組み合わせをブール理論を用いて記述するトップダウンの検索方法

※2　HAZOP：設計で期待した運用から考えうる全ての逸脱と、その逸脱に関連するすべてのハザードを識別する手法

※3　ETA：イベントツリー解析。ディシジョンツリー形式で、何らかの故障とそれに続いて起こり得る一連の事象の結果を識別するために順方向検索を用いる手法

私が「Engineering a Safer World」を執筆したのは、これまでとは異なるタイプの問題が起き始めている現状を伝えるためでもあります。多くの複雑なソフトウェアから成るシステムで、新技術を活用し、どのように安全性を向上させていけばよいのかを説明したいと考えたのです。

松本：つまり、大きな問題は２つ。ひとつは、複雑で大規模なシステムではコンポーネント同士のインターフェースが非常に増えるため、コンポーネントの安全性のみに着目するだけでは不十分だという点と、今までは機械的なもので人間の関与がほとんどなかったのに、現在のシステムにはオペレーターやユーザーといった“人”の関与が多いという点でしょうか。

レブソン：そうですね。ただ、問題はコンポーネント間の相互作用あるいはインターフェースだけではありません。まずは、システム設計がソフトウェア開発の一部になってきていることに注目しなくてはならないのです。実は多くの事故が、特別な故障のないコンポーネントの中で起きています。確かにコンポーネント間の相互作用を制御するのはソフトウェアです。そのうえシステム設計や、従来ならオペレーターが担っていた作業も、ソフトウェアが行うようになってきました。

私はソフトウェアに関する数百件もの事故を調査・研究してきましたが、すべての事例でソフトウェア自体は要件を満たしており、実装も正確でした。しかし、もともとの要件が間違っていれば、コンポーネントが適切であっても事故は起こるのです。問題がソフトウェア・エンジニアリングではなくシステム・エンジニアリングの側にあることは、珍しくないのです。システム・エンジニアはソフトウェアを理解していません。事故をなくしていくためには、ソフトウェア・エンジニアとシステム・エンジニアが協力し、お互いに学び、関わり合うことが必要なのです。

松本：その点についてはまったく同感です。私はソフトウェア・エンジニアですが、システム・エンジニアの方と話すと、考え方に相違があると感じます。特に、ソフトウェアは物理的な法則に基づく装置や機械と違い、ロジカルに考えて作るため、発想の仕方から異なるように思います。ただ、システムの安全性を上げてゆくためには、当然ながら個々のコンポーネントの信頼性を上げていくことも必要ですよね。

レブソン：はい。ただし、どれだけ信頼性の高いコンポーネントだったとしても、システムの問題が設計部分にある場合は、安全性の向上にはつながりません。

松本：個々の信頼性を上げるだけでは十分でないということですね。最近のソフトウェア開発では、要求に合ったソフトウェアなのかをチェックしたり、上流工程での設計品質を上げたりすることを重要視します。そこで最も大切な問題は、きちんとした要求条件をどのように正しく策定するかではないでしょうか。

レブソン：はい。それが現在できる対策だと思います。新しい分析の手法によって可能になってきました。旧型のアプローチでは、ハザードや安全の分析を設計の中で取り入れていかなければなりません。しかし、システム理論に基づいた新しいアプローチを使えば、コンセプトを作る早い段階で分析が可能となるため、形式的手法により数学的に安全要件を導き出せます。ただ、そうした流れを作るには、従来のような信頼性の理論からシステム理論、システム・エンジニアリングの考え方に移行していく必要があるでしょう。

松本：安全性要求の考え方を、システム的な観点からきちんと決めていくことが重要ですね。

ルートコーズを求めるだけでなく全体像をとらえる

松本：それでは、著書にある２つ目の仮説の話に進みます。先生の考案した事故原因の分析モデルについてです。まず、今までイベントの連鎖として考えていた分析方法は、必ずしも正しくないのでしょうか？　私たちは事故の原因分析をする際、やはり「ルートコーズ（根本原因）」の発見を一番の目的にしてしまいがちです。

レブソン：イベントの連鎖がないということではありませんが、複数の原因が絡み合う相互作用や関係を考えていく際、イベントの連鎖を使うとどうしても限界が出てきます。直接の因果関係や原因だけでなく、間

松本 隆明（まつもと　たかあき）

1978 年東京工業大学大学院修士課程修了。同年日本電信電話公社（現 NTT）に入社、オペレーティング・システムの研究開発、大規模公共システムへの導入 SE、キャリア共通調達仕様の開発・標準化、情報セキュリティ技術の研究開発に従事。2002 年に株式会社 NTT データに移り、2003 年より技術開発本部本部長。2007 年 NTT データ先端技術株式会社常務取締役。2012 年 7 月より独立行政法人情報処理推進機構（IPA）技術本部ソフトウェア高信頼化センター（SEC）所長。博士（工学）。

接的な関係に注目しなくてはならないのです。
東日本大震災の福島を例に考えてみましょう。原発に事故が起きた原因は、津波や外壁だけではありません。政府の諸機関と関連する会社の仕組みも、おそらく問題のひとつだったことでしょう。もし直接的なルートコーズだけに注目すると、福島の原発事故では津波や防潮壁の高さ、電源の設置場所にのみ焦点を当てることになります。原発の問題にとどまらず、全体的な社会の安全という視点で問題解決を図り、将来に備えていくのならば、本来は全体像をとらえる必要があります。つまり、原因になりえたすべての事象に目を向けるべきなのです。
人はシンプルな解答を求めるあまり、根本原因さえ直せばよいというシンプルな答えにたどりつく傾向があります。ひとつの原因を解消しさえすれば、問題はすべて解決できると思いたいのです。こうした根本原因の概念は、システム制御に関する幻想だといえるでしょう。マネージャーは自分でコントロールできると思いたいけれど、ひとたび大きな問題が起きれば圧倒され、コントロールが不可能な状態に陥ることもある。私たちが見ているのは、大きな問題の中にある一現象にすぎません。視野が狭いために、同じ根本から別の事故が発生してしまうこともあるのです。
米国で、沿岸警備隊が起こした複数のヘリコプター事故に関する調査が行われました。通常の手法で分析した際には複数の事故に類似性はなく、度重なる事故は偶然の一致であるように見えていました。そこに、私たちの新しい事故分析の手法を使ったのです。すると共通要因が見いだされ、的確な解決を果たすことができました。
松本：原因は必ずしもひとつに集約されるわけではなく、複数の要因が絡んで事故が起きるのですね。ただ、イベントの連鎖を遡って考えること自体は悪くないという認識でよろしいでしょうか。
レブソン：イベントの連鎖は間違った考え方ではないですが、十分ではありません。事故に影響した直接的な原因にとどまらず、因果関係のモデルを拡張して考えていかなければならないでしょう。
現代社会では、様々な新技術が使われています。ソフトウェアはもちろん、システムの複雑性も増しています。こうした時代に安全問題を考えるには、単なるイベントの連鎖を越えて、社会的システムとして思考を巡らせていかなくてはなりません。考え方を抜本的に変えていく必要もあるでしょう。新たな方法を展開していく場面で、人は現在の考え方を少しだけ変えようとしがちですが、それではうまく機能しません。ソフトウェアやシステムの現状を踏まえた、パラダイムシフトが必要になります。従来とはまったく異なるやり方で、システム・エンジニアリングを考えていかなければならないのです。
私たちが集めた多くの事例では、新旧様々な手法で分析しましたが、やはり新しい手法を使ったほうがより多くの原因や予防策を見つけられました。最近、米国の EPRI（米国電力研究所）では、原子力発電所の設計を複数の専門家に渡し、FTA、ETA や HAZOP などを含めた異なる手法で、それぞれ分析してもらう研究を実施しました。そこで私たちの手法「STPA」を使ったグループは、ほかよりも多くの問題を見つけることができました。今までその原子力発電所ではいくつもの事故が起こっており、それは分析者たちには事前に知らされていませんでしたが、私たちのグループのみが実際に起こったその事故を分析から発見できました。問題があったのが、コンポーネントではなくシステム設計だったからです。様々な業界で同種の例が多数あり、コンポーネントの信頼性が安全に関わるという古い考え方は、現在は十分ではないといえるでしょう。

適切な分析を実施するための取り組み

松本：分析をする際は、技術的な問題はもちろん、社会システムの問題や人間の精神的・感情的な問題といった様々な側面から原因を考えなければならないですね。しかし、そのためには幅広いスキルを持った人材が必要になります。その点はどのような解決法があるでしょうか。
レブソン：今後は、システム的な考え方のシステム・エンジニアリングの教育がより重要になってくるでしょう。また、分析の作業をグループ単位で行うことも必要です。ソフトウェア・エンジニアやシステム・エンジニア、アプリケーション・エンジニアたちが一丸となって問題に当たらない限り、本当の解決には至らないと思います。コンピューターに命令を与える機能しかないソフトウェアは、エンジニアたちにとっては胃潰瘍の原因になるかもしれませんが（笑）、それ自体が人に傷害を負わせたり、火災や爆発の原因になることはありません。つまり、ソフトウェア自体に安全が欠如しているというわけではないのです。しかし、安全性の欠如しているシステムの中にソフトウェアがある場合は、危険を生み出しかねない。システムは、火災や爆発といった事故につながってしまう可能性があるのです。ただ、コードを実装するだけのソフトウェア・エンジニアもいるでしょうから、全員がその責任を担わなければならないとは思いません。シス

テム・エンジニアと協業しなければいけないソフトウェア・エンジニアの数は限られるといえるでしょう。

松本：私たちは、各企業から過去の事例を収集して、お互いに共有する事業に取り組んでいます。

レブソン：事例の共有も有効ですね。ただ、残念ながら、事故が起こる原因はほとんど毎回異なるものです。

松本：本当にそうですね。現象を見ると、実は類似性のある事故は頻発しています。たとえば、日本では最近、社会を支えるインフラシステムに障害が起きました。あらかじめサーバーを二重化しておき、片方が壊れた際はもう片方に切り替えるのですが、その切り替えの失敗が多発しているのです。通信、証券、金融、流通など様々な業界で、切り替えがうまくいかず、システム全体がダウンしてしまう事態が起きています。そこには何か共通の原因があるのではないかと思い、現在、分析をしているところです。様々な側面から検討していますが、単に技術的な問題だけでなく、システムの裏には企業ごとの文化や管理方法などもあるため、同じ原因が存在するといえるのか、判断が難しいと考えています。その点をしっかり把握しないと、事例を教訓として共有する意味が薄れ、汎用性がなくなってしまう。そこで私たちは、できるだけバックグラウンドを整理し、事象が起きた状況をコンテキストにまとめるようにしています。そうした活動についてのご意見をお聞かせください。

レブソン：冗長性やコンポーネントの再利用は、ハードウェアの信頼性に対して主たる解決策ですが、ソフトウェアには使えません。冗長性を持つことは、同じエラーを繰り返してしまうことだともいえます。再利用についても同様ですね。過去 10 年間で起きたすべての宇宙船事故の事例では、ソフトウェアが再利用されていました。同じコードが別の宇宙船で使われ、事故が発生していたのです。

松本：冗長性の問題は、ソフトウェアではなかなか解決できません。しかし、事故が多発している現状を踏まえ、ソフトウェアが直接的な原因でないとしても、実態を明らかにする必要性があるのではないでしょうか。

レブソン：共通の要因を探すのは、確かによいことです。ただ、その作業にばかりとらわれて、違う要因だけれどもお互いに結びついているかもしれない部分をなおざりにしないよう、気を付けなければなりません。

たとえばある日本車メーカーの負けた訴訟では、ソフトウェア・エンジニアリングが適切に行われていなかったことが問題となりました。実はその訴訟の中で、“ソフトウェア・エンジニアリングが適切に行われていなかった”ことと“意図しない加速”の因果関係は証明されていません。それにも関わらず、裁判には負けてしまった。この例からも、航空や軍事の分野では長く行われてきたハザード分析やセーフティー・エンジニアリングを、自動車業界にも取り入れていくべきだということが見えてくるでしょう。これまでどおりコンポーネントの信頼性にばかりフォーカスしていては、新しくて複雑な車には機能しません。自動車業界の方は、様々な機能間の相互作用が大きな問題のひとつだとおっしゃっていました。新たな機能を搭載したとき、機能間の意図しない相互作用があるそうです。たとえば、ある状態のときにヘッドライトとワイパーの相互作用で問題が生まれるなど、以前のシンプルなシステムでは予想もできなかった事態が実際に起きています。先ほどの訴訟となったケースも、もし問題があったなら、システム的な見地から見れば発見できたことでしょう。ソフトウェア・エンジニアリングだけでなく、システム・エンジニアリングもやり方を変えていかなければならないのです。

松本：航空業界や軍需業界で行われていた手法を、自動車業界では取り入れていなかったということですね。米国には、そうした問題を解消するための、業界をまたいだ情報共有の仕組みはないのでしょうか？

レブソン：残念ながらありません。このケースでは、従来のやり方に従っていたことが問題だったと思います。航空や軍需の分野は、複雑なシステムと長く向き合ってきたため、手法が開発されました。しかし、これまでの自動車業界では、車全体は複雑なものでもコンポーネントに分割できたので、コンポーネント間の相互作用は限られていると考えられてきたのです。現在私たちは数社の大手自動車企業と協業し、このような問題に対処するために必要なシステム・エンジニアリングの変更を提案しています。

松本：カーナビゲーションシステムのソフトウェアに不具合があり、ブレーキがおかしくなったという話を聞いたことがあります。今までは独立していると思っていた部分がそれぞれ密接に関連して動いていたり、自動運転の登場も間近になっていますから、今後の自動車メーカーはますます車を複雑なひとつのシステムとして認識しなくてはなりませんね。

レブソン：おっしゃるとおりです。また、様々な相互作用を考えるには、車内のコンポーネントだけでは足りません。車を取り囲む環境や道路の設計からドライバーのソーシャルな部分まで、今まで考慮されていなかったことにも目を向ける必要があります。

エンジニアはほとんどのシステムにおいて“人”の部分を無視してきてしまいました。物理的なパーツのみを設計し、オペレーターが適切に使用してくれることを勝手に想定していたのです――たとえば、ブレーキに問題が起こっても、運転手が適切に対応してくれるというように。しかし、システムがあまりにも複雑になってしまった今、そうした想定は成り立ちません。システムが複雑になったことで人に責任を転嫁するのではなく、人はユーザーとしてだけでなく、システムを構成する重要な一部分だと考えなくてはならないのです。

IT のセキュリティも、ひとつの例です。ユーザーが通常できないようなことを、当たり前にさせる想定でセキュリティが組まれています。そうなると、問題があったときに責任をユーザーに押し付けることになる。たとえば IT 管理者が、とても難しいインストールの設定を適切にできなかったり、複雑なパスワードを覚えられなかったりした場合、人が悪いことになるのです。長くて意味のない数字や文字が並んだパスワードを、書き留めずに何百も覚えられる人がいるでしょうか？　これは、想定条件が悪いと言わざるを得ません。

松本：パスワードは頻繁に変更することが求められますが、正直覚えきれないのではないでしょうか。

レブソン：これは、セキュリティの進歩が足りない点だと思います。本来ならユーザーをシステムの一部として扱わなければならないのに、敵として扱ってしまっているのでしょう。

発注側と受注側の距離を埋める

松本：次の話題ですが、最近のシステムは、ソフトウェアがほとんどをコントロールしているといっても過言ではありません。その安全性を確保する困難さについて、お話を伺います。まず、ソフトウェア開発においては要求条件を考える人と実際に作る人が異なる場合が多いため、お互いにうまくコミュニケーションをとることがとても大切です。日本は米国に比べ、発注側と受注側の距離が離れている傾向が強い。要求条件を決める人が自らソフトウェアを作る“内製”が多い米国では、日本よりコミュニケーションの問題が少ないのではないかと思いますが、いかがでしょうか。

レブソン：比較できるほど日本について詳しくはありませんが、コミュニケーションが安全の大きな要求事項だというのはそのとおりだと思います。ビル・カーティスの調査では、100 以上の大規模なソフトウェア・プロジェクトを対象に、要因分析を行いました。ほとんどが失敗したプロジェクトだったのですが、成功例と失敗例の違いを調べたのです。大きな違いとして、成功したプロジェクトには、ソフトウェアが使われるシステム全体を理解している人が常に携わっていたことがわかりました。

解決策や進め方には 2 つあると思います。まずひとつは、最近、システム分析がおろそかになってきており、それを元の状態に戻してシステム分析を充分かつ綿密に行うこと。米国でも、医療のウェブサイトで「オバマケア」の大きな問題がありました。通常の原因は、開発者がソフトウェアの要求を正しく理解しないまま、業務に当たっていることです。ソフトウェア・エンジニアとプログラマーがやる仕事はそれぞれ違うのに、あまり職務の区別がされていないのも問題でしょう。プログラマーはソフトウェア開発全体での設計・実装の詳細を理解している人ですが、ソフトウェア・エンジニアは多くの企業で“コードを作る人”が昇進して大きな肩書きとしてソフトウェア・エンジニアと称しつつ今までと同じ仕事をしている場合がありますが、必要なスキルも経験も異なります。プログラマーとソフトウェア・エンジニアとでは実際やる仕事も、必要とされるスキルも異なります。先ほどお話したカーティスの調査でも、成功したプロジェクトでシステム全体を理解し責任を持っていた人間の多くはコード作りに関しては優秀でなかったという結果がみられました。

日本のように発注者と受注者が異なる場合は特に、両社間できちんとコミュニケーションがとれるインターフェースを持たなくてはなりません。たとえば軍需の分野では、プライムコントラクターが全体の責任を持ちます。もちろん大きなプロジェクトでは多数の会社に委託しますが、それらのインターフェースをつなぐのはプライムコントラクターの役目です。それが、オバマケアのウェブサイト制作では欠けていました。問題が起きた後、委託会社はそれぞれ自社に非がないことを主張していました。実際に各コンポーネントは問題なく機能しており、単体でのテストも注意深く実施されていたのです。

的確に要求仕様をまとめるためには？

レブソン：2 つ目の解決策は、要求の仕様固めです。多くの現場で、リクワイアメント・エンジニアリングがなおざりにされてしまっているのが問題です。デザインの仕様ととらえられ、機能要求として考えられていないケースも多々あります。私は本領域における 20 年ほど

の経験を活かし、要求の仕様固めについて著作でも論じてきました。現在の要求のまとめ方は、決して十分ではありません。その要求が仕様になった根拠や理由は記述されていない場合がほとんどです。使用時の様々な前提条件や、なぜそのような使い方になるのかという理由・想定についても書かれていません。ユーザーに必要なトレーニングやアクションなどは、想定条件にすら入っていないのが現状です。

松本：わかりやすい仕様書を作ることも大きな課題です。文章はどうしても表現が曖昧で誤解を生じやすくなってしまいます。先生のおっしゃるように仕様の理由を書いたとしても、なかなか伝わりにくいのではないでしょうか。

レブソン：確かに、それは大きな問題ですね。形式手法・数学的手法が解決策としてよく挙げられますが、それだけでは難しいでしょう。ソフトウェア・エンジニアには理解しかね、エラーの特定につなげられないのです。

私は 1990 年、米国政府の依頼によって「TCAS」という航空機衝突回避システムを認証しました。非常に複雑なシステムで、曖昧ではない仕様書を作るのが困難でした。使われているロジックも大変難しく、政府も認定に難色を示していたほどです。そこで私は、スペシフィケーション言語を作りました。数学的な基礎に基づいた言語なので、曖昧ではありません。誰もが 10 分で学ぶことができ、エラーを見つけられます。なかには意図を示す仕様やトレーサビリティ、機能要求も入っているため、「インテント・スペシフィケーション」とも呼んでいます。JAXA でも有用だとして、使われているものです。形式手法のコミュニティでは十分な数学的根拠がないと考えられているようですが、根底では数学的モデルを用いているため、数学的分析をすることも可能です。たとえば一貫性や完全性に欠けているものは、このモデルで導き出せます。

「Engineering a Safer World」の後半でも触れていますが、それぞれが商用ツールとしてシステムを提案していることが問題なのかもしれません。ツール開発者は自分のツールを使ってほしい気持ちが強いため、なかなか新しいアイディアが導入されないのです。ただ、私のインテント・スペシフィケーションは、有用性が実証されています。今使われているひどいツールを乗り越えていかなければ、問題は解決できません。

新しい因果関係のモデル「STAMP」

松本：お聞きしたいことがたくさんあるのですが、時間が迫ってきています。最後に、先生が提唱されている STAMP についてご紹介いただけますか。

レブソン：はい。STAMP は新しい理論的因果モデルです。因果関係に対する考え方なので、ツールでもテクニックでもありません。STAMP 以前の古いモデルはイベントの連鎖に基づいたもので、200 年も前から存在しています。現在の様々なツールは、こうした古いモデル上に構築されました。STAMP はそれをさらに人・環境にまで広げていくという考え方で、ソフトウェアが果たした役割まで考えを広く巡らせます。

松本：STAMP の特徴は、安全性を信頼性の面ではなく、コントロールとコンストレイント（制約）としてとらえることだと思っているのですが、その理解でよろしいでしょうか。

レブソン：そうです。故障の予防という従来の考え方から、ユーザーの行動を制約し実行させるというコンセプトに変えていくべきだと思っています。たとえば人間や政府の行動、組織の文化・風土、管理方法といった要素が、制約に対して違反する可能性をはらんでいます。そうした部分を変え、システムやコンポーネントの挙動に対して安全に関する制約を設けていくのです。

また、セキュリティについても攻撃や脆弱性を考えるだけでなく、制約を実行していかなければなりません。これは新たな提案なので、まだ書籍でも触れていない内容です。STAMP の考え方を土台に新たなツールを作っていけば、よりパワフルなツールが生まれます。STAMP は旧来の考え方を含んだうえに、さらに広げたものだからです。

松本：ソフトウェアシステムの安全性に関して大変貴重な示唆を伺うことができました。今後の IPA/SEC の取組みにもぜひ活かしていきたいと思います。本日はどうもありがとうございました。