Vol.2017-SE-196 No.16
2017/7/20

システム理論に基づくSTAMP/STPAを用いたソフトウェアアーキテクチャ分析の提案

日下部 茂[1,a)]

概要：ソフトウェア利用の拡大に伴い，求められる機能や振る舞いの実現といった観点だけでなく，リスクへの対処といった観点からも，ソフトウェアについて適切に記述し分析することがより重要となっている．本稿では，ライフサイクルの早期から安全性といった創発的特性に焦点を当ててソフトウェアのアーキテクチャを記述・分析する際に，システム理論に基づくSTAMP/STPAを活用することを提案する．ソフトウェアアーキテクチャのフレームワークには様々なものがあるが，ここではビューポイントとパースペクティブを用いるものをベースとする．安全性といった創発的な特性に対するパースペクティブを適用しながら，関連するビューポイントでのアーキテクチャ記述にSTAMPのモデル記述も含め，STPAでの分析も行うことを提案する．

キーワード：ソフトウェアアーキテクチャ，STAMP/STPA，安全性，創発的特性

An Extended Software Architecute Framework Using STAMP/STPA

Kusakabe Shigeru[1,a)]

Abstract: The author proposes an extended software architecture framework using STAMP/STPA for making architectural decisions on emergent properties such as safety. As the base of the framework, the author uses a software architecture framework with viewpoints and perspectives. With the perspectives related to the emergent property, we make software architecture descriptions of associated viewpoints using STAMP and analyze them using STPA. The resulting software architecture descriptions are augmented with STAMP/STPA results and quality sections corresponding to the perspectives.

Keywords: Software Architecture, STAMP/STPA, Safety, Emergent property

1. はじめに

コンピュータシステムやその接続に関する技術革新により利便性や効率の向上などがもたらされると同時に，リスクや脅威に対する注意の必要性も高まっている．そのため，機能や振る舞いの実現の観点だけでなく，リスクへの対処といった観点からも，ソフトウェアに関する相互作用を適切に記述し分析することが以前にも増して重要となっている．そのような記述や分析をライフサイクル早期から行う有効なアプローチのひとつにソフトウェアアーキテクチャに関する取組がある．本稿では，ライフサイクル初期に，安全性のような創発的な特性について，ソフトウェアアーキテクチャの記述や分析を行う際に，Nancy Levesonが提案したSTAMP/STPA(Systems-Theoretic Accident Model and Processes/System-Theoretic Process Analysis)[1] を用いるアプローチを提案する．

ソフトウェアアーキテクチャのフレームワークには様々なものがあるが，創発的特性をシステム思考的アプローチでモデル化して分析を行うことを念頭に，ビューポイントとパースペクティブを用いるフレームワーク [2] をベースにする．安全性のような創発的な特性の分析には，パース

[1] 長崎県立大学
University of Nagasaki, 1-1-1 Manabino, Nagayo-cho, Nishi-Sonogi-gun, Nagasaki 851-2195, JAPAN
[a)] kusakabe@sun.ac.jp

ペクティブを適用しながら，適切なビューポイントで記述や分析を行うことが有効と考える．

安全性といった横断的で創発的な特性に対するパースペクティブを適用しながら，コンテクストビューポイントから着手し，関連するビューポイントでのアーキテクチャ記述にSTAMP も用い，STPA での分析も行う．STAMP/STPA は以下のような特徴を持ち，このようなソフトウェアアーキテクチャのアプローチに適合すると考えている．

- STAMP/STPA はトップダウンな手法でライフサイクルの上流から適用可能．
- グラフィカル表現は多様なステークホルダが理解容易．
- STAMP/STPA は形式的な手法ではないが，STAMP/STPA の成果物やプロセスの一部は機械的な処理を前提とした形式な記法や手法との間の親和性が高いと期待でき，スケーラビリティや詳細化などの観点からも有望と考える．[*1]

本稿の第 2 節では，STAMP/STPA の概略を説明する．第 3 節で今回ベースとする，ビューポイントとパースペクティブを持つアーキテクチャフレームワークを，第 4 節では STAMP/STPA を用いたソフトウェアアーキテクチャ分析の手順を説明する．第 5 節でまとめと議論を行う．

2. STAMP/STPA

Nancy Leveson は，従来の解析的還元論や信頼性理論ではなくシステム理論に基づいたアクシデントモデル STAMP を提唱している．ソフトウェアが中心的な役割を果たし，またコンポーネント間での複雑な相互作用による創発的な特性を持つようなシステムのアクシデントは，システム構成要素の故障や人間の操作ミスに起因するイベントの連鎖といった観点でのモデル化は不十分としている．

STAMP では，コントロールする側のコンポーネント（以後コントローラプロセスと記す）から，コントロールを受ける側のコンポーネント（以後被コントロールプロセスと記す）へ，コントロールアクション (Control Action，以後 CA と記す) が発行され，被コントロールプロセスからコントローラプロセスにフィードバックが返される，というコントロール構造をベースにシステムをモデル化する (図 1 参照)．コントローラプロセスは自身が想定する被コントロールプロセスや環境のモデルを持ち (以後プロセスモデルと記す)，プロセスモデルにもとづいて CA に関する判断を行う．また，得られるフィードバックの結果はプロセスモデルの更新に利用する．STAMP モデルでは，コンポーネント間の相互作用において，CA が適切に与えられなかったり，不適切に与えられる，といったコントロールの問題によってアクシデントが起こると考える．

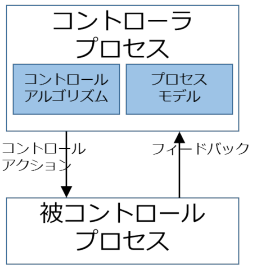

図 1　STAMP モデルの基本構成要素

STAMP 自身はアクシデントを説明するモデルであり分析法ではない．コントローラプロセスのプロセスモデルが被コントロールプロセスの実際の状況を正しく反映できてない，コントローラプロセスの判断方法が不適切，といったことに起因するコントロールの問題を，ハザード分析法 STPA を用いて分析する．STAMP にもとづく分析法 (思考ツール) として，ハザード分析法 STPA の他に，アクシデント分析手法 CAST (Causal Analysis based on STAMP)[1]，STPA-Sec (STPA for Security)[5]，STECA(System-Theoretic Early Concept Analysis)[6] といったものが提案されている (図 2 参照)．

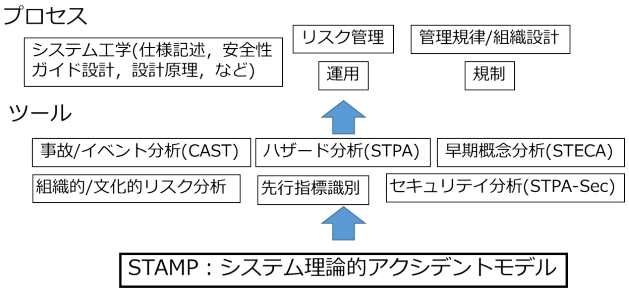

図 2　STAMP に基づく分析方法とその利用プロセス

STAMP/STPA は，FTA（Fault Tree Analysis）や FMEA（Failure Mode and Effect Analysis）といった従来の安全解析手法より (経験的に) 効果が高いとされる事例が増え，システムエンジニアリング分野を中心にその利用が広がっている[*2]．STAMP/STAP は安全工学の知見を幅広く適用可能にすることも念頭においており，ソフトウェア工学への適用に関しても既にモデル検査やテストを支援する研究などがある [7]．STAMP のモデリングはコンセプト段階からトップダウンに適用可能であり，システムエンジニアリングの早期アーキテクチャ検討にも用いられている [6]．このようなモデリングをソフトウェアアーキテクチャに用いれば，高い抽象度から詳細度を高めながらソフトウェア

[*1] その提唱者の Nancy Leveson 教授は形式手法 SpecTRM[3] の開発者でもあり，その研究室では STAMP/STPA と形式的な記述や分析の手法との関連づけも研究されている [4].

[*2] 例えば 2017 年 3 月に Nancy Leveson の研究室主催で MIT にて実施されたワークショップは，参加者は 300 名弱で前年の同様のワークショップと比べ約 12%増，参加者の主要な所属分野は約 40%が航空・軍関係，約 20%が自動車関係と発表されている．

アーキテクチャの記述と分析ができると考える．

STAMP/STPA の適用手順は厳密には定められておらず，いまだ研究の対象でもあるが，本稿では，以下のような，典型的とされている手順の一例に従って説明を行う．

- 準備 1 ： アクシデント，ハザード，安全制約の識別
- 準備 2 ： コントロール構造の構築
- Step1 ： 安全でないコントロールアクション（Unsafe Control Action，以後 UCA と記す）の識別
- Step2 ： Causal factor （誘発要因）の特定

ここで，アクシデントは受容し難い損失を伴うシステムの事象，ハザードはアクシデントにつながるシステムの状態，安全制約はシステムを安全に保つために必要なルールであり，システムが回避すべき事象を事前に設定することで目的に沿ったハザード分析を行うためのものである．STAMP/STPA は，慣習的なアクシデントに対するものだけでなく，多様な望まないイベントに対するハザードに対しても効果的に適用可能とされている．

3. ソフトウェアアーキテクチャのフレームワーク

ソフトウェアアーキテクチャの記法や分析法，そのベースにある考え方には様々なものがある [8][9][10]．ここでは，安全性のような創発的特性を STAMP/STPA で分析することを意識して，ビューポイントとパースペクティブを用いるロザンスキとオウェンのソフトウェアアーキテクチャフレームワークをベースに用いる [2]．

そのフレームワークでは，ビューポイントを「一つのタイプビューを構築するためのパターンやテンプレートおよび慣例を集めたもの．反映される関心事を持つステークホルダとビューを構築するための指針や原理及びテンプレートモデルを定義する」として，以下のようなビューポイントが提案されている．

- コンテクスト： システムとその環境 (システムが相互作用する人々，システム，外部エンティティ) の間にある関係と依存性および相互作用を記述する．
- 機能的： システム実行時の機能要素，その責務，インターフェースおよび主な相互作用を記述する．
- 情報： システムが情報を格納，操作，管理および配布する方法を記述する．
- 並行性： システムの並行性構造を記述し，機能要素を並行性の単位にマッピングし，システムの並行実行できる部分と，その調整および制御方法を明白に特定．
- 開発： ソフトウェア開発のプロセスをサポートするアーキテクチャを記述する．
- 配置： システムが配置される環境と，システムがその要素に対して持つ依存性を記述する．
- 運用： 本番環境での稼働時，システムがどう運用，管理およびサポートされるかを記述する．

ビューポイントに従ってソフトウェアアーキテクチャを記述するだけでは，横断的な関心事や非機能要求を明確にすることは難しい．ロザンスキとウッズは，横断的関心事や非機能要求に近いものをあえてそう呼ばず，パースペクティブとして次のように定義している：「システムの多数あるアーキテクチャビュー全体にわたって熟慮を要する，関連した品質特性の特定セットを，システムが提示することを保証するために用いられるアーキテクチャアクティビティや戦術，指針の集まり」．このようなパースペクティブをビューに適用することにより，洞察と改善につながり得るとしている．

以下のものが主要なパースペクティブとされている．

- セキュリティ
- パフォーマンスとスケーラビリティ
- 可用性とレジリエンス
- 発展性

また，二次的なものとして，アクセシビリティ，開発リソース，国際化，配置場所，規則，使用性が考えられるとしている．STAMP/STPA を安全性に関する記述と分析に用いる場合，対応するパースペクティブを追加して実践する必要がある．STAMP/STPA は，安全工学の知見を幅広く活用することも念頭に提唱されており，慣習的なアクシデントを念頭に安全性だけでなく，多様な望まないイベントに対するハザードに対しても効果的に適用可能とされている．STAMP アクシデントモデルの「アクシデント」を「受容できない損失の発生」ととらえて関連付けたパースペクティブを用いることができると考える．例えばセキュリティのような安全性以外のパースペクティブに対応した分析が可能と考えられる [5]．

4. STAMP/STPA を用いた分析

前述のようなパースペクティブとビューポイントをベースに STAMP/STPA の手順を実施する．STAMP のモデルには，安全制約，階層的なコントロール構造，プロセスモデルという要素が含まれる．分析ではコントロール構造とプロセスモデルに対して，システムの安全制約が正しく適用されているかどうかに着目する．

4.1 準備 1：アクシデント，ハザード，安全制約の識別

最初のステップとして，アクシデント，ハザード，安全制約の 3 つを決める．

- アクシデント： 望んでもいないし計画もしていない，損失につながるようなイベントをアクシデントとする．人命喪失，けが，物損，環境汚染，ミッション喪失，経済的損失などが損失に相当する．STAMP/STPA では安全工学の技法を出来るだけ広く適用する意図で，

アクシデントも広めの定義となっている．

- ハザード： 環境のある最悪な条件と重なることでアクシデントにつながるような，システムの状態の集合もしくは条件をハザードとする．これは，安全性と信頼性の混同を回避するために単なる故障 (failure) とハザードを区別し，分析可能性も高めるといった意図の下，以下のような二つの点を念頭においた定義である．第一に，ハザードがコントロール対象のシステムの境界内のものであり．コントロール対象のシステムの範囲，コントロール範囲外の環境との境界が明確になっている必要がある．第二に，ハザードが実際に損失につながるのは，ハザードと組み合わさる，最悪の環境条件の存在が必要である．
- 安全制約： ハザードが識別されると，それらからシステムを安全に保つための要件もしくは制約を導ける．トップダウンに考える場合，まず高レベルの安全制約が導かれることになる．安全制約はこの段階ですべて確定するとは限らず，後の STPA の手順の実施によっても導出，修正されることもある．

このような STAMP/STPA の準備ステップは，主にコンテクストビューポイントと関連が深く，記述や分析の際，図 3 のような社会技術システムの階層の中での位置づけや関連する相互作用を考慮するとよいと考えられる．

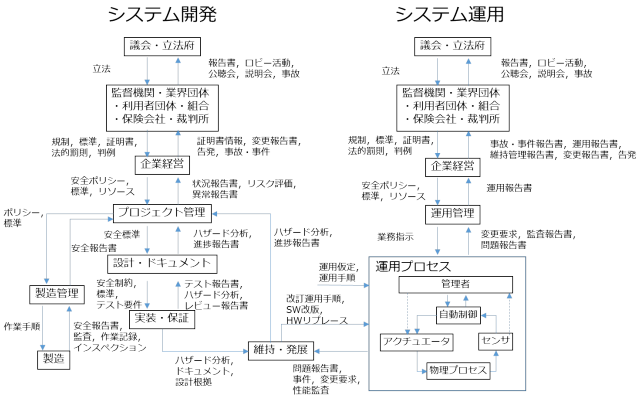

図 3 社会技術システムのモデル例 (書籍 [1] 図 4.4 をもとに作成)

4.2 準備 2：コントロール構造の構築

CA に焦点を当てたコントロール構造を構築する．典型的には，機能の実現の観点からの抽象的なコントロール構造を起点にすることが推奨されている．機能的ビューポイントに関して，UML といった慣習的なものに加え，対応する STAMP での成果物が加わる．システムのコンポーネントに機能を割り当てるといったことは一般的な活動でも行われるが，ドキュメントとしてコントロール構造を使うことは一般に行われていない．また，準備 1 で定めた，アクシデントやハザード，安全制約によっては，他のビューポイントの成果物も加わる．

アクシデント，ハザード，安全制約を考えるのはアクシデントのモデルやハザードの解析法によらず一般的な手順であるが，コントロール構造を構築して分析するのは STAMP/STPA の特徴的な点である．詳細な設計の記述やドキュメントがあるにしても，機能的な振る舞いを把握するための情報はまとまっていなかったり，理解することが難しいことも多い．適切なパースペクティブに従った STAMP のコントロール構造は，システムの機能的な設計をうまくグラフィカルにモデルとして表現でき，優れたソフトウェアアーキテクチャのドキュメントとして有用と考える．これまでの知見では，コントロール構造を作成したり，理解したりするには，まず簡潔な高レベルのものから始め，段階的に詳細化したり追加したりするのがよいとされている．最初の段階では，コントロール対象のプロセスとコントローラだけ，場合によっては自動化や人がかかわるコントローラーの階層が若干加わった程度のコントロール構造からトップダウンに始めるのが典型的である．[*3]

4.3 Step1：非安全なコントロールアクションの識別

STAMP では，安全に関するコントロール構造がシステムの安全制約を守れず，以下のような原因でハザードが発生しアクシデントにつながると考える．

- 対処されない環境外乱や環境条件
- 対処されなかったりコントロールされなかったりするコンポーネントの障害
- コンポーネント間の非安全な相互作用
- 適切に協調されてない複数コントローラによる CA

STAMP モデルの以下の 3 つの要素を用い，コントロールを行う側とその対象プロセスとの間の相互作用において，安全制約が不適切であったり，守られない状態になってしまうシナリオを中心に分析する．

- 構成要素間の相互作用を表すコントロール構造
- コントロールする側がその対象プロセスをコントロールするアルゴリズムとプロセスモデル
- 安全制約

分析のステップは，典型的には，UCA の識別と，UCA の原因の識別のステップに分けることが多い．必ずしもこのような分け方に従う必要はないが，ここではその分け方に従った場合の最初のステップの説明を行う．

コントローラからの制御対象のプロセスに対するアクションのうち，以下の 4 つのタイプの UCA の識別を行う．

- 与えられないとハザード (Not-Providing causes haz-

[*3] 例えば，コントロールケース [11] を拡張した非機能要求記述コントロールケースを用いた記述実験では，最初に簡単に書いたモデルから詳細度を高めていく方法が便利というアンケート結果が得られているが [12]，STAMP/STPA ではそのような方法が典型的なものとなる．

表 1　UCA 分析表フォーマット例

UCA	Providing	Not Providing	Timing	Duration
CA 名	…	…	…	…
…	…	…	…	…

ard)：安全のために必要とされる CA が与えられないことでハザードに至る．

- 与えられるとハザード (Providing causes hazard)：ハザードに至る UCA が与えられる．
- 早過ぎ, 遅過ぎ, 誤順序でハザード (Timing - Too early/too late, wrong order causes hazard)：安全のためのものであり得る CA が，遅すぎて，早すぎて，または順序通りに与えられないことでハザードに至る．
- 早過ぎる停止, 長過ぎる適用でハザード (Duration - Stopping too soon/applying too long causes hazard)：(連続的，または非離散的な CA において) 安全のための CA の停止が早すぎる，もしくは適用が長すぎることでハザードに至る．

ハザードに至る 5 番目のタイプとして，必要な CA が与えられたけれども有効に実行されない，という場合もあり，この 5 番目の可能性については次のステップで分析する．

分析に使うフォーマットに決まりはないが，UCA のドキュメント化のためには，表を用いるのが便利とされている．

UCA は，安全のために必要なルールと関連付けて考えることができるため，UCA の分析から安全制約を作り出すことも可能である．このため，安全制約を最初の準備の段階でなくここで作成したり，作成済みの安全制約を UCA の分析結果をもとに見直したりすることもできる．

4.4　Step2：非安全なコントロールの原因の識別

UCA を識別したのち，非安全なコントロール (につながるシナリオ) の原因を識別する．必ずしもこれらのステップを完全に別のものとして逐次的に行う必要はなく，一部の UCA を識別した段階で原因の分析を行う場合もありえる．このステップで，前述の 5 番目のタイプのシナリオである，安全のために必要なコントロールアクションの不適切な実行を考慮する．このステップは，分析者の経験と考察を最も必要とするため，事前に設定できるガイドラインは少ない．しかしながら，可能性のあるハザードの原因を取り除いたり，緩和したりするなどして安全制約を保つために必要な情報を分析する重要なステップである．基本的に，安全制約を保つためのコントロールループやその構成要素を確認し，以下のような点を分析する．プロセスモデルが正しくなくなってしまう原因も含め，どのようにして非安全なコントロール，ひいてはハザードにつながるかを分析する．必要な CA が与えられたにもかかわらず，適切に実行されないのかの原因も分析する．

- コントロール入力や外部情報の誤りや喪失

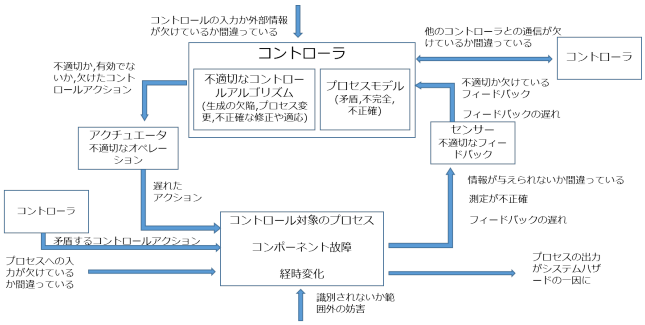

図 4　誘発要因の例

- 不適切なコントロールアルゴリズム（作成時の欠陥，プロセスの変更，誤った修正や適用）
- 不整合，不完全，または不正確なプロセスモデル．不適切な操作．
- コンポーネントの不具合．経年による変化．
- 不適切なフィードバック，あるいはフィードバックの喪失．フィードバックの遅れ．
- 不正確な情報の供給，または情報の欠如．測定の不正確性．フィードバックの遅れ．
- 操作の遅れ．
- 不適切または無効な CA，CA の喪失．
- CA の衝突．プロセス入力の喪失または誤り．
- 未確認，または範囲外の障害．
- システムにハザードを引き起こすプロセス出力

分析によって得られたシナリオは，システムを保護するためのコントロールを識別するために用いられる．通常は，このような保護のためのコントロールの設計は STPA の解析そのものではなく，解析結果を用いてドメインの専門家によってなされる．

5.　まとめと議論

ロザンスキとウッズのフレームワークのサンプルをベースにした，ソフトウェアアーキテクチャ記述の構成例は以下のようになる．システム品質に安全性が加わり，そのための記述としてアーキテクチャビューの記述に STAMP のモデルが記述される．

- 導入
  - 目的とスコープ，対象者，ステータス，アーキテクチャ設計アプローチ
- 用語集
- システムのステークホルダと要求
  - ステークホルダ，要求の概要，システムシナリオ
- アーキテクチャフォース
  - ゴール，制約，アーキテクチャ原理
- アーキテクチャビュー
  - コンテクストビュー，機能的ビュー，情報ビュー，並行性ビュー，配置ビュー，開発ビュー，運用ビュー

- システム品質
  - 安全性，性能とスケーラビリティ，セキュリティ，可用性とレジリエンス，発展性，その他の品質
- 付録
  - 判断と他の選択肢，QA，リファレンス

このようなライフサイクル初期の，多様なステークホルダが関与するソフトウェアアーキテクチャの記述は，人にとっての了解性といった観点の重要度が高く，機械的で厳密な処理が前提の中間成果物や実装レベルの成果物の記述との間のギャップが大きいことも多い．そのようなギャップを埋める，ソフトウェアアーキテクチャと形式手法を関連付けた取り組みもなされている [13][14][15][16].

STAMP は機械的な実行やシミュレーションを想定したものでなく，本アプローチは特にライフサイクル初期のシナリオ分析 [17][18] に適していると考える．一方，STAMP/STPA の詳細化が進むとその過程や成果物に関する機械的支援の有用度も増すと考えらる．例えば，AADL に関連したものとして，STPA の分析結果を AADL モデルに統合し分析報告書を自動作成したり，Error Model Annex を用いて分析の一部を支援する例もある [19][20].

著者らも，形式的なモデル記述の前段階の活動として，STAMP/STPA 分析を行うアプローチに取り組んでいる [21]．そのような STAMP/STPA の活用においても，ソフトウェアエンジニアリングで確立したプラクティスと系統的に関連付けて実践する，このような STAMP/STPA を用いたソフトウェアアーキテクチャ分析は有用と考える．

6. おわりに

本稿では，システム理論に基づく STAMP/STPA を用いたソフトウェアアーキテクチャ分析を提案した．STAMP のモデリングの記法を用いることで，創発性を持つソフトウェアシステムのアーキテクチャの記述と分析を行うという観点からこのようなアプローチは有用と考える．また，ソフトウェア工学で確立しているソフトウェアアーキテクチャのプラクティスと系統的に関連付けて STAMP/STPA を効果的に実践するためのガイドとなることも目指した．今後は公開可能な適用実績を増やしながらより効果的なガイドの開発を目指す予定である．

参考文献

[1] N. G. Leveson. *Engineering a safer world: Systems thinking applied to safety*. MIT press, 2011.
[2] ニック・ロザンスキ, オウェン・ウッズ, 監訳：榊原彰, 訳：牧野祐子. ソフトウェアシステムアーキテクチャ構築の原理 第二版 (Software Systems Architecture). SB Creative, 2014.
[3] N. G. Leveson, M. P. E. Heimdahl, and J. D. Reese. *Designing Specification Languages for Process Control Systems: Lessons Learned and Steps to the Future?*, pp. 127–146. Springer, Berlin, Heidelberg, 1999.
[4] J. Thomas. *Extending and Automating a Systems-Theoretic Hazard Analysis for Requirements Generation and Analysis*. PhD thesis, MIT, 2013.
[5] W. Young and N. G. Leveson. An integrated approach to safety and security based on systems theory. *Communications of the ACM*, 57(2):31–35, 2014.
[6] C. H. Fleming. *Safety-driven Early Concept Analysis and Development*. PhD thesis, MIT, 2015.
[7] A. Abdulkhaleq, S. Wagner, and N. Leveson. A comprehensive safety engineering approach for software-intensive systems based on {STPA}. *Procedia Engineering*, 128:2 – 11, 2015.
[8] P. Kruchten. Architectural blueprints?the “4+ 1” view model of software architecture. *Tutorial Proceedings of Tri-Ada*, 95:540–555, 1995.
[9] R. T. Monroe, A. Kompanek, R. Melton, and D. Garlan. Architectural styles, design patterns, and objects. *IEEE Softw.*, 14(1):43–52, Jan. 1997.
[10] 沢田篤史, 野呂昌満. ソフトウェアアーキテクチャの設計と文書化の技術 (特集 サーベイ論文). コンピュータソフトウェア, 32(1):35–46, feb 2015.
[11] J. Zou and C. J. Pavlovski. Modeling architectural non functional requirements: From use case to control case. In *2006 IEEE International Conference on e-Business Engineering (ICEBE'06)*, pp. 315–322, Oct 2006.
[12] 非機能要求記述ガイド. 経済産業省　ソフトウェア開発力強化推進タスクフォース　要求工学・設計開発技術研究部会 非機能要求とアーキテクチャ WG, 2008.
[13] R. Allen and D. Garlan. Formalizing architectural connection. In *Proceedings of the 16th International Conference on Software Engineering*, ICSE '94, pp. 71–80, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.
[14] J. Magee, J. Kramer, and D. Giannakopoulou. Software architecture directed behaviour analysis. In *Proceedings Ninth International Workshop on Software Specification and Design*, pp. 144–146, Apr 1998.
[15] 福澤公之, 佐伯元司他. 4+ 1 ビューモデルにもとづくソフトウェアアーキテクチャの記述およびその評価法. 情報処理学会研究報告ソフトウェア工学 (SE), 2002(23 (2001-SE-136)):111–118, 2002.
[16] 岸知二, 野田夏子他. モデル検査によるアーキテクチャ設計検証. 情報処理学会研究報告ソフトウェア工学 (SE), 2005(119 (2005-SE-150)):9–16, 2005.
[17] L. Dobrica and E. Niemela. A survey on software architecture analysis methods. *IEEE Transactions on software Engineering*, 28(7):638–653, 2002.
[18] M. A. Babar and I. Gorton. Comparison of scenario-based software architecture evaluation methods. In *Software Engineering Conference, 2004. 11th Asia-Pacific*, pp. 600–607. IEEE, 2004.
[19] S. Procter and J. Hatcliff. An architecturally-integrated, systems-based hazard analysis for medical applications. In *Formal Methods and Models for Codesign (MEMOCODE), 2014 Twelfth ACM/IEEE International Conference on*, pp. 124–133. IEEE, 2014.
[20] 岡本圭史. Aadl を用いた stamp/stpa 支援. ソフトウェア・シンポジウム, 6 月, 2017.
[21] 日下部茂. Pre-formal メソッドとしての stamp モデリング. 第 13 回クリティカルソフトウェアワークショップ, 1 月, 2016.