6. フォーマルメソッドと関連技術の位置づけ

本章では、ソフトウェアに対する要求とそれを満たすために使われる技術の全体像をまとめ、その中でフォーマルメソッドの位置づけ、用途、制約、他の技術との関係などについて整理する。

対象読者	(1)ベンダー上級管理者 (2)開発プロジェクト管理者 (3)開発技術者、等
目的	ソフトウェアに対する要求としての機能要求、非機能要求、ディペンダビリティ等の関係や全体像について整理し、要求が満たされていることを確認するために用いられる技術の全体像とその中におけるフォーマルメソッドの役割、特徴、他の技術との関係を把握することで、フォーマルメソッドの位置づけの適切な理解を促進することを目的とする。
想定知識	ソフトウェア開発に関する基礎
得られる知見等	● ソフトウェアに対する要求(機能性、安全性、信頼性、ユーザビリティ、ディペンダビリティ等)とそれを満たすための技術の全体像 ● 対策技術の全体におけるフォーマルメソッドの位置づけ ● 開発プロセス(V 字モデルなど)全体におけるフォーマルメソッドの適用箇所、用途 ● フォーマルメソッドに適した用途、フォーマルメソッドの制約や課題および他の技術との組合せの必要性

ソフトウェアに対する要求が満たされることを保証するためには、ソフトウェアを含むシステム全体で議論しなければならない[88]。そのようなシステム全体におけるフォーマルメソッドの位置付けや用途について把握することは重要である。それらについて把握し、フォーマルメソッドと他の技術の組合せ利用について以下のようなことを検討する際の参考にするとよい。

(1) 満たすべき要求の全体像を把握し、そのうちのどこにフォーマルメソッドが適しているか把握する。
(2) 要件を満たすために利用される技術とフォーマルメソッドの関係を把握する。
(3) フォーマルメソッドの適用箇所、およびその他の技術の補完的適用を検討する。

[88] Safeware: System Safety and Computers, Nancy G. Leveson, Addison-Wesley Professional, 1995

6.1. ソフトウェアに対する要求の把握

6.1.1. 機能要求と非機能要求

ソフトウェアに対する要求は、大きく以下の2つに分けられる[89]。

表 6-1:ソフトウェアに対する要求

	内容
機能要求	ユーザがソフトウェアに対して求める中心的な機能で、ある入力に対して、出力(画面の表示変更等を含む。)を伴って提供されるもの。
非機能要求	機能要求に対する特性(制約や品質等)に関する要求で、具体的には、信頼性、安全性、性能、操作性、運用性などが含まれる。

機能要求は、ユーザが求める機能であるため、ソフトウェア開発の初期段階から比較的よく検討される事項であるが、非機能要求の中には性能、利用性、セキュリティなどユーザが初期段階で、把握しにくいものもあるため、上流工程で十分な検討がなされないことが多い。人命に関わるようなセーフティクリティカル・システムや、基幹業務に関わるミッションクリティカル・システムなどにおいては、機能要求はもとより、非機能要求に対して高い水準が求められる。安全性等の非機能要求は、ソフトウェア・ベンダー側のステークホルダーよりも、ドメインにおける過去の経験や失敗情報を把握している事業者やユーザの方がより適切に指摘できる場合がある。「情報システムの信頼性向上に関するガイドライン」では、「非機能要求」を明確に定義することが情報システムの信頼性を向上する上で重要であると指摘している。

ソフトウェアに対する要求については、ソフトウェアの要求仕様や品質特性に関する国際標準[90]等の様々な整理がなされている。

表 6-2:ISO/IEC 9126[91] (ソフトウェアの品質特性)

要求	説明	特性
機能性 (functionality)	明示的及び暗示的必要性に合致する機能を提供する	合目的性 正確性 相互運用性
信頼性[92]	指定された達成水準を維持する特	成熟性

[89] IEEE Std 830 Software Requirement Specification Template

[90] ISO/IEC 9126 Software engineering — Product quality

[91] ソフトウェアの品質に関する標準 ISO/IEC 25000:2005 Software Engineering -- Software product Quality Requirements and Evaluation (SQuaRE) -- Guide to SQuaRE に統合される。

[92] 本ガイダンスでは、ソフトウェアの信頼性という用語を、ソフトウェア品質に関する国際標準 ISO9126, ISO25000 における信頼性や、Nancy Leveson のセーフウェアにおける信頼性の考え方に基づくものとして用いる。つまり、ある状況においてソフトウェアがどの程度機能するかに関する特性で、ソフトウェアが要求仕様に遵守している程度や、設計や実装に不具合が無い度合いを指すものとする。一方、通常、ハードウェアを含むシステムにおける信頼性は、与えられた期間、想定された条件の下で、求められた機能を実行する確率によって表される特性で、ハードウェアの故障が原因となる障害も含むもので、より広い概念である。

(reliability)	性	障害許容性 回復性
ユーザビリティ (usability)	理解、習得、利用でき、利用者にとって魅力ある特性	理解性 習得性 運用性 魅力性
効率性 (efficiency)	使用する資源の量に対して適切な性能を提供する特性	時間効率性 資源効率性
保守性 (maintainability)	保守(変更)作業のし易さに関する特性	分析容易性 変更容易性 安定性 試験容易性
移植性 (portability)	保守(変更)作業のし易さに関する特性	環境適応性 設置性 共存性

ソフトウェアの品質に関しては、表 6-1 のソフトウェア品質特性に関する国際標準(ISO/IEC9126)が参考になる。従来、ソフトウェアの品質は、不具合がないことだけに偏っていたが、ISO/IEC9126 により、信頼性やユーザビリティなど幅広い特性について認識が広まってきた。

非機能要求の要素については様々な提案がなされ、統一的な分類には至っていない[93]が、Kotonya と Sonnmerville(KS)[94]の非機能要求[95]を例にとると以下のような要求を含むものとしている。

表 6-3:非機能要求の主な要素とその内容

非機能要求の要素	内容
安全性	ユーザや環境に、破壊的な影響を与えないこと。
セキュリティ	意図的な攻撃に対する耐性があること。
信頼性	意図した通りサービスを提供すること。
ユーザビリティ	美しさ、一貫性、単純性、文書の整備など
性能要求	応答時間、スピード、スループット、リソース使用率、効率性など。

フォーマルメソッドは、技術的な制約などにより、上記の全ての要素に対して適用できるわけで

[93] 非機能要求、山本修一郎、Business Communication, 2006 年 9 月
[94] Gerald Kotonya and Ian Sonnmerville, Requirements Engineering, John Wiley & Sons, 2002
[95] Requirements Engineering : Processes and Techniques, Gerald Kotonya and Ian Sommerville., John Wiley 1998.

はない。そのため、フォーマルメソッドの適性を把握して、適した対象を選ぶことにより効果が得られると期待できる。具体的には、ソフトウェアの品質特性における機能性、信頼性や、非機能要求の主な要素のうち、安全性、セキュリティ、信頼性などがフォーマルメソッドの対象として有望と言える[66,111]。一方、フォーマルメソッドは、ユーザビリティ、性能、効率性、保守性、移植性に関しては、取り扱いが難しい。第 7 章や付録の応用事例集等を参考に、成功事例の多い対象を参考にして、成功確度の高いものから試行するとよい。

6.1.2. ディペンダビリティの要素と対策

セーフティクリティカル・システムやミッションクリティカル・システムにおいては、高い信頼性や安全性が求められる。ソフトウェアを含むシステム全体の観点では、信頼性や安全性を含む広義の概念として、Jean-Claude Laprie などの提唱により[96]、「ディペンダビリティ(dependability)」[97],[98],[99]という言葉が使われるようになっている。A. Aviizienisらは、システムのディペンダビリティを以下の要素を含む広い概念として整理している[100][101]:

表 6-4:ディペンダビリティの構成要素

要素	説明
信頼性(Reliability)	意図した通りサービスを提供すること。
安全性(Safety)	ユーザや環境に、破壊的な影響を与えないこと。
可用性(Availability)	サービスを適切な時に提供できる能力。
完全性(Integrity)	システムに対する不適切な変更が無いこと。
保守性(Maintainability)	変更や修正を行う能力。
機密性(Confidentiality)	権限の無い者への情報開示が無いこと。

ディペンダビリティのうち、機密性、完全性、可用性のことをセキュリティの要素としても捉えられる。

ディペンダビリティを阻害する原因は脅威と呼ばれ、以下のように分類される[102],[103]:

---

[96] J.C. Laprie. "Dependable Computing and Fault Tolerance: Concepts and terminology," in Proc. 15th IEEE Int. Symp. on Fault-Tolerant Computing, 1985

[97] ディペンダビリティの概念は、(狭義の)信頼性(reliability)を要素として含んでいるため、「信頼性」という言葉と区別して「ディペンダビリティ」と呼ぶ。

[98] これらの要素は、排他的ではなく互いに重なりを持っている。これらの要素のうち、信頼性と可用性は、定量的に計測可能だが、その他の要素はより主観的なものである。

[99]国内規格 JIS においても、ディペンダビリティを、availability(可用性)、maintanability(保守性)、integrity (保全性)、 confidentiality or security(機密性)、safety(安全性)などを包含する概念と定義している。

[100] A. Avizienis, V. Magnus U, J. C. Laprie, and B. Randell, "Fundamental Concepts of Dependability," presented at ISW-2000, Cambridge, MA, 2000.

[101] Basic concepts and taxonomy of dependable and secure computing,Avizienis, A. et al, Dependable and Secure Computing, IEEE Transactions on Issue Date: Jan.-March 2004, Vol.1 Issue:1, pp.11 - 33, 2004

[102] Basic concepts and taxonomy of dependable and secure computing,Avizienis, A. et al, Dependable

表 6-5:ディペンダビリティに対する脅威の分類

	分類	説明
脅威	エラー/誤り(error)	欠陥の原因となる(人間の)知識や行為のこと。行為が正しくても知識が間違っていれば、欠陥が生じる。正しい知識に基づいていても、行為が誤っていれば欠陥が生じる
	欠陥/バグ(fault)	エラーが原因で、プログラムや設計などの記述に埋め込まれた間違い。
	障害/故障(failure)	欠陥に基づいて発生した望ましくない事象。

情報セキュリティの分野では、脆弱性という用語が使われる。これは、仕様には明記されず、出荷時には明確になっていないもので、障害に至る原因となるもので、欠陥とは区別される。

さらにディペンダビリティの確保に関する対策は以下のように分類される 。

表 6-6:ディペンダビリティに関する対策の分類

対策の分類	内容
欠陥予防 (Fault prevention)	欠陥が入り込まないための対策。仕様が正しく記述され、正しく実装されるための対策。
欠陥耐性 (Fault tolerance)	障害の発生を前提として、傷害が具現したときに、事故に至らないような対策。
欠陥除去 (Fault Removal)	欠陥がすでに含まれている時に、それらを検出し、除去する対策。
欠陥予測 (Fault forecasting)	欠陥の除去や回避を行うために、欠陥の存在の可能性を予測する対策。

ソフトウェアのディペンダビリティを確保するためには、人間の誤りによるバグの混入をいかに少なくするかという問題がある。テストによって欠陥/バグを見つけられるのは、テストケースにより障害が顕在化した場合のみであり、すべての欠陥を見つけられるものではない。

ディペンダビリティ対策の分類ごとの具体的な手法の例を挙げると以下のようになる。

表 6-7:ディペンダビリティ対策の手法例

対策の分類	手法の例
欠陥予防	● 要求仕様書の作成 ● オブジェクト指向設計プログラミング手法

and Secure Computing, IEEE Transactions on Issue Date: Jan.-March 2004, Vol.1 Issue:1, pp.11 - 33, 2004

[103] J.C. Laprie. "Dependable Computing and Fault Tolerance: Concepts and terminology," in Proc. 15th IEEE Int. Symp. on Fault-Tolerant Computing

	● デザインパターン ● モデルベース開発[104] ● フォーマルメソッド ● 要求管理・インパクト解析
欠陥耐性	● エラー処理 ● 多重化 ● エラー処理を設計(アーキテクチャ記述言語 AADL 等)
欠陥除去	● デバッガー ● 脆弱性チェッカー[105] ● テスト(ブラックボックス、ホワイトボックス) ● コードレビュー、コード・インスペクション ● フォーマルメソッド(不具合解析)
欠陥予測	● ハザード・リスク分析[106] ➢ FMEA[107]:構成部品の故障モードを元にボトムアップで、影響や対策を分析(原因から結果) ➢ FTA[108]:問題となる事象の要因をトップダウンに分析する。(結果から原因にさかのぼる) ➢ HAZOP[109]:システムの持つ危険事象を見つけ出し. 致命度(Seriousness)の程度に従って区別する ● 信頼度成長曲線[110]

ディペンダビリティの対策分類とソフトウェアの開発工程ごとに、実際に利用される技術を整理すると図 6-1 のようになる[111],[112],[113]。

[104] モデルベース開発:ソフトウェア開発においては、仕様をシミュレーションや検証可能なモデルで表現し、各工程内でモデルのシミュレーション等による検証と修正を繰返し構成する開発手法。
[105] 脆弱性チェッカー:ソフトウェアのバッファーオーバーフローなどネットワーク攻撃等に関する不具合を検出するツール。
[106] ハザード・リスク分析:事故の原因とその影響の関係について分析する手法。
[107] FMEA:設計の不完全や潜在的な欠点を見出すために構成要素の故障モードとその上位要素への影響を解析する技法。
[108] FTA:事故などの事象について、発生経路、発生原因及び発生確率を原因の関係を木を用いて解析する手法。
[109] HAZOP:プロセスの目標値からのずれを想定し、そのずれの起こる原因と発生する危険事象を解析し、さらにその原因から危険事象に進展するのを防護する機能を評価し、対策を検討する技術。
[110] 信頼度成長曲線:テスト工程においてテストケース数に応じて発見されたバグの累積件数をグラフ化したもの。
[111] Safeware: System Safety and Computer, Nancy Leveson, Addison Wesley
[112] EASIS D3.2 Part1: Guidelines for establishing dependability requirements and performing harzard analsis
[113] Jackson, D. et al, Software for dependable systems. sufficient evidence?, NATIONAL RESEARCH COUNCIL, 2008

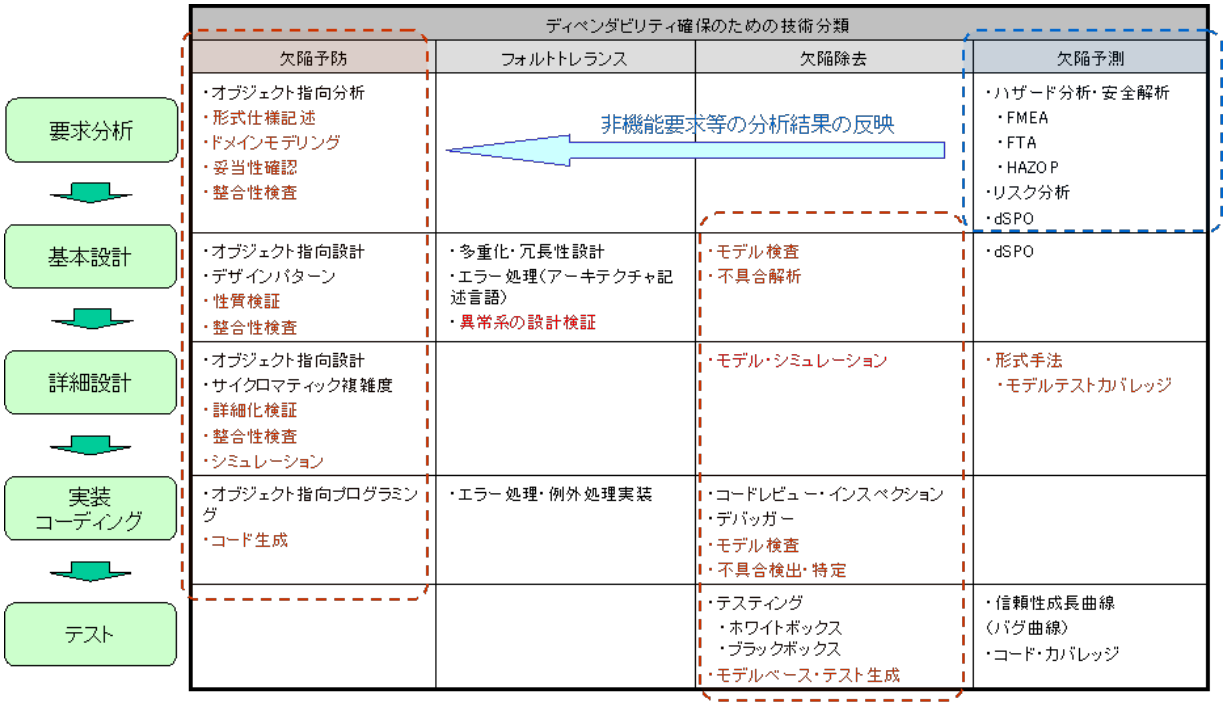

図 6-1:ディペンダビリティ確保の対策分類と利用される技術の関係

フォーマルメソッドは、欠陥予防、欠陥除去を中心に利用され、形式仕様記述による曖昧性の除去、仕様検証、不具合検出などに有効である。一方、図を見るとディペンダビリティを確保するためには、フォーマルメソッド以外の手法が必要な領域があることも示している。不具合から生じる障害や、障害の原因となる不具合などを解析するハザード・リスク分析(FTA, FMEA, HAZOP 等)は、高い安全性・信頼性を求められるシステムにとっては不可欠である。また、従来のソフトウェア・テストは、不具合が無いことを示すためには有効ではないが、コストの面で、フォーマルメソッドで全て置き換えられるわけではない。これらの技術を併用することで、目的とする要求を満たすことが現実的なアプローチと考えられる。

6.1.3. 信頼性・安全性・情報セキュリティの関係

従来から、ディペンダビリティの要素のうち、(狭義の)信頼性、安全性に対する要求は高いものであると認識されてきた。情報システムがネットワークに接続されることが一般的になり外部からの脅威に対して防御が求められるようになった今日では、信頼性、安全性に加えて、情報セキュリティに対する要求が高まっている。

信頼性、安全性、情報セキュリティの関係は、下図に示すような交わりを持つが必ずしも一致しない[114]。

[114] 文献 111 をもとに、情報セキュリティの関係も加えて作成。

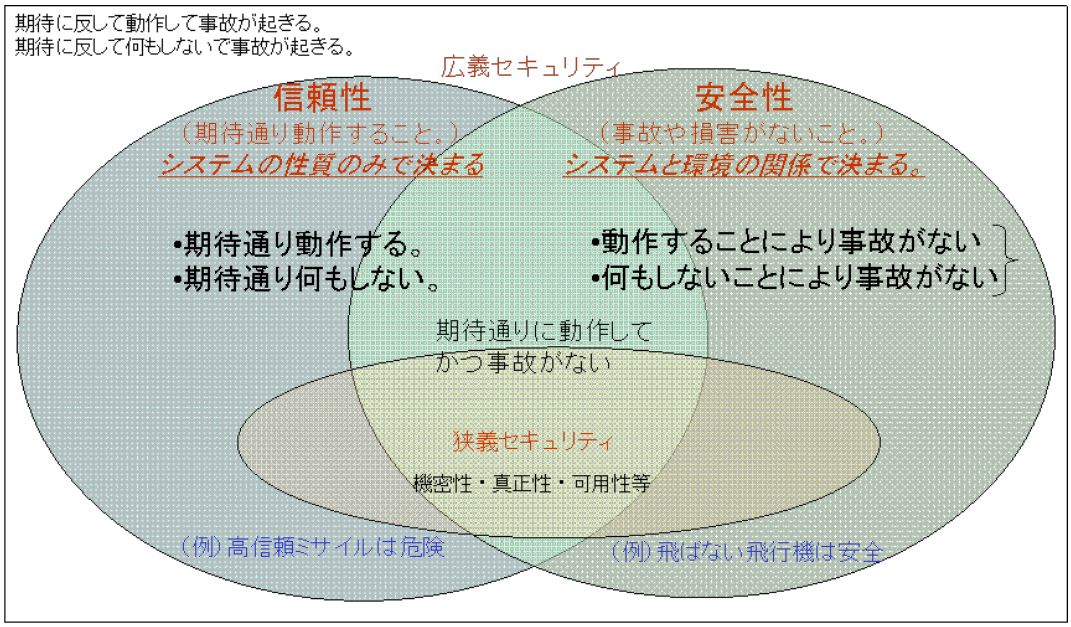

図 6-2:信頼性・安全性・セキュリティの関係

（狭義の）信頼性は、システムが仕様通りに動作することを意味し、安全性は、被害や損害を生じる事象に至らないことを意味する。信頼性と安全性は、互いに関係性を持つが、信頼性が高いシステムは、必ずしも安全とは限らない。実際、多くのセーフティクリティカル・システムでは、仕様通りに正しく動作している時に事故が発生している 。逆に、仕様通りに機能を果たさない、何もしないシステムは、常に安全であるが、目的を達成しないシステムは意味がない。システムが本来期待される機能を提供するという（信頼性が確保されたという）前提の下で、初めて安全性を議論することに意味がある。一方、安全性は、システム単独で議論することはできず、システムと外部環境の関係において議論しなければならない。要求分析において、システム障害時の想定被害やその影響範囲なども分析して、要求に反映させる必要がある。

情報セキュリティは、信頼性と安全性と関係する部分もあるが、狭義の情報セキュリティは、意図的な攻撃による機密性、可用性、真正性が失われるようなことに対する耐性を指す。意図的な攻撃は、情報システムの企画・設計段階で網羅的に把握することが難しいため、要求分析の困難さがある。一般的な設計プロセスにおいては、正しい使い方を前提としたユーザであるアクターに基づきユースケースを用いた要求分析が行われてきたが、セキュリティを確保するためには、従来のユースケースに、攻撃者の視点で行動する攻撃アクターを加えたミスユースケースを作成し、セキュリティ要求を十分に洗い出すことが重要になっている。通常、システムの完全なセキュリティを保証することは経済性の面で現実的ではなく、トレードオフを考慮した脅威と攻撃のリスクに対するセキュリティ保証技術が必要である[115]。

[115] Bashar Nuseibeh, Valuing Security: on risky requirements and expensive design, Software Security Symposium 2007

6.2. ソフトウェアの品質に関する対策技術とフォーマルメソッドの位置づけ

6.2.1. フォーマルメソッドの用途と特徴

ソフトウェア・テストでは、不具合の存在を発見することができても、不具合が存在し無いことを保証することができない[116]。つまり、いくらテストをしても、ソフトウェアの正しさを保証することができない。一方、フォーマルメソッドは、記述した性質に関して、形式検証に成功すれば、システム(設計や実装)が、与えられた性質を満たすことを保証できる点が、テストとの大きな違いである。

プログラムは、ある機能をどのように処理するか記述しなければならない。一方、フォーマルメソッドは、システムが機能を「どのように実現するか」(How)を記述することなく、「何を実現するか」(What)を記述することができる点に特徴がある。テストは、プログラムの実行を前提とするため、以下のような問題がある:

- プログラムが実装された後でなければ検査できない。
- 実行に依存するため、並列システムの動作タイミングなど再現性に依存する性質は、完全には検査できない。

フォーマルメソッドは、実現方法を記述することなく、何を実現するか記述でき、それらに対して検証も可能であるため、テストと比べて、要求分析・設計等の開発上流プロセスにおいて適用することが可能であり、早期に不具合を発見することで、開発手戻りを押さえる効果がある。安全性に関わるエラーの多くは、要求分析に原因であり[117]、要求仕様のレベルで検証を行えるフォーマルメソッドは、コスト低減の効果が大きいといえる。ただし、フォーマルメソッドは、特定の数学モデルに基づく制約の強い言語で記述するため、記述の自由が高いプログラムに比べて、記述の工数が大きくなることがある。したがって、フォーマルメソッドは、上流の設計工程で適用する方が高い費用対効果が得られる。

フォーマルメソッドの主な適用箇所と用途は図 6-3 のようになる[118]。

---

[116] Edsger Wybe Dijkstra による言葉。

[117] Abernethy, K. et al, Technology Transfer Issues for Formal Methods of Software Specification, Proceedings. 13th Conference on Software Engineering Education & Training, 2000, 6-8 March 2000, pp.23 - 31

[118] ソフトウェア開発の分野では、中間成果物に対して様々な用語が用いられる。「RFP(提案依頼書)」「要件定義書」「要求仕様書」は、ユーザがどんなシステムを開発したいのかを記述したもであり、立場や目的により異なる。RFP は、発注者がベンダーを選定するためのものであり、要件定義書は、システム開発プロジェクトにおける要件分析フェーズのアウトプットなどである。ソフトウェア工学の分野では、これらは Requirement(要求)という用語が用いられる。本ガイダンスでは、明確に仕様化されていないユーザ等のニーズを「要求」と呼び、仕様化されたものを「要求仕様」と呼ぶ。また、実現方法を仕様化したものを「設計仕様」(基本設計、詳細設計など、詳細度に応じて複数のレベルが想定される。)。また、「仕様」とは、異なるステークホルダー間の合意文書である。「要求仕様」は、発注者とベンダーの間の合意文書、「設計仕様」は、設計者とプログラマーの間の合意文書である。単に「仕様」というと、「設計仕様」を指す場合もあるが、曖昧であるため、本ガイダンスでは、「要求仕様」と「設計仕様」を明確に区別して表現する。

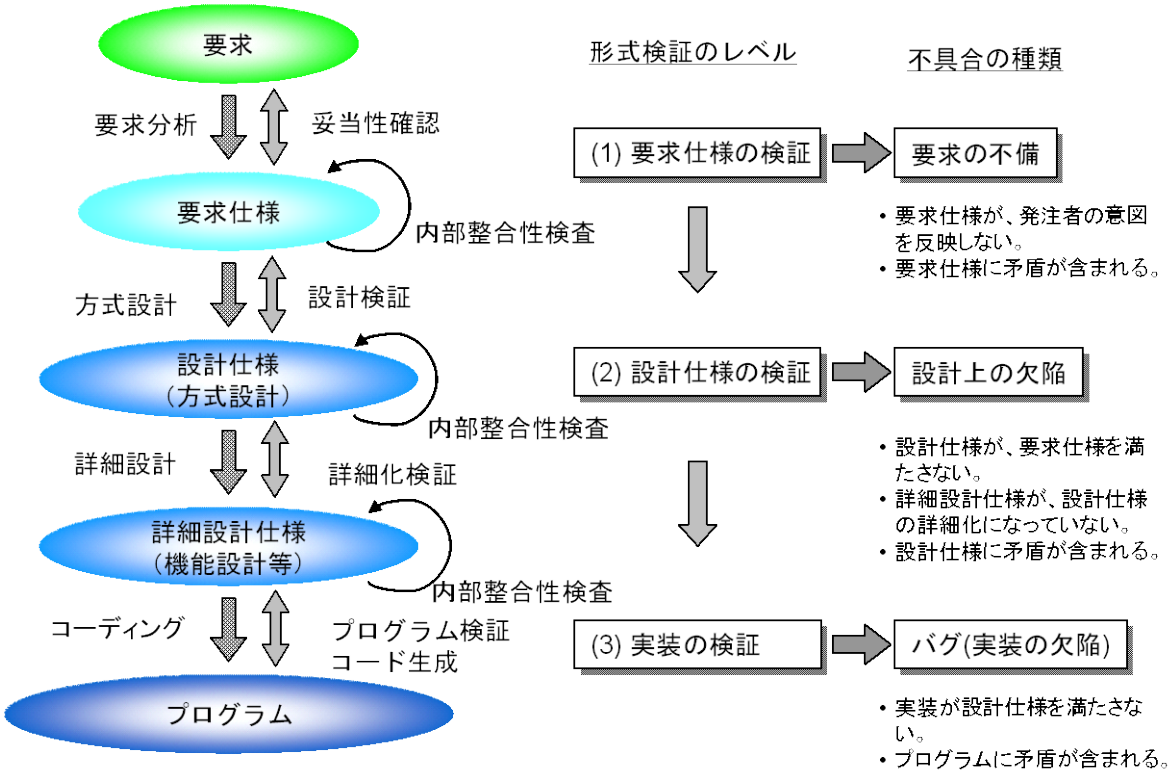

図 6-3:フォーマルメソッドの主な適用箇所と用途

図 6-3 に示したフォーマルメソッドの用途を整理すると、主に以下のようなレベルで適用されるものに分類される。

表 6-8:検証のレベル

検証のレベル	内容
実装の検証	実装に不具合(バグ)がないか。実装が設計を満たしているか。
設計仕様の検証	設計上の不具合はないか。設計仕様が要求仕様を満たしているか。設計に矛盾、定義漏れがないか。
要求仕様の検証	要求仕様はだとうであるか。要求に矛盾がないか。要求は、ユーザの意図に沿っているか。

6.2.2. V 字モデルにおける各手法の位置づけ

フォーマルメソッドとディペンダビリティ等を確保するためのその他の手法を、ソフトウェア開発V字モデルにおける位置づけを整理すると以下のようになる。

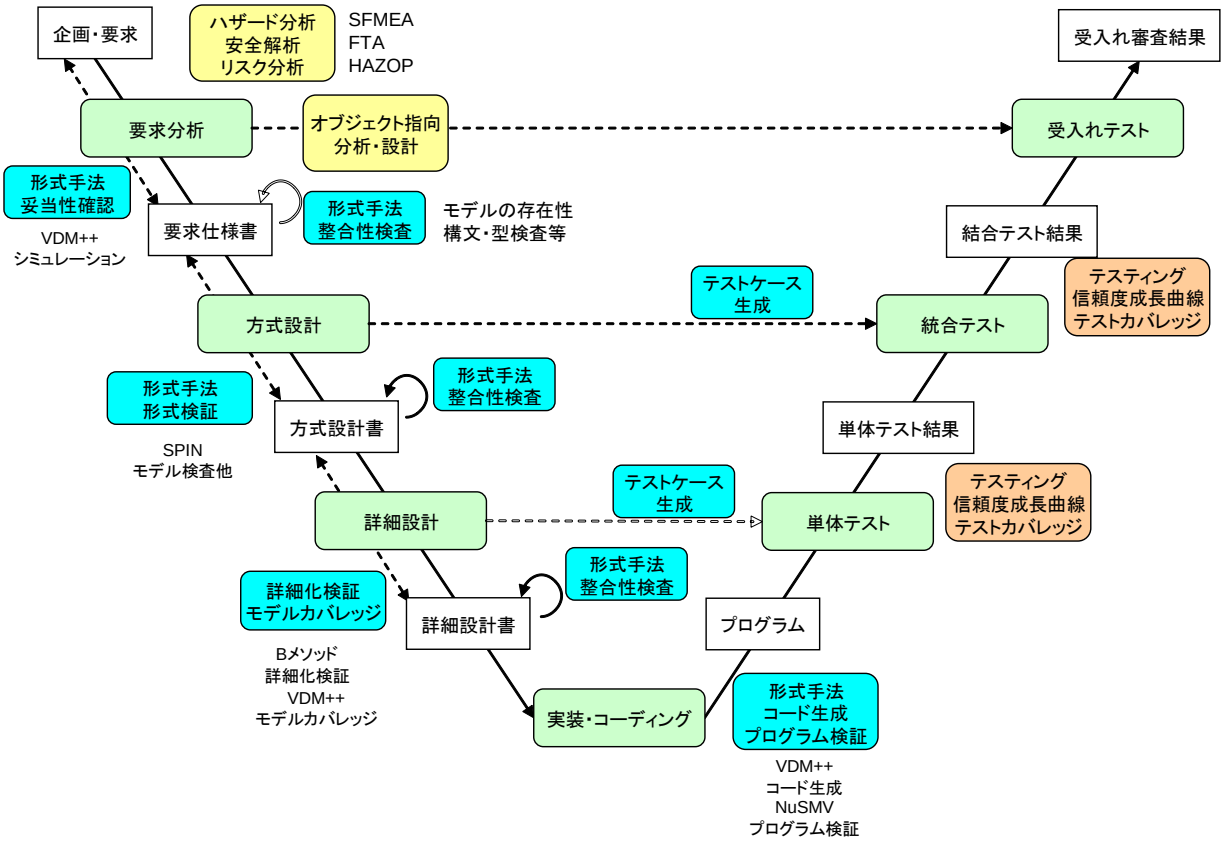

図 6-4:開発 V 字モデルにおける各種手法の適用箇所

フォーマルメソッドは、主に、上流工程で適用される。フォーマルメソッドは、定義された仕様の間の整合性や矛盾について解析することはできるが、ハザードの洗い出しを行うことには適していないため、セークティ・クリティカル・システムに求められるリスクやハザードの洗い出しには、フォーマルメソッド適用の前に、ハザード分析などが使われる。

詳細設計やコード生成が可能なレベルの設計記述では、B メソッドや VDM++などのフォーマルメソッドが使われる。どちらもコード生成機能を持っており、形式仕様記述言語で記述した設計仕様のレビューや検証項目のレビュー、シミュレーション実行、形式証明により設計仕様の品質を高める。

一方、モデル検査法では、自動検証は可能であるが、検証できるシステム規模に制約が強く、通常、上流設計の検証や、ソースコードの一部のモジュールに対する検証に制限される。

テストは、システムに期待する動作が実際に行われるかという機能要求に適しているが、安全性などの非機能要求には適さないことがある。フォーマルメソッドは、陥ってはいけない状態など安全性に関わる検査も対象とできる点に特徴がある。

6.3. システム全体の一部としてのソフトウェアの位置づけ

ソフトウェアは、それが動作するシステムという状況の中でのみ、安全性や信頼性を評価すること

ができる 。したがって、ソフトウェアを含むシステム全体について考えることが重要である。フォーマルメソッドは、エンタプライズ系やパソコンなど汎用のコンピュータシステムだけではなく、航空機、鉄道システム、電力システム、情報家電など組込みシステムへの適用の方が多数を占めている[119],[120]。また、近年、制御対象の物理システムと制御システムのハイブリッドシステムの設計モデリングと検証に対するニーズが高まっている[121],[122],[123],[124]。 これらに対するフォーマルメソッドの位置づけと役割を以下にまとめる。

6.3.1. 機能安全における安全性確保の考え方

機能安全とは、機能的な工夫により極力安全を確保する[125]ための安全機能を実現するシステムに対する国際規格である。機能安全においては、システムの障害を、以下の2つに分類する。

障害の分類	内容
確率的ハードウェア障害	部品などハードウェアの劣化等により確率的に発生する障害
決定論的障害(系統的障害)	設計や実装の欠陥により決定論的に発生する障害

機能安全では、これらの障害に対して図 6-5 のように異なるアプローチで安全性の確保を目指すものである。

[119] Woodcock, J., Larsen, P. G., Bicarregui, J., and Fitzgerald, J. 2009. Formal methods: Practice and experience. ACM Comput. Surv. 41, 4 (Oct. 2009), 1-36.
[120] IPA, 形式手法適用調査, 2010
[121] Cyber-Physical Systems Research Charge, Jeannette M. Wing, Cyber-Physical Systems Summit, 2008
[122] Cyber Physical Systems: Design Challenges, Edward A. Lee, Technical Report No. UCB/EECS-2008-8
[123] 連続系と離散系の統合モデリングと標準化, 制御設計研究会、JASA, 目時 伸哉, 2010
[124] EASIS, Marko Auerswald, Guidelines for the Development of Dependable Integrated Safety Systems, Formal Verification Techniques, 2006
[125] NECA 技術委員会報告 第三の波「機能安全」

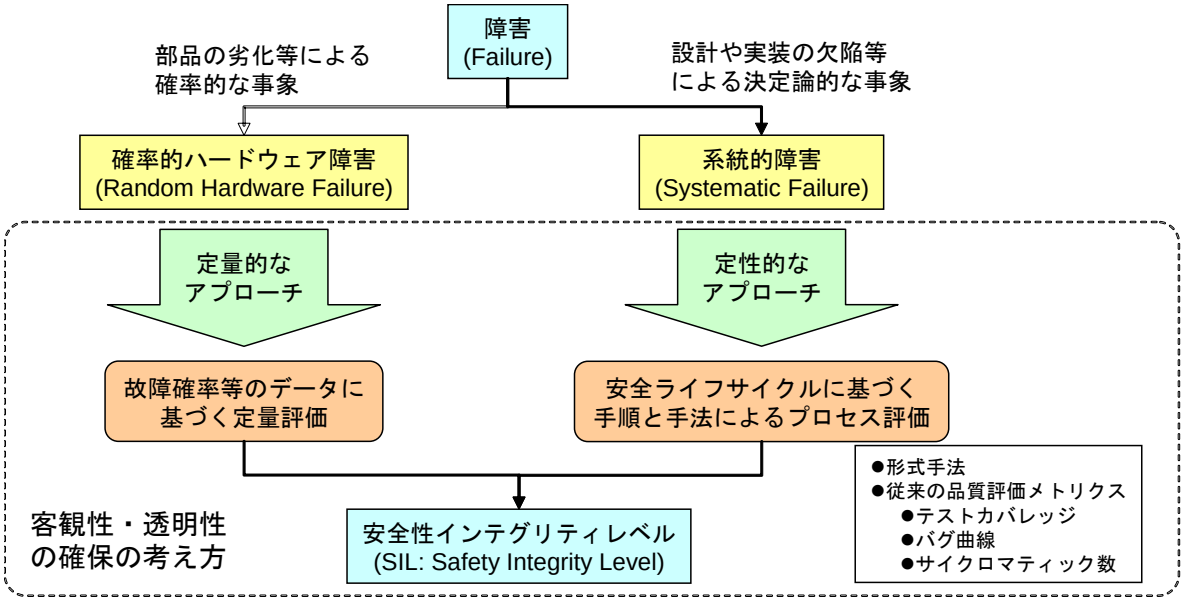

図 6-5:障害の区別と評価対策アプローチ

確率的ハードウェア障害に対しては、故障確率等のデータに基づく定量的に評価する。一方、ソフトウェアを中心に系統的障害(決定論的障害)については、定量的に評価することは困難であるとの認識から、開発手順や適用手法によるプロセス評価によって安全性の確保を目指す。これらの両方のアプローチによりシステムの安全性レベル(SIL)を確保する。

SIL は、システムのユーザである発注者が、自己責任を前提に設定する。SIL は、モジュールごとにリスク評価に基づき設定される。もっとも高いレベル 4 では、フォーマルメソッドの適用が強く推奨され、フォーマルメソッドを適用しない場合は、認証機関に対して合理的な説明が求められる。

6.3.2. ハイブリッドシステムへの応用に関する現状

自動車等の制御システムでは、制御対象の物理システム(連続モデル[126])と制御システムの離散システム(離散モデル[127])のハイブリッドシステムの設計と検証が必要になる。近年、自動車のソフトウェア化が急速に進み、ハイブリッドシステムに対する設計や検証の要求が高まっている。また、医療機器、ビル管理、橋等のインフラ管理などにおいて、連続システムである物理システムと離散システムであるサイバーシステムのハイブリッドシステムに対する設計、検証のニーズが高まっている[128],[129]。

ハイブリッドシステムにおいては、システムの重要な性質や要求は、システムと環境の連結したダイナミクスによって記述される。連続モデル(微分方程式)で表される環境からのフィードバックを扱う必要性がある場合、離散モデルと連続モデルの統合モデルが必要になる。産業レベルの

[126] 物理法則に基づき微分方程式で表されるもの。
[127] 時間的に離散イベントに基づく状態遷移システムなど。
[128] Cyber Physical Systems: Design Challenges, Edward A. Lee, Technical Report No. UCB/EECS-2008-8
[129] Cyber-Physical Systems Research Charge, Jeannette M. Wing, Cyber-Physical Systems Summit, 24 April 2008

ツールとしては、Stateflow で拡張した Simulink、MatrixX、VisSim で拡張した Statemate などがあるが、これらはハイブリッドシステムのモデリング、シミュレーション、プロトタイピングを行うための機能を提供するが、ハイブリッドモデルに対する検証機能がない。

ハイブリッドシステムに対するフォーマルメソッドのアプローチとしては、ハイブリッド・オートマトンがある。ただし、ハイブリッド・システムに対する形式検証は研究途上にあり、以下のような研究が活発に進められている 。

- ハイブリッドシステムを有限オートマトンあるいは時間オートマトンでマニュアルあるいは自動でエンコードし、モデル検査を可能にする。
- 厳密または近似したハイブリッドシステムの到達可能状態の直接生成
- 非線形制御からハイブリッド制御への証明ルールの拡張

これらの分野については、今後の動向を注視し、適用の可能性を検討していくことが重要である。

6.3.3. リアルタイムシステムへの応用に関する現状

リアルタイムシステムは、外部環境との相互作用において、ある入力に対する出力が一定の時間制約を満たすようなシステムである。多くの組込みシステム、特にセーフティクリティカル・システムは、リアルタイム性の要求が求められる。フォーマルメソッドによる自動検証を目的とした場合、時間オートマトンに基づくモデルが用いられる[130]。時間オートマトンのモデル検査系で最も先進的なものの例としてUPPAALが挙げられる 。いくつかの有効なケーススタディは見られるが、以下のような問題点がある:

- ツールは Boolean や制限された整数など単純なデータ構造のみしか扱うことができない。
- 状態爆発の問題を起こしやすい。

時間オートマトンに関するモデル検査において、このような課題を克服するための研究が行われている[131]。また、モデル検査による自動検査に対して、PVSやIsabelleを用いたインタラクティブな証明により状態爆発を回避する方法についても研究されている。

これらの分野についても、今後の動向を注視し、適用の可能性を検討していくことが重要である。

6.4. まとめ

ソフトウェア・テストは、不具合の一部を発見することができても、不具合が無いことを保証することができない。フォーマルメソッドは、定義した性質についての検証が成功すれば、その性質に関して不具合が存在し無いことを保証できるという特徴がある。また、プログラムは、ある機能をどの

[130] EASIS Guidelines for verification and validation of dependability requirements, Formal Verification Techniques

[131] E Asarin, Maler, and A. Pnuell, Data-Structures for the Verlfication of timed automata, In Proc. Of the Int. Workshop on Hybrid and Real-Time Systems, 1997.

ように処理するか記述しなければならないが、フォーマルメソッドは、システムを「どのように実現するか」(How)を記述することなく、「何を実現するか」(What)を記述することができる点が特徴である。これにより、実現方法を記述する前に、開発の上流工程で矛盾を発見したり検証したりすることができる。また、ユーザに近い上位レベルの検証性質を記述することにより、本来の目的に近い検証が可能になる。

一方で、フォーマルメソッドの実践応用例では、フォーマルメソッドが全てのコンポーネントに対して、全ての工程で用いられていない。フォーマルメソッドは、モデル検査における状態爆発や、定理証明における証明の失敗、厳密な仕様を記述することによるコストの増加、記述できる範囲の制約などの問題もある。このようなことから、フォーマルメソッドは、従来のテストを置き換えるものではなく、補完的に組み合わせて利用されることが現実的である。また、フォーマルメソッドは、セーフティクリティカル・システムに求められるハザード分析による障害の原因などの求を洗い出しに適していない。これらの手法と組み合わせて利用されることも必要である。

フォーマルメソッドは、対象システムや手法の適性に応じて、検証範囲を特定し、従来のテストを部分的に置き換えるなどして、フォーマルメソッドと他の手法を組合せすることで、コストを抑えながら全体としてソフトウェアの信頼性・安全性を向上さるといった利用を検討することが重要である。