である。従来のリスク分析では、重要パラメータの値を乗じて、数的リスク指数（クリティカリティ（*criticality*）と呼ばれる）を求める。軍用規格MIL-SPEC 1629Aによれば、通常、この方法はソフトウェアの分析に用いられない。その代わりとして、細菌のシステム安全性規格MIL-SPEC 832Cのガイダンスを用いて、発生率と損失またはハザードの重大性の両方の質的属性の組み合わせを示す表を作成することができる。この単純な例を表1に示す。この表は、図1に示した2次元リスク図と同様であることに注意されたい。

	ハザードの重要性/損失		
発生確率	軽微	中程度	重大
ありそうもない（Improbable）	低い	低い	中程度
ほとんどない（Remote）	低い	中程度	高い
時々ある（Occasional）	中程度	高い	高い
かなりある（Reasonable）	高い	高い	極めて高い
頻繁にある（Frequent）	高い	極めて高い	極めて高い

表I.　単純なリスク指数の例

特定されたリスクの組み合わせを探すためにリスク指数表を用いることは、非常に有用であることがわかっている。この方法を用いる際には、以下の点に注意することが重要である。

- リスク指数表は、それぞれの発生頻度および重要性についての質的パラメータの説明を含め、正式に文書化すべきである。
- 開発チームまたは品質グループは、異なるラベルおよび値を用いて独自の表を定義することができる。

特定されたリスクは、開発チームの多くの者にとって重要な情報であると思われる。これらの者はリスク分析が完了した後にチームに参加することもあるため、リスク判定の状況を容易に理解できるように十分な詳細が提供されなければならない。リスクの高い項目についてマネジメントが判断の責任を共有できるよう、数値と適切な措置とを展開することができる。別表にリスク指数値の許容度を説明する（表II）。例えば、それぞれのリスク指数を特定のハザードまたは安全性の喪失ならびに原因に結びつけることができる。原因は軽減策に関連がある可能性があるため、リスク指数について記載されている文書には、

指数がハザード軽減策の指定前または指定後のどちらで割り当てられたかどうかを示さなければならない。軽減前に割り当てられた場合、リスク指数を用いて軽減メカニズムの必要性を示すことができる。軽減後に割り当てられた場合には、リスク指数により、原因と軽減策の組み合わせで損失がどの程度減るかが明らかになるはずである。

リスク指数値	措置
極めて高い	許容できない－（さらに）軽減策を要す
高い	エンジニアリングおよび品質担当役員の署名がある場合のみ許容できる
中程度	プロジェクトマネジャーの署名がある場合、許容できる
低い	レビューなしで許容できる

表II.　リスク指数値と措置の割り当て表の例

安全性に関する重要変数

通常、プログラム実行には値の設定と変更が必要となる。多くの値は、医療用具のシステム安全性要求事項への適合に対してほとんど影響を及ぼさない。一部の変数、例えば輸液ポンプの投与速度または除細動器が放出するエネルギー量などは、医療用具の安全性に直接的に関連がある。操作者はこのような値を医療用具のフロントパネルから直接入力することができる。重要な管理値を含む計算された変数も、医療用具の安全性に何らかの役割を果たす。この一例としては、所定のポンプ投与量を達成するためのステッパーのモーター速度が挙げられる。その値が医療用具の安全性に影響を及ぼす変数を重要安全変数（*safety-significant variables*）という。

従来のリスク分析には、システムがヒトを脅かす確率の決定が含まれる。MIL-SPEC 1629Aに従って実施される分析には、発生確率値、重要性、検出能の積算が含まれる。このプロセスは、ソフトウェアのリスク分析を熟知していないエンジニアを混乱させることがある。現在、医療用具開発の分野で実施されているソフトウェアのリスク分析は、このレベルで数量化を確実に裏付けるものではない。

安全性に特異的なソフトウェア

ソフトウェアのリスク分析は、すべてのソフトウェアが医療用具の安全性要求事項への適合に直接関与しているわけではないという考えによって決まる。安全性要求事項の支援

は、ソフトウェアのbuilding block間で偏在している。通常、安全性要求事項を満たすモジュールは、*セイフティクリティカルまたは重要安全（safety-significant）*と称する。例えば、患者に供給されるエネルギーレベルを制御するアルゴリズムを含むモジュールは、バックグラウンドのハウスキーピングタスクを行なうものよりもはるかにセイフティクリティカルである。工学文献にも、安全性に関連したソフトウェアはセイフティクリティカルソフトウェアに不具合が発生した場合に安全性を確保するためのセイフティネットを構成するものとして記載されている。

単純な医療用具の場合、ソースコードが実行可能な機械命令のブロックにコンパイルされていることから、医療用具内のすべてのソフトウェアがセイフティクリティカルというわけではないという概念は理解されにくいかもしれない。抽象的境界（abstract boundaries）は機械命令のモノリシックブロックに適用されない。あるソフトウェアの不具合が結果的に安全性要求事項にかかわるソフトウェアの不具合を引き起こす可能性について確認することは容易である。通常、この脅威は以下の3つのポイントに要約される：

- 他のソフトウェアが安全性性能に影響を及ぼす変数を破壊（corrupt）する可能性がある。つまり、破壊（corruption）を検出できるようにセイフティクリティカルな情報を維持しなければならないということになる。
- 他のソフトウェアが実行スレッドの不具合を引き起こし、正常シーケンス外のコードを実行させる可能性がある。良好に設計されたセイフティクリティカルソフトウェアは、クリティカルコードセグメントの正しいシーケンスを保証する。
- 他の設計不良のソフトウェアが、プロセッサ計算処理能力およびワーキングメモリ等の計算リソースを消費したり、誤って運用する可能性がある。その結果、セイフティクリティカルソフトウェアが機能しなくなる可能性がある。C++が一般的になったため、エンジニアは、いわゆるメモリリーク（メモリリソースが解放されない環境に存在するソフトウェア実行スレッドから生じる）に対処しなければならない。安全性に関連したソフトウェアはメモリリークを軽減しなければならない。ひとつの解決策は、ローカライズされたリソースコントロールをシステムコントロールと切り離すことである。

軽減策

リスクマネジメントは、ソフトウェアの不具合がリスク増大をもたらす可能性があるという前提に基づいている。開発者は、リスクの削減（すなわち軽減）のためのアプローチを定めなければならない。これによって設計者は、ソフトウェアの不具合の可能性と軽減策を結びつけて考えることを要求される。このような組み合わせについて、ソフトウェアの不具合の可能性の削減において最低限の信頼性のあるものから最重要なものまで、以下に説明する。

ユーザーへの情報提供　潜在的リスクが情報プロンプトに関連している可能性がある。例えば、「エラー：情報がスクリーンバッファの誤った領域に記述されている。軽減策：予想されるディスプレイに関連するユーザー文書を提供する（Failure: Information written to wrong area of screen buffer. Mitigation: Provide use documentation relating to expected display）」。この特別な軽減策は、開発および品質チームの管理外の活動に依存しているため、明らかに脆弱である。開発者は、スクリーンレイアウトによって異なるスクリーン領域にある情報への手がかりが得られるようにするため、指示事項を見直さなければならない。

情報の表示と関連するハザードは、特に難しい。開発チームが、意味をなさない表示や熟練した医療従事者は意味をなさない表示や間違った値を検出できるはずである、ということを表示するに過ぎないことも多い。軽減策の品質は、医療従事者に提示される画面情報の価値と医療従事者の訓練の程度に応じて異なる。情報が治療に重要である場合、設計の仕方によって、情報の妥当性を遡って確認するディスプレイバッファの判読手段を提供できる。*dead facing*（医療用具のディスプレイに何も表示されないが、医療用具は治療を続行している）はさらに危険である。

開発プロセス　ここでの典型的な組み合わせには以下がある：「エラー：欠陥のある呼吸アルゴリズムの実行。軽減策：独立レビュー（Failure: Flawed breathing algorism implementation. Mitigation: independent review）」。これは、開発プロセス内で例外的な事項が実施されることを示す。プロセス軽減策を完了するには、軽減策をソフトウェア開発プラン上に記載するとともに、監査を行なって、独立レビューが実施されたこと、同様に重要なこととして、あらゆる所見について対処されたことを保証しなければならない。

ソフトウェアのメカニズム　ソフトウェアの問題を表現する組み合わせは以下のように提示されることがある「エラー：ポンプ速度変数が上書きされた。軽減策：変数を単一機能によって重複して保存、アクセス、変更する（Failure: Overwritten pump-speed variable. Mitigation: Variable redundantly stored, accessed, and changed by a single function）」。このような特殊なソフトウェアの追加は一般的であり、これによって構造化されたアクセスが強化され、各アクセス時の破壊の発見が可能になることから、グッドプラクティスとみなされる。しかし、これらのメカニズムは、ずさんな使用（重要変数のローカル使用変数への移動等）が許容される場合、脆弱となる可能性がある。このことからも、使用ルールの検出と強化のため、ソフトウェアのメカニズムが開発プロセスの一部（コード検査等）を必要とすることがあるということがわかる。

ハードウェアのメカニズム　ハードウェアの軽減策に必要なエラーの例には以下がある「エラー：ランナウェイ実行スレッド。軽減策：ハードウェア監視タイマー（Failure: Runaway execution thread. Mitigation: hardware watchdog timer）」。ハードウェアは医療用具のセーフティネットを提供するための独立した技術に依存しているため、個別のハードウェア安全性メカニズムのインストールがグッドプラクティスとみなされる。しかし、ソフトウェアが監視回路と正しく接続できず、スタートアップテストによって機能不良が発見できない場合、この特別なハードウェア軽減策が無効となることがある。

実行の多様性　この種の軽減策は、安全管理者を設けるシステム構造に依存している。安全管理者は、主要プロセッサとソフトウェアストリームが医療用具の安全性要求事項に従って作動するようにするため、独自のソフトウェアを備えた個別のプロセッサを採用している。これは、この種の構造（プロセス管理システムの文献に説明されている）には一般的であり、市販プロセッサとソフトウェアパッケージにみられる[5]。

欧州連合では、特定の医療用具に関する軽減メカニズムの対応速度が重要であると考えられている。例えば、シリンジ注入ポンプの規格は、投与速度がモーター速度に比例していることが要求される。これは、この種の医療用具では過量投与が一般的なハザードであり、モーター速度が高速となるソフトウェアランナウェイによって引き起こされる可能性

があるためである。一般的な軽減策の形式は監視タイマーである。タイマーが実行される方法に応じて、ソフトウェア障害からエラーが検出され、ポンプモーターが停止した安全状態に移行するまでの経過時間が長すぎて、危険な量の薬物が制御不能で投与されることを回避できないことがある。軽減策が存在しても、治療速度よりも遅い場合、リスクエクスポージャーを軽減できないことがある。モーター速度を命令することができるlock-and-key ソフトウェアによってより迅速な軽減策が可能となり、リスクエクスポージャーをソフトウェアのより安全な領域に移動させると思われる。

Lock-and-keyソフトウェアは、操作者が正しいキーを提示する場合にのみ実行されるセイフティクリティカルソフトウェアの機能に依存する。lock-and-keyソフトウェアが正しく実行された場合、これは機能の途中でソフトウェアシステムジャンプの検出も行う。次に、Lock-and-keyは、機能の開始時に不法な実行入力のコマンドを検出するが、開始はしない。入力によって機能がさらにダウンすると思われる場合、lock-and-keyはモーターに対する命令後、数マイクロ秒以内に不法コマンドを検出する。監視とlock-and-keyの併用による解決策によって最大の保護が得られる。

原因と軽減策のパターン

ソフトウェア開発者にとって最近のパラダイムのひとつと思われるものは、パターンという考え方である。パターンは、共通の問題を解決するためソフトウェアオブジェクトの集合体を特定の方法で連結できるという観察に基づいている。この仮定は抽象的であるが、このアプローチによって、複数の問題の解決策はある程度似通った形式をとるという考え方が正しいことがわかる。

パターンのパラダイムを適用することにより、原因と軽減策の組み合わせを能率化することができる。小規模なソフトウェアの不具合のパターンを表IIIに示している。この表の軽減メカニズムの一覧は、より一般的なアプローチからあまり一般的でないアプローチの順に、大まかに示してある。すべてのパターンは、ベースラインとしてコード検査および検証試験を含むしっかりとした開発プロセスを想定している。これらのパターンは開始地点のみを示している。難しいのは、製品の構造や実行環境に特有と考えられる補助的なパターンを特定することである。

不具合	軽減メカニズム
データ/変数の破壊	重複コピー；妥当性の確認、アクセス制限
	CRCまたは保存スペースのチェックサム
	フェッチの合理性検査
ハードウェアに基づく問題	スタートアップ時の厳格なbuilt-in-self-test (BIST)
	合理性検査
	診断ソフトウェアのインターリーブ (不法機能入力；データ破壊を参照)
ソフトウェアランナウェイ；不法機能入力	ハードウェアの監視
	入力および終了時のLock-and-key
	限界／合理性検査
	独立チェックとともに記録される実行スレッド
メモリリークが実行ストリームを困難にする	コード検査チェックリストとコーディングルールを明示する
	メモリー使用量の解析
	使用量ストレス解析に基づく組み込みコード
	セイフティクリティカル機能のためのローカルメモリコントロール
欠陥のある管理値がHWに提示される	合理性検査との独立リードバック
	HWメカニズムが独立制御／安全状態を提供
	安全管理者のコンピュータが値を承認しなければならない
情報表示の欠陥	BISTと使用説明書におけるユーザーレビューの指示
	独立ソフトウェアチェックとのリードバック
	別の表示プロセッサが合理性をチェック
メモリの不法使用が重複	検査チェックリスト項目を明示する
	配置・配置解除に関するコーディングルール
	特別ポインタ割り当てルール

表III　不具合のパターンと軽減メカニズム

軽減策のリンク化

一部の安全性関連ソフトウェアが実施されない可能性と、このため軽減機能が正しくサポートされないことは、多数の開発者が関わる大規模プロジェクトでは重要である。従って、ソフトウェアのトレースを行なうことが重要になりつつある。通常、トレースでは以後の活動と製品開発ライフサイクルの初期に決定された事項とを関連付ける。

トレースは、ソフトウェアのリスクマネジメントの重要部分である。少なくとも、軽減策の実証は試験される原因と軽減策の組み合わせに関連付けられなければならない。より保守的なアプローチ（血液銀行用機器およびシステムにみられるソフトウェアに適用されるようなアプローチ）は、それぞれの原因と軽減策の管理および安全性要求事項を、製品

のソフトウェア要求事項仕様における特定の要求事項に関連付けることである[6]。安全性要求事項と機能を遂行する特定の論理的ルーチンとの関連付けも、認識と軽減策、および認識と実証を結びつける。

ベストプラクティスの追跡

同様の臨床機能を実行する医療用具には以下の傾向がみられる

安全性特異的ソフトウェアの決定

すべてのソフトウェアのbuilding blockに適用される可能性がある一連のカテゴリーを決定することで、重要安全ソフトウェアの決定を早めることができる。これらのカテゴリーは手順になったり、ソフトウェア開発プロセスを定める文書に取り入れられたりすることがある。以下のカテゴリーは主要な重要安全ソフトウェアの特定のための開始点を示す:

- その不具合が直接的に安全性要求事項を損ない、軽微ではないハザードをもたらす可能性があるソフトウェア。ほとんどすべての治療用医療用具の制御ソフトウェアはこのカテゴリーに分類される。通常、診断用医療用具のアッセイ値計算のためのアルゴリズムソフトウェアはこのカテゴリーに分類される。
- ソフトウェアの他のセグメントまたはハードウェアによる不具合を軽減するために用いられるソフトウェア。医療用具のスタートアップに用いられるBuilt-in-self-test (BIST)ソフトウェアは一般にこのカテゴリーに分類される。セイフティクリティカルソフトウェアの不具合の検出に用いられる安全性関連ソフトウェアもこのリストに加える。
- 重要安全値に直接アクセスするソフトウェア。これは、これらの値にアクセスし、変更するための特別ルーチンを用いるgood design practiceとみなされる。このような特殊なソフトウェアはこのカテゴリーに分類される。
- 他のカテゴリーに含まれるソフトウェアによって直接呼び出されるサポートソフトウェア

すべての内蔵ソフトウェアに一定レベルのソフトウェアの品質保証（レビューおよび試験を含む）を適用すべきである。重要安全ソフトウェアの重要性を考慮して、この特別な

ソフトウェアには追加の品質保証活動を適用すべきである。強化された活動には、より正式な設計およびコードレビュー、ならびに機能的（ブラックボックス）試験および構造的（ホワイトボックス）試験が挙げられる。品質保証活動が厳密であるほど、重要安全ソフトウェアの開発費用もかさむ。また、重要安全ソフトウェアが非常に少ない医療用具は、このようなソフトウェアを多く備えた医療用具よりも本質的に安全である。ただし、これはシステム構造に依存する。重要安全ソフトウェアの数（通常、全ソフトウェアに占める割合で示す）はできるかぎり少なくすべきである。この割合は、医療用具の構造と用途によって異なる。一般に単純な診断用医療用具では重要安全ソフトウェアが20–30%を占めるのに対し、生命維持エネルギーを供給する医療用具では、約85%を占めることがある。血液銀行での使用を意図した複雑な自動診断用医療用具でも、重要安全ソフトウェアが高い割合を占める。

開発プロセスの関連付け

ソフトウェアのリスクマネジメントでは、製造業者が確実なソフトウェア開発プロセスを実施していると仮定する。トレーサビリティ、コード検査、インターフェースおよび構造ユニット試験、総合テスト、設計の妥当性確認等の検証プロセスを採用することは、組織にとって一般的になってきている。通常、ソフトウェア開発は、十分に認識され、十分にコミュニケーションが行なわれたプロセス、信頼できるツール、および優れたエンジニアで構成された、一つの機能であると言われている。早期の障害発見が決め手となるソフトウェア開発プロセスを有することは、規制当局の期待するところである。一部の組織は、ソフトウェアの障害を検出するために設計された標準プロセス（すなわち、標準的開発プロセスの一部となる軽減策の形式）を明確に示している。最近は設計管理が重要視されているため、特異的ツールまたは統合的サポートパッケージが開発されるにつれて、このプラクティスはより一般的になると思われる。最後に、今後はソフトウェアのリスク分析の訓練が多くなるとともに、エンジニアリングへの期待も厳しいものになると思われる。

結果の伝達

最近のワーキングセッションにおいて、あるソフトウェア開発マネジャーはソフトウェアのリスク分析に対する反感を示し、「これによって新発見があるわけではない」と述べた。特定の問題のある環境（診断システムの小規模で漸進的調整を行う場合）では、この

マネジャーの反応は正しかった。典型的な新製品の開発環境と異なり、この状況では、開発チームがソフトウェアのリスク分析をレビューして、すでに実施された軽減メカニズムが現在も理にかなったものであること、また、データベースやプロセッシングの諸経費がかさむにつれて生じる可能性がある積層欠陥に基づいてシステムが不具合に至らないことを確かめる必要があった。

ソフトウェアのリスクマネジメントは、4つの伝達経路に従って情報を伝達すべきである。第一に、分析は、すべての開発エンジニアに対し安全性要求事項を満たす上でのソフトウェアの役割を伝達すべきである。この一部として、ハードウェアが、どの程度適切に、また強力な軽減策を支援するソフトウェアとは独立して機能しなければならないかを理解することが必要となる。第二に、開発プロジェクト内で作業産物は軽減メカニズムが機能していることを実証するため、検査、試験、特別検査の形式で検証を必要とする項目に相当するポイントを伝達すべきである。第三に、ソフトウェアの役割を規制当局に伝達する上で作業産物を役立てるべきである。第四に、例に挙げたように、リスクマネジメントが有効かどうかは、変更事項が増えるにつれて医療用具のリスク状況が継続的に検討されているかどうかによって決まる。

システムとソフトウェアのカップリング

ソフトウェアのリスクマネジメントは、システムのリスクマネジメントの一部であることもあれば、システムのリスクマネジメントと別のものであることもある。開発の初期段階で、潜在的ハザードの原因としてソフトウェアの不具合の概要が認識されるのが普通である。詳細なシステムレベルのリスク分析によってもソフトウェアの不具合の分析の寄与が、より詳しく挙げられることがある。考えられる不具合の経路は、通常は全システムではなくサブシステムに注目する設計段階で分析される。この詳細な分析はソフトウェアの不具合に取り込まれることが多く、システム全体の詳細な分析におけるラベルとして現れる。

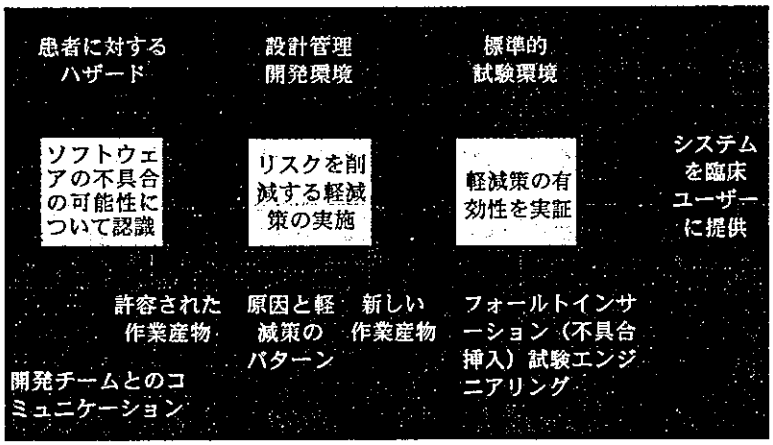


図2. 認識、軽減策、実証ストラテジー

この概念は、認識、軽減策、実証（AMD）ストラテジーを強調するリスクマネジメントの枠組みに完全に拘束される。このストラテジーは前述の主要な概念と関連があり、これを図2に示した。認識は、患者に対するハザードに関連があり、安全性要求事項を満足する上でのソフトウェアの役割を伝達しなければならない。軽減策は確立された原因－軽減策パターンを活用すべきである。これは、現在認識されているソフトウェア開発のためのベストプラクティスに依存する開発環境において発生しなければならない。有効な軽減策の実証は、正式試験環境ならびに欠陥挿入試験に関する工学分野の発展にかかっている。

作業産物

AMDストラテジーは、ソフトウェアのリスクの認識に対する作業産物ならびに意図した医療を実施するための設計と実行に依存している。さらに、リスク削減を実証するためのプロトコールと試験結果にも依存している。実行作業産物は、特定の環境のために確立されたノルマに依存している。同様に、実証作業産物は単純であり、企業に特異的なテンプレートを用いて達成することができる。

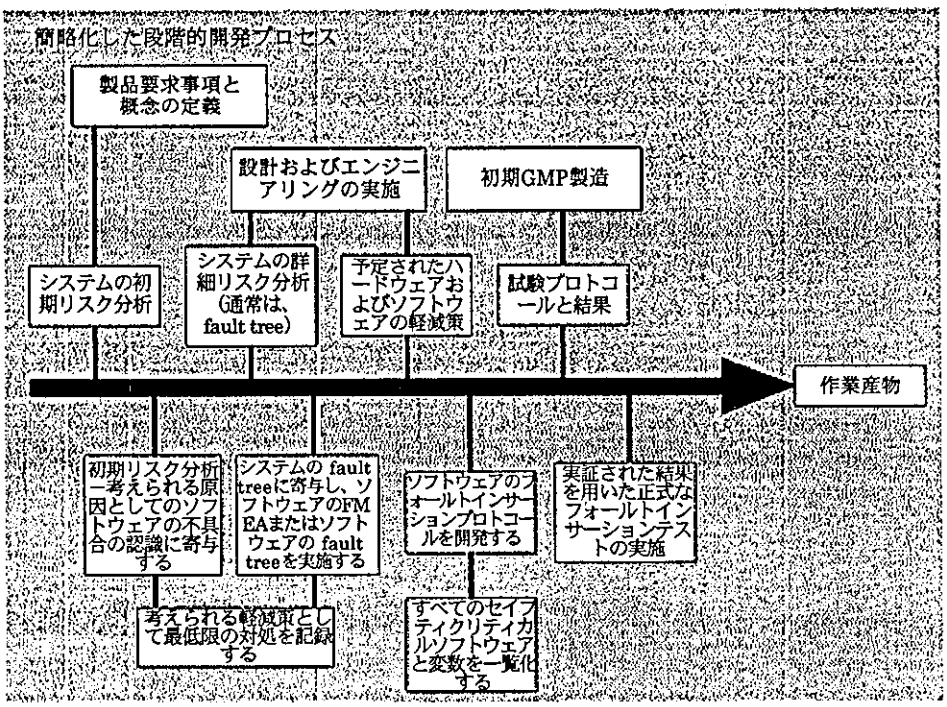


図3. 作業産物の統合と連続性

図3に、ソフトウェアのリスクマネジメントに通常関連がある作業産物の位置を例示している。上部に3つの広義の開発段階（通常は重複する）を例示している。この下に、示した段階にたいていは関連があるリスクマネジメント作業産物を示している。中央のラインの下に、ソフトウェアのリスクマネジメントに特異的な作業産物を示している。この図には、名目上のソフトウェア開発に必要とされる作業産物と活動（コード検査、検証試験等）は示していない。一部の事例では、初期リスク分析(IRA)と同様に、医療用具をトータルシステムとして見た場合、ソフトウェアの分析は作業産物への寄与物であるということに注意されたい。以下のセクションに、各種作業産物について、それぞれのプロセスパターンを含め、重要な詳細を示す。

初期リスク分析

システムのリスク分析には、定義段階の作業および設計段階のより詳細な分析が含まれている。IRAは製品概念とシステムの初期構造レイアウトの知識に従って実施されることが多い。表IVに示したように、IRAは発生の可能性に関連付けられた潜在的ハザードと重大な原因を含むカラム形式でレイアウトされる。発生確率と重要度を指定した後、リスク

指数を特定することができる。第4カラムの見出しに示したように、この分析レイアウトはリスク指数が最低限の対処後に割り当てられることを想定している。また、この分析表には、潜在的ハザードの軽減策に必要とされる最低限の対処をリストしたカラムも含めるべきである。この情報は、安全性要求事項に関連する可能性がある非常に早期の設計および品質作業へのコミットメントに相当するため、重要である。最低限の対処には、リスクマネジメントの軽減策および実証の部分を通したトレーサビリティを向上させるためのタグ(例、IRA_SWFailure1)を含めることができる。

ハザード	重要度	考えられる原因	発生確率	最低限の対処後のリスク指数	最低限の対処
患者の熱傷	重大	制御ソフトウェアの不具合	ありそうもない	中程度	ハードウェアモニタリングと制御回路を切り離す[IRA_SWFailure]
患者の熱傷	重大	温度変数の破壊	ほとんどない	高い	温度変数の重複保存[IRA_CorVar]

表IV. 2つのソフトウェアの原因を示す初期リスク分析フラグメント

軽減策前および最低限の対処による軽減策後のリスク指数を示すカラムを加えることができる。このレイアウトを用いる場合、重要度および発生のカラムを省略することが一般的である。このアプローチは、表においてリスク指数を検索する際に下す判定を省略しているため、問題が生じることもある。しかし、原因と軽減策の組み合わせが適切であるかどうかを決定できるため、リスク指数は最も重要な因子である。

例えば、上の表には、2つのソフトウェアの不具合に関連した原因によって生じる可能性がある患者の熱傷というハザードが示されている。いずれの状況においてもこのハザードの重要度は重大とみなされているが、2つの原因の間で発生の可能性は異なる。これは、適用される軽減メカニズムの種類に起因する可能性がある：一方はソフトウェアから独立しているのに対し、もう一方は軽減ソフトウェアに完全に依存している。表IIに、上級マネジメント2名による署名を必要とする高いリスク指数を示している。代替として、この分析を実施する者は、チェックサム範囲を有する補助的変数の保存（これによって通常、破壊の検出機会が増える）のような軽減メカニズムを選択することができる。

種類	時期	形式	対象者	レビュー	詳細	トレース	影響
認識：軽減策の第一の記述を含むトップダウン分析	開発サイクルの初期に実施	表	開発および規制 変更・管理文書	正式	広い	試験を実証する「最低限の対処」	全体的構造と製品要求事項文書の変更の可能性をもたらす

表V. 初期リスク分析プロセスのパターン

ソフトウェア側の原因は、一般に直接ソフトウェアのサブシステムの特異的セクションに至る可能性のあるモジュールまたは変数名のリストなしで報告されることに注意する。IRAは特異的な設計またはコーディングの前に実施されるべきである(表V)。また、IRAの結果に基づいて、安全性要求事項とリスク分析所見との均衡を図るため、製品要求事項文書が変更される可能性があることに注意する。分析手法は、開発レビューの大部分を実施する一個人によるものから、インタラクティブなワークショップ形式まで様々である[2,3]。

詳細リスク分析

詳細リスク分析は、製品開発の設計段階で実施されるべきである。詳細設計入力が必要な場合でも、詳細リスク分析が設計に影響を及ぼすことがあるため、全体設計と同時に検討することが最善である。ソフトウェアのリスク分析はシステムの詳細リスク分析を裏付けることができる。通常、開発チームもソフトウェアに特異的なリスク分析を構築する。

詳細リスク分析は、3つの方法で実施することができる。第一に、開発チームがfault tree（故障の木）解析を開発できる。これはトップダウンアプローチである。第二に、ソフトウェアの故障モード影響解析(FMEA)を構築するという一般的なアプローチである。FMEAはボトムアップ解析であり、ソフトウェアの障害と潜在的ハザードに対する精査から開始する。第三は、ハイブリッドアプローチで、ソフトウェアのコンポーネントの障害から開始し、下位およびゲートのシステム全体のfault treeまで作業するFMEA法を採用する。fault treeおよびFMEAの開発手法は、文献において論じられている。ここ数年、fault treeのグラフィカル構造を支援するツール（ブール演算子の一般的記号を含む）によって、作成速度が向上した。

ソフトウェアのコンポーネント	故障モード(ソフトウェアエラー)	エラーの原因(障害の種類)	システムへの影響	ハザード	軽減策
シグナル解析	1.シグナルサンプル地が誤って処理された	値がADCから引用される際にチェックされないFetch_Error_Flagに寄与するロジックエラー	過度のエネルギーが患者に供給される	患者の熱傷	重複したプロセッサが患者用プローブからの単一の値も監視する；独立してエネルギーレベルを監視し、命令されたエネルギーレベルがtable lookupと異なる場合、安全状態に移行する[Hazsig_<BRSAMPLE_ERROR]< TD>
	2. エネルギーtable lookupエラーに対するプローブシグナル	テーブルが他のソフトウェアによって破壊された	過度のエネルギー値が患者に供給される	患者の熱傷	値を返す前にテーブルのCRCチェック を実施するTab_fetchに限定されるTable fetch；破壊が検出された場合、Tab_fetchは安全状態に移行する[HazCRC_check]

表VI. 詳細なソフトウェアのリスク分析フラグメント

種類	時期	形式	対象者	レビュー	詳細	トレース	影響
FMEA：障害のある"指定された" ソフトウェアコンポーネントがハザードを引き起こす可能性があることを認識する ボトムアップ解析	詳細なソフトウェア設計中に実施	テーブル：作表機能のあるワープロツール	実行と試験エンジニアによる強力な認識を有する開発チーム 変更-管理文書	チーム；正式	非常に詳細	軽減策から実証試験	ソフトウェアの設計および全体的なフォールトインサーションテストストラテジーの変更に至る可能性がある
fault tree (故障の木) ：通常、システムのfault treeを詳述するトップダウンマッピング；ボトムとゲートは誤り、指定されたコンポーネント、軽減要素を示すべきである	通常、システムのfault treeが完成した後に実施する	記号による階層tree図 fault treeツールによって作成が迅速になる	上記	上記	ラベルに特異的要素を記載する	上記	上記

表VII. 詳細リスク分析プロセスのパターン

ソフトウェアのFMEAのレビューからは、このプロセスが発展し、MIL-1629A規格のすべてのコンポーネントがもはや保持されていないことがわかる。表VIにソフトウェアのFMEAにおいて一般的にみられる要素を示す。表VIIには、ソフトウェアの詳細リスク分析の作業産物についてのプロセスのパターンを概説する。この種の分析における主要な要素には以下が挙げられる：

- 特定のソフトウェアの要素（例えば、機能、または高位ではモジュール）
- 特定コンポーネントの不具合の挙動（特定の障害とともに発生するエラーの種類を含む）
- 特定の患者に対するハザードに至るシステム挙動の説明。患者に対するハザードはIRAに引用されたものにすべきである。新しいハザードは、IRAの改訂が必要な可能性を示唆している。これは、IRAが初期段階を超えて妥当である場合に特に必要である。
- ソフトウェア試験チームが強制的に特定のソフトウェアエラーを発生させたときに、軽減メカニズムを実証できる軽減策の詳細。タグ付けによって、軽減策の追跡が保証される。

セイフティクリティカルソフトウェア

ソフトウェアのリスクマネジメントでは、セイフティクリティカルソフトウェアの特別な処理を必要とすることがある。このソフトウェアは前述のルールを適用して特定することができる。ソフトウェアの設計者はリストに詳細なインプットを提供することができるが、通常、詳細リスク分析によって確定される(表VIII)。

種類	時期	形式	対象者	レビュー	詳細	トレース	影響
一連の正式に承認されたルールに従って決定されたセイフティクリティカルであることが明確なソフトウェアを認識する	設計段階の範囲内で実施する	ルールを添付した一覧表	実行および試験チーム コンフィギュレーション管理はプロジェクトレベルである	詳細リスク分析と一致していることを保証するためのソフトウェアの品質チームによるチェック	詳細	品質計画に従ったソフトウェアの広範囲の品質保証活動を受けると記載されたすべてのモジュールを保証しなければならない	確実なソフトウェア配布のために必要とされる通常のプロセスに加えて、安全性に特異的な検証タスクを定義する傾向がある

表VIII. セイフティクリティカルソフトウェアのプロセスのパターン

ソフトウェアの品質計画は、セイフティクリティカルソフトウェアに適用される追加レベルの検証活動を定義すべきである。これらの活動は一般的に以下の項目が含まれる：

- 正式な従来のFagan式の設計およびコード検査
- 独立したブラックボックステスト
- 非常に詳細な独立した構造テスト（full-statementおよびdecision-predicateテストを含む）
- 発生した措置とすべての欠陥が処理されたことを保証するための進捗監査および最終監査

フォールトインサーション（不具合挿入）テスト

AMDリスクマネジメントモデルの成功は、軽減メカニズムによってリスクが軽減することを実証する最終ステップにかかっている。このステップは、文書化されたプロトコールに基づく正式に承認された試験、結果の収集、発見された問題の処理、監査完了からなる。この試験は、少なくとも2回実施されるべきである。1回目は、すべての検証活動を受けたソフトウェアに関するエンジニアリングプロトタイプについて試験を実施すべきである。次に、試作品、すなわち準備完了ソフトウェアを内蔵したGMP製造医療用具について試験を実施すべきである。

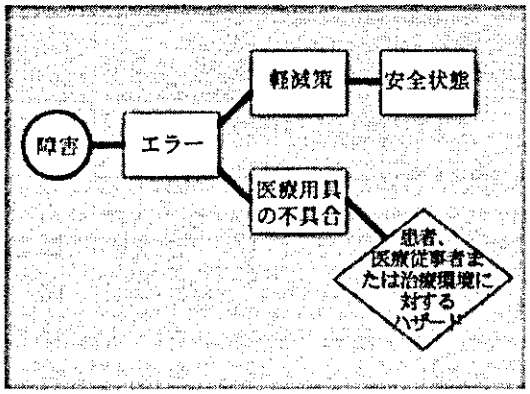
種類	時期	形式	対象者	レビュー	詳細	トレース	影響
以下について実証する：軽減メカニズムがリスク軽減に有効であることを実証するよう努める	プロトタイプおよび試作段階で実施	試験グループが定めたテンプレートに従う	開発；要約が記述され、終了時の監査が実施されるような形式で結果を収集しなければならない	正式である可能性があるが、必ずしも実施されるとは限らない	非常に詳細	IRAおよびDHA作業産物タグへの遡及を支援する	不具合の安全性を決定付ける重要ステップ；修正および再試験サイクルに至ることがある

表IX.　フォールトインサーションテストのプロセスパターン

いわゆるフォールトインサーションテスト（*fault-insertion testing*）、*このテストは軽減ソフトウェアおよびハードウェアを強制的に機能させるもので、その結果は通常、後に*

医療用具が安全状態で作動するというものである。オペレーティングユニットが意図的に組み込んだ障害を発見するまでにはかなりの時間を要する可能性があるため、大部分のテストは意図的に誘発したエラーのパスに従う(表IX)。例えば、試験者はスタックポインターを破壊するために組み込んだ障害の発生を待つのではなく、システムを一時停止させ、スタックポインターを破壊し、停止した実行状態からシステムをリスタートさせることができる。このようなテストは、医療用具の構造についての詳細な知識を有する状態で実施しなければならないため難しい。通常、改変された医療用具は外部の支援ハードウェアを使用することが必要とされる。ハードウェアおよびソフトウェア環境を変えると、通常は臨床使用に適さないユニットが得られる。現在、フォールトインサーションテストは、工学雑誌においてかなりの注目を集めている[8,9]。

障害とエラー



ソフトウェアの障害は開発プロセスの様々なポイントで生じる可能性がある。ソフトウェアの障害は、特別な瞬間に現れるいくつかの状況が組み合わさることによって誘発されるまで、休止していることがある。障害はエラー、またはコンピュータ環境の不正確な状態を招く。ハザードを予防する状態にマシンを移行させるハードウェアまたはソフトウェアの軽減メカニズムによって、このような状態になることもある。リスクマネジメントプロセスに欠陥がある場合、エラーが拡大し、結果的に医療用具の不具合および患者に対する潜在的に有害な作用をもたらす可能性がある。

特別な問題

COTS用途　医療用具へのCOTSソフトウェア内蔵が一般的になりつつある。これは、FDAの「Guidance for Off-the-Shelf Software Use in Medical Devices」という表題の文書案によって認められた[10]。迅速に市販を実現したいという要望に刺激され、将来的に製造業者による医療用具へのCOTSソフトウェアの組み込みは継続すると思われる。しかし、COTSソフトウェアは医療用具の安全性を脅かす可能性があるため、ソフトウェアのリスクマネジメントプロセスに従って検討されなければならない。

AMDアプローチをCOTSソフトウェアに応用することができる。最初に、開発チームはシステムにおけるソフトウェアの役割を認識し、文書化しなければならない。COTSソフトウェアの不具合はIRAに含まれる可能性がある。その特別な性質のため、開発チームがCOTSソフトウェアについて個別のIRAを作成することが妥当と考えられる。この分析では、使用されるソフトウェアパッケージの種類にかかわらず認められる従来の故障モードについて検討すべきである。例えば、ファイルシステムは、破壊されたレコードの復帰に失敗することがある；分析によって、破壊されたレコードに基づいて作動するときのシステム安全性に対する影響が示されなければならない。これが、safety-wrapper philosophyの開発につながることもある。この場合、COTSソフトウェアの不具合を発見し、システムをCOTSソフトウェアと切り離すための特別なソフトウェアが開発される。これには、保存前のレコードの内容に関するエラー検出コードの作成と、その後のこのコードと保存レコードに対して作成されたコードとの照合が含まれると思われる[11]。

COTSソフトウェアの広範囲にわたる検証は、開発チームがソースコードを知ることができないため困難であることがある。しかし、医療用具に特異的なコードとCOTSソフトウェアとのインターフェースにエラーを導入することによって、COTSソフトウェアの不具合に対するシステム安全性の感受性をテストから特定することができる。

ツールとリスクマネジメント　ソフトウェア開発は、ビジュアルデザインツールから欠陥の処理を追跡するデータベースのコンパイラーに至るまでの多数のソフトウェアツールに依存している。これらのツールに組み込まれた障害が、製造コードを破壊する可能性のあるエラーの原因となる可能性がある。これらの障害は革新的な組織が現在検討しているソ

フトウェアのリスクに相当する。組み込まれた障害に対処するために以下のステップを実施することができる：

- 各ツールからの主たるリスクの脅威を認識する。少なくともすべてのツールおよびそのリスク原因への寄与を考慮する初期リスク分析を実施する。
- ツールからのリスクの削減におけるすべての検証プロセスの有効性を考慮する。典型的なアプローチは、決定操作者（decision operator）にとって欠陥のあるコンパイラーが生成する欠陥のあるマシンコードが構造試験によって発見できるかどうかを考慮することであるかもしれない。ツールがセイフティクリティカルソフトウェアに及ぼす影響に特に配慮しなければならない。
- 各ツールとともに用いられる限定的な一連の機能を定義する。ツールのすべての機能が使用されることはまれである。このステップの実施は不可欠である。
- おそらくは典型的な作業で構成される単純な試験を用いて、新たにリリースされた各ツールについて受入れ試験を実施する。欠陥が発見された場合、開発チームに連絡する。
- ソフトウェアのベンダーと直接連絡をとり、すべての既知の欠陥について理解する。一部のベンダーはこの作業をサポートするウェブサイトを開設している。確実に開発チームの全員が欠陥について認識しているようにする。既知の欠陥を誘発する可能性がある使用を発見するためのコード検査チェックリストを調整する。

結論

ソフトウェアのリスクマネジメントは、ソフトウェアを用いる医療用具数が増加するにつれて、ますます重要になりつつある。言い換えれば、このことによって、医療用具の安全性におけるソフトウェアの役割を特定する特異的活動が必要となる。この役割を認識するには、原因を、患者、操作者、臨床環境に対するリスクを軽減する軽減メカニズムと組み合わせる、確実なエンジニアリングが必要である。さらに、プロセスステップをパターンにあてはめることができ、反復した原因と軽減策のパターンの適用によって再利用と有効性が達成される。最後に、ソフトウェアのリスク分析を拡大して、COTSソフトウェアおよび開発に使用される他のツールの役割を明確にし、分離することが重要である。