WHITEPAPER

Making Safety Beautiful:

機能安全と Qt

Contents

はじめに

機能安全要件の重要性が、現在、数多くの分野で高まっています。高速道路や病院、工場など社会基盤となるものが、より一層安全になることは誰にとっても望ましい話です。こうした社会インフラの運営基盤となるシステムの安全性を向上させる製品を設計、製造するには、一貫したプロセスや標準化が役に立ちます。しかしながら機能安全の保守的な方法論は、ソフトウェア開発の標準プロセスの様々な側面とは相いれないことも少なくありません。開発を加速させる最近のツール群、ソフトウェアの適応を可能にする動的フレームワーク、そして魅力的、現代的かつ更新可能なUIを求める製品要件。実際のところ、これらについて機能安全の観点からは許容されない手法に頼っていることもしばしばです。

我々The Qt Companyでは、この2つの世界が交差する場面が増えていると実感しています。エンドユーザーは強化された製品が、自分のスマホ並みに便利で、自由で、使いやすいことを求めています。一方、規制当局や企業の安全責任者は、各製品が可能な限り安全であることを求め、ほとんどのUIフレームワークにとって満たすことが難しい基準に適合することを要求します。では、機能安全を実現しながら、時代に合った魅力的な製品を生み出すのは可能なのでしょうか。

可能である、というのが我々の答えです。ただし当然ながら、それはさらなる計画と作業を要し、様々な落とし穴を回避するために、一定の知見も必要となります。本ホワイトペーパーでは、現代的でダイナミックで機能的なUXを、機能安全対応の製品に組み込む方法について取り上げます。具体的なUXフレームワークとしてはQtを使いますが、できる限り一般的なアドバイスについても触れていきます。

機能安全とは

まずは「機能安全とは何か」を、簡単に説明します。
機能安全の基本的な目標は、故障の影響を低減するか、可能な場合には故障をなくすことによって、許容できない危害が人に及ぶことを避けることです。これは（安全のエキスパートの方々には単純すぎる分類で申し訳ないのですが）、4つの柱に分けることができます。

1) 障害を回避する

すべての障害をなくすことが目標ではありません。よほど些細なソフトウェアでもない限り、それは不可能だからです。むしろ、システム的な障害を回避できるときは回避し、回避できないときには適切にコントロールすることを検討する方が賢明であり、それを目標とすべきです。例えば、動的メモリ割り当ては失敗することもあるため、機能安全を確保するためのアプローチでは、メモリを静的に割り当てます。機能安全対応システムでは偶発的な障害や故障も、回避またはコントロールしようとします。このため、ソフトウェアとハードウェアの両方で冗長性を持たせたり、キープアライブやハートビート等の仕組みを導入してソフトウェアが正しく動作していることを確認できるようにしたりします。

2) リスクを管理する

リスク管理は、リスク評価を通してリスクレベルを把握するところから始まります。つまり、コンポーネントが機能しなくなった場合に最悪どのような事態になるかの見極めです。これは、「怪我の程度（軽傷～死亡）」「発生頻度（めったにない～常に起きている）」「回避できるか（できる～回避不可能）」の3つについてリスクを数値化することによって行います。例えば列車を制御する機構が、どの分岐点でも脱線を生じさせる可能性があるとします。これは程度の重い、頻繁に発生する、回避不可能なリスクであると判定され、故障を低減するための思い切ったリスク削減策を要することになります。一方、トナーカートリッジ交換時に手をすばやく動かさないと小さな切り傷を負う原因になる障害は、程度の軽い、めったにない、回避可能なリスクです。リスクレベルの把握は、リスクをどのように管理していくかにとって非常に重要です。リスクレベルは産業分野によって安全度水準（SIL：Safety Integrity Level）やクラス等で詳しく定義づけられています。

3) 一貫性を維持する

初期のセーフティクリティカルシステムは、機能安全規格が定められる前に作られました。つまり、決まったルールに従わずに安全なシステムを作ることは不可能ではありません。しかしながらソフトウェアエンジニアたちは、安全なソフトウェアを構築する上でやるべきこと、やってはいけないことを他者の間違いから学び、そこにエキスパートたちの手によってベストプラクティスを織り交ぜた各種規格が生まれました。どの業界にも、安全なソフトウェアを構築するための独自のプロセスがありますが、繰り返し可能で一貫性がある点ではどれも共通しています。ルールに縛られないカウボーイ・コーダーでは、安全なソフトウェアは作れません。

4) 最初から安全を組み込む

ソフトウェアは最初から安全を考慮して設計される必要があります。後付けでは、真に安全なシステムを構築するのは極めて難しくなります。一部コンポーネントが（ほぼ）フェイルプルーフであることが必要な場合はシステムの特性も決定的に異なるため、安全要件をアーキテクチャや設計の段階で考えに入れることが重要です。機能不全が許されないシステムでは、たまに単発的に故障することが許容される（時折再起動すればよい）システムと比べ、グラフィクスサブシステムへのアプローチは全く異なるものになるはずです。

機能安全規格と各種業界

安全の世界は、略語や数字から成る様々な規格であふれています。どの業界にも当てはめることができる、ひとつの基準に統一されていたらよいのに、と思いませんか。実際には、医療機器はIEC62304、自動車はISO26262と使われている規格は異なりますが、その多くが安全規格の草分けである産業オートメーションの安全規格、IEC61508と親戚関係にあります。

IEC 62304（医療）は、医療機器用ソフトウェアのライフサイクル・プロセスを規定しています。医療機器業界ではIEC61508認証は求められませんが、安全なシステムを構築する上で考慮すべき実践的な点が多く含まれているため、IEC62304に加えてIEC61508も参考にするのもよいかもしれません。

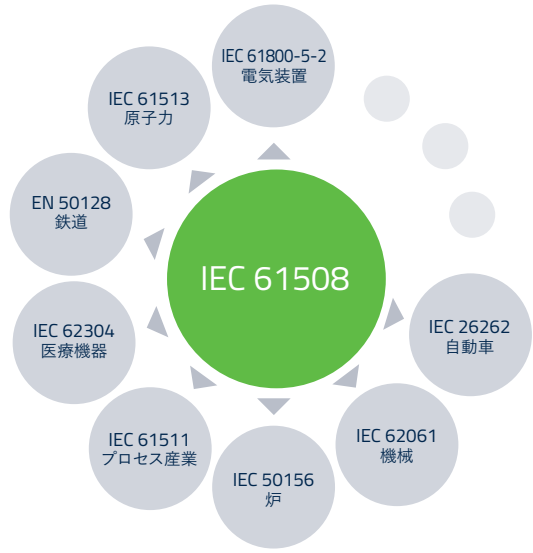


図1: IEC 61508と関連規格

ISO 26262（自動車）は乗用車用の機能安全規格で、IEC 61508から派生したものです。乗用車の設計と製造に関する独自の項目と、乗用車に独特なライフサイクルの特性を含めるよう変更が加えられています。ISO26262は乗用車用ではありますが、商用車やオフロード車でもよく使用されます。

安全度水準

機能安全規格のほぼすべてが、故障リスクをいくつかの安全水準に分類しており、それぞれの水準でソフトウェアに対する要件が異なるという重要な側面を持っています。例えばIEC61508では、対象機器に対する作動要求が高いのか（ほぼ継続的に使用される）、低いのか（最高でも年に1回ほどの使用）で水準を分けています。

作動要求が高い場合の安全水準は次のように規定されています。

- 安全水準度（SIL）4 – 1時間当たりの故障確率が10^{-9}～10^{-8}、すなわち約11,400年間の作動中に1回の故障
- SIL 3 – 1時間当たりの故障確率が10^{-8}～10^{-7}、すなわち約1,140年間の作動中に1回の故障
- SIL 2 – 1時間当たりの故障確率が10^{-7}～10^{-6}、すなわち約114年間の作動中に1回の故障
- SIL 1 – 1時間当たりの故障確率が10^{-6}～10^{-5}、すなわち約11年間の作動中に1回の故障

ISO26262は元はIEC61508に由来しますが、安全水準に関しては別の定義を使用しています。ISO26262の安全水準はIEC61508のようにその内容を規定するものではなく、目標志向の規定になっており、危害度、発生頻度、制御できる度合いの3つの要素を組み合わせてASIL（Automotive Safety Integrity Level）としています。このように定義が異なるため、ISO26262のASILとIEC61508のSILは1対1の関係にはありませんが、大まかに関係付けることは可能です（表1参照）。
ただし実際問題として、この2つの規格は機能安全用のソフトウェアライブラリを両者間でそのまま相互利用できないということを意味します。産業オートメーション分野と自動車分野の両方に対応するには、両方の認証が必要だからです。

	IEC 61508 産業 オートメーション	ISO 26262 自動車
最も高い安全水準	SIL 4	–
	SIL 3	ASIL D
	SIL 2	ASIL B/C
最も低い安全水準	SIL 1	ASIL A
安全要件なし	–	QM (品質管理)

表1　安全度水準の大まかな比較 [1]

1) From Safety Integrity Level to Assured Reliability and Resilience Level for Compositional Safety Critical Systems, Eric Verhulst, Altreonic NV

機能安全とQt

機能安全の基本をおさらいしたところで、次に機能安全と、数多くの組み込みデバイスのUIに使われているQtをどのように組み合わせることができるのかを見ていきます。機能安全対応システムでQtを使用できるか否かを判定するため、The Qt Companyでは認証機関VTT Expert Servicesと共同で2つの調査研究を実施し、製品の中の機能安全コンポーネントを全体のシステムから切り離すことでQtのフレームワークを認証できるという結論に達しました。機能安全コンポーネントは隔離された環境内で実行し、重要なディスプレイのグラフィックス等、認証が必要なコード部分に使用します。QtベースのUIを含むそれ以外すべてについては、機能安全対応でない部分を使用します。もちろん、機能安全には対応していなくとも頑強で回復力が高い場合もあるかもしれませんが、認証は必要でないため、機能安全対応部分と同様の厳格な規則に従う必要もありません。ただし、両者の間には、このような分離を考慮した上での十分なシステム状態の共有がなければなりません。

調査研究では、システムの機能安全対応部分を抽出、隔離することが最も適切なやり方である理由も示しています。

- Qtシステムには、機能安全な環境に合うように後から変えることが非常に難しい部分がある（Qt Coreも認証を得るには大幅修正が必要）。これらのコンポーネントを取り除くと、大幅に軽量化され、価値も下がったフレームワークが残ることになる。
- Qtは、機能安全対応システムに適しているかが疑わしい部分も多いモダンC++の機能をかなり使用するため、コードの大きな部分が悪影響を受けることになる。
- Qtの動的オブジェクト、ポインタ、自動型変換を削減、削除するには、たくさんの変更が必要。これらの変更はAPIを根本から変えてしまうことになり、認証されるQtフレームワークは、現在のQtとは大幅に異なるものになる。
- APIに対して必要となる変更は、既存のコードやライブラリを使えなくしてしまうため、新しいAPIをエンジニアが正しく使えるように、ドキュメンテーション、研修、資料も新たに必要になる。
- ソースをフォークさせた場合、「認証済みQt」を標準的なQtのベースラインと同期させていくことが不可能に近い。頻繁な機能追加と高いリリース頻度も併せて考えると、安全対応のQtフレームワークが通常のQtからますます離れ、メンテナンス工数も2倍かかる。

最後に、機能安全はQtの元の設計基準には含まれていなかったため、機能安全の基準を満たすためにはフレームワーク全体について再評価、ドキュメンテーション、修正が必要となり、大量の作業が発生することが考えられます。

Qt Safe Rendererの登場

ありがたいことに、ほとんどの機能安全対応システムでは、Qtほど複雑なソフトウェアを実行させる必要はなく、もっとよいアプローチを使えます。理想的としては、安全対応用に切り離されたQtシステムはUIの機能安全に関わる部分だけを扱い、安全要件を課せられないソフトウェア本体と安全にやり取りし、そして安全性を最初から考慮して設計されている、ものです。故障する危険性を少しでも減らすため、必要最小限の大きさで、なるべくシンプルなものがよいですが、必要以上に小さかったりシンプルであってはなりません。また、通常のQtの世界の中で既に存在するツールやフレームワークをできるだけ活用すべきです。

以上が、機能安全対応システムでのQtアプリケーション作成のために特に用意された、Qt Safe Rendererの基本的な設計要件です。Qtシステムの本体とセーフティークリティカルな部分を切り離すため、機能安全認証がソフトウェア開発の主要部分に与える影響も最小限に抑えることができます。

一言でいうと、Qt Safe Rendererはインジケータ、警告、アラート、ポインタ等のシンプルなグラフィックスを表示させるためのコンポーネントです。本体のQtアプリケーションとは別のコンテナに隔離され（ハイパーバイザー環境等）、ハードウェアレイヤまたは安全認証済みのコンポジターで本体アプリケーションの上にイメージをオーバーレイします。Qt Safe Rendererは機能安全認証に必要なエビデンスを提供できます。本体のQtアプリケーションの実行を監視し、正しく機能していない場合は再起動させます。

Qt Safe Rendererはフェイルプルーフで100%検証済みのコードパスを持つように設計されているため、UIに項目を表示する方法の選択肢がどうしても制限されます。一般的に安全システムで最も必要とされる静的なビットマップの表示と再配置の機能はありますので、トラブルや診断結果の表示には非常にうまく対応できます。これは医療機器、自動車、産業オートメーションシステムの機能安全部分を扱うには通常十分です。

機能安全対応Qtのアーキテクチャ

QtとQt Safe Rendererを併せ持つシステムのソフトウェアアーキテクチャでは、1つのCPUを仮想的に2つに分けます。使用するOS、求められる機能安全認証の水準によっては、OS内部でこれを行うことができる場合もあります。またそれができない場合でも、ハイパーバイザーを追加することによって、各部分のソフトウェアを互いに隔離することができます。

図2　Qt RTOSのみの環境による機能安全

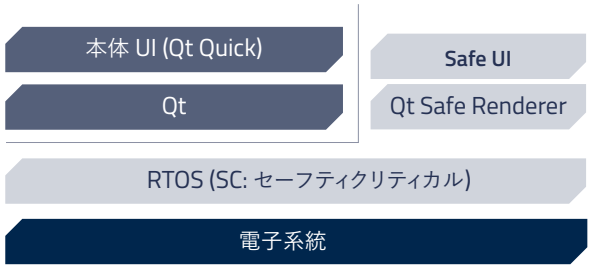
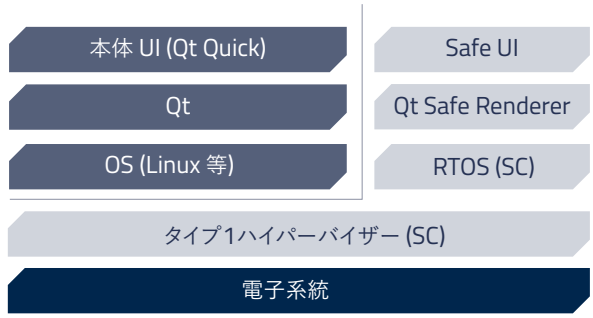


図3　Qtハイパーバイザー環境による機能安全



いずれの場合も、目的は機能安全対応の限定的な部分で認証を取得し、それ以外は標準的な手法や既存の方法論を用いて構築することです。
この2つのアプローチを次に示します。

1. 認証OS – QNX Neutrino RTOS Safe Kernel、Green Hills INTEGRITY RTOS等、安全認証を取得できるOS（通常はリアルタイムOS）。リアルタイムOSの役割は、主に初期実行時のメモリ割り当て、タスクスケジューリング、グラフィックスのレンダリング等、Qt Safe Rendererを読み込んで実行するためのサービスを提供することです。

2. Qt Safe RendererとSafe UI – 機能安全要件の対象となるシステム部分のグラフィックスのレンダリングを扱うコンポーネント。名前が示す通り、Safe UIは機能安全対応のUIを実現するためのアプリケーションで、Qt Safe Rendererは、Safe UIに代わり実際のレンダリングを行うエンジンです。Qt Safe Rendererは本体アプリケーションの状態の監視も行い、アプリケーションが正しく動作していない場合はアプリケーションを再起動します。

3. OS – 本体のQtアプリケーションを実行するOS。このOSは機能安全認証の必要がないため、Linux等、安全に関してそれほど強固なお墨付きがないものでもかまいません。とはいえ、安全対応のOSと同じものであってはいけない理由もありません。同じであった方がコンポーネントの種類も少なくなり、全体のシステムもシンプルになります。本体OSはメモリ管理、ファイルシステム、スレッド処理、同期プリミティブ、ドライバ等、本体アプリケーションで必要な標準サービスをすべて提供できる必要があります。

4. Qt – 本体UIが使用するQtフレームワークライブラリ。アプリケーションが実行時に必要とするすべてのQtコンポーネントを含みます。

5. 本体UI – 安全以外の主要機能を含む本体アプリケーション。本体UIアプリケーションに異常終了や誤動作がないかをQt Safe Rendererが監視し、必要に応じて対応措置を施せるように、Qt Safe Rendererに定期的にハートビートを送ります。このアプリケーションは認証を取得する必要はありませんが、可能な限り信頼性が高く、フェイルプルーフであることが重要です。機能安全対応部分と同じ、または同様のプロセスを使って開発することを検討してもよいかもしれません。

図2のアーキテクチャは、初めから安全認証を受けたOSを使用する場合にはよいのですが、もし対象システムがLinux等の認証取得ができないOSを使用する設計になっている場合は、タイプ1ハイパーバイザーをシステムに追加し、アプリケーションOSと安全OSを確実に隔離して、認証を受けやすくする必要があるかもしれません（図3参照）。これは、機能安全について高い水準の認証（ASIL D等）を狙う場合にも有効な方法です。この場合、ほとんどの部分は図2と同じですが、明らかに追加しなければならないものがあります：

6. 認証済みのタイプ1ハイパーバイザー – 安全対応と非対応のそれぞれの環境が、隔離されたコンテナで同一のプロセッサ上に存在できるようにするソフトウェア。ハイパーバイザーでは、この2つの異なる環境がCPU、GPU、メモリ、フラッシュ、周辺機器等の同一のリソースを共有できるように、ハードウェアからのサポートも利用します。完全に隔離、保護された安全システムの開発に最適なため、認証も取得しやすくなります。

デュアルプロセッサを使用した安全対応

機能安全なアーキテクチャのハイパーバイザー使用モデルの代替案として、図4に示すように、別個のプロセッサを使用する方法があります。

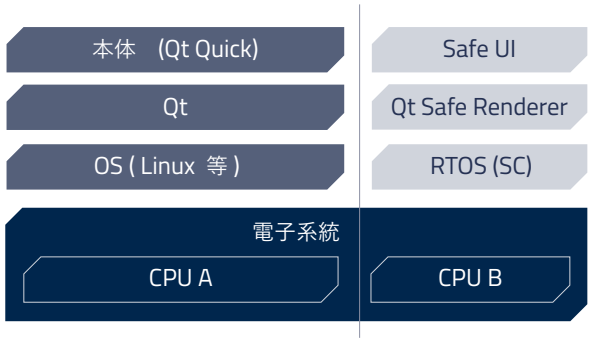


図4　2個のCPUを使った機能安全

このアプローチには多くのメリットがありますが、その中でも最大のメリットのひとつが、ハードウェアを通して機能安全をほぼ完全に隔離できることです。セーフティークリティカルなコード部分は通常は範囲が限定的なため、必要な機能のみのシンプルなRTOSが稼働する小型のマイクロコントローラで処理し、システムの残りの部分には機能をフル装備したCPUとOSを当てる、という設定を組むことができます。ソフトウェアのセーフティークリティカルな部分が保護されることが保証されれば、2つのCPU間でRAMや不揮発性ストレージ、特定の周辺機器等のリソースを共有することもできるかもしれません。

安全対応部分を分離したQt Systemの構築

Qtのランタイムシステムに新しく安全対応のパーティションが導入されたことに対し、「現行のQtのワークフローではどうなるのだろうか?」という疑問を抱く方がいるかもしれません。ここではQtの典型的なツールチェーン内の要素を取り上げ、それがシステム内の機能安全対応部分とQt Safe Rendererと、どのようにインターフェースするのかを見ていきます。

図5　Qt Safe Rendererで使用するためのISOアイコンをQt Quick Designerに追加

- **Qt QuickとQt Quick Designer**

Qt Quick(とQML)は、システムで宣言型のUIを作成できるようにします。Qt Quick Designerで作成し、QMLインターフェースで使用されるインジケータのアセットは、抽出してQt Safe Rendererのバイナリに組み込むことができます。つまり、本体のUIの作成で使用したツールチェーンやワークフローは、Qt Safe Rendererのアセットにも利用できます。

- **Qt for Device Creation**

Qt Safe Rendererのビルドタイムツールは、標準のQtまたはQt for Device Creationと統合できますが、機能安全が必要になるのは主に組み込みシステムであるため、ほとんどの場合はQt for Device Creationの方が望ましいでしょう(Qt Automotive SuiteはQt for Device Creationとバンドルされているため、自動車関連のデザインも同様)。

- **Qt 3DとQt 3D Studio**

アプリケーションで3Dレンダリングが必要な場合はQt 3D Studioでアセットを開発し、Qt 3Dでランタイムにアセットを使用します。QML同様、Qt Safe Renderer もビルド時にQt 3D Studioのツール群と接続・統合して2Dのアセットをインポートできます。こうすることによって、二重にリソースを作成することなく、同じツールを本体のUIと機能安全対応部分の両方に使用できます。

- **Qtフレームワーク全体**

Qtそのものは機能安全対応の環境で実行することはできませんが、本体アプリケーション用の、信頼性の高い、安定した、実績あるUI開発プラットフォームとしてシステム全体にわたって機能します。インジケータのアイコンの位置や表示を制御するためにQt Safe Rendererとやり取りするモジュールも、このアプリケーションの一部になります。

各領域における機能安全対応Qt

ここで提案しているシステムの分離は、各種問題領域でどのように活用できるのでしょうか。機能安全を必要とする3つの分野を選び、使用事例をお伝えします。

自動車

自動車では、機能安全対応Qtは主にデジタルメーターで使用されます。エンジンの回転数が分かっても人の命を救うことに直接的にはつながらないかもしれませんが、ブレーキシステムに障害が起きていることが分かれば助かる命もあるかもしれません。自動車には他にもABS、ステアバイワイヤやドライブバイワイヤといった自動車における電子制御システム、エンジン制御装置等、安全を要求されるモジュールが数多くありますが、これらにはGUIがないため、残念ながらQtの出番がありません。しかし自動車内で画面がある他の場所として、インフォテインメント(ナビゲーション)システムがあります。通常、機能安全要件の対象とはならないものですが、対象となった場合にはHMIで安全が重要な部分について、Qt Safe Rendererを使えます。

従来の運転席には、エンジン、ブレーキ、エアバッグ等の障害を示す警告灯が個別にありましたが、最近では運転手に対するインターフェースがひとつのLEDまたはOLEDディスプレイにまとめられている設計が多くなっています。つまり、機能安全であるためには、このようなデジタルクラスタが、主なインストルメントクラスタのアプリケーションとは別に制御され隔離されていることが保証されなければなりません。Qt Safe Rendererは自動車のインストルメントクラスタ向けに理想的に設計されており、インジケータは完全に隔離されたコンテナのコードで表示される一方、計器の表示レイアウトを変えられるよう、再配置は本体アプリケーションで行えます。

従来のアナログクラスタを画面上に再現することで、自由に構成を設定できるゲージ、3Dでレンダリングする複雑なデジタルゲージ、3Dで回転する自動車のモデル等、これまでにはなかった自由度でのデザインが可能になり、より魅力的、かつ正確に運転手に情報を伝えることができるようになりました。Qt 3D Studioで作成する場合でも、2Dのインジケータのアセットを必要に応じてQt Safe Rendererと併せて使用することができます。

本体アプリケーションは(Qt 3DまたはQt Quick/QMLを通して)ゲージを描画するだけでなく、自動車のバスインターフェース(CAN／MOST)、ハンドル制御、インフォテインメントシステムとの接続・統合しての利用も可能です。通常はQt for Device CreationまたはQt for Automotive Suite(実際にはQt for Device Creationの上位セット)を使用して構築します。

医療

患者へのリスクが最も低い医療機器を、クラスIの機器と呼びます。ただし、この分類の詳細は国によって多少異なります(下の表を参照)。クラスIの例としては、歩行分析器、サーモグラフィカメラ、血圧計等、表示機能が故障しても患者へのリスクがない(または問題にならない程度である)機器があります。

	ヨーロッパ	米国	カナダ
最も高い安全水準	クラス III	クラス III	クラス IV
	クラス IIb	クラス II / III	クラス III
	クラス IIa	クラス I / II	クラス II
最も低い安全水準	クラス IIa	クラス I	クラス I

表3　医療機器に関する各地域の安全分類の比較[2]

クラスIIの機器(ヨーロッパにおけるIIa/bも含む)は、患者に対するリスクが高くなります。このクラスの機器が機能しない場合、患者に危害が及んだり、命に係わる病状が診断されなかったりする可能性があります。クラスIIの機器の例としては、心電計、脳波計、超音波装置、負荷運動モニタ等があります。自動車の場合と同様、これらの機器には極めて重要なモニタリング機能があり、安全なコンテナで実行し、危険が生じた場合にはあらかじめ用意されたアイコンで警告を発することができるようになっています。例えば超音波装置の場合、本体アプリケーションで白黒の超音波画像を表示する一方、異常を検出した場合には安全対応部分で大きな赤い「!」マークを表示する、ということができます。

クラスIIIの機器は、障害が起きた場合のリスクが最も高いものです。厳しい規制がかけられており、求められる認証も最高レベルです。クラスIIIの医療機器は多くの場合、生命維持機能を提供するもので、使用を誤ると重傷や死亡に至る危険があります。例としては輸液ポンプ、埋め込み型神経刺激装置、埋め込み型ペースメーカー、自動除細動器などが挙げられます。埋め込み型の機器の場合は画面はありませんが、Qtが稼働する外部制御装置から設定を行えるケースがあり、その場合、システム全体がクラスIIIの機器に該当します。

[2] Medical device regulations, classification and submissions, MaRS Discovery District.

これらの機器を使用する看護師、医師、技術者たちは、より現代的で、応答性がよく、信頼性の高いユーザーインターフェースが1秒でも早く機器に備え付けられて欲しいと思っています。なぜなら彼らは、患者が生きるか死ぬかの局面で薬を投与、心臓への電気ショック、呼吸器その他の生命維持装置の使用等について素早く決断を下さなければならず、分かりやすく、直観的で、応答性がよく、信頼性の高いユーザーインターフェースを必要としています。医療機器における、エラーの発生しないHMIの重要性は、優れたユーザーインターフェースが見た目に美しいだけではないことを教えてくれます。それは、機能的にも決定的に重要な役割を果たすものなのです。

産業オートメーション

製品間で最も大きな違いが見られるのは、産業オートメーションの分野かもしれません。産業オートメーションは、他の領域に当てはめにくいものをすべて入れてしまえる包括的なカテゴリーだからです。産業オートメーション機器でLCD/OLEDディスプレイを必要とするものは多岐にわたります。研究所でのオートメーション、ロボット製造、ビルオートメーション、材料検査機械、CNC機械、倉庫管理システム、コンベアーシステム…、これらは、ほんの一部の例にすぎません。

これらのシステムの多くに共通して見られるのが、キャリブレーションや他の障害が重大な異常へとつながる恐れがあり、対応措置が施されなかった場合には危険な動作を引き起こす可能性がある点です。この意味では、これらの様々なシステムは自動車の例に似ています。適切に配置された目立つエラー表示があれば、オペレーターが組み立てラインの停止や原料供給の中止等の適切な措置をとり、万が一誤動作があった場合の障害の連鎖や怪我の発生を防止できます。重大なエラーを表示する必要のあるシステムは、他の例で取り上げたものと同様にシステムをパーティショニングすることによって対応できます。隔離された安全対応パーティションまたは仮想マシンでQt Safe Rendererを実行し、安全認証を必要としない残りのUIは安全対応パーティションの外に置きます。

機能安全の課題に取り組む

認証プロセスを進める際に知識やアドバイスを求める先として最もよいのは、安全監査を実施する監査企業です。一方、機能安全プロジェクトを開始する前の段階で、特にそれが初めてのプロジェクトの場合は、疑問が無数に湧き上がってくるかもしれません。どのような場合でも専門家に相談することをお勧めしますが、今後の方針を決める際に考慮すべき点を次にいくつか挙げます。

- 機能安全対応の製品は、最終的に費用節約につながる可能性があります。機能安全対応の製品を開発するためにはツール、教育、エンジニアリング、ドキュメンテーション、プロセスにかなりの投資が必要ですが、完成すればこれまでより信頼性の高い製品を作っているのだという安心感を、会社全体で持つことができます。機能安全は製造物責任訴訟に対して一定の保証にもなり、特に複数マーケットを対象にしている場合は法規制の順守の簡素化にもつながります。また、初期段階でソフトウェアの欠陥をなくすことで、コスト削減にも大きく貢献できます。欠陥当たりのコストの総合分析（“A Short History of the Cost Per Defect Metric”, Jones, 2012）によると、優れた品質のソフトウェアは、平均的な品質のソフトウェアより、その寿命を通してコストを33%削減できるとしています
- ソフトウェアのドキュメンテーションは極めて重要です。きちんと整備されているようにしましょう。ドキュメンテーションが十分に揃っていないことは、驚くほど多くの認証取得失敗の原因となっています。ある調査（“Analysis of Premar- ket Review Times Under The 510(K) Program”, US FDA, 2011）によると、認証申請の20%が、ソフトウェアのドキュメンテーション不足で却下されていると伝えています。ドキュメンテーションの重要性は、コーディングを始める前に理解しておくべきです。
- 機能安全では、どうすればよいUIがデザインできるかは語られませんが、悪いUIでは認証に悪影響が出ます。悪いUIはユーザーを躊躇させたり、誤操作の原因となったり、重要なデータの解釈を誤らせたり、誤った情報の入力を許したりします。ワークフローを設計する際には、安全要素を考慮した人的要因についても必ず検討しましょう。産業分野によっては、UIデザインについて標準ガイドラインを設けている場合もあります。例えば、医療製品の使いやすさについては、ISO/IEC 62366:2007とAAMI/ANSI HE75:2009を参照できます。
- IEC 61508等の規格では個々のコンポーネントについて認証を必要としないため、たとえ認証されていないものであっても商用オフザシェルフ（COTS）のソフトウェアを機能安全対応製品に使用することができます。ただし、機能安全規格ではCOTSのコンポーネントに関しても信頼性と適切性の証明を求めますので、あらかじめ認証済みであれば、証明提出が簡単になります。機能安全対応ソフトウェアのコンポーネントの認証には、特定の仕様、ドキュメンテーション、レポート、計画書を作成し、認証プロセスの中で提出する必要があります。明らかに、ベンダーのフルサポートがなければ、これを実現することはできません。

まとめ

政府当局はより安全な製品を要求し、エンドユーザーにとっては魅力的で直感的に操作できるUIが以前にも増して当たり前になっています。このため、組み込みシステムは機能安全でありながらユーザーにとって使いやすいものでもあるように作る、というのが最近のグローバルなトレンドです。

Qtは、機能安全規格への適合をますます求められている数多くの組み込みデバイスで、ユーザーフレンドリーなUIを提供しています。この事実と連動するかのように、システムを機能安全対応部分とそうでない部分とに分離することによってQtフレームワークの認証を取得することが可能となっています。そして、このプロセスを促進するため、セーフティークリティカルなUIが簡単に作成できるよう、我々はQt Safe Rendererを用意しました。

この機能安全コンポーネントによって、安全が要求される部分を本体Qtシステムから分離できるだけでなく、現行のQtワークフロー内で作業する際に機能安全認証がソフトウェア開発の主要部分に与える影響を最小限に抑えることができます。

機能安全対応に関してQtに詳しいパートナーが必要になった場合には、The Qt Companyが喜んでお手伝いいたします。

The Qt Company

The Qt Company develops and delivers the Qt development framework under commercial and open source licenses. We enable the reuse of software code across all operating systems, platforms and screen types, from desktops and embedded systems to wearables and mobile devices. Qt is used by approximately one million developers worldwide and is the platform of choice for in-vehicle digital cockpits, automation systems, medical devices, Digital TV/STB and other business critical applications in 70+ industries. With more than 250 employees worldwide, the company is headquartered in Espoo, Finland and is listed on Nasdaq Helsinki Stock Exchange. To learn more visit **http://qt.io**