オーム社/雑誌局

ISBN4-274-50113-2

C3055 ¥571E

9784274501135

1923055005716

定価（本体571円【税別】）

IPA® 独立行政法人 情報処理推進機構
ソフトウェア・エンジニアリング・センター

SEC−TN05−011

SEC BOOKS

組込みシステムの安全性向上の勧め（機能安全編）

独立行政法人 情報処理推進機構 ソフトウェア・エンジニアリング・センター 編

Ohmsha

SEC BOOKS

組込みシステムの安全性向上の勧め

（機能安全編）

独立行政法人 情報処理推進機構　ソフトウェア・エンジニアリング・センター　編

Ohmsha

SEC BOOKS

組込みシステムの安全性向上の勧め

（機能安全編）

独立行政法人 情報処理推進機構　ソフトウェア・エンジニアリング・センター　編

Ohmsha

はじめに

昨今、われわれの身の回りには、さまざまなシステムが溢れています。こうしたシステムのユビキタス化の進展とともに、システムに起因する事故やトラブルもまた数多く発生するようになってきています。一般にシステムは、われわれの生活を助け支援することが本来の役割であり、われわれの生活を豊かにするなど、多くのメリットをもたらしてくれます。しかし、昨今のシステム障害に見るように、時としてシステムはわれわれにさまざまな不利益をもたらしてしまう場合もありえます。システム開発に携わる技術者やメーカ、その経営者は、それぞれが携わるシステムが、人や社会に危害を与えることなく、システムを利用することによるメリットを生み出していくように振舞わなくてはなりません。本冊子は組込みシステムに求められる安全や安心といった視点から皆様の関係するシステムについて見つめなおしていただくことを目的として、その考え方などを整理したものです。

安全工学の世界では「ハインリッヒの法則」という有名な経験則があります。これは、「数多くの不具合や不安全行為が発生している場合には、その中にはいくつかの重大事故が含まれる可能性が高く深刻な事態につながる」という経験則で、ゆえに、「どのような小さな事故や不安全行為も見逃してはならない」というものです。組込みシステムが関わる世界においても、この法則は十分に成り立ちます。このため、組込みシステムに関わる技術者、経営者の一人ひとりが、まず身近なところから、システムの安全について考え行動することが重要になります。

また一方で、こうしたシステムの安全や安心、あるいは信頼性について

は、欧米などでも重大な関心ごとになりつつあり、システムの安全を確実なものとするための技術や開発過程での作業の進め方などについて、さまざまな議論が進められています。こうした背景の中、国際的にはIEC（国際電気標準会議）では、システムの安全に対する考え方や、そのための手法を標準化するための国際規格などの整備が続けられており、今や、システムの安全・安心については、開発者や利用者も含めたシステム関係者内できちんと考えられていて、あたり前という時代に入ろうとしています。このようにシステムの安全・安心については開発面での技術的な側面のみではなく、システムを取り巻くビジネス全体の問題としての認識を持つことも必要になってきています。

我が国でも情報システムに関する重大トラブルを契機として、2006年6月に経済産業省から「情報システムに関する信頼性ガイドライン」が発行されましたが、ここでも技術の視点、ビジネスの視点、技術者スキルの視点など多視点からのシステム信頼性や安全性に関する配慮が求められています。

本冊子は冒頭にも述べましたが、さまざまなシステムが氾濫する中で特に組込みシステムに目をやり、その安全性や信頼性を確保・向上していくことを目的に、そのための基本的な考え方や所作を整理したものです。編集に際しては、経済産業省と独立行政法人 情報処理推進機構　ソフトウェア・エンジニアリング・センター（IPA SEC）の連携のもとで組織した、「機能安全準備部会」のメンバが中心になって執筆しました。ぜひ、システムの安全・安心や機能安全の考え方を理解し、行動に結び付けるための最初の第一歩として、参考にしていただければと思います。

2006年11月　機能安全準備部会　一同

目　次

第1章　組込みシステムの安全性

1.1　システムの信頼性と安全性

(1) システム信頼性

われわれが身の回りで利用する製品などが故障した際に「この製品、信頼性低いんじゃない」などと口にします。「製品あるいはシステムの信頼性」とは、どのようなものなのでしょうか？　辞書をひくと「信頼」とは「信じて頼る」という意味があります。これを考慮すると「システムの信頼性」とは、“そのシステムを信じて頼れるかどうか”という性質と理解することができます。

通常、一つ一つのシステムは、それぞれが提供する機能や役割を持っています。システムのユーザが、そのシステムが提供する、こうした機能や役割を利用するときに、ユーザの期待どおりにシステムが動作すれば、そのシステムを「信じて頼る」ことができたこととなり、結果として、そのシステムの信頼性は、問題がないことになります。逆に、そのシステムが、当初、ユーザの期待したとおりに動いていたとしても、しばらくして動かなければ、そのシステムの信頼性は、問題があることになります。

これをもう少し厳密に定義すると、システムの信頼性は、以下のように定義することができます。

システム信頼性：機能単位が要求された機能を与えられた条件のもとで、与えられた期間実行する能力（JIS X 0014：情報処理用語－信頼性・保守性および可用性）

(2) システム安全性

一方でわれわれの身の回りのシステムの多くは、われわれの生活に直結している分、それらのシステムが誤動作などをした結果、ユーザであるわれわれ自身やその周囲の人々、あるいはもっと広範囲の人々に、被害を与える影響を及ぼしかねません。本来、われわれの生活や社会活動の助けとなるべきシステムが、われわれに害を及ぼすことは、あってはならないことです。しかし、現実には、残念なことに、システムの誤動作などに端を発する被害は後をたちません。この1年ほどをとっても、新幹線の自動列車制御装置の誤動作によって列車が緊急停止したり、証券取引所のシステム障害によって多大な経済損失を出すなど、事故の報告は、枚挙にいとまありません。システムは通常、それが動作することによって、ユーザその他になんらかの利益や効能をもたらすべきものです。

しかし、システムが、誤動作などをすることによって、ユーザに不利益や危害をもたらすこともあります。システムの安全性とは、システムが結果として、ユーザに対してこうした不利益や危害をもたらすことのない度合いをさします。

これをもう少し厳密に定義すると、システムの安全性は以下のように定義することができます。

システム安全性：システムが規定された条件のもとで、人の生命、健康、財産またはその環境を危険にさらす状態に移行しない期待度合い(JIS X 0134：システム及びソフトウェアに課せられたリスク抑制の完全性水準)

このようにみてくると、システム信頼性とシステム安全性は、その根本の考え方において、大きな開きがあることが、理解できるかと思います。

ただし、一般的には、システム信頼性が低いとシステムの動作中に誤動作をする可能性が高くなり、結果的にシステム安全性が損なわれる事態が発生します。

1.2　ランダム故障と決定論的原因故障

(1) ランダム故障

自動車では、新車を除くと2年に1回点検をする自動車検査制度が義務づけられています。なぜ、こうした2年に1回と義務づけられているのでしょうか？　自動車は、エンジン制御をつかさどるコンピュータプログラムなども搭載されていますが、その一方でタイヤ、エンジン、ブレーキやハンドルなど、さまざまなハードウェアユニットが搭載されています。例えば、タイヤを例にとると、走行距離が長くなるとタイヤは摩耗し、最後にはパンクしてしまいます。では、新品のタイヤは“いつパンクするのか”については、明確にいついつにパンクするとは明言できません。タイヤ自身の製造時のばらつきや、ユーザがどのような路面の道路を多く走行したかなど、さまざまな条件によって、タイヤの寿命は変わってきます。そのためタイヤのパンクという故障は、時間的には、無秩序に発生すると考えることができます。ただ、そうはいっても、ある程度の距離や時間を走行したタイヤは、パンクする確率が高くなることは、想像に難くありません。

さて、このタイヤの例を見てもわかるように、システムあるいはシステムの構成要素で発生する故障の中には、このように時間的に無秩序に発生するタイプの故障があります。これをランダム故障と呼びます。

ランダム故障：構成部品・機器などの多様な劣化のメカニズムのもとで時間的に無秩序に発生する故障(ハードウェアの劣化、偶発故障、単純作業におけるヒューマンエラーなど)

(2) 決定論的原因故障

数年前にソフトウェアの2000年問題として、世間は大騒ぎをしました。2000年問題にはさまざまなものがありましたが、その代表的なものは「年号を下２桁で管理していて、年号として00が入力された場合には、データ入力の終わりと自動判断する」といったタイプのプログラムでした。これはこうしたシステムが2000年まで利用されることを想定せず、開発した70年代、80年代には、考えもしなかった状況が起きようとしていたものです。

この2000年問題では、プログラム内の論理構造上、「年号として00年を入力すると必ずシステムが入力終了と判定してしまう」。ある意味で、すでにプログラムが作られた当時から、論理的には起こるべくして起こる故障でした。このように故障と呼ばれるものの中には、そのシステムや機能要素を作成する過程で、すでに故障の要因が作り込まれており、必然的に故障が起きるタイプのものがあります。こうした故障を**決定論的原因故障**と呼びます。

> **決定論的原因故障：設計過程、製造過程、運転手順、文書化などに直接関わり、これらの中で故障要因が入り込むことで、必然的に発生する故障。決定論的原因故障は、これらの過程でのアウトプットなどを修正することによってのみ除去することができる。（ソフトウェアの不具合、安全解析・安全管理の不都合、設計の誤りなどが起因する）**

このように考えると、ソフトウェアシステムにおける故障は、論理的には決定論的原因故障の範囲に属すると考えることができます。

(3) 組込みシステムの故障

故障には「ランダム故障」と「決定論的原因故障」の２種類に分けて考えることができます。さて、それでは、本冊子が取り上げる組込みシステムにおける故障は、どのように考えることができるでしょうか？

一般的な組込みシステムは、ハードウェアとソフトウェアという２つの大きな要素ブロックから構成されています。このうち、前者のハードウェアについては、さまざまなセンサやアクチュエータ、あるいは回路やマイコンなどが含まれますが、これらの多くは「ランダム故障」の範囲に入れて考えることができますが「決定論的原因故障」もあります。

一方、組込みソフトウェアは、基本的にソフトウェアとしての性質を有しますので「決定論的原因故障」の範囲に入れて考えることが一見、妥当のように思われます。

しかし、近年の組込みソフトウェアは、非常に規模が大きく、複雑に進化しています。そして機能的にも非常に多くの機能が搭載されていますが、そのシステムの一生の中で搭載された機能の全てが、同じような頻度で利用されるとは限りません。非常に良く使われる機能もあれば、一度も動作しない機能ユニットもあります。そして、どの機能をどの程度利用するかという点で考えてみると、これは特に明確に決められるとは限らず、大局的にみるとあたかも「**ランダムに実行される**」という形に限りなく近くなっています。

このように考えると、組込みソフトウェアの故障は、決定論的原因故障であることは間違いないものの、その発生の傾向としては、**限りなく「ランダム故障」に近い**ものとして扱う必要が出てきます。

1.3　安全度水準

(1) 安全度水準の背景

我が国の工業製品の特徴の一つは、高い品質レベルにあるといわれています。例えば「どのような状況下においても絶対に故障や事故を起こさない」、あるいはソフトウェアに関していうと「絶対にバグはあってはならない」ことが絶対条件のように求められています。

しかし、こうした考え方は最近の大規模複雑な情報処理システムや組込みシステムを前にして達成がむずかしくなっています。

これに対しプラントなどの産業分野では、事故発生などに関する確率論に基づいた考え方を採用する方向になりつつあります。その根本には、プラントなどで発生する事故には、軽微なものから重大なものまで、さまざまな程度のものがあり、かつ、それらはプラント構成要素部品の故障確率をベースに、それらの事故の発生頻度を管理し、最適な事故予防策を講じる考え方があります。こうした流れの中で確率論的な考えに基づいた障害発生への対処方法は、プラント業界のみならず、それ以外のシステムや電気計装装置などの分野にも、広がりを見せ始めています。

こうした考えを後押ししているものの一つがIEC 61508と呼ばれる規格です。この規格の中では、システムにおける障害や危機管理の目安として、**安全度水準**と呼ばれる考え方が要求されています。

(2) IEC 61508における安全度水準

IEC 61508では、システムに求められる目標とする安全機能が、どの程度の割合で機能不全にいたるかの許容程度を、安全度水準として管理することが、求められています。すなわち、通常、原子炉内には、100本以上の制御棒がありますが、安全性の視点からこれらが「どの程度の割合まで故障するのを許容するか」といった考え方にたって設計されています。IEC 61508における安全度水準は、SIL(Safety Integrity Level)という略語で呼ばれ、SILは、SIL 1からSIL 4までの4段階に区分されています。たとえば、SIL 1の場合、こうした故障が発生する確率は、10～100回の作業要求当たり1回以下でなければならないなどと、定められています。

一方、こうした故障の発生確率とともに、人間にどの程度の影響の事故が起きるかという視点もあわせて考える必要があります。IEC 61508ではこれに関しても、A～Dまでの4段階の被害程度を規定しています。

程度A：一時的な身体機能喪失

程度B：１名以上の深刻な傷害または１名の人命損失

程度C：数名の人命損失

程度D：かなり多数の人命損失

IEC 61508では、このようにシステムに求められる安全度水準と想定される被害程度を考慮し、その程度に応じたシステム障害の防止に向けた対策を実施するように求めています。

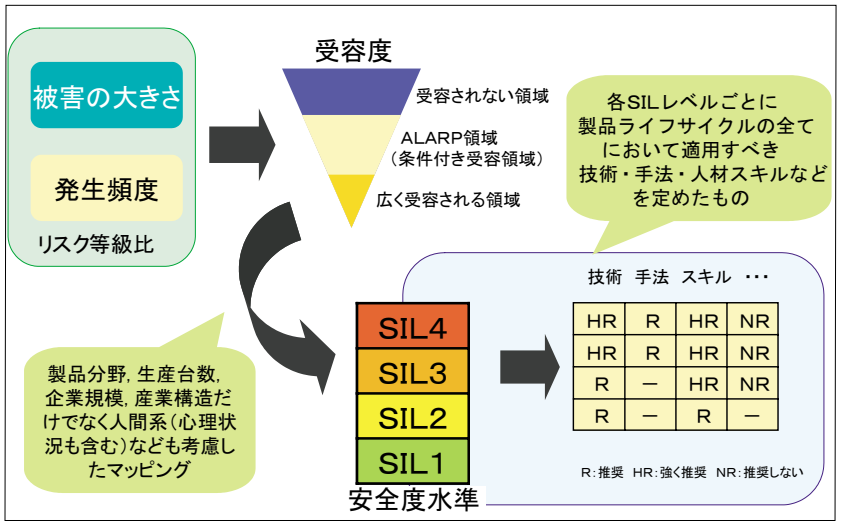


図1.1　安全度水準 SIL（Safety Integrity Level）とは

第2章　ソフトウェアの事故事例

ソフトウェアによる事故について、それがなぜ起きたかを分析・評価して、公開した例はあまり多くありません（表2.1 最近のソフトウェアによる不具合事例参照)。ソフトウェア製作の失敗の多くは、製作者の人的ミスであり、本人の不注意や管理不行き届きで終わってしまい、公開して、今後の役に立てようとする意識が低いのではないか、と思われます。

本章では、公開情報をもとに、ソフトウェア事故の例を紹介します。

本冊子が対象とする組込みシステムは、その構造上や利用状況（局面）において、さまざまな特徴を持っています。

① システムのユーザは、必ずしもシステムの専門家とは限らず、システムに不慣れなユーザが操作する場合も少なくない

② 組込みシステムの中には、不特定多数のユーザが利用するシステムなども少なくない

③ システムの構造上、さまざまな動作環境や条件化で動作することが求められ、それらの外部環境の変化などに対応した動作が求められる場合が多い

④さまざまな周辺機器と連動して動作する場合が少なくなく、限られた時間内での処理が求められることが多い

以下に示す4つの事例は、こうした組込みソフトウェア特有の性質を背景として、実際のフィールドで、不幸にもシステム障害という形でユーザに不利益をもたらしたり、あるいはその一歩手前の危機的状況を引き起こしてしまったものの例です。

表 2.1　最近のソフトウェアによる不具合事例

新聞掲載日	不具合発生機器	不具合の内容
2005年 6 月	ハイブリッド車	「走行中に突然エンストを起こす（米国)」「エンジン走行から突然モータ走行に切り替わる（日本)」というもの。車両制御ソフトの不具合が原因と考えられている。
2005年 6 月	携帯電話	「着信時に終了キーを連続して押した場合に、着信履歴が表示されない場合がある」「受信メールの送信者が実際の送信者と違う名前で表示される場合がある」というもの。
2005年 7 月	高温水供給式ガス給湯器	「出荷時に設定されている追い炊き時の沸き上がり温度を、ユーザ設定モードを使って変更し、沸かしなおしや追い炊きをした場合、浴槽内の湯が溢れたり、設定温度以上の沸きあがり温度になる」というもの。制御用部品のマイコンソフトの不具合に原因。
2005年 7 月	ETC（自動料金収受システム）	「首都高速の一部出口から発信される信号を読み取れず、特定区間割引サービスを受けられない場合がある」というもの。ETC 搭載ソフトの不具合に基づくもの。
2005年 8 月	26V 型液晶 TV	「リモコンで電源が入らなくなる」というもの。ソフトウェアの不備により、ロゴマークダウンロードの開始された 8 ／ 1 以降、リモコンで電源が入らなくなっていた。無償で点検・修理（ソフトウェアのアップデート）実施。
2005年 8 月	HDD レコーダ	「600GB の容量があるにもかかわらず、録画領域として500GB しか使用できない」というもの。HDD のファームウェアに起因する。修正ソフト開発に 1 ヶ月程度要するとのことである。修理の対応は保守サービス会社から技術者がユーザ宅を訪問し、無償修理を実施する予定。

ETC : Electronic Toll Collection
HDD : Hard Disk Driver

事例ー 1 ：ユーザビリティへの対応：ソフトウェアの対応が追いつかず
～「テラック25」放射線照射装置の事故～

事故の状況

1985年から1987年の間に、カナダ AECL 製の LINAC 加速器の異常によって、 6 件の被爆事故が発生し、うち 4 名が死亡する事故が起きました。事象は、オペレータが「弱い電子線」を表す“e”キーを押すべきところを間違えて「強い X 線」の“x”キーを押し、まちがえに気づき、これをカーソルキーを使って修正する場合、 8 秒以内に修正すると、ソフトウェアが正常に動作せず、高エネルギー放射線を直接患部に照射してしまったものです。

事故の直接的な原因

この事故は、ユーザが、直接システムに接する部分であるユーザインタフェース部分の組込みシステムの不具合が引き金になって発生したものです。このシステムに対しては、開発段階で、あらかじめ潜在危険分析がなされていたものの、ソフトウェアのミスについての可能性は、除外されていました。また、コンピュータエラーによる過剰照射の可能性は FTA（Fault Free Analysis：故障木解析）の段階でも分析されていましたが、コンピュータが間違った照射エネルギーを選択する可能性は、10～11という低い確率とされていました。(「SAFEWARE」、N. リバソン、1995年）

機能安全面からの教訓

機能安全の観点で検討すると、この装置では、ある程度のリスク解析がなされていましたが、ソフトウェアによる事故を想定していなかったことが問題といえます。ソフトウェアとハードウェアが連携して、一つの装置を作り出す場合、その両側面からのリスク解析が、必須要件であることを物語っています。また同時に、システムのユーザビリティという落とし穴についても、リスクとして認識し対処していく必要があるといえます。

事例ー２：不特定ユーザへの配慮：コンテキストの想定ミス
～都内ビルの回転ドアでの死亡事故～

事故の状況

2004年３月26日、都内のビルで、男児が自動回転ドアとドア枠に挟まれ死亡しました。人が近づくとセンサが作動してドアを停止するプログラムになっていましたが、センサ検知領域が、背の低い子供を検知できない範囲に設定されていたのが、直接原因とされています（朝日新聞、2004年３月27日）。

事故の直接的な原因

この事故の場合、防ぐことができた機会は何度もありましたが、機能安全の点から見ると、最初にリスク評価フェーズを着実に実行すべきであったと考えられます。すなわち、ビジネスビルでありながら、子供連れも来訪する状況（コンテキスト）を想定して、リスク評価を行っていれば、事故は防げたと思われます。そして、保守段階で、センサ調整などの部分改修を行う際の安全妥当性確認を十分に行うべきでした。

機能安全面での教訓

この事故では、事故後の調査で、初めて過去の事故情報が収集され公開されました。しかし、そこには過去にも類似の事故が、複数発生していたことが記載されていました。こうしたシステムや機器に関する危険情報収集は、機能安全管理の必須項目の一つであり、もしこれらが適切な時期に適切な形で公開され、それをメーカサイドで真摯に受けとめて安全対策を実施していれば、このような悲惨な事故は防げたと思われます。

事例ー３：システム動作環境への対応：リアクティブ性に起因する誤動作
～東海道新幹線 ATC のプログラムミス～

事故の状況

新型 ATC（Automatic Train Control：列車自動制御）装置のコンピュータが列車を緊急停止させた事故が2006年３月18日以来、19件発生しました。一つは、各新幹線に搭載した ATC 用コンピュータのメモリ不足（12件）で、先行している新幹線の速度が相対的に速いと、自列車の ATC の取得する相手位置の情報が増えます。この情報を格納しておく領域が不足した結果、コンピュータが異常と判断しました。もう一つは、ATC 用コンピュータの処理能力の問題（３件）で、先行列車と自列車との両位置情報を（線路上のトランスポンダから）0.1秒以内に受け取ると、コンピュータが処理しきれずに異常と判断したものです（日経コンピュータ、2006年４月12日）。

事故の直接的な原因

この事故では、列車の相対速度という要因をシステム設計段階で十分に織り込んで検討できていなかったことが、直接的な原因となっています。列車速度が上昇すれば、信号やデータを受け取る頻度が上昇し、搭載されるコンピュータの処理速度や、メモリスペースが追いつかなくなったと考えられます。組込みシステムの場合、動作条件が、わずかに変わっただけでも、その処理に大きな影響を及ぼすことが少なくありません。

機能安全面での教訓

この事故は、ATC の故障（ソフトウェアの不具合）によって列車が停止したため人身事故につながる危険側故障には至りませんでしたが、機能安全の要求事項であるソフトウェア設計製作段階でのライフサイクルの仕組み（フォールトアボイダンス技法とフォールトトレランス技法）を採用していれば、当該事故は防げたと思われます。

事例－４：システム動作へのタイミング対応：リアルタイム性に起因する誤動作
～エレベータでの死亡事故～

事故の状況

2006年６月３日、エレベータから自転車に乗ったままの高校生が出ようとしたところ、突然、上昇したエレベータに挟まれて死亡事故となりました。この事故の直接原因は調査中ですが、本来、扉が開いたままでは、上下動しないように、コンピュータが監視制御していなければならないはずでした。

この事故後、事故を起こしたメーカの他のエレベータを調査した結果、過去のプログラム欠陥が明らかになりました。扉が閉まった瞬間から0.25秒以内に「開」ボタンを押した場合、扉が開いた状態でカゴが昇降してしまう、という欠陥でした。当該欠陥は、すでに1993年に判明しており、同社はプログラムを修正しましたが、３基が修正リストから漏れ、６基は別の改修をした際、欠陥プログラムを誤って再び搭載してしまいました（朝日新聞、2006年６月17日）。

事故の直接的な原因

最近のエレベータシステムは、地震や火事への対応などを含めて、さまざまな動作モードが用意されており、そのシステム機能の７割近くは、異常処理対応機能といわれています。この結果、きわめて複雑な構造となっており、ボタン一つの押されるタイミングなどによっても、動作モードのコンフリクト（衝突）といった事象が発生し、誤動作につながる危険性があります。この事故も、まさにこうしたシステム制御の微妙なタイミングに起因する事故の可能性があり、いわゆる、組込みソフトウェア特有のリアルタイム性が深く影響しています。

機能安全面での教訓

この死亡事故の直接原因は調査中ですが、扉が開いたまま上昇したわけですから、安全系を構成するセンサ、コンピュータ、アクチュエータ（ブレーキなど）の組み合わせのいずれかの欠陥であることは明らかです。特に、機能安全の観点からは、現場の劣悪な環境で使用されるセンサ・アクチュエータなどの故障や異常を監視制御すべきコンピュータに対して、安全要求機能と、その安全度水準を明示し、当該要求事項を実施すれば、事故防止の一助となったと思われます。

一方、適切な保守がなされていなければ、事故につながる場合もあります。とりわけ、保守は人間が関与する部分ですので、悪意がなくてもヒューマンエラーは起こりえます。

したがって、保守段階でのヒューマンエラーを想定し、これを防衛するという、機能安全に基づいた設計を採用すべきでした。なお、別のエレベータで起きたプログラム欠陥については、保守フェーズにおける部分改修での機能安全管理がなされれば、適切な処理となったはずです。

第3章　安全のための開発プロセス

3.1　ソフトウェア開発プロセスと機能安全

(1) ソフトウェア開発プロセスとは

組込みソフトウェアに限らず通常、ソフトウェアを開発する場合には、本来、実施しなければならない作業があります。例えば、まず始めにどのようなソフトウェアを開発するかを明確にする必要があり、このための作業として要求獲得や分析、要求定義などを実施します。また、こうした要求事項をソフトウェアとして実現するためには、どのような造りにするかを考える必要があり、いわゆる、ソフトウェア設計という作業が必要になります。さらに、実現方式が決まった後には、それに従いソフトウェアをプログラミング言語などを用いて実装します。そして、実装されたソフトウェアは、当初の要求や設計に合致しているか、どうかを確認するために、テストを行います。

このように一つのソフトウェアを開発するためには、要求分析・定義、設計、実装、テストなど、さまざまな作業を行う必要があります。これらを整理したものを開発プロセスと呼びます。

(2) 機能安全を織り込んだ開発プロセス

ソフトウェアの安全性や信頼性を高め機能安全を実現するためには、こうしたソフトウェア開発プロセスに機能安全の視点を盛り込んでいく必要があります。すなわち、機能安全を実現するためには、要求分析・定義の段階、設計の段階、実装の段階、テスト検証の段階など、それぞれで機能安全を実現していくための作業項目を実施することが重要になります。

例えば、一般的な要求分析や要求定義では、システムとして実現する機能面の要求や性能などの非機能と呼ばれる側面の要求を分析したり整理したりします。これに対し、要求分析や定義の段階で、機能安全的な側面を加味しようとすると、システム障害時の想定被害や、その影響範囲、あるいはこれらのシステム障害の発生の可能性なども分析して、ソフトウェアの要求に反映させることが求められます。通常の機能要求や非機能要求を、このような視点からレビューし機能安全という視点で見たときに、これらの要求事項が適切であるか、どうか、などを確認することも求められます。

本冊子が対象とする組込みソフトウェアや組込みシステムは、われわれの日常の身の回りの、さまざまなところで利用されています。

そして、こうしたシステムやソフトウェアが障害を起こすと、その被害の程度や範囲は、非常に大きなものに発展する場合も少なくありません。このため、要求分析や定義の段階のみならず、開発のあらゆるプロセスで、機能安全的な側面からの工夫や確認を常に行う必要があります。

3.2　機能安全を実現する開発プロセス上のしくみ

(1) 要求・設計、実装への機能安全の織り込み

(a) 機能安全を織り込んだシステム要求

組込みシステムの機能安全を実現する上での最初の第一歩としては、システムに、どの程度の安全性や信頼性が求められているかを整理し、システムに対する安全要求を明確にすることが求められます。システムの安全要求の分析や検討については、対象システムのユーザやユーザのシステムへの接し方なども考慮する必要があります。さらに多くのシステムでは、障害が発生した場合、直接的なユーザ以外の周囲の人々にも広範な影響を及ぼす場合も少なくなく、これらの影響の広がりも考慮しておくことが求められます。

(b) 機能安全を実現するシステムの設計

システムに関する安全要求が明確になったとしても、それをいかにシステムとして実現するかがセットとして検討されないと意味をなしません。その意味では、機能安全を実現するためのメカニズムとして、システムの多重化をはじめとして、さまざまなテクニックが提案されています。また、ソフトウェアシステムにおけるエラー処理など、システム全体に共通する処理の切り出しと一元化など、基本的なシステム構築のノウハウなども存在します。

(c) 機能安全を実現するシステム実装

組込みシステムの場合、ハードウェアとソフトウェアが協調動作をすることが最大の特徴です。このソフト／ハード協調という中で、機能安全の要素を相互に補完実現しながら、それぞれを実装していくことが求められます。特に組込みソフトウェアで多用される C 言語による実装では、単純なコーディングミスや、言語仕様や利用するハードウェア特性に依存する誤り（ハードウェアとの相性）まで、さまざまなミスが入り込む余地があります。これらのミスを意識した実装方法をとることが求められます。

(2) 機能安全ファクタの織り込み確認

機能安全の実現に向けては、第一義的には上記のように要求・設計、実装といった、それぞれの作業の中で機能安全ファクタを織り込んでいくことが基本となりますが、一方で、これらが適切に織り込まれているかを確認することも必要です。その基本動作は、開発の節目節目で実施されるべきデザインレビューなどの局面で、それぞれの対象成果物が機能安全面を十分に考慮しているかどうか、を確認する必要があります。

(3) IEC 61508における安全ライフサイクル

機能安全に関する考え方や基本動作を整理した国際規格として IEC 61508

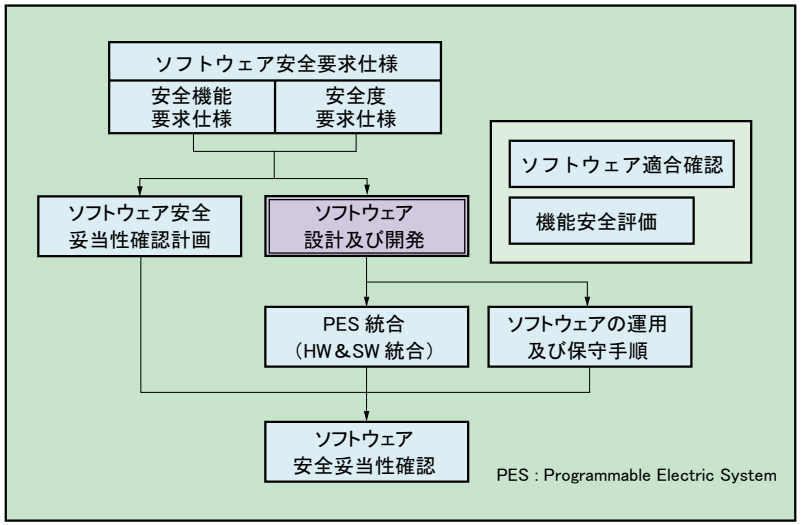
が規定されています。この規格の中でもシステムの機能安全を実現するための作業を「安全ライフサイクル」という形で整理してあります。ここでは詳細には触れませんが、前述したように、

① システムの利用シーンやユーザを考慮して安全要求を整理し

② それらを達成するための方式を考慮したうえでシステム構成要素に割り付けて

③ それらが適切に実現されているかを確認し、必要に応じて是正措置を講じる

といった安全性実現の基本概念は、大きく変わるものではありません。ただし、この安全ライフサイクルでは、システムの運用・保守や設置といった局面まで考慮する点で、システムの機能安全実現に向けて、より広範な活動を求めています。



出典　吉岡律夫：機能安全規格と適合認証、機能安全規格と IEC 61508とその認証、(独)産業技術総合研究所、産総研ワークショップ、2006年２月

図3.1　ソフトウェアの安全ライフサイクル

こうした考え方は、2006年夏に経済産業省から発行された「システム信頼性ガイドライン」の中でも、**情報システム全般に関して、その企画・開発から保守・運用にわたり関係者が信頼性・安全性向上に向けた活動を行う**ように求められています。

このガイドラインでいう情報システムの中には、組込みシステムも含まれています。また、これらのガイドラインの遵守を求められる対象である情報システム関係者とは、情報システムの企画・開発および保守・運用に携わる発注者、利用者、受注者、開発者および運用者を含んでいます。安全ライフサイクルを考える上での参考としてぜひ、一読することをお勧めします。

3.3　組込みソフトウェアの開発プロセス

組込みソフトウェアの開発では、ハードウェアとソフトウェアのすりあわせなどを含め、一般的な情報処理システムを開発する場合とは異なる作業や留意点があります。これらを軽視すると、システムとしての安全性や信頼性を損なうことになりかねません。しかし、組込みソフトウェアや組込みシステムの開発プロセスという点では、標準的なものやお手本となるものが整備されていませんでした。

SEC では、こうした状況を踏まえ、2006年10月に、組込みソフトウェアに関する開発プロセスのひな形として「組込みソフトウェア開発プロセスガイド　ESPR－Ver1.0」を策定しました。このガイドでは、組込みソフトウェア開発の要求フェーズからテストフェーズまでの各フェーズで実施すべき作業やその注意点、作業結果のまとめ方などを参考情報として整理してあります。

これらを参考に開発に必要な作業を整理し、それぞれの組織やグループの開発作業（プロセス）を見直していくことは、結果として開発されるシステムの機能安全を実現する上でも重要となります。

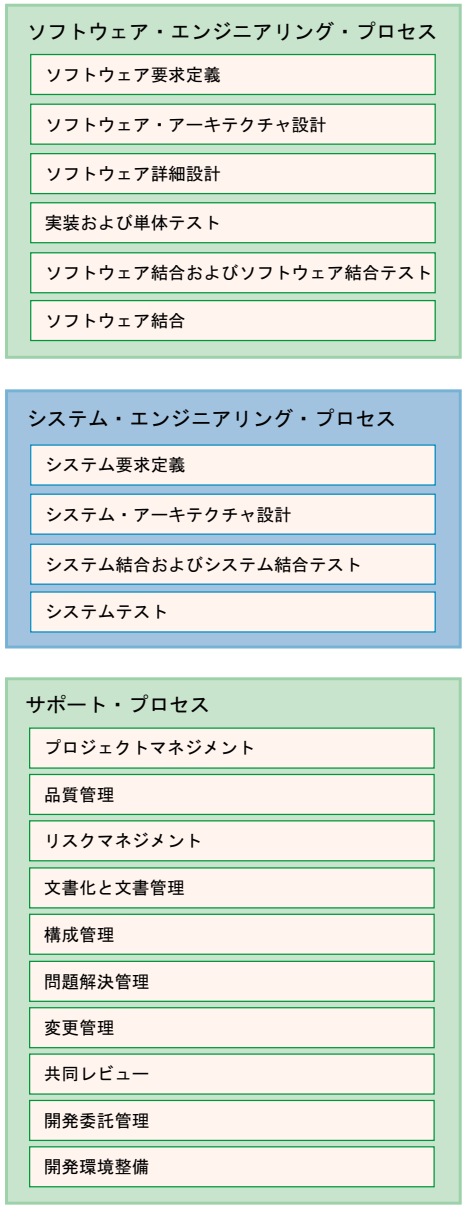


図3.2 組込みソフトウェア開発プロセスガイド（ESPR-Ver1.0）の構成

第 4 章　ハザードリスク分析

4.1　ハザードの考え方と対処法

(1) ハザードとは

ハザードとは通常、潜在危険と訳されます。これをソフトウェアシステムについて考えると、これらのシステムが誤動作などをして利用者や関係する社会などに危害や損害をもたらす場合に、それらの誤動作を引き起こす障害の芽と考えることができます。

機能安全の視点では、こうしたシステム障害を引き起こすであろう危険の芽を、どのように検知し、あらかじめ摘んでいくかが、きわめて重要な要素になります。これらのハザードを洗い出し、それらへの対応を考える活動の一つを、安全計画と呼びます。安全計画には、一度発生した障害や事故の教訓を、その先の開発に生かしていくことを目的とする事後安全計画と、新規に開発に取り組む際に安全面からの分析や検討を加える事前安全計画の 2 つがあります。

(2) 事後安全計画と事前安全計画

(a) 事後安全計画

事故が発生すると、事故原因を調査し、さらに再発防止のための対策を実施します。このような一連の過程を事後安全計画といいます。事後安全計画は、事故の再発防止によるシステムの安全性向上のために必須の条件です。事後安全計画は、原則として、類似のシステム開発などを前提として、同様の事故や障害が繰り返し発生するようなシステムで、特に有効な考え方です。しかし、そうでない、すなわち、類似開発などがあまり想定さ

れていないシステムの場合には、必ずしも効果的な手法にはなりえません。

（b）事前安全計画

一方、一般的に、新規の開発プロジェクトなどでは、開発段階で安全計画を実施することが標準的な要求事項となっています。ここで、対象システムの運用経験が皆無あるいは乏しく、したがって、プロジェクトの開発段階での事故の経験もない場合、開発段階での事後安全計画の実施は不可能です。新規の開発プロジェクトでは、設定された新規システムに対して、起こりうる事故、すなわちハザードとそれによる危害発生機構とを想定し、事故の発生以前に安全確保のための対策を検討し実施することが必要です。このような業務を事前安全計画といいます。

（c）事前安全計画の実施手順

ここでは、まず、事前安全計画実施手順について簡単に紹介します。事前安全計画は、一般的に下記のような順序で実施されます。

手順-1：対象範囲の定義
手順-2：ハザードの同定
手順-3：ハザード抑制措置の検討
手順-4：危害発生機構の定性的解析
手順-5：危害発生機構の定量的解析・評価

まず、事前安全計画を策定実行するために、システム安全業務が十分に実施できるように、対象プロジェクトとシステムおよび、それがおかれる物理的、法的環境などを理解し、当該システム安全の対象範囲を決定します（**手順−1**）。次いで、**手順−2**では、対象プロジェクトまたはシステムが**「どのような状況で、どのような事象や原因によって（危険源）、どのような事象（危害）が生ずるか（これを総称してハザードと呼ぶ）」**を予測・同定します。**手順−3**では、同定されたハザードを除去または抑制する方策を検討します。**手順−4**では、対象とするプロジェクトまたはシステムにおいて、手順-2で洗い出したハザードがシステムの障害や事故として発現するメカニズムを定性的あるいは論理的に解析します。**手順−5**では、手順-4での解析結果を確率論的に定量化し、実際にシステム障害が発生した場合の危害リスクを推定します。こうした危害リスクが当該ステムに求められる許容リスク以下になるように目標危険事象率などを設定し、この危険事象率内に収まるように具体的な対策を実施し、安全性が目標値内に収まるようにしていきます。

4.2　ハザード解析手法

(1) ハザードの解析方法

ハザードを洗い出す手法については、下記のようないくつかの方法があります。

① チェックリストによる確認手法
② HAZOP

このうち、チェックリストによる確認方法は、システム開発における経験やノウハウに基づき、システムの障害発生の元を洗い出す方法です。この手法はどれだけ意味のあるチェック項目を事前に用意するかがきわめて重要であり、通常は、先に述べた事後安全計画におけるシステム障害事象を分析して、障害の芽を洗い出し、それをチェック項目として取り込んでいくことで、より意味のあるチェックリストを整備することができます。

(2) HAZOP（Hazard & Operability)スタディズ

HAZOP は化学プラントなどを対象として危害へ至るシナリオの洗い出し、すなわちハザードの同定のために開発された手法です。安全ライフサイク

ルのどの段階でも適用でき、化学工業では、チェックリスト HAZOP とガイドワード HAZOP が使用されています。前者は予備的ハザード解析(PHA)あるいは HAZOP 1 、後者は HAZOP 2 とも呼ばれます。

チェックリスト HAZOP：プロジェクトの初期の段階で使用されますが、この段階では、システムの詳細設計に関する情報は入手できないので、在来のハザードに関するチェックリスト（チェックリスト HAZOP)を用いて、当該システムの機器で使用される物質や装置から生ずるハザードをおおまかに把握します。

ガイドワード HAZOP：ガイドワードと呼ばれるキーワードを用いて、システム要素に起こりうる正常な状態・条件からの逸脱を同定し、それらの考えられる原因および影響を特定して評価する手法です。ガイドワードには次のものがあります。

- 無し（意図された状況が起きない）
- より多く（意図された状況に関する数量が多すぎる）
- より少なく（意図された状況に関する数量が少ない）
- 余分に（意図された状況に余分な状況が付加される）
- 不十分に（意図された状況が完全には達成されない）
- 逆に（意図された状況と逆の状況が起きる）
- 以外の（意図された状況と無関係な状況が起きる）
- 早く（状況が定められた時刻よりも早く起きる）
- 遅く（状況が定められた時刻よりも遅く起きる）
- 以前に（状況が定められた順序よりも前に起きる）
- 以後に（状況が定められた順序よりも後で起きる）

システム	XXX
運転モード	通常走行時
部位	XXX

No.	パラメータ	ガイドワード	ズレの内容	ズレの原因	システムへの影響	安全対策
24	回転速度	Less	回転速度 小	回転機構への 異物の噛込み	過電流による 電子機器の発熱 機器の過大振動	電流リミッタ 変位センサ による電源 遮断
89	温度	More	冷却水温度 高	ソフトウェアエラーに 起因する熱交換器 流量制御バルブの 誤閉止	温度上昇による 電子制御系の 誤作動 機器の劣化	温度モニタ 流量モニタ

事故シナリオ：ソフトウェアのエラーで、熱交換器の入り口弁が閉まると
電子機器が冷却できず熱暴走することで、装置が制御不能に陥る。
ただし、冷却水の状態は、温度モニタと流量モニタで常時監視されている。

出典　川原卓也：潜在危険分析とリスク分析、（株）日本機能安全、機能安全エキスパート・セミナー、2006年 9 月 1 日

図4.1 HAZOP シートの例

4.3　システム障害の解析手法

(1) システム障害の解析方法

システムの安全性を阻害するシステム障害や誤動作などは、既に述べたように、実際に発生した後に、その原因を追究する事後安全計画と、事象が発生する前段階で、可能性を検討する事前安全計画の 2 つの側面で分析評価をする必要があります。

このシステム障害や誤動作の原因と、それが引き起こす事象を解析するための代表的な手法が　FMEA（Failure Mode & Effects Analysis）と FTA（Fault Tree Analysis）です。前者はシステムを構成する部分に着目し、それらの一つ一つが故障したり、誤動作した場合に、システム全体として、どのような事象が発生し、それがどのような影響を及ぼすかをボトムアップ的に分析する方式です。

一方、後者は、システムにとって望ましくない事象が、どのようなケースで発生するかをブレイクダウンしながら分析していく手法で、トップダウン型の分析手法です。

(2) FMEA (Failure Mode & Effects Analysis)

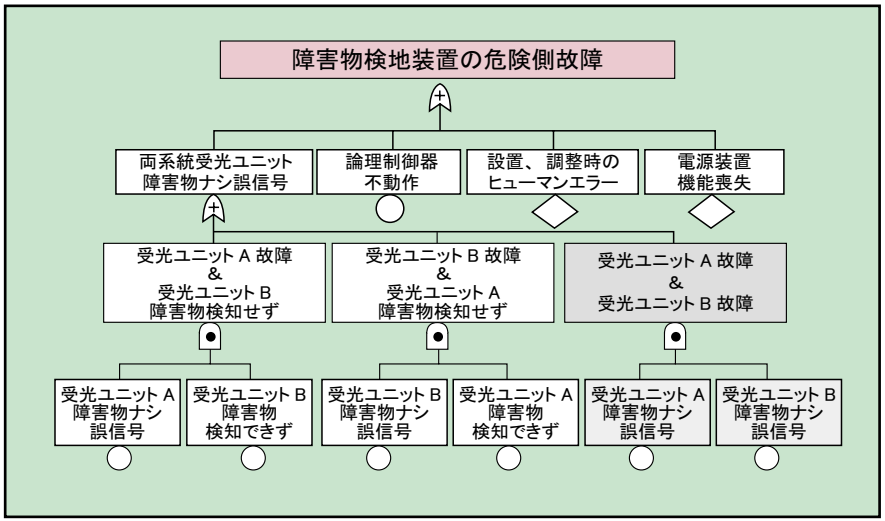
システムを構造化分析設計やオブジェクト指向分析設計などの考え方でとらえた場合、システムは、それぞれタスク、モジュール、あるいはクラス、サブクラスなどの粒度の細かい構成要素の集合体に分割することができます。こうしたシステムの構成要素一つ一つについて、どのような故障の芽（フォールトモード）が存在し、その原因は何か、またそれらが上位階層の構成要素に、どのような影響を与えるかについて、定められたフォーマットを用いて系統的に分析することができます。

このような手法を FMEA と呼びます。FMEA は、元来、信頼性工学・品質管理の分野で開発された手法で機械システムの故障解析などで多用されていますが、この考え方は組込みシステムなどの故障解析やハザードの同定にも利用することが可能です。

(3) FTA (Fault Tree Analysis)

FTA は通常、システムにとって望ましくない事象がなぜ発生するかを系統的に分析する手法です。機能安全の側面では、FTA では、HAZOP スタディズなどによってハザードと危害が特定された後に、それらの事象がなぜ生起するのかを、順次原因にさかのぼって調べ、FT（Fault Tree）を展開していく形になります。

このとき用いられる記号には論理ゲート記号、ゲート入出力記号、転移記号などがあり、「ある事象と別の事象が同時に起きると当該事象が発生する」あるいは「ある事象または別の事象のいずれかが起きると、当該事象が発生する」といったように記述していきます。Fault Tree のルートに当たる事象（頂上事象：最も好ましくない事象）は、その原因となる事象、または状態の出力とゲート記号を接続して、ツリー構造の子ノードとして展開されます。さらにそれぞれの子ノードを親ノードとして、それを引き起こす別の事象を次のレベルの子ノードとして逐次展開して、ツリー状にそれぞれの中間事象のトリガとなる事象を分析していきます。このようにしてツリー状に展開された Fault Tree において末端のノード（それ以上に展開することのできない事象）を基本ゲート入力といいます。この基本ゲート入力は、通常、その発生頻度または状態確率に関して、定量的または定性的な判断が可能な事象または状態となります。



出典　川原卓也：潜在危険分析とリスク分析、(株)日本機能安全、機能安全エキスパート・セミナー、2006年 9 月 1 日

図4.2 FTA の例

第5章　安全のためのシステム設計

5.1　組込みソフトウェアの設計手法

組込みソフトウェアは、システムに組み込まれて動作することを特徴としています。このため開発しているシステムの機能安全を実現するためには、その要素である組込みソフトウェアの設計も、適切な設計手法を用いて開発されなければなりません。

現在、ソフトウェアの設計の考え方で最もよく用いられている設計手法には、**「構造化設計手法」「オブジェクト指向設計手法」**などがあります。それぞれ長所短所はありますが、このような標準的な考え方や、設計手法を用いてソフトウェアを設計することは、機能安全を実現する上でもきわめて重要なことといえます。

(1) オブジェクト指向設計手法、構造化設計手法

構造化設計手法は、1960年代の大型コンピュータ全盛時代に、規模の大きなシステム構築のための設計手法として、体系化され整理された設計手法です。また、オブジェクト指向設計手法は1980年代以降に急速にソフトウェア設計の世界で用いられるようになった手法で、システムを構成し実現する要素（オブジェクト）という視点でとらえ、分析や設計を進めていく考え方です。これらはシステムを提供する機能という視点でとらえるか、実現するオブジェクトという視点でとらえるか、といった違いはあるものの、これらの手法に共通するのは、

① 規模の大きなシステムやソフトウェアであっても、それを構成し実現する小さな単位に分割して考えることで、問題領域を簡潔に整理していく。

② これらの複数の小さな単位を連結・連係動作させることで、一つのシステムを構成し実現していく。

という点にあります。ソフトウェアの設計作業は、基本的に技術者一人ひとりが分析検討し考える作業を通して進められるものです。どのような優秀な技術者であっても、考える範囲やボリュームが多いと、どうしてもその細部にまで目が行き届かなかったり、検討が漏れてしまうこともあります。構造化設計手法やオブジェクト指向設計手法は、こうしたことを未然に防ぐ意味で、一人ひとりの検討範囲を小さく分割することで、致命的な検討ミスを押さえ込むことが期待できます。

(2) 状態遷移モデルによるシステム設計

一方、組込みシステムでは、それが動作する外部の環境からの信号やセンサなどでとらえられる、さまざまな情報をトリガとして動作を行うイベントドリブン型のシステムが、比較的多くを占めています。こうしたシステムの設計では、どのようなイベント（動作を引き起こすきっかけ）が発

表5.1 状態遷移表の例

□	S	停止中	プロペラ駆動計算中	飛行船制御用信号送信中	位置情報受信中	着陸中
E		0	1	2	3	4
発信命令受信完了	0	⇒プロペラ駆動計算中	/	/	/	/
プロペラ駆動方法計算終了	1	×	⇒飛行船制御用信号送信中	×	×	×
飛行船制御用信号送信完了	2	×	×	⇒位置情報受信中	×	×
位置情報受信完了	3	×	×	×	⇒プロペラ駆動計算中	×
着信通知用信号送信完了	4	×	×	⇒着陸中	×	×
着信完了	5	×	×	×	×	⇒停止中

生した場合に、どのような動作を行うかといった点を簡潔に整理しておく必要があります。特に、機能安全の側面では、システムが動作する状況下で発生する、さまざまな想定外の事象に対して、システムがどのような動作や振る舞いをするかを同じようにイベントと、それの対応動作（アクション）という形で整理しておかなければなりません。こうしたタイプの組込みシステムには、状態遷移モデルと呼ばれる設計の考え方を採用した方が適している場合もあります。さらに、組込みシステム特有の微妙な動作タイミングなども考える必要があり、その場合には、システムの動作時間軸を機軸にした、イベント／アクションの整理を試みることも有効です。

5.2　システムの見える化

ソフトウェアは、一般的に「構造やからくりがよく見えない、よくわからない」という声をよく耳にします。例えどのような設計手法を用いて、きちんと内部の構造や振る舞いを考慮しても、それらが第三者に見えなかったりわからない場合、それらが適切かどうかを判断することはできず、機能安全の面からも好ましくありません。その意味で、「機能安全を実現するためのシステム設計」の第一歩は、まず、**設計の段階で「システムを見えるようにする」**ことといえます。

システムの構造やからくり、振る舞いを見えるようにするための手段としては、それらを日本語できちんと説明する方法、いろいろな図や表で整理する方法など、いくつかの方法があります。

もちろん、先に示した構造化設計手法やオブジェクト指向設計手法を用いる場合には、これらの手法の考え方を反映した設計結果の表現方法などが合わせて提案されています。いずれにせよ、どのような方法を利用しても設計が可視化され、第三者が確認できるようになれば問題はありませんが、これらの記述方法が自己流だと、第三者に誤って伝わってしまったり、

誤解を招く可能性があります。その意味では、例えば、オブジェクト指向設計手法を採用するのであれば、近年、その設計情報を整理し表現する標準的な記法としての UML(Unified Modeling Language)　などを設計表現方法として用いることもでき、これによりシステムの構造や振る舞いを適切に表記し確認することが可能になります。

こうした UML などの設計表記法を用いて、システムやソフトウェアの構造や振る舞いを抽象的に整理することを設計モデリングといいます。設計モデリングのための手法や記述方法は、いろいろなものが提案され利用されています。詳しくは SEC 発行の書籍『組込みソフトウェア開発における品質向上の勧め［設計モデリング編］』を参考にしていただければと思います。

5.3　機能安全を実現するシステム方式

ここまでは機能安全を実現する上での設計の方法について説明をしてきました。しかし、一方で、実際に機能安全をシステムの構造や振る舞いの中で実現するためには、システム構造の中に、そのための仕組みを組み込んでおく必要があります。ここでは、システムの中で機能安全性を高める方式について簡単に説明します。

(1) フェールセーフとフールプルーフ

システムの安全性や信頼性向上を目的として、従来からさまざまな実現方式が考案されています。代表的なところでは、フェールセーフやフールプルーフといったものがあります。

フェールセーフ：システムがなんらかの事情でトラブルを発生した場合にも、システム障害の影響を最小限に食い止めるという視点から、システムを安全側に帰着させるという考え方。

フールプルーフ：システム利用者が誤った操作などをした場合にも、直接的なシステム障害につながらないようにする、あるいは系としての安全性を保持するようにする考え方。

例えば、航空機などでは、昇降蛇を操作する油圧系統は、異なる複数系統が用意されており、一つの系統が故障しても、残った系統での操作が可能となっています。また、洗濯機の上蓋(ふた)をうっかり開けるとモータが止まる仕組みや、電子レンジの蓋を開けると電波照射が止まる装置など、一般のユーザが誤った操作をしても、事故につながらないような設計思想をフールプルーフといいます。

表5.2 ヒューマンインタフェースでの安全対策の考え方

安全対策	考え方	例
フールプルーフ fool proof	馬鹿なことをしても大丈夫な仕掛け	・ギアがパーキングに入っていないとエンジンが始動しない自動車 ・正しい向きにしか入らない電池ボックス ・パスワードを入れないとできない登録処理
アフォーダンス affordance	自然にそうしたくなる仕掛け	・形を使う（シェイプコーディング） ・色を使う（カラーコーディング） ・場所を使う（ポジションコーディング）
フェールセーフ fail-safe	起きた際の被害を、最小限にとどめる仕掛け	・揺れの検知で消化する石油ファンヒータ ・ネットワークの２重系 ・停電時の自家発電への自動切換 ・ファイルの自動バックアップ
多層防御 defense-in-depth	一つがだめでも、次、その次がある仕掛け	・コンピュータセキュリティの多層防御

(2) システム安全性向上のメカニズム

それでは、こうしたフェールセーフやフールプルーフといった仕組みを実際のシステムとして実現するためには、どのような方式が考えられるの

でしょうか。

ここでは、一般的な情報処理分野のシステムで多用される2つの方式を紹介しておきます。一般的な情報処理システムでは、さまざまなビジネスデータの処理（トランザクション処理）が、その中核となります。このためシステムの機能や負荷をどのように分散し、円滑な処理を実現するかといった考え方の中で、クライアント／サーバ方式などの分散処理方式と集中処理方式といった使い分けが行われています。また、システムの信頼性向上の手法として、デュプレックス方式、デュアル方式など、いわゆる、システムの2重化方式などが採用される場合も少なくありません。

デュプレックス方式：2系統のCPU（中央演算処理装置）と周辺装置を持ち、通常運用する主系と予備の従系がある。主系に障害が起きたら、従系に切り換えて運用を続ける。運用を中断しないホットスタンバイ型。

デュアル方式：2系統のCPU（中央演算処理装置）と周辺装置を持ち、互いに処理結果を照合（クロスチェック）しながら、常に同一の処理を行うシステム。片方に障害が起きた場合には、これを切りはなしてもう一方で運用を続ける。

これらの方式は、一般的な情報処理システムの世界で多用される考え方ですが、本冊子が対象とする組込みシステムの場合、システムに搭載されるマイコンやメモリなどに制約があったり、さらに消費電力面も含めさまざまな制約があります。

このため、上記のような考え方を直接導入することは難しい場合もあるかもしれませんが、一方で、発電所の制御システムや交通他の社会インフラなどに関わる組込みシステムなどでは、それらに求められる安全性や信頼性を考慮して、適宜、これらの考え方を応用していくことも視野に入れる必要があります。

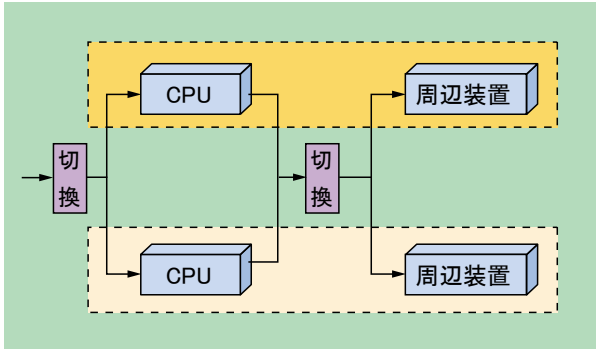


図5.1 デュプレックス方式

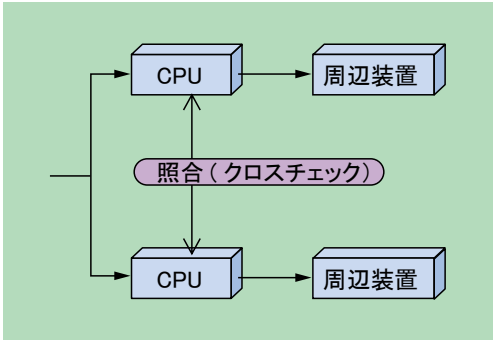


図5.2 デュアル方式

第 6 章　機能安全を実現する実装技術

6.1　設計資産の再利用

(1) IP(Intellectual Property)

システムの世界、特に半導体や電子回路などでは、それぞれを構成する要素をパッケージ化して再利用する IP（Intellectual Property）の考え方が確立されています。IP は多様な製品などで広く利用されることで、それ自身の品質や信頼性が、さらにブラッシュアップされることが多く、安全・安心なシステムを構築する上で、欠くことのできないものとなっています。

一方、こうしたシステムに組み込まれるソフトウェアは、**ソフトウェアコンポーネント（COTS：Component off the Shelf）**やソフトウェア再利用など設計資産の再利用が重要になります。COTS やソフトウェア再利用においても、実績のあるソフトウェアを使うことによって、システムの品質や信頼性を維持し、安全・安心なシステムを実現する考え方につながっています。

(2) ソフトウェア設計資産の再利用

ソフトウェアの設計資産の再利用は、最終的に利用するものは、いわゆる、ソースコードになります。しかし、それぞれのソースコードは、それぞれの設計情報や設計思想があり、また、同時に動作確認時のテストケース、テストデータといった情報があります。このため、**「ソフトウェアの再利用＝ソースコードの再利用」と考えるのは大きな誤り**です。ソフトウェア設計資産を再利用する場合には、直接的な再利用対象であるソースコードのみでなく、その実現の一部としての設計情報やテストの情報なども含

めて再利用する必要があります。

(3) 再利用の粒度

ソフトウェアを再利用する場合、どのような単位で再利用するかを考える必要があります。再利用の粒度としては、

① システム全体の骨格をなすアーキテクチャのレベル

② 一つないし複数の機能を実現するサブシステムのレベル

③ 個々のサブシステムを実現する機能構成要素のレベル

④ C 言語などのモジュールなどに相当する実装単位のレベル

など、いろいろな粒度があります。当然のことながら、上記の①、②に相当する粒度の大きな再利用では、再利用した場合の効果や影響範囲は大きいものの、再利用できる範囲は限られてしまいます。

一方、③、④のような粒度の小さな再利用では、再利用による効果は限られますが、いろいろな製品やシステムでの再利用が考えられます。

(4) ソフトウェア設計資産の再利用戦略の立案

いずれにせよ、ソフトウェア設計資産の再利用は、どのような粒度で、どの部分を再利用するかをあらかじめ検討し、設計や実装を行い、パッケージとしての切り出しを可能にしておく必要があります。近年、ソフトウェア工学の分野で話題となっている「ソフトウェア・プロダクトライン」はこうしたソフトウェアの設計資産の戦略的再利用の考え方の一つと理解することもできます。

6.2　ソースコードの標準化

(1) 組込みシステムの実装の難しさ

組込みシステムの特徴は、ハードウェアとソフトウェアが連係動作する点にあります。このため、そのソースコードもハードウェア要素との接点の部分の実装が多く、結果的に複雑になってしまう場合が少なくありません。一方で、こうしたハードウェア依存の部分を実装する点において、C 言語が多用される傾向にあります。C 言語は、こうした組込みソフトウェア特有の部分の処理に適している側面を持つ反面で、利用する技術者の経験やスキルによって、さまざまな記述方法（表現）をとることもでき、利用者の技量を反映しやすい特徴を持っています。

結果として、開発する技術者個々人の癖や経験により、作成されるソースコードの記述には、さまざまなバリエーションが生まれたり、品質にバラツキが生じてしまう状況が生まれかねません。

企業活動の一環として開発されるソフトウェアが開発者個人の能力などによって、バリエーションが増えたり、品質にばらつきが生まれてしまうことはできれば避けたいものです。

(2) ソースコードの標準化のための方法

最近の組込みソフトウェアのほとんどは、複数人の技術者で開発されています。こうした場合、開発者が10人いれば10通りのソースコードの書き方が存在することになります。この状態を放置すると、前述したように、ソースコードの記述様式や品質のレベルはバラバラになってしまいます。こうした事態を避けるためには、個々の組織や開発プロジェクト、開発チームの関係者間で、ソースコードの書き方などのルールを決め、守っていくことが必要になります。このようなソースコードの書き方に関するルールをコーディング規約と呼びます。

(3) コーディング作法／規約とは

一般にいわれる、コーディング規約、コーディングルール、コーディン

グスタイル、コーディング作法、コーディング基準などのソースコードを取り巻くこれらの規約やルールは、これまで数多くのものが提案されています。これらはいずれも、過去の先人たちが、さまざまなソフトウェア開発の中で得た経験やソースコードを記述する際に、どのような点に注意すべきかといった事項を整理したものです。

すなわち、コーディング規約の中には、いままでに起こしたソースコードにまつわる問題の原因を参考に、「このような書き方をするとトラブルを引き起こす」という経験をルールや規約の形に整理したものなども含まれています。

また、コーディング規約では、個々の会社の経験に基づいて決められたルール以外にも、ドメインや製品などの特性に基づいて決められている場合もあります。このようなルールの制定経緯を考えると、これらのコーディング規約を守らなかった場合、ソースコード上の不具合を引き起こす可能性があると考えることができます。

SEC でも組込みソフトウェア開発に造詣の深い皆さんに協力をいただき、わが国の産業界で、コーディング規約を整備していただく際の参考として、「コーディング作法ガイド」を整理しました。

このガイドは、ISO/IEC 9126で定義されたソフトウェアの品質特性を軸に、それらを向上されるためのソースコード作成時の注意点を体系的に整理し、対応するコーディングルールを参考としてまとめてあります。

表 6.1　コーディングルールの一覧表

基本名称	目　的	開発年度	特　色
GNU Coding Standards	クリーンで矛盾のないインストールが容易な GNU システムを作成すること	2005/01	ドキュメントやソフトウェアのインストール、Makefile の規約まで論じられている
Linux Kernel coding style	Linux カーネルのソースコードの好ましい書き方	2004/02	Linux Kernel のソースに付属〈Linux Kernel のソース〉／Documentation/Coding Style
Java Code Conventions (SUN Microsystems,Inc.)	Java 言語の標準的なコーディングスタイルの取り決め	1999/04	SUN Microsystems,Inc. が作成したコーディングスタイルの取り決め
Programming in C++, Rules and Recommendations(Ellemtel)	C++プログラミングの１つのスタイルを定義する	1992	AT&T の C++言語システムに基づいている
Recommended C Style and Coding Standards (Indian Hill)	移植性を高め、メンテナンスの手数を減らし、プログラムをわかりやすくする	1990/06	Cプログラムのために推奨された標準のコーディング
C STYLE GUIDE (SOFTWARE ENGINEERING LABORATORY SERIES SEL-94-003)	C プログラムを書くため、ソフトウェア工学研究室(SEL)が推奨するスタイルを記述、ここで「良いコード」とは、組織化した、読みやすい、理解しやすい、維持しやすい、効率的なコードと定義される。	1994/08	ソフトウェア工学原則が論議されて示され、コード例題がよい慣例を示すために提供される
MISRA－C	車載用ソフトウェアを対象とした C 言語プログラミングガイドライン	1998	ソフトウェア開発プロセス全体についての開発ガイドラインがあり、この流れの上での、C言語でプログラムを作成する工程についての開発ガイドライン。127のコーディングルールを規定している。
Sater－C（書籍）	安全関連ワークスでの C 言語の利用ガイダンス	1995	ソフトウェア品質について述べられている

コラム　組込みソフトウェア向けコーディング作法ガイド

組込みソフトウェアの規模は拡大の一途をたどり、多人数による開発形態に移行しています。開発者数の増大とともに様々なスキルレベルの開発者が参画するようになり、スキルレベルの統一のための施策が今まで以上に必要となっています。特に、安全性、保守性、および移植性を考慮した高品質なC言語のコーディングは、組込みソフトウェア開発の中で基礎体力と位置付けられるものと考えます。高品質なC言語コーディングのためには、開発者の暗黙の了解でコードを記述するのではなく、コーディング規約を形式知として定め、規約に基づいてコードを記述することが有効と考えます。しかし、コーディング規約を制定するためには、C言語の文法に精通している必要があることなど、高いスキルと多大な労力が必要です。また、開発者への理解を深め、普及・徹底を図ることに大きな障壁があることも事実です。このような背景から経済産業省組込みソフトウェア開発力強化推進委員会では、産官学連携の取組みの一環として、C言語のコーディング規約を策定している方々を対象にした「コーディング作法ガイド」を編纂しました。このコーディング作法ガイドでは、品質特性に対応して分類した作法と、それに対応するルール群から構成されています。コーディング規約を策定する方々は、ルール群の中から必要部分を取捨選択することにより、目的に見合ったコーディング規約を策定することができると考えます。

『組込みソフトウェア開発向け
コーディング作法ガイド [C言語版]』
SEC編　2006年5月発行

第７章　高度な機能安全の実現に向けた技術

7.1　システムの確かさの確認

どのようにきちんとした安全要求をもとにシステムを設計・実装したとしても、当初の計画どおりのシステムが構築されるとは限りません。一般のソフトウェア開発では、「ソフトウェア開発のＶ字型モデル」と呼ばれる考え方があります。これは、開発の前半（上流）で検討した要求事項や設計内容をもとに、それらが実際のシステムやソフトウェアで実現できているかをテストによって確認する考え方です。

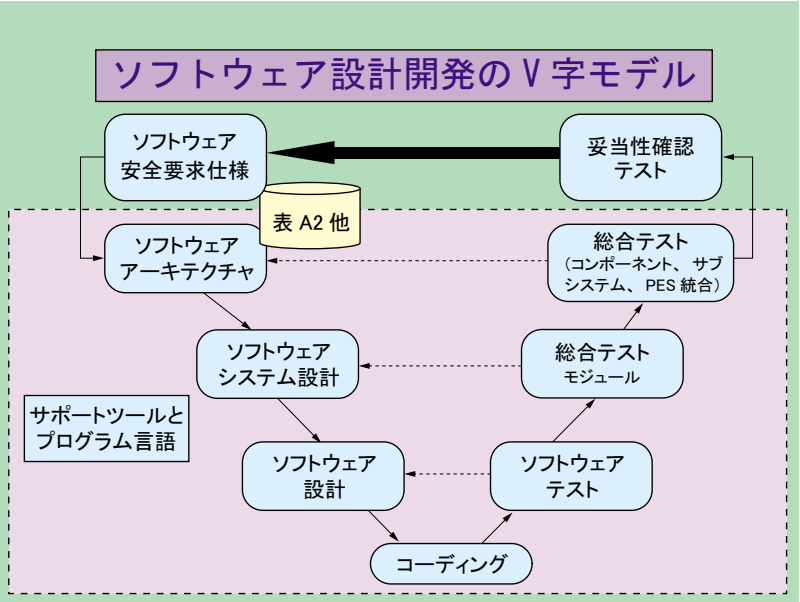
(1) ソフトウェアテストの考え方

実際のソフトウェアテストのフェーズは**単体テスト、結合テスト、総合テスト**の通常３つの段階に分かれます。

単体テスト：これはシステムを構成する小さな単位（タスクやクラス）ごとにそれぞれの設計内容どおりの動作をするかどうかを確認します。

結合テスト：これは上記で確認した個々の単位を複数連結（リンク）させて、システムで実現する個別の機能的な側面が動作するかどうかを確認します。

総合テスト：ソフトウェアとして当初の要求段階で求められた事項が実現できているかどうかを全体として確認します。



出典　吉岡律夫：機能安全規格と適合認証、機能安全規格と IEC 61508とその認証、（独）産業技術総合研究所、産総研ワークショップ、2006年２月

図7.1 ソフトウェア開発のＶ字モデルと安全要求

(2) ソフトウェアテストのテクニック

ソフトウェアのテストを実際に行う場合には、何をテストするかというテスト項目（テストケース）と、実際にソフトウェアを動作させて確認するためのテストデータが必要になります。通常、テストケースやテストデータは、ソフトウェアやシステムの要求仕様書や設計書などを参考に作成しますが、それらの作成方法として、いろいろな手法が利用されています。ここでは代表的な考え方として、ブラックボックステストとホワイトボックステストを取り上げて紹介しておきます。

(a) ブラックボックステスト

対象となるソフトウェアやそれを構成する要素をあたかも一つの箱と理解し、その中身である処理内容や論理構造には着目せず、単純に、その箱に対する入力と出力が妥当であるかどうかをテストする方式です。

(b) ホワイトボックステスト

ホワイトボックステストは、逆にソフトウェアやそれを構成する要素の中身である処理内容や論理構造に着目し、その流れを追いかけてテストしていく方式です。

これらの方式に付随して、境界値／限界値分割法、原因結果グラフの活用など、いろいろなテクニックが利用されますが、詳しくは、ソフトウェアテスト関連の書籍などを参考いただければと思います。

システムとしての機能安全を実現する上では、当該システムが当初の要求事項を適切に実現できているかどうか、が最初の出発点となります。その意味では、ソフトウェア開発の後半で適切なテストを実行して、システムの動作上の問題点を早めに洗い出しておくことはきわめて重要になります。

(3) システムとしてのテスト

本冊子で扱う組込みシステムでは、ソフトウェアとハードウェアが協調して動作することが一つの特徴ともいえます。しかし、このソフト・ハードの協調動作は、時として設計者の想像を超えたトラブルを生み出すことがあります。すなわち、ソフトウェア開発段階でのテストでは、正しく要求どおりの動作をしていたとしても、ひとたびハードウェアと結合され実際のフィールドで動作させた段階で、思わぬ動作をする場合が少なくありません。こうした事故を未然に防ぐため、組込みシステムの場合には、必ず、実フィールドでの動作確認やテストを行う必要があります。特にこの場合、

① どのようなユーザが

② どのような状況下で

③ どのような操作をするか

といった点をきちんと洗い出して、実機テストを行う必要があります。実フィールドで発生するシステム障害やシステム安全性関連の事故の多くは、これらの 3 点のいずれかを十分に洗い出せず、結果として、動作確認がもれていた場合が少なくありません。

7.2　システム検証

(1) テスト技術の限界

ソフトウェアやシステムの正しさを確認する代表的な手法は前述した「テスト」です。テストも含めてソフトウェアの正しさを確認するには、「対象ソフトウェアを実際に動作させて確認する」方法と「対象ソフトウェアを動作させずに確認する」方法の 2 種類があります。

通常、実際の開発現場で行われるテストは前者に属しており、動的テスト(Dynamic Testing)とも呼ばれます。この動的テストでは開発の現場で、最も多用される手法の一つですが、次のようないくつかの欠点があります。

欠点 1 ：ソフトウェアを実際に動かすため、実際にソフトウェアがコンパイルビルドされた後でないと実施できない。

欠点 2 ：ソフトウェアを実際に動かすという点で、ソフトウェア内部の条件分岐などを考慮して、あらゆる動作パターンをテスト仕様とすると膨大な数のテストケースやデータが必要となり現実には全てを網羅しきれない。

このため、実際のソフトウェアやシステムの正しさを動的テストで確認しようとする場合には、これらの欠点も考慮して行うことが重要になります。

一方で、近年、こうした動的テストの欠点を補う観点から検証技術が着目されています。

(2) 検証技術

上記の動的テストの欠点であるテスト実施時期の問題と、テストの網羅性の問題を解決するためには、ソフトウェアの実装よりも早い段階で、ソフトウェアで実現する動作を網羅的に確認する必要があります。

その代表的な手法は、レビューやインスペクションと呼ばれる方法です。システムやソフトウェア開発の途中段階で作成した仕様書や設計書を、第三者の手によって、精査し、その誤りをチェックしようというものです。

レビューやインスペクションでは、レビューア（レビューやインスペクションをする技術者）の経験やスキルに依存する部分も多いため、適切なレビューアを選ぶことがきわめて重要になります。

しかし、その一方で、どのようなベテランの技術者にレビューアの協力を得ても、やはり、規模の大きなシステムや複雑なシステムの場合には、チェックの網羅性という観点で不安が残ってしまいます。これを解決するための技術の一つが形式検証技術です。

(3) 形式検証技術

形式検証技術とは、ソフトウェアの仕様や設計などをある決まったフォーマルな書式で記述し、その情報を利用して、仕様や設計に誤りがないかどうかを検査する方法です。

代表的な考え方としては、ソフトウェアの仕様や設計を数学的な記述を用いて記述し、検査する方式（定理証明法）やいわゆるモデリングと呼ばれる方式で表現された仕様や設計を検査するモデル検査法と呼ばれる方式があります。

実際の開発現場に導入するためには、これらの手法の基本的な考え方を理解した上で、仕様や設計をこれらの手法に合致するように数式で記述したり、状態遷移モデルなどで記述し、検証ツールなどを利用して検証する

必要があります。

現在、さまざまな研究機関などで形式検証技術を実際のソフトウェア開発に適用するための工夫や研究が進められています。形式検証技術は、仕様や設計をある記述方法にのっとって記述すれば、検証ツールによって、その内容を網羅的にチェックすることができるため、高い信頼性や安全性を求められるシステムやソフトウェアの検査方法としては、きわめて有効な手法の候補の一つです。

例えば、モデル検査を行うツールの代表的なものとして、SPIN が知られています。SPIN では、システムの動作仕様を Promela という記述言語で記述して、自動検査を行います。検査により、状態遷移上のデッドロック（行き止まり）や到達不能状態などの検出が可能です。

付録　機能安全と国際規格 IEC 61508

IEC 61508[1]（日本では JIS C 0508[2]）は国際電気標準会議(IEC)によって、2000年までに策定された国際規格の一つであり、コンピュータ技術を用いた場合の安全を実現するための機能安全国際規格と呼ばれています。この規格は電気・電子技術やコンピュータ技術を応用した産業における安全確保を促進するために個別の分野規格の制定を促すことを目的の一つとしており、対象とする産業を特定していません。

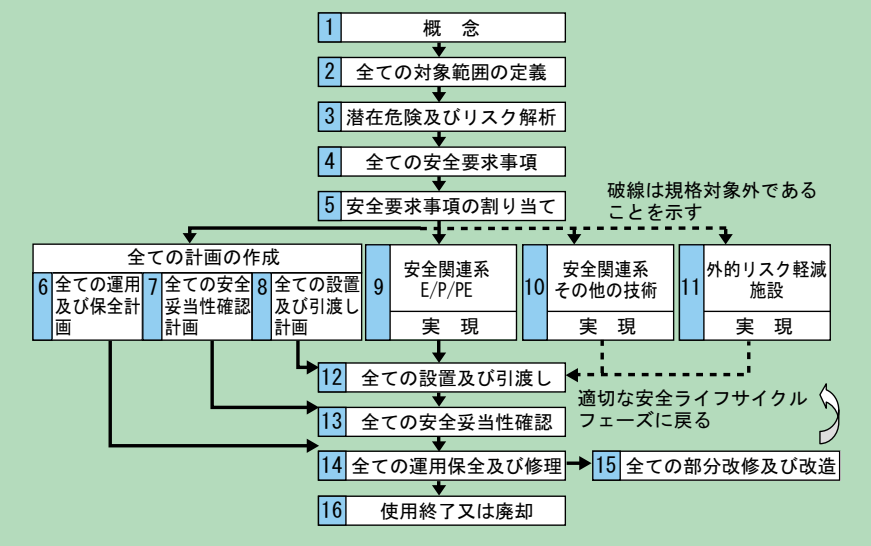
コンピュータ技術は電子部品からなるハードウェア技術と、これを動作させるソフトウェア技術とからなります。規格は、安全関連系(SRS：Safety Related System)のハードウェア・ソフトウェアの設計指針であると同時に、業務遂行に関するマネージメントの規格という側面もあわせ持っています。規格ではシステムの安全という視点からのリスク評価、安全度水準(SIL：Safety Integrity Level)、全安全ライフサイクル、機能安全評価方法や、機能安全を実現するためのさまざまな技法の導入などについても述べられています。

(1) 安全の枠組み

IEC 61508では、電気・電子・プログラマブル電子系（以下 E/E/PES と略)、すなわち、コンピュータ技術を応用した産業やシステムを対象としています。

ISO/IEC ガイド51では、品質と安全は同義語ではなく、品質規格と安全規格の役割を混同すべきではないことを述べています。すなわち、すべての産業システムで、品質が良ければ安全であるとは限りません。IEC 61508は

例えば、品質問題等で事故に至るような事象が発生した場合に、SRS(Safety Related System：安全関連系）を正しく作動させることで、リスクが大きくならないようにするための安全規格です。安全の確保は、部品メーカ、サブシステム統合会社、エンジニアリング会社、システム運用会社といったSRSの供給チェーンに広く関係します。このため、規格は付図１に示す全安全ライフサイクルという考え方を採用し、概念・設計・保守・改修・廃却にいたる、ライフサイクルを通じた業務管理手順を規定しています。



付図１　全安全ライフサイクル

(2) IEC 61508の構成

IEC 61508は全７部から構成され、翻訳規格であるJIS C 0508では全350頁からなる大きな規格です。

付表１　JIS C 0508（電気・電子・プログラマブル電子安全関連系の機能安全）

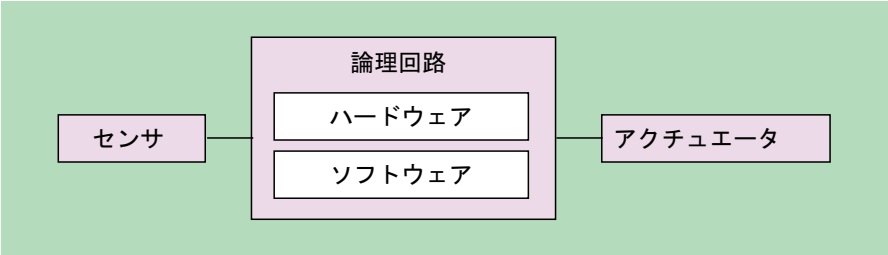
パート	概　　要
第１部 一般要求事項	全体の考え方をまとめており、機能安全を達成するための管理および技術上の要求事項をまとめた機能安全管理、概念・設計・改修・廃却にわたる機能安全の維持を目的とした全安全ライフサイクル、潜在危険およびリスク解析、E/E/PESの安全度水準、適合性確認、評価者の独立性を示した機能安全評価などが想定されています。
第２部 電気・電子・プログラマブル電子安全関連系に対する要求事項	E/E/PESのハードウェアに対する要求事項であり、E/E/PESの設計に関わる要求事項を中心に機能失敗確率の推定法や安全度水準決定法などが述べられています。
第３部 ソフトウェア要求事項	E/E/PESのソフトウェアに対する要求事項をまとめており､オペレーティングシステムからアプリケーションまで、安全度水準に応じたソフトウェアが使われなければならないことを述べています。
第４部 用語の定義および略語	機能安全に関わる用語の定義をまとめています。
第５部 安全度水準決定方法の事例	SRSの信頼度、すなわち安全度水準（SIL）を解析で求める方法をまとめています。
第６部 第２部および第３部の適用指針	第２部、第３部の適用指針。プログラマブル電子系の信頼度を、アーキテクチュア構成をもとにして算出する事例をまとめています。
第７部 技術および手法の概観	第２部、第３部に関係する安全技術・手法を紹介しています。

(3) 機能安全

「機能安全」とは新しい用語であり、本質安全と対比する用語です。鉄道が良い例ですが、立体交差にすれば、踏み切りを渡って事故に遭遇する可能性はなくなります。このような対策によって根源からリスクを無くして達成する安全が本質安全です。しかし、一般には全てを立体交差にできないので、ある程度のリスクが残ります。すなわち、そのままでは本質安全を達成できない場合には、リスクが残るので、信号や列車停止装置などの周辺の安全機能により、相対的にリスクを軽減させ、許容されるリスク

以下にして安全を確保することを機能安全と呼びます。

IEC 61508が対象とする電気・電子・プログラマブル電子系(E/E/PES)は、ハードウェアおよびソフトウェアからなる論理回路と、センサやアクチュエータなどのサブシステムから構成されます。こうした系の安全機能は個々のサブシステムによってではなく、これらのサブシステムがすべて機能したときに正しく実現されると考えることができます（付図２参照）。



付図２　典型的な E/E/PE 安全関連系の構成

（4）故障と安全度水準

システムに故障は付き物ですが、こうした故障がどの程度の頻度で発生するかを念頭に、その許容範囲（許容リスク）を表す指標を IEC 61508では安全度水準(SIL : Safety Integrity Level)と呼んでいます。

IEC 61508では、付表２に示す４段階の数値目標をそれぞれの SIL に対して定めています。

（5）機能安全評価

IEC 61508では、組織などで採用した機能安全実現の仕組みが IEC 61508の要求事項に適合することを調査するための機能安全評価を実施することが求められています。機能安全評価は、「根拠に基づいて、一基以上の E/E/PE 安全関連系、他技術安全関連系または外的リスク軽減施設によって、

付表２　安全度水準：E/E/PE 安全関連系に割り当てられる安全機能に対する目標機能失敗尺度

SIL	低頻度作動要求モード運用（注１）	高頻度作動要求または連続モード運用（注２）
4	１０$^{-5}$以上１０$^{-4}$未満	１０$^{-9}$以上１０$^{-8}$未満
3	１０$^{-4}$以上１０$^{-3}$未満	１０$^{-8}$以上１０$^{-7}$未満
2	１０$^{-3}$以上１０$^{-2}$未満	１０$^{-7}$以上１０$^{-6}$未満
1	１０$^{-2}$以上１０$^{-1}$未満	１０$^{-6}$以上１０$^{-5}$未満

（注１）作動要求当たりの設計上の機能失敗平均確率
（注２）単位時間当たりの危険側故障確率［1/ 時間］

機能安全が達成されることを判定するための調査」であり、機能安全評価実施者を１名以上任命する必要があります。機能安全評価は対象とする E/E/PES の安全度水準の程度、または災害の過酷度に応じて、評価者などを準備し、IEC 61508の要求事項と照らし合わせて、受容、条件付受容、拒否のいずれかの評価を行います。

（6）国際規格の動向

IEC 61508の規格制定の目的の一つには、分野ごとの規格の制定を促すことがあり、運用側の分野規格や指針類が制定され始めています（付表３）。

また、IEC 61508自身も５年に１回の割合で改訂作業が進められており、現在、次の改定に向けた検討が進められています。日本国内でも、この改訂作業に合わせて、国内対応委員会が組織され検討が進められています。

付表３　機能安全分野規格の制定状況

年	国際規格	備考
1998	IEC 61508-1,3,4,5	—
2000	IEC 61508-2,6,7	—
2001	IEC 61513	原子力
2002	IEC 62278	鉄道
2003	IEC 61511-1 IEC 62279 IEC 61800	プロセス産業(注) 鉄道 電動モータ
2004	IEC 61511-2,3	プロセス産業
2005	IEC 62061	機械類
2008	IEC 61508 Rev1	予定

（注）化学，石油精製，石油・ガス製造，パルプ・製紙，原子力以外の発電など

参 考 文 献

1） 片山卓也・土居範久・鳥居宏次監訳：ソフトウェア工学大事典、朝倉書店、ISBN：4254121237

2） IEC 61508，Functional Safety of Electrical/Electronic/Programmable Electronic Safety Related Systems，1998

3） JIS C 0508，電気・電子・プログラマブル電子安全関連系の機能安全、日本規格協会、2000．ISO/IEC GUIDE51，1999

4） IEC ホームページ　http : //www.iec.ch/zone/fsafety/pdf_safe/brochure.pdf

5） IEC 61511，Functional safety-Safety instrumented systems for the process industry sector ，2003

6） IEC 62061，Safety of machinery-Functional safety of safety related electrical，electronic and programmable electronic control systems，draft

7） EN 954－１，Safety of machinery-Safety-related parts of control systems -Part１：General principles for design，1996

8） ISO 13849－１，Safety of machinery，Safety related parts of control systems，General principles for design，1999

9） IEC 61513，Nuclear power plants-Instrumentation and control for systems important to safety-General requirements for systems，2001.

10） IEC 61800－５－２，Adjustable speed electrical power drive systems，draft.

その他、参考とした資料

(1)　中嶋洋介著、向殿政男監修：安全とリスクのおはなし（おはなし科学・技術シリーズ）安全の理念と技術の流れ、日本規格協会、ISBN：４－542－90277－３　2006.6

(2)　向殿政男著：よくわかるリスクアセスメント、事故未然防止の技術、中央労働災害防止協会、中災防新書、ISBN：４－8059－0901－３、2003.10

(3)　向殿政男監修：国際化時代の機械システム安全技術、安全技術応用研究会編、日刊工業新聞社、ISBN：４－526－04566－７、2000.4

(4)　重大事故の舞台裏―技術で解明する真の原因、日経ものづくり編集、日経BP社　ISBN：４－8222－1885－６、2005.10

(5)　（独）産業技術総合研究所：産総研ワークショップ、機能安全規格と適合認証－IEC 61508のさらなる理解に向けて－、速記録、2006年２月

(6)　（株）日本機能安全：機能安全エキスパート・セミナー：2006年９月１日

おわりに

医療機器、エレベータ、列車など身の回りのさまざまな機器や装置に、組込みソフトウェアが利用されている中で、あたり前のように、さまざまなシステム障害や事故が発生しています。本冊子でも触れたように組込みソフトウェアに関わる関係者は、経営者から現場の技術者、マネージャ、そしてシステムのユーザまで多岐にわたります。その一人ひとりが常に携わるソフトウェアの安全性や信頼性を頭の隅に置いてシステムと関わることで、いま以上に安心・安全なシステムを作り上げることができるようになるのではないかと思います。その意味で、本冊子は、こうしたシステム関係者の皆様が「機能安全」というキーワードのもと、システムの安全性や信頼性を見つめなおすきっかけとしていただければ幸いです。

なお、本冊子は経済産業省およびSECの連携による組込みソフトウェア設計力強化タスクフォースの機能安全部会での議論をもとにして編集したものです。

機能安全は、安全・安心なシステムを提供していく、ものづくりに携わる技術者の原点となるテーマであり、一方で、IEC 61508に見るように国際的にも大きな関心が寄せられているテーマです。

SECでは機能安全部会の中で引き続きこのテーマについての深堀を行い、広く一般の技術者の皆様の役に立つ情報を提供していく予定です。

○**執筆者**（敬称略）

経済産業省 組込みソフトウェア開発力強化推進委員会 機能安全準備部会（2006年度）

主　査：佐藤　吉信　　東京海洋大学

向殿　政男　　明治大学

平尾　裕司　　長岡技術科学大学／財団法人 鉄道総合技術研究所

兼本　　茂　　会津大学

程子　　学　　会津大学

木下　佳樹　　独立行政法人 産業技術総合研究所

水口　大知　　独立行政法人 産業技術総合研究所

松岡　　聡　　独立行政法人 産業技術総合研究所

吉岡　律夫　　株式会社 日本機能安全/日本システム安全研究所 有限会社

田邊　安雄　　株式会社 日本機能安全

門田　　浩　　IPA SEC/日本電気 株式会社

田丸喜一郎　　IPA SEC/株式会社 東芝

平山　雅之　　IPA SEC/株式会社 東芝

大野　克巳　　IPA SEC/トヨタテクニカルディベロップメント 株式会社

及川　　健　　株式会社 三菱総合研究所［事務局］

小迫　光貴　　株式会社 三菱総合研究所［事務局］

田中　秀明　　株式会社 三菱総合研究所［事務局］

監　修：組込みソフトウェア開発力強化推進委員会

編 者 紹 介

独立行政法人 情報処理推進機構 ソフトウェア・エンジニアリング・センター

2004年10月に独立行政法人 情報処理推進機構（IPA）内に設立されたソフトウェア・エンジニアリング・センター（SEC）は、エンタプライズ系ソフトウェアと組込みソフトウェアの開発力強化に取り組むとともに、その成果を実践・検証するための実践ソフトウェア開発プロジェクトを産学官の枠組みを越えて展開している。

［所在地］ 〒113-6591 東京都文京区本駒込2-28-8
文京グリーンコート センターオフィス
電話 03-5978-7543，FAX 03-5978-7517
http://sec.ipa.go.jp/

SEC BOOKS
組込みシステムの安全性向上の勧め（機能安全編）

平成18年11月10日　　第1版第1刷発行

編　　者　独立行政法人 情報処理推進機構
　　ソフトウェア・エンジニアリング・センター
発 行 者　佐 藤 政 次
発 行 所　株式会社 **オーム** 社
　　郵便番号　101-8460
　　東京都千代田区神田錦町3-1
　　電 話　03(3233)0641(代表)
　　URL　http://www.ohmsha.co.jp/

印刷・製本　　報光社
ISBN 4-274-50113-2　Printed in Japan