ソフトウェアと安全関連システム

独立行政法人自動車技術総合機構
交通安全環境研究所　鉄道認証室

森　　崇

0. 登場人物

鉄道信号メーカー　カバ興業チーム

カバ興業　社長
座右の銘:技術と直感

カバ興業　営業　カバお
「怒られてナンボの毎日」

カバ興業　技術　オタかば
「面白くなければ技術じゃない」

カバ興業　プログラマ
ハッキングカバ
「俺しかできないことをやる」

カバ興業設計課長　カバ実
「全体のレベルアップ」

謎な奴

謎のフリーコンサル
なぞカバ
「知識は力！」

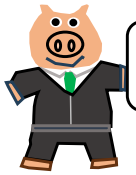
バチバチ

鉄道事業者　カバ鉄道チーム

カバ鉄道　社長
品格の経営、根拠ある経営

カバ鉄道　電気課長
口癖:安くてエエもん持って来い！

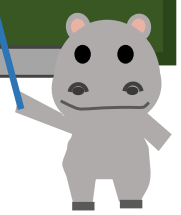
カバ鉄道　乗務員　カバどん
いつかは自動化されるかも。ドキドキ



ブタ工業　社長
カバ興業を凌ぐ強い体質づくり

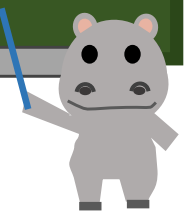
1. はじめに

- ソフトウェアは、「このようなものが必要」という要求を「CPUに分かるように変換する」ことであると考えています。
- 開発は、要求事項を挙げて、どのような構成にするか決めて、作ったもの、買ってきたものを組みつけていく。まさに建築と同じと考えています。その中で、どのような安全の度合いで作るかを加味しつつ、お客さんが「こんなもの欲しいな」というのをプロが注意深くブレークダウンしていくために、RAMS関係の規格ではどのように考えられているのかを説明します。



1. はじめに（本日の項目）

- どんな故障が保安システムにはあるのか？
- ソフトウェアの観点では、故障を防ぐためにはどうすればいいか？
- SILやTFFRと、ソフトウェアの安全性インテグリティの関係
- 開発手法と規格
- これはちょっと。本末転倒では？



2. ソフトウェアとフェールセーフのカバ興業？

社長！いつもいつもしつこくてゴメン。でもな、やっぱりゆうとかなアカンことはあるねん。御社の保安装置、安全機能はSIL 4で作っとるよな。

そりゃまあ、そうしてますよ。そこについては私も口を酸っぱくして、ちゃんとせい、妥協するな！と社内で言っていますよ。フェールセーフのカバ興業ですし。

そこやねんな。この前ウチの社長から、「お前ら二言目にはフェールセーフフェールセーフゆうけどな。ワシはうたがっとんねん。」って言われたわ。

それは誠に遺憾ですな。社長さんにはちゃんと理解していただかなくては。

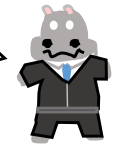
いやー、この前な、「そんなもん、電子連動ゆうても、ちゅーりんぐましんやろ。言われたとおりしか処理せーへんモン、データ間違いやったら、どうしようもないやろ。そもそもソフトが間違っとっても、どうしようもないやろ。なんでそれがフェールセーフやねん。ワシまちごうてるか？」って言われたわ。大体ちゅーりんぐましんってなんや！てなったわ。

そこについては私も口を酸っぱくして、ちゃんとせい、妥協するな！と社内で言っていますよ。なんせウチは優秀なスタッフたちが。。。

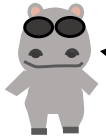
いやー、多分社長は、すでにあらかた答え持っとんねん。あれはかなりタヌキやからな。「電子連動止めろとは言ってない。安全関連機能はSIL 4やろ。確率論だけやない。」ともゆうとったからな。
あれは地雷や。間違うと爆発する。タヌキは事務系のくせにヤバいで。ちゃんと理屈まとめて、報告頼むで。

（ま、丸投げやん。。）

2. ソフトウェアとフェールセーフのカバ興業？

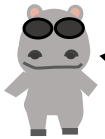


おいハッキングカバ、オマエのソフトSIL 4やいう証拠出せ！100万年に一回しか間違わんという証拠出せ！



ハァ？なんやそれ。「百万年生きたカバ」か社長？まず100万ってどこから来てんねん。まーー度も仕事してないヤツやったら、100万年に1度も間違わんやろうなあ。サボりでええんやな。今日からブラ勤させてもらうわありがとな。

SIL 4ゆうたら、10^{-9}/hやろ。100万年に一回やろ!

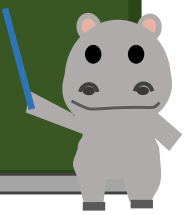


ハ～！それって、ずっと動いてる装置の中で、部品とかが壊れた時、安全に関係する機能を失ってしもて危険につながる、一時間当たりの回数とかそういうのちゃうか。ワシは機械やないで。カバやで。ソフトウェアの故障って、急に壊れてランダムに出るか？ちがうやろ。

IEC 62278-1 5.6.2 Classes of failures, 5.6.3 Derivation of detailed railway-specific influencing factors

故障の区分

- ランダム故障
  - 経年故障など、離散的に起こる確率的な故障
  - →許容できるレベルまで頻度を低減
- システマティック故障
  - ヒューマンエラーなどでライフサイクル全体で起こりえる決定論的な故障
  - →故障のヤバさに応じて管理手法を調整
- 上記故障のヤバさ
  - どれくらいまで許容できるか
  - →TFFR及びそこから導出されるSIL

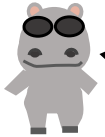


3. 社長の統制下のカバ興業？

そうや、管理やそれや管理や。オレが箸の上げ下ろしまで管理させてもらうからゆうこと聞け！今からのワシのお言葉全部業務命令や！

ハァ？なんやそれ。それがシステマティック故障となんの関係があるねん。単なるイジメ、不当労働行為やろ。その合理性について、ちょっと団交させてもらうで。IEC 62279の4.6にも、「ソフトウェアの安全性インテグリティレベルに応じた対応」が書かれていて、メリハリ付けるんちゃうんか！

そうやそれが言いたかったんや。



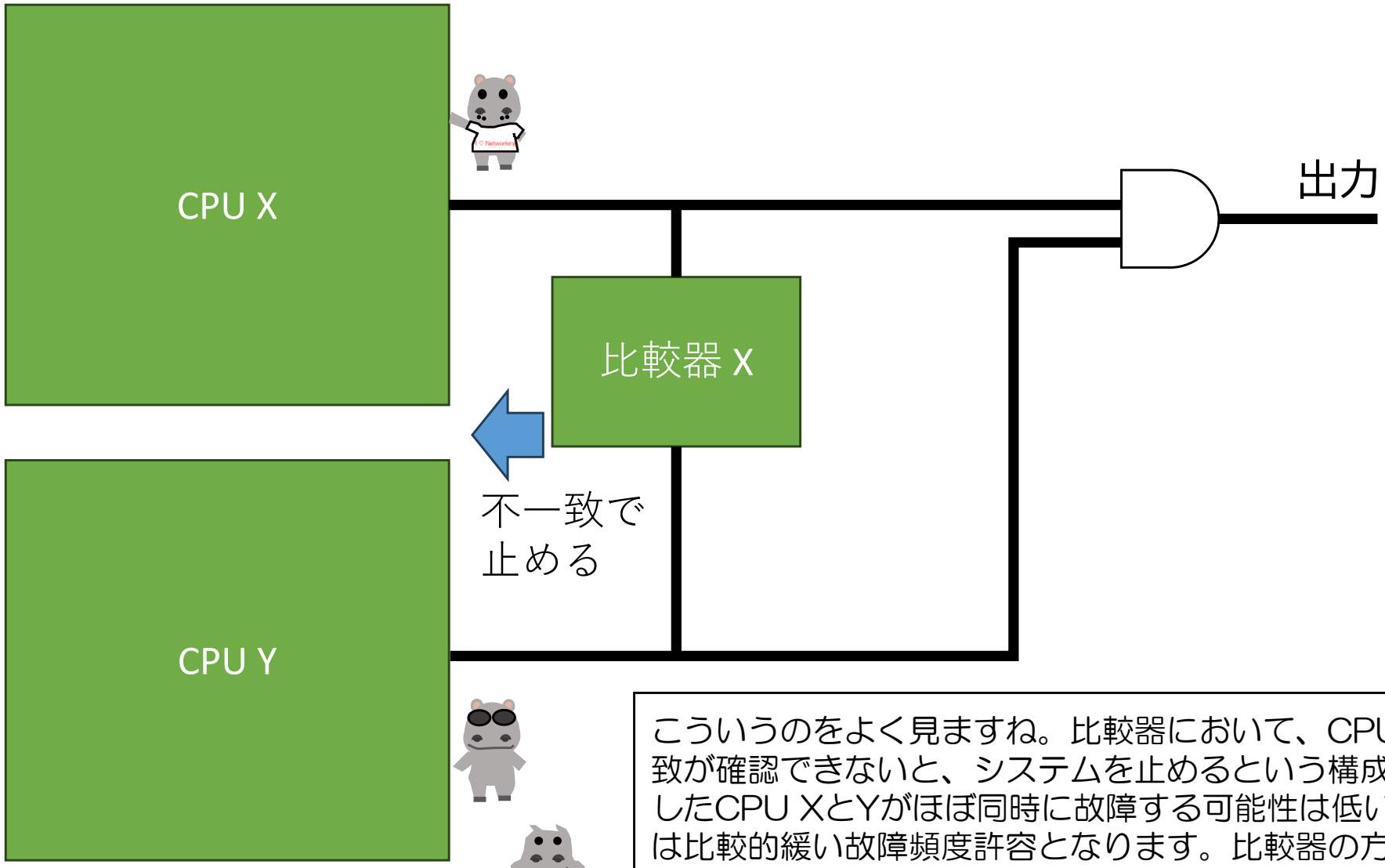
ハ～！ホンマ適当なことばっかりいうねんな。リソースは有限なんやで。社長。
そのリソースを安全のレベルに応じて振り分けるのは、「組合」やなくて「社長」ちゃうか？そんな事を「カバ興業労働組合」から説教されるっておかしいやろ。反省してや。

システマティック故障

IEC 62279 4 Objectives, conformance and software safety integrity levels

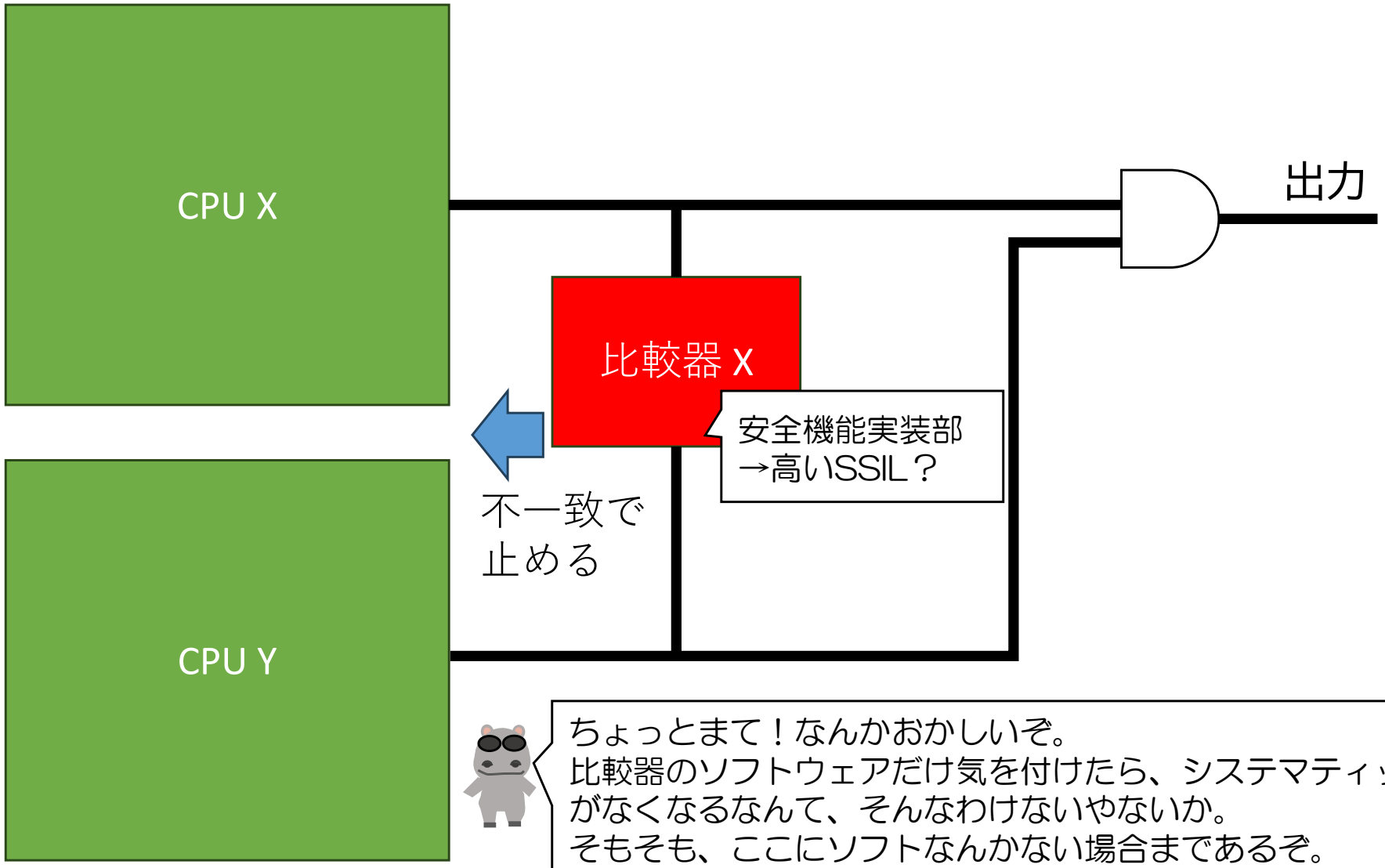
- “to the extent required by the software safety integrity level”→ソフトウェア安全性インテグリティレベル(SSIL)に応じた対応
- Annex Aの技術と対応(T&M)を使用して対処。
- じゃあSSILはどう決める？
- どのような段階で管理し、T&Mを活用する？

4. 安全機能の実装部が高いSSIL？



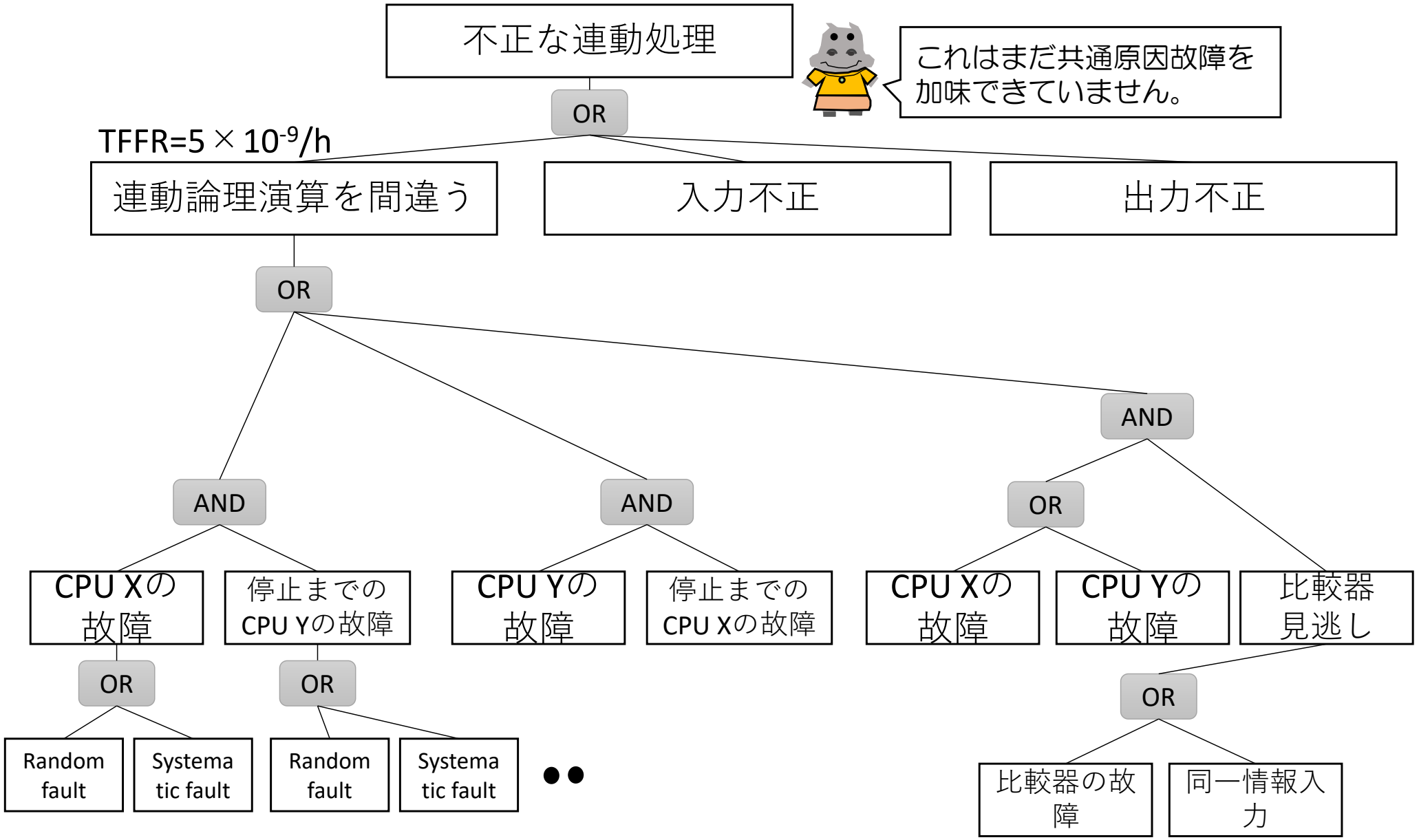
こういうのをよく見ますね。比較器において、CPU X, Yの一致が確認できないと、システムを止めるという構成です。独立したCPU XとYがほぼ同時に故障する可能性は低いため、CPUは比較的緩い故障頻度許容となります。比較器の方は最後の砦なので、比較器が故障した場合、安全側に遷移するような回路構成(inherent fail-safety)として、厳しめのTFFRを満たすようにします。

4. 安全機能の実装部が高いSSIL？

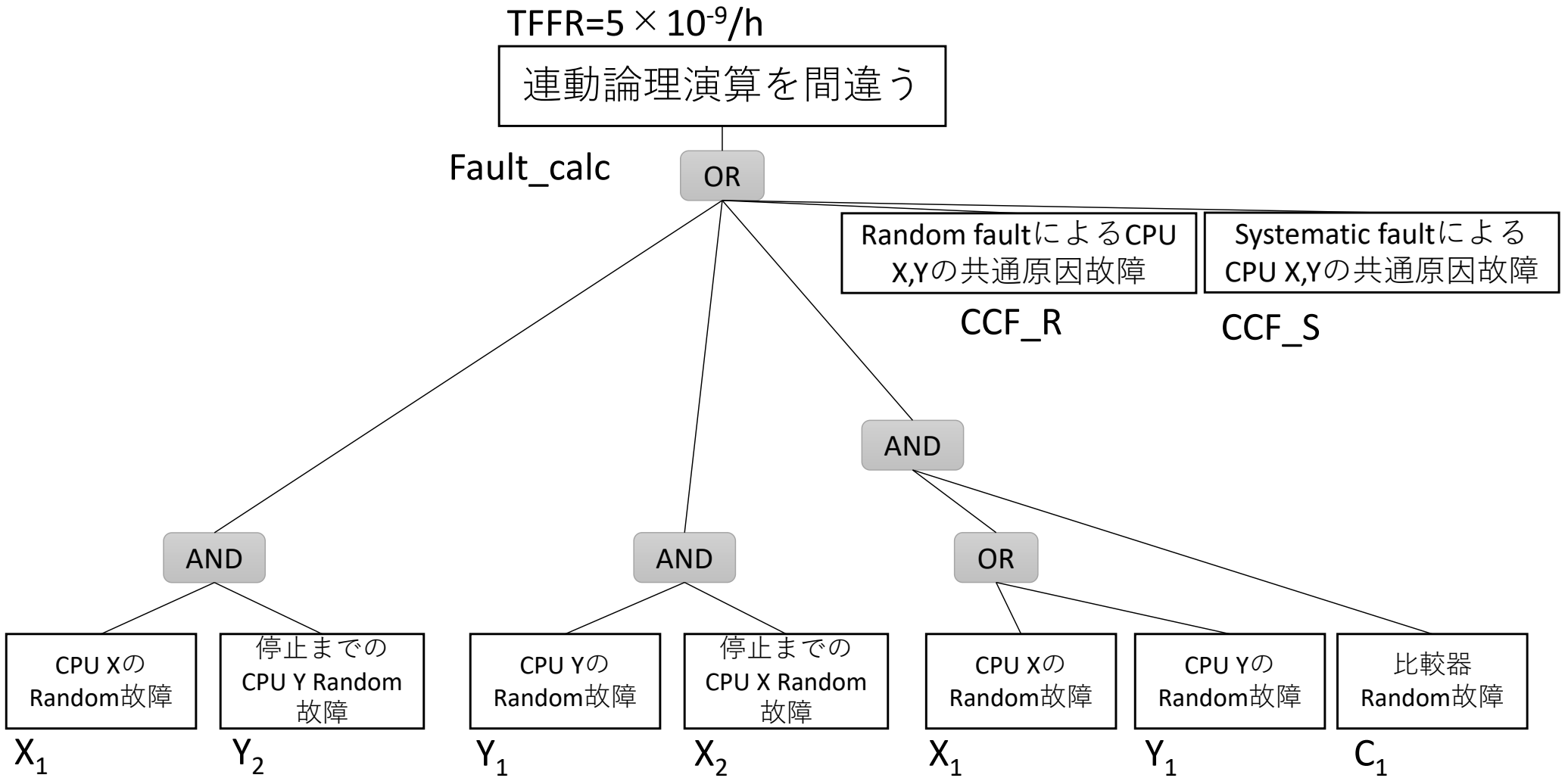


ちょっとまて！なんかおかしいぞ。
比較器のソフトウェアだけ気を付けたら、システマティック故障がなくなるなんて、そんなわけないやないか。
そもそも、ここにソフトなんかない場合まであるぞ。
CPUにロードするソフトウェアがどうでもいいとか、絶対にオレの感覚と違うんやけど。。

4. SIL/SSILの割り当てを再確認しよう



4. SIL/SSILの割り当てを再確認しよう



$Fault\_calc = X_1 * Y_2 + X_2 * Y_1 + (X_1 + Y_1) * C_1 + CCF\_R + CCF\_S$

$X_1 = Y_1$, $X_2 = Y_2$とすると(CPUなり回路なりが同一)

$Fault\_calc = 2X_1 * X_2 + 2X_1 * C_1 + CCF\_R + CCF\_S = 2X_1 (X_2 + C_1) + CCF\_R + CCF\_S$

4. SIL/SSILの割り当てを再確認しよう

TFFR=5 × 10^{-9}/h

連動論理演算を間違う

Fault_calc= $2X_1 (X_2 + C_1)$ +CCF_R+CCF_S

一つ目のCPUが壊れるのは仕方ない

2つ目のCPUが壊れるのは避けたい。
1つ目が壊れたら、2つ目が壊れる前に止める作戦が良い

比較器が危険側に壊れたらおしまい。危険側に壊れないようにInherent fail-safetyで回路構成する。

CPUがいくら2つあっても、
共通のランダム故障部分があっては意味がない

ソフトウェアや、データはCPUどっちも共通
CPUに実装する安全関連機能は注意深く
TFFRから導かれるSILに応じて構築！

FTA解析と論理演算をシンプルな形に変形すれば、わかりやすく、何を重視すればいいか一目瞭然です。
FTAは、作りっぱなしではほぼ役に立たないです。

4. SIL/SSILの割り当てを再確認しよう

連動論理演算を間違う

TFFR=5 × 10^{-9}/h

Fault_calc= $2X_1 (X_2 + C_1)$ +CCF_R+CCF_S

ユニット寿命15年

CPUユニットの故障頻度を10年に一度と保守的に設定すると、一時間当たり、1×10^{-5}/hの故障頻度となる。

Detection and Negationを100msとすると、2つ目のCPUがその間に故障する確率は3.1×10^{-10}。

比較器のライフサイクル15年、Inherent fail-safetyの場合は、IEC 62425により、危険側故障頻度を0としてもよいが、保守的に1×10^{-9}/h (=SIL 4)とするとライフサイクル全体で危険側故障する確率は、1.3×10^{-4}.
このため、$2X_1 (X_2 + C_1)$ の項は、
$2(1\times10^{-5})(3.1\times10^{-10}+1.3\times10^{-4})$でほぼ$2.6\times10^{-9}$/h
これによりTFFRを満たしていると判断。

共通原因故障は、規格の規定により独立させることにより解消。

TFFRを加味して考えると、連動論理演算を間違ってしまう可能性のあるソフトウェア機能は、SSIL 4

4. SIL/SSILの割り当てを再確認しよう

そういう目で改めて、SSILの割り当てについて書いている規格について確認してみましょう。

4.3 The required software safety integrity level shall be decided and assessed at system level, on the basis of the system safety integrity level and the level of risk associated with the use of the software in the system.

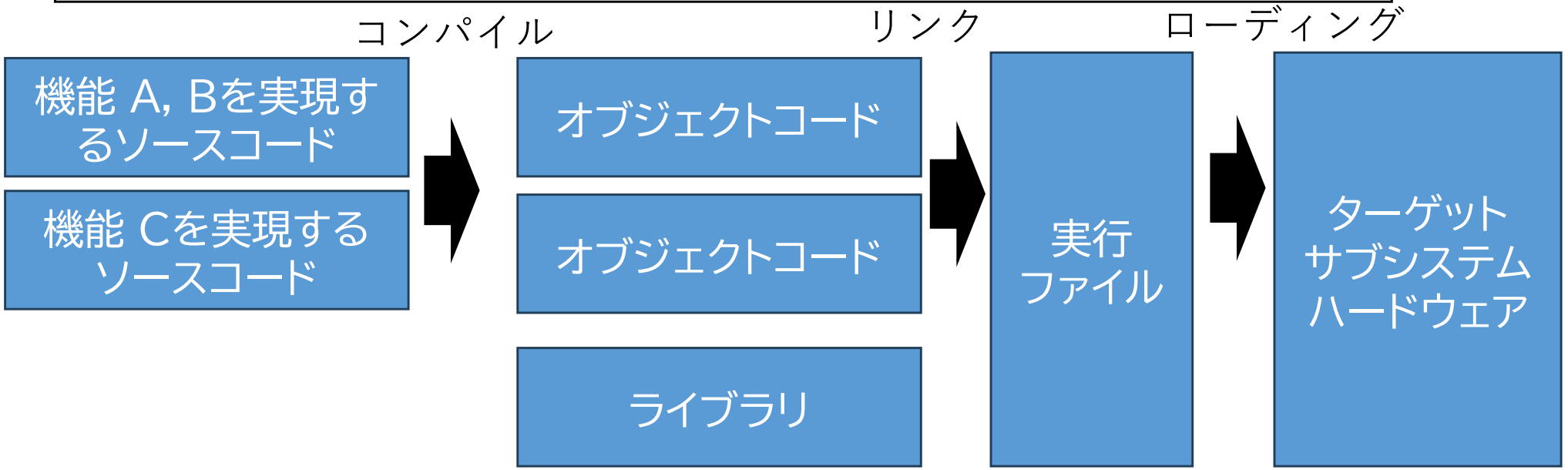
- IEC 62279

- 要求されるSSILは、システムレベルで決めて調査せよ。
- システムSILと、システムにおけるソフトウェアの使い方に関するリスクを基準に決めろ。

→CPU自体のTFFRは低いから、SSIL0とかそういうのではなくて
システムレベルで調査し、決めることになる。

4. SIL/SSILの割り当てを再確認しよう

TFFRから考えると、機能AはSIL3，機能BはSIL4、機能CはSIL2ですね。そんでもって、その機能は、ソフトで実現し、リンクされて一つのExecutable fileになるってわけ。
これって、SSIL分けて作る普通？？



NOTE The allocation of safety integrity requirements could lead to different SIL for well-separated software and hardware parts of a subsystem. This allocation depends on the contribution of the software and hardware parts of the subsystem to the safety-related functions and on the mechanisms for the failure mitigation including the separation of function with different SIL. IEC 62279 4.1

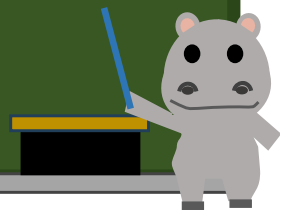
Where the software consists of components of different software safety integrity levels then all of the software components shall be treated as belonging to the highest of these levels unless there is evidence of independence between the higher software safety integrity level components and the lower software safety integrity level components. IEC 62279 7.3.4.9

サブシステムハードウェアが共通だったら、その中でSSILを分けて作るっていうのは制約が大きいから、結局サブシステムレベルで決める(→システムレベルの解析)ということになりそうです。

5. どのような手順で開発するのか

そろそろワシの大好きな「管理」について話しはないんか？

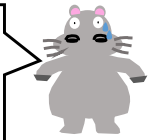
4 Objectives, conformance and software safety integrity levels	ソフトウェアのSILを決める方法
5 Software management and organisation	SSILに応じた組織、能力 またライフサイクルの定義
6 Software assurance	ソフトウェアの品質確保のための試験、V&V及びアセスメント手法
7 Generic software development	ソフトウェア開発のブレークダウン及びSSILに応じた手法の選択と実装
8 Development of application data or algorithms: systems configured by application data or algorithms	開発ソフトウェアにパラメータを入れることにより個別の客先への対応
9 Software deployment and maintenance	導入と保守の管理手法



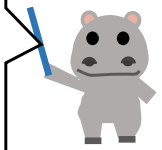
5. どのような手順で開発するのか（組織と能力）

安全性が強度に必要なソフトウェアは、出来るヤツに任せたい。

ハ～！オレの時代が来たわ！給料アップ頼むで社長！



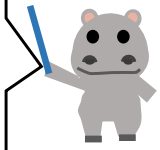
それね、いつもお客さんに「ウチはできるカバで仕事するから単価高いんです」と言うんだけど、「じゃあできるカバってどういう条件やねん。カバおくん給料高いんかぁ、うらやましいな。」と嫌味を言われていつも終わりなんだけど。出来るって何？



この規格の考え方は

職務に応じた必要能力の差は、安全性の上下で付けていません。
例えば：Designerは、SSILが変わっても要件は同一です。

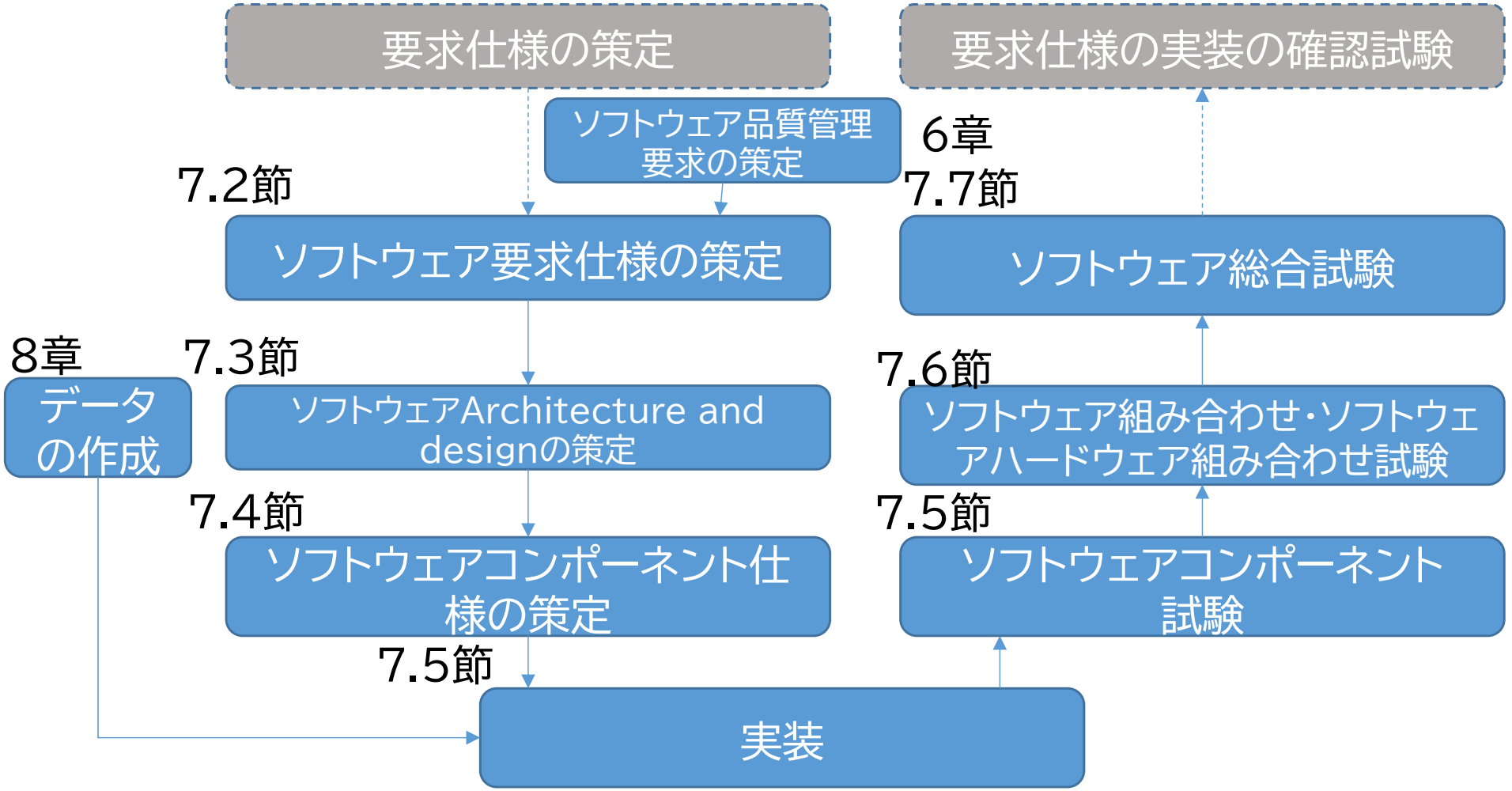
そんなんやったら、どこで差をつけるんや！安全に関係ないから安くしろっていつもお客さんに言われてんねんで。一回経営者になってみたらええねん。



この規格の考え方は

安全性が必要なほど、チェックするカバを別の者にして、独立性を高めることで担保しようとしているんですよ。別の目ってやつです。
安全性があまり要らないものは、兼務出来るってことです。

5. どのような手順で開発するのか（要求仕様から実装、試験）



上位で決まったことが下位にすべて含まれ、矛盾がないことを確認することで、要求仕様を実現する。

5. どのような手順で開発するのか
（プログラム開発 7章の中身）

「ソフトウェア要求段階（7.2節）」、「ソフトウェアArchitecture and design段階（7.3節）」、「ソフトウェアコンポーネント段階（7.4節）」と細分化されるのですが、やることはほぼ変わらないです。

目的を把握する
7.x.1に書かれている。

入出力される文書を確認する
7.x.2,3に書かれている。どの文書をもとにしなければならないか分かる。

やることを確認する
7.x.4に書かれている。
- 仕様を作る
- 試験仕様を作る
- 確認（Verification）をする

この3点が順番に述べられている。

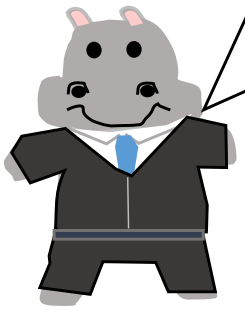
小さなことからコツコツと

具体的な方法

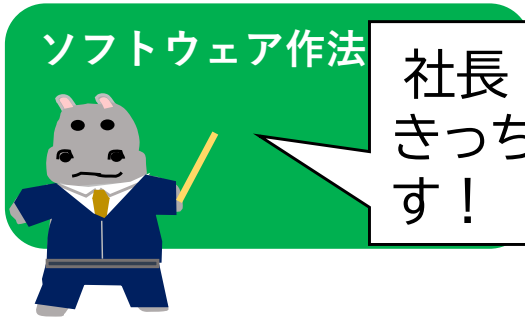
5章及び6章、Annex Aに沿って実施する

6. これはちょっと、本末転倒

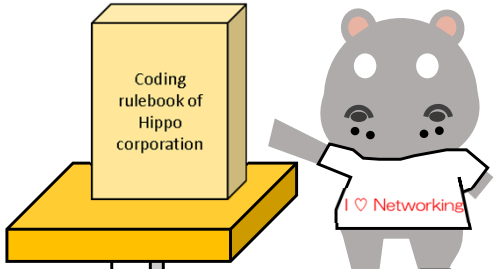
立派すぎるルールは正直怪しい。事の本質をとらえているかどうかじゃないでしょうか。
別に認証機関が泣いて喜ぶようなものはいらないです。



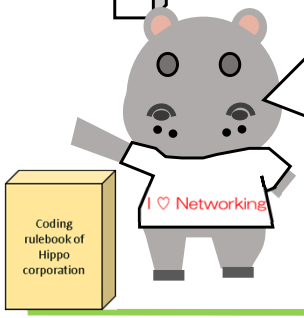
オオカバ
大蒲教授、我々カバ興業のために、認証機関が泣いて喜ぶような先進的なコーディングルール作成よろしくお願いします。



社長！お任せください。きっちりさせて頂きます！

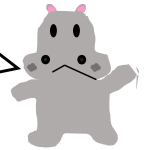


大蒲教授の渾身の作だけど
英語だし、内容は難しいし、これを協力会社まで完全に理解してもらうのは絶対無理。。。。



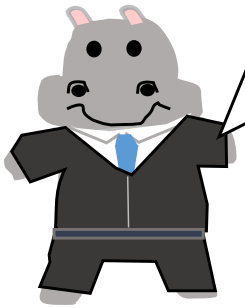
我がカバ興業は新コーディングルールを作った。IEC 62279に完全に合致しているし、世の中の主要コーディングルールの内容も矛盾なく包含している。これを守ることが下請負契約の条件だ！

ハイ分かりました。
(守らんけど)



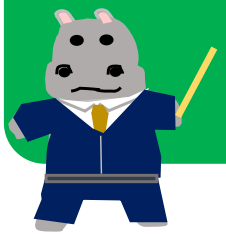
6. このようなルールはどうですか

現実的で無理なく守れることが重要と思います。



オオカバ
大蒲教授、我々カバ興業のために、品質が向上でき、みんなから喜ばれるようなコーディングルール作成よろしくお願いします。

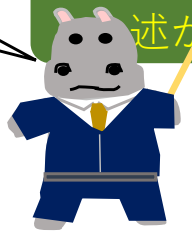
ソフトウェア作法

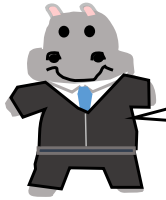


社長！分かりました。
現場の皆さんの意見も聞いて作ります。

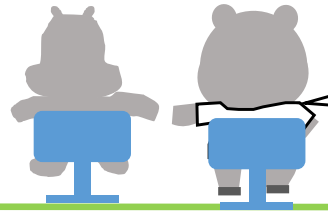
禁止の理由はこれです。実務との差異はありませんか？

再帰呼び出しのメリットデメリット
メリット：短いコードで実現できる。
デメリット：メモリが呼び出すごとに消費され、文法やアルゴリズムが正しくても動作が正常に行われない場合がある。
ルールXX:再帰呼び出しは他の記述が出来る場合は禁止





これなら何とかなりそうだ。



これは大丈夫です。
守れそうです。

7. まとめ

- どんな故障が保安システムにはあるのか？

→一般的に、「どの頻度で危険側に遷移するのか？」ということに興味が行きますが、ソフトウェアについては、そうではなく、ヒューマンエラーを中心にした、決定論的なSystematic faultを防止します。

- ソフトウェアの観点では、故障を防ぐためにはどうすればいいか？

→管理です。しかしカバ社長のように何でもかんでもではなく、要求される機能の重篤度をシステムレベルで考えて調整します。

- SILやTFFRと、ソフトウェアの安全性インテグリティの関係

→整合して決定します。但し、部品やアイテム単位の許容故障頻度で決めるのではなく、サブシステムレベルのSILを考慮し決定します。

- 開発手法と規格

→規格では、SSILの決め方、体制、教育、管理手法をのべて、その次にソフトウェアをブレークダウンし、次に各お客さんごとの対応をするためのパラメータを入れるという順番で述べられています。