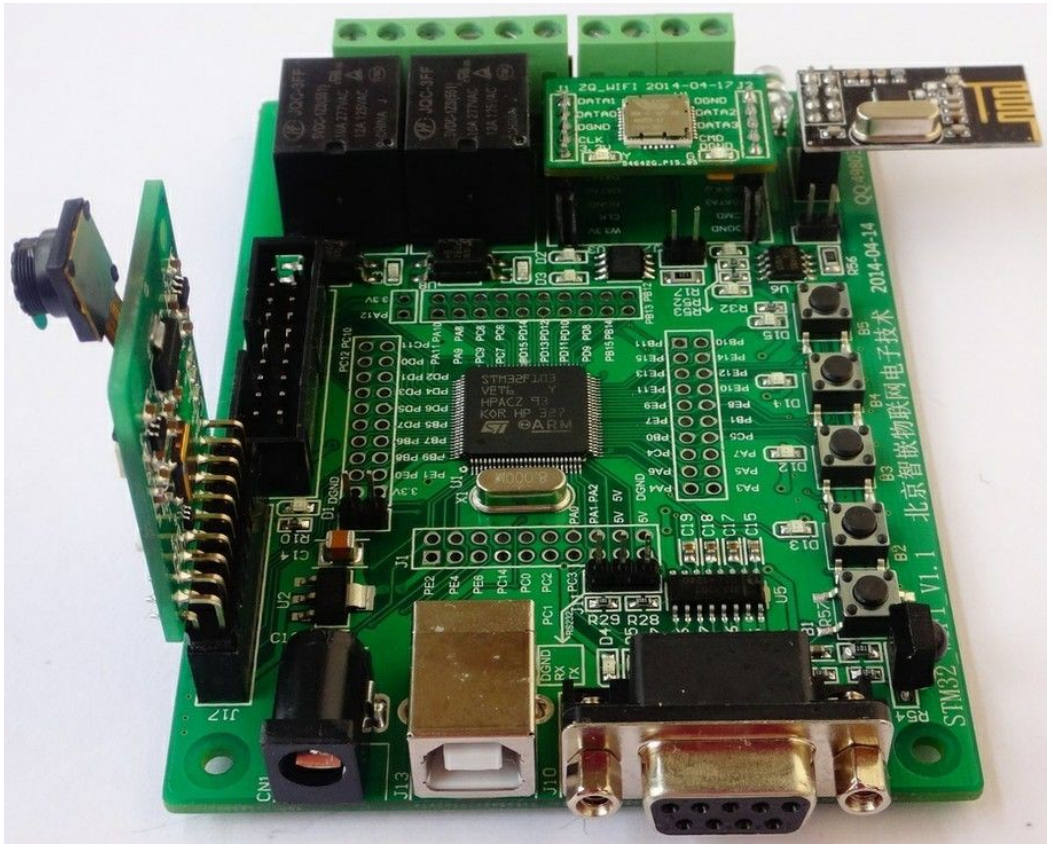
Kiedy od softu zależy ludzkie życie o systemach safety-critical

Maciej Gajdzica

Solw'IT | Let's Solve It



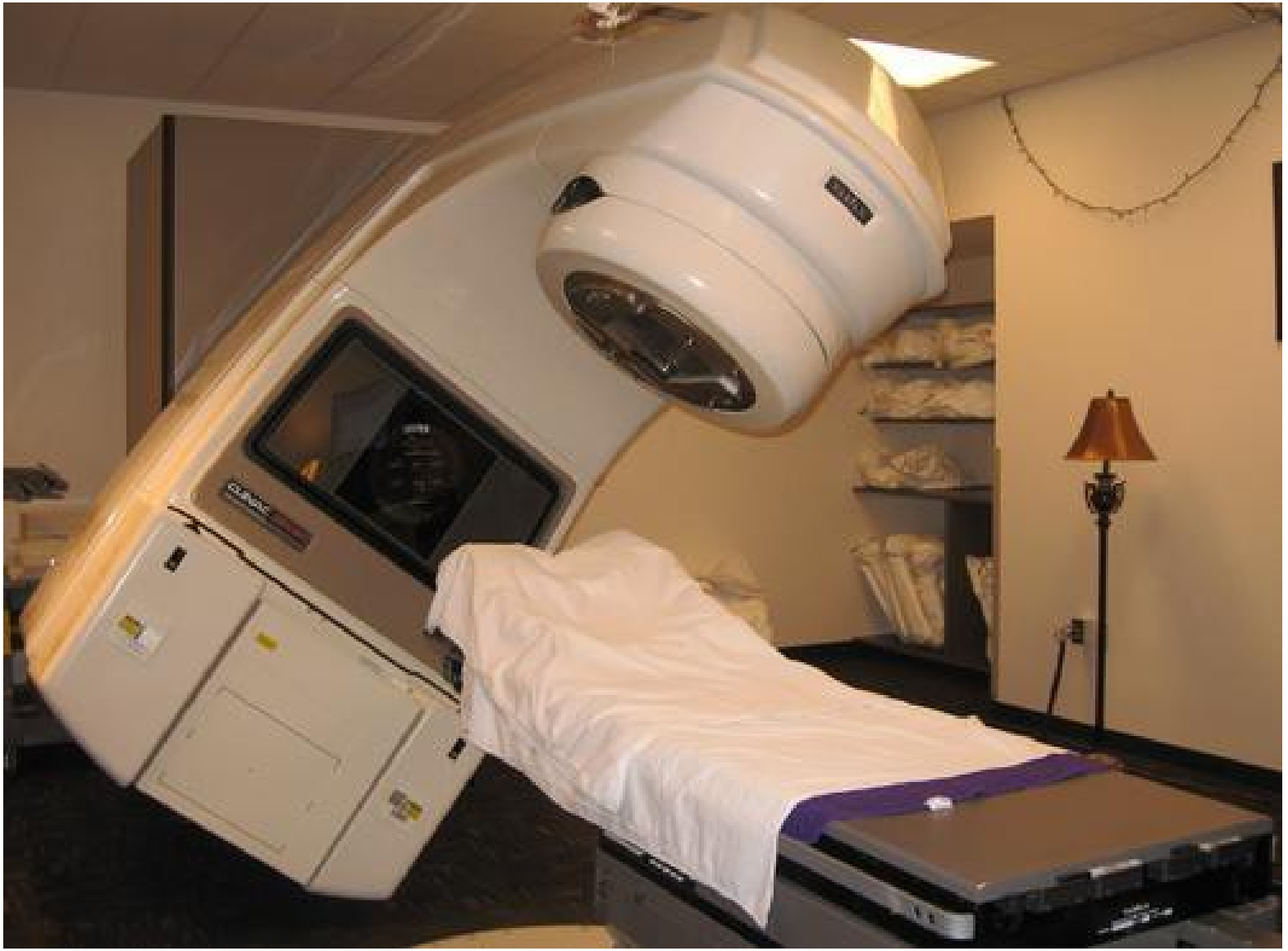
ucgosu.pl

Przyzwyczailiśmy się, że programy zawierają błędy

„Will we continue on our undisciplined course, blown by the chaotic winds of business and government, **until one of us finally blunders badly enough to wake the sleeping giant of government regulation?**”

Robert C. Martin

Therac-25

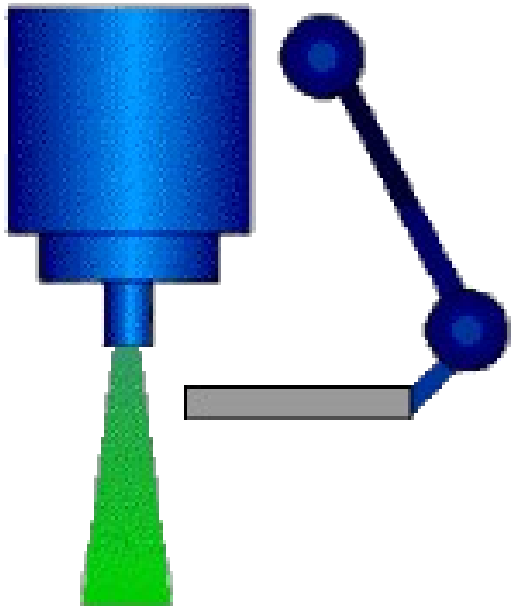


Therac-25

- W latach 1983-87 działało 11 maszyn tego typu
- Odnotowano 6 wypadków (5 osób zmarło)
- Aparatura podawała sto razy większą dawkę promieniowania
- Przeciwko producentowi toczył się proces w sądzie
- Sprawą zainteresowała się opinia publiczna i FDA

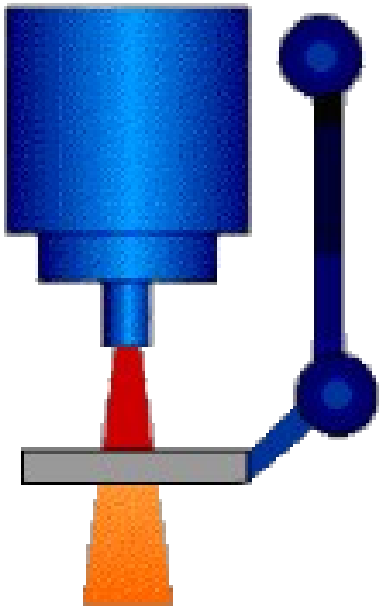
Therac-25

low current
electron beam
was scanned
across the field



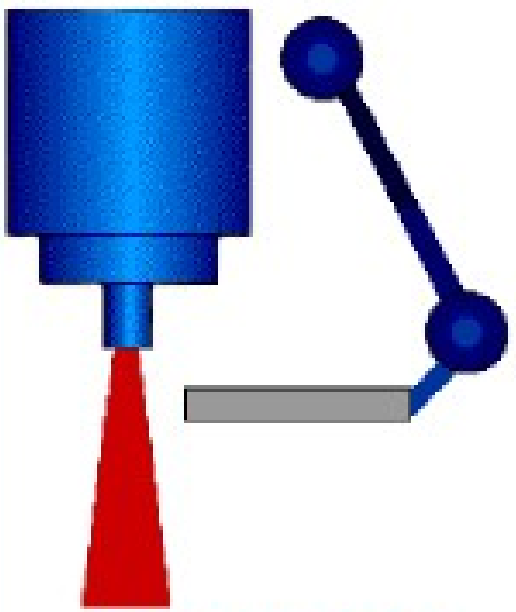
Electron Mode

high current
electron beam
was tracked
at the target



X-Ray Mode

high current
electron beam
with no target
> 'lightning'



THE PROBLEM

tray including the target, a flattening filter, the collimator jaws and an ion chamber was moved OUT for "electron" mode, and IN for "photon" mode.

Therac-25

- Jedna osoba zaimplementowała cały program
- Praktycznie zerowa dokumentacja
- Zrezygnowano z hardware'owych zabezpieczeń – uznano, że są nadmiarowe, bo software nie zawiera błędów
- Integracja SW/HW nie była testowana podczas developmentu
- Producent ignorował zgłaszane problemy

Ariane 5

Ariane 5

- Bezzałogowa rakieta Europejskiej Agencji Kosmicznej
- Podczas pierwszego lotu 4 czerwca 1996 wybuchła niedługo po starcie
- Rakieta nagle obróciła się o 90 stopni i oderwały się jej silniki odrzutowe
- Spowodowało to włączenie mechanizmu autodestrukcji

Ariane 5

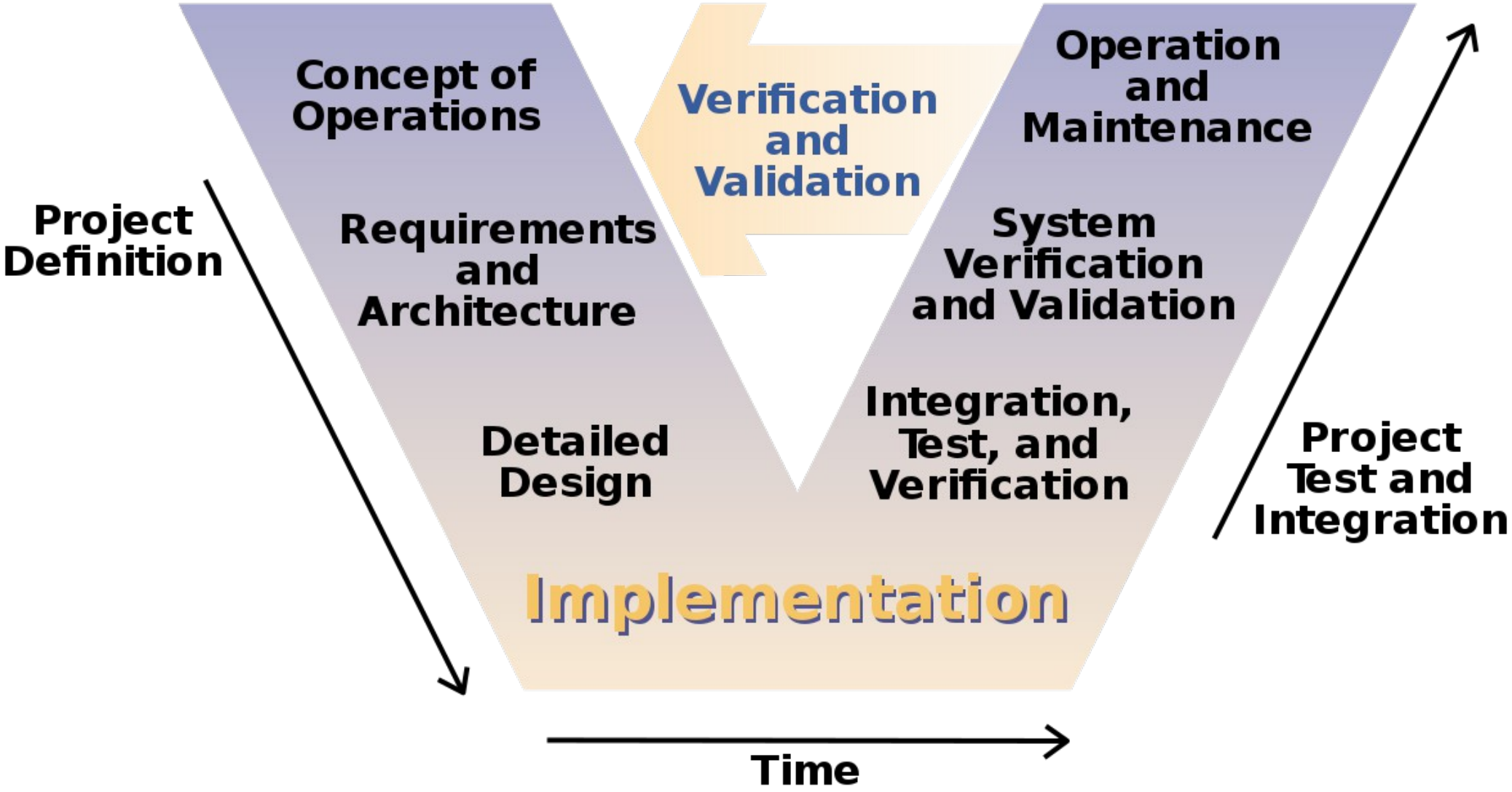
- Konwersja double → int16 zbyt dużej wartości spowodowała overflow
- Była to część kodu odpowiedzialnego za określanie pozycji rakiety w przestrzeni
- Rakieta zawierała dwa takie same moduły z tym samym błędem
- Zmienna, która spowodowała wypadek nie była w ogóle potrzebna
- Kod skopiowano z Ariane 4
- Brak testów symulacyjnych tego modułu

Błędy są nie do uniknięcia, potrzebny jest odpowiedni proces, aby nie przedostały się one do końcowego produktu

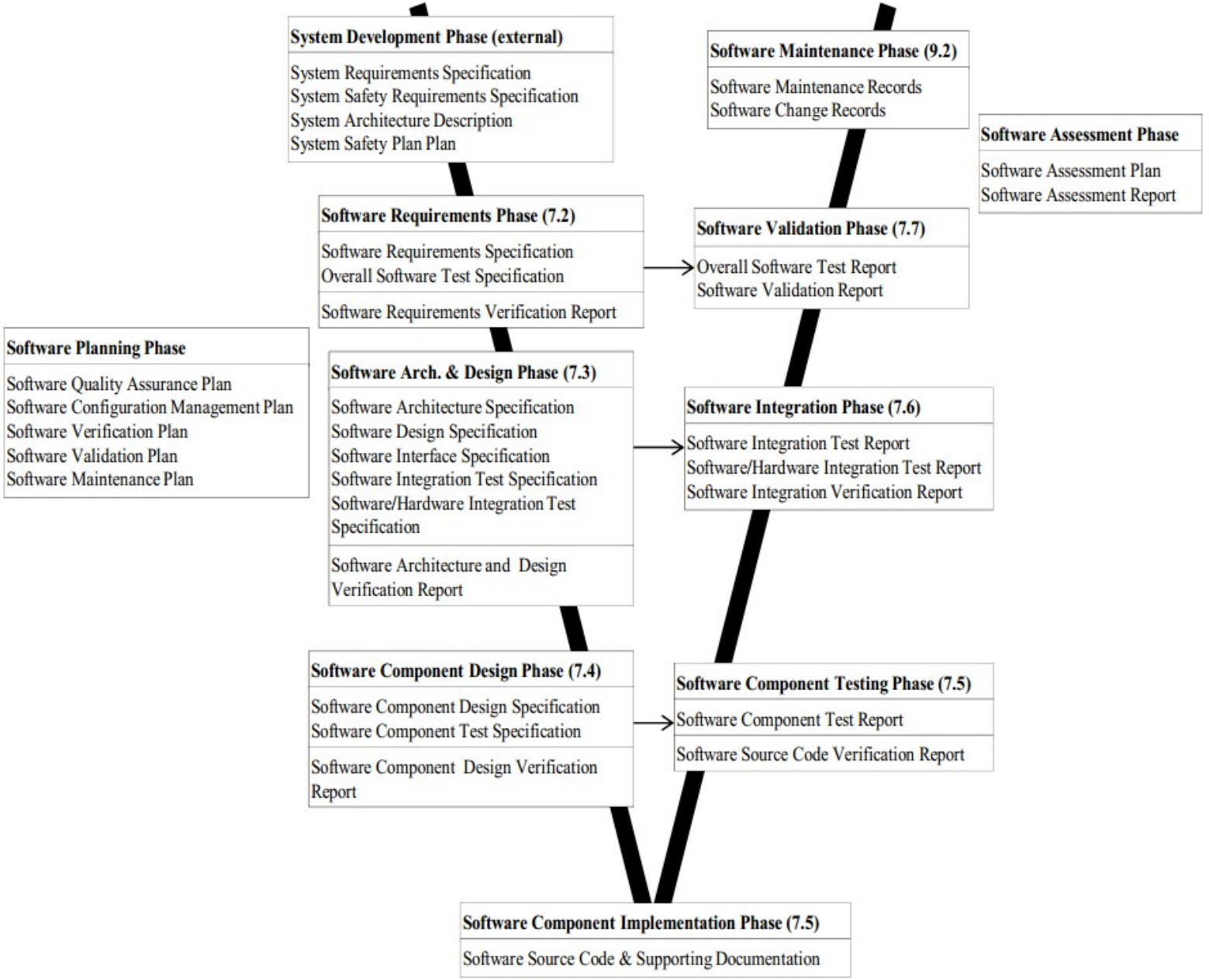
Procesy są opisane w normach

- IEC 61508: Ogólnie o systemach safety-critical
- IEC 62304: Systemy medyczne
- ISO 26262: Branża automotive
- IEC 61513: Elektrownie atomowe
- EN 50128: Branża kolejowa
- DO-178C: Branża lotnicza
- NASA Safety Critical Guidelines

V-model


Concept of Operations
Requirements and Architecture
Detailed Design
Implementation
Integration, Test, and Verification
System Verification and Validation
Operation and Maintenance
Verification and Validation
Project Definition
Project Test and Integration
Time

V-model


System Development Phase (external)
System Requirements Specification
System Safety Requirements Specification
System Architecture Description
System Safety Plan Plan
Software Maintenance Phase (9.2)
Software Maintenance Records
Software Change Records
Software Assessment Phase
Software Assessment Plan
Software Assessment Report
Software Requirements Phase (7.2)
Software Requirements Specification
Overall Software Test Specification
Software Requirements Verification Report
Software Validation Phase (7.7)
Overall Software Test Report
Software Validation Report
Software Planning Phase
Software Quality Assurance Plan
Software Configuration Management Plan
Software Verification Plan
Software Validation Plan
Software Maintenance Plan
Software Arch. & Design Phase (7.3)
Software Architecture Specification
Software Design Specification
Software Interface Specification
Software Integration Test Specification
Software/Hardware Integration Test Specification
Software Architecture and Design Verification Report
Software Integration Phase (7.6)
Software Integration Test Report
Software/Hardware Integration Test Report
Software Integration Verification Report
Software Component Design Phase (7.4)
Software Component Design Specification
Software Component Test Specification
Software Component Design Verification Report
Software Component Testing Phase (7.5)
Software Component Test Report
Software Source Code Verification Report
Software Component Implementation Phase (7.5)
Software Source Code & Supporting Documentation

„It does not require that any particular lifecycle model is used, but it does require that the plan include certain ACTIVITIES and have certain ATTRIBUTES.”

IEC 62304 (norma medyczna)

Poziomy bezpieczeństwa

Safety Integrity Level	Probability of Dangerous Failure per hour
SIL 4	>= 10^{-9} to 10^{-8}
SIL 3	>= 10^{-8} to 10^{-7}
SIL 2	>= 10^{-7} to 10^{-6}
SIL 1	>= 10^{-6} to 10^{-5}

Software Level	Effect of software anomalous behavior
Level A	Multiple loss of life, usually with loss of aircraft
Level B	The aircraft, or crew, is less capable to deal with adverse operating conditions
Level C	The aircraft, or crew, is less able to deal with unfavorable operational conditions
Level D	No significant reduction in the aircraft's level of safety
Level E	*No effect on safety*

DO-178C Software Levels

Norma medyczna (62304):
Class A: No injury or damage to health is possible
Class B: Non-SERIOUS INJURY is possible
Class C: Death or SERIOUS INJURY is possible

SIL4 – zagrożenie dla życia wielu ludzi.
SIL3 – zagrożenie dla życia pojedynczej osoby.
SIL2 – zagrożenie poważnego uszczerbku na zdrowiu.
SIL1 – zagrożenie drobnego urazu.

Jak osiągnąć wymagane prawdopodobieństwo błędu w sofcie?

Nie możemy go zmierzyć dopóki produkt nie zacznie działać.

Możemy jednynie stosować pewne sprawdzone techniki.

Table A.4– Software Design and Implementation (7.4)

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Formal Methods	D.28	-	R	R	HR	HR
2. Modelling	Table A.17	R	HR	HR	HR	HR
3. Structured methodology	D.52	R	HR	HR	HR	HR
4. Modular Approach	D.38	HR	M	M	M	M
5. Components	Table A.20	HR	HR	HR	HR	HR
6. Design and Coding Standards	Table A.12	HR	HR	HR	M	M
7. Analysable Programs	D.2	HR	HR	HR	HR	HR
8. Strongly Typed Programming Language	D.49	R	HR	HR	HR	HR
9. Structured Programming	D.53	R	HR	HR	HR	HR
10. Programming Language	Table A.15	R	HR	HR	HR	HR
11. Language Subset	D.35	-	-	-	HR	HR
12. Object Oriented Programming	Table A.22 D.57	R	R	R	R	R
13. Procedural programming	D.60	R	HR	HR	HR	HR
14. Metaprogramming	D.59	R	R	R	R	R

Table A.5 – Verification and Testing (6.2 and 7.3)

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Formal Proof	D.29	-	R	R	HR	HR
2. Static Analysis	Table A.19	-	HR	HR	HR	HR
3. Dynamic Analysis and Testing	Table A.13	-	HR	HR	HR	HR
4. Metrics	D.37	-	R	R	R	R
5. Traceability	D.58	R	HR	HR	M	M
6. Software Error Effect Analysis	D.25	-	R	R	HR	HR
7. Test Coverage for code	Table A.21	R	HR	HR	HR	HR
8. Functional/ Black-box Testing	Table A.14	HR	HR	HR	M	M
9. Performance Testing	Table A.18	-	HR	HR	HR	HR
10. Interface Testing	D.34	HR	HR	HR	HR	HR

Requirements:

1) For software Safety Integrity Levels 3 and 4, the approved combination of techniques is 3, 5, 7, 8 and one from 1, 2 or 6.

2) For Software Safety Integrity Level 1 and 2, the approved combinations of techniques is 5 together with one from 2, 3 or 8.

NOTE 1 Techniques/measures 1, 2, 4, 5, 6 and 7 are for verification activities.

NOTE 2 Techniques/measures 3, 8, 9 and 10 are for testing activities.

Table A.12 – Coding Standards

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. Coding Standard	D.15	HR	HR	HR	M	M
2. Coding Style Guide	D.15	HR	HR	HR	HR	HR
3. No Dynamic Objects	D.15	-	R	R	HR	HR
4. No Dynamic Variables	D.15	-	R	R	HR	HR
5. Limited Use of Pointers	D.15	-	R	R	R	R
6. Limited Use of Recursion	D.15	-	R	R	HR	HR
7. No Unconditional Jumps	D.15	-	HR	HR	HR	HR
8. Limited size and complexity of Functions, Subroutines and Methods	D.38	HR	HR	HR	HR	HR
9. Entry/Exit Point strategy for Functions, Subroutines and Methods	D.38	R	HR	HR	HR	HR
9. Limited number of subroutine parameters	D.38	R	R	R	R	R
10. Limited use of Global Variables	D.38	HR	HR	HR	M	M
Requirement: 1) It is accepted that techniques 3, 4 and 5 may be present as part of a validated compiler or translator.						

Table A.15 – Textual Programming Languages

TECHNIQUE/MEASURE	Ref	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
1. ADA	D.54	R	HR	HR	HR	HR
2. MODULA-2	D.54	R	HR	HR	HR	HR
3. PASCAL	D.54	R	HR	HR	HR	HR
4. C or C++	D.54 D.35	R	R	R	R	R
5. PL/M	D.54	R	R	R	NR	NR
6. BASIC	D.54	R	NR	NR	NR	NR
7. Assembler	D.54	R	R	R	R	R
8. C#	D.54 D.35	R	R	R	R	R
9. JAVA	D.54 D.35	R	R	R	R	R
10. Statement List	D.54	R	R	R	R	R

Requirements:

1) The selection of the languages shall be based on the requirements given in 6.7 and 7.3.

2) There is no requirement to justify decisions taken to exclude specific programming languages.

NOTE 1 For information on assessing the suitability of a programming language see entry in D.54 'Suitable Programming Languages'.

NOTE 2 If a specific language is not in the table, it is not automatically excluded. It should, however, conform to D.54.

NOTE 3 Run-time systems associated with selected languages which are necessary to run application programs should still be justified for usage according to the Software Safety Integrity Level.

Bezpieczeństwo jako element systemu

- Nie da się najpierw zapewnić funkcjonalności, a dopiero potem dodać bezpieczeństwa
- Bezpieczeństwo poszczególnych elementów nie gwarantuje bezpieczeństwa całego systemu
- Bezpieczeństwo musimy mieć na uwadze od samego początku projektu

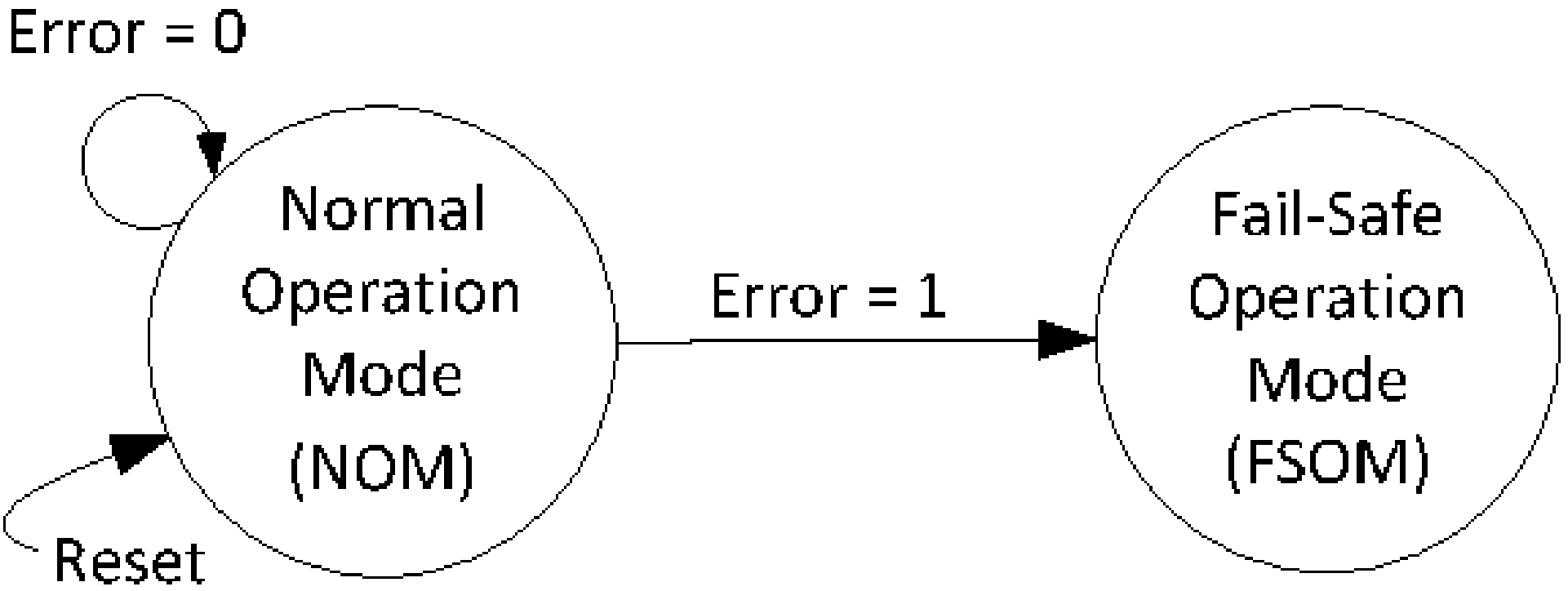
Analiza ryzyka

RISK ASSESSMENT MATRIX				
SEVERITY / PROBABILITY	Catastrophic (1)	Critical (2)	Marginal (3)	Negligible (4)
Frequent (A)	High	High	Serious	Medium
Probable (B)	High	High	Serious	Medium
Occasional (C)	High	Serious	Medium	Low
Remote (D)	Serious	Medium	Medium	Low
Improbable (E)	Medium	Medium	Medium	Low
Eliminated (F)	Eliminated			

Składowe systemu

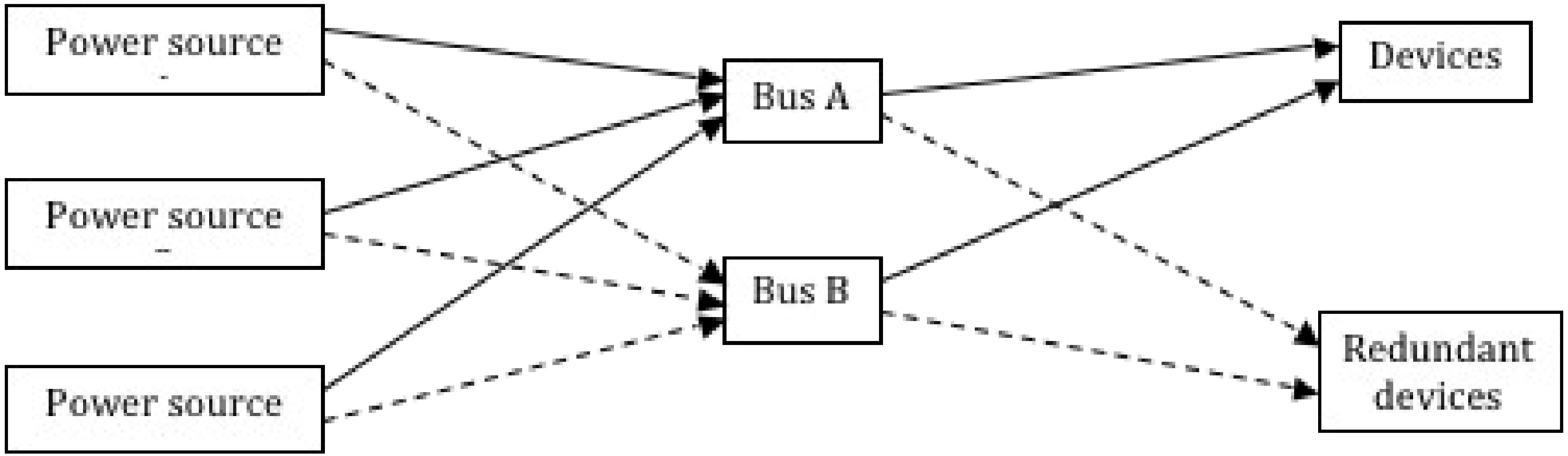
- Hardware
- Software
- Procedury i ludzie

Safe state

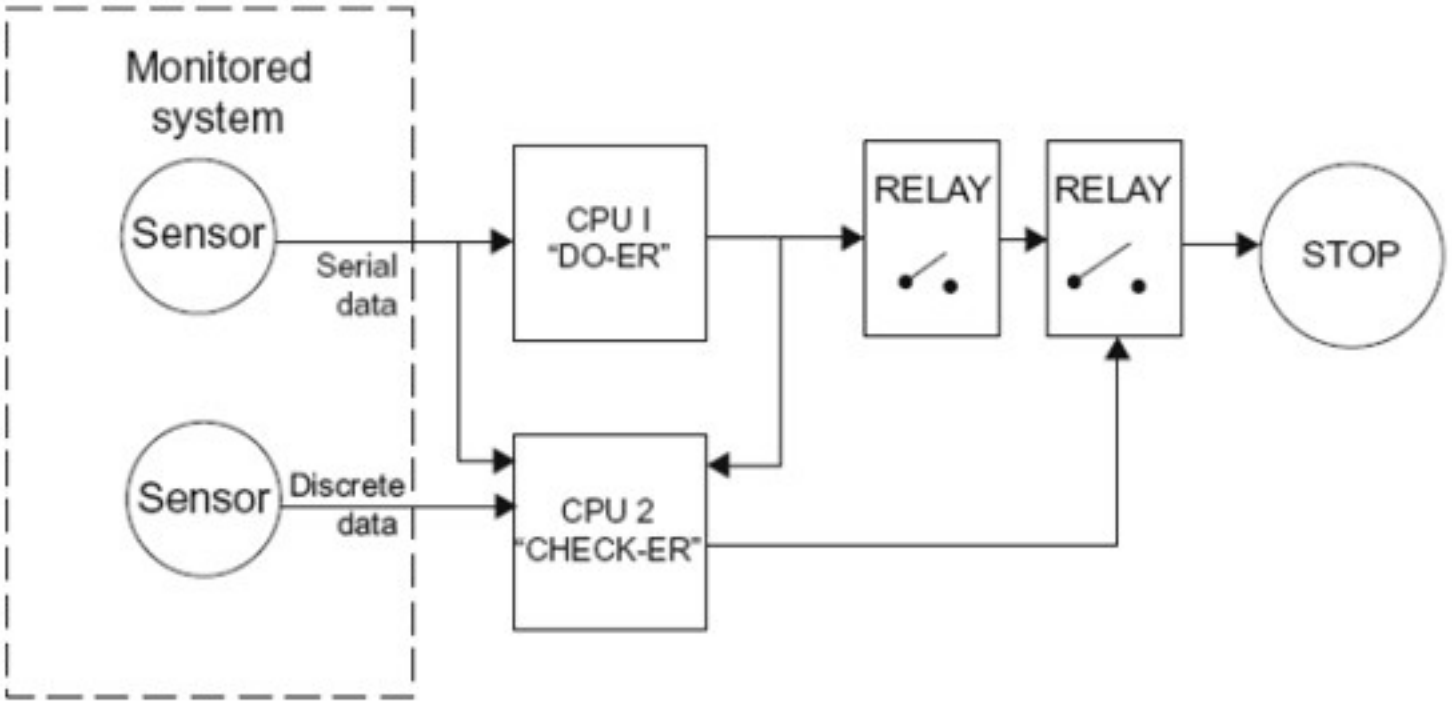

Error = 0
Normal Operation Mode (NOM)
Error = 1
Fail-Safe Operation Mode (FSOM)
Reset

Safety-critical vs mission-critical

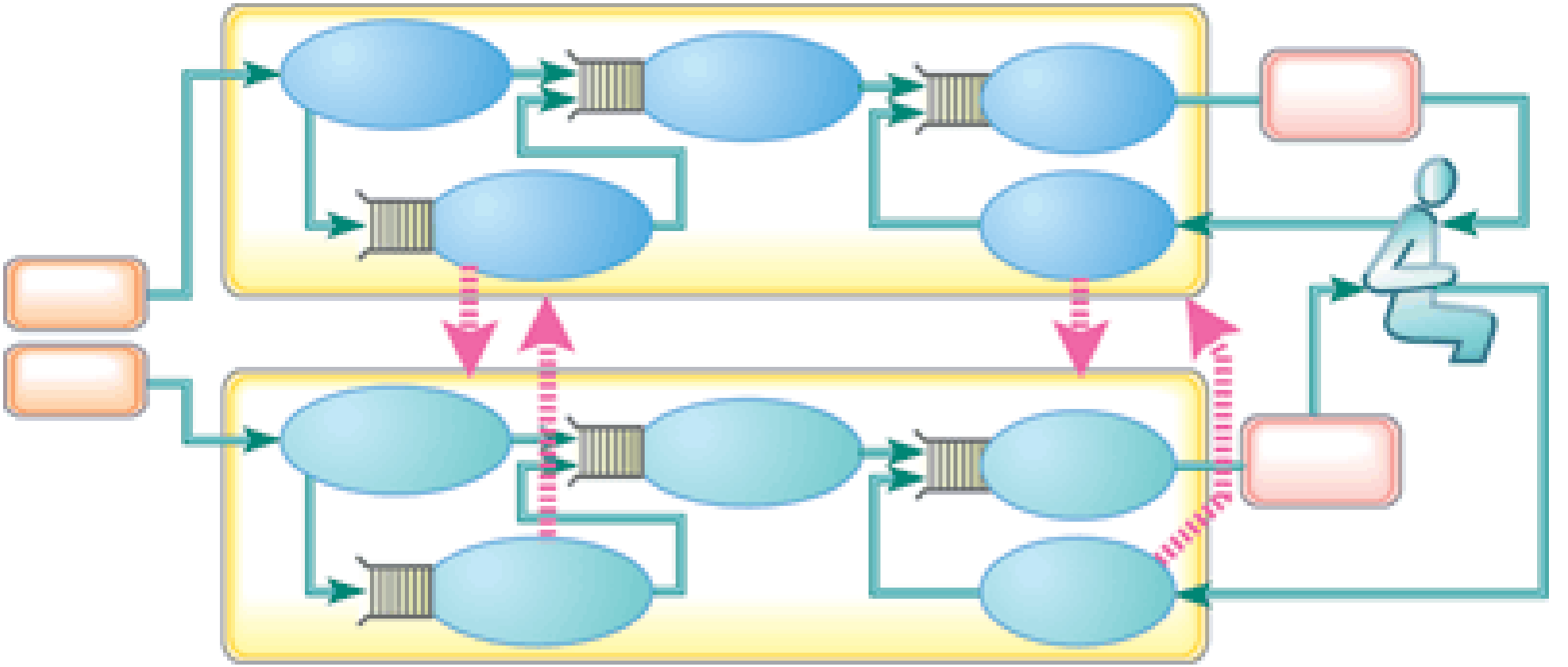
Redundancja



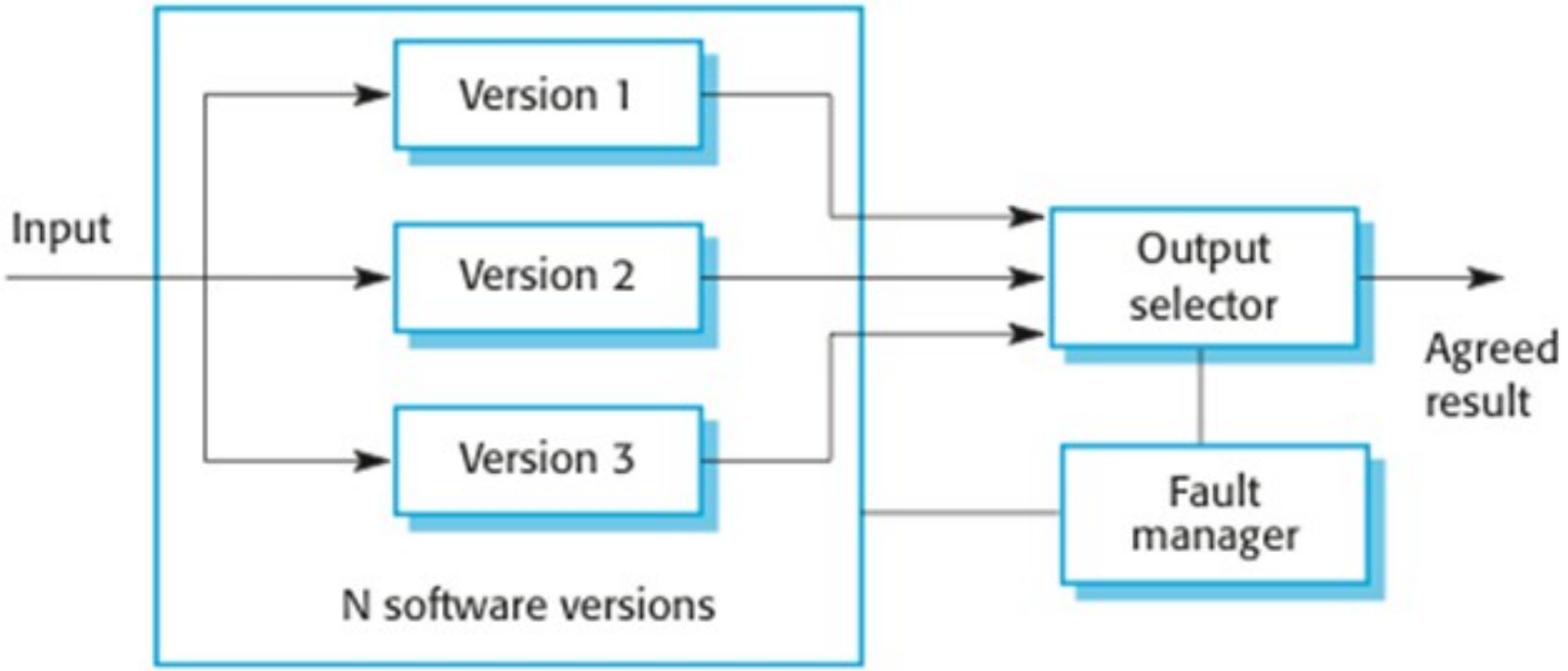
Procesor nadzorujący



Dwa niezależne kanały



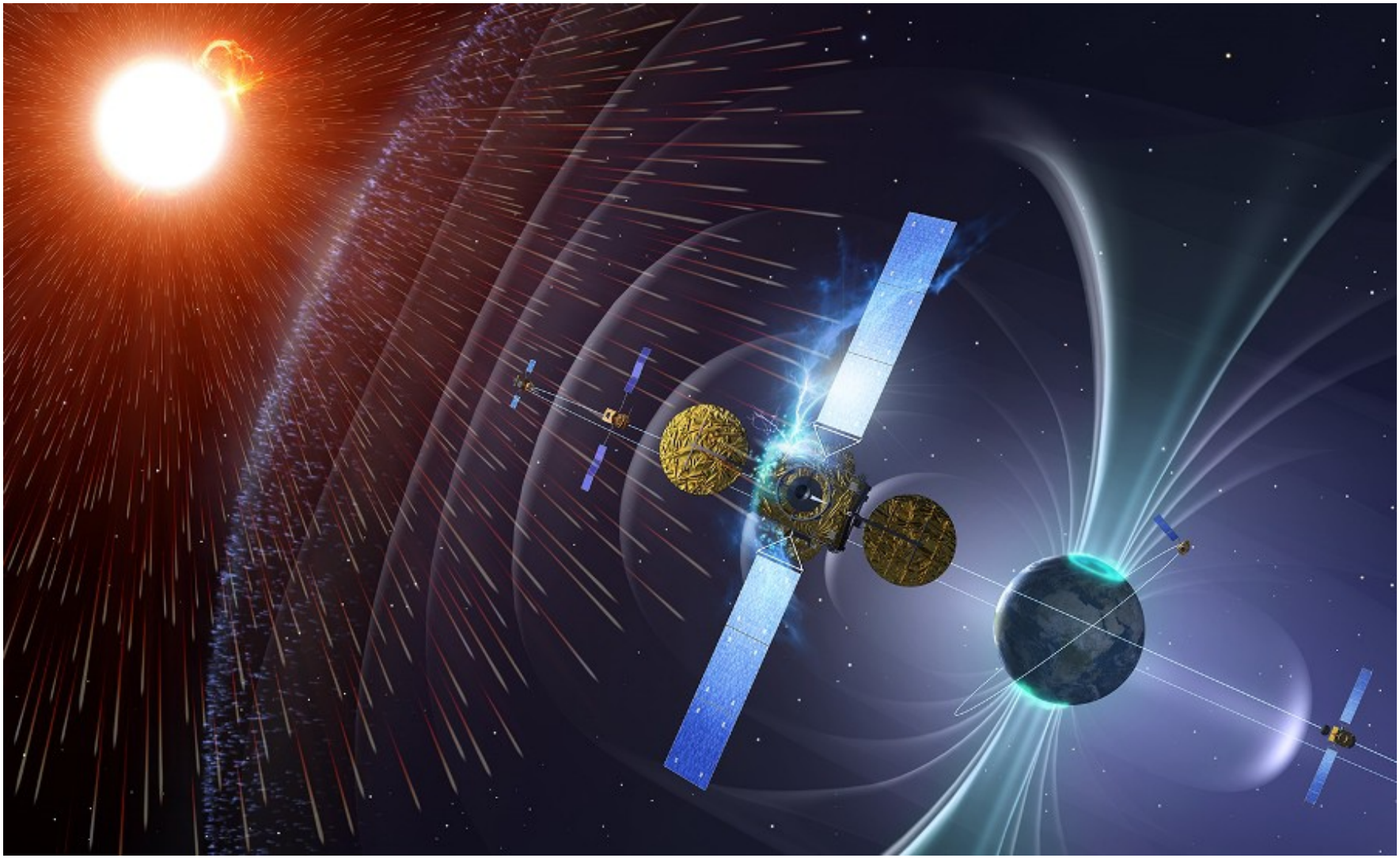
System głosujący



Diverse programming

- Wersje robione przez niezależne zespoły
- Na podstawie tych samych wymagań
- Celem jest zmniejszenie ryzyka tych samych błędów systematycznych (bugów)
- Możliwe są inne sposoby dywersyfikacji: hardware, język programowania, inne paradygmaty

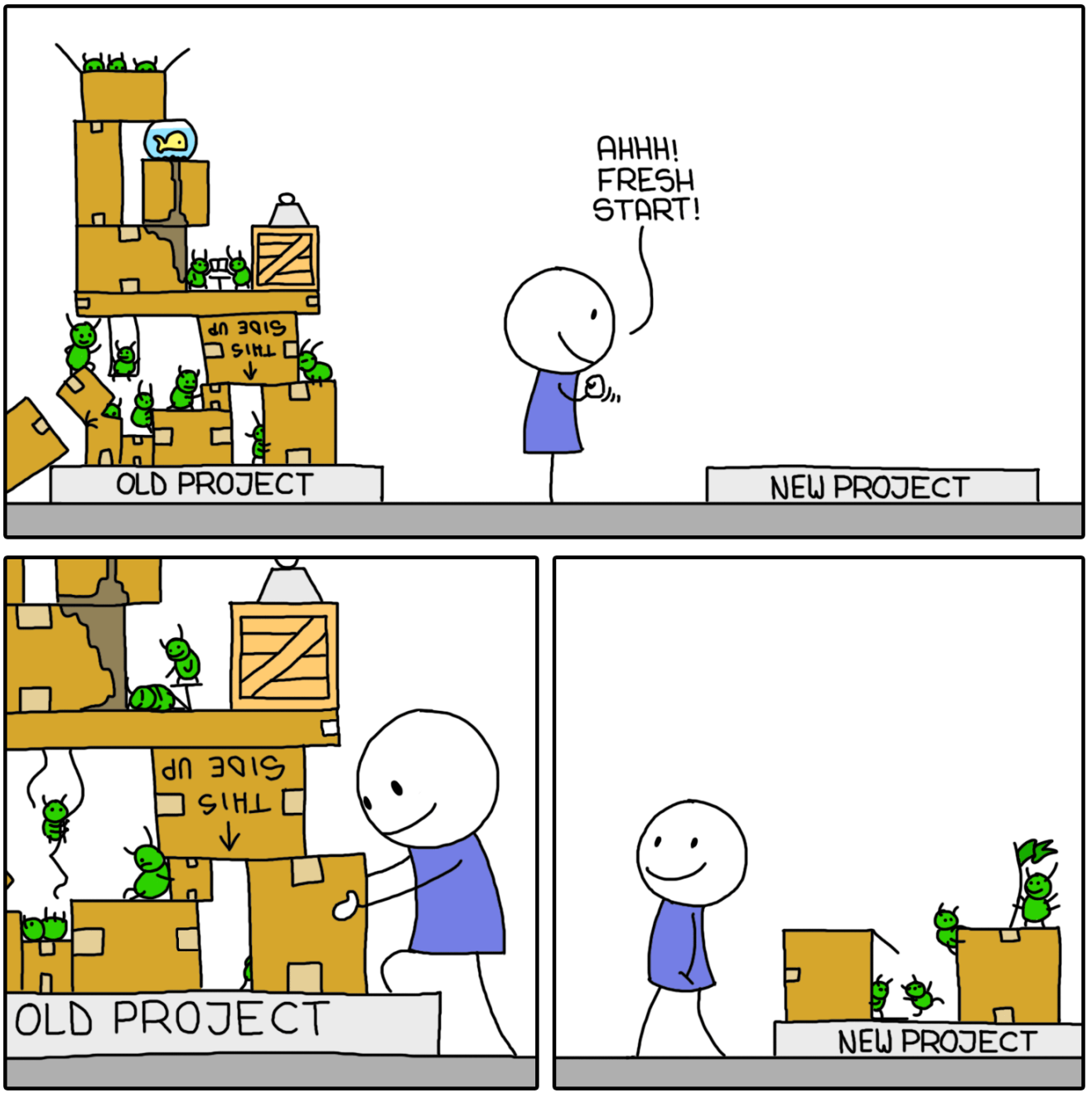
Promieniowanie kosmiczne



Sanity checki

- Testy RAM
- Testy pamięci nieulotnej
- Testy CPU – rejestry i instrukcje
- Testy zasilania
- Testy czujników
- Watchdog

CODE REUSE



MONKEYUSER.COM

SOUP
Software of unknown provenance

SOUP - problemy

- Niewystarczająca dokumentacja
- Brak analizy ryzyka
- Nieznane procedury developmentu i testów
- Brak dostępu do kodu

SOUP – kryteria wyboru

- Stabilność
- Support
- Baza użytkowników

Koszt projektu safety-critical jest 10x większy niż w przypadku zwykłego projektu

Jak tworzyć bezpieczniejsze systemy?

- Planować, dokumentować i weryfikować
- Wcześnie myśleć o możliwych błędach
- Patrzeć na system jako całość
- Stosować sprawdzone praktyki

Dodatkowe materiały:
https://ucgosu.pl/safety-critical/

Twitter:

@MaciekGajdzica

DZIĘKUJĘ!