

BRNO UNIVERSITY OF TECHNOLOGY

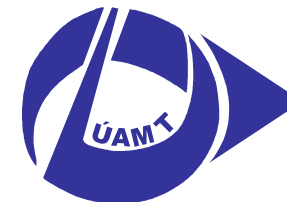
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF CONTROL AND INSTRUMENTATION

Kolejní 4, 616 00 Brno

tel.: +420 5 4114 1113 fax: +420 5 4114 1123

E-mail: kucera@feec.vutbr.cz <http://taceo.eu>

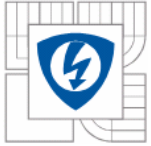


Fault Tolerant systémy

Pavel Kučera

CAK, E112

kucera@feec.vutbr.cz

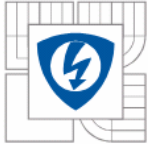


Motto:

„If anything can go wrong, it will.“

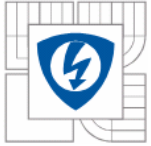
Murphy

If there is a possibility of several things going wrong, the one that will cause the most damage will be the one to go wrong.



Neexistuje nic takového, jako nikdy neselhávající systém. Zvláště pak, když uvažujeme systém vytvořený člověkem. Je to zejména proto, že sám člověk je selhávající systém; a selhávající systém nemůže vytvořit neselhávající systém, což je logický krok k pochopení základní myšlenky fungování vysoce funkčních a vysoce bezpečných systémů:

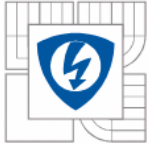
„t.j. na určité úrovni návrhu přijmeme myšlenku, že námi vytvořený systém selže, ale my na to musíme být připraveni.“



Problematikou vysoce bezpečného a vysoce funkčního řízení se zabývá disciplína:

Fault-Tolerant systems (zkratka FT).

Česky bychom hovořili o tzv. systémech, které jsou odolné proti poruchám.



Literatura:

Barry, W., Johnson: **Design and analysis of fault-tolerant digital system.**

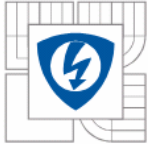
Addison-Wesley, 1989

Hlavička, J., a kolektiv: **Číslicové systémy odolné proti poruchám.**

ČVUT, 1992

Walker, N., W.: **The Design Analysis Handbook.**

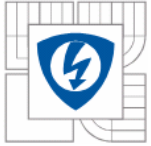
Butterworth-Heinemann, 1998



V mnoha aplikacích může chybná funkce řídicího systému vést k finančním ztrátám, poškození životního prostředí nebo dokonce ke ztrátám na životech.

1979, USA Pennsylvania, Three Mile Island nuclear power plant. Over 140,000 people evacuated within a 15 mile area.

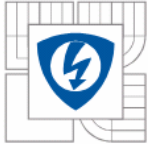
zdroj: www.atomicarchive.com



V mnoha aplikacích může chybná funkce řídicího systému vést k finančním ztrátám, poškození životního prostředí nebo dokonce ke ztrátám na životech.

1982, Therac-25, a compounding of process design, and implementation failures, software defect caused massive radiation killing 3 people.

zdroj: Levenson, Nancy. *Safeware*, Reading, Mass.:Addison-Wesley, 1995



V mnoha aplikacích může chybná funkce řídicího systému vést k finančním ztrátám, poškození životního prostředí nebo dokonce ke ztrátám na životech.

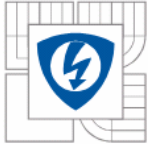
1985, Cement factory, a failure of 8080-based control system caused a large pile of boulders (about 6-8 feet in diameter) to pile up on top of the conveyor (about 80 feet up), eventually falling off and crushing several cars on the parking lot, and damaging a building.

zdroj: Levenson, Nancy. *Safeware*, Reading, Mass.:Addison-Wesley, 1995



V mnoha aplikacích může chybná funkce řídicího systému vést k finančním ztrátám, poškození životního prostředí nebo dokonce ke ztrátám na životech.

1991, Patriot v Perském zálivu, řídicí systém pracoval více než 100 hodin bez restartu na což nebyl testován. Díky zaokrouhlování hodin při ukládání do paměti došlo k odchylce 0.3433 s oproti reálnému času a radar při druhém zaměření hledal střelu Scud na nesprávném místě (o 687 metrů dál). Střela Patriot nebyla vypuštěna a Scud zasáhla kasárna, kde usmrtila 28 lidí a zranila 97.



Fault Tolerant Systems



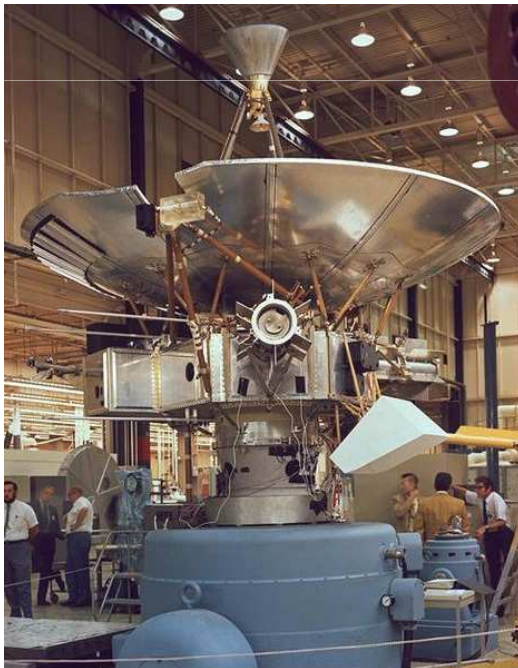
Fatal Accident Causes by Category (percent)						
Cause	1950s	1960s	1970s	1980s	1990s+	All
Pilot Error	42	33	26	29	30	32
Pilot Error (weather related)	9	18	16	17	20	16
Pilot Error (mechanical related)	7	5	4	4	6	5
Total Pilot Error	58	56	46	49	55	53
Other Human Error	2	7	9	7	6	6
Weather	15	10	12	14	9	12
Mechanical Failure	20	20	21	19	20	20
Sabotage	5	4	9	11	8	8
Other Cause	0	3	3	1	1	1



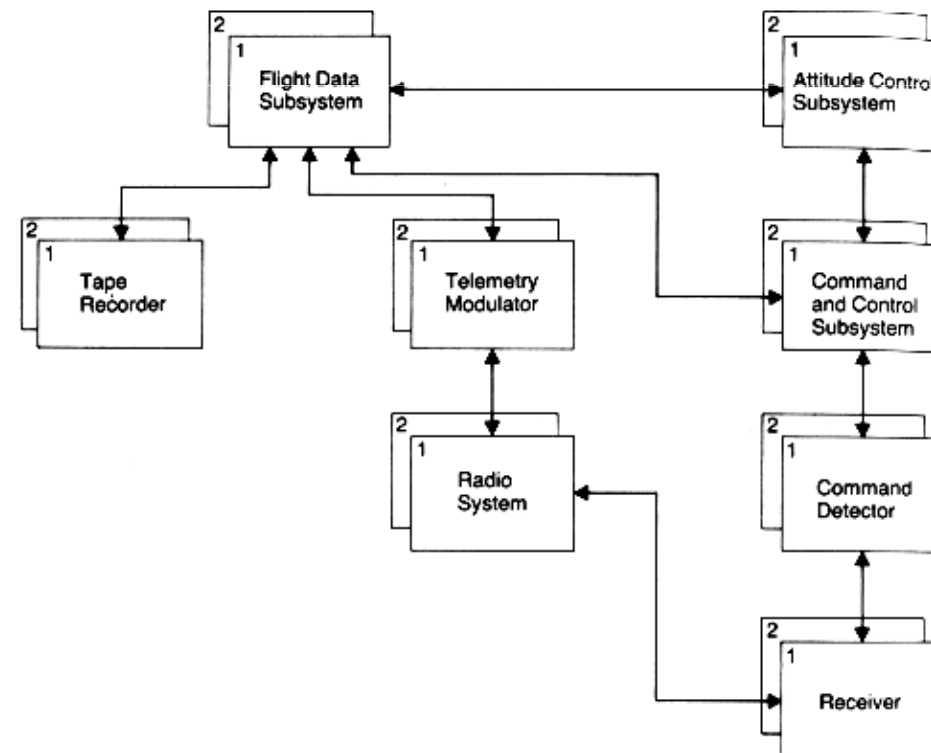
Long-Life Applications

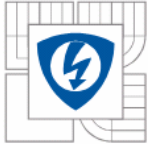
0,95% reliability after 10 years

Pioneer 10 – 1972 to 1983



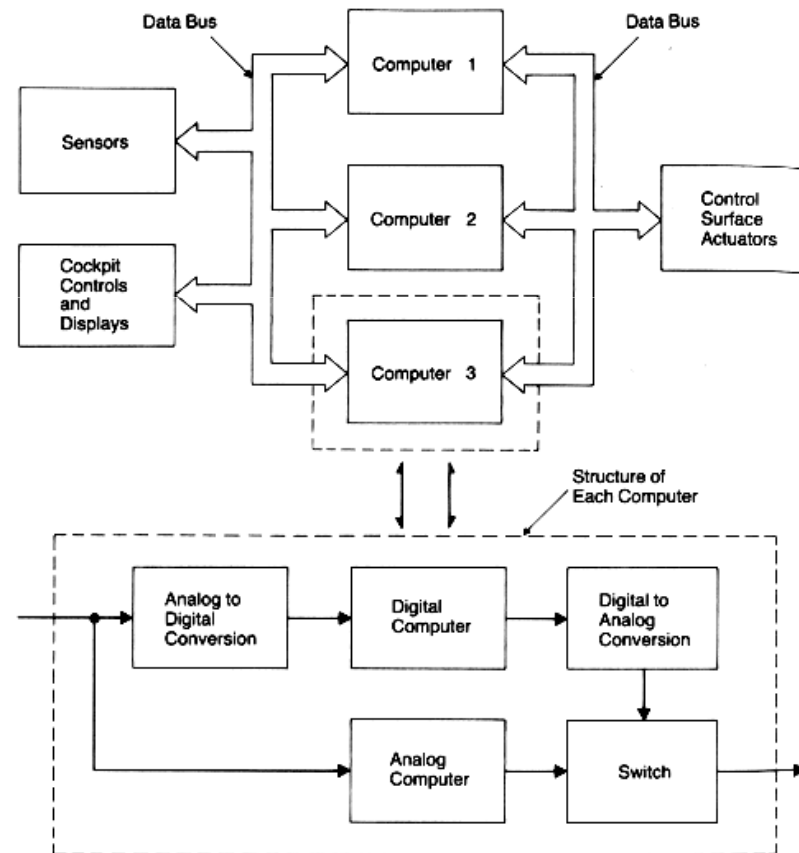
zdroj: solarsystem.nasa.gov

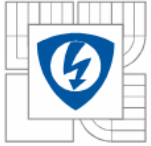




Long-Life Applications

Aircraft X29

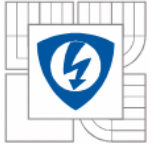




Základní pojmy – **Bezpečnost**

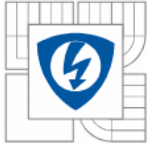
je pravděpodobnost $S(t)$, že systém bude buďto pracovat správně, nebo ukončí svoji činnost takovým způsobem, že nenaruší činnost jiného systému nebo neohrozí lidské životy.

Bezpečnost je mírou bezporuchového provozu systému; pakliže systém nevykonává svoji činnost správně je nutné zajistit, aby alespoň přešel do tzv. bezpečného stavu.



Příklad:

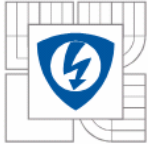
Selže-li řídicí ventil v chemickém procesu, pak existuje poloha (otevřeno nebo zavřeno), ve které by se měl nacházet, aby byla zajištěna výše definovaná bezpečnost. Míru schopnosti řídicího systému tento požadavek splnit nazýváme bezpečností.



Základní pojmy – **Spolehlivost**

$R(t)$ je pravděpodobnost, že se systém v časovém intervalu $[t_0, t]$ neporouchá, přičemž v čase t_0 nebyl v poruše.

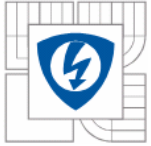
Pojem **bezpečnost** systému se často nesprávně zaměřuje se **spolehlivostí** systému. Základní rozdíl je, že spolehlivost, na rozdíl od bezpečnosti, **neřeší** problematiku selhání systému.



Základní pojmy – **Vykonavatelnost**

$P(L, t)$ systému je funkce času, definovaná jako pravděpodobnost, že výkonnost systému bude vyšší nebo rovna určité hranici L v čase t .

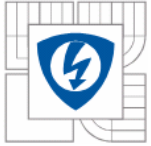
V mnoha případech je možné navrhnout systém tak, aby pokračoval ve vykonávání své činnosti poté co se vyskytla SW nebo HW chyba za cenu, že se sníží výkonnost systému.



Základní pojmy – **Postupná degradace**

je definovaná jako schopnost systému autonomně snížit svůj výkon za účelem kompenzace selhání HW nebo SW části systému.

- Operační systémy
- GSM
- Internet

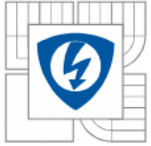


Základní pojmy – **Chyba**

je fyzický defekt nebo závada, která se vyskytne na některém SW nebo HW prvku v systému.

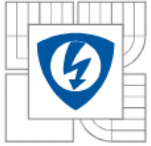
Typickým příkladem selhání HW prvku je elektrický zkrat.

Typickým selháním SW elementu v systému je neplánovaná nekonečná programová smyčka nebo odskok na neexistující či nesprávnou adresu.



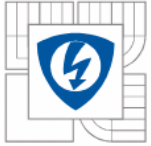
Základní pojmy – **Porucha**

je projevem chyby. Porucha spočívá v ukončení schopnosti systému plnit požadovanou funkci, a to z jakékoliv příčiny a do jakéhokoliv stupně.



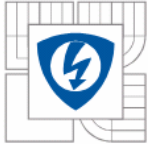
Základní pojmy – **Selhání**

je projevem poruchy. Systém jednak přestává plnit svoji funkci a navíc nekontrolovaně ohrožuje své okolí.



Základní pojmy – **Havárie**

system selže takovým způsobem, že dojde ke ztrátám na životech, k poškození životního prostředí, nebo k finančním ztrátám.



Řetězec *Chyba - Porucha - Selhání*



Př.:

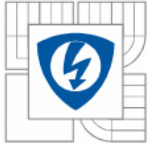
Polohovací ventil se log. 1 zavírá, log. 0 otevírá.

Chyba: Trvalá logická nula na sběrnici (zkrat).

Porucha: Systém se snaží ventil zavřít.

Selhání: Nádrž přeteče.

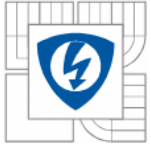
Havárie: Kousky fabriky se povalují v okolí.



Řetězec *Chyba - Porucha - Selhání*

Selhání systému se projeví tehdy, když vyskytnuvší se porucha zabrání dalšímu plnění činnosti systému - t.j. když systém není navržen na správnou reakci na tuto poruchu - systém není FT.

Selhání systému je známkou špatně navrženého a provedeného projektu.

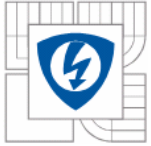


TMI 1979 - sled událostí

TMI - 1979, zatavení jádra

Postup událostí od $t = 04:00$ am.





TMI 1979

t + 0 s

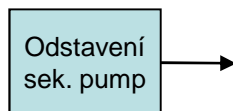
Pumpy v sekundárním okruhu se kvůli malé poruše automaticky vypnuly.
Chyba 1.

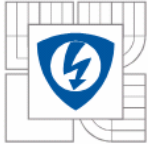
t + 2 s

Protože teplo není dále sekundárním okruhem přenášeno, vzrůstá teplota a tlak vody v primárním okruhu.

t + 3 s

Tlakový ventil primárního okruhu (PORV) se automaticky otvírá, aby vypustil nahromaděnou páru do odpadní nádrže.





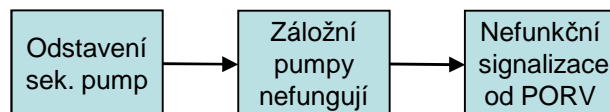
TMI 1979

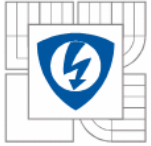
t + 4 s

Naběhnou záložní pumpy sekundárního okruhu, nicméně od systému jsou fyzicky odpojeny uzavíracími ventily - což operátoři netuší. **Chyba 2.**

t + 9 s

Řídicí tyče jsou zasunuty do reaktoru. Zpětná signalizace od PORV hlásí, že se tento uzavřel, což ve skutečnosti není pravda. Unikající pára s vodou odhalují jádro → primární okruh ztrácí schopnost udržet teplotu v reaktoru na bezpečné úrovni. **Porucha.**





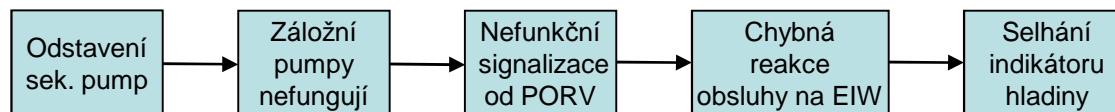
TMI 1979

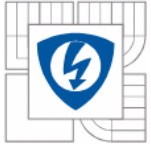
t + 2 min

Je aktivován automatický systém vstřikování vody do primárního okruhu (EIW). Operátory to nepřekvapí, neboť v minulosti se to již několikrát stalo, aniž by k nějakému úniku vody z primárního okruhu došlo.

t + 5 min

Operátoři zjišťují, že hladina vody v primárním okruhu stoupá, proto systém EIW vypnou. Ve skutečnosti hladina vody klesá - PORV ventil je stále otevřen. **Selhání** - systém způsobuje škody, neboť voda unikající přes PORV do atmosféry je radioaktivní.

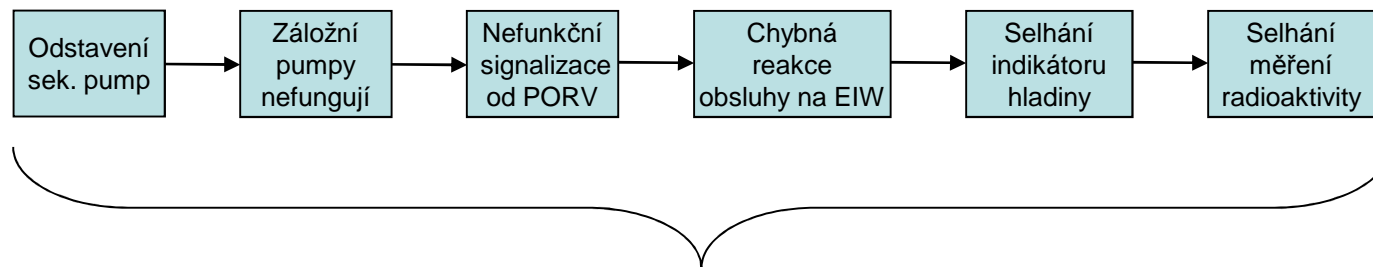




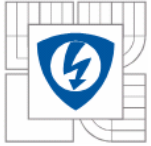
TMI 1979

t + 15 min

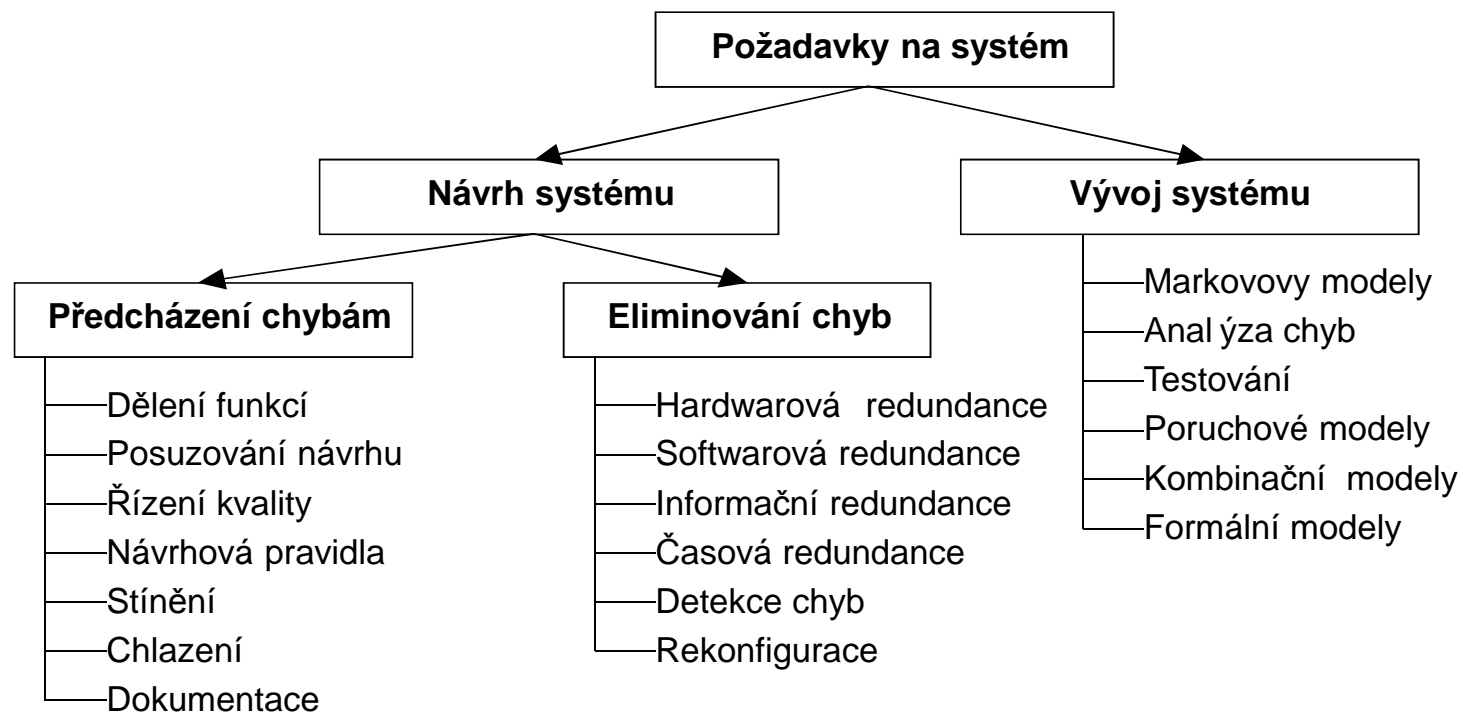
Do tohoto okamžiku unikne z primárního okruhu přes 12 tisíc litrů vody, avšak měřiče radioaktivity nespustí alarm. **Havárie.**

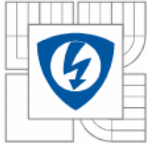


pravděpodobnost řetězce událostí $\ll 0.000001$



Metodika FT návrhu

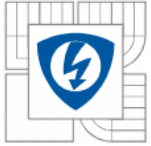




Boj proti chybám

Existují tři základní metodiky, které pomáhají eliminovat vliv chyb na systémy:

1. Předcházet chybám - **Fault avoidance.**
2. Maskovat chyby - **Fault masking.**
3. Eliminovat vliv chyby - **Fault tolerance.**



Fault avoidance

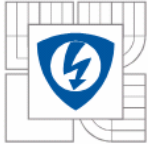
- Řízení kvality
- Dokumentace
- Stínění, chlazení, předimenzování
- Testování

Fault masking

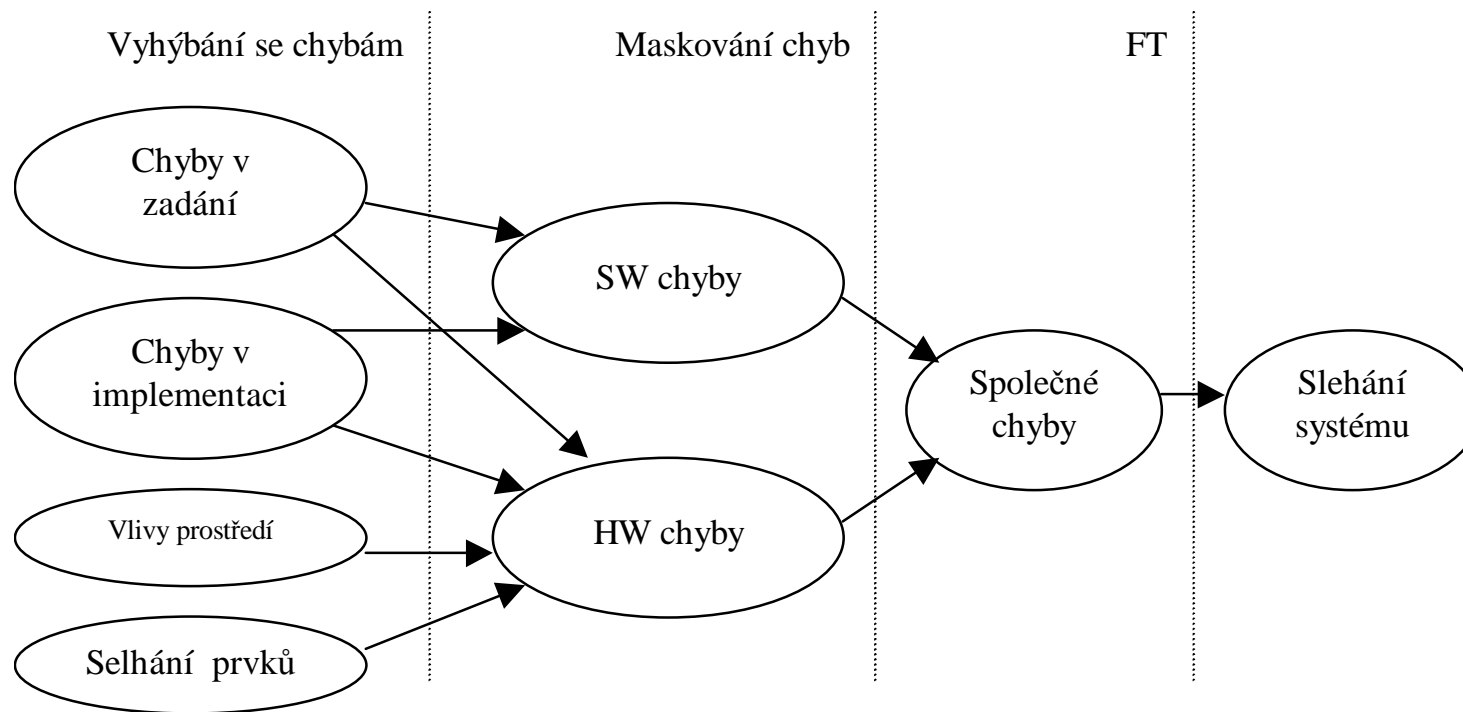
- Samoopravné kódy
- NMR systémy

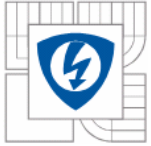
Fault tolerance

- Rekonfigurace



Vztahy mezi FT metodikami

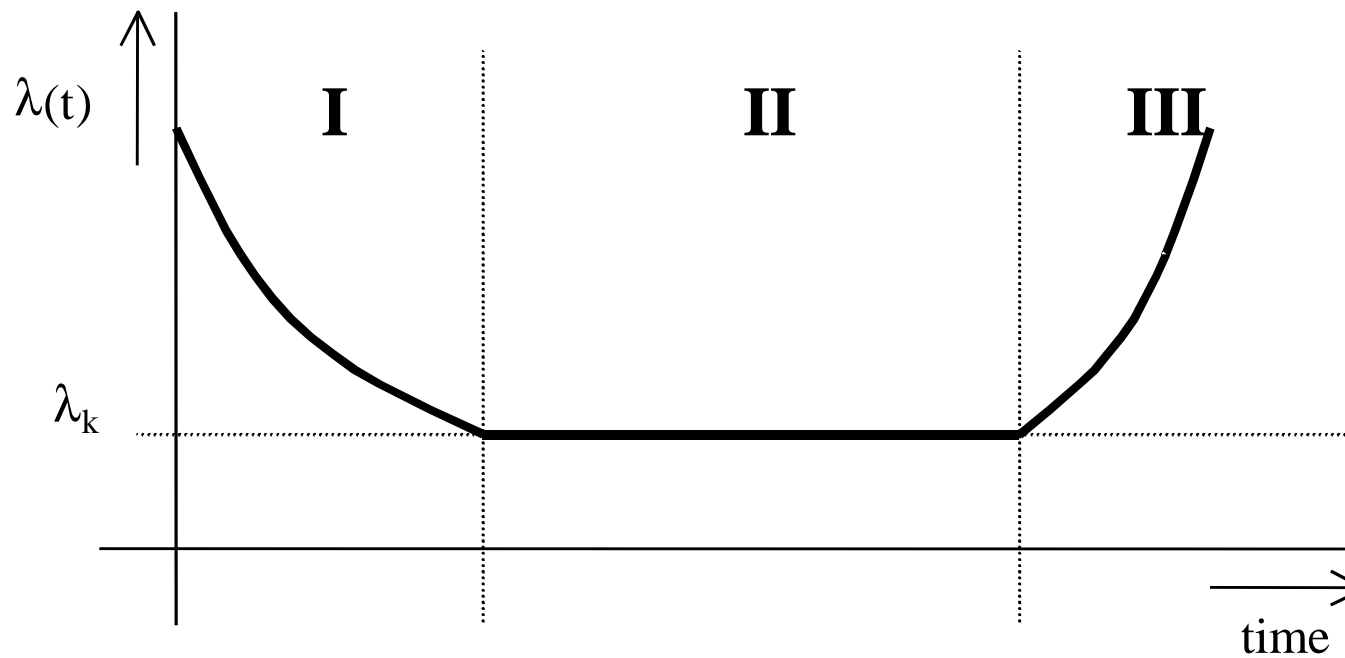


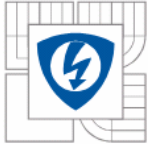


Ukazatele spolehlivosti - **Intenzita poruchy**

$$\lambda(t)$$

očekávaný počet selhání za jednotku času





Ukazatele spolehlivosti

Střední doba mezi poruchami

V anglické literatuře se označuje jako **MTBF** (Mean Time Between Failures) a je to zdaleka nejčastější spolehlivostní ukazatel, který výrobci poskytují.

Dobrý výrobce se neobává MTBF svého produktu zveřejnit.

$$MTBF = \frac{\sum_{i=1}^n t_{pi}}{n} \quad \lambda_k = \frac{1}{MTBF}$$

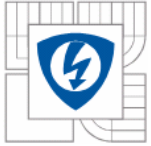


Ukazatele spolehlivosti

Střední doba mezi poruchami

Příklady:

<i>Prvek/systém</i>	<i>MTBF [h]</i>
Intel 8086	30×10^6 (3000 let)
AMD 80386	10×10^5 (100 let)
PLC	30×10^4 (30 let)
Motherboard MX3L	76×10^3 (9 let)
Motherboard MX3Z	40×10^3 (5 let)
Unix	1×10^3 (měsíc)
Windows	?



Ukazatele spolehlivosti

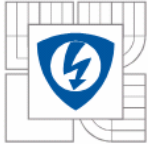
Pravděpodobnost bezporuchového provozu

$R(t)$

je pravděpodobnost, že v čase $\tau \leq t$ nedojde k poruše systému.

Pro $R(t)$ platí rovnice:

$$R(t) = e^{-\lambda_k t}$$

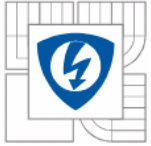


Sériový spolehlivostní model

Porucha kteréhokoliv prvku v systému způsobí poruchu v celém systému.

$$R_s(t) = \prod_{i=1}^n e^{-\lambda_i t}$$

$$\lambda_s = \sum_{i=1}^n \lambda_i$$



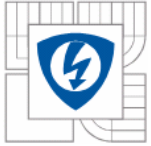
Sériový spolehlivostní model - příklad

Řízení letu proudového letadla se skládá ze tří číslicových regulačních obvodů, z nichž každý je tvořen:

snímačem polohy s intenzitou poruchy λ_s ,
snímačem povelů pilota s intenzitou poruchy λ_p ,
aktorem s intenzitou poruchy λ_a ,
mikropočítačem s intenzitou poruchy λ_m ,

Prvky mezi sebou komunikují po řídicí sběrnici s intenzitou poruchy λ_{bc} a datové sběrnici s intenzitou poruchy λ_{bd} .

Stanovte pravděpodobnost bezporuchového provozu systému po dobu 1 hodiny letu, způsobí-li jeho selhání výpadek libovolného prvku řízení.



Sériový spolehlivostní model - příklad

Intenzita poruchy celého systému je dána součtem intenzit poruch jednotlivých komponent, tedy:

$$\lambda_{\text{system}} = 3\lambda_s + 3\lambda_p + 3\lambda_a + 3\lambda_m + \lambda_{bc} + \lambda_{bd} \quad (\text{s}^{-1})$$

a pravděpodobnost, že systém řízení letu bude pracovat správně:

$$R(18000) = e^{-\lambda_{\text{system}} 18000}$$

Sériový spolehlivostní model – příklad

Dosadíme-li za definované intenzity konkrétní hodnoty letounu X-29:

$$\lambda_s = 10^{-6} \text{ h}^{-1}$$

$$\lambda_p = 10^{-6} \text{ h}^{-1}$$

$$\lambda_a = 10^{-5} \text{ h}^{-1}$$

$$\lambda_m = 4 \times 10^{-4} \text{ h}^{-1}$$

$$\lambda_{bc} = 10^{-6} \text{ h}^{-1}$$

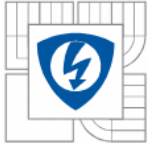
$$\lambda_{bd} = 2 \times 10^{-6} \text{ h}^{-1}$$

pak je výsledná intenzita poruch

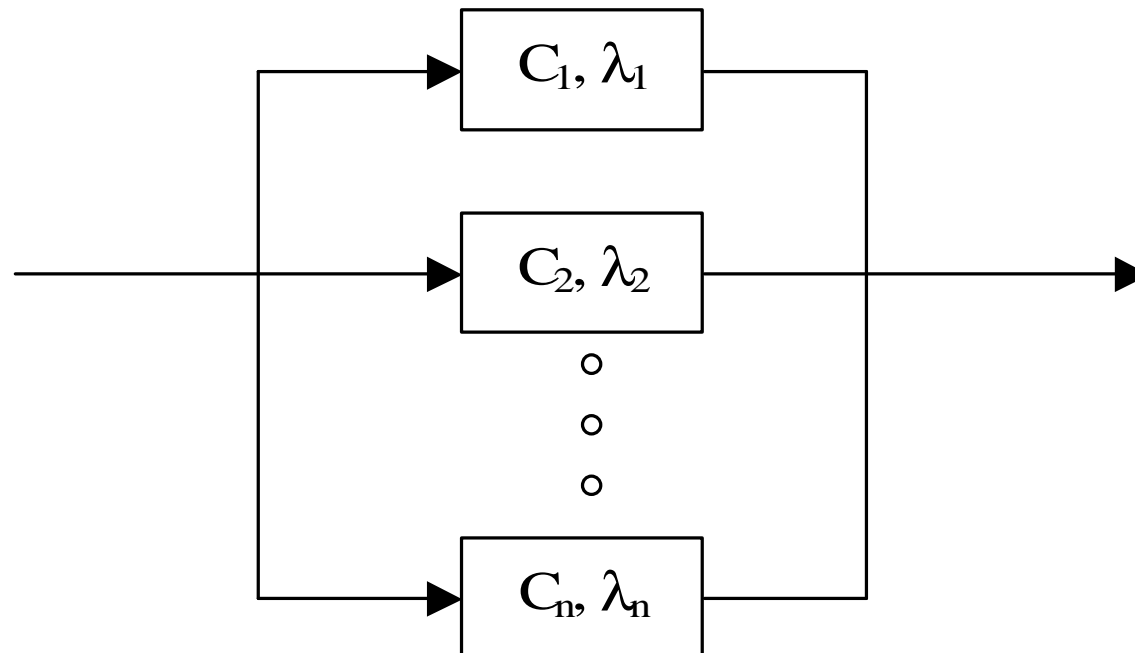
$$\lambda_{\text{system}} = 1,239 \times 10^{-3} \text{ h}^{-1}$$

a pravděpodobnost bezporuchového provozu po dobu 5-ti hodin je:

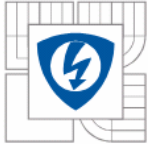
$$R(5 \text{ h}) = 0,995$$



Paralelní spolehlivostní model



$$R_p(t) = 1 - \prod_{i=1}^n (1 - e^{-\lambda_i t})$$

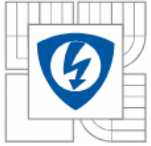


M z N systémy

Jsou generalizací ideálních paralelních systémů. V M z N systému je nutné ke správné činnosti systému jeho **M** prvků z celkových **N** prvků.

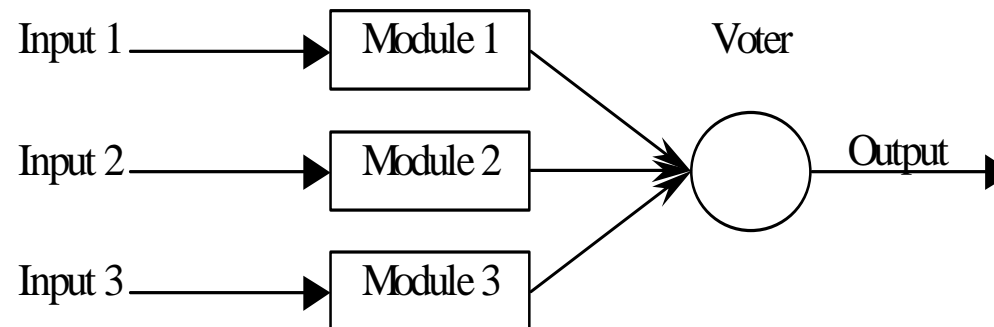
Pravděpodobnost bezporuchového provozu systému M z N lze spočítat jako:

$$R_{MzN}(t) = \sum_{i=0}^{N-M} \binom{N}{i} R^{N-i(t)} (1-R(t))^i$$



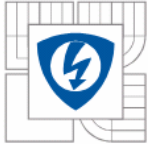
TMR systémy

TMR je zkratka z angl. Triple Modular Redundant. Tímto termínem se rozumí uspořádání tří prvků tak, aby výpadek jednoho vedl k maskování poruchy v systému.



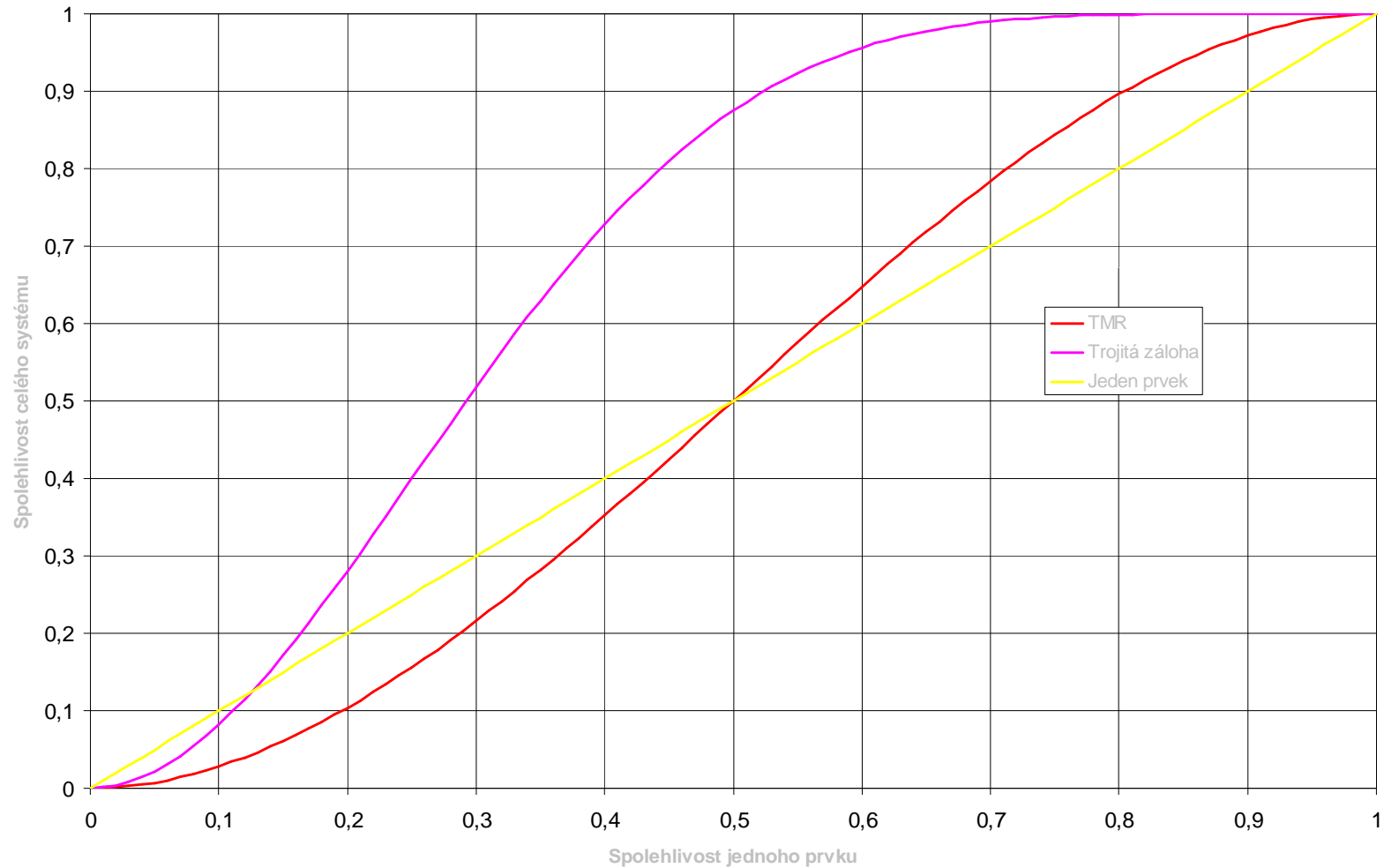
$$R_{TMR} = 3R^2 - 2R^3$$

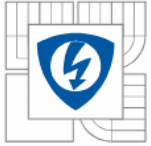
$$R_{TMR}(t) = 3e^{-2\lambda t} - 2e^{-3\lambda t}$$



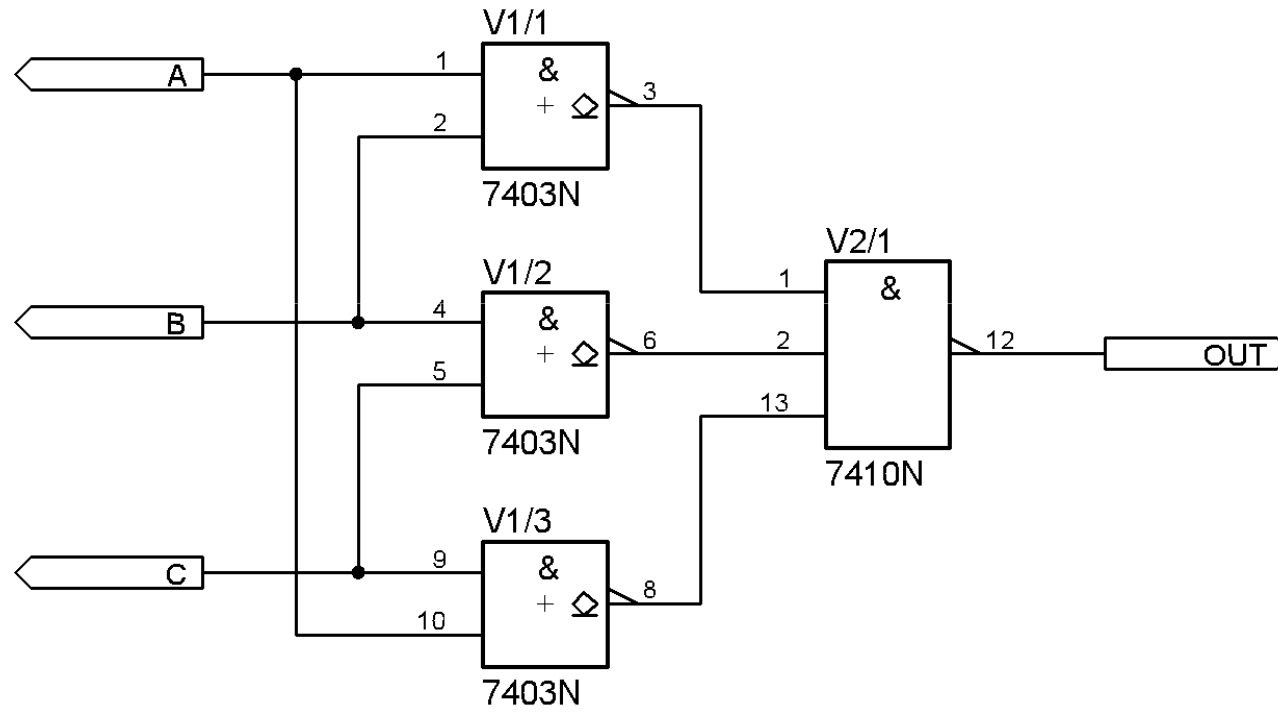
Srovnání

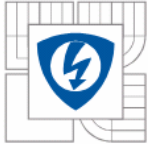
Srovnání spolehlivostí TMR, paralelní zálohy trojice prvků a jednoho prvku



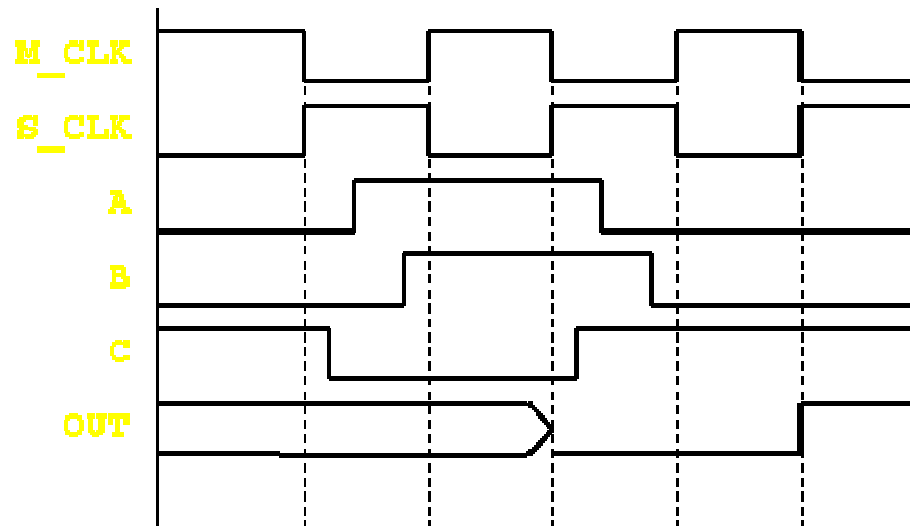
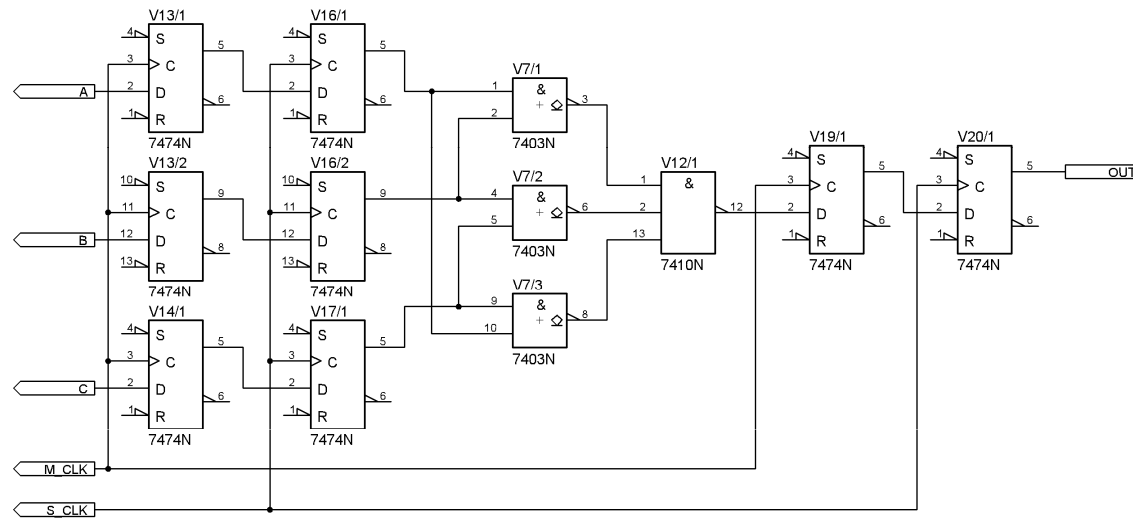


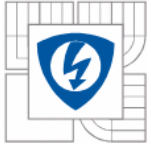
TMR Aktor - I



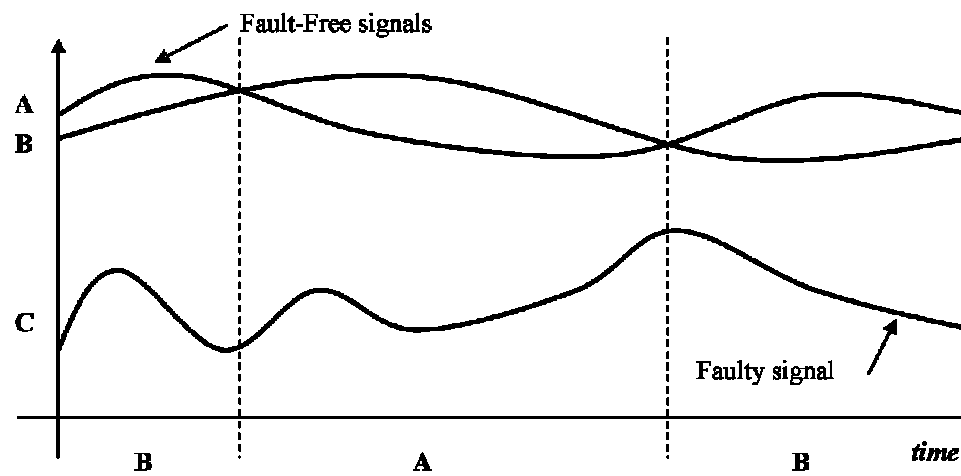
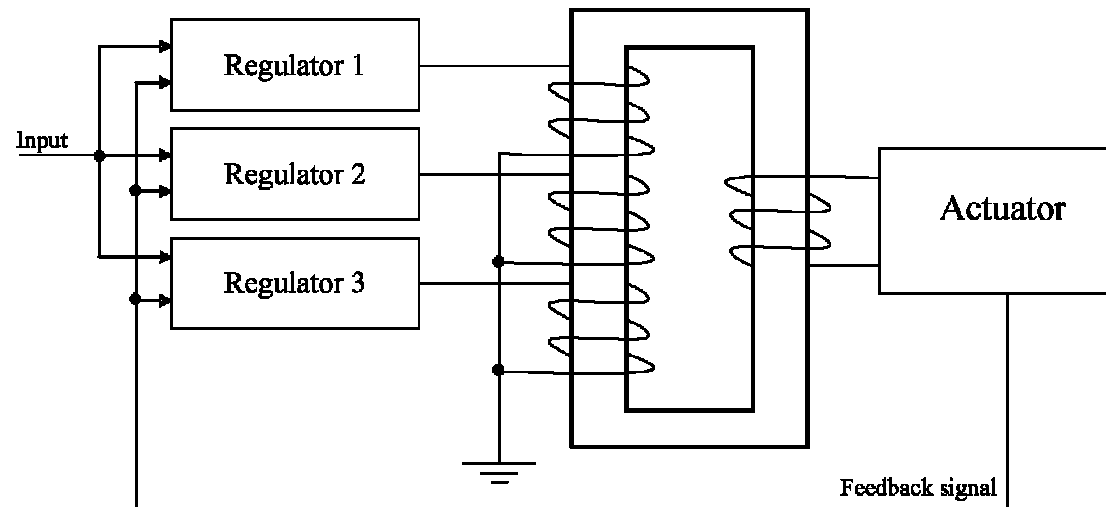


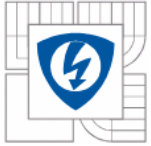
TMR Aktor - II



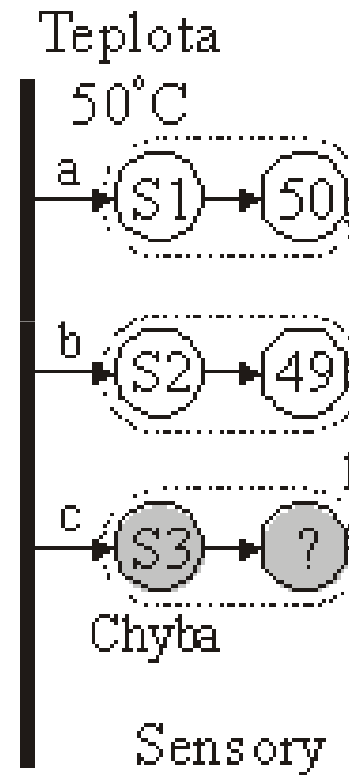


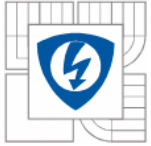
TMR Aktor - III



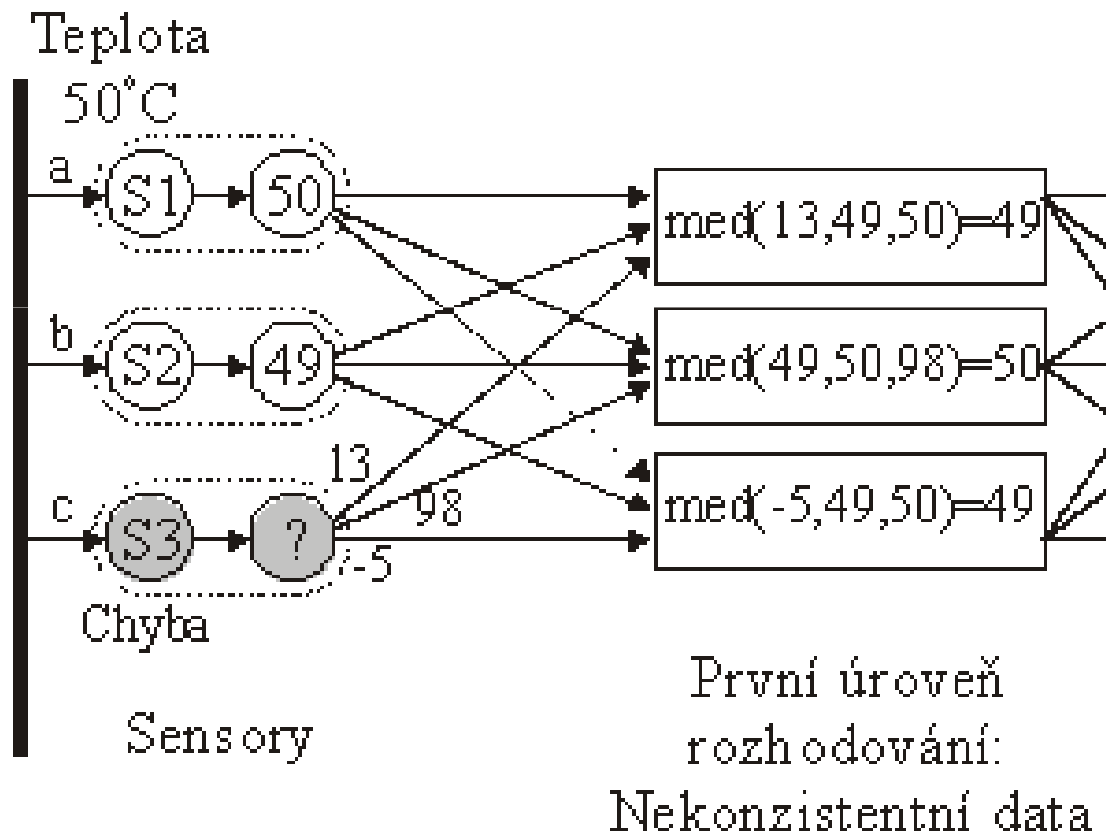


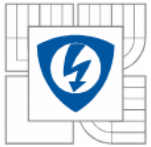
TMR Sensor - I



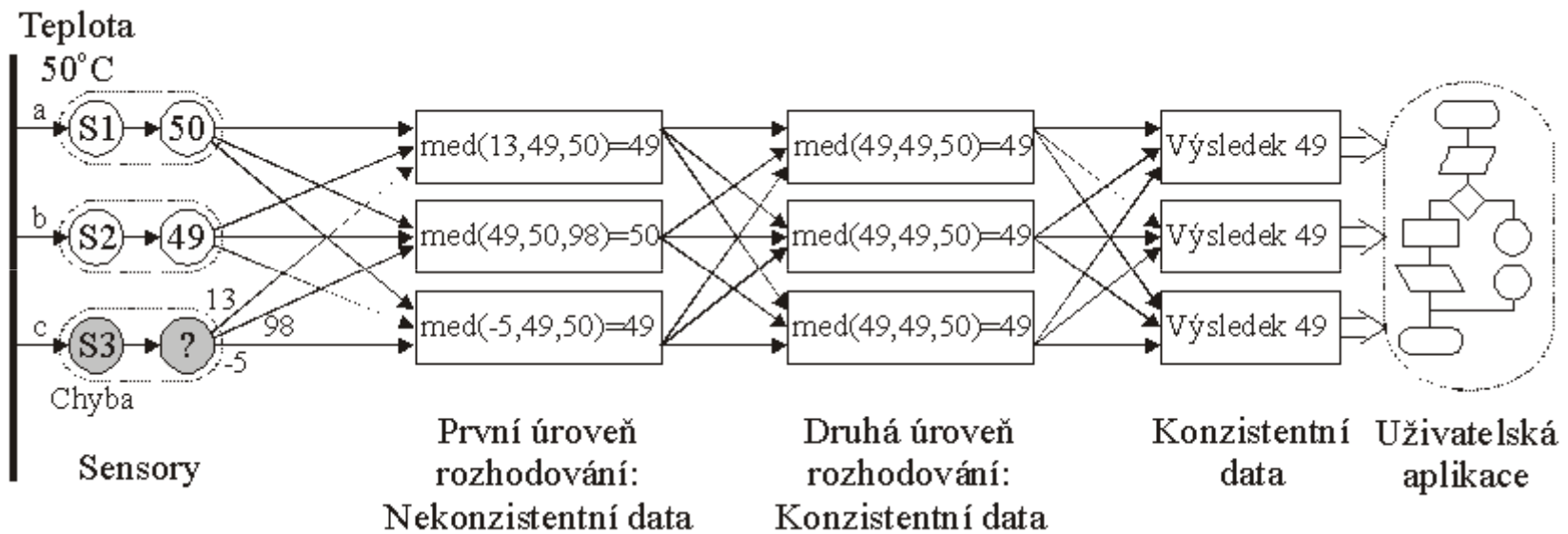


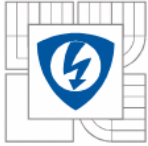
TMR Sensor - II



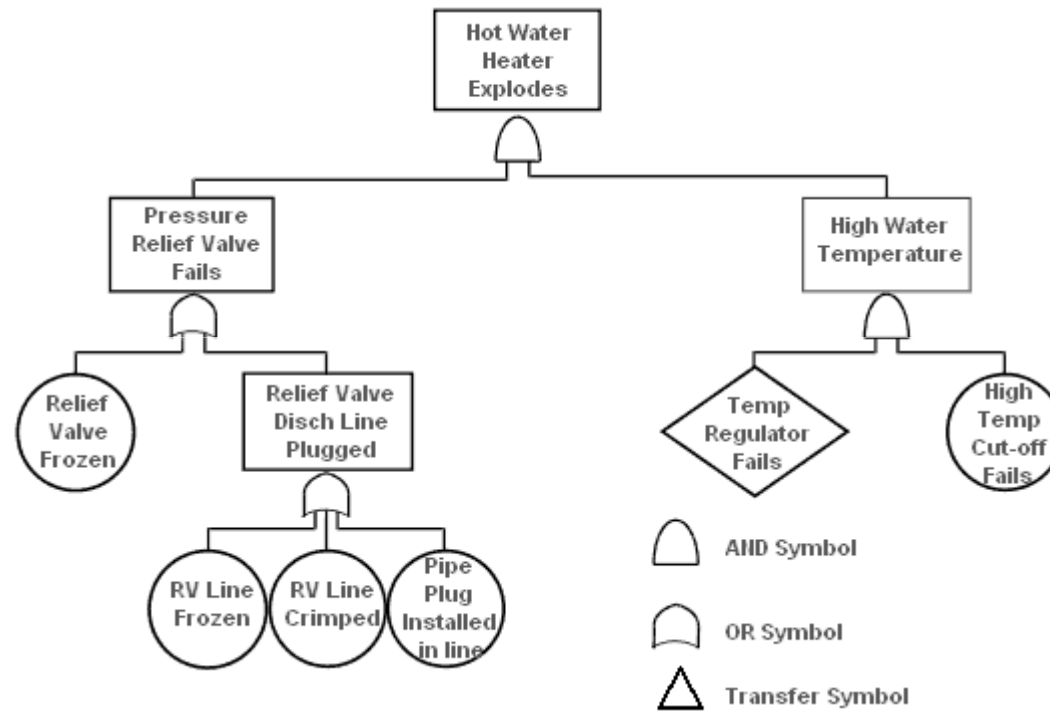


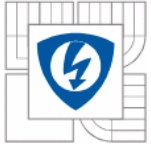
TMR Senzor - III



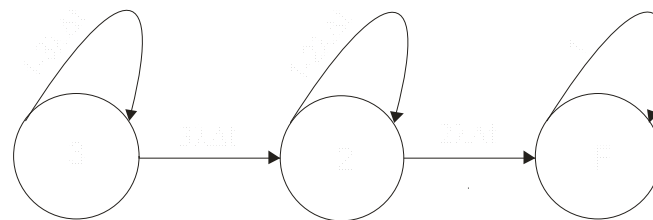
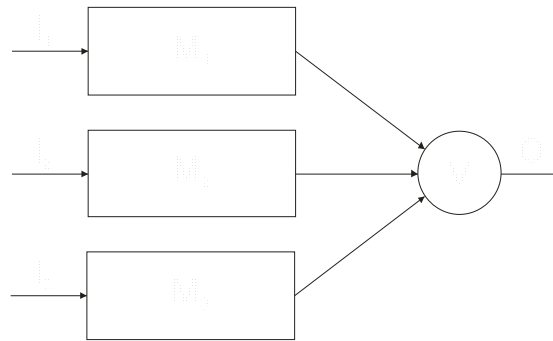


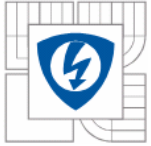
Fault-Tree Analysis (FTA)



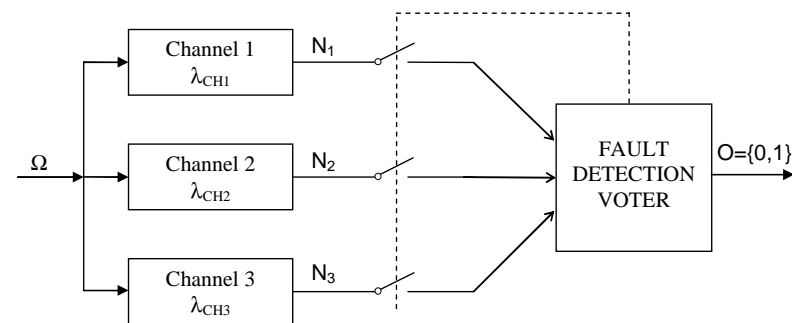
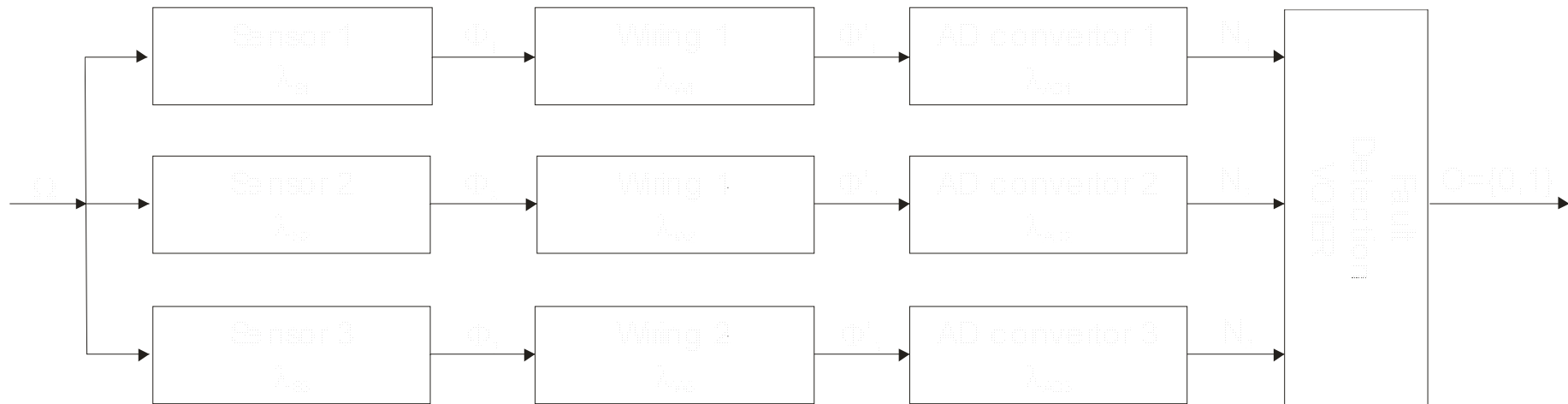


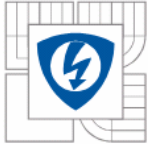
Markovovy procesy



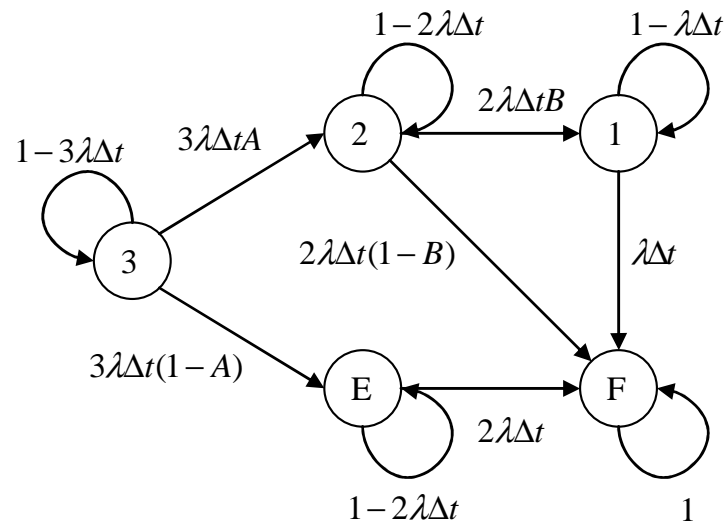


Analogový TMR systém





Analogový TMR systém



$$P_3(p) = \frac{1}{p+3\lambda}$$

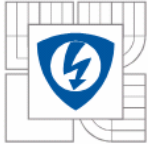
$$P_2(p) = \frac{3\lambda A}{p^2 + 5p\lambda + 6\lambda^2}$$

$$P_1(p) = \frac{6\lambda^2 AB}{p^3 + 6p^2\lambda + 11p\lambda^2 + 6\lambda^3}$$

$$P_E(p) = \frac{3\lambda(A-1)}{p^2 + 5p\lambda + 6\lambda^2}$$

$$P_F(p) = -\frac{6\lambda^2(ABp - p - \lambda)}{p(p^3 + 6p^2\lambda + 11p\lambda^2 + 6\lambda^3)}$$

$$R_{FDV} = 1 - p_F = (3AB - 2)R^3 + (3 - 6AB)R^2 + 3ABR$$



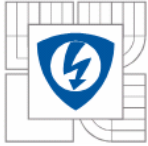
Chyby v SW – nesprávná konstrukce

Elektromagnetické rušení ovlivní činnost samotného mikroprocesoru a uvede mikroprocesor do stavu, v jehož důsledku dojde k nekorektnímu vykonání instrukce. Příčinou může být např. elektrostatický výboj (ESD).

Elektromagnetické rušení způsobí neplatnost údajů na sběrnici v okamžiku, kdy je ze sběrnice čteno, nebo je na sběrnici zapisováno.

Krátkodobý pokles či výpadek napájecího napětí, případně rušivý impuls indukovaný na napájení může procesor uvést do nestandardního stavu, v jehož důsledku dojde k nesprávné činnosti programu.

Chyba v návrhu nebo při výrobě hardwaru, která se projevuje pouze za určitých specifických okolností. Příkladem mohou být poruchy, které se projevují pouze za určitých teplot, v určité nadmořské výšce, nebo pouze při současném spuštění jiných zařízení.



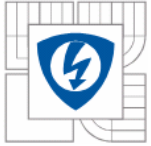
Chyby v SW – nesprávný program

Chyby a nedokumentované vlastnosti mikroprocesoru. Chyby v mikroprocesorech jsou zcela běžné u nových modelů procesorů a výrobci mikroprocesorů běžně zveřejňují Errata k jednotlivým výrobním šaržím.

Chyby ve vývojových prostředcích (kompilátorech, linkerech, atd.). Takovéto chyby jsou běžné zejména u vývojových prostředků pro nové modely procesorů a způsobují, že správně napsaný zdrojový kód provádí nesprávnou činnost.

Chyby ve zdrojovém kódu udělané programátorem.

Činnost periferních obvodů a připojených zařízení nebo lidské obsluhy, která nebyla při programování aplikace očekávána.



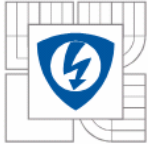
Chyby v SW – zvýšení spolehlivosti

Zabezpečení obsahu nonvolatilních pamětí (PROM, EPROM, ...) kontrolním součtem, přičemž kontrola neporušenosti obsahu se provádí nejen při startu aplikace, ale v určitých časových intervalech i za běhu aplikace.

Ověření, že zápis dat do paměti byl úspěšný, porovnáním obsahu paměti s tím, co bylo zapisováno. Je-li zjištěno, že obsah paměti není shodný se zapsanými daty, je zápis dat do paměti zopakován. Je-li i opakovaný zápis neúspěšný, došlo zřejmě k poruše trvalejšího charakteru (např. porucha paměti nebo přerušení vodiče sběrnice).

Zabezpečení dat ukládaných do paměti za běhu aplikace kontrolním součtem vždy, když je to jen trochu možné, a následná kontrola před použitím těchto dat.

Sledování správného vykonávání jednotlivých částí programu pomocí trasovacích proměnných. Na začátku souvislého bloků kódu je nastaven příslušný bit v trasovací proměnné. Při opouštění tohoto bloku se zkontroluje, zda je v trasovací proměnné nastaven příslušný bit a je nastaven jiný bit, který indikuje, že daný blok byl vykonán. Při opouštění bloku je tak prováděna kontrola, zda do daného bloku procesor vstoupil na správném místě (na začátku bloku).

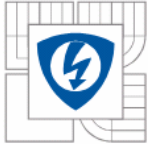


Chyby v SW – zvýšení spolehlivosti

Sledování běhu programu pomocí trasovacích proměnných. Použití trasovacích proměnných umožňuje sledovat odkud do daného bloku programu procesor vstupuje. Mnohdy platí, že do určitých částí programu se procesor může dostat pouze z několika málo jiných částí programu. Nastavením trasovacího bitu při výstupu z daného bloku programu tak poskytujeme následujícímu úseku kódu informaci o tom, která část programu byla vykonávána před tím, než byl daný blok spuštěn.

Sledování doby potřebné na vykonání úseku programu. V mnoha případech lze určit, za jak dlouho se procesor dostane z bodu A programu, do bodu B. Při průchodu bodem A je uložena informace o aktuálním čase a při průchodu bodem B je zkontrolováno za jak dlouho se procesor z bodu A do bodu B dostal. Došlo-li k chybě, která způsobila že procesor mezi body A a B někde „bloudil“, tak je zjištěno, že přechod z bodu A do bodu B trval nepřipustně dlouho. Tento způsob může zajistit mnohem rychlejší detekci selhání softwaru, než jakou poskytuje watchdog.

Neustálá kontrola smysluplnosti výsledků výpočtů. Může-li se výsledek pohybovat v předem známém intervalu hodnot, je vhodné po dokončení výpočtu zkontrolovat, zda se výsledek v tomto intervalu skutečně nachází. (Je-li např. výsledkem výpočtu teplota vody v nádrži, je zřejmé, že z fyzikálních důvodů vypočtená hodnota musí ležet v nějakém omezeném intervalu.)



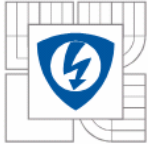
Chyby v SW – zvýšení spolehlivosti

Úplné vyhodnocování podmínek. Například může-li proměnná A nabývat pouze hodnot 1 nebo 2, tak je mnohdy použita konstrukce:

```
if (A = 1) then Pohon_stop;           /* Je-li A = 1, tak pohon vypneme */  
else Pohon_start;                    /* Je-li A=2, tak pohon zapneme */
```

Při použití této konstrukce předpokládáme, že pokud je hodnota proměnné A různá od jedné, tak musí být rovna dvěma. Avšak dojde-li k chybě a proměnná A bude mít hodnotu A=3, tak dojde ke spuštění pohonu, což nemusí být žádoucí. Proto je vhodné výše uvedenou konstrukci upravit např. následovně:

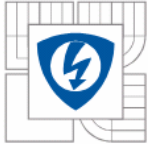
```
if (A = 1) then Pohon_stop           /* Je-li A = 1, tak pohon vypneme */  
else if (A = 2) then Pohon_start     /* Je-li A=2, tak pohon zapneme */  
    else Nastala_chyba;              /* Dostane-li se procesor sem, tak nastala chyba */
```



Chyby v SW – zvýšení spolehlivosti

Používání watchdogu. Mnoho procesorů používaných pro vestavěné (embedded) systémy obsahuje speciální obvod zvaný watchdog, který slouží ke resetování procesoru v případě, že se aplikace zhroutí. Při použití watchdogu musí aplikace čas od času zapsat určitou hodnotu na adresu watchdogu - tzv. občerstvení obvodu watchdog. Pokud je na tuto adresu zapsána nesprávná hodnota, nebo není na tuto adresu po určitou dobu zapsáno vůbec nic, tak watchdog vyvolá reset procesoru. V případě, že použitý procesor neobsahuje interní watchdog, je možné zajistit jeho funkci externím obvodem.

Zápis na adresu watchdogu by měl být prováděn pouze na jednom místě programu. V případě použití operačního systému reálného času, by pro zápis na adresu watchdogu měl být vyhrazen samostatný proces s nejnižší prioritou.



Chyby v SW – zvýšení spolehlivosti

Kontrola kódu po kompilaci, zda neobsahuje kombinaci, která by mohla provést nežádoucí občerstvení obvodu watchdog. Například má li instrukce zajišťující občerstvení kód 0DD8h, tak vyskytne li se v programu následující sekvence instrukcí:

CPI	A, #0D	= 370Dh
JRNZ	JMP1	= D8xxh

tak je vhodné upravit výše uvedený kód vložením instrukce NOP, což zajistí, že se ve výsledném kódu eliminuje nežádoucí výskyt po sobě jdoucích hodnot 0Dh a D8h:

CPI	A, #0D	= 370Dh
NOP		= 04h
JRNZ	JMP1	= D8xxh

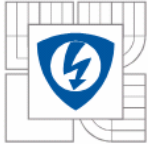
U mnoha procesorů je možné provést zápis na adresu watchdogu několika různými způsoby. Je proto nutné identifikovat všechny možné varianty, které mohou způsobit občerstvení watchdogu a eliminovat jejich výskyt ve výsledném kódu.



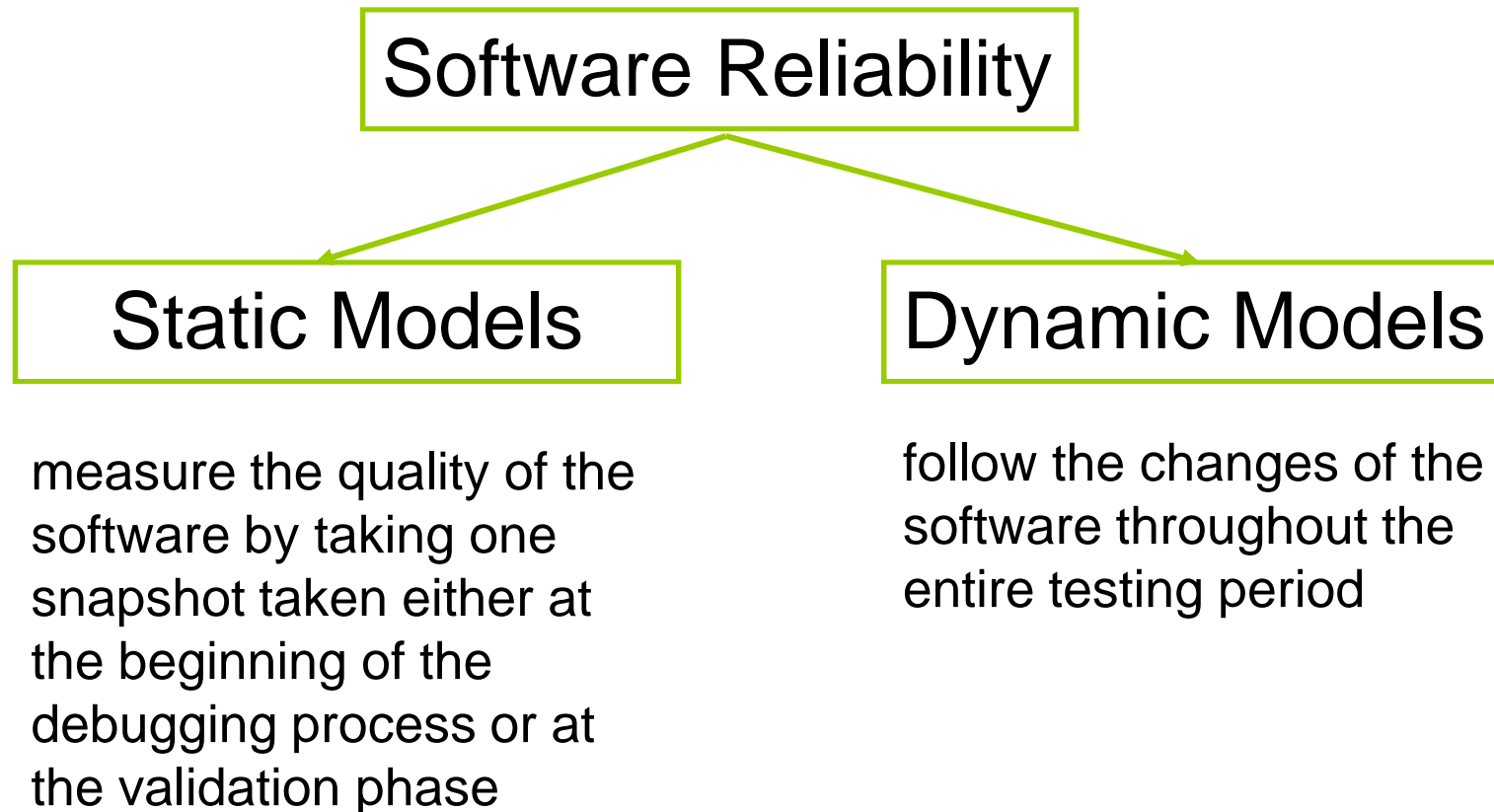
Chyby v SW – zvýšení spolehlivosti

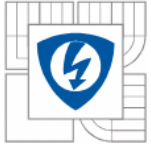
Zaplnění nepoužívané paměti programu instrukcí NOP (No operation) nebo ještě lépe instrukcí vyvolávající reset procesoru.

Periodický restart aplikace. Je rozumné předpokládat, že běžící aplikace díky své neodhalené chybě ve zdrojovém kódu postupně degraduje. Příkladem takové postupné degradace může být např. nesprávná práce s pamětí, která vede k jejímu postupnému zaplnění. Proto se doporučuje, je-li to možné, dlouhodobě běžící aplikaci po určité době restartovat. Restart a náběh je u některých aplikací natolik rychlý, že jej uživatel ani nepostřehne, a proto je možné pevně stanovit, s jakou periodou se má aplikace restartovat. U jiných zařízení, u kterých není žádoucí restart s pevně nastavenou periodou, bývá někdy možné nalézt pracovní režim, ve kterém aplikací vynucený restart nepoškodí uživatele. Příkladem vhodného okamžiku k restartu může být okamžik, kdy zařízení přechází z režimu nízké spotřeby po delší době, kdy zařízení nebylo používáno.



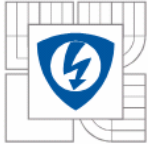
SW Reliability





SW Reliability

- Software reliability is defined to be the probability of failure free operation of a computer program in a specific environment for a specified period of time.
- Professional programmers average six software defects for every 1000 lines of code (LOC) written.

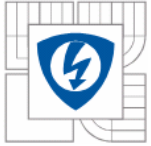


SW Reliability

Yu supposes a following form of the defect model of the software product:

$$D = f(M, ED, T, other)$$

- D* is the number of defects to be found a certain time period,
- M* is static program metrics - such as lines of code, volume, etc.,
- ED* is the defects found in the earlier stages of testing,
- T* is the testing time in CPU time, calendar time, etc.,
- other* stands for additional parameters such as programmer's skills, programming language, complex of the task, etc.



SW Reliability

Defect density

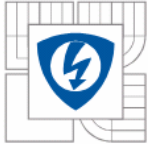
Compton model assumes a quadratic relationship between errors and program length, he defines defect density E_d :

$$E_d = \min(E_{d1}, E_{d2})$$

$$E_{d1} = 1.56 + \frac{69}{\hat{N}} + 4.7 \cdot 10^{-4} \hat{N}$$

$$E_{d2} = 2.2351 + \frac{998}{\hat{N}} + 9.778 \cdot 10^{-5} \hat{N}$$

\hat{N} is estimated program length.



SW Reliability

Number of defects - Halstead

Halstead's estimated program length \hat{N} :

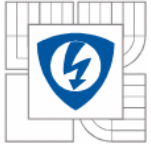
$$\hat{N} = n_1 \log_2(n_1) + n_2 \log_2(n_2)$$

n_1 is a number of unique operators,

n_2 is a number of unique operands.

Number of defects D :

$$D = \frac{E_d \hat{N}}{1000}$$

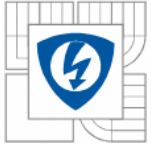


SW Reliability

Number of defects - Lipow

$$\frac{D}{\hat{N}} = A_0 + A_1 \ln \hat{N} + A_2 \ln^2 \hat{N}$$

A_i is dependent on the average number of usages of operators and operands per LOC for a particular language.

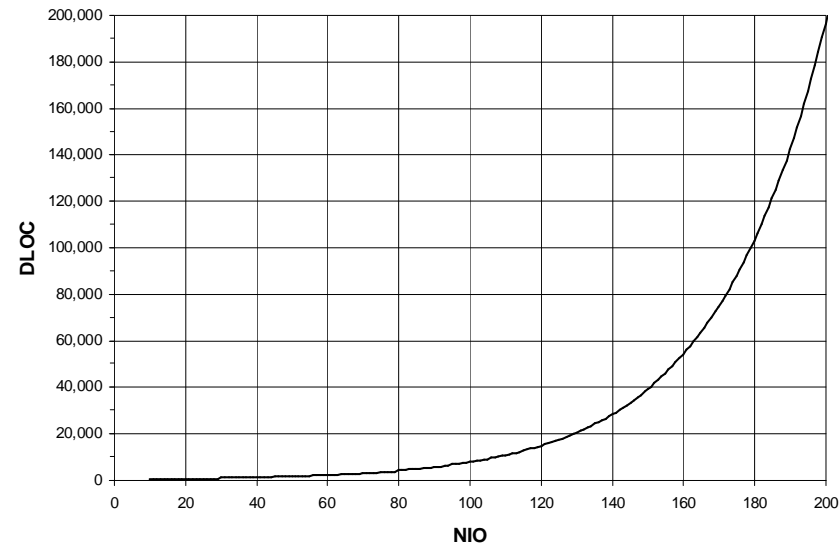


SW Reliability

Complexity of the automation task

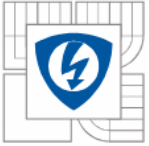
The relationship between number of inputs and outputs (NIO) and the deliverable lines of code in the PLC (DLOC) is an exponential function in the form:

$$DLOC = \alpha e^{\frac{NIO \cdot 10}{\alpha}}$$



where α is a relation coefficient between programming language and number of physical I/O.

For IL (assembler) $\alpha = 310$.

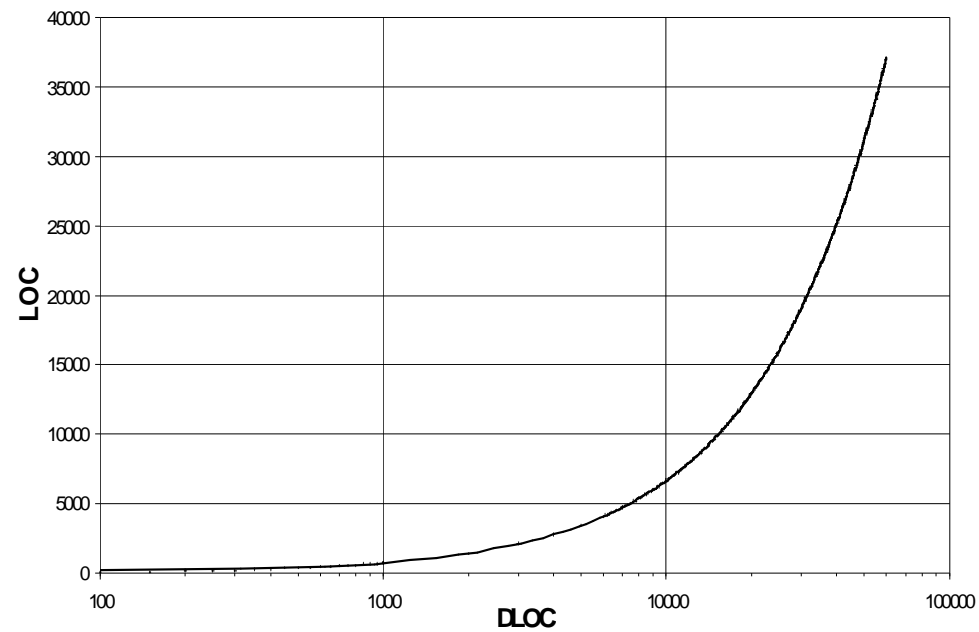


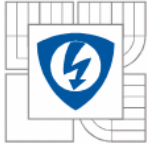
SW Reliability

DLOC vs. LOC

$$LOC = \gamma DLOC^\gamma$$

γ depends on implementation language;
for IL (assembler) $\gamma = 0.96$.





SW Reliability

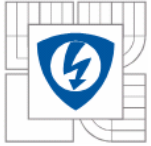
Program Volume and Level

Program Volume V is a metric for the size of any implementation of any algorithm:

$$V = 2 \cdot \hat{N} \cdot \log_2(n_1 + n_2)$$

Program Level L :

$$L = \frac{2n_2}{n_1 \hat{N}}$$

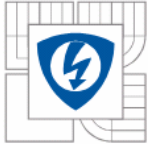


SW Reliability

Programming Effort

Programming Effort E is restricted to the mental activity required to convert an existing algorithm to an actual implementation in a programming language:

$$E = \frac{V}{L}$$



SW Reliability

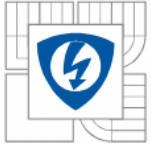
Time of design

The time equation for design time T is:

$$T = \frac{E}{S} [s]$$

Stroud defined a moment as the time required by the human brain to perform the most elementary discrimination. The Stroud number S is then Stroud's moments per second:

$$S \in \langle 5, 20 \rangle$$



SW Reliability

Debugging phase

Goel and Okumoto assume that the number of software failures during non-overlapped time intervals is independent and the software failure rate is proportional the residual fault content:

$$\lambda(t) = \frac{dm(t)}{dt} = b(a - m(t)),$$



$$m(t) = a(1 - e^{-bt})$$

a denotes the initial number of faults contained in a program

b represents the fault detection rate