TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

VYTVOŘENÍ VÝUKOVÉ APLIKACE ŘEŠÍCÍ BLOKOVÉ DIAGRAMY BEZPORUCHOVOSTI

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **Oldřich Taufer**
Vedoucí práce: Ing. Josef Chudoba, Ph.D.

Liberec 2015

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: 2014/2015

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Oldřich Taufer**
Osobní číslo:	**M11000128**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Vytvoření výukové aplikace řešící blokové diagramy bezporuchovosti**
Zadávající katedra:	**Ústav nových technologií a aplikované informatiky**

Zásady pro vypracování:

1. Seznámit se s metodou RBD a minimálně s jedním komerčním softwarem řešící problematiku.
2. Zvolit vhodný programovací jazyk a uvést jeho výhody oproti ostatním
3. Vytvořit výukovou aplikaci s grafickým rozhraním umožňující řešit seriové, paralelní a m z n zálohování.
4. Export a vytvoření výsledků.
5. Sepsat vzorové příklady a návod k aplikaci.

Rozsah grafických prací: **dle potřeby**

Rozsah pracovní zprávy: **40-50 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] Fuchs P., Vališ D., Chudoba J., Kamenický J., Zajíček J., Řízení spolehlivosti, učební text, Technická univerzita v Liberci 2006
[2] ČSN EN 61078 (010677) Techniky analýzy spolehlivosti Blokový diagram bezporuchovosti a booleovské metody, Česká technická norma, 2007
[3] Pecinovský R., Návrhové vzory, Computer press, Praha 2007, ISBN 978-80-2511-582-4
[4] Pecinovský R., OOp naučte se myslet a programovat objektově, Computer press, Praha 2010, ISBN 978-80-2512-126-9
[5] Knuth D.E., Umění programování, Computer press, Praha 2010, ISBN 978-80-2512-898-5

Vedoucí bakalářské práce: **Ing. Josef Chudoba, Ph.D.**
Ústav nových technologií a aplikované informatiky

Datum zadání bakalářské práce: **20. října 2014**
Termín odevzdání bakalářské práce: **15. května 2015**

prof. Ing. Václav Kopecký, CSc.
děkan

Technická univerzita v Liberci - 14 -
L.S.

prof. Dr. Ing. Jiří Maryška, CSc.
vedoucí ústavu

V Liberci dne 20. října 2014

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 15.5.2015

Podpis: 

Poděkování

Mé poděkování patří vedoucímu této bakalářské práce Ing. Josefu Chudobovi, Ph.D. za odborné vedení, cenné rady, trpělivost a odborný dohled.

Mé poděkování patří též všem, kteří se podíleli na testování vyvíjené aplikace.

Abstrakt:

Bakalářská práce se zabývá vývojem aplikace, která řeší spolehlivost systémů pomocí metody RBD. Výhodou této aplikace, oproti demoverzím komerčních softwarů je, že není omezena délkou licence nebo počtem bloků. Aplikace byla vytvořena pomocí programovacího jazyka C#. Vstupní data je možné zadávat přímo z aplikace, nebo textovým souborem. Výsledkem jsou grafy pravděpodobnosti bezporuchového provozu a funkcí okamžité pohotovosti. Důležitou funkcionalitou aplikace je export výsledků i vstupních dat. Součástí práce je i manuál popisující všechny dostupné funkce a vzorové příklady.

Klíčová slova:

RBD, analýza spolehlivosti, aplikace, C#

Abstract:

This bachelor thesis deals with the development of application that solves the reliability of systems by using the RBD method. The advantage of this application in comparison with casual – commercial - software demo versions is not being limited by the duration of the license or number of blocks. The application was created by using the C# programming language. Input data can be entered directly from the application itself or through the text file. The final result is graph of reliable operation probability with immediate alert function. An important functionality of the application is export of the results and input data. The thesis also includes manual describing all available functions and examples.

Keywords:

RBD, reliability analysis, application, C#

Obsah

Seznam obrázků

Seznam zkratek

RBD Reliability block diagram, blokové schéma spolehlivosti

PC Parts count reliability prediction, předpověď bezporuchovosti výpočtem z dílů

FTA Fault tree analysis

ETA Event tree analysis

MA Markov analysis, Markovova analýza

FMEA Fault mode and effect analysis

FMECA Fault mode effects and criticality analysis

.NET Soubor technologií umožňující snadnější propojováních různých zdrojů informací

PNG Portable network graphic

Úvod

V dnešní době, která je prakticky řízená technikou, se musí člověk zabývat spoustou aspektů. Mezi nejvíce diskutované patří kvalita, technické parametry výrobku a cena, ale zapomíná se na neméně důležitou spolehlivost. Parametry spolehlivosti jednotlivých systémů lze spočítat různými metodami (např. RBD, FTA, FMEA, FMECA) a tím zjednodušit a zlevnit provoz.

Metody analýzy spolehlivosti lze uplatnit ve všech průmyslových odvětvích. Mezi nejvýznamnější patří ty, ve kterých by selhání systému mohlo způsobit ztráty na lidských životech. Patří mezi ně například chemický průmysl, zdravotnická technika, výroba dopravních prostředků, transport látek a osob atd.

Mezi nejznámější metody analýzy bezporuchovosti patří:

- RBD (Blokové diagramy bezporuchovosti)
- PC, FTA, ETA, MA, FMEA, FMECA a další

Jejich výhody a nevýhody jsou uvedeny například v pracích:

- Reliability Analysis Technique Comparison, as Applied to the Space Shuttle od Alex Keisner
- METODY ANALÝZ SPOLEHLIVOSTI SYSTÉMŮ A JEJICH VÝBĚR od Doc. Ing. Antonína Mykisky, CSc.

Tématem této práce je vytvořit aplikaci, která stanoví parametry spolehlivosti systému pomocí metody RBD. Aplikace, které se tímto problémem zabývají již existují, ale pro učební účely bývají zbytečně drahé a hlavně složité. Některé z komerčních aplikací budou v této práci porovnány v kapitole 3. Samotná aplikace musí studentům zabývajících se tímto oborem dovolit vyzkoušet si, navrhnout a vytvořit vlastní blokový diagram bezporuchovosti a pohotovosti komplexního systému obsahující například tisíce bloků.

Pro vytvoření aplikace je nejprve potřebné pochopit základní principy této analýzy. V první části práce, přesněji v kapitole 1, naleznete teoretické poznatky i praktické ukázky.

V dalších částech se budu zabývat převážně samotným vývojem aplikace. Popíši jak probíhal návrh systému, výběr programovacího jazyka, rozeberu nejdůležitější třídy a metody a ukáži průběh testování na různých systémech.

Poslední částí bude manuál, který uživatele seznámí s funkcionalitou aplikace. Mimo to v něm bude popsán způsob zadávání komponent, bez kterého se uživatel neobejde.

1 Metoda RBD

1.1 Popis metody RBD

Metoda RBD – tj. metoda blokového diagramu bezporuchovosti – je ve své aplikaci použitelná jako kvantitativní metoda analýza bezporuchovosti. Bývá využívána jako vhodná „příprava“ ostatních kvantitativních metod analýzy spolehlivosti, jako jsou například metody FTA a MA.

Analýza blokového diagramu bezporuchovosti (RBD – Reliability Block Diagram) je deduktivní (shora dolů) metoda analýzy bezporuchovosti systému. Výsledek její aplikace je blokový diagram bezporuchovosti (RBD), který je grafickým zobrazením logické struktury systému v podobně subsystémů, popř. množiny elementárních prvků systému. To umožňuje zobrazit cestu úspěchu (bezporuchového stavu systému) tak, jak jsou bloky (tzn. vyjádření jednotlivých subsystémů a elementárních prvků) logicky propojeny. Struktura zapojení bloků (sériové, paralelní, smíšené atd.) blokového diagramu bezporuchovosti pak vyjadřuje závislost poruchy systému na poruchách jeho komponent (prvků).

Metoda RBD umožňuje analyzovat až několik tisíc prvků systému. Konkrétní limitní počet prvků předurčuje složitost struktury analyzovaného systému. Je způsobilá zpracovávat zálohované struktury a kombinace poruch nebo událostí. Analýza není závislá na charakteru průběhu intenzity poruch. Neanalyzuje případné procesy obnovy a nezpracovává statisticky závislé situace oprav.

Výstupem analýzy pomocí metody RBD jsou kvalitativní a kvantitativní část. V případě užití této metody pouze jako kvalitativní analýzy bezporuchovosti, je zobrazena pouze cesta úspěšného „průchodu“ mezi vstupem a výstupem RBD, která modeluje bezporuchový stav analyzovaného systému (například libovolná porucha nesmí ohrozit provozuschopnost systému). V případě využití jako kvantitativní analýzy, budou výstupem grafy bezporuchovosti a pohotovosti v závislosti na čase. Metodu RBD charakterizuje norma ČSN IEC 1078:1993. [1]

1.2 Komponenta

Základní jednotkou celého systému je tzv. blok neboli komponenta. Samotná komponenta může dosahovat pouze dvou stavů – provozuschopný a poruchový. V systému každý blok symbolizuje jeden funkční celek (jednu složku) reálné soustavy. Pro každou komponentu lze stanovit intenzitu poruch a intenzitu obnov.

Pro ukázku si představme například auto jako celý systém. Jeho jednotlivými komponentami budou kola, motor, převodovka, dostatek paliva, atd. Nutno dodat, že většinu komponent lze rozdělit ještě do menších subsystémů. Např. motor lze rozdělit na válce, sání, alternátor, atd.

1.3 Kvantitativní výstupy metody RBD

1.3.1 Pravděpodobnost bezporuchového provozu bloku

Pravděpodobnost bezporuchového provozu R(t) nám označuje pravděpodobnost závislou na čase, že objekt může vykonávat požadovanou funkci za daných podmínek v daném časovém intervalu (0,t). Bývá označována jako $R_i(t)$, R_B, R_C, …, R_N. [2]

U každého objektu musí být zadefinovaná tzv. konstantní intenzita poruch bloku, která bývá označována jako λ_A. Tato veličina vyjadřuje frekvenci, při které dojde k poruše výrobku.

Výpočet pravděpodobnosti bezporuchového provozu každého definovaného bloku lze stanovit pomocí vzorce:

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

1. Vzorec- Výpočet R(t) bloku

Příklad

Pro názornou ukázku uveďme například žárovku v pravém předním světle u automobilu. Její intenzita poruch bude 0,01 h^{-1}. Pravděpodobnost bezporuchového provozu bloku v simulačním čase 1000 h lze stanovit: $R(t) = e^{-\lambda t} = e^{-1000*0,01} \approx 0,368$

1.3.2 Funkce okamžité pohotovosti

Funkce okamžité pohotovosti A(t) představuje pravděpodobnost, že objekt je ve stavu schopném plnit v daných podmínkách a v daném časovém okamžiku požadovanou funkci, za předpokladu, že požadované vnější prostředky jsou zajištěny.

Funkce okamžité pohotovosti udává s jakou pravděpodobností bude (nebude) výrobek v provozuschopném stavu, pokud při začátku simulace byla komponenta provozuschopná.

Funkci lze stanovit pomocí dob bezporuchového a poruchové stavu, které lze přepočítat na intenzitu poruch a obnov. [3]

V kapitole 1.3.1 je uvedeno, že pro intenzitu poruch se využívá parametr λ. Obdobně pro konstantní intenzitu oprav se používá μ [h^{-1}].

Funkce okamžité pohotovosti v čase je poté označována jako A(t) a její funkční závislost každého bloku lze stanovit pomocí vzorce:

$$A(t) = \frac{\mu}{\lambda + \mu} + \frac{\lambda}{\lambda + \mu} \cdot exp[-(\lambda + \mu) \cdot t]$$

2. Vzorec: Výpočet A(t) bloku

1.4 Struktura komponent

Celková spolehlivost systému je dána topologicky spolehlivostním uspořádáním (strukturou) jednotlivých komponent. Mezi základní typy uspořádání patří sériové, paralelní a M z N zapojení. Každé z nich má své klady a zápory, což bude podrobněji rozepsáno v kapitolách 1.4.1, 1.4.2 a 1.4.3.

1.4.1 Systém bez zálohy

Nejjednodušším zapojením v systému je zapojení sériové. K nefunkčnosti systému zde stačí selhání jediného prvku. Jednotlivé komponenty bývají zakreslovány ve stejném pořadí jako jsou řazeny v reálném systému, ale není to nutná podmínka.

Typickým příkladem je odpor a k němu připojený paralelní kondenzátor připojený k zemi. Při změně hodnoty kapacity nemusí docházet k filtrování šumů.

Pro systémy bez zálohy (sériové zapojení komponent) je pravděpodobnost bezporuchového provozu systému R_s (t) dána výrazem:

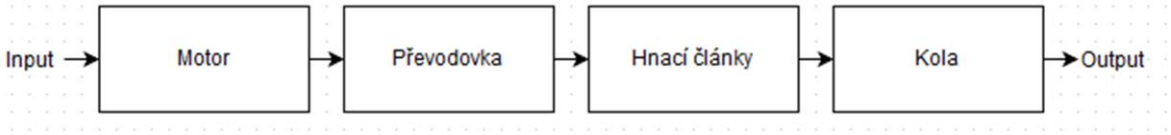
$$R_s = R_a \cdot R_b \cdot R_c \cdot \ldots \cdot R_z$$

3. Vzorec- Výpočet Rs sériového zapojení

tj. vzájemným vynásobením pravděpodobností bezporuchového provozu všech bloků tvořících systém. [2]

Příklad

Příkladem takového zapojení je například hnací motor auta. Jednotlivými prvky jsou motor, převodovka, hnací články a kola. Porucha jediného prvku způsobí kolaps celého



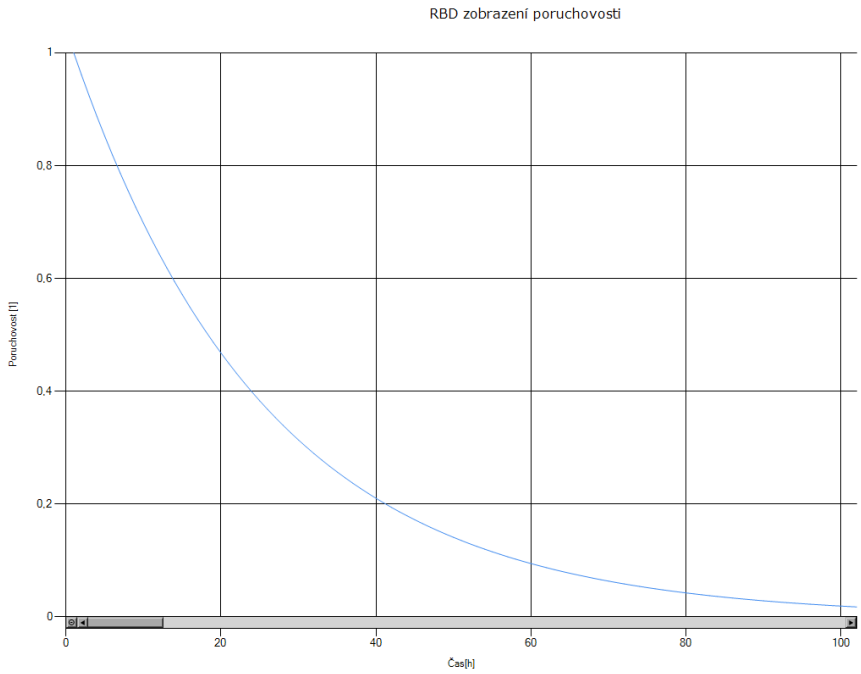
Obrázek 1: Sériové zapojení zobrazené pomocí RBD

systému. Tento systém zakreslený pomocí RBD lze vidět na Obrázek 1: Sériové zapojení zobrazené pomocí RBD

V případě, že každý blok má konstantní intenzitu poruch $\lambda_{a,b,c,d}$= 0,01 h^{-1}, výpočet pravděpodobnosti bezporuchové provozu lze stanovit:

$$R(100) = e^{(-0{,}01 \cdot t)} \cdot e^{(-0{,}01 \cdot t)} \cdot e^{(-0{,}01 \cdot t)} \cdot e^{(-0{,}01 \cdot t)}$$

Průběh poruchovosti tohoto systému v čase 0 až 100 hodin si lze prohlédnout na obrázku Obrázek 2: Zobrazení poruchovosti sériového systému.



Obrázek 2: Zobrazení poruchovosti sériového systému

1.4.2 Paralelní zálohování

Jedním ze způsobů jak zvýšit spolehlivost systému je použití paralelního zálohování. Pro toto zapojení je charakteristické větvení komponent. Obsahuje uzly a bloky, které mohou být umístěny v různých větvích. Selhání systému nastává až v případě, kdy dojde k poruše všech funkčních bloků v paralelním zapojení.

Tyto systémy mají pravděpodobnost bezporuchového provozu systému R_s danou výrazem:

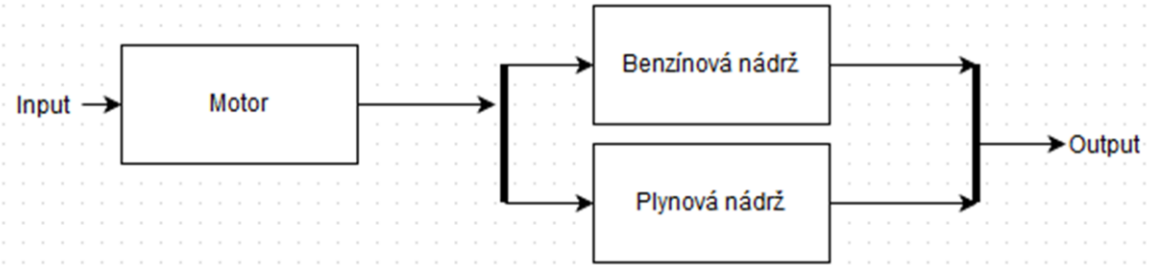
$$R_s = 1 - \sum_{i}^{n} 1 - Ri$$

4. Vzorec- Výpočet Rs paralelního zapojení

V těchto systémech je potřeba zohlednit i tzv. poruchy se společnou příčinou (značí se CCF), které mají tu vlastnost, že z jisté příčiny nastane ztráta funkce na více (v nejhorším případě všech) komponentách systému souběžně a celý systém přestane plnit požadovanou funkci.

Může se jednat například o ztrátu napájení, poruchu vyhodnocovacího členu, klimatický jev, mechanické poškození, apod. [4]

Příklad



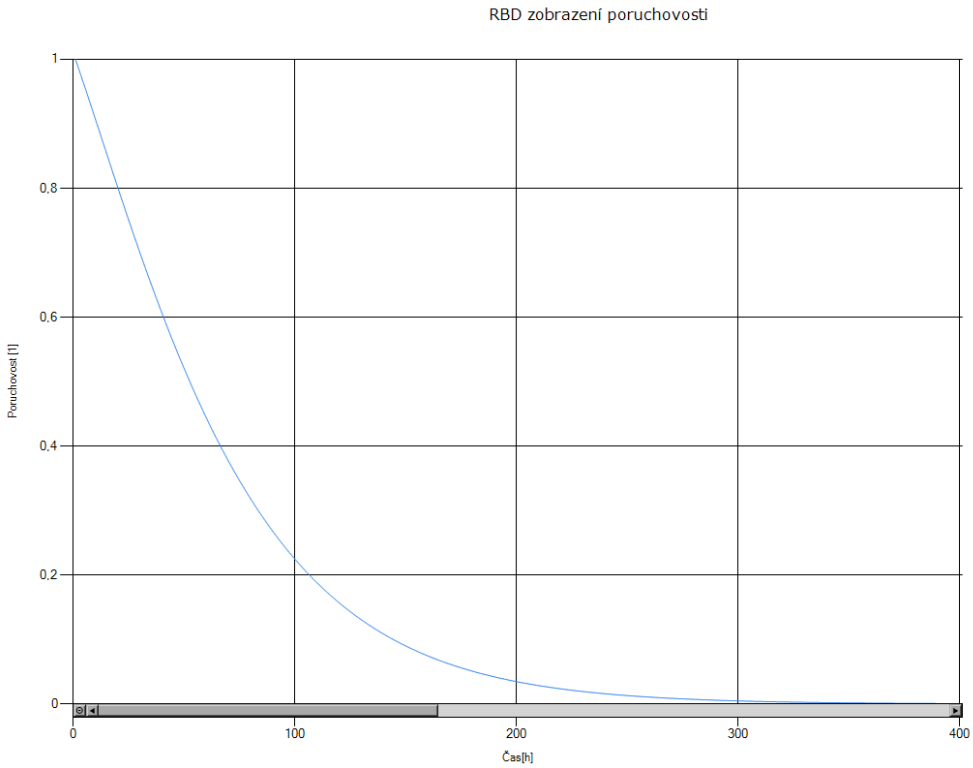
Obrázek 3: Paralelní zapojení zobrazené pomocí RBD

Příkladem takového zapojení mohou být například dvě nádrže u automobilu. Ve chvíli kdy jedné z nich dojde palivo, nebo nastane nějaká porucha, tak bude vozidlo stále funkční. Ve chvíli kdy se porouchají obě, tak systém přestane fungovat. Tento systém zakreslený pomocí RBD lze vidět na Obrázek 3: Paralelní zapojení zobrazené pomocí RBD.

Pokud budeme uvažovat, že každá komponenta v takovémto systému bude mít hodnotu $\lambda_{a,b,c}$= 0,01, bude výpočet poruchovosti vypadat takto:

$$Rs(t) = e^{(-0{,}01 \cdot t)} \cdot (1 - (1 - e^{(-0{,}01 \cdot t)}) \cdot (1 - e^{(-0{,}01 \cdot t)}))$$

Průběh poruchovosti tohoto systému v čase 0 až 400 hodin si lze prohlédnout na obrázku Obrázek 4: RBD zobrazení poruchovosti paralelního systému.



Obrázek 4: RBD zobrazení poruchovosti paralelního systému

1.4.3 M z N zálohování

Tento typ zálohování patří obdobně jako Paralelní zálohování mezi tzv. redundantní zapojení. Je reprezentováno veličinami M a N, kde M označuje počet větví, které musí být minimálně v provozuschopném stavu z celkového N prvků. Toto zálohování bývá používáno převážně při návrhu kritických systémů.

Tyto systémy mají pravděpodobnost bezporuchového provozu systému R_s danou několika výrazy, které závisejí na specifických vstupních podmínkách.

Prvním případem je zapojení, kde k bezporuchovému stavu stačí funkční pouze jedna větev. V tu chvíli se jedná o paralelní zapojení a výpočet R_s je popsaný v kapitole 1.4.2.

Další variantou, která může nastat, je zapojení M z N, kde všechny prvky mají stejné hodnoty. V takovém případě lze R_s spočítat jako:

$$R_s(m,n) = \sum_{r=m}^{n} \binom{n}{i} R^i (1-R)^i$$

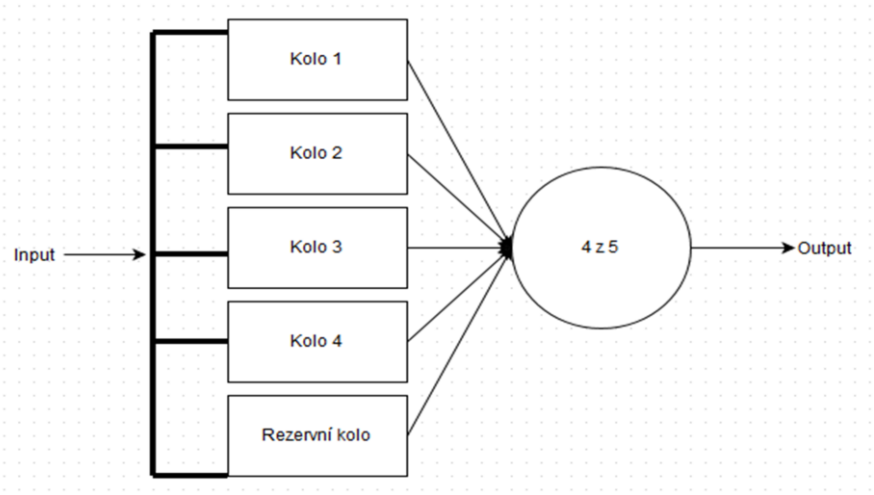
5. Vzorec- Výpočet M z N stejných prvků

Například pro konfiguraci M=2 a N=3, kde každá komponenta má jiné parametry λ (μ), lze stanovit výslednou bezporuchovost pomocí vzorce:

$$R_s = R_1R_2R_3 + R_1R_2(1-R_3) + R_1R_3(1-R_2) + R_2R_3(1-R_1)$$

6. Vzorec- Výpočet M z N různých prvků kde M = 2 a N = 3

V situaci, kdy máme různé M a N, musíme postupovat jiným způsobem, protože počet možných kombinací je programovatelně nepopsatelný. Je nutné spočítat průměr hodnot ze všech N a postupovat podle vzorce 5. Vzorec- Výpočet M z N stejných prvků. V systému se tímto vytvoří pouze zanedbatelná odchylka od reálných hodnot.



Obrázek 5: Zapojení M z N zobrazené pomocí RBD

Příklad

Typickém příkladem tohoto zapojení jsou kola u automobilu, pokud počítáme s tím, že časový interval pro výměnu za rezervní, není brán jako výpadek systému. Názorné zobrazení je vidět na Obrázek 5: Zapojení M z N zobrazené pomocí RBD.

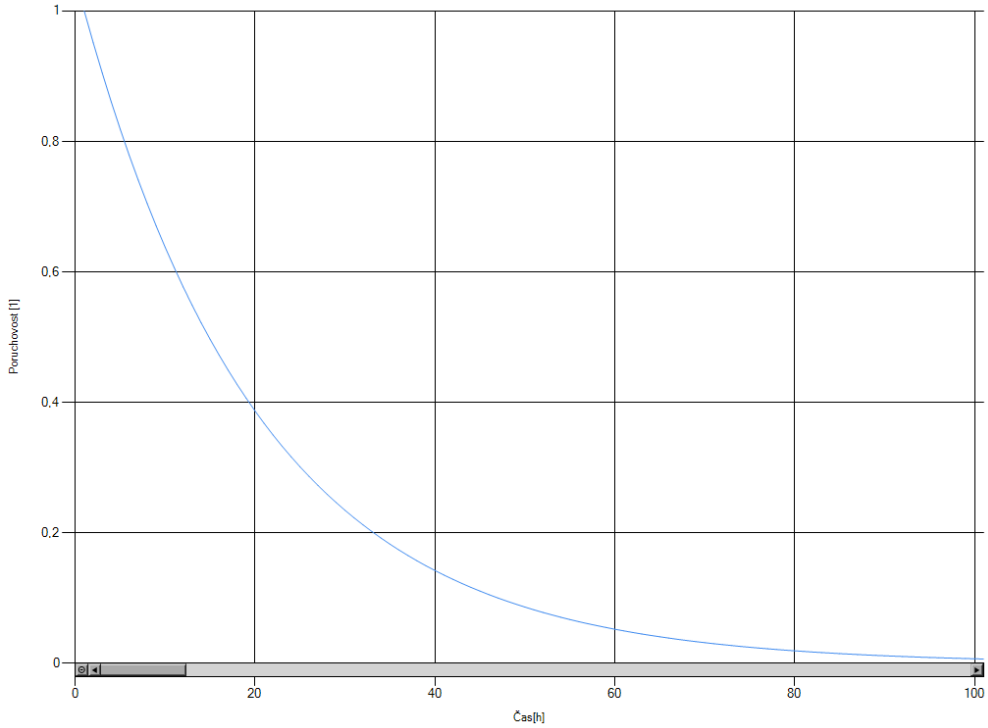
Vezměme v potaz, že každé kolo bude mít λ_k= 0,01 h^{-1}, kromě rezervního, které bude mít λ_r= 0,05 h^{-1}. V takovém případě nezbývá nic jiného, než počítat podle vzorce s průměrováním intenzit poruch. Průměrná lambda jednoho prvku bude:

$$\lambda = (4 \cdot \lambda_k + \lambda_r)/5\ 0{,}018h^{-1}$$

Teď už lze počítat s vzorcem pro 5 stejných prvků:

$$R(t) = \sum_{i=4}^{5} \binom{5}{i} R(t)^i (1 - R(t))^{(5-i)}$$

Graf pravděpodobnosti bezporuchového provozu systému lze vidět na grafu Obrázek 6: Graf poruchovosti systému M z N.

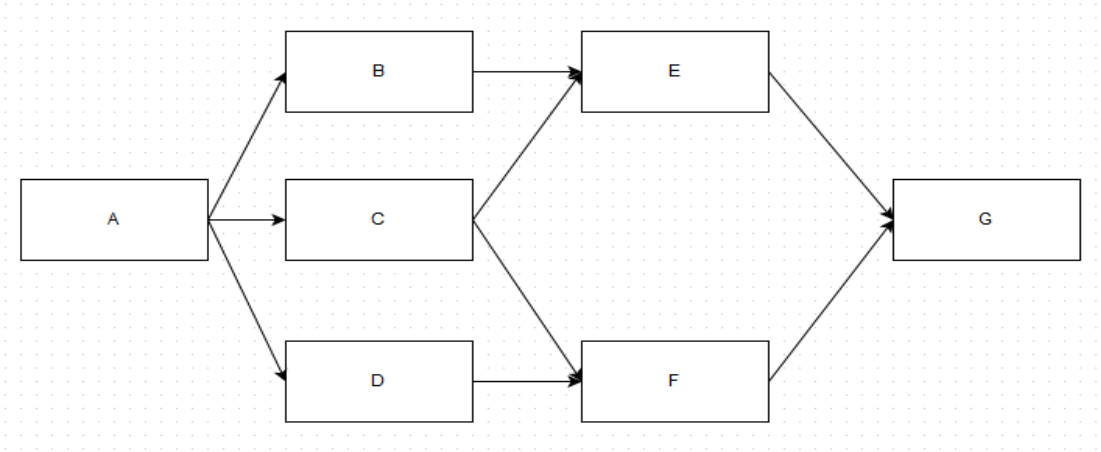


Obrázek 6: Graf poruchovosti systému M z N

1.4.4 Krizová zapojení

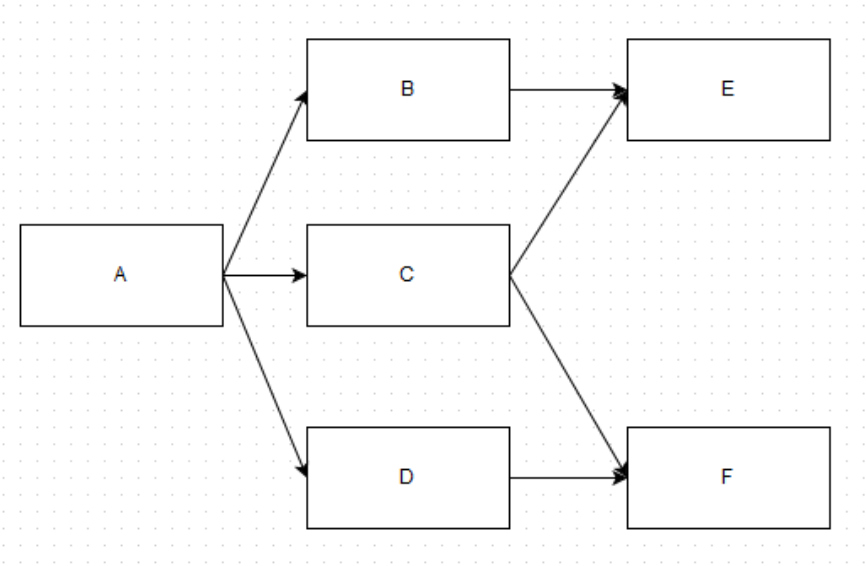
V komplexních systémech může vždy nastat určitý druh zapojení, který není možné postupně zjednodušit pomocí základních zapojení uvedených dříve. V této kapitole bude ukázka některých z nich.

Prvním je zapojení znázorněné na obrázku Obrázek 7: Nerozpoznatelný typ zapojení. Nelze zde zjednodušit paralelní bloky E a F, neboť jsou oba součástí jiných sériových větví.



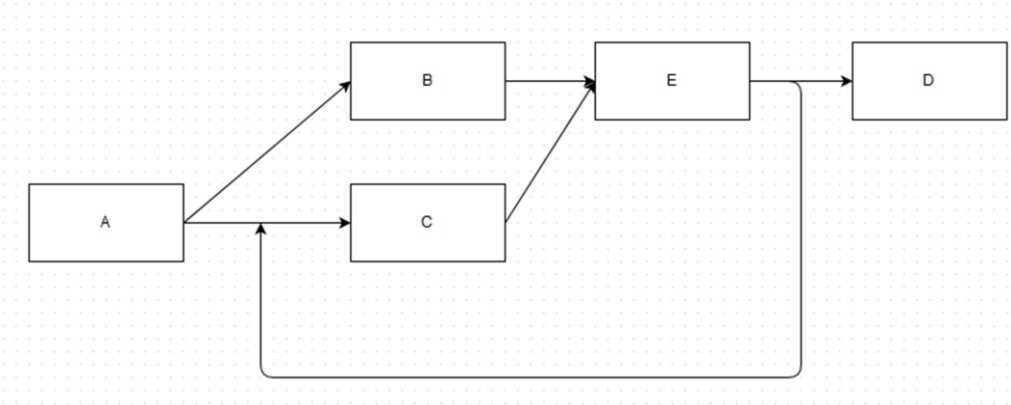
Obrázek 7: Nerozpoznatelný typ zapojení

Dalším takovým zapojením je schéma zobrazené na obrázku Obrázek 8: Zapojení s dvěma výstupy. Jedná se zde sice o zjednodušený případ nerozpoznatelného zapojení, ale není zde uveden jednoznačný výstup.



Obrázek 8: Zapojení s dvěma výstupy

Posledním zapojením, o kterém se zde zmíním, je tzv. zacyklení. Schéma si lze prohlédnout na obrázku Obrázek 9: Zacyklení. Jak je z ilustrace patrné, bloky E a C směřují do kruhu.



Obrázek 9: Zacyklení

1.4.5 Sestavení systému

V případech, kdy má systém jen jednu funkci, je sestavení diagramu jednoduché. Takový systém bude tvořit pár skupin prvků v sérii, popřípadě pár skupin s redundancí. U komplexních systémů s více funkcemi je potřeba dodržovat určité postupy.

Prvním krokem je volba specifikace úspěchu/poruchy systému. Jestliže je možná více než jedna specifikace, může být nutné pro každou z nich vypracovat samostatný blokový diagram bezporuchovosti. Následujícím krokem je rozdělení systému do bloků, které odrážejí logické chování tak, že je každý blok statisticky nezávislý na jiných blocích a je co největší. Současně nemá žádný blok (pokud možno) obsahovat žádné zálohy.

Než se vypracuje konečný vhodný blokový diagram, může být v praxi nezbytné konstruovat blokový diagram bezporuchovosti na několikrát (přičemž je vždy nutné mít na zřeteli výše uvedené kroky).

V následujícím kroku je nutné se odvolat na specifikaci poruchového stavu systému a zkonstruovat diagram, který spojí bloky a vytvoří "cestu úspěchu". [2]

1.4.6 Funkčnost systému

Funkční systém vyžaduje vždy alespoň jednu cestu mezi vstupem a výstupem. K popisu minima kombinací, které způsobí selhání systému, může být použita Booleova algebra.

Pro definování provozuschopného systému lze sériové zapojení převést na hradlo AND. Obdobně paralelní na hradlo OR. M z N vyžaduje složitější zápis pomocí logických hradel vyplývající z počtu M, N. Na řešení systému lze aplikovat De Morganův teorém.

Například systém kde blok A a B budou spolu propojeny sériově a s blokem C paralelně lze zapsat jako:

$$R = (A \wedge B) \cup C$$

Pro jednodušší zapojení lze místo jednotlivých bloků zakreslit spínače. Sepnutý spínač by reprezentoval funkční komponenty a rozepnutý nefunkční. V takovém případě je systém stále v provozu, pokud existuje alespoň jedna cesta od vstupu (prvnímu prvku) k výstupu (poslednímu prvku).

2 Návrh aplikace

Při návrhu této aplikace je hlavním kritériem to, že bude využívána převážně studenty. Aplikace by však mohla být provozována i v komerční sféře, protože její omezení v počtu hradel oproti placeným softwarům není zaznamenatelné. Z toho plyne, že je potřeba eliminovat co nejvíce možností, kde by mohl uživatel udělat chybu. Mezi takové případy patří zacyklení, neuvedené vstupní hodnoty, atd. Většina komerčních programů sice nabízí možnost zadat systém pomocí grafického prostředí, ale já jsem se rozhodl, že použiji textový vstup s následným vykreslením systému. U takového vstupu se dají lépe odhalovat uživatelské chyby a celkové zapsání systému může být rychlejší. Jednou z nevýhod tohoto způsobu zadávání je, že uživatel musí mít větší systém již promyšlený a nejlépe načrtnutý.

2.1 Výběr jazyka

2.1.1 Rozhraní aplikace

Při výběru programovacího jazyka je potřeba se zabývat dvěma hlavními faktory. Prvním z nich je možnost grafického vykreslování (struktura zapojení, grafy) a druhým celková rychlost a přesnost matematických operací. Vzhledem k rozsáhlým možnostem grafického vykreslování jazyka C#, jsem se rozhodl, že základní rozhraní aplikace bude napsáno v něm. Pro matematické operace mi připadalo nejvýhodnější použít skriptovací jazyk Matlab, ale rozhodl jsem se udělat krátké testování vybraných programovacích jazyků, aby bylo jasné, zda se externí skript pro výpočty vyplatí použít či ne. Mezi mnou vybrané testovací jazyky patří C, C#, Java, Matlab, Python a R.

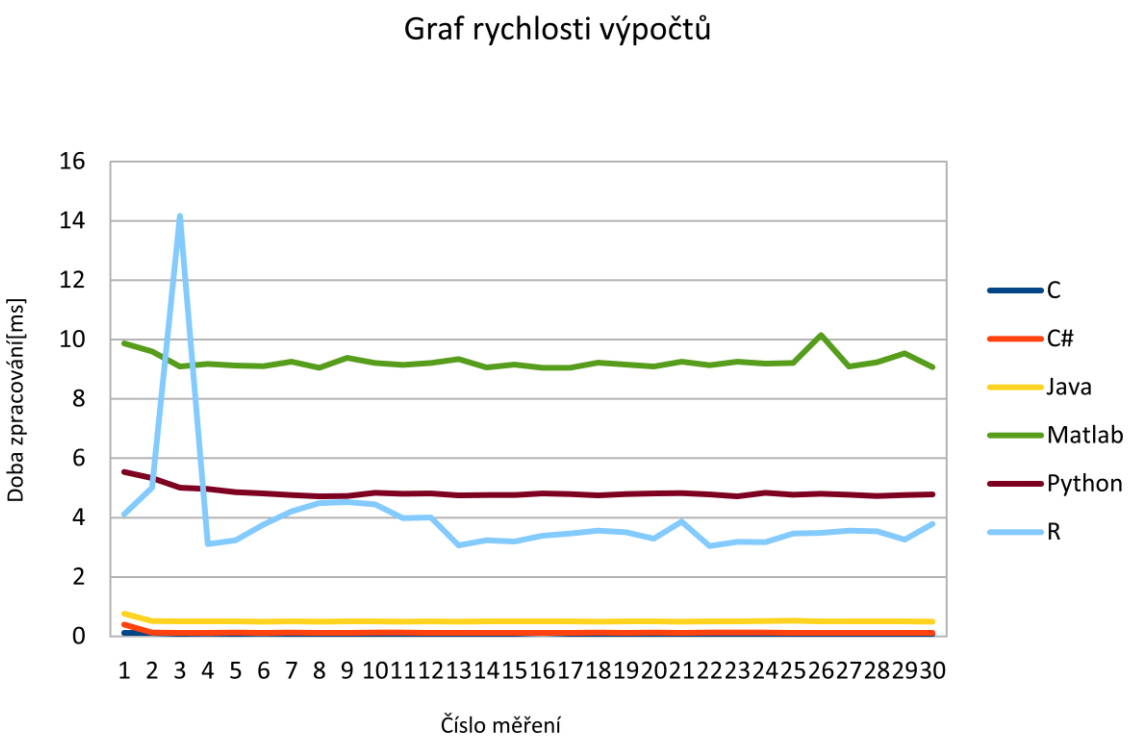
2.1.2 Rychlost matematických výpočtů jednotlivých jazyků

Vzhledem k účelu aplikace se vytvořil vzorový příklad z matematických operací využitých při výpočtech bezporuchového provozu a okamžité pohotovosti a aplikoval se nad jednotlivými jazyky. Ze vzorců popsaných v kapitolách 1.4.1, 1.4.2 a 1.4.3 lze odvodit, že využíváno bude především sčítání, odčítání, násobení, dělení a mocnina. Je zapotřebí také otestovat, jak jednotlivé jazyky pracují s množinou dat a cykly, proto výpočty probíhají vždy pro tisíc prvků. Vzorový příklad byl navržen tak, aby nedocházelo k přetečení základních datových typů v jednotlivých jazycích a v konečné fázi získal podobu:

$$X = 1{:}\,1{:}\,1000$$
$$result[i] = X_i^4 + X \div 2$$

Nad každým programovým jazykem bylo provedeno celkem 30 měření a výsledky si můžete prohlédnout na Obrázek 10: Výsledky všech testovaných měření. Překvapivě se

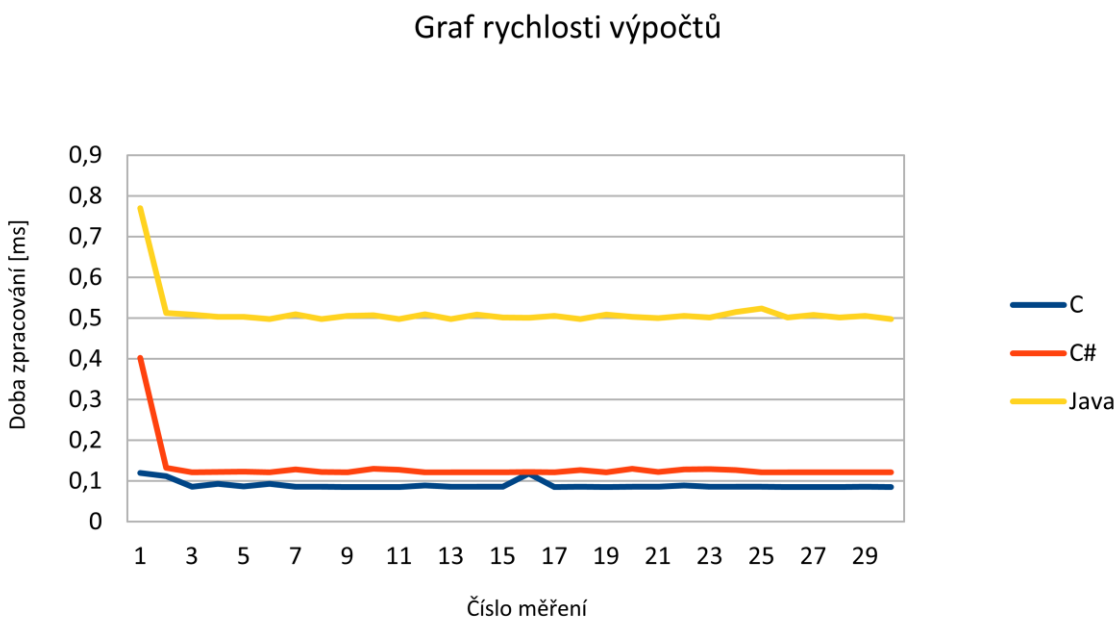
mezi třemi nejrychlejšími jazyky umístily C, C# a Java. Matlab se umístil až na posledním místě, proto v aplikaci využíván nebude.



Obrázek 10: Výsledky všech testovaných měření

Když se podíváme na Obrázek 11: Výsledky třech nejrychlejších programovacích jazyků, zjistíme, že jsou zde rozdíly opravdu minimální. Přičítám to tomu, že Java a C# jsou odvozené z jazyka C.

Na závěr této kapitoly je nutné uvést, že i přes to, že pro srovnání byl použit VirtualBox s čistě nainstalovanými Windows, aby měly všechny kompilátory a skripty stejné podmínky, tak jsou výsledky spíše orientační. Nehledě na to, výsledky se liší v řádu ms, což je pro běžného uživatele nepostřehnutelný rozdíl a proto jsem se rozhodl, že bude celá aplikace napsaná v programovacím jazyce C#.



Obrázek 11: Výsledky třech nejrychlejších programovacích jazyků

2.2 Návrh tříd

Celá aplikace bude rozdělena do tří hlavních částí: *Entities*, *Parsers* a *Views*. Do *Entities* patří základní datové složky, sloužící pro uchování návrhu systému a jeho struktury. Část *Parsers* slouží ke zpracování textových vstupů (zadávaných z aplikace nebo exportovaných ze souboru) a poslední část *Views* slouží k vykreslení celé aplikace.

2.2.1 Entities

Složka *Entities* bude obsahovat základní interface *IElement*, který má jedinou metodu, a to *CalculateFailure*, se vstupními proměnnými polem integerů a textovým řetězcem. Pole integerů symbolizuje časy, pro které se má poruchovost počítat a textový řetězec odlišuje, jestli počítáme pravděpodobnost bezporuchového provozu R(t) nebo funkci okamžité pohotovosti A(t). Návratová hodnota této metody bude kolekce typu slovník, kde klíčem je čas pro který se aktuálně počítá poruchovost (pohotovost) a hodnotou bude výsledek v daném čase typu double.

Od tohoto interfacu budou odvozené další třídy *Element*, *SerialConnection*, *ParallelConnection* a *KFNConnection*.

Třída *Element* bude symbolizovat nejmenší komponentu v daném systému, respektive prvek u kterého je zadána hodnota λ, nebo λ a μ. Z toho plyne, že třída bude muset mít dva

konstruktory. První, ve kterém se zadává pouze intenzita poruch a název komponenty. A v druhém se přidá k intenzitě poruch ještě intenzita oprav. Ostatní třídy určují pouze strukturu zapojení těchto bloků. Výpočet hodnot pro jednotlivé časy bude naprogramován podle vzorce 1. Vzorec- Výpočet R(t) bloku.

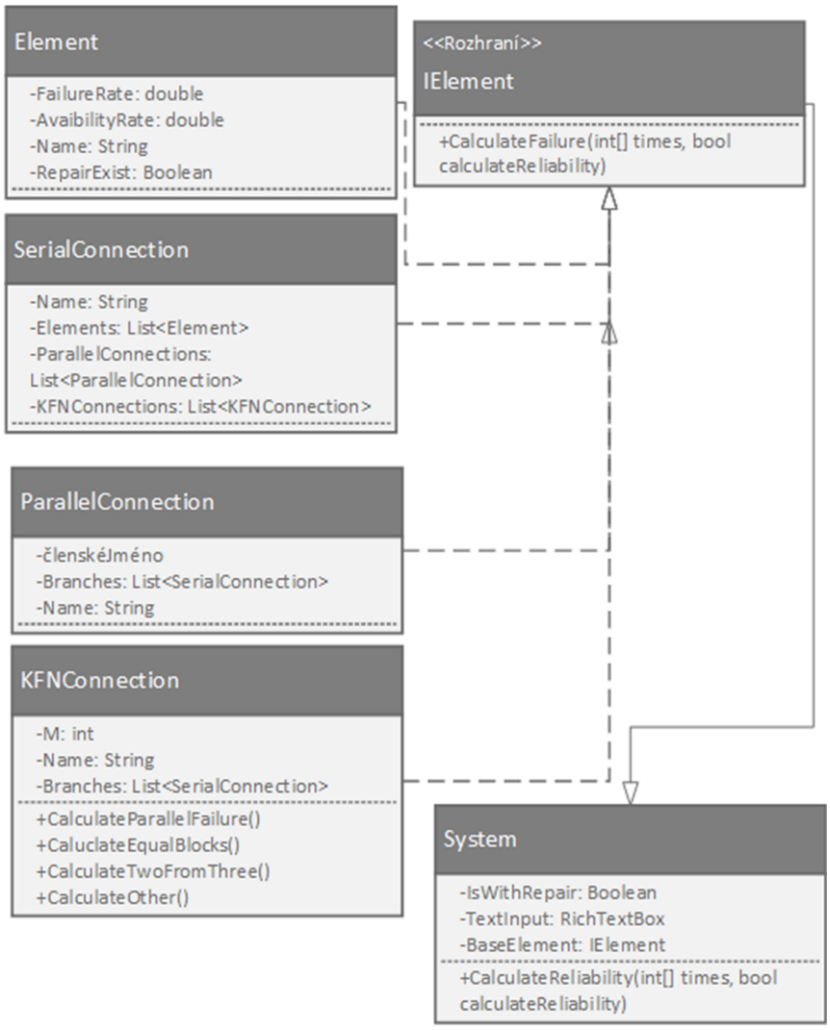
Nejzákladnější sériové zapojení, uvedené v kapitole 1.4.1, bude reprezentovat třída *SerialConnection*. Mezi proměnné této třídy musí patřit kolekce prvků, paralelních zapojení a M z N zapojení. Výpočet poruchovosti a pohotovosti je realizován postupným násobením výsledků všech ostatních komponent a systémů podle vzorce 3. Vzorec Výpočet Rs sériového zapojení.

Paralelní zapojení bude reprezentované třídou *ParallelConnection*. Každá větev tohoto zapojení bude reprezentována zapojením sériovým. O uchování této informace se bude starat kolekce typu List, obsahující sériová zapojení. Poruchovost takového systému je dána vzorcem 4. Vzorec- Výpočet Rs paralelního zapojení, kde vstupními proměnnými budou všechna sériová zapojení v dané třídě.

Zapojení M z N je téměř totožné jako zapojení paralelní (jedná se také o redundantní zapojení), ale přibývá zde proměnná N, která udává, kolik větví je potřebných k fungování tohoto systému. Samotný výpočet bude rozdělen do několika privátních tříd, neboť výpočty jsou závislé na hodnotách M a N, jak je uvedeno v kapitole M z N zálohování.

Poslední třídou v tomto balíčku bude *System*, který bude sloužit k uchování struktury právě počítaného systému. Bude udržovat *IElement*, textový vstup a boolean hodnotu pro rozhodnutí, zda má systém všechny potřebné informace k výpočtu pohotovosti (viz. kapitola 1.3.2).

Návrh datových tříd tohoto systému si lze prohlédnout na Obrázek 12: UML- Datové třídy.



Obrázek 12: UML- Datové třídy

2.2.2 Views

Pro vykreslování dat a ovládání aplikace bude sloužit část *Views*. Ta bude dále rozdělena na *Controls* a *Forms*. Složka *Forms* bude obsahovat všechny *Windows forms* se kterými aplikace pracuje a v *Controls* se budou nacházet tzv. *UserControls* (vlastní komponenty vytvořené uživatelem).

Nejdříve je potřeba popsat všechny uživatelské komponenty, které budou vytvořeny. První z nich je *InputTextControl*. Jedná se o *RichTextBox*, který bude sloužit pouze k zadávání vstupních dat přímo z aplikace.

Druhou je *CanvasControl*. Tato komponenta bude sloužit k vykreslování dat vybraného systému. Bude obsahovat tři komponenty pro vykreslování jednotlivých bloků schématu. Nejzákladnější bude *DrawSeriesSystem()*, která bude mít za úkol vykreslování sériového zapojení systému. Zbylé dvě metody (*DrawParallelSystem()* a *DrawMFNSystem*) pro

vykreslování redundantních systémů budou vykreslovat pouze jednotlivé větve těchto zapojení.

Poslední komponentou bude *SystemElementControl*. Je to komponenta o určené velikosti, kterou překrývá *Label* s názvem prvku či vnitřního systému. Název komponenty (text *Labelu*) se bude generovat a kreslit již v konstruktoru. Jednou z nejdůležitějších metod této komponenty bude dvojitý klik na ni. Tato metoda musí rozpoznat o jaký typ komponenty se jedná. Pokud půjde o paralelní nebo M z N zapojení, tak se otevře nový formulář a vykreslí větve vybraného systému. Pokud se jedná o typ *Element*, musí se zobrazit formulář s výčtem zadaných hodnot a názvem.

Část *Forms* musí, z principu fungování jazyka C#, obsahovat hlavní formulář, který bude v této aplikaci defaultně pojmenovaný *Form1*. Tento formulář se bude skládat ze dvou částí. Komponenty *CanvasControl* jejíž význam jsem vysvětlil dříve a hlavního menu sloužícího pro plnohodnotné ovládání aplikace.

Hlavní část menu se bude skládat z položek *Nový* a *Ukončit*. Jejich funkčnost není třeba nijak podrobně popisovat. *Nový* vymaže již zadaný systém a *ukončit* vypne aplikaci.

Nejdůležitější částí menu budou *Úpravy* a je třeba se jimi zabývat. První položkou v úpravách bude *Vložit textový vstup*. Při kliknutí na něj se vyvolá *TextInputForm*. Tento formulář bude obsahovat *RichTextBox* pro zadání struktury systému a dále potvrzovací tlačítko, které předá data *Parseru* pro validaci. V případě, že zadaný systém validací projde, bude vykreslen v hlavním okně a vyvolá se *ViewResultForm*.

Dalšími položkou bude *Načíst systém ze souboru*. Načtení systému musí fungovat téměř stejně jako *TextInputForm*, ale data do parseru budou vstupovat z externího souboru.

Poslední částí menu je část O Aplikaci. Zde se uživatel může dostat k informacím o verzi aplikace, poděkování vedoucímu bakalářské práce a manuálu. Manuál je zde pravděpodobně nejpodstatnější část, protože zde nebudou uvedeny jen jednotlivé funkce systému, ale i způsob zadávání komponent do textové části a manuální vytváření externích souborů.

Aplikace bude umožňovat exportovat výsledky z *ViewResultForm*. Data se exportují do souborů obsahujících výsledný graf, spočítané hodnoty poruchovosti (pohotovosti) v jednotlivých časech a vstupní data.

2.3 Parsers

Jednou z nejdůležitějších částí aplikace je složka *Parsers*. Ta bude obsahovat třídy, které

poslouží ke zpracování vstupních textů.

TextInputParser aplikace bude rozpoznávat, zda je uživatelský kód ve správném formátu a hlásit případné chyby. Rozezná jednotlivé prvky systému a poskládá je do zapojení popsaném v kapitole 2.2.1.

FileToTextParser se bude starat o načtení dat ze souboru a jejich následné předání do *TextInputParser.*

3 Srovnání existujících komerčních softwarů

Tato kapitola se bude zabývat hlavními výhodami a nevýhodami existujících programů sloužících pro výpočty spolehlivostí systému pomocí metody RBD. Rozhodl jsem se zhodnotit jedny z nejpoužívanějších softwarů:

- ITEM toolkit (http://www.itemsoft.com/item_toolkit.html)
- PTC windchill (http://www.ptc.com/product-lifecycle-management/windchill)
- Isograph (http://www.isograph.com/software/reliability-workbench/rbd-analysis/)
- RAM Commander (http://aldservice.com/)

Všechny softwary budou srovnávány na jedné počítačové sestavě. Tato sestava se skládá z procesoru Intel inside CORE i5 vPro, 500GB SSHD, 4GB operační paměti a operačního systému Windows 7 Professional.

3.1 PTC Windchill Quality Solution

Jak výrobce toho softwaru píše na svých stránkách, jedná se o software jenž nabízí komplexní funkce, které pomáhají výrobcům spravovat své výrobky ve všech fázích životního cyklu výrobku. Software by měl sloužit společnostem ke zrychlení uvádění produktů na trh, zlepšit kvalitu, výkonnost a snižovat rizika. [5]

3.1.1 Stažení a instalace

Na oficiálních stránkách aplikace:

www.ptc.com/product-lifecycle-management/windchill

je volně ke stažení 30denní trial verze. Ke stažení je ovšem nutné vyplnit krátký dotazník obsahující osobní informace, včetně účelu použití aplikace. Na konci dotazníku může uživatel odškrtnout kolonku, že si přeje být kontaktován. Do druhého dne se z firmy ozvali a zjišťovali, zda by naše univerzita s nimi nechtěla spolupracovat. Vzhledem ke komplexnosti aplikace je její velikost v komprimovaném stavu necelých 300MB, což je poměrně dost.

Po rozbalení aplikace na pevný disk se objevil jediný soubor a to instalátor. Instalace je u tohoto produktu tzv. user-friendly a nevyskytly se žádné problémy. Jediným zklamáním byla absence české lokalizace (stejně jako v celé aplikaci) a doba instalace, která trvala přibližně 10 minut.

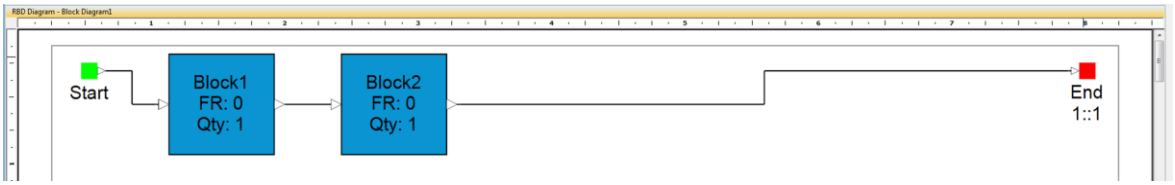
3.1.2 Rozhraní

Po spuštění aplikace se v prvním okně zobrazí, jakou analýzu budeme používat. Lze zde

nalézt např. ALT, Event Tree, Markov, FMEA a spoustu dalších. Pro účely bakalářské práce bylo využito pouze RBD, které je v základním výčtu také.

Po tomto výběru už vidíme základní rozhraní aplikace, kde můžeme vybírat mezi rozpracovanými projekty nebo založit nový. Založení nového projektu trvalo asi jednu minutu, mezi kterou program tzv. neodpovídal.

Rozhraní pro daný projekt se skládá ze dvou částí. Z project navigator, který zobrazuje prvky v textové podobě a plátna, na které se dají objekty zanášet. Propojení těchto bloků se dělá tak, že člověk postupně kliká na vstupy a výstupy daných objektů a spojnice se dokreslují automaticky. Jednoduché schéma sériového zapojení lze vidět na obrázku Obrázek 13: Sériové zapojení v PTC Windchill Quality Solutions.



Obrázek 13: Sériové zapojení v PTC Windchill Quality Solutions

Je třeba se ještě zmínit o tom, že když uživatel vytvoří paralelní zapojení, tak se automaticky na jeho konečný uzel přidá červený blok, značící kolik funkčních větví je potřeba pro správnou funkčnost systému.

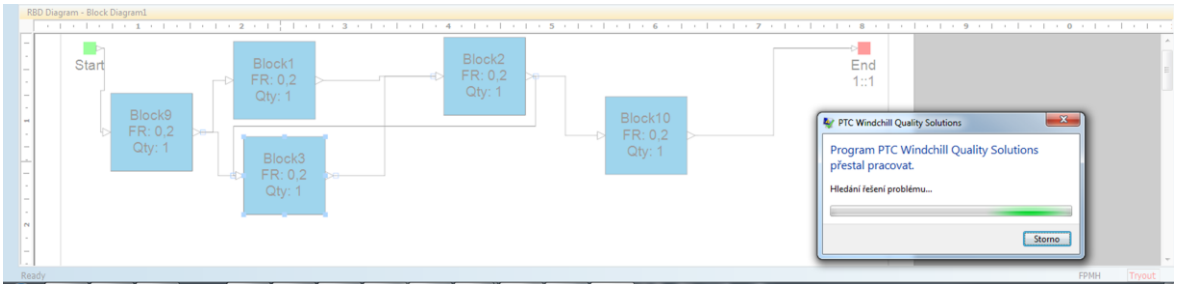
Intenzitu poruch a oprav lze nastavovat jednotlivým blokům po dvojitém poklepání na ně. Zobrazení výsledků v časech, popřípadě zobrazení grafu, poté můžeme vyvolat z hlavního menu.

3.1.3 Krizové případy

Případ, který je vidět na Obrázek 7, jde do systému zanést bez problémů. Poté proběhne i výpočet a zobrazení grafu, což pravděpodobně značí to, že výpočty probíhají nejprve zjednodušením přes Booleovu algebru.

Zapojení z obrázku Obrázek 8: Zapojení s dvěma výstupy není v tomto programu možné realizovat.

Zacyklení, které je vidět na Obrázek 9: Zacyklení, v tomto programu není vůbec dobře ošetřeno. Několikrát jsem vyzkoušel i různé jiné případy tohoto problému a aplikace pokaždé přestala pracovat, jak je vidět na obrázku Obrázek 14: PTC windchill špatně ošetřené zacyklení. Je dost možné, že se jedná jen o bug v aktuální verzi, který bude brzy odstraněn.



Obrázek 14: PTC windchill špatně ošetřené zacyklení

3.1.4 Závěr a zhodnocení

Tato aplikace toho ve své podstatě umí přehršel, ale jako učební pomůcka mi nepřijde příliš vhodná. Mezi její hlavní zápory patří zdlouhavé načítání, některé neodladěné funkce, nepříliš intuitivní ovládání a především cena, která se pohybuje okolo 200 $. Navzdory tomu se jedná o velice komplexní a silný nástroj, který jistě spousta vývojových firem ocení.

3.2 RAM Commander

Jak píše výrobce aplikace A.L.D., jedná se o komplexní softwarový nástroj pro spolehlivost, udržovatelnou analýzu a predikci, optimalizaci náhradních dílů, FMEA/ FMECA, testovatelnost, analýzu stromové struktury a posuzování bezpečnosti. Spolehlivostní a bezpečností moduly by měli pokrývat všechny obecně známé spolehlivostní standardy a analýzy poruch. Měl by to být nepostradatelný nástroj pro zajištění spolehlivosti sofistikovaných systémů. [6]

3.2.1 Stažení a instalace

Instalátor této aplikace najdeme, stejně jako u jeho konkurenta PTC Windchill popsaného v kapitole 3.1, na oficiálních stránkách aplikace (http://www.aldsoftware.com). Na výběr jsou zde tři verze: DEMO, plná a studentská. Plná verze a studentská vyžaduje autorizační kód, a proto jsem si vybral demo verzi, ve které je jediné 30 denní omezení. Po vyplnění krátkého formuláře, na jaké účely bude aplikace používána, začalo stahování přibližně 60MB velké aplikace.

Při instalaci se nevyskytl jediný problém, ale opět chyběla česká lokalizace.

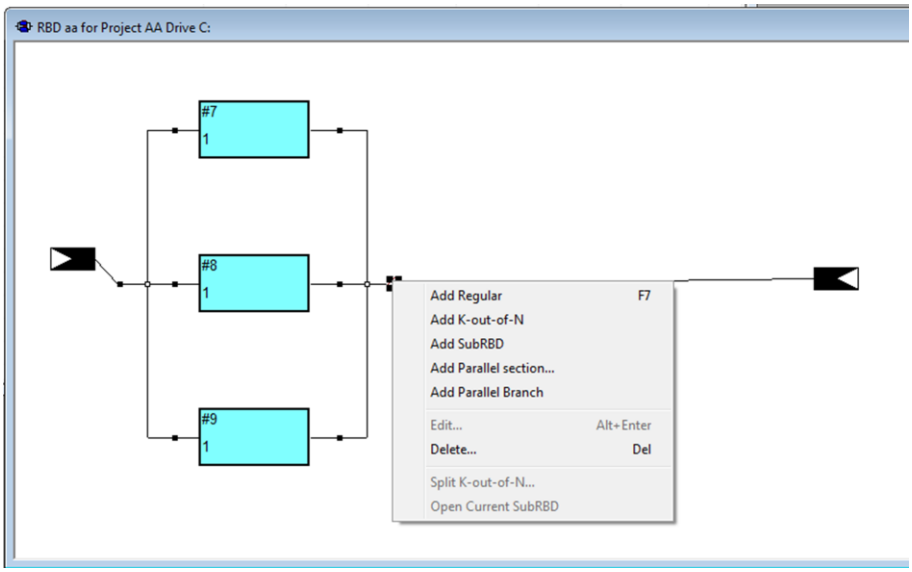
3.2.2 Rozhraní

Uvítací obrazovka ihned po spuštění nabídne výběr modulu, který budeme využívat. Pro účel bakalářské práce byl vybrán modul RBD.

Poté bylo potřeba vytvořit nový projekt a zadat název schématu. Orientace v tomto programu není úplně intuitivní, ale po čase v něm dá zorientovat. Po vytvoření schématu naskočí grafické plátno se vstupem a výstupem.

Každý blok zde má svůj uzel a při jeho vybrání lze vložit paralelní zapojení, zapojení M z N nebo další prvek. Způsob vkládání schématu zapojení lze vidět na obrázku Obrázek 15: RAM Commander způsob vkládání komponent.

Dvojitým poklepáním na element lze upravovat jeho vlastnosti. Výpočty můžeme vyvolat z hlavního menu pod položkou Calculation.



Obrázek 15: RAM Commander způsob vkládání komponent

3.2.3 Krizová zapojení

Všechna krizová zapojení popsaná v kapitole 1.4.4, jsou v tomto softwaru eliminována způsobem zadávání.

Zapojení zobrazená na Obrázek 7 a Obrázek 9: Zacyklení nejdou zadat díky přidávání jednotlivých způsobů zapojení hierarchicky k uzlům komponent nebo zapojení.

Zapojení zobrazené na Obrázek 8: Zapojení s dvěma výstupy je odstraněno tím, že máme daný pevný vstup i výstup.

3.2.4 Závěr a zhodnocení

Opět se jedná o komplexní řešení pro různé způsoby vyhodnocování rizik, které je ovšem mnohem jednodušší než jeho konkurence. Způsob zadávání komponent je pro výukové účely ideální a nedovolí uživateli spoustu chyb, které by mohl vytvořit.

Větší systémy by se do tohoto programu pravděpodobně zanášely obtížněji.

Mezi velké zápory této aplikace patří absence české lokalizace a cena.

3.3 Isograph

Isograph nabízí na svých oficiálních stránkách samostatnou aplikaci s názvem Reliability Workbench. Jak o ní sám Isograph píše, je to jejich vlajková loď nástrojů pro spolehlivost, bezpečnost a udržovatelnost. Software se neustále vyvíjí od roku 1980. Měl by být snadno ovladatelný a mělo by se jednat o nástroj především pro profesionály v této oblasti. [7]

3.3.1 Stažení a instalace

Na stránkách Isographu lze nalézt odkaz pro stažení zkušební verze aplikace. Po vyplnění formuláře s údaji o tom, na jaké účely bude využívána a pracovní pozice uchazeče se přibližně 120MB velký instalační soubor začne stahovat.

Pro instalaci softwaru je vyžadováno heslo. Po emailové korespondenci jsem obdržel telefonát od jejich zaměstnance, že s naší univerzitou nespolupracují a tudíž žádné přístupové údaje (ani k zkušební verzi) nedostanu.

3.3.2 Závěr a zhodnocení

Ze zjištěných údajů ze stránek výrobce lze zkonstatovat pouze to, že se jedná o placený software.

3.4 ITEM Toolkit

Jak lze nalézt na oficiálních stránkách aplikace, ITEM Toolkit je sada nástrojů komplexní predikce a analytických modulů v jednom integrovaném prostředí. Toolkit je integrované prostředí, využívající objektově orientované architektury, která přináší přesnost, flexibilitu a jednoduchost použití. To nabízí funkce, které poskytují konzistentní formát pro všechny analýzy. To je umožněno přenosem mezi moduly.

Využívají celosvětově uznávané standardy a metodiky k analýze komponent, systémů a projektů. [8]

3.4.1 Stažení a instalace

Na stránkách itemsoftu nalezneme odkaz ke stažení zkušební verze. Ten je zpřístupněn po zadání osobních informací do jejich formuláře. Poté je stažen přibližně 50MB velký instalační soubor.

Při instalaci je vyžadováno heslo, které je zasláno na požádání emailovou korespondencí. Samotná instalace byla svižná a probíhala standardně.

3.4.2 Rozhraní

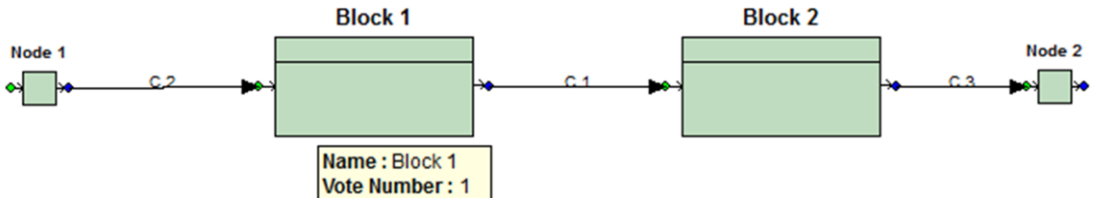
Po spuštění se zobrazí hlavní obrazovka rozdělená do tří částí: výčet projektů, výčet prvků v systému a zobrazení samotného systému. Je třeba zvolit z menu vytvoření nového projektu, do kterého se musí přidat RBD systém.

Samotný RBD systém je rozdělen do několika záložek. Těmi nejpodstatnějšími jsou: Grid, RBD, Chart a Result.

V záložce GRID lze vidět všechny prvky a spojnice. Prvkům zde můžeme přidávat jejich základní parametry.

Záložka RBD slouží pro zakreslení systému. Prvky a spojnice lze vkládat libovolně a není zde žádné omezení. Příklad sériového zapojení lze vidět na obrázku Obrázek 16: ITEM Toolkit sériové zapojení.

Pro zobrazení výsledků v záložkách chart a result musíme data nejdříve ověřit v menu analýza a poté nechat nechat vykonat výpočet.



Obrázek 16: ITEM Toolkit sériové zapojení

3.4.3 Krizová zapojení

Neidentifikovatelné schéma zapojení, zobrazené na Obrázek 7: Nerozpoznatelný typ zapojení, v programu vytvořit lze. Projde všemi validacemi a vypočítá požadované hodnoty.

Příklad zobrazený na obrázku Obrázek 8: Zapojení s dvěma výstupy sice zapojit lze také, ale při kontrole dat je zobrazena chyba a bez opravy Vás program nepustí dále.

V posledním příkladu Obrázek 9: Zacyklení, schéma kontrolou projde. Chybová hláška se zobrazí až při vykonání výpočtů nad danými daty.

3.4.4 Shrnutí a závěr

Toto komplexní řešení vyniká přehledností a po přečtení částí manuálu se v něm velice dobře pracuje. Program hlídá všechny kritické chyby na které jsem se zaměřil, jen u zacyklení by mohl zobrazovat cestu, ve které k němu dochází.

Mezi největší nevýhody patří, jako u ostatních softwarů na trhu, cena a absence české lokalizace.

4 Aplikace

Tato kapitola se bude zabývat podrobným popisem celé aplikace, ať už se jedná o aplikační logiku nebo popis důležitých metod v jednotlivých třídách.

4.1 Logika aplikace

Jedná se o celkový popis jak aplikace funguje na základě vstupu uživatele a rozhodnutí programu. Nákres této logiky můžete vidět na Obrázek 17: Aplikační logika.

Prvním krokem musí být vždy samotné spuštění aplikace. Vzhledem k tomu, že se jedná o aplikaci počítající pouze s modelem RBD, tak se vyhneme úvodní obrazovce s výběrem parametrů, jako u porovnávaných softwarů v kapitole 3.

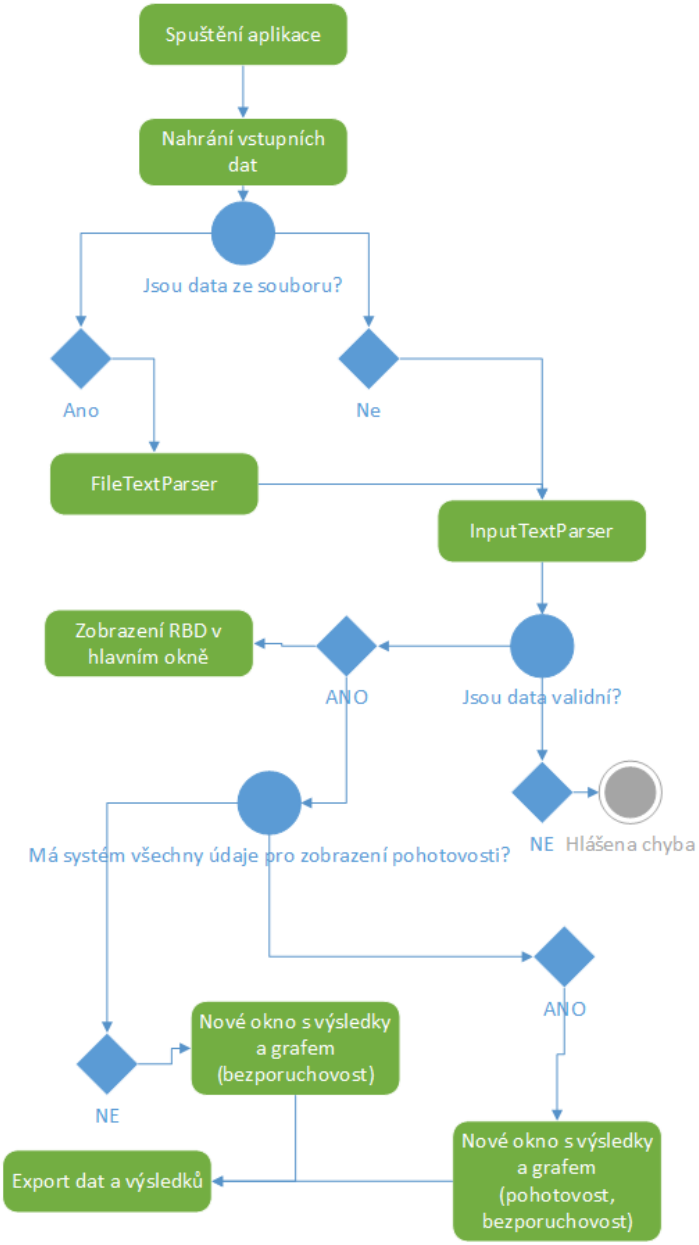
Dalším krokem je vložení vstupních dat. Zde je na uživateli, zda vybere data ze souboru či začne psát nový vstupní kód. V případě, že se uživatel rozhodne pracovat s textovým souborem, tak musí mít koncovku .rbd a musí být přístupný z disku s právy pro čtení. Po úspěšném načtení se otevře další okno aplikace se zobrazenými vstupními daty a možností editace. V případě psaní nového kódu se pouze otevře nové okno s možností zápisu.

Po dopsání (nebo kontrole) kódu, nechá uživatel data validovat programem. Pokud data validní nejsou, zobrazí se chybové hlášení obsahující typ chyby a číslo řádku na kterém se vyskytuje.

Když je kód validní, zobrazí se v hlavním okně schéma zapojení systému. Současně s tím program kontroluje, zda-li jsou u všech zadaných komponent dostupné i informace o pohotovosti. Pokud ano, zobrazí se průběh parametrů bezporuchovosti a pohotovosti v jednom grafu. V opačném případě se zobrazí pouze průběh bezporuchovosti. V obou případech je ve stejném okně dostupná i možnost zobrazit pravděpodobnost bezporuchového provozu v jednotlivých časech. U funkce okamžité pohotovosti není funkce naprogramována, z důvodu rychlého ustálení.

Průběh výpočtů bezporuchovosti (pohotovosti), probíhá postupným voláním metody *calculate* od vrchních komponent po ty nejspodnější. Když se program dostane až k nim, vrací postupně číselné hodnoty až do počátečního systému.

Posledním krokem je export dat a výsledků. Uživatel je při této volbě vyzván, aby vybral složku pro uložení. Aby byl příkaz úspěšný, musí složka obsahovat práva pro zápis pro daného uživatele.



Obrázek 17: Aplikační logika

4.2 Rozbor důležitých metod

Tato kapitola slouží k vysvětlení účelu a funkce vybraných metod mé aplikace. Vzhledem k obsáhlosti zde uvedených kódů jsem se rozhodl tuto kapitolu přesunout do přílohy. Lze si jí prohlédnout v Příloha C.

4.3 Testy

Součástí každé vyvíjené aplikace musí být testy. Programovací jazyk C# nabízí řešení ve formě zvané Unity Tests. Jde o otestování dílčích částí aplikace, pro ověření správných výsledků, stability a časového vyhodnocení. V mé aplikaci jde především o kontrolu správnosti výpočtů bezporuchovosti.

Bylo třeba otestovat co možná největší škálu případů, které mohou nastat. Tyto zapojení byli testovány pro deset a sto hodin. Z důvodu lepšího zadávání byli výsledné hodnoty zaokrouhleny na čtyři desetinná místa.

Všechny testy prošly s úspěchem a průměrná rychlost (počítána z 20 měření) je 0,003 s.

Vstupní data, na kterých testy proběhly, si můžete prohlédnout v Příloze A. Pro lepší orientaci v testovaných systémech si můžete prohlédnout jejich strukturu v Příloze B.

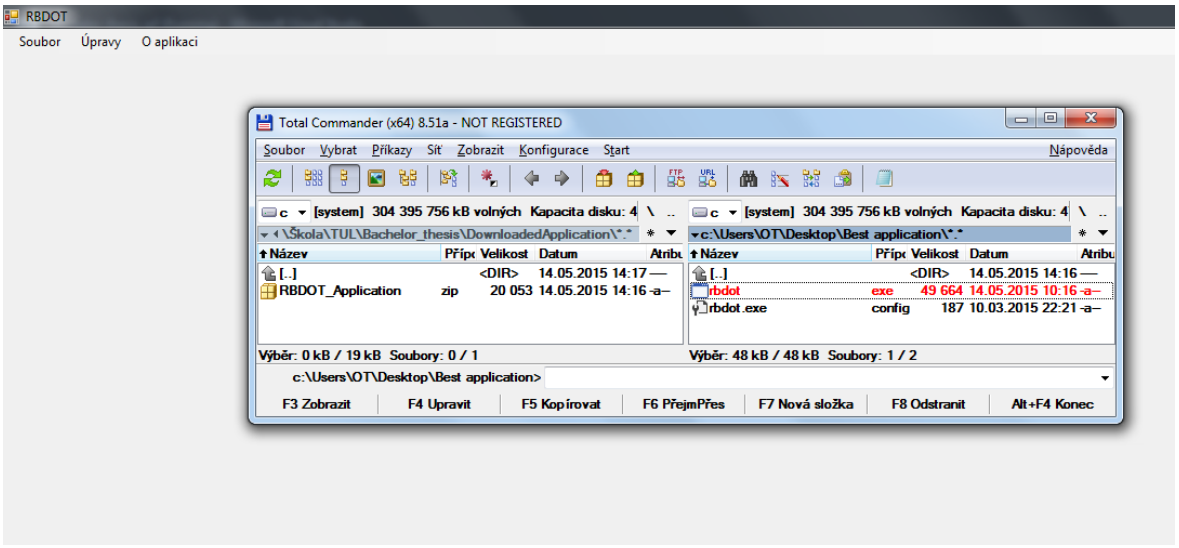
5 Manuál

5.1 Instalace a spuštění

Pro bezproblémovou práci aplikace rozbalte konfigurační conf a spustitelný exe soubor na místo disku s povoleným zápisem a čtením.

Pro běh softwaru musíte mít na vašem operačním systému nainstalovaný .NET Framework verze 4.5, který je volně ke stažení na oficiálních stránkách Microsoftu (https://www.microsoft.com/cs-cz/download/details.aspx?id=30653).

Aplikace se spouští přes soubor rbdot.exe. Po úspěšném spuštění by se vám měla zobrazit uvítací obrazovka.



Obrázek 18: Uvítací obrazovka aplikace

5.2 Vložení dat z aplikace

Pro tento způsob zadávání komponent je nutné z menu vybrat *Úpravy→Vložit textový vstup*. Zobrazí se okno *Data systému*. Do něj pak postupně po řádcích vkládáme požadována data podle postupu popsaném v kapitole 5.4.

Na posledním řádku je vždy hlavní část systému. Komponenty, které nebudou obsaženy v něm nebo v některém jeho podsystému nebudou vůbec zaznamenány.

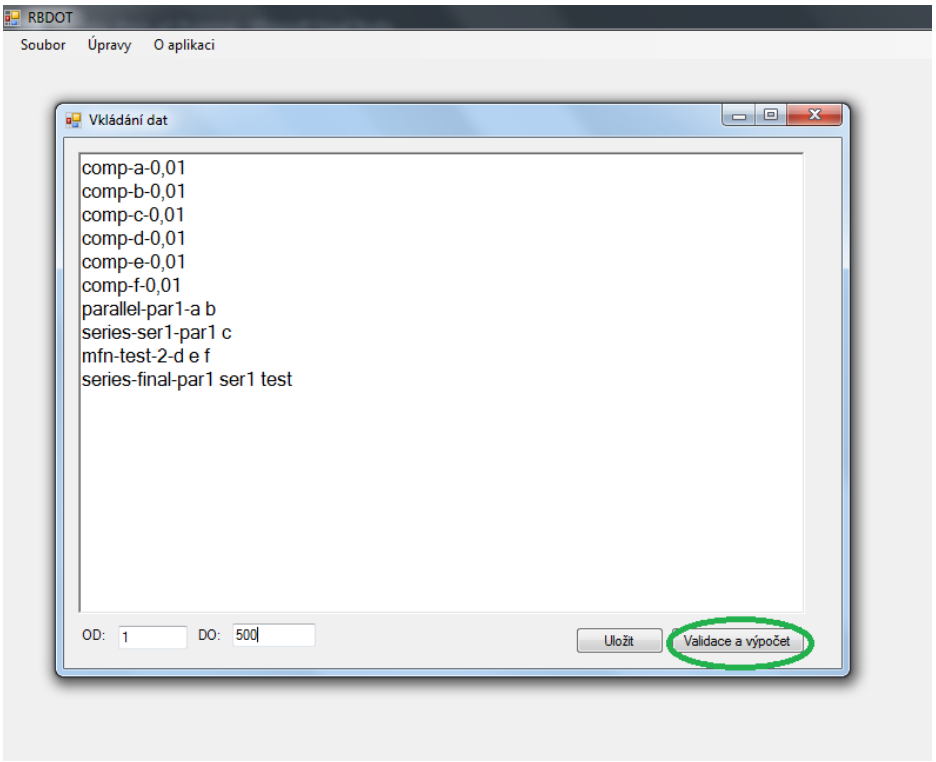
Příklad:

Comp-a-0,01-0,1
Comp-b-0,01-0,1
Series-serial-a a a a

Tento kód je validní a systém se vykreslí, ovšem zadefinovaná komponenta *b* není obsažena v systému *serial*, ani v žádném jeho podsystému, a proto nebude nikde zařazena.

V levém dolní rohu nalezneme kolonky *OD* a *DO*. Jedná se o nastavení intervalu, ve kterém chceme poruchovost (pohotovost) daného systému zobrazit. Časy jsou uvedeny v hodinách a krokem je vždy jedna hodina. Tyto údaje musí být vyplněny!

Pro zobrazení výsledků je nutné stisknout tlačítko *Zobrazit*, umístěné v pravém dolním rohu. Po úspěšné validaci se v hlavním okně vykreslí zadaný systém a zobrazí se okno *výsledky*.

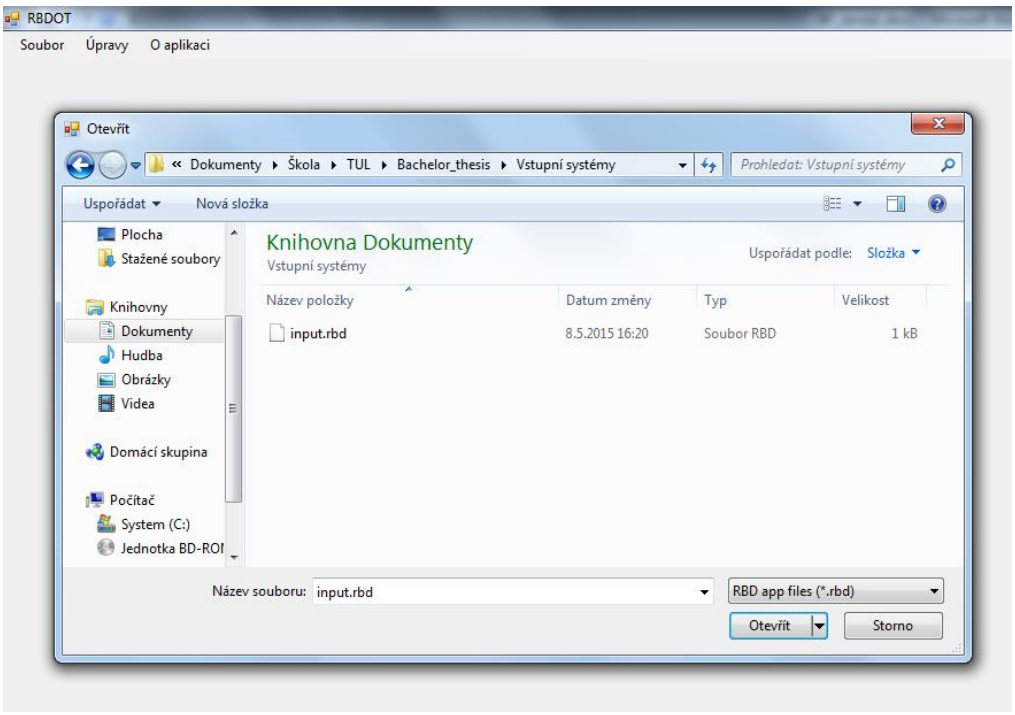


Obrázek 19: Vstupní data a validace systému

5.3 Vložení dat ze souboru

Pokud chcete zvolit tento způsob zadávání komponent, je nutné vybrat *Úpravy→ Vložit data ze souboru*. Otevře se okno, ve kterém můžete vybrat jakýkoliv *.rbd soubor z disku nebo jiného datového média.

Po úspěšném otevření souboru je zbylý postup stejný jako v kapitole 5.2.



Obrázek 20: Vstupní data ze souboru

5.4 Formát vstupních dat

Vstupní data systému lze zadat ručně v aplikaci nebo importem ze souboru *.rbd. Struktura obsažených dat zůstává v obou případech stejná. Jednotlivé systémy, subsystémy a komponenty vkládáme postupně po řádcích.

1. **Komponenta s uvedenou intenzitou poruch a konstatní intenzitou oprav**

   comp-nazev_komponenty-λ-μ

2. **Komponenta s uvedenou intenzitou poruch**

   comp-nazev_komponenty-λ

3. **Sériové zapojení se třemi prvky**

   series-nazev_serioveho_zapojeni_prvek1 prvek2 prvek3

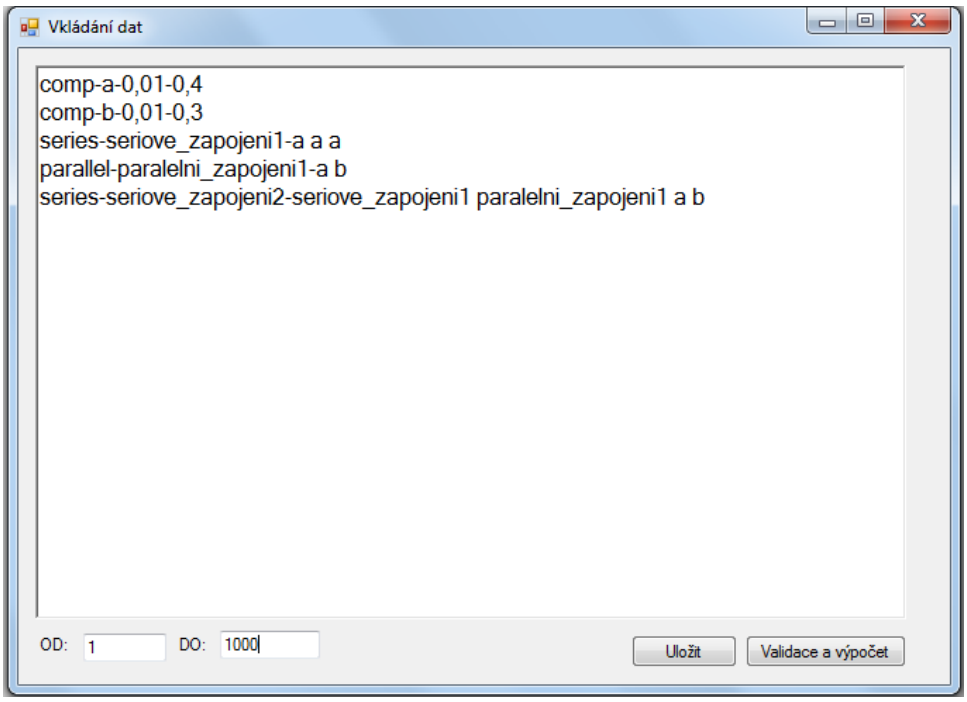
4. **Paralelní zapojení se dvěma prvky a sériovým systémem**

   parallel-nazev_paralelniho_zapojeni_prvek1 prvek2 system3

5. **M z N zapojení se čtyřmi prvky**

   mfn-nazev_M_z_N_zapojeni-hodnota_M-prvek1 prvek2 prvek3 prvek4

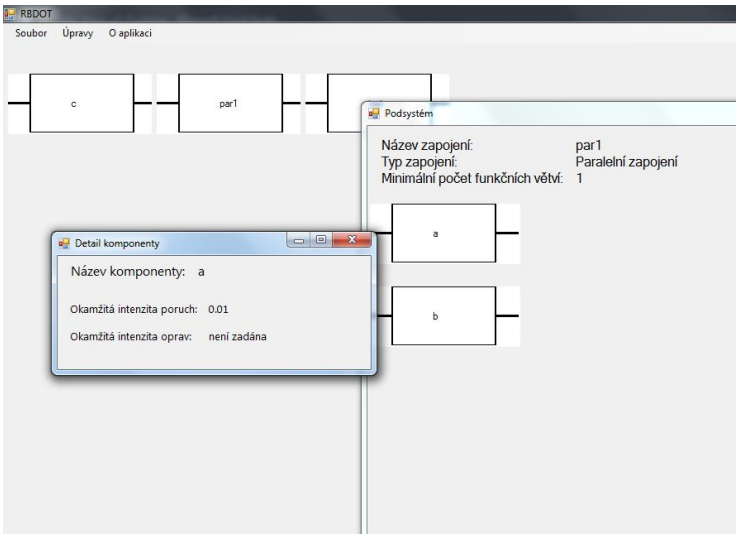
Příklad jak by mohlo vypadat zadání jednoduchého složeného systému si lze prohlédnout na obrázku Obrázek 21: Příklad zapsání systému.



Obrázek 21: Příklad zapsání systému

5.5 Zobrazení podsystému

Pokud Váš systém obsahuje kromě komponent i podsystémy, je možné je zobrazit. Ve vykresleném systému dvakrát poklepeme na vybraný podsystém, který se otevře v novém okně. Tento postup lze opakovat až do nejmenších komponent. Při poklepání na komponentu se otevře nové okno s detailními informacemi o ní.



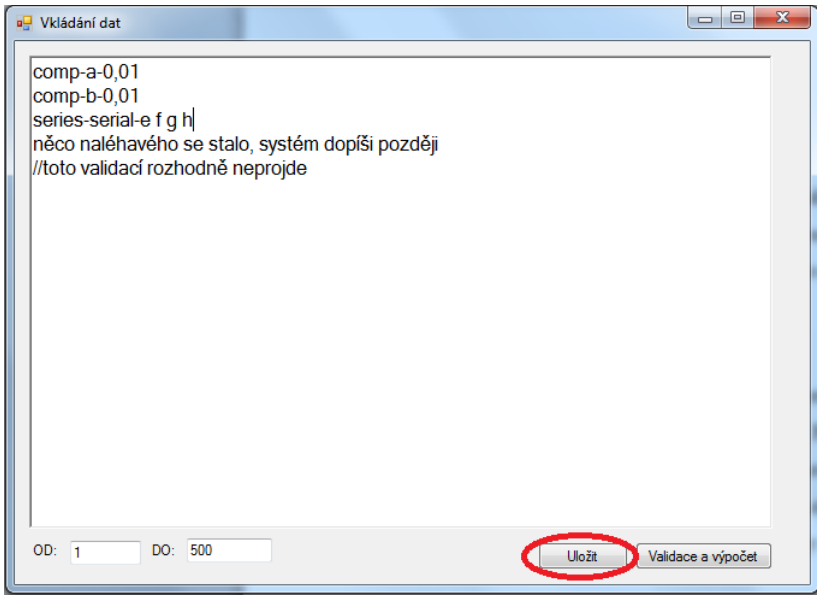
Obrázek 22: Rozklikávání komponent a podsystémů

5.6 Export dat

5.6.1 Rozpracovaný systém

Pro uložení Vámi rozpracovaného a nevalidovaného systému, je nutné stisknout tlačítko

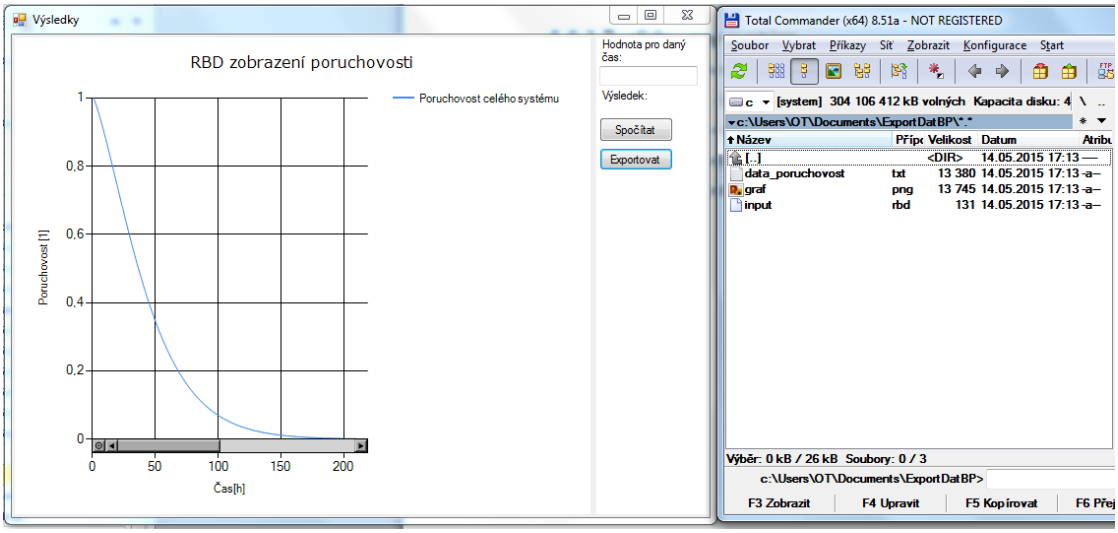
uložit v okně určeném pro zadávání dat (více v kapitole 5.2). Dále již budete vyzváni k výběru složky, kam chcete data umístit. Místo na datovém úložišti musí mít povolen zápis dat pro daného uživatele.



Obrázek 23: Uložení rozpracovaného systému

5.6.2 Validovaný systém

Pro uložení již validovaného systému vyberte tlačítko *Exportovat data*. Otevře se okno s výběrem složky, kam chcete data umístit. Po ověření, že místo má povolen zápis pro daného uživatele bude exportováno několik souborů. Prvním z nich je soubor *.rbd, který obsahuje vstupní data systému. Druhý z nich obsahuje graf ve formátu *.png. Poslední dva jsou textové soubory jenž obsahují výsledky v daném rozsahu.



Obrázek 24: Export výsledků

Závěr

Jedním ze základních cílů této bakalářské práce bylo seznámit se s metodou RBD. Jedná se o poměrně obsáhlou problematiku, a proto se v teoretické části shrnují její základní části. Všechny druhy zapojení jsem pro lepší pochopení uvedl na příkladech.

Kromě samotného vytvoření aplikace bylo cílem porovnat již existující komerční programy. Byly vybrány čtyři firmy, které se metodikou RBD zabývají a ke třem z nich se podařilo získat přístup. Mezi největší nevýhody, které mají všechny společné, patří absence české lokalizace a vysoká cena.

Dalším krokem byl návrh aplikace. Základní rozhraní aplikace je implementováno v programovacím jazyce C#. Nabízela se eventualita, že matematické výpočty budou probíhat v externím skriptu. Po porovnání rychlostí matematických výpočtů v jiných jazycích bylo zjištěno, že rozdíly jsou tak nepatrné, že v jazyce C# bude napsaná celá aplikace.

Při důležitém rozhodnutí, zda vytvořit textový nebo grafický způsob zadávání systému, zvítězil textový. Grafický vstup sice umožní uživateli lepší průběžnou orientaci v systému, ale v textovém se lépe odchytávají špatná vstupní data a nedovolí chyby typu zacyklení, nerozpoznatelný typ zapojení, atd. Grafické rozhraní se v aplikaci po validaci systému pouze zobrazí.

Jednou z nejdůležitějších částí návrhu aplikace bylo vytvoření schématu tříd, převážně těch datových. Třídy kopírují strukturu systému a jsou vnořeny do sebe. Postupným voláním od nejvrchnější komponenty směrem dolů, se systém dostane až k nejmenší jednotce, od které naopak zespoda nahoru počítá výsledky v jednotlivých časech.

Export dat je uživateli umožněn dvojím způsobem. První z nich ukládá rozpracovaný (ještě nevalidovaný) vstup do systému ve formátu textového souboru *.rbd. Druhý z nich již exportuje výsledky a to ve formě čtyř souborů: data bezporuchovosti v daném rozsahu, data pohotovosti v daném rozsahu, graf hodnot ve formátu png a datový soubor pro znovuotevření systému v programu.

Posledním krokem této práce bylo vytvoření manuálu pro ovládání softwaru. Ten si můžete prohlédnout v bakalářské práci nebo spustit přímo z aplikace.

Literatura

[1] MYKISKA, Antonín. *Bezpečnost a spolehlivost technických systémů*. Vyd. 2. přeprac. V Praze: Vydavatelství ČVUT, 2004, 206 s. ISBN 80-01-02868-2.

[2] ČSN EN 61078. *Techniky analýzy spolehlivosti – Blokový diagram bezporuchovosti a Booleovské metody*. 2007. Praha: Český normalizační institut.

[3] FUCHS, Pavel, David VALIŠ, Josef CHUDOBA, Jan KAMENICKÝ a Jaroslav ZAJÍČEK. 2006. *Řízení spolehlivosti*. Liberec. Skripta. Technická univerzita v Liberci.

[4] KAMENICKÝ, Jan, Jaroslav ZAJÍČEK a Pavel FUCHS. 2015. *Bezporuchovost a pohotovost*. Praha: Česká společnost pro jakost.

[5] *Technology Solutions for Ongoing Product & Service Advantage | PTC* [online]. 2009. [cit. 2014-11-27]. Dostupné z: http://www.ptc.com/

[6] *RAMS (Reliability, Availability, Maintainability and Safety) Software* [online]. 2015. [cit. 2014-11-29]. Dostupné z: http://aldservice.com/en/reliability-products/rams-software.html

[7] *RBD Analysis - Isograph* [online]. [cit. 2015-03-11]. Dostupné z: http://www.isograph.com/software/reliability-workbench/rbd-analysis/

[8] *Fully Integrated Reliability Analysis and Safety Software - ITEM ToolKit* [online]. 2015. [2015-03-11]. Dostupné z: http://www.itemsoft.com/item_toolkit.html

Přílohy

Příloha A- Vstupní data testovacích prvků

a) Samostatná komponenta

```
comp-A-0,01
```

b) Sériové zapojení

```
comp-A-0,01
series-final-A A A
```

c) Paralelní zapojení

```
comp-A-0,01
parallel-par-A A
```

d) Smíšené zapojení I

```
comp-A-0,01
comp-B-0,02
series-ser-A B
parallel-par-ser A
```

e) Smíšené zapojení II

```
comp-A-0,01
comp-B-0,02
comp-C-0,03
comp-D-0,04
parallel-par1-B C
series-ser1-par1 A
parallel-par2-ser1 D
```

f) M z N typ paralelního zapojení

```
comp-A-0,01
comp-B-0,02
comp-C-0,03
mfn-mfn-1-a b c
```

g) M z N dva ze tří zapojení

```
comp-A-0,01
comp-B-0,02
comp-C-0,03
mfn-mfn-2-a b c
```

h) M z N stejné prvky zapojení

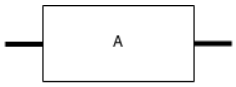
```
comp-A-0,01
mfn-mfn-2-a a a a
```

i) Smíšené zapojení III

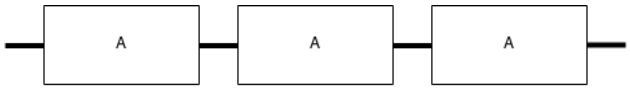
```
comp-A-0,01
comp-B-0,02
comp-C-0,03
comp-D-0,04
series-ser1-A A
series-ser2-A B
parallel-par1-ser2 C
parallel-par2-C C
mfn-mfn-2-par1 B par2
series-ser3-A A mfn
parallel-par4-ser3 D
```

Příloha B- Schémata testovaných zapojení

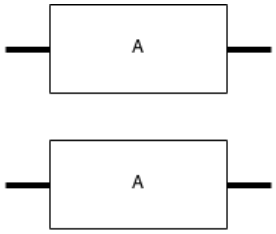
a) Samostatná komponenta



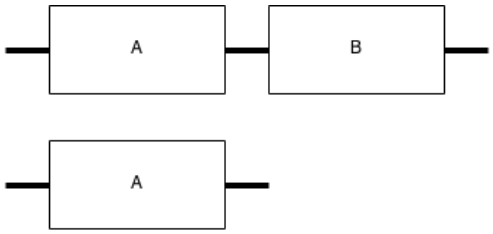
b) Sériové zapojení



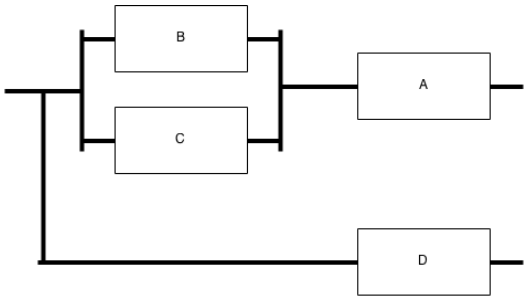
c) Paralelní zapojení



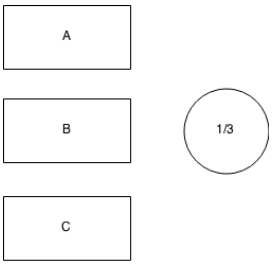
d) Smíšené zapojení I



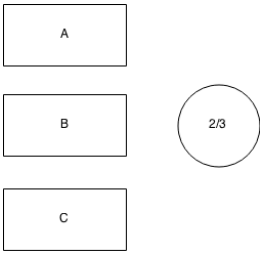
e) Smíšené zapojení II



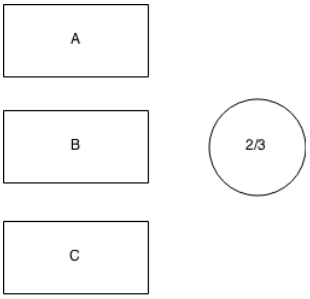
f) M z N typ paralelního zapojení



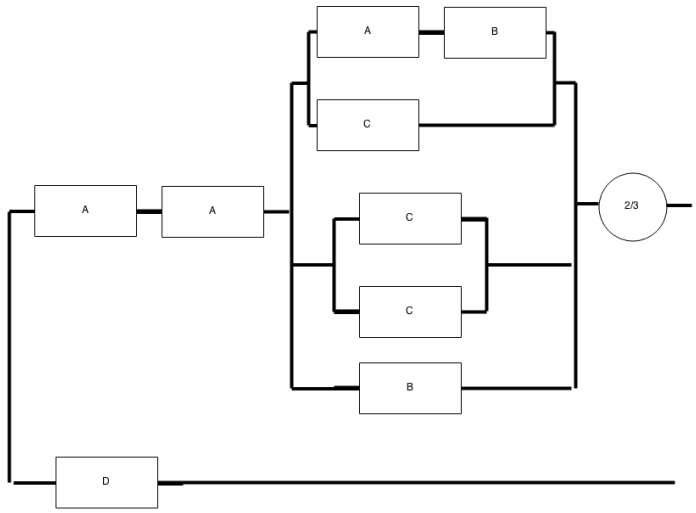
g) M z N dva ze tří zapojení



h) M z N stejné prvky zapojení



i) Smíšené zapojení III



Příloha C- rozbor důležitých metod

CalculateFailure

Jedná se o metodu, kterou obsahují všechny datové třídy popsané v kapitole 2.2.1. Její rozhraní je implementováno v interfacu *IElement*. Vstupní a výstupní proměnné popsané v kapitole 2.2.1 při návrhu systému, byly dodrženy.

Nyní se podíváme na její zpracování ve třídě Element.

```
public Dictionary<int,double> CalculateFailure(int[] times, bool
calculateRepairibility)
   {try
      {Dictionary<int, double> results= new Dictionary<int, double>();
                  foreach (var time in times)
             {if (calculateRepairibility == false)
                  {var result = Math.Pow(Math.E, -time*FailureRate);
                      results[time] = result;
                  }
                  else
                  {
                      var result = (AvaibilityRate/(AvaibilityRate +
FailureRate)) + (FailureRate/(FailureRate + AvaibilityRate))*
                                      Math.Pow(Math.E,(-(FailureRate +
AvaibilityRate)*time));
                      results[time] = result;
                  }
               }
               return results;
           }
           catch (Exception)
           {

               throw new Exception("Při výpočtu prvku "+ this.Name+ " nastala
chyba");
           }

       }
```

Logika této metody spočívá na rozhodnutí, jestli se počítají parametry pohotovosti nebo bezporuchovosti. Samotné výpočty (*var result*) spočívají pouze v přepsání vzorečků použitých v kapitole 1.4 do programové podoby. Výsledky jednotlivých časů se vždy po dokončení ukládají to návratového slovníku.

Další aplikaci této metody nalezneme ve třídě *ParallelConnection*.

```
public Dictionary<int, double> CalculateFailure(int[] times, bool
calculateAvaibility)
       {
```

```
            var results = new Dictionary<int, double>();
            var branchValues = Branches.Select(branch =>
    branch.CalculateFailure(times,
    calculateAvaibility)).ToList();


            var tmp = new Dictionary<int, double>();
            foreach (var value in
        branchValues.SelectMany(branch => branch))
            {
                if (!tmp.ContainsKey(value.Key))
                {
                    tmp.Add(value.Key, (1 - value.Value));
                }
                else
                {
                    tmp[value.Key] *= (1 - value.Value);
                }
            }
            return tmp.ToDictionary(value => value.Key,
    value => 1 - tmp[value.Key]);
        }
```

Vzhledem k tomu, že jednotlivé větve jsou zde reprezentovány jako pole sériových zapojení, tak je nejprve nutné spočítat hodnoty větví v daných časech. Toho je docíleno klasickým foreach cyklem. Předpokládá se, že existuje alespoň jedna větev, protože jinak by data nemohla projít validací přes *parser*.

Podle vzorce pro paralelní výpočet, uvedený v kapitole 1.4.2, se rozdělil výpočet do 2 částí. V té první se vždy hodnota větve pro daný čas odečte od jedné a násobí se s již takto upravenými hodnotami v daném čase. V druhém kroku se tento mezivýsledek odečte od jedné a tím získáme daného paralelního systému (podsystému) v čase.

Tuto metodu nalezneme i v třídě *SerialConnection*.

```
public Dictionary<int, double> CalculateFailure(int[] times, bool
calculateAvaibility)
        {
            var results = new Dictionary<int, double>();
            foreach (var value in Elements.Select(element =>
        element.CalculateFailure(times,
    calculateAvaibility)).SelectMany(tableValues => tableValues))
            {
                double prod;
                if (results.TryGetValue(value.Key, out prod))
                {
                    results[value.Key] = prod*value.Value;
                }
                else
```

```
                {
                    results.Add(value.Key, value.Value);
                }
            }
            foreach (var value in ParalelsConnections.Select(parallel =>
       parallel.CalculateFailure(times,
   calculateAvaibility)).SelectMany(resultParallel => resultParallel))
            {
                double prod;
                if (results.TryGetValue(value.Key, out prod))
                {
                    results[value.Key] = value.Value*results[value.Key];
                }
                else
                {
                    results.Add(value.Key, value.Value);
                }
            }
            foreach (var value in KfnConnections.Select(mfn =>
       mfn.CalculateFailure(times,
   calculateAvaibility)).SelectMany(resultMfn => resultMfn))
            {
                double prod;
                if (results.TryGetValue(value.Key, out prod))
                {
                    results[value.Key] = value.Value*results[value.Key];
                }
                else
                {
                    results.Add(value.Key, value.Value);
                }
            }
            return results;
        }
```

Jak je patrné z vzorce pro sériové zapojení, popsané v kapitole 1.4.1, tak k dosažení výsledků v čase, je nutné mezi sebou násobit všechny komponenty. Tato část kódu ukazuje, jakým způsobem je toho docíleno.

V třídě *KFNConnection* (M z N zapojení) vypadá tato metoda následovně.

```
public Dictionary<int, double> CalculateFailure(int[] times, bool
calculateAvaibility)
        {
            var branchInTime= new Dictionary<int, List<double>>();
            var tmp= Branches.Select(branch => branch.CalculateFailure(times,
       calculateAvaibility)).ToList();
            foreach (var resultInTime in tmp.SelectMany(branch => branch))
```

```
            {
                if (branchInTime.ContainsKey(resultInTime.Key))
                {
                    branchInTime[resultInTime.Key].Add(resultInTime.Value);
                }
                else
                {
                    var collection = new List<double> {resultInTime.Value};
                    branchInTime.Add(resultInTime.Key, collection);
                }
            }
            var areSame = true;
            foreach (var value in branchInTime)
            {
                var hashset = new HashSet<double>();
                foreach (var oneValue in value.Value)
                {
                    hashset.Add(oneValue);
                }
                if (hashset.Count <= 1) continue;
                areSame = false;
                break;
            }
            var result= new Dictionary<int, double>();
            if (M == 1)
            {
                result = CalculateParalleFailure(times, calculateAvaibility);
            }
            else if (areSame)
            {
                result= CalculateEqualBlocks(times, calculateAvaibility);
            }
            else if (M == 2 && Branches.Count == 3)
            {
                result = CalculateTwoFromThree(times, calculateAvaibility);
            }
            else
            {
                result = CalculateOther(times, calculateAvaibility);
            }
            return result;
        }
```

Vzhledem ke všem možnostem tohoto zapojení (popsaných v kapitole 1.4.3), se zvolil trochu odlišný způsob oproti ostatním třídám.

V prvním kroku se spočítají hodnoty jednotlivých větví, ale v druhém je ukládáme do slovníku, kde je klíčem čas a hodnotou pole typu *double*. Do tohoto vnořeného pole postupně ukládáme výsledky všech větví v jednotlivých časech. Toto bylo zvoleno záměrně, abych v dalším kroku mohl zkontrolovat, zda se jedná o stejné komponenty či nikoliv.

Poslední krok obsahuje všechny 4 varianty, které nám mohou nastat.

První z nich je případ, kdy M se rovná jedné. V takovém případě se jedná o prostě paralelní zapojení, jehož kód byl vysvětlen výše.

Druhým případem je, že mají všechny prvky stejnou hodnotu (takové zapojení bývá nejčastější). V tomto případě vypadá kód takto:

```
private Dictionary<int, double> CalculateEqualBlocks(int[] times, bool
calculateRepairibility)
        {
            var branch = Branches.First();


             var   valuesForTime = branch.CalculateFailure(times,
      calculateRepairibility);


            var result = new Dictionary<int, double>();
            foreach (var time in valuesForTime)
            {
                for (var i = 0; i < Branches.Count - M + 1; i++)
                {
                    if (!result.ContainsKey(time.Key))
                    {
                        result.Add(time.Key,
                            CalculateBinomialCoefficient(Branches.Count,
i)*Math.Pow(time.Value, (Branches.Count - i))*
                            Math.Pow((1 - time.Value), i));
                    }
                    else
                    {
                        result[time.Key] +=
                    CalculateBinomialCoefficient(Branches.Count,
             i)*Math.Pow(time.Value, (Branches.Count - i))*
                           Math.Pow((1 - time.Value), i);
                    }
                }
            }
           return result;
        }
```

Jedná se o vzorec uvedený v kapitole 1.4.3, převedený do jazyka C#.

Kód pro výpočet poruchovosti při zapojení M z N, kde M= 2 a N= 3, je opět pouze přepis vzorce do syntaxe programovacího jazyka a vypadá takto:

```
private Dictionary<int, double> CalculateTwoFromThree(int[] times, bool
calculateReparibility)
        {
            var branchInTime = new Dictionary<int, List<double>>();
            var resultTimes= new Dictionary<int, double>();
            var tmp = Branches.Select(branch => branch.CalculateFailure(times,
      calculateReparibility)).ToList();
            foreach (var resultInTime in tmp.SelectMany(branch => branch))
            {
                if (branchInTime.ContainsKey(resultInTime.Key))
                {
                    branchInTime[resultInTime.Key].Add(resultInTime.Value);
                }
                else
                {
                    var collection = new List<double> { resultInTime.Value };
                    branchInTime.Add(resultInTime.Key, collection);
                }
            }
            foreach (var time in branchInTime)
            {
                var values = time.Value;
                var result = values[0]*values[1]*values[2] + (1 -
             values[0])*values[1]*values[2] +
                    values[0]*values[1]*(1 - values[2]) + values[0]*(1-
             values[1])*values[2];
                resultTimes.Add(time.Key, result);
            }
            return resultTimes;
        }
```

Naopak relativně odlišný je kód pro výpočet při jakémkoliv jiném zapojení, než je uvedeno výše. Jak bylo popsáno dříve, je nejprve nutné spočítat průměrné hodnoty jednotlivých větví a až poté dosadit do vzorce. V takovém případě jsem postupoval takto:

```
private Dictionary<int, double> CalculateOther(int[] times, bool
calculateRepairibility)
        {
            var branchInTime = new Dictionary<int, List<double>>();
            var tmp = Branches.Select(branch => branch.CalculateFailure(times,
      calculateRepairibility)).ToList();
            foreach (var resultInTime in tmp.SelectMany(branch => branch))
            {
                if (branchInTime.ContainsKey(resultInTime.Key))
                {
```

```
                branchInTime[resultInTime.Key].Add(resultInTime.Value);
            }
            else
            {
                var collection = new List<double> { resultInTime.Value };
                branchInTime.Add(resultInTime.Key, collection);
            }
        }
        var averageBranch = new Dictionary<int, double>();
        foreach (var value in branchInTime)
        {
            var average= value.Value.Average();
            averageBranch.Add(value.Key, average);
        }
        var result = new Dictionary<int, double>();
        foreach (var time in averageBranch)
        {
            for (var i = 0; i < Branches.Count - M; i++)
            {
                if (!result.ContainsKey(time.Key))
                {
                    result.Add(time.Key,
                        CalculateBinomialCoefficient(Branches.Count, i) *
                  Math.Pow(time.Value, (Branches.Count - i)) *
                         Math.Pow((1 - time.Value), i));
                }
                else
                {
                    result[time.Key] +=
                CalculateBinomialCoefficient(Branches.Count, i) *
          Math.Pow(time.Value, (Branches.Count - i)) *
                       Math.Pow((1 - time.Value), i);
                }
            }
        }
        return result;
    }
```

Parse

Jedná se o nejdůležitější metodu v třídě *TextInputParser*. Vstupní proměnná je textový řetězec, obsahující data pro vytvoření systému. V případě úspěšného rozčlenění kódu a úspěšné validaci dat, vrací vytvořený systém. Vzhledem ke komplexnosti kódu, ho budu rozebírat po částech.

První část vytvoří slovník s hodnotou *IElement* a klíčem *String*. *String* symbolizuje název komponenty. Dále je pak text rozdělen pomocí znaku '-'. Tento kód vypadá takto:

```
var elements = new Dictionary<string, IElement>();
            for (var i = 0; i <= inputText.Lines.Length-1; i++)
            {
                var text = inputText.Lines[i];
                text = text.ToLower();
                try
                {
                    var data = text.Split('-');
                    if (data.Length < 1 || data.Length > 4)
                    {
                        throw new SystemException("Nesprávně uvedená data na
                    řádku " + i);
                    }
              //další kód
          }
       //další kód
       }
```

Další částí je rozlišení o jakou komponentu se jedná. To je založeno na principu rozlišení první části rozdělených dat. Ve třídě jsou definované všechny dostupné názvy všech dostupných typů připojení, se kterými lze pracovat. Pokud textový řetězec neodpovídá ani jednomu z nich, je vytvořena výjimka.

Druhou částí rozdělených dat je název komponenty. Název komponenty může obsahovat jakékoliv znaky kromě pomlčky a mezery. Důležité je říct, že jméno systému není case-sensitive, protože je vždy převeden na malá písmena.

Další postup už záleží na typu komponenty. Pokud se jedná o *Element*, jsou dalšími daty λ a μ. Prvním argumentem je vždy λ a druhým (nepovinným) μ. Pokud se jedná některý z jiných systémů, je vždy další částí výčet komponent nebo subsystémů oddělený mezerou.

Pokud jsou úspěšně validovány všechny prvky systému, proběhne kontrola zda jsou zadány všechny potřebné údaje pro výpočet pohotovosti.

Na závěr této metody je vytvořena instance třídy *System*, která je předána zpět do hlavního formu.

Příloha D- Obsah přiloženého CD

1. Text bakalářské práce
    a. bakalarska_prace_2015_Oldrich_Taufer.pdf
2. Zdrojový kód programu
    a. aplikace pro PC (v programovacím jazyce C#)
3. Spouštěcí soubory programu (zkomprimované)
    a. rbdot.rar