VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY
a BIOMECHANIKY
INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

TYPOVÝ PŘÍKLAD VYUŽITÍ METOD MODEL-BASED DESIGN PRO VÝVOJ LETADLOVÝCH SOUSTAV

ILLUSTRATIVE EXAMPLE OF MODEL-BASED DESIGN FOR COMPLEX AIRCRAFT SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Radek Sopoušek
AUTHOR

VEDOUCÍ PRÁCE — Ing. Luděk Janák
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Radek Sopoušek**
Studijní program:	Aplikované vědy v inženýrství
Studijní obor:	Mechatronika
Vedoucí práce:	**Ing. Luděk Janák**
Akademický rok:	2020/21

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Typový příklad využití metod model-based design pro vývoj letadlových soustav

Stručná charakteristika problematiky úkolu:

Model-based design (MBD) je jedním z fenoménů současné praxe ve vývoji komplexního software pro řízení technických soustav-například v letectví, automotive, vývoji drážních vozidel a zabezpečení. V letectví je zavádění model-based design spojeno zejména s platností poradního oběžníku FAA AC.20-115 "Airborne Software Assurance", který "povoluje" použití metod model-based design. Cílem nasazení metod model-based design v letectví je zejména snížení počtu softwarových požadavků, snažší dokumentovatelnost vývoje, zjednodušení testovaní a možnost automatického generování kódu z modelů. Standardem, který popisuje životní cyklus software při využití metod MBD je RTCA DO-331. Pro vývoj na úrovni systému je to pak SAE ARP4754A. Tyto standardy ovšem postrádají názorné/typové příklady využití metod MBD a nastavení vývojového procesu. Pro široké praktické nasazení MBD je nutné tyto názorné příklady připravit a rozšířit do inženýrské praxe. Typový příklad by měl obsahovat minimálně popis vývojového procesu, systémové požadavky, případně softwarové požadavky, samotné modely, knihovnu komponent modelu. Příklad může být vystavěn na některém z prvků automatického systému řízení letu (CAS logika, autopilot monitory, autothrottle).

Cíle bakalářské práce:

1) Literární rešerše standardů, specifikací a požadavků souvisejících s nasazením metod "model-based design" pro vývoj letadlových soustav na úrovni systému a software.
2) Popis možných způsobů (příkladů procesů) vývoje letadlových soustav a software s využitím metod "model-based design".
3) Výběr a aplikace vybraného procesu vývoje systému a software (tj. příprava "metodiky"/"kuchařky" jak soustavu vyvíjet).
4) Ověření navrženého způsobu vývoje na ilustrativním příkladu.

Seznam doporučené literatury:

RTCA DO-331. Model-Based Development and Verification Supplement to DO-178C and DO-278A, RTCA, 2011.

RTCA DO-178C. Software Considerations in Airborne Systems and Equipment Certification, RTCA, 2011

SAE ARP 4754A. Guidelines for Development of Civil Aircraft and Systems, SAE, 2010.

SAE AIR 6110. Contiguous Aircraft/System Development Process Example, SAE, 2011.

FAA AC 20-115D, Airborne Software Development Assurance Using EUROCAE ED-12 and RTCA DO-178, FAA, 2017.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2020/21

V Brně, dne

L.S.

prof. Ing. Jindřich Petruška, CSc. — ředitel ústavu

doc. Ing. Jaroslav Katolický, Ph.D. — děkan fakulty

Abstrakt

Tato bakalářská práce se věnuje problematice vývoje leteckých soustav, zejména jejich softwaru, s využitím metod *„model-based design“* (*MBD*) při zachování doporučených postupů vytvořených leteckým průmyslem a certifikačními autoritami (zejména RTCA DO-178C/331). Obecné možnosti nasazení *MBD* dané současnými leteckými regulativy a standardy jsou popsány v teoretické části práce. Dále je v této práci popsáno, jak by mohl vypadat proces vývoje software pro dílčí část soustavy letadla (*„altitude hold mode“* softwaru autopilota“) s využitím metody *„model-based design"*. Částečně je řešen i přesah na úroveň vývoje soustavy (systému dle SAE ARP 4754A). Součástí práce je ilustrativní příklad, ve kterém byly specifikovány systémové požadavky, vývoj software prostřednictvím *„design modelu“* a zajištění trasovatelnosti mezi systémovými požadavky a design modelem.

Abstract

This bachelor's thesis is on the development of aviation systems, in particular their software, using *model-based design* (*MBD*) methods while maintaining recommended practices developed by the aviation industry and certification authorities (in particular RTCA DO-178C/331). The general *MBD* deployment options given by current aviation regulators and standards are described in the theoretical part of the work. Furthermore, this work describes what the process of developing software for a sub-segment of an aircraft system (altitude hold mode of autopilot software) could look like using the *model-based design* method. The overlap to the system development level (SAE ARP 4754A) is also partially addressed. The work includes an illustrative example in which system requirements were specified, software development through *"Design model*" and ensuring traceability between system requirements and model design.

Klíčová slova

letecká soustava, životní cyklus softwaru, vývoj softwaru, letecký software, model-based design, design model, MBD, SAE ARP 4754A, DO-178C, DO-331

Key words

aircraft system, software life cycle, software development, aircraft software, model-based design, design model, MBD, SAE ARP 4754A, DO-178C, DO-331

Bibliografická citace

SOPOUŠEK, R. Typový příklad využití metod model-based design pro vývoj letadlových soustav. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2021. 46 s. Vedoucí práce bakalářské práce Ing. Luděk Janák

Bibliographic entry

SOPOUŠEK, R. A specimen example of the use of model-based design method for the development of aircraft systems. Brno: Brno University of Technology, Faculty of mechanical Engineering, 2021. 46 s. Supervisor of bachelor's thesis: Ing. Luděk Janák

Prohlášení autora

Prohlašuji, že jsem svoji bakalářskou práci vypracoval samostatně s využitím informačních zdrojů, které jsou v práci citovány.

Brno 21. května 2021

……

Radek Sopoušek

Poděkování

Rád bych poděkoval Ing. Luďkovi Janákovi za odborné vedení a cenné rady při vypracovávání mé bakalářské práce.

Obsah

1 Úvod

Vývoj, zdokonalování a integrace vestavěných (embedded) systémů do elektromechanických soustav vede k vytvoření kvalitnějších, víceúčelových zařízení, jejichž vývoj oproti stejně funkčním, pouze elektromechanickým systémům, je jednodušší. Návrh zařízení ovládaných pomocí softwaru ale může být náročný na společnou spolupráci inženýrských týmů jednotlivých oborů, které mají různé přístupy a metody řešení.

Spojením mechanických systémů s elektronikou se zabývá specializovaný obor mechatronika. Tento pojem se začal používat v polovině sedmdesátých letech minulého století a jeho součástí je modelování systému. Jedním z přístupů, které mechatronika používá je metoda „model-based design“ (*MBD*). Úkolem tohoto přístupu je snížení počtu systémových a softwarových požadavků, snazší tvorba dokumentace, testování, verifikace a automatického generování zdrojového a objektového kódu.

Tato metoda je čím dál více preferována a umožňuje inženýrským týmům vytvářet kvalitní, sofistikované produkty s vysokou časovou i finanční efektivitou. *MBD* se uplatňuje pro vývoj systémů v mnoha oblastech, ve vývoji strojních zařízení, v automobilovém průmyslu[1], v železniční dopravě, v leteckém průmyslu, v kosmonautice ale také i ve vojenských systémech.

Díky *MBD* lze použít řadu nástrojů pro tvorbu softwaru. Například vývojové a simulační prostředí *Simulink*, který je součástí programu *MATLAB* od společnosti *MathWorks* (1). Pro uvedení nového systému na trh je třeba splnit podmínky certifikačních autorit a státních institucí. Pro snadnější získání certifikačního souhlasu vznikly napříč průmyslovými odvětvími standardy, předpisy a směrnice, které mají pomoci inženýrům nejenom zachovat formální stránku a splnit požadavky na systém a software při využití metodiky *MBD,* ale i vést k přehlednosti, trasovatelnosti a vytvoření správné dokumentace složitých systémů a softwaru.

K vývoji leteckých systémů se používá směrnice SAE ARP 4754A. Vývoj softwaru těchto systémů může být řízen pomocí směrnice RTCA[2] DO-178C (2). K té slouží jako doplněk dokument RTCA DO-331 (3) pro vývoj softwaru, za použití metod *MBD*. Těmito směrnicemi, a jejich aplikací na vybraném příkladě se zabývá tato práce.

[1] MBD v automobilovém průmyslu je řízen podle normy ISO 26262

[2] RTCA – „Radio Technical Commision for Aeronatuics“ dobrovolnická organizace působící jako poradní orgán FAA (Federal Avation Administration)

2 Model-based design

Základní myšlenkou metody *MBD* je sestavení virtuálního modelu za pomoci jednoduchých bloků, které mohou představovat rovnice, logické členy, funkce nebo i podsystémy. Tento přístup pak slouží jako základ pro vytvoření ovládajícího a řídícího systému a softwaru. Jedná se o metodu, která umožňuje pomocí grafického popisu zapsat chování systému nebo softwaru, včetně popisu dynamického chování systému.

Díky možnosti simulace a testování chování modelu systému je možné předejít mnoha chybám již brzy, během vývojového cyklu systému nebo softwaru, a hledat nejvhodnější řešení, které klasický přístup, využívající prototypy neumožňuje. Toto řešení je též časově a finančně nákladnější.

Metoda *MBD* může využívat vysoké míry automatizace. Podstatnou výhodou je možnost automatického generování zdrojového kódu z vytvořeného modelu. Automatizace je ale využívána i při trasování systémových a softwarových požadavků a na jejich adresování v návrhu, nebo na části zdrojového kódu, který těmto požadavkům odpovídá. Lze automaticky generovat i kvantitativní testy systému, získávat jejich výsledné zprávy a verifikovat tak správnost systému.

2.1 Model

Prvním ze tří základních pilířů metody *MBD* je model. Tímto modelem rozumíme strukturu s definovanou syntaxí, která popisuje jakékoliv chování, například s pomocí matematiky, logiky nebo fyziky. V případě inženýrských aplikací je model matematickým popisem chování dynamického systému. Vytvoření správného modelu umožňuje mnohem lépe porozumět fungování navrhovaného systému, prozkoumat možné alternativy řešení a navrhnout nejvhodnější z nich. To vše bez nutnosti vytváření funkčních prototypů nebo fyzických maket. Metoda *MBD* tak umožňuje rychlejší a efektivnější vývoj. Díky tomu mohou i méně početné vývojové týmy s nižším rozpočtem, vytvářet kvalitní produkty v kratším čase než při běžném prototypovém návrhu.

Model se skládá z bloků, které pak společně popisují reálné chování systému. Jednotlivé bloky spolu interagují pomocí vstupů a výstupů, které reprezentují datový tok jednotlivých proměnných. Proměnnými v modelu mohou vystupovat jako veličiny reprezentující napětí, moment síly, teplotu, tlak, logické proměnné atd. Vstupy do modelu jsou pak parametry představující zásahy z vnějšku, jimiž se snažíme dosáhnout požadovaného chování systému. Cílem je dosáhnout takového modelu, jehož výstup a chování odpovídá co nejvíce reálnému systému. Vnitřní stavba modelu může obsahovat různé zpětné vazby, které jsou reprezentovány otevřenými či uzavřenými smyčkami. Je též používáno stavového řízení, kontrolní logiky atd.

2.2 Simulace a testování

Druhým pilířem metody *MBD* je simulace a testování. Virtuální prostředí, ve kterém je model vytvářen, musí umožňovat testování a verifikaci systému kdykoliv v průběhu procesu vývoje. Díky tomu lze zjistit, zda se systém chová podle našeho očekávání a ověřit tak jeho

správnost, nebo upozornit na chyby v systému či modelu, které by jinak byly odhaleny mnohem později a za vyšších nákladů, jako je tomu při klasickém vývojovém přístupu.

Využití vhodného prostředí pro návrh modelu dovoluje vytvářet automatizované scénáře, jež testují chování systému. Zprávy a hodnocení výsledků testů lze automaticky generovat a archivovat. Účelem těchto testů je získat plné pokrytí modelu, které zajišťuje spolehlivý popis chování ve všech možných případech, definovaných pomocí vstupních parametrů.

2.3 Automatické generování kódu

Třetím pilířem je automatická generace zdrojového a objektového kódu. Díky ní lze jednoduše a rychle vytvářet zdrojové a objektové kódy, které můžeme poté spouštět na cílovém hardwaru, nebo simulovat na hostitelském výpočetním zařízení. Tento přístup zabraňuje chybám, které by mohly vzniknout při ručním psaní zdrojového kódu. To umožňuje testovat chtěné chování vyvíjeného systému před samotnou implementací na fyzický prototyp.

Pro zaručení správné funkce zdrojového kódu a usnadnění implementace mají inženýři možnost měnit nastavení automatické generace zdrojového kódu, jako jsou datové objekty, datové typy a třídy. Je tak možné upravit kód, tak aby odpovídal standardům určeným během vývoje. Prostředí pro vývoj modelu, jež umožňuje následnou automatickou generaci kódu, jsou často přednastavené. Na tuto skutečnost je třeba pamatovat a upravit prostředí tak, aby vyhovovalo následnému implementování.

2.4 Vývojová a simulační prostředí pro metody MBD

Zájemce o použití metody *MBD* může dnes vybírat z velkého množství profesionálních vývojových a simulačních prostředí. Obvykle je třeba pro vývoj systémů kombinovat více prostředí, ať už pro jednotlivé fáze procesu vývoje nebo pro podpůrné činnosti. Uživatel si může vybrat ta prostředí, která mu vyhovují, nebo jsou nejvhodnější pro zamýšlený proces vývojového životního cyklu. Vždy by měl ale upřednostňovat prostředí a programy, které spolu dobře spolupracují. Tímto dojde k větší bezproblémovosti a zefektivnění vývojového procesu a vytvoření dostatečné důvěry v produkt, že bude plnit svoji danou funkci s požadovanou spolehlivostí.

Pro účely této práce bylo použito prostředí *Simulink* (1) poskytované společností *MathWorks*, ve kterém bude vytvářen model zvoleného příkladu řídícího softwaru.

3 Vývoj systému podle směrnice SAE ARP 4754A

Pro potřeby vývoje systémů v leteckém průmyslu vznikla směrnice SAE ARP[3] 4754A, publikovaná organizací pro vytváření standardizací SAE International.[4] Tato směrnice je velice podobná směrnici DO-178C,[5] ale poskytuje odborné vedení pro splnění podmínek letové způsobilosti pro vyvíjený systém.

Dodržení směrnice ARP 4754A zajišťuje, aby byl systém vyvíjen s požadovanou pravděpodobností vytvoření chyb, které by mohly ohrozit bezpečnost letadla používající vyvinutý řídící systém. Dodržení směrnice ARP 4754A při vývoji systému má zajistit, že jeho úroveň bezpečnosti odpovídá kategorii poruchového stavu (viz **Tabulka 1**).

Životní cyklus systému podle směrnice ARP 4754A se skládá z fáze plánování, vývojové fáze a integrálních procesů. Popis jednotlivých fází je uveden v následujících kapitolách.

[3] Zkratka pro plný název směrnice „Aerospace Recommended Practice“ („Doporučená praxe pro letecký a kosmický průmysl“)

[4] Zkratka dřívějšího názvu: Society of Automotive engineers („Společnost automobilových inženýrů“)

[5] Název směrnice: DO-178C Software Considerations in Airborne Systems and Equipment Certification („Softwarové aspekty při certifikaci vzdušných systémů a zařízení“)

Tabulka 1: Kategorie poruchových stavů s příslušnými úrovněmi softwaru převzato z (**2**).

Kategorie	Popis	Úroveň zajištění vývojového procesu (*DAL*)[6]
Katastrofická (Catastrophic)	Porucha vedoucí k mnohačetné ztrátě životů či letadla.	A
Riziková (Hazardous)	Porucha snižující schopnost letadla nebo posádky vypořádat se s nepříznivými podmínkami do míry, kdy by mohlo dojít k: • výraznému snížení koeficientu bezpečnosti nebo funkčních schopností. • fyzickému nepohodlí nebo nadměrné zátěži, která snižuje schopnost posádky vykonávat svoje úkony přesně nebo plně. • vážným či smrtelným zraněním relativně malého množství pasažérů.	B
Závažná (Major)	Porucha, která by snížila schopnosti letadla nebo posádky vypořádat se s nepříznivými podmínkami do míry, kdy by nastalo například významné snížení koeficientu bezpečnosti nebo funkčních schopností, významné zvýšení pracovní zátěže posádky a fyzické nepohodlí pro pasažéry či palubní personál s možnými zraněními.	C
Nezávažná (Minor)	Porucha, nesnižující významně bezpečnost letadla a vyžadující činy posádky, které jsou plně v jejich schopnostech. Nezávažným selháním může například být malé snížení bezpečnostního koeficientu nebo funkčních schopností, nepatrné zvýšení pracovní zátěže posádky, nebo fyzické nepohodlí pro posádku či palubní personál.	D
Bez vlivu (No effect)	Porucha, která nemá vliv na bezpečnost. Například selhání, které neovlivní operační schopnosti letadla nebo pracovní zátěž posádky.	E

[6] Zkratka pro Design assurance level („úroveň zajištění vývojového procesu “)

3.1 Plánování vývoje systému

Účelem plánovacího procesu je určit metody, prostředky, činnosti, vztahy, standardy, nástroje a další případné faktory, které budou využívány při vývoji a zajistí tak potřebnou úroveň systému. Cílem této fáze je naplánovat jednotlivé procesy životního cyklu a jejich vztahy, a zajistit tak řádné vedení projektu společně s dokumentací vývoje systému.

3.2 Vývoj systému

Při vývoji systému letadla jsou funkce letadla rozděleny do skupin, kterým jsou přiřazeny systémy, jež zajistí jejich funkci (například varovný systém motoru sleduje stav motoru a jeho poruchy). K systému jsou posléze definovány požadavky, které zajišťují správné vykonání přiřazených funkcí. Na základě analýzy rizik provozu letadla jsou stanoveny možné poruchové stavy, vlivy a bezpečnostní požadavky. Z této analýzy a ze stanovených systémových požadavků je vyvíjena systémová architektura. Ta určuje strukturu propojených součástí systému a jejich omezení určené požadavky. Při vývoji architektury systému by měla být zvážena i další rizika samotného systému určená předběžným hodnocením bezpečnosti systému. Během vývoje může být zvažováno více možných architektur systému. Výsledná architektura pak může být vybrána na základě technologické dostupnosti, výkonosti, předchozích zkušeností či ekonomické výhodnosti. Z vytvořené architektury je potřeba určit požadavky na samotné součásti systému (hardware a software). Vývojem těchto součástí a jejich implementací se zabývá směrnice DO-178C pro software. Pro vývoj a implementaci hardwaru slouží směrnice DO-254 *„Design Assurance Guidance for Airborne Electronic Hardware“*.

3.3 Integrální procesy

Jedná se o procesy, které probíhají během vývoje systému a mají podpořit ostatní procesy. Těmito procesy jsou: ověřování (verifikace), určení konfigurace systému, zajištění kvality a proces spolupráce s certifikační autoritou. Tyto integrální procesy jsou doplněny o procesy určení a validace požadavků, hodnocení bezpečnosti a přiřazení úrovně zajištění vývojového procesu odpovídající kategorii selhání (viz. **Tabulka 1**).

K hodnocení bezpečnosti slouží různé procesy, jež mohou určit další případné požadavky na systém, architekturu nebo na jeho součásti. K posouzení potenciálních selhání funkcí letadla a systémů slouží analýza funkčních rizik *„Functional Hazard Analysis“* (*FHA*) společně s předběžným hodnocením bezpečnosti systému. Analýza funkčních rizik se snaží předpovědět, jak selhání v systému mohou vést k poruchovým stavům. Pomocí dalších analýz je přezkoumáváno vzájemné oddělení systémů, projevy jejich možných selhání a příčiny vedoucí k těmto selháním, dále pak je ověřena nezávislost mezi detekovanými selháními.

Vývoj systémů probíhá na základě stanovených požadavků, mezi ně můžeme zařadit (4):

- Funkční požadavky – určují požadované funkce a chování systémů.
- Požadavky zákazníka – jsou dány zákazníkem, jeho očekávání od výsledného produktu.
- Požadavky pro komunikaci – jsou určeny k zajištění rozhraní a komunikace, mezi jednotlivými systémy, příp. personálem (posádka, údržba atd.) a letadlem.
- Požadavky pro bezpečnost – zajištující správnou funkci a integritu systémů.
- Požadavky na výkonnost – určující efektivitu a rychlost systémů.
- Certifikační požadavky – jejichž splnění zajišťuje soulad s regulacemi pro letovou způsobilost.

Správně určené požadavky by měly být jasné, důkladné, dosažitelné, ověřitelné, trasovatelné a nenadbytečné.

Na základě těchto požadavků jsou pak vyvíjeny a implementovány jednotlivé součásti systému – hardware a software.

4 Vývoj softwaru podle směrnic DO-178c a DO-331

Pro vývoj softwaru v leteckém průmyslu slouží směrnice DO-178C *„Software Considerations in Airborne Systems and Equipment Certification“* (2) vytvořená zvláštní komisí *„Radio technical commitie for Aeronautics“* (*RTCA*) a pracovní skupinou číslo 71 *„European Organisation for Civil Aviation Equipment“* (*EUROCAE*). Směrnice DO-178C vznikla na základě potřeby odborného vedení při vývoji softwaru pro splnění podmínek letové způsobilosti. Tento dokument byl pak na základě nově nabytých zkušeností, a postupujícího vývoje několikrát upravován až do nejaktuálnější verze DO-178C (2).

S rostoucím využitím metody *MBD* nastala potřeba definovat vhodné postupy pro tuto metodu. K tomuto účelu vznikla doplňující směrnice DO-331 *„Model-Based Development and Verification Supplement to DO-178C and DO-278A“*, která obsahuje modifikace a doplňky k DO-178C. Předpokládá se tak, že při použití metody *MBD* jsou zároveň naplněny cíle a činnosti doporučované DO-178C a DO-331, případně kombinované s dalšími doplňky DO-178C (3).

Použití směrnic DO-178C a DO-331 je dovoleno oběžníkem AC 20-115D. Splněním jejich cílů, spojených s úrovní zajištění vývojového procesu (**Tabulka 1**), a vytvořením souvisejících dat životního cyklu softwaru, bude vyvinutý software splňovat základní podmínky pro certifikaci dané certifikačními autoritami.

Při vývoji softwaru mohou být použity i další směrnice, například doplňková směrnice DO-332 *„Object-Oriented Technology and Related techniques Supplement to DO-178C and DO-278A“* pro objektově orientovaný vývoj, nebo směrnice DO-333 *„Formal Methods Supplement to DO-178C and DO-278A“*, pro použití jiných metod vývoje softwaru, než je metoda *MBD*. (5).

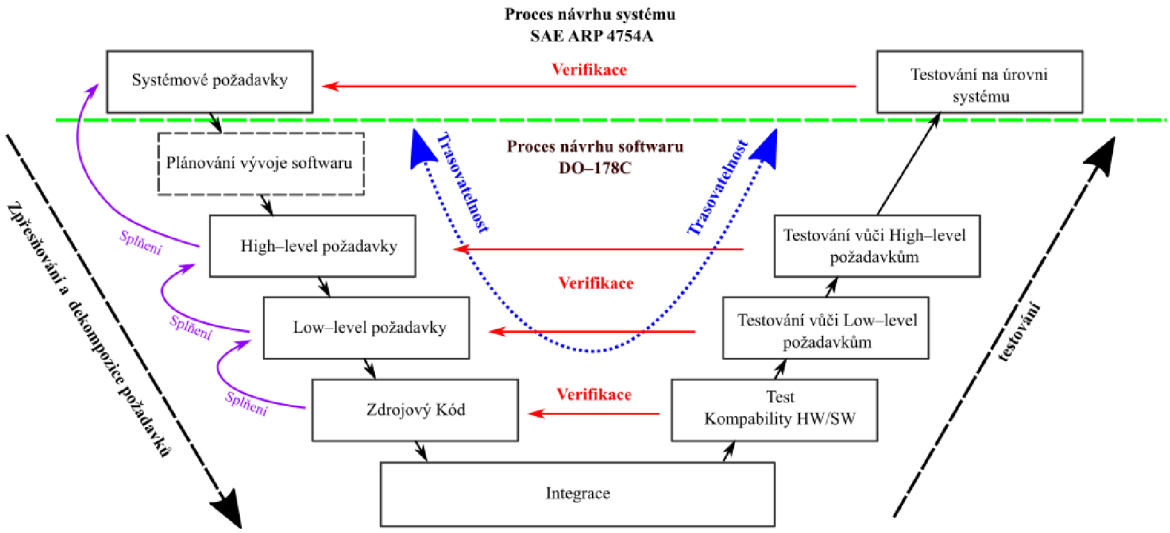
Veškeré zmíněné směrnice popisují, jaké vlastnosti a cíle, by jednotlivé fáze životního cyklu softwaru měly mít. Tyto cíle pro jednotlivé fáze jsou shrnuty v tabulkách, které se nachází v přílohách jednotlivých směrnic. Jako příklad je uvedena **Tabulka 2** v kapitole 4.2.

4.1 V-model pro vývoj softwaru

Při metodě *MBD*, je vždy nutné splnit cíle a požadavky směrnice DO-178C a současně i doplňku DO-331. Postup při vývoji softwaru s využitím a bez využití *MBD,* podle těchto dokumentů, je obdobný. **Obrázek 1** ukazuje tzv. *V-model*[7] vývoje softwaru čistě podle směrnice DO-178C a **Obrázek 2** v následující kapitole ukazuje *V-model* pro metodu *MBD* řídící se i směrnicí DO-331.

[7] Označení V je odvozeno od tvaru diagramu pro „*V-model*“, který připomíná uvedené písmeno. Předchůdcem *V-modelu* je tzv. *Waterfall model.*

4.1.1 V-model podle směrnice DO-178C



Obrázek 1: V-model pro DO-178C

Prvním krokem podle směrnice DO-178C je určení systémových požadavků, které jsou kladeny na software. Tyto požadavky pocházejí z předcházejícího vývoje systému, který je veden podle směrnice SAE ARP4754A (viz **Kapitola 3**).

Před samotným vývojem softwaru podle směrnice DO-178C, je třeba nejprve naplánovat jednotlivé aspekty vývojového životního cyklu softwaru. Těmi jsou například vývojová prostředí a nástroje, které budou použity pro vývoj a testování vyvíjeného softwaru**,** definice standardů pro psaní softwarových požadavků či standardy softwarového kódu. V průběhu tohoto procesu je nutno zohledňovat požadavky dané certifikačními autoritami jako jsou instituce *FAA*[8], *EASA*[9], či vnitřními směrnicemi firmy, která systém nebo software vyvíjí.

Na základě požadavků alokovaných z úrovně systému definujeme takzvané *high-level* požadavky. Jsou to požadavky na software, které určují předpokládané vstupy, výstupy a základní požadovanou funkci software. Nejedná se zatím o samotnou logiku softwaru, ale předpokládané chování na vyšší úrovni abstrakce. Příkladem *high-level* požadavku může být: *„Posádka je vizuálně a zvukově upozorněna na neschopnost autopilota vykonávat svoji požadovanou funkci.“*

Rozvíjením *high-level* požadavků získáváme nižší úrovně softwarových požadavků, tzv. *low-level* požadavky. Ty určují detailní návrh softwaru. Cílem je určit posloupnosti, návaznosti a vztahy jednotlivých operací, které software bude vykonávat pro zajištění své funkčnosti. *Low-level* požadavky udávají implementační detaily pro návrh softwaru. Mohou

[8] Federal Aviation Administration – agentura Ministerstva dopravy Spojených států amerických, regulující civilní letectví.
[9] European Aviation Safety Agency – Evropská agentura letecké bezpečnosti, regulující civilní letectví.

to být výpočty (matematické funkce) použité ve funkcích vyvíjeného softwaru, přesné určení typu dat a manipulace s nimi, a další specifika, která naplňují *high-level* požadavky.

Posledním krokem vývoje je samotné psaní kódu. Díky předchozím krokům, zejména pak díky určeným *low-level* požadavkům, může i programátor, který není důkladně obeznámen s technologickou a funkční stránkou projektu, jednoduše vytvářet zdrojový kód.

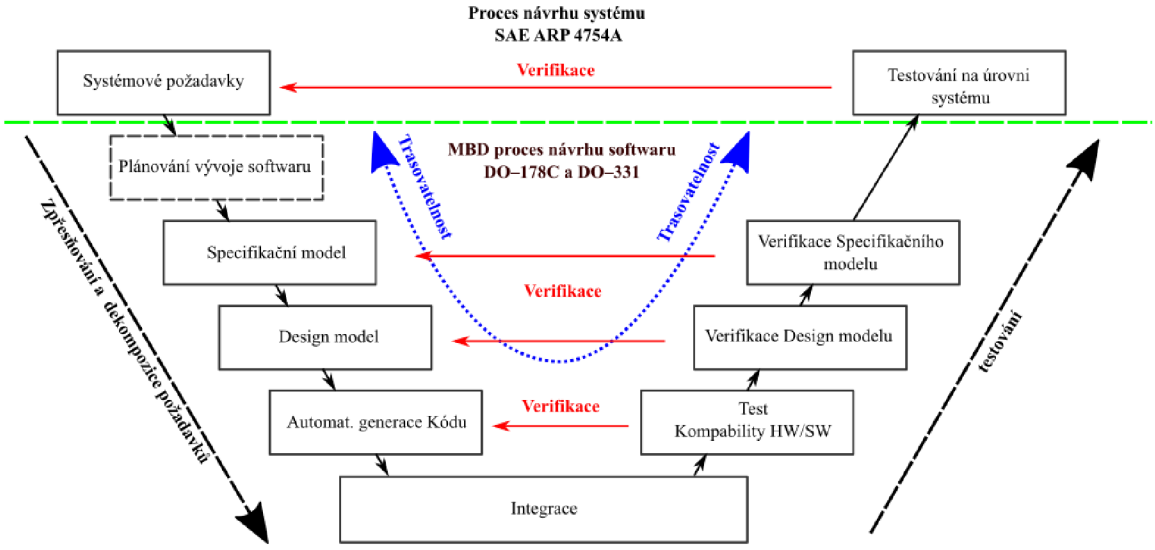
Pro zajištění správnosti a bezchybnosti softwaru je třeba ověřit, zda výstupy jednotlivých procesů a fází životního cyklu softwaru splňují svoje požadované cíle.

Výstupy získané během všech fází vývojové části životního cyklu softwaru kontrolujeme, zda splňují požadavky na vyšších úrovních. Ověřujeme, zda *high-level* požadavky splňují systémové požadavky, *low-level* požadavky splňují *high-level* požadavky, a tedy i systémové požadavky. Zdrojový kód musí splňovat *low-level*, *high-level* a systémové požadavky. Všechny výstupy vývojové části musí splňovat i metody, standardy a další požadavky na jejich formu získání.

Testováním softwaru pak ověřujeme, zda výsledný, integrovaný software je kompatibilní s hardwarem na kterém je nahrán, odpovídá a naplňuje *low-level* požadavky a *high-level* požadavky. Výsledný systém je pak ověřen, zda byly naplněny systémové požadavky.

V průběhu vývoje softwaru je kladen důraz na vytváření trasovatelnosti mezi jednotlivými fázemi životního cyklu softwaru. Trasovatelnost slouží pro lepší orientaci ve vývoji, ověřování požadavků, verifikaci atd.

4.1.2 V-model podle směrnice DO-178C a dodatku DO-331



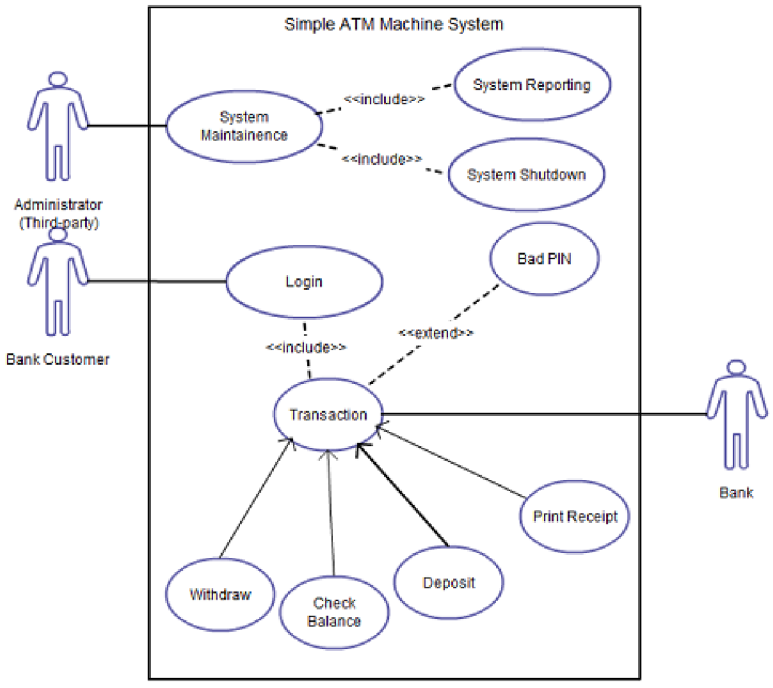
Obrázek 2: Příklad V-modelu podle DO-178C a DO-331

V případě vývoje softwaru podle směrnice DO-178C a dodatkové směrnice DO-331, je postup obdobný jako v předcházející kapitole. **Obrázek 2** ukazuje vývoj softwaru dle metody *MBD*, který odpovídá příkladu MB 2. Zde jsou nahrazeny *high-level* požadavky takzvaným *„specifikačním modelem“* a *low-level* požadavky takzvaným *„design modelem“*. Nakonec, je-li třeba, může být využito možnosti automatického generování počítačového kódu

Specifikační model reprezentuje *high-level* požadavky. *Specifikační model* by měl vyjadřovat tyto požadavky jednoznačně pro porozumění softwarové funkcionality. Popisuje, jak by se měl software chovat, ale ne do takové míry, aby určoval, jak tohoto chování dosáhne. Neudává ani softwarové podrobnosti jako jsou použité datové typy, jejich tok a řízení. (3) Příklad jednoduchého *specifikačního modelu*, ukazuje **Obrázek 3**. Tento model byl vytvořen v prostředí softwaru *UML*.

Ze *specifikačního modelu* pak lze vytvářet návrh softwaru pomocí *Design modelu*, který je obrazem *low-level* požadavků. Tento model je již představitelem funkčního softwaru se všemi požadovanými vlastnostmi. Obsahuje určení datových struktur, jejich tok a kontrolu. (3)

Díky vytvoření *Design modelu* v patřičném vývojovém prostředí, můžeme využít nástroje, které umožňují vygenerovat zdrojový kód v daném programovacím jazyce. V případech, kdy automatické generování je nežádoucí, lze vytvářet zdrojový kód ručně.



Obrázek 3: Příklad jednoduchého specifikačního modelu z bankovnictví v prostředí vývojového softwaru UML **(21)**

Při životním cyklu softwaru podle obou výše uvedených postupů, je důležité udržovat trasovatelnost mezi požadavky. Díky tomu můžeme rychle ověřovat, zda požadavky byly splněny. Lze snadno vyhledávat chybné části a efektivněji verifikovat či validovat celý vývoj.

4.2 Systémová stránka vývoje softwaru

Pro software je třeba stanovit požadavky, které vyplývají z požadavků na samotný systém, kterého je součástí. Tyto požadavky je třeba splnit pro zajištění bezpečnosti, spolehlivosti, výpočetního výkonu a správné funkce navrhovaného softwaru a systému. Požadavky na systém mohou vést k různým požadavkům na funkce, integritu a spolehlivost nebo k restrikci některých možností softwarové architektury. (2).

Součástí posudku bezpečnosti vyvíjeného systému je i posouzení odolnosti softwaru vůči poruchovým stavům. K zabránění či omezení poruch vznikají požadavky pro detekci, toleranci, odstranění nebo vyhnutí se chyb. Tyto požadavky jsou přepracovány na softwarové požadavky a ověřeny činnostmi verifikačního procesu softwaru, pro vykonávání zamýšlených funkcí softwaru v jakýchkoliv předvídatelných situacích. (2)

Na základě vlivu chyby ve vyvíjeném softwaru na potenciálním poruchovém stavu systému je určena tzv. *úroveň softwaru, „Design Assurance Level“* (*DAL*). Tato úroveň se odvíjí od závažnosti poruchy systému a jejího projevu na úrovni letadla. Běžně, například pro velká transportní letadla, rozlišujeme pět kategorií poruchových stavů, kterým náleží příslušná úroveň softwaru (A až E), a které uznává i směrnice DO-178C (viz **Tabulka 1**). Tyto úrovně určují i cíle a aktivity, které musí být naplněné během životního cyklu softwaru. V případě úrovní odpovídajících nebezpečnějším kategoriím poruchových stavů (nejzávažnější je úroveň A) je třeba, aby cíle některých procesů, byly dosaženy skupinou inženýrů, která je nezávislá na skupinách provádějících ostatní činnosti. Například v případě softwaru úrovně A je potřeba, aby tým, odpovědný za verifikaci souladu *low-level* požadavků s *high-level* požadavky, byl nezávislý na týmu, který tyto požadavky určoval. V případě úrovní odpovídající méně nebezpečným kategoriím poruchových stavů lze některé cíle a aktivity vynechat. Pro software úrovně E není třeba provádět jakékoliv aktivity. Jako příklad je uvedena **Tabulka 2.** (2)

Pro zamezení dopadů chyb lze během vývoje softwaru přistoupit k různým návrhovým řešením, které budou vůči těmto chybám odolné. Pomocí rozdělení softwaru do dílčích části, lze případné chyby izolovat a zabránit tak jejich šíření do dalších částí. Takto rozdělené části softwaru mívají přiřazený vlastní hardware. Je možné použít i jiný software, který má sice stejnou funkci, ale vykonává ji jiným způsobem, který nedovoluje výskyt stejných chyb. Můžeme vytvořit i funkce, které budou sledovat, zda nenastane konkrétní selhání a zabránit mu. (2)

Tabulka 2: Verifikace výstupů procesu určení softwarových požadavků převzato z (**2**). Označení pomocí kolečka (● nebo ○), určují, které cíle mají být naplněny pro odpovídající úroveň softwaru. V případě černého kolečka (●) pak tyto cíle mají být splněny s nezávislostí. V případě, že chybí tento způsob označení u cíle, není ho třeba splnit pro danou úroveň softwaru.

	Cíle	Platnost pro úroveň softwaru (*DAL*)				Výstup
	Popis	A	B	C	D	Soubor dat
1	High-level požadavky souhlasí se systémovými požadavky	●	●	○	○	Výsledky verifikace softwaru
2	High-level požadavky jsou přesné a konzistentní	●	●	○	○	Výsledky verifikace softwaru
3	High-level požadavky jsou kompatibilní s cílovým výpočetním zařízením	○	○			Výsledky verifikace softwaru
4	High-level požadavky jsou ověřitelné	○	○	○		Výsledky verifikace softwaru
5	High-level požadavky odpovídají standardům	○	○	○	○	Výsledky verifikace softwaru
6	High-level požadavky jsou trasovatelné k systémovým požadavkům	○	○	○		Výsledky verifikace softwaru
7	Algoritmy jsou přesné	●	●	○		Výsledky verifikace softwaru

Uvedená **Tabulka 2** je příkladem cílů, kterých se snaží dosáhnout proces verifikace výstupů, které vznikly během určení softwarových požadavků. Výsledky tohoto procesu (zda byly cíle splněny a jak byly ověřeny) jsou pak uvedeny v samostatném dokumentu: *„Výsledky verifikace softwaru“*.

4.3 Životní cyklus softwaru

Fáze životní cyklu softwaru jsou (2):

- plánovací proces softwaru, jehož účelem cílem je stanovení použití a výběru prostředků pro všechny ostatní procesy.
- vývoj softwaru, během kterého jsou určeny požadavky, vytvořen návrh, kódován zdrojový kód a proběhl proces integrace.

- v průběhu ostatních fází probíhají integrální procesy, které zajišťují správnost a kontrolu softwaru. Jsou to procesy verifikace, určení konfigurace, zajištění kvality a proces spolupráce s certifikační autoritou.

4.3.1 Určení životního cyklu pro Model Based Design

Životní cyklus softwaru je definován stanovením jednotlivých aktivit vykonávaných během jednotlivých fází vývoje softwaru, jejich pořadím a zodpovědností za ně. Použít metodu *MBD* lze různými způsoby v různých fázích vývoje softwaru. V některých případech lze i určité aktivity jednotlivých fází nahradit. **Tabulka 3** ukazuje jakými způsoby lze nahradit procesy vývoje softwaru použitím metody *MBD* u pěti příkladů.

Tabulka 3: Příklady použití metody MBD převzato z **(3)**

<table>
<tr><th>Procesy generující data životního cyklu</th><th>Příklad MB 1</th><th>Příklad MB 2</th><th>Příklad MB 3</th><th>Příklad MB 4</th><th>Příklad MB 5</th></tr>
<tr><td rowspan="2">Procesy definující systémové požadavky a návrhy systému</td><td rowspan="2">Požadavky přiřazené softwaru</td><td rowspan="2">Požadavky, ze kterých je vyvíjen model</td><td rowspan="2">Požadavky, ze kterých je vyvíjen model</td><td rowspan="2">Požadavky, ze kterých je vyvíjen model</td><td>Požadavky, ze kterých je vyvíjen model</td></tr>
<tr><td rowspan="3">Design model</td></tr>
<tr><td rowspan="2">Procesy softwarových požadavků a návrhů software</td><td>Požadavky, ze kterých je vyvíjen model</td><td>Specifikační model</td><td>Specifikační model</td><td rowspan="2">Design model</td></tr>
<tr><td>Design model</td><td>Design model</td><td>Textový popis návrhu softwaru (=softwarové požadavky)</td></tr>
<tr><td>Proces kódování softwaru</td><td>Zdrojový kód</td><td>Zdrojový kód</td><td>Zdrojový kód</td><td>Zdrojový kód</td><td>Zdrojový kód</td></tr>
</table>

MB příklad 1: Tento příklad ukazuje použití metody *MBD*, která nahrazuje návrhový proces, kdy jsou určeny *low-level* požadavky *Design modelem*. Z tohoto *Design modelu* pak můžeme automaticky vygenerovat kód. Požadavky, ze kterých je vyvíjen model (*high-level* požadavky) jsou textové.

MB příklad 2: Zde je uvedený příklad, kdy oproti prvnímu je použit pro softwarové požadavky *Specifikační model*. *Design model* je vytvořen z tohoto modelu.

MB příklad 3: *MBD* může být použit během návrhu na vyšší úrovni pomocí specifikačního modelu. Z něho můžeme vytvořit textové *low-level* požadavky a popis návrhu. V tomto příkladu není použit *Design model*, nelze tak automaticky generovat kód, který musí být vytvořen manuálně.

MB příklad 4: Uvádí příklad, kdy systémové požadavky splývají se softwarovými *high-level* požadavky. Systémové požadavky, ze kterých je vyvíjen model musí také splňovat cíle pro *high-level* požadavky dle směrnice DO-178. Z nich je rovnou tvořen *Design model*. I v takovémto případě musí být splněny cíle dodatku DO-331.

MB příklad 5: Systémová a softwarová data životních cyklů nelze v některých případech oddělit. I přesto platí, že cíle dané směrnicemi DO-178C a DO-331 musí být splněny nezávisle na tom, odkud model pochází. *Design model* v tomto případě bude muset splnit i procesní požadavky kladené na systémové požadavky, protože je z části definuje.

Software se může skládat z více komponent, které mají různé životní cykly v závislosti na jejich funkcionalitě a komplexnosti. Životní cyklus ovlivňují i způsoby jakým je vývoj veden, případně použití již vyvinutého softwaru nebo dostupnost vhodného hardwaru. (3)

4.3.2 Přechodová kritéria mezi procesy

Přechodová kritéria určují podmínky, které musí být splněny předchozím procesem pro započatí činností následujícího procesu. Pokud jsou splněna přechodová kritéria procesu, lze tento proces zahájit i přesto že postrádá některé vstupní informace. Při jejich pozdějším získání je však třeba přezkoumat, zda předchozí výstupy vývojového a verifikačního procesu softwaru jsou stále platné. (2)

Každý proces může vyvolat zpětnou vazbu pro jiný proces, který ovlivní. Pro zpětnou vazbu je třeba určit, jak je s jejími informacemi nakládáno, jak jsou rozpoznány, kontrolovány a použity procesem, který tuto zpětnou vazbu přijímá. (2)

4.4 Plánování softwaru

Jako první proces v životním cyklu softwaru je samotné naplánování jednotlivých činností. Tento proces definuje činnosti, vztahy, prostředí, metody, nástroje, standardy a další případné ohledy, které budou zajišťovat splnění a adresování požadavků na úroveň softwaru, jejich posloupnost, zpětnou vazbu a přechod mezi jednotlivými procesy životního cyklu softwaru. Plánování dále určuje, jak budou naplněny požadavky na bezpečnost a na data potřebná pro fáze životního cyklu softwaru. (2)

Během tohoto procesu vznikají následující plány:

- **plán pro softwarové aspekty certifikace *„Plan for Software aspects of certification“* (*PSAC*)**, který určuje prostředky pro získání certifikace softwaru od dané autority.
- **vývojový plán *„Software Development Plan“* (*SDP*)** pro prostředí a prostředky nutné pro vývoj softwaru.
- **plán pro verifikaci *„Software Verification Plan“* (*SVP*)** určující prostředky pro ověření softwaru.
- **plán určení konfigurace softwaru *„Software Configuration Managment Plan“* (*SCMP*)** zajišťující prostředky pro dosažení cílů určení konfigurace softwaru.
- **plán pro zajištění kvality softwaru *„Software Quality Assurance Plan“* (*SQAP*)** definující prostředky pro zajištění cílů procesu zajištění kvality.

4.4.1 Plánování prostředí životního cyklu softwaru

Plánování prostředí životního cyklu softwaru určuje metody, nástroje, procedury, programovací jazyk a hardware který bude použit v jednotlivých procesech pro vývoj, verifikaci, kontrolu a vytvoření dat životního cyklu softwaru. Výběr prostředí a způsob jejich kombinace (v případě použití více prostředí) může mít vliv na celkovou kvalitu vyvíjeného softwaru. Vhodným výběrem lze dosáhnout snížení potenciálního vzniku chyb a jejich dopad na celkový systém. (2)

Při plánování použití *MBD*, je potřeba ověřit, zda je pro certifikační autoritu přijatelné použití této metody, to platí i o všech případných jiných doplňcích, které by uživatel chtěl použít. Vždy musí být splněny požadavky a cíle směrnice DO-178C, a následně splněny cíle dodatku DO-331 a případných dalších dodatků. Pokud dojde ke konfliktu mezi nimi, je třeba navrhnout řešení uvedené v plánu *PSAC*. Směrnice DO-331 upřednostňuje použití jednoho plánu děleného do sekcí zabývajících se doplňky než více plánů, které zvolené doplňky adresují zvlášť. (3)

Vývojové prostředí

Pro zajištění kvalitního softwaru, který nebude trpět chybami vzniklými během vývoje je třeba určit vývojové prostředí, které bude do jisté míry jistoty schopné detekce vzniklých chyb v softwaru. Vývojové prostředí by mělo umožnit dodržení a verifikaci standardů. Plánování vývojového prostředí by mělo určit i postup použití jednotlivých nástrojů pro účely certifikace a zhodnotit možný nepříznivý dopad použitého prostředí a nástrojů. (2)

Faktory použitého programovacího jazyka a kompilátoru

Pro úspěšnou certifikaci softwaru, je třeba zvážit specifické prvky použitého programovacího jazyka a kompilátoru.[10] Některé kompilátory jsou přizpůsobeny pro optimalizaci výkonu objektového kódu. Pokud testování nevykazuje odpovídající výsledky je třeba přezkoumat

[10] Program převádějící zdrojový kód z programovacího jazyka do strojového kódu.

dopad prvků použitého kompilátoru. Kompilátor taky nemusí poskytnout přímou trasovatelnost sobě příslušných částí objektového a zdrojového kódu. Plánovací proces softwaru by měl zajistit prostředky pro tuto trasovatelnost kvůli verifikaci. Zároveň při změně nastavení nebo použití nové verze kompilátoru, *linkage editoru*[11] či *loaderu*[12] nemusí být předchozí testy a analýzy kódu platné. (2)

Testovací prostředí softwaru

Plán prostředí pro testování vyvíjeného softwaru určuje metody, nástroje, procedury a hardware, které budou použity pro testování výstupů integračního procesu. Pro testování na cílovém výpočetním zařízení, emulátoru či simulaci na hostitelském počítači, je třeba zajistit jejich kvalifikaci, pro provádění těchto testů a popsat rozdíly mezi cílovým výpočetním zařízením, emulátorem či simulací a jejich dopadem na chybové detekce a verifikační funkce. (2)

Simulační prostředí-

Plánování simulačního prostředí modelu softwaru definuje metody, nástroje, procedury a další vlastnosti použitého prostředí, kterých je využíváno k simulaci a verifikaci výstupů procesu vývoje modelu. Je třeba vzít v potaz, zda lze dané prostředí a nástroje kvalifikovat stejně jako jiné nástroje. Měla by být zvážena schopnost a limitace simulátoru modelu s ohledem na jeho zamýšlené použití, detekci chyb a verifikační funkcionalitu. Detekce chyb, kterou nezvládne modelová simulace, musí být adresována jiným softwarovým verifikačním procesem a příslušnými metodami určenými v plánu verifikace softwaru. (3)

4.4.2 Vývojové standardy

Standardy požadavků, návrhu a kódu určují další pravidla a omezení pro vývoj softwaru. Verifikační proces pak používá tyto standardy pro ověření souladu mezi skutečnými a zamýšlenými výstupy. Vývojové standardy jsou základem pro jednotný vývoj softwarového produktu nebo příbuzné sady produktů. Jejich účelem je mj. zamezit použití konceptů nebo metod, jejichž výstupy nelze ověřit, nebo nesplňují bezpečnostní požadavky. (2)

4.5 Vývoj softwaru

Vývoj softwaru je řízen plánem *SDP*. Součástí vývoje je proces určení požadavků, návrhu, kódování a integrace. Vývoj softwaru může produkovat požadavky, které mohou mít jednu a více úrovní. Z analýzy systémových požadavků a architektury systému vznikají *high-level* požadavky. Tyto požadavky jsou v průběhu vývoje více rozvíjeny a vznikají z nich *low-level* požadavky. Ovšem v odůvodněném případě lze z těchto *high-level* požadavků vytvářet kód. Jsou pak považovány zároveň za *low-level* požadavky a je s nimi i stejné zacházeno. (2) To neplatí v případě použití metody *MBD*. Při použití *Specifikačního modelu*,

[11] Program zodpovědný spojení objektových souborů do jednoho, který je pak sputitelný.
[12] Program, zodpovědný za načtení strojového kódu nebo knihoven.

reprezentujícího *high-level* požadavky, a *Design modelu*, reprezentujícího *low-level* požadavky. Existuje tak vždy více jak jedna úroveň požadavků softwaru. (3)

Během vývoje softwaru mohou vzniknout *odvozené* (*„derived“*) požadavky. Tyto požadavky nevznikají z vyšších požadavků (z *high-level* nebo systémových požadavků) a tak nejsou ani k nim trasovatelné. Tyto požadavky by měly být přezkoumány pro zjištění jejich dopadů na *high-level* nebo systémové požadavky. (2)

4.5.1 Proces určení požadavků pro software

Na základě systémových požadavků, vlastností komunikačního rozhraní hardwaru, vývojového plánu softwaru (*SDP*) a standardů pro požadavky softwaru, jsou vytvářeny *high-level* a *odvozené high-level* požadavky. Tyto požadavky by měly souhlasit s určenými standardy, a také by měly být ověřitelné a konzistentní. Jejich zápis by měl být proveden v kvantitativní formě s přípustnými tolerancemi (například: *„Systém Autothrottle*[13] *udržuje rychlost letadla s tolerancí ±3 m/s.“*). Pokud není odůvodněno, neměly by *high-level* požadavky předepisovat návrh softwaru. V případě *odvozených high-level* požadavků je potřeba účel jejich existence řádně zdůvodnit. (2)

V případě nahrazení *high-level* požadavků S*pecifikačním modelem*, musí tento model odpovídat softwarovým standardům modelu, stejně tak jako jeho prvky, které by v tomto standardu měly být popsány. (3)

4.5.2 Návrh softwaru

Opakovaným propracováním je z *odvozených* a *high-level* požadavků vytvářena softwarová architektura a *low-level* požadavky, ze kterých lze vytvářet zdrojový kód. Stejně jako u *high-level* požadavků, mohou vzniknout *odvozené low-level* požadavky, které musí být řádně zdůvodněny. *Low-level* požadavky mohou mít i více než jednu úroveň, a platí, že i ony se řídí vývojovým plánem softwaru (*SDP*) a standardy pro návrh softwaru. (2) *Design model*, který nahrazuje *low-level* požadavky musí splňovat softwarové standardy modelu společně se svými prvky, které by v něm měly být popsány. (3)

Výstupem tohoto procesu je návrh softwaru, který je tvořen popisem softwarové architektury a *low-level* požadavky. Obsahem návrhu softwaru je stanovení algoritmů, datových struktur, implementací architektur, vstupů a výstupů, toku dat, rozvrhování procesů, dělení oddílů atd. (2)

4.5.3 Kódování a integrování softwaru

Během programování softwaru je vytvářen zdrojový kód na základě *low-level* požadavků, vývojového plánu softwaru (*SDP*) a kódovacích standardů. Tento proces je spojen s integračním procesem, a končí teprve po naplnění všech svých cílů a současně cílů integračního procesu. (2) V případě použití *MBD*, lze zdrojový kód automaticky vytvářet z *Design modelu*, pokud to prostředí, ve kterém je modelován dovoluje.

[13] Systém udržující stálou rychlost letadla.

Zdrojový kód může být nahrán jak na cílovém výpočetním zařízení, emulátoru nebo hostitelském výpočetním zařízení. V případě, že používaný software pro implementaci má novou aktualizaci, opravující chyby nebo zavádějící změny v softwaru, neměla by tato aktualizace být použita bez řádného odůvodnění a prozkoumání jeho dopadů na vývoj a systém. (2)

4.6 Trasovatelnost vývojových procesů

Požadavky vzniklé během vývoje softwaru by měly být mezi jednotlivými úrovněmi trasovatelné, a to obousměrně. Trasovatelnost by měla být mezi (2):

- systémovými požadavky alokovanými na software a *high-level* požadavky.
- *high-level* požadavky a *low-level* požadavky.
- *low-level* požadavky a zdrojovým kódem.

Odvozené požadavky, které nelze trasovat na vyšší úrovně, by měly byt náležitě zvýrazněny v dokumentaci. (2)

4.7 Verifikace softwaru

Verifikací se rozumí technické hodnocení výstupů plánovacího, vývojového a verifikačního procesu z hlediska stanovených cílů a úkolů. Nejedná se jen o testování, ale o kombinaci revizí, analýz a testů provedených podle plánů pro verifikaci. Cílem je odhalit chyby, které vznikly během vývojových procesů. (2)

4.7.1 Revize a analýza softwaru

Revizí vývojových procesů softwaru dochází k inspekcí výstupů, zda řádně splnily cíle svých procesů. Analýza vyvíjeného softwaru detailně prozkoumává funkčnost, výkonnost, trasovatelnost, implementaci softwarových součástí a vztah mezi nimi. (2)

Revizí a analýzou *high-level, low-level* požadavků, softwarové architektury a zdrojového kódu ověřujeme, že vyhovují o úroveň vyšším požadavkům (tj., že *high-leve*l vyhovují systémovým požadavkům, *low-level* vyhovují *high-level* požadavkům atd.), jsou přesné a konzistentní, kompatibilní s cílovým výpočetním zařízením, verifikovatelné, odpovídající standardům a jsou trasovatelné. (2)

Testování softwaru

Testování softwaru ukazuje, do jaké míry a s jakou jistotou software splňuje *high-level* a *low-level* požadavky. Pro účely testování může být zvoleno více druhů testovacích prostředí. To by mělo co nejvíce odpovídat skutečným provozním podmínkám. Testovat lze různými způsoby vytvořením scénářů, které otestují požadavky softwaru, odezvu softwaru na správné vstupy a nesprávné vstupy atd. (2)

Analýza pokrytí modelu pro Design model

Analýza pokrytí modelu určuje, které požadavky definované modelem, nebyly obsaženy v požadavcích, ze kterých byl model vyvinut. Slouží také pro posouzení důkladnosti verifikačních aktivit modelu a pomáhá k zjištění nezamýšlených funkcí *Design modelu*. Existuje více metod a kritérií pro analýzu pokrytí modelu. Použitá kritéria by měla být definována během plánovacího procesu. (3)

Simulace modelu

Cílem simulace modelu je poskytnout důkazy o dodržení požadavků, na základě kterých je model vytvářen. V případě simulace modelu, by mělo být ověřeno, že cíle verifikace *high-level* požadavků, softwarového návrhu, zdrojového kódu a testování softwaru, byly splněny. V případě že simulace modelu je použita jako podpora dosažení těchto cílů, je třeba zajistit že tato simulace je plánována, rozvinuta a vykonána úplně a správně. Pro *Specifikační* a *Design model* může být simulace použita pro ověření a analýzu *high-level/low-level* požadavků a softwarové architektury. Pro *Design model* může být simulace také použita pro testování a analýzu a pro zjištění splnění cílů spjatých s verifikací zdrojového kódu. Nesmí být použita pro celkovou verifikaci kódu, vzhledem k tomu, že může probíhat v jiném prostředí a s jiným zdrojovým kódem. To neplatí v případě, že simulace probíhá na cílovém výpočetním zařízení se stejným zdrojovým kódem. (3)

4.8 Určení konfigurace softwaru

Tento proces je jedním z integrálních procesů. Jeho účelem je zachování a řízená kontrola konfigurace softwaru během životního cyklu softwaru. Cílem je poskytnout informace pro případné replikování zdrojového kódu a dat, vstupů a výstupů mezi fázemi životního cyklu, zajištění řešení chyb a změny pro jejich odstranění atd. Pro zajištění těchto informací je veden záznam o verzích konfigurovaných položek, nesouladu procesů se standardy, nedostatcích ve výstupech procesů, kontrole změn a nástrojů pro jejich dohledání. (2)

4.9 Zajištění kvality softwaru

Proces zajištění kvality dohlíží během životního cyklu softwaru, na dodržení cílů jednotlivých fází životního cyklu. Zajištění kvality by mělo být prováděno nezávisle na ostatních procesech. Tento proces zodpovídá za soulad softwarových plánů a standardů se směrnicemi DO-178C a DO-331. Dohlíží na dodržování těchto plánů a standardů během ostatních fází životního cyklu softwaru. Tento proces dohlíží, aby přechodová kritéria mezi fázemi životního cyklu softwaru, byla dodržena. Potvrzuje, že veškeré fáze a procesy byly řádně dokončeny a jsou kompletní pro certifikační účely. (2)

4.10 Proces spolupráce s certifikační autoritou

Účelem tohoto procesu je zajištění spolupráce s certifikačními autoritami během životního cyklu software. Mělo by být dohodnuto s certifikačními autoritami, jak bude vývojový proces veden, aby bylo zajištěno splnění požadavků pro certifikaci výsledného produktu. Plán pro softwarové aspekty certifikace (*PSAC*) se předkládá certifikačním autoritám společně

s dalšími případnými daty, které jsou vyžadovány. Nastalé problémy by měly být vyřešeny a získán tak souhlas certifikační autority, pro daný plán softwarových aspektů certifikace. (2)

4.11 Certifikace softwaru

V souvislosti s certifikací softwaru se používá spíše pojmu *„schválení“*. Software je součástí systémů a vybavení, pro které musí být schválen, aby byla získána certifikace těchto systémů a vybavení. Software je schválen teprve po demonstraci a přezkoumání a je vždy omezen jen na daný certifikovaný systém. Certifikační úřady stanovují základní předpisy, potřebné pro software, aby byl schválen. V případě splnění konkrétních speciálních podmínek, lze software schválit i mimo předpisy. (2)

4.12 Dodatečné úvahy

V některých případech lze cíle a činnosti fází životního cyklu upravit. Jedná se o případy použití již vyvinutého softwaru, použití nástrojů či použití alternativních metod. (2)

4.12.1 Použití vyvinutého softwaru

Při použití již schváleného softwaru, není potřeba znovu usilovat o schválení, jestliže software splňuje novou úroveň bezpečnosti (*DAL*) a certifikační podmínky. Pro vývoj nové verze softwaru může být použito jiných vývojových prostředí a nástrojů, avšak jejich dopad by měl být zjištěn, analyzován a překontrolován, zejména dopad na výsledný zdrojový kód. Podobné podmínky platí i na změnu hardwaru. (2)

4.12.2 Kvalifikace nástrojů a prostředí

Nástroje a prostředí, které pomáhají při vývoji, tj. automatizují, nebo odstraňují potřebu pro některé činnosti fází životního cyklu softwaru, vyžadují *kvalifikaci*. Cílem *kvalifikace* je zajištění, že používaný nástroj nebo prostředí poskytuje dostatečnou důvěru, ekvivalentní důvěře procesů odstraněných, nebo procesů zautomatizovaných. *Kvalifikace* může být udělena pro jeden nástroj, sadu nástrojů nebo některou funkci nástroje či prostředí, pokud nemůže ovlivnit jinou funkci. *Kvalifikovaný nástroj* lze pak použít jen pro specifický systém, podle plánu pro *softwarové aspekty certifikace* (*PSAC*). Typickým příkladem mohou být nástroje pro automatické generování nebo verifikaci kódu z *Design modelu*, nástroje pro zajištění trasovatelnosti apod. (2)

Při kvalifikaci nástroje nebo prostředí, je přezkoumán dopad jeho použití a určena úroveň podle tří kritérií (2):

Kritérium 1: Nástroj, jehož výstup je součástí softwaru, který bude nahrán na palubě letounu a může tak být přímým zdrojem chyby.

Kritérium 2: Nástroj, který automatizuje verifikační proces a nemusí objevit chybu, a odůvodňuje odstranění nebo omezení verifikačních procesů či vývojových procesů softwaru.

Kritérium 3: Nástroj, který nemusí objevit chybu při použití účelu svého použití.

Na základě těchto kritérií a podle *úrovně softwaru* (*DAL*), pro který je nástroj nebo prostředí použito, je možné určit potřebnou kvalifikaci nástroje tak jak ukazuje **Tabulka 4**. TQL-1 je nejvíce nekompromisní, TQL-5 nejméně. Kvalifikací nástrojů se zabývá směrnice RTCA DO-330 *„Software Tool Qualification Consideration“*. (2)

Tabulka 4: Určení úrovně kvalifikace nástroje převzato z (2)

Úroveň softwaru (Viz. **Tabulka 1**)	Kritéria		
	1	2	3
A	TQL-1	TQL-4	TQL-5
B	TQL-2	TQL-4	TQL-5
C	TQL-3	TQL-5	TQL-5
D	TQL-4	TQL-5	TQL-5

5 Příklad využití metody MBD podle DO-178C a DO-331 – návrh dílčí části SW autopilota

Pro účely této práce, je jako příklad použití metody *MBD* podle směrnic DO-178C a DO-331, vybrána část softwaru autopilota, tzv. *„Altitude Hold Mode“*, jehož funkcí je udržovat stálou výšku letu. Vývoj tohoto softwaru odpovídá vývoji příkladu MB 4, který uvádí **Tabulka 3**. Vývoj podle tohoto příkladu byl vybrán, protože odpovídá častému návrhu mechatronických soustav, a ilustruje výhody metody *MBD*.

První z fází životních cyklů tohoto vývoje softwaru, je určení systémových požadavků na software. Před jejich definicí se doporučuje popsat daný systém nejen pro lepší pochopení ale i soudržnost požadavků a jejich úplnost. (6) Na celém vývojovém cyklu pracují různé týmy, které mají určenou svoji specializaci. Popis systému a systémové požadavky tak napomáhají jednotlivým týmům pochopit a orientovat se v daném systému na jednotlivých vývojových úrovních systému, bez předchozí znalosti problematiky daného systému.

5.1 Autopilot

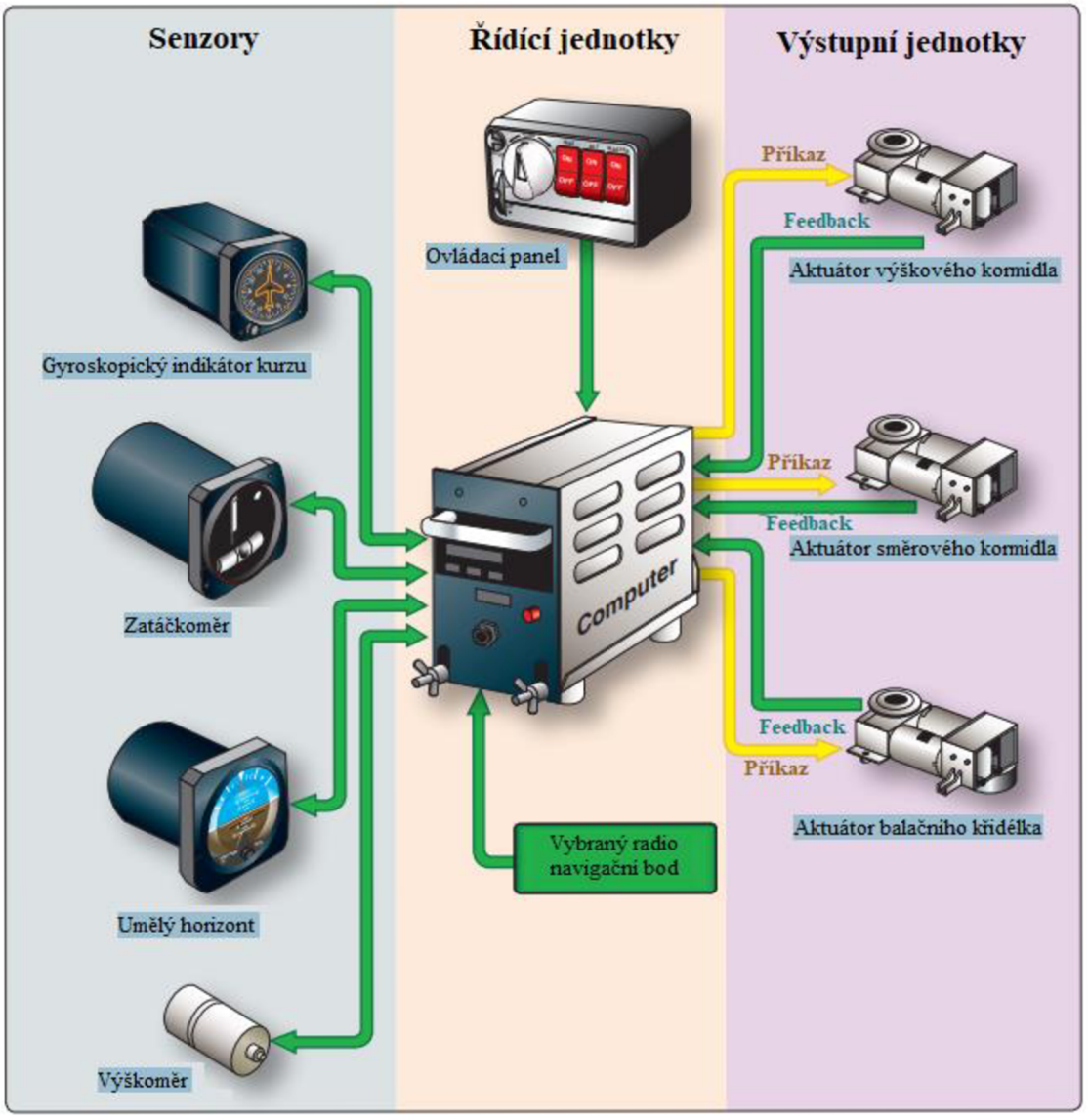
Autopilot je v dnešní době součástí téměř všech letadel. Díky jeho schopnostem ovládat letadlo ať už operativně, nebo na základě vstupů ze systému řízení letu *„Flight managment system“* (*FMS*) podle předem naprogramovaných příkazů, má posádka možnost věnovat pozornost i jiným parametrům chování letadla, než je samotné řízení letadla. Díky tomu má pilot možnost snížit svoji zátěž a udržovat potřebný přehled o celkovém stavu letadla. To je zejména důležité při letech, které trvají delší dobu a pilot při nich může trpět únavou a sníženou pozorností.

Autopilot může mít celou řadu funkcí, od základních jako je udržování kurzu nebo výšky až ke složitějším jako je kontrolované stoupání či klesání. Díky vstupům od systému *FMS* je schopen autopilot řídit letadlo podle navigačních bodů po celou dobu letu, podle předem naprogramované trasy uložené v systému *FMS*. Autopilot společně s dalšími systémy, které letadlo může mít (například *Autothrotle* a *FMS*), je schopný v ideálních podmínkách vykonat celý let bez pomoci posádky. Nutnost lidské posádky však v žádném případě nelze vyloučit, neboť může dojít k poruše, turbulencím nebo jiným neočekávaným situacím, které vyžadují převzetí kontroly posádkou.

5.1.1 Součásti autopilota

Autopilot se skládá z několika součástí. Jejich příklad zobrazuje **Obrázek 4**. Různé typy senzorů slouží k poskytnutí informací o poloze v prostoru a stavu letadla (výška, úhel náklonu atd.). Signály ze senzorů zpracovává výpočetní jednotka, která na základě vstupů od řídící jednotky, dává signály servopohonům. Ty jsou propojeny s ovládacími prvky letadla, které tak řídí. Servopohony jsou zpravidla dva (pro ovládání náklonu letadla pomocí křidélek a výškového kormidla). Může být využito i třetího servopohonu pro ovládání směrového kormidla letadla. Ovládací panel slouží jako rozhraní pro pilota a umožňuje mu

zadávat instrukce pro autopilota. Součásti softwaru autopilota je funkce *„Altitude Hold Mode“*, která slouží k udržování stálé výšky v průběhu letu.



Obrázek 4: Součásti autopilota převzato z **(14)**

5.2 Altitude Hold Mode

Účelem funkce *„Altitude Hold Mode“* (*AHM*) je udržování konstantní výšky letu (barometrické výšky) s patřičnou přesností při zachování bezpečnosti a ovladatelnosti letadla. Výstupní akční veličinou je požadavek na úhel stoupání/klesání, který je dále zpracován na polohu servomotoru připojeného k táhlu ovládajícímu výchylku výškového kormidla. Poloha natočení výškového kormidla následně ovlivní úhel stoupání/klesání letadla.

Základním principem *AHM* je ovlivnit úhel stoupání/klesání, aby došlo k zachování letové výšky.

Použití v těchto režimech letu:

- přímý vyrovnaný let,
- let se změnou kurzu,
- přechod ze stoupání/klesání.

Prostředí ovlivňující funkci:

- teplota,
- tlak,
- vítr.

Interakce s pilotem:

- vizuální upozornění na zapnutí/vypnutí,
- zapnutí manuálně nebo přechodem z jiného modu,
- vypnutí manuálně/přechodem do jiného modu,
- vizuální a zvukové upozornění na nefunkčnost (nefunkční zařízení, nemožnost udržet letovou hladinu za daných podmínek).

Systémová omezení:

- omezení maximálního úhlu stoupání/klesání letadla (zamezení odtržení proudu/vysoké rychlosti klesání),
- vypnutí nebo upozornění pilota při nemožnosti udržet zadanou výšku,
- vypnutí při nesprávných, nebo chybějících datech ze senzorů.

Vstupy do funkce:

- zapnutí/vypnutí,
- skutečná výška,
- úhel náběhu *„Angle of Attack“* (*AoA*),

- Pravá vzdušná rychlost „*True Air Speed*“ (*TAS*), [14]
- signál *SSM* ze sběrnice *ARINC 429*[15]pro skutečnou výšku,
- signál *SSM* ze sběrnice *ARINC 429* pro úhel náběhu,
- signál *SSM* ze sběrnice *ARINC 429* pro pravou vzdušnou rychlost.

Výstupy funkce:

- digitální signál pro autopilot o zapnutí/vypnutí modu,
- digitální signál upozorňující na chybu zařízení,
- požadavek na úhel stoupání/klesání.

5.2.1 Koncept návrhu funkce AHM

Aktivování funkce *AHM* bude softwarově prováděno autopilotem. Autopilot aktivuje *AHM* po zmáčknutí příslušného tlačítka, umístěném na ovládacím panelu, pilotem, nebo pokud aktivace *AHM* byla předem naprogramována na systému řízení letu a splňuje podmínky pro aktivaci. Deaktivování je provedeno obdobně, po zmáčknutí příslušného tlačítka pilotem, nebo pokud je vyžadován jiný předem naprogramovaný mód, se kterým by byla funkce *AHM* v konfliktu, či v případě chybných vstupních údajů v softwaru. Funkce *AHM* poskytuje autopilotovi informaci o svém stavu. Na základě tohoto stavu dává autopilot pokyn primárnímu letovému displeji *„Primary Flight Fisplay“* (*PFD*) pro indikaci stavu *AHM*.

Skutečná a požadované barometrická výška bude pomocí *PID regulátoru*[16] určovat vertikální úhel dráhy letu *„Flight Path Angle“* (*FPA*). K té bude přičten úhel náběhu (*AoA*) pro výstup úhlu stoupání/klesání letadla. Maximální a minimální hodnota úhlu stoupání/klesání bude omezena. Rychlost růstu a klesání bude pomocí hodnot, vypočtených z pravé vzdušné rychlosti, omezena tak, aby na letadlo nepůsobilo zrychlení větší jako 0,1 G (tíhové zrychlení). Tyto maximální a minimální hodnoty jsou spočítány pomocí následujícího vztahu:

$$d\theta = \frac{g}{v_{TAS}}(n-1) \tag{1}$$

Kde $d\theta$ je rychlost růstu úhlu stoupání/klesání, g je gravitační zrychlení (9,81 m/s^2), v_{TAS} je skutečná rychlost letu a $(n-1)$ je tzv. load faktor, který je roven 0,1 G.

Úhel stoupání/klesání je pak dále zpracován subsystémem autopilota, který udává pomocí úhlu stoupání/klesání (*Pitch controller*, není součástí modelu AHM), požadavek na polohu servopohonu ovládající výškové kormidlo.

---

[14] Rychlost letadla vůči okolnímu vzduchu

[15] ARINC 429 je standardizovaná datová směrnice o 32 bitech. *SSM – Sign/Status Matrix* je druhý a třetí bit v této směrnici, a udává, zda jsou poslaná data správná.

[16]Regulátor, který využívá proporcionální, integrační, a derivační složky vstupního signálu, nebo kombinace těchto složek.

Autopilot má poskytnut kontrolní *SSM* signál ze sběrnice *ARINC 429* příslušící vstupním datům skutečné výšky letu a *AoA*. Pokud tento signál bude potvrzovat správná data na vstupu, bude funkce *AHM* aktivní. V opačném případě je funkce *AHM* deaktivována.

Funkce *AHM* bude kontrolovat, zda vstupní hodnoty výšek nepřesahují předpokládané hodnoty. Přesah hodnot mimo předpokládaný interval by mohl být způsoben vadným nebo poškozeným senzorem. Pokud tato chyba není nárazová a přetrvává (např. více jak 10 ms) je funkce *AHM* vypnuta.

AHM bude udržovat výšku, ve které byla provedena aktivace autopilotem. Snaží se tuto výšku udržet s přesností ± 25 m. V případě, že není funkce *AHM* schopna udržet požadovanou výšku letu v tomto patřičném rozmezí, je posádka upozorněna zvukovým anebo i vizuálním výstupem, dokud není funkce *AHM* vypnuta, nebo znovu schopna správně vykonávat svoji funkci.

V **Příloze 2** je ukázána dílčí část autopilota odpovídající *Altitude Hold Mode*.

5.3 Systémové požadavky pro Altitude Hold Mode

Při psaní systémových požadavků je třeba dodržovat standardy určené během fáze životního cyklu, plánování pro software – zejména standardy pro psaní požadavků. Tento standard určuje metody, pravidla a nástroje pro vývoj systémových požadavků. Následující podkapitola slouží jako příklad dílčí části takového standardu a uvádí zásady pro definici požadavků.

5.3.1 Standard psaní systémových požadavků:

Systémové požadavky musí být psány následujícím způsobem:

- určený požadavek má tři části – název, popis a zdůvodnění pro tento požadavek,
- jsou psány písmem Times New Roman velikosti 12,
- název požadavku a popis je zvýrazněn tučně,
- jednotlivé části názvu jsou oddělené podtržítkem; název požadavku začíná zkratkou *„SYS“* značící že se jedná o systémový požadavek; poté následuje pořadové číslo požadavku a končí klíčovým slovem, nebo slovy bez mezer v anglickém jazyce s velkými počátečními písmeny jednotlivých slov,
- popis požadavku je psán, pokud možno co nejstručněji, s použitím pozitivního přístupu (vyvarovat se slovům jako nemůže, nesmí atd.),
- pro zachování přehlednosti, název popis a důvod požadavku je vždy společně na jedné stránce,
- pro zjednodušení je pro funkci *„Altitude Hold Mode“* použita zkratka *AHM*.

5.3.2 Systémové požadavky:

SYS_01__StatusIndicator

Popis: Při aktivním *AHM* je na primárním letovém displeji (*PFD*) zobrazen text *„ALT“*.

Důvod: Zobrazení *ALT* na komunikačním rozhraní autopilota potvrzuje funkční a aktivní *AHM* pilotovi.

SYS_02_ManualActivation

Popis: Po zmáčknutí tlačítka *ALT* pilotem, autopilot aktivuje *AHM*.

Důvod: Umožní posádce aktivovat *AHM* manuálně kdykoliv během letu po zmáčknutí tlačítka.

SYS_03_FDActivation

Popis: Po přechodu z jiného modu pro klesání či stoupání letadla, při dosažení zadané výšky je *AHM* aktivován autopilotem.

Důvod: Umožnění zapnutí *AHM* autopilotem pro přechod do tohoto modu z jiného módu, který mění nebo udržuje výšku letu, při dosažení předem zadané výšky letu.

SYS_04_ManualDeactivation

Popis: Jestliže je *AHM* aktivní, při zmáčknutí tlačítka *ALT* pilotem autopilot *AHM* deaktivuje.

Důvod: V případě že je *AHM* aktivní umožňuje pilotovi dát signál autopilotovi pro vypnutí tohoto modu.

SYS_05_FDDeactivation

Popis: Pokud je vyžadován jiný mód, který mění výšku letu, je autopilotem *AHM* deaktivován.

Důvod: Umožnění vypnutí *AHM* autopilotem, pokud je po něm vyžadován jiný mód, který mění výšku letu, jako například konstantní stoupání/klesání.

SYS_06_AHMFunction

Popis: *AHM* udržuje letadlo ve stálé letové výšce s přesností na ±25 m.

Důvod: Hlavní funkce *AHM* je udržovat letadlo v dané výšce s patřičnou mezí přesnosti

SYS_07_TargetAltitude

Popis: *AHM* udržuje výšku letu, ve které letělo letadlo při aktivaci *AHM*.

Důvod: Při manuálním zapnutí, nebo při přechodu z jiného modu, má *AHM* za úkol udržovat výšku ve které byl aktivován.

SYS_08_MaxFPAAngle

Popis: Maximální hodnota pro úhel stoupání letadla je 10°.

Důvod: Omezení úhlu stoupání letadla pro zabránění případné ztrátě vztlaku (*„stalling“*) letadla.

SYS_09_MinFPAAngle

Popis: Maximální hodnota pro úhel klesání letadla je 7°.

Důvod: Omezení úhlu klesání letadla, pro zabránění příliš rychlé ztrátě výšky, která by mohla být fatální.

SYS_10_MAXG

Popis: Maximální dovolené normálové zrychlení letadla je 0,1 G.

Důvod: Zabránění příliš prudkým pohybům letadla, které by mohlo vést k fyzické zátěži, nebo nepohodlí pro posádku a pasažéry.

SYS_11_AHMWarningSound

Popis: Při rozdílu mezi požadovanou a skutečnou výškou letu větším jak 25 m, je pilot varován zvukovou výstrahou.

Důvod: V situaci, kdy *AHM* není schopný udržet výšku letu, je pilot informován zvukovým signálem o této situaci. To je provedeno, aby pilot vyhodnotil situaci a pokud uzná, že *AHM* si s ní není schopný poradit a nastalo by ohrožení letadla či posádky, převzal kontrolu nad letadlem.

SYS_12_Inputs

Popis: Vstupními daty do *AHM* jsou:

- zapnutí/vypnutí *AHM,*
- skutečná barometrická výška,
- úhel náběhu (*AoA*),
- Pravá vzdušná rychlost (TAS)
- signál *SSM* ze sběrnice *ARINC 429* příslušící barometrické výšce,
- signál *SSM* ze sběrnice *ARINC 429* příslušící *AoA*.
- signál *SSM* ze sběrnice *ARINC 429* příslušící *TAS*

Důvod: Vstupní data pro *AHM*, která zajišťují správnou funkci *AHM*.

SYS_13_Outputs

Popis: Výstupními daty *AHM* jsou:

- signál pro primární letový displej, indikující zapnutí modu,
- vizuální a zvukové upozornění na chybu zařízení,
- pitch úhel

Důvod: Výstupní data *AHM*, zajišťující řízení letadla a indikaci jeho funkčnosti či případné nefunkčnosti.

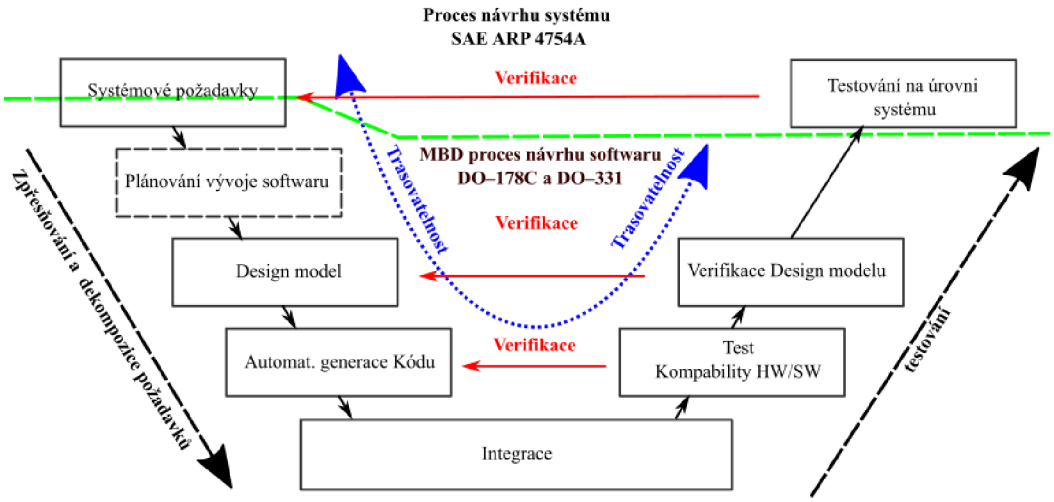
SYS_14_WrongInputs

Popis: V případě neplatných, nebo při nedostupných datech je *AHM* deaktivován.

Důvod: Pro zabránění nežádoucích účinků *AHM*, které by mohly nastat v případě neplatných, chybějících nebo neúplných dat. K tomuto účelu slouží i datové sběrnice *ARINC 429*.

5.4 Proces vývoje Altitude Hold Mode podle směrnic DO-178C a DO -331

Během hodnocení bezpečnosti systému by byla určena úroveň zajištění vývojového procesu softwaru. V případě funkce *AHM* by chyba v této funkci vedla k zvýšení zátěže posádky a snížení její efektivity. Tato chyba neohrožuje život posádky či pasažérů, ale může způsobit fyzické nepohodlí či zranění. Na základě tohoto hodnocení je pro *Altitude hold mode* určena úroveň softwaru *DAL: C*. V-model pro tento vývoj zobrazuje **Obrázek 5.** Lze si na něm povšimnout, že zelená čára, oddělující vývoj podle SAE ARP 4754A a DO-178C/DO-331, prochází systémovými požadavky. To je z důvodů, že ze systémových požadavků je tvořen *Design model.* Na systémové požadavky se tak vztahují jak *SAE ARP 4754A*, tak *DO-178C* a *DO-331*.**Tabulka 5** shrnuje cíle, které mají být dosaženy pro software úrovně *DAL C,* jak bude vývoj *AHM* odlišný při použití metod *MBD.*



Obrázek 5: V-model pro vývoj *AHM*

Tabulka 5: Specifika pro AHM v DAL C a využití MBD

DO-178C/331 proces	Specifika vývoje AHM s hodnocením DAL C	Využití a specifika MBD
Plánování softwaru	Všechny cíle tohoto procesu musí být naplněny.	Součástí plánů *PSAC, SDP, SVP, SCMP* a *SQAP* bude popis způsobů zapojení metody *MBD* do vývoje a verifikace *AHM*. Specifikem pro metodu *MBD* je nutnost určení standardu modelu a jeho prvků, kde bude popsáno využití prostředí *Simulink* pro vytvoření *Desing modelu*. Standard pro softwarové požadavky bude určovat způsob zápisu systémových požadavků, protože se jedná o požadavky ze kterých je model vyvinut a přebírají tak roli high-level požadavků.
Vývoj softwaru	Všechny cíle tohoto procesu musí být naplněny.	Protože systémové požadavky budou použity jako požadavky, ze kterých je model vyvinut, musí odpovídat standardům pro softwarové požadavky. *Low-level* požadavky jsou nahrazeny *Design modelem*, který musí odpovídat standardům modelu. Zdrojový kód bude generován nástrojem kvalifikovaným podle DO-330 *na TQL-5*.

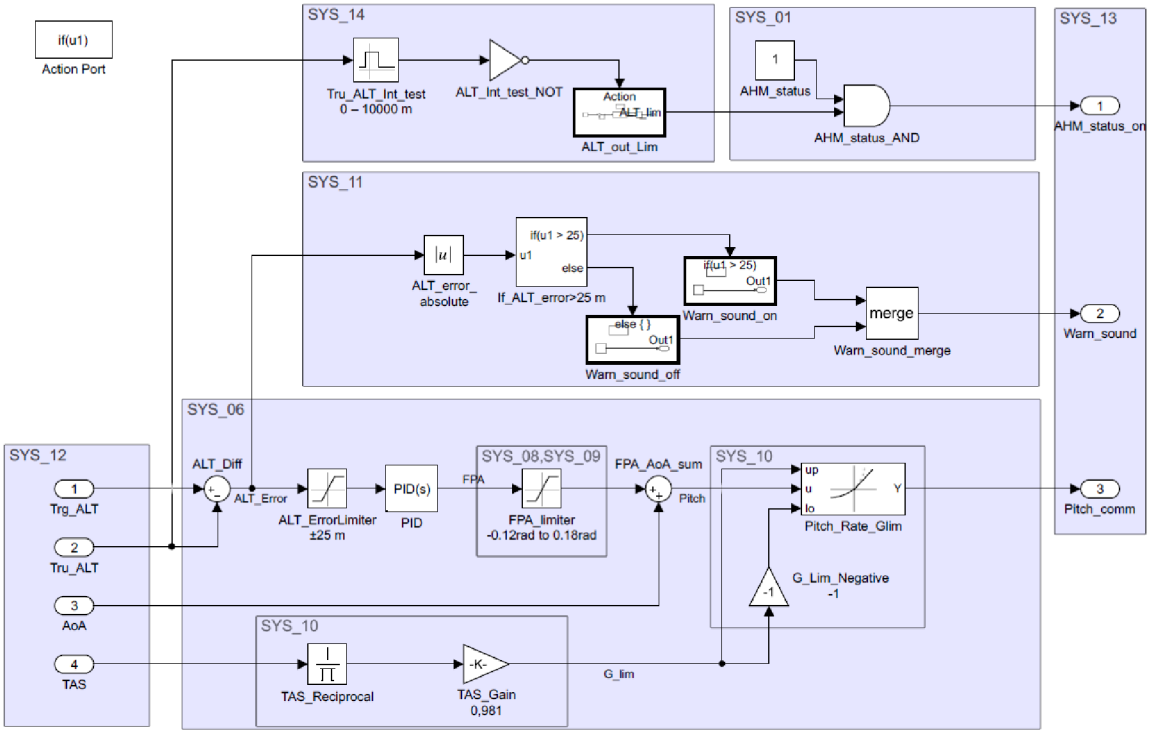
Tabulka 5 - pokračování

DO-178C/331 proces	Specifika pro AHM v DAL C	Využití a specifika MBD
Verifikace výstupů procesu určení požadavků pro software	Pro *DAL C* není nutné splnit cíl 3: *high-level* požadavky jsou kompatibilní s cílovým výpočetním zařízením.	Vytvářením modelu ze systémových požadavků, tyto požadavky přebírají roli *high-level* požadavků. Systémové požadavky tak musí splňovat cíle i pro *high-level* požadavky. Pro ověření některých cílů na *high-level* požadavky bude použito simulace. Je pak třeba ověřit že vstupy, procedury a výsledky jsou správné.
Verifikace výstupů návrhu softwaru	Pro *DAL C* není nutné splnit cíle: 3 – *low-level* požadavky jsou kompatibilní s cílovým výpočetním zařízením. (tj. není potřeba ověřit, jestli není zdrojový kód v konfliktu s prvky hardware jako jsou sběrnice, *I/O* atd.). 4 – *low-level* požadavky jsou ověřitelné. 10 – Softwarová architektura je kompatibilní s cílovým výpočetním zařízením (tj. není třeba ověřit že nejsou konflikty v synchronizaci, asynchronní operaci, přerušení atd.). 11 – Softwarová architektura je ověřena (např. ověření, zda se nevyskytují neomezené rekurzivní algoritmů)	Pro splnění některých cílů tohoto procesu bude opět použita simulace. Simulace pro tento proces a předchozí proces bude společná. Lze tak s ní dosáhnout cílů obou procesů.

Tabulka 5 - pokračování

DO-178C/331 proces	Specifika pro AHM v DAL C	Využití a specifika MBD
Verifikace výstupů kódování a integrace	Pro *DAL C* není nutné splnit cíl 3: zdrojový kód je ověřitelný. (není třeba ověřovat správnost příkazů a struktur zdrojového kódu).	I při použití automatické generace kódu z *Design modelu*, je potřeba prověřit zdrojový kód zda splnit cíle tohoto procesu.
Testování výstupů integračního procesu	Všechny cíle tohoto procesu musí být naplněny.	Při použití metody *MBD* je třeba zajistit, že je použit stejný *Design model* pro simulaci a vytvoření zdrojového nebo objektového kódu.
Ověření Výsledků verifikačního procesu.	Pro *DAL C* není nutné splnit cíle: 5 – Testového pokrytí softwarové struktury (modifikované pokrytí podmínek/rozhodnutí) je dosaženo (tj. není třeba testovat software na všechny možné kombinace vstupních podmínek a nemusí být dosaženo všech výstupů a vstupů). 6 – Testového pokrytí softwarové struktury (pokrytí rozhodnutí) je dosaženo (tj. nemusí být dosaženo všech výstupů a vstupů). 9 – Verifikace doplňkového kódu, který není trasovatelný ke zdrojovému kódu. (tj. pro účely testování může být vytvořen ve zdrojovém kódu nový kód. Tento kód nemusí být ověřen.).	Pro splnění některých cílů bude opět použita simulace, pro kterou musí být použit zdrojový kód, ze kterého je vytvořen výsledný objektový kód.
Určení konfigurace softwaru	Všechny cíle tohoto procesu musí být naplněny.	Pro zajištění reprodukce dat, je potřeba archivovat použité nastavení prostředí a metod pro metodu *MBD* použitých.
Zajištění kvality softwaru	Všechny cíle tohoto procesu musí být naplněny s nezávislostí.	Při použití metody *MBD* musí být zajištěno vyvinutí standardů a plánů v souladu s DO-331
Schválení softwaru	Všechny cíle tohoto procesu musí být naplněny.	Bez konkrétního specifika

5.5 Design model Altitude hold modu



Obrázek 6: Kontrolní logika AHM autopilota

Obrázek 6 vyobrazuje funkce AHM autopilota v prostředí *Simulink*. Jedná se o kontrolní funkci zapnutého AHM.

Vstupními proměnnými je cílová výška (**TRG_Alt**), kterou se AHM snaží udržet, Aktuální barometrická výška (**Tru_ALT**) a úhel náběhu (**AoA**). Pomocí bloku **ALT_Diff** je spočítán rozdíl mezi cílovou výškou a skutečnou barometrickou výškou. Tento rozdíl (**ALT_Error**) je omezen do hodnot od -25 m do +25 m blokem **ALT_ErrorLimter**. Pomocí PID regulátoru (**PID**) je z omezeného rozdílu výšek spočítán požadovaný úhel dráhy letu (**FPA).** Ten je omezen na hodnotu -0.12 radiánů (-7°) do 0.18 radiánů (10°) pomocí **FPA_limiter**. Pro získání požadovaného úhlu stoupání/klesání (**Pitch**) je k **FPA** přičten **AoA**. Úhel stoupání/klesání je pak přes omezení rychlosti klesání/stoupání **Pitch_Rate_Glim** omezen tak, aby zrychlení letadla odpovídalo 0,1G. Výstup z **Pitch_Rate_Glim** je přiveden na **Pitch_comm** a předán do *Pitch controlleru*.

Hodnoty pomocí kterých je **Pitch** omezen, jsou spočítány pomocí bloků **TAS_Reciprocal, TAS_Gain** na hodnotu odpovídající zrychlení 0,1G. Ta je pak přivedena jako horní mez na vstup **up** bloku **Pitch_Rate_Glim**. Na dolní mez **Io** je přivedena záporná hodnota spočítána pomocí bloku **G_Lim_Negative**.

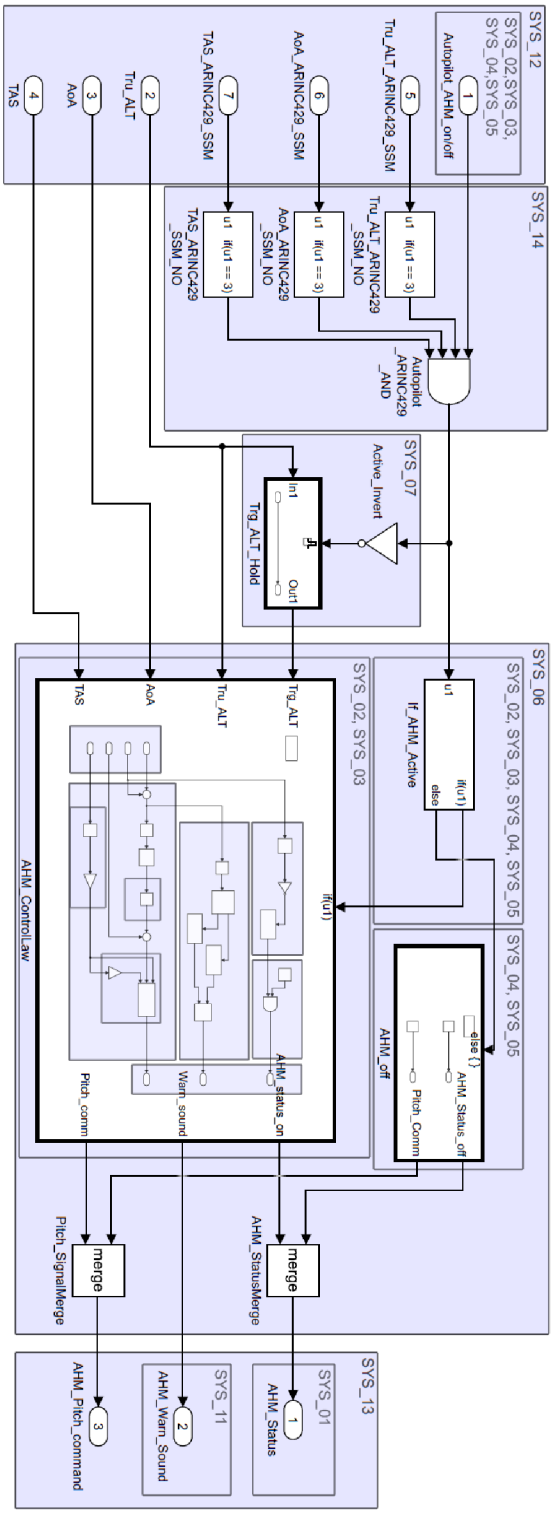
Rozdíl skutečné a požadované výšky **ALT_Error** je použit i pro varování pilota o oddálení skutečné výšky letadla od požadované výšky o 25 m. Pomocí bloku **ALT_error_absolute** je

určena absolutní hodnota **ALT_error**. V případě že je větší jak 25 m (**If_ALT_error>25 m)** je aktivní if-subsytém **Warn_sound_on**, jehož výstupem je logická 1. V případě že je rozdíl skutečné a požadované výšky letadla menší jak 25 m, je aktivní else-subsystém **Warn_sound_off**, jehož výstup je logická 0. Výstupy if a else subsystémů, jsou spojeny pomocí merge bloku (**Warn_sound_merge**), který na výstupu má výstup právě aktivního subsystému. Výstupem je pak logická 1 nebo 0 na **Warn_sound**, která udává zda má být zapnut varující signál autopilotem.

Skutečná barometrická výška letu je pomocí bloku **Tru_ALT_Int_test** kontrolována, zda je v rozmezí 0 až 10 000 m. V případě chybného měření senzorem výšky, může dojít k vytvoření nesmyslných údajů, které jsou mimo kontrolované rozmezí. Výstupem **Tru_ALT_Int_test** je logická 0, jestliže je vstupní hodnota mimo kontrolované rozmezí. Tato hodnota je převrácena blokem **NOT_ALT_Int_test** do logické 1, při které je aktivní subsystém **ALT_out_Lim**. Subsystém **ALT_out_Lim** pak počítá při určité frekvenci dobu po kterou je skutečná výška letu mimo rozmezí 0 až 10 000 m. Pokud tato doba nepřesáhne stanovený čas (např. 10 ms) je na výstupu **ALT_out_Lim** logická 1. Pomocí logického hradla AND (**AHM_Status_AND)**, kde je přivedena logická hodnota z **ALT_out_Lim** a logická 1 z **AHM_Status_on,** je určeno na výstupu **ALT_status_on** zda je AHM aktivní, nebo ne pomocí logické 1 a 0.

Trasovatelnost *Design modelu* na systémové požadavky je pro názornost docílena použitím fialových obdélníků. Tyto obdélníky mají v levém horním rohu udáno, ke kterým systémovým požadavkům se vážou. Bloky uzavřené v bledě modrých obdélnících naplňují systémové požadavky jim přiřazené. Běžnou praxí je vytvoření hyper odkazů mezi jednotlivými systémovými požadavky a bloky, které tyto systémové požadavky naplňují.

Na **Obrázku 7** je pak ukázána další kontrolní logika *AHM*, která je mimo řídící funkci. Pomocí bloků **ARINC429** je přivedena hodnota *SSM*, která pokud neodpovídá hodnotě správných dat, vypne kontrolní logiku **AHM_ControlLaw.** Díky tomu, že při zapnuté funkci *AHM* je přes **Active_Invert** vypnut subsystém **Trg_ALT_Hold**. Jeho výstupem je tak poslední hodnota výšky, ve kterou byl *AHM* aktivován. Tato hodnota pak slouží jako referenční výška, kterou se **AHM_ControlLaw** snaží udržet.



Obrázek 7: Funkce AHM v Autopilotovi

6 Závěr

Bezpečnost v leteckého provozu je prioritou, která musí být vždy na prvním místě. Vývoj spolehlivých leteckých systémů a softwaru má řadu úskalí. K jejich překonání napomáhají směrnice, vytvořené certifikačními autoritami a zástupci leteckých technologických a průmyslových firem. Řada směrnic slouží k certifikaci softwaru, který má být součástí leteckých systémů. Pro vývoj leteckých systémů je využívána směrnice *ARP 4754A*. Pro vývoj softwaru těchto systémů slouží směrnic *DO-178C* doplněná směrnicí DO-331, v případě využití metod *MBD*. Každá z nich definuje cíle, kterých musí být dosaženo během činností a procesů životního cyklu softwaru, pro zajištění nezbytných podmínek pro certifikaci systému a schválení softwaru tohoto systému.

U vyvíjeného systému a jeho softwaru je na základě analýzy bezpečnosti na případ výskytu chyby v systému nebo softwaru, určena úroveň zajištění vývojového procesu (*DAL)* v kategoriích A až E. Některé nižší bezpečnostní úrovně systému a softwaru dovolují vynechat určité činnosti a cíle fází životního cyklu systému a jeho softwaru. Jiné, zpravidla nejvyšší úrovně spolehlivosti, zase kladou důraz na provedení činností a dosažení cílů s nezávislostí na ostatních činnostech fází životního cyklu systému nebo softwaru. Typicky je tato nezávislost vyžadována u úrovní spolehlivosti kategorie A.

Vývoj softwaru pomocí metody *MBD* dovoluje nahrazení některých činností fází životního cyklu a cílů jinými, více vyhovujícími činnostmi a cíli. Metodiku *MBD* lze použít na celý vývojový cyklus softwaru, nebo pouze na některé jeho fáze.

Směrnice *DO-178C* a dodatek *DO-331* dovolují použití i převzatých nástrojů, které se stanou součástí výsledného softwaru, nebo automatizují některé procesy životního cyklu softwaru. U vedené nástroje je však potřeba kvalifikovat na požadovanou úroveň podle stanovených kritérií a úrovně spolehlivosti softwaru, pro který je nástroj použit.

Cílem stanoveným zadáním této práce bylo provést literární rešerši standardů, specifikací a požadavků s nasazením metod *MBD* pro vývoj leteckých soustav a software. Toto bylo provedeno v teoretické části práce, popsáním fází a procesů životního cyklu vývoje leteckých systémů a softwaru společně s jejich cíli. Popsané směrnice neslouží ke stanovení jak má vypadat vývoj těchto systémů konkrétně. Určují, jakých cílů by jednotlivé procesy měli dosáhnout, pro splnění základních požadavků pro certifikaci systémů a softwarů. Zároveň napomáhají uživatelům s aspekty, se kterými se mohou během životního cyklu software setkat. Současně byl v teoretické části prezentován popis možných způsobů vývoje leteckých systémů při využití metod *MBD*.

Jako příklad, na kterém byl jeden ze způsobů využití MBD, byla vybrána a aplikována na funkci *„Altitude Hold Mode“* (*AHM*) leteckého systému *„Autopilot“*. Na příkladu bylo v **Kapitole 5.4** určeno, jaké cíle by byly vyžadovány během jednotlivých fází životního cyklu softwaru a jak by tyto fáze byly ovlivněny při použití metody *MBD*. V **Kapitole 5.5** je pak ukázán možný výstup vývojových procesů na úrovni *Design modelu AHM* v prostředí *Simulink*, který je doplněn o kontextový *Design model AHM* jako součásti softwaru autopilota.

Vývoj systémů a softwaru podle výše zmíněných směrnic usnadňuje splnění základních podmínek pro svoji certifikaci. Letecké systémy vyžadují jednu z největších spolehlivostí mezi technickými systémy. Dodržením cílů a doporučení zmíněných směrnic má za následek vyvinutí systémů a softwarů s touto požadovanou spolehlivostí.

Použitím metod *MBD* lze dosáhnout většího přehledu, během vývoje, o vyvíjeném systému nebo softwaru. Použitím modelů a automatického generování kódu z těchto modelů může být značně snížena časová náročnost vývoje. Díky možnosti simulací modelů, lze odhalit chyby mnohem dříve, než při běžném vývoji systému nebo softwaru. Tato skutečnost umožňuje vytvářet systémy a software finančně a časově úsporněji. Tento vývoj je v práci ilustrován a dovolil vytvořit software systému bez profesionální znalosti tohoto systému.

Na základě výsledků této práce lze říct, že obsah směrnic, které byly v práci sledovány, dodržení postupů při vývoji a kvalifikaci systémů požadovaných certifikačními autoritami a používání vývojových prostředí a jejich nástrojů, vede k zvýšení efektivity vytváření spolehlivého softwaru a tím i spolehlivost celých leteckých systémů

7 Seznam zdrojů

1. **Mathworks.** Simulink from Mathworks. *Mathworks.* [Online] [Citace: 13. Květen 2021.] https://www.mathworks.com/products/simulink.html.

2. **SC–205, RTCA.** DO–178C. *Software Consideration in Aribrne Systems and Equipment Certification.* s.l. : RTCA, Inc, 2011.

3. —. DO–331. *Model–Based Development and Verification Supplement to DO–178C and DO–278A.* s.l. : RTCA, Inc, 2011.

4. **Esterel Technologies.** *Efficien Avionics Systems Engineering with ARP – 4754A Objectives Using SCADE System.* [Online Dokument] Červen 2013. Methodology Handbook.

5. **FAA.** AC 20 – 115D. [Online] 21. Červenec 2017. [Citace: 27. Duben 2021.] https://www.faa.gov/documentLibrary/media/Advisory_Circular/AC_20-115D.pdf.

6. —. Requirements Engineering Management Handbook. *Design Approvals.* [Online] Červen 2009. [Cited: Duben 15, 2021.] https://www.faa.gov/aircraft/air_cert/design_approvals/air_software/media/ar-08-32.pdf.

7. **Lennon, Tony, The MathWorks Inc. and Natick, Mass.** Model–based design for mechatronics systems. *MachineDesign.* [Online] listopad 21, 2007. [Cited: březen 24, 2021.] https://www.machinedesign.com/archive/article/21812097/modelbased-design-for-mechatronics-systems.

8. **FAA.** AC 25.1329–1C Approval of Flight Guidance Systems. *Advisory Circular.* [Online] 10 27, 2014. [Cited: Duben 6, 2021.] https://www.faa.gov/documentLibrary/media/Advisory_Circular/AC_25_1329-1C.pdf.

9. —. Automated Flight Control. *Advanced Avionics Handbook.* [Online] [Cited: Duben 2, 2021.] https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/advanced_avionics_handbook/media/aah_ch04.pdf.

10. **MAIXNER, Ladislav.** *Mechatronika.* 1. vyd. Brno : Computer Press, 2006.

11. **MathWorks.** Model–Based Design for Embedded Control Systems. [Online] [Cited: Březen 24, 2021.] https://www.mathworks.com/content/dam/mathworks/white-paper/gated/model-based-design-with-simulation-white-paper.pdf.

12. **Sapir, Jacob.** A New Approach to Embedded Systems: Model–Based Design. *REAL TECHNOLOGY TOOLS.* [Online] [Cited: březen 26, 2021.] https://realtechnologytools.com/model-based-design/.

13. **Kanardia d.o.o.** Autopilot – Installation manual. *kanardia.* [Online] Únor 2020. [Cited: Duben 24, 2021.] https://www.kanardia.eu/wp-content/uploads/2020/03/Autopilot-Installation-Manual.pdf.

14. **FAA.** Aviation Maintenance Technician Handbook–Airframe, Volume 2. *Handbooks manuals.* [Online] 2015. [Cited: Duben 16, 2021.]

https://www.faa.gov/regulations_policies/handbooks_manuals/aircraft/media/amt_airframe_hb_vol_2.pdf.

15. **MathWorks.** Model–Based Design for DO–178C Software Development with MathWorks Tools. *MathWorks videos series.* [Online] [Citace: 12. únor 2021.] https://www.mathworks.com/videos/series/model-based-design-for-do-178c-software-development-using-matlab-and-simulink-95160.html.

16. **Spitzer, Carry R.** *Digital avionics handbook: avionics.* Boca Raton : CRC Press, 2007. Sv. 1. ISBN 0-8493-8441-9.

17. **MathWorks.** Using Qualified Tools in a DO–178C DEvelopment Process. *Video series.* [Online] [Citace: 28. březen 2021.]

18. **Eisemann, Ulrich.** Applying Model-Based Techniques for Aerospace Projects in Accordance with DO178C, DO-331, and DO-333. *HAL.* [Online] Leden 2016. [Cited: Duben 5, 2021.] https://hal.archives-ouvertes.fr/hal-01291337/document.

19. **Esterline Control Systems.** *Guidelines for Development of Civil Aircraft and Systems.* [Online Prezentace] Červen 23, 2014. introduction to ARP4754A.

20. **FAA.** SAE ARP 4754A Linkage with DO – 178 and DO – 254. [Online] 2011. [Citace: 5. Květen 2021.] http://www.atlanticcertgroup.com/strasburger_sae_arp_4754a_linkage_with_do178_and_do254_.pdf.

21. **Nishadha.** Use Case Diagram Tutorial (Guide with Examples). [Online] 21. Prosinec 2020. [Citace: 28. Duben 2021.] https://creately.com/blog/diagrams/use-case-diagram-tutorial/.

Seznam příloh

Příloha 1: Použité prvky Simulinku (2 s)

Příloha 2: Model AHM v prostředí Simulinku (datový soubor .sxl)

Příloha 1: Použité prvky Simulinku

Konstanta – Blok s konstantou, který vytváří signál s konstantní hodnotou. Může se jednat o číslo, vektor nebo matici hodnot stejného datového typu (boolean, int8, float, atd)

Vstup – Vstupní signál z vnějšího systému do aktuálního systému

Výstup – Výstupní signál z aktuálního systému do vnějšího systému

Součet – Sečte vstupní čísla, může mít více vstupů. Vstup může být i záporný.

Násobek – vynásobí vstupní signál konstantou.

Převrácená hodnota – vytvoří převrácenou hodnotu vstupního signálu.

Absolutní hodnota – Převede hodnotu vstupního signálu na jeho absolutní hodnotu.

Logické hradlo AND – Pokud na všech vstupních hodnotách jsou logické 1, pak výstupem je logická hodnota 1, pokud ne výstupem je logická hodnota 0.

Logické hradlo NOT – Převrátí logickou vstupní hodnotu (boolean).

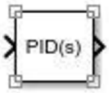
Intervalový test – kontroluje zda vstupní signál se nachází v zadaných mezích. Pokud ano výstupem je logická 1, pokud ne výstupem je logická 0-

If – V případě, že vstupní hodnota u1 splňuje podmínku if(u1), bude vykonán subsystém připojený na tento výstup. Jinak bude vykonán subsystém připojený na výstup else.

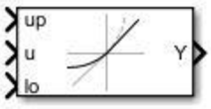
Akční subsystém – subsystém, jehož vykonání je podmíněno vstupem Action. Vstup Action může být připojen na If blok, nebo jiný blok jehož výstupem je logický výraz.

Sloučení – Sloučí vstupní signály do jednoho signálu. Výstupní signál je signál aktuálně vykonávaného bloku. Vstupy jsou často spojeny s if a else akčními subsystémy.

Saturace – omezuje maximální a minimální hodnotu vstupního signálu



PID regulátor – Regulátor spojený z proporcionální, integrační a derivační části vstupního signálu. (může mít i pouze některé složky, nebo jejich kombinaci)



Dynamický omezovač rychlosti – omezí rychlost klesání/stoupání vstupního signálu **u**, na základě signálů na vstupech **up** (pro horní mez) a **lo** (pro dolní mez).