UNIVERSIDADE DO VALE DO RIO DOS SINOS - UNISINOS
UNIDADE ACADÊMICA DE PESQUSA E PÓS-GRADUAÇÃO
ESPECIALIZAÇÃO EM ENGENHARIA DE SOFTWARE

Gustavo Giroletti Mottin

IMPACTO DO DESENVOLVIMENTO BASEADO EM MODELOS PARA PRODUÇÃO E CERTIFICAÇÃO DE SOFTWARE AVIÔNICO

Porto Alegre 13 de Julho de 2018

UNIVERSIDADE DO VALE DO RIO DOS SINOS - UNISINOS
UNIDADE ACADÊMICA DE PESQUSA E PÓS-GRADUAÇÃO
ESPECIALIZAÇÃO EM ENGENHARIA DE SOFTWARE

Gustavo Giroletti Mottin

IMPACTO DO DESENVOLVIMENTO BASEADO EM MODELOS PARA PRODUÇÃO E CERTIFICAÇÃO DE SOFTWARE AVIÔNICO

Trabalho de Conclusão de Curso apresentado como requisito parcial para a obtenção do título de Especialista em Engenharia de Software, pelo curso de Pós-Graduação Lato Sensu em Engenharia de Software da Universidade do Vale do Rio dos Sinos - UNISINOS.

Orientadora: Profa. Me. Josiane Brietzke Porto

Porto Alegre 13 de Julho de 2018

Impacto do desenvolvimento baseado em modelos para produção e certificação de software aviônico

Gustavo Giroletti Mottin[1]

[1]Unidade Acadêmica de Pesquisa e Pós-Graduação
Universidade do Vale do Rio dos Sinos (UNISINOS) – São Leopoldo – RS – Brasil

gustavogirolettimottin@gmail.com

Abstract. *The development of avionics software usually follows guidelines defined by the industry which establishes objectives and activites that are critical to enable it to be operated. The DO-331 is one out of these guidelenes recently published that refers to the usage of model based development. This article presents a survey done with professionals working in the area, looking forward to gather perceptions about the impact of model based development in avionics software production. The analysis points out that using the methodology normally is beneficial, especially in some development steps, where the benefits might be augmented or diminished according to how it is applied.*

Resumo. *O desenvolvimento de software aviônico normalmente segue guias criados pela indústria que estabelecem objetivos e atividades fundamentais para habilitar este a entrar em operação. A DO-331 é um destes guias, entre os mais recentes, que refere-se à utilização do desenvolvimento baseado em modelos. Este artigo apresenta um survey realizado com profissionais da área, buscando as percepções sobre o impacto do desenvolvimento baseado em modelos na produção de software aviônico. A análise aponta que a utilização da metodologia normalemente é benéfica, especialmente em algumas etapas do desenvolvimento, podendo estes benefícios serem potencializados ou reduzidos de acordo com a forma de aplicação.*

1. Introdução

Normalmente na prática da corrida como atividade física, utiliza-se um *smartwatch* para registrar informações sobre o exercício, como por exemplo batimentos cardíacos, quantidade de passos por minuto, distância e trajeto percorrido. Na necessidade de pagar uma conta, o aplicativo do banco no smartphone permite facilmente acesso à conta, leitura do código de barras através da câmera e, após autenticação, a operação é finalizada em questão de minutos com a devida segurança. Para locomoção de um ponto A até um ponto B, tem-se o GPS embarcado no veículo ou smartphone, o qual determina o caminho mais otimizado e guia por comandos de voz, inclusive corrigindo o trajeto em eventuais erros. Os exemplos citados são apenas alguns dos inúmeros os quais software está presente rotineiramente.

Essa onipresença é uma tendência e, em algumas situações, os usuários se deparam com falhas que podem causar aborrecimento ao tomar muito tempo ou impedir que uma tarefa seja finalizada. Em outras circunstâncias, aplicações possuem um nível elevado de criticidade, ou seja, podem levar a eventos perigosos ou catastróficos em caso de

falha. Segundo IEEE (1990, p. 25) , "software crítico é aquele o qual a falha pode ter impacto na segurança, causando perdas financeiras ou sociais relevantes". Por exemplo, a indicação errônea de altitude em uma aeronave pode acarretar na queda da mesma, provavelmente, causando fatalidades e perda de propriedade. Em uma aplicação voltada para a área da medicina, uma falha de cálculo pode causar o óbito de um paciente durante um procedimento cirúrgico.

A existência dessas características denominam esse tipo de aplicação como críticas para a segurança. Para que as aplicações deste tipo sejam colocadas em uso, existe a necessidade de que as empresas responsáveis pelo desenvolvimento sigam certas normas e padrões internacionais. Por exemplo, RTCA DO-178 é um padrão de garantia de qualidade de software relacionado à segurança para o desenvolvimento de software em sistemas aeroespaciais (Zoughbi et al, 2011). Logicamente, apenas seguir as normas não é suficiente, ou seja, é necessário comprovar através de evidências que os objetivos foram cumpridos e todas as medidas necessárias foram tomadas para garantir a confiabilidade do produto. Esta avaliação independente é conhecida como certificação, sendo realizada por uma autoridade certificadora como a Agência Nacional de Aviação Civil (ANAC) e a *Federal Aviation Administration* (FAA), voltadas para o setor de aviação.

O design correto de um projeto a partir de seus requisitos é uma das partes vitais para o sucesso do mesmo. Explorar a técnica de desenvolvimento baseado em modelos pode ser uma das formas de mitigar os erros que podem ocorrer nessa fase, bem como auxiliar em processos de desenvolvimento, verificação, validação e certificação. A forma padrão para modelagem de software ocorre através da utilização da *Unified Modelling Language* (UML), porém o desenvolvimento de aplicações de software crítico embarcado lida com essa abordagem de forma particular. Conforme descrito por Santamaría et al. (2012, p. 1-2) , "na comunidade de desenvolvimento de aplicações de software crítico, o termo *model-based* possui um paradigma diferente. Nesta área, este refere-se ao desenvolvimento de modelos de controle com capacidade de simulação, a partir de ferramentas próprias para essa tarefa".

Há empresas no setor aeroespacial que desenvolvem software e tem interesse em adotar o desenvolvimento baseado em modelos, porém em muitos casos evitam esta transição por alguns motivos. Em primeiro lugar, as ferramentas existentes no mercado, mesmo que de qualidade, em muitos casos não atendem a maior parte das necessidades de uso. Além disso, não é possível prever a aplicabilidade e efetividade antes de experimentá-las, ou seja, como determinar se o processo de desenvolvimento será melhor ao adotar uma ferramenta e também como os desenvolvedores irão lidar com tal novidade. Em instituições do mesmo ramo que já empregam o uso de tais ferramentas, é possível encontrar dúvidas sobre o verdadeiro ganho de seu uso, por serem mal compreendidas ou pela complexidade da análise comparativa frente os métodos tradicionais de desenvolvimento.

Considerando esta contextualização, este trabalho teve por objetivo principal analisar os impactos da aplicação do desenvolvimento baseado em modelos em projetos de software aviônico, a partir da percepção de pessoas com experiência na área. A questão de pesquisa a ser respondida é: Quais são os impactos da utilização do desenvolvimento baseado em modelos para aplicações aviônicas que buscam atingir os objetivos da norma RTCA DO-178? Para tanto, os seguintes objetivos específicos foram estabelecidos:

1. Identificar as principais dificuldades no desenvolvimento de software frente o padrão RTCA DO-178 através de referências bibliográficas.
2. Criar um questionário que permita capturar a experiência dos participantes frente o uso do desenvolvimento baseado em modelos em projetos de software aviônico.
3. Analisar os resultados coletados a fim de obter conclusões sobre o impacto da utilização do desenvolvimento baseado em modelos para software aviônico.

Na próxima seção deste trabalho encontra-se a fundamentação teórica utilizada como base, abrangendo sistemas de segurança crítica, software aviônico e desenvolvimento baseado em modelos. Os métodos de pesquisa utilizados são apresentados em seguida, somados à apresentação de resultados e respectiva análise. Ao final, as considerações finais são apresentadas.

2. Fundamentação Teórica

2.1. Sistemas de segurança crítica

Sommerville (2011) afirma que os sistemas críticos de segurança são os sistemas nos quais é essencial que a operação do sistema seja sempre segura, ou seja, que o sistema nunca deve causar danos às pessoas ou ao ambiente, mesmo que ocorra uma falha.

Segundo Sommerville (2003), a falha de muitos sistemas controlados por software causa inconveniência, mas não danos sérios e prolongados. Contudo, há certos sistemas em que as falhas podem resultar em perdas econômicas significativas, danos físicos ou ameaças à vida humana. Esses sistemas, em geral, são chamados de sistemas críticos.

Para Sommerville (2011), a confiança é um atributo essencial dos sistemas críticos e todos os aspectos da confiança (disponibilidade, confiabilidade, segurança e proteção) podem ser importantes. Uma relação destes fatores é apresentada na Figura 1, adaptado de Sommerville (2011), que são especificados da seguinte forma:

- Confiabilidade: É a probabilidade de operação livre de falhas durante um tempo especificado, em um dado ambiente, para um propósito específico.
- Disponibilidade: É a probabilidade de um sistema, em determinado instante, ser operacional e capaz de fornecer os serviços requeridos.
- Segurança: Reflete a capacidade do sistema de operar, normal e anormalmente, sem ameaçar as pessoas ou o ambiente.
- Proteção: Avaliação do ponto em que o sistema protege a si mesmo de ataques externos, que podem ser acidentais ou deliberados.

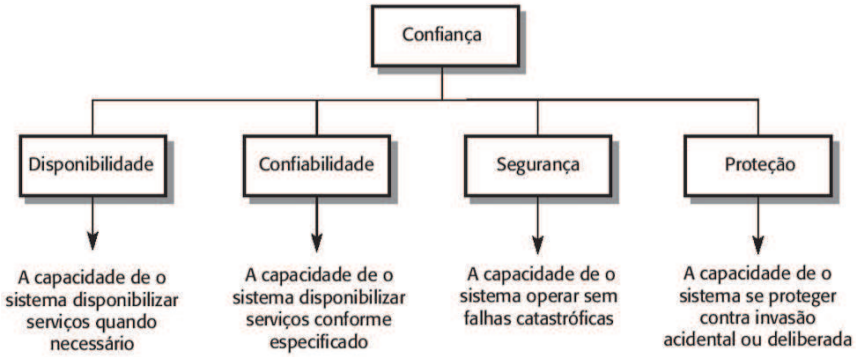

Figura 1. Dimensões da confiança

Há sistemas críticos que, por necessidade, focam no aspecto de segurança, os tornando únicos dentro deste universo. Conforme Sommerville (2003), quando a segurança é um atributo essencial de um sistema crítico, esse sistema é um 'sistema de segurança crítica'. Nem todos os sistemas que tem essa característica possuem software, em especial os que não tem complexidade elevada, e o comportamento esperado pode ser todo implementado via hardware. Por outro lado, quando o software faz parte desse tipo de sistema, ele tem a designação de software crítico de segurança, conforme explica Sommerville (2011), divindo este em duas classes:

1. Software crítico de segurança primária - Esse é um software que é embutido como um controlador em um sistema. A disfunção desse software pode causar uma disfunção de hardware, o que resulta em ferimentos em pessoas ou em dano ambiental.
2. Software crítico de segurança secundária - Esse é um software que pode indiretamente resultar em ferimentos.

Um software confiável inclui código extra, frequentemente redundante, para realizar a verificação necessária dos estados de sistema excepcionais e para possibilitar recuperação a partir de falhas do sistema (Sommerville, 2003). Naturalmente, um sistema crítico de segurança deve ser confiável no que se refere a atender suas especificações e operar sem falhas (Sommerville, 2003).

Considerando a necessidade de sistemas críticos possuírem um nível muito elevado de confiança, o custo de desenvolvimento destes também se torna muito alto. Além disso, o desenvolvimento de software crítico normalmente é regulado por normas exigentes, de acordo com a sua finalidade, a fim de garantir a segurança. Segundo Sommerville (2011, p. 205),

> por causa dos custos extras com projeto, implementação e validação, aumentar a confiança de um sistema amplia significativamente seus custos de desenvolvimento. Em particular, os custos de validação são altos para sistemas que devem ser ultraconfiáveis, como sistemas críticos de controle de segurança. Além de validar se o sistema atende aos seus requisitos, o processo de validação pode precisar provar para um regulador externo que o sistema é seguro. Por exemplo, sistemas de aeronaves precisam demonstrar aos órgãos reguladores, como a Autoridade Federal de Aviação, que a probabilidade de uma falha catastrófica de sistema que afete a segurança de uma aeronave é extremamente baixa.

Sommerville (2011) retrata ainda que quando o software não é muito confiável, existe a possibilidade de obter melhorias significativas com custos relativamente baixos, mas ao utilizar boas práticas o custo de melhorias são muito maiores e os benefícios menores. O gráfico exponencial extraído de m (o), na Figura 2, retrata a relação entre custo e confiança:

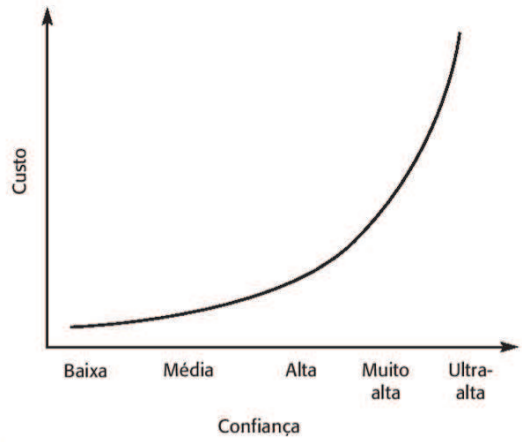

Figura 2. Relação custo/confiança

Para Sommerville (2011), a chave para garantir a segurança é assegurar que os acidentes não ocorram e que as consequências de um acidente sejam mínimas em caso de ocorrência. Sommerville (2011) ainda afirma que existem 3 formas complementares de alcançar essa garantia necessária:

1. Prevenção de perigos: O sistema é projetado de modo que os riscos sejam evitados.
2. Detecção e remoção de perigos: O sistema é projetado de modo que os perigos sejam detectados e removidos antes que resultem em um acidente.
3. Limitação de danos: O sistema pode incluir recursos de proteção que minimizem os danos que possam resultar em um acidente.

2.2. Software aviônico

Mesmo que este trabalho tenha foco no desenvolvimento de software aviônico baseado em modelos, existe a necessidade de compreender um pouco o cenário o qual este está inserido. Isto permite um melhor entendimento sobre os detalhes que são abordados quanto os processos de desenvolvimento de software desta natureza e suas especificidades. O software aviônico se encontra num contexto único, fazendo parte de um sistema que se comunica com outros sistemas que, em conjunto, formam outro sistema: a aeronave. O sistema para o usuário final é visto somente como um e cada uma de suas partes impacta de forma diferente na segurança. Segundo Rierson (2013, p. 13),

> Software opera no contexto de um sistema e segurança é uma propriedade geral do sistema. Software não é nem seguro nem inseguro. De qualquer forma, software influencia em uma variedade de sistemas, incluindo sistemas para aeronaves, sistemas automotivos, sistemas nucleares e sistemas médicos. Para desenvolver software que melhore a segurança ao invés de prejudicar, devemos primeiro compreender o sistema no qual o software opera e o processo de segurança do sistema.

Existem normas e padrões criados pela indústria aeronáutica que funcionam como diretrizes para as empresas produzirem aplicações para esse domínio. Nos itens a seguir esses padrões serão abordados, buscando identificar como estes afetam o processo e comentando sobre as principais fases de desenvolvimento existentes.

2.2.1. Normas, padrões e conceitos

Desenvolver a *expertise* e competência para construitr sistemas críticos de segurança requer tempo significativo, recursos e integridade (Rierson, 2013). Para a aviação civil, desenvolvedores são encorajados a utilizar a *Aerospace Recommended Practice* (ARP) 4754, intitulada como *Guidelines for Development of Civil Aircraft and Systems* (Rierson, 2013).

A ARP4754 divide o processo de desenvolvimento de sistemas em seis fases: planejamento, desenvolvimento de funções do avião, alocação de funcões do avião para sistemas, desenvolvimento da arquitetura do sistema, alocação de requisitos do sistema para itens (incluindo alocação de software e hardware) e implementação do sistema (incluindo desenvolvimento de software e hardware). Além disso, a ARP4754 elenca os seguintes processos integrais: avaliação de segurança, designação de nível de confiança, captura de requisitos, validação de requisitos, verificação de implementação, gerenciamento de configuração, garantia de processos e coordenação com autoridades certificadoras (Rierson, 2013). A Figura 3, também extraída e adaptada de Rierson (2013), ilustra de forma geral o processo de desenvolvimento de sistemas. Os processos relacionados a ARP4754 mencionados estão em cinza.

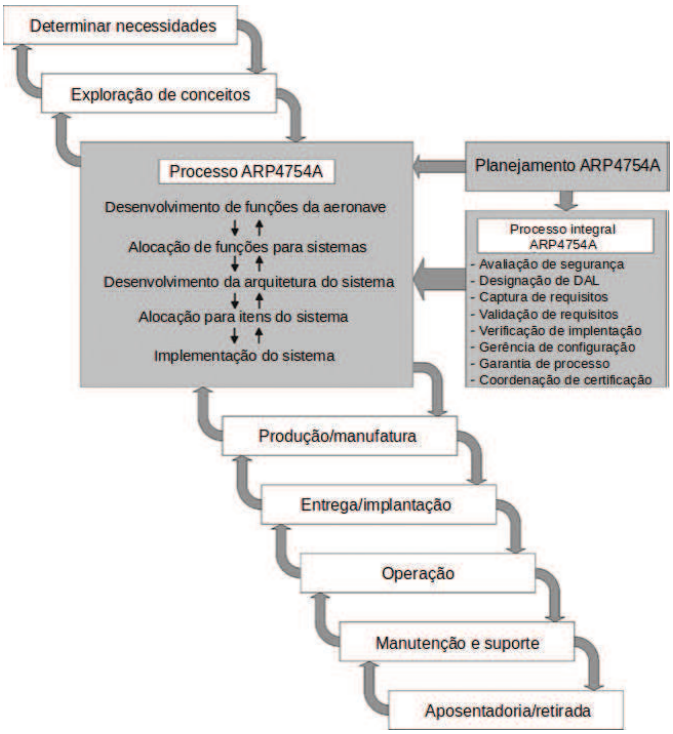

Figura 3. Processo geral de desenvolvimento de sistemas

Para Rierson (2013), software faz parte da implementação do sistema. A arquitetura do sistema e os requisitos direcionam o desenvolvimento de software. Dessa forma, é fundamental para engenheiros de sistema, software, segurança e hardware se comunicarem. O mais cedo e frequente que a comunicação ocorrer ao longo do processo de desenvolvimento, menores serão a quantidade de problemas. É necessário para estas quatro categorias desenvolverem uma relação próxima de trabalho. É possível comparar essas

quatro disciplinas (sistema, segurança, software e hardware) com as quatro pernas de uma cadeira. Sem uma das pernas, a cadeira é muito limitada. Sem duas pernas, é inútil. As falhas de comunicação levam a desafios desnecessários, potencialmente causando erros de intepretação e implementação de requisitos de sistema, problemas para identificar dependências, testes inadequados e desperdício de tempo e dinheiro.

No desenvolvimento de sistemas para aeronaves, as condições de falha associadas com as funcionalidades do avião e suas combinações devem ser identificadas e classificadas. Os efeitos da condição de falha perante o avião são avaliadas frente regulamentações e guias para estabelecer os requisitos de segurança (Rierson, 2013). As falhas identificadas são classificadas de acordo com sua severidade, contendo potenciais efeitos da falha, probabilidades de ocorrência e níveis de confiança necessários.

O Quadro 1, extraido de Rierson (2013), apresenta as classificações de severidade para as falhas, com respectivas probabilidades, níveis de confiança e condições e efeitos. Rierson (2013) afirma que em sistemas complexos e altamente integrados com software e hardware programável, não é plausível testar todas as combinações de entradas e saídas para associar uma probabilidade de falha. Essa limitação, somada ao fato de que programas de computador não detorioram ou falham conforme o tempo como componentes físicos, leva à necessidade de um conceito chamado garantia de desenvolvimento.

Esse conceito é utilizado para assegurar a devida confiança no proceso utilizado para desenvolver o sistema e respectivos itens. A garantia de desenvolvimento são todas as ações sistemáticas e planejadas usadas para substanciar, em um nível adequado de confiança, que erros em requisitos, design e implementação tenham sido identificados e corrigidos a fim de que o sistema satisfaça os critérios de certificação. A garantia de desenvolvimento assume que um processo mais rigoroso está mais apto a identificar e remover erros antes que o produto seja entregue do que um processo menos rigoroso (Rierson, 2013).

O processo de avalição de segurança é responsável por estabelecer os níveis de garantia de desenvolvimento, baseados nos potenciais impactos de segurança sobre o sistema. A ARP4754 define o nível de garantia de desenvolvimento como uma medida de rigor aplicada ao processo de desenvolvimento para limitar, à um nível aceitável de segurança, a probabilidade de que erros ocorram durante o processo de desenvolvimento de funcionalidades da aeronave ou do sistema e itens que possuem efeito adverso à segurança em caso de exposição durante o uso (Rierson, 2013).

As afirmações de Rierson (2013) e Sommerville (2003) são complementares quanto a aplicabilidade de testar todas as possibilidades do sistema, conforme mostra também a Figura 2.

O uso de software para sistemas aeroespaciais aumentou dramaticamente nas últimas décadas. Como resposta à esse aumento no uso de software, a indústria aeroespacial desenvolveu um padrão específico para prover suporte à certificação de sistemas de segurança crítica (Sarkis et al., 2014). Duas fases de desenvolvimento são identificadas na ARP4754: desenvolvimento de função e desenvolvimento de item. A fase de desenvolvimento de função inclui o desenvolvimento, validação, verificação, gerenciamento de configuração, garantia de processo e coordenação de certificação para o sistema. No mais baixo nível, requisitos de sistema são alocados para software e hardware, que são referen-

Classificação de Severidade	Condicões de falha e potenciais efeitos	Probabilidade de ocorrência	Exposição por hora de vôo	Nível de confiança
Catastrófico	Condições de falha que resultariam em múltiplas fatalidades, normalmente com a perda da aeronave.	Extremamente improvável	1E-9	A
Perigoso	Condições de falha que reduziriam as capacidades da aeronave ou limitariam a tripulação a lidar com situações adversas de operação, que poderiam levar à: - Larga redução em margens de segurança ou funcionalidades - Sofrimento físico ou carga de trabalho excessiva a ponto de que a tripulação não fosse confiável para exercer suas tarefas - Lesões sérias ou fatais para um número relativamente pequeno de ocupantes que não sejam tripulantes	Extremamente remota	1E-7	B
Grande	Condições de falha que reduziriam as capacidades da aeronave ou limitariam a tripulação a lidar com situações adversas de operação, que poderiam levar à reduções significativas em margens de segurança ou funcionalidades, aumento significativo na carga de trabalho ou condições que prejudiquem a eficiência dos tripulantes, desconforto para os tripulantes, sofrimento físico para os passageiros ou tripulantes, possivelmente causando lesões.	Remota	1E-5	C
Pequena	Condições de falha que não reduziriam significativamente a segurança da aeronave e que envolveriam ações da tripulação as quais estão bem preparados para realizar. Condições de falha pequena podem incluir pequenas reduções nas margens de segurança ou funcionalidades; Pequeno aumento na carga de trabalho da tripulação, como mudança de plano de voo rotineiro; ou algum desconforto físico para os passageiros ou tripulantes.	Razoavelmente provável	1E-3	D
Sem efeitos de segurança	Condições de falha que não teriam efeito algum sobre a segurança, por exemplo falahas que não afetariam a capacidade operacional da aeronave ou aumento na carga de trabalho dos tripulantes.	Provável	1.0	E

Quadro 1. Condições de falha - Severidade, probabilidades e níveis de confiança

ciados como itens. Os itens de software e hardware possuem seus próprios processos de desenvolvimento. A Figura 4 apresenta o modelo conhecido como desenvolvimento em V, retirado e adaptado de Rierson (2013), em que as fases e respectivos processos estão delimitados.

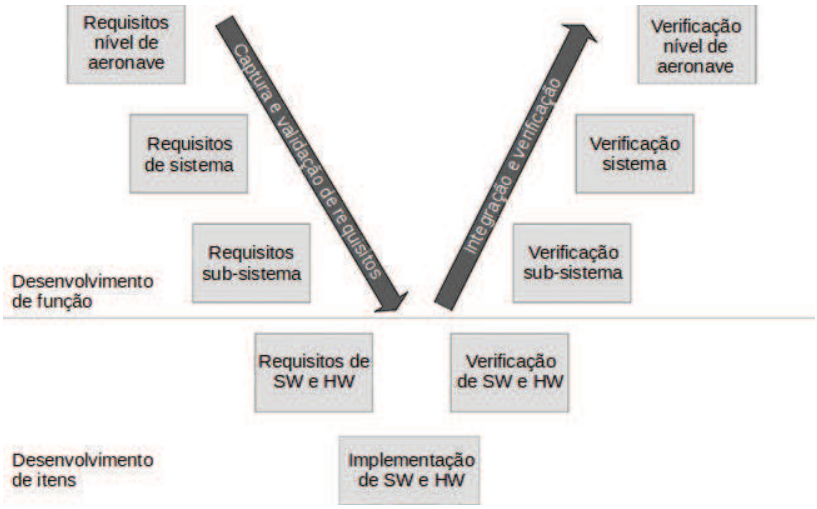

Figura 4. Modelo de desenvolvimento em V

Rierson (2013) afirma que as técnicas tradicionais de análise de falha e confiabilidade não funcionam com software, por causa de sua complexidade e singularidade. Algumas outras características únicas são:

- Programas podem ser complexos e difíceis de compreender, mesmo pequenos.
- Software tende a melhorar com o tempo devido à descoberta e correção de erros.
- Corrigir erros em uma área pode introduzir erros em outra aparentemente sem relação.
- Software pode ter interface com dezenas ou centenas de módulos de software.
- Diferentemente do hardware, o software não fornece um aviso prévio quando falha. Erros ocultos podem permanecer invisíveis por anos e aparecer de repente.
- Software pode ser facilmente modificado.
- Encontrar um erro de software pode demorar e ser trabalhoso, especialmente se não for tratado proativamente.

O padrão criado pela indústria aeroespacial para desenvolvimento de software é conhecido como DO-178 e está atualmente em sua quarta revisão, também conhecida como DO-178 revisão C ou simplesmente DO-178C (Rierson, 2013). A DO-178 fornece recomendações para produção de software em sistemas aeroespeciais e equipamentos que executem sua função com um nível de confiança que cumpra com requisitos de aeronavegabilidade (C (T)).

Por causa da incapacidade de aplicar modelos de confiabilidade de forma efetiva para software, o conceito de garantia de desenvolvimento é aplicada para a maioria das indústrias com foco em segurança, incluindo a de aviação. Quanto mais crítica for a funcionalidade, maior a quantidade de atividades de desenvolvimento e verificação que são aplicáveis (Rierson, 2013).

A DO-178 elenca cinco níveis de software baseado no processo de segurança (identificados pelas letras A, B, C, D e E, visíveis no Quadro 1). Software classificado com nível A é o mais rigoroso e todos os objetivos do padrão são aplicáveis. Por outro lado, o software de nível E não necessita atingir nenhum dos objetivos (Rierson, 2013). O Quadro 2, de e (i), apresenta o aumento na quantidade de objetivos e consequente dificuldade conforme a variação de nível.

É virtualmente impossível provar cientificamente as alegações da abordagem de garantia de desenvolvimento. Ter um bom processo também não necessariamente significa software seguro e robusto. É possível ter um ótimo processo e ainda produzir software com defeitos, frequentemente por requisitos de sistema ruins e pessoas inexperientes. Da mesma forma, um time pode ter um processo abaixo dos padrões e produzir software robusto e confiável, principalmente por possuir engenheiros experientes e altamente capacitados (Rierson, 2013).

Segundo Rierson (2013), para empresas com poucos princípios de segurança, pode ser tentador escolher um nível de garantia de software sem entender o porquê esse nível é necessário. Segregar o processo de desenvolvimento de software dos requisitos de sistema e diretivas de segurança podem levar a grandes problemas de segurança. Por essa razão, a DO-178, ARP4754 e ARP4761 tentam melhorar as práticas para lidar com a implementação de software nos sistemas.

Nível	Objetivos	Objetivos com Independência	Resumo dos objetivos
E	0	0	Nenhuma atividade necessária
D	26	2	- Planos - Requisitos de alto nível desenvolvidos - Arquitetura desenvolvida - Código objeto executável desenvolvido - Arquivos de parametrização desenvolvidos e verificados quanto exatidão e completude - Algumas revisões e análise de requisitos de alto nível - Revisão ou análise da arquitetura em caso de uso de particionamento - Testes normais e de robustez (requisitos de alto nível) - Cobertura de requisitos de requisitos de alto nível - Gerenciamento de configuração - Garantia de qualidade (conformidade com planos e revisão de conformidade) - Resumo de realização e indíce de configuração
C	62	5	- Todas as atividades de nível D - Padrões de desenvolvimento - Requisitos de baixo nível desenvolvidos - Dados rastreáveis desenvolvidos - Código fonte desenvolvido - Revisões e análise de requisitos de alto nível adicionais - Algumas revisões e análise de requisitos de baixo nível - Algumas revisões e análise da arquitetura - Revisão e análise de partes do código fonte - Verificação de arquivos de parametrização - Testes normais e de robustez (requisitos de baixo nível) - Revisão de procedimentos de teste - Revisão de resultados de teste - Declaração de análise de cobertura - Análise de acoplamento de controle e dados - Garantia de qualidade adicional (revisão de planos, padrões, conformidade com padrões e critérios de transição).
B	69	18	- Todas as atividades de nível C - Revisões adicionais e análise de requisitos de alto nível (compatibilidade com *target*) - Revisões adicionais e análise de requisitos de baixo nível (compatibilidade com *target* e verificabilidade) - Revisões adicionais e análise de requisitos de arquitetura (compatibilidade com *target* e verificabilidade) - Revisões adicionais de código fonte (verificabilidade) - Cobertura de decisão
A	71	30	- Todas as atividades de nível B, C e D - Análise de cobertura de múltiplas condições e decisões - Rastreabilidade de código fonte para código objeto

Quadro 2. Resumo de objetivos - DO178C

2.2.2. Requisitos

A produção de software bem sucedida depende diretamente da correta execução de todas as fases de seu ciclo de vida. A escrita de requisitos é a primeira delas e pode ser considerada a base fundamental para todos os processos, conforme escreve Rierson (2013, p. 99):

> Eu não sou arquiteta, mas o bom senso me diz que, ao construir uma casa, a fundação é extremamente importante. Se a fundação for fraca ou feita com pouco material, faltarem partes, ou não estiver nivelada, a casa que será construída sobre a mesma terá problemas no longo prazo. O mesmo é verdade para o desenvolvimento de software. Se os requisitos (a fundação) forem fracos, terá efeitos no longo prazo que levará a problemas indescritíveis e dificuldades para todos os envolvidos, incluindo o cliente.

Todas as fases de desenvolvimento são iterativas e sujeitas a mudanças conforme o andamento do projeto. Contudo, realizar atividades sucessivas baseadas em entradas incompletas e incorretas é um dos erros mais comuns e causa de ineficiência na engenharia de software. Requisitos ruins conduzem ao efeito de bola de neve, ou seja, os problemas e complexidade acumulam até que o projeto vire uma enorme bola de neve não gerenciável (Rierson, 2013). A Figura 5, de Rierson (2013), demonstra o efeito bola de neve dos requisitos até as funcionalidades.

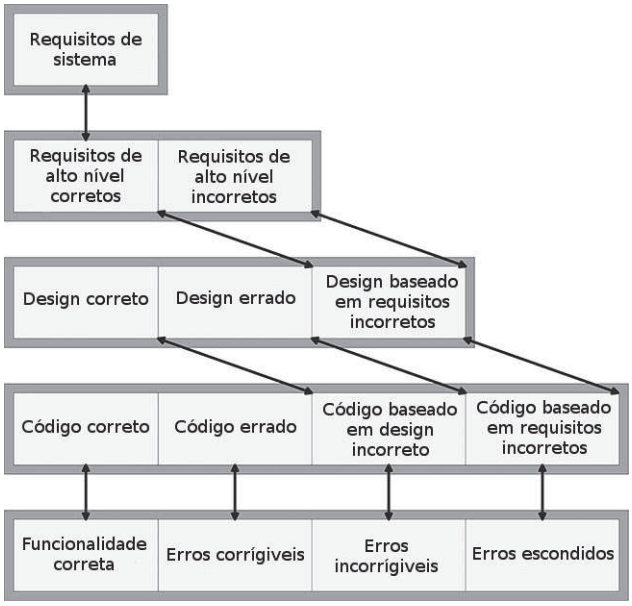

Figura 5. Efeito bola de neve de requisitos

A DO-178 demanda rastreabilidade bidirecional entre requisitos de sistema e requisitos de alto nível de software, requisitos de alto nível de software e requisitos de baixo nível de software, requisitos de baixo nível de software e código fonte, requisitos de software e casos de teste, casos de teste e procedimentos de teste e procedimentos de teste e resultados de teste (Rierson, 2013). Essa relação de rastreabilidades pode ser melhor compreendida pela Figura 6, adaptado de Rierson (2013).

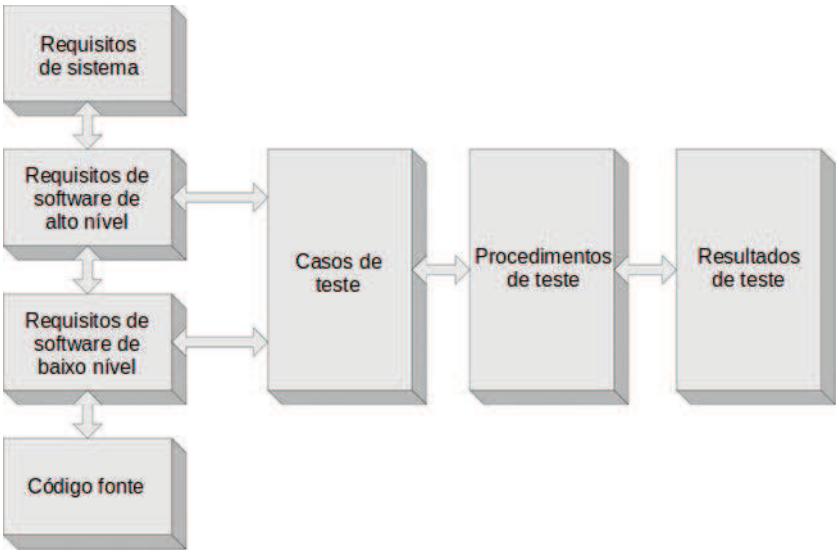

Figura 6. Rastreabilidade bidirecional entre artefatos do ciclo de vida

A rastreabilidade dentre todos os itens mencionados e presentes na Figura 6 é fundamental, especialmente nas fases de certificação de software. Durante auditorias muitas vezes é necessário demonstrar a relação existente em ambas as direções, ou seja, chegar até o código fonte a parte de um requisito de sistema e vice-versa. Os requisitos de sofware também possuem rastreabilidade para os itens de teste, permitindo que estes e respectivos resultados sejam relacionados até o código fonte ou requisitos de sistema.

Além da questão de certificação, essa rastreabilidade também permite aos desenvolvedores maior facilidade em uma eventual análise de impacto de mudança. Por exemplo, se um requisito for modificado, será possível identificar todos os outros itens que serão afetados em diferentes níveis a partir da rastreabilidade, permitindo modificações mais controladas. Os três níveis de requisito visíveis na Figura 6 são desenvolvidos em etapas diferentes do projeto, sendo os requisitos de baixo nível de software normalmente parte da fase de design.

2.2.3. Design

Para Rierson (2013), existe um desvio na DO-178C frente o que encontramos na literatura sobre o design de software. A DO-178C explica que o design de software é formado pela arquitetura e também os requisitos de baixo nível. Rierson (2013) analisa também que o design serve como um *blueprint* para a fase de implementação, descrevendo como o software deve ser construído (arquitetura) e como deve realizar a funcionalidade (requisitos de baixo nível).

A arquitetura é parte crucial do processo de criação de um design. Para cumprir com os objetivos da DO-178C é necessário que a arquitetura seja compatível com os requisitos e, para tanto, algumas formas de garantir essa compatibilidade são necessárias. Em muitos casos, a rastreabilidade ou mapeamento entre requisitos e arquitetura são utilizados (Rierson, 2013). Além disso, a arquitetura deve ser documentada utilizando um formato claro e consistente, sendo importante considerar para isto quem irá implementar

e manter o código usando este design. A arquitetura deve ser definida de forma clara para que seja precisamente implementada e mantida (Rierson, 2013).

Vários estilos de arquitetura existem e a maioria inclui componentes e conectores. O tipo destes componentes e conectores dependem da abordagem utilizada para arquitetura. A maior parte do software de tempo real em aeronaves utiliza a estrutura funcional, onde componentes representam funções e conectores demonstram as interfaces entre as funções tanto para fluxo de controle quanto de dados (Rierson, 2013).

Os requisitos de baixo nível são a decomposição dos requisitos de alto nível para um nível do qual código fonte pode ser diretamente escrito. A DO-178C essencialmente coloca o detalhemento na fase de design e não na fase de codificação. Se o design for documentado apropriadamente, o esforço para codificação deve ser relativamente pequeno (Rierson, 2013).

Da mesma forma que acontece com os requisitos, o design de software também necessita ser verificado. Normalmente essa verificação ocorre através de revisões, que incluem avaliar a arquitetura e os requisitos de baixo nível. É comum a utilização de um *checklist* para a avaliação, que auxilia os engenheiros a realizarem essa tarefa de forma minusciosa (Rierson, 2013).

2.2.4. Implementação

Todos os processos são fundamentais no desenvolvimento de software aviônico bem sucedido, porém o produto final integrado e operante na aeronave é o código fonte. Conforme afirma Rierson (2013), a atividade de codificação é o processo de desenvolvimento do código a partir do design e, mesmo que design e requisitos sejam críticos, código compilado e linkado é o que realmente voa. Mesmo considerando esta afirmação, Rierson (2013, p. 166) destaca que, em sua experiência,

> o código mais problemático é criado quando programadores são forçados a criar código rapidamente. Os requisitos e o design ou são imaturos ou inexistentes e os programadores simplesmente produzem código. É como construir uma casa nova sem ter a planta baixa e sem ter materiais de qualidade; Vai desmoronar - somente não sabemos quando.

O código fonte necessita implementar única e somente o design definido (incluindo requisitos de baixo nível e arquitetura), ter rastreabilidade para os requisitos de baixo nível e ser compatível com os padrões de código estabelecidos. As saídas da fase de codificação são o código fonte e dados de rastreabilidade. Dados como instruções de *build* (incluindo diretivas de compilação e *link*) e carregamento também são criadas durante a fase de codificação, mas são tratadas como parte da integração (Rierson, 2013).

Os padrões de código que são usados normalmente são definidos durante a fase de planejamento do projeto e devem ser seguidos ao longo do mesmo. Rierson (2013) esclarece que esses padrões explicam como usar uma linguagem específica corretamente, restrições sobre diretivas específicas da linguagem críticas para segurança, convenções de nomenclatura, utilização de dados globais e como desenvolver código legível e sustentável. Precisão, consistência, conformidade com design e conformidade com padrões são alguns exemplos do que a verificação do código fonte busca garantir (Rierson, 2013).

Frequentemente código é escrito sem a devida atenção aos padrões e durante revisões de código problemas significativos são encontrados e o retrabalho é inevitável (Rierson, 2013).

2.2.5. Verificação

Em um projeto de software aviônico, a verificação deve estar estar presente desde a fase de planejamento até a entrega e manutenção. A DO-178C especifica como guia de verificação uma combinação de revisões, análises e testes. As revisões e análises tem como objetivo avaliar a precisão, completude e verificabilidade das saídas de cada fase do ciclo de vida do projeto (incluindo planejamento, requisitos, design, codificação, desenvolvimento de testes e execução de testes). Os testes exercitam o sistema para garantir que o mesmo satisfaz os requisitos e para detectar erros (e (i)).

Rierson (2013) analisa que metade dos objetivos da norma DO-178C são identificados como objetivos de verificação sendo que, quanto mais crítico for o software, maior a quantidade de atividades a fim de atingir os objetivos e aumentar a confiança de identificação e remoção de erros. Da forma complementar, Rierson (2013) menciona sua experiência e conversas com gerentes de projeto, relatando que mais da metade do orçamento dos projetos são utilizados nas atividades de verificação.

Rierson (2013) ratifica que a verificação é essencial para a segurança, que é utilizada para satisfazer as regulamentações ao confirmar que o software faz somente o que se propõe a realizar. Basicamente, sem uma verificação de qualidade, a garantia de desenvolvimento não tem nenhum mérito, pois é a verificação que traz a confiança ao produto.

Ter conhecimento sobre características temporais do software é essencial para o design e execução bem sucedida de sistemas de tempo real. Uma mensuração crítica de tempo é conhecida como *Worst Case Execution Time* ou WCET, que define a maior quantidade de tempo necessária para completar a execução de certas tarefas em um determinado processador (Rierson, 2013). Essa análise normalmente é realizada para garantir que o tempo máximo utilizado está conforme com a quantidade de tempo alocada e, mesmo que a abordagem utilizada dependa da arquitetura do software e do sistema, os tempos são mensurados e analisados (Rierson, 2013). Para tanto, é necessário avaliar os caminhos possíveis e iterações existentes nos mesmos, elegendo quais são os piores em cada caso. A soma dos piores caminhos que podem ocorrer num mesmo ciclo de execução são somados, resultando no valor de WCET (Rierson, 2013).

Testes de requisitos de alto e baixo nível são atividades majoritárias necessárias para conformidade com a norma DO-178C. O propósito dos testes de software é descobrir erros feitos durante as fases de desenvolvimento. A DO-178C tem o foco dos testes baseados em requisitos, ou seja, testes que são escritos e executados para provar que os requisitos são cumpridos e que não exista nenhuma funcionalidade não prevista (Rierson, 2013). Quanto mais crítico o software for, maior é o esforço necessário nos testes.

Na norma DO-178C são previstos duas formas de casos de teste, que são os normais e os de robustez. Os casos de teste normais são aqueles casos que procuram erros no software considerando entradas normais e condições esperadas. Já os casos de teste de robustez são criados para mostrar como o software reage quando exposto à condições

Nível de Garantia	Normal / Alto nível	Robustez / Alto nível	Normal / Baixo nível	Robustez / Baixo nível
A	Mandatório	Mandatório	Mandatório com independência	Mandatório com independência
B	Mandatório	Mandatório	Mandatório com independência	Mandatório com independência
C	Mandatório	Mandatório	Mandatório	Mandatório
D	Mandatório	Mandatório	Opcional	Opcional

Quadro 3. Requisitos de teste DO-178C

anormais ou entradas inesperadas (Rierson, 2013). O Quadro 3, adaptado de Rierson (2013), mostra a demanda da DO-178C quanto testes de requisitos de alto e baixo nível.

A independência mencionada no Quadro 3 e requisitada em certas atividades da DO-178C é a segregação de responsabilidades para garantir uma avaliação objetiva. Como exemplificado por Rierson (2013, p. 187) em sua analogia:

> Se você já tentou editar seu próprio trabalho, sabe o quão difícil é encontrar os erros de digitação. Você sabe como ele supostamente deve ser lido, então esses erros são simplesmente ignorados. O mesmo é válido quando criamos software.

Há também uma análise muito importante no desenvolvimento de software aviônico conhecida como análise de cobertura estrutural ou análise dinâmica. Essa investigação tem como objetivo identificar qualquer estrutura de código que não tenha sido exercitada durante a execução dos testes baseados em requisitos (Rierson, 2013). Conforme descreve Rierson (2013), a análise de cobertura estrutural tem os seguintes propósitos:

- Garantir que todo o código executa ao menos uma vez
- Encontrar funcionalidades não intencionais ou não testadas
- Identificar código
- Confirmação de que código desativado realmente está desativado
- Identificar combinações mínimas de teste (evita testes exaustivos)
- Identificar lógicas incorretas

O tamanho e a complexidade de software aviônico seguirá crescendo ao passo que as funcionalidades seguirem sendo realocadas do hardware para o software. Para gerenciar o nível de complexidade enquanto lidam com questões como o ambiente de operação e requisitos de confiabilidade significativos, os engenheiros de software responsáveis precisam de métodos, ferramentas e processos para melhorar a abstração, compreensibilidade, precisão e previsibilidade dos sistemas. (Petit et al. 2013).

2.3. Desenvolvimento de software baseado em modelos

O desenvolvimento de software baseado em modelos é um paradigma que coloca modelos como um artefato primário no processo de desenvolvimento. Em contraste com os métodos tradicionais, modelos não são usados somente para documentação, mas servem como artefato central para implementação do sistema. Modelos podem ser fonte para geração de código, bem como executáveis através de um interpretador. Além disso, podem ser

usados em diversas outras tarefas, como para a comunicação com partes interessadas ou análises baseadas em modelos (Streekmann, 2011).

Desde sua concepção, o uso de modelos para desenvolvimento de software se difundiu dentre as principais indústrias e a aeroespacial não foi exceção à regra (Sarkis et al., 2014). Quando o padrão DO-178, mencionado no capítulo 2.2, foi atualizado para a revisão C, um apêndice conhecido como DO-331 foi criado. Este apêndice trouxe novas orientações, incluindo as atividades e objetivos que devem ser atingidos para o desenvolvimento de software utilizando modelos (RTCA, 2018).

Rierson (2013) traz a definição de modelo da DO-331 como uma abstração de um conjunto de aspectos de um sistema que pode ser usado para análise, verificação, simulação, geração de código e qualquer combinação destes. Rierson (2013) afirma também que os modelos podem fazer parte da hierarquia de requisitos na forma de requisitos de sistema, requisitos de software e design de software. De acordo com Rierson (2013), as principais vantagens na utilização do desenvolvimento baseado em modelo são:

- Redução no ciclo de desenvolvimento de software: o uso de modelos tem o potencial de reduzir de forma geral o tempo utilizado para o desenvolvimento de software. Isto pode ocorrer por duas razões principais. Primeiramente, os modelos oferecem uma descrição mais intuitiva do sistema comparado aos requisitos textuais. Isso facilita o entendimento entre os desenvolvedores em nível de software e sistema. Modelos também trazem a possibilidade de simulação e, por isso, simplicifam a identificação e correção de erros. A outra razão para redução no ciclo de desenvolvimento é a possibilidade de geração automática de código fonte a partir do modelo. Essa alternativa faz com que a codificação manual, atividade que consome muito tempo em projetos de software, seja desnecessária. Caso ferramentas qualificadas sejam utilizadas para geração de código, não será essencial verificar o mesmo.
- Nível de abstração mais alto: quando comparado aos métodos tradicionais de desenvolvimento, o uso de modelos possuí um nível de abstração maior. Os ganhos são similares aos que podem ser observados na de transição linguagens de baixo nível para linguagens de alto nível. Quando geradores de código são utilizados, o desenvolvedor de software pode dar foco ao desenvolvimento, validação e verificação do modelo, ficando livre dos detalhes de implementação. Dessa forma, o produto final entregue será geralmente de maior qualidade.
- Melhor comunicação com partes interessadas: uma vez que os modelos possuem uma linguagem mais natural e intuitiva, a comunicação entre as partes interessadas ocorre de forma melhor. Maior grau de alinhamento, entendimento e maturidade sobre os requisitos de sistema são alcançados mais rapidamente no projeto.

Rierson (2013) destaca também os potenciais riscos e benefícios da utilização de desenvolvimento baseado em modelos para software aviônico, avaliando que os mesmos virão a tona dependendo da forma com que a abordagem é inserida no projeto. As possíveis vantagens elencadas são:

- Ciclo de vida em Y (Figura 7) ao invés de ciclo de vida em V (Figura 4)
- Maior foco em requisitos
- Antecipar verificação
- Maior compreensibilidade de requisitos

- Melhoria na interação com clientes
- Suporte de ferramentas
- Utilização de métodos formais

Por outro lado, as desvantagens que podem ser percebidas são:

- Perda de múltiplas revisões de requisitos
- Expansão das responsabilidades da engenharia de sistemas
- Dificuldades com rastreabilidade
- Controvérsia nos créditos por simulação
- Frequente confusão requisitos e design
- Desafios para separar modelos de especificação e modelos de design
- Desafios para validar os modelos
- Confusão entre as responsabilidades das áreas de sistema e software
- Interpretação incosistente de modelos
- Manutenção
- Controvérsia sobre código gerado
- Instabilidade de ferramentas
- Limitações de modelagem

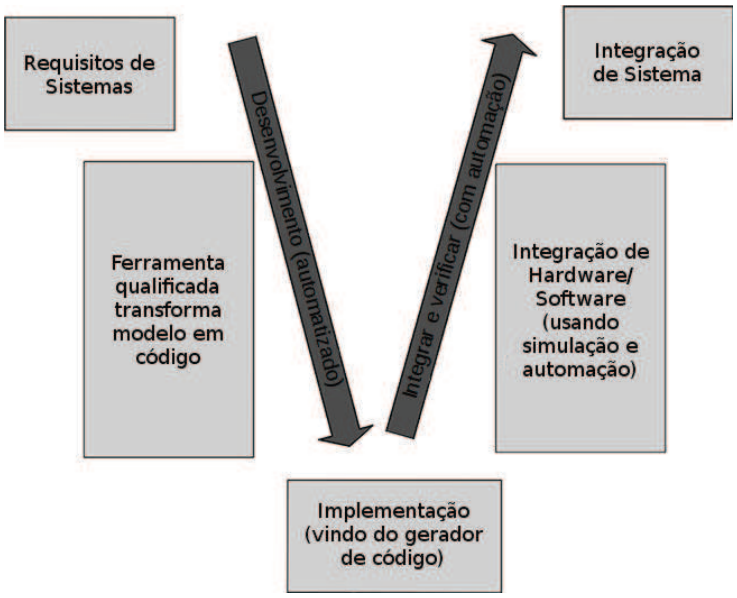

Figura 7. Modelo de desenvolvimento em Y

Para Pettit et al. (2013), chegamos a um ponto em que precisamos deixar abordagens de baixo nível, centradas em código e adotarmos práticas rigorosas de modelagem para lidar com as crescentes demandas colocadas sobre sistemas modermos de software. Mesmo que muitos esforços tenham sido realizados para utilização de engenharia baseada em modelos para projetos recentes de software aeroespacial, ainda existem diversos desafios que complicam a adoção sistêmica. Muitos desses desafios estão relacionados à utilização de modelos em larga escala ao invés de focar no desenvolvimento de componentes individuais.

Pettit et al. (2013) afirma também que é fundamental coordenar a metodologia que será utilizada para o desenvolvimento dos modelos entre times, caso contrário estes

tendem a divergir sobre o uso. Essa tendência leva à inconsistências nos quesitos de qualidade, completude e consistência entre os modelos. Para facilitar a coordenação e para que os times possam aplicar modelagem de forma homogênea, as organizações devem estabelecer uma visão consistente sobre os modelos, ou seja, quais artefatos devem ser gerados, quão completos devem ser em cada fase do ciclo de desenvolvimento e quais métodos serão utilizados para avaliar a qualidade dos artefatos gerados.

A engenharia baseada em modelos facilita a validação e verificação do design, porém, os modelos devem ser corretos por si só. Um modelo é somente tão válido quanto o design o qual tenta descrever, frequentemente de forma abstrata. Mesmo que um design esteja bem representado num modelo, isso não necessariamente faz o mesmo um modelo válido do sistema, uma vez que o design pode não estar correto (Pettit et al., 2013).

A maioria das ferramentas de modelagem possui geração automática de código fonte diretamente do modelo, mas a qualidade e eficiência desse pode ainda demandar certificação, particularmente quando aplicado à sistemas críticos. Mesmo que abordagens para certificar geradores de código existam, grande parte das organizações precisam utilizar o mesmo empenho para verificar um código automaticamente gerado ou uma codificação manual. Como consequência disto, frequentemente as empresas desistem de utilizar a geração automática de código e implementam manualmente o que é especificado pelo modelo. Essa forma de desenvolvimento requer um gerenciamento minuscioso para mitigar inconsistências entre o modelo e o código, aumentando muito probabilidade de que o modelo deixará de ser utilizado no ciclo de desenvolvimento conforme o aumento de complexidade tanto do modelo como do código (Pettit et al., 2013).

Rierson (2013) comenta que a utilização de ferramentas de modelagem qualificadas que geram código automaticamente e, em alguns casos, geram vetores de testes é um paradigma bem recente para a comunidade de software aviônico. Considerando que o apêndice DO-331 da norma DO-178C é relativamente recente e essa afirmação podemos inferir que ainda existe muita incerteza frente a utilização de modelos no desenvolvimento de software aviônico, especialmente tratando a geração de código automática.

As primeiras perguntas que normalmente surgem sobre código fonte gerado a partir de modelos são sobre sua eficiência. Podemos decompor eficiência de código em duas áreas: performance de execução e consumo de memória. Os geradores automáticos produzem código que, comparado à um código manual equivalente, possuí uma diferença de performance de 5 a 15%, para melhor ou pior. Considerando que esta afirmações foram extraídas de Selic (2003) , podemos afirmar com certa convicção que este índice deve ter melhorado de acordo com os avanços tecnógicos que ocorreram desde então.

Selic (2003) afirma que podem existir ocasiões críticas em que código manual pode ser necessário em partes específicas do modelo. Este fato em muitos casos é utilizado como razão para reijeitar o uso do desenvolvimento baseado em modelos, mesmo que envolva uma parte muito pequena do sistema como um todo.

2.4. Trabalhos relacionados

O artigo de Prosvirin et al. (2015) apresenta como problema o aumento na demanda de desenvolvimento de sistemas e veículos aéreos não tripulados, destacando as dificuldades dada a complexidade do processo e das aplicações que demandam finalização cada vez mais rápida. Como possível solução, o desenvolvimento formal baseado em modelos

com a utilização da ferramenta SCADE é apresentado, demonstrando as possibilidades de simulação e geração de código através da ferramenta, bem como principais benefícios da utilização dessa metodologia.

No estudo de Paz et al. (2016) , diversas abordagens de desenvolvimento são avaliadas quanto seu suporte ao desenvolvimento e certificação frente a norma DO-178C. Um *framework* foi construído para caracterização das abordagens de acordo com vários critérios, focado especialmente na cobertura dos dados para conformidade com a norma DO-178C. A árvore representativa deste *framework*, considerada um dos principais resultados da pesquisa, demonstra quatro principais domínios ligados à abordagem de desenvolvimento baseada em modelos: filosofia, cobertura frente a norma DO-178C, tratamento de informações e utilização. A partir dos dados criados, uma análise sobre 12 abordagens diferentes para o desenvolvimento baseado em modelos é retratada, com foco em software aviônico e software crítico em geral. Nesta, são mencionadas possíveis ferramentas e notações não proprietárias que podem ser utilizadas como suporte para áreas como design e análise de arquitetura, testes e especificação de sistema.

A observação dos principais desafios da engenharia baseada em modelos para sistemas de software aeroespaciais foi realizada por Pettit et al. (2013) . Os fatores predominantes de dificuldades experienciadas pelos autores com modelos no desenvolvimento são evidenciados, explicando em cada caso a relevância, suas causas, efeitos e mitigações. As causas apresentadas e analisadas são a falta de coordenação de desenvolvimento, integração de múltiplas linguagens de modelagem, aspectos de verificação e validação, consistência entre código e modelo e captura e verificação de propriedades críticas do sistema. Este trabalho por fim conclui que há claras oportunidades para o crescimento do desenvolvimento baseado em modelos na área de software embarcado para aeronaves, tanto considerando inovações quanto na correta aplicação das tecnologias existentes.

Os trabalhos relacionados identificados no Quadro 4 possuem claramente um contexto em comum, o qual também é tema central deste artigo: o desenvolvimento baseado em modelos para software aviônico ou crítico.

A análise de Prosvirin et al. (2015) demonstra as percepções dos autores sobre várias abordagens através da utilização do *framework* desenvolvido, incluindo questões como a finalidade dos modelos, tema também abordado por este estudo durante o levantamento e análise. No relato de experiência sobre os desafios de aplicação do desenvolvimento baseado em modelos para software aeroespacial, Pettit et al. (2013) comentam sobre dificuldades que tiveram com questões de comunicação, simulação, implementação e requisitos. O levantamento realizado nessa pesquisa abordou todas estas, mas com foco em compreender em uma escala mais larga o impacto dos modelos através das percepções dos profissionais da área, ou seja, trazer uma visão mais generalista sem expor os detalhes. Já Paz et al. (2016) apresenta um estudo de caso sobre uma estação de solo para controle de veículos aéreos não tripulados, desenvolvida com a utilização de uma ferramenta específica de desenvolvimento baseado em modelos. Durante a análise do estudo de caso, o autor comenta sobre o bom suporte dos modelos para questões como design, requisitos, rastreabilidade e implementação, que também fizeram parte do questionário aplicado neste estudo.

Trabalho relacionado 1: Provisrin et al. (2015)		
Contexto	Objetivos	Método
Uso de modelos para atingir custos e certificação de sistemas para veículos aéreos não tripulados dentro do prazo	Solução baseada em modelos para sistemas em veículos aéreos não tripulados; Ganhos para conformidade com padrões de segurança, principalmente automação de processos custosos; Considerações sobre formas de aplicar tecnologias modernas de desenvolvimento baseado em modelos;	Estudo de caso
Trabalho relacionado 2: Paz et al. (2016)		
Contexto	Objetivos	Método
Formas com que o desenvolvimento baseado em modelos suporta a produção e certificação de software frente DO-178C	*Framework* de caracterização de abordagens baseadas em modelo; Análise de diferentes abordagens considerando o *framework* proposto; Apresentação de ferramentas para propósitos diferentes no processo de desenvolvimento frente a norma DO-178C;	Pesquisa-ação
Trabalho relacionado 3: Pettit et al. (2013)		
Contexto	Objetivos	Método
Dificuldades na aplicação efetiva do desenvolvimento baseado em modelos em larga escala para software aeroespacial	Apresentar principais desafios evidenciados pelos autores na aplicação de modelos para software no ramo aeroespacial; Causas, consequencias e possíveis mitigações destes principais desafios;	Relato de experiência
Trabalho: pesquisa do autor		
Contexto	Objetivos	Método
Experiências e percepções da aplicação do desenvolvimento baseado em modelos para software aviônico	Analisar projetos de software aviônico para demonstrar os impactos perceptíveis pelo uso de uma ferramenta de modelagem; Identificar as experiências e percepções com o uso de modelos pela comunidade de desenvolvimento de sistemas para aeronaves;	*Survey*

Quadro 4. Comparação de trabalhos relacionados

3. Metodologia

Esta pesquisa teve enfoque quantitativo e foi realizada utilizando os métodos de levantamento e pesquisa bibliográfica. Para Hernandez Sampieri et al. (2013) o enfoque quantitativo utiliza a coleta de dados para testar hipóteses, baseando-se na medição numérica e na análise estatística para estabelecer padrões e comprovar teorias. Segundo Silva et al. (2012, p. 93), "mesmo a análise mais objetiva necessitará de discussão e espírito crítico por parte do pesquisador sobre os dados coletados, de forma a gerar conclusões claras e válidas". Os levantamentos, de acordo com Gil (2017, p. 33)l (i),

> caracterizam-se pela interrogação direta das pessoas cujo comportamento se deseja conhecer. Basicamente, procede-se à solicitação de informações a um grupo significativo de pessoas acerca do problema estudado para, em seguida, mediante análise quantitativa, obterem-se as conclusões correspondentes aos dados coletados.

A pesquisa bibliográfica, que consiste em elaborar a pesquisa com base em material já publicado (Gil, 2017), desmontrou os obstáculos existentes no desenvolvimento de software crítico para a segurança implicados pela norma RTCA DO-178, bem como a técnica de desenvolvimento baseado em modelos.

3.1. Técnica de coleta de dados

Conforme descrito por Gil (2017), para a coleta de dados nos levantamentos são utilizadas as técnicas de interrogação: o questionário, a entrevista e o formulário. Este estudo utilizou um questionário para seu levantamento que, segundo Gil (2017), entende-se como um conjunto de questões que são respondidas por escrito pelo pesquisado.

O questionário utilizado, que pode ser visto no Apêndice 1, teve como objetivo o levantamento das percepções da abordagem de desenvolvimento baseado em modelos para diversas propriedades e atividades da produção de software aviônico. Essas atividades e propriedades foram elencadas a partir da experiência do autor, bem como de acordo com os estudos realizados, em especial a partir de Rierson (2013) e Pettit et al. (2013).

A população alvo consistiu de engenheiros que atuam ou atuaram com o desenvolvimento de software aviônico mundialmente. O mesmo foi distribuído diretamente via email para contatos do pesquisador, que atua na área, e em dois grupos de interesse na área de sistemas aviônicos na rede social *LinkedIn* dos quais o autor participa. Estes grupos da rede social são denominados de *DO-178C For Avionics Engineers & Managers, from DO-178B* e *Avionics Engineers* e continham respectivamente 6289 e 7061 integrantes no momento da distribuição do questionário, totalizando 13350 pessoas, caracterizando como uma amostra não probabilística (Azevedo et al., 2011) , selecionada por conveniência. A coleta foi realizada de 14/06/2018 até 13/07/2018, recebendo um total de 80 respostas das quais 79 foram consideradas válidas para fins de análise.

Para descoberta de variáveis relevantes sobre o padrão RTCA DO-178 em projetos de software e sobre o desenvolvimento baseado em modelos, a técnica de coleta de dados foi a pesquisa bibliográfica. Esta abrange toda a bibliografia já tornada pública em relação ao tema de estudo, desde publicações avulsas, boletins, jornais, revistas, livros, pesquisas, monografias, teses, artigos científicos impressos ou eletrônicos, material cartográfico e até meios de comunicação oral: programas de rádio, gravações, audiovisuais, filmes e programas de televisão (Marconi et al., 2017).

3.2. Técnica de análise de dados

Os dados quantitativos obtidos pelo levantamento foram analisados através de técnicas estatísticas. Segundo Silva et al. (2012, p. 90), "dados quantitativos são aqueles que originalmente estão na forma numérica (valores monetários, taxas, quantidades em geral) e também aqueles coletados por meio de questões fechadas em questionários e convertidos em dados numéricos". A análise estatística é desenvolvida em dois níveis: a descrição dos dados e a avaliação das generalizações obtidas a partir deles (Silva et al., 2012). Dentre as técnicas mais comuns destacadas por Silva et al. (2012), a análise de frequência de respostas (com o uso de tabelas, quadros e gráficos) e medidas de localização e tendência central (média, moda e mediana) foram utilizadas.

4. Apresentação e análise de resultados

4.1. Perfil do respondente

A pesquisa foi respondida por 79 pessoas, das quais não se sabe o gênero por não considerar tal informação relevante para o estudo. As questões sociodemográficas do questionário incluíram informações como, por exemplo, nacionalidade, faixa etária, anos de experiência e número de projetos de software aviônico participados. Esses dados agregaram valor à pesquisa uma vez que identificaram informações fundamentais do perfil dos respondentes e permitiram avaliar as respostas em diferentes perspectivas.

No total, pelo menos pessoas de 13 países diferentes enviaram suas respostas, porém existe a possibilidade de uma diversidade ainda maior uma vez que 14 não especificaram sua nacionalidade. A Figura 8 mostra a relação de participantes por país, incluindo os quais optaram por não fazer tal identificação.

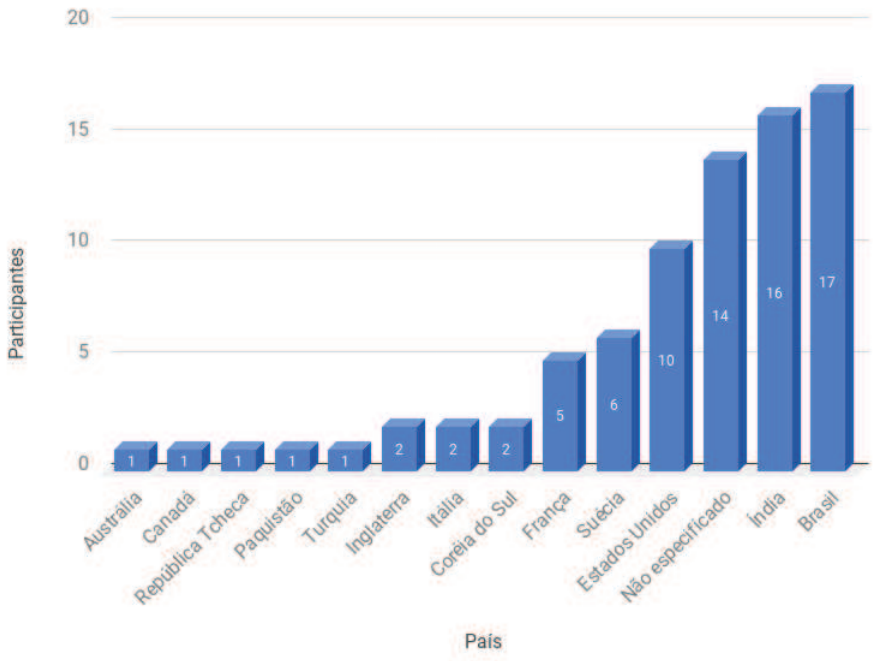

Figura 8. Nacionalidade dos participantes

A faixa etária dos participantes teve um equilíbrio bem visto pelo pesquisador, possuindo desde jovens até pessoas com maior vivência. Essa distribuição é considerada interessante principalmente porque a metodologia de desenvolvimento baseada em modelos para software aviônico é relativamente recente, como aponta Hilderman (2018) . Dessa forma, pessoas mais jovens provavelmente avaliaram a mesma com uma perspectiva diferente e mais moderna do que pessoas mais velhas, que participaram de projetos quando a metodologia estava longe de ser realidade e conseguiram ter um olhar mais crítico e comparativo. A coleta realizada não identificou a idade dos mesmos especificamente, mas utilizando intervalos. Os dados podem ser visualizados no gráfico presente na Figura 9.

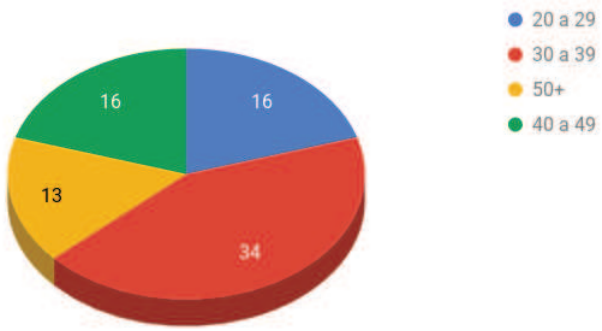

Figura 9. Faixa etária dos participantes

Idade e anos de experiência na área são caracterísitcas distintas e não necessariamente tem relação, ou seja, podemos ter pessoas novas e que já atuaram por vários anos, bem como pessoas mais velhas e com pouca experiência. O tempo despendido no desenvolvimento de software aviônico influencia diretamente nas percepções dos respondentes quanto a utilização do desenvolvimento baseado em modelos. Nesta pesquisa, os anos de experiência foram especificados pelos participantes através dos intervalos de 0 a 2, 3 a 5, 6 a 9 e mais de 10 anos. Para o pesquisador, estes intervalos classificam os mesmos, respectivamente, como pouco experientes, experientes, muito experientes e extremamente experientes.

Os resultados demonstraram que 54 dos participantes tem 6 ou mais anos de prática, isto é, são a maioria e muito ou extremamente experientes. Ter a opinião de pessoas tão habilitadas na área aumenta drasticamente a credibilidade da pesquisa uma vez que estas são fundamentadas através de suas vivências. Da mesma forma, ter a visão de pessoas com menos tempo na área também tem importância para demonstrar as percepções de quem ainda não possui vícios quanto o desenvolvimento de software aviônico e permitir uma análise comparativa. A Figura 10 mostra o gráfico com a quantidade de participantes em cada intervalo definido.

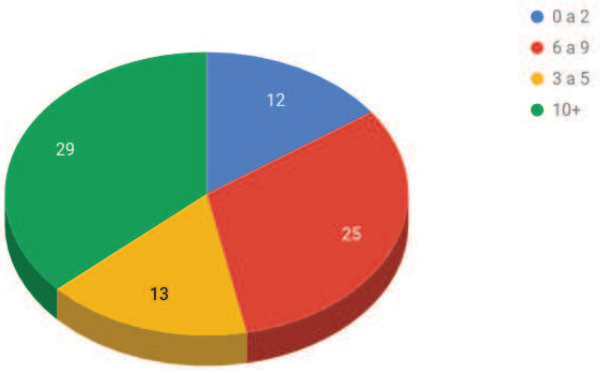

Figura 10. Tempo de experiência dos participantes

O desenvolvimento de sistemas aviônicos possui quatro principais disciplinas (sistemas, software, hardware e segurança) que foram apresentadas e analisadas no item 2.2.1. Uma vez que essa pesquisa tem enfoque na utilização de desenvolvimento baseado em modelos para produção de software, uma das questões elencou com quais das disciplinas os participantes já trabalharam, visando identificar se os mesmos possuíam experiência na área desejada. As estatísticas apontaram que 75 dos 79 respondentes possuíam experiência com a área de software, o que representa 94,9% do total. A Figura 11 apresenta as respostas obtidas, considerando que cada pessoa pode ter atuado em somente uma ou em todas as áreas.

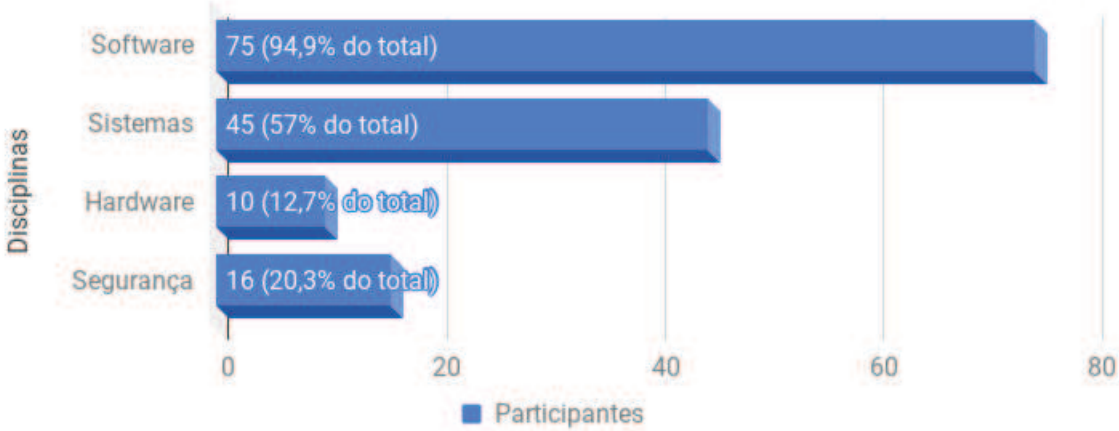

Figura 11. Disciplinas atuadas pelos participantes

Foram solicitados também em quantos projetos de software aviônico os respondentes participaram e em quantos destes houve a aplicação do desenvolvimento baseado em modelos. Questionou-se também em quantos dos projetos participados ocorreu a desistência da utilização da metodologia em questão. As informações obtidas foram relacionadas a fim de levantar, em média, quantos projetos de software aviônico que utilizaram o desenvolvimento baseado em modelos. O resultado demonstrou que aproximadamente a

cada 2 dos projetos mencionados 1 utilizou modelagem. Uma relação similar foi realizada para a taxa de desistência, que verificou-se em torno de 1 desistência a cada 4 projetos.

4.2. Impacto de desenvolvimento baseado em modelos em software aviônico

Com o objetivo de identificar as razões pelas quais os projetos deixaram de empregar o desenvolvimento baseado em modelos, a pesquisa abordou este aspecto com os participantes. Uma vez que nem todos apontaram desistência da metodologia nos projetos, a quantidade de respostas recebidas para essa pergunta foi reduzida, num total de 33. As seguintes causas prováveis levantadas pelo pesquisador foram listadas como opção de resposta por padrão:

- Projeto desistiu de utilizar modelos porque eles ficaram obsoletos ao longo do ciclo de desenvolvimento.
- Projeto desistiu de utilizar modelos porque as ferramentas de desenvolvimento foram consideradas muito caras.
- Projeto desistiu de utilizar modelos porque os engenheiros não se adaptaram ao desenvolvimento baseado em modelos.
- Projeto desistiu de utilizar modelos porque o processo não facilitava a utilização de modelos.

Uma vez que não é plausível listar todas as causas possíveis, a questão permitiu aos respondentes também mencionarem outros motivos que não os colocados por padrão. A Figura 12 demonstra as estatísticas relativas somente aos motivos pré-definidos. No Quadro 5, são apresentadas as razões compartilhadas pelos participantes que não foram listadas e complementos ao questionamento.

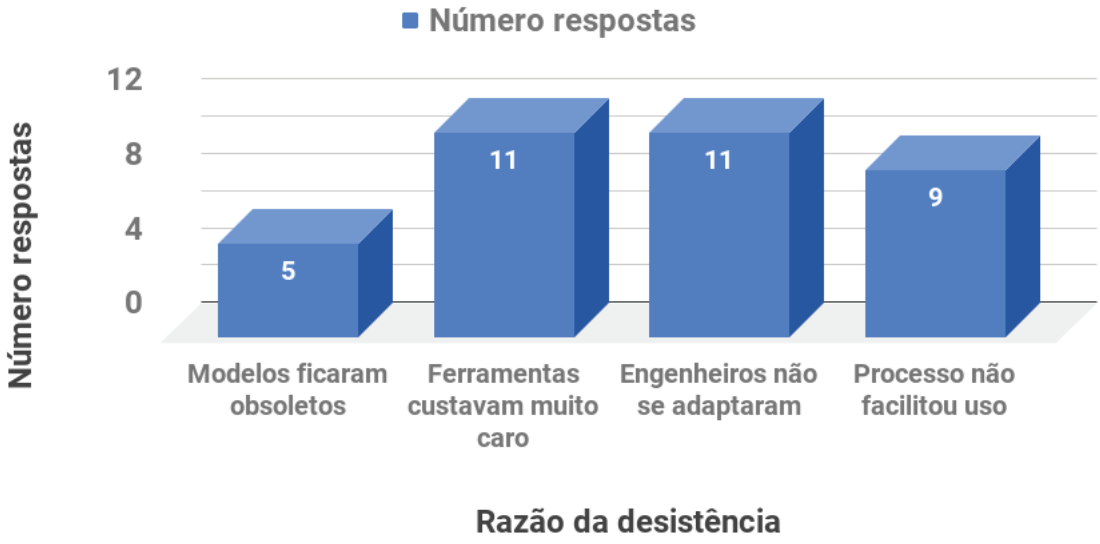

Figura 12. Respostas em motivos pré-definidos para desistência de modelos

Os participantes tiveram uma contribuição considerável para esta pesquisa com as respostas apresentadas no Quadro 5, pois um número significativo de outros motivos relevantes para a desistência do desenvolvimento baseado em modelos foram mencionados.

Número	Motivo(s) para desistir do desenvolvimento baseado em modelos
1	Alto custo de CPU e memória, além de dificultar o *debugging*
2	O código gerado consumia muitos recursos, o que não é bom quando estes são limitados. O projeto teve preferência por colocar foco em apenas um ambiente de desenvolvimento e fazer o mesmo o melhor possível ao invés de ter que dividir os recursos humanos para focar em ambientes diferentes. O ganho de utilizar o desenvolvimento baseado em modelos não é maior que o seu custo
3	Preferências do time e natureza do software
4	O projeto considerou o custo inicial para criar os modelos muito elevados
5	Alguns requisitos do cliente não se enquadravam para a utilização de modelos
6	Código fonte gerado excessivamente grandes e introdução da DO-331
7	Ineficiente e caro
8	Ônus de qualificar a ferramenta usada
9	Modelos consumiam muito tempo e tinham muitos *bugs*. As ferramentas eram difíceis de utilizar e com diversos problemas
10	Modelos eram vistos como perda de tempo, projeto preferiu focar na escrita do código somente
11	Necessita trabalho excessivo que não pode ser realizado com perfeição.
12	Processo utilizando modelos levou aproximadamente 3 vezes mais tempo para alcançar em torno de um quarto das funcionalidades comparado ao processo tradicional de codificação

Quadro 5. Outros motivos para desistência de modelos

Modelos podem ter variadas aplicações no desenvolvimento de software aviônico e inclusive terem relação, isto é, funcionarem como entrada e saída um para o outro. O mercado oferece algumas ferramentas para a criação de modelos, cada com suas particularidades, que podem ser benéficas ou prejudiciais dependendo da forma e com qual finalidade são empregadas. A Figura 13 apresenta um levantamento realizado pelo questionário que indica para quais propósitos os respondentes já usaram modelos dentre os casos mais comuns, que foram pré-definidos pelo pesquisador. Os respondentes também tinham a oportunidade de relatar outras possibilidades para os modelos, onde foram mencionadas análise operacional, requisitos e testes.

Figura 13. Finalidade dos modelos

Avaliando as finalidades apresentadas na Figura 13, podemos perceber que os modelos, na maioria dos casos, são utilizados para implementação nos projetos de software aviônico. Na visão do pesquisador, esse fato pode ser percebido uma vez que os modelos podem contribuir substancialmente para a implementação, fazendo parte da definição de questões fundamentais no desenvolvimento de software como arquitetura, estruturas de dados, algoritmos, fluxo de controle e fluxo de dados. Em vários casos, dependendo das ferramentas utilizadas para modelagem, existe a possibilidade de geração automática de código a partir das definições mencionadas, o que torna os modelos ainda mais valiosos pela garantia de que o código fonte é compatível com os modelos.

Outras funções em destaque na Figura 13 são documentação, simulação e prototipação. A questão de documentação, especialmente no desenvolvimento de software aviônico, é fundamental uma vez que estes são artefatos de software que são verificados pelas autoridades para declaração e operação do produto. Conforme mencionado por Rierson (2013), os modelos nesse caso podem contribuir como, por exemplo, documentação de design e até mesmo requisitos de baixo nível dependendo dos processos e planos de desenvolvimento seguidos. A prototipação é muito bem vista na área de software aviônico também, principalmente para que as ideias de como implementar os requisitos do sistema sejam experimentadas e validadas sem necessitar grande esforço. Ao realizar essa etapa desenvolvedores amadurecem muitas vezes design, algoritmos e soluções propostas ao conseguirem perceber detalhes que não seriam vistos sem prototipar. A simulação por sua vez permite que os desenvolvedores identifiquem erros cometidos cedo, evitando um esforço maior e desnecessário posteriormente para encontrá-los na fase de testes.

A Tabela 1 traz um resumo das ferramentas utilizadas e o número de ocorrências de cada nas respostas.

Número	Ferramenta	Ocorrências	Total de respostas	% de ocorrência
1	Simulink	46	75	61,33%
2	SCADE	35		46,67%
3	Raphasody	22		29,73%
4	Bridgepoint (xtUml)	10		13,51%
5	LabView	7		9,46%
6	VAPS	4		5,41%
7	Enterprise Architect	2		2,70%
8	Atego Artisan	1		1,35%
9	MATRIXx System Build	1		1,35%
10	Papyrus	1		1,35%

Tabela 1. Ferramentas utilizadas para criação de modelos

Dentre todas as ferramentas utilizadas pelos participantes na produção de software aviônico dispostas na Tabela 1, podemos dar destaque especial para duas: Simulink e SCADE. Estas duas ferramentas tem propostas similares, permitindo aos usuários prototiparem e criarem modelos e, a partir desses, simularem o comportamento do mesmo, além de poder gerar código pronto para integração no sistema. Para Selic (2003), uma das vantagens importantes de modelos executáveis (simulação) é que eles fornecem uma forma de experimentar cedo o sistema que está sendo desenvolvido e que esta é uma forma

de aprendermos sobre o mesmo. A possibilidade de implementação, prototipação e simulação estão entre os de maior destaque na finalidade dos modelos conforme analisado e visto na Figura 13, o que indica a busca pela maior abrangência possível dentro das etapas de desenvolvimento quando se trata de ferramentas de modelagem.

Mesmo que certas ferramentas de modelagem disponham de muitos recursos, nem sempre todos são adotados. Os processos de desenvolvimento variam de organização para organização e entre projetos de uma mesma organização, o que pode limitar os recursos utilizáveis. Outras ferramentas podem ter enfoque maior ao buscar apenas uma finalidade e serem preferidas por atenderem exclusivamente as necessidades dos projetos, cenário que provavelmente caracteriza as ferramentas menos mencionadas na Tabela 1.

As últimas perguntas do questionário aplicado foram direcionadas a classificar como a utilização do desenvolvimento baseado em modelos afeta um conjunto de propriedades e atividades num projeto de software aviônico. A classificação possuia uma escala de 0 a 10, onde 0 significava que o respondente acredita que a atividade ou propriedade em questão apenas piora ao utilizar o desenvolvimento baseado em modelos. Por outro lado, a nota 10 significava que este pensa que somente melhora. Uma classificação com nota 5 indicava a percepção de que não há efeito sobre a propriedade ou atividade em questão. Foi solicitado aos participantes também que em propriedades e atividades as quais não possuiam conhecimento ou experiência fossem deixadas em branco, podendo também classificá-las caso tivessem alguma ideia de acordo com suas impressões de outros times e já haviam ouvido falar a respeito. As respostas contabilizadas podem ser vistas na Tabela 2.

Propriedade / Atividade	**Moda**	Média	Mediana	Respostas
Design	**10**	7,7	8	75
Prototipação	**10**	7,4	8	73
Qualidade de software	**9**	6,7	7	74
Simulação	**8**	7,6	8	70
Documentação	**8**	7,2	8	74
Implementação	**8**	7,2	8	73
Testes	**8**	7,1	8	73
Comunicação	**8**	6,8	7	73
Revisões	**8**	6,7	7	73
Rastreabilidade	**8**	6,4	7	73
Manutenção	**8**	6,3	7	72
Legibilidade do código	**8**	5,0	5	72
Identificação de requisitos	**7**	6,1	6	71
Integração	**5**	6,4	7	70
Análise estática	**5**	6,1	6	67
Tempo de aprendizagem para novos membros	**5**	5,9	6	67
Análise dinâmica	**5**	5,9	5	64
Análise WCET	**5**	5,4	5	62
Performance do código	**5**	4,8	5	67

Tabela 2. Impacto do desenvolvimento baseado em modelos

Generalizando os resultados presentes na Tabela 2, pode-se afirmar que, considerando o valor de moda, a maioria das atividades e propriedades questionadas são impactadas beneficamente pela utilização do desenvolvimento baseado em modelos e algumas pode ser considerado neutro. É possível destacar dentre elas as atividades de design, simulação, prototipação, documentação, implementação e testes, uma vez que estas obtiveram média acima de 7, com moda e mediana iguais ou superiores a 8.

Considerando uma perspectiva individual das respostas, pode-se identificar situações em que a utilização do desenvolvimento baseado em modelos pode ser avaliada traumática para o pesquisador. Nestas, todas as atividades e propriedades da Tabela 2 foram avaliadas com notas baixas e houveram casos em que todas foram classificadas como 0. Houveram casos também em que nenhuma das avaliações de impacto do uso da metodologia foi preenchida. Um participante canadense com mais de 20 anos de experiência na área entrou em contato para conversação sobre as questões e argumentação de ter mantido as respostas em branco. Este julgou que, para poder responder as questões das atividades e propriedades, seria necessário saber a natureza do software que está sendo desenvolvido, bem como o tamanho do projeto uma vez que estes influenciam diretamente. Este também mencionou acreditar no uso de metodologias que se apliquem ao problema a ser resolvido, mas também pesando que muitos fatores devem ser considerados para tomar a decisão de utilizar algo ou não.

Para o pesquisador, uma das questões que poderia ter opiniões variadas refere-se ao tempo de aprendizagem para novos membros. Essa questão poderia receber uma nota 10 considerando novos membros com facilidade e experiência com modelos, fazendo com que estes se insiram no projeto e entendam seus detalhes mais rapidamente. Pelo lado contrário, podemos ter uma nota 0 para desenvolvedores que tem dificuldade ou nenhuma experiência com modelos, o que teria o efeito contrário do primeiro caso. O valor de moda igual 5 obtido como resultado pode indicar que os respondentes pesaram os dois cenários e, como ambos se equivalem de certa forma, mantiveram uma resposta neutra. Outra forma de visualizar isto é que, dada suas experiências, o uso de modelos realmente não afeta o tempo de aprendizagem, independente dos membros em questão. Para o autor, o uso de modelos reduz o tempo de aprendizagem uma vez que este permite um entendimento mais fácil do design e arquitetura do software.

Dentre os casos que ficam em evidência pelo valor de moda mais baixo, que são iguais a 5, alguns estão relacionadas ao código como a performance. Nesse caso é necessário considerar alguns fatores que podem influenciar este, como a utilização ou não de geração de código a partir dos modelos. As ferramentas de geração podem ou não ter certificados de qualificação, ou seja, geram uma saída que precisa de pouca ou nenhuma verificação dependendo de sua aplicação. A performance do código pode ser diretamente influenciada para atingir estes padrões de qualidade, uma vez que podem garantir a não ocorrência de determinadas exceções como ponteiros nulos, variáveis não inicializadas entre outras. De qualquer forma, avalia-se fundamental estudar como o gerador produz código a partir de suas diretivas de modelagem para aperfoiçoar a forma de utilização da ferramenta, visando obter a melhor performance possível.

Uma vez que a geração de código normalmente toma como referência as definições feitas no modelo e aplica padrões de nomenclatura próprios, a saída do mesmo provavelmente se caracterizará como de difícil leitura. Isto pode explicar a ocorrência de

uma moda de valor 8 na questão de legibilidade de código enquanto a média e mediana estão no valor 5. Na visão do autor o código pode ser muito mais legível quando o mesmo é escrito manualmente a partir do modelo, mas considerando um código gerado isso pode não ser verdade.

Logicamente, nos casos em que não há uso de um gerador automático de código, a legibilidade e performance do código fonte depende diretamente das capacidades de programação do desenvolvedor. Se o modelo existir, este irá determinar fatores importantes para a implementação, como uma arquitetura que deve ser seguida, um fluxo de controle ou fluxo dados que deverão ser respeitados, mas cabe também a quem implementa o que está especificado possuir um olhar crítico para identificar oportunidades de melhoria. Neste sentido há também outro fator avaliado pela pesquisa que é considerado como um dos benefícios dos modelos por Streekmann (2011) e Rierson (2013), considerado por muitos como uma das chaves para o sucesso de projetos aviônicos: a comunicação. Este fato poder ser visualizado claramente nos dados apresentados na Tabela 3, que possui as respostas obtidas no questionário sobre os principais fatores para o sucesso de projetos de software aviônico. A Tabela 2 demonstra também a comunicação com moda 8, o que reforça estas percepções.

Fator	Ocorrências	Total respostas	% de ocorrência
Requisitos maduros	59	79	74,7%
Processo bem definido	57		72,2%
Engenherios experientes	47		59,5%
Boa comunicação	46		58,2%
Uso de ferramentas adequadas	36		45,6%
Seguir os planos estabelecidos	25		31,6%
Reuso	20		25,3%

Tabela 3. Principais fatores para o sucesso de projetos aviônicos

Os fatores mencionados na Tabela 3 foram pré-determinados para escolha dos participantes de acordo com o estudo bibliográfico e experiência do autor, porém também havia a opção para descrever outros fatores. Os respondentes definiram também como chave para o sucesso os seguintes motivos:

- Rápida adaptação à mudanças
- Quantidade baixa de burocracia
- Arquitetura bem definida
- Testes bem realizados
- Automatizar processos para atingir objetivos
- Paciência
- Conhecimento da área
- Tempo e financiamento suficientes
- Revisão e atualização de requisitos

Visualizando a Tabela 3, é possível identificar também o fato de que aproximadamente 3 a cada 4 respondentes consideraram que ter requisitos maduros e um processo de desenvolvimento bem definido como fatores determinantes. O primeiro é considerado fundamental para que não sejam necessárias mudanças e adaptações bruscas ao decorrer

da implementação ou até mesmo da verificação. Dependendo da quantidade de mudanças dos requistos e respectivas funcionalidades, pode ser necessário retrabalhar o design para obter a solução mais otimizada, mas muitas vezes isto acaba não sendo possível por diversos motivos como a escassez de tempo. Nesses casos, desenvolvedores finalmente são obrigados a encontrar uma solução alternativa e que pode não se enquadrar como deveria no design e na arquitetura estabelecida. Existe também a possibilidade de que mais mudanças sejam realizadas nos requisitos, afetando exatamente o que já foi implementado de forma alternativa, tornando a situação mais crítica ainda. Enfim, requisitos são a base fundamental de qualquer projeto e a utilização destes quando não estão maduros podem ser catastróficos, conforme já apresentado no para capítulo 2.2.2.

A outra questão vital para o sucesso dos projetos aviônicos é a existência de um processo de desenvolvimento bem definido. A criação de um sistema aviônico pode ser feito de inúmeras maneiras e até não seguir processos, mas este só entrará em operação após ser aprovado em auditorias realizadas pelas entidades responsáveis. O processo bem definido permite a criação do sistema e geração dos artefatos necessários para evidenciar que o mesmo foi desenvolvido conforme planejado, atendendo a todas as especificações de segurança necessárias. Este também traz clareza ao desenvolvedor sobre quais os passos que são realizados durante o projeto, mencionando entradas necessárias e saídas esperadas. Caso o processo não esteja bem definido, há a possibilidade de que uma etapa seja realizada parcialmente ou de forma inesperada, gerando problemas nas etapas subsequentes e inevitavelmente elevando o tempo e custo de desenvolvimento.

5. Considerações finais

A técnica de desenvolvimento baseado em modelos pode ser usada em diversas partes do processo de desenvolvimento de software aviônico conforme a DO-178C, como requisitos, design, implementação, documentação, testes e análises. A aplicação em cada uma destas tem suas particularidades e trazem impactos diferentes, cabendo a cada projeto e organização definir como e se irão utilizar a metodologia, buscando os melhores benefícios que atendam os processos definidos para o projeto em questão.

Este artigo teve como objetivo analisar os impactos do desenvolvimento baseado em modelos para a produção de software aviônico a partir das diferentes percepções e experiências de profissionais que já trabalharam com esta tecnologia. Para tanto, um questionário foi criado considerando o referencial teórico pesquisado e as vivências do autor, sendo este aplicado para coletar as informações necessárias a fim de possibilitar a análise.

Com os resultados obtidos na pesquisa, pode-se concluir que, geralmente, a utilização do desenvolvimento baseado em modelos impacta de forma benéfica a produção de software aviônico, aumentando a qualidade deste e trazendo grandes ganhos especialmente em questões como design e prototipação. Pode-se deduzir também que a aplicação da metodologia depende diretamente de alguns fatores como a natureza do software em desenvolvimento, finalidades dos modelos, processo utilizado e a experiência dos desenvolvedores com modelagem. Em outras palavras, deve-se aplicar a metodologia para as atividades pertinentes em cada caso, considerando fatores como estes mencionados a fim de que os modelos sejam positivos para o projeto.

A utilização de modelagem não é uma novidade para a área de software em geral,

mas pode ser considerada relativamente nova no âmbito de software aeroespacial visto que foi recentemente formalizada através da publicação da DO-331. A modelagem de software ainda pode evoluir muito e, da mesma forma com que ocorreu com a DO-178, é provável que novas revisões da DO-331 sejam lançadas a fim de aprimorar os objetivos e esclarecer a suas finalidades nos projetos.

A maioria dos projetos de software aviônico possuem questões sigilosas, seja por proteção à propriedade intelectual das empresas ou por restrições dos clientes por exemplo, que não permitem o compartilhamento de muitas informações para estudos como o realizado neste artigo. Essa realidade é considerada uma limitação e, especificamente nessa pesquisa, afetou diretamente a elaboração do questionário. As questões aplicadas trouxeram percepções generalistas de profissionais que já trabalharam com software aviônico e estas são influenciadas pelas experiências de cada um, principalmente considerando fatores como o processo de desenvolvimento usado, natureza do projeto e quais ferramentas e como foram aplicadas. Variáveis como estas dificultaram a análise dos resultados, demandando uma visão mais ampla do autor frente os aspectos que foram considerados mais relevantes.

Tendo em vista a possibilidade de expansão e atualização sobre os modelos, bem como as experiências que irão surgir a partir das novas realidades, sugere-se como possíveis trabalhos futuros:

- Estudo exploratório sobre possíveis melhorias na DO-331 para facilitar aplicação do desenvolvimento baseado em modelos
- Estudos em profundidade sobre a aplicação dos modelos em cada uma das finalidades
- Estudo comparativo entre as ferramentas mais utilizadas para modelagem
- Estudo comparativo sobre possíveis revisões da DO-331 para demonstrar a evolução da aplicação dos modelos em software aviônico

Referências

Azevedo, D., Silva, L. V. d., and Machado, L. (2011). *Métodos e procedimentos de pesquisa : (do projeto ao relatório final).* EAD. São Leopoldo : Ed. UNISINOS, 2011.

Gil, A. C. (2017). *Como elaborar projetos de pesquisa.* Rio de Janeiro Atlas.

Hernandez Sampieiri, R., Collado, C. F., and Lucio, M. d. P. B. (2013). *Metodologia de pesquisa.* Porto Alegre AMGH.

Hilderman, V. (2018). Do-331 model based development for engineers and managers. Technical report, Afuzion.

IEEE (1990). Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84.

Marconi, M. d. A. and Lakatos, E. M. (2017). *Fundamentos de metodologia científica.* Rio de Janeiro Atlas.

Paz, A. and Boussaidi, G. E. (2016). On the exploration of model-based support for do-178c-compliant avionics software development and certification. In *2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 229–236.

Pettit, R. G. and Mezcciani, N. (2013). Highlighting the challenges of model-based engineering for spaceflight software systems. In *2013 5th International Workshop on Modeling in Software Engineering (MiSE)*, pages 51–54.

Prosvirin, D. A. and Kharchenko, V. P. (2015). Model-based solution and software engineering environment for uav critical onboard applications. In *2015 IEEE International Conference Actual Problems of Unmanned Aerial Vehicles Developments (APUAVD)*, pages 312–315.

Rierson, L. (2013). *Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance*. CRC Press.

RTCA (2018). Rtca inc. [Online]. https://www.rtca.org/. [Acessado em 23 de maio de 2018].

Santamaría, D., Alarcón, F., Jiménez, A., Viguria, A., Béjar, M., and Ollero, A. (2012). Model-based design, development and validation for uas critical software. *Journal of Intelligent & Robotic Systems*, 65(1):103–114.

Sarkis, A. and Dias, L. A. V. (2014). A set of rules for production of design models compliant with standards do-178c and do-331. In *2014 11th International Conference on Information Technology: New Generations*, pages 27–32.

Selic, B. (2003). The pragmatics of model-driven development. *IEEE Software*, 20(5):19–25.

Silva, L. V. d., Saccol, A. Z., Azevedo, D., and Machado, L. (2012). *Metodologia de pesquisa em administração : uma abordagem prática.* EAD. São Leopoldo : Ed. UNISINOS, 2012.

Sommerville, I. (2003). *Engenharia de Software - 6a edição.* Pearson.

Sommerville, I. (2011). *Engenharia de Software - 9a edição.* Pearson.

Streekmann, N. (2011). *Model-Driven Software Development*, pages 55–69. Vieweg+Teubner Verlag, Wiesbaden.

Zoughbi, G., Briand, L., and Labiche, Y. (2011). Modeling safety and airworthiness (rtca do-178b) information: conceptual model and uml profile. *Software & Systems Modeling*, 10(3):337–367.

Desenvolvimento Baseado em Modelos - Software aviônico

Esse formulário contém questões a fim de obter dados para um artigo científico que está estudando os impactos da utilização de desenvolvimento baseado em modelos na produção de software aviônico.

1. **Qual a sua nacionalidade?**

_____

2. **Qual a sua idade?**
   Marcar apenas uma oval.

   - ◯ Até 19 anos
   - ◯ 20 a 29 anos
   - ◯ 30 a 39 anos
   - ◯ 40 a 49 anos
   - ◯ 50+

3. **Quantos anos de experiência você tem com o desenvolvimento de software aviônico?**
   Marcar apenas uma oval.

   - ◯ 0 a 2
   - ◯ 3 a 5
   - ◯ 6 a 9
   - ◯ 10+

4. **Com que níveis de garantia de software você já trabalhou? Selecione mais de um se aplicável.**
   Marque todas que se aplicam.

   - ☐ DAL A
   - ☐ DAL B
   - ☐ DAL C
   - ☐ DAL D
   - ☐ DAL E

5. **Com quais das quatro principais disciplinas de desenvolvimento de sistemas aviônicos você já trabalhou? Selecione mais de uma se aplicável.**
   Marque todas que se aplicam.

   - ☐ Software
   - ☐ Sistema
   - ☐ Hardware
   - ☐ Segurança

6. **Na sua opinião, quais são os principais fatores para o desenvolvimento de software aviônico com sucesso? Selecione mais de um ou nenhum se aplicável.**
Marque todas que se aplicam.

- [ ] Boa comunicação
- [ ] Requisitos maduros
- [ ] Processo bem definido
- [ ] Seguir os planos estabelecidos
- [ ] Engenheiros experientes
- [ ] Reuso
- [ ] Utilização de ferramentas adequadas
- [ ] Outro: _____

7. **Quantos projetos de software aviônico você já participou?**
Marcar apenas uma oval.

- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5
- ◯ 6
- ◯ 7
- ◯ 8
- ◯ 9
- ◯ 10
- ◯ 11
- ◯ 12
- ◯ 13
- ◯ 14
- ◯ 15
- ◯ 16
- ◯ 17
- ◯ 18
- ◯ 19
- ◯ 20

8. **Quantos desses projetos que você participou utilizaram desenvolvimento baseado em modelos?**

Marcar apenas uma oval.

- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20

9. **Quantos desses projetos que você participou desisitiu de utilizar desenvolvimento baseado em modelos?**
   Marcar apenas uma oval.

   - ◯ 0
   - ◯ 1
   - ◯ 2
   - ◯ 3
   - ◯ 4
   - ◯ 5
   - ◯ 6
   - ◯ 7
   - ◯ 8
   - ◯ 9
   - ◯ 10
   - ◯ 11
   - ◯ 12
   - ◯ 13
   - ◯ 14
   - ◯ 15
   - ◯ 16
   - ◯ 17
   - ◯ 18
   - ◯ 19
   - ◯ 20

10. **Selecione as razões que você presenciou para o desistência da utilização do desenvolvimento baseado em modelos, se existirem.**
    Marque todas que se aplicam.

    - ☐ Projeto desistiu de utilizar modelos porque eles ficaram obsoletos ao longo do ciclo de desenvolvimento.
    - ☐ Projeto desistiu de utilizar modelos porque as ferramentas de desenvolvimento foram consideradas muito caras.
    - ☐ Projeto desistiu de utilizar modelos porque os engenheiros não se adaptaram ao desenvolvimento baseado em modelos.
    - ☐ Projeto desistiu de utilizar modelos porque o processo não facilitava a utilização de modelos.
    - ☐ Outro: _____

11. **Nos projetos de software aviônico que você participou, para quais objetivos foram utilizados os modelos? Selecione mais de uma ou nenhuma se aplicável.**
    Marque todas que se aplicam.

    - ☐ Documentation
    - ☐ Implementation
    - ☐ Prototyping
    - ☐ Simulation
    - ☐ Safety
    - ☐ Outro: _____

12. **Quais das seguintes ferramentas você já utilizou no desenvolvimento de software aviônico? Selecione mais de uma ou nenhuma se aplicável.**
Marque todas que se aplicam.

- [ ] Simulink
- [ ] SCADE
- [ ] Bridgepoint (xtUml)
- [ ] Raphasody
- [ ] LabVIEW
- [ ] Outro: _____

Nesta seção você deve classificar como a utilização do desenvolvimento baseado em modelos afeta um conjunto de propriedades e atividades num projeto de software aviônico.

A classificação possui uma escala de 0 a 10, onde 0 significa que você acredita que a atividade ou propriedade em questão apenas piora na utiliza do desenvolvimento baseado em modelos. Por outro lado, a nota 10 significa que você pensa que apenas melhora. Uma classificação com nota 5 indica que você acredita que não há efeito sobre a propriedade ou atividade em questão.

Para propriedades que você não possui conhecimento ou experiência, por favor deixa-as em branco ou classifique-as se você tiver alguma ideia de acordo com suas impressões de outros times e o que você já ouviu falar a respeito.

13. **Comunicação**
Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
Completamente pior	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Completamente melhor

14. **Documentação**
Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
Completamente pior	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Completamente melhor

15. **Rastreabilidade**
Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
Completamente pior	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Completamente melhor

16. **Prototipagem**
Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
Completamente pior	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Completamente melhor

17. **Identificaçao de requisitos**
Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
Completamente pior	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Completamente melhor

18. **Design**
Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
Completamente pior	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Completamente melhor

19. **Implementação**
Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
Completamente pior	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Completamente melhor

20. **Integração**
Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
Completamente pior	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Completamente melhor

21. **Testes**
Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
Completamente pior	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Completamente melhor

22. **Revisões**
Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
Completamente pior	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Completamente melhor

23. **Qualidade de software**
Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
Completamente pior	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Completamente melhor

24. **Simulação**
Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
Completamente pior	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Completamente melhor

25. **Manutenção**
Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
Completamente pior	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Completamente melhor

26. **Performance do código**
Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
Completamente pior	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Completamente melhor

27. **Legibilidade do código**
Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
Completamente pior	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Completamente melhor

28. **Análise estática**
Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
Completamente pior	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Completamente melhor

29. **Análise dinâmica**
Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
Completamente pior	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Completamente melhor

30. **Análise WCET (Worst Case Execution Time)**
Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
Completamente pior	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Completamente melhor

31. **Tempo de aprendizagem para novos membros (nesse caso, melhorar significa menor tempo de aprendizagem)**
Marcar apenas uma oval.

	0	1	2	3	4	5	6	7	8	9	10	
Completamente pior	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Completamente melhor