Architecture Software - Toolchain

Eric Silva

Eric Silva

Plan du cours

N°	Prof	Type	Dates	Thème
1	SI	Cours	18/02/2025	Gestion de projet et Organisation du cours
2	SI	Cours	25/02/2025	Patrons de conception et styles d'architecture
3	SF	Cours	04/03/2025	DevOps: Intégration Continue (slides et exercices)
4	SF	Cours	11/03/2025	DevOps: Intégration Continue
5	ES	Cours	18/03/2025	Evaluation des risques et définitions des fonctions de sécurité
6	ES	Cours	25/03/2025	Ecriture d'exigences, du système au logiciel, spécification des tests
7	ES	TP	01/04/2025	TP1 - Spécification des exigences du projet
8	SF	TP	08/04/2025	TP2 - Mise à jour du gitlab board en fonction des exigences
9	SF	Cours	15/04/2025	DevOps: Automatisation des tests
			22/04/2025	Vacances
10	SF	TP	29/04/2025	TP3 - Spécifications et Réalisation des tests automatisés
11	ES	Cours	06/05/2025	Processus de développement et toolchain
12	ES	Cours	13/05/2025	Cybersécurité et Communication
13	ES	Cours	20/05/2025	Tests statiques de code
14	ES	TP	27/05/2025	TP4 - Tests statiques et finalisation du projet

Plan du cours d'aujourd'hui

- La qualification des outils
- La chaîne de compilation
- Les librairies de test

Qualification d'outil

Lors de l'utilisation d'un logiciel tiers dans le développement d'un système critique, il faut garantir que ce logiciel ne génère pas d'erreurs non détectées pouvant compromettre la sécurité du système final.

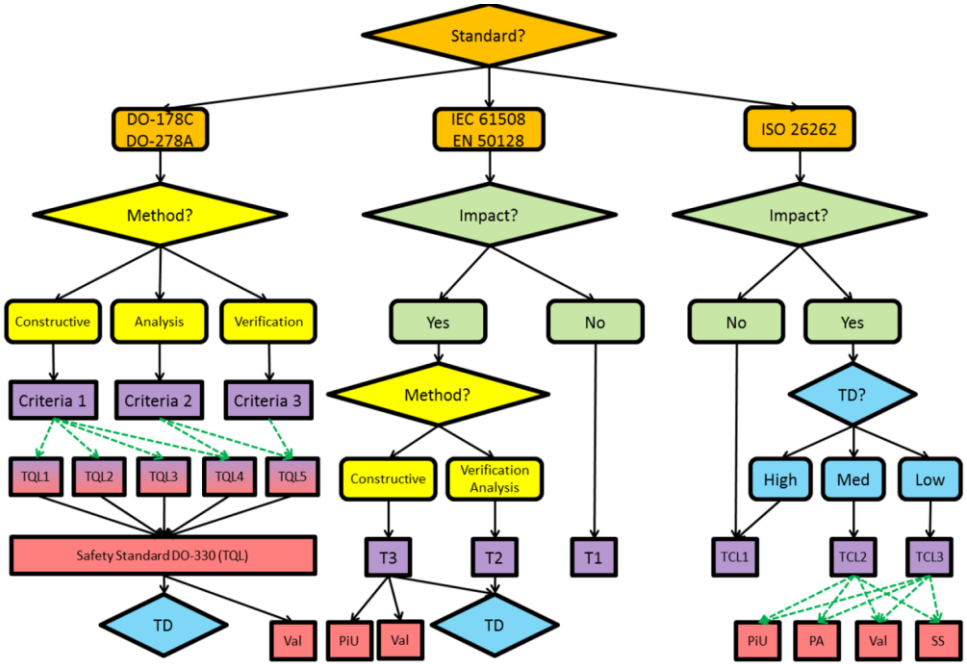
Exemples de logiciels:

- Compilateur
- Gitlab
- Word / Excel
- Visual Studio Code
- …

Source: https://www.verifysoft.com/Tool_Qualification_Plan_for_Testwell_CTC.pdf

Qualification d'outil (1)

Lors de l'utilisation d'un logiciel tiers dans le développement d'un système critique, son risque doit être estimé et mitigé.



Source: https://www.verifysoft.com/Tool_Qualification_Plan_for_Testwell_CTC.pdf

Qualification d'outil (2)

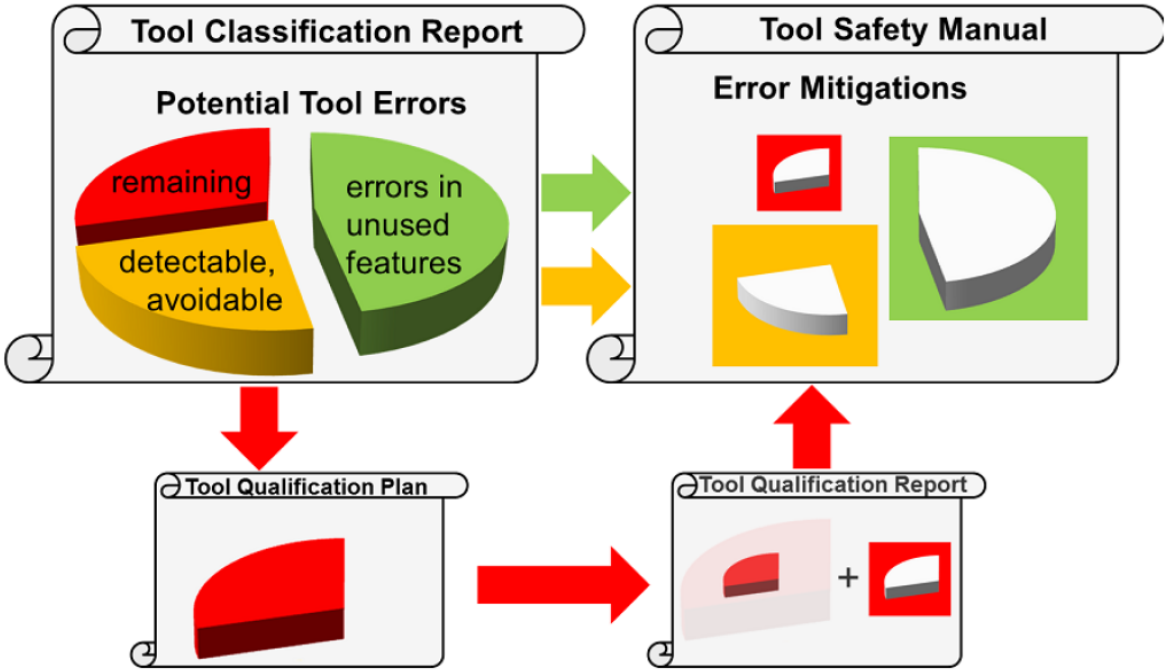
Pour évaluer la confiance en un outil donné, le modèle suivant peut être appliqué:

- Cas d'utilisation : décrire un scénario d'application de l'outil
- Fonction : une fonction de l'outil utilisée dans les cas d'utilisation
- Erreur potentielle : une erreur potentielle qui pourrait se produire lors de l'application d'un outil
- Atténuation des erreurs : vérification ou restriction appliquée pendant la phase d'utilisation de l'outil.
- Qualification : méthode permettant de montrer qu'un outil ou une fonction satisfont aux exigences spécifiées en démontrant l'absence d'erreurs potentielles.

Source: https://www.verifysoft.com/Tool_Qualification_Plan_for_Testwell_CTC.pdf

Qualification d'outil (3)

Un rapport doit ensuite être produit.



Source: https://www.verifysoft.com/Tool_Qualification_Plan_for_Testwell_CTC.pdf

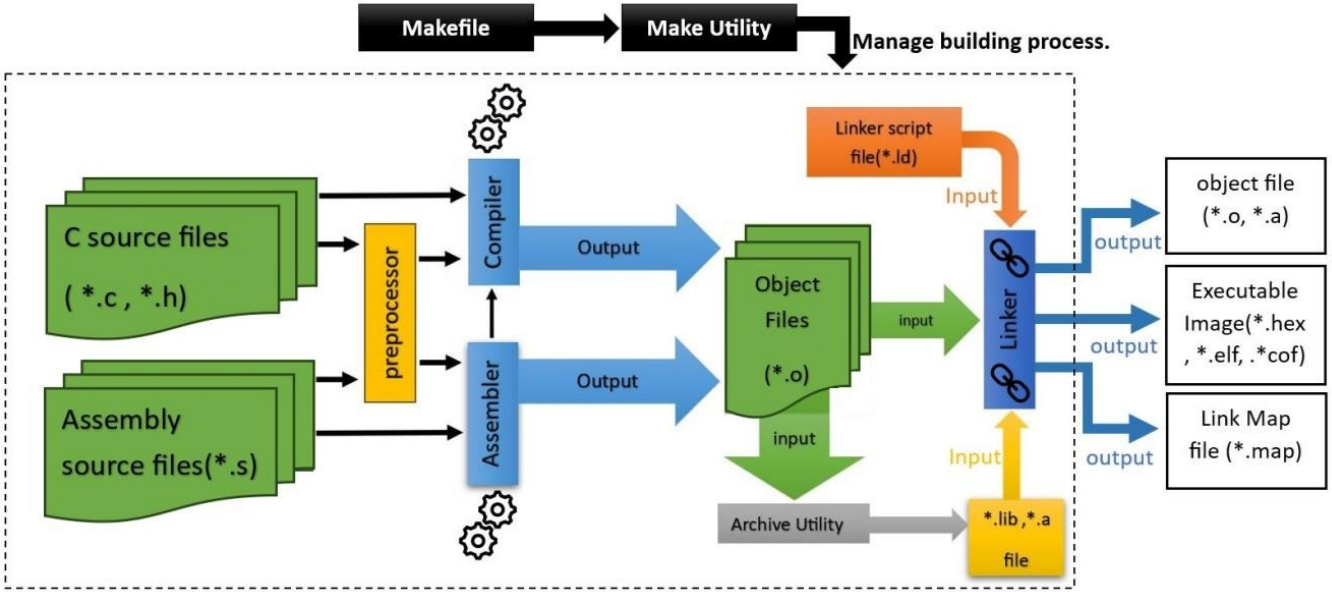
La chaîne de compilation

- Ensemble d'outils permettant de transformer un code source (C/C++) en programme exécutable.
- Elle comprend plusieurs étapes et utilitaires, adaptés à l'architecture cible.

Exemple : la chaîne GNU inclut gcc (compilateur), binutils (assembleur, éditeur de liens), glibc (bibliothèque C), gdb (débogueur).

Source: https://www.linkedin.com/pulse/buildingcompilation-process-using-gnu-arm-toolchain-mohamed-ali/

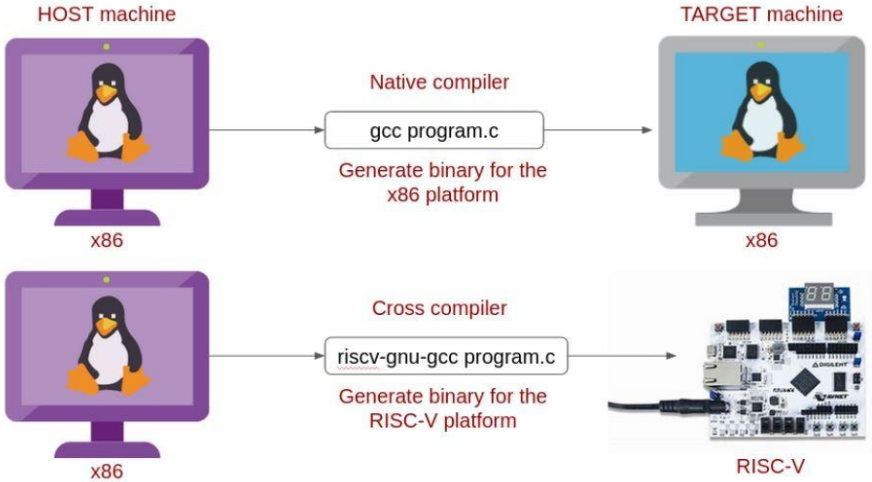
La chaîne de compilation (2)



Source: https://www.linkedin.com/pulse/buildingcompilation-process-using-gnu-arm-toolchain-mohamed-ali/

Compilation native ou cross-compilation

On parle de compilation native lorsqu'une machine compile un programme pour sa propre utilisation ou pour une machine ayant le même système d'exploitation par exemple.
Lorsque l'on compile pour un système différent, on parle de cross-compilation.

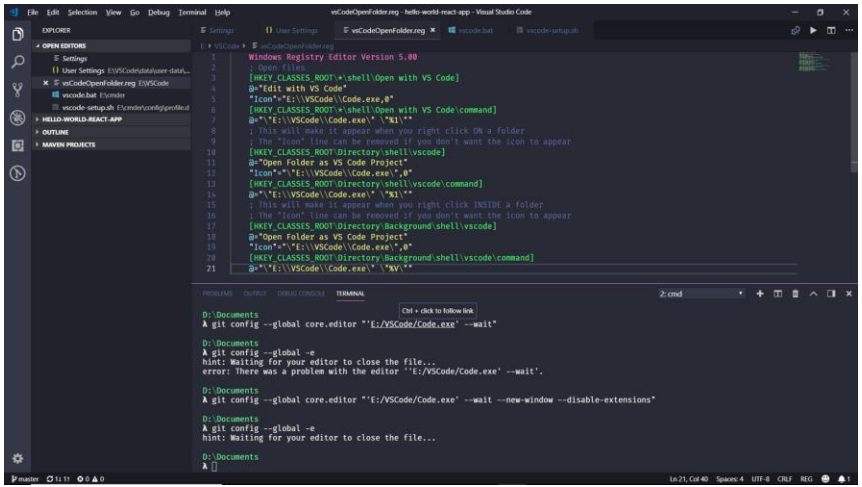


Source: https://www.linkedin.com/pulse/buildingcompilation-process-using-gnu-arm-toolchain-mohamed-ali/

Eric Silva

1ère étape, la programmation

Au moment de programmer du code, le plus souvent on utilise un “IDE” (integrated development environment) cet outil nous offre plus de flexibilité, il complète parfois notre code mais lors du développement d’un système critique il faut s’assurer qu’il n’ajoute pas d’erreurs dans notre programme.



Source: https://medium.com/@fawwazyusran/create-a-portable-ide-with-visual-studio-code-fb0c6bc198ef

1ère étape, la programmation (2)

Risques liés à l'utilisateur :

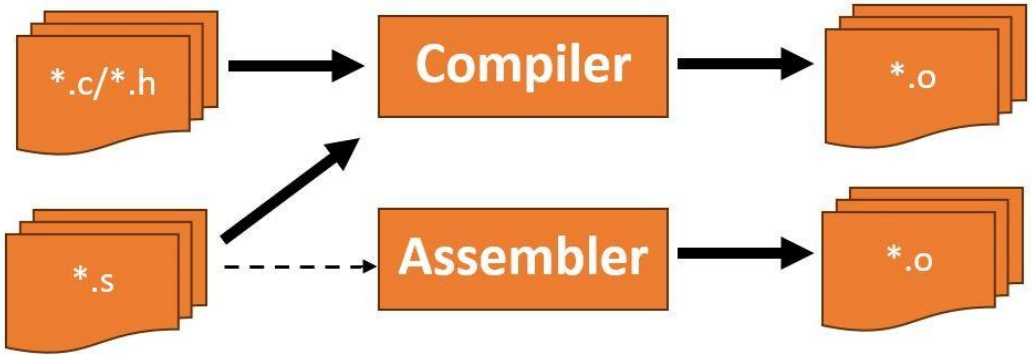
- Erreurs de logique ou de conception (ex. : mauvaise gestion des erreurs, absence d vérification des entrées)
- Vulnérabilités classiques (injection SQL, XSS, buffer overflow, mots de passe codés en dur)
- Mauvaise utilisation de fonctions dangereuses (system(), accès direct mémoire)
- Dépendance excessive aux fonctionnalités d'auto-complétion ou de correction automatique de l'IDE, pouvant masquer des erreurs logiques

Risques liés à l'outil (IDE) :

- Auto-complétion générant du code vulnérable ou non adapté au contexte critique
- Débogueur masquant des conditions de course ou des erreurs de synchronisation
- Plugins ou extensions non vérifiés pouvant introduire des failles de sécurité
- Fusion automatique de code (gestion de versions) créant des conflits ou des erreurs silencieuses
- Surcouche graphique (glisser-déposer) cachant la complexité réelle du code généré

2ème étape, compilation et assemblage

La compilation consiste à transformer un fichier contenant du code. Par exemple des fichiers “.c” ou “.h” en un fichier de type objet “.o” ce fichier contient les informations de programmation mais certains des symboles qu’il contient nécessitent encore d’être reliés à des librairies par exemple.

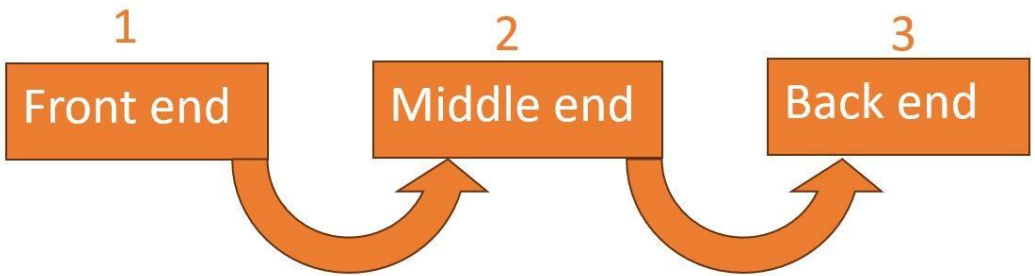


Source: https://www.linkedin.com/pulse/buildingcompilation-process-using-gnu-arm-toolchain-mohamed-ali/

2ème étape, compilation et assemblage (2)

La compilation est un processus complexe qui peut être divisé en 3 étapes effectuées consécutivement:

- **Front end** : Responsable de l'analyse du code source.
- **Middle end** : Responsable de l'optimisation du code.
- **Back end** : Responsable de la génération du code du fichier objet.



Source: https://www.linkedin.com/pulse/buildingcompilation-process-using-gnu-arm-toolchain-mohamed-ali/

2ème étape, compilation et assemblage (3)

Risques liés à l'utilisateur :

- Mauvaise configuration des options de compilation (optimisations inadaptées, flags de sécurité désactivés)
- Utilisation de versions non certifiées du compilateur pour des systèmes critiques
- Non-prise en compte des avertissements ou erreurs du compilateur

Risques liés à l'outil (compilateur) :

- Optimisations agressives supprimant des vérifications de sécurité nécessaires (ex : suppression de code "mort" critique)
- Bugs de compilation générant un code machine incorrect sans avertissement
- Corruption des fichiers intermédiaires (headers précompilés, objets)
- Incompatibilités entre options de compilation (flags contradictoires)
- Mauvaise gestion des dépendances et des bibliothèques externes

2ème étape, compilation et assemblage (4)

Chacune de ces étapes comporte des risques mais permet aussi de détecter certaines erreurs et de les corriger.

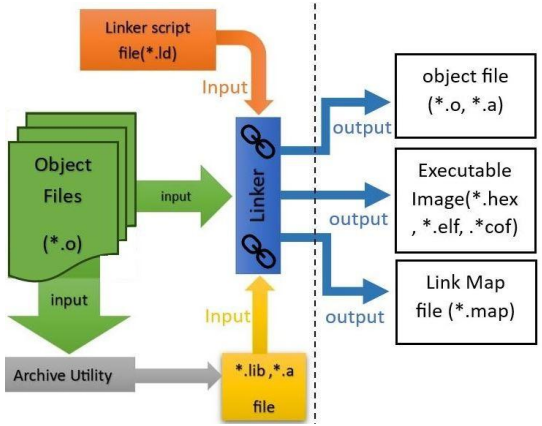
Par exemple:

- Détecter certaines boucles sans fin
- Détecter des portions de code inutilisées
- Avertir au sujet de mauvaise manipulation de variables
- …

Eric Silva

3ème étape, l'édition des liens (Linker)

Dans cette étape, le "linker" prend les fichiers objets, les bibliothèques et les autres dépendances générées lors de la compilation et les combine en un seul programme exécutable. L'édition de liens résout les références aux fonctions et aux variables, attribue les adresses mémoire et garantit que le programme ou la bibliothèque est complet et prêt à être exécuté ou utilisé.



Source: https://www.linkedin.com/pulse/buildingcompilation-process-using-gnu-arm-toolchain-mohamed-ali/

3ème étape, l'édition des liens (Linker) (2)

Risques liés à l'utilisateur :

- Oubli de l'inclusion de bibliothèques critiques ou de dépendances nécessaires
- Mauvaise configuration du linker script (adresses mémoire, sections, alignement)
- Non-vérification de la compatibilité des versions de bibliothèques

Risques liés à l'outil (éditeur de liens) :

- Gestion incorrecte des adresses mémoire ou des symboles, pouvant causer des plantages
- Mauvaise résolution des symboles externes (symboles non résolus ou mal associés)
- Limitations ou bugs dans l'éditeur de liens pouvant générer des binaires incorrects

Le choix du compilateur

Au vu de l'impact qu'il peut avoir sur l'exécutable qu'il va générer, le compilateur est un élément critique dans la chaîne de développement de systèmes critiques.

C'est pourquoi certaines entreprises ont investi un temps de développement conséquent pour proposer des compilateurs certifiés.

Chez ARM, par exemple, le compilateur de sécurité contient:

- Une librairie C
- Un kit de qualification
- Une librairie d'interface certifiée et open-source (CMSIS)
- Un système d'exploitation temps-réel
- Une librairie de test logiciel

Source: https://www.isit.fr/fr/article/iso-26262-repondre-aux-exigences-de-la-norme-de-surete-de-fonctionnement-des-vehicules-automobiles.php

Les librairies de test

Vérification de la fiabilité et de la sécurité

- Vérifier l'intégrité d'exécution du processeur et du système logiciel, en détectant la présence de défauts matériels ou logiciels au démarrage et pendant l'exécution.
- Répondre aux exigences des normes de sécurité fonctionnelle en fournissant des routines de test efficaces qui valident que le matériel fonctionne correctement avant et pendant l'exécution des applications critiques.

Automatisation et couverture des tests

- Intégration des tests dans les applications, que ce soit sur un système bare-metal ou avec un système d'exploitation.
- Elles permettent d'automatiser les tests, d'obtenir des résultats rapides (pass/fail), et d'assurer une couverture diagnostique élevée, essentielle pour la certification des systèmes critiques.

Support au développement et à la certification

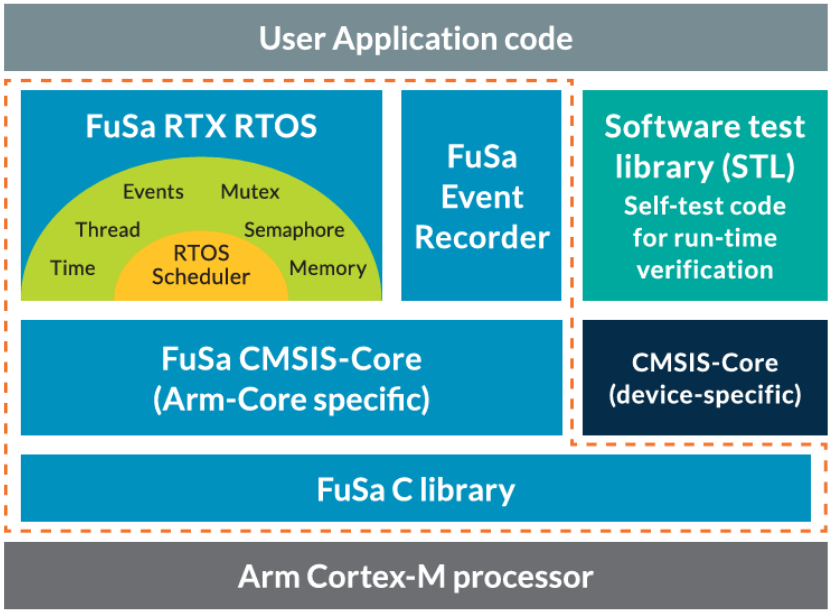
- ARM propose des librairies certifiées (par exemple, bibliothèque C certifiée avec certificat TÜV SÜD) pour accélérer la mise sur le marché et simplifier la justification de la chaîne d'outils lors de l'audit de sécurité.
- Les rapports de test et de défauts fournis avec ces librairies aident à démontrer la conformité du compilateur et de la bibliothèque aux standards ISO, et à documenter les éventuels écarts ou défauts connus.

Source: https://www.isit.fr/fr/article/iso-26262-repondre-aux-exigences-de-la-norme-de-surete-de-fonctionnement-des-vehicules-automobiles.php

Exemple -> ARM FuSa RTS

Arm FuSa RTS: Run-time system for functional safety

Software components certified for safety-critical applications



FuSa RTS components certified with safety Arm C/C++ Compiler

Covered safety standards:

+ Automotive: ISO 26262, ASIL D
+ Industrial: IEC 61508, SIL 3
+ Medical: IEC 62304, Class C
+ Railway: EN 50128, SIL 4

Supported processors:

+ Cortex-M0/M0+
+ Cortex-M3
+ Cortex-M4
+ Cortex-M7

Source: https://www.tecnologix.it/it/arm-keil-fusa-rts.html

Continuation du TP

Rappel de la notation:

Le projet se compose de 4 parties principales avec les objectifs suivants:

- TP1 (25% de la note du projet): Evaluation des risques et définition des exigences de sécurité pour le produit / projet en question.
- TP2 (25% de la note du projet): l'adoption de bonnes pratiques d'intégration continue et d'utilisation de gitlab pour le projet selon les consignes du cours.
- TP3 (25% de la note du projet): conception et la réalisation de tests automatiques pour ce projet.
- TP4 (25% de la note du projet): Tests statiques, justification des choix et finalisation du projet.