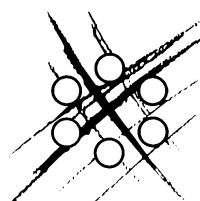


USER DOCUMENTATION :

Plan de Gestion de Configuration de TrioCFD v1.9.1




Code Version	Date	Code manager J. DARONA	Authors
v1.9.1	12 décembre 2022		J.DARONA
DES/ISAS/DM2S CEA SACLAY 91191 GIF-SUR-YVETTE CEDEX		<i>Input file :</i> PGC_TrioCFD.tex <i>Software :</i> TrioCFD	
		DES/ISAS/DM2S/STMF/LMSF/UD	

Table des matières

Table des matières	2
Table des figures	5
Liste des tableaux	6
I Introduction - Historique du code	6
I.1 Un peu d'histoire	8
I.2 Présentation générale et cartographie	11
II Définitions	14
II.1 Définition des termes utilisés	15
II.2 Définition des acteurs	18
III Outils	19
III.1 Tracker informatique - TULEAP	20
Constitution du projet TrioCFD	20
Fonctionnement général du BugTracker informatique	21
Constitution d'une fiche de demande d'intervention	21
États de traitement d'une fiche de Demande d'Intervention	23
III.2 Suivi et archivage des études	25
Objectifs du suivi et de l'archivage des études	25
Constitution d'une fiche de Suivi d'Etude	25
États de traitement d'une fiche de suivi d'étude	29
Archivage des études	30
III.3 Outil de gestion de versions (GIT)	32
Définition de la gestion de versions	32
Logiciel actuel de gestion de versions de TrioCFD : GIT	33
Organisation générale et Workflow de TrioCFD sur GIT	33
III.4 Messagerie	36
III.5 Interfaçage pour l'utilisateur et le développeur	37
Installation et compilation de TrioCFD	37
Lancement d'un cas test	37

IV	Description des Outils de Vérification et Validation	38
IV.1	Parc des machines	39
IV.2	Base de Vérification	41
	Objectifs et étapes de la Vérification	41
	La base de cas-tests de TrioCFD	41
	Fréquence de lancement de la base de cas-tests de TrioCFD	42
	Méthodologie de lancement de la base de cas-tests de TrioCFD	42
	Dépouillement des résultats de la base de cas-tests de TrioCFD	44
IV.3	Base de Validation de TrioCFD	47
	La base de validation de TrioCFD	47
	Fréquence de lancement de la base de validation de TrioCFD	48
	Méthodologie de lancement de la base de validation de TrioCFD	48
	Dépouillement des résultats de la base de validation de TrioCFD	50
V	Documentation	51
V.1	Appui sur la documentation TRUST	52
V.2	Documentation spécifique à TrioCFD	53
	Release Notes	54
	TrioCFD Reference Manuel	54
	Plan de développement TrioCFD 2020-2025	54
	Description des modèles	55
	Rapport de validation	55
	Plan de Gestion de Configuration	55
VI	Méthodologies et Procédures	55
VI.1	Processus de développement	56
	Étape 1 : Spécifications	56
	Étape 2 : Conception	57
	Étape 3 : Réalisation	57
	Étape 4 : Validation	58
	Étape 5 : Relecture et Intégration	59
VI.2	Processus de Livraison	60
	Étape 1 : Vérification et Validation de la non-régression	61
	Étape 2 : Finalisation de la documentation et dernières intégrations	61
	Étape 3 : Génération de l'archive et livraison via SourceForge/GitHub	61
	Étape 4 : Mise à disposition sur les Clusters	61
VI.3	Demande de maintenance	62
	Prise en compte et analyse des demandes	62
	Traitement des Assistances Aux Utilisateurs (AAU)	62
	Traitement des Maintenances Correctives (MC)	63
	Traitement des Maintenances Evolutives (ME)	63
	Traitement des DI LIV	64
	Traitement des DI FOR	64

VII	Communication	65
VII.1	Site TrioCFD	66
VII.2	GT et réunion de développement	67
VII.3	Séminaires	68
VIII	Conclusion	68
	Bibliographie	70

Table des figures

I.1.1	Historique de Trio_U. code couleur : version livrée - outils de visualisation et maillage - outils de documentation et validation - outils informatiques et de couplage - physique et numérique	9
I.1.2	Historique de TrioCFD	10
III.1.1	Structure de la base documentaire sur TULEAP	20
III.1.2	Récupération de l'adresse du dépôt GIT via TULEAP	21
III.1.3	Création d'une Pull Request depuis Tuleap	21
III.1.4	Champs à renseigner pour la création d'une Demande d'Intervention TrioCFD	22
III.1.5	Vue générale du BugTracker TrioCFD et de quelques fiches de Demandes D'Intervention en cours	23
III.1.6	Workflow du BugTracker TrioCFD	24
III.2.1	Constitution d'une fiche de Suivi d'Etude - partie générale	26
III.2.2	Constitution d'une fiche de Suivi d'Etude - partie contexte	26
III.2.3	Constitution d'une fiche de Suivi d'Etude - partie informations sur l'étude	27
III.2.4	Constitution d'une fiche de Suivi d'Etude - partie avancement de l'étude	27
III.2.5	Constitution d'une fiche de Suivi d'Etude - partie finalisation de l'étude	28
III.2.6	Constitution d'une fiche de Suivi d'Etude - partie pièces jointes	28
III.2.7	Constitution d'une fiche de Suivi d'Etude - partie permissions	28
III.2.8	Constitution d'une fiche de Suivi d'Etude - numéro d'identification de la fiche	29
III.2.9	Workflow du gestionnaire de suivi des études de TrioCFD	30
III.2.10	Organisation de la sauvegarde des études sous Titania	31
III.3.1	Workflow de TrioCFD sous GIT pour une branche de développement - en vert : tâches développeur - en jaune : tâche relecteur - en bleu : tâche intégrateur - en gris : tâches Atelier Logiciel	35
III.3.2	Workflow général de TrioCFD sous GIT	36
IV.2.1	Exemple de bilan du lancement de la base de vérification lancée en local	43
IV.2.2	Page html générée pour l'analyse quotidienne de la base de vérification de TrioCFD	46
IV.3.1	Méthodologie de lancement du Pipeline de Validation de TrioCFD sous Jenkins	48
IV.3.2	Les différentes étapes du processus de Validation de TrioCFD sous Jenkins	49
IV.3.3	Dépouillement résultats de validation - partie 1 : Summary	50
IV.3.4	Dépouillement résultats de validation - partie 3 : Failed PRMS	50
IV.3.5	Dépouillement résultats de validation - partie 4 : Failed PRMS au niveau de la comparaison des PDF	51
IV.3.6	Dépouillement résultats de validation - partie 4 : Failed PRMS, exemple de comparaison pixel à pixel	51
IV.3.7	Dépouillement résultats de validation - partie 5 : Successfull PRMS	51
V.1.1	Release notes de TRUST	52
VI.1.1	Exemple d'échange lors de la relecture d'une branche	60

VII.1.1	Structure actuelle du site TrioCFD.	66
VII.1.2	1 ^{ère} maquette de la nouvelle structure du site TrioCFD.	67

Liste des tableaux

I.1.1	Evolution des performances de calcul de la plateforme TRUST/TrioCFD	11
I.2.1	Cartographie des BALTIKS et sous-BALTIKS de TrioCFD	13
III.5.1	Options disponibles pour le lancement d'un cas-test	38
IV.1.1	Parc des machines de tests de TrioCFD	40
IV.2.1	Liste des options utilisées par l'atelier logiciel pour le lancement de la base de vérification	44

I. Introduction - Historique du code

LA Gestion de Configuration Logicielle (GCL) est une discipline de management de projet qui permet de définir, d'identifier, de gérer et de contrôler les outils de configuration tout au long du cycle de développement d'un logiciel. Cette gestion de configuration logicielle est régie par la norme internationale ISO 10007 :2017 [3]. Le respect des normes internationales en terme de gestion de configuration est indispensable pour tout code, d'autant plus lorsque celui-ci est utilisé dans le cadre de la sûreté nucléaire comme l'est TrioCFD. Elle a pour objectif de répondre à la question : " Quelqu'un a obtenu un résultat. Comment le reproduire ? " Le plus souvent, il ne s'agit pas de reproduire à l'identique, mais de reproduire avec des modifications incrémentales. La question est donc de comparer des résultats et d'analyser leurs différences.

La gestion de configuration du logiciel se concentre sur les aspects informatiques du logiciel et du système qui le concerne. Ainsi, la GCL s'appuie sur la gestion de version pour pouvoir identifier avec fiabilité la version du logiciel utilisé, mais prend également en compte l'environnement matériel (machines hôtes, équipements en interface) et système (système d'exploitation, type de réseau,...) dans lequel celui-ci fonctionne.

Elle s'attache également à tracer le suivi des évolutions (correctifs, évolutions) en regard des adaptations du produit. A ce titre, un outil de type *système de suivi de problèmes (issue tracking system)* est fortement recommandé dans le processus de gestion. Les adaptations qui en découlent se font en veillant en maintenir à jour la matrice de conformité qui garantit l'assurance fonctionnelle du produit.

En résumé, gérer la configuration d'un logiciel consiste à gérer :

- les différentes versions de ses composants (code, outils, données de vérification/validation,...)
- sa documentation
- les anomalies et les réponses apportées à leur résolution
- les demandes de modification ou d'évolution
- les environnements : espaces de développement, de vérification, de validation, de production

Cela consiste également à définir les règles de passage d'un environnement à l'autre. Cette méthodologie apporte au génie logiciel les moyens de réaliser un produit logiciel avec une qualité et une maîtrise du processus de développement élevées.

La Gestion de Configuration Logicielle permet de nombreux bénéfices :

- fournit une approche disciplinée et documentée pour définir, organiser et maintenir les éléments applicatifs,
- garantit l'intégrité des applications,
- assure que les versions précédentes de tout livrable contrôlé par configuration peuvent être restaurées et recréées,
- assure que tous les changements ne sont réalisés que lorsque cela est requis et seulement par les personnes autorisées.

Le Plan de Gestion de Configuration (PGC) permet, quant à lui, de fournir à l'équipe de développement et de validation du logiciel une méthodologie de travail et une description de l'utilisation des outils afin de répondre aux exigences de fiabilité qui leur sont demandées. LE PGC est utilisé comme base pour réaliser l'ensemble des activités sur le code (maintenance, développement, correction, gestion de version, livraison,...).

L'objectif de ce Plan de Gestion de Configuration (PGC) de TrioCFD est donc de définir, pour les différents acteurs intervenants sur TrioCFD, les méthodologies pour résoudre les différentes actions qu'ils ont en charge, les outils à leur disposition ainsi que la documentation nécessaire à la bonne utilisation du code.

Pour ce faire, nous commencerons par nous intéresser aux différents termes et acteurs de TrioCFD, puis aux outils de gestion. Les outils techniques spécifiques seront ensuite décrits ainsi que la méthodologie de vérification et de validation. Nous finirons par la description de la conduite à tenir pour chacune des actions possibles sur le produit et les outils de communication autour de TrioCFD. Mais débutions, tout d'abord, par un petit historique de TrioCFD et une présentation générale de la structure du code.

I.1

Un peu d'histoire

LE développement de TRIO_U a débuté en 1994 au CEA de Grenoble avec l'ambition d'unifier TRIO VF, développé à Grenoble, Trio EF, développé à Saclay, et Genepi.

L'idée initiale était de créer un seul code unifié (d'où le _U de TRIO_U) code dans un langage plus récent à savoir le C++ puisque TRIO VF et Genepi étaient en Esope, TRIO EF en FORTRAN. Une première maquette avait été réalisée en se basant sur le principe de créer une classe C++ par maille. Or, cette méthodologie nécessitait des ressources machine beaucoup trop conséquentes pour les calculs fins. Un nouveau maquetage a été alors réalisé avec une classe C++ par problème (Conduction, Hydraulique,...), par équation (*e.g.* EDP), par opérateur (gradient, divergence, laplacien,...), par variable physique (*rho*, *mu*,...),... C'est cette seconde structure qui a été finalement retenue et qui est, aujourd'hui encore, appliquée dans TRUST et ses Baltiks.

En ce qui concerne l'unification des codes, celle-ci ne s'est finalement pas faite et TRIO_U a donc débuté uniquement avec TRIO VF et été développé sur Grenoble jusqu'à la migration du service de Thermohydraulique de Grenoble vers Saclay dans les années 2010. Genepi est resté un code indépendant. Quant à TRIO EF, son développement n'a pas été poursuivi.

Au fil des années, les modèles, fonctionnalités et outils du code ont été enrichis progressivement. L'historique de ces principales améliorations sont précisées dans la frise chronologique [I.1.1](#).

En 2015, TRIO_U v1.7.1 a été scindé en deux parties : TrioCFD et TRUST. Cette séparation entre TRUST et TrioCFD a été faite de manière à ce que le code TRUST contienne toute la partie **noyau logiciel** (**kernel**) à savoir les opérateurs, les solveurs, la discrétisation, les outils de maillage et de post-traitement, les schémas en espace et en temps et le parallélisme. TRUST est ainsi capable de résoudre de manière autonome des problèmes laminaires monophasiques incompressibles ou quasi-compressibles en 2D ou 3D. TrioCFD regroupe, quant à lui, la partie **modèles physiques poussés pour la CFD** comme la turbulence (LES et RANS), le diphasique (Front-tracking et Interface diffuse), les interactions fluide-structure (méthode ALE),... Ces modèles physiques sont rangés dans des modules distincts appelés **Baltik** faisant de TrioCFD un code modulaire.

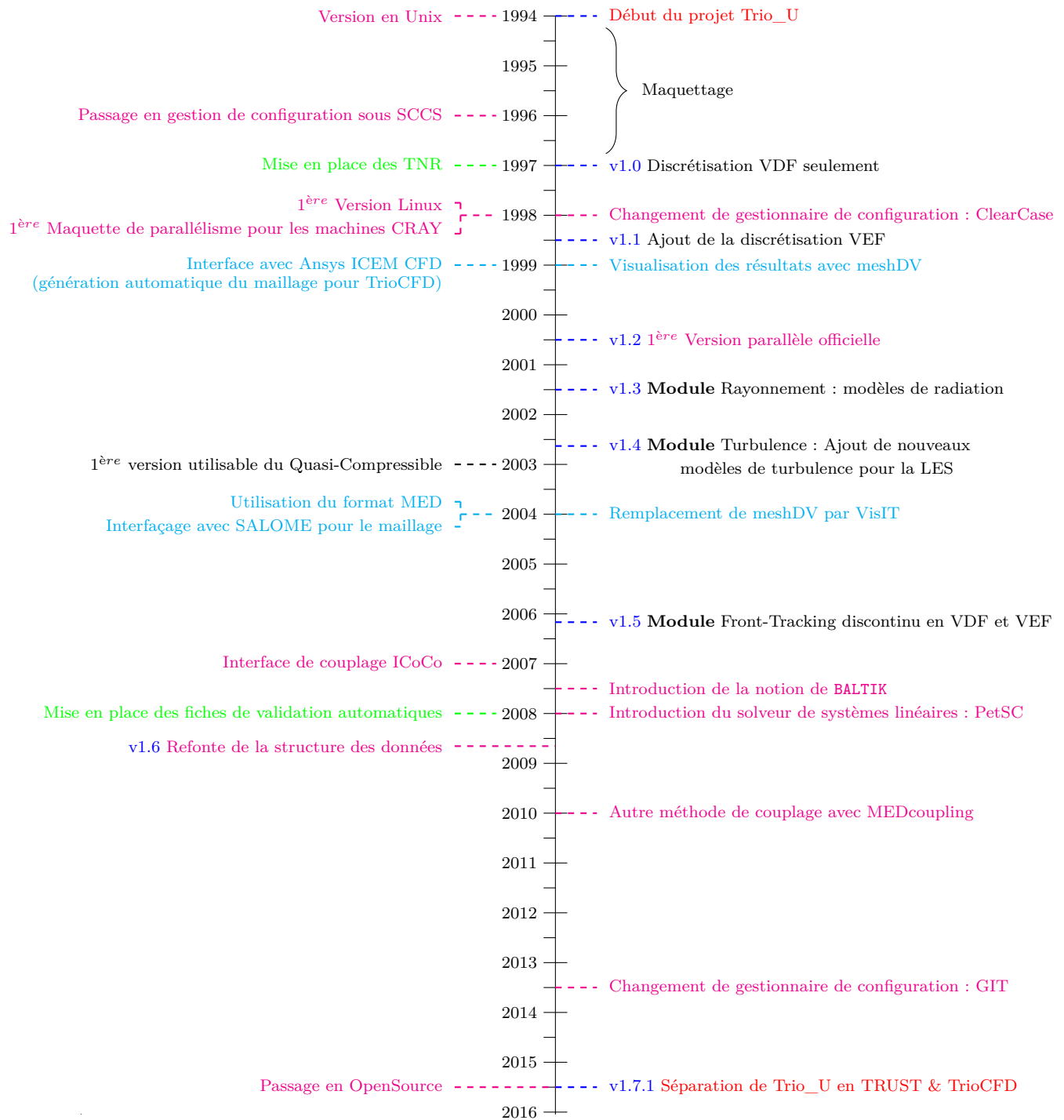


FIGURE I.1.1: Historique de Trio_U. code couleur : version livrée - outils de visualisation et maillage - outils de documentation et validation - outils informatiques et de couplage - physique et numérique

Malgré cette scission, TrioCFD reste très proche de TRUST autant au niveau des outils que de la gestion quotidienne. Les deux équipes de développement s'attachent à sortir les versions en même temps pour faciliter le processus de livraison. Chaque projet est géré par une forge TULEAP propre (TRUST & TrioCFD), il existe une forge dédiée à l'administration des deux projets (TRUST admin CEA). Une bonne partie des documents utiles tout comme les formations utilisateurs et développeurs sont communes pour TrioCFD & TRUST.

Depuis 2015 et la scission de TRUST et TrioCFD, les 2 codes ont continué à évoluer, pour TRUST, sur les aspects outils, performances et optimisations et pour TrioCFD, sur les modèles physiques et la documentation. Les évolutions de TRUST depuis 2015 ne sont pas détaillées dans ce présent document puisque ce Plan de Gestion Logiciel concerne TrioCFD. Il est toutefois à noter qu'en 2020, deux améliorations majeures ont été apportées à TRUST :

- possibilité de lancement de calculs sur la partie GPU des processeurs
- passage du code en 64 bits

Pour TrioCFD, les évolutions majeures sont explicitées sur la frise chronologique [I.1.2](#).

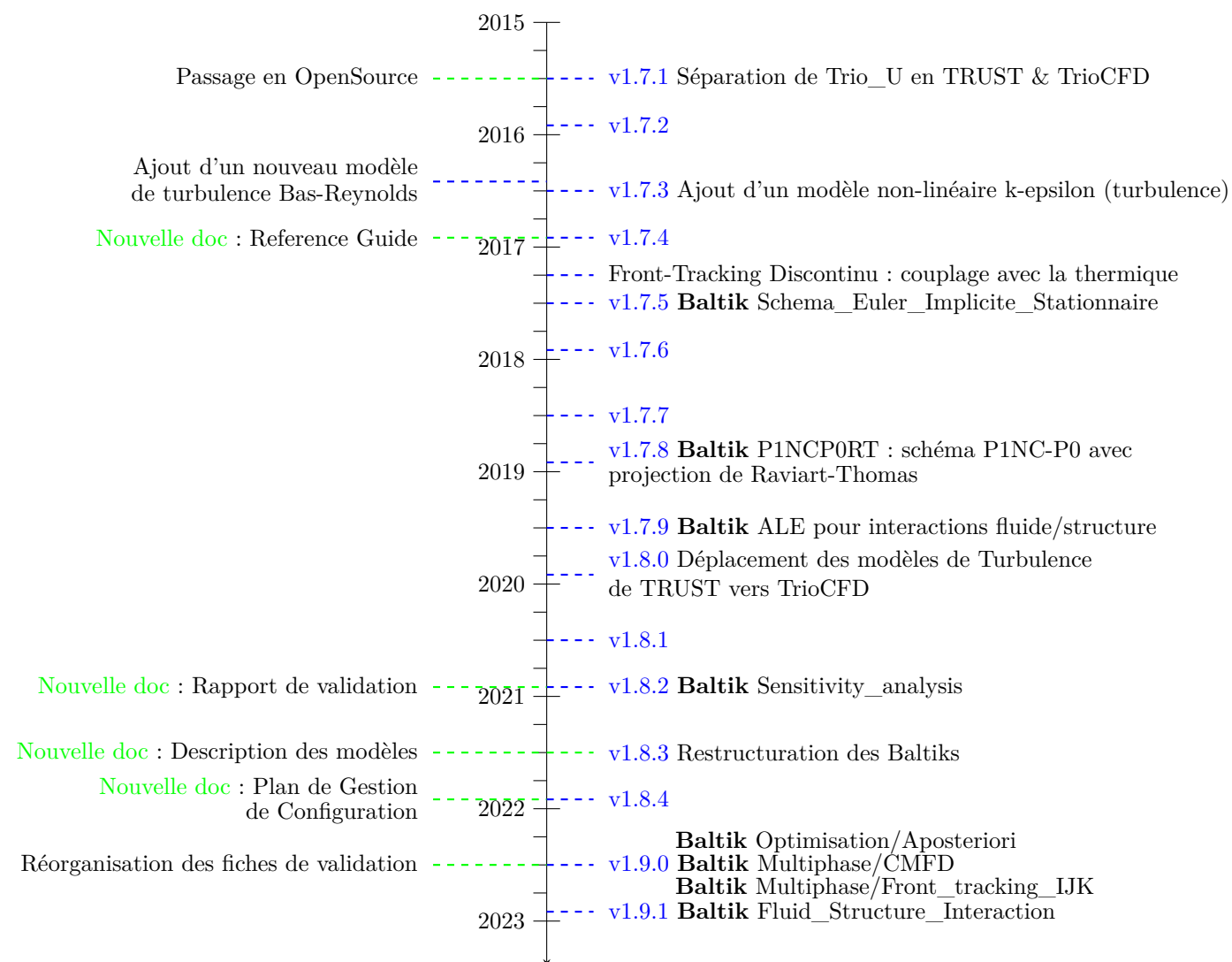


FIGURE I.1.2: Historique de TrioCFD

Il est à noter que même si TRUST supporte la partie "schémas numériques", lorsqu'un nouveau schéma est amené à être développé pour des besoins TrioCFD, l'élaboration de ce nouveau schéma est, dans un premier temps développé, vérifié et validé dans TrioCFD avant d'être reversé dans TRUST si d'autres Baltiks sous base TRUST (FLICA5, Genepi2,...) sont intéressés par ce schéma.

Depuis la création du projet, de nombreuses améliorations sur la parallélisme du produit ont permis d'augmenter, de manière très importante, les capacités de calcul. La plateforme TRUST/TrioCFD est maintenant en mesure d'effectuer des calculs haute performance (en anglais : High Performance Computing ou HPC). Les progrès récents (2020-2021) sur cet aspect viennent de 3 améliorations majeures :

- **l'amélioration des processus d'entrées/sorties** : le code utilise une base de processeurs MPI pour écrire certaines de ses données de sortie. Cela signifie qu'il est nécessaire d'avoir autant de processeurs MPI que de fichiers pour un seul calcul. Si cela est acceptable pour de petites simulations (jusqu'à 500-1000 nœuds MPI), cette méthodologie n'est plus envisageable lorsque les calculs sont lancés sur 10 000 nœuds MPI ou plus. Grâce au support du format HDF5, le processus a été rationalisé. La structure des données reste similaire (une donnée par processeur) mais un fichier physique donné est désormais remplacé par un ensemble de données correspondant, dans un seul fichier HDF5. Cette stratégie a été mise en place dans les cas/situations les plus gourmands en terme de ressources machine (fichiers de vérification, fichiers de sauvegarde/reprise, stockage des informations de division de domaine,...) permettant ainsi une réduction drastique de nombre d'i-nodes utilisés sur le système de fichiers du cluster.
- **le portage du code sur des identifiants 64 bits** : pour les tailles de maillage nécessaires dans une simulation massive, le format 32 bits s'est avéré trop restrictif. En effet, les diverses entités du maillage (sommets, faces, volumes) d'un maillage de 2 milliards d'éléments ne peuvent pas être indexées en utilisant seulement 4 octets (32 bits). Le noyau du code C++ ainsi que divers outils accompagnant la plateforme (notamment les plugins de visualisation pour le logiciel VisIt) ont été portés sur une indexation en 64 bits.
- **l'optimisation de la mémoire** : la plateforme TRUST/TrioCFD s'appuie fortement sur la bibliothèque PETSc (manipulation des vecteurs et matrices denses et creuses et résolution de systèmes linéaires) et certaines des matrices du calcul, notamment la matrice jacobienne utilisée dans les schémas numériques implicites, n'étaient, jusqu'à présent, pas remplies de façon optimale. Des optimisations ont été faites dans ce sens.

Grâce à ces améliorations sur la parallélisation et les performances, TRUST/TrioCFD est désormais capable de résoudre des simulations extrêmement fines en utilisant la parallélisation massive. Le tableau [I.1.1](#) retrace les évolutions en terme de puissance de calcul de la plateforme.

Année	nombre d'éléments du maillage	nombre de cœurs
1999	10 millions	
2010	500 millions	10 000
2020	1 milliard	17 000
2021	2 milliards	50 000

TABLE I.1.1: Evolution des performances de calcul de la plateforme TRUST/TrioCFD

I.2

Présentation générale et cartographie

DEPUIS la séparation de TRUST et TrioCFD, TrioCFD est devenu un **Baltik** (**B**uild an **A**pplication **L**inked to **T**rio_ **U** **K**ernel) de TRUST. Cela implique que TrioCFD hérite de tous les outils de TRUST à savoir :

⇒ La discrétisation :

- Finite Volume Difference (VDF) ou Volumes Finis
- Finite Volume Element (VEF) ou Eléments Finis
- PolyMach

⇒ Conditions aux limites : pour les parois et le fluide**⇒ Schémas en temps :**

- explicite : Euler, Runge-Kutta, Adams-Bashforth, Crank-Nicholson
- semi-implicite
- implicite : Euler, Adams-Moulton, backward differentiation

⇒ Schémas en espace pour la convection :

- jusqu'au 4^{ème} ordre
- upwind, centered stabilized, MUSCL, QUICK, ...

⇒ Les outils de de génération de maillage :

- l'outil interne à TRUST pour les cas les plus simples
- SALOME [1]
- Gmesh [2]

⇒ Post-traitement :

- sur les variables principales
- sur les propriétés physiques
- sur (quasiment) ce qu'on veut via des sondes définies dans le jdd
- visualisation : SALOME, VisIT, GnuPlot

⇒ Cas tests automatisés :

- fiches de validation (.prm)
- génération d'un rapport pdf par prm
- comparaison pixel à pixel des pdf pour le processus de validation

⇒ Suivi de l'impact des autres BALTIKs sur TrioCFD**⇒ HPC (High Performance Computing) :**

- parallélisme massif (MPI)
- présent sur les calculateurs CEA (cluster CURIE : 2 petaFLOPS - cluster AIRAIN : 420 teraFLOPS)

Tous ces outils étant hérités de TRUST, leur gestion est du ressort de l'équipe TRUST. Par conséquent, leur processus ne sera pas décrit dans ce présent PGC. Toutefois, une description de certains de ces outils sera faite dans les sections suivantes.

TrioCFD est en langage C++ pour la partie source du code tandis que les procédures sont en Python ou en bash. Le code source est constitué d'environ 1 500 classes et 513 636 lignes de code (fichiers .cpp et .h). La modélisation d'un applicatif considéré est définie dans un Jeu De Données (.data) et toutes les exécutions (compilation du code, lancement du jeu de données,...) se font en ligne de commande.

TrioCFD est lui-même composé de plusieurs Baltiks et sous-Baltiks ayant tous un domaine de compétence spécifique. Ils sont à l'heure actuelle au nombre de 10 et 2 d'entre eux contiennent eux-même 2 sous-Baltiks. Le tableau [I.2.1](#) dresse la liste de ces Baltiks et sous-Baltiks, leur domaine de compétence ainsi que leur dépendance interne.

Baltik	Description	Dépendances
Critere_Entrainement_Gaz	Critère numérique basé sur la voriticité	Turbulence
	pour prédire l'apparition d'un vortex	Turbulence
Fluid_Structure_Interaction	Méthode Arbitrary Lagrangian-Eulerian	Turbulence
	pour les interactions fluide-structure	
	modèle vibratoire de poutre	
Multiphase		
↔ CMFD	Computational Multiphase Fluid Dynamics	SO
↔ Front_tracking_IJK	Intégration de TrioIJK dans TrioCFD	Turbulence & FTD
↔ Front_tracking_discontinu	Suivi d'interface avec la méthode de Front-Tracking	Turbulence
↔ Phase_field	Écoulements diphasiques incompressibles de	SO
	fluides non miscibles	
P1NCP0RT	Approximation P1/P0 non conforme avec les	
	éléments de Raviart-Thomas	SO
Rayonnement	Rayonnement thermique	
↔ Rayonnement_milieu_transparent	dans différents	Turbulence
↔ Rayonnement_semi_transp	milieux	Turbulence
Schema_Euler_Implicite_Stationnaire		Turbulence
Sensitivity_analysis	Quantification des changements dans la solution d'un système d'EDP (ici Navier-Stokes) dus aux variations des paramètres d'entrée	SO
Turbulence	Baltik principal de TrioCFD dont quasiment tous les autres Baltiks dépendent. Il contient l'ensemble des modèles de turbulence (Bas-Reynolds, k-epsilon, lois de parois,...) quelle que soit la discrétisation considérée	SO
validation	Baltik contenant les fiches de validation pour les tests à effets intégraux. Les fiches de validation des tests à effets séparés sont présents dans le répertoire /share du Baltik concerné	Turbulence
Zoom		Turbulence

TABLE I.2.1: Cartographie des BALTIKS et sous-BALTIKS de TrioCFD

Ces différents Baltiks permettent de couvrir les domaines de modélisation physique suivants :

- ⇒ Hydraulique avec ou sans turbulence
- ⇒ Thermohydraulique avec ou sans turbulence
- ⇒ Quasi-compressible
- ⇒ Écoulements diphasiques
 - Front-Tracking
 - Interface diffuse incompressible
 - Modèle Homogène Équilibré (HEM) présent dans le Baltik CMFD

⇒ Interactions fluide/structure par la méthode ALE

⇒ Chimie

Les outils seront décrits dans la troisième partie de ce document mais avant cela, intéressons nous aux différents termes spécifiques qui seront utilisés par la suite.

II. Définitions

LE projet TrioCFD fait intervenir différents acteurs et utilise diverses notions qui seront expliqués dans cette seconde partie.

II.1

Définition des termes utilisés

TULEAP : logiciel OpenSource permettant la gestion du cycle de vie de TrioCFD. Il dispose de plusieurs outils dont plusieurs sont utilisés dans le cadre de la gestion du code TrioCFD :

- ⇒ **BugTracker** : Système de "tracker" pour suivi des bugs, tâches, demandes de support, exigences, stories...
- ⇒ **Gestion de version** : Tuleap supporte le dépôt GIT de TrioCFD. C'est via Tuleap que les développeurs vont demander l'intégration de leur branche de travail dans la version de développement par des Pull Request. Tuleap permet également de gérer les droits d'accès à la base GIT du code ;
- ⇒ **MediaWiki** : Système de gestion de contenu permettant de générer des wikis pour les développeurs et utilisateurs du code. Il est collaboratif et peut être enrichi ou mis à jour par n'importe quel membre du projet TULEAP. Il fournit des informations sur la méthodologie de développement dans TrioCFD, des mémos, un état des lieux de la validation pour les versions livrées,...
- ⇒ **Documents** : Stockage des documents relatifs à TrioCFD à destination des développeurs/utilisateurs internes CEA et TMA (*e.g.* notes techniques CEA, supports des réunions de développement, spécifications techniques émises,...)

Atelier de Génie Logiciel : Ensemble de procédures en Python et/ou bash constituant l'atelier "maison" de la plateforme et permettant sa vérification quotidienne, son bon fonctionnement, sa livraison, la génération de la documentation, ...

GIT : outil de gestion de versions décentralisé sur lequel TrioCFD est stocké et l'évolution de son contenu d'arborescence géré.

Branche master : branche d'origine du dépôt GIT. C'est elle qui est la référence et qui permet de générer les versions de livraison. Celle-ci est mise à jour à chaque livraison de version, une fois que la version est considérée comme propre et stable. L'Atelier de Génie Logiciel part de cette branche du code pour générer l'archive pour la livraison qui sera distribuée aux utilisateurs et installée sur les différentes machines/clusters.

Branche triou/TMA : branche de développement de TrioCFD dans laquelle sont régulièrement intégrées les branches GIT des développeurs ou de la TMA. C'est sur cette branche que l'Atelier de Génie Logiciel lance quotidiennement les tests de vérification sur les différentes machines du parc.

Branche GIT : branche GIT d'un développeur ou d'un membre de la TMA issue de la branche de développement **triou/TMA** à un instant donné. La Branche GIT a une vie propre en gestion de configuration. Elle peut être synchronisée avec une autre branche par les développeurs et la TMA. Tout nouveau développement ou correctif est effectué dans une nouvelle branche dédiée à ce travail. Le nom de cette branche doit respecter un certain formalisme, à savoir TCFDXXXXXX_activité où XXXXXX correspond à l'Artifact ID (numéro d'identification) de la fiche de suivi correspondante ouverte dans le BugTracker et activité, à un descriptif sur quelques caractères du domaine concerné par la branche (*e.g.* turbulence, documentation,...). Une fois arrivée à maturité, celle-ci sera intégrée dans la version de développement de TrioCFD.

Tag GIT : répertoire GIT des versions "figées" du projet.

Pull Request : mécanisme qui permet au développeur de prévenir les membres de son équipe et notamment l'intégrateur que sa branche GIT est fonctionnelle, testée et donc prête à être intégrée dans la branche de développement de TrioCFD (branche **triou/TMA**). 4 informations sont nécessaires pour faire une Pull Request : le dépôt source, la branche source, le dépôt cible et la branche cible.

Jeu De Données : (JDD) il s'agit d'un ensemble de valeurs (ou données) permettant de définir les caractéristiques et les modèles qui seront utilisés lors de la simulation d'un applicatif. Le Jeu De Données est structuré en différentes parties : définition de la géométrie, du maillage, des problèmes traités (*e.g.* turbulence, convection, diffusion,...), des conditions aux limites, des variables à extraire pour le post-traitement,...

Demande d'intervention : (DI) Demande d'intervention d'un Initiateur ou d'un Utilisateur sur le code. Pour chaque Demande d'Intervention, une fiche de suivi est ouverte dans le BugTracker de la forge TULEAP de TrioCFD. Cet outil permet le suivi de la réalisation de son cycle de vie autant par le CEA que par la TMA. Il garantit la traçabilité des demandes et permet le calcul des indicateurs de la TMA. Ces Demandes d'Intervention peuvent être de différentes natures :

- ⇒ **Maintenance Corrective (MC) :** action permettant la correction d'anomalies dans les sources du code, dans les outils de maintenance, dans la documentation et dans les Jeux De Données de la base de test.
- ⇒ **Maintenance Evolutive (ME) :** action de modification des sources du code et des outils de maintenance associés qui ont pour conséquence :
 - L'ajout de nouvelles fonctionnalités ;
 - L'adaptation à un changement d'environnement logiciel et système ;
 - L'amélioration des performances informatiques ;
 - La réalisation de fiches de vérification automatisées.
- ⇒ **Assistance Aux Utilisateurs (AAU) :** action permettant de répondre au besoin d'un utilisateur sans que cela implique la réalisation d'une maintenance corrective ou évolutive. Cela couvre les actions d'installation ou d'aide à l'utilisation des codes du lot considéré mais exclut les actions de formation ou d'expertise physique.
- ⇒ **Action de PORTage (POR) :** service ayant pour objet le portage des versions des codes sur les systèmes d'exploitation (OS) supportés.
- ⇒ **Tests de Non-Régression (TNR) :** action permettant d'assurer le suivi quotidien de l'état d'une version d'un code suite à l'intégration de corrections d'anomalies ou de développements.
- ⇒ **FORmations (FOR) :** ensemble des actions permettant de préparer et réaliser une session de formation sur le code. Les formations sont planifiées et organisées par le CEA (spécification dans les lettres de cadrage semestrielles) qui en confie la préparation et l'exécution à la TMA.

⇒ **LIVraison de nouvelles versions (LIV)** : ensemble des actions permettant d'assurer la production de versions sur les différents systèmes d'exploitation (OS) et leur documentation en vue de la livraison des versions du code.

Suivant la nature et/ou la complexité de la Demande d'Intervention, celle-ci peut être traitée soit par l'équipe TMA soit par l'équipe CEA.

Régression : Dégradation du comportement du code après intégration d'une modification ou d'une correction.

TNR : Test de Non Régression, ensemble de calculs réalisés après intégration d'une modification du code (dans le cadre du processus de développement ou d'action de maintenance), pour s'assurer de l'intégrité de la version du code.

Robustesse : notion qui rend compte du fait que l'utilisateur, lançant un calcul aura la certitude que celui-ci ne soit pas interrompu avant l'atteinte de la fin du calcul, par le code lui-même sur critère de mauvaise convergence.

Portabilité : notion qui rend compte du fait qu'un même calcul, réalisé dans différents environnements (compilateur, OS, machine PC ou serveur de calcul), produira un résultat le moins sensible possible à cet environnement de calcul. Pour statuer sur la bonne portabilité d'un code de calcul, on accepte généralement une tolérance de 2% d'écart entre les différentes configurations testées.

Mode de compilation : afin de s'assurer du bon fonctionnement du code après chaque intégration, celui-ci est compilé chaque nuit par l'atelier logiciel avec différents modes de compilation et différentes options. 3 modes de compilation sont testés régulièrement :

- le mode optimisé : on autorise, dans ce cas, au compilateur à effectuer des opérations d'optimisation afin de générer le code pour une vitesse maximale d'exécution ou une taille minimale du code compilés ; le code compilé est alors plus complexe, une compilation plus longue mais une taille de code compilé réduite. Par exemple, en mode optimisé, seules les variables globales sont initialisées à 0 et non pas l'ensemble des variables. Ainsi, quand les variables sont lues dans des contextes particuliers, comme à l'intérieur d'une boucle, ces absences d'initialisation ne sont pas toujours détectées par le compilateur.
- le mode debug : ce mode de compilation est principalement dédié au débogage lors du développement. Dans ce cas-là, toutes les optimisations sont désactivées (informations symboliques conservées, ajout de points d'arrêt,...). Ce mode de compilation permet de mieux comprendre le fonctionnement d'un programme, d'exécuter le code pas à pas (grâce un debugger type gdb) et de corriger facilement les erreurs. Le temps de compilation est alors plus court, l'utilisation de la mémoire est moins importante mais la taille du code compilé sera plus importante ainsi que son temps d'exécution. Ce mode d'exécution est souvent couplé à des outils de debug/analyse de performance, type valgrind pour détecter d'éventuelles fuites mémoire parfois indétectables par le compilateur.
- le mode semi-optimisé : il s'agit d'un mode de compilation intermédiaire entre le mode optimisé et le mode debug.

II.2

Définition des acteurs

TMA : Tierce Maintenance Applicative est la maintenance appliquée du logiciel. Elle est assurée par un prestataire externe dans les domaines de l'informatique et de la thermo-hydraulique. Un cahier des charges est émis par le CEA auquel plusieurs candidats vont répondre. A l'issue de cette consultation, un des candidats sera retenu et deviendra le titulaire du contrat de maintenance. Le contrat actuel de TMA est établi pour deux ans fermes avec la possibilité de lever des options pour trois années supplémentaires. Sur **TrioCFD**, l'équipe technique de TMA est composée de deux intervenants principaux :

- ⇒ Le **Responsable Maintenance Logiciel (RML)** est généralement le membre le plus expérimenté de l'équipe de TMA. Il est responsable de différentes tâches comme :
- la centralisation des Demandes d'Intervention (réception des actions de maintenance, vérification de leur cohérence et leur recevabilité) ;
 - la répartition du traitement des Demandes d'Intervention au sein de l'équipe TMA ;
 - le bon fonctionnement de l'Atelier de Génie Logiciel ;
 - le suivi de la base de vérification quotidienne ;
 - la tenue de la bonne résolution des Demandes d'Intervention (qualité, temps de traitement,...) ;
 - le reporting des statistiques sur les travaux de la TMA ;
 - l'archivage rigoureux de l'ensemble des documents associés aux Demandes d'Intervention en cours ou closes ;
 - le chiffrage et l'émission d'une proposition de travail pour les Demandes d'Intervention de type **Maintenance Evolutive** ;
 - la résolution des Demandes d'Intervention (généralement celles nécessitant la plus grande compétence métier ;
 - ...
- ⇒ L'**Intervenant Maintenance Logiciel (IML)** est, quant à lui, chargé de la résolution des Demandes d'Intervention qui lui sont attribuées par le RML et de l'accomplissement des tâches que le RML peut lui avoir déléguées ;

Le **Responsable de Produit Logiciel (RPL)** (agent CEA) s'assure du bon déroulé du contrat de TMA et remonte au chargé d'affaire CEA, les performances de l'équipe TMA qui seront étudiées lors du Comité de Pilotage (COPIL) qui se déroule tous les 2 mois. Il statue sur les priorisations entre les différents codes couverts par l'équipe de TMA.

Le **Responsable de Code** (agent CEA) est en charge du bon fonctionnement du code et des outils associés, de sa gestion ainsi que de la cohérence techniques des différentes actions menées sur le code. Ainsi, sont de son ressort :

- ⇒ la bonne conduite de l'intégration de nouveaux développements ou de correctifs dans la version considérée,
- ⇒ la réalisation périodique des tests de non-régression lorsque la branche de développement a fait l'objet de modifications,

- ⇒ le dépouillement des résultats associés, au moyen d'outils du projet et prononce la validité (ou non) de cette version autant d'un point de vue physique que numérique,
- ⇒ la mise à jour de la liste des tests de non-régression de la branche considérée (ajout/suppression),
- ⇒ la fourniture d'un cadre de travail optimum aux développeurs et utilisateurs du code en terme d'outils, de préconisations, de documentation,...
- ⇒ le lien entre l'équipe CEA et l'équipe TMA travaillant sur les Demandes d'Intervention émanant de l'équipe de développement
- ⇒ les vérifications du ressort du CEA lors de la livraison des versions
- ⇒ la communication autour du code (réunions de développement, séminaires, site web,...)

Le **Relecteur** s'occupe de la relecture des branches des développeurs avant que celles-ci ne soient intégrées dans la branche de développement du code. Suite à cette relecture, des corrections sur la branche considérée peuvent être nécessaires afin que le code réponde aux recommandations de codage et aux exigences. Ce rôle est, la plupart du temps, endossé par le Responsable de Code.

L'**Intégrateur**, après approbation d'une branche de développement, sera en charge de fusionner cette branche dans la branche de développement de TrioCFD. Ce rôle est également majoritairement endossé par le Responsable de Code.

L'**Initiateur** est celui qui détecte une anomalie sur les versions en développement. La détection de cette anomalie entrainera l'ouverture d'une Demande d'Intervention et donc d'une fiche dans le BugTracker.

L'utilisateur : L'utilisateur formalise une Demande de maintenance et est destinataire de sa correction.

III. Outils

TRIOCFD dispose de deux Trackers sous l'outil TULEAP [4]. Ils ont tous deux des utilités différentes l'un étant dédié à la gestion du code (Demandes d'Intervention, Développements, Bugs,...) tandis que l'autre constitue un outil de suivi et d'archivage des études effectuées sous TrioCFD. Par conséquent, leur constitution et leur utilisation sont totalement différentes et vont être détaillées dans les deux premiers chapitres de cette partie. Nous nous intéresserons ensuite à l'outil de gestion du code (GIT) puis, les différentes messageries du projet pour finir par une cartographie des différents outils d'interfaçage.

III.1

Tracker informatique - TULEAP

Le code TrioCFD dispose d'un projet propre sous TULEAP accessible à l'adresse <https://codev-tuleap.intra.cea.fr/projects/triocfd/>. Il a pour but de répertorier l'ensemble des actions de correction, développement, assistance et livraison menées sur le code.

Constitution du projet TrioCFD

Dans ce projet TULEAP, différents outils sont disponibles via le bandeau de gauche :

⇒ **TrioCFD BugTracker** : outil de trackage des différents développements, correctifs, demandes d'assistance,... qui concernent le code TrioCFD. Chaque fiche dispose d'un seul et unique numéro d'identification qui est utilisé pour la dénomination des branches du gestionnaire de contrôle des versions, GIT. Cet outil, assez conséquent est décrit plus en détail par la suite.

⇒ **Documents** : regroupe les différents documents à destination des utilisateurs CEA de TrioCFD (archivage des présentations des séminaires, des notes techniques CEA, des spécifications techniques...). La structure est illustrée sur la figure III.1.1.

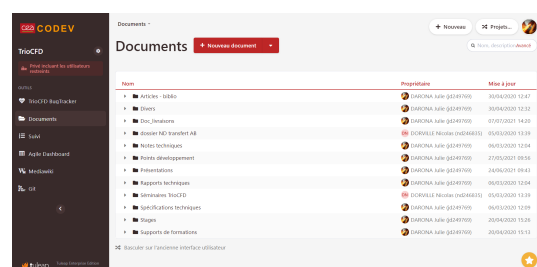


FIGURE III.1.1: Structure de la base documentaire sur TULEAP

⇒ **Agile Dashboard** : outil de planification des intégrations en vue des livraisons du code. Cet outil a commencé à être mis en place à partir de la version v1.8.1 (juin 2020). Son utilisation n'est pas encore optimale mais monte progressivement en puissance.

⇒ **MediaWiki** : outil d'aide à destination des développeurs pour l'installation de leur environnement de travail

⇒ **Git** : le système de contrôle des versions sous base GIT est hébergé sous TULEAP et est accessible ici. On y trouve l'adresse du dépôt nécessaire pour faire le clonage (voir figure III.1.2). Aucune action GIT ne doit être lancée depuis TULEAP car source de nombreuses erreurs. Seules les Pull Request sont effectuées depuis TULEAP en choisissant, dans les menus déroulants, d'une part la branche voulant être intégrée et d'autre part, la branche recevant l'intégration (voir figure III.1.3).

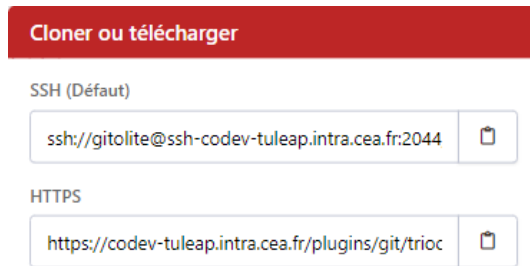


FIGURE III.1.2: Récupération de l'adresse du dépôt GIT via TULEAP

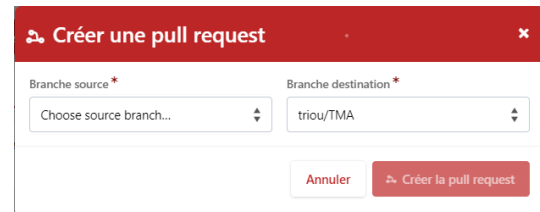


FIGURE III.1.3: Création d'une Pull Request depuis Tuleap

Nous allons maintenant détailler la partie BugTracker du projet TULEAP de TrioCFD.

Fonctionnement général du BugTracker informatique

Le BugTracker informatique est utilisé autant par l'équipe de TMA que par les agents CEA. A chaque action concernant le code, une fiche spécifique est créée (fiche de Demande d'Intervention). Les fiches répertoriées sur le BugTracker peuvent être traitées soit par l'équipe TMA soit par l'équipe CEA suivant la nature de la fiche. La répartition du traitement des fiches est gérée par le Responsable de Code via le champs **Way of treatment** de chaque fiche.

Pour les demandes reçues sur les adresses électroniques du projet (voir Chapitre III.4), les fiches sont exclusivement ouvertes par l'équipe TMA. En revanche, pour les fiches attribuées à l'équipe CEA, celles-ci sont ouvertes soit directement par un membre de l'équipe soit par le responsable de code.

Afin de garantir une bonne traçabilité, chaque action de modification, correction ou évolution faite dans le code nécessite l'ouverture au préalable, d'une fiche de Demande d'Intervention. A chaque fiche d'intervention, correspond une branche de développement spécifique dans le gestionnaire de versions dont le nom inclut l'identifiant de la fiche. Ainsi, à tout moment, il est possible de connaître la raison du mouvement des sources du code mais également de connaître l'influence de ce mouvement sur la validation. Nous reviendrons, par la suite, plus en détails sur ces aspects de traçabilité.

Constitution d'une fiche de demande d'intervention

Une fiche de demande d'intervention est constituée de multiples champs. Tous ne sont pas systématiquement définis mais les champs vivement recommandés sont détaillés ci-après :

- ⇒ **liste_diffusion** (*type* : liste ouverte avec lien vers les membres du projet) : permet de retrouver, dans l'annuaire du BugTracker, les personnes potentiellement intéressées/concernées par la fiche. Toute personne identifiée dans ce champ recevra un e-mail automatique du BugTracker la renseignant sur les modifications apportées dans la fiche.
- ⇒ **Summary** (*type* : chaîne de caractère - **champ obligatoire**) : correspond au titre de la Demande d'Intervention. Celui-ci doit être court tout en identifiant clairement le problème constaté.
- ⇒ **Original Submission** (*type* : Texte) : décrit en détails le problème rencontré ou le développement prévu. Comme, il s'agit d'un champ sans restriction, il est recommandé de donner le maximum d'informations sur le contexte de la fiche et les attendus.

- ⇒ **Priority** (*type* : liste de choix) : définit la priorité de la fiche considérée par rapport aux autres ouvertes. Ce champ est principalement à destination de l'équipe TMA afin qu'elle soit renseignée sur la priorité de traitement des fiches en fonction des contraintes projet. L'affectation à une priorité induit un délai de résolution : 1-High -> 10 jours ouvrés ; 2-Medium -> 22 jours ouvrés ; 3-Low -> avant la prochaine livraison de version. Le compteur débute à la date renseignée dans le champs *Date hiérarchisation*.
- ⇒ **Category** (*type* : liste de choix) : correspond à une des catégories de Demandes d'Intervention listées au Chapitre II.1 et identifiées dans le contrat de marché de la TMA.
- ⇒ **Sub project associated** (*type* : liste de choix) : permet d'identifier le Baltik impacté par la DI.
- ⇒ **Authorization to work on this** (*type* : case à cocher) : champ destiné à la TMA pour lui donner l'autorisation de débiter les travaux sur la DI.
- ⇒ **Date hiérarchisation** (*type* : date) : champ à mettre à jour avec le champ précédent (Authorization to work on this) et à renseigner avec la date du jour du lancement des travaux sur la DI. Cette date permettra de vérifier que le délai de réalisation de la DI est bien tenu.
- ⇒ **Status** (*type* : liste de choix) : correspond à l'état de vie de la fiche. Cette notion sera plus approfondie dans la section suivante (voir section III.1.5).
- ⇒ **Assigned to** (*type* : liste à choix multiples avec lien vers les membres du projet) : correspond à(aux) membre(s) du projet travaillant sur le traitement de la DI.
- ⇒ **Way of treatment** (*type* : liste de choix) : 2 choix sont possibles ; soit la demande est traitée par le CEA (*ResolutionByCEA*) soit par la TMA (*ResolutionByTMA*).
- ⇒ **Work load** (*type* : flottant) : champ renseigné uniquement lorsque la DI est traitée par l'équipe TMA afin de s'assurer que la charge de traitement n'a pas été trop conséquente et a bien respecté les indicateurs contractuels.

Ainsi, l'aspect général de la fiche au moment de sa création est visualisable ci-contre (voir Figure III.1.4)

Tous les champs ne sont pas obligatoires à la création de la fiche mais peuvent le devenir en fonction de son état de traitement. De même, tous les champs sont modifiables à tout moment. Ainsi, on veillera à mettre à jour le titre de la fiche si, lors de son traitement, on constate que le problème avait été initialement mal identifié.

La gestion du BugTracker Tuleap est du ressort du Responsable de Code. En fonction de l'évolution des besoins, des adaptations de ces champs pourront ponctuellement être faites.

FIGURE III.1.4: Champs à renseigner pour la création d'une Demande d'Intervention TrioCFD

Une fois la fiche de DI créée, celle-ci est répertoriée dans le BugTracker avec un identifiant unique à 6 chiffres qui servira à nommer la branche GIT associée. On observera sur la figure III.1.5, cet identifiant dans la 1^{ère} colonne du tableau.

Artifact ID	Local ID	Summary	Submitted on	Last Modified On	Priority	Status	Way of treatment	Category	Submitted by	Assigned to	Authorization to work on this
199144	155	Development of a new component	21/10/2021 20:12	25/10/2021 17:06	2 - Medium	Merged	ResolutionByCEA	ME	Matthieu (CEA)	Matthieu (CEA)	
198632	152	Improvement of the user interface	12/10/2021 12:20	25/10/2021 17:42	3 - Low	Merged	ResolutionByTMA	MC	Matthieu (CEA)	Matthieu (CEA)	Yes
198117	151	Integration of a new module	30/09/2021 09:37	12/10/2021 09:09	2 - Medium	InProgress	ResolutionByTMA	AAU	Matthieu (CEA)	Matthieu (CEA)	Yes
196898	149	Integration of a new module	24/08/2021 15:43	24/08/2021 15:44	2 - Medium	InProgress	ResolutionByCEA	ME	Matthieu (CEA)	Matthieu (CEA)	
196896	148	Integration of a new module	24/08/2021 15:08	06/09/2021 14:09	2 - Medium	InProgress	ResolutionByTMA	AAU	Matthieu (CEA)	Matthieu (CEA)	Yes
196368	147	Development of a new component	30/07/2021 14:41	30/07/2021 14:41	2 - Medium	New	ResolutionByCEA	ME	Matthieu (CEA)	Matthieu (CEA)	
196284	146	Integration of a new module	27/07/2021 14:59	21/09/2021 11:06	1 - High	ReceivedByTMA	ResolutionByTMA	MC	Matthieu (CEA)	Matthieu (CEA)	Yes
196117	141	Integration of a new module	20/07/2021 13:09	27/08/2021 10:39	2 - Medium	ReceivedByTMA	ResolutionByTMA	AAU	Matthieu (CEA)	Matthieu (CEA)	Yes
195869	140	Integration of a new module	12/07/2021 09:57	12/07/2021 09:59	2 - Medium	New	ResolutionByCEA	ME	Matthieu (CEA)	Matthieu (CEA)	

FIGURE III.1.5: Vue générale du BugTracker TrioCFD et de quelques fiches de Demandes D'Intervention en cours

La vue générale de la figure III.1.5 permet d'avoir rapidement la visualisation des fiches ouvertes, leur priorité, leur état d'avancement et les personnes en charge de leur résolution. Cette vue peut être ajustée en fonction des besoins de suivi et ses différentes configurations, conservées.

États de traitement d'une fiche de Demande d'Intervention

Une fiche de DI du BugTracker TrioCFD suit un processus de traitement clairement défini et a, suivant son état, un statut particulier défini dans le champ **Status**. Ce cycle de vie est illustré par la figure III.1.6. Il est composé de 6 états (dans les boîtes rectangulaires) qui sont :

- ⇒ **NEW** : pour une DI saisie directement par l'utilisateur (agent CEA), une fois les champs précisés au paragraphe précédent, remplis et la création de la fiche validée, celle-ci passe automatique en statut NEW.
- ⇒ **RECEIVED BY TMA** : (pour une DI créée par l'équipe TMA) une fois les champs précisés au paragraphe précédent remplis et la création de la fiche validée, la TMA la définit en statut RECEIVED BY TMA. Il en est de même pour les DI en statut NEW lorsque son traitement est du ressort de la TMA. Cet état signifie que la TMA a menée une première analyse de la demande et a commencé à identifier l'origine du problème. Ces premiers résultats d'analyse seront discutés lors de la réunion hebdomadaire entre le RPL et le RML. Il sera alors décidé de la pertinence de cette demande, de sa priorité de traitement (champ **Priority**) et du lancement ou non des travaux sur celle-ci (champ **Authorization to work on this**).
- ⇒ **IN PROGRESS** : lorsque le traitement d'une fiche débute, il appartient à la personne en charge de son traitement de la passer en statut IN PROGRESS. Ce changement de statut nécessite de renseigner le champ **Assigned to** afin qu'il soit pris en compte. Cet état permet de savoir que les travaux ont débuté sur le sujet. C'est dans ce statut que la DI passera la plus grande partie de sa vie. Lorsque le travail sera achevé, si la DI nécessite une modification du code source, une Pull Request (voir figure III.1.3) sera effectuée par la personne ayant traitée la DI.
- ⇒ **MERGED** : une fois la branche correspondante à la DI intégrée dans la branche de développement du code (branche **trio/TMA**) par l'Intégrateur, celui-ci la passe en statut MERGED. L'équipe de TMA

aura ainsi connaissance des intégrations d'un jour sur l'autre et prêter une attention toute particulière au retour hebdomadaire de l'atelier de Vérification. En cas d'impacts non attendus sur la base de vérification, la DI est repassée en statut IN PROGRESS afin d'investiguer sur ces écarts imprévus et les justifier ou apporter des correctifs pour les résorber. Ce statut concerne uniquement les DI de type MC (Maintenance Corrective) ou de type ME (Maintenance Évolutive).

- ⇒ **TREATED/RESOLVED** : cet état signifie que l'ensemble des travaux nécessaires pour mener à bien le traitement de la fiche ont été faits. Dans le cas d'une ME ou d'une MC, ce statut est atteint lorsque la branche de travail a été intégrée avec succès dans la branche en développement du code et que les éventuels impacts ont été validés. Pour une fiche de type AAU, l'initiateur de la demande a retourné un avis positif sur la solution proposée pour résoudre son problème.
- ⇒ **CLOSED** : le travail réalisé dans le cadre de cette Demande d'Intervention est validé par le RPL (demandes en traitement TMA) ou par le Responsable de Code (demandes en traitement CEA). Une fois cet état atteint, plus aucun travail ne doit être mené sur la demande. Si d'aventure, d'avantage de travail était identifié à posteriori, une nouvelle fiche devra alors être créée.

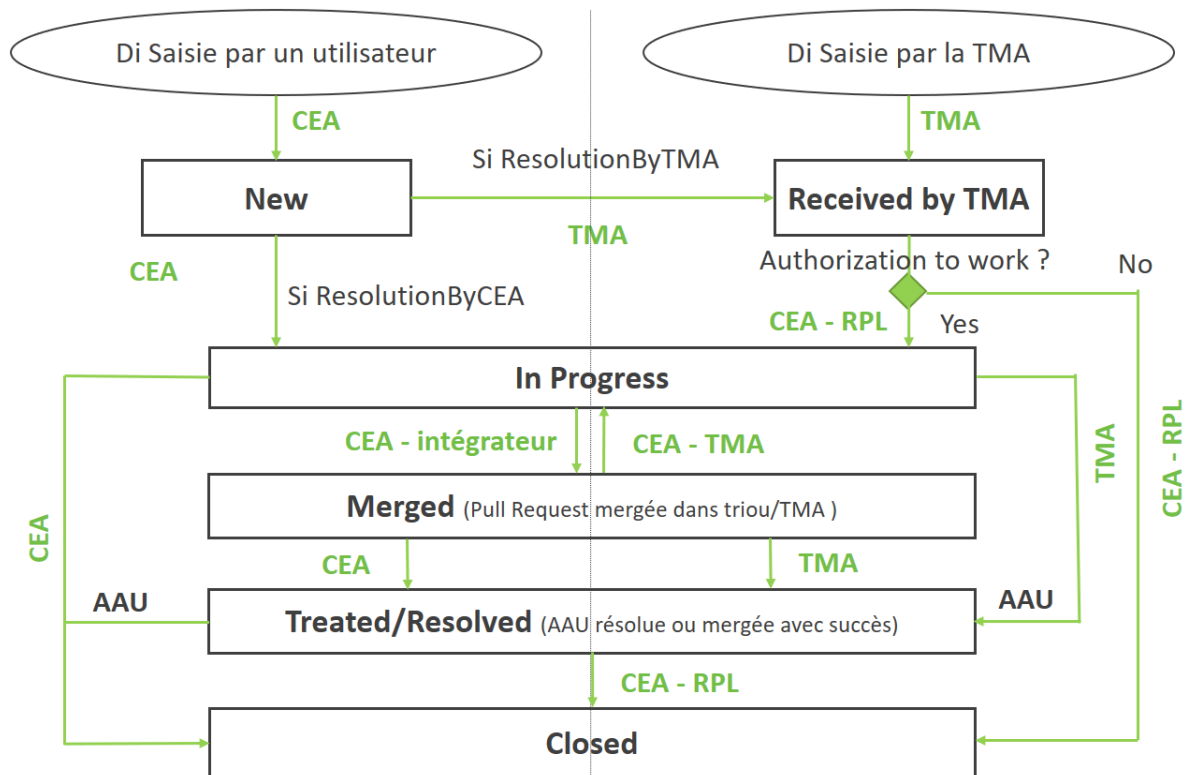


FIGURE III.1.6: Workflow du BugTracker TrioCFD

L'ensemble de cette démarche est également expliquée dans le MediaWiki du BugTracker TrioCFD.

Le respect de ces procédures garantit une bonne traçabilité de l'ensemble des actions relatives au code mais également de toute modification apportée dans les sources. Le Responsable de Code a donc pour mission des les expliquer et de veiller à leur bonne application. En cas de non respect, il a à sa charge d'apporter les modifications nécessaires afin de rétablir leur bon déroulement (saisie des fiches de DI, rappel à l'ordre, blocage provisoire des droits d'intégration,...).

III.2

Suivi et archivage des études

Pour le suivi et l'archivage des études menées sous TrioCFD, le code dispose d'un Tracker propre sous Tuleap dans le projet Etudes-STMF à l'adresse : <https://codev-tuleap.intra.cea.fr/projects/etudes-stmf/>. D'autres codes du service sont également hébergés sous ce projet Tuleap mais chacun d'eux dispose de son propre Tracker. Si quelques adaptations sont nécessaires d'un code à l'autre pour répondre au mieux à leurs besoins, ils restent néanmoins très proches afin d'en faciliter la consultation. Ce projet a été mis en place début 2019 et est rentré en production mi 2019. Le Tracker de chacun des codes permet d'assurer un suivi de chacune des études menées sur le code concerné. Il n'est en aucun cas un lieu d'archivage des études. Pour l'archivage, un espace dédié a été créé sur le nouveau système de stockage Linux du service nommé TITANIA. Le lien avec cet espace de stockage ainsi que sa structure seront détaillés dans la dernière section de ce chapitre.

Objectifs du suivi et de l'archivage des études

Avant la création de ce système, aucun outil n'était proposé pour conserver une mémoire de l'ensemble des études menées sur un code. Chacun conservait, en local sur son poste de travail, les études qu'il avait réalisées. Or cette méthodologie de travail présente de forts inconvénients et risques parmi lesquels, on peut citer :

- Pertes scientifiques très conséquentes en cas de départ de la personne ayant réalisé des études sur un code ou en cas de changement de localisation des activités
- Pas de "mémoire" des codes sur les études pour lesquelles ils ont été utilisés
- Anciennes études inexploitable au fur à mesure que le temps passe
- Incapacité de reproductibilité des dossiers de sureté
- Limitations de la mutualisation des connaissances (échanges sur les pistes suivies ayant abouti ou non à la résolution des problèmes, avis des développeurs pouvant aider la partie étude,...)
- Benchmark entre les différents codes peu nombreux
- Couplage entre les différents codes peu nombreux
- Doublement d'études en cas de manque de dialogue entre les projets ou les laboratoires
- Pas de réelle vision d'ensemble de la multitude des actions menées dans le service

Les objectifs de ce suivi sont donc de limiter autant que possible les problèmes identifiés précédemment.

Constitution d'une fiche de Suivi d'Etude

Comme les fiches de Demande d'Intervention, les fiches de Suivi d'Etude contiennent plusieurs champs. Ils sont moins nombreux que ceux d'une Demande d'Intervention informatique car ce Tracker ne s'adresse pas au même public et son utilisation se doit d'être plus simple. Une fiche de Suivi d'Etude est constituée de plusieurs rubriques, chacune étant constituée de différents champs. L'ensemble de cette structure est détaillée

ci-dessous.

Général

- ⇒ **Titre de l'étude** (*type* : chaîne de caractères - **champ obligatoire**) : correspond au titre de l'étude menée. Celui-ci doit être relativement court tout en relatant sans ambiguïté le ou les domaine(s) traité(s) dans l'étude
- ⇒ **Description** (*type* : texte - **champ obligatoire**) : décrit, en détails, l'étude qui va être menée avec notamment ses objectifs, les phénomènes observés, les moyens utilisés pour valider l'étude,...
- ⇒ **Mots-clés** (*type* : chaîne de caractères) : à renseigner de quelques mots clés bien choisis afin de retrouver plus rapidement les études traitant d'un sujet spécifique

The 'Général' tab contains the following fields and controls:

- Titre de l'étude (?) ***: A text input field.
- Description (?) ***: A large text area with a 'Format: Markdown' dropdown menu and 'Aperçu' and 'Aide' buttons.
- Mots-clés**: A text input field.

FIGURE III.2.1: Constitution d'une fiche de Suivi d'Etude - partie générale

Contexte

- ⇒ **Projet** (*type* : liste de choix) : à renseigner si l'étude est réalisée dans le cadre du financement par un projet
- ⇒ **Lot/application** (*type* : liste de choix) : spécifie le lot du projet si le champ précédent a été renseigné
- ⇒ **Accords de confidentialité** (*type* : liste de caractères) : renseigne si l'étude est soumise à des accords de confidentialité et dans quel cadre d'accord
- ⇒ **Partenariat** (*type* : liste de caractères) : permet de préciser si l'étude a été réalisée en partenariat avec une autre entité (laboratoire, université, partenaire industriel,...)
- ⇒ **Jalon** (*type* : bouton radio - **champ obligatoire**) : définit si l'étude constitue un jalon pour un projet
- ⇒ **Livable** (*type* : liste de choix - **champ obligatoire**) : définit si un livrable doit être émis suite à l'étude
- ⇒ **DR** (*type* : liste de choix - **champ obligatoire**) : renseigne si l'étude est soumise à une règle restreinte de diffusion

The 'Contexte' tab contains the following fields and controls:

- Projet**: A dropdown menu with 'Aucun' selected.
- Lot/application**: A dropdown menu.
- Accords de confidentialité (?)**: A text input field.
- Partenariat (?)**: A text input field.
- Jalon ? ***: Radio buttons for 'Oui' (red) and 'Non' (green).
- Livable ? ***: Radio buttons for 'Oui' (red) and 'Non' (green).
- DR ? ***: Radio buttons for 'oui' (red) and 'non' (green).

FIGURE III.2.2: Constitution d'une fiche de Suivi d'Etude - partie contexte

Informations sur l'étude

- ⇒ **Autre personne potentiellement intéressée** (*type* : liste ouverte avec lien vers les membres du projet) : permet de retrouver dans l'annuaire du Tracker, les personnes potentiellement intéressées/concernées par la fiche. Toute personne stipulée dans ce champ recevra un e-mail automatique

du BugTracker la renseignant sur les modifications apportées dans la fiche.

⇒ **Catégorie** (*type* : case à cocher) : permet d'identifier si il s'agit d'une nouvelle étude, de la reprise d'une étude,...

⇒ **Versión** (*type* : liste de choix) : renseigne sur la version du code utilisée pour mener l'étude. Les différentes versions livrées depuis la v1.7.9 sont disponibles dans le menu déroulant ainsi que la version de développement. Si l'étude était menée sur la version de développement, il est indispensable de renseigner le champ suivant.

⇒ **Préciser SHAI-ID si version développement ou autre version** (*type* : chaîne de caractères) : correspond au "code barre" ou SHA1 ID sous le gestionnaire de configuration GIT, de la version de développement utilisée pour mener l'étude.

FIGURE III.2.3: Constitution d'une fiche de Suivi d'Etude - partie informations sur l'étude

⇒ **Système d'exploitation** (*type* : liste de choix) : renseigne sous quel système d'exploitation l'étude a été menée. En effet, on peut parfois observer de petits écarts de portabilité entre les machines. Si l'étude archivée était amenée à être reprise ou en cas d'inspection des Autorités de Sureté, cette information est capitale.

⇒ **Autre(s) outil(s) utilisé(s) lors de l'étude** (*type* : chaîne de caractères) : si d'autres outils ont été utilisés dans le cadre de l'étude (*eg* SHAPER pour la construction de la CAO, SALOME pour l'établissement du maillage), le ou les outil(s) ainsi que leur version sont à renseigner ici.

Avancement

⇒ **Etat d'avancement** (*type* : liste de choix - **champs obligatoire**) : correspond à l'état de vie de la fiche. Cette notion sera plus approfondie dans la section suivante.

⇒ **Traité par** (*type* : liste à choix multiple avec lien vers les membres du projet) : correspond à(aux) membre(s) du projet travaillant sur le traitement de la DI.

FIGURE III.2.4: Constitution d'une fiche de Suivi d'Etude - partie avancement de l'étude

Finalisation de l'étude

Cette partie est à renseigner lorsque l'étude est terminée.

⇒ **Données finales sauvegardées sur Titania** (*type* : cases à cocher) : correspond à la liste des données qui auront été sauvegardées sur Titania. Ainsi, à la consultation d'une fiche, l'utilisateur saura immédiatement quelles données pourront être accessibles rapidement puisque sauvegardées sur le système d'archivage des études.

⇒ **Autres données sauvegardées sur Titania** (*type* : chaîne de caractères) : si d'autres données ont été sauvegardées sur Titania mais ne correspondent pas aux catégories proposées dans le champ précédent, elles peuvent être renseignées manuellement dans celui-ci.

- ⇒ **Référence note émise** (*type* : chaîne de caractères) : si dans le cadre de cette étude, une note a été émise, il est possible d'en renseigner son numéro d'émission ici afin de la retrouver plus rapidement dans la base documentaire de notes du département.
- ⇒ **Temps passé (jours)** (*type* : chaîne de caractères) : peut être rempli pour donner un retour d'expérience sur le temps nécessaire pour traiter chaque catégorie d'étude et faciliter les chiffrages postérieurs.

Pièces jointes

Des données légères peuvent être sauvegardées ici mais il ne s'agit, en aucun cas du lieu d'archivage de l'étude. Tuleap n'est pas conçu pour supporter de gros volumes de données et au delà de 50 Go de données stockées, son temps de réponse devient très important, nuisant fortement à son utilisation. Ainsi, seules des données de petites tailles (*eg* échanges de mail, supports de présentation,...) peuvent être sauvegardées ici. Ces données correspondent plutôt à des données nécessaires lors de l'étude et devront ensuite être sauvegardées sur le système d'archivage.

D'autre part, en cas d'une étude contenant des données sensibles (type Diffusion Restreinte ou Diffusion Restreinte - Spéciale France), ces données ne devront pas figurer "en clair" dans la fiche de Suivi de l'Etude. Elles devront avoir été préalablement placées dans le conteneur du projet hébergeant l'étude. C'est ce conteneur qui sera ensuite sauvegardé en pièce jointe dans la fiche.

Permissions

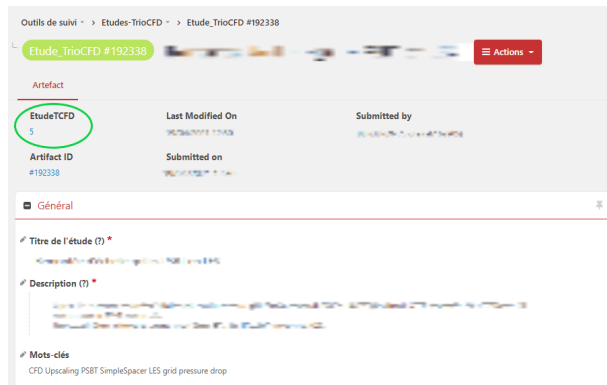
Le dernier champ à remplir avant la finalisation de la fiche de Suivi d'Etude concerne les droits d'accès à celle-ci. Ce champ est **obligatoire**. Si il s'agit d'une étude contenant des données sensibles (type Diffusion Restreinte ou Diffusion Restreinte - Spéciale France), il faudra veiller à sélectionner uniquement le groupe DRSF-STMF. Seul ce groupe-là pourra alors voir les études de ce type. Pour les autres études n'ayant aucune donnée sensible, le groupe Permanents-STMF sera à sélectionner ainsi que le groupe temporaires-STMF si on souhaite mettre la fiche à disposition des personnes n'étant pas en contrat permanents (stagiaires, doctorants, post-doc, CDD,...).

Lorsque tous les champs ont bien été remplis, veillez à bien appuyer sur le bouton **Valider** afin de générer la création de la fiche. L'ensemble des champs peut être modifié après la création de la fiche.

FIGURE III.2.5: Constitution d'une fiche de Suivi d'Etude - partie finalisation de l'étude

FIGURE III.2.6: Constitution d'une fiche de Suivi d'Etude - partie pièces jointes

FIGURE III.2.7: Constitution d'une fiche de Suivi d'Etude - partie permissions



Une fois la fiche créée, elle dispose alors d'un identifiant unique (voir encadré vert sur figure ci-contre). Cet identifiant servira à nommer le répertoire de sauvegarde de l'étude afin de le retrouver immédiatement sur le système d'archivage sous Titania.

FIGURE III.2.8: Constitution d'une fiche de Suivi d'Etude - numéro d'identification de la fiche

États de traitement d'une fiche de suivi d'étude

Contrairement au BugTracker informatique, le gestionnaire de suivi des études comporte un Workflow beaucoup plus simple avec seulement 4 états pour une fiche de Suivi d'Etude. Ces 4 états sont les suivants :

- ⇒ **NEW** : à la création de la fiche de Suivi d'Etude, celle-ci est automatiquement en statut NEW. L'étude a été identifiée mais le travail dessus n'a pas encore été amorcé.
- ⇒ **IN PROGRESS** : dès que le travail est amorcé, la personne en charge de l'étude passe la fiche de suivi correspondante à l'état IN PROGRESS. Pour le passage dans ce statut, il est nécessaire de renseigner le champ **Traité par**. En effet, à ce stade, il y a nécessairement une personne qui a été identifiée pour mener à bien cette étude. La fiche de Suivi de l'Etude va passer la majorité de sa vie dans cet état.
- ⇒ **SAVED** : une fois l'étude, à proprement parler, terminée, les données nécessaires à sa reproduction ou reprise doivent être archivées sur le système d'archivage sous Titania. Une fois que cet archivage est fait, la fiche passe alors en statut SAVED. Pour ce passage, 2 champs de la fiche doivent avoir été au préalable remplis à savoir **Données sauvegardées sur Titania** et **Version**. Ce dernier champ est, en effet, absolument nécessaire pour que les données archivées puissent être ré-exploitées. Un jeu de données peut différer d'une version à l'autre du code mais également être fonctionnel sur des versions antérieures et, nécessiter une mise à jour pour qu'il le soit sur une nouvelle version.
- ⇒ **CLOSED** : une fois l'étude menée et les documents associés archivés, la fiche de suivi peut alors être clôturée.

Il s'agit là du déroulé le plus standard d'une fiche d'étude. Il est toutefois possible qu'il y ait plusieurs itérations avant qu'une fiche ne soit clôturée. La figure III.2.9 représente l'ensemble du Workflow et récapitule les champs nécessaires pour le passage d'un statut à l'autre.

	New	In Progress	<u>Saved</u>	<u>Closed</u>
(Nouvelle fiche)	⬆	⬆ Traité par		
New		⬆ Traité par		
In Progress			⬆ Données finales sauvegardées sur <u>Titania</u> + Version	
<u>Saved</u>		⬆		⬆
<u>Closed</u>		⬆		

FIGURE III.2.9: Worflow du gestionnaire de suivi des études de TrioCFD

Archivage des études

Avant de passer dans l'état **SAVED** décrit au paragraphe précédent, toutes les données nécessaires à la reprise de l'étude doivent avoir été archivées sur Titania. Cette méthodologie d'archivage va être décrite dans cette section.

Le système d'archivage Linux est présent sur le nouveau système Linux du DM2S nommé Titania. Il est accessible autant depuis Linux que depuis Windows. Celui-ci s'appelle *etudes-stmf* (même nom que le projet sur Tuleap, moyennant la casse). L'accès à l'espace de stockage via LINUX se fait à partir d'un poste connecté à Titania. L'espace est ensuite accessible sur `/home/etudes-stmf` :

```
> cd /home/etudes-stmf
```

On trouve ici les répertoires des différents codes participant au projet, donc TrioCFD, et présents sur le gestionnaire d'études dans Tuleap.

Pour avoir accès à l'espace d'archivage via Windows, il suffit d'ouvrir un explorateur de fichier et de saisir l'adresse suivante dans la barre d'accès en haut de la fenêtre :

```
\\titania\home_projet\partage\etudes-stmf
```

Il est conseillé de connecter un lecteur sur cet emplacement afin de ne pas avoir à le rechercher à chaque fois. Comme sur Linux, vous aurez alors accès aux répertoires des différents codes participant au projet et présents sur le gestionnaire d'études dans Tuleap (voir figure III.2.10).

Sous chacun des répertoires des codes, autant de répertoires que de fiches de Suivi d'Etude présentes sur Tuleap devront être créés. A un répertoire correspond les données d'une fiche d'étude. Le lien entre la fiche étude sous Tuleap et le répertoire d'archivage sur Linux se fera via le nom donné au répertoire sur le système d'archivage qui correspondra au numéro de l'étude généré par Tuleap. Chaque répertoire d'étude est composé de plusieurs sous-dossiers avec une structure identique pour chaque étude :

- ⇒ **01-MEMO** : un ou plusieurs mémos permettant de bien identifier les données qui ont été archivées comme l'organisation des répertoires de calculs ou toute information qui semble utile pour permettre la bonne compréhension et reproductibilité de l'étude.
- ⇒ **02-Donnees_Entrees** : ce répertoire a pour but de stocker les données qui ont été nécessaires à la modélisation de l'étude (plans de l'expérience si une comparaison par rapport à l'expérimental a été faite, le jdd d'origine si il s'agit d'une reprise d'étude,...)
- ⇒ **03-Calculs** : c'est ici que sont stockés l'ensemble des jdds, maillages, post-traitement et éventuellement les résultats. Il se peut que de nombreux jdds aient été créés pour mener à bien l'étude. Il est recommandé de procéder, tout d'abord à un tri de ces calculs puis de ranger chacun d'entre eux dans un répertoire différent sous 03-Calculs par exemple Calcul01, Calcul02,... (nommage à la discrétion de chacun). Chaque répertoire de calcul contiendra le maillage propre au calcul, le jdd, le fichier de post-traitement et éventuellement, les résultats. Afin de mieux comprendre en quoi consiste chacun de ces répertoires de calcul une fiche mémo pourra être élaborée et sauvegardée dans **01-MEMO**

- ⇒ **04-Documents_Emis** : seront sauvegardés ici tous les documents émis suite à cette étude. Cela pour être un article, une présentation, une note technique,...
- ⇒ **05-Verification** : seront sauvegardés dans ce répertoire tous les retours et échanges qui auraient pu y avoir dans le cadre de la vérification de l'étude
- ⇒ **06-References** : seront sauvegardés ici les articles ou autres références bibliographiques ayant contribué aux réflexions menées durant l'étude
- ⇒ **07-Codes-scripts** : si des scripts ou des codes non suivis en gestion de configuration (git, github,...) ont été élaborés dans le cadre de l'étude, ils seront stockés dans ce sous-répertoire.

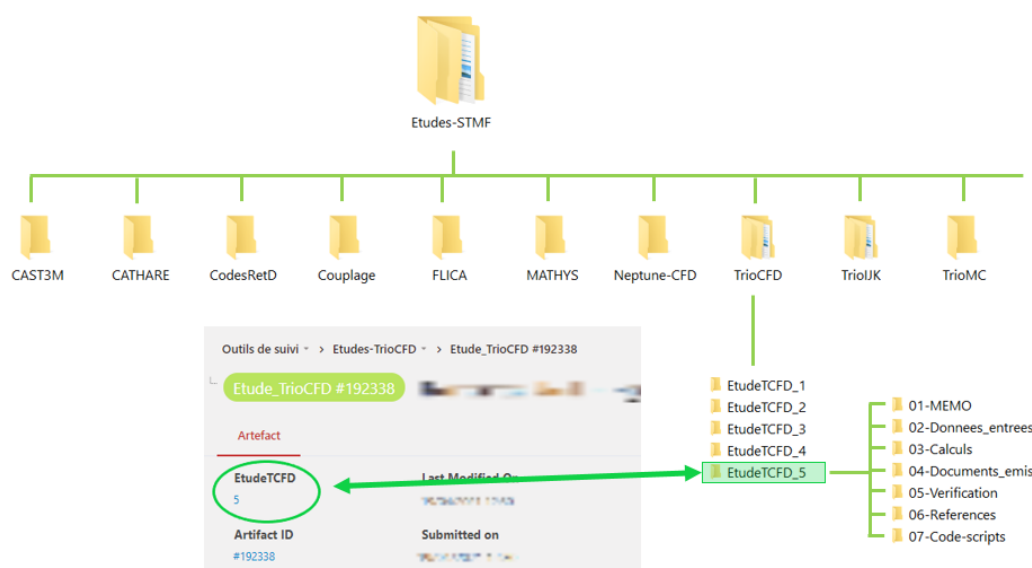


FIGURE III.2.10: Organisation de la sauvegarde des études sous Titania

Afin de générer rapidement cette arborescence, un script est présent dans le répertoire de TrioCFD. Celui-ci se nomme `gen_dossier.sh`. Il est alors demandé le numéro de la fiche pour laquelle le dossier doit être créé et l'ensemble est alors automatiquement généré. Son déroulé a lieu de la façon suivante (exemple de la figure III.2.10) :

```
> ./gen_dossier.sh           % lancement script
Numero de la fiche Tuleap ?   % demande du numero de la fiche Tuleap correspondante
5                             % numero renseigne par l'utilisateur
Repertoire EtudeTCFD_5 cree   % message de succes de la creation
```

Le remplissage de tous les répertoires présentés ci-dessus n'est, en aucun cas, obligatoire tant que toutes les données indispensables à la compréhension et la reproductibilité de l'étude sont présentes. Si un répertoire est vide, il est demandé de ne pas le supprimer et le laisser vide, en l'état.

Pour le stockage des documents, les formats de stockage des données doivent être les plus standard possibles et lisibles par une majorité d'OS à savoir des fichiers texte (jeux de données, post-traitement, scripts, latex,...) ou des fichiers pdf. En cas de sauvegarde de fichiers à format propriétaire, type Office ou LibreOffice, il est recommandé d'archiver à la fois ce type de fichier mais également leur impression au format pdf.

L'ensemble de cette méthodologie pour le suivi et l'archivage d'une étude est détaillé dans le MediaWiki du projet Tuleap Etudes-STMF à l'adresse : <https://codev-tuleap.intra.cea.fr/plugins/mediawiki/wiki/etudes-stmf/index.php?title=Accueil>.

III.3

Outil de gestion de versions (GIT)

Deux ans après le début du développement de la plateforme, celle-ci est passée en gestion de configuration, d'abord sous SCCS, puis sur ClearCase et depuis mi 2013 sur GIT. L'ensemble du code TrioCFD y est stocké que ce soit le code source, les procédures, les fiches de V&V ou la documentation. Cette base GIT est hébergée par Tuleap et se nomme triocfd-code.

Définition de la gestion de versions

Le contrôle de version, également appelé contrôle de source, désigne la pratique consistant à suivre et à gérer les changements apportés au code d'un logiciel. Les systèmes de contrôle de version sont des outils logiciels qui permettent aux équipes de développement de gérer les changements apportés au code source au fil du temps. Les logiciels de contrôle de version gardent une trace de chaque changement apporté au code dans un type spécial de base de données. Si une erreur est commise, les développeurs peuvent revenir en arrière et comparer les versions antérieures du code, ce qui leur permet de corriger l'erreur tout en minimisant les perturbations pour tous les membres de l'équipe. Les développeurs qui travaillent en équipe écrivent continuellement du nouveau code source et modifient le code source existant. Le code d'un projet logiciel est généralement organisé en une structure de dossiers ou « arborescence de fichiers ». Un développeur de l'équipe peut travailler sur une nouvelle fonctionnalité, tandis qu'un autre corrige un bug sans rapport en changeant le code. Chaque développeur peut apporter ses changements dans plusieurs parties de l'arborescence de fichiers.

Le contrôle de version aide les équipes à résoudre ce genre de problèmes, en suivant chaque changement individuel de chaque contributeur et en aidant à prévenir les conflits entre les tâches concomitantes. Les changements apportés à une partie du logiciel peuvent être incompatibles avec ceux apportés par un autre développeur travaillant en même temps. Ce type de problème doit être identifié et résolu de manière ordonnée sans bloquer le travail du reste de l'équipe. En outre, dans tout projet de développement, le moindre changement peut introduire de nouveaux bugs, et la nouvelle version du code ne peut être considéré comme fiable tant qu'elle n'a pas été testée. Les tests et le développement se poursuivent donc en parallèle jusqu'à ce qu'une nouvelle version soit prête.

Les principaux avantages du contrôle de version sont les suivants :

- **Un historique complet des changements à long terme de chaque fichier** : le moindre changement effectué par de nombreuses personnes au fil des ans. Les changements comprennent la création et la suppression de fichiers, ainsi que l'édition de leur contenu. Cet historique doit également comprendre l'auteur, la date et des notes écrites sur l'objet de chaque changement. Le fait de disposer de l'historique complet permet de revenir aux versions précédentes pour aider à l'analyse des causes profondes des bugs, ce qui est crucial lorsqu'il est question de corriger des problèmes dans les anciennes versions d'un logiciel.
- **Branching et merges** : Le fait que les membres d'une équipe travaillent simultanément est une évidence, mais il peut être intéressant pour les personnes autonomes de pouvoir travailler sur des flux de changements indépendants. La création d'une « branche » permet de garder plusieurs flux de travail indépendants les uns des autres, tout en offrant la possibilité de fusionner ces tâches, permettant ainsi aux développeurs de vérifier que les changements sur les différentes branches n'entrent pas en conflit.
- **Traçabilité** : Pouvoir retracer chaque changement apporté au logiciel et le connecter à des logiciels de gestion de projet et de suivi des BugTracker comme Tuleap, et être capable d'annoter chaque

changement avec un message décrivant son but peut contribuer à l'analyse des causes profondes d'un bug. Avoir sous la main l'historique annoté du code pendant sa lecture, essayer de comprendre ce qu'il fait et pourquoi il est ainsi conçu, peut permettre aux développeurs d'apporter des changements appropriés et harmonieux, en accord avec l'architecture prévue du système. Cela peut être particulièrement important pour travailler efficacement avec le code existant et pour permettre aux développeurs d'estimer les futures tâches avec précision.

Logiciel actuel de gestion de versions de TrioCFD : GIT

GIT est un logiciel de gestion de versions décentralisé. Le code informatique développé est stocké non seulement sur l'ordinateur de chaque contributeur du projet, mais également sur le serveur dédié. Il s'agit d'un outil de bas niveau, qui se veut simple et performant et dont la principale tâche est de gérer l'évolution du contenu d'une arborescence. GIT indexe les fichiers d'après leur somme de contrôle calculée avec la fonction de hachage SHA1 ID. Quand un fichier n'est pas modifié, la somme de contrôle ne change pas et le fichier n'est stocké qu'une seule fois. En revanche, si le fichier est modifié, les deux versions sont stockées sur le disque. Le SHA1 ID protège le code et l'historique des changements contre toute modification accidentelle ou malveillante, tout en assurant une traçabilité complète de l'historique. GIT prend en charge le branching et le tagging en priorité et les opérations qui concernent les branches et les tags (comme les merges et les reverts) sont également stockées dans l'historique des changements. GIT présente également une très grande flexibilité et est capable de prendre en charge le workflow propre à chaque projet. Celui de TrioCFD est décrit dans la section suivante.

Organisation générale et Workflow de TrioCFD sur GIT

Pour chaque modification, correction ou développement dans le code, que cela concerne les sources à proprement parler mais également la documentation, les cas tests ou les procédures, une fiche de Demande d'Intervention (voir chapitre [III.1](#)) doit être créée. Une branche correspondante sera, à son tour, créée sous GIT. La méthodologie 1 problème = 1 DI = 1 branche permet d'assurer un excellent suivi de tout changement fait dans le code et de pouvoir reconstruire une version du code pour chacun d'eux.

Après avoir créé la fiche de Demande d'Intervention, le déroulé sous GIT est le suivant :

1. **Définition du nom de la branche de travail** - assuré par le développeur : le nom de la branche dans laquelle le travail va être réalisé devra porter le formalisme suivant : TCFDXXXXXX_ACTIVITE où XXXXXX correspondra à l' "Artifact ID" (numéro d'identification) de la fiche qui a été créé suivant la méthodologie décrite dans la section [III.1](#) et ACTIVITE, l'activité sur laquelle porte la branche.
2. **Création de la branche sur le dépôt GIT local** - assuré par le développeur : à partir du dépôt local de TrioCFD, le développeur crée sa branche à partir de la version en développement du code (branche triou/TMA) :

```
> git checkout -b TCFDXXXXXX_ACTIVITE origin/triou/TMA
```

En lançant ensuite la commande `git branch`, le développeur s'assurera que la branche TCFDXXXXXX_ACTIVITE a bien été créée sur son dépôt local.

3. **Sauvegarde régulière de la branche de travail** - assurée par le développeur : après avoir effectué une partie de son développement (voir chapitre [VI.1](#) décrivant le processus de développement et sa structuration), le développeur effectue une sauvegarde de son travail sur le dépôt GIT distant. Il doit, au préalable s'être assuré que le code compilait correctement et que la base de vérification avait tourné avec succès. La sauvegarde se fait via un point de commit et la branche de développement est ensuite publiée sur le dépôt distant de TrioCFD :

```
> git add <path/to/fileName>          % ajout des fichiers voulant etre sauvegardes
> git commit                          % creation d'un point de commit
> git push origin TCFDXXXXXX_ACTIVITE % envoi de la branche sur le depot distant
```

Cette étape sera répétée autant de fois que nécessaire jusqu'à ce que les actions menées dans le cadre de cette Demande d'Intervention soient considérées comme finies.

4. **Demande de Pull Request** - assuré par le développeur : une fois l'ensemble des commits réalisés sur la branche TCFDXXXXXX_ACTIVITE ainsi que les vérifications de bonne compilation du code et du bon déroulé de la base de vérification, le développeur informe le Responsable de Code que sa

branche est prête à être intégrée dans TrioCFD via une Pull Request de sa branche vers la branche triou/TMA (voir Figure III.1.3)

5. **Relecture de la branche de développement** - assurée par le relecteur : le Responsable de Code demande à un des relecteurs de l'équipe de développement de TrioCFD de procéder à la relecture de la branche (voir la section VI.1). Il s'assure du bon respect des règles de méthodologie et de codage dans la branche TCFDXXXXXX_ACTIVITE. Il itère autant de fois que nécessaires avec le développeur afin que la branche soit propre et fonctionnelle. Suite à la relecture, le développeur peut être amené à effectuer des modifications afin que sa branche atteigne un niveau de conformité satisfaisant. Une fois cet objectif atteint, il appose le tag **OK pour intégration** au niveau de la Pull Request.

6. **Intégration de la branche de développement** - assurée par l'intégrateur : une fois la relecture achevée avec succès, l'intégrateur prend en charge son intégration dans la branche en développement de TrioCFD. Il commence par fusionner la branche de développement (TCFDXXXXXX_ACTIVITE) avec la branche en développement de TrioCFD (triou/TMA) via un **git merge**. En cas de conflits entre elles, il s'occupe de leur résolution en faisant appel, si nécessaire, au développeur. Si cette fusion n'entraîne aucun conflit, il compile cette nouvelle version obtenue par la fusion et lance une base de vérification. En cas de problème sur la base de vérification, il se tourne vers le développeur pour que celui-ci résolve les problèmes observés. Si la base de vérification passe sans difficulté et qu'aucun impact n'est prévu suite à la branche de développement, il entérine la fusion des 2 branches et envoie sur le dépôt distant la nouvelle version de la branche **triou/TMA** via la commande **git push**. Si des impacts sont à prévoir sur la base de Validation du code, celle-ci est lancée et les impacts observés, validés, avant que le développement ne soit intégré dans **triou/TMA**.

Le Workflow sous GIT, décrit ci-dessus, est illustré en Figure III.3.1.

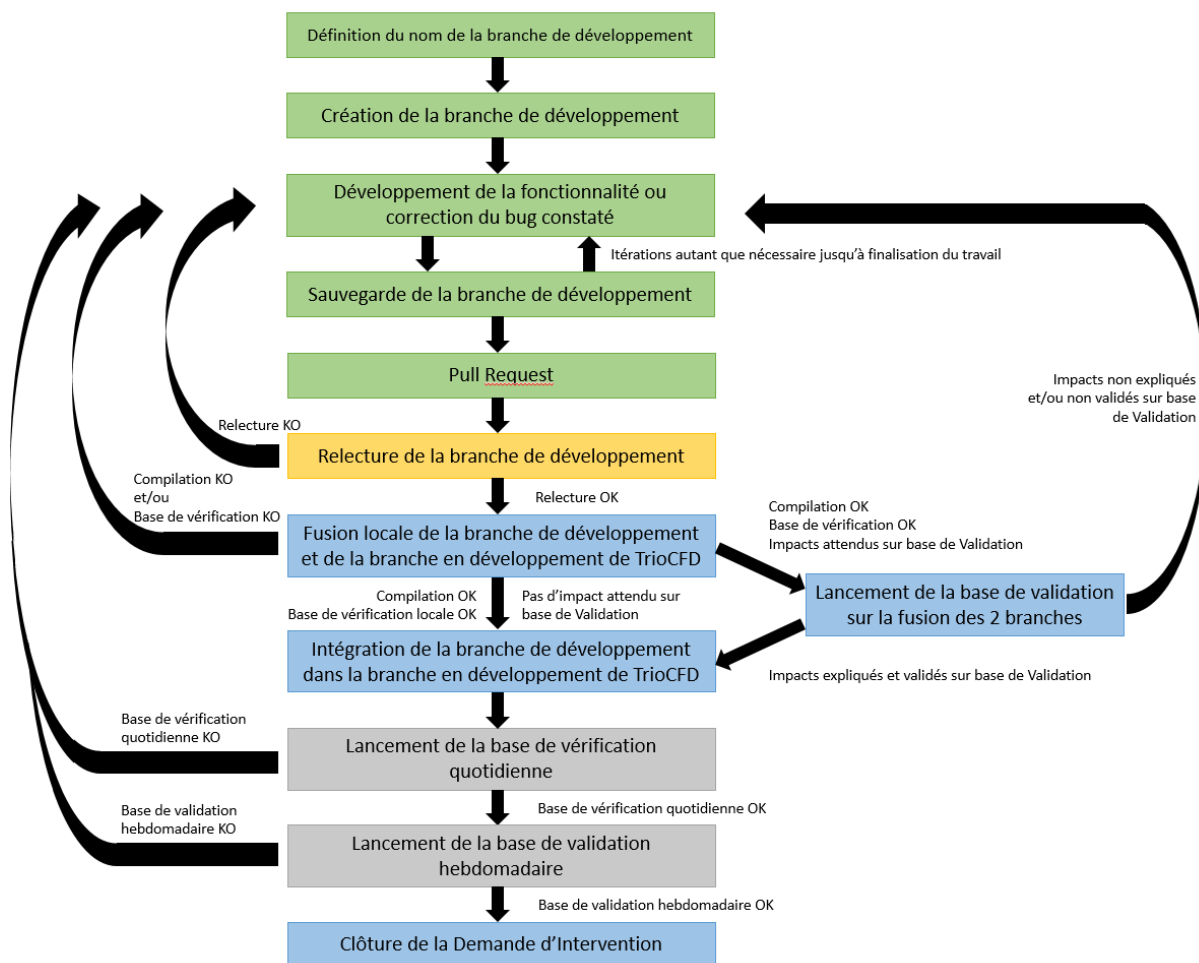


FIGURE III.3.1: Workflow de TrioCFD sous GIT pour une branche de développement - en vert : tâches développeur - en jaune : tâche relecteur - en bleu : tâche intégrateur - en gris : tâches Atelier Logiciel

En plus de garantir une bonne traçabilité pour tout changement apporté dans le code, cette méthodologie permet d'avoir une lecture facilitée de la base GIT du code. Quand cette méthodologie est étendue à tous les développements en cours sur le code, le flux sur GIT pour TrioCFD peut être illustré par la figure III.3.2. L'ensemble de cette méthodologie est décrite sur le MediaWiki du projet TrioCFD sur Tuleap (<https://codev-tuleap.intra.cea.fr/plugins/mediawiki/wiki/triocfd/index.php?title=Accueil>)

La branche `trio`/`TMA` correspond donc à la version en développement de TrioCFD qui évolue au gré des intégrations en continu dans le code et sur laquelle tous les développeurs du code CEA doivent travailler. La branche `master`, quant à elle, est mise à jour à chaque sortie de version et correspond à la dernière version livrée du code. A la mise à jour de la branche `master` pour la sortie de version, un tag est également apposé et nommé suivant le numéro de la version livrée.

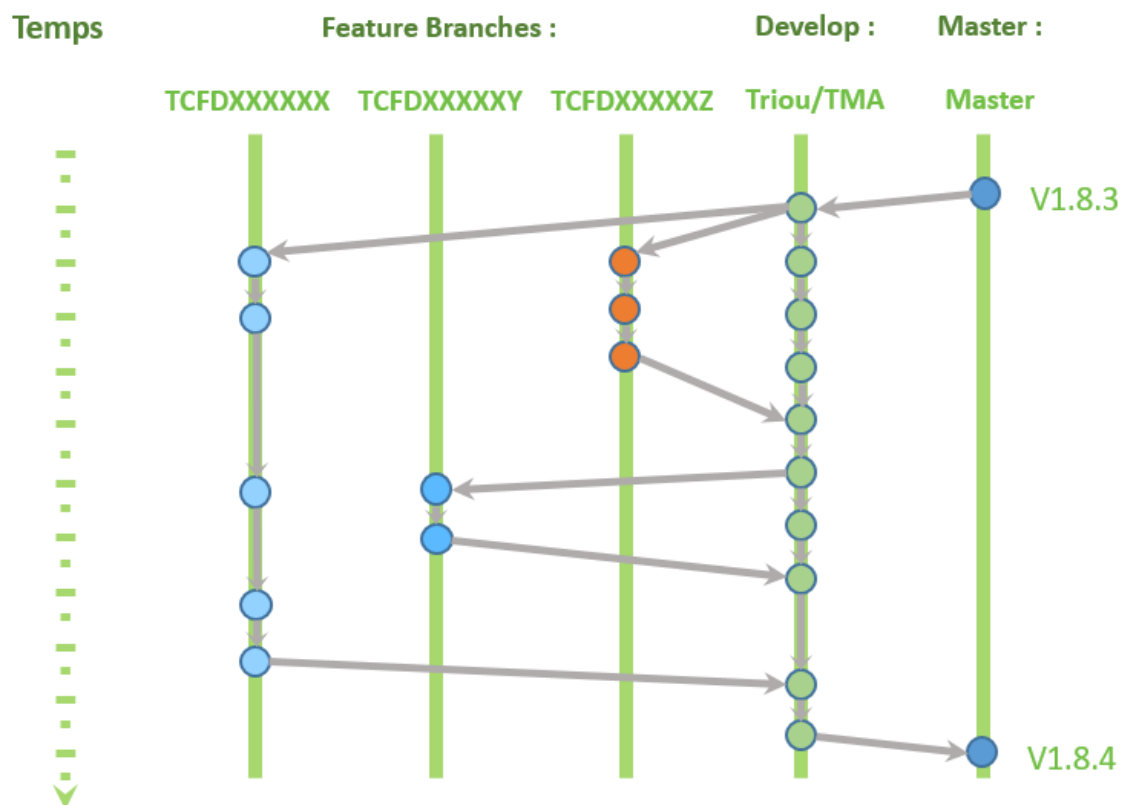


FIGURE III.3.2: Workflow général de TrioCFD sous GIT

La branche **triou/TMA** est mise à jour quotidiennement par l'Atelier de Génie Logiciel sur le dépôt extérieur de TrioCFD hébergé sur GITHUB à cette adresse : <https://github.com/cea-trust-platform/TrioCFD-code>. Ce dépôt extérieur permet aux développeurs hors CEA de travailler de la même façon que les développeurs CEA avec une version en développement du code à jour, nommée **next** sur GITHUB. Pour ce faire, ils créent également une **Issue** (qui est l'équivalent d'une Demande d'Intervention sur Tuleap) et nomment leur branche, en fonction, sous GITHUB. Afin de centraliser sur un seul Tracker l'ensemble des actions concernant TrioCFD, le Responsable de Code aura la charge de basculer l'ensemble des données présentes sur GITHUB sur Tuleap et renommer la branche de GITHUB en conséquence sur GIT avant son intégration dans le code.

III.4

Messagerie

Afin d'échanger avec la TMA, une adresse de messagerie est à disposition des utilisateurs : trust@cea.fr. Cette adresse est exclusivement gérée par la TMA.

Il s'agit de l'unique moyen de contact entre les utilisateurs du code et l'équipe de TMA. L'adresse personnelle des agents TMA n'est pas utilisée dans le cadre du projet. Cette boîte unique permet de conserver un historique complet des échanges de la TMA avec les utilisateurs. Chaque membre de l'équipe TMA a les droits en lecture et écriture sur cette messagerie.

Elle est utilisée pour répondre aux questionnements des utilisateurs mais également pour communiquer des informations pratiques pour les sessions de formations par exemple. Le délai contractuel de traitement des mails reçus via cette adresse par l'équipe TMA est d'un jour ouvré. Il est donc fortement recommandé de passer exclusivement par ce biais pour contacter le support (cela n'empêche cependant pas de mettre en copie le RPL ou le Responsable de Code) afin de garantir la bonne traçabilité des échanges et d'être également assuré d'avoir un retour dans les plus brefs délais.

III.5

Interfaçage pour l'utilisateur et le développeur

Plusieurs scripts et outils sont à la disposition des utilisateurs afin de faciliter l'utilisation de la plateforme TRUST/TrioCFD. Une procédure est dédiée à chaque action effectuée sur la plateforme. Ces outils étant nombreux, ils n'ont pas pu tous être détaillés pour la première version de ce document mais cette partie sera étayée au fil des différentes versions.

Installation et compilation de TrioCFD

On considère qu'une version TRUST au même niveau de développement est à disposition. On commence par charger l'environnement de TRUST de cette version :

```
> source trust-code/env_TRUST.sh
```

On configure le passage en mode "Baltik" :

```
> baltik_build_configure
```

On lance la configuration de TrioCFD :

```
> ./configure
```

On lance la compilation de TrioCFD dans le mode souhaité :

```
> make optim semi-optim debug
```

Lancement d'un cas test

Pour lancer un calcul en mode séquentiel, la méthodologie est la suivante :

```
> cd $my_test_directory
> trust my_data_file %whitout .data
```

Il s'agit du lancement standard d'un calcul. D'autres options sont disponibles pour effectuer cette action et sont détaillés ci-après. La syntaxe de lancement devient alors :

```
> trust [option] my_data file [nb_cpus] [1>file.out] [2>file.err]
```

où **nb_cpus** correspond au nombre de processeurs utilisés pour lancer le calcul (ce nombre doit être égal au nombre de sous-domaines définis dans le jeu de données),

où les **options** disponibles sont :

OPTION	Description
-help -h	List options
-baltik	Instantiate an empty Baltik project
-index	Access to the TRUST ressource index
-doc	Access to the TRUST manual (Generic Guide)
-config nedit vim emacs gedit	Configure nedit or vim or emacs with TRUST keywords
-edit	Edit datafile
-xedit	Edit datafile with xdata
-xcheck	Check the datafile's keywords with xdata
-partition	Partition the mesh to prepare a parallel calculation (Creation of the .Zones files)
-mesh	Visualize the mesh(es) contained in the data file
-probes	Monitor the TRUST calculation only
-evol	Monitor the TRUST calculation (new IHM)
-wiz	Create a TRUST datafile from a MED file (new IHM)
-prm	Write a prm file
-clean	Clean the current directory from all the generated files by TRUST
-search keywords	Know the list of test cases from the data bases which contain keywords
-copy	Copy the test case datafile from the TRUST database under the present directory
-check all testcase list	Check the non regression of all the test cases or a single test case or a list of tests cases specified in a file
-check function class class : :method	Check the non regression of a list of tests cases covering a function, a class or a class method
-gdb	Run under gdb debugger
-valgrind	Run under valgrind
-valgrind_strict	Run under valgrind with no suppressions
-create_sub_file	Create a submission file only
-prod	Create a submission file and submit the job on the main production class with exclusive resource
datafile -help_trust	Print options of TRUST_EXECUTABLE [CASE[.data]] [options]

TABLE III.5.1: Options disponibles pour le lancement d'un cas-test

IV. Description des Outils de Vérification et Validation

POUR tout code, les processus de Vérification et Validation sont primordiaux puisqu'ils permettent de connaître l'état du code à tout moment et de garantir son comportement sur un large panel de cas allant des plus simples aux plus compliqués. Ces 2 processus ont des objectifs différents : Le processus de **Vérification** permet de s'assurer que les équations sont **correctement** résolues tandis que le processus de **Validation** permet de vérifier que **les** bonnes équations sont résolues.

Dans cette partie, l'application de ces 2 processus dans le code **TrioCFD** sera détaillée.

Une autre notion est également extrêmement importante dans le processus de V&V du code, à savoir la portabilité du code. En effet, un code de calcul doit être le moins sensible possible à l'environnement sur lequel il est utilisé que ce soit la configuration matérielle (processeur) ou l'environnement logiciel (système d'exploitation, compilateur,...). Ainsi, afin d'avoir la meilleure portabilité possible, un code ne doit utiliser que la bibliothèque logicielle "standard" disponible sur plusieurs environnements et non pas certaines fonctions spécifiques à un environnement particulier. Le code pourra ainsi être porté sans difficulté d'une machine à l'autre en le recompilant avant utilisation. Grâce à l'utilisation de bibliothèques "standard", le code est alors adapté à un plus grand nombre d'utilisateurs.

Afin de garantir une bonne portabilité de la plateforme **TRUST/TrioCFD**, des bibliothèques nécessaires, plus riches que les bibliothèques communes sont fournies, pour chaque version, dans les **externalpackages** de **TRUST** (*e.g.* PETSc, Miniconda,...). La plateforme **TRUST/TrioCFD** dispose donc d'une bonne portabilité native. La surveillance régulière de celle-ci permet donc de s'assurer qu'aucun codage ambigu ou non conforme soit introduit par inadvertance dans le code. Celui-ci pourrait être interprété différemment suivant l'environnement considéré, entraînant des écarts inexplicables. Le diagnostic de ce genre de problème étant souvent complexe à établir, une identification au plus tôt permet alors de rectifier rapidement. Pour ce faire, **TRUST** et **TrioCFD** sont testés quotidiennement sur un large panel de configurations qui sera cartographié dans le premier chapitre de cette partie.

IV.1

Parc des machines

Afin de s'assurer de la bonne portabilité de **TrioCFD**, un parc d'un peu plus de vingt de machines, présentant des configurations distinctes, a été mis en place et est régulièrement mis à jour au en fonction de l'évolution des systèmes d'exploitation, compilateurs et autres bibliothèques. En voici la constitution :

CIBLE	CORES	RAM	CACHE	FREQ.	OS	COMPILATEUR	MPI
cluster1-amd-gpu	8	131	512	2095	CentOS release 7.6.1810	g++ 9.1.0	OpenMPI 3.1.4
cluster1-amd	8	131	512	2095	CentOS release 7.6.1810	icpc 19.0.3.199	Intel MPI 2019
cluster1-intel	8	97	16896	2300	CentOS release 7.6.1810	icpc 19.0.3.199	Intel MPI 2019
cluster2	56	131	19712	3157	CentOS release 7.9.2009	g++ 4.8.5	MPICH 3.2
cluster3	8	131	35840	2401	Red Hat Server release 7.9	icpc 18.0.3	OpenMPI 4.0.2rc3
cluster4-amd-build64	8	263	512	2500	Red Hat Server release 7.9	icpc 19.1.3.304	OpenMPI 4.0.2rc3
cluster4-amd-gpu	8	263	512	2500	Red Hat Server release 7.9	g++ 8.3.0	OpenMPI 4.0.2rc3
cluster4-amd	8	263	512	2500	Red Hat Server release 7.9	icpc 19.1.3.304	OpenMPI 4.0.2rc3
cluster4-build64	8	196	33792	2701	Red Hat Server release 7.9	icpc 19.0.5.281	OpenMPI 4.0.2rc3
cluster4	8	263	512	2500	Red Hat Server release 7.9	icpc 19.1.3.304	OpenMPI 4.0.2rc3
station1	6	16	9216	3989	CentOS release 7.9.2009	g++ 4.8.5	MPICH 3.2
station2	8	8	8192	1596	CentOS release 7.9.2009	g++ 4.8.5	MPICH 3.2
station3	24	16	12288	2538	Ubuntu 18.04.5 LTS	g++ 7.5.0	MPICH 3.2
station4	4	8	6144	3199	Ubuntu 16.04.7 LTS	g++ 5.4.0	MPICH 3.2
station5	24	16	12288	1596	CentOS release 7.9.2009	g++ 4.8.5	MPICH 3.2
station6	12	12	12288	1801	Ubuntu 20.04.2 LTS	g++ 9.3.0	MPICH 3.2
station7	4	8	8192	3402	Fedora release 24	g++ 6.1.1	MPICH 3.2
station8	24	32	15360	1200	CentOS release 6.4	g++ 10.3.0	MPICH 3.2
station9	24	32	15360	1340	Fedora release 26	g++ 7.1.1	MPICH 3.2
station10	4	16	8192	3311	CentOS release 7.9.2009	g++ 4.8.5	MPICH 3.2
station11	4	16	8192	1501	CentOS release 7.9.2009	g++ 4.8.5	MPICH 3.2
station12	4	16	8192	3157	CentOS release 7.9.2009	g++ 4.8.5	OpenMPI 2.0.4
station13	32	65	11264	800	Ubuntu 18.04.5 LTS	g++ 7.5.0	MPICH 3.2
station14	6	16	9216	3910	CentOS release 7.9.2009	g++ 4.8.5	MPICH 3.2
station15	6	16	9216	3964	CentOS release 7.9.2009	g++ 4.8.5	MPICH 3.2
station16	16	16	16384	2551	Fedora release 30	g++ 9.0.1	MPICH 3.2
station17	12	32	8448	1673	Ubuntu 20.04.2 LTS	g++ 9.3.0	MPICH 3.2
station18-clang	8	32	12288	3600	Fedora release 32	clang++ 10.0.0	OpenMPI 2.0.4
station19-gpu	8	32	12288	4514	Fedora release 32	cuda-g++ 8.3.0	OpenMPI 4.1.0
cluster5-build64	8	263	512	2184	Red Hat release 8.3	icpc 19.1.3.304	OpenMPI 4.0.5

TABLE IV.1.1: Parc des machines de tests de TrioCFD

C'est sur ce parc de machines que sont lancées la vérification et la validation.

IV.2

Base de Vérification

Objectifs et étapes de la Vérification

Le processus de **Vérification** a pour but de s'assurer que le code satisfait pleinement toutes les exigences attendues. La vérification doit répondre à la question suivante : " Est-ce que nous construisons le code correctement ? " Cela correspond à vérifier que les spécifications sont correctement mises en œuvre par le système. Elle permet de déterminer si le résultat d'une phase de développement donnée satisfait les conditions imposées au début de cette phase. La vérification du logiciel garantit que celui-ci a été bien construit et confirme que le code répond aux plans des développeurs.

La vérification quotidienne porte sur les aspects suivants :

- ⇒ **la bonne compilation du code** : toutes les nuits, la compilation du code est lancée en mode optimisé et semi-optimisé sur plusieurs machines du parc sur la branche de développement (branche triou/TMA). La compilation en mode debug est lancée tous les week-end car ce mode de compilation nécessite un temps trop conséquent au vu des contraintes du temps d'occupation des machines du parc. En cas de non succès de la compilation, la version de développement du code est considérée comme étant en défaut et un correctif doit lui être apportée dans les plus brefs délais.
- ⇒ **la bonne réalisation des cas-tests** : après la compilation, la base de cas-tests (comprenant à la fois des tests élémentaires et des tests issus des fiches de validation) est lancée sur 3 pas de temps. On s'attache à vérifier que ces 3 pas de temps ont été réalisés avec succès mais également que les résultats physiques obtenus sur ces 3 pas de temps sont conformes aux résultats attendus. Pour ce faire, les résultats attendus sur ces 3 pas de temps sont stockés en gestion de configuration dans la base GIT du code sous la forme de fichiers lml zippés (fichiers .lml.gz). Les résultats obtenus avec la vérification sont alors comparés sous format binaire avec les résultats en gestion de configuration (attendus). En cas d'interruption de cas-tests ou de non-conformité des résultats physiques avec l'attendu, le code est considéré comme défaillant. Une action doit être menée, soit sur le code, soit sur le(s) cas-test afin de rétablir la situation.
- ⇒ **la portabilité du code** : la compilation du code ainsi que la base de tests de vérification sont lancées sur des ordinateurs ou stations de travail ayant différentes configurations matérielles et/ou logicielles (voir chapitre IV.1). Sur chacune de ces machines, la version de développement du jour de TrioCFD doit remplir correctement les deux critères précédents (compilation et réalisation de la base de cas-test), assurant la bonne portabilité quotidienne du code.
- ⇒ **la génération de la documentation** : la bonne génération de la documentation est également testée toute les nuits. Toutefois, à l'heure actuelle, l'ensemble de la documentation n'est pas testé. Seul le TrioCFD Reference Manuel, construit à partir de balises XDATA présentes dans le code (voir paragraphe), l'est. La génération automatique de l'ensemble de la documentation et sa vérification constituent une voie d'amélioration d'ores et déjà identifiée.

La base de cas-tests de TrioCFD

Revenons maintenant un peu plus en détails sur la base de cas-tests de TrioCFD. Celle-ci comprend 1533 tests séquentiels et 444 tests parallèles (voir tableau récapitulatif en Annexe). Ces cas-tests couvrent l'ensemble des baltiks de TrioCFD et assurent un taux de couverture de 80.05% des mot-clés (1349 mots-clés sont testés

sur les 1685 mots-clés de **TrioCFD**). Les cas-tests sont rangés dans le repertoire **tests** de chacun des baltiks. Un cas-test a pour extension **.data** et peut être soit un test simple (test de Référence), soit un test extrait d’une fiche de validation. Dans ce dernier cas, comme une fiche de validation peut comporter plusieurs cas-tests, il aura un suffixe de la forme **_jddX.data** où X variera de 1 au nombre de jeux de données de la fiche de validation.

Chaque cas-test **.data** est rangé dans un répertoire du même nom qui contient également le résultat de référence de ce cas-test (fichier au format **.lml.gz**). Comme mentionné précédemment, ces cas-tests sont lancés sur les 3 premiers pas de temps dans le cadre du processus de vérification afin de s’assurer que :

⇒ le calcul se déroule correctement sur ces 3 pas de temps ;

⇒ les résultats obtenus sur ces 3 pas de temps ne présentent pas d’écarts avec les résultats de référence.

Afin que la base de vérification puisse être lancée aussi souvent que nécessaire, son temps de réalisation doit rester abordable. Sur un poste de travail standard (*e.g.* is241762 - voir tableau [IV.1.1](#)), les 1977 cas-tests s’exécutent en 2047.2s (2s/test), soit un peu moins d’une heure, le plus long test durant 89 secondes.

Fréquence de lancement de la base de cas-tests de TrioCFD

Chaque nuit, l’Atelier de Génie Logiciel lance, sur les différentes machines du parc, la vérification de **TrioCFD**. Ces machines, correspondant, pour la plupart aux stations de travail des développeurs, ne peuvent donc être utilisées pour la vérification en dehors des heures de travail. Par conséquent, le processus de vérification quotidien doit être mené sur un temps d’exécution raisonnable durant la semaine. La vérification plus poussée aura lieu durant le week-end. La base de vérification est automatiquement lancée tous les soirs à 19h15 et est stoppée à 7h le matin si elle n’est pas parvenue à terme.

Outre son lancement quotidien automatique, le développeur doit s’assurer de son bon fonctionnement avant chaque demande d’intégration (Pull-Request) dans le code, vérification qui devra être de nouveau effectuée par l’intégrateur avant d’intégrer une branche de développement/correction dans la branche de développement de **TrioCFD** (branche **triou/TMA**) (voir chapitre [III.3](#)).

Méthodologie de lancement de la base de cas-tests de TrioCFD

Pour le lancement de la base de vérification de façon manuelle, l’utilisateur doit avoir préalablement compilé sa branche de développement de **TrioCFD** (suivant la procédure détaillée dans la section [III.5](#)). Il crée alors le lien vers la version compilée et peut lancer la base de vérification de la façon suivante :

```
> source env_TrioCFD.sh      % source de l’environnement de TrioCFD
> make check_all_optim      % lancement de la base de verification
```

Une fois la base de vérification terminée, la procédure renvoie le bilan de celle-ci , dont un exemple peut être visualisé sur la figure [IV.2.1](#).

```

16:56:34|16:56:35|0.3s|.1|. ....|. ....0|. ....0|. ..2400|. ..52Mo|. ....|OK|wl_vef_correlation_jdd5
16:56:35|16:56:36|0.1s|.1|. ....|. ....0|. ....0|. ..2400|. ..41Mo|. ....|OK|wl_vef_coupling_jdd1
16:56:37|16:56:38|0.1s|.1|. ....|. ....0|. ....0|. ..2400|. ..41Mo|. ....|OK|wl_vef_coupling_jdd2
16:56:38|16:56:46|6.6s|.1|. ....|. ....0|. ....0|. ..2400|. ..43Mo|. ....|OK|wl_vef_coupling_jdd3
16:56:46|16:56:54|6.9s|.1|. ....|. ....0|. ....0|. ..2400|. ..43Mo|. ....|OK|wl_vef_coupling_jdd4
16:56:54|16:56:55|0.1s|.1|. ....|. ....0|. ....0|. ..2400|. ..39Mo|. ....|OK|wl_vef_laminar_jdd1
16:56:56|16:56:57|0.1s|.1|. ....|. ....0|. ....0|. ..2400|. ..40Mo|. ....|OK|wl_vef_laminar_jdd2
16:56:57|16:56:58|0.1s|.1|. ....|. ....0|. ....0|. ..2400|. ..39Mo|. ....|OK|wl_vef_laminar_jdd3
16:56:58|16:56:59|0.1s|.1|. ....|. ....0|. ....0|. ..2400|. ..39Mo|. ....|OK|wl_vef_laminar_jdd4
16:57:00|16:57:01|0.0s|.1|. ....|. ....0|. ....0|. ....4|. ..35Mo|. ....|OK|FTD_couplage2
16:57:01|16:57:02|0.0s|.2|. 0.505|. ....|. ....0|. ....4|. ..71Mo|. ..10/10|OK|PAR_FTD_couplage2
16:57:03|16:57:04|0.1s|.1|. ....|. ....0|. ....0|. ..500|. ..35Mo|. ....|OK|CondVDF_VEF
16:57:04|16:57:05|0.2s|.1|. ....|. ....0|. ....0|. ....51|. ..34Mo|. ....|OK|DocondVDF_VDF
-----
Successful tests cases :1531/1532
Successful tests cases in parallel mode :443/443
1975 test cases run in 2941.7 s ( 2 s/test), so less than 1 hours
Parallel tests cases :
  127 PARALLEL NOT
  434 PARALLEL OK
   32 PARALLEL ONLY
   88 PARALLEL RUNS
    4 PARALLEL STOPS
4 test cases skipped
4 test cases not compared to the reference
List of unsuccessful tests cases :
Turbul_RANS_Robin_wall_law

```

FIGURE IV.2.1: Exemple de bilan du lancement de la base de vérification lancée en local

Pour le lancement automatique quotidien, l'atelier de TrioCFD, contenant un ensemble de procédures en Bash et en Python, est exécuté chaque soir par l'atelier de TRUST par un crontab (procédure de lancement automatique et périodique) sur le script `lance_test_nuit` contenu dans le répertoire `bin/admin` de TRUST. Ce script met à jour TrioCFD par rapport à ce qui a été intégré la veille dans la branche `triu/TMA`, installe TRUST, puis envoie TrioCFD (et les autres baltiks sur base TRUST) sur les machines du parc (voir tableau IV.1.1). Les machines sur lesquelles TRUST et ses Baltiks sont testés chaque nuit sont spécifiées dans le fichier `bin/admin/liste.machines` de TRUST. Les machines concernant TrioCFD sont celles qui contiennent le paramètre `-TrioCFD` dans ce fichier au niveau de la colonne `baltiks`. Ce fichier contient également les options de compilation et de lancement des tests. Quelques-unes de ces options sont données dans le tableau IV.2.1.

Option	Signification
-g	lancement des cas tests en mode debug
-semi_opt	lancement des cas tests en mode semi optimisé
-gui	test des balises XDATA (génération Référence Manual)
-clang	permet de spécifier un compilateur CLANG (variante de compilateur - par défaut GNU)
-cuda	lancement des tests sur les GPC - par défaut sur les CPU
-gcov	test la couverture des mots clés par la base de vérification
-limited	lance une base plus succincte de cas tests (20)
-all	lance l'ensemble des cas tests)
-build_64_bit	lancement sur 64 bits
-test	lancement des cas tests de TRUST avec l'exécutable de TrioCFD
-platform	profiling : analyse les temps de passage dans les différentes parties du code
-disable-mpi	désactive le lancement en parallèle des cas tests
-download-visit	télécharge la dernière version de Visit plutôt que d'utiliser la version présente dans TRUST

TABLE IV.2.1: Liste des options utilisées par l'atelier logiciel pour le lancement de la base de vérification

Le script `lance_test_nuit` installe d'abord TrioCFD sur la machine de l'atelier, puis crée un fichier qui permet de spécifier les options de compilation et de tests pour chacune des machines distantes. Ce fichier s'appelle `Run.liste` et contient autant de lignes de machines et spécifie, entre autre, le nom de la machine, le chemin vers TRUST, le chemin d'installation, le mode de tests,...

Le lancement des installations sur les machines distantes se fait avec le script `/export/home/-triuou/tuleap/Livraison/TRUST/bin/baltik/share/baltik/bin/baltik_check_portability`
`Run.liste`. Ce script va :

1. créer l'archive `trio CFD.tar`, qui contient le `trio CFD.tar.gz` et les autres scripts pour préparer, installer, configurer, vérifier, ..
2. la copier sur chacune des machines distantes et lancer la compilation
3. récupérer les logs en local de chacune des machines et générer la page `nuit_trio CFD.html` qui permettra l'analyse des résultats de vérification.

Le script `lance_test_nuit` rapatrie les résultats de TRUST à 7h du matin, et stoppe la procédure de vérification de TrioCFD sur les machines pour lesquelles l'installation ou les tests ne sont pas terminés. Ceci garantit que la page `nuit_trio CFD.html` est correctement générée. Cette page servira au dépouillement des résultats de la base de vérification.

Dépouillement des résultats de la base de cas-tests de TrioCFD

L'analyse des résultats obtenus par la base de vérification de TrioCFD commence par l'analyse de la page html générée par le script `lance_test_nuit` (voir figure IV.2.2). Cette page de rapport a été refondue en 2021 afin de faire figurer les éléments nécessaires pour analyser efficacement les résultats de la base de vérification.

Elle est composée de 15 colonnes, les colonnes 1, 4, 5 et 6 donnent les informations générales sur les machines testées (nom, OS, architecture, et compilateur C++). Les colonnes 2 et 3 renseignent sur la date et l'heure de démarrage de l'atelier de vérification sur chacune des machines. La 7^{ème} colonne identifie si la machine est considérée comme une machine cible (machine sur laquelle le succès de la vérification est indispensable pour considérer la version du jour comme stable). Ces machines sont au nombre de 4 sur TrioCFD. Elles correspondent à des machines présentant une configuration standard en terme d'OS et compilateur par rapport aux machines des utilisateurs/développeurs. Ainsi, si l'atelier de vérification ne fonctionne pas sur ces machines cibles, l'utilisation du code sera dégradée dans l'équipe. La 13^{ème} colonne correspond au mode

de compilation de de TrioCFD sur la machine concernée. Toutes les autres colonnes sont quant à elles destinées à l'analyse de la base de vérification :

- ⇒ **Colonne 8 - prepare** : vérifie que le désarchivage de l'archive envoyée par la machine de l'atelier s'est bien déroulé et génère les scripts de lancement propres à chaque machine (`configure.sh`, `make.sh`,...) en fonction des options définies dans le fichier `liste.machines`
- ⇒ **Colonne 9 - configure** : vérifie que la configuration de TrioCFD s'est bien déroulée (commandes `baltik_build_configure` et `./configure`)
- ⇒ **Colonne 10 - make** : vérifie que la compilation de TrioCFD a été faite avec succès en respectant les options de compilations spécifiées
- ⇒ **Colonne 11 - make_check** : lancement des tests de vérification en respectant les options spécifiées dans le fichier `liste.machines`
- ⇒ **Colonne 12 - make_install** : vérifie que l'installation générale a été bien faite et le baltik bien chargé. Cet état est toujours valide si les 4 présents ont abouti avec succès. Cette colonne pourra être supprimée lors de la prochaine évolution de l'atelier de vérification
- ⇒ **Colonne 14 - test_par** : nombre de tests en mode parallèle réussis par rapport au nombre de tests lancés
- ⇒ **Colonne 15 - test_seq** : nombre de tests en mode séquentiel réussis par rapport au nombre de tests lancés

Le résultat pour chacune de ces 6 colonnes est analysable dans le détail en cliquant sur le lien présent dans chacune des cases qui renvoie aux fichiers `.log` ou `.err`. L'équipe TMA assure le suivi quotidien du bon déroulement de l'atelier de vérification. En cas de problème, l'équipe TMA et les Responsable de Code s'attacheront à résoudre au plus vite les problèmes constatés et aucune nouvelle intégration ne sera faite dans la branche en développement de TrioCFD tant que l'état de l'atelier de vérification ne sera pas revenu dans son état nominal.

Machine	date_debut	date_fin	Os	model	release	cible	prepare	configure	make	make_check	make_install	CC	make	test_pas	test_req
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 17:30:00 CEST 2021	CentOS Linux release 7.6.1810 (Core)	x86_64	[3.10.0-957.1.3.el7.x86_64]	0	OK	OK	OK	OK		g++ 9.1.0	option	444/444	1515/1536
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 14:39:19 CEST 2021	CentOS Linux release 7.6.1810 (Core)	x86_64	[3.10.0-957.1.3.el7.x86_64]	0	OK	OK	OK	OK		icpc 19.0.3.199	option	444/444	1527/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 13:03:28 CEST 2021	CentOS Linux release 7.6.1810 (Core)	x86_64	[3.10.0-957.1.3.el7.x86_64]	0	OK	OK	OK	OK		icpc 19.0.3.199	option	444/444	1527/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 16:39:06 CEST 2021	CentOS Linux release 7.9.2009 (Core)	x86_64	[3.10.0-1160.3.2.el7.x86_64]	0	OK	OK	OK	OK		g++ 4.8.5	debug	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 04:29:53 CEST 2021	Ubuntu 20.04.2 LTS	x86_64	[5.4.0-74-generic]	0	OK	OK	OK	OK		g++ 9.3.0-17ubuntu1~20.04	option	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 14:01:55 CEST 2021	Ubuntu 18.04.5 LTS	x86_64	[4.15.0-14-generic]	0	OK	OK	OK	OK		g++ 7.5.0-3ubuntu1~18.04	debug	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 03:48:09 CEST 2021	CentOS Linux release 7.9.2009 (Core)	x86_64	[3.10.0-1160.3.11.el7.x86_64]	0	OK	OK	OK	OK		g++ 4.8.5	option	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 03:12:55 CEST 2021	Red Hat Enterprise Linux Server release 8.2 (Ootpa)	x86_64	[4.18.0-193.el8.x86_64]	0	OK	OK	OK	OK		icpc 18.0.3	option	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 07:16:03 CEST 2021	Red Hat Enterprise Linux Server release 7.9 (Maipo)	x86_64	[3.10.0-1160.3.11.el7.x86_64]	0	OK	OK	OK	OK		icpc 19.1.3.304	option	444/444	1527/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 06:49:37 CEST 2021	Red Hat Enterprise Linux Server release 7.9 (Maipo)	x86_64	[3.10.0-1160.3.11.el7.x86_64]	0	OK	OK	OK	OK		g++ 8.3.0	option	444/444	1515/1536
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 14:12:16 CEST 2021	Red Hat Enterprise Linux Server release 7.9 (Maipo)	x86_64	[3.10.0-1160.3.11.el7.x86_64]	0	OK	OK	OK	OK		icpc 19.1.3.304	option	444/444	1527/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 06:06:17 CEST 2021	Red Hat Enterprise Linux Server release 7.9 (Maipo)	x86_64	[3.10.0-1160.3.11.el7.x86_64]	0	OK	OK	OK	OK		icpc 18.0.3	option	444/444	1527/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 06:00:03 CEST 2021	Red Hat Enterprise Linux Server release 7.9 (Maipo)	x86_64	[3.10.0-1160.3.11.el7.x86_64]	0	OK	OK	OK	OK		icpc 19.1.3.304	option	444/444	1527/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 03:06:07 CEST 2021	Red Hat Enterprise Linux Server release 7.9 (Maipo)	x86_64	[3.10.0-1160.3.11.el7.x86_64]	0	OK	OK	OK	OK		icpc 18.0.3	option	444/444	1527/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 03:06:06 CEST 2021	CentOS Linux release 7.9.2009 (Core)	x86_64	[3.10.0-1160.25.1.el7.x86_64]	0	OK	OK	OK	OK		g++ 4.8.5	option	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 06:07:19 CEST 2021	CentOS Linux release 7.9.2009 (Core)	x86_64	[3.10.0-1160.25.1.el7.x86_64]	1	OK	OK	OK	OK		g++ 4.8.5	option	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 17:14:18 CEST 2021	Ubuntu 18.04.5 LTS	x86_64	[4.15.0-14-generic]	0	OK	OK	OK	OK		g++ 7.5.0-3ubuntu1~18.04	debug	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 03:51:55 CEST 2021	Ubuntu 16.04.7 LTS	x86_64	[4.4.0-10-generic]	1	OK	OK	OK	OK		g++ 5.4.0-6ubuntu1~16.04.12	option	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 04:19:59 CEST 2021	Debian GNU/Linux 9	x86_64	[4.9.0-14-amd64]	0	OK	OK	OK	OK		g++ 6.3.0-18-deb9u1	option	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 03:36:34 CEST 2021	Fedora release 34 (Twenty Four)	x86_64	[4.5.3-300.fc24.x86_64]	0	OK	OK	OK	OK		g++ 8.1.1	option	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 04:21:56 CEST 2021	CentOS release 6.4 (Final)	x86_64	[2.6.32-558.14.1.el6.x86_64]	0	OK	OK	OK	OK		g++ 10.2.0	option	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 03:38:47 CEST 2021	CentOS Linux release 7.9.2009 (Core)	x86_64	[3.10.0-1160.25.1.el7.x86_64]	0	OK	OK	OK	OK		g++ 4.8.5	option	438/444	1526/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 03:26:54 CEST 2021	CentOS Linux release 7.9.2009 (Core)	x86_64	[3.10.0-1160.25.1.el7.x86_64]	0	OK	OK	OK	OK		g++ 4.8.5	option	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 03:39:48 CEST 2021	CentOS Linux release 7.9.2009 (Core)	x86_64	[3.10.0-1160.25.1.el7.x86_64]	0	OK	OK	OK	OK		g++ 4.8.5	option	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 03:52:08 CEST 2021	Ubuntu 18.04.5 LTS	x86_64	[4.4.0-74-generic]	0	OK	OK	OK	OK		g++ 7.5.0-3ubuntu1~18.04	option	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 08:49:27 CEST 2021	CentOS Linux release 7.9.2009 (Core)	x86_64	[3.10.0-1160.25.1.el7.x86_64]	0	OK	OK	OK	OK		g++ 4.8.5	option	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 03:05:04 CEST 2021	CentOS Linux release 7.9.2009 (Core)	x86_64	[3.10.0-1160.25.1.el7.x86_64]	0	OK	OK	OK	OK		g++ 8.3.0-6	option	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 04:37:17 CEST 2021	Debian GNU/Linux 10	x86_64	[4.19.0-16-amd64]	0	OK	OK	OK	OK		g++ 9.0.1	debug	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 07:26:50 CEST 2021	Fedora release 30 (Thirty)	x86_64	[5.0.9-301.fc30.x86_64]	1	OK	OK	OK	OK		clang++ 10.0.0	option	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 03:20:08 CEST 2021	Ubuntu 20.04.2 LTS	x86_64	[5.8.0-59-generic]	0	OK	OK	OK	OK		g++ 9.3.0-17ubuntu1~20.04	option	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 03:00:27 CEST 2021	Fedora release 32 (Thirty Two)	x86_64	[5.6.6-300.fc32.x86_64]	0	OK	OK	OK	OK		g++ 8.3.0	option	444/444	1533/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 03:03:37 CEST 2021	Fedora release 32 (Thirty Two)	x86_64	[5.6.6-300.fc32.x86_64]	0	OK	OK	OK	OK		icpc 19.1.3.304	option	0/444	0/1533
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 06:24:57 CEST 2021	Red Hat Enterprise Linux release 8.3 (Ootpa)	x86_64	[4.18.0-147.el8.x86_64]	0	OK	OK	OK	OK		icpc 19.0.5.281	option		
	Sat Jul 17 01:46:13 CEST 2021	Sat Jul 17 16:24:45 CEST 2021	Red Hat Enterprise Linux release 8.1 (Ootpa)	x86_64	[4.18.0-147.el8.x86_64]	0	OK	OK	OK	OK		icpc 19.0.5.281	option		

FIGURE IV.2.2: Page html générée pour l'analyse quotidienne de la base de vérification de TrioCFD

IV.3

Base de Validation de TrioCFD

Si la vérification logicielle garantit que le résultat de chaque phase du processus de développement logiciel réalise effectivement les spécifications initiales (*e.g.* implémentation correcte des équations et modèles spécifiés initialement), la validation logicielle garantit, quant à elle, que le produit logiciel répond aux besoins de toutes les parties prenantes (*e.g.* la bonne résolution des équations définies dans les spécifications et résultats physiques conformes à l'attendu).

La base de validation de TrioCFD

La validation du code est effectuée via les fiches de validation (PRM). Ces PRM peuvent être composées d'un ou plusieurs cas tests de la base de vérification. Lorsqu'une fiche de validation est composée de plusieurs cas tests, c'est parce que ceux-ci comportent des variantes (différents maillages, différents modèles,...) La base de validation est actuellement composée de 162 fiches. Ce nombre est en perpétuelle augmentation puisqu'à chaque nouveau développement, le développeur se doit de fournir une fiche de validation associée. Ces fiches de validation sont disponibles dans le répertoire `/share/Validation/Rapports_automatiques` de chacun des Baltiks et peuvent être ensuite ordonnées dans différents sous-répertoires pour mieux identifier les sujets d'intérêt de chacune. A l'heure actuelle, cette sous-structuration n'est pas optimale et sera améliorée dans les futures versions de TrioCFD.

Le tableau récapitulatif des différentes fiches de validation est donné en Annexe A du document intitulé `models_report_TrioCFD.pdf` présent dans le répertoire `share/doc` de TrioCFD. Il sera amené à être prochainement migré vers le document `validation_report_TrioCFD.pdf` situé dans ce même répertoire.

La validation est actuellement exécutée sur une seule machine (cluster2 du tableau IV.1.1). Actuellement, seule cette machine est en mesure de faire tourner la base de validation : les autres clusters sont des machines partagées avec d'autres codes, le processus de validation nécessite trop de temps de calcul et les stations ne sont pas suffisamment puissantes pour que le processus de validation soit achevé dans des temps raisonnables. Prochainement, un nouveau cluster sera disponible pour la validation avec des caractéristiques de configuration différentes afin d'analyser la portabilité sur ce processus. Le processus de validation présente également une contrainte forte en terme de temps de résolution et doit être en mesure de fournir les résultats dans un délai acceptable. Avant la mise en service du cluster2, mi 2020, celui-ci était lancé sur une station de travail et prenait entre 12 et 13. Cette durée ne permettait pas de garantir une validation suffisamment performante du code.

La base de validation est exécutée sur le cluster via le code Open Source Jenkins, développé en Java, qui permet de bien gérer tous les processus d'un code en intégration continue comme TrioCFD. Jenkins est relativement ergonomique avec une interface graphique simple pour la création de files de tâches mais dispose également d'un mode un peu plus évolué avec la possibilité pour l'utilisateur d'implémenter ses propres scripts en langage Groovy. TrioCFD utilise ces deux fonctionnalités de Jenkins pour le lancement de la base de validation de TrioCFD.

La validation hebdomadaire du logiciel porte actuellement sur les aspects suivants :

- ⇒ la bonne compilation du code
- ⇒ la bonne résolution des fiches de validation
- ⇒ la bonne génération des fiches de validation

Ce processus de validation sera prochainement enrichi afin d'être également en mesure de contrôler la portabilité du code de façon plus poussée que la base de vérification mais s'assure également de la bonne génération de la documentation utilisateur de TrioCFD.

Fréquence de lancement de la base de validation de TrioCFD

Avec l'arrivée du cluster2, la fréquence de lancement de la base de validation a pu être considérablement augmentée. En effet, elle tournait auparavant seulement avant chaque sortie de version à cause de sa durée trop conséquente. La réduction de cette durée permet maintenant de l'exécuter de façon hebdomadaire. Son lancement est actuellement réalisé manuellement par le Responsable de Code le vendredi, après que les dernières intégrations prévues dans la branche `trio/TMA` aient été réalisées. Ce lancement hebdomadaire est réalisé sur la branche en développement de TrioCFD en s'appuyant sur la version de même niveau de TRUST. Il sera prochainement automatisé.

Lorsque des développements pour lesquels des impacts sont prévus doivent être intégrés dans la version, la base de validation peut être amenée à être lancée à tout moment par le Responsable de Code.

Méthodologie de lancement de la base de validation de TrioCFD

La machine maître de lancement des tâches de Jenkins (Pipelines) est actuellement la station 9 du tableau IV.1.1. Un Pipeline spécifique a été créé pour le lancement de la validation, nommé `TrioCFD_Validation` et rattaché au cluster 2. Ce Pipeline est lancé manuellement en remplissant la fiche de paramètres de la figure IV.3.1.

Pipeline TrioCFD_Validation

Ce build nécessite des paramètres :

validation_label	<input type="text"/>	Unique label identifying this validation session.
reference_dir	<input type="text"/>	Where the reference PRMs are stored
	<input type="checkbox"/> resume	Whether to resume a previously interrupted validation session.
	When this is ticked, the build of TRUST and TrioCFD won't be done again, thus avoiding the generation of a newer version of '\$exec' that would, in turn, re-trigger **all** PRM.	
ext_pkg_git_ref	<input type="text" value="refs/heads/triou/TMA"/>	Gti reference used when cloning externalpackages.
trust_git_ref	<input type="text" value="refs/heads/triou/TMA"/>	Gti reference used when cloning TRUST.
trio_git_ref	<input type="text" value="refs/heads/triou/TMA"/>	TrioCFD git reference that should be checked out.
	<input type="checkbox"/> run_test	Whether to launch the tests.
nb_proc_test	<input type="text" value="40"/>	Number of procs on which tests should run.

FIGURE IV.3.1: Méthodologie de lancement du Pipeline de Validation de TrioCFD sous Jenkins

Six principaux paramètres sont à renseigner manuellement pour le lancement de la base de validation qui sont :

- ⇒ **validation_label** : nom du répertoire de validation résultant de ce lancement
- ⇒ **reference_dir** : nom du répertoire d'une version antérieure de validation par rapport à laquelle la version lancée sera comparée
- ⇒ **ext_pkg_git_ref** : branche, SHA1 ID ou tag du dépôt git de externalpackages à partir duquel la validation sera lancée
- ⇒ **trust_git_ref** : branche, SHA1 ID ou tag du dépôt git de TRUST à partir duquel la validation sera lancée
- ⇒ **trio_git_ref** : branche, SHA1 ID ou tag du dépôt git de TrioCFD à partir duquel la validation sera lancée

⇒ **nb_proc_test** : nombre de processeurs de la machine sur laquelle le processus de validation sera lancé (par défaut 40)

Deux cases à cocher sont utilisées pour la réitération du lancement en cas de problème lors du précédent. La première, nommée **resume** permet de relancer le processus de validation à partir de l'endroit exact où celle-ci a été interrompu. La seconde, nommée **run_test**, permet de s'affranchir de toute la première partie qui concerne le clonage des différentes bases git de la compilation de TRUST et TrioCFD.

Ce Pipeline lance le script **TrioCFD_validation.py** présent dans le répertoire **/home/trust_trio/jenkins/scripts/** avec en arguments, les paramètres précédemment renseignés. Ce script Groovy est composé de plusieurs étapes :

1. **check-ref-dir** : vérification de l'existence du répertoire de la version de référence (2ème paramètre de la figure ci-dessus)
2. **git-ext-pkg** : clonage du dépôt GIT externalpackages et positionnement par rapport au paramètre renseigné dans **ext_pkg_git_ref**
3. **git-TRUST** : clonage du dépôt GIT TRUST et positionnement par rapport au paramètre renseigné dans **trust_git_ref**
4. **configure-TRUST** : configuration de TRUST
5. **build-optim-TRUST** : compilation de TRUST en mode optimisé
6. **configure-MEDICoCo** : configuration des outils de couplage (MEDCoupling et ICoCo)
7. **build-optim-MEDICoCo** : compilation des outils de couplage en mode optimisé
8. **git-TrioCFD** : clonage du dépôt GIT TRUST et positionnement par rapport au paramètre renseigné dans **trio_git_ref**
9. **configure-TrioCFD** : construction et configuration de TrioCFD
10. **build-optim-TrioCFD** : compilation de TrioCFD en mode optimisé
11. **prepare_valid** : génération du makefile de la validation et des fichiers CTest (fiches de validation à lancer)
12. **clean_status** : nettoyage des résultats du précédent lancement
13. **ctest_valid** : lancement des fiches de Validation, génération du répertoire des résultats de ce lancement suivant le paramètre renseigné dans **validation_label**, création de la page html de résultat et bilan sur l'état de chaque fiche de validation dès qu'une arrive à son terme
14. **ctest_valid_agg** : finalisation du processus de validation

Le suivi de l'avancée du processus de validation se fait en direct dans Jenkins et permet de savoir, à tout moment, où en est le processus mais également connaître le bon déroulé ou non de chaque étape et le temps pris par chacun (voir figure IV.3.2).

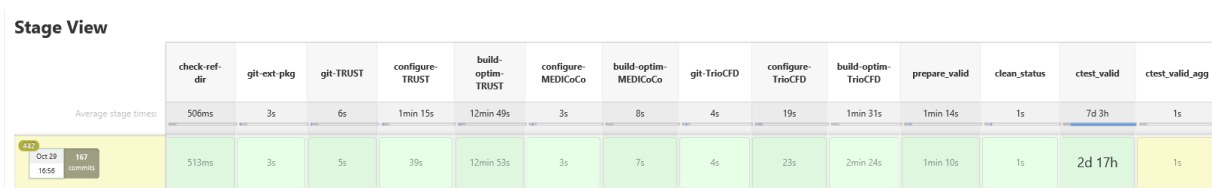


FIGURE IV.3.2: Les différentes étapes du processus de Validation de TrioCFD sous Jenkins

Une fois le processus arrivé dans l'étape **ctest_valid**, l'état de chaque fiche peut être consulté en temps réel dans la page html de suivi qui sera décrite dans la section suivante.

Les résultats de la validation sont stockés dans le répertoire nommé suivant le premier paramètre donné au lancement du Pipeline situé dans **/volatile/projet/trust_trio/jenkins/workspace/TrioCFD_Validation_pegasi2/validation**.

Une fois le processus de validation terminée, le Responsable de code l'archivera dans **/volatile/projet/trust_trio/validation/main_Validation18X/**, répertoire qui comprendra l'ensemble des résultats de validation obtenus entre la version X et X-1 de TrioCFD.

Dépouillement des résultats de la base de validation de TrioCFD

Pour le dépouillement des résultats, le script Groovy `TrioCFD_validation.gy` génère une page HTML consultable depuis l'interface Jenkins, ou pouvant être ouverte par le biais d'un navigateur web lancé depuis la machine où la validation a tourné, à l'adresse : `/volatile/projet/trust_trio/jenkins/workspace/TrioCFD_Validation_pegasi2/validation/validation_label/prm_report.html`. Cette page est composée de cinq rubriques.

Summary

A total of **162** PRM(s) were run:

Item	Count
Failed PRMS	23
Successful PRMS	139
TOTAL (/expected)	162 (/162)

La première nommée **Summary** est un résumé du nombre de fiches ayant tourné avec succès et des fiches en échec. Par échec, on entend à la fois les fiches n'ayant pas abouti et les fiches présentant des écarts avec la version de référence.

FIGURE IV.3.3: Dépouillement résultats de validation - partie 1 : Summary

La deuxième nommée **Running PRMS** liste les fiches de validation en cours de calcul lorsque la page HTML est consultée avant la fin du processus de validation.

Failed PRMS (run failed)

Following PRMS have encountered an issue during their run:

Item	Status	Log
FTD_particles_coupling (2021-09-20 10:54)	Failed running!	/logs/FTD_particles_coupling.log
Verification_CEG (2021-09-20 10:54)	Failed running!	/logs/Verification_CEG.log
FTD_hysteresis (2021-09-20 10:54)	Failed running!	/logs/FTD_hysteresis.log
Channel_T1_T2_QC (2021-09-20 10:54)	Failed running!	/logs/Channel_T1_T2_QC.log
FTD_particles_transfo (2021-09-20 10:54)	Failed running!	/logs/FTD_particles_transfo.log
Turb_coupled_pipeFlow (2021-09-20 10:54)	Failed running!	/logs/Turb_coupled_pipeFlow.log
test_div_grad_Prep1b (2021-09-20 10:54)	Failed running!	/logs/test_div_grad_Prep1b.log
PeriodicBox (2021-09-20 14:02)	Failed running!	/logs/PeriodicBox.log

La rubrique 3, **Failed PRMS (run failed)**, correspond à un tableau des fiches de validation qui ont planté. Le plantage peut être dû à l'échec d'un des jdds de la fin de validation ou à une erreur lors de la génération du pdf de la fiche.

FIGURE IV.3.4: Dépouillement résultats de validation - partie 3 : Failed PRMS

La quatrième nommée **Failed PRMS (comparison)** correspond aux fiches ayant tourné avec succès, mais qui présentent des écarts par rapport à la version de référence. Ces écarts sont détectés automatiquement par une comparaison pixel à pixel du nouveau pdf généré et du pdf de la version de référence. C'est alors au Responsable de Code d'intervenir pour faire une analyse manuelle de ces écarts. Ceux-ci peuvent être sans réelle importance et correspondre seulement à un décalage d'une ligne sur le pdf nouvellement généré par rapport à la fois précédente, ou à une légère variation du temps de calcul. A contrario, cet échec de comparaison peut venir d'un impact physique ou numérique important qui devra être justifié ou corrigé.



FIGURE IV.3.5: Dépouillement résultats de validation - partie 4 : Failed PRMS au niveau de la comparaison des PDF

FIGURE IV.3.6: Dépouillement résultats de validation - partie 4 : Failed PRMS, exemple de comparaison pixel à pixel

Successful PRMS

The PRM that ran successfully:

Chimie_FT (2021-09-20 10:55)	New report	Old report
decroissance keps (2021-09-20 10:55)	New report	Old report
Verification flux implicite (2021-09-20 10:55)	New report	Old report
Test ghost visit (2021-09-20 10:55)	New report	Old report
Nusselt Correlation 2D (2021-09-20 10:55)	New report	Old report
pena_ellipsoide (2021-09-20 10:55)	New report	Old report
Poreux_VEF (2021-09-20 10:55)	New report	Old report
ftd_gravite (2021-09-20 10:55)	New report	Old report
PorousWithPloss_VEF (2021-09-20 10:55)	New report	Old report
Marche k eps T_steady (2021-09-20 10:55)	New report	Old report
RotationALE (2021-09-20 10:55)	New report	Old report
ftd_remaillage (2021-09-20 10:55)	New report	Old report
Uniform keps front field from ud (2021-09-20 10:56)	New report	Old report
Test tparoi (2021-09-20 10:56)	New report	Old report
iodure-iodate (2021-09-20 10:59)	New report	Old report
T_paro (2021-09-20 10:59)	New report	Old report
wl_vof_laminar (2021-09-20 10:59)	New report	Old report
2D_VEF_Cylindre_steady (2021-09-20 10:59)	New report	Old report
pena_couette (2021-09-20 10:59)	New report	Old report
FTD PhaseChange 1D (2021-09-20 11:00)	New report	Old report
iodure-iodate-molasse (2021-09-20 11:00)	New report	Old report

FIGURE IV.3.7: Dépouillement résultats de validation - partie 5 : Successfull PRMS

V. Documentation

V.1

Appui sur la documentation TRUST

TrioCFD étant un Baltik de TRUST, toute la documentation de TRUST est applicable pour TrioCFD. Cette documentation de TRUST est disponible soit via la commande `trust -index` dans un terminal après avoir chargé l'environnement TRUST soit dans le répertoire `doc/TRUST` de TRUST. Cette documentation est composée de :

⇒ La Documentation Générale

- *Release notes* : Elle détaille toutes les intégrations majeures ou les changements ayant un impact sur l'utilisation de TRUST entre deux livraisons de version. Elle est renseignée par le développeur à la fin de son développement/correctif. Pour chacun d'eux, une ligne est ajoutée précisant la date d'intégration dans la version en développement de TRUST, le code sur lequel le travail a eu lieu, la nature de la modification des sources est précisée (New feature, Major Change, Bug fixed,...) et termine par un bref descriptif du travail (figure V.1.1).

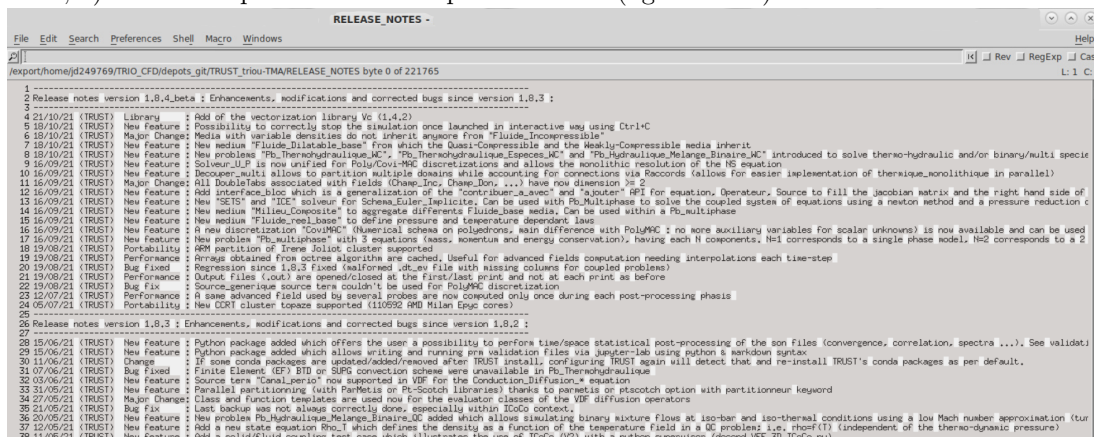


FIGURE V.1.1: Release notes de TRUST

- *Generic Guide* : Ce guide est le premier document à lire lorsque l'on commence à utiliser la plateforme TRUST. Il présente la plateforme TRUST de façon générale, décrit la méthodologie de lancement des cas-tests, la constitution générale d'un jeu de données, la méthodologie de post-traitement et de lancement d'un calcul en mode parallèle.
- *Reference Manual* : Il est généré automatiquement à partir des balises XDATA présentes dans le code et définies par le développeur. Ce manuel explicite tous les mots clés de TRUST et détaille la façon de les mettre en œuvre dans le jeu de données
- *Description note* : Cette note, établie en 2007, présente les différents modèles et équations résolus par Trio_U puisqu'elle a été émise avant la séparation TRUST/TrioCFD et n'a pas été mise à jour depuis. En ce qui concerne les informations de TrioCFD, celles-ci ne sont pas à prendre en compte dans cette documentation puisqu'une version à jour de ces aspects est présente dans la note spécifique à TrioCFD, nommée **Description des modèles** (voir section V.2).

- *Best practices note* : Tout comme le document précédent, celui-ci, émis en 2013, concerne à la fois TRUST et TrioCFD puisqu'il s'agit d'un guide de bonnes pratiques pour la base de validation de Trio_U. La partie concernant TrioCFD de cette note a été mise à jour avec la sortie de la documentation spécifique à TrioCFD à savoir le **Rapport de validation** (voir section V.2).
- *Validation note* : Cette note émise en 2009 décrit l'organisation de la base de données des cas tests de validation de Trio_U ainsi que la méthodologie mise en place pour leur automatisation. Comme la note précédente, les informations relevant de TrioCFD ont été mises à jour dans le **Rapport de validation** (voir section V.2).

⇒ Ressources pour les utilisateurs

- *User tutorial* : ce tutoriel, à destination des utilisateurs de TRUST et TrioCFD, propose de nombreux exemples et exercices afin de prendre en main l'utilisation de la plateforme. Ce tutoriel est largement utilisé dans la formation utilisateur dispensée par l'équipe TMA.
- *Test cases* : tableau récapitulatif des cas tests de référence de TRUST
- *Keywords* : tableau de tous les mots clés de TRUST avec lien vers 1 ou plusieurs cas tests utilisant chacun d'eux.
- *Memo scripts* : petit mémo des principaux scripts devant être connus par les utilisateurs de TRUST dans leurs prémices sur la plateforme.

⇒ Ressources pour les développeurs

- *Developer tutorial* : ce tutoriel, à destination des développeurs de TRUST ou de tout Baltik sous base TRUST, propose de nombreux exemples et exercices afin de prendre en main le codage sur la plateforme. Ce tutoriel est largement utilisé dans la formation développeur dispensée par l'équipe TMA.
- *ICoCo tutorial* : ce tutoriel explique la principe de couplage disponible sur la plateforme TRUST grâce à l'outil ICoCo (Interface for Code Coupling), comment le mettre en œuvre sur TRUST et TrioCFD via des exercices et des descriptions de méthodologie pas à pas.
- *Other baltik tutorial* : il s'agit d'un document expliquant la création d'un Baltik sous base TRUST. Dans le cadre de TrioCFD, elle est également très utile pour créer des Baltiks ou des sous-Baltiks de TrioCFD, la méthodologie étant la même.
- *C++ classes* : documentation DOxygen des classes de TRUST afin d'avoir une vision générale des classes de TRUST, des méthodes rattachées à celles-ci et une description des variables. Elle est générée automatiquement avec la commande `trust -gui` à partir des balises DOxygen présentes dans le code. Pour les parties de code non balisées, des scripts spécifiques ont été conçus afin qu'elles soient néanmoins prises en compte dans cette documentation indispensable aux développeurs.
- *Test coverage* : tableau récapitulatif du nombre de lignes testées dans la partie `source` de TRUST mais également sur l'ensemble du projet TRUST.
- *.prm syntax* : page html expliquant les différents éléments de syntaxe pour la création d'une fiche PRM (fiche de Validation). Si la majorité des mots clés sont applicables à TrioCFD, une nouvelle structure a été mise en place en 2020 dans TrioCFD et les nouveaux mots clés sont décrits dans le rapport de validation de TrioCFD qui sera présenté dans le chapitre suivant.

V.2

Documentation spécifique à TrioCFD

Outre cette documentation de TRUST, TrioCFD dispose de sa propre documentation en complément. Une grande partie de celle-ci a été mise en place récemment (à partir de 2019) afin de fournir d'avantage de

supports à la communauté TrioCFD mais également d'accroître la qualité du code. Jusqu'alors les ressources humaines n'étaient pas suffisantes pour mener de front l'enrichissement des modèles et la mise en qualité. L'ensemble de ces documents sont gérés en gestion de configuration (sous GIT). Ils sont régulièrement enrichis et mis à jour au fur et à mesure des évolutions du code. En effet, leur mise à jour est une étape obligatoire dans tous les processus d'actions menées sur TrioCFD et ils sont relus, au même titre que le code à proprement parler, avant d'être intégrés dans la version. Mise à part la Release Notes qui est un simple document texte, ils sont tous en latex et disposent de leur propre script de génération. Les sources latex de ces documents sont accessibles dans le répertoire `/share/doc_src` et disposent d'un répertoire spécifique chacun. Les fichiers PDF correspondants sont, quant à eux, disponibles dans le répertoire `/share/doc`. Tout comme TRUST, TrioCFD dispose d'une page Html facilitant l'accès à ces différents documents, qui peut être ouverte par la commande `triocfd -index` après avoir chargé l'environnement TrioCFD.

Release Notes

TrioCFD dispose de sa propre Release notes qui présente la même structure que celle de TRUST : date d'intégration dans le code - code concerné - nature du travail - bref descriptif. Elle présente l'ensemble des modifications/évolutions majeures faites dans le code depuis la version v1.4.0 (mi-2002), version dans laquelle, elle a été mise en place pour la première fois. Elle est organisée par livraison afin que l'utilisateur soit bien informé des changements d'une version à l'autre.

Elle est également mise à jour par le développeur lorsque ses travaux font apparaître de nouvelles fonctionnalités dans le code ou modifient le fonctionnement ou les résultats des fonctionnalités existantes. Elle est relue au moment de l'intégration de la branche d'un développement dans le code. Il est recommandé à l'utilisateur de la consulter avant l'utilisation de toute nouvelle version. Des renseignements supplémentaires sur chacun des points peuvent être obtenus auprès de l'équipe de TMA ou du Responsable de Code.

TrioCFD Reference Manuel

Tout comme le Manuel de Reference de TRUST, celui de TrioCFD est généré automatiquement à partir des balises XDATA présentes dans le code et définies par les développeurs pour chaque implémentation de nouveau mot clé. En effet, ce manuel répertorie l'ensemble des mots-clés de TrioCFD à disposition des utilisateurs pour la construction des jeux de données. Il précise la syntaxe exacte à employer pour chacun et décrit brièvement leur utilité.

Si ce document n'est pas extrêmement ergonomique et convivial, il présente néanmoins l'intérêt d'avoir les informations sur les mots-clés directement dans le code, ce qui facilite son évolution et sa maintenance. Des travaux sont actuellement en cours afin de mettre en place une documentation des mots-clé plus accessible et riche.

Plan de développement TrioCFD 2020-2025

Sorti en 2019 sous la forme d'une note technique CEA, il a été mis en gestion de configuration début 2021 afin de le mettre à disposition des utilisateurs mais également pour en assurer la sauvegarde.

Comme son nom l'indique, il répertorie les développements et les actions de Recherche et Développement prévus dans le code pour la période 2020-2025. Il décrit le programme technique à réaliser et les ressources associées. Il ne concerne que le code de CFD monophasique aux échelles RANS et LES. Les travaux envisagés sur le module de CFD diphasique moyenné, les baltiks FT et IJK font l'objet de documents séparés. Aucune date n'est spécialement définie en face de chaque action. La démarche consiste plutôt à choisir, chaque année, en fonction des sollicitations des utilisateurs et des ressources disponibles, certaines de ces actions et à les traiter. Des développements non définis dans ce document peuvent également être menés afin de répondre aux besoins exprimés par des projets ou des industriels après la rédaction de ce document. Il représente néanmoins la ligne directrice de l'évolution que l'équipe de développeurs CEA souhaite donner au code. D'autre part, il est également possible que certaines actions identifiées à l'époque perdent de leur pertinence au cours du temps et ne soient finalement pas traitées. Finalement, il est à prévoir que certaines actions de R&D, qui sont longues à réaliser par nature, seront initiées durant cette période et s'achèveront au-delà de ce terme.

Description des modèles

La première version de cette description des modèles est également une note CEA émise en 2019 qui a été mise en gestion de configuration début 2021, en même temps que le document précédent. Cette note décrit les principaux modèles physiques du code de TrioCFD.

Dans ce document, la présentation des modèles se concentre sur la description des écoulements monophasiques, incompressibles, Newtoniens et turbulents. Sa deuxième section rappelle les équations de conservation de masse, quantité de mouvement et d'énergie sont rappelées. Sont également rappelées les définitions des tenseurs de contraintes, de déformation et de rotation qui reviendront dans tout le document. La section se termine par une description des matrices de masse et de rigidité lorsque le problème de Stokes est discrétisé par la méthode des Volumes-Éléments Finis (VEF). La section suivante présente les différentes approches qui permettent de simuler la turbulence en LES (Large Eddy Simulations) et en RANS (Reynolds Averaged Navier-Stokes). Pour les modèles RANS, plusieurs modèles de type $\kappa - \epsilon$ sont présentés : le $\kappa - \epsilon$ standard, le $\kappa - \epsilon$ réalisable, et le $\kappa - \epsilon$ bas Reynolds. Depuis la première version de cette note, le document a été enrichi avec les modèles présents dans les Baltiks Phase_Field, ALE et Sensitivity_Analysis. Ce document ne fait, à l'heure actuelle, pas état de l'ensemble de modèles présents dans TrioCFD mais est régulièrement enrichi avec les modèles déjà existants dans TrioCFD et les nouveaux modèles régulièrement développés.

Rapport de validation

Le rapport de validation a été mis en place fin 2020 et rendu disponible pour la première fois à la livraison de la version v1.8.2. Dès sa création, il a immédiatement été mis en gestion de configuration car il s'agit d'un document évolutif qui fournit aux utilisateurs des exemples d'utilisation de TrioCFD sur les domaines physiques couverts par le code.

Il commence par décrire la nouvelle syntaxe des fiches de validation qui a été justement mise en place pour la création de ce document. En effet, jusqu'alors chaque fiche de validation avait sa structure propre définie par son auteur. L'idée a été de mettre en place une structure définie et identique pour toutes les fiches de validation afin de faciliter la lecture et la compréhension de ce rapport. Dans la première version de ce rapport, les fiches de validation couvraient les domaines suivant :

- les écoulements laminaires
- les écoulements laminaires avec échanges thermiques
- les écoulements turbulents
- les écoulements turbulents avec échanges thermiques
- les écoulements diphasiques avec le Baltik de Front-Tracking

Chaque domaine contient 3 fiches de validation dont les résultats numériques obtenus avec TrioCFD sont comparés soit à l'analytique, soit à des résultats expérimentaux, soit aux résultats numériques obtenus avec d'autres codes. L'entrée d'une fiche de validation dans le rapport de validation nécessite la réécriture de la fiche afin de la mettre au nouveau format. Certaines fiches de validation ne sont également pas suffisamment consistantes pour avoir leur place au sein de ce rapport. Ceci explique pourquoi l'intégralité des fiches de TrioCFD ne figure pas dans ce rapport de validation. En 2021, trois fiches de validation sur le Baltik ALE, Baltik permettant la modélisation des interactions fluide-structure, y ont été intégrées. Ce rapport sera progressivement enrichi avec de nouvelles fiches dans les domaines existants mais également avec de nouveaux domaines non abordés jusqu'ici.

Plan de Gestion de Configuration

Le PGC ou Plan de Gestion de Configuration de TrioCFD (ce présent document) est le dernier à rejoindre les nouveaux documents mis à disposition des utilisateurs de TrioCFD. Il est disponible, pour la première fois fin 2021 à l'occasion de la sortie de la version v1.8.4. Il a également été immédiatement placé en gestion de configuration, car il s'agit aussi d'un document évolutif qui sera enrichi et mis à jour pour chaque évolution ou modification des pratiques ou des outils du code.

VI. Méthodologies et Procédures

A chaque type d'action, un processus spécifique est défini afin que celle-ci soit traitée au mieux. On dénombre 4 types différents d'action : développement, étude, demande de maintenance (ayant elle-même plusieurs catégories) et livraison.

Chacune a sa propre méthodologie de traitement qui est spécifiée dans cette partie. Le responsable de l'action doit suivre cette méthodologie afin de garantir sa qualité de réalisation et éviter d'oublier certaines étapes cruciales.

VI.1

Processus de développement

Un développement consiste à introduire un nouveau modèle, une nouvelle fonctionnalité ou nouvel outil dans le code. Le respect d'une méthode de développement (ou bonnes pratiques) a plusieurs avantages :

- ⇒ Elle permet de réduire considérablement la dette technique du code. Les coûts ultérieurs, tant pour la maintenance corrective (correction de bugs informatiques) que pour la maintenance évolutive (mise à jour ou ajout de fonctionnalités) peuvent être plus ou moins importants selon la mise en oeuvre de ces bonnes pratiques. Ces coûts additionnels représentent la dette technique d'un logiciel. Une méthodologie de développement permet ainsi de maîtriser les temps de développement mais aussi de réduire la fréquence des bugs.
- ⇒ Elle permet d'accroître la sécurité du logiciel.
- ⇒ Elle permet d'obtenir un code ayant en permanence une bonne qualité jusqu'au moment où une opération de valorisation est envisagée (préparation des audits).
- ⇒ L'application de bonnes pratiques rend propoice le travail en collaboration en établissant un "code de conduite" commun à tous les acteurs.

La méthodologie de développement appliquée à TrioCFD va donc être explicitée dans ce chapitre.

Étape 1 : Spécifications

Cette étape d'expression du besoin doit être faite en tout premier lieu par le développeur en charge de l'action. Elle doit couvrir l'intégralité des cas d'utilisation du code, en expliquant ce qui doit être fait et non pas comment le faire. Chaque spécification doit pouvoir être testée lors de la dernière phase du développement. Pendant cette phase de spécification, le développeur doit :

- ⇒ identifier les besoins puis les traduire en termes de fonctionnalités, d'interfaces avec les autres Baltiks de TrioCFD, avec TRUST, avec les autres Baltiks de TRUST mais également entre elles ;
- ⇒ préciser les enchaînements des actions ;
- ⇒ préciser les contraintes liées au développement (performances, priorités,...) ;
- ⇒ analyser, en fonction des besoins à couvrir, les classes ou Baltiks qui pourraient être réutilisés et évaluer les impacts de leur réutilisation sur le développement.

Ces spécifications fonctionnelles doivent s'affranchir de la façon dont est implémentée le code. Il est important de les établir pour ne pas se lancer aveuglément dans le codage.

Lors de cette étape, il est pertinent de faire une analyse bibliographique des différents modèles à disposition pour arriver au besoin ciblé. Par cette analyse bibliographique, les avantages et inconvénients des différents modèles pouvant répondre au besoin seront identifiés, de même que leur adéquation avec la démarche scientifique du code. Ainsi, le plus pertinent sera plus rapidement identifié.

TrioCFD étant un code Open Source, il sera important de s'assurer, à cette étape, que les choix faits rentrent bien dans cette démarche.

En fin de cette étape de spécification, une fiche de Demande d'Intervention de type **Maintenance Évolutive** devra être créée dans le BugTracker et le bilan de cette étape devra y être reporté.

Étape 2 : Conception

Cette étape, également à la charge du développeur, permet décrire comment le code est censé fonctionner au terme du développement, avant de rentrer dans la phase de codage. Il s'agit de la réponse technique aux spécialités fonctionnelles faites précédemment.

L'activité de conception consiste à :

- ⇒ définir le découpage structurel du développement de chacune des fonctionnalités identifiées précédemment en classes, méthodes et templates puis de détailler chacun d'eux. Ce découpage doit respecter les règles de la plateforme TRUST/TrioCFD disponibles dans le **Developer tutorial** ;
- ⇒ identifier si le développement doit être rattaché à un Baltik existant ou si il est nécessaire de créer un nouveau Baltik ou sous-Baltik spécifique ;
- ⇒ identifier la nécessité ou non de modifier ou surcharger des classes de TRUST. **Attention la surcharge de classes de TrioCFD est interdite.** Si des classes de TrioCFD ont été identifiées comme pouvant être réutilisées dans le cadre du nouveau développement, moyennant certaines adaptations, des travaux préparatoires amont devront être menés afin de rendre générique cette classe existante puis de créer deux classes filles : une pour l'application déjà existante et une nouvelle pour la nouvelle application, dérivant de la classe mère ;
- ⇒ effectuer l'estimation du temps et des ressources nécessaires pour mener à bien les actions de réalisation et de validation afin de prévoir son intégration et la validation générale du code associée ;
- ⇒ commencer à identifier l'activation de ce nouveau modèle ou application dans le jeu de données ainsi que la création des variables de post-traitement nécessaires pour mettre en évidence son intérêt et sa validité.

Cette partie architecture (algorithmique) doit être conçue en amont de l'écriture du code. Elle permet de structurer le développement, de séparer ses différentes fonctions et d'obtenir un codage modulaire et optimisé. Sans codage modulaire, le développement serait sous forme d'un seul fichier de plusieurs centaines de lignes de code, le rendant difficile à maintenir et peu évolutif.

Étape 3 : Réalisation

C'est pendant la phase de réalisation à proprement parler que sont réalisés les développements en vue de répondre aux besoins exprimés lors de la phase de spécification, en suivant l'architecture définie lors de la phase de conception. Il est demandé de respecter la découpe architecturale du code en scindant en plusieurs classes ou fonctions le développement afin d'en améliorer sa maintenance et sa mutualisation avec d'autres applications. TrioCFD dispose d'une architecture modulaire avec ses différents Baltiks. Lors de la phase de codage, il est demandé de poursuivre cette démarche de modularité.

Pour rappel, TrioCFD est codé en langage C++ avec une surcharge des classes standard C++ via TRUST. Il est obligatoire de respecter ce langage. Aucun source dans un autre langage ne sera intégré dans le code. En ce qui concerne les procédures, celles-ci sont en Python (3) ou en Bash. Il n'y a pas d'outil de codage imposé pour mener à bien le développement. Ainsi, le développeur peut choisir, à sa convenance d'utiliser soit un éditeur de code classique soit utiliser un environnement de développement intégré (IDE) type Eclipse ou autre. Le code doit être explicitement commenté et les messages d'entrées/sorties, clairs et pertinents. Chaque Baltik ou sous-Baltik comporte la même architecture générale composée, à sa racine, d'un fichier `project.cfg` qui permet de caractériser le Baltik (ou sous-Baltik) avec son nom, son auteur, la dénomination

de son exécutable ainsi que sa ou ses dépendance(s) potentielle(s) avec d'autres Baltiks (ou sous-Baltiks). Au niveau de sa racine, on trouve également 3 répertoires :

- le répertoire **share** : il contient les documents propres au Baltik dans le sous répertoire **doc_src** et les fiches de validation propres au Baltik dans le sous répertoire **Validation**. Celles-ci seront lancées lors du processus de Validation (voir Chapitre IV.3).
- le répertoire **src** : il regroupe l'ensemble des fichiers source du Baltik (.cpp et .h). Si des classes de TRUST sont menées à être surchargées pour le Baltik, celles-ci seront placées dans un sous-répertoire nommé **Modif_TRUST** dans **src**.
- le répertoire **tests** : celui-ci contient l'ensemble des cas-tests de vérification du Baltik concerné et seront lancés lors du processus de Vérification (voir Chapitre IV.2).

Tout développement devra respecter cet agencement. Les étapes de codage que devra respecter le développeur sont les suivantes :

1. mise à jour du dépôt GIT local de TRUST et TrioCFD dans lequel le développement va être fait ;
2. création de la branche GIT spécifique au développement nommée **TCFDXXXXXX_Activité** à partir de la branche de développement de TrioCFD nommée **trio/TMA**. Le numéro XXXXXX correspond au numéro d'identification de la fiche relative au développement créée dans le BugTracker TrioCFD (voir chapitre III.1). Il s'agit d'un identifiant unique permettant de faire immédiatement le lien entre chaque modification apportée dans le code et sa fiche descriptive associée. Le mot **Activité** va donner un aperçu en 1 mot du sujet sur lequel porte le développement (*eg* Turbulence_keps, ALE_vibration,...) ;
3. codage à proprement parler d'une des fonctionnalités du développement général en respectant les quelques recommandations données ci-dessus ;
4. vérification de la bonne compilation du Baltik et de TrioCFD en mode optimisé et debug ;
5. définition et implémentation d'un cas-test permettant de vérifier la fonctionnalité implémentée avec vérification de son bon fonctionnement ;
6. commit dans la branche du développement de cette première fonctionnalité et push de la branche sur le dépôt GIT distant ;
7. répétition des étapes 4, 5, 6 et 7 autant de fois que nécessaire suivant le nombre de fonctionnalités ajoutées dans le cadre de ce développement. Si pertinent, le cas-test pourra être mutualisé pour les différentes fonctionnalités ;
8. création d'une fiche de Validation (PRM) testant l'ensemble du développement pouvant reprendre tout ou partie des cas-tests de vérification établis au fur et à mesure ;
9. lancement de la base de vérification complète de TrioCFD, analyse des résultats obtenus, corrections ou justification des écarts si nécessaire ;
10. documentation du développement qui sera ajouté dans un des documents décrits dans la partie V ;
11. vérification de la bonne compilation du Baltik et de TrioCFD en mode optimisé et debug ;
12. commit des travaux effectués aux étapes 8, 9 et 10 et publication de la branche de développement sur le dépôt distant ;
13. demande de Pull Request via Tuleap (voir méthodologie expliquée ici III.1.3)

Suite à toutes ces actions, l'étape de codage est terminée et le développement vérifié. Le développeur doit maintenant mettre à jour la fiche Tuleap de son développement en expliquant les différents points de commit de celui-ci. Il y explicite également la possibilité d'impacts physiques ou numériques dans la base de Validation. En fonction des résultats obtenus lors de la Validation (étape suivante), il est possible que d'autres interventions dans le code soient nécessaires.

Étape 4 : Validation

L'étape suivante consiste à valider ce développement. Cette étape est à la main du Responsable de Code qui, voyant l'apparition de la demande de Pull Request dans Tuleap va faire une première relecture du développement et consulter la fiche de BugTracker associée. En fonction de l'analyse qu'il a mené sur le développement et du message du développeur sur le potentiel impact de cette branche sur la base de Validation, il lancera ou pas une validation complète.

En cas de lancement de la base de Validation à ce moment-là, le Responsable de Code la lancera à partir du dernier point de commit de la branche du développement (voir Chapitre IV.3) sur la station dédiée à la validation (Pegasi 2) en prenant comme version de référence la dernière version considérée comme validée. Cette étape dure environ 60h, durant lesquelles, toutes les fiches de validation sont lancées et une comparaison pixel à pixel est faite pour chacune. Les fiches ressortant en écart (les procédures ont détecté des différences de pixel entre la version de référence et la version à valider) ou en échec (le rapport PDF de la fiche de validation n'a pas été généré ou un ou plusieurs cas-tests de la fiche n'a pas tourné) sont analysées par le Responsable de Code. Dans de nombreux cas, les écarts observés sur les fiches sont en réalité des écarts anodins. Ces écarts correspondent en réalité à de légères translation de texte ou de figure, à des arrondis différents au delà de la 5^{ème} décimale, à des figures tracées à quelques centièmes de secondes d'écart par rapport à la version de référence ou à de légers écarts sur les résultats obtenus ($< 2\%$). Pour ce type d'écarts, le Responsable de Code considère que les résultats de Validation sont corrects.

En revanche, lorsque les écarts physiques ou numériques sont significatifs ($> 2\%$), le développeur est sollicité afin d'apporter un avis d'expert sur leur validité ou non. En cas de non validité, le développement doit être repris pour apporter les corrections adéquates. Les fiches de validation problématiques ayant été détectées lors du lancement de la base de Validation, celui-ci s'attachera à les relancer en local, une fois les corrections/adaptations effectuées et s'assurera de leur non-régression.

Pour le cas des fiches en échec, il sera également de la responsabilité du développeur d'apporter les correctifs nécessaires (dans le code en lui-même ou sur la PRM) afin de la rendre de nouveau fonctionnelle. Ce travail sera mené sur la station personnelle du développeur.

Une fois les écarts et échecs résorbés, le développeur effectue un dernier commit sur sa branche et la pousse sur le dépôt distant. La base de Validation complète n'est pas relancée à cette étape.

Étape 5 : Relecture et Intégration

La dernière étape du processus de développement concerne, dans un premier temps, la relecture de la branche du développement. Cette étape est extrêmement importante car elle permet d'avoir un code plus lisible, plus homogène et globalement de meilleure qualité. Sachant son code relu, l'auteur s'oblige à relire lui-même son propre code avant de faire sa Pull Request. Il va donc ajouter spontanément des commentaires et faire en sorte que son code soit le plus compréhensible possible pour que la revue de code se passe bien. Par ailleurs, un second regard permet de limiter la duplication de code grâce à une deuxième vision et le relecteur peut alors proposer de réutiliser des parties de code déjà existantes. Même si le relecteur ne teste pas forcément le code, son point de vue extérieur peut identifier certains bugs en lisant le code (tel que des cas limites que le développeur aurait pu oublier) ou des problèmes d'implémentation mais également constater une réponse partielle aux spécifications initiales. Il peut également suggérer l'ajout de tests unitaires pour des cas qui n'auraient pas été prévus initialement ou l'enrichissement de la documentation si certains aspects restent flous pour le relecteur. Lors de la revue de code, le relecteur s'assure :

- de la bonne dénomination de la branche ;
- de la pertinence des points de commit et du commentaire associé ;
- du bon respect des règles de codage (non duplication de code, dénomination des classes, fonctions et variables, optimisation du code,...) ;
- de l'adéquation entre le développement produit et les besoins et spécifications initialement émis ;
- de la présence et de la clarté de la documentation ;
- de la présence et de la pertinence des cas-tests et de la fiche de validation.

En cas d'un soucis sur un de ces aspects, le relecteur et le développeur travailleront ensemble sur les points controversés afin de converger sur la meilleure solution pour le code. Les échanges sur la revue de code se font via Tuleap au niveau de la Pull Request. La figure VI.1.1 est un exemple de relecture d'une branche.

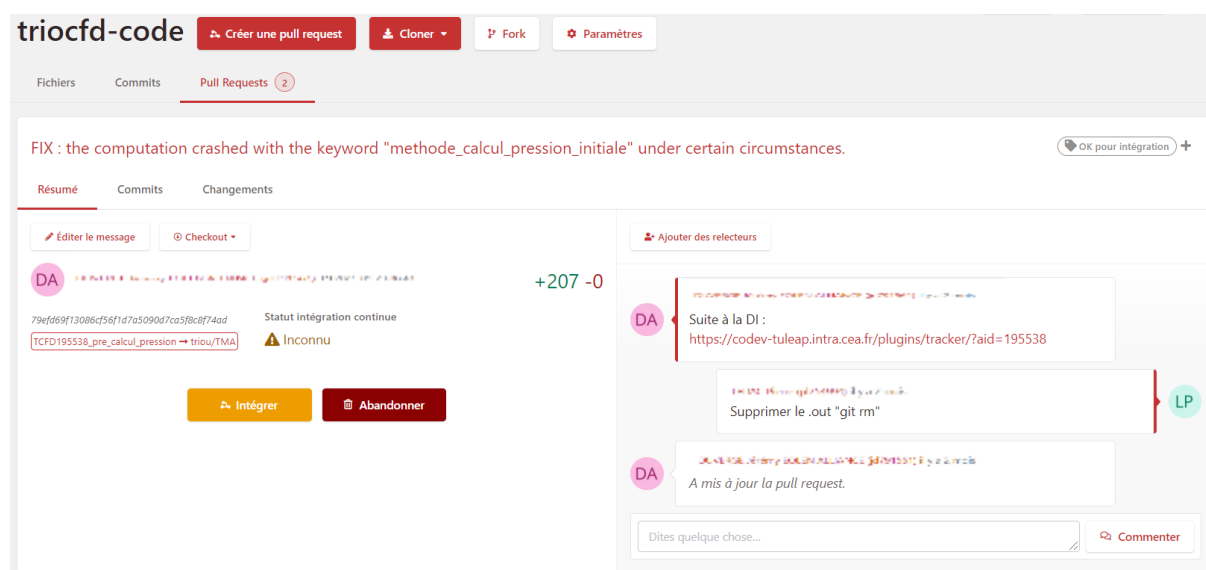


FIGURE VI.1.1: Exemple d'échange lors de la relecture d'une branche

Suite à la relecture, un commit supplémentaire pourra amené à être fait sur la branche du développeur et la Pull Request, mise à jour.

Une fois la revue de code approuvée, le relecteur placera un tag **OK pour intégration** (voir figure VI.1.1 en haut à droite) signifiant que la branche du développeur a passé avec succès cette étape et peut alors être intégrée dans le code.

L'intégrateur intervient alors en rapatriant la branche du développeur dans la branche de développement de TrioCFD (branche triou/TMA) à date. La branche de développement ayant pu recevoir d'autres intégrations depuis la création de la branche du développeur, il vérifie la bonne compilation du code sur la branche de développement avec les nouvelles fonctionnalités et lance la base de vérification. Si une de ces actions ne donnaient pas le résultat escompté, l'intégrateur corrigera la branche de développement avec, potentiellement, l'aide du développeur. Une fois ces 2 contrôles (compilation + vérification) atteints avec succès, la branche du développeur est committée dans la branche triou/TMA et celle-ci est poussée sur le dépôt distant. Les tests de vérification quotidiens tourneront donc, le soir même, sur cette nouvelle version du code et le statut de la fiche Tuleap associée passera alors en état **MERGED** (voir figure III.1.6). La base de Validation sera lancée sur cette nouvelle version du code le week-end suivant et lorsque la non-régression sera atteinte, la fiche Tuleap associée sera fermée (statut **CLOSED**). L'ensemble du processus de développement sera alors achevé et l'action de développement sera considérée comme terminée.

VI.2

Processus de Livraison

La livraison est effectuée conjointement par l'équipe TMA et l'équipe CEA. Elle commence par la définition exacte de la date de sortie de version. 15 jours avant cette date, l'ensemble des branches (développements et correctifs) doivent avoir été intégrés dans la branche en développement (**triu/TMA**). D'autres intégrations mineures pourront éventuellement avoir lieu ensuite, notamment en cas d'échec de l'étape de validation de la non-régression. Les étapes importantes qui rythment la livraison sont décrites dans les sections suivantes.

Étape 1 : Vérification et Validation de la non-régression

Une fois l'ensemble des correctifs et développements intégrés dans la version, la base de vérification tourne une dernière fois sur l'ensemble des machines du parc afin de s'assurer qu'il n'y ait aucune différence de comportement entre les différentes configurations couvertes. L'atelier de génie logiciel est alors désactivé jusqu'à la livraison de la version.

La base de validation est lancée une première fois et les non régressions vis à vis de la dernière exécution de la base de validation sont analysées afin de maîtriser les éventuels impacts des dernières intégrations (voir méthodologie décrite au chapitre IV.3). Elle est également lancée en prenant comme version de référence, la dernière version livrée du code. Les écarts obtenus correspondent à la concaténation de l'ensemble des différences identifiées lors du lancement hebdomadaire du processus de validation. Des jdds qui n'ont pas subi d'écarts lors des différentes validations intermédiaires ne doivent donc pas apparaître à ce moment-là. Une fois ces 2 exécutions complètement analysées et justifiées, la nouvelle version du code est considérée comme validée. Les sources du code et les jdds ne devront plus être touchés jusqu'à ce que la livraison soit achevée.

Étape 2 : Finalisation de la documentation et dernières intégrations

L'étape suivante consiste, dans un premier temps, à mettre à jour toute la documentation afin notamment de faire évoluer le numéro de version, mais également de faire apparaître l'ensemble des évolutions qui ont eu lieu dans le code depuis la dernière livraison. Ceci concerne les documents cités au chapitre V.2. Une fois l'ensemble de la documentation mise à jour, celle-ci est intégrée dans la branche `triou/TMA` et constituera le dernier point de commit. La branche de développement de TrioCFD est alors mergée avec la branche `master` et un tag est apposé sur chacune de ces 2 branches avec le nom de la version livrée.

Étape 3 : Génération de l'archive et livraison via SourceForge/GitHub

Vient ensuite l'étape de la mise à disposition aux utilisateurs et développeurs extérieurs au CEA. La nouvelle version leur est transmise via GITHUB depuis 2021 à l'adresse <https://github.com/cea-trust-platform/TrioCFD-code>. Auparavant, la plateforme utilisée était Sourceforge. La version est disponible sous GITHUB via 2 formats :

- format archive : regroupe dans une archive l'ensemble du code source, des procédures ainsi qu'un pannel représentatif des fiches de validation du code. Cette archive doit être téléchargée et décompressée sur la machine où l'installation doit avoir lieu.
- format base git : contient l'ensemble du code, des jdds et des procédures de TrioCFD identique au dépôt GIT de TrioCFD hébergé sur Tuleap. La dernière version livrée correspondra à la branche `master` du dépôt GITHUB.

Pour les utilisateurs, il est recommandé de récupérer la version livrée sous le format archive, tandis que pour les développeurs, le dépôt GITHUB est à privilégier.

Étape 4 : Mise à disposition sur les Clusters

Finalement, la nouvelle version sera livrée aux utilisateurs CEA avec son installation sur l'ensemble des clusters répertoriés (5). Ils pourront également utiliser la version réseau de cette nouvelle version disponible sous `/home/triou`.

A l'issue de toutes ces étapes, un mail sera envoyé à la communauté CEA de TrioCFD, décrivant les évolutions du code depuis la dernière version livrée et donnant les différents liens pour l'utiliser. Pour la communauté extérieure au CEA, l'information sera communiquée par le site de TrioCFD (voir chapitre VII.1).

A l'issue de cette démarche, les intégrations reprendront dans le code en vue de l'enrichir pour la version suivante.

VI.3

Demande de maintenance

Les actions de la TMA sur TrioCFD se font dans le cadre d'un cahier des charges relatif à la prestation de maintenance informatique des codes de calcul en thermohydraulique. Le contrat en cours est régi par le cahier des charges référencé : DES/ISAS/DM2S/STMF/DIR/ST/2021-66801/A. Celui-ci a débuté le 01/04/2021 pour une durée de deux ans avec la possibilité de le renouveler 3 fois pour une durée d'un an à chaque fois. Ce contrat est commun à différents codes du CEA.

Dans le cadre de ce contrat, diverses actions sur le code sont gérées par la TMA avec un suivi régulier de la part du CEA par le RLP et le Responsable de Code. Outre les échanges quotidiens, plusieurs réunions contractuelles doivent avoir lieu pour assurer le suivi :

- ⇒ les **réunions de suivi hebdomadaires** qui permettent de faire un suivi rapproché des actions en cours, revoir les priorisations sur le code concerné en cas de besoins urgents des utilisateurs, ou échanger sur les difficultés rencontrées lors de la résolution des demandes de maintenance ;
- ⇒ les **COSUIV**, se déroulant de façon mensuel, permettent de faire le même suivi que les réunions hebdomadaires mais la priorisation est faite en regard des besoins de l'ensemble des codes à la charge de la TMA. Ainsi, si une échéance importante est prévue sur un code, les actions le concernant seront mises en priorité haute par rapport à celle des autres codes. Il se tient en présence du RPL, de l'ensemble des Responsables de Code et de l'ensemble de l'équipe TMA et donne lieu à compte-rendu de réunion rédigé par le RML.
- ⇒ les **COPIL** ont lieu de façon bi-mensuelle et réunit les chargés d'affaire (CEA et TMA), les RPL, les Responsables de Code, les RML (et éventuellement les Chefs de Laboratoires portant les codes et Chef de Service). Les COPIL permettent de s'assurer que les attendus contractuels sont bien respectés de part et d'autre. Un compte-rendu de réunion trace l'ensemble des constats fait lors du COPIL.

Les différentes actions à la charge de la TMA ainsi que leur méthodologie de traitement sont décrites dans ce chapitre.

Prise en compte et analyse des demandes

Toute question arrivant dans la boîte mail trust@cea.fr est prise en charge par la TMA avec un délai de traitement d'un jour ouvré. Après une première analyse, une fiche de Demande d'Intervention est créée par la TMA pour chaque problème rencontré. Cette étape préliminaire d'analyse permet de lui affecter de catégorie parmi celles détaillées ci-après. Lors du traitement de la Demande d'Intervention, si il s'avère que la catégorie n'a pas été correctement identifiée, elle pourra être mise à jour lors de son traitement.

La saisie sous forme de fiche de chacun des mails reçus permet de garantir un bon suivi des difficultés rencontrés et de garantir une réponse à l'ensemble des sollicitations reçues.

Traitement des Assistances Aux Utilisateurs (AAU)

L'équipe de TMA est souvent sollicitée par les utilisateurs car un de leurs cas tests n'aboutit pas favorablement. La Demande d'Intervention est alors classée comme Assistance Aux Utilisateurs. A ce stade, rien ne garantit encore que l'erreur vienne effectivement d'un problème au niveau de la construction du jdd. Les travaux commenceront par la modification de plusieurs paramètres, bien connus pour jouer sur la stabilité des calculs comme le pas de temps, le facsec ou le solveur utilisé. Si ces premiers tests s'avèrent concluant et permettent d'arriver à une fin propre du calcul, cela voudra dire que le jdd envoyé nécessitait

juste de jouer sur ces paramètres numériques.

En cas contraire, si le jdd fourni par l'Initiateur est complexe, on essaiera de simplifier progressivement le jdd afin de mieux cerner l'élément problématique. Ce procédé de simplification couplé à l'analyse des messages obtenus dans le fichier de sortie (fichier .err) permettront de simplifier les travaux. Si le problème relève bien du jdd, l'équipe de TMA sera en mesure d'identifier rapidement le problème de modélisation que l'utilisateur aurait pu introduire dans ce jdd. La TMA proposera alors une version fonctionnelle du jdd à l'utilisateur en lui expliquant les modifications effectuées et la raison. Si l'utilisateur est satisfait de ces modifications, la DI passe en statut **Treated/Resolved** et les travaux sont considérés comme achevés.

Parfois, il s'avère que la TMA ne parvienne pas à résoudre le problème immédiatement car celui-ci ne provient pas du jdd en lui-même, mais relève d'un bug au niveau du code. La DI sera alors requalifiée dans la catégorie Maintenance Corrective.

Le contrat de TMA actuel prévoit 25 demandes Assistance Aux Utilisateurs par an sur TrioCFD.

Traitement des Maintenances Correctives (MC)

Pour rappel, une maintenance corrective est la correction d'anomalies dans les sources du code, dans les outils de maintenance, dans la documentation et dans les jeux de données de la base de test.

Une fois la fiche de Demande d'Intervention créée et identifiée dans cette catégorie, les travaux sont initiés sur le jdd fourni par l'Initiateur pour reproduire le problème constaté. Si le jdd fourni est complexe, celui-ci sera simplifié afin de mieux cerner l'élément source de bug dans le jdd. Une fois le problème reproduit sur un cas test élémentaire, les travaux de débogage débiteront jusqu'à la bonne résolution du jdd fourni. Une fois cette étape atteinte en local, une branche GIT sera créée avec la même dénomination que décrite précédemment (voir la section III.3), c'est-à-dire avec le numéro de la DI correspondante sur le BugTracker, contenant l'ensemble des sources modifiées ainsi que le cas test ayant permis sa détection. Le Workflow présenté précédemment sera suivi, jusqu'à la clôture de la DI.

Le contrat de TMA actuel prévoit 5 demandes de Maintenance Corrective par an sur TrioCFD.

Traitement des Maintenances Evolutives (ME)

Toute intervention relevant de la maintenance évolutive sera précisément circonscrite, spécifiée et tracée et représentera au plus un volume de travail estimé à 10 h. jours (estimation validée par le CEA). Les interventions de maintenance évolutive demandant plus que 10 jours et moins de 20 jours de travail sont hors des prestations forfaitaires mais s'inscrivent dans le cadre des prestations sur devis. Les interventions de maintenance évolutive demandant plus de 20 jours de travail seront traitées dans un autre cadre que celui du contrat de TMA. Elles entrent également dans la partie forfaitaire du marché actuel.

Une Maintenance Évolutive correspond à une action légère de développement avec une répartition des actions décrites au Chapitre VI.1 de cette partie entre l'équipe CEA et l'équipe TMA. Ce type de Demande d'Intervention se fait par sollicitation du RPL vers l'équipe de TMA. Le RPL commence par créer une fiche de Demande d'Intervention dans le BugTracker TrioCFD. Il y précise, à l'aide l'Initiateur, les spécifications (voir section VI.1) attendues dans le cadre de l'action.

Lors d'un COSUIV, l'action est détaillée par le RPL à l'équipe TMA (plusieurs rebouclages peuvent être faits afin de bien cerner l'action) puis le RML établit un chiffrage pour mener à bien la demande de Maintenance Évolutive. Suite à ce chiffrage, le RPL donne son accord (ou non) pour lancer l'action et précise dans la fiche du BugTracker, l'échéance attendue. Le responsable de l'action dans l'équipe TMA est alors en charge d'effectuer les étapes 2, 3 et 4 précisées dans le Chapitre VI.1 de cette partie, avec un point d'avancée à la fin de chacune avec le RPL. L'Initiateur peut également intervenir à l'étape 4 afin de s'assurer que le développement fournit bien les fonctionnalités et résultats attendus. L'étape 5 de relecture et d'intégration est à la charge du CEA.

Le contrat de TMA actuel prévoit 2 demandes de Maintenance Évolutive par an sur TrioCFD.

Traitement des DI LIV

L'équipe de TMA est en charge d'assurer 2 livraisons du code chaque année, généralement une fin juin et la seconde mi-décembre. Dans le cadre du marché actuel, il s'agit d'une action sur bordereau de prix (périmètre et chiffrage associé bien défini). Le CEA définit le planning et les destinataires des livraisons de ses logiciels dans les lettres de cadrage semestrielles. Un déplacement sur le site du client du CEA peut éventuellement être nécessaire. La livraison des nouvelles versions est planifiée par le CEA, qui en confie une partie de la préparation et de l'exécution à l'équipe TMA (voir chapitre [VI.2](#)).

Les tâches à accomplir comprennent notamment :

- La compilation sur différents OS (une douzaine) ;
- Le passage des différents tests (non régression, vérification, ...) ;
- La génération de la version, la mise à disposition sur l'outil de partage (GITHUB) et la création de l'archive ;
- La mise à jour d'une partie la documentation ;
- L'établissement de la note de version ;
- Les installations de la version pour chaque OS sur le réseau interne du CEA et les calculateurs (3 à 4) du CCRT.

Traitement des DI FOR

Ce type de Demande d'Intervention correspond à la dispense de formation auprès des utilisateurs du code. Elles sont au nombre de 2 par an sur TrioCFD, généralement en Mars et en Octobre. Dans le cadre du marché actuel, il s'agit également d'une action sur bordereau de prix (périmètre et chiffrage associé bien défini). Elles ont lieu sur les sites CEA en région parisienne ou en province dans des locaux prévus à cet effet. La durée des sessions est variable, de 1 à 2 jours suivant les codes et les sujets traités. Le nombre de participants est aussi variable, au maximum 12 par session. Les formations peuvent être dispensées en Français ou en Anglais suivant la demande du CEA. Les formations sont planifiées et organisées par le CEA (présentation dans les lettres de cadrage semestrielles) qui en confie la préparation et l'exécution au titulaire du contrat de TMA.

Ce type d'activité est identifié comme DI dans le BugTracker à des fins d'organisation/planification de l'activité de maintenance. La fiche de BugTracker permet également d'assurer le suivi des personnes assistant à la formation. Les tâches incombant à la TMA comprennent la préparation des machines et logiciels de la salle de formation, la préparation/mise à jour des supports de formation et la dispense d'une partie des formations.

La formation dispensée sur TrioCFD est une formation à destination des utilisateurs avec, pour commencer, une phase de prise en main puis une description et une démonstration des capacités du code et également de nombreux exercices illustrant les différents domaines et utilisations possibles. Une seconde formation à destination des développeurs est supportée par l'équipe TRUST mais convient également parfaitement aux développeurs de TrioCFD puisque tous deux utilisent le même langage, la même architecture et la même philosophie de codage. Elle dure également 2 jours et nécessite la formation dite "utilisateur" comme pré-requis.

En terme de déroulé, le CEA et la TMA commencent par définir la date exacte pour la dispense de la formation, environ 2 mois avant les mois habituels, soit en janvier et en août. Une fois cette date définie, le RPL s'assure de réserver la salle de formation. Un mail d'annonce est envoyé, dans la foulée, par la TMA aux utilisateurs répertoriés du projet. Toutes les réponses sont archivées dans la fiche correspondante du BugTracker. 1 mois avant la formation, les participants sont sélectionnés par date de réception de leur demande avec un ajustement possible en cas de besoin dûment justifié. La TMA les en informe en rappelant la date et le lieu exacte. 15 jours avant la formation, la TMA installe la dernière version de TRUST et de TrioCFD sur les machines sur lesquelles la formation sera dispensée et fournit les supports au CEA pour relecture et impression. 1 semaine avant la formation, la TMA envoie un dernier mail de rappel aux participants de la session. Durant la formation, l'équipe TMA est en charge de la présentation des supports

et de l'encadrement des exercices avec un support de l'équipe CEA. En fin de formation, un questionnaire de satisfaction est fourni aux participants afin de faire évoluer leur contenu si besoin.

VII. Communication

PLUSIEURS canaux de communications ont été mis en place progressivement autour de TrioCFD afin de permettre aux utilisateurs et développeurs de TrioCFD d'échanger sur le code, sur les difficultés qu'ils rencontrent, d'exprimer leurs besoins autant en terme de développements que d'outils, mais également de les informer sur les capacités et évolutions de TrioCFD. Chaque canal est dédié à un type d'information particulier avec une description de chacun d'eux ci-dessous.

VII.1

Site TrioCFD

Le site TrioCFD a été construit en 2019 et est accessible à l'adresse suivante : "http://trio CFD cea.fr/Pages/Presentation/TrioCFD_code.aspx". Il est actuellement construit comme illustré sur la figure VII.1.1. Les rubriques encadrées en rouge ne sont actuellement pas renseignées et celles en jaune, sont en cours de rédaction. Les rubriques non encadrées sont achevées.

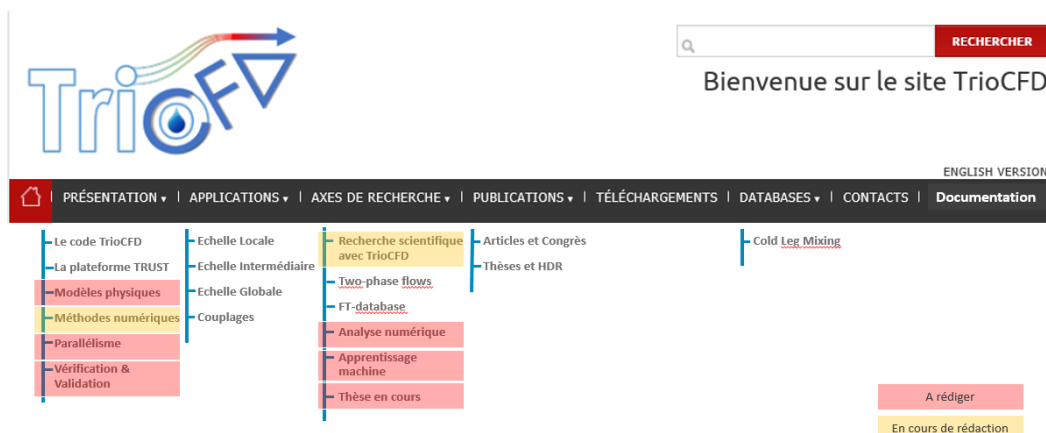


FIGURE VII.1.1: Structure actuelle du site TrioCFD.

L'objectif du site est de présenter de façon générale le code TrioCFD, les modèles qui y sont implémentés et les applicatifs pour lequel il est utilisé/applicable. Un bon nombre d'articles ou de rapport de thèses présentant les travaux effectués avec TrioCFD y sont disponibles ainsi que la documentation du code (rapport de validation, documentation des modèles, manuel de référence). La page d'accueil regroupe les actualités du code comme l'annonce de séminaire, de sortie de version,... Le lien vers GitHub sur lequel le code source de TrioCFD est disponible est également référencé.

Le site s'adresse autant aux utilisateurs interne CEA du code qu'aux utilisateurs universitaires ou industriels. Un formulaire de contact est à la disposition des utilisateurs (pour ceux ne connaissant pas l'adresse du projet) afin qu'ils soient néanmoins en mesure de prendre contact avec l'équipe en cas de question ou de problème.

Pour l'instant, le site est très majoritairement en français et toutes les parties nommées ci-dessus ne sont pas achevées. Un travail de refonte du site est actuellement en cours afin notamment de l'enrichir avec les nouveaux applicatifs pour lesquels TrioCFD est utilisé. Lorsque le nouveau format du site sera terminé, celui-ci sera exclusivement en anglais. Une première maquette de la nouvelle structure du site est donnée en figure VII.1.2

Applications de TrioCFD		
Le code TrioCFD a la capacité de traiter différents types d'écoulements auxquels peuvent être ajoutés d'autres phénomènes suivant l'applicatif à modéliser. Voici une vue d'ensemble des capacités de TrioCFD. Pour chaque catégorie, les modèles résolus ainsi que des exemples sont décrits.		
Hydraulique		
RANS	LES	DNS
Thermohydraulique		
RANS	LES	DNS
Quasi compressible		
Multi-phase Flow		
Front-Tracking	Phase Field	CMFD (HEM)
Interactions Fluide/structure	Chimie	Scalaire passif
Discrétisation		
VDF	VEF	Galerkin Discontinu

FIGURE VII.1.2: 1^{ère} maquette de la nouvelle structure du site TrioCFD.

Il s'agit là d'une première ébauche qui pourra être quelque peu modifiée lors de son implémentation. Cette nouvelle structure a pour but de créer une cartographie complète et structurée des domaines d'utilisation de TrioCFD, des modèles utilisés pour chaque applicatif et des illustrations de ceux-ci par des exemples. Dans sa forme finale, le site présentera également l'équipe TrioCFD, le formulaire de contact sera conservé ainsi que la base de données (DATABASE) et les publications relatives à TrioCFD.

VII.2

GT et réunion de développement

Deux réunions avec des formats différents ont lieu chaque mois :

- ⇒ **Groupe de Travail TrioCFD** : il s'agit d'une réunion interne au laboratoire supportant TrioCFD, à savoir le LMSF, avec une orientation plutôt monphasique ; elle est animée par le chef de laboratoire. L'équipe fait un état d'avancement sur les différentes tâches qui lui sont attribuées sur TrioCFD. Les nouvelles collaborations sont également présentées à l'ensemble de l'équipe avec les enjeux et les difficultés rencontrées. Le chef de laboratoire communique également à l'ensemble de l'équipe les informations issues de la ligne hiérarchique, et le responsable de lot (SELMA) les informations issues de la ligne projet.
- ⇒ **Réunion de développement TrioCFD** : cette réunion est ouverte à l'ensemble des utilisateurs et développeurs CEA de TrioCFD (quelque soit son laboratoire ou son site de rattachement) ; elle est animée par le responsable de code. Celui-ci évoque les sujets d'actualité sur TrioCFD (sortie de version, développements en cours ou à venir, séminaires,...). Il propose également à l'équipe, des outils et pratiques pour améliorer la qualité du code, les conditions de travail pour mener les études et les développements,... Après consensus sur ces outils et mise en place, leur utilisation est détaillée dans une réunion de développement suivante.

Si un besoin particulier est exprimé, une réunion supplémentaire peut être provoquée, ou la fréquence augmentée sur une période donnée.

VII.3

Séminaires

Différents formats de séminaires rythment l'année :

- ⇒ **Séminaire des étudiants** : Au départ des stagiaires, doctorants ou post-doctorants, un séminaire est organisé en interne CEA afin qu'ils présentent les travaux qui ont été effectués dans ce cadre. En ce qui concerne les stagiaires, le séminaire regroupe plusieurs présentations traitant d'une même thématique. Ainsi, plusieurs séminaires de ce type seront organisés chaque année. Pour les doctorants, le séminaire dédié lui permet un entraînement en grandeur nature pour sa soutenance de thèse.
- ⇒ **Séminaire des permanents** : Lorsque un sujet de recherche, une étude ou un livrable est arrivé à maturité, un séminaire spécifique est organisé en interne CEA afin d'échanger sur le sujet.
- ⇒ **Séminaire TrioCFD** : Tous les 2 ans, le séminaire TrioCFD est organisé avec l'ensemble des utilisateurs internes ou externes CEA. La journée s'articule autour de présentations techniques autant sur des applications pour lesquelles TrioCFD est utilisé pour la modélisation que sur des développements majeurs de nouvelles fonctionnalités. C'est une occasion pour les utilisateurs de découvrir les avancées du code sur les deux dernières années et d'avoir un panel complet des applicatifs. Une partie importante de la journée est également dédiée aux échanges et au partage d'expériences.

VIII. Conclusion

CE Plan de Gestion de Configuration détaille les différents outils, méthodes et processus de TrioCFD en vue d'une démarche soit R&D soit industrielle. Celui-ci sera mis à jour régulièrement en fonction des évolutions de TrioCFD et enrichi pour chaque livraison de la version.

Bibliographie

- [1] DES-CEA. Site web du logiciel salome. <https://www.salome-platform.org>.
- [2] Gmesh. Site web du logiciel gmesh. <https://gmsh.info>.
- [3] ISO. Iso 10007 : 2017 - systèmes de management de la qualité - lignes directrices pour la gestion de la configuration. <https://www.iso.org/fr/standard/70400.html>.
- [4] Tuleap. Site web de l'outil tuleap. <https://www.tuleap.org/1>.