Retour au Dossier Organisationnel

Sujets connexes	Cartographie fonctionnelle de LoGeAs Proc#145 : Procédure de Gestion des Versions [OBSOLÈTE] Procédure de versionning des codes sources (ancienne Procédure #30) [OBSOLÈTE] Procédures de gestion du dépôt GitLab [OBSOLÈTE] Procédure de Dépôt des Codes Sources (ancienne Procédure #09)

Gestion des tickets

Informations qualité

Suivi des modifications majeures	juin 2026 - Nicolas Marchand - Refonte complète 5 mars 2015 - Nicolas Marchand - Création du document 27 novembre 2015 - Nicolas Marchand - Evolution vocabulaire, non-gestion du temps dans projeQtOr 27 novembre 2015 - Valentin Barrere - Correction orthographique 02 aout 2017 - Nicolas Marchand - Portage sur DoKuWiKi & Evolutions ajout “A tester”
Suivi des approbations	ce fait sur la plateforme de cartographie fonctionnelle
Objet	L'objet de ce document est d'indiquer la procédure à suivre lors de la vie d'un ticket
Destinataires	**- Validation des modifications :** Gérant **- Approbation du document** : Equipe dev & Equipe Ass

Qualification des tickets

Types de tickets

Type	Quand l'utiliser ?	Info clé à fournir	Priorité	Traitement
Bug	Le logiciel produit un résultat incorrect ou inattendu	Étapes de repro + résultat observé vs attendu	Bloquant → Majeur → Mineur	Investigation immédiate si bloquant
Évolution	La fonctionnalité demandée n'existe pas encore	Besoin métier + cas d'usage + critères d'acceptation	Selon impact métier et charge estimée	Planification sprint / roadmap
Amélioration	La fonction existe mais pourrait être mieux conçue	Usage concerné + problème concret + gain estimé	Faible à moyenne selon contexte	Backlog, trié par valeur
Question / Assistance	L'utilisateur cherche à comprendre comment faire	Ce qu'il essaie de faire + ce qu'il a tenté	Non applicable	Réponse assistance, pas devs

Arbre de décision

1. Le logiciel produit un résultat **incorrect ou inattendu** ? → **Bug**
2. La fonctionnalité demandée **n'existe pas encore** dans le logiciel ? → **Évolution**

3. La fonctionnalité **existe mais est difficile** à utiliser ou trop lente ? → **Amélioration**
4. L'utilisateur **cherche à comprendre** comment utiliser le logiciel ? → **Question / Assistance**

<note tip> Si aucune condition n'est remplie → demander des précisions à l'utilisateur avant de classifier. </note>

Bug

Un comportement incorrect ou inattendu du logiciel. Quelque chose qui devrait fonctionner ne fonctionne plus ou fonctionne mal.

Quand utiliser ce type ?

Situations
Le logiciel produit un résultat incorrect
Une action déclenche une erreur ou un plantage
Un affichage ne correspond pas à la réalité
Une donnée est perdue, dupliquée ou corrompue
Un comportement a changé sans raison apparente (régression)

Critères de classification

Critère	Valide ?
Reproductible de façon fiable	✓ obligatoire
Résultat observé ≠ résultat attendu	✓ obligatoire
Lié à un comportement existant du logiciel	✓ obligatoire
Pas une demande de nouvelle fonctionnalité	✓ obligatoire

Informations indispensables à fournir

Information	Description
Étapes de reproduction	Séquentielles : 1, 2, 3…
Résultat observé	Ce qui se passe réellement
Résultat attendu	Ce qui devrait se passer
Environnement	Navigateur, version logiciel, env (prod/test)
ID ou données de test	Identifiant adhérent, exercice, etc.
Capture d'écran ou message d'erreur	Si disponible

Pièges fréquents

Piège
“Ça marche pas” sans décrire le comportement
Confondre avec une évolution non encore développée
Signaler un bug sur une fonctionnalité jamais définie
Mélanger plusieurs bugs dans un seul ticket
Omettre les données de test nécessaires à la reproduction

Exemples

✗ À éviter	✓ Bon exemple
Objet : Bug adhérents *Dans les adhérents ya un truc bizarre. Des fois ça marche des fois pas. C'est urgent.*	**Objet :** Filtre statut non réinitialisé après navigation — liste adhérents *Filtre “statut actif” reste appliqué après retour liste. Résultat : 247 affichés au lieu de 183. Reproduction : Adhérents > filtrer actif > ouvrir fiche #1042 > retour liste. Firefox 124, prod v2.4.1.*

Confusion fréquente

<note warning> Si la fonctionnalité n'a jamais existé, ce n'est **pas un bug** — c'est une **évolution** à développer.
À distinguer de : Évolution (fonctionnalité absente), Amélioration (fonctionne mais mal conçue)
</note>

Évolution

Une demande de nouvelle fonctionnalité absente du logiciel. Le logiciel fonctionne comme prévu — l'utilisateur veut quelque chose de plus.

Quand utiliser ce type ?

Situations
La fonctionnalité demandée n'existe pas encore
Un nouveau besoin métier émerge
Un processus manuel pourrait être automatisé
Une intégration avec un autre outil est souhaitée
Un nouveau type de données doit être géré

Critères de classification

Critère	Valide ?
La fonctionnalité n'existe nulle part dans le logiciel	✓ obligatoire
Exprime un besoin métier nouveau ou non couvert	✓ obligatoire
Implique un développement significatif	✓ obligatoire
Doit être priorisé et planifié	✓ obligatoire

Informations indispensables à fournir

Information	Description
Besoin métier	Le pourquoi — pas seulement le quoi
Cas d'usage concret	Qui fait quoi, quand, dans quel contexte
Critères d'acceptation	Mesurables : oui/non, valeur, comportement
Périmètre fonctionnel	Ce qui est inclus ET ce qui est exclu
Impact si non réalisé	Temps perdu, risque, coût

Pièges fréquents

Piège
“Refaire entièrement” le module : trop vague
Mélanger 5 demandes dans un seul ticket
“Améliorer” sans dire en quoi concrètement
Omettre le contexte métier (le pourquoi)
Ne pas préciser les critères de succès

Exemples

✗ À éviter	✓ Bon exemple
Objet : Amélioration événements *Il faudrait améliorer la gestion des événements, c'est vraiment pas pratique pour nos bénévoles.*	**Objet :** Export CSV des inscrits depuis la fiche événement *Pouvoir exporter la liste des inscrits à un événement en CSV depuis la fiche événement, pour alimenter notre outil de relance. Actuellement : export manuel copier-coller. Impact : 2h/mois perdues par la secrétaire.*

Confusion fréquente

<note warning> Si la fonctionnalité existe mais est mal conçue, c'est une **amélioration**. Si elle n'a jamais existé, c'est une **évolution**.
À distinguer de : Amélioration (existe déjà mais perfectible), Bug (fonctionne de façon incorrecte)
</note>

Amélioration

Une fonctionnalité existante qui fonctionne correctement, mais qui pourrait être plus fluide, plus ergonomique ou plus efficace. Le besoin est couvert — l'expérience peut être optimisée.

Quand utiliser ce type ?

Situations
La fonction existe et fonctionne, mais est lente à utiliser
Le nombre de clics ou d'étapes est trop élevé
L'interface est source d'erreurs fréquentes
La terminologie ou l'organisation est confuse
Un usage récurrent mériterait d'être simplifié

Critères de classification

Critère	Valide ?
La fonctionnalité concernée existe déjà	✓ obligatoire
Elle fonctionne correctement (pas de bug)	✓ obligatoire

Critère	Valide ?
L'objectif est de gagner en efficacité ou clarté	✓ obligatoire
Peut porter sur l'ergonomie ou la performance	✓ obligatoire

Informations indispensables à fournir

Information	Description
Fonctionnalité existante identifiée	Module, écran, action précise
Problème d'usage concret	Ce qui est pénible, source d'erreur
Proposition ou direction souhaitée	Ce qu'on voudrait à la place
Fréquence d'usage	Combien de fois par jour/semaine/mois
Gain estimé si amélioré	Temps, erreurs, clics économisés

Pièges fréquents

Piège
“C'est moche” ou “C'est lent” sans données concrètes
Ne pas quantifier le gain potentiel
Confondre avec un bug si le comportement est incorrect
Demander une refonte complète sans périmètre défini
Oublier de préciser la fréquence d'utilisation

Exemples

✗ À éviter	✓ Bon exemple
Objet : Interface pas très belle *L'interface est pas très belle et c'est lent des fois. Vous pourriez améliorer ça ?*	**Objet :** Réduction du nombre de clics — saisie adhérent *La création d'un adhérent requiert 12 clics sur 4 écrans. Regrouper les champs obligatoires sur un seul écran réduirait à 4 clics. Constat sur 3 secrétaires : ~15 min/jour perdues. Risque d'erreur de saisie élevé.*

Confusion fréquente

<note warning> Si la fonctionnalité fonctionne de façon incorrecte, c'est un **bug**. Si elle n'existe pas encore, c'est une **évolution**.
À distinguer de : Bug (comportement incorrect), Évolution (fonctionnalité absente) </note>

Question / Assistance

L'utilisateur ne sait pas comment utiliser une fonctionnalité existante, ou croit à tort que c'est un bug. Ce n'est ni un bug ni une demande — c'est un besoin d'accompagnement.

Quand utiliser ce type ?

Situations
L'utilisateur ne trouve pas comment faire quelque chose

Situations
Il pense que le logiciel bug mais ce n'est pas le cas
Il cherche à confirmer un comportement normal
Il a besoin d'une explication ou d'un tutoriel
La documentation ne couvre pas son cas

Critères de classification

Critère	Valide ?
Le logiciel fonctionne correctement	✓ obligatoire
L'utilisateur ne maîtrise pas la fonctionnalité	✓ obligatoire
Résolu par de l'aide, pas du développement	✓ obligatoire
Peut révéler un manque de documentation ou d'UX	à vérifier

Informations indispensables à fournir

Information	Description
Ce que l'utilisateur essaie de faire	L'objectif, pas le problème technique
Ce qu'il a déjà tenté	Les actions déjà réalisées
L'écran ou module concerné	Contexte de navigation
Son niveau de connaissance	Novice, intermédiaire, expert

Pièges fréquents

Piège
Traiter comme un bug alors que ça fonctionne
Transmettre aux devs sans qualification préalable
Ne pas détecter le bug réel derrière la question
Ignorer que la question révèle un problème d'UX

Exemples

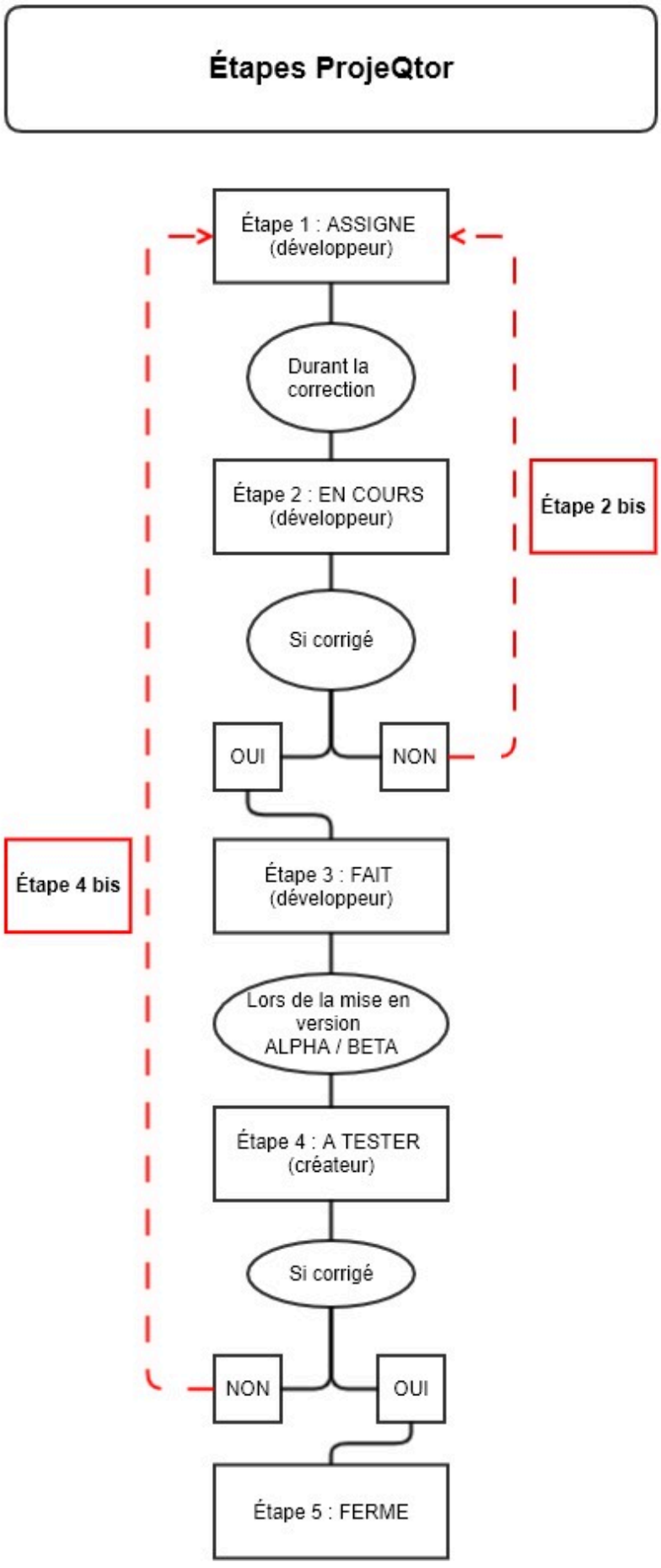
✗ À éviter	✓ Bon exemple
Objet : Ça marche pas les cotisations *Ça marche pas pour les cotisations, je comprends pas. Pouvez-vous regarder ?*	**Objet :** Bouton Générer grisé — appel de cotisations 2025 *Comment générer l'appel de cotisations pour l'exercice 2025 ? Menu Cotisations > Appels > bouton Générer grisé. Exercice 2025 créé et validé. Tentative : clic sur Générer sans résultat.*

Confusion fréquente

<note warning> Une question peut cacher un vrai **bug** ou révéler un besoin d'amélioration de l'**ergonomie**. Toujours investiguer avant de classifier.
À distinguer de : Bug (si le comportement est vraiment incorrect), Amélioration (si l'UX est systématiquement confuse) </note>

Les éléments nécessaires

Etapes de traitement


Étapes ProjeQtor
Étape 1 : ASSIGNE (développeur)
Durant la correction
Étape 2 : EN COURS (développeur)
Étape 2 bis
Si corrigé
OUI
NON
Étape 4 bis
Étape 3 : FAIT (développeur)
Lors de la mise en version ALPHA / BETA
Étape 4 : A TESTER (créateur)
Si corrigé
NON
OUI
Étape 5 : FERME

	Responsable	**Type de ticket**	**Urgence**
Niveau 1 (plus urgent)	Nicolas	Problème sur base Client	Bloquant
Niveau 2	Nicolas	Problème sur base Client	Urgent
Niveau 3	Nicolas	Problème sur base Client	Non Urgent
Niveau 4	Nicolas	Dysfonctionnement	Bloquant
Niveau 5	Nicolas	Dysfonctionnement	Urgent
Niveau 6	Nicolas	Dysfonctionnement	Non Urgent

Etape 1 : Référencement d'une demande

Procédure vis-à-vis du client

La première action à faire vis-à-vis du client est d'identifier sur quelle version du logiciel il travaille. S'il n 'est pas sur la dernière version disponible, il lui est demandé de bien vouloir le mettre à jour, puis de vérifier si le problème persiste.

AUCUNE assistance sur les fonctionnalités n'est faite sur les versions antérieures du logiciel à partir du moment où une mise à jour est publiée. Seules l'assistance à l'évolution de la version et/ou l'aide à la correction conséquente à un problème dû à une version antérieure seront prises en compte.

Chaque cas sera étudié et une réponse sera émise au client dans tous les cas. Celle-ci sera obligatoirement tracée au travers de la procédure d'assistance

Cas où une correction est nécessaire

Si lors d'une demande d'assistance, d'une formation ou de test, un problème nécessitant une action sur le code ou les fichiers annexes est repéré, la procédure est la suivante :

Cas 1 : Problème impliquant uniquement une correction du code

1. Lancer une session Projeqtor
    - Sélectionner le Projet et la sous-version correspondant à la prochaine version [PROC-Nflog-5 : gestion des versions]
    - Dans “Travail\Bugs”, créer un nouveau Ticket et le remplir comme indiqué plus loin en tenant compte des spécificités listées ci-dessous :
        - **Projet** = **Prochaine sous-version** (par défaut si le projet est sélectionné)
        - **Type de ticket = “Anomalie / Bug issue - origine client” ou “Anomalie / Bug issue - origine autre”**
        - **Référence externe = Numéro de ticket OTRS s'il existe.** Exemple :*Ticket#2015022310000059*
        - **Responsable = vide**
    - NB : Dans le cas où la demande est un doublon par rapport à une demande existante, on se contentera de compléter le ticket existant en ajoutant la référence à la demande du client.
2. Revenir sur le ticket OTRS

- Envoyer une réponse à l'utilisateur
- Faire une note en indiquant en titre “PROJEQTOR –BUGS#XX” et le texte de votre choix

Cas 2 : Problème impliquant uniquement une décision du groupe de travail EPUdF-Logeas

1. Lancer une session Projeqtor
   - Sélectionner le Projet et la sous-version correspondant à la prochaine version [certif:procedure:develop:gestionversion|PROC-Nflog-5 : gestion des versions]
   - Dans “Journaux des revues\Questions”, créer un nouveau Ticket et le remplir comme indiqué plus loin en tenant compte des spécificités listées ci-dessous :
     - **Projet = Prochaine sous-version** (par défaut si le projet est sélectionné)
     - **Type de ticket = “Assistance - demande d'explication EPUdF” ou “Assistance - demande de correction EPUdF“**
     - **Référence externe = Numéro de ticket OTRS s'il existe.** Exemple :*Ticket#2015022310000059*
     - **Responsable = “Jean-Marc DEGON & Michel HAFFNER”**
   - NB : Dans le cas où la demande est un doublon par rapport à une demande existante on se contentera de compléter le ticket existant en ajoutant la référence à la demande du client.
2. Revenir sur le ticket OTRS
   - Envoyer une réponse à l'utilisateur
   - Faire une note en indiquant en titre “PROJEQTOR – TICKET #XX” et le texte de votre choix (obligatoire, mais sans intérêt ..)

Cas 3 : Demande d'évolution du logiciel

1. Lancer une session Projeqtor
   - Sélectionner le Projet et la sous-version correspondant à la prochaine version [PROC-Nflog-5 : gestion des versions]
   - Dans “Travail\Evolutions”, créer un nouveau Ticket et le remplir comme indiqué plus loin en tenant compte des spécificités listées ci-dessous :
     - **Projet = Prochaine sous-version** (par défaut si le projet est sélectionné)
     - **Type de ticket = “Assistance - demande d'évolution ''**
     - **Référence externe = Numéro de ticket OTRS s'il existe.** Exemple :*Ticket#2015022310000059*
     - **Responsable = vide**
   - NB : Dans le cas où la demande est un doublon par rapport à une demande existante, on se contentera de compléter le ticket existant en ajoutant la référence à la demande du client.
2. Revenir sur le ticket OTRS
   - Envoyer une réponse à l'utilisateur
   - Faire une note en indiquant en titre “PROJEQTOR – TICKET #XX” et le texte de votre choix

Information générique sur la saisie d'un ticket

Description	Projet	Selon le cas
Description	Type de ticket	Selon le cas
Description	Nom	**Donner un titre compréhensible**

Description	Référence externe	Selon le cas
Description	Urgence	Non utilisé
Description	Date de création	Automatique
Description	Émetteur	Automatique, nom de l'utilisateur connecté NB : Chaque utilisateur doit avoir une session distincte
Description	Demandeur	Non utilisé
Description	Origine	Non utilisé
Description	Ticket en doublon	Utilisation postérieure
Description	Contexte	A remplir si connu
Description	Produit	Indiquez le produit
Description	Version d'origine	Indiquer la version du produit où a été constaté le problème, si connue
Description	Description	Indiquer le détail du problème
Traitement	Activité de planning	Ne pas utiliser
Traitement	Etat	Mettre “Assigné”
Traitement	Responsable	Assigner le ticket à la personne qui doit s'en occuper, ce qui envoie un mail à la personne
Traitement	Criticité	Selon le cas
Traitement	Priorité	Selon le cas
Traitement	Échéance initiale,; actuelle	Non utilisé
Traitement	Travail estimé, restant	Non utilisé
Traitement	Pris en charge, Fait, Clos	Automatique
Traitement	Version cible	Non utilisé
Avancement		Sans objet
Élément prédécesseur / Successeur		Généralement Sans Objet
Élément liés		Ne pas utiliser
Fichiers attachés		Ajouter les documents, bases ... qui permettent de mettre en lumière le problème et de tester la solution
Notes		Permet d'ajouter des éléments textuels

Etape 2 : Prise en charge par le dev d'une demande

Lancer une session Projeqtor

1. Sélectionner le ticket correspondant
2. Le basculer en “En cours”
3. Initialiser le **Traitement\Responsable** s'il ne l'est pas

Etape 2bis : Prise en charge par le dev d'une demande (Echec)

Lancer une session Projeqtor

1. Sélectionner le ticket correspondant : **“Travail\Bugs“**
    - Compléter le ticket dans la partie “**Traitement**”:
        - Initialiser le **Traitement\Responsable** s'il ne l'est pas

Activité de planning	Initialisé à la création du ticket
Etat	**Mettre “Assigné”**
Responsable	**Changer le responsable pour l'assigner à quelqu'un d'autre**
Criticité	Non utilisé
Priorité	Non utilisé
Échéance initiale,; actuelle	Non utilisé
Travail estimé, restant	Non utilisé
Pris en charge, Fait, Clos	Automatique
Version cible	Non utilisé
Résultat	vide

- Enregistrer
- Lier les documents, si il y a lieu (base de test, copie ...)
- Mettre une note si il y a lieu

Etape 3 : Prise en charge par le dev d'une demande (Succès)

Au niveau développement :

1. Effectuer la correction, la tester
2. Publier le code sur SVN, en indiquant dans le commentaire le tag du ticket (id=#20)

Au niveau ProjeQtOr

Lancer une session Projeqtor

1. Sélectionner le ticket correspondant : **“Travail\Bug“**
    - Compléter le ticket dans la partie “**Traitement**”:
        - Initialiser le Traitement\Responsable s'il ne l'est pas

Activité de planning	Initialisé à la création du ticket
Etat	**Mettre “FAIT”**
Responsable	**ne pas changer**
Criticité	Non utilisé
Priorité	Non utilisé
Échéance initiale,; actuelle	Non utilisé
Travail estimé, restant	Non utilisé

Pris en charge, Fait, Clos	Automatique
Version cible	Non utilisé
Résultat	vide

- Enregistrer
- Lier les documents, si il y a lieu (base de test, copie ...)
- Mettre une note si il y a lieu

Etape 4 : Mise en place d'une version de test (alpha ou beta)

1. Lancer une session Projeqtor
2. Dans les tickets sur le projet concerné, cliquer sur le bouton à droite “Mise à jour multiple”
3. Sélectionner le(s) ticket(s)/activité(s) correspondant à un état “FAIT”
4. Initialiser le Traitement\Responsable au créateur du ticket
5. Les basculer en “A TESTER”

From:
https://wiki-logeas.fr/certif/ - **dokuwiki-certif**

Permanent link:
https://wiki-logeas.fr/certif/doku.php?id=certif:procedure:develop:gestionprojeqtor&rev=1780410293

Last update: **2026/06/02 16:24**