Redmine

L'objectif est de proposer des préconisations sur l'usage de Redmine. Cela s'adresse à tous les acteurs d'un projet.

```
Ce document est une proposition.
Il est écrit en ASCIIDOC.
Son code est disponible ici https://git.ias.u-psud.fr/DevTeam/DevGuide/tree/develop
```

1. Présentation de REDMINE

Redmine[1] est un gestionnaire de projet OpenSource, mis en oeuvre à l'IAS[2]. Il propose un panel de services très utiles comme le suivi de demandes, la visualisation sous forme de Gantt, etc…

```
A compléter
```

[1] http://www.redmine.org
[2] https://idoc-projets.ias.u-psud.fr/redmine

Conventions

Nous distinguons trois niveaux d'exigence :

- les obligations et interdictions
- Les recommandations
- Les possibilités et les suggestions

Les exigences apparaissent clairement et en gras.

Tableau 1. Les obligations et interdictions

Indicateur	Exemple
le verbe *devoir* au présent indicatif	"Un projet **doit** avoir un nom…" "l'accès **ne doit pas** être possible…
le terme *obligatoire* et ses dérivés	"l'accès est **obligatoirement** refusé…"
le terme *interdit* et ses dérivés	"Il est **interdit** d'accorder l'accès…"

Tableau 2. Les recommandations

Indicateur	Exemple
le verbe *devoir* au conditionnel présent	"Un projet **devrait** avoir une roadmap …" "l'accès **ne devrait pas** être interdit…
le terme *recommander* et ses dérivés	"Il est **recommandé** d'accorder l'accès à…"

Tableau 3. Les possibilités et les suggestions

Indicateur	Exemple
le verbe *pouvoir* au présent indicatif	"Un projet **peut** avoir une roadmap …" "Un projet **peut ne pas** avoir une roadmap …"
le terme *possible* et ses dérivés	"Il est **possible** d'accorder l'accès à…"

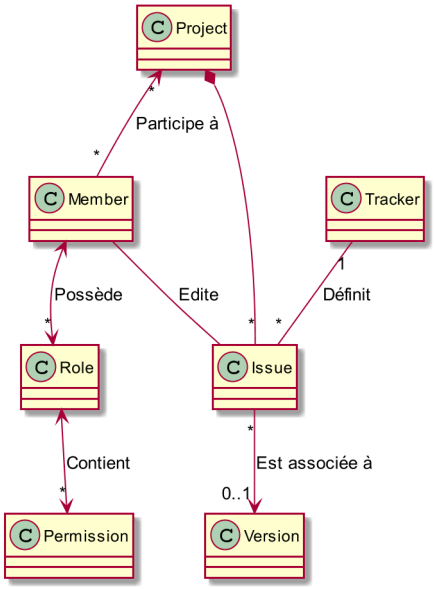
Si Redmine utilise de nombreux concepts pour permettre de gérer des projets, nous nous intéresserons aux principaux :

Tableau 4. Concepts Redmine

Concept	Définition
`project`	Ensemble des ressources associées à un projet au sens large.
`member`	Membre d'un `project`.
`permission`	Droit d'exercer une action (voir, supprimer, modifier, …) sur un objet du projet (document, liste des membre, …)

Concept	Définition
`role`	Ensemble de permissions.
`issue`	Problème ou demande.
`tracker`	Système de gestion des `issues`.
`version`	Etat d'un artefact du `projet`.

Les relations entre ces concepts sont illustrées dans un diagramme de classe:



2. Autour du projet

Cette partie est dans le chapitre REDMINE, mais elle gagnerait à être dans un chapitre plus générique.

La raison d'être de REDMINE est d'outiller la gestion de projet. Encore faut-il se mettre d'accord sur ce qu'est un projet. Plusieurs définitions existent [3].

> Un projet est un ensemble d'activités coordonnées et maîtrisées comportant des dates de début et de fin, entrepris dans le but d'atteindre un objectif conforme à des exigences spécifiques.
>
> — AFNOR X50-115

L'atteinte de l'objectif d'un projet produit des *artefacts* ou des *livrables* : des logiciels, des instruments, des jeux de données, des plans d'action, etc… C'est parfois même l'objectif principal d'un projet. Il peut y avoir un ou plusieurs artefacts, ou plusieurs sous-projets avec leurs propres artefacts.

Ces artefacts sont le résultats d'étapes, d'évolutions et de livraisons successives. Ils existent ou ont existé sous différentes versions.

[3] norme ISO 10006: Un projet est un processus unique qui consiste en un ensemble d'activités coordonnées et maîtrisées comportant des dates de début et de fin, entrepris dans le but d'atteindre un objectif conforme à des exigences spécifiques, incluant les contraintes de délais, de coûts et de ressource.

Version et roadmap

Ces versions **doivent** être nommées selon une logique signifiante, par exemple

- `VersionMajeur.VersionMineur.Correctif` (ex. 2.3.5)
- `Prefix-Itération` (ex. ACME-16)
- `YYYY-MM`, année, mois pour les productions récurrentes (ex. 2018-05)
- …

La version **devrait** avoir une date d'échéance et **peut** avoir une date de début.

Les évolutions et livraisons des *artefacts* **devraient** être associées à des *versions* pour faciliter leur identification.

La succession des versions constituent la *roadmap*.

Les projets **devraient** avoir une *roadmap*, c'est à dire un plan d'évolution.

3. Vision IAS

Dès son installation, Redmine propose une vision par défaut s'articulant autour d'une configuration minimale.

Rôles par défaut

- Manager
- Developer
- Reporter

Etats par défaut

- Non member (non membre d'un projet)
- Anonymous (non identifié)

Trackers par défaut

- Bug
- Feature
- Support

Status par défaut

- New
- In Progress
- Resolved
- Feedback
- Closed
- Rejected

L'IAS utilise une instance REDMINE qu'elle a modifiée pour l'adapter à ses besoins, notamment dans le domaine Qualité.

Tableau 5. Status IAS ajoutés ou modifiés

Nom	Evolution
Rejected / Cancelled	Modifié
Waiting For Validation	Ajouté

Tableau 6. Trackers IAS ajoutés ou modifiés

Nom	Evolution
Action	Ajouté
Change request CR	Ajouté (qualité)
Configuration	Ajouté
Design	Ajouté
Documentation	Ajouté
Infrastructure	Ajouté
Meeting	Ajouté
NewDev	Ajouté
Non-conformance NC	Ajouté (qualité)
Production	Ajouté
Request for Deviation RFD	Ajouté (qualité)
Test	Ajouté

Tableau 7. Rôles IAS ajoutés ou modifiés

Nom	Evolution
Admin	Ajouté
Documentation Validator	Ajouté

La philosophie de Redmine est toutefois très ouverte, elle n'impose aucune vision établie sur la signification de ces noms: il n'y a même pas de description associée. [4]

C'est à nous de définir, par convention, la sémantique de chacun des termes

3.1. Définition commune des statuts

Nous donnons les définition des *principaux* statuts. Elles s'appliquent *par défaut* à tous les `trackers`, sauf bien sur indication contraire ou pour des besoins spécifiques. Nous ne considérons par les changements de `trackers`.

Les définitions sont décrites par un tableau et un schéma des transitions possibles vers d'autres statuts.

Tableau 8. exemple de tableau descriptif

Sémantique	La sémantique portée par le statut.
Remarques	Les éventuelles remarques sur le statut.
Obligatoire	Les aspects **obligatoires** liés au statut.

[4] Mais une demande d'évolution en ce sens existe https://www.redmine.org/issues/442

Recommandé	Les aspects *recommandés* liés au statut.
Transitions	Les remarques sur les transitions entre statuts.

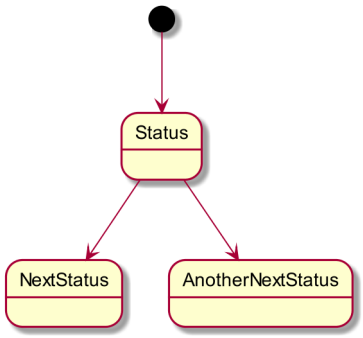
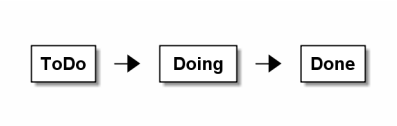


Figure 1. Exemple des Transitions par défaut d'un status

Un parallèle sera fait avec l'approche *KANBAN* [5] où les demandes sont classées dans des listes correspondant à 3 états.

- `TODO` à faire
- `DOING` en cours
- `DONE` fait

La demande passe alors d'une liste à la suivante en fonction de sa progression.



Statuts

Les statuts, par définition, permettent de caractériser l'état d'une demande.

Au delà de l'outil, quels sont les états que nous voulons voir ? Cela dépend de l'approche que nous avons des demandes.

L'approche retenue, plutôt commune [6], est de les considérer comme des demandes de client injectées dans une chaine de traitement et chaque étape du traitement de la demande est labellisée par un état.

Prenons par analogie la révision de la voiture d'un client.

Les états *abstraits* de la commande à observer:

Etape	**Illustration**
La création	Le client demande à faire réviser sa voiture
L'acceptation	Le garagiste accepte
Le rejet	La voiture est trop vieille ou c'est un garage moto.
La planification	Le garagiste inscrit la révision dans le planning

[5] cf. https://fr.wikipedia.org/wiki/Tableau_kanban
[6] ITIL avec ses requêtes par exemple

Etape	Illustration
La prise en charge	Un mécanicien prend en charge la révision
La validation	Le client réceptionne la voiture, la teste et paie
La clotûre	La révision est terminée

Statut "New"

C'est le statut par défaut à la création d'une demande.

Sémantique	La demande est officiellement formalisée.
Remarques	Même si la demande est officielle, aucun engagement n'est pris à ce stade.
Transitions	La demande ne doit pas rester en l'état *New* longtemps. Après un certain délai elle doit être examinée et passer à *Accepted* ou *Rejected*. Si la décision de la traiter est immédiate, elle doit alors passer en état *InProgress*.

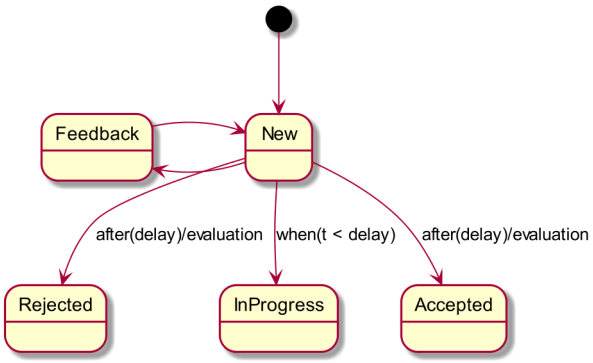


Figure 2. Transitions par défaut du status New

Statut "Accepted"

Sémantique	La demande est jugée pertinente et sera traitée. Elle rentre dans la liste TODO (à faire).

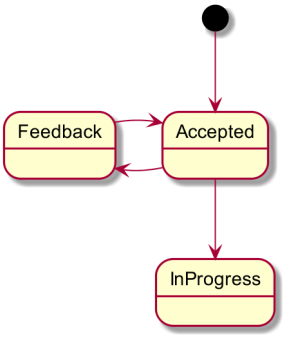


Figure 3. Transitions par défaut du status Accepted

Dans la situation actuelle (janvier 2018) le statut `Accepted` est à créer.

Pouvons-nous nous en passer ? Oui, mais il faudrait une convention d'usage très précise pour différencier une demande *déposée et oubliée* d'une demande *en cours de traitement*.

> Cela pourrait être :
>
> ```
> Toute demande acceptée est affectée à une version
> ```
>
> ou
>
> ```
> Toute demande acceptée est prise en compte, mais avec un poucentage de réalisé
>  de 0%
> ```
>
> ou
>
> ```
> Toute demande nouvellement créée doit être examinée et rejetée dans les 10
>  jours
> sinon elle est considérée comme acceptée.
> ```
>
> Ces conventions sont tout de même fragiles et ambigües. Mieux vaut garder des sémantiques simples et intuitives.

Statut "Rejected / Canceled"

Sémantique	La demande n’est pas ou plus jugée pertinente et ne sera pas traitée.
Obligatoire	La demande ne **doit** pas être réouverte. Une autre demande doit être créée en lien avec celle-ci.
Remarques	Le statut est final. En cas de désaccord une nouvelle demande doit être ouverte.

Statut "Closed"

Sémantique	La demande est clôturée, la solution est en place.
Obligatoire	La demande ne **doit** pas être réouverte. Une autre demande doit être créée en lien avec celle-ci.
Recommandé	La version indiquée est bien celle mise en place.

Statut "In Progress"

Sémantique	La demande est prise en charge et est en cours de traitement Elle rentre dans la liste DOING (en cours).
Obligatoire	Être assignée à quelqu’un
Recommandé	Être associée à une version cible, Avoir une estimation de la durée ou de la date de fin provisionnelle. Mettre à jour le pourcentage de réalisation si la durée dépasse quelques jours.

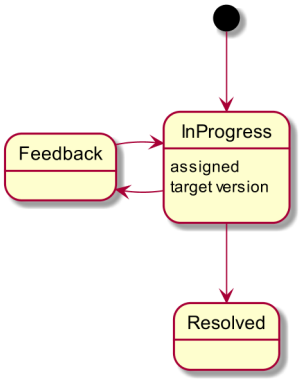


Figure 4. Transitions par défaut du status Rejected

Statut "Resolved"

Sémantique	La demande est résolue, la solution est trouvée et testée par le développeur. Elle est prête à être mise en place dans un environnement de validation pour être validée. Elle rentre dans la liste DONE (fait).
Obligatoire	Le pourcentage de résolution est à 100%.
Recommandé	Indiquer le temps consommé.
Transitions	La demande peut passer en statut *Closed* si les conditions de légitimité sont assurées. C'est à dire que l'acteur répond à sa propre demande, qu'il est apte à juger que le besoin est pleinement satisfait ou que les modifications peuvent passer en production par exemple. Dans le cas contraire, la demande doit passer en statut *WaitForValidation* avec assignation de la personne légitime.

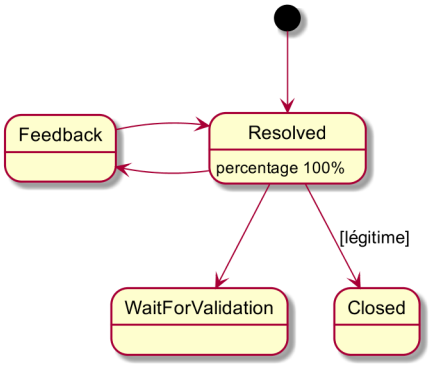


Figure 5. Transitions par défaut du status Rejected

Statut "WaitForValidation"

Sémantique	La solution de la demande est mise à disposition et doit être validée.
Obligatoire	La demande est réassignée à un validateur

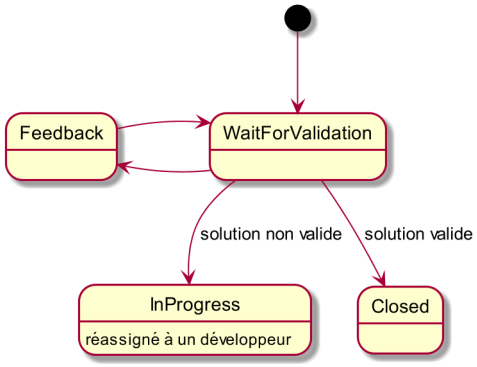


Figure 6. Transitions par défaut du status WaitForValidation

Statut "Feedback"

Sémantique	La demande est en attente d'un complément d'informations.
Remarques	Cet état signifie que la progression de la demande est vraiment conditionnée par l'obtention du complément d'information.
Obligatoire	Être assignée à celui qui peut donner les informations.
Transitions	Depuis tous les états sauf les états finaux (*Closed* et *Rejected/Cancelled*). Vers tous les états, y compris finaux.

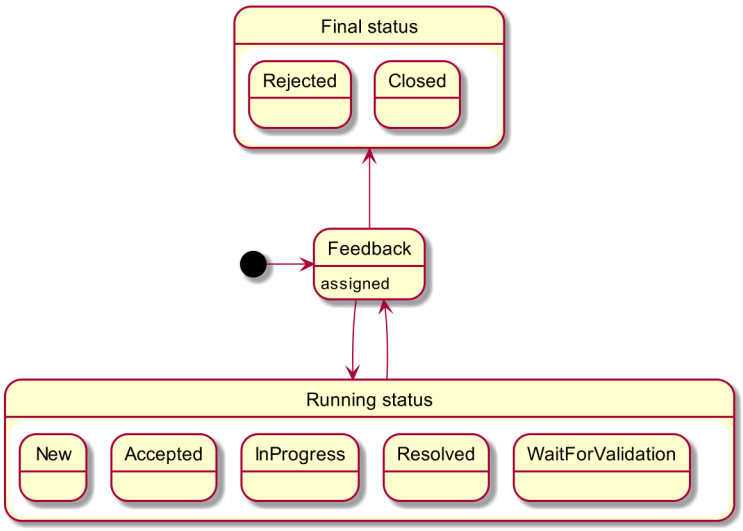


Figure 7. Transitions par défaut du status Feedback

Resumé

Statut	**Sémantique**
New	La demande est officiellement formalisée.
Accepted	La demande est jugée pertinente et sera traitée. TODO
Rejected	La demande n'est pas ou plus jugée pertinente et ne sera pas traitée
InProgress	La demande est prise en charge et est en cours de traitement. DOING
Resolved	La demande est résolue DONE et peut être mise à dsiposition.
WaitForValidation	La solution de la demande est mise à disposition et doit être validé.

Statut	Sémantique
Closed	La demande est clôturée, la solution est en place.
Feedback	La demande est en attente d'un complément d'informations.

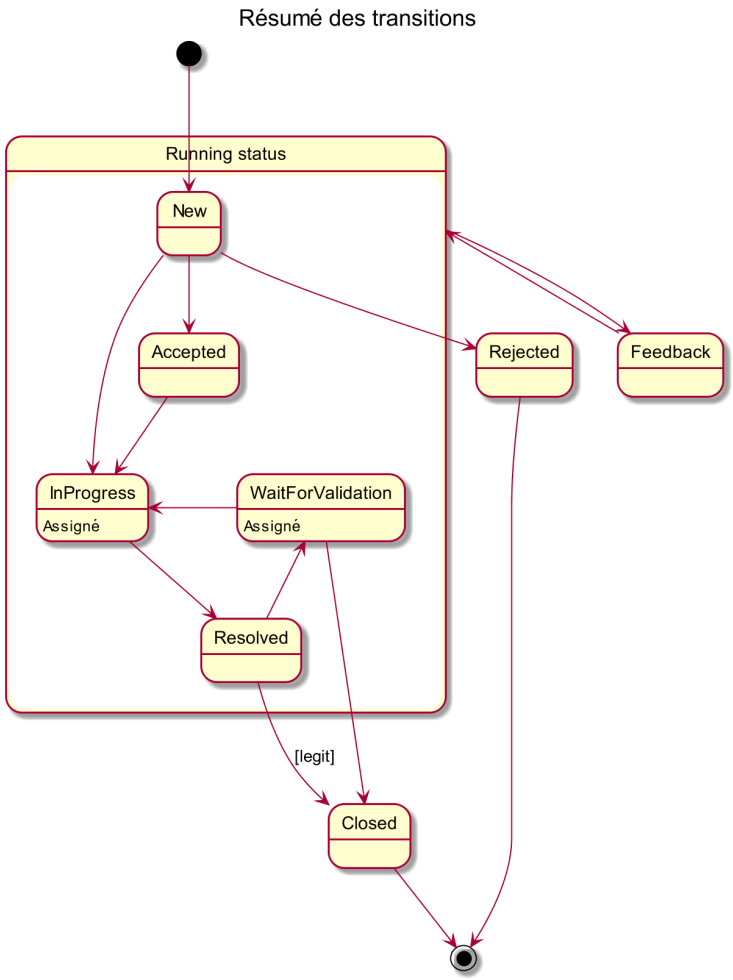


Figure 8. Résumé de toutes les transitions

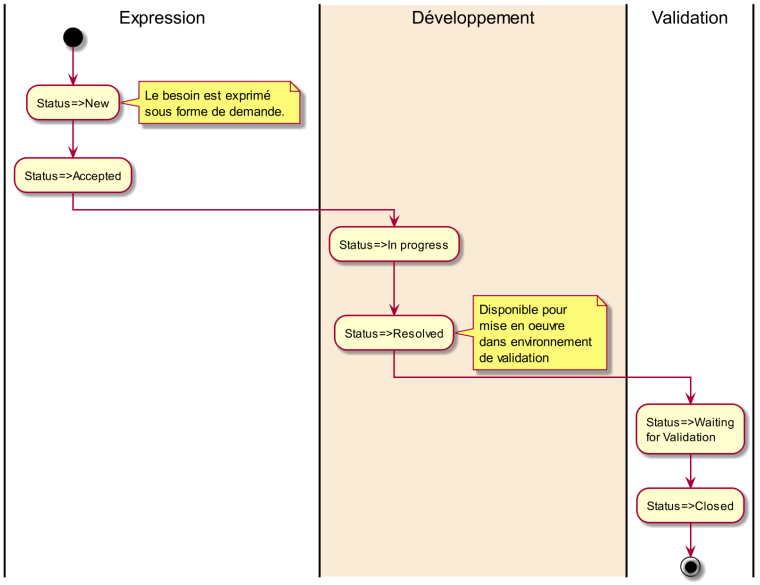


Figure 9. Résumé de toutes les transitions par phase

3.2. Définition commune des trackers

Nous donnons les définitions des *principaux* `trackers`. Elles s'appliquent *par défaut* à tous les projets sauf indication pour des cas spécifiques. De même sauf indication pour des cas spécifiques, le Workflow à utiliser est le Workflow standard.

La vision est très liée à la conduite de projet de développement logiciels. Elle tente néanmoins d'être plus raisonnablement généraliste.

Définitions

Pour mémoire, Redmine n'en définit que 3 à l'installation

- Bug
- Feature
- Support

Les définitions sont sous forme d'un tableau

Tableau 9. Exemple de tableau de définition.

Objet	Objet du tracker
Remarques	Remarques sur le tracker.
Utilisation	Utilisations du tracker, estimé sur le nombre des demandes associées (Majoritaire, Forte, Notable, Faible, Très faible)

Tracker "Action"

Objet	Description et suivi des actions sous forme de *Todo List*
Remarques	Tracker généraliste à utiliser si les autres ne conviennent pas.
Utilisation	Majoritaire

Tracker "NewDev"

Objet	Description et suivi des développements.
Remarques	**ATTENTION** comme c'est le premier tracker proposé dans la liste, pas sur qu'il ait été toujours utilisé à bon escient.
Utilisation	Forte

Le nom de ce tracker est trompeur, car il induit l'idée que les développement sont toujours nouveaux, et ceux déjà réalisés ne peuvent évoluer que sous la forme de Bug.

Mélanger un état et un sujet n'est pas pertinent.

Il serait préférable de renommer `NewDev` en `Development` ou, en complémentarité à `Feature`, `Technical`, voire `Realisation` pour dépasser un peu le cadre du développement logiciel.

Tracker "Documentation"

Objet	Description et suivi des actions orientées Documentation.
Utilisation	Notable

Tracker "Bug"

Objet	Description des défauts et suivi des corrections.
Utilisation	Notable

Tracker "Production"

Objet	Description et suivi des actions dans un contexte de production.
Remarques	Spécialisation de *Action*.
Utilisation	Notable

Tracker "Feature"

Objet	Description et suivi des évolutions des fonctionnalités offertes.
Utilisation	Notable

Tracker "Test"

Objet	Description et suivi des tests.
Utilisation	Notable

Tracker "Infrastructure"

Objet	Description et suivi des actions concernant les infrastructures.
Remarques	Spécialisation de *Action*, sans particularité. Concerne principlement les infrastructures réseau, serveur.
Utilisation	Notable

Tracker "Support"

Objet	Description et suivi des demandes de support.
Utilisation	Faible

Tracker "Configuration"

Objet	Description et suivi des actions de configuration dans un contexte de production.
Remarques	Spécialisation de *Action*, sans particularité.
Utilisation	Faible

Trackers de type Action

Nous constatons l’existence de plusieurs trackers de type Action, dont l’usage est faible.

- Production
- Infrastructure
- Configuration

Il sera souhaitable de les différencier, de les définir, voire d'envisager d'utiliser le plugin Tags pour avoir une politique plus simple et efficiente.

Versions

Redmine ne fait pas de différence entre les différentes versions possible d'une demande. Il ne considère que la version cible, celle sur laquelle un travail **va** être réalisé, dans le futur. Or certaines demandes (Bug, demande de support) sont ancrées dans le présent, et se rapportent à une version en cours.

Pour cela nous avons créée un *champs personnalisé* en plus `Involved Version`, qui se rapporte à la version présente.

Il est activé dans les trackers `Bug` et `Support` (02/02/2018)

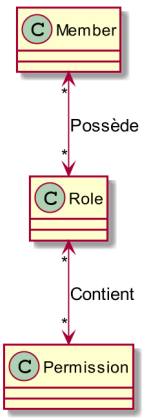
3.3. Définition commune des rôles

Nous donnons les définitions des *principaux* rôles. Elles s'appliquent *par défaut* à tous les projets sauf indication pour des cas spécifiques.

Toujours fortement orienté développement, les rôles présentés ne sont pas les seuls possibles. L'interprétation sémantique peut, dans un autre contexte, différer : Un rôle de *développeur* pourrait être associé à un *projeteur* ou un *tourneur* dans un projet de mécanique.

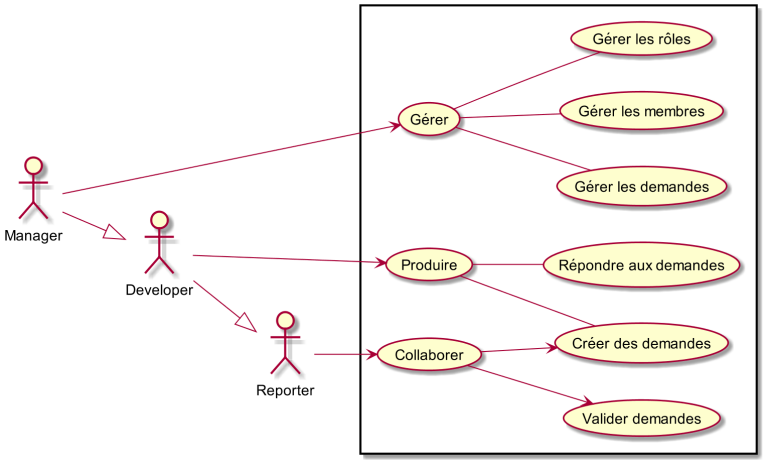
Rôles

Les rôles sont des ensembles de permissions. Ces permissions peuvent être associées à d'autres rôles.



Ces permissions sont assez nombreuses et il est difficile d'avoir une vision synthétique. L'objectif est plus ici de s'accorder sur les différences de responsabilité que sur les capacités fines de gestion.

Les rôles sont cumulables sur un projet. Ils sont parfois eux même "emboités", tels que *Manager*, *Developper* et *Reporter*.



Role "Manager"

Sémantique	Le `Manager` gère les aspects du projets
Remarques	Il pilote le projet, fait des choix, décide.

Role "Developer"

Sémantique	Le `Developper` produit, réalise.
Remarques	C'est un rôle très orienté développement informatique. Il est encadré par un manager.

Role "Reporter"

Sémantique	Le `Reporter` constate, rapporte, demande.
Remarques	C'est un rôle sans capacité de réalisation directe, mais il contribue au projet en apportant ses retours.

Role "Admin"

Sémantique	Le rôle `Admin` est un `Manager` avec de moindres permissions.
Remarques	**Attention** : Ne pas confondre avec la propriété `Administrator` d'un utilisateur qui lui donne des capacités d'administration du service REDMINE.

Role "Documentation Validator"

Sémantique	Le rôle `Documentation Validator` est un rôle d'observateur, toutes les informations lui sont accessibles, mais en lecture seule. Il peut nénamoins ajouter des demandes et les annoter.
Remarques	C'est un rôle très peu utilisé.

Je recommande de ne pas utiliser le rôle `Documentation Validator` car il n'apporte aucune plus value *pour l'instant*, sauf à l'augmenter par la suite.

Role "Non member"

Sémantique	Le rôle `Non member` est associé à toute personne identifiée dans REDMINE **mais** extérieure au projet.
Remarques	C'est un état plus qu'un rôle. Mais il a certaines permissions si le projet est publique.

Role "Anonymous"

Sémantique	Le rôle `Anonymous` est associé à toute personne **non identifiée** dans REDMINE.
Remarques	C'est un état plus qu'un rôle. Mais il a certaines permissions si le projet est publique.

Annexes définitions des rôles

Différences entre manager et admin

Par rapport à `Manager`, le role `Admin` a des permissions en moins (au 18/01/2018)

- Aucun permissions sur le plugin Agile
- Gestion des Issues
  - copy_issues
  - set_issues_private
  - set_own_issues_private
  - import_issues
  - view_issue_recurrence
  - add_issue_recurrence
  - edit_issue_recurrence
  - delete_issue_recurrence

Différences entre Reporter et Documentation Validator

Il n'y a quasiment aucune diférence entre `Reporter` et `Documentation Validator`, `Documentation Validator` n'a pas les permissions suivantes (au 18/01/2018)

- :edit_own_messages
- :log_time

Droits d'un Non-Member

Si un projet n'est pas publique, un *non-member* ne peut y accéder.

Par contre si il est publique, le *non-member* se voit appliquer la matrice de permissions

Il peut notamment, en l'état (février 2018):

- créer une demande
- ajouter des commentaires
- voir le gantt (mais pas le modifier)

- idem pour le calendrier
- idem pour le wiki
- idem pour les fichiers

3.4. Définition commune des processus

Nous donnons les définitions des *principaux* processus, qui s'appliquent *par défaut* à tous les projets sauf indication pour des cas spécifiques.

Les processus sont juste des noms sur des suites continues d'opérations, d'actions constituant la manière de faire, de fabriquer quelque chose [7]. Ce ne sont pas des procédures [8].

Les processus sont décrits par

- des acteurs avec des rôles (`Manager`, `Developer`, `Reporter` principalement), parfois cumulés,
- des entrées (input) et des sorties (output),
- des étapes et des situations,
- des procédures,
- des outils.

Convention

Redmine ne permet pas [9] de s'affranchir d'une convention d'usage. Il est des situations qui ne sont pas explicitement indiquées par l'outil. Il faut décrire ce qui doit être perçu et compris à travers plusieurs indicateurs indirects [10].

Cette convention n'est pas figée, et est construite au fur et à mesure de la vie du projet, des difficultés et enseignements rencontrés.

Outillage

Redmine met à disposition des outils.

- Les filtres de demande enregistrés : Quand vous filtrez les demandes, vous avez la possibilité de le sauvegarder et de le mettre à disposition d'autres personnes
- Les taches récurrentes: créer une demande de type action , puis dans l'écran *Recurent issue*, lui donner une fréquence adaptée.

Processus et méthodologies

Les processus sont très liés à la méthodologie utilisée. Il n'y a pas de *meilleure* approche universelle, il n'en existe que des plus adaptées que d'autres à la situation.

[7] extrait de http://www.larousse.fr/dictionnaires/francais/processus/64066#vHXCpObEEvld9p26.99

[8] Un processus c'est "Faire le repas pour la famille le soir", une procédure, c'est suivre la recette de cuisine pour faire des pâtes à la Carbonara

[9] Il n'est pas souhaitable, voire envisageable de tout verrouiller ou tout définir via Redmine. Si les procédures à appliquer deviennent trop lourdes, elles ne sont plus suivies.

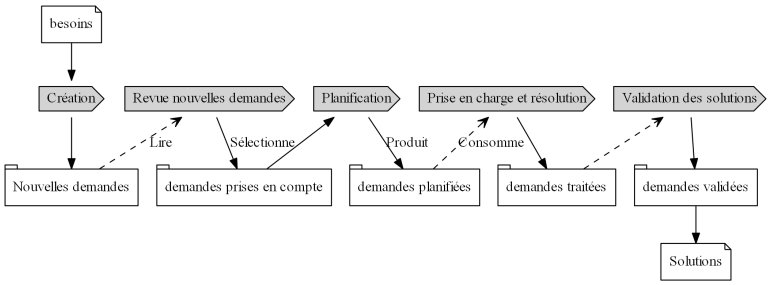
[10] Par exemple convenir qu'une demande avec une version doit être traitée et qu'en l'absence de version peut être rejetée

Liste des processus de gestion des demandes

Nous distinguerons les processus

- Création de demande
- Revue des demandes à évaluer
- Planification des prises en charges
- Prise en charge et résolution des demandes
- Validation des solutions

Ces processus permettent de répondre par des solutions à des besoins.



Dans ce qui va suivre, nous opterons, par défaut, pour une approche méthodologique de type Agile.

Processus de création de demande

La création de demande est le point de départ de l'ensemble des processus. Tous les rôles peuvent saisir des demandes.

Le sujet d'une demande peut-être:

- une **fonctionnalité**,
  c'est à dire un point de vue *métier*, *externe*, *utilisateur*, sans aborder la technique à mettre en oeuvre.
- une **réalisation**, un **développement**,
  c'est à dire un point de vue *technique*, *interne*, *développeur*.
- un **défaut**, un **dysfonctionnement**
- une demande d'**aide**, de **renseignement**, d'**information**
- une **action**, une **tache**, un **travail**,
  c'est à dire une demande au sens le plus large, qui ne peut être associée aux autres catégories plus spécialisées du projet.

Une demande qui mélange plusieurs sujets, ou qui est trop ambitieuse, **doit** être scindée en plusieurs demandes *filles*.

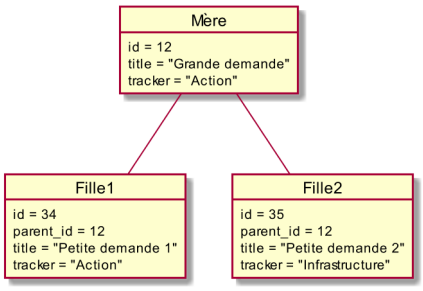


Tableau 10. Synthèse de la création des demandes

Acteurs	Tous les rôles
Entrées	Une demande non formalisée à propos de fonctionnalités, réalisations, dysfonctionnements, aide ou action.
Sortie	Une demande en statut `New` et - tracker `Dev` pour une réalisation - tracker `Feature` pour une fonctionnalité, - tracker `Bug` pour un dysfonctionnement, - tracker `Support` pour de l'aide, - tracker `Action` sinon. Plusieurs demandes filles éventuellement.
Procédures	Suivre les actions préconisées suivant la nature de la demande - la section intitulée « Processus de création d'une demande pour une fonctionnalité » - la section intitulée « Processus de création d'une demande pour une réalisation » - la section intitulée « Processus de création d'une déclaration de dysfonctionnement » - la section intitulée « Processus de création d'une demande de support » - la section intitulée « Processus de création d'une demande d'action »

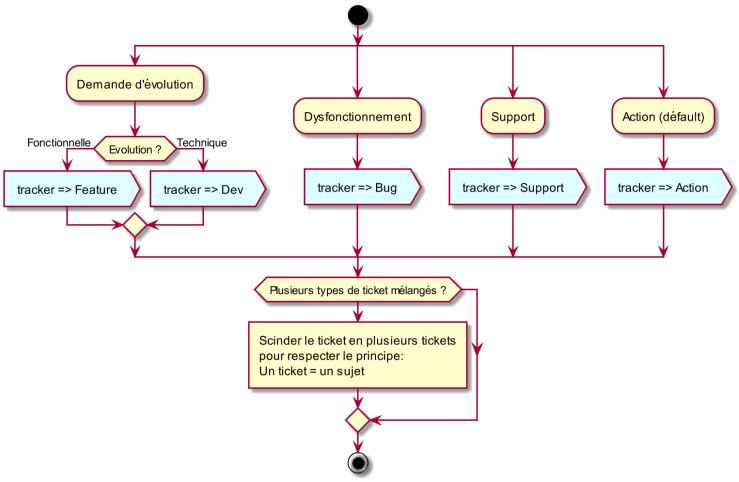


Figure 10. Processus de création de demande

Processus de création d'une demande pour une fonctionnalité

Objectifs	Fournir les éléments pour vérifier la cohérence, la valeur ajoutée, l'effort requis de la demande de fonctionnalité.
Obligatoire	Utiliser le tracker `Feature`, Indiquer la référence fonctionnelle *extérieure* (Règle de gestion, exigence, Spécification, autres `Feature`). **ou** Décrire - la fonctionnalité (ce que l'on veut), - sa finalité (à quoi cela va servir), - ses utilisateurs (qui va l'utiliser)
Recommandé	Décrire ou référencer la description des éléments permettant de valider la solution au plus tôt (use cases, scénarios, situations…).

Processus de création d'une demande pour une réalisation

Objectifs	Fournir les éléments pour vérifier la pertinence, la faisabilité, la cohérence, la valeur ajoutée, l'effort requis de la demande.
Obligatoire	Utiliser le tracker `Dev`, Décrire - La réalisation (ce que l'on veut), - Sa finalité (à quoi cela va servir), - La référence fonctionnelle (Règle de gestion, exigence, Spécification, `Feature`) si elle existe.
Recommandé	Fournir - les références techniques, préconisations - les attendus (maquettes, …)

Processus de création d'une déclaration de dysfonctionnement

Objectifs	Fournir les éléments pour vérifier l'existence, la reproductibilité, la gravité du dysfonctionnement.
Obligatoire	Utiliser le tracker `Bug`, Décrire - Le défaut ou le dysfonctionnement constaté (ce qui est anormal), - La fréquence d'apparition (aléatoire, rare, fréquente, systèmatique), - Les étapes pour le reproduire, - La version sur laquelle le dysfonctionnement est constaté.
Recommandé	Fournir - les références techniques, préconisations, - la référence fonctionnelle (Règle de gestion, exigence, Spécification, `Feature`) - Le comportement ou l'état jugé normal

Processus de création d'une demande de support

Objectifs	Fournir les éléments pour faciliter l'aide sollicitée, favoriser la pertinence de l'information à fournir, donner le renseignement juste.
Obligatoire	Utiliser le tracker `Support`, Décrire - la demande (la motivation de la demande de support) - le contexte (cas d'usage, écrans, matériel utilisé, etc..) - La version sur laquelle le dysfonctionnement est constaté. - sa finalité (à quoi cela va servir),
Recommandé	Fournir les ressources déjà exploitées mais insatisfaisantes.

Processus de création d'une demande d'action

Objectifs	Fournir les éléments pour apprécier la pertinence, la cohérence, la légitimité, l'urgence de l'action.
Obligatoire	Utiliser le tracker `Action`, Décrire - le constat (dans quel contexte l'action s'inscrit) - l'action (ce que l'on veut obtenir), - sa finalité (à quoi cela va servir),
Recommandé	Fournir les informations de légitimité si cela peut faciliter la prise de décision. Si l'action est complexe, en donner les étapes. Si elle est urgente, le préciser avec la propriété *priority*

Processus de revue des demandes à évaluer

Les demandes **ne doivent pas** rester indéfiniment en l'état `New`. Leur évolution se fait au travers d'une revue.

Tableau 11. Synthèse de la revue des demandes

Acteurs	Elle **doit** être menée par un manager. La participation des *développeurs* est **recommandée**. Celle des *rapporteurs* est **possible**.
Entrées	Une liste de demandes en status `New`
Sortie	Plus aucune demande en statut `New`. Toutes les demandes en sortie sont soit - status `Rejected` si irrecevable, - statut `Feedback` si besoin de complément d'information pour évaluer, - tracker `Support` si du à une incompréhension, - statut `Accepted` si acceptée mais planifiée ultérieurement.
Procédure	Effectuer les actions préconisée en fonction des situations. cf Tableau 12, « Situations et actions préconisées pour la revue des demandes »
Outillage	**Filtre de demande enregistré**: - Filtrer sur le status `New`, - Ordonner par ordre décroissant de création

	Tache récurrente: - Créer une demande de type action *Revue des demandes entrantes*, - Associer récurrence (au minimum 1/mois).

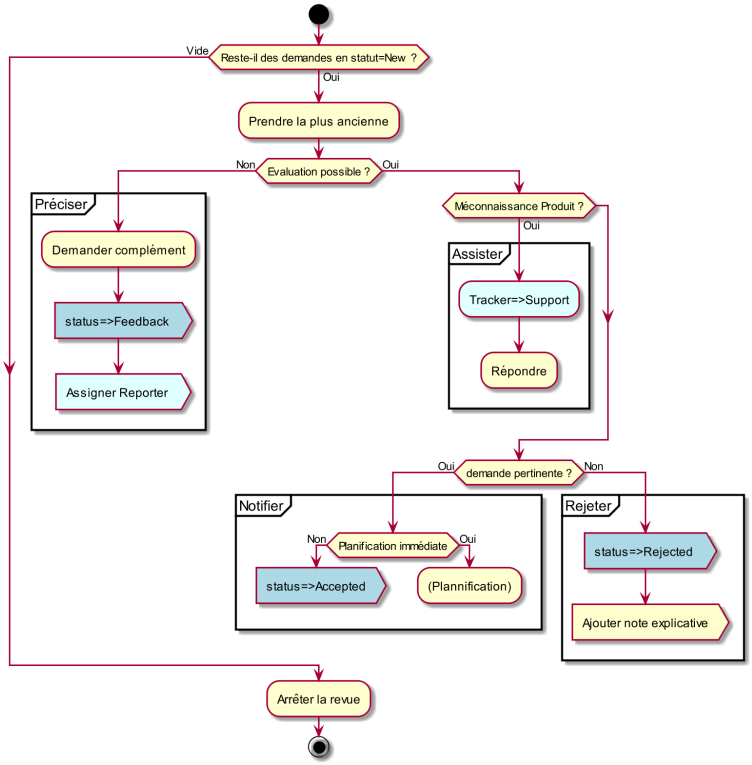


Figure 11. Processus de la revue des nouvelles demandes

Tableau 12. Situations et actions préconisées pour la revue des demandes

Situation	Action
Il manque des informations pour évaluer	**Obligations** Passer le statut à `Feedback`, Assigner la personne la plus apte à répondre. **Recommandations** Si l'information est nécessaire pour faire un choix important, mettre une date limite (*Due date*)
La demande n'est pas pertinente ou obsolète.	**Obligations** Passer le statut à `Rejected` **Recommandations** Ajouter une note explicative.
La demande est due à une incompréhension ou une méconnaissance du produit.	**Obligations** Passer le tracker à `Support` **Recommandations** Répondre dans la foulée, passer le statut à `Feedback` et assigner au demandeur. **Possibilités** Alimenter une FAQ.
La demande est recevable et planifiée ultérieurement.	**Obligations** Passer le statut à `Accepted`.

Situation	Action
La demande est recevable et planifiable immédiatement.	**Obligations** Suivre le processus de planification.

Processus de planification des prises en charge des demandes

La planification consiste à définir, pour chaque demande acceptée, les conditions de prise en charge par un développeur pour la résoudre. C'est un processus fondamental pour répartir au mieux le traitement des demandes. Les décisions reposent sur des critères comme la charge envisagée, la criticité, les priorités ou la cohérence globale.

La planification d'une demande **doit** reposer sur *tout ou partie* des propriétés suivantes

- la version du produit ou de l'itération
- la date prévisionnelle ou attendue (Due Date)
- la criticité de la demande (Low, Normal, High, Urgent, Immediate)
- l'assignation d'un développeur

Mode de consommation des demandes

Si nous nous plaçons du point de vue du développeur, la question revient à identifier la prochaine demande à prendre en charge parmi toutes celles planifiées sans trop se poser de questions. Il n'existe pas de statut *planifié*. L'état planifié repose donc sur une convention propre au projet.

Il existe deux modes de *consommation* des demandes :

- un mode *poussé* où un développeur se voit assigné une demande (par un manager, en cours de réunion, après discussion, …)
- un mode *tiré* où un développeur s'assigne de lui-même la demande.

Le mode *poussé* est simple à mettre en oeuvre à priori, il suffit au développeur de sélectionner les demandes qui lui sont assignés et les trier selon un critère à convenir.

Le mode *tiré* n'est efficace que si le produit est versionné et que les versions ont une date de fin. Il suffit au développeur d'aller sur la roadmap, de rebondir sur la liste des demandes à traiter et de sélectionner la demande à traiter selon son expertise.

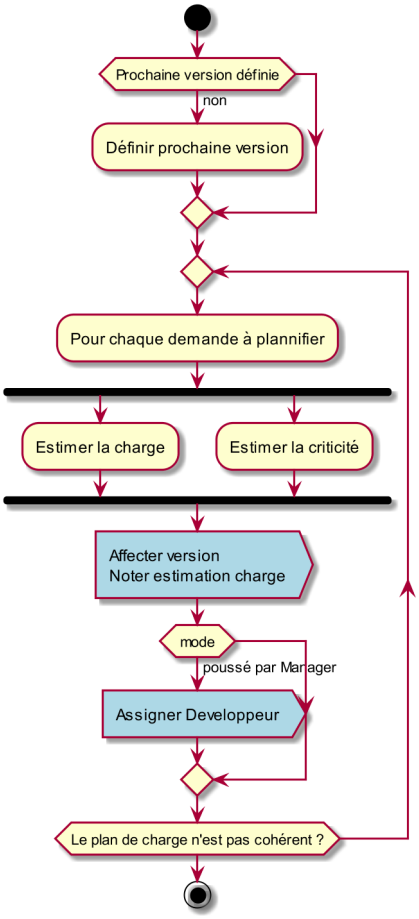
Nous ne saurions encore fortement recommander l'usage des versions pour les artefacts (produits) de votre projet. Les choses sont beaucoup plus simples et explicites :

```
Est planifié ce qui est affecté à une version.
Est à prendre en charge ce qui est dans la version en cours de réalisation.
```

Tableau 13. Synthèse de la planification des demandes

Acteurs	La planification **doit** être menée par le manager, la participation des développeurs est **recommandée**, celle des rapporteurs est **possible**.
Entrées	A la suite de la revue des demandes ou

	la liste de demandes en statut `Accepted`. Une convention d'usage établie sur le mode de consommation des demandes.
Sortie	Toute les demandes sont *planifiées*. C'est à dire que les développeurs savent comment sélectionner les demandes à prendre en charge. Il est **recommandé** que chaque demande soit associée à une version. La convention d'usage **peut** être mise à jour
Procédure	**Obligatoire** (Re)estimer la criticité, Estimer la charge, En mode *poussé*, assigner la demande à un développeur. En mode *tiré*, associer la demande à une version OU une date d'échéance **Recommandé** Vérifier que le produit est versionnée, créer la version, *et la suivante* au besoin. Affecter chaque demande à une version, la prochaine ou la suivante. Vérifier la cohérence du plan de charge associée à la version, Estimer la charge avec le développeur, **tests inclus**.



Processus de prise en charge et résolution des demandes

Le processus de prise en charge et résolution consiste à effectuer les opérations nécessaires à la résolution de la demande.

Résolution et définition du Fini

Derrière cette évidence se cachent des notions bien plus subtiles. Quand peut-on dire qu'une demande est résolue ?

En fait c'est encore une question de convention. En agilité on appelle cela la *définition de fini* [11]. Est-ce quand le code fonctionne, que la pièce est faite ? Ou est-ce quand les tests sont faits, la documentation écrite ? A chaque projet de (re)définir le *Fini*.

Exemple 1. Exemple de définition de fini

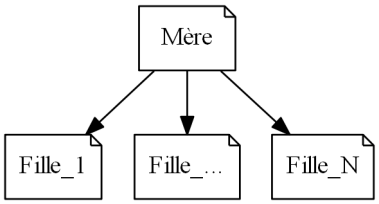
Pour un développement logiciel, il est raisonnable de penser que la demande est finie quand

- ❏ Le code est réalisé.
- ❏ Le code peut être exécuté
- ❏ Les tests unitaires sont réalisés.
- ❏ Les tests unitaires passent tous.
- ❏ Le taux de couverture des tests [12] est satisfaisant (ex. supérieur ou égale à 60%).
- ❏ La documentation est modifiée.

Le pourcentage de réalisé

Le développeur **doit** mettre à jour le pourcentage de réalisé *dès que cet indicateur est révélateur d'une situation*. Quand la demande est résolue, le pourcentage de réalisé **doit** être 100%.

Il ne s'agit pas de "pointer", mais dès que la durée de la résolution de la demande dépasse quelques jours, indiquer le pourcentage de réalisé permet de mieux percevoir la situation globale, notamment à traves le mécanisme de décomposition d'une demande $D^{Mère}$ en N demandes filles $D^{fille}{}_P$ avec $P=1..N$.



Le pourcentage $P^{Mère}$ de la demande mère est alors déduit automatiquement de celui des filles $P^{Fille}{}_P$ selon la formule:

$$P^{Mère} = \sum_{i=0}^{N} \frac{P_i^{Fille}}{N}$$

La mise à jour du pourcentage de réalisé d'une demande fille est automatiquement répercuté sur le pourcentage de la demande mère.

[11] cf. http://www.aubryconseil.com/post/Definition-de-fini-multi-niveaux

[12] Le taux de couverture se calcule en estimant le nombre de lignes parcouru lors des tests unitaires sur le nombre total de lignes de codes. Un taux de couverture de 60% signifie qu'il reste 40% de lignes de code qui ne sont pas testés. le taux de couverture est un indicateur parmi d'autre, il ne peut à lui-seul être synonyme de qualité.

> Ainsi dès que toutes les demandes filles sont réalisées à 100%, la demande mère est considéré également résolue à 100%.

Résolution et validité de la solution

Le développeur **doit** vérifier que la solution est valide dans son domaine de responsabilité et en accord avec les ressources accordées [13].

Le développeur **doit** s'assurer que la solution proposée est validable, c'est à dire qu'il a vérifié, *dans les limites de son domaine de compétence*, qu'elle répond aux attentes.

Le développeur peut travailler avec le validateur pour s'assurer, **le plus en amont possible**, que la future solution répondra aux attentes.

Nombre de demandes en parallèle

Le nombre de demandes traitées en parallèle par un même développeur doit être limité pour rester crédible. Si une demande en cours de prise en charge est stoppée à cause d'une attente, faire évoluer le statut de la demande !

Tableau 14. Synthèse de la prise en charge et résolution des demandes

Acteurs	La prise en charge et la résolution **doivent** être menée par le développeur. Les rapporteurs **peuvent** participer.
Entrées	Une demande sélectionnée : - En mode *poussé* dans la liste des demandes assignées. - En mode *tiré* dans la liste des demandes associées à la prochaine version. Une demande passée en `WaitingForValidation` mais invalidée.
Sortie	Toute les demandes sont *résolues* et finies selon la convention établie (définition de fini)
Procédure	**Obligatoire** Passer le statut à `InProgress`, Suivre la définition de finie. Quand la demande est résolue: - Passer le statut à `Resolved`, - Passer le pourcentage de réalisé à 100% **Recommandé** Ne pas prendre en charge plus de 3 demandes en parallèle.

Processus de validation des solutions

La validation consiste à vérifier que la solution apportée rempli les attentes. C'est une étape importante qui officialise la solution et clôture la démarche initiée par la demande .

> Cet aspect est éminement lié à la nature de votre projet, sa culture, ses exigences et à la méthodologie suivie. Aussi n'entrerons nous pas trop, ici, dans le menu détail.

[13] A l'impossible nul n'est tenu

Néanmoins il reste des points fondamentaux à respecter :

- les attentes doivent être vérifiables et documentées
- La vérification doit être menée sous la responsabilité d'un acteur dont la légitimité est reconnue.
- Tout écart à l'attente doit lui même être constaté et documenté.

Attentes vérifiables et documentées

Pour être vérifiables, les attentes **doivent** doivent être objectives, mesurables, réalistes et compréhensibles. Elles doivent être documentées au plus tôt.

Elles peuvent se construire au fur et à mesure par des échanges entre acteurs. A la fin, elles **doivent** être *écrites* et *acceptées* par l'ensemble des acteurs dans le référentiel utilisé (Spécifications Détaillées, Exigences, …).

Vérification et légitimité

La validation d'une solution est réalisée par un validateur. Sa responsabilité est s'assurer que la solution répond aux attentes, qu'elle ne génère pas d'autres problèmes, qu'elle est cohérente avec le reste du système.

L'identification du validateur doit être explicite [14] et si possible être décrite dans la demande initiale.

Un validateur ne doit pas valider au delà du domaine de compétence ou de responsabilité. Il doit alors faire valider les autres aspects par des experts du domaine.

Par exemple, si la solution est la mise en oeuvre d'une fonctionnalité qui implique des compétences scientifiques très poussées, seul un expert ayant ces compétences peut valider la solution.

Souvent, le rapporteur de la demande est aussi le validateur.

Si la validation impliques plusieurs acteurs et/ou opérations lourdes, un conseil:

- Ajouter autant de sous-demandes de *test* avec le tracker `Test` que d'acteurs à la demande de validation.
- Jouer sur les filiations pour ordonnancer les tests.

Après, il faut rester pragmatique et user de bon sens, à condition de l'écrire ! Par exemple :

- quand les spécifications existent, que la demande est un `Bug`, que l'impact est mineur, le développeur peut lui même valider la correction.
- Si c'est une demande de fonctionnalité métier, le validateur est un expert métier.

Défaut, écarts et documentation

Quand le validateur constate un défaut, un écart, une non-conformité, il la documente de la façon la plus objective, factuelle, compréhensible possible.

Toutefois, il est possible de fluidifier la dynamique validation → constat → correction en travaillant de concert avec le développeur en amont.

[14] Cela est normalement décrit dans le Plan Qualité Projet si vous en avez un.

Tableau 15. Synthèse de la prise de la validation de la solution

Action	La validation **doit** opérée par un validateur, qui peut être le rapporteur. Les développeurs **peuvent** participer.
Entrées	Une demande sélectionnée en statut `Resolved`
Sortie	La demande est clôturée.
Procédure	**Obligatoire** Passer le statut à `Closed` quand la validation est effective, Quand la solution n'est pas valide: - Créer une note, - Assigner le développeur **Recommandé** Faire la validation conjointement avec le développeur, Eclater les validations lourdes en plusieurs sous-demandes Préférer les interactions que les échanges de demandes.

Principe des revues

Une revue consiste à balayer les demandes et identifier celles nécessitant une décision. Elle est plus efficace quand elle est courte et orientée en utilisant un filtre. Quasiment tous les processus listés ici pourraient démarrer par une revue.

C'est aux acteurs du projets de créer leurs propres filtres pour les aider à piloter, en combinant les différents filtres [15].

Les revues usuelles sont

- revue des nouvelles demandes à évaluer (statut `New`)
- revue des demandes en sommeil (dernière date de mise à jour supérieure à 1 mois)
- revue de la version en préparation (filtre sur la version)
- revue des demandes en retard (date d'échéance dépassée de *n* jours)

Les filtres les plus pertinents ou fréquemment utilisés peuvent être sauvegardés.

Filtres consommables

La forme la plus efficace de revue est celle qui sort la demande de la liste après examen. Pour cela il faut respecter quelques principes

- toute demande examinée **doit** faire l'objet d'une décision
- toute décision **devrait** rendre la demande impossible à sélectionner avec le filtre

Par exemple la revue des nouvelles demandes à évaluer:

1. Sélection des demandes par filtrage sur le statut `New`.
2. Après examen, la demande change de statut.

[15] Certain plugins comme http://www.redmine.org/plugins/computed_custom_field doivent pouvoir apporter une certaine souplesse

3. La demande sort de la liste

Usage des versions

L'usage intensif des filtres dans les revues est le meilleur argument pour l'utilisation des roadmaps et des versions. L'attaque par l'angle des versions permet d'avoir des listes de demandes courtes et rapides à examiner et évite de se disperser.

Fréquence des revues

La fréquence des revues doit être adaptée en fonction de l'activité du projet et du volume des demandes.

Le principe de pragmatisme doit être respecté et la revue devrait durer 1h au plus. Les revues se font souvent successivement en réunion. Si vous passez 5 minutes en moyenne par demande, dès qu'une douzaine de demandes sont en attente, déclenchez une revue.

En régime de croisière, une revue quotidienne est nécessaire. Dès que le régime baisse, diminuer la fréquence. Tant que le projet est actif, ne descendez pas en dessous de 1 par mois.

Mieux vaut des réunions courtes et fréquentes que rares et longues

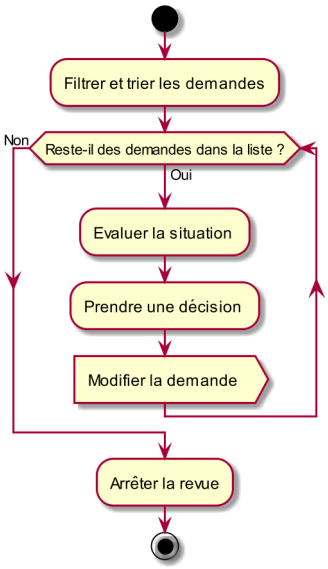


Figure 12. Processus de la revue des demandes

3.5. Templates

```
Ceci sont des exemples de templates.
Tous les retours ou propositions sont bonnes à prendre.
```

Template de création de demande de fonctionnalité

Ceci est un gabarit (template) fourni pour votre projet.
Vous pouvez l'utiliser tel quel ou l'adapter à vos besoins.

Objectifs	Fournir les éléments pour vérifier la cohérence, la valeur ajoutée, l'effort requis de la demande de fonctionnalité.

Obligatoire	Utiliser le tracker `Feature`, Indiquer la référence fonctionnelle *extérieure* (Règle de gestion, exigence, Spécification, autres `Feature`). **ou** Décrire - la fonctionnalité (ce que l'on veut), - sa finalité (à quoi cela va servir), - ses utilisateurs (qui va l'utiliser)
Recommandé	Décrire ou référencer la description des éléments permettant de valider la solution au plus tôt (use cases, scénarios, situations…).

Procédure

Pour vous aider, voici une checklist à suivre.

Checklist de création d'une demande de fonctionnalité

Préalable

- ❏ Le tracker est bien *Feature* ?
- ❏ Le status est bien *New* ?
- ❏ Le titre est explicite ?

Description

- ❏ La *fonctionnalité* est détaillée ?
- ❏ Les *finalités* sont décrites ?
- ❏ Les *utilisateurs* sont identifiés ?
- ❏ Les *Règles de gestion* ou *Exigences*, ou leurs références, sont décrites ?
- ❏ La présentation est lisible et rapide à parcourir ?

Propriétés

- ❏ La priorité est indiquée ?
- ❏ Les bons acteurs sont en *watchers* ?

Le template suivant peut-être copié collé dans le champs *Description* de la demande.

Template prêt-à-copier pour le champs description Redmine.

```
h2. Fonctionnalité
//Détailler la fonctionnalité (quoi obtenir)

h2. Finalités
// Détailler les finalités de cette fonctionnalité (pourquoi ?)

h2. Utilisateurs
```

```
// Détailler les utilisateurs qui utiliseront la fonctionnalité (pour qui ?)

h2. Règles de gestion
//  Décrire les règles de gestion ou exigences associées à la fonctionnalité
// ou les références
// (selon quels principes ?)
```

Exemple de mise en oeuvre

Exemple de création d'une demande de fonctionnalité pour redmine

- Titre: *Afficher la liste des jeux de données d'observation disponibles*
- Description :

```
h2. Fonctionnalité

Afficher la liste des jeux de donnée disponibles avec des informations essentielles telles que

* le nom,
* l'origine,
* les dates de débute et dates de fin d'observation
* le domaine (solaire, cosmo, planéto)
* le nombre d'observation

h2. Finalités

* Avoir une liste exhaustive des jeux de données,
* Pouvoir rapidement localiser les jeux de données souhaités selon plusieurs critères,
* Pouvoir accéder aux jeux de données en soi.

h2. Utilisateurs

Tout utilisateur

h2. Règles de gestion

La liste peut être triée par

* nom
* dates
* domaine

A partir de cette liste l'utilisateur doit pouvoir accéder à la vue de détail du jeux de
 données.
Si le détail du jeu de données n'est pas accessible, il n'est pas affiché.
```

- criticité: normale