AMÉLIORATION DE L‘EFFICACITÉ ET DU TRAVAIL COLLABORATIF

La technologie des flux de travail dans le domaine de l'architecture

Zheng LYU n.21164

SOMMAIRE

1.Introduction :

J'ai travaillé pendant 4 ans au sein d'un cabinet d'architecture en Chine, spécialisé dans la conception de bâtiments avec une équipe d'environ 100 personnes. Ayant obtenu mon diplôme d'ingénieur, j'ai commencé en tant que dessinateur au niveau le plus bas. Au cours de ces quatre années, j'ai progressé de dessinateur à chef de projet, assumant des responsabilités telles que le traitement des images, la création de dessins, la conception indépendante, la formation des stagiaires, la planification du travail, ainsi que la communication avec le client.

J'ai occupé pratiquement tous les postes au sein de l'entreprise, ce qui m'a permis d'acquérir une compréhension claire du processus de conception et de la structure organisationnelle de l'entreprise.

En tant que cabinet spécialisé dans la conception, toutes nos activités sont axées sur les besoins du client. Notre travail se divise principalement en trois parties. Tout d'abord, nous aidons le client à analyser le site et formulons des hypothèses en fonction du plan local d'urbanisme, telles que la maximisation des rendements, la création de paysages attrayants, ou des agencements optimaux pour le confort. Ensuite, nous commençons à concevoir et à dessiner en fonction des décisions finales, y compris la forme du bâtiment, les plans, les fonctionnalités, le style et les modèles 3D. Enfin, nous soumettons le résultat final à des entreprises spécialisées en dessin technique, tout en maintenant une communication constante pour collaborer activement aux modifications et garantir que les choix esthétiques des architectes ne soient pas compromis par des problèmes techniques.

Afin d'assurer le bon fonctionnement de l'entreprise de 100 employés, la société Gee utilise une structure organisationnelle basée sur un système de groupes combiné à un système de niveaux pour classer les employés.

Les 100 employés sont répartis en 8 groupes, chacun ayant un classement en termes de niveaux : A1, A2, B1, B2, C1, C2, D1, D2, etc.

Les niveaux plus élevés représentent une plus grande capacité, et par conséquent, un salaire plus élevé. Par exemple, un chef de projet de niveau B2 peut être responsable de la planification d'un projet et du réajustement du personnel, ce qui signifie qu'il est également compétent pour effectuer toutes les tâches de niveaux inférieurs, tels que C1-2, D1-4.

Chaque niveau a ses propres domaines d'expertise. Par exemple, le groupe H, auquel j'appartiens, a une forte demande pour la conception de dessins, ce qui se traduit par une demande plus importante pour les niveaux C1 et C2. En revanche, le groupe D, chargé des compétitions, aura besoin de personnel de niveaux D3-4 spécialisé dans les rendus et la modélisation.

Comment mener un projet :

Au sein du groupe H, le processus de conception pour un projet résidentiel peut être divisé en quatre phases principales.

La première phase consiste en une analyse approfondie de la recherche préliminaire. Pendant cette étape, nous nous penchons sur la partie urbaine du site cible, en nous familiarisant avec le plan local d'urbanisme (PLU) et en effectuant des visites sur site. Il s'agit de trouver des points d'entrée pertinents et de réaliser une première simulation d'ensoleillement.

La deuxième phase est celle des hypothèses. Au cours de cette étape, des réunions et des discussions sont organisées, plusieurs essais sont effectués,et les progrès sont régulièrement présentés au client jusqu'à la finalisation complète de la conception.

La troisième phase est dédiée à la finalisation des dessins. À ce stade, tous les dessins sont achevés et soumis aux professionnels spécialisés dans la structure, l'électricité, le paysage et l'intérieur.

La quatrième phase est la maintenance post-projet. Durant cette période, nous restons en contact avec les différentes spécialités pour leur fournir un soutien technique. Nous surveillons également de près les progrès des autres entreprises spécialisées pour garantir la qualité de la partie architecturale du projet. Cette phase marque également le début de la construction, nécessitant une surveillance continue sur site.

Pour un projet complet, la durée de conception varie généralement de 3 à 6 mois, ce qui signifie que les phases un à trois doivent être achevées dans ce délai.

En ce qui me concerne, prenons un exemple concret. Imaginons que je sois de niveau B2 et que, en janvier, j'ai débuté un projet P1 avec trois collègues. En mars, le projet a atteint la troisième phase, la majeure partie de mon travail est terminée, mais le projet n'est pas encore achevé. À ce stade, j'ai la possibilité de rejoindre un autre projet P2 qui se trouve actuellement dans la phase de dessin, en assumant le rôle de niveau C1. Ainsi, je suis impliqué simultanément dans deux projets avec des responsabilités différentes.

Dans un autre contexte, un projet antérieur, P3, est en phase de construction, et en tant que responsable, je suis chargé de la communication avec diverses spécialités. De plus, le projet P4, sur le point de commencer, me verra participer en tant que niveau D3.

Une fois familiarisé avec l'ensemble du processus, les architectes de niveau CD peuvent choisir de ne pas rejoindre un projet dès le début. Il leur suffit de se familiariser avec le projet avant le début des travaux.
Cette organisation du travail et cette structure semblent solides, efficaces, et chaque étape est clairement définie. Cependant, ma réalité professionnelle quotidienne se résume ainsi : chaque jour, je me rends au bureau à 9h30, ouvre mes logiciels de communication pour recevoir les retours sur le projet P3, que je redistribue ensuite aux personnes concernées. À ce moment-là, les membres de l'équipe du projet P1 me

signalent un problème insoluble, nécessitant l'organisation d'une réunion de discussion. En même temps, la date limite de remise des dessins pour le projet P2 approche, et je dois intensifier les travaux de dessin. De 14h à 17h, une réunion de lancement du projet P2 est prévue, et ainsi de suite... Pour moi, de mon arrivée au bureau jusqu'à 17h, je n'ai aucun moment pour réfléchir aux aspects architecturaux du projet. C'est une mécanique constante d'accomplissement des tâches immédiates. Après 17h, lorsque je suis enfin seul sans interruption, je peux commencer à travailler sur les tâches du projet P3. Lorsque ma journée de travail se termine, j'ai déjà travaillé pendant 14 heures.

Après avoir vécu cette routine, je peux résumer mon travail d'architecte en trois grandes parties.

Première partie : La première partie de mon travail implique des échanges avec différents architectes, responsables de domaines spécialisés, ainsi que mes collègues. Les discussions portent principalement sur la résolution des problèmes rencontrés dans un projet donné. Ces problèmes ne sont généralement pas très complexes, par exemple, le département de la structure propose l'ajout de piliers à certains endroits, mais cela aurait un impact sur l'espace intérieur. Je dois alors contacter le collègue responsable des plans et demander à l'ingénieur de fournir ses plans. Ensuite, mon collègue tente de modifier les plans en collaboration avec l'ingénieur pour trouver une solution. Une fois cela fait, je dois archiver les plans et les envoyer à tous les membres du projet, en les informant des modifications apportées à certaines zones qui nécessitent une mise à jour rapide. Mon rôle principal dans ce processus est la communication et l'organisation, en veillant à ce que tous aient accès aux versions les plus récentes des plans.

Deuxième partie : La deuxième partie concerne la phase de dessin. Le travail de dessin de l'architecte comprend trois types : déterminer la version finale du concept de conception, produire les plans finaux à

soumettre, et effectuer de petites modifications après des échanges avec des entreprises spécialisées. Cette partie du travail est généralement ce sur quoi je me concentre après 17 heures, car elle ne nécessite pas de communication avec d'autres personnes, me permettant de travailler dans un environnement calme. Enfin, la dernière partie concerne les discussions et les échanges. Lorsque je suis confronté à des problèmes que je ne peux résoudre seul, comme dans l'exemple précédent où des ajustements aux piliers étaient nécessaires, ces petites modifications nécessitent généralement une attention particulière.

Troisième partie : La troisième partie du travail concerne l'organisation, la gestion et la planification du calendrier. Lorsque je travaille sur plus de deux projets en même temps, mes pensées passent fréquemment d'un projet à l'autre. Cela rend difficile de savoir sur quel projet je suis en train de dessiner à un moment donné, et quelle est la version la plus récente du plan. Cela demande beaucoup de temps pour suivre et comprendre l'évolution d'un projet spécifique.

Cette situation me plonge souvent dans une période de travail intense et prolongée, d'autant plus avec les déplacements professionnels. En fin de compte, je ne dispose pratiquement que d'un jour de repos par semaine. Le travail intensif limite mes activités quotidiennes normales, car tout est centré sur le travail. Les résultats de ce travail intensif se traduisent par le fait que la plupart des conceptions architecturales sont inspirées par des pratiques existantes, car cela permet un achèvement plus rapide.

Résumé : J'ai pris conscience qu'en tant qu'architecte travaillant au sein d'une entreprise, deux compétences essentielles sont nécessaires : les "compétences en gestion de projet" et les "compétences en conception". La deuxième compétence, liée au temps de conception de l'architecte, est facile à comprendre. La première compétence concerne l'organisation du travail et la communication, que j'ai subdivisée en 5 catégories : la capacité à communiquer de manière globale entre différentes spécialités,

la capacité à collaborer sur les détails lors de la conception indépendante, la capacité à organiser les tâches de travail, la capacité à organiser les documents et la capacité à suivre en temps réel l'avancement du projet.
Y a-t-il une technologie qui, par des moyens scientifiques, pourrait aider les architectes à développer ces "compétences en gestion de projet" en utilisant des outils et des techniques, leur permettant ainsi de se concentrer davantage sur le processus de conception ?

Ce mémoire se concentre principalement sur cette question, en intégrant la structure organisationnelle et les flux de travail de mon travail chez GEE. J'essaie de trouver une solution pour réduire le temps que les architectes consacrent aux "compétences en gestion de projet".

J'ai décidé d'explorer un domaine en particulier : la technologie des flux de travail.

2. Etat de l’art

2.1 État de l'art de la technologie des flux de travail

La technologie des flux de travail a ses origines dans la recherche sur la technologie de l'automatisation de bureau (OA) des années 70, mais en raison des contraintes de l'époque, le développement de la technologie des flux de travail a progressé lentement. Ce n'est qu'à la fin des années 80, avec le développement rapide de domaines tels que le traitement numérique d'images et la messagerie électronique, qu'ont émergé progressivement des systèmes de gestion des flux de travail de grande envergure. À partir des années 90, le développement rapide de la technologie Internet a accéléré la recherche et le développement des systèmes de gestion des flux de travail.

Aujourd'hui, après plus de 20 ans de développement, bien que la technologie des flux de travail ait atteint des niveaux élevés de réussite, la définition des flux de travail n'est toujours pas claire et uniforme. Différents chercheurs et fournisseurs de produits de flux de travail ont donné des définitions différentes du concept de flux de travail, chacun adoptant une perspective particulière. Cet article présente quelques-unes des définitions représentatives.

La définition du flux de travail donnée par Dimitrlos Georgakopoulos et Mark Hornick est la suivante : un flux de travail consiste à organiser de manière ordonnée un ensemble de tâches selon des conditions spécifiques afin d'accomplir un processus métier. Cette définition englobe la spécification des conditions de déclenchement et de l'ordre de déclenchement des tâches. Chaque tâche peut être accomplie par une ou plusieurs personnes, ou encore par un ou plusieurs systèmes logiciels.

[Référence : Dimitrlos Georgakopoulos, Mark Hornick. Aperçu de la gestion des flux de travail : de la modélisation des processus à l'infrastructure d'automatisation des flux de travail [J], Distributed and Parallel Databases, 1995, 3(2) : P119-P153.]

Le centre de recherche IBM Almaden définit le flux de travail comme une représentation informatisée des processus métier. Il permet de définir les paramètres nécessaires à l'ensemble du processus métier, comprenant la définition de chaque étape, l'ordre d'exécution, les conditions requises pour l'exécution des tâches, ainsi que la personne chargée de chaque tâche au cours du processus. [Référence : Mohan C, Voorhoeve M. Tendances récentes dans les produits, normes et recherches en gestion des flux de travail [J]. Système d'information, 1997, 95(4) : p127-p131.]

Dans les années 90, la création de la Workflow Management Coalition (WfMC) marquait le développement progressif de la technologie des flux de travail. En tant qu'organisation de normalisation pour la gestion des flux de travail, la WfMC définit le flux de travail comme suit : le flux de travail concerne l'automatisation des procédures où des documents, des informations ou des tâches sont transmis entre les participants selon un ensemble défini de règles pour atteindre ou contribuer à un objectif commercial global. Le workflow est souvent associé à la réingénierie des processus métier (Business Process Re- engineering, BPR), qui s'intéresse à l'évaluation, à l'analyse, à la modélisation, à la définition et à la mise en œuvre opérationnelle ultérieure des processus métier principaux d'une organisation (ou autre entité commerciale).

En résumé, le workflow est une forme de règles qui doit être élaborée et mise en œuvre avant le début du travail, en fonction du processus métier, et ces règles visent à maximiser l'automatisation pour atteindre une efficacité maximale. [Référence : David Hollingsworth. The Workflow Reference Model [M]. Workflow Management Coalition, 1995: 5-6]

Pour créer un flux de travail, trois étapes sont nécessaires au plus haut niveau : définir les étapes du flux de travail, contrôler la phase d'exécution et la phase d'utilisation par l'utilisateur.

Tout d'abord, avant de créer le flux de travail, il est nécessaire de définir le flux de travail et de comprendre comment le travail est effectué,

puis d'optimiser en utilisant des applications et des outils informatiques. Tout travail peut être défini par un flux de travail. Par exemple, pour préparer du café, une personne doit utiliser des grains de café, une meuleuse, une machine à café et une tasse. Le processus peut être simplifié en une séquence linéaire : démarrage - prendre les grains de café - mettre les grains de café dans la meuleuse - retirer la poudre de café - mettre la poudre de café dans la machine à café - placer la tasse pour recevoir le café - terminer. Ainsi, un flux de travail simple est défini. Ensuite, une optimisation peut être réalisée, comme la substitution d'outils pour certaines étapes ou la réduction de la participation humaine. Supposons que l'on trouve une machine à café automatique capable de moudre automatiquement le café, d'ajouter la poudre de café à la machine et disposant également d'un grand réservoir pour stocker les grains de café. Les étapes ci-dessus peuvent alors être simplifiées en introduisant le nouvel outil : démarrage----vérifier le stock de grains de café (s'il y a du stock, passer à l'étape suivante, sinon mettre les grains de café dans la meuleuse)---placer la tasse pour recevoir le café---terminer. Ainsi, en observant, on obtient un modèle simple, et en analysant et en recherchant des outils, on optimise le processus, réduisant la participation humaine et maximisant l'efficacité.

La deuxième étape consiste à contrôler la phase d'exécution, qui se déroule dans un contexte de travail réel, en faisant fonctionner ce flux de travail et en gérant et ajustant le processus. Prenons l'exemple du café mentionné précédemment : si le gestionnaire du flux de travail souhaite préparer du café pour 10 personnes, il ajustera ce modèle de flux de travail en fonction de ses besoins réels, par exemple en augmentant le stock de café en poudre, en introduisant un outil d'alerte pour une faible quantité de café en poudre, en introduisant des gobelets en papier, afin de réduire l'étape consistant à mettre la tasse pour recevoir le café.

La troisième étape est la phase d'utilisation, où le modèle de flux de travail commence officiellement à travailler pour l'utilisateur et fournit

constamment des commentaires à la deuxième étape, permettant des ajustements continus jusqu'à ce que le modèle final soit achevé. Dans la littérature de référence, la figure 1 - Caractéristiques du système de flux de travail exprime cette relation.

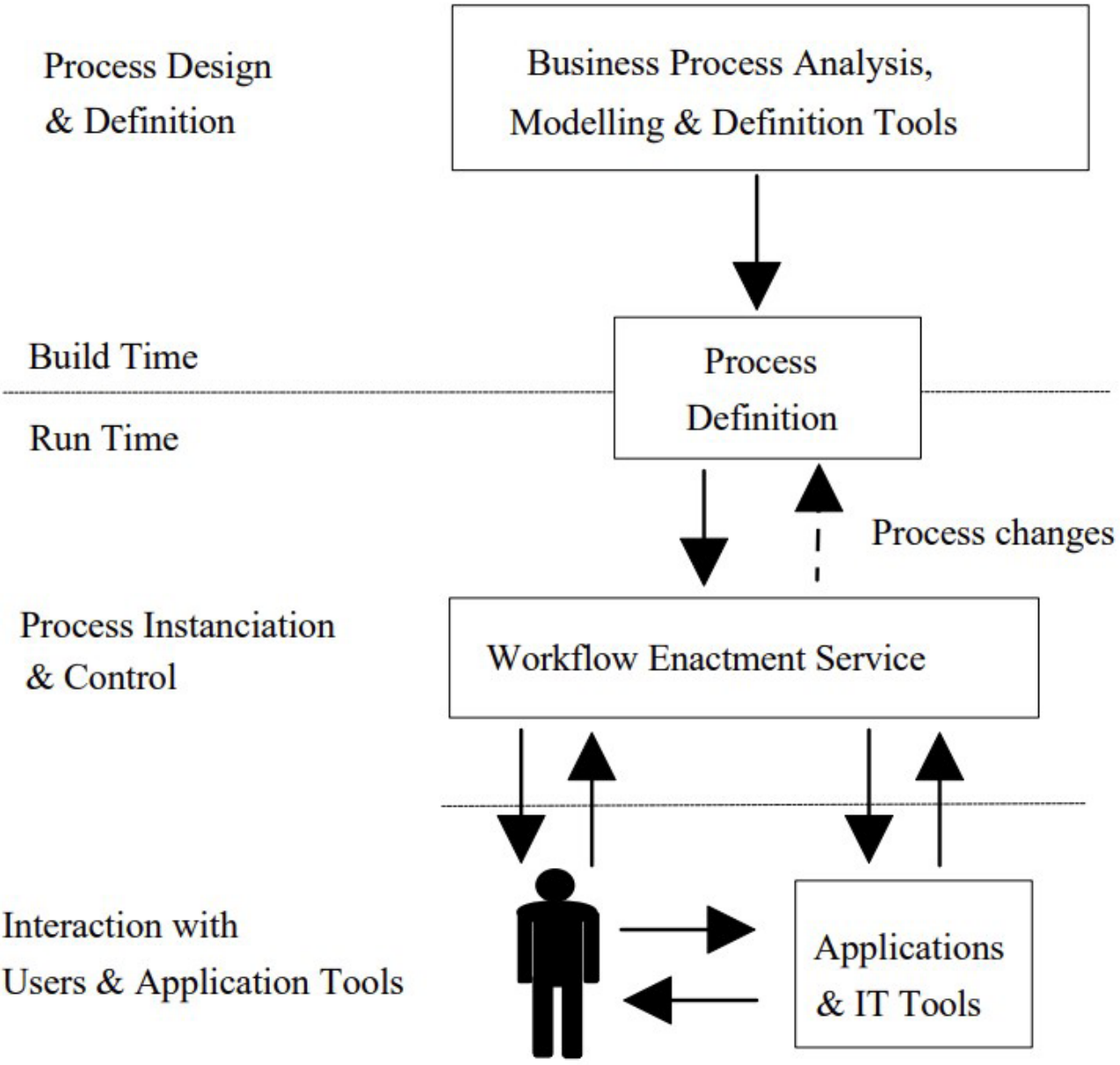


figure 1 - Caractéristiques du système de flux de travail exprime cette relation.

[ref-David Hollingsworth. The Workflow Reference Model [M]. Workflow Management Coalition, 1995: 6- 7]

Si l'on a besoin d'une définition pour comprendre comment un flux de travail est concrètement mis en œuvre et quels outils sont nécessaires, voici une explication.

Tout d'abord, l'objectif initial d'un flux de travail est d'améliorer la capacité à traiter des tâches complexes et de minimiser les erreurs humaines, tout comme dans une usine où l'on privilégie les machines pour remplacer le travail manuel.

De même, pour les flux de travail, il est nécessaire d'utiliser de nombreux outils informatiques pour soutenir le flux de travail.

La WfMC (Workflow Management Coalition), dans son "Modèle de référence de flux de travail" [Référence : David Hollingsworth. The Workflow Reference Model [M]. Workflow Management Coalition, 1995: 10-11], a présenté de nombreux outils de gestion de flux de travail présents sur le marché des technologies de l'information. Ces logiciels couvrent différents domaines tels que le traitement d'images, la gestion de documents, le courrier électronique et les annuaires, les applications de collaboration d'équipe, les applications basées sur des transactions, les logiciels de support de projet, la réingénierie des processus métier et les outils de conception de systèmes structurés, entre autres.

La WfMC aspire à établir une norme et à construire un modèle de mise en œuvre de système de flux de travail générique qui puisse correspondre à la plupart des produits du marché, fournissant ainsi une base commune pour le développement de scénarios d'interopérabilité.

Cette norme repose sur un cadre de classification où toutes les applications peuvent être trouvées dans une certaine catégorie, et les développeurs peuvent choisir leur domaine en fonction de ce cadre. Le cadre de la WfMC fournit des interfaces pour la conversion mutuelle des processus au sein de ce modèle. En résumé, cette norme constitue un cadre qui unifie les différentes catégories d'applications sur le marché, les rendant ainsi cohérentes en un tout. Ainsi, lors de la création d'un modèle

de flux de travail, il est possible de trouver les outils appropriés dans une catégorie donnée pour accomplir le processus de flux de travail.

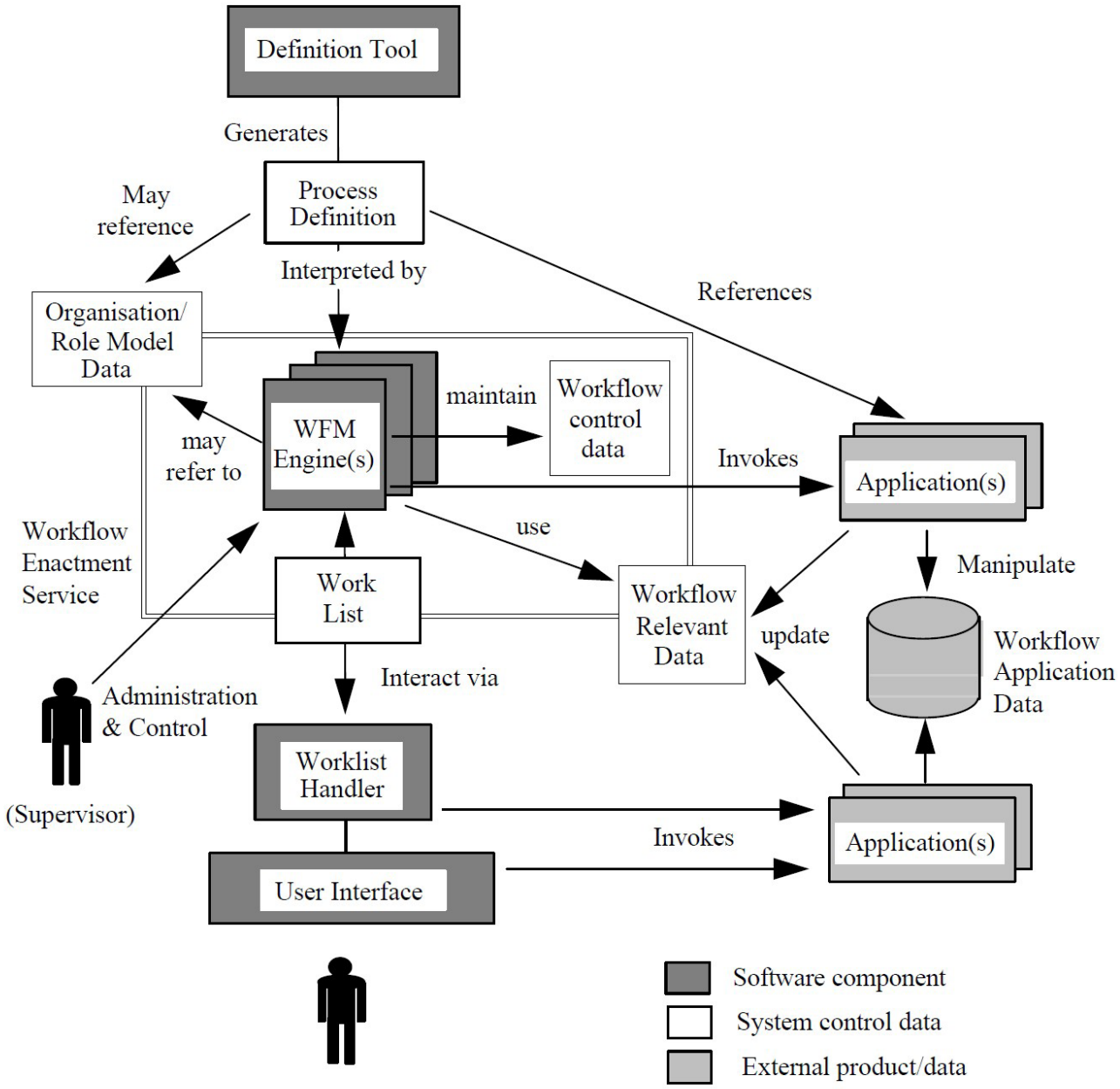


Figure 2-Generic Workflow Product Structure

[ref-David Hollingsworth. The Workflow Reference Model [M]. Workflow Management Coalition, 1995: 13]

La WfMC fournit également un modèle de référence universel pour les flux de travail, dans lequel sont inclus tous les éléments nécessaires à la conception et à la mise en œuvre d'un flux de travail. Veuillez consulter la figure 3.

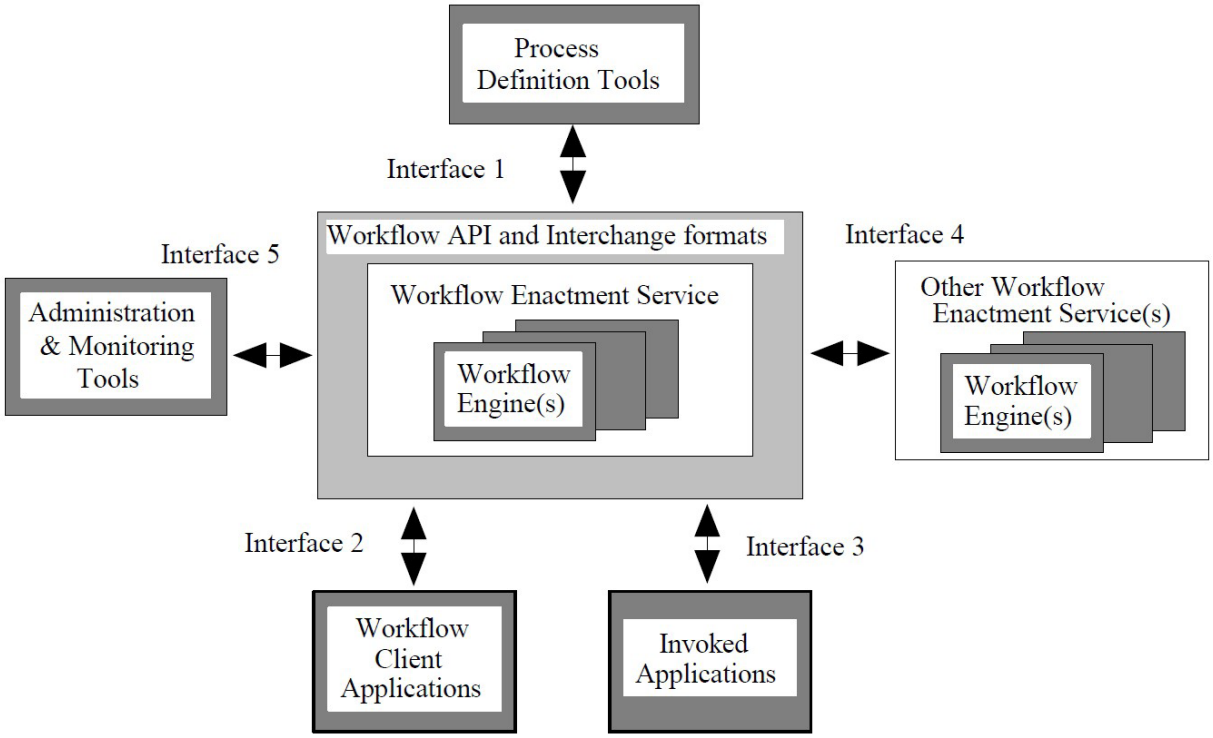


Figure 3-Workflow Reference Model-Componsents&Interfaces

[ref:David Hollingsworth. The Workflow Reference Model [M]. Workflow Management Coalition, 1995: 20]

Dans cette illustration, le WfMC définit les principaux outils nécessaires à la réalisation d'un flux de travail ainsi que leurs relations. Au cœur de l'image, le module "Workflow API and Interchange formats" agit comme le pivot du flux de travail, offrant le stockage et l'échange de données, tout en intégrant un moteur de flux de travail (définition et exécution de processus). Les quatre autres outils principaux comprennent les "Process Definition Tools" (outils de définition des processus), les "Administration & Monitoring Tools" (outils d'administration et de surveillance), les "Other Workflow Enactment Service(s)" (autres services d'exécution de flux de travail), ainsi que les "Workflow Client Applications" et "Invoked Applications" (applications clientes de flux de travail et applications invoquées). Ces fonctionnalités, constituant la base de l'ensemble de la structure, interagissent mutuellement. Les interfaces 1 à 5 dans le schéma représentent les connexions fondamentales entre ces modules, appelées l'API Workflow (interface de programmation d'application de flux de travail).

2.2 État de l'art sur la structure organisationnelle et les processus de travail de la société Gee.

Prendons l'exemple du cabinet de conception Gee, une agence spécialisée dans la partie conception architecturale, avec environ 100 architectes. Afin de garantir le bon fonctionnement de cette entreprise comptant 100 employés, Gee est organisée en deux parties : la section architecturale et la section de gestion. Pour la partie architecturale, Gee utilise un système de groupes combiné à une hiérarchie pour classer ses employés. Le système de groupes consiste à diviser les 100 personnes en 8 groupes, A à H, chaque groupe étant responsable de projets différents avec des emphases et des effectifs variés.

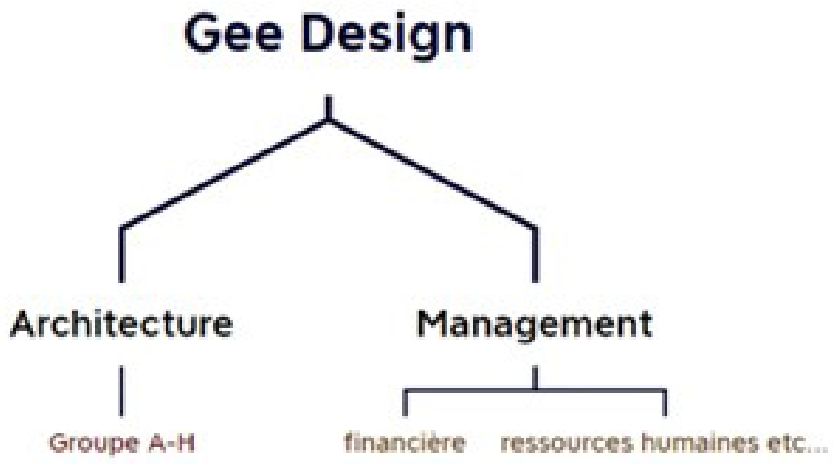


Fig. 2.2-1 Organigramme de l'entreprise Gee

Et voici la hiérarchie des architectes :

Il existe un total de 10 niveaux pour classer les architectes : A1, A2, B1, B2, C1, C2, D1, D2, D3, D4.

A1 : Patron

A2 : Les Partenaires B1 : Chef d’équipe B2 : Chef de projet

C1 : L’architecte polyvalent

C2 : Architecte Spécialiste D1-D2 : Assistant Architecte D3-D4 : Designer, Graphiste

Ci-dessus sont utilisées les abréviations A1, A2, etc. pour représenter les différents niveaux. Les niveaux plus élevés représentent une capacité plus grande, et par conséquent, un salaire plus élevé. Par exemple, un chef de projet de niveau B2 peut être responsable de la planification et du personnel d'un projet, ce qui signifie qu'il peut accomplir toutes les tâches des niveaux B2 et inférieurs, ainsi que des niveaux C1-2 et D1-4.

Chaque niveau a ses propres domaines d'expertise. Par exemple, le groupe H auquel j'appartiens a une forte demande pour la conception de

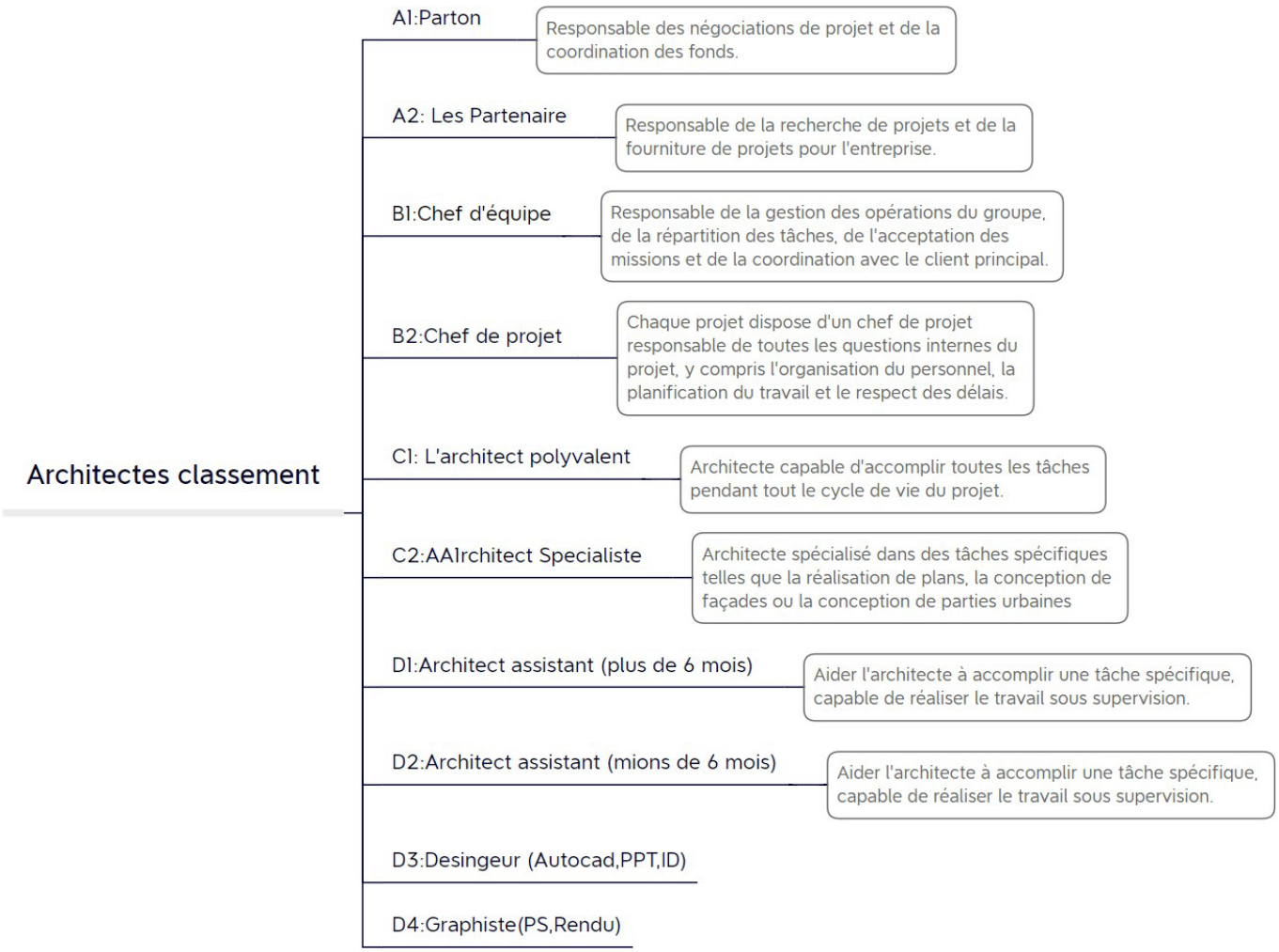


Fig2.2-2 Niveaux et compétences des architectes chez Gee

plans, ce qui entraîne une demande accrue pour les niveaux C1 et C2. De même, le groupe D, qui est responsable des compétitions, nécessite davantage de personnes spécialisées en rendus et modélisation, de niveaux D3 à D4.

L'organisation de Gee Design Studio se concentre sur des équipes de projet et des groupes chargés de prendre en charge ces projets. Toutes les tâches sont organisées par le chef de projet, et tous les membres de l'équipe sont des architectes possédant des compétences en conception. Au sein des équipes, des sous-groupes sont formés en fonction des projets. Par exemple, dans mon équipe H, composée de 12 personnes, il y a 2 B1, 2 B2, 3 C1-2, 4 D1-2 et 1 D3-4 en fonction des besoins du projet.

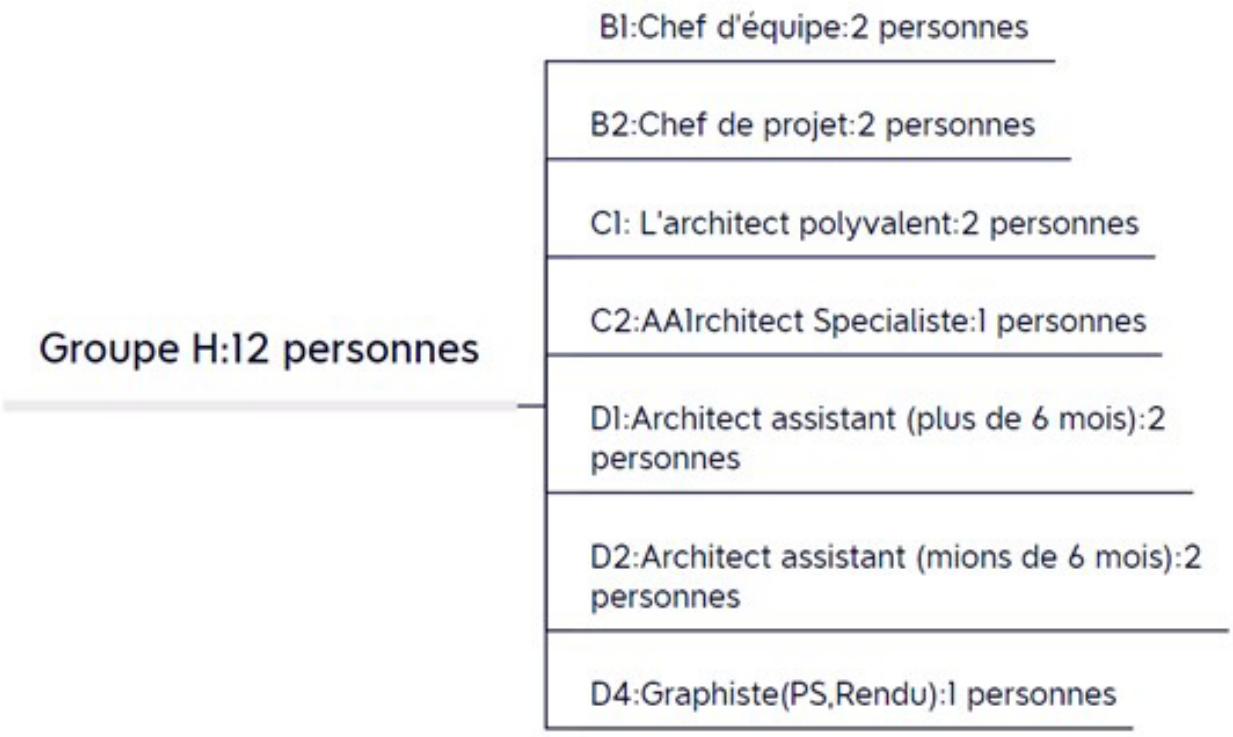


Figure 2.2-3 Proportion des membres dans le groupe H

2.3 Le processus de conception de Gee Design

Gee design comme la plupart des entreprises chinoises de conception résidentielle, adopte un modèle où le gouvernement fournit des ressources foncières, le propriétaire achète les droits d'utilisation, puis les confie à une entreprise de conception.

Bien que Gee respecte la créativité architecturale dans le processus de conception, étant donné que c'est la société propriétaire qui dirige le projet, la période de conception des projets de construction est coordonnée avec la société propriétaire. Chaque projet a un responsable côté propriétaire qui travaille avec nous pour traiter des questions connexes, telles que la recherche d'une société de conception structurelle, d'une société de conception paysagère, d'une société de conception intérieure. Le responsable côté propriétaire prend également les décisions finales sur le schéma final. Par conséquent, pour garantir que le projet de conception soit mené à bien, il est essentiel de guider la participation du côté propriétaire tout au long du processus de conception.

Prenons l'exemple du groupe H, après avoir pris en charge un projet, le processus est lancé selon le processus de conception, et le processus global du projet commence linéairement, se décomposant en 5 étapes comme illustré dans la figure 2.2-4.

fig2.2-4 Processus de conception architecturale du groupe H

Analyse du site

À cette étape, l'objectif principal est de comprendre le site, de proposer des plans d'aménagement intérieur du site, et d'étudier son utilisation en collaboration avec le propriétaire. Au cours de ce processus, il y a une étape très particulière appelée "Nadi", que tous les propriétaires effectuent. L'objectif principal de cette étape est de trouver, dans diverses contraintes, les possibilités de maximiser les intérêts. Ce processus peut nécessiter que les architectes fournissent 6 à 7 dispositions possibles, puis le propriétaire, après examen, détermine un schéma urbain approximatif. Ainsi, la direction générale du plan de masse est essentiellement établie à cette étape.

Chaque étape spécifique est illustrée dans la figure 2.2-5.

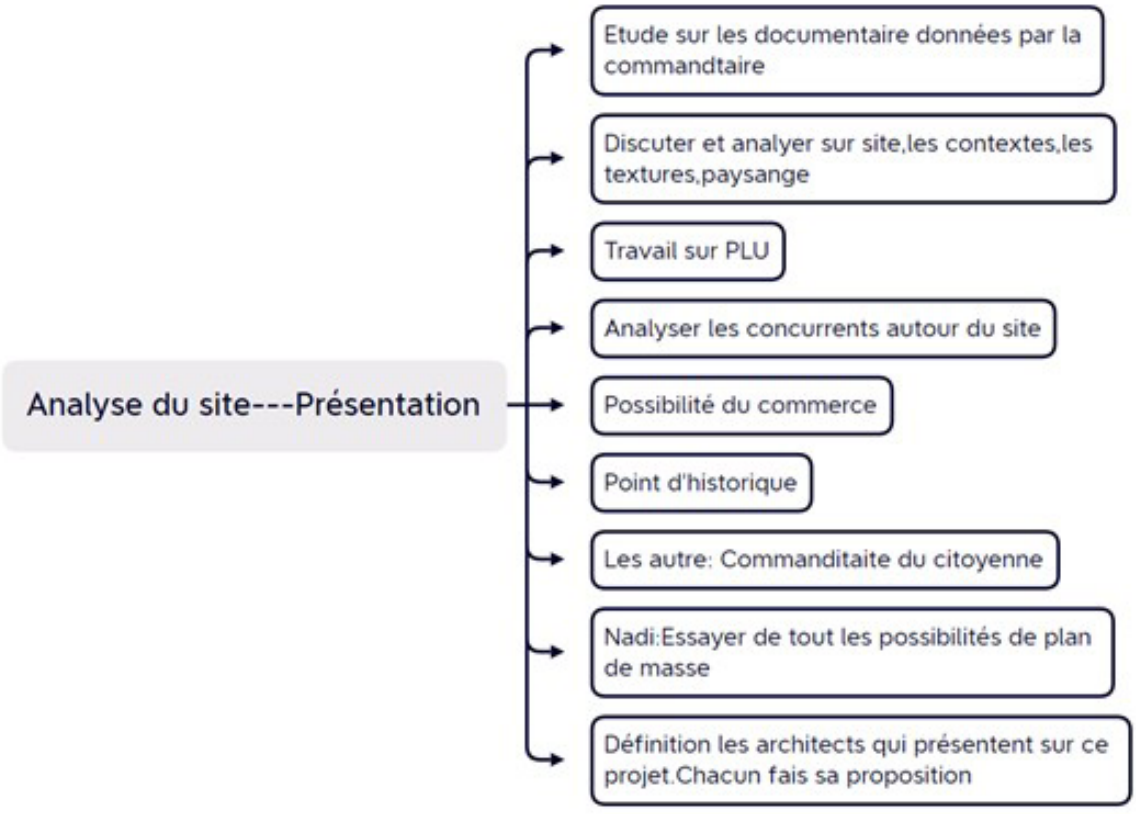


fig2.2-5 Processus de conception sur Analyse du Site

Hypothèse du Projet

À cette étape, l'accent est principalement mis sur la partie architecturale du projet, avec pour objectif de discuter et de déterminer l'effet architectural final. Bien entendu, tous les résultats finaux doivent être exprimés à l'aide d'une présentation, et une confirmation par la signature de toutes les parties concernées est requise.

Cette étape se divise en deux phases. La première phase concerne les discussions internes au sein de l'entreprise. Chaque vendredi après-midi, l'entreprise organise un événement de thé l'après-midi, invitant tous les architectes à y participer et à partager des fruits et des collations. Pour tirer pleinement parti de ce moment, l'entreprise organise des présentations sur les nouveaux projets, permettant à l'équipe de se familiariser avec les dernières actualités du projet. Dans cet environnement interactif, chacun a l'occasion d'exprimer ses idées sur le projet, de proposer des points de vue et de participer aux discussions. Tout au long de ce processus, nous recueillons des opinions qui serviront de base à notre conception.

La deuxième phase implique une discussion collective avec toutes les personnes impliquées dans le projet, y compris les plans de logement, les références graphiques et divers pourcentages. Une fois que tout est confirmé sans erreur, une présentation finale est faite au client. Si la deuxième phase n'obtient pas l'approbation finale, des ajustements continus sont effectués jusqu'à ce que tous les départements participants soient d'accord.

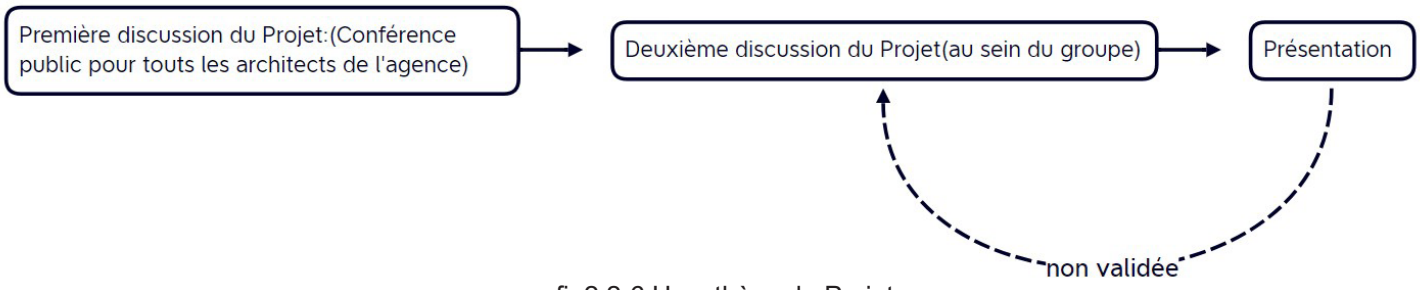


fig2.2-6 Hypothèse du Projet

Phase de dessin de la conception finale

À cette étape, toutes les conceptions sont finalisées, ne laissant que le travail de dessin. Certains architectes se retirent temporairement à cette étape, ne laissant que les architectes chargés du dessin. Le travail ici est plus concret et lourd, constituant l'étape la plus chronophage et énergivore de l'ensemble du processus. Les étapes de conception ici suivent également un processus linéaire, chaque étape devant être confirmée avant de passer à la suivante.

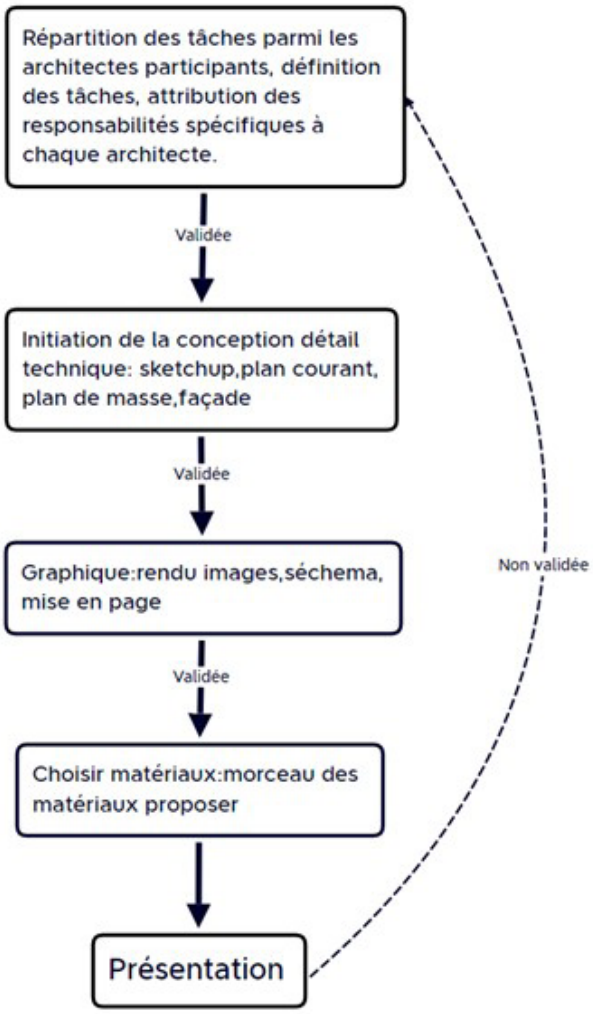


fig2.2-7 Phase de dessin de la conception finale

Collaboration en conception avec différentes entreprises de design

À cette étape, tout le travail est achevé, la phase de conception est complètement terminée. Ensuite, il revient au client (le promoteur) de prendre la tête pour rechercher des entreprises partenaires telles que des sociétés de plans techniques de construction, des bureaux d'études structure, des entreprises de conception paysagère, des entreprises de conception intérieure, des fournisseurs de matériaux, etc. Dans cette phase, le principe de base pour les architectes est de collaborer autant que possible avec les autres entreprises tout en assurant la stabilité de la conception. Cette étape constitue la dernière préparation avant la construction.

Collaboration ultérieure pour aider les entreprises techniques à résoudre les problèmes

À cette étape, l'accent est mis sur la collaboration pendant la construction.

Si nécessaire, des visites sur site peuvent être effectuées pour fournir des conseils et garantir autant que possible la réalisation de l'effet de conception architectural, tout en résolvant rapidement les problèmes éventuels.

Répartition des ressources humaines :

Prenez l'exemple du groupe H, la répartition des ressources humaines nécessaire à chaque étape d'un projet tout au long du processus varie. Comme illustré dans la figure fig2.2-8.

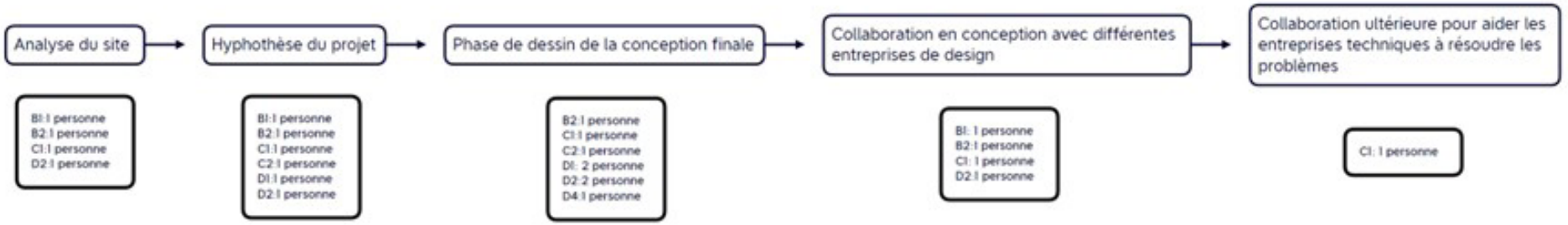


figure 2.2-8 des ressources humaines pour différents projets.

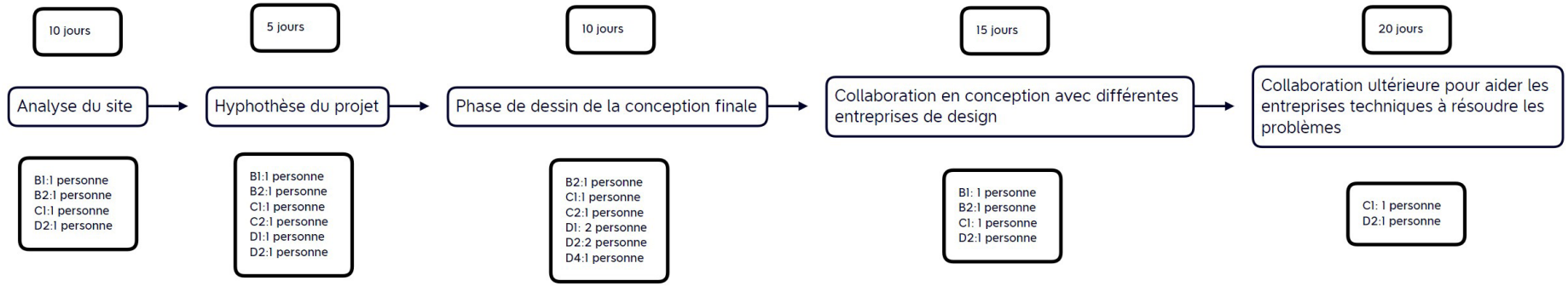


figure2.2-9 représente la répartition des ressources temporelles pour différents projets.

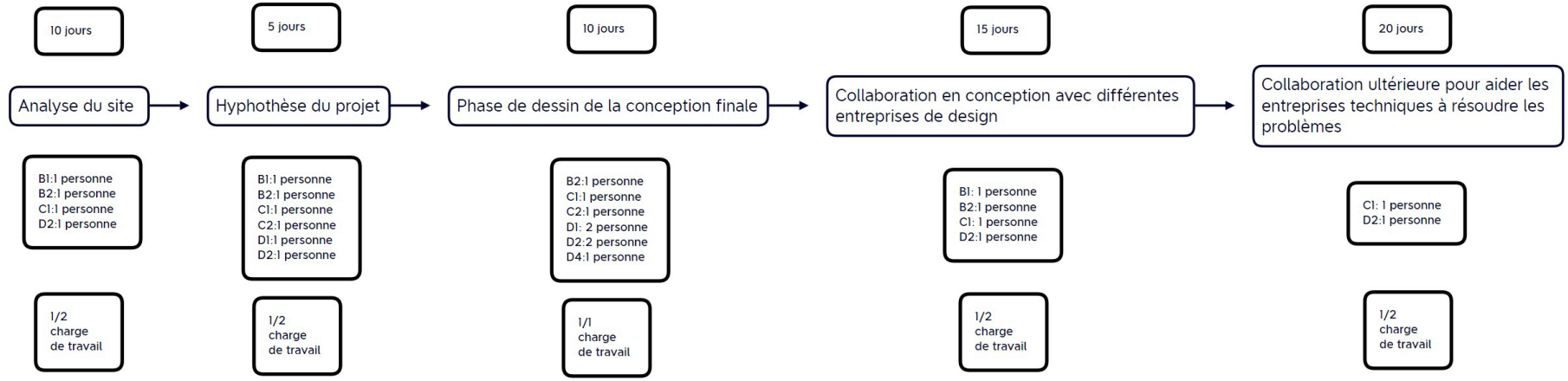


figure2.2-10 représente la répartition des ressources avec charge

L'allocation des ressources temporelles :

L'hypothèse de répartition du temps est basée sur le processus de travail réel de la société Gee, où la phase de conception a une période moyenne de 60 jours, ce qui donne une durée totale d'environ 90 jours pour l'ensemble du processus. Il est important de noter que ces paramètres temporels sont des estimations virtuelles basées sur l'expérience de l'auteur, n'ayant pas de valeur de référence pour des projets réels. De plus, ces paramètres temporels n'affecteront pas les résultats de cette mémoire. La figure2.2-9 représente la répartition des ressources temporelles pour différents projets.

L'allocation des charges de travail :

Sur la base de ce qui précède, une autre variable est introduite, la charge de travail. La charge de travail est quantifiée en fonction du temps que chaque personne consacre à chaque étape. L'auteur fait des hypothèses sur la charge de travail dans chaque étape (la charge de travail à chaque étape est virtuelle et basée sur l'expérience de l'auteur, sans valeur de référence pour des projets réels, et les paramètres temporels n'affectent pas les résultats de ce mémoire). Dans la première étape, "Analyse du Site", les 4 personnes travaillant sur ce projet doivent collaborer pendant 10 jours, mais ils ne consacrent que 4 heures par jour à ce projet pendant ces 10 jours.

Ainsi, ces quatre personnes contribuent à une charge de travail équivalente à la moitié pendant ces 10 jours. Et ainsi de suite.

Dans des projets réels, plusieurs projets peuvent se chevaucher simultanément. Chaque projet sera réorganisé en fonction de ce processus, et tout le travail excédentaire sera réparti sur d'autres projets pour garantir que chaque personne puisse accomplir un travail complet de 8 heures par jour sans heures supplémentaires.

Comment assurer que les projets progressent normalement dans un laps de temps limité, et comment organiser rationnellement le travail de

chacun pour accomplir autant de projets que possible sans faire d'heures supplémentaires est la responsabilité des architectes de niveau B1. Ils doivent maîtriser tous les projets en cours, gérer les informations et les progrès quotidiens de chaque projet, attribuer des tâches et veiller à ce que chaque personne accomplisse son travail de manière raisonnable selon son propre calendrier.

A partir de l'analyse détaillée du processus de conception réel de Gee Design, plusieurs caractéristiques majeures peuvent être identifiées :

- Caractère séquentiel et linéaire du processus

L'ensemble du déroulement suit strictement une progression en cinq étapes : analyse du site → hypothèse du projet → dessins de conception → coordination interdisciplinaire → accompagnement de chantier.

Chaque étape doit être entièrement validée avant que la suivante puisse débuter, garantissant ainsi une certaine rigueur et une organisation linéaire du travail.

- Forte implication du maître d'ouvrage

Depuis les premières analyses jusqu'à la validation du concept, le maître d'ouvrage joue un rôle déterminant.

Les architectes doivent non seulement respecter les logiques de conception, mais également répondre de manière continue aux exigences économiques, fonctionnelles et opérationnelles imposées par le client.

- Allocation dynamique des ressources au sein de l'équipe

Les responsables de projet (B1, B2) ajustent la répartition des effectifs selon les phases du projet :

plus de concepteurs mobilisés durant la phase d'hypothèse, puis un effectif réduit et ciblé lors de la phase de coordination ou de suivi de chantier.

- Importance centrale de la communication et de la gestion documentaire

Une part significative du temps de travail est consacrée à la coordination interdisciplinaire, à la gestion des modifications et au contrôle des différentes versions des documents.

La circulation fréquente de fichiers entre sous-équipes génère des redondances d'information et réduit l'efficacité globale.

- Tension temporelle du cycle de projet

Pour un projet résidentiel par exemple, la durée totale moyenne de conception est de 90 jours, dont environ 60 jours consacrés à la phase de conception.

Cette pression temporelle pousse les architectes à produire rapidement, souvent en s'appuyant sur des références existantes ou des modèles éprouvés.

En résumé, si le système de travail de Gee permet d'assurer une progression rapide et efficace des projets, il repose néanmoins fortement sur la coordination humaine, entraînant pour les architectes — en particulier les chefs de projet — une charge importante de tâches non créatives.

Cette situation limite leur investissement dans la conception architecturale proprement dite et accroît sensiblement leur charge de travail.Ainsi, l'introduction des technologies de flux de travail apparaît comme une solution pertinente pour optimiser les processus, réduire les échanges répétitifs et la surcharge informationnelle, et permettre aux architectes de se recentrer sur l'essentiel : la création architecturale.

3.Expérimentale:Étude sur le système de flux de travail collaboratif en conception architecturale.

Nous nous pencherons sur la recherche d'un système de workflow collaboratif pour la conception architecturale.

Dans la précédente partie, nous avons discuté en détail des principes fondamentaux de la technologie des workflows ainsi que du processus de workflow et de la structure organisationnelle de l'entreprise Gee. Nous entamons maintenant la phase expérimentale. Selon la définition du WfMC (Workflow Management Coalition) [Référence : David Hollingsworth. The Workflow Reference Model [M]. Workflow Management Coalition, 1995: 28-29], la méthode d'obtention d'un modèle de workflow nécessite trois types d'outils (Tools) et un moyen de connexion (APIs) : un outil de définition de processus, un outil de définition de structure organisationnelle, un outil de serveur de moteur de workflow et un moyen d'échange. Ces trois types d'outils sont centrés autour du serveur de moteur de workflow, qui devrait posséder les caractéristiques suivantes : 1. Être un outil numérique informatisé ne nécessitant pas l'intervention humaine. 2. Pouvoir distribuer automatiquement les tâches aux parties concernées selon le processus prédéfini et obtenir des retours. 3. Pouvoir stocker des fichiers pertinents et définir des services d'archivage.Comme illustré dans la figure 3-1.

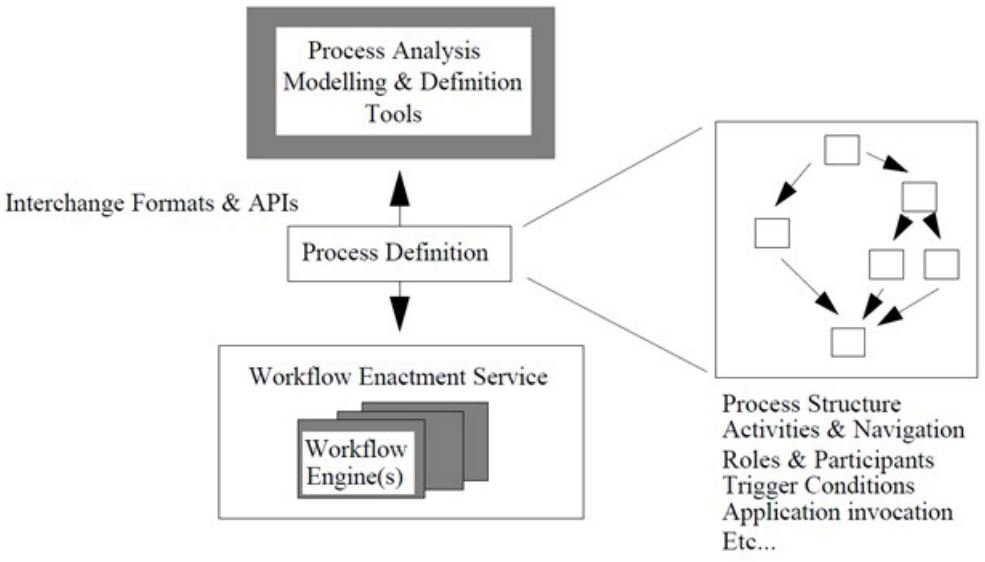


figure3-1 Process Definition Interchange[ref□David Hollingsworth. The Workflow Reference Model [M].Workflow Management Coalition, 1995: 28-29]

3.1 Modélisation pour le flux de travail

Nous allons maintenant modéliser le flux de travail de l'équipe H pour obtenir l'outil de définition de processus.

Dans le deuxième chapitre, le processus global de travail de l'équipe H a été présenté. Nous allons maintenant analyser la première étape, l'analyse du site. Dans cette première étape, tous les travaux peuvent être réalisés de manière indépendante et simultanée. Ces travaux ne nécessitent pas de discussions collectives ou de conception, et les architectes n'ont pas besoin de mettre en œuvre leurs compétences en conception. Une fois tous ces travaux terminés, ils doivent être confirmés par le niveau hiérarchique actuel le plus élevé. S'il n'y a pas de problème après l'inspection du niveau hiérarchique le plus élevé, la prochaine étape peut commencer. Dans le cadre de ce processus, les travaux sont principalement axés sur la collecte centrale de documents. Tous les participants peuvent accéder aux fichiers documentaires centraux, et tous les travaux sont effectués à cet endroit, comme un serveur central de documents.

Il y a deux personnes participantes : l'architecte qui effectue le travail concret récupère les données brutes dans le serveur public, effectue son travail et le télécharge sur le serveur. Ensuite, le serveur distribue la tâche d'examen au plus haut niveau actuel de l'architecte. L'architecte actuel achève l'examen et renvoie les résultats au serveur. Si l'examen n'est pas réussi, le serveur réattribuera le fichier d'examen à l'architecte pour modification. Ce processus se répète jusqu'à ce que l'approbation du superviseur soit terminée, puis le serveur archive, et le travail est terminé. Ainsi, chaque étape du flux de travail ici peut être exprimée en langage de programmation comme illustré dans la figure Fig3.1-1.(Process Analysis Modelling&Definition Tools)

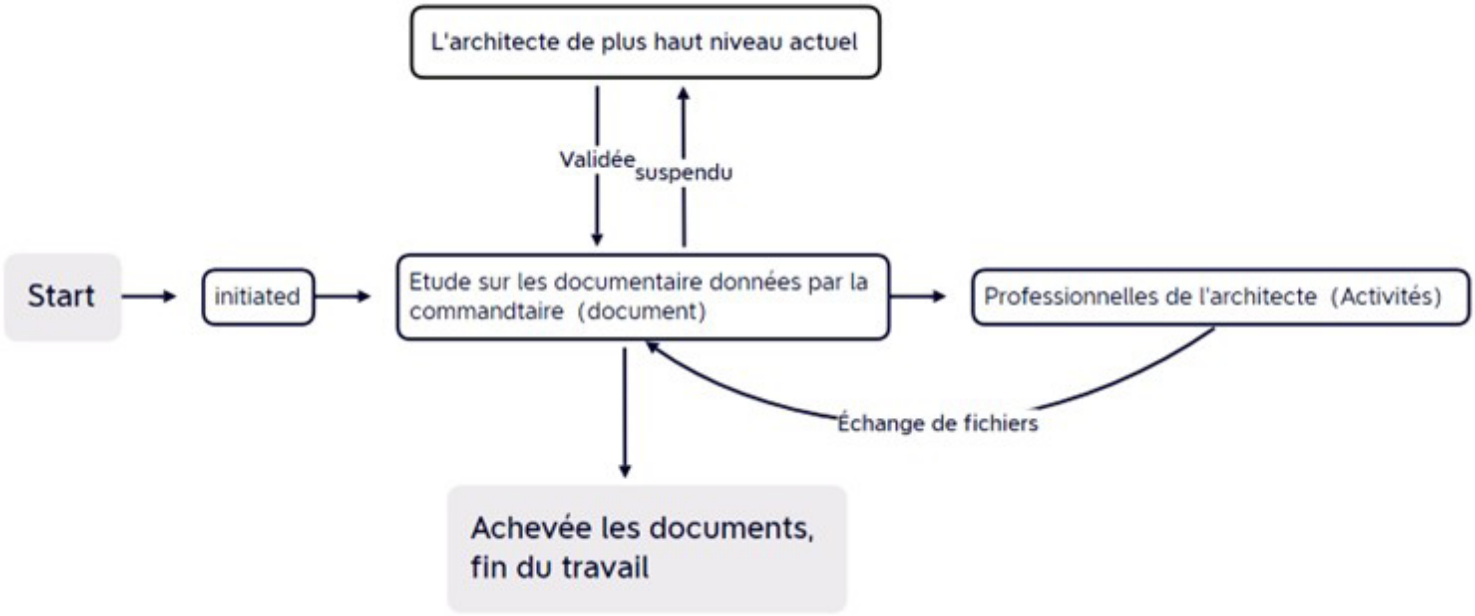


Fig3.1-1 Étapes pour accomplir une tâche linéaire, avec l'exemple de l'analyse du site

3.2 Modélisation du flux de travail de la phase d'analyse du site

Dans le processus réel du groupe H, l'analyse du site constitue la première étape du projet. Les principales tâches comprennent :

- La collecte des données (informations d'urbanisme, analyse d'ensoleillement, relevés et observations sur site) ;
- La rédaction d'un rapport préliminaire ;
- La validation par l'architecte de niveau supérieur ;

Dans un système de flux de travail, cette étape peut être représentée de la manière suivante :

- L'architecte télécharge les données brutes depuis le serveur central;
- Une fois la tâche réalisée, il dépose les résultats sur le serveur;

- Le système attribue automatiquement le dossier à l'architecte de niveau supérieur pour examen;
- Si l'examen est validé → archivage → fin de l'étape ;
- Si l'examen est refusé → retour pour correction → nouvelle soumission → reprise du cycle de révision.

Ce flux de travail met en place une boucle complète assignation automatique des tâches → retour d'information → archivage, réduisant ainsi les échanges répétitifs et les risques de perte de documents.

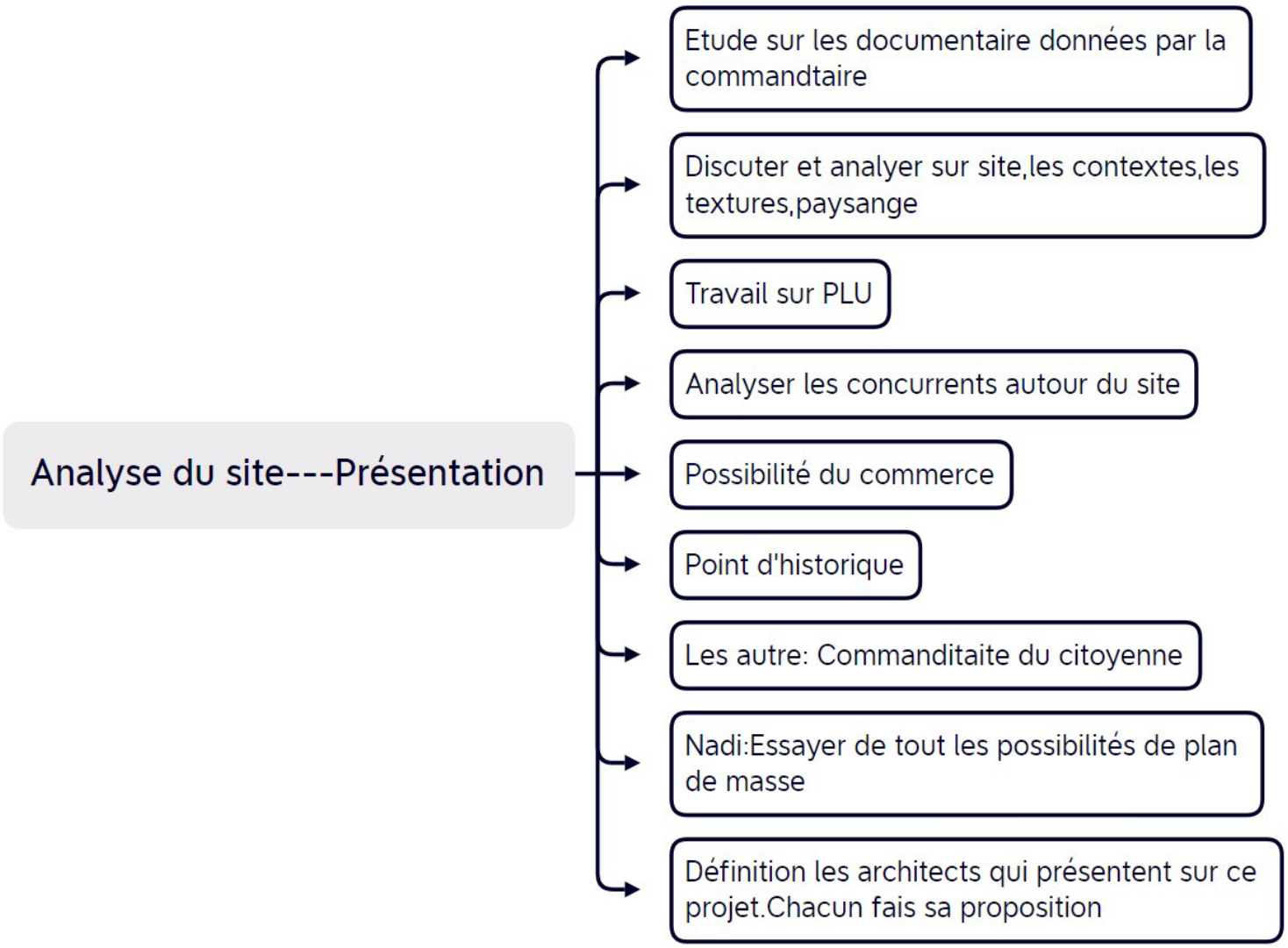


Fig3.2-1Processus de conception sur Analyse du Site

Fig3.2-2 répertorie toutes les étapes du processus structure actuel. Lorsque toutes les étapes du processus sont suivies conformément au processus de Fig3.1-1, la première partie de l'analyse du site sera complètement achevée. Le chef architecte de niveau supérieur actuel préparera alors une présentation et rendra compte à la partie prenante (dans cette expérience, le feedback par défaut de la partie prenante est considéré comme validé).

Comme illustré dans la Figure 3.2-2, et en référence au modèle de gestion présenté dans la Figure 3-1, l'enchaînement opérationnel peut être exprimé de la manière suivante :

la partie en bleu représente les opérations effectuées automatiquement par le serveur (échanges, distribution et mise à jour des données), tandis que la partie en rouge correspond aux interventions humaines nécessaires dans le processus.

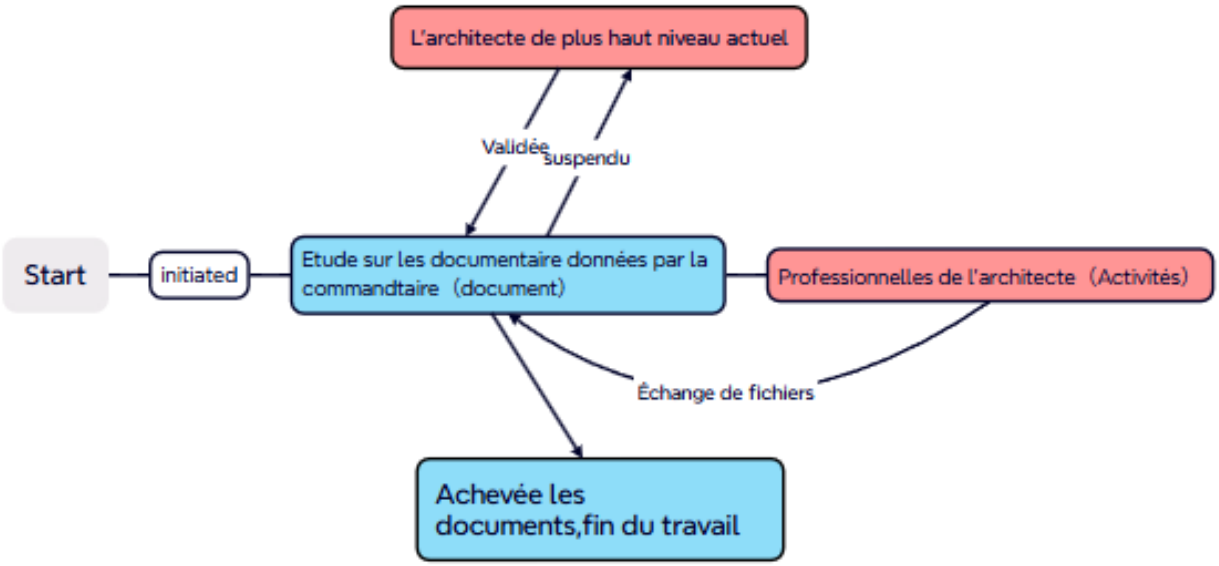


Fig3.2-2Processus definition de Etude sur les documentaire

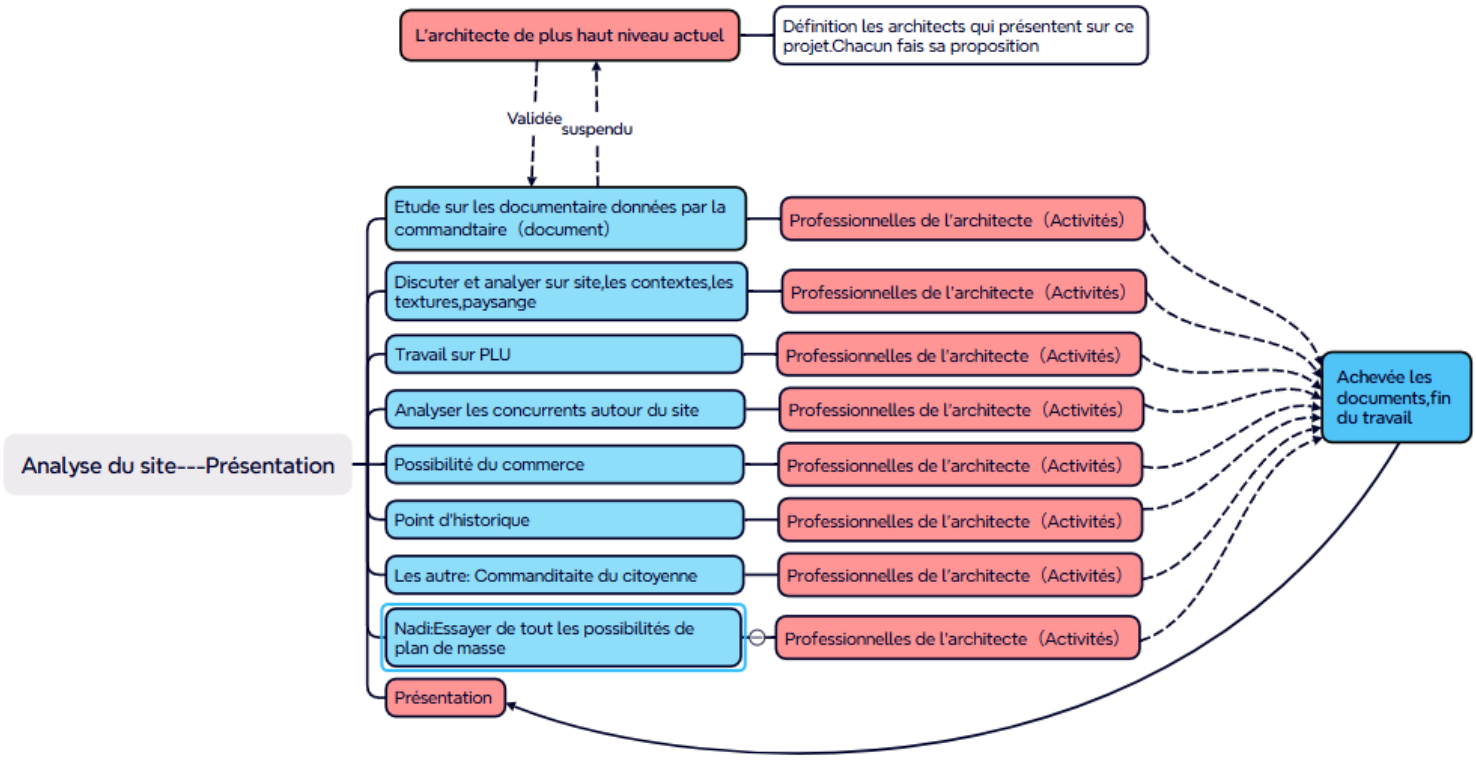


Fig3.2-3Processus definition de conception sur Analyse du Site

Au cours de la phase d'analyse du site, le groupe H doit généralement accomplir neuf sous-tâches spécifiques.

Chacune de ces sous-tâches implique une interaction entre l'utilisateur et le serveur, et les rôles attribués varient selon le niveau hiérarchique des architectes impliqués.

La correspondance entre chaque sous-tâche et les niveaux d'intervention est présentée dans le schéma Fig-3.2-3.

Ainsi, il apparaît que pour les architectes de niveau supérieur, la majorité du travail consiste à prendre des décisions et à valider ou corriger les propositions.

Ce mode de fonctionnement correspond à la pratique de nombreux architectes expérimentés, capables d'exercer un contrôle global sur l'ensemble du projet.

Par ailleurs, les différentes tâches peuvent être réalisées en parallèle, sans interférer les unes avec les autres : chaque membre de l'équipe transmet directement ses résultats au décideur, tandis que les retours de ce dernier sont systématiquement enregistrés dans le serveur.

Cela permet à tous les participants de visualiser en temps réel l'état d'avancement du projet et d'assurer un suivi continu du processus.

Pour le système de flux de travail, bien que la phase d'analyse du site comporte une grande variété de sous-tâches, chacune d'elles suit une même logique cyclique :

démarrage → exécution → dépôt → validation → retour → archivage → achèvement.

Les avantages de ce modèle sont les suivants :

1. Assurer le déroulement ordonné de chaque sous-tâche, évitant les oublis ou ruptures de processus ;
2. Centraliser l'ensemble des documents, avec une traçabilité complète des versions ;
3. Garantir la qualité des résultats grâce au cycle continu d'examen et de rétroaction.

3.3 Modélisation du flux de travail de la phase d'hypothèse du projet

Dans la pratique de conception de Gee, la phase d'hypothèse du projet joue un rôle crucial de transition entre les études préliminaires et la définition définitive du concept architectural.

L'objectif de cette phase n'est pas seulement de proposer des intentions architecturales viables, mais surtout de faire converger, à travers des comparaisons entre plusieurs variantes et une série de discussions successives, vers une solution capable de répondre à la fois aux logiques de conception et aux exigences du maître d'ouvrage.

Dans le modèle traditionnel, ce processus s'accompagne souvent de communications fréquentes et de nombreuses modifications successives, ce qui peut entraîner une multiplication des versions, des retards dans la transmission de l'information, ainsi qu'une baisse d'efficacité globale.

Pour répondre à ces problématiques, cette recherche associe chacun des sous-tâches de cette phase à une logique de flux de travail, transformant ainsi l'ensemble de la procédure en un processus systématisé, structuré, traçable, et fonctionnant en boucle fermée.

Afin de simplifier la présentation, les relations entre le processus de conception et le flux de travail numérique ont été synthétisées sous forme de schémas.

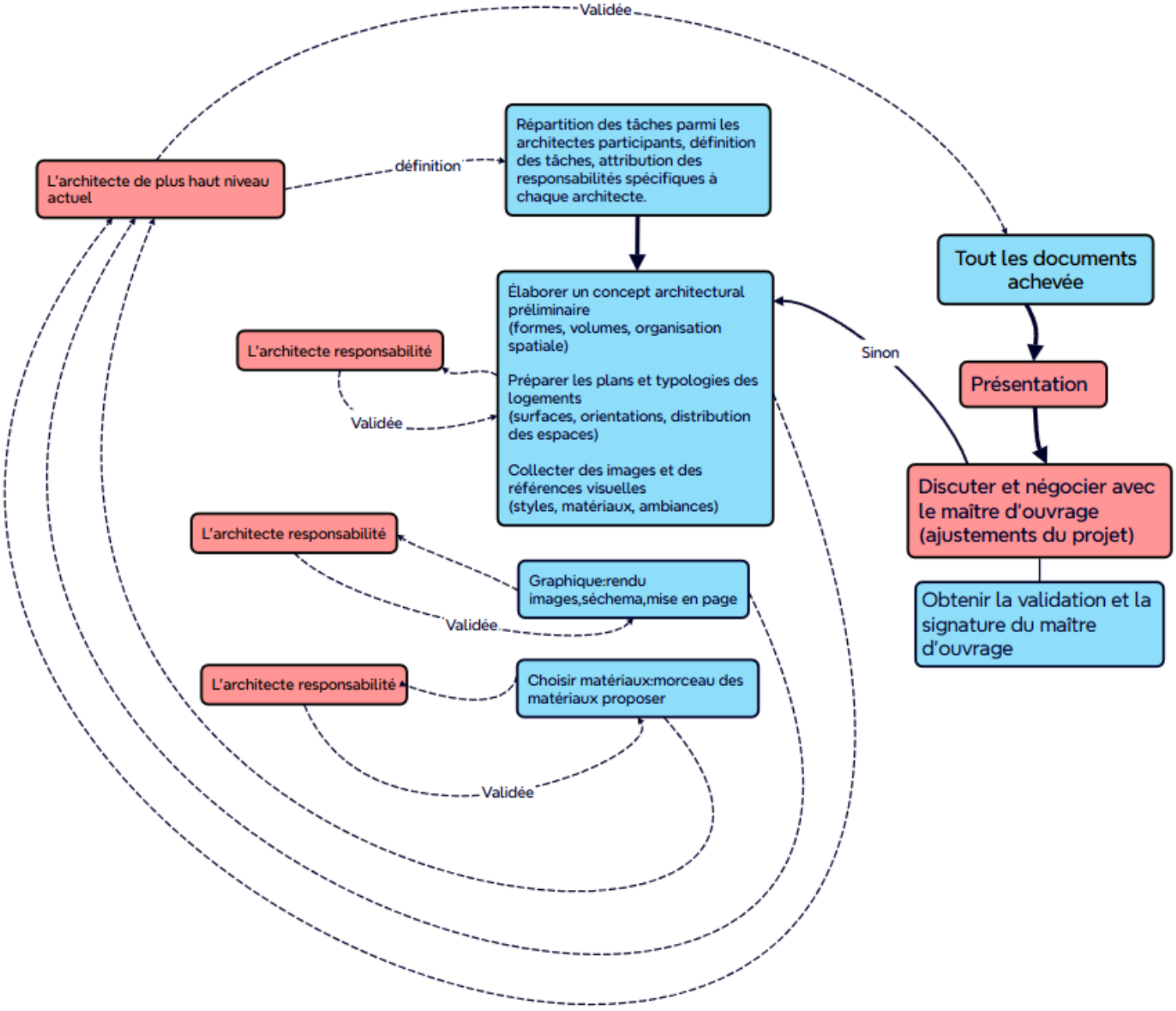


Fig3.3-1Processus definition de conception sur hypothèse du projet

L’analyse du flux de travail de la phase d’hypothèse du projet révèle que la majeure partie du travail de conception est assurée par les architectes opérationnels.Cette répartition entraîne une augmentation de la charge de travail “manuelle” pour ces derniers, mais réduit en parallèle la pression liée à la responsabilité de validation.À l’inverse, les architectes responsables de l’examen portent une charge décisionnelle importante, tout en assumant une quantité moindre de tâches productives.

Dans cette phase, chaque hypothèse de conception constitue une référence essentielle pour la traçabilité du projet.Il n’est donc pas possible d’écraser ou de remplacer les documents précédents: toutes les versions doivent être archivées et classées par date.Ainsi, le serveur doit assumer des fonctions plus avancées, telles que:

1. **Gestion multi-versions des propositions**:le système enregistre et archive automatiquement chaque variante, évitant les difficultés de suivi des versions propres aux méthodes traditionnelles.
2. **Double niveau de validation:**en plus de la validation interne par les responsables B1/B2, le processus intègre une étape incontournable de confirmation par le maître d’ouvrage.

3.Réduction des redondances informationnelles:toutes les remarques et modifications sont directement associées aux fichiers concernés, éliminant les pertes d’information liées aux échanges oraux ou aux courriels dispersés.

4.Amélioration de l’efficacité décisionnelle:les retours du maître d’ouvrage sont transmis via le système, permettant une révision rapide des propositions et une nouvelle itération d’examen, ce qui réduit considérablement les délais de communication.

En somme, l’introduction d’un flux de travail structuré permet d’assurer une articulation efficace entre la génération créative et la validation par le maître d’ouvrage.Cette organisation renforce la stabilité et l’ordre du processus, constituant une base solide pour la phase suivante de dessin final.

3.4 Modélisation du flux de travail de la phase de dessin de la conception finale

Dans le processus global de Gee, la phase de dessin de la conception finale constitue l'étape la plus longue et la plus chargée en travail. Contrairement aux phases antérieures de « recherche — hypothèse », principalement dédiées à la formulation et à la validation des intentions architecturales, cette phase a pour objectif central de transformer le projet validé en un ensemble complet, précis et exploitable par les disciplines techniques ultérieures.

Cette étape n'implique pas encore de collaboration avec les autres corps de métier.

Au contraire, elle exige que l'équipe d'architecture réalise elle-même l'ensemble des approfondissements liés au projet, notamment :

- le plan masse définitif,
- les plans des logements et l'organisation fonctionnelle des espaces,
- les élévations et coupes architecturales,
- les descriptions de matériaux, les tableaux de surfaces et les informations techniques,
- les perspectives et images illustratives nécessaires à la communication du projet,
- l'ensemble des documents standardisés destinés à être transmis aux équipes structure, paysage, CVC, électricité et plomberie.

Dans le mode de fonctionnement traditionnel, cette phase repose souvent sur la communication continue entre les architectes chargés du dessin : toute modification doit être immédiatement transmise verbalement ou via des messages instantanés, et les fichiers sont ensuite échangés par messagerie une fois la tâche terminée.

Si cette méthode peut fonctionner dans un petit atelier, elle devient rapidement insuffisante dès que le projet prend de l'ampleur, que le nombre

d’architectes impliqués augmente ou que la durée du projet s’allonge.
Dans ces situations, il devient extrêmement facile de générer des incohérences entre les versions des plans, des décalages dans la synchronisation des modifications, ou encore une confusion dans la nomenclature des fichiers.

Ainsi, durant cette étape, la traçabilité et la visualisation en temps réel de l’avancement de chaque tâche deviennent essentielles. Lorsqu’un problème apparaît, il doit être possible de suspendre immédiatement une partie du processus afin de corriger l’erreur sans entraîner d’autres conséquences.

Par conséquent, les priorités du modèle de flux de travail à ce stade diffèrent nettement de celles des deux phases précédentes.
Dans les premières phases, le serveur joue principalement un rôle d’agrégation et de transmission linéaire de l’information, avec un accent mis sur l’attribution des responsabilités et la réduction du temps perdu en communication inutile.

En revanche, dans la phase de dessin final, les enjeux majeurs résident dans la traçabilité des opérations, la quantification du travail, et la visualisation claire et collective de l’ensemble du processus.
Afin de simplifier la présentation, le flux de travail de la phase de dessin de la conception finale ont été synthétisées sous forme de schémas.

La phase de dessin de la conception finale constitue l’étape la plus dense et la plus complexe du processus de conception chez Gee. Son objectif essentiel est de transformer avec précision le projet validé en un ensemble complet de documents techniques directement exploitables par les disciplines ultérieures.

L’analyse du modèle de flux de travail montre que les méthodes traditionnelles, reposant principalement sur la communication humaine et la gestion manuelle des fichiers, conduisent facilement — dans les projets de grande envergure — à des incohérences de versions, des retards dans la synchronisation des modifications et des pertes d’information.

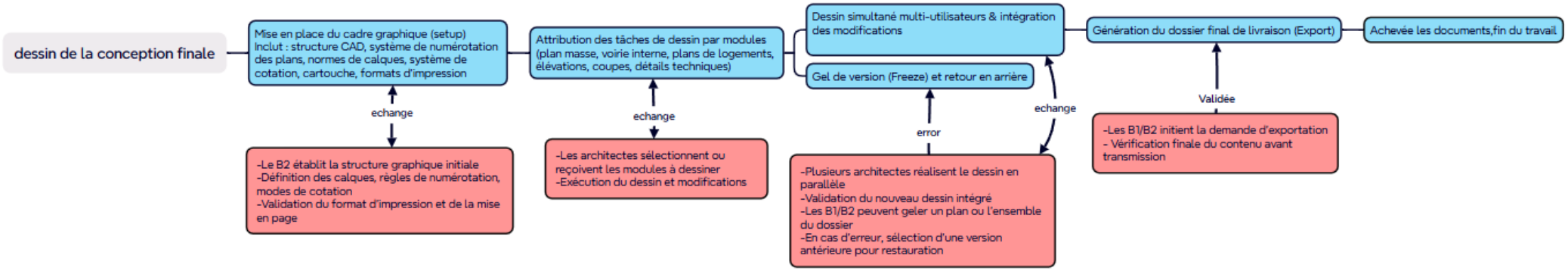


Fig3.3-1Processus definition de conception sur dessin de la conception finale

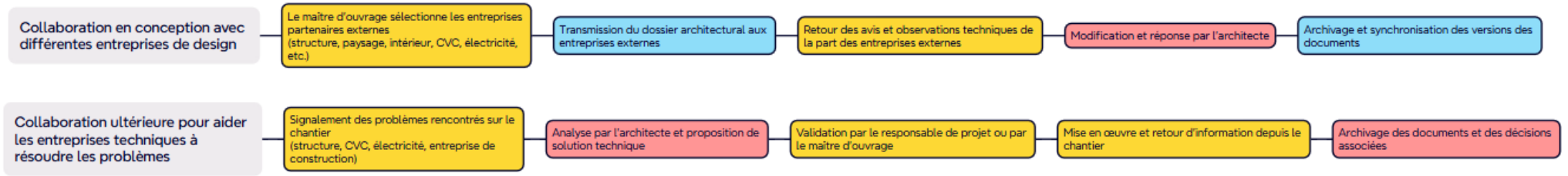


Fig3.3-1Processus definition de conception sur collaboration interdisciplinaire et de collaboration ultérieure

Le système de flux de travail, en structurant l'ensemble des tâches de dessin et en distinguant clairement les opérations humaines de celles automatisées par le serveur, confère au processus une meilleure traçabilité, une visibilité accrue de l'avancement et une capacité de retour en arrière. Contrairement aux phases précédentes, centrées sur la collecte d'informations et l'efficacité décisionnelle, cette phase met davantage l'accent sur le contrôle des versions, la gestion de l'avancement et la cohérence des livrables.

3.5Modélisation du flux de travail de la phase de collaboration interdisciplinaire et de la phase de collaboration ultérieure

Dans le processus global de Gee, la phase de collaboration interdisciplinaire marque le moment où la conception architecturale est essentiellement finalisée et entre dans une étape de coordination élargie. À ce stade, le maître d'ouvrage fait intervenir des entreprises externes telles que les bureaux d'études structure, paysage, design intérieur, CVC/électricité ainsi que des fournisseurs de matériaux. Le rôle de l'architecte évolue alors du pilotage de la conception vers une coopération active, consistant à garantir que l'intégrité du projet architectural et l'intention du concepteur soient préservées malgré l'intervention d'autres disciplines.

La phase de collaboration ultérieure, quant à elle, intervient principalement au cours de la phase de chantier. Ses caractéristiques reposent sur le fait que l'architecte n'assume plus la direction de la conception, mais devient un acteur dédié à l'assistance technique et au contrôle qualitatif. L'architecte doit répondre aux problèmes rencontrés sur le chantier, y compris les détails constructifs, les substitutions de matériaux et la coordination in situ.

Dans les modes de fonctionnement traditionnels, ces deux phases reposent largement sur des appels téléphoniques, des réunions sur site et des échanges par courriel. Les informations y sont dispersées, les cycles de retour sont longs, et cela peut facilement entraîner des retards de chantier.

Pour répondre à ces limites, cette recherche modélise le flux de travail de ces phases en le transformant en un système structuré basé sur un système de tickets (Ticket System). Chaque question provenant des disciplines externes est traitée comme un ticket, doté d'un numéro d'origine,

d’un contenu, d’un responsable, d’un statut, d’une date limite et d’un lien avec la version correspondante des documents graphiques. Le ticket suit un mécanisme cyclique : création — attribution — traitement — validation — clôture, garantissant que toutes les problématiques techniques soient enregistrées, suivies et résolues de manière exhaustive.

Dans le processus de modélisation, une couleur supplémentaire — le jaune, représentant les entreprises externes — a été introduite. L’analyse du tableau comparatif montre que, dans ces deux étapes, l’architecte se trouve dans une position réceptive et passive, c’est-à-dire qu’il n’intervient qu’à partir du moment où une demande ou un problème lui est transmis. En l’absence de sollicitation, aucune action n’est requise et l’architecte peut se consacrer à d’autres tâches.

Ainsi, la logique du programme pour cette partie du flux de travail peut être structurée selon un mode de ticket : chaque tâche émise par une partie externe constitue un ticket contenant l’ensemble des informations nécessaires. Tout participant au projet disposant d’un niveau d’autorisation suffisant peut alors prendre en charge le ticket et y répondre de manière autonome. Une fois le ticket traité, le système doit diffuser l’information à l’ensemble des intervenants, assurant une notification collective et une transparence complète du suivi.

3.6 Synthèse du chapitre

Ce chapitre a procédé à la décomposition et à la reconstruction du processus opérationnel réel de Gee Design, afin d'établir un modèle expérimental basé sur la logique des flux de travail. Plus précisément, le chapitre a analysé cinq phases — analyse du site, hypothèse du projet, dessin final, collaboration interdisciplinaire et collaboration ultérieure — et a associé leurs tâches respectives à un cadre unifié de flux de travail, formant ainsi un mécanisme de collaboration traçable et à boucle fermée.

Les principaux résultats sont les suivants :

- Phase d'analyse du site : le système de flux de travail permet une gestion en boucle fermée des tâches telles que l'étude documentaire, l'analyse sur site et l'interprétation des réglementations, évitant ainsi la perte d'informations et les conflits de versions.
- Phase d'hypothèse du projet : l'introduction d'une « double boucle fermée » intégrant à la fois la validation interne du responsable et le retour du maître d'ouvrage améliore l'efficacité décisionnelle.
- Phase de dessin final : l'intégration des productions architecturales permettant une vérification et une archivisation unifiées garantit la cohérence et la fiabilité des versions des documents graphiques.
- Phase de collaboration interdisciplinaire : la circulation systématisée des tâches remplace la communication manuelle traditionnelle, réduisant les redondances et rendant le processus plus transparent et plus efficace.
- Phase de collaboration ultérieure : l'utilisation d'un système de "tickets" pour traiter les problématiques du chantier assure la rapidité des retours et la traçabilité des décisions.

De manière générale, l'application du système de flux de travail dans ces cinq phases démontre sa valeur dans l'optimisation du processus de conception architecturale :

- Amélioration de l'efficacité : réduction des communications répétitives

et du pilotage manuel, permettant aux architectes de se concentrer sur leur cœur de métier.

- Renforcement de la transparence : toutes les tâches sont traçables dans le système, évitant omissions et pertes d'informations.
- Assurance qualité : grâce aux boucles de rétroaction et aux validations multi-niveaux, les résultats gagnent en fiabilité.
- Archivage systématique des fichiers : la visualisation processuelle du projet introduit une logique de chronologie permettant la documentation précise de chaque contribution.
- Ainsi, le modèle expérimental présenté dans ce chapitre constitue la base du chapitre suivant. Le Chapitre 4 — Résultats et analyse examinera, au moyen de comparaisons expérimentales et de données quantitatives, l'impact du système de flux de travail sur l'efficacité, la qualité de la collaboration et la réduction de la charge de gestion pour les architectes.

4. Mise en œuvre et application du prototype de flux de travail

- Dans ce quatrième chapitre, l'objectif est de valider et d'illustrer l'application concrète du modèle de flux de travail architectural proposé au chapitre précédent.
- Cependant, étant actuellement en situation de salarié, et mes collègues ne souhaitant pas intégrer un dispositif expérimental externe dans les projets professionnels en cours, il m'est impossible de mettre en place une expérimentation réelle impliquant plusieurs collaborateurs, un déploiement à l'échelle de l'agence ou un suivi opérationnel sur une longue durée.
- Face à cette contrainte, le choix méthodologique s'est porté sur le développement d'un prototype expérimental, permettant de simuler de manière contrôlée le fonctionnement d'un projet architectural et d'en analyser les mécanismes internes.
-

4.1 Méthodologie de la recherche

Afin de conserver une pertinence scientifique malgré l'absence de terrain réel, cette recherche s'appuie principalement sur une démarche de simulation expérimentale, fondée sur un prototype d'application numérique développé sous Xcode (Swift/SwiftUI), implémentant un modèle de flux de travail architectural.

Les données produites par cette simulation font ensuite l'objet d'une modélisation déductive, permettant d'analyser de manière systématique la logique du workflow et ses effets sur l'organisation du projet, la distribution des tâches et la charge de travail.

Une enquête qualitative légère auprès d'acteurs du secteur pourra éventuellement être menée dans un second temps à titre complémentaire,

mais elle ne constitue pas le cœur du dispositif expérimental présenté dans ce chapitre.

4.1.1 Simulation expérimentale

La méthode de simulation consiste à construire un environnement expérimental contrôlé à partir d'un projet résidentiel de petite échelle (50–80 logements), servant de support de test.

Le processus réel de conception observé au sein de l'agence Gee Design est abstrait sous la forme d'un scénario structuré, puis implémenté dans un modèle multi-agents, au sein duquel plusieurs architectes numériques interagissent avec un ensemble de tâches organisées selon les cinq grandes phases du projet :

- analyse du site,
- hypothèse de projet,
- dessin de la conception finale,
- collaboration interdisciplinaire,
- accompagnement en phase chantier.

La simulation se déroule selon un calendrier journalier, permettant d'observer la manière dont les tâches se distribuent, s'enchaînent et se valident à l'intérieur du flux de travail.

- L'analyse se concentre notamment sur :
- la durée relative des différentes phases,
- la répartition des tâches entre les membres de l'équipe,
- la charge de travail quotidienne de chaque agent,
- l'effet des règles de workflow (dépendances, validations, archivage) sur la continuité du projet.

4.1.2 Modélisation déductive des données

Compte tenu de la complexité inhérente aux projets architecturaux

— multiplicité des acteurs, interdépendance des tâches, itérations successives de conception — les résultats issus de la simulation ne sont pas interprétés de manière ponctuelle, mais intégrés dans un cadre de modélisation déductive.

Les données produites par le prototype (dates de début et de fin, états des tâches, niveaux de charge de travail, enchaînement des phases) sont réorganisées sous la forme :

- de tableaux synthétiques,
- de diagrammes temporels (type diagramme de Gantt),
- d'indicateurs quantitatifs.
- Cette modélisation permet de dériver des indicateurs tels que :
- le nombre de conflits potentiels entre tâches ou versions,
- la durée des cycles de validation et de retour,
- la fréquence des reprises et corrections,
- la distribution de la charge de travail entre conception et gestion,
- l'impact des règles de workflow sur la continuité du projet.

4.2 Architecture du prototype de workflow collaboratif

Afin de vérifier la faisabilité et l'opérationnalité du modèle élaboré au chapitre 3, cette recherche a développé un prototype d'application numérique, conçu comme une preuve de concept (Proof of Concept).

Ce prototype n'a pas vocation à devenir un logiciel professionnel finalisé. Il constitue un dispositif expérimental permettant d'observer, dans un environnement contrôlé et reproductible, les effets du workflow sur la répartition des tâches, l'avancement des phases, la circulation de l'information et la distribution de la charge de travail au sein d'une équipe de conception architecturale.

Le système repose sur un modèle multi-agents, dans lequel chaque

agent représente un acteur réel du processus de conception (B2, B1, C1, D1, etc.), doté de comportements différenciés selon son niveau hiérarchique.

4.2.1 Principes généraux de l'architecture

L'architecture du prototype repose sur quatre composantes principales.

(1) Les agents (A1–D4)

Chaque agent correspond à un membre de l'équipe de conception et possède :

- un rôle hiérarchique,
- une capacité maximale de travail quotidienne (8 heures),
- une logique décisionnelle autonome dépendant de son niveau.

Les agents de niveaux B, C et D peuvent exécuter des tâches, tandis que les agents de niveau A conservent exclusivement un rôle décisionnel et de validation, sans intervention directe dans la production.

L'agent B2 joue un rôle central de supervision :Il contrôle la complétion des phases, autorise l'ouverture de la phase suivante et régule le rythme global du projet.

(2) Les tâches structurées sous forme de workflow

Chaque tâche est définie par :

- sa phase,
- ses agents exécutants,
- une charge horaire estimée,
- une charge restante,
- des dates de début et de fin,
- un statut,
- des dépendances préalables.

Cette structure garantit un enchaînement contrôlé des actions.

(3) Le mécanisme temporel (timeline simulée)

La simulation progresse jour par jour, selon une logique répétitive : capacité journalière, sélection des tâches disponibles, exécution, validation et libération des dépendances.

(4) Le registre de charge de travail

La charge de travail est exprimée sous la forme d'un indice normalisé, calculé comme le rapport entre le temps effectivement mobilisé et la capacité journalière théorique :

Charge = heures travaillées / 8

Cet indicateur permet d'identifier les situations de surcharge, de sous-utilisation ou d'attente.

4.2.2 Structure interne du prototype

Le prototype est organisé autour de modules distincts :

- Module Task : définition des tâches, dépendances, timeline, versions.
- Module Agent : sélection et exécution autonome des tâches.
- Module Manager Agent : validation des phases et contrôle du processus.
- Moteur principal : gestion du temps simulé et mise à jour globale.
- Module Workload : calcul et analyse de la charge de travail.
- Capacité multi-projets : gestion simultanée de plusieurs projets et analyse de la charge cumulative.

4.2.3 Formalisation stricte des étapes et matrice d'habilitation (rôles → tâches)

Afin d'assurer une correspondance stricte entre le modèle théorique (chapitre 3) et l'expérimentation, le prototype intègre un catalogue normatif des étapes. Chaque étape est définie par :

1. sa phase (Analyse, Hypothèse, Dessin, Collaboration, Chantier),

2. un code unique (A1…E5),
3. un intitulé,
4. une matrice d'habilitation précisant quels rôles ont le droit d'exécuter l'étape.

Cette formalisation permet d'éviter un prototype uniquement "piloté par l'interface" : ici, les actions des agents (accepter/démarrer/terminer) sont contraintes par une règle d'autorisation, dérivée directement de la structure du workflow.

Remarque : la version complète (A1…E5) est placée en annexe afin de ne pas surcharger le corps du mémoire.

```swift
/// Étape du workflow : phase + code + libellé + rôles autorisés
struct StepDef {
    let phase: Phase
    let code: String     // A1…E5
    let name: String     // Libellé
    let allowed: Set<Role>
}

/// Catalogue normatif (extraits) : A1…E5 suivent la grille du mémoire
private let stepCatalog: [String: StepDef] = {
    func S(_ phase: Phase, _ code: String, _ name: String, _ allowed: [Role]) -> (String, StepDef)
        (code, StepDef(phase: phase, code: code, name: name, allowed: Set(allowed)))
    }
    var dict: [String: StepDef] = [:]
    for (k, v) in [
        // Phase 1 – Analyse (exemples)
        S(.analyse, "A3", "Travail sur PLU", [.b1, .b2, .c1, .c2]),
        S(.analyse, "A8", "NADI – Essai de toutes les possibilités de plan de masse", [.b2, .c1, .
        // Phase 3 – Dessin (exemple)
        S(.dessin, "C3", "Dessin simultané multi-utilisateurs", [.c1, .c2, .d1, .d2]),
        // Phase 5 – Chantier (exemple)
        S(.chantier, "E3", "Validation par responsable / MOA", [.b2]),
    ] { dict[k] = v }
    return dict
}()
```

Fig4.1 – Définition formelle des étapes du workflow et des règles d'autorisation associées

4.2.4 Contrôle d'accès et machine à états des tâches

Le prototype modélise l'avancement sous la forme d'une machine à états (En attente → Disponible → Acceptée → En cours → Terminée, avec Refusée comme branche alternative). Avant toute transition (accepter, démarrer, terminer), le système vérifie que le rôle de l'acteur appartient à l'ensemble autorisé par l'étape. En cas de non-conformité, l'action est refusée et une entrée est ajoutée au journal d'activité, garantissant traçabilité et auditabilité.

4.2.5 Génération automatique des tâches et capacité multi-projets

Pour rendre l'expérimentation reproductible, la création d'un projet déclenche automatiquement la génération de toutes les tâches correspondant aux étapes A1…E5. Le prototype instancie une tâche par étape, en conservant explicitement le stepCode, ce qui permet :

la construction de diagrammes temporels (Gantt),

la comparaison multi-projets,

la mesure de charge par agent,

l'analyse des dépendances par phase (verrouillage du passage à la phase suivante).

```
private func allowedRoles(for task: Task) -> Set<Role> {
    if let def = stepCatalog[task.stepCode] { return def.allowed }
    return []
}

private func roleAllowed(_ role: Role, for task: Task) -> Bool {
    allowedRoles(for: task).contains(role)
}

func accept(_ task: Task, by user: Architect?) {
    if let u = user, !roleAllowed(u.role, for: task) {
        log(action: "Accès refusé : rôle \(u.role.rawValue) non autorisé pour \(task.stepCode).",
        return
    }
    update(taskId: task.id) { t in
        guard t.status == .available || t.status == .pending else { return }
        t.status = .accepted
    }
    log(action: "Tâche acceptée", task: task, by: user)
}

func start(_ task: Task, by user: Architect?) {
    if let u = user, !roleAllowed(u.role, for: task) {
        log(action: "Accès refusé : rôle \(u.role.rawValue) non autorisé pour \(task.stepCode).",
        return
    }
    update(taskId: task.id) { t in
        guard t.status == .accepted || t.status == .available else { return }
        t.status = .inProgress
    }
    log(action: "Démarrage de la tâche", task: task, by: user)
}

func finish(_ task: Task, by user: Architect?) {
    if let u = user, !roleAllowed(u.role, for: task) {
        log(action: "Accès refusé : rôle \(u.role.rawValue) non autorisé pour \(task.stepCode).",
        return
    }
    update(taskId: task.id) { t in t.status = .done }
    log(action: "Tâche terminée et validée", task: task, by: user)
}
```

Fig 4-2Mécanisme de contrôle des droits d’exécution et de transition des états des tâches

```
func createProject(name: String, type: ProjectType, units: Int, manager: Architect) {
    let start = suggestedStartDateForNewProject()

    let proj = Project(
        name: name,
        type: type,
        units: units,
        startDate: start,
        endDate: start.addingDays(120),
        managerId: manager.id
    )
    projects.append(proj)

    // Génération des tâches A1…E5 selon le catalogue
    generateStandardTasks(for: proj, start: start, manager: manager)

    // Planification automatique sous contraintes (jours ouvrés / 8h / non fractionnable)
    autoScheduleAllTasksRespectingCapacityAndWeekends(for: proj)

    tasks = tasks.sorted { $0.startDate < $1.startDate }
}

private func generateStandardTasks(for project: Project, start: Date, manager: Architect) {
    let phaseOrder: [Phase] = [.analyse, .hypothese, .dessin, .collaboration, .chantier]
    var cursor = start

    for phase in phaseOrder {
        let defs = stepCatalog.values
            .filter { $0.phase == phase }
            .sorted { $0.code < $1.code }

        for def in defs {
            let days = 2 // (paramètre expérimental ; la table complète est en annexe)

            let task = Task(
                name: "\(project.name) - \(def.code) \(def.name)",
                projectId: project.id,
                phase: def.phase,
                subPhase: def.name,
                type: .work,
                stepCode: def.code,
                assignees: [],              // (initialisés ensuite par règles)
                reviewer: manager,          // relecteur par défaut = responsable projet
                estimatedHours: Double(days * 8),
                startDate: cursor,
                endDate: cursor.addingDays(days),
                status: .pending
            )
            tasks.append(task)
            cursor = cursor.addingDays(days + 1)
        }
    }
}
```

Figure 4.3 – Extrait du catalogue normatif des étapes du workflow et de la matrice d'habilitation des rôles

4.2.6 Ordonnancement automatique sous contraintes (jours ouvrés, 8h/jour, tâches indivisibles)

L'élément déterminant pour la validité expérimentale du prototype est l'ordonnancement automatique. Plutôt que d'attribuer des dates arbitraires, le moteur simule une logique réaliste d'agence en appliquant quatre contraintes :

- jours ouvrés uniquement (samedi/dimanche exclus),
- capacité maximale de 8h/jour/personne,
- tâches indivisibles (non préemptives : une tâche occupe un bloc continu de jours ouvrés),
- décalage automatique (si saturation, report au prochain créneau disponible).

Techniquement, ces contraintes sont gérées via un registre de capacité journalière (ledger) : pour chaque date, il mémorise les heures déjà consommées par chaque agent, tous projets confondus. Avant de placer une tâche, le système vérifie qu'un bloc de jours ouvrés consécutifs est disponible pour tous les exécutants.

```swift
// ledger : [date : [personId : usedHours]]
typealias Ledger = [Date: [UUID: Int]]

func autoScheduleAllTasksRespectingCapacityAndWeekends(for project: Project) {
    var ledger: Ledger = [:]
    preloadExistingTasksIntoLedger(&ledger, excludingProjectId: project.id)

    let orderedTasks = tasks
        .filter { $0.projectId == project.id }
        .sorted {
            if $0.phase.order != $1.phase.order { return $0.phase.order < $1.phase.order }
            return $0.stepCode < $1.stepCode
        }

    for t in orderedTasks {
        guard !t.assignees.isEmpty else { continue }

        let needDays = max(1, Int(ceil(t.estimatedHours / 8.0)))
        var candidateStart = nextWorkingDay(from: max(project.startDate, currentDate))

        while true {
            if canPlace(task: t, startAt: candidateStart, needDays: needDays, ledger: ledger) {
                let end = advanceWorkingDays(from: candidateStart, count: needDays - 1)

                update(taskId: t.id) { tt in
                    tt.startDate = candidateStart
                    tt.endDate = end
                }

                write(task: t, startAt: candidateStart, needDays: needDays, into: &ledger)
                break
            } else {
                candidateStart = nextWorkingDay(from: candidateStart.addingDays(1))
            }
        }
    }
}

func canPlace(task: Task, startAt: Date, needDays: Int, ledger: Ledger) -> Bool {
    let startWD = isWorkingDay(startAt) ? startAt : nextWorkingDay(from: startAt)
    var ok = true

    iterateWorkingDays(from: startWD, count: needDays) { day in
        let d = day.startOfDay()
        let dayMap = ledger[d] ?? [:]
        for p in task.assignees {
            let used = dayMap[p.id] ?? 0
            if used + 8 > p.dailyCapacityHours { ok = false; return }
        }
    }
    return ok
}

func write(task: Task, startAt: Date, needDays: Int, into ledger: inout Ledger) {
    let startWD = isWorkingDay(startAt) ? startAt : nextWorkingDay(from: startAt)

    iterateWorkingDays(from: startWD, count: needDays) { day in
        let d = day.startOfDay()
        var dayMap = ledger[d] ?? [:]
        for p in task.assignees {
            dayMap[p.id] = min(8, (dayMap[p.id] ?? 0) + 8)
        }
        ledger[d] = dayMap
    }
}
```

Fig 4-4 Planification automatique des tâches sous contraintes de capacité et de jours ouvrés

4.2.7 Journalisation des actions et exploitation analytique (Simulation / Rapport)

Chaque action significative (acceptation, démarrage, terminaison, accès refusé, etc.) est enregistrée dans un journal d'activité structuré (Activity). Ce registre constitue un matériau directement exploitable dans la modélisation déductive : regroupement par jour (Day 1, Day 2…), analyse des validations, comparaison inter-agents et identification de goulots d'étranglement.

```
struct Activity: Identifiable {
    let id = UUID()
    var date: Date
    var actorName: String
    var actorRole: String
    var projectName: String
    var taskName: String
    var action: String
    var taskId: UUID
}

private func log(action: String, task: Task, by user: Architect?, overrideDate: Date? = nil) {
    let date = overrideDate ?? currentDate
    let actorName = user?.name ?? "Système"
    let actorRole = user?.role.rawValue ?? "Automatique"
    let projName = project(for: task.projectId)?.name ?? "Projet inconnu"

    activities.append(Activity(
        date: date,
        actorName: actorName,
        actorRole: actorRole,
        projectName: projName,
        taskName: task.name,
        action: action,
        taskId: task.id
    ))
}
```

Fig 4-5 Système de journalisation des actions (Activity Log)

4.3 Présentation du prototype et déroulement de la simulation

Après avoir décrit l'architecture interne du prototype et les principes de fonctionnement du modèle de workflow, cette section présente le prototype tel qu'il a été effectivement mis en œuvre et manipulé.

Il ne s'agit pas encore d'analyser les résultats produits par la simulation, mais de montrer concrètement comment le système fonctionne, comment les acteurs interagissent avec l'application, et comment les différentes règles de workflow se traduisent dans l'interface et dans le déroulement du projet simulé.

Les captures d'écran présentées ci-après illustrent le fonctionnement opérationnel du prototype développé sous Xcode (Swift/SwiftUI). Elles permettent de comprendre :

- la structure générale de l'application,
- la manière dont les tâches sont visualisées et filtrées,
- les interactions possibles selon le rôle de l'architecte,
- la progression temporelle du projet jour par jour,
- ainsi que la visualisation des activités et de la charge de travail.

Cette présentation constitue une étape intermédiaire indispensable entre la conception du modèle (chapitres 3 et 4.1–4.2) et l'analyse des résultats de la simulation, qui sera développée dans la section suivante.

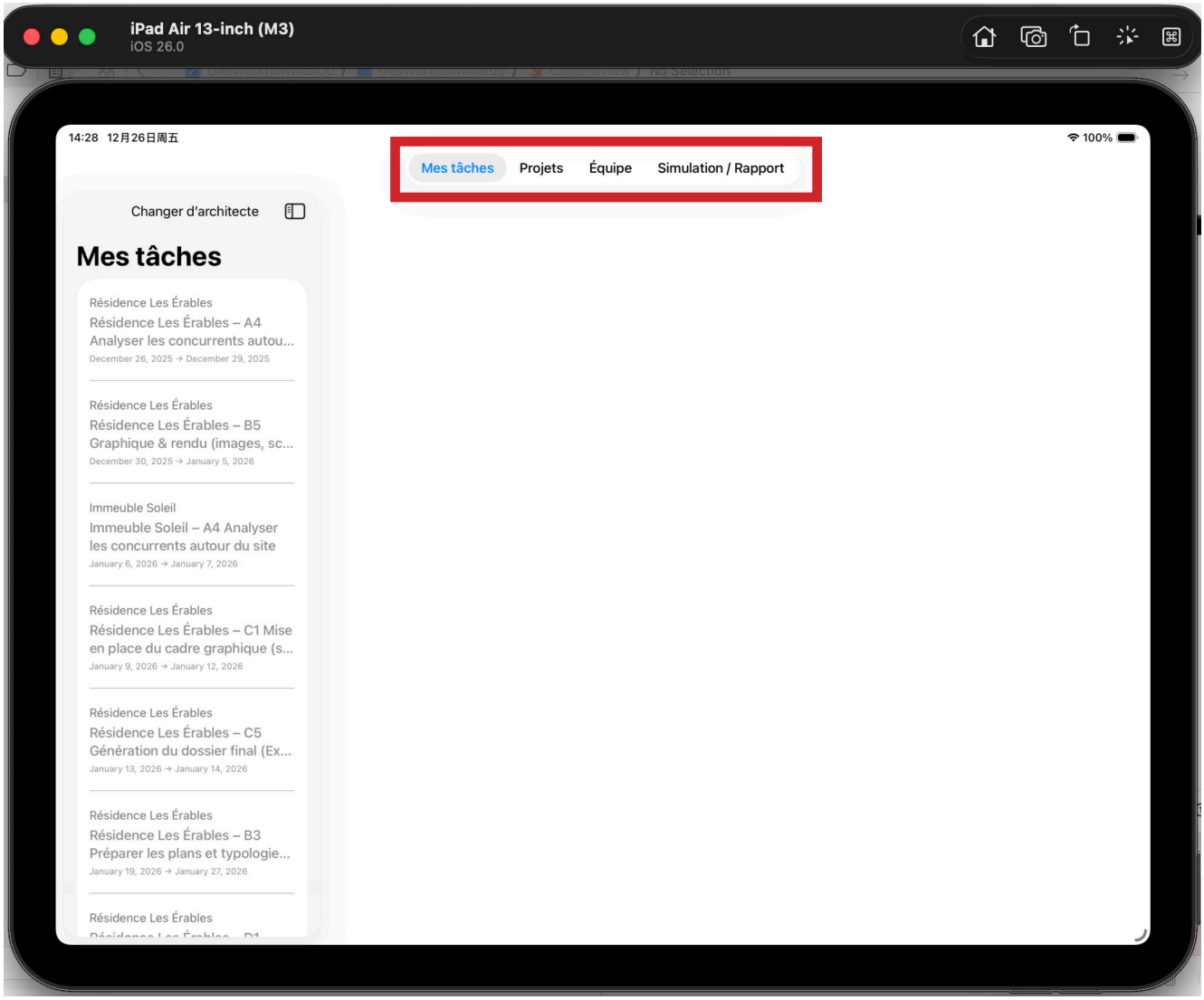
4.3.1 Vue générale de l'application et organisation en onglets

Le prototype est structuré autour de quatre onglets principaux, correspondant chacun à une dimension essentielle du workflow collaboratif :

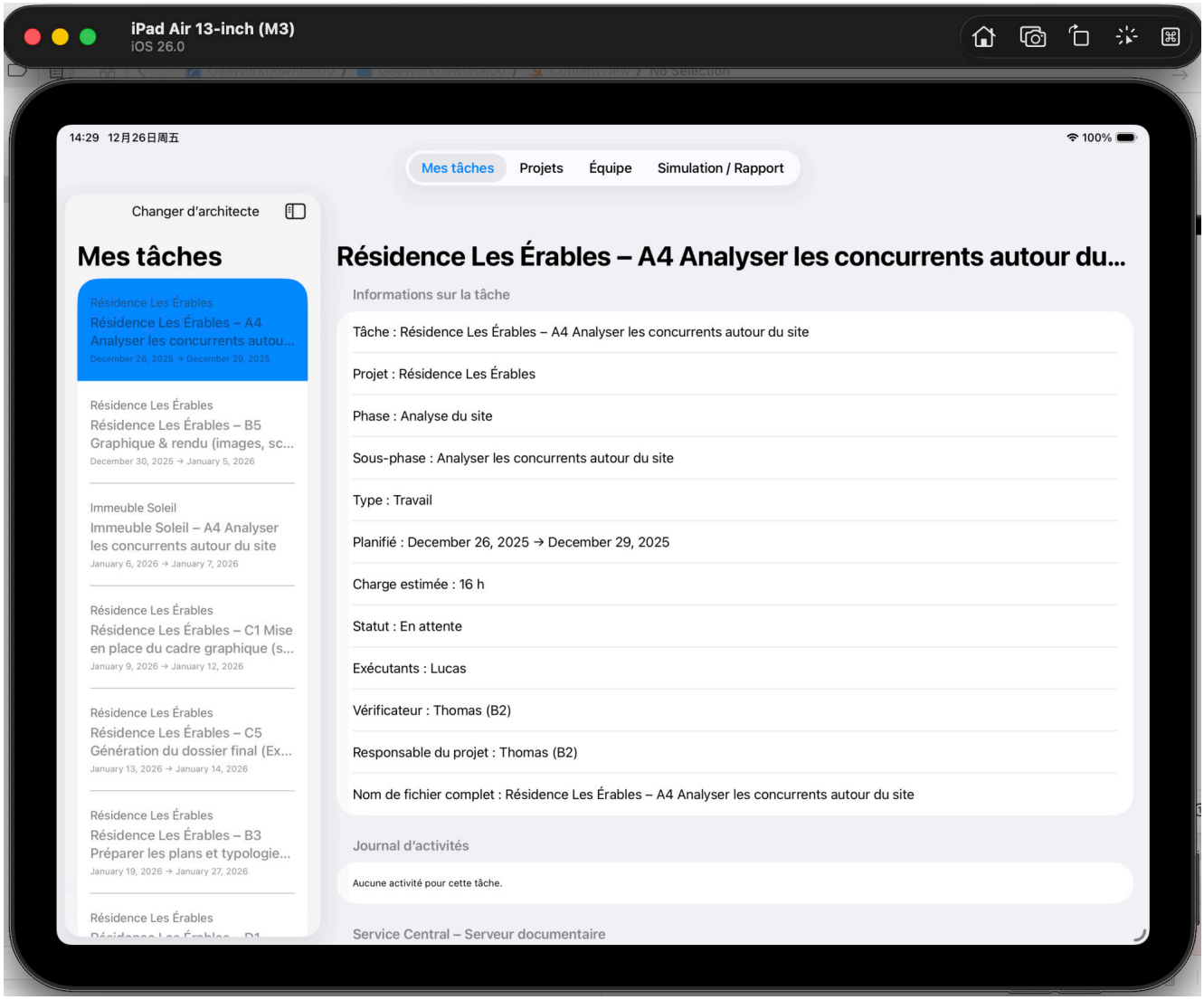
- Mes tâches : visualisation des tâches en cours pour l'architecte sélectionné, filtrées selon la phase active du projet et les règles de dépendance.

- Projets : vue synthétique des projets en cours, avec indication de l'avancement global et accès aux détails par phase.
- Équipe : représentation de la structure hiérarchique (Direction / groupe H / intervenants externes).
- Simulation / Rapport : interface de pilotage temporel permettant de faire avancer la simulation jour par jour et d'observer les actions générées.

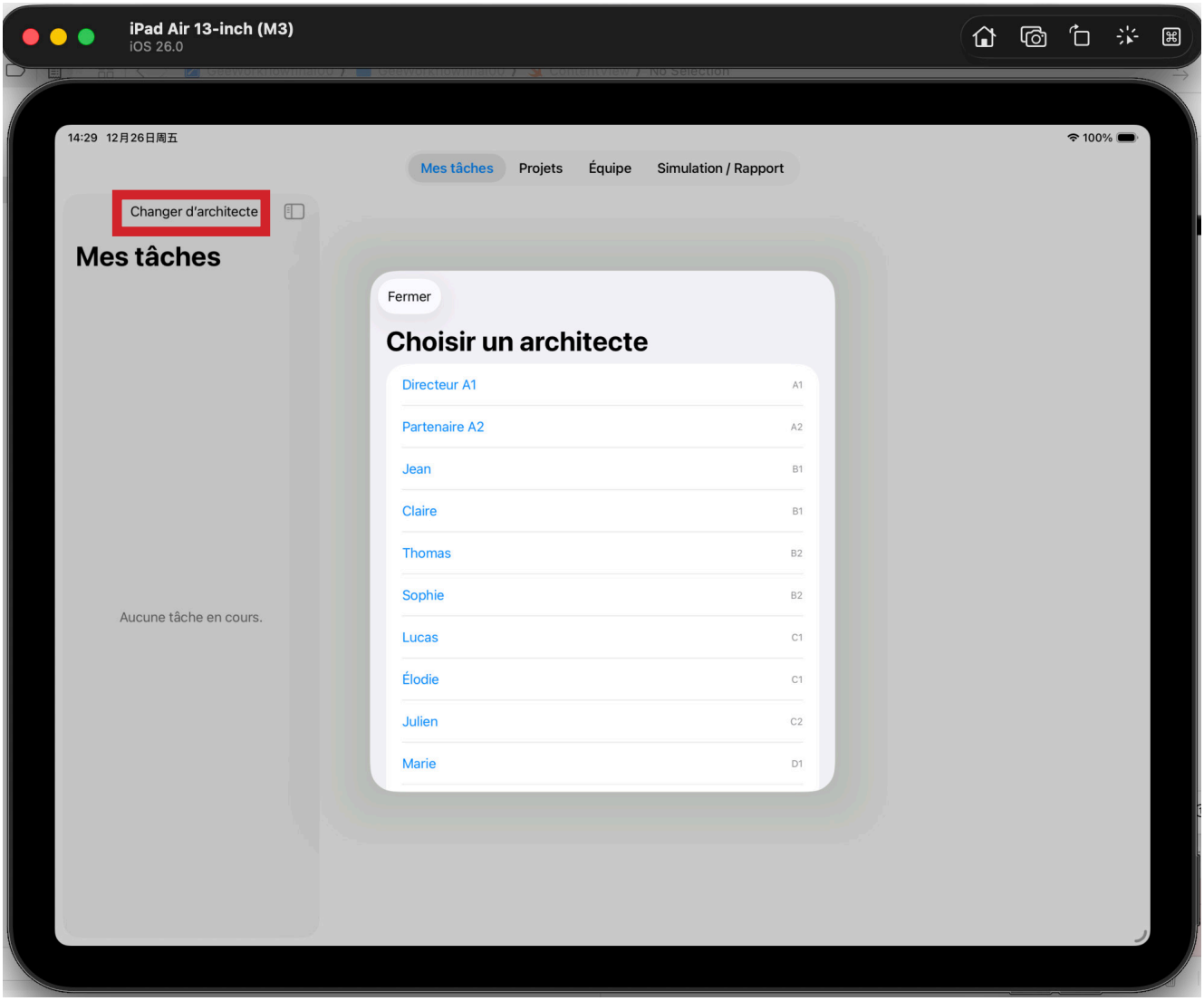
Les figures suivantes présentent ces différentes vues à travers des captures d'écran commentées.



Mes tâches : affiche la liste des tâches accessibles à l’architecte sélectionné, filtrées en fonction de la phase active du projet et des règles de dépendance. En sélectionnant une tâche, l’utilisateur accède à une vue détaillée permettant de consulter les attributs et l’état de cette tâche.



4.3.2 Sélection d’un agent et visualisation des tâches disponibles



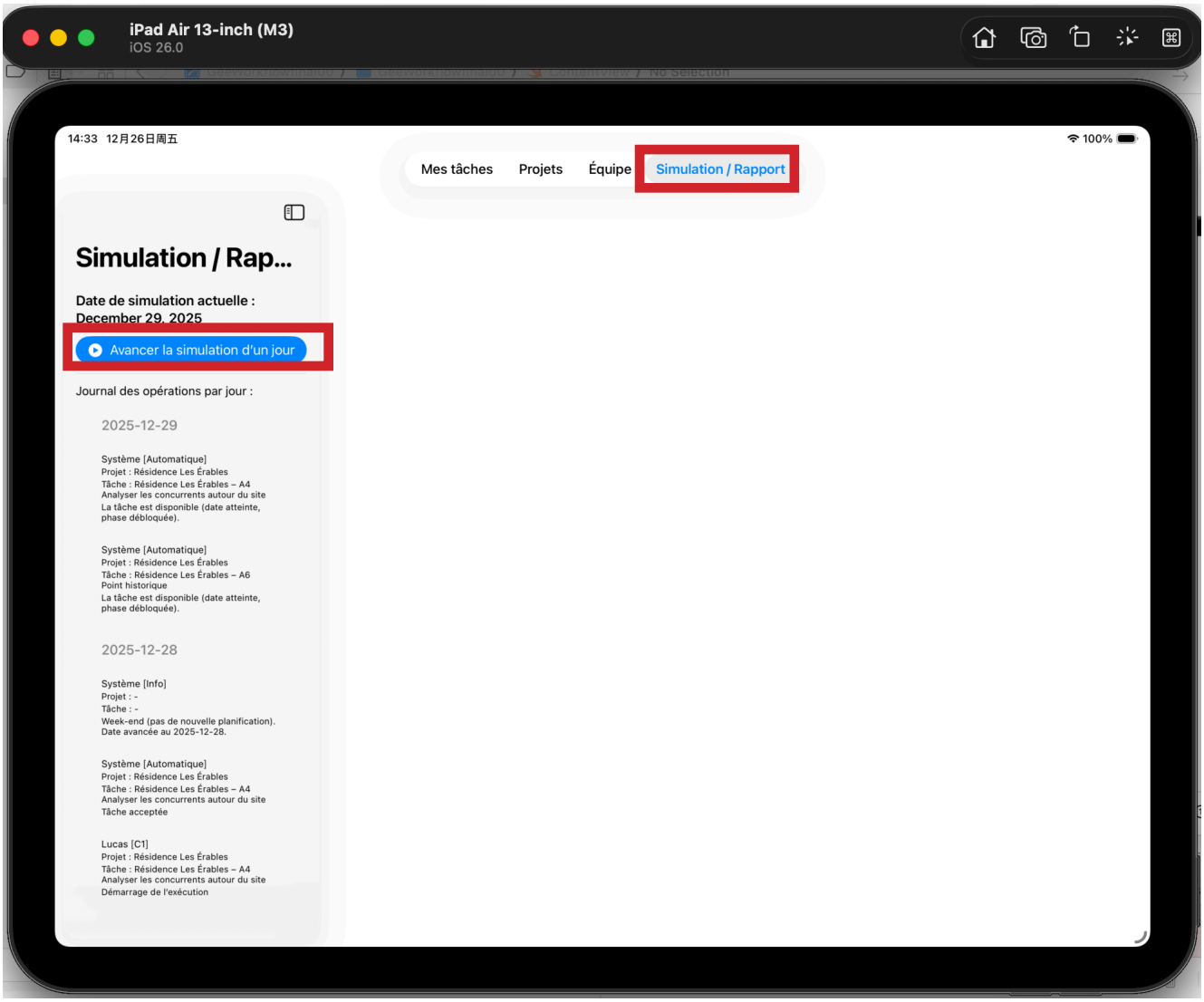
Cette vue montre la possibilité de simuler la connexion de différents architectes (C1, C2, D1, etc.).

Selon le rôle sélectionné, l’application filtre automatiquement les tâches visibles et les actions disponibles, en fonction :

- du niveau hiérarchique,
- des autorisations définies par le catalogue de workflow (A1–E5),
- de l’état d’avancement du projet.

Ainsi, un même projet génère des interfaces et des possibilités d’action différentes selon l’agent simulé.

4.3.3 Déroulement temporel de la simulation



La simulation progresse selon une temporalité discrète, basée sur des journées de travail simulées.

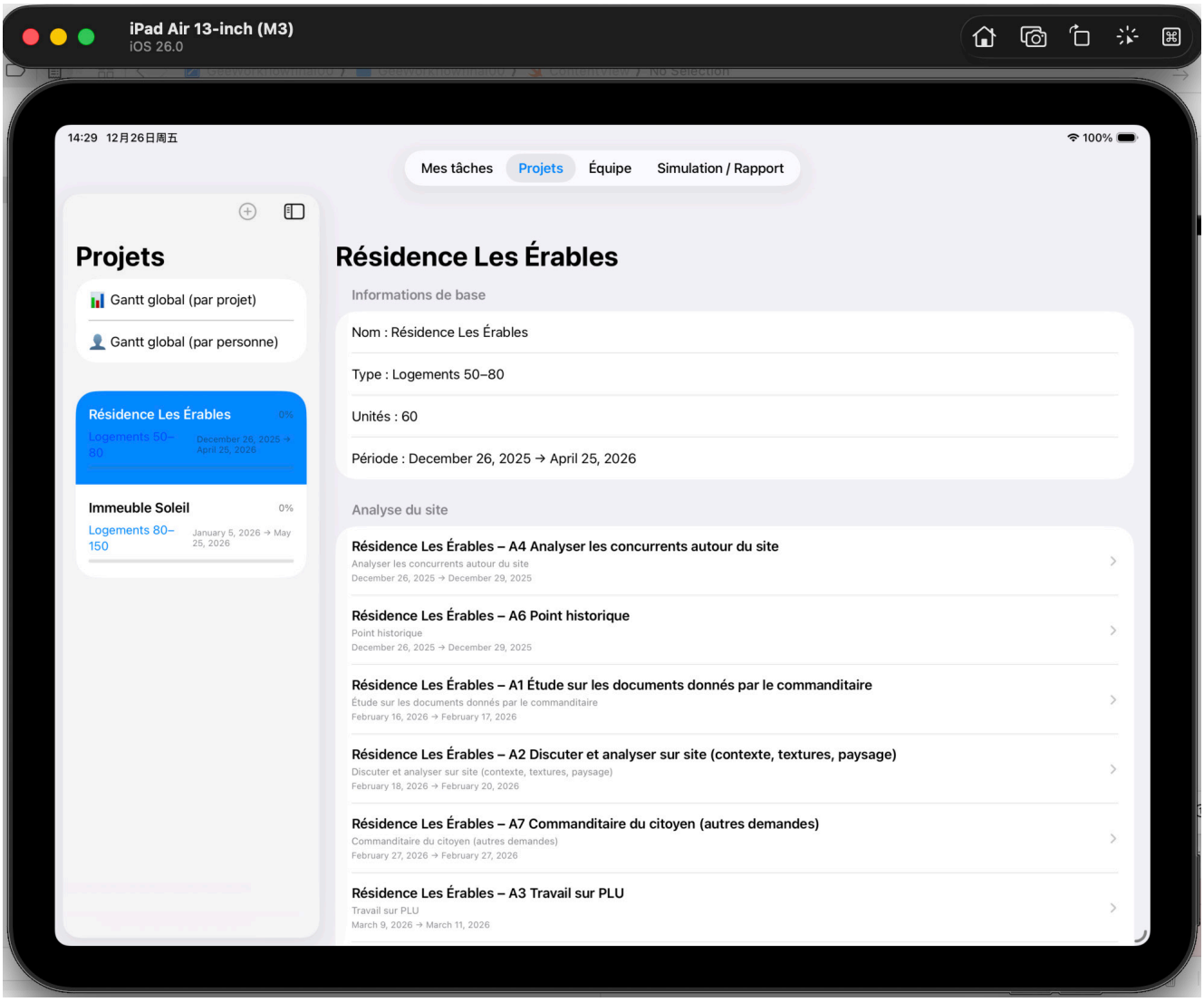
À chaque itération :

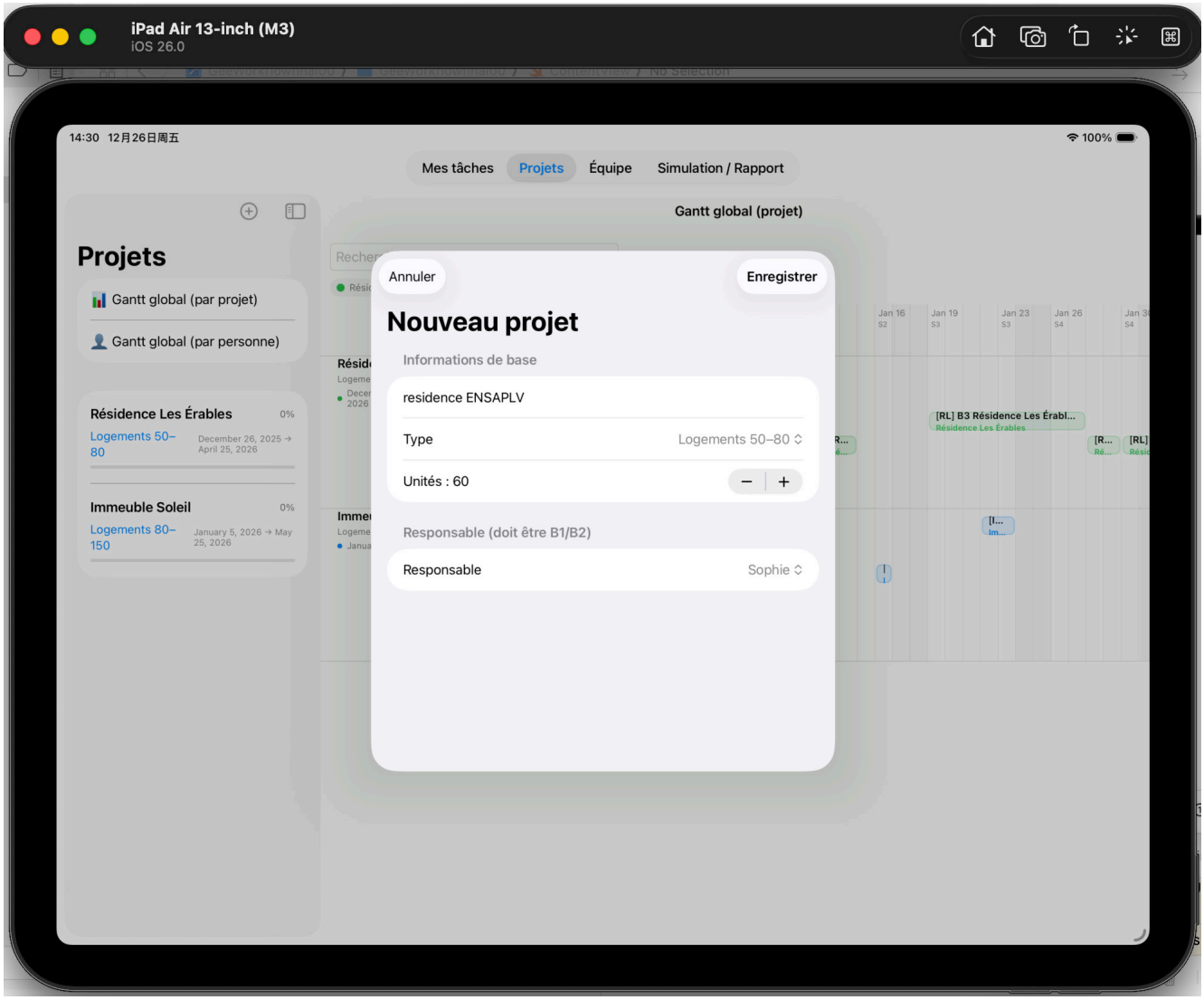
- les tâches deviennent disponibles lorsque leurs dépendances sont satisfaites,
- les agents sélectionnent et exécutent automatiquement des tâches compatibles,
- les activités sont enregistrées dans un journal horodaté.

Ce mécanisme permet d'observer la dynamique du projet dans le temps, indépendamment des décisions manuelles de l'utilisateur.

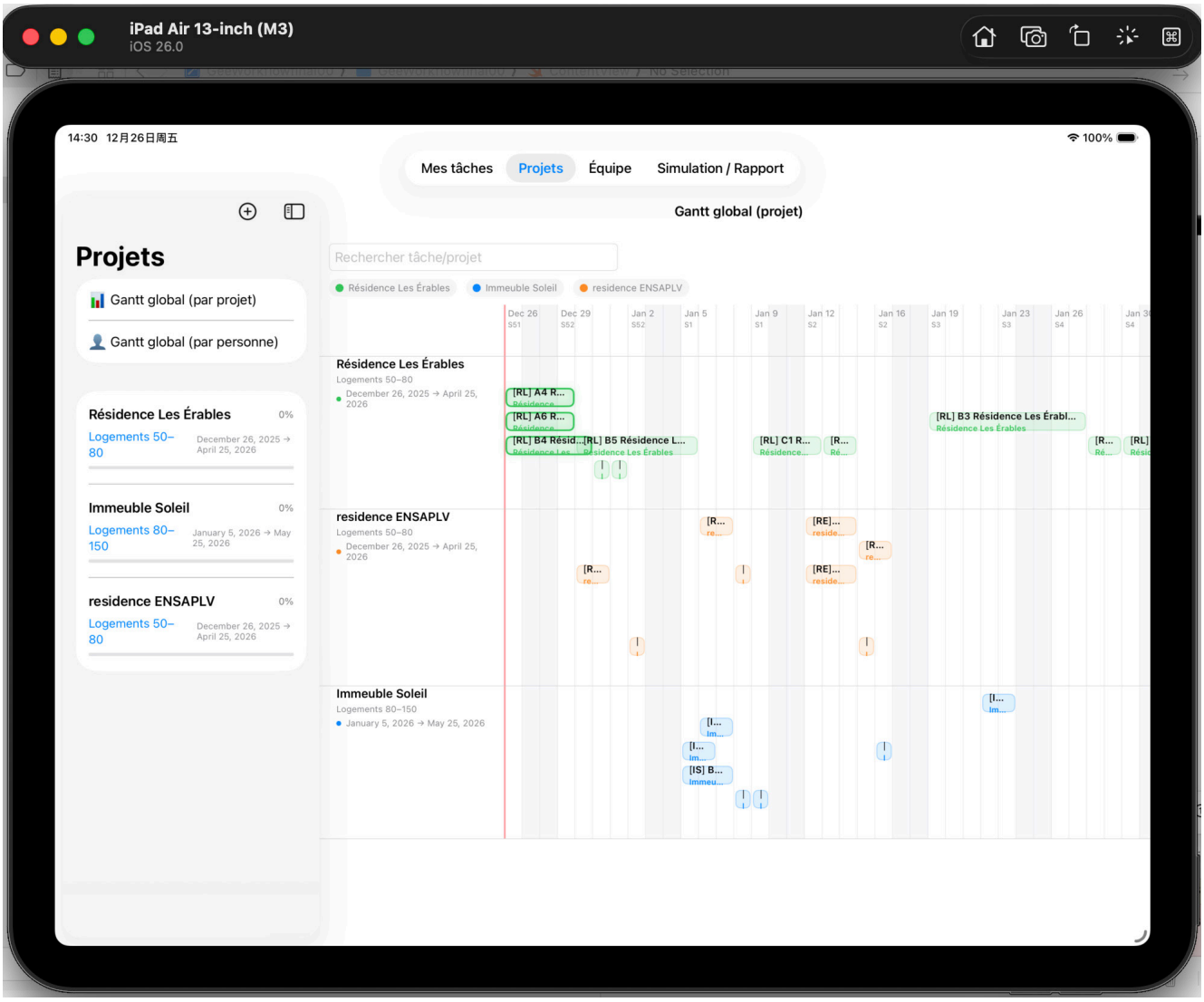
4.3.4 Visualisation multi-projets et charge de travail

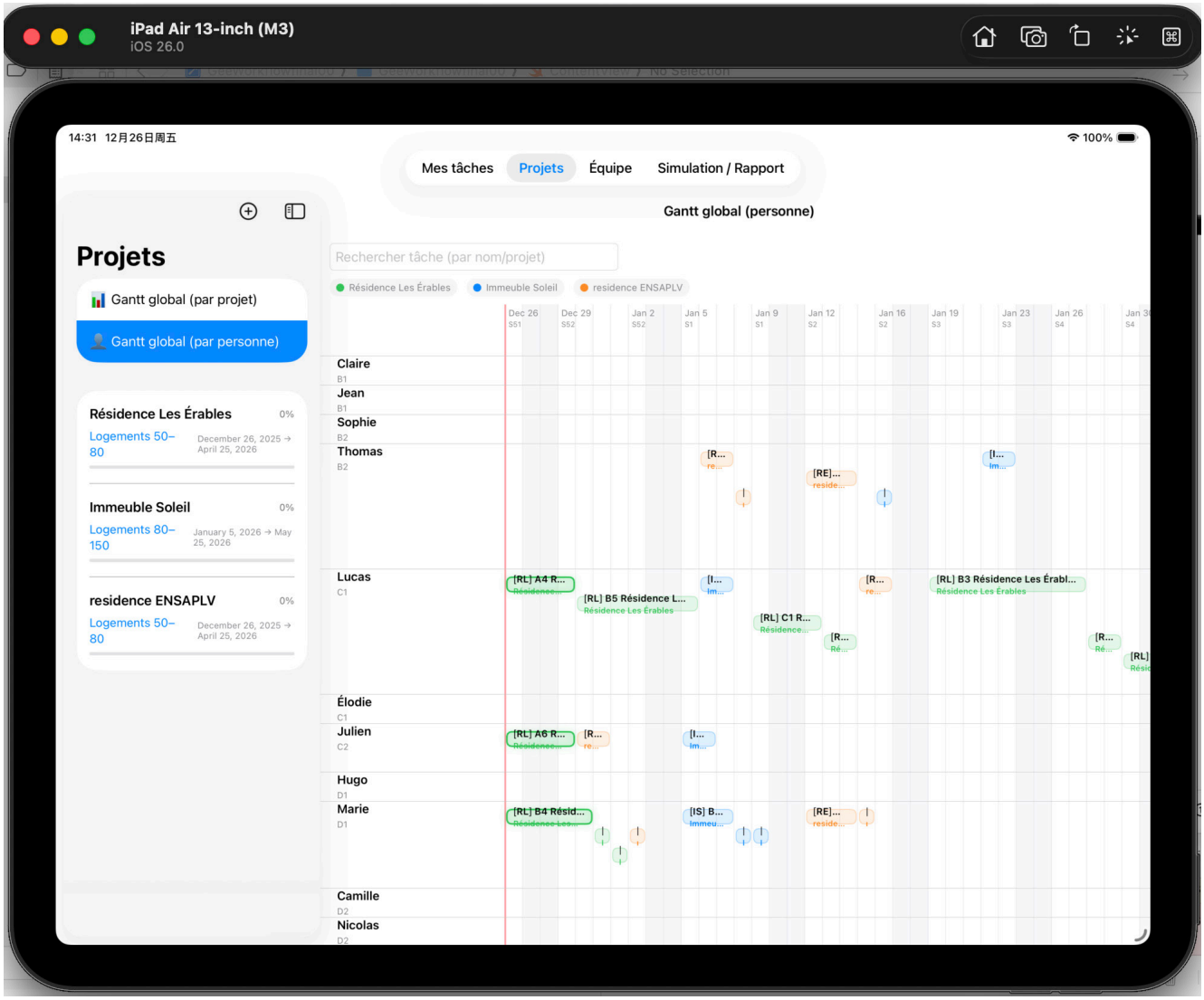
Dans le prototype, l'onglet « Projets » permet l'ajout libre de nouveaux projets, cette fonctionnalité étant réservée aux rôles de niveau A1 et A2. Lors de la création d'un projet, il est possible de désigner un responsable de projet ainsi que de définir le type de projet.




iPad Air 13-inch (M3)
iOS 26.0
14:30 12月26日周五
100%
Mes tâches
Projets
Équipe
Simulation / Rapport
Gantt global (projet)
Projets
Gantt global (par projet)
Gantt global (par personne)
Résidence Les Érables
0%
Logements 50–80
December 26, 2025 → April 25, 2026
Immeuble Soleil
0%
Logements 80–150
January 5, 2026 → May 25, 2026
Annuler
Enregistrer
Nouveau projet
Informations de base
residence ENSAPLV
Type
Logements 50–80
Unités : 60
Responsable (doit être B1/B2)
Responsable
Sophie
Jan 16
S2
Jan 19
S3
Jan 23
S3
Jan 26
S4
[RL] B3 Résidence Les Érabl...
Résidence Les Érables

Dans le suivi de l’avancement des projets, il est possible de consulter l’état d’avancement global, soit selon une organisation par projet, soit selon une organisation par acteur et par tâches.Le prototype permet de gérer plusieurs projets simultanément, reflétant la réalité des agences d’architecture.





Les tâches issues de différents projets entrent alors en concurrence pour les mêmes ressources humaines, ce qui met en évidence :

- les situations de surcharge individuelle,
- les périodes d'attente dues à des dépendances bloquantes,
- l'effet des règles de priorité et d'assistance (B2 vers C).

Ces éléments constituent la matière première de l'analyse développée dans la section suivante.

4.4 Analyse et interprétation des résultats de la simulation

La simulation réalisée à partir de ce prototype ne cherche pas à reproduire fidèlement l'ensemble de la complexité d'un projet architectural réel, mais à observer, au sein d'un système contrôlé, certains problèmes organisationnels récurrents du processus de conception architecturale. Les résultats doivent donc être compris comme des phénomènes structurels apparaissant progressivement au cours de l'expérimentation et de l'implémentation, plutôt que comme des conclusions quantitatives strictes.

Au fur et à mesure de la construction du prototype et du développement du code — notamment du système de tâches, du mécanisme de progression temporelle et de la logique multi-agents — plusieurs caractéristiques organisationnelles sont devenues plus lisibles. L'analyse s'appuie sur le journal d'activités ainsi que sur les représentations temporelles générées automatiquement (diagrammes de Gantt par projet et par acteur), permettant une lecture globale du processus selon la continuité du flux, la répartition de la charge de travail et le rôle effectif des différents niveaux hiérarchiques dans l'avancement du projet.

Un premier phénomène particulièrement visible concerne les architectes de niveau C (C1 / C2). Dans la version initiale du modèle, la majorité des tâches de conception centrales était attribuée à ces rôles, ce qui correspondait logiquement à leur position de concepteurs principaux. Toutefois, lors de l'exécution de la simulation, cette configuration a rapidement montré ses limites : les tâches s'accumulaient sur C1 / C2, entraînant des temps d'attente prolongés pour les tâches dépendantes et un ralentissement global du rythme du projet.

C'est précisément au cours de cette phase de tests et de débogage que le modèle a été ajusté, en introduisant un mécanisme plus souple permettant, sous certaines conditions, l'intervention des architectes de

niveau B sur des tâches initialement réservées aux niveaux C. Il convient de souligner que cette modification ne visait pas à remettre en cause la hiérarchie du projet, mais à instaurer une forme d'assistance contrôlée, destinée à soulager les points de surcharge critiques du processus.

Au cours du développement et des itérations successives du prototype, un autre type de difficulté est également apparu. Lorsque plusieurs tâches se répétaient ou se concentraient fortement au sein d'une même phase, la charge de travail de cette phase augmentait rapidement, provoquant des déséquilibres dans la répartition des ressources humaines.

Pour répondre à cette situation, les versions ultérieures du modèle ont intégré des ajustements liés à la date de modification des tâches ainsi qu'à la logique de répartition des ressources, afin de mieux prendre en compte, dans la dimension temporelle, les effets d'accumulation des tâches et d'adapter dynamiquement l'allocation du travail.

Les simulations menées après ces ajustements montrent que ces mécanismes permettent de réduire efficacement les blocages liés à la surcharge individuelle, d'améliorer la continuité du flux de travail et de se rapprocher davantage des pratiques observées en agence, notamment celles où un supérieur intervient ponctuellement pour soutenir un collaborateur en difficulté. À l'inverse, les rôles de direction (A1 / A2) demeurent volontairement exclus des tâches opérationnelles, leur intervention se limitant aux phases de validation et de prise de décision, ce qui apparaît de manière claire dans le fonctionnement du modèle.

L'ensemble de ces ajustements et itérations s'est étalé sur une dizaine de cycles de mise à jour environ, aboutissant à la version stabilisée du prototype présentée dans ce mémoire.

Par ailleurs, le journal d'activités s'est révélé être un outil essentiel pour l'analyse a posteriori. Chaque démarrage, achèvement ou validation de tâche étant enregistré, il est possible de reconstituer le déroulement du projet jour par jour après la simulation. Cette capacité de « relecture » rend visibles les points de blocage, les modifications

répétées et les phases de stagnation, et constitue un support direct pour la réflexion critique sur le fonctionnement du workflow.

Dans son ensemble, le processus de simulation s'apparente davantage à une démarche expérimentale itérative, marquée par des allers-retours constants entre conception, implémentation et ajustement. Le modèle n'a pas été entièrement défini dès le départ, mais s'est progressivement stabilisé à mesure que les problèmes apparaissaient au cours de l'exécution. Cette dynamique reflète, à sa manière, une réalité fréquemment rencontrée dans la pratique architecturale.

5.Conclusion

Ce mémoire s'est inscrit dans une interrogation centrale :

comment structurer et améliorer l'efficacité du travail collaboratif en architecture, dans un contexte où la complexité des projets, la multiplication des acteurs et la fragmentation des outils numériques rendent le processus de conception de plus en plus difficile à maîtriser ?

À partir d'une expérience professionnelle concrète au sein de l'agence Gee Design, ce travail a mis en évidence un décalage récurrent entre, d'une part, les intentions organisationnelles affichées (planning, hiérarchie, coordination), et d'autre part, la réalité quotidienne du travail architectural, souvent marquée par des blocages, des pertes d'information, des surcharges individuelles et une faible visibilité globale du processus.

Synthèse de la démarche

Afin de répondre à cette problématique, le mémoire a développé une approche en trois temps.

Dans un premier temps, une analyse critique du processus de conception architectural a permis de montrer que les difficultés rencontrées ne relèvent pas uniquement des outils de dessin ou des compétences individuelles, mais surtout de l'absence de structuration explicite du flux de travail : dépendances implicites, responsabilités floues, validations informelles et gestion fragmentée des versions.

Dans un second temps, cette analyse a conduit à la formulation d'un modèle théorique de workflow, structuré autour de phases clairement définies, de rôles hiérarchisés et de règles explicites d'enchaînement, de validation et d'archivage. Ce modèle ne cherche pas à rigidifier la conception, mais à fournir un cadre lisible et partagé, capable de soutenir la complexité du projet architectural.

Enfin, dans un troisième temps, ce modèle a été mis à l'épreuve

à travers la conception et l'implémentation d'un prototype d'application développé sous Xcode en Swift/SwiftUI. Ce prototype, conçu comme une simulation expérimentale, a permis de traduire le modèle théorique en un système opérationnel, intégrant des agents, des tâches, une timeline, des règles de permission et des mécanismes de visualisation.

Apports principaux du travail

L'un des apports majeurs de ce mémoire réside dans la démonstration qu'un workflow architectural peut être formalisé sans appauvrir la conception. Au contraire, la simulation montre que la clarification des rôles, des responsabilités et des dépendances contribue à réduire les blocages organisationnels et à améliorer la continuité du projet.

Le prototype met également en lumière l'importance de la dimension temporelle. En rendant visibles la charge de travail individuelle et collective, ainsi que les effets cumulatifs de plusieurs projets menés simultanément, l'outil révèle des déséquilibres souvent invisibles dans la pratique courante des agences. Cette visualisation constitue un support précieux pour la prise de décision, la planification et la discussion collective.

Par ailleurs, le recours à un modèle multi-agents permet de dépasser une vision purement linéaire du projet. Il rend compte d'une réalité essentielle du travail architectural : la coexistence de niveaux hiérarchiques distincts, la possibilité de coopération ponctuelle entre rôles, et la nécessité d'une flexibilité contrôlée, notamment lorsque certains acteurs deviennent des points de saturation du processus.

Limites et posture critique

Ce travail assume pleinement les limites de son dispositif. Le prototype repose sur une simulation, et non sur une expérimentation réelle en agence. Les comportements humains y sont modélisés par des règles, sans intégrer les dimensions subjectives, relationnelles ou politiques qui traversent inévitablement tout projet architectural.

Cependant, ces limites ne constituent pas une faiblesse du mémoire, mais un choix méthodologique. L'objectif n'était pas de reproduire fidèlement la réalité humaine, mais de mettre en évidence les effets structurels du flux de travail. À ce titre, le prototype fonctionne comme un outil d'analyse et de projection, capable de révéler des tendances organisationnelles et des points de tension.

Ouverture et perspectives

Les résultats de ce mémoire ouvrent plusieurs perspectives.

D'un point de vue professionnel, le prototype pourrait évoluer vers un outil d'aide à la décision, utilisé non pas comme un système contraignant, mais comme un support de dialogue au sein des agences : discussion sur la charge de travail, anticipation des phases critiques, réflexion collective sur l'organisation du projet.

D'un point de vue académique, cette recherche montre l'intérêt d'un dialogue entre architecture, sciences de l'organisation et technologies numériques. Elle invite à considérer le workflow non comme un simple outil de gestion, mais comme un véritable objet de projet, au même titre que l'espace, la structure ou le programme.

Enfin, à un niveau plus personnel, ce mémoire reflète une position assumée : celle d'un architecte qui refuse de se limiter à la production de formes, et qui considère l'organisation du travail comme une dimension essentielle de la qualité architecturale. Dans un contexte de transformation profonde du métier, cette réflexion constitue une tentative de reprendre la maîtrise des conditions mêmes de la conception.

Bibliographie

Hollingsworth D, Hampshire U. Workflow management coalition: The workflow reference model[J]. Document Number TC00-1003, 1995, 19(16): 224.

MDPI and ACS Style Mei, T.; Zhong, S.; Lan, H.; Guo, Z.; Qin, Y. Configuration Analysis of Integrated Project Delivery Principles' Obstacle to Construction Project Level of Collaboration. Sustainability 2023, 15, 3509. https://doi.org/10.3390/su15043509

Vishal Singh, Ning Gu, Xiangyu Wang,A theoretical framework of a BIM-based multi-disciplinary collaboration platform, Automation in Construction,Volume 20, Issue 2,2011,Pages 134-144, ISSN 0926-5805,

Annexes

L’application est structurée en trois modules principaux : ContentView, Models et WorkflowStore.
Le développement a été réalisé dans Xcode, avec l’appui de ChatGPT, en prenant en charge l’architecture ainsi que l’implémentation du code :
ContentView : responsable de l’interface utilisateur et des interactions ;
Models : définit les structures de données et les modèles métier ;
WorkflowStore : couche centrale de gestion d’état et de données, assurant le stockage, la mise à jour et l’orchestration de la logique de workflow.

Code de WorkflowStore :

```
import Foundation
import SwiftUI
import Combine

/// Définition des étapes: table stricte Phase/Étape/Rôle autorisé
struct StepDef {
  let phase: Phase
  let code: String   // A1…E5
  let name: String   // Nom affiché
  let allowed: Set<Role>
}

private let stepCatalog: [String: StepDef] = {
  func S(_ phase: Phase, _ code: String, _ name: String, _ allowed: [Role]) -> (String, StepDef) {
    (code, StepDef(phase: phase, code: code, name: name, allowed: Set(allowed)))
  }
  var dict: [String: StepDef] = [:]
  for (k, v) in [
    // Phase 1 – Analyse
    S(.analyse, "A1", "Étude sur les documents donnés par le commanditaire", [.b1, .b2, .c1, .c2]),
    S(.analyse, "A2", "Discuter et analyser sur site (contexte, textures, paysage)", [.b2, .c1, .c2]),
    S(.analyse, "A3", "Travail sur PLU", [.b1, .b2, .c1, .c2]),
    S(.analyse, "A4", "Analyser les concurrents autour du site", [.c1, .c2, .d1]),
    S(.analyse, "A5", "Possibilité du commerce", [.b2, .c1]),
    S(.analyse, "A6", "Point historique", [.c2, .d1]),
    S(.analyse, "A7", "Commanditaire du citoyen (autres demandes)", [.b2]),
    S(.analyse, "A8", "NADI – Essai de toutes les possibilités de plan de masse", [.b2, .c1, .c2, .d1, .d2]),
    S(.analyse, "A9", "Présentation (interne)", [.b1, .b2]),
    // Phase 2 – Hypothèse
    S(.hypothese, "B1", "Répartition des tâches", [.b1, .b2]),
    S(.hypothese, "B2", "Élaborer un concept architectural préliminaire", [.b2, .c1, .c2]),
    S(.hypothese, "B3", "Préparer les plans et typologies des logements", [.c1, .c2]),
    S(.hypothese, "B4", "Collecte des images et références visuelles", [.d1, .d2, .d3]),
    S(.hypothese, "B5", "Graphique & rendu (images, schémas, mise en page)", [.c1, .d2, .d3, .d4]),
    S(.hypothese, "B6", "Choix des matériaux proposés", [.b2, .c1]),
    S(.hypothese, "B7", "Présentation & validation MOA", [.b2]),
    // Phase 3 – Dessin
    S(.dessin, "C1", "Mise en place du cadre graphique (setup)", [.c1]),
    S(.dessin, "C2", "Attribution des tâches de dessin par modules", [.b2, .c1]),
    S(.dessin, "C3", "Dessin simultané multi-utilisateurs", [.c1, .c2, .d1, .d2]),
    S(.dessin, "C4", "Gel de version (Freeze) et retour en arrière", [.c1, .b2]),
    S(.dessin, "C5", "Génération du dossier final (Export)", [.c1, .d1]),
    // Phase 4 – Collaboration en conception
    S(.collaboration, "D1", "Transmission du dossier architectural", [.c1, .d1]),
    S(.collaboration, "D2", "Retour des avis techniques", [.c1, .c2]),
    S(.collaboration, "D3", "Modification & réponse par l’architecte", [.b2, .c1]),
    S(.collaboration, "D4", "Archivage & synchronisation des versions", [.d1]),
    // Phase 5 – Chantier
    S(.chantier, "E1", "Signalement des problèmes du chantier", [.c1]),
    S(.chantier, "E2", "Analyse & proposition de solution technique", [.b2, .c1]),
    S(.chantier, "E3", "Validation par responsable / MOA", [.b2]),
    S(.chantier, "E4", "Mise en œuvre & retour d’information", [.c1]),
    S(.chantier, "E5", "Archivage des décisions", [.d1]),
  ] { dict[k] = v }
  return dict
}()

/// Magasin de données global de l’app
class WorkflowStore: ObservableObject {

  // Architectes internes (groupe H, jusqu’à 12 personnes)
  @Published var architects: [Architect] = []

  // Autres spécialistes externes
  @Published var externalCollaborators: [ExternalCollaborator] = []

  // Projets & tâches
  @Published var projects: [Project] = []
  @Published var tasks: [Task] = []
```

```
  /// Utilisateur « connecté » (simulation)
  @Published var currentUser: Architect?

  /// Date simulée (détermine quelles tâches « arrivent à échéance »)
  @Published var currentDate: Date = Date()

  /// Journal d'activités: qui a fait quoi quel jour (onglet 2)
  @Published var activities: [Activity] = []

  /// Service documentaire centralisé (texte explicatif)
  let documentServiceDescription =
    "Tous les plans et documents des projets sont stockés et distribués via un « Service Central – Serveur documentaire » unifié.
" +
    "Les entrées et sorties de chaque tâche sont considérées comme des opérations de lecture/écriture sur ce service, afin de
soutenir la gestion des versions et la traçabilité des archives."

  init() {
    seedSampleData()
  }

  // MARK: - Outils: retrouver un projet par id

  func project(for id: UUID) -> Project? {
    projects.first(where: { $0.id == id })
  }

  /// Pour un projet, trouver la phase actuellement active
  func activePhase(for project: Project) -> Phase? {
    for phase in Phase.allCases {
      let phaseTasks = tasks.filter { $0.projectId == project.id && $0.phase == phase }
      if phaseTasks.contains(where: { $0.status != .done && $0.status != .refused }) {
        return phase
      }
    }
    return nil
  }

  // MARK: - Avancement global du projet (0...1)

  func progress(for project: Project) -> Double {
    let projectTasks = tasks.filter { $0.projectId == project.id }
    guard !projectTasks.isEmpty else { return 0.0 }
    let doneCount = projectTasks.filter { $0.status == .done }.count
    return Double(doneCount) / Double(projectTasks.count)
  }

  // MARK: - Autorisations

  private func allowedRoles(for task: Task) -> Set<Role> {
    if let def = stepCatalog[task.stepCode] {
      if def.phase != task.phase {
        log(action: "Avertissement: la tâche \(task.name) a un stepCode(\(task.stepCode)) dont la phase ne correspond pas à la
phase de la tâche.", task: task, by: nil)
      }
      return def.allowed
    } else {
      return []
    }
  }

  private func roleAllowed(_ role: Role, for task: Task) -> Bool {
    allowedRoles(for: task).contains(role)
  }

  // MARK: - Groupe H

  var hGroupMembers: [Architect] {
    architects.filter { $0.isHGroupMember }
  }

  var isHGroupFull: Bool {
    hGroupMembers.count >= 12
  }

  // MARK: - Mutations d'état des tâches (avec contrôle d'accès + logs)

  func accept(_ task: Task, by user: Architect?) {
    if let u = user, !roleAllowed(u.role, for: task) {
      log(action: "Tentative d'accepter la tâche, mais le rôle \(u.role.rawValue) n'est pas autorisé pour l'étape \(task.stepCode).",
task: task, by: user)
      return
    }
    update(taskId: task.id) { t in
      guard t.status == .available || t.status == .pending else { return }
      t.status = .accepted
    }
    log(action: "Tâche acceptée", task: task, by: user)
  }

  func refuse(_ task: Task, by user: Architect?) {
    update(taskId: task.id) { t in
      guard t.status == .available || t.status == .accepted || t.status == .pending else { return }
```

```
      t.status = .refused
    }
    log(action: "Tâche refusée", task: task, by: user)
  }

  func start(_ task: Task, by user: Architect?) {
    if let u = user, !roleAllowed(u.role, for: task) {
      log(action: "Tentative de démarrer la tâche, mais le rôle \(u.role.rawValue) n'est pas autorisé pour l'étape \(task.stepCode).",
task: task, by: user)
      return
    }
    update(taskId: task.id) { t in
      guard t.status == .accepted || t.status == .available else { return }
      t.status = .inProgress
    }
    log(action: "Démarrage de l'exécution", task: task, by: user)
  }

  func finish(_ task: Task, by user: Architect?) {
    if let u = user, !roleAllowed(u.role, for: task) {
      log(action: "Tentative de terminer la tâche, mais le rôle \(u.role.rawValue) n'est pas autorisé pour l'étape \(task.stepCode).",
task: task, by: user)
      return
    }
    update(taskId: task.id) { t in
      t.status = .done
    }
    log(action: "Tâche terminée et validée", task: task, by: user)
  }

  func addAssignee(_ architect: Architect, to task: Task, by user: Architect?) {
    guard architect.isHGroupMember && architect.role.canDraw else {
      log(action: "Tentative d'ajouter un membre hors groupe H ou non-dessinateur — opération bloquée.", task: task, by: user)
      return
    }
    guard roleAllowed(architect.role, for: task) else {
      log(action: "Tentative d'ajouter \(architect.name) (\(architect.role.rawValue)) à l'étape \(task.stepCode), mais ce rôle n'est pas
autorisé.", task: task, by: user)
      return
    }
    if !hGroupMembers.contains(where: { $0.id == architect.id }) && isHGroupFull {
      log(action: "Tentative d'ajouter un nouveau membre, mais le groupe H est complet (12 personnes) — opération bloquée.",
task: task, by: user)
      return
    }

    update(taskId: task.id) { t in
      if !t.assignees.contains(where: { $0.id == architect.id }) {
        t.assignees.append(architect)
      }
    }
    log(action: "Ajout de \(architect.name) à la tâche", task: task, by: user)
  }

  // MARK: - Gestion projet (création/suppression réservée A1/A2)

  func canEditProjects(current: Architect?) -> Bool {
    guard let u = current else { return false }
    return u.role.isDirectorLevel
  }

  /// Créer un projet: génère automatiquement toutes les étapes
  func createProject(name: String, type: ProjectType, units: Int, manager: Architect) {
    let start = suggestedStartDateForNewProject()
    let proj = Project(
      name: name,
      type: type,
      units: units,
      startDate: start,
      endDate: start.addingDays(120),
      managerId: manager.id
    )
    projects.append(proj)
    logGlobal("Nouveau projet « \(name) » créé, responsable: \(manager.name). Date d'intervention suggérée: \
(formattedDate(start))")

    // Génération des tâches standard (basée sur stepCatalog)
    generateStandardTasks(for: proj, start: start, manager: manager)

    // Planification automatique (jours ouvrés, 8h/pers/j, non fractionnable, décalage si nécessaire)
    autoScheduleAllTasksRespectingCapacityAndWeekends(for: proj)

    // Tri pour l'affichage UI
    tasks = tasks.sorted { $0.startDate < $1.startDate }
  }

  func deleteProject(_ project: Project, by user: Architect?) {
    guard canEditProjects(current: user) else { return }
    projects.removeAll { $0.id == project.id }
    tasks.removeAll { $0.projectId == project.id }
```

```
    logGlobal("Projet « \(project.name) » supprimé.")
  }

  /// Date de début suggérée pour un nouveau projet
  private func suggestedStartDateForNewProject() -> Date {
    let calendar = Calendar.current
    let maxDaysAhead = 60
    let peopleCount = max(architects.count, 1)

    func loadIndex(on day: Date) -> Double {
      let activeTasks = tasks.filter {
        $0.status != .done && $0.status != .refused &&
        $0.startDate <= day && $0.endDate >= day
      }
      return Double(activeTasks.count) / Double(peopleCount)
    }

    var candidate = currentDate

    for offset in 0..<maxDaysAhead {
      guard let day = calendar.date(byAdding: .day, value: offset, to: currentDate) else { continue }
      let weekday = calendar.component(.weekday, from: day)
      if weekday == 1 || weekday == 7 { continue }
      if loadIndex(on: day) < 2.0 {
        candidate = day
        break
      }
    }

    let weekday = calendar.component(.weekday, from: candidate)
    if weekday == 1 || weekday == 7 {
      let daysToAdd = weekday == 7 ? 2 : 1
      return candidate.addingDays(daysToAdd)
    }
    return candidate
  }

  private func formattedDate(_ date: Date) -> String {
    let df = DateFormatter()
    df.locale = Locale(identifier: "fr_FR")
    df.dateFormat = "yyyy-MM-dd"
    return df.string(from: date)
  }

  // MARK: - Charge de travail

  func workload(for architect: Architect) -> Int {
    tasks.filter {
      $0.assignees.contains(where: { $0.id == architect.id }) &&
      $0.status != .done &&
      $0.status != .refused
    }.count
  }

  private func isFreeToday(_ person: Architect, on day: Date) -> Bool {
    tasks.first(where: { t in
      t.status == .inProgress &&
      t.assignees.contains(where: { $0.id == person.id }) &&
      t.startDate <= day && t.endDate >= day
    }) == nil
  }

  private func pickWorker(for task: Task, on day: Date) -> Architect? {
    let allowed = allowedRoles(for: task)

    let cCandidates = architects.filter { ($0.role == .c1 || $0.role == .c2) && allowed.contains($0.role) && task.assignees.
contains($0) }
    let freeC = cCandidates.filter { isFreeToday($0, on: day) }
    if let bestC = freeC.min(by: { workload(for: $0) < workload(for: $1) }) {
      return bestC
    }

    let b2All = architects.filter { $0.role == .b2 && allowed.contains(.b2) }
    let b2Assigned = b2All.filter { task.assignees.contains($0) }
    let freeB2Assigned = b2Assigned.filter { isFreeToday($0, on: day) }
    if let bestB2 = freeB2Assigned.min(by: { workload(for: $0) < workload(for: $1) }) {
      return bestB2
    }

    let freeB2Any = b2All.filter { isFreeToday($0, on: day) }
    if let bestB2Any = freeB2Any.min(by: { workload(for: $0) < workload(for: $1) }) {
      return bestB2Any
    }

    let others = task.assignees.filter { allowed.contains($0.role) && isFreeToday($0, on: day) }
    return others.min(by: { workload(for: $0) < workload(for: $1) })
  }

  // MARK: - Simulation: avancer d'un jour (auto-acceptation + exécution exclusive + soutien B2 équilibré)

  func advanceOneDay() {
    let calendar = Calendar.current
```

```
  let newDate = currentDate.addingDays(1)

  let weekday = calendar.component(.weekday, from: newDate)
  if weekday == 1 || weekday == 7 {
    currentDate = newDate
    logGlobal("Week-end (pas de nouvelle planification). Date avancée au \(formattedDate(newDate)).")
    return
  }

  for idx in tasks.indices {
    guard tasks[idx].status == .pending else { continue }
    guard tasks[idx].startDate <= newDate else { continue }
    guard
      let proj = project(for: tasks[idx].projectId),
      let active = activePhase(for: proj),
      tasks[idx].phase == active
    else { continue }

    tasks[idx].status = .available
    log(action: "La tâche est disponible (date atteinte, phase débloquée).",
      task: tasks[idx],
      by: nil,
      overrideDate: newDate)
  }

  var schedulable = tasks.enumerated().compactMap { (i, t) -> (Int, Task)? in
    guard (t.status == .available || t.status == .accepted),
       t.startDate <= newDate,
       let proj = project(for: t.projectId),
       let active = activePhase(for: proj),
       t.phase == active
    else { return nil }
    return (i, t)
  }

  schedulable.sort { a, b in
    if a.1.startDate != b.1.startDate { return a.1.startDate < b.1.startDate }
    return a.1.name < b.1.name
  }

  var busyToday: Set<UUID> = Set(
    tasks
      .filter { $0.status == .inProgress && $0.startDate <= newDate && $0.endDate >= newDate }
      .flatMap { $0.assignees.map(\.id) }
  )

  for (indexInArray, task) in schedulable {
    let alreadyRunning = tasks[indexInArray].status == .inProgress &&
              tasks[indexInArray].startDate <= newDate &&
              tasks[indexInArray].endDate >= newDate
    if alreadyRunning { continue }

    let anyAssigneeBusy = task.assignees.contains { busyToday.contains($0.id) }
    if anyAssigneeBusy { continue }

    if let worker = pickWorker(for: task, on: newDate),
      !busyToday.contains(worker.id),
      roleAllowed(worker.role, for: task) {

      if !task.assignees.contains(where: { $0.id == worker.id }) {
        addAssignee(worker, to: task, by: nil)
      }

      if tasks[indexInArray].status == .available {
        accept(tasks[indexInArray], by: nil)
      }
      start(tasks[indexInArray], by: worker)
      busyToday.insert(worker.id)
    }
  }

  for idx in tasks.indices {
    guard tasks[idx].status == .inProgress else { continue }
    if tasks[idx].endDate <= newDate {
      finish(tasks[idx], by: nil)
    }
  }

  currentDate = newDate

  autoAssignBHelpersIfNeeded()
}

// MARK: - Journal d'activité

private func log(action: String,
         task: Task,
         by user: Architect?,
         overrideDate: Date? = nil) {
```

```
  let date = overrideDate ?? currentDate
  let actorName: String
  let actorRole: String

  if let u = user {
    actorName = u.name
    actorRole = u.role.rawValue
  } else {
    actorName = "Système"
    actorRole = "Automatique"
  }

  let projName = project(for: task.projectId)?.name ?? "Projet inconnu"

  let entry = Activity(
    date: date,
    actorName: actorName,
    actorRole: actorRole,
    projectName: projName,
    taskName: task.name,
    action: action,
    taskId: task.id
  )
  activities.append(entry)
}

private func logGlobal(_ message: String) {
  let entry = Activity(
    date: currentDate,
    actorName: "Système",
    actorRole: "Info",
    projectName: "-",
    taskName: "-",
    action: message,
    taskId: UUID()
  )
  activities.append(entry)
}

// MARK: - Utilitaire privé: modification d'une tâche

private func update(taskId: UUID, mutate: (inout Task) -> Void) {
  guard let idx = tasks.firstIndex(where: { $0.id == taskId }) else { return }
  var t = tasks[idx]
  mutate(&t)
  tasks[idx] = t
}

// MARK: - Données d'exemple (groupe H, deux projets)

private func seedSampleData() {
  // Direction (hors groupe H)
  let a1 = Architect(name: "Directeur A1", role: .a1)
  let a2 = Architect(name: "Partenaire A2", role: .a2)

  // Groupe H
  let b1_1 = Architect(name: "Jean",  role: .b1)
  let b1_2 = Architect(name: "Claire", role: .b1)
  let b2_1 = Architect(name: "Thomas", role: .b2)
  let b2_2 = Architect(name: "Sophie", role: .b2)
  let c1_1 = Architect(name: "Lucas",  role: .c1)
  let c1_2 = Architect(name: "Élodie", role: .c1)
  let c2_1 = Architect(name: "Julien", role: .c2)
  let d1_1 = Architect(name: "Marie",  role: .d1)
  let d1_2 = Architect(name: "Hugo",   role: .d1)
  let d2_1 = Architect(name: "Camille", role: .d2)
  let d2_2 = Architect(name: "Nicolas", role: .d2)
  let d3_1 = Architect(name: "Anaïs",  role: .d3)

  architects = [
    a1, a2,
    b1_1, b1_2,
    b2_1, b2_2,
    c1_1, c1_2, c2_1,
    d1_1, d1_2, d2_1, d2_2,
    d3_1
  ]

  currentUser = c1_1

  externalCollaborators = [
    ExternalCollaborator(name: "BE Structure X", specialty: "Structure"),
    ExternalCollaborator(name: "BE CVC-Y",    specialty: "CVC / Électricité")
  ]

  let start = Date()
  currentDate = start

  // Projets initiaux
  let p1 = Project(
    name: "Résidence Les Érables",
    type: .smallHousing,
```

```
    units: 60,
    startDate: start,
    endDate: start.addingDays(120),
    managerId: b2_1.id
  )

  let p2 = Project(
    name: "Immeuble Soleil",
    type: .mediumHousing,
    units: 120,
    startDate: start.addingDays(10),
    endDate: start.addingDays(150),
    managerId: b1_1.id
  )

  projects = [p1, p2]

  // Générer toutes les tâches pour l'exemple (affichage Gantt)
  generateStandardTasks(for: p1, start: p1.startDate, manager: b2_1)
  generateStandardTasks(for: p2, start: p2.startDate, manager: b1_1)

  // Planification automatique pour les exemples
  autoScheduleAllTasksRespectingCapacityAndWeekends(for: p1)
  autoScheduleAllTasksRespectingCapacityAndWeekends(for: p2)

  // Tri
  tasks = tasks.sorted { $0.startDate < $1.startDate }
}

// MARK: - Génération automatique des tâches standard (durées par défaut + exécuteurs)

private func generateStandardTasks(for project: Project, start: Date, manager: Architect) {
  // Pools de rôles
  let b1s = architects.filter { $0.role == .b1 }
  let b2s = architects.filter { $0.role == .b2 }
  let c1s = architects.filter { $0.role == .c1 }
  let c2s = architects.filter { $0.role == .c2 }
  let d1s = architects.filter { $0.role == .d1 }
  let d2s = architects.filter { $0.role == .d2 }
  let d3s = architects.filter { $0.role == .d3 }
  let d4s = architects.filter { $0.role == .d4 }

  func any(_ list: [Architect]) -> Architect? { list.randomElement() }
  func anyAllowed(_ allowed: Set<Role>) -> Architect? {
    // Priorité simple: B2 > C1 > C2 > B1 > D1 > D2 > D3 > D4
    let pools: [[Architect]] = [
      b2s, c1s, c2s, b1s, d1s, d2s, d3s, d4s
    ]
    for pool in pools {
      if let pick = pool.first, allowed.contains(pick.role) { return pick }
    }
    return nil
  }

  // Durées par défaut (jours)
  let defaultDurations: [String: Int] = [
    // A*
    "A1": 2, "A2": 3, "A3": 3, "A4": 2, "A5": 2, "A6": 2, "A7": 1, "A8": 7, "A9": 1,
    // B*
    "B1": 1, "B2": 5, "B3": 7, "B4": 3, "B5": 5, "B6": 2, "B7": 1,
    // C*
    "C1": 2, "C2": 2, "C3": 15, "C4": 2, "C5": 2,
    // D*
    "D1": 2, "D2": 3, "D3": 4, "D4": 1,
    // E*
    "E1": 2, "E2": 3, "E3": 1, "E4": 3, "E5": 1
  ]

  // Ordre des phases
  let phaseOrder: [Phase] = [.analyse, .hypothese, .dessin, .collaboration, .chantier]

  var cursor = start

  for phase in phaseOrder {
    let defs = stepCatalog.values
      .filter { $0.phase == phase }
      .sorted { $0.code < $1.code }

    for def in defs {
      let days = defaultDurations[def.code] ?? 2

      // Exécuteurs par défaut
      var assignees: [Architect] = []
      if let main = anyAllowed(def.allowed) {
        assignees.append(main)
      }
      // Étapes typiquement multi-exécuteurs
      if ["A8", "B5", "C3"].contains(def.code) {
        if let extra1 = anyAllowed(def.allowed), !assignees.contains(extra1) { assignees.append(extra1) }
        if let extra2 = anyAllowed(def.allowed), !assignees.contains(extra2) { assignees.append(extra2) }
```

```
        }

        // Vérificateur: manager du projet si son rôle le permet, sinon B1/B2
        let reviewer: Architect? = {
          if manager.role.isManagerLevel { return manager }
          return any(b2s) ?? any(b1s)
        }()

        let task = Task(
          name: "\(project.name) – \(def.code) \(def.name)",
          projectId: project.id,
          phase: def.phase,
          subPhase: def.name,
          type: .work,
          stepCode: def.code,
          assignees: assignees,
          reviewer: reviewer,
          estimatedHours: Double(days * 8),
          startDate: cursor,
          endDate: cursor.addingDays(days),
          status: .pending
        )
        tasks.append(task)

        // Sérialisation simple: chaque étape décale la suivante (sera surchargé par la planification auto)
        cursor = cursor.addingDays(days + 1)
      }
    }
  }

  // MARK: - Soutien équilibré: ajouter B2 libres pour les tâches C surchargées

  private func autoAssignBHelpersIfNeeded() {
    let today = currentDate

    let candidateTasks = tasks.indices.compactMap { idx -> Int? in
      let t = tasks[idx]
      guard t.status != .done && t.status != .refused else { return nil }
      guard let proj = project(for: t.projectId),
         let active = activePhase(for: proj),
         t.phase == active else { return nil }
      guard t.startDate <= today && t.endDate >= today else { return nil }
      return idx
    }

    for idx in candidateTasks {
      let t = tasks[idx]

      let cAssignees = t.assignees.filter { ($0.role == .c1 || $0.role == .c2) && allowedRoles(for: t).contains($0.role) }
      guard !cAssignees.isEmpty else { continue }

      let anyCFreeToday = cAssignees.contains { isFreeToday($0, on: today) }
      if anyCFreeToday { continue }

      let b2All = architects.filter { $0.role == .b2 && allowedRoles(for: t).contains(.b2) }
      let b2Assigned = b2All.filter { t.assignees.contains($0) }
      if let freeAssignedB2 = b2Assigned.filter({ isFreeToday($0, on: today) })
        .min(by: { workload(for: $0) < workload(for: $1) }) {

        if !t.assignees.contains(where: { $0.id == freeAssignedB2.id }) {
          addAssignee(freeAssignedB2, to: t, by: nil)
        }
        continue
      }

      if let freeAnyB2 = b2All
        .filter({ isFreeToday($0, on: today) })
        .min(by: { workload(for: $0) < workload(for: $1) }) {

        if !t.assignees.contains(where: { $0.id == freeAnyB2.id }) {
          addAssignee(freeAnyB2, to: t, by: nil)
        }
      }
    }
  }
}

// MARK: - Planificateur automatique (jours ouvrés + 8h/pers/j + non fractionnable + décalage)

private extension WorkflowStore {

  // ledger: [date(startOfDay) : [personId: usedHours]]
  typealias Ledger = [Date: [UUID: Int]]

  func autoScheduleAllTasksRespectingCapacityAndWeekends(for project: Project) {
    // 1) Précharger le « ledger » de capacité avec les tâches existantes (tous projets)
    var ledger: Ledger = [:]
    preloadExistingTasksIntoLedger(&ledger, excludingProjectId: project.id)

    // 2) Tâches du projet, triées par ordre Phase puis code d'étape
    let orderedTasks = tasks
      .filter { $0.projectId == project.id }
```

```
    .sorted {
      if $0.phase.order != $1.phase.order { return $0.phase.order < $1.phase.order }
      return $0.stepCode < $1.stepCode
    }

  // 3) À partir de project.startDate, chercher une fenêtre continue de jours ouvrés
  for t in orderedTasks {
    guard !t.assignees.isEmpty else { continue }

    let needDays = max(1, Int(ceil(t.estimatedHours / 8.0)))
    var candidateStart = nextWorkingDay(from: max(project.startDate, currentDate))

    while true {
      if canPlace(task: t, startAt: candidateStart, needDays: needDays, ledger: ledger) {
        let end = advanceWorkingDays(from: candidateStart, count: needDays - 1)
        update(taskId: t.id) { tt in
          tt.startDate = candidateStart
          tt.endDate = end
        }
        write(task: t, startAt: candidateStart, needDays: needDays, into: &ledger)
        break
      } else {
        candidateStart = nextWorkingDay(from: candidateStart.addingDays(1))
      }
    }
  }
}

// Précharger les tâches existantes dans le ledger (jours ouvrés uniquement)
func preloadExistingTasksIntoLedger(_ ledger: inout Ledger, excludingProjectId: UUID) {
  for t in tasks {
    let days = workingDayCount(from: t.startDate, to: t.endDate)
    if days <= 0 { continue }
    iterateWorkingDays(from: t.startDate, count: days) { day in
      let d = day.startOfDay()
      var dayMap = ledger[d] ?? [:]
      for p in t.assignees {
        dayMap[p.id] = min(8, (dayMap[p.id] ?? 0) + 8)
      }
      ledger[d] = dayMap
    }
  }
}

// Vérifier si chaque exécuteur a 8h libres pour needDays jours ouvrés consécutifs
func canPlace(task: Task, startAt: Date, needDays: Int, ledger: Ledger) -> Bool {
  let startWD = isWorkingDay(startAt) ? startAt : nextWorkingDay(from: startAt)
  var ok = true
  iterateWorkingDays(from: startWD, count: needDays) { day in
    let d = day.startOfDay()
    let dayMap = ledger[d] ?? [:]
    for p in task.assignees {
      let used = dayMap[p.id] ?? 0
      if used + 8 > p.dailyCapacityHours { ok = false; return }
    }
  }
  return ok
}

// Écrire l'occupation de la tâche dans le ledger
func write(task: Task, startAt: Date, needDays: Int, into ledger: inout Ledger) {
  let startWD = isWorkingDay(startAt) ? startAt : nextWorkingDay(from: startAt)
  iterateWorkingDays(from: startWD, count: needDays) { day in
    let d = day.startOfDay()
    var dayMap = ledger[d] ?? [:]
    for p in task.assignees {
      dayMap[p.id] = min(8, (dayMap[p.id] ?? 0) + 8)
    }
    ledger[d] = dayMap
  }
}

// MARK: - Jours ouvrés

func isWorkingDay(_ date: Date) -> Bool {
  let wd = Calendar.current.component(.weekday, from: date)
  return wd != 1 && wd != 7
}

func nextWorkingDay(from date: Date) -> Date {
  var d = date
  while !isWorkingDay(d) {
    d = d.addingDays(1)
  }
  return d.startOfDay()
}

// Avancer de count jours ouvrés à partir de start (count=0 → jour ouvré d'alignement)
func advanceWorkingDays(from start: Date, count: Int) -> Date {
  var d = nextWorkingDay(from: start)
```

```
    if count == 0 { return d }
    var remaining = count
    while remaining > 0 {
      d = d.addingDays(1)
      if isWorkingDay(d) { remaining -= 1 }
    }
    return d
  }

  // Itérer sur des jours ouvrés consécutifs (inclut le premier)
  func iterateWorkingDays(from start: Date, count: Int, _ body: (Date) -> Void) {
    var d = nextWorkingDay(from: start)
    var remaining = count
    while remaining > 0 {
      body(d)
      remaining -= 1
      if remaining == 0 { break }
      d = d.addingDays(1)
      while !isWorkingDay(d) {
        d = d.addingDays(1)
      }
    }
  }

  // Nombre de jours ouvrés entre deux dates (inclusifs)
  func workingDayCount(from start: Date, to end: Date) -> Int {
    if end < start { return 0 }
    var d = start.startOfDay()
    let e = end.startOfDay()
    var count = 0
    while d <= e {
      if isWorkingDay(d) { count += 1 }
      d = d.addingDays(1)
    }
    return count
  }
}
```

Code de Model.swift

```
// MARK: - Rôles / Niveaux
enum Role: String, CaseIterable, Identifiable {
  case a1 = "A1"
  case a2 = "A2"
  case b1 = "B1"
  case b2 = "B2"
  case c1 = "C1"
  case c2 = "C2"
  case d1 = "D1"
  case d2 = "D2"
  case d3 = "D3"
  case d4 = "D4"

  var id: String { rawValue }

  /// Appartient au groupe H (les personnes qui participent réellement au dessin)
  var isHGroupMember: Bool {
    switch self {
    case .b1, .b2, .c1, .c2, .d1, .d2, .d3, .d4:
      return true
    case .a1, .a2:
      return false
    }
  }

  /// Autorisé à dessiner (intervenir sur la production)
  var canDraw: Bool {
    switch self {
    case .b1, .b2, .c1, .c2, .d1, .d2, .d3, .d4:
      return true
    case .a1, .a2:
      return false
    }
  }

  /// Fait partie de la Direction (gouverne, ne dessine pas)
  var isDirection: Bool {
    switch self {
    case .a1, .a2: return true
    default: return false
    }
  }

  /// Peut assurer la revue / prise de décision
  var isReviewer: Bool {
    switch self {
    case .a1, .a2, .b1, .b2:
      return true
```

```
    default:
      return false
    }
  }

  /// Force de travail principale (prioritaire dans l'affectation)
  var isPrimaryWorker: Bool {
    switch self {
    case .c1, .c2: return true
    default: return false
    }
  }

  /// Niveau "management" (B1/B2 et au-dessus)
  var isManagerLevel: Bool {
    switch self {
    case .a1, .a2, .b1, .b2:
      return true
    default:
      return false
    }
  }

  /// Niveau "direction" (A1/A2)
  var isDirectorLevel: Bool {
    switch self {
    case .a1, .a2:
      return true
    default:
      return false
    }
  }
}

struct Architect: Identifiable, Hashable {
  let id = UUID()
  let name: String
  let role: Role

  /// Capacité maximale par jour (heures)
  let dailyCapacityHours: Int = 8

  /// Appartient au groupe H (effectif de 12 personnes)
  var isHGroupMember: Bool { role.isHGroupMember }
}

// MARK: - Projets

enum ProjectType: String, CaseIterable, Identifiable {
  case smallHousing = "Logements 50–80"
  case mediumHousing = "Logements 80–150"
  case largeHousing = "Logements 150+"

  var id: String { rawValue }
}

struct Project: Identifiable {
  let id = UUID()
  var name: String
  var type: ProjectType
  var units: Int
  var startDate: Date
  var endDate: Date?
  /// Responsable: doit être B1/B2
  var managerId: UUID?
}

// MARK: - Phases / Types de tâches / Statuts

enum Phase: String, CaseIterable, Identifiable {
  case analyse = "Analyse du site"
  case hypothese = "Hypothèse du projet"
  case dessin = "Dessin de la conception finale"
  case collaboration = "Collaboration en conception"
  case chantier = "Collaboration ultérieure (chantier)"

  var id: String { rawValue }

  /// Ordre de comparaison des phases (1 → 5)
  var order: Int {
    switch self {
    case .analyse:    return 1
    case .hypothese:  return 2
    case .dessin:     return 3
    case .collaboration: return 4
    case .chantier:   return 5
    }
  }
}
```

```
enum TaskType: String, CaseIterable, Identifiable {
  case work = "Travail"
  case review = "Revue"
  case ticketResponse = "Ticket – réponse"
  case coordination = "Coordination"

  var id: String { rawValue }
}

enum TaskStatus: String, CaseIterable, Identifiable {
  case pending  = "En attente"  // Créée mais pas encore à son tour
  case available = "Disponible"  // Arrivée à la date, peut être acceptée
  case accepted  = "Acceptée"
  case refused  = "Refusée"
  case inProgress = "En cours"
  case done    = "Terminée"

  var id: String { rawValue }
}

// MARK: - Tâche (sous-phase + chronologie + statut)

struct Task: Identifiable, Equatable {
  let id = UUID()
  var name: String
  var projectId: UUID
  var phase: Phase
  var subPhase: String
  var type: TaskType

  // Étape codifiée strictement (ex. "A1", "B3"...)
  var stepCode: String

  var assignees: [Architect]    // Exécutants (plusieurs possibles)
  var reviewer: Architect?     // Vérificateur (optionnel)

  var estimatedHours: Double    // Charge estimée (heures)
  var startDate: Date         // Début planifié
  var endDate: Date          // Fin planifiée
  var status: TaskStatus      // Statut actuel

  static func == (lhs: Task, rhs: Task) -> Bool {
    lhs.id == rhs.id
  }
}

// MARK: - Journal d'activité (qui a fait quoi et quand)

struct Activity: Identifiable {
  let id = UUID()
  var date: Date     // Utilise la "date simulée": WorkflowStore.currentDate
  var actorName: String  // Qui
  var actorRole: String  // A1/B2/C1…
  var projectName: String // Projet concerné
  var taskName: String  // Tâche concernée
  var action: String   // Action: accepter / démarrer / terminer / valider, etc.
  var taskId: UUID    // Pour retrouver la tâche
}

// MARK: - Outils

extension Date {
  func addingDays(_ days: Int) -> Date {
    Calendar.current.date(byAdding: .day, value: days, to: self) ?? self
  }
}

// Intervenant externe (utilisé ailleurs dans le projet)
struct ExternalCollaborator: Identifiable {
  let id = UUID()
  var name: String
  var specialty: String
}
```

Code de ContentView

ContentView

```
import SwiftUI

struct ContentView: View {

  @StateObject var store = WorkflowStore()

  var body: some View {
    TabView {
      MyTasksView(store: store)
        .tabItem {
          Label("Mes tâches", systemImage: "person.fill.checkmark")
        }
```

```
      ProjectsView(store: store)
        .tabItem {
          Label("Projets", systemImage: "building.2.fill")
        }

      TeamView(store: store)
        .tabItem {
          Label("Équipe", systemImage: "person.3.fill")
        }

      SimulationView(store: store)
        .tabItem {
          Label("Simulation / Rapport", systemImage: "clock.arrow.circlepath")
        }
    }
  }
}

// MARK: - Onglet 1 : Mes tâches

struct MyTasksView: View {

  @ObservedObject var store: WorkflowStore
  @State private var showUserPicker = false

  private func relatedTasks(for user: Architect) -> [Task] {
    store.tasks
      .filter { task in
        task.assignees.contains(where: { $0.id == user.id })
      }
      .sorted { $0.startDate < $1.startDate }
  }

  private func isUnfinished(_ t: Task) -> Bool {
    t.status != .done
  }

  var body: some View {
    NavigationView {
      Group {
        if let user = store.currentUser {
          let allRelated = relatedTasks(for: user)
          let unfinished = allRelated.filter(isUnfinished)

          if unfinished.isEmpty {
            Text("Aucune tâche en cours.")
              .foregroundColor(.secondary)
              .padding(.top, 32)
          } else {
            List {
              ForEach(unfinished) { task in
                NavigationLink {
                  TaskDetailView(store: store, task: task)
                } label: {
                  SimpleTaskRow(
                    task: task,
                    projectName: store.project(for: task.projectId)?.name ?? "Projet non associé"
                  )
                }
              }
            }
            .animation(.default, value: store.tasks.map(\.id))
          }
        } else {
          Text("Veuillez choisir un architecte comme utilisateur courant.")
        }
      }
      .navigationTitle("Mes tâches")
      .toolbar {
        ToolbarItem(placement: .navigationBarTrailing) {
          Button {
            showUserPicker = true
          } label: {
            Label("Changer d’architecte", systemImage: "person.crop.circle.badge.arrow.right")
          }
        }
      }
      .sheet(isPresented: $showUserPicker) {
        UserPickerView(store: store, isPresented: $showUserPicker)
      }
    }
  }
}

private struct SimpleTaskRow: View {
  let task: Task
  let projectName: String

  var body: some View {
```

```
    VStack(alignment: .leading, spacing: 6) {
      Text(projectName)
        .font(.subheadline)
        .foregroundColor(.secondary)

      Text(task.name)
        .font(.body.weight(.medium))
        .foregroundColor(nameColor)
        .lineLimit(2)

      Text("\(task.startDate, style: .date) → \(task.endDate, style: .date)")
        .font(.caption2)
        .foregroundColor(.gray)
    }
    .padding(.vertical, 6)
  }

  private var nameColor: Color {
    switch task.status {
    case .inProgress:
      return .primary
    case .pending, .available, .accepted, .refused:
      return .gray
    case .done:
      return .gray
    }
  }
}

struct UserPickerView: View {
  @ObservedObject var store: WorkflowStore
  @Binding var isPresented: Bool

  var body: some View {
    NavigationView {
      List {
        ForEach(store.architects) { arch in
          Button {
            store.currentUser = arch
            isPresented = false
          } label: {
            HStack {
              Text(arch.name)
              Spacer()
              Text(arch.role.rawValue)
                .font(.caption)
                .foregroundColor(.secondary)
            }
          }
        }
      }
      .navigationTitle("Choisir un architecte")
      .toolbar {
        ToolbarItem(placement: .cancellationAction) {
          Button("Fermer") { isPresented = false }
        }
      }
    }
  }
}

func makeDefaultArchitects() -> [Architect] {
  var list: [Architect] = []
  list.append(Architect(name: "Directeur A1", role: .a1))
  list.append(Architect(name: "Partenaire A2", role: .a2))
  list.append(Architect(name: "Chef d'équipe B1-1", role: .b1))
  list.append(Architect(name: "Chef d'équipe B1-2", role: .b1))
  list.append(Architect(name: "Chef de projet B2-1", role: .b2))
  list.append(Architect(name: "Chef de projet B2-2", role: .b2))
  list.append(Architect(name: "Architecte C1-1", role: .c1))
  list.append(Architect(name: "Architecte C1-2", role: .c1))
  list.append(Architect(name: "Architecte C2-1", role: .c2))
  list.append(Architect(name: "Assistant D1-1", role: .d1))
  list.append(Architect(name: "Assistant D1-2", role: .d1))
  list.append(Architect(name: "Assistant D2-1", role: .d2))
  list.append(Architect(name: "Assistant D2-2", role: .d2))
  list.append(Architect(name: "Designer D3-1", role: .d3))
  return list
}

struct TaskRow: View {

  let task: Task
  let project: Project?
  @ObservedObject var store: WorkflowStore
  let currentUser: Architect

  var body: some View {
    VStack(alignment: .leading, spacing: 6) {
      Text(project?.name ?? "Projet non associé")
        .font(.headline)
        .foregroundColor(.blue)
```

```
    HStack(alignment: .firstTextBaseline) {
      Text(task.name)
        .font(.subheadline.weight(.medium))
        .lineLimit(2)

      Spacer()

      Text(task.phase.displayName)
        .font(.caption2.smallCaps())
        .padding(.horizontal, 6)
        .padding(.vertical, 2)
        .background(phaseColor.opacity(0.15))
        .foregroundColor(phaseColor)
        .clipShape(RoundedRectangle(cornerRadius: 6))
    }

    HStack(spacing: 8) {
      if !task.subPhase.isEmpty {
        Text(task.subPhase)
          .font(.caption)
          .foregroundColor(.secondary)
      }

      if let reviewer = task.reviewer {
        Text("Vérif.: \(reviewer.name) (\(reviewer.role.rawValue))")
          .font(.caption2)
          .foregroundColor(.secondary)
      }
    }

    HStack(spacing: 8) {
      Text(statusText)
        .font(.caption2)
        .foregroundColor(statusColor)

      Text("\(task.startDate, style: .date) → \(task.endDate, style: .date)")
        .font(.caption2)
        .foregroundColor(.gray)

      Spacer()
    }

    HStack {
      switch task.status {
      case .pending:
        Text("Pas encore à la date de début prévue")
          .font(.caption2)
          .foregroundColor(.gray)

      case .available:
        Button("Accepter") {
          store.accept(task, by: currentUser)
        }
        Button("Refuser") {
          store.refuse(task, by: currentUser)
        }

      case .accepted:
        Button("Démarrer") {
          store.start(task, by: currentUser)
        }
        Button("Refuser") {
          store.refuse(task, by: currentUser)
        }

      case .inProgress:
        Button("Terminer") {
          store.finish(task, by: currentUser)
        }

      case .refused:
        Button("Accepter de nouveau") {
          store.accept(task, by: currentUser)
        }

      case .done:
        Text("□ Terminé")
          .font(.caption)
          .foregroundColor(.green)
      }

      Spacer()
    }
    .buttonStyle(.bordered)
    .font(.caption)
  }
  .padding(.vertical, 4)
}
```

```
  private var phaseColor: Color {
    switch task.phase {
    case .analyse:    return .orange
    case .hypothese:   return .purple
    case .dessin:    return .green
    case .collaboration: return .teal
    case .chantier:   return .red
    }
  }

  private var statusText: String {
    switch task.status {
    case .pending:  return "En attente"
    case .available: return "Disponible"
    case .accepted:  return "Acceptée"
    case .inProgress: return "En cours"
    case .done:    return "Terminée"
    case .refused:  return "Refusée"
    }
  }

  private var statusColor: Color {
    switch task.status {
    case .pending:  return .gray
    case .available: return .orange
    case .accepted:  return .blue
    case .inProgress: return .purple
    case .done:    return .green
    case .refused:  return .red
    }
  }
}

// MARK: - Onglet 2 : Projets

struct ProjectsView: View {
  @ObservedObject var store: WorkflowStore
  @State private var showNewProjectSheet = false

  var body: some View {
    NavigationView {
      List {
        Section {
          NavigationLink("□ Gantt global (par projet)") {
            ProjectGroupedGanttView(store: store)
          }
          NavigationLink("□ Gantt global (par personne)") {
            PersonGroupedGanttView(store: store)
          }
        }

        ForEach(store.projects) { project in
          NavigationLink(destination: ProjectDetailView(store: store, project: project)) {
            ProjectRow(project: project, progress: store.progress(for: project))
          }
        }
      }
      .navigationTitle("Projets")
      .toolbar {
        ToolbarItem(placement: .navigationBarTrailing) {
          Button {
            showNewProjectSheet = true
          } label: {
            Label("Nouveau projet", systemImage: "plus.circle")
          }
          .disabled(!store.canEditProjects(current: store.currentUser))
        }
      }
    }
    .sheet(isPresented: $showNewProjectSheet) {
      NewProjectView(store: store, isPresented: $showNewProjectSheet)
    }
  }
}

private struct ProjectRow: View {
  let project: Project
  let progress: Double

  var body: some View {
    VStack(alignment: .leading, spacing: 8) {
      HStack {
        Text(project.name)
          .font(.headline)
        Spacer()
        Text("\(Int(progress * 100))%")
          .font(.caption)
          .foregroundColor(.secondary)
      }
      HStack {
        Text(project.type.rawValue)
          .font(.subheadline)
```

```
          .foregroundColor(.blue)
        Spacer()
        if let end = project.endDate {
          Text("\(project.startDate, style: .date) → \(end, style: .date)")
            .font(.caption2)
            .foregroundColor(.gray)
        } else {
          Text("\(project.startDate, style: .date)")
            .font(.caption2)
            .foregroundColor(.gray)
        }
      }
      ProgressView(value: progress)
        .tint(.green)
    }
    .padding(.vertical, 4)
  }
}

import SwiftUI

// MARK: - Gantt global par projet

struct ProjectGroupedGanttView: View {
  @ObservedObject var store: WorkflowStore

  private let baseRowHeight: CGFloat = 40
  private let laneBarHeight: CGFloat = 22
  private let laneSpacing: CGFloat = 8
  private let leftColumnWidth: CGFloat = 220
  private let dayWidth: CGFloat = 22
  private let headerHeight: CGFloat = 64

  // Filtres
  @State private var searchText: String = ""

  var body: some View {
    let range = overallDateRange()
    let days = max(1, daysBetween(range.start, range.end) + 1)
    let timelineWidth = CGFloat(days) * dayWidth

    // Tri des projets: par startDate puis par nom
    let projects = store.projects.sorted {
      if $0.startDate != $1.startDate { return $0.startDate < $1.startDate }
      return $0.name < $1.name
    }

    ScrollView(.vertical) {
      VStack(spacing: 0) {

        // Haut: recherche + légende des couleurs
        VStack(alignment: .leading, spacing: 8) {
          HStack(spacing: 12) {
            TextField("Rechercher tâche/projet", text: $searchText)
              .textFieldStyle(.roundedBorder)
              .frame(maxWidth: 360)
          }
          ProjectLegendView(projects: store.projects, colorMap: projectColorMap())
        }
        .padding(.horizontal, 12)
        .padding(.vertical, 8)

        // Échelle de temps
        ScrollView(.horizontal, showsIndicators: true) {
          TimeHeaderView(
            start: range.start,
            end: range.end,
            dayWidth: dayWidth,
            leftColumnWidth: leftColumnWidth,
            headerHeight: headerHeight
          )
          .frame(width: leftColumnWidth + timelineWidth, height: headerHeight)
        }

        Divider()

        // Contenu: une ligne par projet (multi-voies)
        ScrollView(.horizontal, showsIndicators: true) {
          VStack(spacing: 0) {
            ForEach(projects) { project in
              let projectTasks = tasks(for: project)
              let layout = GanttLayout.computeLanes(for: projectTasks)
              let rowHeight = baseRowHeight + CGFloat(layout.laneCount) * (laneBarHeight + laneSpacing)

              ProjectGanttRowView(
                project: project,
                tasks: projectTasks,
                layout: layout,
                barHeight: laneBarHeight,
                laneSpacing: laneSpacing,
```

```
                projectColor: projectColorMap()[project.id] ?? .blue,
                start: range.start,
                end: range.end,
                dayWidth: dayWidth,
                leftColumnWidth: leftColumnWidth,
                rowHeight: rowHeight,
                store: store
              )
              Divider()
            }
          }
          .frame(width: leftColumnWidth + timelineWidth)
        }
      }
    }
    .navigationTitle("Gantt global (projet)")
    .navigationBarTitleDisplayMode(.inline)
  }

  private func tasks(for project: Project) -> [Task] {
    // Ordre strict: phase.order → stepCode → startDate → name
    let filtered = store.tasks
      .filter { $0.projectId == project.id }
      .filter { t in
        if searchText.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty { return true }
        let key = searchText.lowercased()
        return t.name.lowercased().contains(key)
      }
    return filtered.sorted { a, b in
      if a.phase.order != b.phase.order { return a.phase.order < b.phase.order }
      if a.stepCode != b.stepCode { return a.stepCode < b.stepCode }
      if a.startDate != b.startDate { return a.startDate < b.startDate }
      return a.name < b.name
    }
  }

  private func overallDateRange() -> (start: Date, end: Date) {
    let all = store.tasks
    guard let minStart = all.map(\.startDate).min(),
       let maxEnd = all.map(\.endDate).max()
    else {
      let d = store.currentDate
      return (d, d)
    }
    return (minStart.startOfDay(), maxEnd.startOfDay())
  }

  private func projectColorMap() -> [UUID: Color] {
    let palette: [Color] = [.blue, .green, .orange, .purple, .teal, .pink, .red, .indigo, .brown, .mint, .cyan]
    var idx = 0
    var map: [UUID: Color] = [:]
    for p in store.projects.sorted(by: { $0.name < $1.name }) {
      map[p.id] = palette[idx % palette.count]
      idx += 1
    }
    return map
  }

  private func daysBetween(_ a: Date, _ b: Date) -> Int {
    Calendar.current.dateComponents([.day], from: a.startOfDay(), to: b.startOfDay()).day ?? 0
  }
}

// Ligne d'un projet (rendu multi-voies)
struct ProjectGanttRowView: View {
  let project: Project
  let tasks: [Task]
  let layout: GanttLayout

  let barHeight: CGFloat
  let laneSpacing: CGFloat

  let projectColor: Color

  let start: Date
  let end: Date

  let dayWidth: CGFloat
  let leftColumnWidth: CGFloat
  let rowHeight: CGFloat

  @State private var selectedTask: Task?
  @State private var showTaskDetail: Bool = false

  @ObservedObject var store: WorkflowStore

  var body: some View {
    let days = max(1, Calendar.current.dateComponents([.day], from: start, to: end).day ?? 0 + 1)
    let timelineWidth = CGFloat(days) * dayWidth
    let today = Date().startOfDay()

    return HStack(spacing: 0) {
```

```
      // Colonne gauche: infos projet
      VStack(alignment: .leading, spacing: 4) {
        Text(project.name)
          .font(.subheadline.weight(.semibold))
          .lineLimit(1)
        Text(project.type.rawValue)
          .font(.caption2)
          .foregroundColor(.secondary)

        HStack(spacing: 6) {
          Circle().fill(projectColor)
            .frame(width: 6, height: 6)
          if let endDate = project.endDate {
            Text("\(project.startDate, style: .date) → \(endDate, style: .date)")
              .font(.caption2)
              .foregroundColor(.secondary)
          } else {
            Text("\(project.startDate, style: .date)")
              .font(.caption2)
              .foregroundColor(.secondary)
          }
        }
      }
      .padding(.horizontal, 10)
      .frame(width: leftColumnWidth, height: rowHeight, alignment: .topLeading)
      .background(Color(UIColor.systemBackground))

      // Droite: barres Gantt
      ZStack(alignment: .topLeading) {
        // Fond: week-end grisé + grille journalière
        HStack(spacing: 0) {
          ForEach(0..<days, id: \.self) { i in
            let date = Calendar.current.date(byAdding: .day, value: i, to: start) ?? start
            let isWeekend = Calendar.current.isDateInWeekend(date)
            Rectangle()
              .fill(isWeekend ? Color.gray.opacity(0.05) : Color.clear)
              .frame(width: dayWidth, height: rowHeight)
              .overlay(alignment: .trailing) {
                Rectangle()
                  .fill(Color.gray.opacity(0.12))
                  .frame(width: 1)
              }
          }
        }

        // Ligne « aujourd'hui »
        if start <= today && today <= end {
          let offsetDays = Calendar.current.dateComponents([.day], from: start, to: today).day ?? 0
          Rectangle()
            .fill(Color.red.opacity(0.5))
            .frame(width: 2, height: rowHeight)
            .offset(x: CGFloat(offsetDays) * dayWidth - 1)
        }

        // Barres multi-voies
        ForEach(layout.items, id: \.task.id) { item in
          let t = item.task
          let x = xOffset(for: t.startDate)
          let w = barWidth(for: t.startDate, t.endDate)
          let y = 10 + CGFloat(item.lane) * (barHeight + laneSpacing)

          let isToday = t.startDate.startOfDay() <= today && t.endDate.startOfDay() >= today

          GanttBar(
            title: "[\(shortProjectCode(project.name))] \(t.stepCode) " + shortTitle(t.name, limit: 24),
            subtitle: project.name,
            color: projectColor,
            status: t.status,
            emphasize: isToday
          )
          .frame(width: max(10, w), height: barHeight)
          .offset(x: x, y: y)
          .contentShape(Rectangle())
          .onTapGesture {
            selectedTask = t
            showTaskDetail = true
          }
          .contextMenu {
            Text("Tâche : \(t.name)")
            Text("Projet : \(project.name)")
            Text("Phase : \(t.phase.rawValue)")
            Text("Période : \(t.startDate.formatted(date: .abbreviated, time: .omitted)) → \(t.endDate.formatted(date: .abbreviated, time:
.omitted))")
            Text("Statut : \(t.status.rawValue)")
          }
        }
      }
      .frame(width: timelineWidth, height: rowHeight, alignment: .topLeading)
      .popover(isPresented: $showTaskDetail) {
        if let t = selectedTask {
```

```
          VStack(alignment: .leading, spacing: 10) {
            Text("Détails de la tâche").font(.headline)
            Text("Nom de fichier complet : \(t.name)").font(.body)
            Text("Tâche : \(t.name)").font(.body)
            Text("Projet : \(project.name)").font(.body)
            if let manager = (store.architects.first { $0.id == project.managerId }) {
              Text("Responsable du projet : \(manager.name) (\(manager.role.rawValue))").font(.body)
            } else {
              Text("Responsable du projet : non défini").font(.body)
            }
            Text("Phase : \(t.phase.rawValue)").font(.body)
            Text("Période : \(t.startDate.formatted(date: .long, time: .omitted)) → \(t.endDate.formatted(date: .long, time: .omitted))").font(.
body)
            Text("Statut : \(t.status.rawValue)").font(.body)
            let names = t.assignees.map { $0.name }.joined(separator: ", ")
            if !names.isEmpty {
              Text("Exécutants : \(names)").font(.body)
            }
            Button("Fermer") { showTaskDetail = false }
              .buttonStyle(.borderedProminent)
              .padding(.top, 6)
          }
          .padding()
          .frame(minWidth: 360)
        } else {
          Text("Aucune tâche").padding()
        }
      }
    }
    .background(Color(UIColor.systemBackground))
  }

  private func xOffset(for date: Date) -> CGFloat {
    let d0 = start.startOfDay()
    let d1 = date.startOfDay()
    let diff = Calendar.current.dateComponents([.day], from: d0, to: d1).day ?? 0
    return CGFloat(diff) * dayWidth + 2
  }

  private func barWidth(for s: Date, _ e: Date) -> CGFloat {
    let s0 = s.startOfDay()
    let e0 = e.startOfDay()
    let diff = Calendar.current.dateComponents([.day], from: s0, to: e0).day ?? 0
    return CGFloat(max(1, diff + 1)) * dayWidth - 4
  }

  private func shortTitle(_ name: String, limit: Int) -> String {
    if name.count <= limit { return name }
    let idx = name.index(name.startIndex, offsetBy: limit)
    return String(name[..<idx]) + "…"
  }

  private func shortProjectCode(_ name: String) -> String {
    let parts = name.split(separator: " ")
    let letters = parts.prefix(2).compactMap { $0.first?.uppercased() }
    if letters.isEmpty { return "PRJ" }
    return letters.joined()
  }
}

// MARK: - Gantt global par personne

struct PersonGroupedGanttView: View {
  @ObservedObject var store: WorkflowStore

  private let baseRowHeight: CGFloat = 36
  private let laneBarHeight: CGFloat = 18
  private let laneSpacing: CGFloat = 6
  private let leftColumnWidth: CGFloat = 220
  private let dayWidth: CGFloat = 22
  private let headerHeight: CGFloat = 64

  @State private var searchText: String = ""

  var body: some View {
    let range = overallDateRange()
    let days = max(1, daysBetween(range.start, range.end) + 1)
    let timelineWidth = CGFloat(days) * dayWidth

    let people = store.architectsSortedForGantt().filter { $0.isHGroupMember }

    ScrollView(.vertical) {
      VStack(spacing: 0) {

        // Haut: recherche + légende des projets
        VStack(alignment: .leading, spacing: 8) {
          HStack(spacing: 12) {
            TextField("Rechercher tâche (par nom/projet)", text: $searchText)
              .textFieldStyle(.roundedBorder)
              .frame(maxWidth: 360)
          }
          ProjectLegendView(projects: store.projects, colorMap: projectColorMap())
```

```
      }
      .padding(.horizontal, 12)
      .padding(.vertical, 8)

      // Échelle de temps
      ScrollView(.horizontal, showsIndicators: true) {
        TimeHeaderView(
          start: range.start,
          end: range.end,
          dayWidth: dayWidth,
          leftColumnWidth: leftColumnWidth,
          headerHeight: headerHeight
        )
        .frame(width: leftColumnWidth + timelineWidth, height: headerHeight)
      }

      Divider()

      // Contenu: une ligne par personne (multi-voies)
      ScrollView(.horizontal, showsIndicators: true) {
        VStack(spacing: 0) {
          ForEach(people, id: \.id) { person in
            let personTasks = tasks(for: person)
            let layout = GanttLayout.computeLanes(for: personTasks)
            let rowHeight = baseRowHeight + CGFloat(layout.laneCount) * (laneBarHeight + laneSpacing)

            PersonGanttRowView(
              person: person,
              tasks: personTasks,
              layout: layout,
              barHeight: laneBarHeight,
              laneSpacing: laneSpacing,
              projectColors: projectColorMap(),
              start: range.start,
              end: range.end,
              dayWidth: dayWidth,
              leftColumnWidth: leftColumnWidth,
              rowHeight: rowHeight,
              store: store
            )
            Divider()
          }
        }
        .frame(width: leftColumnWidth + timelineWidth)
      }
    }
  }
  .navigationTitle("Gantt global (personne)")
  .navigationBarTitleDisplayMode(.inline)
}

private func tasks(for person: Architect) -> [Task] {
  let filtered = store.tasks
    .filter { t in
      t.assignees.contains(where: { $0.id == person.id })
    }
    .filter { t in
      if searchText.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty { return true }
      let key = searchText.lowercased()
      let projectName = store.project(for: t.projectId)?.name.lowercased() ?? ""
      return t.name.lowercased().contains(key) || projectName.contains(key)
    }

  // Ordre: startDate → phase.order → stepCode → name
  return filtered.sorted { a, b in
    if a.startDate != b.startDate { return a.startDate < b.startDate }
    if a.phase.order != b.phase.order { return a.phase.order < b.phase.order }
    if a.stepCode != b.stepCode { return a.stepCode < b.stepCode }
    return a.name < b.name
  }
}

private func overallDateRange() -> (start: Date, end: Date) {
  let all = store.tasks
  guard let minStart = all.map(\.startDate).min(),
    let maxEnd = all.map(\.endDate).max()
  else {
    let d = store.currentDate
    return (d, d)
  }
  return (minStart.startOfDay(), maxEnd.startOfDay())
}

private func projectColorMap() -> [UUID: Color] {
  let palette: [Color] = [.blue, .green, .orange, .purple, .teal, .pink, .red, .indigo, .brown, .mint, .cyan]
  var idx = 0
  var map: [UUID: Color] = [:]
  for p in store.projects.sorted(by: { $0.name < $1.name }) {
    map[p.id] = palette[idx % palette.count]
    idx += 1
```

```
    }
    return map
  }

  private func daysBetween(_ a: Date, _ b: Date) -> Int {
    Calendar.current.dateComponents([.day], from: a.startOfDay(), to: b.startOfDay()).day ?? 0
  }
}

// Ligne d'une personne (multi-voies)
struct PersonGanttRowView: View {
  let person: Architect
  let tasks: [Task]
  let layout: GanttLayout

  let barHeight: CGFloat
  let laneSpacing: CGFloat

  let projectColors: [UUID: Color]

  let start: Date
  let end: Date

  let dayWidth: CGFloat
  let leftColumnWidth: CGFloat
  let rowHeight: CGFloat

  @State private var selectedTask: Task?
  @State private var showTaskDetail: Bool = false

  @ObservedObject var store: WorkflowStore

  var body: some View {
    let days = max(1, Calendar.current.dateComponents([.day], from: start, to: end).day ?? 0 + 1)
    let timelineWidth = CGFloat(days) * dayWidth
    let today = Date().startOfDay()

    return HStack(spacing: 0) {
      // Colonne gauche: info personne
      VStack(alignment: .leading, spacing: 4) {
        Text(person.name)
          .font(.subheadline.weight(.semibold))
          .lineLimit(1)
        Text(person.role.rawValue)
          .font(.caption2)
          .foregroundColor(.secondary)
      }
      .padding(.horizontal, 10)
      .frame(width: leftColumnWidth, height: rowHeight, alignment: .topLeading)
      .background(Color(UIColor.systemBackground))

      // Droite: barres Gantt
      ZStack(alignment: .topLeading) {
        // Fond: week-end + grille
        HStack(spacing: 0) {
          ForEach(0..<days, id: \.self) { i in
            let date = Calendar.current.date(byAdding: .day, value: i, to: start) ?? start
            let isWeekend = Calendar.current.isDateInWeekend(date)
            Rectangle()
              .fill(isWeekend ? Color.gray.opacity(0.05) : Color.clear)
              .frame(width: dayWidth, height: rowHeight)
              .overlay(alignment: .trailing) {
                Rectangle()
                  .fill(Color.gray.opacity(0.12))
                  .frame(width: 1)
              }
          }
        }

        // Ligne « aujourd'hui »
        if start <= today && today <= end {
          let offsetDays = Calendar.current.dateComponents([.day], from: start, to: today).day ?? 0
          Rectangle()
            .fill(Color.red.opacity(0.5))
            .frame(width: 2, height: rowHeight)
            .offset(x: CGFloat(offsetDays) * dayWidth - 1)
        }

        // Barres (seulement les tâches de cette personne)
        ForEach(layout.items, id: \.task.id) { item in
          let t = item.task
          let x = xOffset(for: t.startDate)
          let w = barWidth(for: t.startDate, t.endDate)
          let y = 10 + CGFloat(item.lane) * (barHeight + laneSpacing)
          let proj = store.project(for: t.projectId)

          let isToday = t.startDate.startOfDay() <= today && t.endDate.startOfDay() >= today

          GanttBar(
            title: "[\(shortProjectCode(proj?.name ?? "PRJ"))] \(t.stepCode) " + shortTitle(t.name, limit: 24),
            subtitle: proj?.name ?? "Projet non associé",
            color: projectColors[t.projectId] ?? .blue,
```

```
            status: t.status,
            emphasize: isToday
          )
          .frame(width: max(10, w), height: barHeight)
          .offset(x: x, y: y)
          .contentShape(Rectangle())
          .onTapGesture {
            selectedTask = t
            showTaskDetail = true
          }
          .contextMenu {
            Text("Tâche : \(t.name)")
            Text("Projet : \(proj?.name ?? "–")")
            Text("Phase : \(t.phase.rawValue)")
            Text("Période : \(t.startDate.formatted(date: .abbreviated, time: .omitted)) → \(t.endDate.formatted(date: .abbreviated, time:
.omitted))")
            Text("Statut : \(t.status.rawValue)")
          }
        }
      }
      .frame(width: timelineWidth, height: rowHeight, alignment: .topLeading)
      .popover(isPresented: $showTaskDetail) {
        if let t = selectedTask {
          let proj = store.project(for: t.projectId)
          VStack(alignment: .leading, spacing: 10) {
            Text("Détails de la tâche").font(.headline)
            Text("Nom de fichier complet : \(t.name)").font(.body)
            Text("Tâche : \(t.name)").font(.body)
            Text("Projet : \(proj?.name ?? "–")").font(.body)
            if let p = proj, let manager = (store.architects.first { $0.id == p.managerId }) {
              Text("Responsable du projet : \(manager.name) (\(manager.role.rawValue))").font(.body)
            } else {
              Text("Responsable du projet : non défini").font(.body)
            }
            Text("Phase : \(t.phase.rawValue)").font(.body)
            Text("Période : \(t.startDate.formatted(date: .long, time: .omitted)) → \(t.endDate.formatted(date: .long, time: .omitted))").font(.
body)
            Text("Statut : \(t.status.rawValue)").font(.body)
            let names = t.assignees.map { $0.name }.joined(separator: ", ")
            if !names.isEmpty {
              Text("Exécutants : \(names)").font(.body)
            }
            Button("Fermer") { showTaskDetail = false }
              .buttonStyle(.borderedProminent)
              .padding(.top, 6)
          }
          .padding()
          .frame(minWidth: 360)
        } else {
          Text("Aucune tâche").padding()
        }
      }
    }
    .background(Color(UIColor.systemBackground))
  }

  private func xOffset(for date: Date) -> CGFloat {
    let d0 = start.startOfDay()
    let d1 = date.startOfDay()
    let diff = Calendar.current.dateComponents([.day], from: d0, to: d1).day ?? 0
    return CGFloat(diff) * dayWidth + 2
  }

  private func barWidth(for s: Date, _ e: Date) -> CGFloat {
    let s0 = s.startOfDay()
    let e0 = e.startOfDay()
    let diff = Calendar.current.dateComponents([.day], from: s0, to: e0).day ?? 0
    return CGFloat(max(1, diff + 1)) * dayWidth - 4
  }

  private func shortTitle(_ name: String, limit: Int) -> String {
    if name.count <= limit { return name }
    let idx = name.index(name.startIndex, offsetBy: limit)
    return String(name[..<idx]) + "…"
  }

  private func shortProjectCode(_ name: String) -> String {
    let parts = name.split(separator: " ")
    let letters = parts.prefix(2).compactMap { $0.first?.uppercased() }
    if letters.isEmpty { return "PRJ" }
    return letters.joined()
  }
}

// MARK: - Partagé : en-tête, layout, barre

struct ProjectLegendView: View {
  let projects: [Project]
  let colorMap: [UUID: Color]
```

```
  var body: some View {
    ScrollView(.horizontal, showsIndicators: false) {
      HStack(spacing: 12) {
        ForEach(projects) { p in
          HStack(spacing: 6) {
            Circle()
              .fill((colorMap[p.id] ?? .blue))
              .frame(width: 10, height: 10)
            Text(p.name)
              .font(.caption)
              .foregroundColor(.secondary)
          }
          .padding(.vertical, 4)
          .padding(.horizontal, 8)
          .background(Color.secondary.opacity(0.08))
          .clipShape(Capsule())
        }
      }
      .padding(.vertical, 2)
    }
  }
}

struct TimeHeaderView: View {
  let start: Date
  let end: Date
  let dayWidth: CGFloat
  let leftColumnWidth: CGFloat
  let headerHeight: CGFloat

  var body: some View {
    let days = max(1, Calendar.current.dateComponents([.day], from: start, to: end).day ?? 0 + 1)
    let today = Date().startOfDay()

    return ZStack(alignment: .topLeading) {
      Rectangle()
        .fill(Color(UIColor.systemBackground))
        .frame(width: leftColumnWidth, height: headerHeight)

      HStack(spacing: 0) {
        Rectangle()
          .fill(Color(UIColor.systemBackground))
          .frame(width: leftColumnWidth, height: headerHeight)

        ZStack(alignment: .topLeading) {
          HStack(spacing: 0) {
            ForEach(0..<days, id: \.self) { i in
              let date = Calendar.current.date(byAdding: .day, value: i, to: start) ?? start
              let isWeekend = Calendar.current.isDateInWeekend(date)
              Rectangle()
                .fill(isWeekend ? Color.gray.opacity(0.06) : Color.clear)
                .frame(width: dayWidth, height: headerHeight)
            }
          }

          HStack(spacing: 0) {
            ForEach(0..<days, id: \.self) { _ in
              Rectangle()
                .fill(Color.clear)
                .frame(width: dayWidth, height: headerHeight)
                .overlay(alignment: .trailing) {
                  Rectangle()
                    .fill(Color.gray.opacity(0.15))
                    .frame(width: 1)
                }
            }
          }

          ForEach(0..<days, id: \.self) { i in
            let date = Calendar.current.date(byAdding: .day, value: i, to: start) ?? start
            if i % 7 == 0 || Calendar.current.component(.weekday, from: date) == 2 {
              VStack(alignment: .leading, spacing: 2) {
                Text(date, format: .dateTime.month().day())
                  .font(.caption2).foregroundColor(.secondary)
                Text("S\(Calendar.current.component(.weekOfYear, from: date))")
                  .font(.system(size: 9)).foregroundColor(.secondary)
              }
              .offset(x: CGFloat(i) * dayWidth + 4, y: 4)
            }
          }

          if start <= today && today <= end {
            let offsetDays = Calendar.current.dateComponents([.day], from: start, to: today).day ?? 0
            Rectangle()
              .fill(Color.red.opacity(0.5))
              .frame(width: 2, height: headerHeight)
              .offset(x: CGFloat(offsetDays) * dayWidth - 1)
          }
        }
        .frame(width: CGFloat(days) * dayWidth, height: headerHeight)
      }
    }
```

```
  }
}

struct GanttLayout {
  struct Item {
    let task: Task
    let lane: Int
  }
  let items: [Item]
  let laneCount: Int

  static func computeLanes(for tasks: [Task]) -> GanttLayout {
    // Ordre strict: phase.order → stepCode → startDate → name
    let sorted = tasks.sorted { a, b in
      if a.phase.order != b.phase.order { return a.phase.order < b.phase.order }
      if a.stepCode != b.stepCode { return a.stepCode < b.stepCode }
      if a.startDate != b.startDate { return a.startDate < b.startDate }
      return a.name < b.name
    }

    var laneEnds: [Date] = []
    var placed: [Item] = []

    for t in sorted {
      var assignedLane: Int? = nil
      for i in 0..<laneEnds.count {
        if laneEnds[i] <= t.startDate {
          assignedLane = i
          laneEnds[i] = t.endDate
          break
        }
      }
      if assignedLane == nil {
        laneEnds.append(t.endDate)
        assignedLane = laneEnds.count - 1
      }
      placed.append(Item(task: t, lane: assignedLane!))
    }

    return GanttLayout(items: placed, laneCount: laneEnds.count)
  }
}

struct GanttBar: View {
  let title: String
  let subtitle: String
  let color: Color
  let status: TaskStatus
  var emphasize: Bool = false

  var body: some View {
    let opacity = statusOpacity(status)

    return RoundedRectangle(cornerRadius: 6)
      .fill(color.opacity(0.24 * opacity))
      .overlay(
        RoundedRectangle(cornerRadius: 6)
          .stroke(color.opacity(emphasize ? 1.0 : (0.65 * opacity)), lineWidth: emphasize ? 2 : 1)
      )
      .shadow(color: emphasize ? color.opacity(0.35) : .clear, radius: emphasize ? 4 : 0, x: 0, y: 0)
      .overlay(alignment: .leading) {
        VStack(alignment: .leading, spacing: 2) {
          Text(title)
            .font(.system(size: 12, weight: .semibold))
            .lineLimit(1)
          Text(subtitle)
            .font(.system(size: 10, weight: .semibold))
            .foregroundColor(color)
            .lineLimit(1)
        }
        .padding(.horizontal, 8)
        .padding(.vertical, 4)
      }
      .accessibilityLabel("\(title), \(subtitle)")
  }

  private func statusOpacity(_ s: TaskStatus) -> Double {
    switch s {
    case .pending:  return 0.5
    case .available: return 0.9
    case .accepted:  return 1.0
    case .inProgress: return 1.0
    case .done:    return 0.6
    case .refused:  return 0.3
    }
  }
}

extension Date {
  func startOfDay() -> Date {
```

```
    Calendar.current.startOfDay(for: self)
  }
}

extension WorkflowStore {
  func architectsSortedForGantt() -> [Architect] {
    let order: [Role: Int] = [
      .a1: 0, .a2: 1,
      .b1: 2, .b2: 3,
      .c1: 4, .c2: 5,
      .d1: 6, .d2: 7, .d3: 8, .d4: 9
    ]
    return architects.sorted {
      (order[$0.role] ?? 99, $0.name) < (order[$1.role] ?? 99, $1.name)
    }
  }
}

struct NewProjectView: View {
  @ObservedObject var store: WorkflowStore
  @Binding var isPresented: Bool

  @State private var name: String = ""
  @State private var type: ProjectType = .smallHousing
  @State private var units: Int = 60
  @State private var manager: Architect?

  var body: some View {
    NavigationView {
      Form {
        Section(header: Text("Informations de base")) {
          TextField("Nom du projet", text: $name)
          Picker("Type", selection: $type) {
            ForEach(ProjectType.allCases) { t in
              Text(t.rawValue).tag(t)
            }
          }
          Stepper("Unités : \(units)", value: $units, in: 10...150, step: 10)
        }

        Section(header: Text("Responsable (doit être B1/B2)")) {
          Picker("Responsable", selection: Binding(
            get: { manager?.id },
            set: { newId in
              manager = store.architects.first(where: { $0.id == newId })
            })) {
              ForEach(store.architects.filter { $0.role == .b1 || $0.role == .b2 }) { arch in
                Text(arch.name).tag(arch.id as UUID?)
              }
            }
        }
      }
      .navigationTitle("Nouveau projet")
      .toolbar {
        ToolbarItem(placement: .cancellationAction) {
          Button("Annuler") { isPresented = false }
        }
        ToolbarItem(placement: .confirmationAction) {
          Button("Enregistrer") {
            if let m = manager, !name.isEmpty, units <= 150 {
              store.createProject(name: name, type: type, units: units, manager: m)
            }
            isPresented = false
          }
        }
      }
    }
  }
}

struct ProjectDetailView: View {

  @ObservedObject var store: WorkflowStore
  let project: Project

  var body: some View {
    let tasks = store.tasks.filter { $0.projectId == project.id }

    return List {
      Section(header: Text("Informations de base")) {
        Text("Nom : \(project.name)")
        Text("Type : \(project.type.rawValue)")
        Text("Unités : \(project.units)")
        if let end = project.endDate {
          Text("Période : \(project.startDate, style: .date) → \(end, style: .date)")
        }
      }

      ForEach(Phase.allCases) { phase in
        let phaseTasks = tasks.filter { $0.phase == phase }
        if !phaseTasks.isEmpty {
          Section(header: Text(phase.rawValue)) {
```

```
            ForEach(phaseTasks) { task in
              NavigationLink(destination: TaskDetailView(store: store, task: task)) {
                VStack(alignment: .leading, spacing: 4) {
                  Text(task.name)
                    .font(.headline)
                  Text(task.subPhase)
                    .font(.caption)
                    .foregroundColor(.secondary)
                  Text("\(task.startDate, style: .date) → \(task.endDate, style: .date)")
                    .font(.caption2)
                    .foregroundColor(.gray)
                }
              }
            }
          }
        }
      }
    }
    .navigationTitle(project.name)
  }
}

struct TaskDetailView: View {
  @ObservedObject var store: WorkflowStore
  let task: Task

  var body: some View {
    let project = store.project(for: task.projectId)

    let relatedActivities = store.activities
      .filter { $0.taskId == task.id }
      .sorted { $0.date < $1.date }

    return List {
      Section(header: Text("Informations sur la tâche")) {
        Text("Tâche : \(task.name)")
        Text("Projet : \(project?.name ?? "–")")
        Text("Phase : \(task.phase.rawValue)")
        Text("Sous-phase : \(task.subPhase)")
        Text("Type : \(task.type.rawValue)")
        Text("Planifié : \(task.startDate, style: .date) → \(task.endDate, style: .date)")
        Text("Charge estimée : \(Int(task.estimatedHours)) h")
        Text("Statut : \(task.status.rawValue)")
        Text("Exécutants : \(task.assignees.map { $0.name }.joined(separator: ", "))")
        if let rev = task.reviewer {
          Text("Vérificateur : \(rev.name) (\(rev.role.rawValue))")
        }
        if let p = project,
          let manager = (store.architects.first { $0.id == p.managerId }) {
          Text("Responsable du projet : \(manager.name) (\(manager.role.rawValue))")
        }
        Text("Nom de fichier complet : \(task.name)")
      }

      Section(header: Text("Journal d’activités")) {
        if relatedActivities.isEmpty {
          Text("Aucune activité pour cette tâche.").font(.caption)
        } else {
          ForEach(relatedActivities) { act in
            VStack(alignment: .leading, spacing: 2) {
              Text("\(act.date, style: .date)")
                .font(.caption2)
                .foregroundColor(.gray)
              Text("\(act.actorName) [\(act.actorRole)]")
                .font(.caption)
              Text(act.action)
                .font(.caption)
            }
            .padding(.vertical, 2)
          }
        }
      }

      Section(header: Text("Service Central – Serveur documentaire")) {
        Text(store.documentServiceDescription)
          .font(.caption)
      }
    }
    .navigationTitle(task.name)
  }
}

struct TeamView: View {

  @ObservedObject var store: WorkflowStore
  @State private var showAddExternal = false
  @State private var newExternalName = ""
  @State private var newExternalSpec = ""

  private var direction: [Architect] {
```

```
  store.architects.filter { $0.role == .a1 || $0.role == .a2 }
}
private var hGroup: [Architect] {
  store.architects.filter { $0.isHGroupMember }
}

var body: some View {
  NavigationView {
    List {
      Section(header: Text("Direction (A1/A2, hors groupe H)")) {
        if direction.isEmpty {
          Text("Aucun membre de niveau A").font(.caption)
        } else {
          ForEach(direction) { arch in
            HStack {
              Text(arch.name)
              Spacer()
              Text(arch.role.rawValue)
                .font(.caption)
                .foregroundColor(.secondary)
            }
          }
        }
      }

      Section(header: Text("Membres du groupe H (B/C/D) – \(hGroup.count) pers.")) {
        if hGroup.isEmpty {
          Text("Aucun membre dans le groupe H").font(.caption)
        } else {
          ForEach(hGroup) { arch in
            HStack {
              Text(arch.name)
              Spacer()
              Text(arch.role.rawValue)
                .font(.caption)
                .foregroundColor(.secondary)
            }
          }
        }
      }

      Section(
        header: HStack {
          Text("Autres spécialistes / Consultants")
          Spacer()
          Button {
            showAddExternal = true
          } label: {
            Image(systemName: "plus.circle")
          }
        }
      ) {
        if store.externalCollaborators.isEmpty {
          Text("Aucun intervenant externe. Ajoutez-en via le bouton + ci-dessus.")
            .font(.caption)
        } else {
          ForEach(store.externalCollaborators) { ext in
            VStack(alignment: .leading, spacing: 2) {
              Text(ext.name)
              Text("Spécialité : \(ext.specialty)")
                .font(.caption)
                .foregroundColor(.secondary)
            }
          }
        }
      }

      Section(header: Text("Service Central – Serveur documentaire")) {
        Text(store.documentServiceDescription)
          .font(.caption)
      }
    }
    .navigationTitle("Équipe")
  }
  .sheet(isPresented: $showAddExternal) {
    NavigationView {
      Form {
        TextField("Nom de l'entité", text: $newExternalName)
        TextField("Spécialité (ex.: Structure / CVC / Paysage)", text: $newExternalSpec)
      }
      .navigationTitle("Ajouter un intervenant externe")
      .toolbar {
        ToolbarItem(placement: .cancellationAction) {
          Button("Annuler") { showAddExternal = false }
        }
        ToolbarItem(placement: .confirmationAction) {
          Button("Enregistrer") {
            if !newExternalName.isEmpty {
              let ext = ExternalCollaborator(
                name: newExternalName,
                specialty: newExternalSpec.isEmpty ? "Consultant" : newExternalSpec
              )
```

```
                store.externalCollaborators.append(ext)
              }
              newExternalName = ""
              newExternalSpec = ""
              showAddExternal = false
            }
          }
        }
      }
    }
  }
}

struct SimulationView: View {

  @ObservedObject var store: WorkflowStore

  var body: some View {
    NavigationView {
      VStack(alignment: .leading, spacing: 12) {
        Text("Date de simulation actuelle : \(store.currentDate, style: .date)")
          .font(.headline)

        Button {
          store.advanceOneDay()
        } label: {
          HStack {
            Image(systemName: "play.circle.fill")
            Text("Avancer la simulation d'un jour")
          }
          .frame(maxWidth: .infinity)
        }
        .buttonStyle(.borderedProminent)

        Divider()

        Text("Journal des opérations par jour :")
          .font(.subheadline)

        List {
          ForEach(groupedActivitiesSortedByDate(), id: \.key) { day, acts in
            Section(header: Text(day)) {
              ForEach(acts) { act in
                VStack(alignment: .leading, spacing: 2) {
                  Text("\(act.actorName) [\(act.actorRole)]")
                    .font(.caption)
                  Text("Projet : \(act.projectName)")
                    .font(.caption2)
                  Text("Tâche : \(act.taskName)")
                    .font(.caption2)
                  Text(act.action)
                    .font(.caption2)
                }
                .padding(.vertical, 2)
              }
            }
          }
        }
        .padding()
        .navigationTitle("Simulation / Rapport")
      }
    }

    private func groupedActivitiesSortedByDate() -> [(key: String, value: [Activity])] {
      let formatter = DateFormatter()
      formatter.locale = Locale(identifier: "fr_FR")
      formatter.dateFormat = "yyyy-MM-dd"

      let grouped = Dictionary(grouping: store.activities) { act in
        formatter.string(from: act.date)
      }

      return grouped
        .map { ($0.key, $0.value.sorted { $0.date < $1.date }) }
        .sorted { $0.key > $1.key }
    }
}

struct ContentView_Previews: PreviewProvider {
  static var previews: some View {
    ContentView()
  }
}

extension Phase {
  var displayName: String {
    switch self {
    case .analyse:    return "Analyse du site"
    case .hypothese:   return "Hypothèse du projet"
```

```
        case .dessin:    return "Dessin final"
        case .collaboration: return "Collaboration"
        case .chantier:   return "Chantier"
        }
    }
}
```