Gérer un projet informatique

Table des matières

La gestion de projet permet de définir les principales étapes phares qui assureront une construction efficiente et efficace du projet. En effet, naturellement, le porteur de projet a tendance à aller directement dans la partie visible du projet, c'est-à-dire celle qui prend en compte l'action. Or, sans période de préparation, le passage à l'action n'est pas toujours optimal, de sorte que la mise en œuvre du projet est parsemée de problèmes qu'il convient de traiter au fur et à mesure de leur apparition.

1. Conception

1.1. Prise de tête (brainstorming)

La première étape de la conception, c'est le "brainstorming", le moment où l'on crée une tempête dans son cerveau, où on laisse bouillonner les idées et on voit ce qu'il en sort.

Votre premier défi est donc simple : déterminez les contraintes de votre projet :

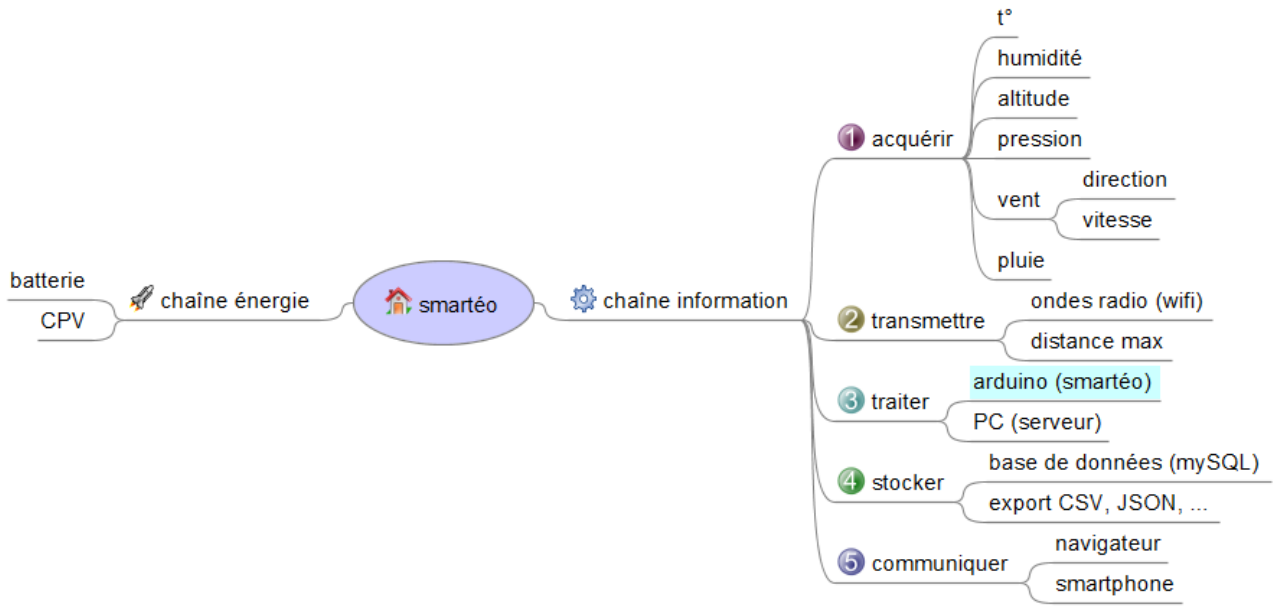
1. de combien de temps je dispose pour réaliser ce projet ?
2. quelle(s) technologie(s) je souhaite / je suis en mesure d'utiliser pour ce projet ?
3. quel type de projet je souhaite faire (jeu, histoire animée, quiz, site web, objet connecté, …) ?

Utilisez le logiciel Freeplane et écrivez toutes les idées qui vous passent par la tête. Seule contrainte : une seule idée par nœud ! L'objectif est de faire ressortir un maximum d'idées, sans jugement ni auto-censure.

Vous pouvez même commencer à trier vos idées en utilisant des couleurs de nœud différentes : selon qu'il s'agit d'une fonctionnalité, d'une technologie à utiliser, d'une idée générale encore vague, d'un type de projet, d'une notion dans une discipline particulière, etc.

Ensuite discutez des post-it de chacun, pour les classer.

Voici le classement pour une station météo en IoT :



1.2. Restez SMART !

Vous avez bien "brainstormé" et vous avez une multitude d'idées mais ne savez pas encore quoi en faire ? Dans la phase de réajustement nous allons sélectionner, alléger ou au contraire creuser un peu plus loin en fonction des besoins, trier tout ça pour transformer un tas d'idées en vrac en un futur projet.

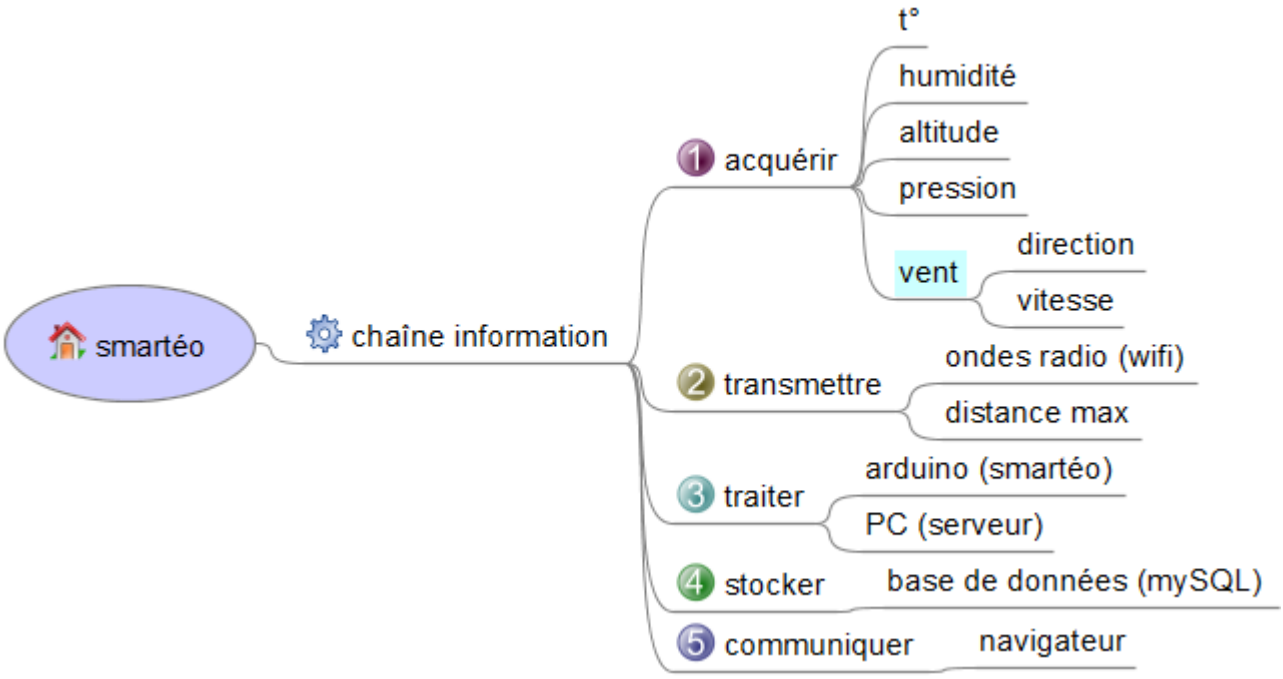
Il est temps de reprendre toutes les idées proposées dans la phase de brainstorming et de réfléchir à

leur faisabilité. Commencez par vous demander si vous avez le matériel et/ou les connaissances pour réaliser chaque idée.

Ainsi, pour chaque idée, vous allez devoir lister ce qu'elle implique et ses différents besoins (en temps, matériel, notions à connaître). Vous serez ainsi à même de déterminer si l'idée est réaliste ou non.

Votre premier défi consiste à éliminer les idées non réalistes, qu'elles fassent appel à trop de connaissances, de temps ou encore de matériel que vous ne pouvez pas vous procurer.

Dans mon cas, j'ai éliminé les nœuds chaîne d'énergie, pluviométrie, export CSV/JSON et application sur smartphone, car cela me prendrait trop de temps de faire des recherches sur ces sujets spécifiques :



L'idée de base est de permettre d'avoir un projet certes plus modeste mais réaliste. Il n'y a rien de plus frustrant que de partir sur un projet génial et ambitieux et de terminer avec quelque chose de bancal et qui ne fonctionne pas car on n'a pas su ou eu le temps de le finir.

A la fin, le projet doit être justifié en termes d'objectifs selon des critères « SMART » :

- Spécifique : le projet n'est pas trop généraliste et son périmètre est convenablement défini
- Mesurable : des indicateurs (délai, objectif) peuvent être mis en place pour le suivi du projet
- Approuvé : tous les membres de l'équipe sont d'accord pour s'investir dans le projet
- Réaliste : les membres disposent, ou pourront acquérir, les compétences pour réaliser le projet
- Temporel : le délai pour la réalisation du projet doit être réaliste

2. Planification

Vous avez à présent des idées réalistes. Mais les éléments de votre projet ne sont pas forcément tous indispensables !

Votre deuxième défi sera de développer vos différents éléments et de les classer dans ces trois catégories :

1. Indispensable : les choses sans lesquelles le projet ne fonctionnerait pas du tout.
2. Important : oui le projet tourne sans mais franchement, c'est clairement moins bien.
3. Souhaitable : oui ça aussi c'est super, mais on fait déjà le reste et puis on avisera en fonction du temps.

Vous devriez maintenant avoir une vue assez détaillée de votre projet avec ses différentes fonctionnalités. Il va donc falloir synthétiser toutes ces informations dans un diagramme de Gantt.

Le diagramme de Gantt est un outil de planification qui permet de visualiser dans le temps les diverses tâches composant un projet. Il s'agit d'une représentation d'un graphe connexe, valué et orienté, qui permet de représenter graphiquement l'avancement du projet.

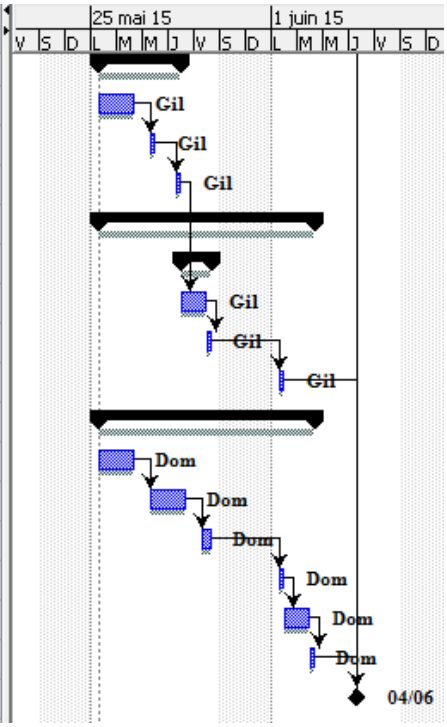
Vous avez défini les différentes tâches à effectuer en les classant par ordre d'importance. Maintenant, il va falloir les classer dans l'ordre de réalisation (on ne peut pas tester une fonction d'export CSV si la base de données n'a pas été préalablement créée par exemple) et attribuer une ressource (c-à-d une personne) à chaque tâche. Attention : un tâche = une personne ; on ne peut pas être deux à faire la même chose sinon c'est une perte de temps.

Le défi est d'équilibrer en terme de charge de travail tous les membres de l'équipe !

A la fin, on doit avoir toutes les tâches à réaliser pour aboutir au projet avec le délai de réalisation et lu durée total du travail pour chaque membre du projet.

Voici un exemple de diagramme de Gantt pour la station météo IoT que vous pouvez réaliser avec les logiciels GanttProject ou LibreProject :

	Nom	Durée	Début	Fin
11	**arduino**	3,5 jours?	**25/05/15 08:00**	**28/05/15 13:00**
12	programmation arduino	2 jours?	25/05/15 08:00	26/05/15 17:00
13	test XBee	0,5 jours?	27/05/15 08:00	27/05/15 13:00
14	test capteurs	0,5 jours?	28/05/15 08:00	28/05/15 13:00
15	**serveur**	7 jours?	**25/05/15 08:00**	**02/06/15 17:00**
16	**base de données**	1,5 jours?	**28/05/15 13:00**	**29/05/15 17:00**
17	modèle et schéma	1 jour?	28/05/15 13:00	29/05/15 13:00
18	configuration	0,5 jours?	29/05/15 13:00	29/05/15 17:00
19	lecteur série (SQL)	0,5 jours?	01/06/15 08:00	01/06/15 13:00
20	**interface web**	7 jours?	**25/05/15 08:00**	**02/06/15 17:00**
21	installateur	2 jours?	25/05/15 08:00	26/05/15 17:00
22	page accueil	2 jours?	27/05/15 08:00	28/05/15 17:00
23	configuration	1 jour?	29/05/15 08:00	29/05/15 17:00
24	mesure temps réel	0,5 jours?	01/06/15 08:00	01/06/15 13:00
25	historique	1 jour?	01/06/15 13:00	02/06/15 13:00
26	export CSV	0,5 jours?	02/06/15 13:00	02/06/15 17:00
27	recette	1 jour?	04/06/15 08:00	04/06/15 17:00



NB : vous remarquerez que l'on définit des sous tâches pour certaines tâches. Que chaque tâche s'enchaîne et qu'une ressource a bien été affectée pour chaque tâche !

3. Réalisation

3.1. Développez (proprement) votre programme

Reprenez la liste de vos tâches du diagramme de Gantt et commencez à la développer en faisant bien attention à la lisibilité du code…

Au fur et à mesure que vous réaliserez ces tâches, complétez l'avancée dans le diagramme de Gantt. Vous pourrez comparer le temps approximatif qu'a pris chacune des tâches avec l'estimation qu'on en avait faite avant, pour nous aider à mieux estimer le temps la prochaine fois. Cela permet, quand on est un peu coincé sur une tâche de la fin, plus difficile que les autres, de se redonner la pêche en voyant tout ce qu'on a déjà fait jusque là.

Si on est déjà désespéré en faisant la première tâche, là, regarder ce qu'on a déjà fait n'aidera pas beaucoup, mais ça signifie peut-être simplement qu'il faut refaire une fiche de conception un peu moins ambitieuse ;) ou redécouper les tâches plus finement.

Et pendant que vous développez votre projet, remplissez la fiche de suivi de bugs avec au moins un bug que vous rencontrerez au cours du développement de votre projet.

3.2. Rendez votre code lisible

Il est TRÈS important d'avoir un code lisible. Voici à nouveau quelques éléments à garder en tête pendant que vous codez :

3.2.1. Choisissez des noms de variables explicites

Pour l'ordinateur, que votre variable s'appelle toto ou ornithorynque, ça n'a aucune importance. Il fonctionnera pareil dans tous les cas. Par contre pour la personne qui vous aide dans votre projet, ou celle qui voudra l'utiliser pour le modifier plus tard, ça change tout.

Personnalisez donc au mieux ce que vous pouvez, pour rendre les choses plus claires.

Par exemple, l'instruction :

```
int   nbr_de_pas = 10 ;
```

est plus parlante que :

```
int   x = 10 ;
```

On peut personnaliser les noms de variables, de listes ou de messages à transmettre, en choisissant un nom qui, d'une part, nous dit ce dont on parle, et d'autre part, n'est pas trop long ou alambiqué.

3.2.2. Commentez votre code

Les commentaires, que vous pouvez ajouter dans votre code permettent de décrire un bout de votre code : à quoi sert une variable dans le premier bloc qui l'initialise, ou ce que fait une boucle, ou une instruction un peu compliquée.

Essayez de penser à ce qui pourrait être un peu obscur pour un lecteur et comment l'aider à comprendre. Inutile en revanche d'ajouter des commentaires sans plus de valeur (par exemple pour une variable "compteur de point" qui est déjà explicite, inutile de redire que c'est un compteur de points). Ajouter en revanche des informations sur la valeur d'une variable (qu'elle doit être toujours positive, ou à choisir dans une liste donnée, …) ou mettre une référence (par exemple un article de

wikipédia) qui a inspiré la portion de code, ou simplement expliquer ce que fait cette partie du code en une ligne est très utile.

L'exemple ci-dessous utilise des commentaires en C++ qui commencent par // :

```
cout << "Combien vaut pi ?" << endl;
//On crée une case mémoire pour stocker un nombre réel
double piUtilisateur(-1.);
//Et on remplit cette case avec ce qu'écrit l'utilisateur
cin >> piUtilisateur;
```

3.2.3 Découpez votre code en petites unités

Il y a plein de bonnes raisons à utiliser les fonctions, en voici quelques-unes :

- tout d'abord le code qui dessine un carré (la définition du bloc), on ne va l'écrire qu'une seule fois, et puis pour chaque carré qu'on veut dans notre programme, on va juste utiliser le bloc carré et l'ajouter dans notre script. Du coup notre code est plus propre et plus court.
- écrire une fonction permet de la tester séparément (en l'exécutant dans différentes situations, pour différentes valeurs d'entrées) et puis une fois qu'on est sûr que la fonction fonctionne bien, on peut le réutiliser en confiance, sans risque d'introduire des bugs en recopiant.
- on peut modifier le code par un autre plus rapide, ou plus facile à comprendre ou plus élégant… il nous suffit de réécrire cette fonction et c'est réglé !
- on peut commenter une fonction pour expliquer ce qu'elle fait et si une autre personne veut modifier le programme, alors elle n'a pas besoin de comprendre en détail le code de cette fonction. Elle peut le refaire à sa manière, à partir du moment où les explications de ce qui est attendu de cette fonction sont claires.

De manière générale, pour n'importe quel programme informatique et quel que soit le langage, découper son code en petites unités qui ont du sens est toujours une bonne pratique.

3.3. Anticipez les bugs

De manière inévitable, les bugs font partie intégrante de l'activité de programmation, qu'il s'agisse de logiciel ou de matériel d'ailleurs. Trouver quel composant électronique fait bugger le robot est aussi amusant que de trouver quel bout de code fait bugger le programme.

Si votre projet tourne, c'est que vous avez fait une bonne partie du boulot, et corrigé les bugs trouvés sur votre chemin. Seulement votre projet est interactif, la personne qui va l'utiliser ne va pas forcément le faire comme vous. Il va donc falloir essayer de débusquer les utilisations non prévues qui pourraient faire bugger votre programme / objet numérique.

3.3.1. Votre programme se comporte-t-il comme prévu ?

D'abord il faut définir ce qui doit marcher ou pas. Par exemple si le couloir est moins large que le robot, forcément ça ne va pas marcher (mais votre robot ne doit pas ni s'abîmer ni tout casser non plus). Pour la vitesse par exemple, il faut d'abord dire à l'utilisatrice les valeurs acceptables, et puis décider ce qu'on fait si elle ne respecte pas ce qui est demandé (on arrête le jeu et elle a perdu ? on met la vitesse à la valeur max autorisée ? on lui demande d'entrer une autre valeur jusqu'à ce qu'elle soit convenable ?).

On va donc essayer de voir si, quels que soient les choix qu'on fait en tant qu'utilisateur, le programme se comporte comme prévu. Si un personnage doit attraper une clé et une pièce dans

l'ordre qu'il veut, il faut tester les deux cas : la pièce en premier ou la clé en premier. Si on peut répondre oui ou non à une question, pareil on va tester les deux cas. De manière générale, on va essayer de tester/couvrir le maximum de comportements possibles, afin d'augmenter d'autant ses chances de trouver un bug caché. On parle de « taux de couverture » pour une série de tests, pour évaluer le pourcentage de comportements qui ont été pris en compte dans les tests.

De même il ne faut pas qu'en faisant n'importe quoi l'utilisateur réussisse à contourner le jeu. Si par exemple on choisit une vitesse négative du personnage et qu'il gagne en franchissant une ligne d'arrivée en marche arrière, ce n'est pas ce que l'on voulait, et il faut le tester.

3.3.2. Testez les cas de bords

Comme on n'a évidemment pas le temps de "couvrir" tous les cas, il va falloir bien choisir ceux qu'on teste, de préférence ceux qui ont le plus de "chances" de planter. Pour ça nous avons des candidats de choix : ce qu'on appelle les « cas de bord ».

Imaginez que mon programme imite un ascenseur qui va de y = -100 (rez-de-chaussée) à y = 100 (premier étage) dans lequel l'utilisateur a deux boutons : monter et descendre. Quand on appuie sur monter, le lutin monte de 10 en 10 pixels jusqu'à la position 100. Quand on appuie sur descendre il le fait de 10 en 10 jusqu'à la position -100.

Le problème arrive quand on demande de descendre et qu'on est en bas (ou on demande de monter et on est en haut). Il commence par descendre (et donc perforer le sol au fond de la cage d'ascenseur) puis il teste s'il est arrivé, ce qui est trop tard car il a déjà creusé son trou jusqu'à presque sortir de l'écran.

L'idée est que quand on a une variable qui peut prendre des valeurs dans un ensemble fini, les valeurs extrêmes ont plus de chances de provoquer un bug que les autres.

3.3.3. Faire et défaire

C'est bien beau, je corrige mes bugs au fur et à mesure que je les trouve, mais comment savoir si je n'en crée pas de nouveaux… ou si je ne retombe pas sur les anciens ?

Une solution pour s'en protéger est de noter les bugs qu'on a déjà vus et les tests qui permettent de les détecter, pour vérifier de temps en temps qu'ils ne sont pas revenus. Pour des logiciels importants, quand la sécurité des utilisateurs ou d'autres enjeux économiques sont dans la balance, les équipes de développement mettent en place des « tests de non régression », qui sont lancés régulièrement après chaque modification (ajout de fonctionnalité, correction de bug) pour vérifier qu'on n'a rien cassé en modifiant quelque chose.

Vous n'êtes pas obligés de le faire à chaque fonction modifiée. Mais noter les tests à faire et les refaire de temps en temps, ça peut être une bonne idée pour ne pas tourner en rond.

Et si je veux tester le niveau 12, je dois vraiment re-gagner les niveaux 1 à 11 ?

Heureusement que non, sinon il serait impossible de réaliser des jeux difficiles qu'on ne pourrait pas tester jusqu'au bout. Et pour ça il y a une solution utilisée par les développeurs : introduire dans son programme des "cheat codes", ou codes de triche, qui permettent d'obtenir immédiatement un avantage non négligeable, comme récupérer des vies supplémentaires, ou passer au niveau suivant, ou même directement à un niveau plus difficile. C'est très utile pour tester un programme en zappant un bout, ou pour réussir à passer une étape difficile. Libre à vous de les enlever une fois les tests terminés, ou de les laisser pour les joueurs malins qui les trouveraient.

3.3.4. Bêta testeurs

Un très bon moyen de tester efficacement son projet, c'est de le faire faire par quelqu'un d'autre, d'avoir un regard extérieur. Une fois que votre projet est finalisé, ou même quand vous avez une version intermédiaire un peu stable, passez la main à quelqu'un d'autre. Il va peut-être tester les choses autrement que vous et tomber sur un bug qui vous avait échappé.

Un oeil extérieur peut vous dire qu'il ne comprend pas les consignes, qu'appuyer en même temps sur les touches A, B et X pour faire monter la fusée ce n'est pas super pratique, ou que c'est dommage de ne pas pouvoir annuler une étape de construction tant qu'on n'a pas validé. En bref il peut vous permettre d'enrichir votre projet, ou faire naître en vous de nouvelles idées.

Et une fois les modifications faites, n'hésitez pas à retourner vers celle ou celui qui vous a aidé, pour savoir si les consignes sont plus compréhensibles, la combinaison de touches plus pratique ou la fonction d'annulation conforme à ce qu'elle/il attendait.

4. Terminaison

Cette étape porte sur l'évaluation du déroulement du projet, les résultats obtenus et la capitalisation de l'expérience acquise durant le projet. Une formalisation consciencieuse effectuée en début de projet favorise le bon déroulement du projet et de l'atteinte des objectifs. Elle est une source d'information riche pour les projets futurs. L'évaluation a posteriori permet de mettre en avant les faiblesses ou les dysfonctionnements apparus lors du projet.

Une façon d'établir un bilan est de guider les commentaires par des questions comme :

- Qu'avez-vous compris de ce projet ?
- Comment fonctionne-t-il ?
- Qu'aimez-vous le plus dans ce projet ?
- Qu'est-ce qui vous a le plus surpris ?
- Avez-vous rencontré des difficultés en le testant ?
- Avez-vous des suggestions à apporter ?

Ce ne sont jamais les personnes porteuses du projet qui sont jugées, mais bien les projets eux-mêmes dans un but d'amélioration !