Avant chaque conception issue d'un programme informatique, il est toujours bon de savoir pour quelle raison on le fait, le besoin d'un client, amélioration d'un système/programme déjà existant, puis en suite viens la première phase qu'est une phase uniquement analytique, cette première phase est une phase uniquement fait via brainstorming, lié à la programmation ou non bien que celle -ci aura une place très importante, puisque cette phase définira la direction ou l'on ira par la suite, et donc que les développeurs devront suivre par la suite.

Cette première phase commence par définir les exigences et donc les besoins nécessaire pour atteindre le but défini par un client ou par l'équipe de design (dans les Jeux-Vidéos). Le(s) besoin(s) du client sera ensuite appliqué pendant cette phase plus en détail, comme savoir quel langage utilisé, savoir si c'est possible de l'utiliser pour X besoin par rapport au client. En fonction du domaine à laquelle ce programme sera créé, d'autres facteurs seront pris en compte ou changeront complètement par rapport au besoin. Dans un jeu vidéo, le but premier à ce stade est de savoir le langage qui sera utilisé mais aussi une partie de son architecture, qui doit être clair dès le début.

On peut diviser celle-ci en plusieurs vues, qui se regroupe en un modèle d'analyse.

Un cas d'utilisation est défini comme un ensemble de scénarios d'utilisation, chaque scénario représentant une séquence d'interaction des utilisateurs (acteurs) avec le système, chacun étant développé dans cette phase mais renforcé et fait de manière pratique à partir de la 2nd phase, ces 5 vues regroupent tous les aspects nécessaire pour établir un programme (généralement logiciel, mais s'applique aussi dans certains domaines comme les sites Internet et les jeux-vidéos par exemple)

La vue logique est essentielle et décrit, de façon statique et dynamique, le système en termes d'objets et de classes. La vue logique permet d'identifier les différents éléments et mécanismes du système à réaliser. On utilisera un maximum de composants des différentes bibliothèques et Framework à sa disposition. Une recherche active de composants libres et/ou commerciaux est également envisagée

La vue de processus décrit les interactions entre les différents processus, threads (fils d'exécution) ou tâches, elle permet également d'exprimer la synchronisation et l'allocation des objets. Cette vue permet avant tout de vérifier le respect des contraintes de fiabilité, d'efficacité et de performances des systèmes multitâches.

La vue de réalisation permet de visualiser l'organisation des composants (bibliothèque dynamique et statique, code source...) dans l'environnement de développement. Elle permet aux développeurs de se retrouver dans le capharnaüm (un bazar) que peut être un projet de développement informatique.

La vue de déploiement représente le système dans son environnement d'exécution. Elle traite des contraintes géographiques (distribution des processeurs dans l'espace), des contraintes de bandes passantes, du temps de réponse et des performances du système ainsi que de la tolérance aux fautes et aux pannes. Cette vue est fort utile pour l'installation et la maintenance régulière du système.

Chacune des 5 vues auront possiblement des diagrammes et autres schémas dédiés pour décorti quer le travail nécessaire et donc à accomplir pour être sûr que cela fonctionne et permettant donc d'afficher leur cohérence envers elles-mêmes, envers les interactions l'une envers l'autre, et éviter un possible capharnaüm entre membres de l'équipe de dé veloppement pour se retrouver face à toutes les options et contraintes à laquelle ils doivent se plier.

Pour validé la première partie de cette phase qui est purement matière à débat et suggestions, à la fin de celle-ci on devrait avoir des réponses précises lié au besoin (comme montre les 5 vues ci-dessus) : Est-ce que le besoin à bien été traduit par rapport à la demande du client, est-ce que le client sera satisfait du résultat si son besoin à bien été traduit, et finalement voir si ce que l'on a fait est donc cohérent à la demande de départ et voir si tout cela est bien réaliste avec les moyens déployés/nécessaire en plus du temps imparti (s'il y en a un). On peut aussi brièvement mentionner comment se passeraient tel ou tel simulation, voir ces conséquences mais aussi voir plus haut, donc le besoin nécessaire pour avoir ces conséquences.

Par la suite on doit trouver une méthode de conception, comme cité plus haut avec les vues, l'architecture du programme sera donc nécessaire, et les 3 vont ensembles car l'un sera une méthodologie, utilisé pour créer et définir l'architecture du système et donc du programme.

Ensuite on arrive à la 2e phase qui est la phase pratique, qui est donc la programmation. Elle est séparée en 3 parties qui sont respectivement : L'algorithmique, la Programmation et la Modernisation du code, puis sera finalisé par des tests de bouts de codes, puis de l'optimisation de celui-ci.

La phase d'algorithmique est la dernière phase purement théorique de l'opération du processus, le but de cette phase est de préparer le terrain pour la programmation et facilité celle-ci. Le but d'un algorithme est de gérer et résoudre les problèmes via des solutions grâce à des processus systématiques permettant en quelque sorte d'autogérer certains problèmes récurrents dans un programme, comme la gestion de certaines données, des boucles qui doivent être utilisé pour certaines raisons.

Certains langages comme le C, le Pascal et l'assembleur utilisent le système d'algorithmique suivant qui est assez général mais pas aussi simple à percevoir partout selon les langages de programmations :

Les structures de contrôles, séparés en séquences, conditionnelles et boucles et les structures de données contenant les constantes, les variables les tableaux et parfois les structures comme les listes, arbres et graphes.

Pour corriger et compléter celui-ci on aura normalement cela :
Cela utilise les notions correction, complétude et terminaison. La terminaison étant la certitude qu'un programme termine et donc ne créer pas de boucle infini, prouvé par des fonctions positives généralement entières, qui décroit à chaque avancée de l'algorithme. La correction apportera le fait que l'algorithme satisfait la solution posé et donc retourne le problème posé. La complétude garanti à la fin donc après la correction, que celle-ci donne une solution à chaque entrée au problème posé par l'algorithme. Ensuite on passe à la programmation, qui est littéralement l'application de l'algorithmique, qui peut avoir quelques complications selon des bugs, ou bien d'autres facteurs ayant été potentiellement oublié, ou bien secondaires voire tertiaire par rapport au problème, étant juste un obstacle.

On continue ça en modernisant le code, c'est-à-dire en le maintenant à jour, en le gardant aussi visible que possible et aussi général pour la compréhension de chacun, qui se poursuit par des tests

de bouts de codes et validé les 2 procédés du dessus. Et après tout cela on fait une optimisation cu code, pour le rendre le plus fonctionnel et optimisé possible tout en le gardant fonctionnel.

On passe ensuite à la dernière phase qui est la phase reliant le client et l'équipe de développement. On montre tout d'abord au client un test du programme ou du logiciel, tout en expliquant pourquoi avoir fait ça comme ça et pas comme ils auraient pu dire si des changements par rapport au plan de départ ont été fait (ce qui est mieux d'éclaircir dès la première étape car des changements peuvent être fait à tout moment pour maintes raisons), on précise aussi les logiciels, supports (OS), langages et les outils utilisés pour qu'on puisse appliquer le tout sur les ordinateurs des utilisateurs. Suite à ça le client valide ou non, après des derniers tests du programme en utilisant des paramètres algorithmique les plus défavorables possibles (donc les plus bas en tout point) comme le test des composants minimaux pour utiliser le programme, la quantité de ram nécessaire, le temps de réaction, des tests avec des valeurs qui pourraient être cause de casses (donc des paramètres limites, considérés comme critique car ça peut casser le programme facilement), cela permets de souligner l'innocuité du programme envers les utilisateurs et le client mais aussi son efficacité, respectant le besoin du client, en vérifiant sa conformité et sa cohérence envers ce que le client à demander.
Si tout c'est bien passé on finis par l'inclusion du programme/logiciel dans l'entreprise du client et conclut la finalisation du produit.

On peut voir ci-dessous un bref schéma résumant de manière vulgaire tous les processus jusqu'à la fin.