Série de TD 1 avec corrections détaillées

Exercice 1 : Questions de cours.

1) Citer les quatre principaux facteurs de qualité d'un logiciel. Proposer deux autres facteurs et dire dans quelles circonstances, ces deux derniers deviennent – ils plus importants que les autres ?

2) Expliquer pourquoi le modèle de cascade ne reflète pas exactement les activités de développement du logiciel.

3) Établir un tableau montrant pour chacune des étapes du modèle en V du cycle de vie des logiciels, les documents à délivrer en sortie.

4) Citer les points forts et les points faibles de chacun des modèles en V et à incréments.

5) Comparer le prototypage jetable par rapport au prototypage non jetable.

6) Montrer, pour chaque modèle de cycle de vie, les domaines d'application les plus appropriés.

7) Quelles sont les qualités requises dans un cahier de charges ?

8) Comment peut-on réduire l'écart entre les besoins réels d'un client et les besoins exprimés dans le cahier de charges ?

Réponses Exercice1:

1. Les quatre principaux facteurs de qualité d'un logiciel sont : fiabilité, efficacité, ergonomie et maintenabilité (voir cours).

Les facteurs de qualité d'un logiciel peuvent être décrits comme suit :

Fiabilité : La fiabilité est la capacité de réaction adéquate lors d'une présence de conditions anormales. Le logiciel doit fonctionner conformément aux besoins des clients (précision des fonctionnalités développées), avec absence de bugs et intégrité (protection du code et des données contre les accès non autorisés) ainsi que des tests réussis pour tous les cas.

Efficacité : L'efficacité est la capacité d'un système logiciel à optimiser la consommation des ressources avec une performance optimale (exemple : économie de mémoire, rapidité d'exécution). Pour cela, il faudrait prendre en considération la complexité des algorithmes et utiliser des machines plus performantes.

Ergonomie : L'interface utilisateur devrait s'adapter avec les capacités réelles des usagers:

- Facilité d'utilisation (apprentissage facile, interprétation d'erreurs, rattrapage en cas d'erreur d'utilisation).
- Accessibilité.
- Documentation pour l'utilisateur.

Maintenabilité (facilité de maintenance) : Les logiciels ayant une longue durée de vie doivent être conçus et réalisés de façons à minimiser les coûts des corrections d'erreurs non détectées dans les phases antérieures ainsi que ceux des adaptations aux évolutions éventuelles. Trois types de maintenance peuvent être distingués :

Maintenance corrective : corriger les erreurs de développement non détectées par les tests.

Maintenance adaptative : adapter les fonctionnalités du logiciel aux changements technologiques (hard et soft).

Maintenance évolutive : réaliser et intégrer de nouvelles fonctionnalités exigées par l'évolution de son environnement.

Afin de permettre la maintenabilité, le développement de logiciel devrait respecter certains critères (qualités pour le développeur) :

- Conception modulaire,
- Structuration et lisibilité du code,
- Extensibilité (facilité d'extension) : capacité des éléments logiciels à servir à la construction d'autres applications.
- Documentation technique lisible, structurée, extensible
- Portabilité : la facilité avec laquelle des produits logiciels peuvent être transférés d'un environnement logiciel ou matériel à l'autre. Pour cela, il faudrait utiliser des langages standardisés, indépendants du matériel et du système d'exploitation.
- Compatibilité : est la facilité avec laquelle des éléments logiciels peuvent être combinés à d'autres. Pour cela, il faudrait assurer un découplage données/traitements, externaliser certaines fonctions en utilisant des API (Application Program Interface), utiliser des formats standards (exemple : XML).

Autres critères : D'autres qualités spécifiques concernent par exemple la sûreté de fonctionnement et la sécurité informatique. Ces deux paramètres peuvent influencer directement les autres critères. Par exemple si pas de sécurité on n'aura pas l'intégrité du code ni celle des données et donc le critère de fiabilité sera remis en cause.

2. Le modèle en cascade ne reflète pas exactement les activités de développement du logiciel car le client n'intervient qu'au début et à la fin du développement, donc ses besoins réels risquent d'être interprétés de manière superficielle ou erronée dans le cahier des charges, ce qui sera répercuté négativement sur les phases suivantes de conception et de réalisation.

3. Les documents du modèle en V :

Phase du cycle de vie	**Document**
Analyse des besoins ***Spécification***	**Cahier des charges.** Spécification générale, spécification technique des besoins.
Conception architecturale,	Livrable de conception (modélisation des données, description des composants, sous-composants).
Conception détaillée	Rapport conception détaillée (fonctions par composants, algorithmes).

Programmation	Code + commentaires,
Tests unitaires,	*Rapport de tests :* - Jeux de tests
Tests d'intégration,	- Plan de tests - Scénario de tests,
Tests d'acceptation	- Rapport (PV) indiquant si le logiciel répond aux besoins du client ou non.

Nb. Une fois le logiciel accepté, il sera livré au client (ou commercialisé) avec la documentation nécessaire d'installation et d'utilisation.

4. Les points forts et faibles des modèle V et par incréments :

Modèle V	**Modèle par Incréments**
(-) Si chaque étape dure plus longtemps alors le coût sera plus important. **(-)** Validation tardive du système. **(+)** Prévention des erreurs car la vérification se fait après chaque étape. **(+)** Bonne réflexion côté client et prestataires (se poser les bonnes questions au début du projet).	**(-)** Nécessité de définir précisément le noyau (difficulté d'identifier le noyau). **(-)** Se focaliser sur le noyau risque de compromettre le développement des fonctionnalités secondaires. **(+)** Grande intégration et cohérence dans un système complexe. **(+)** Minimiser les risques d'inadéquation avec le produit final.

5. Différence entre prototype jetable et non jetable : voir le cours.

6. Modèles de cycles de vie selon les domaines d'application: voir le cours.

7. Qualités d'un cahier des charges : voir le cours

8. Pour réduire l'écart entre les besoins réels du client et les besoins exprimés dans le cahier des charges :

- Exprimer les besoins de manière exacte.
- Le client suit le processus et le valide à chaque étape.
- Faire des simulations
- Avoir des connaissances du domaine afin de pouvoir valider.
- Etc.

Exercice 2 : Choix d'une Solution Informatique

Une université voudrait s'équiper d'un système intégré de gestion des étudiants et qui prendrait en compte tous les détails concernant les étudiants y compris les informations personnelles, les cours suivis et les notes obtenues aux examens. Les trois approches possibles sont :

1. Acheter un système de gestion de bases de données et développer son propre système basé sur cet outil.
2. Acheter un système comparable à celui d'une autre université et le modifier pour ses propres besoins.
3. Se joindre à un groupe d'autres universités, établir un cahier des charges commun, contacter une société de logiciels qui développera un seul système pour tous.

Question : Identifier deux risques possibles pour chacune de ces stratégies et proposez des techniques de résolution de risque qui permettraient de décider quelle approche adopter ?

Réponse Exercice 2 :

Solution 1 : développement par l'université

- Achat du SGBD qui peut être trop élevé
- Temps de développement trop important
- Manque de compétences en matière de développement
- Apparition d'autres coûts supplémentaires comme l'achat d'équipements et les recrutements.

Solution 2 : Achat d'un logiciel et adaptation

- Le logiciel peut ne pas couvrir correctement/totalement les besoins
- Difficultés d'adaptation
- Implication de changements profonds dans l'organisation

Solution 3 : Elaboration d'un même cahier des charges et sous-traitance (développement effectué par une société de développement)

- Difficulté d'établir un cahier de charges commun
- Lenteur du processus d'analyse des besoins et difficultés de coordination
- Apparition de conflits concernant le financement du projet
- *Non équité des frais* à engager par rapports aux besoins des différentes parties
- Problème des mises à jour qui peuvent ne pas arranger toutes les parties

Décision :

1- Prise en compte de l'avis d'un expert en développement qui va nous orienter sur le choix selon son expérience dans le développement de logiciels, tout en prenant en compte des conditions de l'entreprise (budget, délais, compétences, matériels disponibles…).

2- L'utilisation de méthodes plus rigoureuses (basées sur des fonctions mathématiques) par exemple on va optimiser notre choix en considérant les différents facteurs (budget, délais, compétences, fonctionnalités à développer…) et on va pondérer chaque facteur et en sortie on aura le choix le plus approprié pour notre cas (exemple le moins couteux, ce qui reviendrait à minimiser la fonction objective dans ce cas).

Exercice 3 : Difficulté de Spécifier un Besoin Fonctionnel

Soit la spécification de la règle de notation à un examen suivante :

« L'examen est un ensemble de 20 questions à réponses multiples. Chaque bonne réponse à une question rapporte 1 point. Chaque mauvaise réponse fait perdre 1/3 de point. Chaque question sans réponse donne 0 point. »

Question :

1. Pensez-vous que cette spécification est claire ? Afin de le vérifier, calculez chacun la note des 3 étudiants suivants : (Recensez les résultats possibles et Proposez une spécification plus précise).

Réponse Exercice 3 :

L'application numérique donne lieu aux résultats suivants :

	Réponse correcte	*Incorrecte*	*Sans*	*Double réponse*	*Résultat*
Ahmed	*10*	*5*	*5*		***10 – 5/3 = 8,33/20 ou 8,50/20 ou 8,3333333/20 selon la précision de calcul utilisée ou possible.***
Nora	*4*	*16*			***4 – 16/3 = -1,33/20 ou -1,50/20 ou 0/20 ou -1,33333333/20 selon la précision de calcul utilisée ou possible.***
Wissam	*10*	*3*	*4*	*3 (1 juste, 1 faux)*	***Trois réponses sont possibles pour la formule 10 – 3/3 + ? (problème des doubles réponses).*** ***1. 10 – 3/3 + 3 = 12/20 en ne comptant que les réponses justes pour la double réponse.*** ***2. 10 – 3/3 + 3/3 = 8/20 en ne comptant que les réponses fausses.*** ***3. 10 – 3/3 + 0 = 9/20 en considérant qu'il n'y a pas de réponses.***

Rien n'a été spécifié concernant la précision de calcul ou la méthode d'arrondissement du résultat obtenu en cas de présence de la partie décimale.

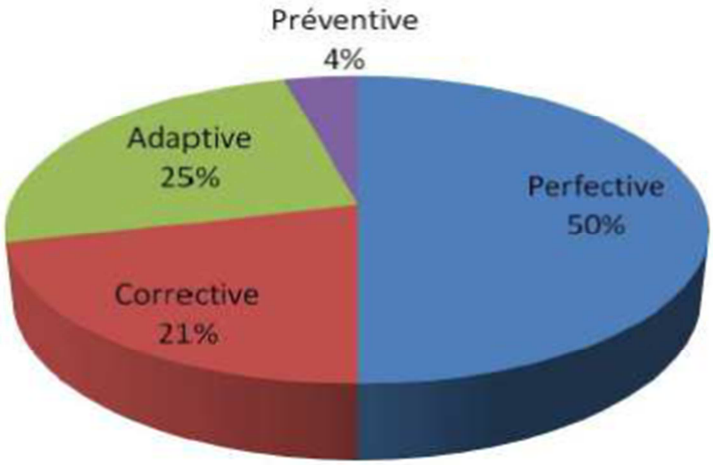
Rien n'a été clairement spécifié quant au traitement du cas de double réponse, i-e, présence de réponses fausses et de réponses justes pour une même question à plusieurs réponses.

Donc, le manque de précision dans l'expression du système a donné lieu à diverses interprétations possibles (ambiguïté).

Dans ce cas, il faudrait modifier la spécification comme suit :

« L'examen est un ensemble de 20 questions à réponses multiples. Chaque bonne réponse à une question rapporte 1 point. Chaque mauvaise réponse fait perdre 1/3 de point. Chaque question sans réponse **ou avec des réponses multiples** donne 0 point. **La note est arrondie au demi-point le plus proche**. **La note minimum est 0»**.

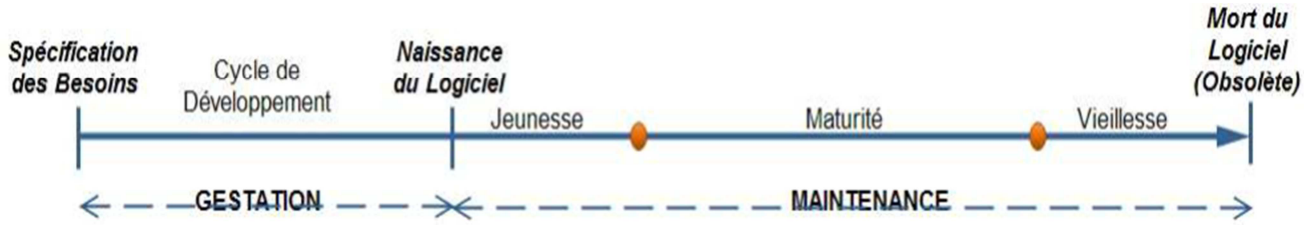
Exercice 4 : Importance de la Maintenance du Logiciel



Réponse Exercice 4 :

Ce schéma représente les variantes de maintenance logicielle existant sur le marché. En effet, après la mise en production ou exploitation d'un produit logiciel commence une phase très importante pour la **longévité** de ce dernier. Cette phase se nomme maintenance.

Rappelons d'abord à l'aide du schéma suivant, l'importance de la phase de maintenance dans le cycle de vie d'un logiciel :



Comme on peut le constater, la phase de maintenance doit assurer la longévité du produit logiciel en prenant en compte différents types de maintenances, y compris ceux relatifs à des aspects de marketing et de compétition.

1. D'après le schéma, quels sont les types de maintenance considérés et qu'elle est sa signification ?

Il y a sur le schéma quatre types de maintenance :

Maintenance	*Rôle*
Perfective	*Permet de perfectionner continuellement un produit logiciel. Ce type de maintenance est fortement recommandé pour rester compétitif sur le marché.*
Corrective	*Permet de corriger les erreurs ou bugs contenus dans le code. Tout logiciel ne peut être exempt d'erreurs, l'essentiel c'est d'arriver à corriger les erreurs et à veiller à vérifier de façon continuelle les doléances des utilisateurs en la matière.*
Adaptive	*Permet d'adapter le produit logiciel aux changements matériels ou logiciels qui surviennent sur le marché comme par exemple la sortie d'un nouveau modèle de machine ou bien l'arrivée d'un nouveau système d'exploitation.*
Préventive	*Permet de prévenir certains risques surtout dans les systèmes pouvant mettre en péril des vies humaines ou bien des équipements très coûteux.*

2. Selon vous que représente le pourcentage associé à chaque type de maintenance ?

Il s'agit des parts du marché pour chaque type de maintenance. Comme nous pouvons le constater, la maintenance perfective est celle qui est la plus utilisée car elle permet d'améliorer continuellement le produit logiciel. La corrective et l'adaptive dépendent de l'environnement (utilisateurs, autres constructeurs hard et soft). La préventive coûte généralement cher et l'investissement n'est pas toujours rentable.

3. Sur un même produit logiciel, pensez-vous qu'il soit possible d'opérer plus d'un type de maintenance ?

Oui, il est possible d'opérer plusieurs types de maintenance sur un même produit à condition qu'il n'y ait pas interférence entre les différentes tâches, i-e, éviter la redondance des tâches d'un type de maintenance à l'autre.

Exercice 5 : Analyse d'un Système « La Pompe à insuline »

Une pompe à insuline est un appareil médical que l'on place sur une partie du corps d'un malade diabétique (bras, cuisse, jambe, …) afin de contrôler la glycémie et injecter la quantité appropriée d'insuline dans le corps du patient de manière automatique. Les figures 1 et 2 illustrent respectivement un modèle de pompe ainsi que le principe général de son fonctionnement.

Figure 1 : La Pompe à Insuline

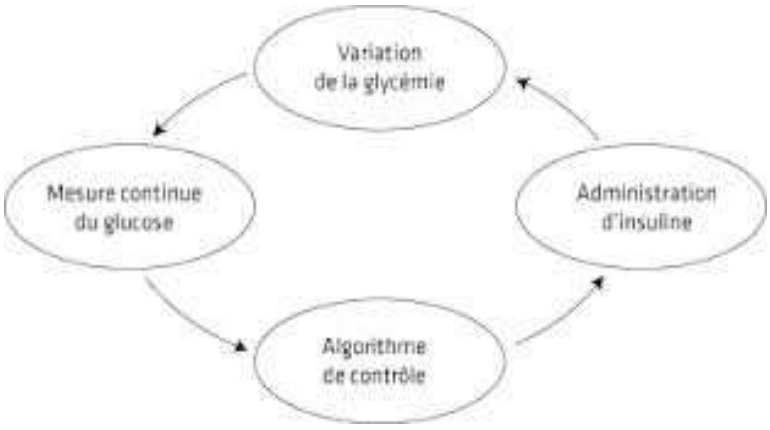


Figure 2 : Principe Général de Fonctionnement

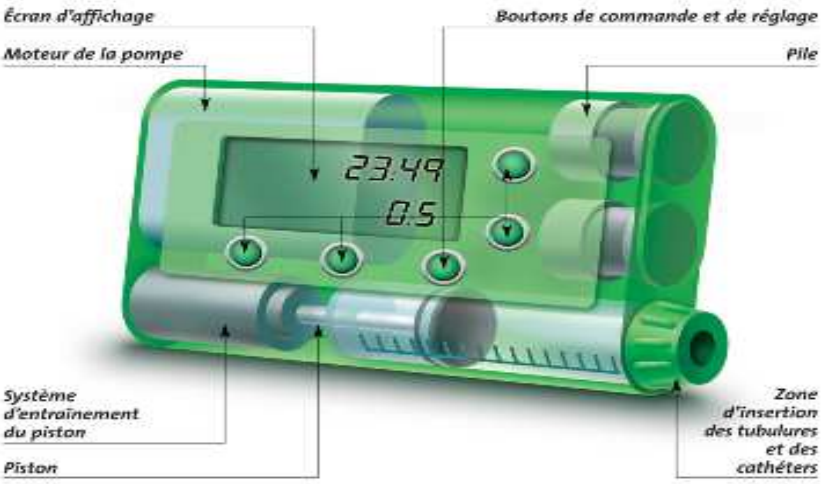


Figure 3 : Détail des Composants de la Pompe

Comme on peut le deviner, la pompe est un système composé d'une partie matérielle « Hardware » et d'une partie logicielle « Software » comme le montre la figure 3.

Questions :

1. D'après le schéma général de fonctionnement (figure1) et les détails (figure 3), quelles sont d'après vous les fonctions à offrir à l'utilisateur final, i-e, le malade ?
2. En termes de risques, donnez les risques majeurs qui pourraient mettre en péril la sécurité ou la vie du diabétique en justifiant.
3. D'après ce qui a été vu en cours, quel modèle de cycle de vie pourrait être le plus adapté pour le développement d'un tel système.
4. A quels domaines d'expertise doit-on faire appel pour la compréhension, la conception puis la programmation de l'algorithme de contrôle (figure 2).

Réponses Exercice 5:

1. Type du système :

- Il s'agit ici d'un système « temps-réel » et « critique », car non seulement la précision de l'algorithme est importante mais aussi le temps de réponse pour l'affichage des résultats.
- C'est aussi un système embarqué car il est basé sur une partie matérielle (hardware) et une partie logicielle (software).

2. Les fonctions à offrir à l'utilisateur final (le malade):

L'application implémentée inclut un ensemble de fonctionnalités accessibles à travers des **clics sur des boutons**. Nous citons à titre non exhaustif quelques fonctionnalités :

- Réglage du doseur d'insuline.
- Information sur la quantité restante d'insuline dans la cartouche.
- Consultation de l'historique des prélèvements et dosages d'insuline.
- Commande de mise en marche et d'arrêt.
- Information sur l'état de la batterie.
- Vidage de la mémoire.
- Etc.

3- Les risques majeurs:

- Le système ne signale pas l'état de la cartouche d'insuline : ce qui risque de causer des conséquences dramatiques pour le patient, qui se croit administrer la quantité d'insuline nécessaire mais en réalité cette quantité **est insuffisante** à la dose nécessaire. Pour cela, il faudrait que le système prévoie la notification du patient, en affichant un message sur écran et/ou signal sonore d'avertissement.
- Le système doit surveiller l'alimentation en énergie et informer l'utilisateur en cas cette dernière a atteint un niveau faible (pour procéder au rechargement de la batterie). De même, il faudrait que le système prévoie la notification du patient, en affichant un message sur écran et/ou signal sonore d'avertissement (pour protéger la sécurité du patient qui pourrai risquer sa vie en cas l'appareil s'éteint et le patient ne trouve pas un moyen pour rechargement).

2 cas : hypoglycémie / hyperglycémie :

- Dans le cas où le système détecte ***un état d'hyperglycémie*** (un taux de sucre élevé dans le sang), l'algorithme de calcul de la dose d'insuline doit être implémenté de manière **précise** et en **temps réel.** Les conséquences d'une erreur de calcul pourraient mettre la vie du patient en danger (exemple : si le patient est en état d'hypoglycémie ou normal et le système détecte un état d'hyperglycémie alors le patient va essayer de faire monter le taux de glycémie, ce qui risquerai d'aggraver son état).
- Par contre, dans ***le cas d'hypoglycémie*** (un taux de sucre faible dans le sang) alors le système **ne doit pas administrer de l'insuline au patient** (sinon il pourrait risquer sa vie).

4. Modèle de cycle de vie le plus adapté :

Le modèle par prototypage est le plus approprié, à commencer par un prototype jetable qui ne va inclure que les fonctions essentielles du système comme **le déclenchement du doseur d'insuline**. Une fois, cette fonctionnalité a été validée par les spécialistes du domaine médical, un prototype réel sera développé incluant toutes les autres fonctionnalités du système. Afin de valider ce système, des tests sur des animaux seront effectués, puis sur un échantillon d'humains volontaires. Le système ne sera validé que si les tests sont satisfaisants (concluants) et auquel cas l'appareil sera mis en production et commercialisé.

5- Domaines d'expertise impliqués dans ce projet :

- Experts du domaine médical.
- Spécialistes dans le développement matériel (hardware) pour réaliser la partie matérielle.
- Spécialistes dans le développement logiciel (software) pour la réalisation du logiciel, incluant toutes les fonctionnalités dont l'algorithme de calcul du doseur d'insuline. Ce logiciel sera accompagnée d'une interface utilisateur (affichage sur écran / saisie de données).

Exercice 6 : Elaboration d'un Cahier des Charges

Imaginez-vous un moment dans la peau d'un informaticien « professionnel » qui a pour mission d'informatiser un cabinet médical. Une première phase importante est l'élaboration d'un cahier des charges qui fera office de contrat entre vous et le médecin « votre client ».

Question :

Expliquer les étapes par lesquelles vous passeriez pour accomplir la mission importante d'élaboration d'un cahier des charges, dont dépendront le succès et la suite du travail de développement.

Exercice 7 : Etude d'un cas d'échec « ARIANE 5 »

Le crash d'ARIANE 5 est le résultat d'une succession d'erreurs techniques et stratégiques. Le calculateur en cause était chargé de fournir des données au cœur du contrôle de vol. Il avait été développé pour Ariane 4, et avait effectué sans problèmes plusieurs vols.

Les erreurs de stratégies sont que le calculateur est repris tel quel sur Ariane 5, sans aucune reprise des spécifications, ni tests complémentaires. Malheureusement les données d'entrées d'ARIANE 5 sont totalement différentes de celles d'ARIANE 4. Ce qui aura pour conséquence de provoquer le bug qui entrainera la destruction du lanceur.

Première erreur technique :

Le code contient tout bêtement une affectation d'une donnée 64 bits vers une donnée 16 bits. Comme les données d'entrées d'ARIANE 5 sont plus importantes que prévu, il arrive qu'une valeur de plus de 16 bits soit affectée à la variable codée sur 16 bits. Une exception est alors logiquement levée.

Deuxième erreur technique :

En langage ADA, un mécanisme permet de récupérer ce type d'exception. Ici aucun mécanisme de protection n'est prévu. L'exception est transmise au cœur du contrôle de vol, qui la traite comme n'importe quelle autre valeur fournie par le calculateur. D'où son comportement aberrant, qui entraîne la perte de trajectoire, puis par conséquent le mécanisme d'autodestruction.

Un deuxième calculateur fonctionnait en redondance à chaud (fonctionnement en parallèle). Mais comme sa conception était en tout point identique au premier, les mêmes causes ont produit les mêmes effets.

Questions :

1. Recenser et classifier les défaillances et les fautes conduisant à ces dernières dans ce cas.
2. Quelles leçons peut-on tirer de ce cas ?
3. Le développement de systèmes industriel ou de pointe est souvent complexe et demande beaucoup de temps pour sa réussite. Quels sont les modèles les mieux adaptés pour de tels projets et pourquoi ?

Réponses Exercice 7:

1. Les défaillances: les données d'entrées d'ARIANE 5 sont plus importantes que prévu, il arrive qu'une valeur de plus de 16 bits soit affectée à la variable codée sur 16 bits au lieu de 64 bits (affectation d'une donnée **64 bits** vers une donnée **16 bits**), ce qui impliquera une exception.

L'exception est traitée comme n'importe quelle autre valeur fournie par le calculateur. D'où son comportement aberrant, qui entraîne la perte de trajectoire, suivi par un mécanisme d'autodestruction.

2. Les leçons à tirer de ce cas :

- Ne jamais réutiliser un composant matériel ou logiciel sans le tester dans son nouvel environnement.
- Le manque ou l'insuffisance de test d'intégration entre le module de calcul d'ARIANE 4 et les nouvelles parties d'ARIANE 5 a conduit à l'échec du projet.
- certaines pannes n'apparaissent qu'au moment du fonctionnement réel du système.

3. Modèles adaptés

- Pour des modèles de ce type il est préférable d'utiliser les approches itératives et incrémentales car elles permettent un développement progressif du produit tout en veillant à des intégrations de qualité.

- Si le système est composé d'une partie matérielle et une autre logicielle alors une approche par prototypage serait préférable. Si la complexité du système nécessite de considérer certains critères ou propriétés alors le prototypage jetable permettra de fixer les réponses qui permettront au projet de continuer ou pas.

- Si le projet présente un important facteur de risque alors l'approche en spirale prendra correctement cet aspect en charge tout en permettant l'itération et l'incrémentation.

- Possibilité de combiner différents modèles de cycle de vie (modèles hybrides) et qui peuvent répondre aux besoins dans des systèmes spécifiques.