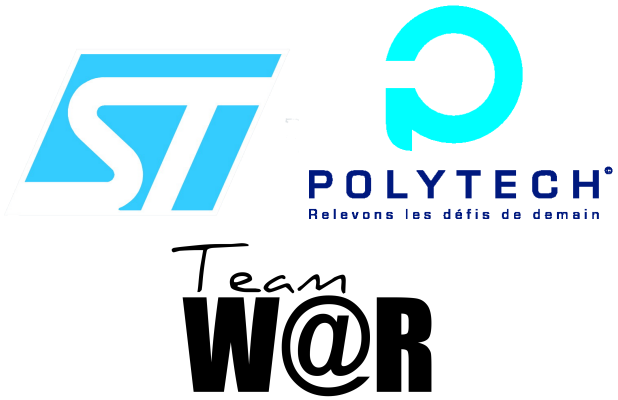
Projet de Fin d'Étude
Rapport Final

« *Recherche de méthode d'estimation de volume de production à risque* »

Équipe 5ème Année : Team-War
Jaafar AMRANI-MESBAHI
Fabien GARCIA
Abdelali NAIT BELKACEM
Rahma NAKARA
Philippe NGUYEN



Génie Industriel et Informatique

Tuteur :
Claudia Frydmann

Mercredi 19 Janvier 2011

Team-war@prunetwork.fr

Table des matières

Introduction

Première partie

Contexte de l'étude

CHAPITRE 1

Présentation de STMicroelectronics

STMicroelectronics (plus connu sous ST) est une société internationale de semi-conducteurs, qui développe, fabrique et commercialise des puces électroniques. Elle a été initialement créée en 1987, de la fusion de SGS Microelettronica et de Thomson Semiconducteurs, sous le nom de SGS-THOMSON. Puis renommée en STMicroelectronics, suite au retrait de Thomson du capital en 1998.

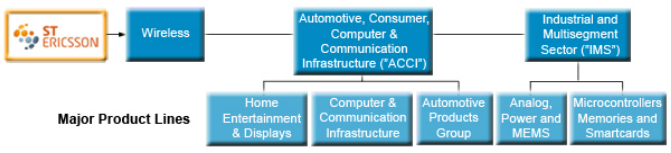
La société mère STMicroelectronics est de droit hollandais (enregistrée à Amsterdam), la direction administrative est regroupée en grande partie sur le site de Genève en Suisse, mais la direction opérationnelle du groupe est toujours italienne, implantée sur le principal site du groupe à Agrate (à côté de Milan). C'est une société en expansion économique depuis sa création, son chiffre d'affaire a même doublé en dix ans depuis son renommage (CA : $9,84 milliards en 2008 contre $4,25 milliards en 1988).

S'appuyant sur sa technologie et son expertise, ST se positionne au 5^e rang mondial dans la fabrication des semi-conducteurs depuis 2005. Elle comprend approximativement 51000 employés répartis dans plus de 15 sites de production (cf Fig 1.0.1), plus de 39 centres de R&D et plus de 78 bureaux de vente dans le monde.

FIGURE 1.0.1. Principaux sites de production

ST propose divers groupes de produit, chaque groupe étant lui-même composé de plusieurs divisions qui gèrent de façon autonome la conception, la fabrication et la commercialisation de leurs produits (cf Fig 1.0.2) :

FIGURE 1.0.2. Divers groupes de produit



Voici quelques exemples d'application des produits semi-conducteurs et de la technologie de ST dans la vie quotidinne (cf Fig 1.0.3) :

FIGURE 1.0.3. Exemple des produits

CHAPITRE 2

Cahier des charges

2.1. Fonctionnement actuel de l'entreprise

La pratique actuelle de STMicroelectronics consiste à compter le nombre de plaques de silicium travaillées entre deux mesures. Ce comptage peut être référencé à une machine de production ou à une technologie et il ne prend pas en compte les paramètres des plaques.

Le cycle de production moyen est de trois mois, tracé par un système GPAO. La production est du type « jobshop [1] » par lots de 25 plaques de silicium. Ce dernier ne suit pas le principe FIFO [2] ce qui constitue une des difficultés majeure de cette étude.

Il existe deux types de contrôle pour assurer les volumes de production à risque :

- Contrôle de qualité sur des plaques de silicium [3] travaillées à l'aide des appareils de métrologie. Ce type de contrôle est très onéreux et nécessite de quelques heures pour contrôler seulement 3 plaques.
- Contrôle de fiabilité des machines de production, c'est une tâche de maintenance périodique qui consiste à vérifier son bon fonctionnement à l'aide d'une plaque de silicium vierge. Ce type de contrôle n'est pas onéreux ni trop long mais qui engendre l'indisponibilité de la machine de production.

Les règles de contrôle sont basées sur l'expérience des contremaîtres, par pifométrie.

2.2. Mission du PFE

Le coût de la non-qualité est souvent l'ancre qui peut faire couler une entreprise. D'une part, par son coût important qui ne rentre pas dans la valeur ajoutée, et puis aussi par son impacte sur le marché concurrentiel.

Notre travail consiste à rechercher une ou des solutions susceptibles d'améliorer la gestion des contrôles et de pouvoir maîtriser la production à risque.

La mission telle qu'elle est mentionnée dans le cahier des charges du client est la suivante : « Le travail consistera en une étude de faisabilité sur l'évaluation des 'wafer@risk' [4] ».

En principe, notre mission se limite à la recherche des solutions et l'étude de leur faisabilité. Étant donné le domaine d'étude risque d'être limité en source d'information due à la confidentialité et à la spécificité du domaine d'activité (micro-électronique). Nous serons peut-être amener à imaginer notre propre solution.

1. Production en atelier, mouvement brownien des produits dans le cycle de production.
2. First In First Out
3. Contrôle du produit par échantillonnage
4. « Wafers At Risk » (W@R) est un indicateur qui représente le nombre de plaques traitées sur un équipement depuis la dernière tâche de contrôle.

2.3. Plan de l'étude

Une étude préliminaire pour balayer le sujet et les pistes de recherche d'information possibles. Ce qui nous amène à l'étude de l'art, une sorte de travail de *Benchmarking*. A partir du bilan de cette étude, de fouille d'information, nous essaierons de trier et de garder les solutions pertinentes pour une étude plus approfondie. Dans le cas où l'état de l'art ne permet pas de répondre pertinemment à nos attentes. Une tentative solution sera développer par notre équipe pour répondre à notre problème d'optimisation, suivi de la phase de simulation et de validation de la solution.

Deuxième partie

Etat de l'art

CHAPITRE 3

Recherches récentes et autre thèses

Dans le cadre de notre recherche, une série de document de recherche nous a été confié, ces articles et supports ont été présenté lors de différentes conférences en rapport avec la micro electronique.

3.1. Operational risk evaluation and control plan design

Ce document est un support de présentation de Belgacem Bettayeb, qui présente sa thèse « An approach for operational risk evaluation and its link to control plan » en collaboration avec P. Vialletelle, S. Bassetto et M. Tollenaere.

Cette approche d'évaluation des risques est basée sur l'évolution du « Potential Loss » (défectivité potentielle). Cette évolution est supposée linéairement croissante avec le nombre d'exécution (traitement par la machine), sans contrôle, sur un horizon donné. Ce risque potentiel est réinitialisé, à une valeur seuil, après chaque tâche de contrôle ou de maintenance. L'évolution du « Potential Loss » dépend de la gestion (planification) des tâches de qualité et de maintenance. De plus s'ajouter à cela une fonction d'évaluation de la valeur ajoutée des tâches de contrôles mises en place afin de déterminer la bonne planification de ces tâches pour un résultat optimal du risque.

Cependant ce support de présentation n'apporte pas plus d'information pertinente que l'extrait de la thèse pour être retenu comme solution de notre problème.

3.2. A novel approach to minimize the number of controls in the defectivity area

Cette présentation expose une nouvelle approche de gestion des tâches de contrôle, issue des travaux de J. Nduhura Munga, S. Dauzère-Pérès, C. Yugma et P. Vialletelle.

Les objectifs de cette nouvelle approche sont :

- d'évaluer les contrôles en défectivité.
- de déterminer les processus de fabrication qui sont sur-contrôlés ou sous-contrôlés
- de pouvoir maîtriser dynamiquement les risques [1] sur les divers équipements de production par des contrôles en défectivité.
- de pouvoir réduire les contrôles inutils en évitant « intelligemment » les produits qui n'apportent pas d'information pertinente.

Cette étude repose sur l'indicateur « Wafers At Risk » (W@R), qui represente le nombre de plaques traitées sur un équipement depuis la dernière tâche de contrôle. Cet indicateur sera exploité par un algorithme de calcul des « Permanent Index per Context (PIC) ». Un PIC est une sorte de compteur qui s'incrémente à chaque fois que le « context » est vérifié et se réinitialise lors d'un évènement spécial (maintenance préventive ou tâche de contrôle). Le « context » peut être un équipement, une cellule ou une recette.

1. nombre de produits traités sur un équipement entre deux contrôles en défectivité.

L'implémentation de cet algorithme sur une ligne de production, nécessite de définir au préalable certains produits qui vont être contrôlés lors de la permière exécution, permet d'atteindre les objectifs définis précédemment.

Les résultats de l'implémentation de l'algorithme sur une ligne de production montrent, au bout de deux semaines, qu'une réduction de 35% des produits qui peuvent être exemptés de contrôle sans incident sur le taux de défectivité de la production.

Malheureusement, nous aboutissons à la même conclusion pour la thèse de « Optimizing return on inspection through defectivity smart sampling » qui traite plus ou moins la même approche. Résultats intéressants mais manquent de consistance. De plus cette approche ne permet pas réduire les risques en défectivité mais seulement l'échantillonnage pour la mesure. Cette la réduction de la population des échantillons réduit par conséquence des coûts des tâches de qualité.

CHAPITRE 4

Méthode D^3

4.1. Généralités

Cette algorithme[**?**] consiste à regarder à la sortie tous les produits (ayant des chemins de productions différentes) et déterminer quels sont les points dans la production qui sont le plus sensible. L'idée est d'observer à la fin les produits finis et de déterminer les opérations à risques par analyse statistique des défauts.

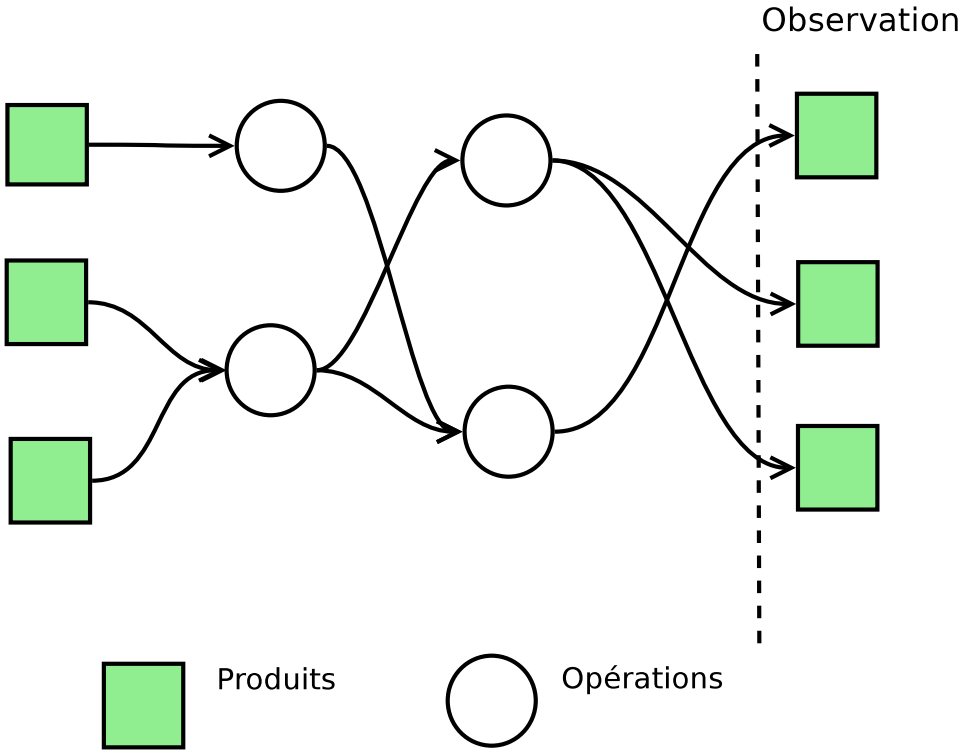
4.2. Objectifs de la méthode

Objectif minimiser le nombre maximum de test pour identifier tous les objets défectueux.

4.3. Hypothèses de la méthode

– Il n'y a que 2 état pour le produit *defective* ou *good*.

FIGURE 4.1.1. Schéma de la situation traité par D^3



Les produits issus de la production proviennent de différents chemins l'analyse des produits finis permet de connaître les opérations sensible grâce à la méthode D^3

- Chaque test se fait sur un sous ensemble n objets et n'ont que deux possibilités *positive* ou *negative*.
- Un résultat *positive* indique que l'échantillon (ou sous ensemble) contient au moins un élément *defective*
- Un résultat *negative* indique que l'échantillon est pur et qu'il ne contient que des éléments *good*.
- Deux type de méthodes :
  - Stochastique : les objets ont une probabilité d'être *defective* et le but est de diminuer le nombre de test attendu pour identifier tous les éléments objet *defective*.
  - Déterministe : le nombre d'éléments *defective* est connu à l'avance, l'objectif est de minimiser le nombre maximal de test pour identifier tous les éléments *defective*.
- $M(n,d)$ nombre optimal de test optimal pour identifier les d éléments *defective* quand on connaît le nombre d parmi n éléments.
- $T_A(n,d)$ nombre de test requis par l'algorithme A pour identifier parmi n éléments tous les d *defective* élément quand on connaît d à l'avance.
- Si $d = 0$ l'algorithme effectuera $\log(n)$ tests inutile. Dans ce cas on vérifiera les éléments inconnus.
- Donc A est α-compétitif si pour $0 \leq d \leq n$, $T(n,d) \leq \alpha M(n,d) + \beta$ avec $\alpha = 2,75$[1] et β une constante et un algorithme de ce genre existe.

4.4. L'algorithme

L'algorithme utilise la stratégie suivante. Il teste des collections disjointes d'item de taille $1, 2, ..., 2^i$ jusqu'à ce qu'il trouve un élément *defective*.

À cette étape l'algorithme a détecté $1+2+3+...+2^{i-1} = 2^i - 1$ bon éléments et a trouvé une collection contaminée de taille 2^i. En utilisant une recherche binaire l'élément peut-être détecté par i tests additionnels.

L'algorithme effectuera $i+1$ tests, donc pour le prix de $2i+1$ tests l'algorithme apprend l'état de 2^i objets.

En d'autre mots l'état de a nouveaux éléments est connu en effectuant $2\log a + 1$ tests.

4.5. Algorithme détaillé

Algorithme DOUBLE.

Algorithm 1 Algorithme Double pour tester n objets

```
Algorithm Double
    U:={1,...,n}
    D:=0;
    G:=0;
    While|U|>= 3 do
        {x,y,z}:= 3 objets arbitraire de U
        TEST({x,y,z});
        if positive 3-TEST({x,y,z});
        if negative
            U:=U-{x,y,z};
            TEST(U);
            if negative
                G:=G+U;
                U:=0;
            if positive
                k:=4;
                repeat
                    C:=k arbitraire de U ou U si k>=|U|;
                    TEST(C);
                    if positive
                        x:=BINARY-TEST(C);
                        D:=D+{x};
                        U:=U-{x};
                        abort-repeat;
                    if negative
                        k:=2k;
                        G:=G+C;
                        U:=U-C
                end-repeat
    end-While
    FINAL-TESTS;
end-algoritm
```

Algorithm 2 Procédure 3-TEST

```
Procedure 3–TEST({x,y,z})
    TEST({x});
    if positive
        D:=D+{x};
        TEST({y};
        if positive D:=D+{y};
        if negative G:=G+{y};
        U:=U–{x,y};
    if negative
        G:=G+{x};
        TEST({y});
        if positive
            D:=D+{y};
            U:=U–{x,y};
        if negative
            D:=D+{z};
            U:=U–{x,z};
```

Algorithm 3 Algorithme Double pour tester n objets

```
Procedure BINARY–TEST(C);
    k:=|C|;
    repeat
        C':=|k/2| éléments arbitraire de C
        TEST(C');
        if positive C:=C';
        if negative C:=C–C';
        k:=|C|;
    until k=1;
    x:= le seul élément de C;
    return (x);
end–procedure;
```

Algorithm 4 Procédure FINAL-TESTS

```
Procedure FINAL–TESTS
    while U!=o do
        x:= un élément arbitraire de U;
        TEST({x});
        if positive D:=D+{x};
        if negative G:=G+{x};
        U:=U–{x};
    end–while;
end–procedure;
```

CHAPITRE 5

Bilan sur l'état de l'art

Durant notre étude de l'art nous avons pu traiter différents documents qui se rapproche plus ou moins de notre sujet mais qui avaient toutes un rapport avec ce dernier. Dans ce qui suit nous allons vous présenter les résultats de l'étude de l'art ainsi les motivations pour retenir ou non une méthode et par la suite l'approfondir et l'appliquer a notre étude :

5.1. La démarche AMDEC :

- une démarche qualitative et quantitative qui se traduit par l'analyse des modes de défaillance de leurs effets et de leurs criticités.
  - Elle repose sur une décomposition fonctionnelle de la machine, une identification des modes de défaillances afin de les classés ainsi que des méthodes de détection des de défaillances et des actions préventives ou correctives.
  - L'avantage de cette démarche est d'évaluer et de garantir : la fiabilité, la maintenabilité, la disponibilité, la sécurité, le rendement maximum au moindre cout. Par contre cette démarche ne permet pas de combiner plusieurs défaillances, en effet elle ne peut detecter qu'une défaillance à la fois.

En conclusion nous n'allons pas utiliser cette démarche pour la suite de notre étude car ce type d'étude requiert un niveau de détail beaucoup trop important pour une production de ce type (non liénaire, multi produits, grand nombre d'étape de fabrication).

5.2. L'étude de la gestion du risque

- Cette étude pour la détection et la réduction du risque rentre dans la prévention des phénomènes dangereux, elle a pour objectif d'apprécier le risque et de le réduire
  - Une démarche complète et entière, consiste en une action en amont qui gère et corrige le risque en aval.

Cette étude peut être intéressante car elle a pour finalité de réduire le risque, malgré ceci c'est une démarche très longue et qui exige une connaissance parfaite des machines, donc vu les contrainte de temps que nous avons-nous ne retiendrons pas cette études pour la suite de notre projet.

5.3. Plan de controle optimisé en se basant sur le risque

Cet article propose une méthode pour élaborer un plan de contrôle en 2 étapes : d'abord consiste à identifier les risques possibles et de faire une répartition de risque et puis à ajuster le plan construit en ajoutant ou en supprimant des contrôles selon la capacité.

Cette méthode a pour inconvénient de devoir définir au préalable tous les types de risque et la capacité des ressources de contrôle

5.4. An Approach for Operational Risk Evaluation and its Link to Control Plan

- Un article qui a pour idée d'évaluer les risques des machines, en effet a chaque traitement la fiabilité du processus diminue (c'est l'usure de la machine) jusqu'à la tache de qualité (maintenance)
  - Son avantage est de réduire fortement le risque du processus. D'autre part cette méthode a pour inconvénient la proportionnalité de la réduction du risque par rapport au nombre de plan de contrôle.

Cette méthode très gourmande en arrêt de la chaine de production lors des plans de contrôle ne sera pas développée par la suite car elle est industriellement infaisable.

5.5. Optimizing Return On Inspection Trough Defectivity Smart Sampling

- Évoque une méthode d'échantillonnage "intelligente" permettant d'identifier les produits qui ont besoins de contrôle sans la perte d'information
  - Son avantage est de réduire le nombre de contrôle non nécessaire par contre les résultats ne sont pas suffisant par rapport aux exigences industrielles

Cette méthode ne sera pas retenue même les résultats sont intéressants. En effet cette approche ne permet pas réduire les risques en défectivité mais seulement l'échantillonnage pour la mesure.

5.6. Computation of Wafer-At-Risk from Theory To Real Life Demonstration

- Introduit une méthode d'évaluation des risques et d'optimisation de l'utilisation des équipements de contrôle en introduisant les indicateurs W@R et W@R-Réduction
  - Son avantage est de réduire fortement le risque presque en temps réel par contre elle est complexe et sa faisabilité reste à prouver.

Malgré des promesses de grandes réduction de risque, nous ne pouvons pas nous engager sur sa fiabilité ni sur son utilisation dans notre étude, donc nous ne l'utiliserons pas.

5.7. Operational risk evaluation and control plan design

5.7.1. Support de Présentation.

- Cette approche d'évaluation des risques est basée sur l'évolution du « Potential Loss » (défectivité potentielle).
  - Cette évolution est supposée linéairement croissante avec le nombre d'exécution (traitement par la machine), sans contrôle, sur un horizon donné. Ce risque potentiel est réinitialisé, à une valeur seuil, après chaque tâche de contrôle ou de maintenance.
  - L'évolution du « Potential Loss » dépend de la gestion (planification) des tâches de qualité et de maintenance. De plus s'ajoute à cela une fonction d'évaluation de la valeur ajoutée des tâches de contrôles mises en place afin de déterminer la bonne planification de ces tâches pour un résultat optimal du risque.

Cependant ce support de présentation n'apporte pas plus information pour être retenu comme solution de notre problème.

5.8. La méthode D^3

- Cette méthode est quantitative basée sur l'analyse systématique d'ensemble de produit, en effet elle est décrite sous la forme d'un algorithme qui cherche les produits défectueux en optimisant le nombre de test.
  - Cette méthode est programmable et a priori correspond au sujet. Son inconvénient et qu'elle demande beaucoup de tests et donc systématique.

Nous avons décidé de garder cette méthode au début, mais une fois que nous avons commencé à faire la correspondance avec notre étude, nous nous sommes rendu compte que cette dernière demande beaucoup de tests dans le cas ou dans le premier échantillon il n'y a aucun éléments défectueux, de plus nous avons une taille de lot fixe et nous ne pouvons faire des tests simultanés sur plusieurs lots et finalement cette méthode imposera l'arrêt de la production plusieurs fois afin de prélever l'échantillons à tester, la taille de ce dernier peut devenir trés grande. Cette méthode s'applique dans un cadres ou la taille de l'échantillon n'est pas fixe et que le contrôle n'en est pas pertubé [1]. Malheuresement les contraintes fortes liés au contrôle des lots nous empêchent d'avoir des contrôles des tailles de lot de plus en plus importante.

5.9. Conclusion de l'état de l'art

Cette abscence de solution concrète dans cette industrie, nous a poussé à développer notre propre méthode de détection.

L'idée étant d'utiliser nos contraintes pour développer notre propre algorithme de calcul, nous détaillerons cet algorithme ainsi que ces résultats dans la partie suivante.

1. Dans le cadre d'une analyse épidémologique par exemple, tester un échantillon de sang d'une personne ou cent personnes ne fait pas de différence dans le cas de la recherche d'un virus dans une population

Troisième partie

Développement de la solution

CHAPITRE 6

Algorithme du marcheur

6.1. Introduction

Au terme de notre étude sur l'existant et l'état de l'art, il nous est apparu que le développement d'un outils adapté au problème spécifique nous semblait nécessaire. En effet le développement d'un modèle complet parait plus réalisable, à l'heure actuelle, que l'adaptation d'un outils inadapté à ce type de problématique.

Pour résumer notre problème nous allons détailler le modèle que nous nous proposons de développer par la suite. Le modèle du marcheur [1], ce modèle permet dans un premier temps de déterminer un indice de risque pour les pièces (*W@R* [2]). Dans un deuxième temps de se servir de cette indice de confiance pour déterminer les pièces les plus risqués, priorisé leur contrôle, et remonter plus rapidement aux lots impactées. Afin de représenter ces données nous coupleront ces données avec l'avancement du lot dans l'usine.

L'idée de départ est assez simple, on se propose de regarder la probabilité d'un marcheur dans une ville d'être agressé en se baladant.

6.2. Hypothèses

Afin de rendre le propos le plus clair possible nous allons poser toutes les hypothèses nécessaires au développement de l'algorithme.

- Un marcheur (un lot ou *Batch*) augmente sa « chance » d'être blessé (défectueux) au fur et à mesure qu'il se ballade dans la ville (Traverse beaucoup de postes de travail ou *Workstation*).
- Le marcheur qui traverse des quartiers dangereux (Machine de production, *Workstation*) a plus de chance d'être blessé dans un quartier dangereux.
- Propriété du quartier dynamique (Machine de production)
  - Le taux de criminalité augmente linéairement. Dans notre cas la sureté d'un quartier correspond à la fiabilité de la machine, de fait la fiabilité de la machine vaut donc $Fiabilité = Sureté_{Défaut} - Taux_{Criminalité}$ ou dans notre problème $F_{Machine} = F_{MachineDefaut} - D_{Machine}$
    - Avec $D_{Machine}$ est le déreglement de la machine
    - $F_{Machine}$ est la fiabilité de la machine ou sûreté du quartier
    - $F_{MachineDefaut}$ est le niveau de fiabilité après un contrôle de qualité, ou le niveau de sureté après le passage de la police.
  - La criminalité augmente en fonction du nombre de marcheurs qui traverse le quartier. Ce qui nous donne $D_{Machine} = N_{Produit} \times \lambda_{Dereglement}$
    - Où $\lambda_{Dereglement}$ est le coefficient de déregelment
  - La criminalité est remise à un niveau seuil quand la police passe. (Tache qualité)

1. Le nom « modèle du marcheur » est un clin d'œil à la « marche de l'ivrogne » qui est un modèle utilisé dans la thermodynamique et le mouvement brownien.
2. Wafer at Risk

  - La criminalité est remise à un seuil de confiace si la santé du marcheur est bonne en sortant du quartier.
    - On s'assure de la santé d'un marcheur en faisant passer un médecin (Mesure métrologique).
    - Si la santé du marcheur est bonne, cela veut dire que la criminalité a été sur-évaluée et il convient de la diminuer.
- Le contrôle de la santé d'un marcheur donne de l'information uniquement sur le dernier quartier vu par le marcheur.
- Il y a trois type de marcheur, un petit, moyen et un grand (technologie logique, mémoire et mixte)
  - Plus un marcheur est petit plus il aura de chance d'être agressé.
- On formalise le problème :

$$Indice_{Santé} = \prod_{i=1}^{N} \left(F_{Machine[i]} \times F_{Produit} \right) \tag{6.2.1}$$

  - $I_{Machine[i]}$ est la fiabilité de la machine i
  - $I_{Produit}$ est la fiabilité du produit.
  - $Indice_{Santé}$ est un indice de confiance plus cette valeur est basse plus le lot a de chance d'être deffectueux et un contrôle s'impose au plus vite.

6.3. L'algorithme

On définit différents niveaux de complexité à notre algorithme, pour chaque niveau de complexité nous détaillerons les différences. Il est à noter que tous ces calculs sont donné pour le calcul du *W@R* d'un seul marcheur (produit). On fera tourner cet algorithme dans une boucle pour comparer les différents produits entre eux. Ensuite on déterminera un classement des produits les plus risqué et cela permettra de savoir rapidement quel produit contrôler en priorité et suivant le resultat du contrôle impacter les autres pièces qui sont passés sur les machines similaires

6.3.1. Niveau 0 : Modèle simple. On suppose que les machines ont une fiabilité $F_{Machine} = 0,5 = 50\%$, il y a N étapes de fabrication, et $Indice_{santé}$ est la probabilité du produit d'être défectueux[3].

- pour i, de 1 à N
  - $F_{Produit} = F_{Produit} \times F$
- Fin Pour

6.3.2. Niveau 1 : Modèle des quartiers. On suppose maintenant que la fiabilité est différente sur chaque machine $F_{Machine[i]}$

- pour i, de 1 à N
  - $F_{Produit} = F_{Produit} \times F_{Machine[i]}$
- Fin Pour
- $W@R = 1 - F_{Produit}$

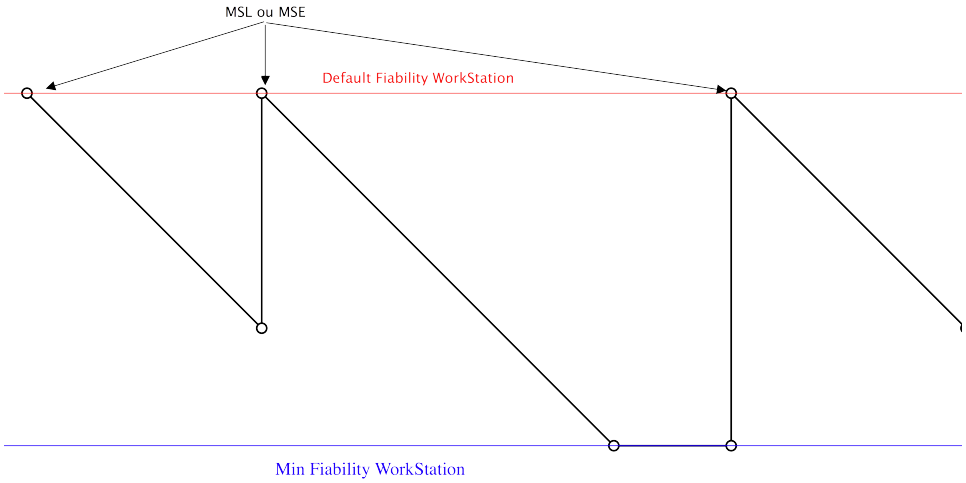
Exemple de loi de distribution de $F_{Machine[i]}$: Répartition Uniforme, Loi Normale, Loi de Poisson, Loi définie par morceaux (retour d'expérience).

6.3.3. Niveau 2 : Modèle des quartiers avec criminalité croissante. On suppose maintenant que la fiabilité est différente sur chaque machine et décroit linéairement :

$$F_{MachineProduits[i]} = F_{MachineDefaut} - (\lambda_{Machine} \times N_{Produit}) \tag{6.3.1}$$

3. Probabilité au marcheur d'être blessé.

FIGURE 6.3.1. Modèles d'évolution fiabilité (ou la criminalité) d'une machine (ou quartier)



Le graphique suivant représente l'évolution de la fiabilité d'une machine en fonction du nombre de produits qui passent sur la machine. Certains évenement font remonter le niveau de fiabilité de la machine à un seuil par défaut (trait rouge). De plus la fiabilité est minoré par un seuil minimal de fiabilité (trait bleu). Tant qu'aucun évenement de contrôle (MSL ou MSE) se produit sur la machine la fiabilité reste au même niveau.

- $N_{Produit}$ le nombre de produits passés sur la machine depuis le dernier contrôle.
- $F_{MachineDefaut}$ est la fiabilité maximal par défaut, suite à un contrôle de la machine.

Afin de définir un seuil minmal de Fiabilité de la machine on définira la Fiabilité de la manière suivante

$$F_{Machine} = \max\left(F_{MachineMin}, F_{MachineProduits[i]}\right) \tag{6.3.2}$$

- pour i, de 1à N
  - $F_{Produit} = F_{Produit} \times F_{Machine[i]}$
- Fin Pour
- $W@R = 1 - F_{Produit}$

6.3.4. Niveau 3 : Modèle du marcheur in-homogène. On suppose maintenant que le type de marcheur (la technologie), est plus susceptible d'être blessé suivant son type intrinsèque

$$F_{Produit} = \beta_{type} \times F_{Produit}$$

avec $\beta_{type} \in \left\{\beta_{petit}; \beta_{moyen}; \beta_{grand}\right\}$.

Comme dans la vie réelle un grand aura moins de chance d'être agressé qu'un petit.

Afin de faire le parallèle avec les technologie :

- β_{grand} correspond aux technologies de logique, qui sont les plus faciles à mettre en oeuvre

- β_{moyen} correspond aux technologies de mémoires, qui sont légèrement plus complexe.
- β_{petit} correspond aux technologies mixtes[4] qui sont les technologies les plus délicates à mettre en oeuvre et ayant de fortes chances de générer de la non qualité.

Le calcul s'effectue de la même manière ensuite, on introduira simplement la nouvelle fiabilité du produit ($F_{Produit}$). Il est à noter que les informations de complexité ne nous ayant pas été communiqué, cette partie de l'algorithme n'a pas été implémenté.

6.4. Specification du Code

Vu la complexité du projet nous avons décidé de développer notre code en Java, la structure des données et l'algorithme nous incite à utiliser ce type de langage. De plus comme il s'agit simplement de la validation d'algorithme, ce code n'a pas pour vocation d'être développé ultérieurement. En effet le Java est le langage enseigné dans notre formation et permettra une implication de tous.

6.4.1. Traitement des données.

6.4.1.1. *Analyse des données.* Afin de tester notre algorithme nous avons récupéré un fichier de données de *STMicroelectronics*, afin de bien comprendre ces données nous allons les analyser.

On distingue différents champs de données :

taskID (Champs Vide): Le premier champs vide est le numéro de la tâche, chaque tâche est numéroté de manière unique.

Date: Il s'agit de la date à laquelle l'évènement étudié a eu lieux. Il est à noter que notre fichier de données repose sur 15 jours de productions, le temps est donné en secondes.

- Le champs de la date est composé de 14 chiffres, exemple : 20101010162720
- Les 4 premier chiffres représentent l'année, ici 2010
- Les 2 chiffres suivants représentent le mois, ici 10 donc octobre
- Les 2 chiffres suivants représentent le jour, ici 10
- Les 6 chiffres suivants représentent l'heure, ici 162720 qui donne 16 heure 27 min et 20 secondes

Lot (batch): Il s'agit du numéro du lot, impacté par la tâche. Dans le cas où ce champ est vide, cela veut dire que la machine ne traite pas de produit mais qu'on effectue une tâche qualité sur la machine.

Oper (operation): Ce champs qualifie le type d'opération effectués sur le lot

Nbr (quantity): Représente le nombre de Wafers concernés par un événement de process, c'est-à-dire le nombre de plaques passées dans la machine.

Event: Permet de qualifier le type d'évènement. Dans notre exemple nous avons trois types d'évènements différents :

PRP: représente une étape de fabrication du produit

MSL: indique une mesure effectuée sur un lot

MSE: indique une mesure réalisée sur un équipement (tâche de qualité, ou « TQ »)

4. mémoire et logique

Tech (technology): Représente la technologie, nous disposons de cinq technologies différentes. Sur nos données il y a visiblement 5 types de technologie, cependant dans l'idée de rendre réalisable cette étude de faisabilité. Aucun détail n'est donné sur les technologies.

Eqpt (workstationID): Représente le numéro de l'équipement, chaque équipement est représenté par un identifiant unique.

6.4.1.2. *Classe des entités.* Afin d'utiliser les données au mieux, nous allons organiser les données sous forme de la structure suivante détaillé dans la suite. Ces classes font partie du package *entities*. Nous allons organiser les données par produit, machine et ordre. Dans le détail :

MyDate: classe qui organise une date, il est à noter que cette structure particulière repose sur la classe native Java qui s'appelle *Date* avec quelques petites adaptations.

Entier year:

Entier month:

Entier day:

Entier hour:

Entier minutes:

Entier secondes:

Traçabilité (Tracability): structure de la mémoire des événements, un objet de cette classe représente un événement élémentaire de l'usine de production.

Charactere Tâche: numéro de tâche

Date date: Donne la date de chaque tâche.

Charactere Type: Précise le type de tâche

Machine (WorkStation): Qualifie la ressource qui usine le procédé

Collection de traçabilité: « mémoire » de la ressource, c'est elle qui stocke les tâches se produisant sur la machine

Charactere Nom: Le nom de la machine

Lot (Batch): Classe qui qualifie un lot

Collection de traçabilité: « mémoire » du produit, c'est elle qui stocke les tâches se produisant sur la machine.

Charactere Nom: Le nom du produit

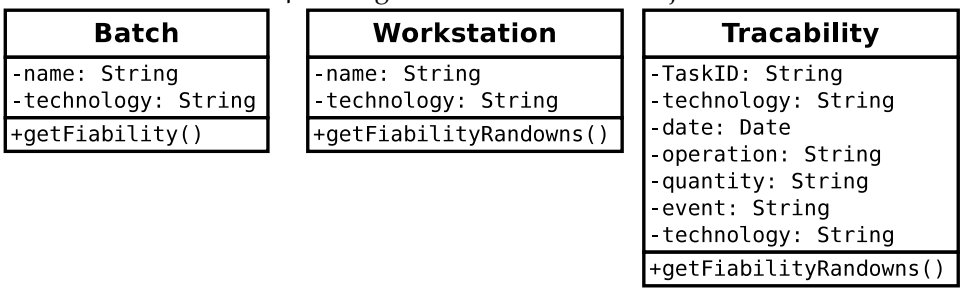
Entier Technologie: Permet de définir le type de technologie du lot

Cette structure nous permet d'organiser les données de manière logique, et permet de piocher facilement dans les données.

6.5. Implémentation du marcheur

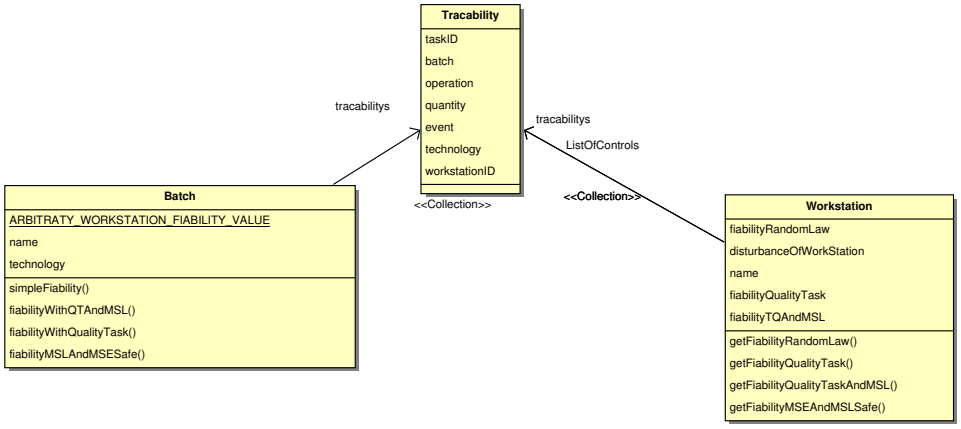
6.5.1. Marcheur. Contient l'algorithme qui lance les méthodes de calculs qui se trouvent sur les batch et workstations. En effet les méthodes de calcul de fiabilité se font sur l'objet batch et Workstation. En effet à la vue de notre architecture, les données nécessaires au calcul sont présentes sur ces objets il est naturel d'associer les méthodes à ces objets.

FIGURE 6.4.1. Diagramme de classe des objets entités



Ce diagramme de classe représente les objets sur lesquels nous travaillerons et utiliseront notre algorithme

FIGURE 6.5.1. Diagramme de classe des différentes entitées



Ce diagramme de classe représente l'ensemble des classes entities, les Batch et Workstation contiennent des collections de tracability, qui constituent la mémoire de chacun des éléments. De plus

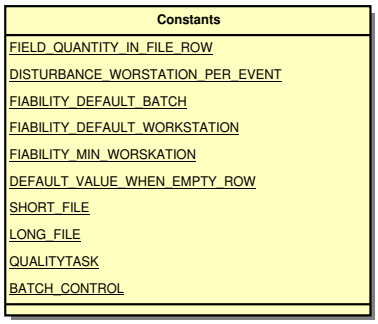
6.5.2. Lecture de fichier (Storage). Nous organisons notre lecture de fichier de façon à ranger les informations dans la bonne structure.

On définit une classe qui lit les fichiers et instancie les traçabilités associées à chaque ligne. De plus ce chargeur de fichier est spécificié pour comprendre le fichier donné par *STMICROELECTRONICS*. Le fichier donné est une extraction de la base de donnée *ST*, qui pour des raisons de confidentialité a été anonymisé.

6.5.3. Extracteur. Ces classes interprètent les traçabilité extraites par le load et crée les objets batch, workstation et insère les traçabilités associées. Il fait en quelque sorte le trie des données et permet d'associer chaque traçabilité à la machine et au lot qui lui correspond. De plus ces classes créent les objets et instance d'objet nécessaire à l'excution de l'algorithme.

6.5.4. Package entities. C'est dans ce package que se situe le coeur de l'algorithme. En effet les calculs de fiabilités sont des méthodes associées au objets *Batch* et *Workstation*. Ces objets possèdent comme atributs leur nom, leur technoloie et leur mémoire qui est représenté par une collection de *tracabilies* ce qui rend le calcul d'indice de santé plus facile à effectuer (détail du code source en section 11.2 et 11.1).

FIGURE 6.5.2. Diagramme de classe des constantes



Cette classe permet de paramétrer l'ensemble du projet en definissant les constantes utilisées dans le projet.

6.5.5. Classe Constants. Les connaissances métiers pouvant être à même d'évoluer nous avons effectué une classe constante qui regroupe toutes les données utilisés pour les calculs d'algorithme et peuvent être regler très facilement. On retrouve dans cette classe les attributs suivants :

FIELD_QUANTITY_IN_FILE_ROW: il s'agit là d'un paramètre pour lire et interpréter le fichier de départ

DEFAULT_VALUE_WHEN_EMPTY_ROW: Permet de donner le texte à écrire lors de la lecture d'un caractère vide

DISTURBANCE_WORSTATION_PER_EVENT: Il s'agit là du déreglement d'une machine après le passage d'un lot il s'agit de $\lambda_{Dereglement}$

FIABILITY_DEFAULT_BATCH: Il s'agit de l'indice de santé par défaut d'un lot $Indice_{Santé}$

FIABILITY_DEFAULT_WORKSTATION: Il s'agit là de la fiabilité d'une machine par défaut $F_{MachineDefaut}$

FIABILITY_MIN_WORSKATION: Il s'agit du seuil minimal de fiabilité d'une machine $F_{MachineMin}$

SHORT_FILE: Il s'agit là du chemin des données courtes (pour les tests rapides)

LONG_FILE: le chemin pour les données complètes

QUALITYTASK: La chaine de charactère qui indique une tâche de qualité sur une machine

BATCH_CONTROL: Chaine de caractère qui indique un contrôle de lot

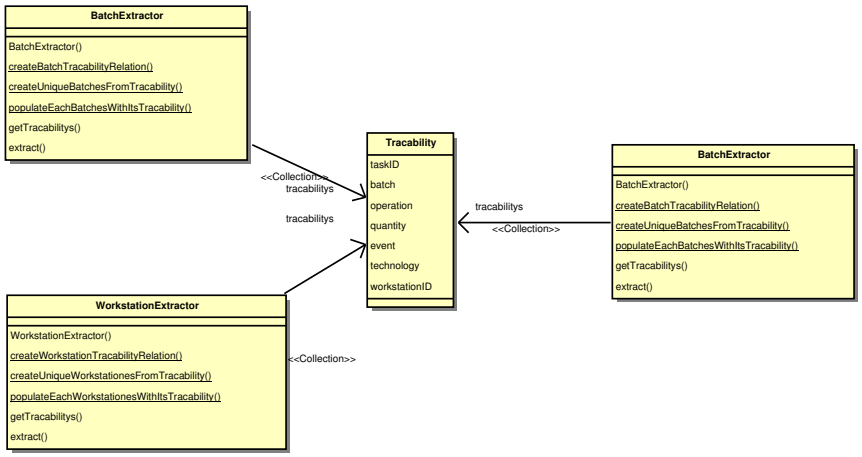
6.5.6. Package Extractors. C'est le package qui contient toutes les classes permettant de créer les *Batch* et les *Workstation* à partir de l'ensemble des traçabilités générées par les classes StoreData[5].

6.6. Résultats

Pour représenter les résultats de l'algorithme. Nous représentons dans un graphique en deux dimensions dans lequel on affiche l'indice de santé en fonction de l'avancement du produit (cf Fig 6.6.1)

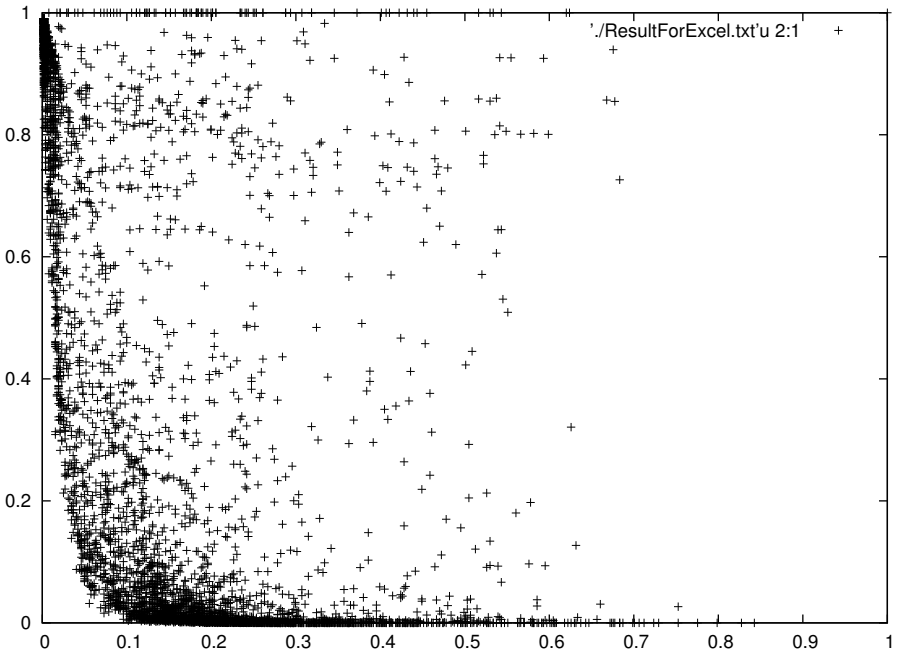
5. Ces classes ne sont pas détaillés, car elles ne font que lire et écrire dans des fichiers. Comme il n'y a rien de nouveau sous le soleil nous passons cette partie sous silence

FIGURE 6.5.3. Diagramme de classe des extractors



Cet ensemble de classe permet d'affecter les traçabilités au seins des bons lots et des bonnes machines

FIGURE 6.6.1. Représentation des résultats après calculs



Sur l'axe des abscisses nous avons l'avancement du produit et l'axe des ordonées représente l'indice de santé. Chaque croix représente un produit à l'instant t.

6.7. Inconvénient de cette méthode

6.7.1. Complexité. L'un des plus gros problèmes de ce type de solution est la complexité du code. En effet des boucles imbriquées entre elles fait que pour n

données nous avons $\log(3 \times n)$ calculs. Ce qui est génant dans ce type de problème car le nombre de donnée est assez considérable et le temps de calcul qui en découle est important.

6.7.2. Manque de réalisme avec la réalité. Le code actuel ne fait pas état de la réalité complète. Pour les besoins de l'études quelques hypothèses simplificatrices ont été faites :

- Les technologies ne sont pas détaillées dans les données traitées.
- Les contrôles MSE et MSL sont sur une technologie particulière et non pas sur la $F_{Produit}$ globale de la *Workstation* et le *Batch*. Une adaptation du modèle serait le bienvenue pour adapter de manière plus fine l'analyse avec le problème. Les *MSL* et *MSE* sont découpées en des sous catégories qui dépendent des technologies analysées.
- Dans notre projet toutes les machines ont le même poid pour le calcul de la fiabilité. Ce qui diffère avec la réalité, c'est que certaines machines ont un poid plus important que d'autres.

Ces propositions constituent des modifications à faire pour un développement ultérieur.

Quatrième partie

Gestion de projet

CHAPITRE 7

Organisation du projet

7.1. Organisation de l'équipe

Après notre première réunion, nous avons décidé de nous répartir les responsabilités en fonctions des centres d'intérêts de chacun. Nous avons répartis les responsabilités en fonction du sujet de PFE lui-même.

NOM Prénom	Rôle	Description du rôle
Nakara Rahma	Chef d'équipe	Son rôle est similaire à celui du **coach** dans une équipe, il fait en sorte de tirer le meilleur de chacun. Il est aussi celui qui tranche les problèmes épineux.
Garcia Fabien	Responsable de communication	Il interface l'équipe et les intervenants extérieurs.
Amrani-Mesbahi Jaafar	Responsable Gestion de Projet & Document	Il met en place la méthode **SCRUM** et gère le système d'information propre à l'équipe.
Ait Belkacem Abdelali	Responsable Développement	Il a pour mission de manager la partie développement du **PFE**
Nguyen Philippe	Responsable Réunion	Son rôle est de contrôler le bon déroulement de tous les réunions de l'équipe, l'animation et l'ordre du jour.

TABLE 7.1.1. Tableau récapitulatif des rôles

7.2. Gestion de projet

7.2.1. Choix de la méthode agile. Après une explication détaillée du sujet par notre client. Il est apparue que la gestion de projet collant au mieux au sujet du PFE est la méthode agile. En effet le projet pourra subir de nombreuses modifications en fonction de l'avancement.

Plutôt que de subir ces modifications avec une gestion de projet traditionnelle, nous préférons en prendre notre partie et l'intégrer dans notre gestion de projet.

En effet la méthode agile est une méthode de suivie de projet héritée des développements informatiques.

7.2.2. Caractéristiques de la méthode agile. Notre équipe fonctionne sur le modèle de la gestion de projet "Agile". La méthode agile retenue est la méthode *SCRUM*. Ces caractéristiques principales sont :

Simplicité : la gestion de projet a été simplifiée au maximum pour n'en garder que la quintessence.

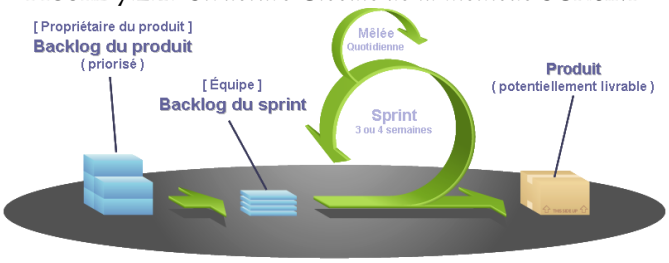
Flexibilité : cette flexibilité est obtenu par une forte implication du client. Le client peut faire des modifications sur ses besoins mais pas à n'importe quel moment. Uniquement entre deux itération. Cela permet de mieux répondre aux besoins du client.

Auto-gestion : basée sur la confiance mutuelle entre les membres de l'équipe ainsi qu'avec le client. Motivation de l'équipe.

Application : satisfaction du client en lui livrant quelque chose de fonctionnel.

Nous pouvons visualiser l'ensemble de cette méthode par le schéma (Réf : 7.2.1).

FIGURE 7.2.1. *Structure Globale de la méthode SCRUM*



Source : Wikipedia.

Voici les itérations de notre projet (Réf : 7.2.2).

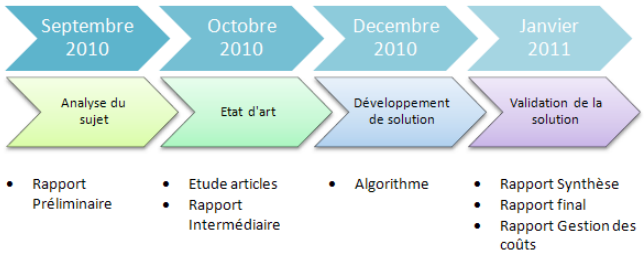


FIGURE 7.2.2. Planning du projet

Cinquième partie

Gestion des coûts

CHAPITRE 8

Gestion des coûts

Notre étude de la gestion des côuts repose sur deux hypothèses :
- La première est plutôt pessimiste : notre solution apporte -5% de non qualité.
- La deuxième plus optimiste : notre soltuion apporte jusqu'à -30% de non qualité.

TABLE 8.0.1. synthése

	1ère solution	2ème solution
VAN		
IP		
Coût		
Bénéfices		

CHAPITRE 9

Informations financières

CHAPITRE 10

Calculs des indicateurs de gestion des coûts

10.1. La valeur Actualisée Nette (VAN)

La VAN d'un investissement est la différence entre les flux actualisés à la date t_0 générés par un investissement et le montant de cet investissement :

$$VAN = \sum_{i=1}^{n} \frac{C_i}{(1+t)^i} - I$$

- I = Investissement de départ
- i = années
- t = taux d'actualisation
- C_k = flux générés par l'investissement à la date k

la VAN mesure la création de valeur engendrée par l'investissement.

10.2. L'indice de Profitabilité (IP)

L'Indice de Profitabilité est le quotient de la somme des flux positifs actualisé par le montant du capital investi :

$$IP = \frac{1}{I} \sum_{i} C_i (1+t)^{-i}$$

Qui s'écrit également sous la forme

$$IP = \frac{VAN}{I}$$

- i = année ;
- C_i = *cash-flow* (flux $\oplus$)
- t = le taux d'intérêt (taux d'actualisation)
- I = investissement initial

10.3. Le Taux de Rentabilité Interne (TRI)

Le Taux de Rentabilité Interne est le taux pour lequel la VAN est nulle ($VAN = 0$).

C'est-à-dire qu'il y a équivalence entre le capital investi et l'ensemble des *cash-flow* (flux $\oplus$) actualisés.

Il faut donc chercher la valeur de t dans l'équation.

$$VAN(t) = -I + C_1(1+t)^{-1} + C_2(1+t)^{-2} + \cdots = 0$$

10.4. Delai de récupération du capital investi

Le délais de récupération d'un capital investi est le temps au bout duquel le montant cumulé des flux financier positifs ($\oplus$) actualisés est égal au capital investi.

C'est donc le temps nécessaire pour que les flux nets de trésorerie générés par l'investissement remboursent la mise de fond initiale en tenant compte du temps.

10.5. Synthèses des différents indicateurs

Sixième partie

Conclusion de l'étude

Conclusion

Septième partie

Annexes

CHAPITRE 11

Code Source

11.1. Classe Batch

```
/*
 *  Copyright (C) 2010 Team-W@R (team-war@prunetwork.fr)
 *
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
package fr.prunetwork.teamwar.entities;

import fr.prunetwork.teamwar.Constants;
import fr.prunetwork.teamwar.utilities.MyDate;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Iterator;

/**
 *
 * @author jpierre03+teamwar@prunetwork.fr
 * @author GARCIA Fabien
 * @author NAIT BELKACEM Abdelali
 */
public class Batch {

    private static final double ARBITRATY_WORKSTATION_FIABILITY_VALUE = 0.5;
    private String name;
    private String technology;
    private Collection<Tracability> tracabilitys =
            new ArrayList<Tracability>();

    public Batch(String name, String technology) {
        this.name = name;
```

```
        this.technology = technology;
    }

    /**
     * @return the name
     */
    public String getName() {
        return name;
    }

    /**
     * @return the technology
     */
    public String getTechnology() {
        return technology;
    }

    public void addTracability(Tracability tracability) {
        tracabilitys.add(tracability);
    }

    /**
     * @return the tracabilitys
     */
    public Collection<Tracability> getTracabilitys() {
        return Collections.unmodifiableCollection(tracabilitys);
    }

    public String description() {
        StringBuilder sb = new StringBuilder();
        sb.append(getName());
        sb.append("\t");
        sb.append(getTechnology());
        sb.append("\t");
        sb.append("tracabilitysCount: ");
        sb.append(getTracabilitys().size());
        return sb.toString();
    }

    public double simpleFiability() {
        double fiability = Constants.FIABILITY_DEFAULT_BATCH;
        fiability *= Math.pow(ARBITRATY_WORKSTATION_FIABILITY_VALUE,
                (double) getTracabilitys().size());
        return fiability;
    }

    public double fiabilityWithQTAndMSL(MyDate currentDate) {
        double fiability = Constants.FIABILITY_DEFAULT_BATCH;
        for (Iterator<Tracability> it = tracabilitys.iterator(); it.hasNext();) {
            Tracability tracability = it.next();
            if (tracability.getDate().before(currentDate)) {
                if (tracability.getEvent().equalsIgnoreCase(Constants.BATCH_CONTROL)) {
```

```
                fiability = Constants.FIABILITY_DEFAULT_BATCH;
            } else {
                fiability *= StoreEntities.getWorkstation(
                        tracability.getWorkstationID()).
                        getFiabilityQualityTask(tracability.getDate());
            }
        }
    }
    return fiability;
}

public double fiabilityWithQualityTask(MyDate currentDate) {
    double fiability = Constants.FIABILITY_DEFAULT_BATCH;
    for (Iterator<Tracability> it = tracabilitys.iterator(); it.hasNext();) {
        Tracability tracability = it.next();
        if (tracability.getDate().before(currentDate)) {
            fiability *= StoreEntities.getWorkstation(
                    tracability.getWorkstationID()).
                    getFiabilityQualityTask(tracability.getDate());
        }
    }
    return fiability;
}

public double fiabilityWithQTAndMSLandWorkstation(MyDate currentDate) {
    Tracability previousTracibility = null;
    double fiability = Constants.FIABILITY_DEFAULT_BATCH;

    for (Iterator<Tracability> it = tracabilitys.iterator(); it.hasNext();) {
        Tracability tracability = it.next();
        if (tracability.getDate().before(currentDate)) {
            if ((tracability.getEvent().equalsIgnoreCase(Constants.BATCH_CONTROL))
                    && (!(previousTracibility == null))) {
                fiability = Constants.FIABILITY_DEFAULT_BATCH;
                StoreEntities.getWorkstation(
                        previousTracibility.getWorkstationID()).setDateLastControle(pre
            } else {
                fiability *= StoreEntities.getWorkstation(
                        tracability.getWorkstationID()).
                        getFiabilityQualityTaskAndMSL(tracability.getDate());
            }
        } else {
        }
        previousTracibility = tracability;
    }
    return fiability;
}

public double fiabilityMSLAndMSESafe(MyDate currentDate) {
    double fiability = Constants.FIABILITY_DEFAULT_BATCH;
    for (Iterator<Tracability> it = tracabilitys.iterator(); it.hasNext();) {
        Tracability tracability = it.next();
```

```
            if (tracability.getDate().before(currentDate)) {
                if (tracability.getEvent().equalsIgnoreCase(
                        Constants.BATCH_CONTROL)) {
                    fiability = Constants.FIABILITY_DEFAULT_BATCH;
                } else {
                    fiability *= StoreEntities.getWorkstation(
                            tracability.getWorkstationID()).
                            getFiabilityMSEAndMSLSafe(tracability.getDate());
                }
            }
        }
        return fiability;
    }
}
```

11.2. Classe Workstation

```
/*
 *  Copyright (C) 2010 Team-W@R (team-war@prunetwork.fr)
 *
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation\t either version 3 of the License\t or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful\t
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program.  If not\t see <http://www.gnu.org/licenses/>.
 */
package fr.prunetwork.teamwar.entities;

import com.sun.corba.se.impl.orbutil.closure.Constant;
import fr.prunetwork.teamwar.Constants;
import fr.prunetwork.teamwar.utilities.MyDate;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Date;
import java.util.Iterator;
import java.util.Vector;

/**
 *
 * @author jpierre03+teamwar@prunetwork.fr
 * @author GARCIA Fabien
 * @author NAIT BELKACEM Abdelali
 */
public class Workstation {
```

```
private double fiabilityRandomLaw = Math.random();
private double disturbanceOfWorkStation =
        Constants.DISTURBANCE_WORSTATION_PER_EVENT;
private String name;
private Collection<Tracability> tracabilitys =
        new ArrayList<Tracability>();
private Collection<Tracability> ListOfControls =
        new ArrayList<Tracability>();
private double fiabilityQualityTask =
        Constants.FIABILITY_DEFAULT_WORKSTATION;
double fiabilityTQAndMSL = Constants.FIABILITY_DEFAULT_WORKSTATION;

public Collection<Tracability> getListOfControls() {
    return ListOfControls;
}

public void setListOfMSL(Collection<Tracability> setOfMSL) {
    this.ListOfControls = setOfMSL;
}
private MyDate dateLastControle = new MyDate();

public Workstation(String name) {
    this.name = name;
}

/**
 * @return the disturbanceOfWorkStation
 */
public double getDisturbanceOfWorkStation() {
    return disturbanceOfWorkStation;
}

/**
 * @param disturbanceOfWorkStation the disturbanceOfWorkStation to set
 */
public void setDisturbanceOfWorkStation(double disturbanceOfWorkStation) {
    this.disturbanceOfWorkStation = disturbanceOfWorkStation;
}

public void addTracability(Tracability tracability) {
    tracabilitys.add(tracability);
}

/**
 * @return the tracabilitys
 */
public Collection<Tracability> getTracabilitys() {
    return Collections.unmodifiableCollection(tracabilitys);
}

/**
 * @return the name
```

```
     */
    public String getName() {
        return name;
    }

    public String description() {
        StringBuilder sb = new StringBuilder();
        sb.append(getName());
        sb.append("\t");
        sb.append("tracabilitys count: ");
        sb.append(getTracabilitys().size());
        return sb.toString();
    }

    public double getFiabilityRandomLaw() {
        while (Double.compare(fiabilityRandomLaw, 0.0) == 0) {
            fiabilityRandomLaw = Math.random();
        }
        return fiabilityRandomLaw;

    }

    /**
     * Fonction which calculate the fiability thanks to the number of events
     * since the last quality task
     * The operation done is this one :
     *  fiabilityQualityTask = Math.max(fiabilityQualityTask,
     *  Constants.FIABILITY_MIN_WORSKATION);
     * @param currentDate
     * @return
     */
    public double getFiabilityQualityTask(MyDate currentDate) {

        fiabilityQualityTask = Constants.FIABILITY_DEFAULT_WORKSTATION
                - getNumberOfEventSinceLastQT(currentDate)
                * Constants.DISTURBANCE_WORSTATION_PER_EVENT;
        fiabilityQualityTask = fiabilityQualityTask
                < Constants.FIABILITY_MIN_WORSKATION
                ? Constants.FIABILITY_MIN_WORSKATION
                : fiabilityQualityTask;
        return fiabilityQualityTask;
    }

    /**
     *
     * @param currentDate
     * @return
     */
    public double getFiabilityQualityTaskAndMSL(MyDate currentDate) {

        fiabilityTQAndMSL = fiabilityTQAndMSL
                - getNumberOfEventSinceLastControl(currentDate)
```

```
                * Constants.DISTURBANCE_WORSTATION_PER_EVENT;
        fiabilityTQAndMSL = fiabilityTQAndMSL
                < Constants.FIABILITY_MIN_WORSKATION
                ? Constants.FIABILITY_MIN_WORSKATION
                : fiabilityTQAndMSL;

        return fiabilityTQAndMSL;
    }

    private Double getNumberOfEventSinceLastQT(MyDate currentDate) {

        Date lastQualityTask;
        Double numberOfEvents = new Double(0);

        for (Iterator<Tracability> it = getTracabilitys().iterator();
                it.hasNext();) {
            Tracability tracability = it.next();
            if (tracability.getDate().before(currentDate)) {
                if (tracability.getEvent().equalsIgnoreCase(
                        Constants.QUALITYTASK)) {
                    lastQualityTask = tracability.getDate();
                    numberOfEvents = new Double(0);
                } else {
                    numberOfEvents++;
                }
            }
        }
        return numberOfEvents;
    }

    public Double getFiabilityMSEAndMSLSafe(MyDate currentDate) {
        getDateLastControle(currentDate);
        Double nbEvents = getNumberOfEventSinceLastControl(currentDate);
        fiabilityTQAndMSL = Math.max(Constants.FIABILITY_MIN_WORSKATION,
                Constants.FIABILITY_DEFAULT_WORKSTATION - nbEvents
                * Constants.DISTURBANCE_WORSTATION_PER_EVENT);
        return fiabilityTQAndMSL;
    }

    private void getDateLastControle(MyDate currentDate) {

        MyDate lastQualityTask = null;
        Double numberOfEvents = new Double(0);
        Iterator<Tracability> it = getListOfControls().iterator();

        while ((it.hasNext()) && (lastQualityTask == null)) {
            Tracability tracabilityBefor = it.next();
            if (it.hasNext()) {
                Tracability tracabilityAfter = it.next();
                if (tracabilityBefor.getDate().before(currentDate)
                        && tracabilityAfter.getDate().after(currentDate)) {
                    lastQualityTask = tracabilityBefor.getDate();
```

```
                }
            } else {
                if (tracabilityBefor.getDate().before(currentDate)) {
                    lastQualityTask = tracabilityBefor.getDate();
                }
            }

        }
        if (lastQualityTask != null) {
            dateLastControle = lastQualityTask;
        }
    }

    /**
     * @param currentDate
     * @return
     */
    private Double getNumberOfEventSinceLastControl(MyDate currentDate) {
        Double numberOfEvents = new Double(0);
        for (Iterator<Tracability> it = getTracabilitys().iterator();
                it.hasNext();) {
            Tracability tracability = it.next();
            if ((tracability.getDate().before(currentDate))
                    && tracability.getDate().after(dateLastControle)) {
                numberOfEvents++;
            }
        }

        return numberOfEvents;
    }

    /**
     * @param lastControleMSE the lastControleMSE to set
     */
    public void setDateLastControle(MyDate lastControleMSE) {
        this.dateLastControle = lastControleMSE;
    }

    /**
     * @return the lastControleMSE
     */
    public MyDate getDateLastControle() {
        return dateLastControle;
    }
}
```

11.3. Classe Marcheur

```
/*
 *  Copyright (C) 2010 Team-W@R (team-war@prunetwork.fr)
 *
 *  This program is free software: you can redistribute it and/or modify
```

```
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
package fr.prunetwork.teamwar;

import fr.prunetwork.teamwar.entities.Batch;
import fr.prunetwork.teamwar.entities.StoreEntities;
import fr.prunetwork.teamwar.entities.Tracability;
import fr.prunetwork.teamwar.entities.Workstation;
import fr.prunetwork.teamwar.extractor.BatchAndWorkstationLinkExtractor;
import fr.prunetwork.teamwar.extractor.BatchExtractor;
import fr.prunetwork.teamwar.extractor.WorkstationExtractor;
import fr.prunetwork.teamwar.storage.reader.ExtractDataFromFile;
import fr.prunetwork.teamwar.storage.writer.StoreDataToFile;
import fr.prunetwork.teamwar.utilities.MyDate;
import java.io.IOException;
import java.text.DecimalFormat;
import java.util.Collection;
import java.util.Iterator;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author jpierre03+teamwar@prunetwork.fr
 * @author GARCIA Fabien
 * @author NAIT BELKACEM Abdelali
 */
public class Marcheur {

    private String fichier;
    private StoreDataToFile sdtf;
    private Collection<Tracability> tracabilitys;
    private WorkstationExtractor we;
    private BatchExtractor be;
    private Collection<Workstation> workstations;
    private Collection<Batch> batchs;

    public Marcheur() {
        fichier = Constants.SHORT_FILE;
        //       fichier = Constants.LONG_FILE;
        sdtf = new StoreDataToFile("./ResultForExcel.txt");
```

```
        tracabilitys = ExtractDataFromFile.createTracabilityCollection(fichier);

        we = new WorkstationExtractor(tracabilitys);
        workstations = we.extract();

        be = new BatchExtractor(tracabilitys);
        batchs = be.extract();

        BatchAndWorkstationLinkExtractor batchAndWorkstationLinkExtractor =
                new BatchAndWorkstationLinkExtractor();
        batchAndWorkstationLinkExtractor.link();
        Double maxNumberOfSteps = StoreEntities.getNumberMaxOfSteps();
        MyDate lastDate = StoreEntities.getLastDate();
        MyDate firstDate = StoreEntities.getFirstDate();
        System.out.println(firstDate.toString());
        System.out.println(lastDate.toString());
        int nunmberOfProcessDone = 0;
        for (Iterator<Batch> it = batchs.iterator(); it.hasNext();) {
            Batch batch = it.next();

            DecimalFormat decimalFormat = new DecimalFormat();
            decimalFormat.setMaximumFractionDigits(4); //arrondi Ã  2 chiffres apres la virg
            decimalFormat.setMinimumFractionDigits(4);
            Double percentOfavencement =
                    new Double(batch.getTracabilitys().size() / maxNumberOfSteps);
            sdtf.add(decimalFormat.format(batch.fiabilityMSLAndMSESafe(lastDate))
                    + ";" + decimalFormat.format(percentOfavencement) + "\n");
            System.out.println("Nombre de batch traitÃ© :"
                    + nunmberOfProcessDone++ + " / " + batchs.size());
        }

        try {
            sdtf.commit();
        } catch (IOException ex) {
            Logger.getLogger(Marcheur.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public static void main(String[] args) {
        new Marcheur();
    }
}
```

Bibliographie

[1] D.Z. Du and F.K Hwang. *Discrete Applied Math*, chapter Competitive Group Testing. AT&T Bell Laboratories Technical Memorandum, December 1990. 4.3