# SCRUM : Une extension au langage de modèles pour le développement logiciel hyper-productif


Mike Beedle beedlem@fti-consulting.com

Martine Devos mdevos@argo.be

Yonat Sharon yonat@usa.net

Ken Schwaber virman@aol.com

Jeff Sutherland jeff.sutherland@idx.com

**En résumé**

Les modèles de la méthode de développement SCRUM se présentent comme une extension aux langages organisationnels existants. Ces dernières années, la méthode SCRUM s'est fait rapidement connaître comme étant un outil efficace en vue d'obtenir un développement logiciel hyper-productif. Cependant, lorsqu'ils sont combinés avec d'autres modèles organisationnels, les modèles SCRUM conduisent à des organisations de développement logiciels fortement adaptables tout en restant structurés. De plus, décomposer SCRUM sous la forme de modèles permet d'adopter uniquement les parties applicables à une situation spécifique.

## 1. Introduction

NOTE : Tout au long de cet article, nous prenons comme postulat que le lecteur est familier des autres modèles organisationnels existants [OrgPatt], [Coplien95]. Tous les noms des modèles seront écrits en caractères gras-italiques de la manière suivante : ***DéveloppeurContrôleLeProcessus***, ou ***ArchitecteContrôleLeProduit***.

Est-ce qu'un processus répétitif et défini existe réellement dans le développement logiciel ? Certains pensent que ce n'est pas seulement possible mais également nécessaire, comme par exemple ceux en faveur de l'approche CMM (Capability Maturity Model) [1]. Le CMM définit cinq étapes du processus de maturité: initial, répétable, défini, géré et optimisé, et enjoint aux utilisateurs de définir les processus de 18 KPA (key process areas - zones processus clés - NdT).

Cependant, un certain nombre d'entre nous, qui travaillent sur le terrain, ont réalisé au fur et à mesure que l'approche d'un processus « répétable ou défini » repose sur des hypothèses erronées, telles que :

1) Un problème répétitif/défini. Un processus répétable et défini part du postulat qu'il y a une étape pour lister les exigences d'une application, mais dans la plupart des cas, il n'est tout simplement pas possible de les définir, car elles sont soit pas bien définies soit changeantes.

2) Une solution répétable et définie. Un processus répétable et défini part du postulat qu'une architecture puisse être complètement spécifiée, mais en réalité une architecture évolue, d'une part en raison d'exigences manquantes ou changeantes (comme décrit plus haut), et d'autre part en raison de la nature même du processus créatif utilisé lors de la création de celle-ci.

3) Des développeurs clonables/définis. Les compétences d'un développeur informatique varient grandement d'un individu à un autre, par conséquent un processus qui fonctionne pour un développeur peut ne pas fonctionner pour un autre.

4) Un environnement organisationnel répétable/défini. La pression des délais, des priorités (autrement dit la qualité vs. le prix), le comportement du client, etc. ; ne sont jamais répétables ou définis.

Le principal problème avec ces hypothèses est de supposer que n'importe lequel des éléments de cette liste ait un comportement non-chaotique. Même la plus petite des inconnues peut avoir de grosses répercussions sur le résultat.

Parce que dans la vraie vie l'élimination de ces incertitudes s'avère impossible, un certain nombre d'entre nous ont cherché des réponses au-delà de l'approche répétable/définie du développement logiciel pour une « approche davantage adaptative ».

SCRUM prend comme postulat de départ que l'hypothèse de l'existence d'un certain chaos dans les éléments de cette liste est fausse, mais fournit des pratiques pour résoudre ces problèmes avec des techniques ayant leurs racines dans la gestion de la complexité, autrement dit l'auto-organisation, la gestion des processus empiriques, la création de savoir, etc.

En ce sens, SCRUM n'est pas seulement une méthode de développement « itérative et incrémentale » mais aussi une méthode de développement informatique « adaptative ».

## 2. Comment fonctionne SCRUM ?

Le but de SCRUM est de livrer un logiciel de la meilleure qualité possible au cours d'une série (3-8) d'intervalles de temps dénommés ***Sprints,*** qui durent généralement environ un mois.

Chaque étape du cycle de développement (Exigences, Analyse, Conception, Évolution et Livraison) est maintenant intégré dans un ***Sprint*** ou dans une série de ***Sprints***. Les étapes du développement informatique traditionnel sont conservées par commodité pour le suivi des jalons. Donc, par exemple, l'étape des exigences peut utiliser un ***Sprint***, y compris la livraison d'un prototype. Les étapes d'Analyse et de Conception peuvent prendre chacun un Sprint. Alors que l'étape d'Évolution peut prendre quelque chose comme 3 à 5 Sprints.

Au contraire d'un processus défini et répétable, il n'existe pas dans SCRUM de processus prédéfini au sein d'un ***Sprint***. À la place, les ***Réunions Scrum*** pilotent la réalisation des activités allouées.

Chaque Sprint prend en charge un certain nombre de tâches à travers ce qui s'appelle un ***Backlog***. En règle générale, aucune tâche extérieure n'est ajoutée au ***Backlog*** dans un ***Sprint***. Des tâches internes résultant du contenu du ***Backlog*** pré-alloué peuvent y être ajoutées. L'objectif d'un ***Sprint*** est de réaliser un logiciel de la meilleure qualité possible, mais en pratique généralement la quantité de logiciel livré est moindre (modèle PireEstMieux). Le résultat final est des ***BasesStablesNommées*** non-parfaites livrées à chaque ***Sprint***.

**Figure 1. Une équipe de rugby fait aussi des Réunions Scrum (non représentées ici).**

Lors d'un ***Sprint***, les ***Réunions Scrum*** ont lieu chaque jour afin de vérifier

1) quels sont les items terminés depuis la dernière ***Réunion Scrum***.
2) quels sont les problèmes ou les ***Blocages*** qui ont été trouvés et qui sont à résoudre. (Le ***ScrumMaster*** est un rôle de leader, au sein de l'équipe, responsable pour résoudre les ***Blocages***.)
3) quelles nouvelles tâches font sens pour l'équipe à réaliser jusqu'à la prochaine ***Réunion Scrum***.

Les ***Réunions Scrum*** permettent à l'équipe de développement de « socialiser les connaissances des membres de l'équipe », et de connaître une transcendance culturelle profonde.

En retour, cette « socialisation du savoir » conduit à une structure d'équipe auto-organisée, dans laquelle le processus est inventé de manière adaptative quotidiennement.

À la fin de chaque ***Sprint***, il y a une ***Démo*** qui :

1) montre au client ce qui se passe, (***ImpliquerLeClient***)
2) donne au développeur un sentiment d'accomplissement (***RécompenserLeSuccès***)
3) intègre et teste une portion raisonnable du logiciel qui a été développé (***ImpliquerAQ***)
4) s'assure qu'il y a de **réels progrès** - une diminution du backlog et pas uniquement davantage de papiers ou d'heures passées (***BasesStablesNommées***).

Après avoir rassemblé et re-priorisé d'une part ce qui reste et d'autre part les nouvelles tâches, un nouveau ***Backlog*** est constitué et un nouveau ***Sprint*** démarre. Potentiellement, beaucoup d'autres modèles organisationnels peuvent être utilisés avec les modèles de SCRUM. Alors que les modèles de Coplien (organisationnels et de processus) semblent un choix évident en raison de leur portée, d'autres modèles organisationnels issus d'autres sources peuvent s'avérer également un très bon choix [OrgPatt], [Coplien].

Les phrases « C'est le système qui l'exige » ou « Le système ne le permet pas » sont devenues des justifications communément acceptées (et souvent inévitables) du comportement humain. Ce que nous appréhendons c'est que les méthodes d'aujourd'hui ne nous permettent pas de réaliser des logiciels *assez souples*, car les méthodes et les paradigmes de conception actuels semblent inhiber l'adaptabilité. C'est pourquoi la majorité des professionnels de l'informatique ont tendance à se spécialiser sur ce qu'ils peuvent *spécifier à l'avance*, travaillant avec la conviction implicite qu'il existe une solution optimale qui peut être planifiée à l'avance.

Une fois qu'une technologie a été adoptée par une organisation, elle devient un cadre contraignant qui structure une partie de l'espace d'action de l'utilisateur. Donc nous construisons du logiciel de la même manière que nous construisons du matériel comme si c'était difficile de changer quoi que ce soit et comme si cela devait être difficile de changer quoi que ce soit.

En contraste SCRUM nous permet de construire un logiciel de manière plus *souple*, il n'est donc pas nécessaire d'essayer d'écrire des exigences en amont. L'utilisateur ne sait pas ce qui est possible et aura tendance à demander une solution papier pré-technologique qu'il perçoit comme étant possible. Pas même un développeur informatique est en mesure de savoir de A à Z ce qui peut être construit avant que ça le soit. Et par conséquent l'utilisateur n'a lui aussi aucune idée de ce qui est possible avant qu'il l'ait senti ou qu'il l'ait touché [Blu96].

Nous avons ainsi besoin d'une autre approche, plus souple, pour faire du logiciel. Donc il vaut mieux réaliser qu'*il est impossible* d'avoir l'ensemble des exigences détaillées en amont ou de geler le contexte et l'environnement. Les exigences sont écrites dans un certain contexte. Nos systèmes transforment ce contexte. De nouveaux problèmes arrivent dans le système et le nouveau contexte.

Ce problème ne se résout pas à l'aide de nouvelles méthodes d'identification d'exigences utilisateurs. Il est nécessaire, à la place, d'avoir un processus plus complexe pour générer de nouvelles manières de faire. Scrum et sa manière de travailler empirique est l'une de ces alternatives possibles.

## 3. Le langage de modèles SCRUM

Le diagramme suivant montre les relations entre les différents modèles SCRUM et les autres modèles organisationnels

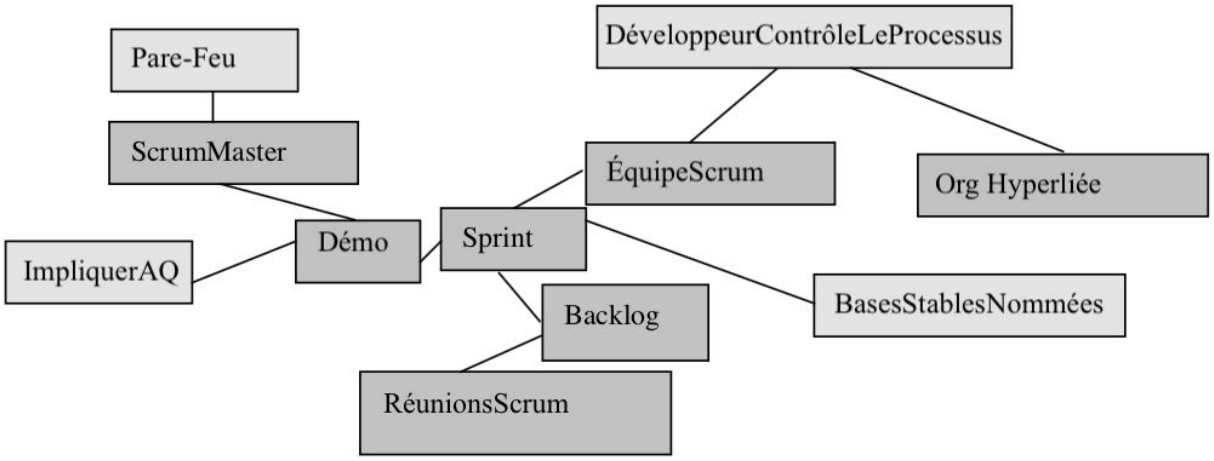


**Figure 2. Représentation du maillage du langage de modèles de SCRUM (Coplien = gris clair, SCRUM=gris foncé)**

## 4. Les modèles de Scrum

# Réunion scrum

***Contexte***

***(DE : Backlog)***

Vous, si vous êtes un développeur informatique ou un coach s'occupant d'une équipe de développement informatique dans laquelle il existe un fort degré de découverte, de créativité ou de test. Comme par exemple, la première livraison où la problématique doit être spécifiée, ou qu'un modèle objet doit être créé, ou qu'une évolution ou une nouvelle technologie est utilisée.

Des activités comme la recherche scientifique, l'innovation, l'invention, l'architecture, l'ingénierie et une myriade d'autres situations métiers peuvent aussi montrer ce type de comportement.

Vous pouvez aussi être un « travailleur intellectuel », un ingénieur, un écrivain, un chercheur, ou un coach ou un manager qui supervise les activités d'une équipes dans ces milieux.

(Variables inappropriées : estimation, planification, suivi, confort humain)

NOTE : les variables inappropriées sont des variables qui peuvent être ajustées contextuellement pour s'adapter à la solution et résoudre le problème.

## Problème

Quelle est la meilleure manière de contrôler un processus aussi empirique et imprévisible que le développement logiciel, la recherche scientifique, les projets artistiques ou des conceptions innovantes dans lesquelles il est difficile de définir les artefacts à produire et les processus permettant de les réaliser ?

## Forces (Analyse des variables inappropriées dans le contexte)

[NOTE : certains utilisent le terme d'anti-modèles pour parler d'exemples inappropriés. J'utilise ici certains noms à-la-Dilbert pour montrer leur nature déviante.]

**Estimation**

(+) Une estimation précise pour des activités incluant découverte, créativité ou test est quelque chose de difficile parce que ce type d'estimation implique généralement de grandes variations, et que de légères différences dans les circonstances peuvent donner des différences significatives.

Ces incertitudes sont au moins de quatre types différentes :

a) Les exigences ne sont pas bien comprises

b) Les dépendances au niveau de l'architecture ne sont pas faciles à comprendre et changent constamment

c) Des défis techniques imprévus avec la technologie peuvent se produire, etc. Même si les défis sont connus à l'avance, leurs solutions et les efforts associés ne sont pas connus.

d) Des anomalies difficiles à résoudre dans le logiciel peuvent arriver, et par conséquent, il est naturel de voir des estimations au niveau du projet ayant plusieurs ordres de magnitude. Vous ne pouvez pas « prévoir les anomalies ». Vous pouvez seulement prévoir la gestion des anomalies et les schémas de prévention, et pour prévoir cela, vous devez tout d'abord accepter la possibilité d'anomalies inattendues.

Exemple inappropriée : ***VousAvezTiréLeMauvaisNuméro*** – dans des projets ayant des exigences nouvelles ou changeantes, une nouvelle architecture ou des technologies changeantes, et des anomalies difficiles à éliminer ; il est normal de voir des estimations au niveau du projet à des années-lumières les unes des autres.

(-) Les estimations sont importantes. Nous devons être capable de déterminer quelles vont être les futures tâches dans un certain horizon de temps et organiser les ressources à l'avance.

Exemple inapproprié : ***PréparezMoiPourÉchouer*** – les projets sans estimation sont difficiles à gérer.

**Planification**

(+) Les tâches de planification et de re-priorisation prennent du temps. Utiliser du temps pour des réunions de planification n'a que comme seul résultat une baisse de la productivité. Si le système est en plus chaotique, quelle que soit la quantité de planification que vous ferez, il sera impossible de réduire les incertitudes au niveau des résultats de la planification..

Exemple inapproprié : ***ParalysieParPlanification*** – les projets qui font perdre du temps à tout le monde en planifiant tout dans les moindres détails mais qui ne sont pas capables de les concrétiser.

(+) Un plan qui est trop détaillé devient énorme et difficile à suivre. Il peut être plus facile d'appliquer le bon sens ou d'appeler le client. Et, plus la planification est importante, plus elle contiendra d'erreurs (de manière alternative, le coût de vérification des corrections augmentera).

(-) Ne pas avoir de planification du tout aboutira à une augmentation de l'incertitude au sein de l'équipe, ce qui pourrait finir par diminuer le moral de l'équipe.

Exemple inapproprié : ***PerteDeVision*** – les projets sans planification ont tendance à perdre de vue leurs objectifs. Sans pression d'un planning quel qu'il soit, personne ne fera rien, et encore pire, il deviendra difficile d'intégrer les différentes parties qui ont pu être faites séparément.

**Suivi**

(+) Trop de suivi fait perdre du temps et suffoquer les développeurs.
(+) Faire un suivi ne peut pas améliorer pas l'exactitude des indicateurs à cause de la nature même du système qui est chaotique.
(+) Trop de données tue la donnée - Le syndrome de la botte de foin.

Exemple inapproprié : ***MesuréJusqu'àLaMort*** – les projets qui font perdre leur temps à tout le monde en faisant un suivi exhaustif de tout dans les moindres détails mais qui finalement n'arrivent jamais à atteindre leur objectif. (Vous avez mesuré la pression des pneus jusqu'à ce toute l'air sorte !)

(-) Un manque de suivi mène à des Blocages et potentiellement à des temps d'attente entre les assignations (de tâches).

Exemple inapproprié : ***Qu'estCeQuiS'estPasséIci*** ? – les projets qui ne font aucun suivi ont tendance à perdre tout contrôle sur ce qui est en train d'être fait. Et éventuellement, personne ne sait vraiment ce qui a été fait.

## Solution

**(Une chose, bien que temporaire)**
Faire une réunion de courte durée (~15 minutes) avec les membres de l'équipe lors d'une ***Réunion Scrum*** quotidienne, dans laquelle chaque participant répond seulement aux 3 questions suivantes :

**(Un processus et ce qui ne change pas)**
1) Sur quoi a-t-il travaillé ces 24 dernières heures. Le ***Scrum Master*** enregistrent quelles tâches ont été terminées et lesquelles restent inachevées.
2) Ce qui bloque – a-t-il trouvé quoi que ce soit qui l'empêche de réaliser ses tâches lors des 24 dernières heures. Le ***Scrum Master*** enregistre tous les blocages et trouve plus tard une manière de résoudre ces blocages.
3) Ce sur quoi il va travailler dans les 24 prochaines heures. Le ***Scrum Master*** aide les membres de l'équipe à choisir les tâches appropriées sur lesquelles travailler avec l'aide de l'Architecte. Les tâches étant planifiées sur une base de 24 heures, elles sont généralement de petite taille (***PetitesTâches***).

Les ***Réunions Scrum*** se déroulent généralement tous les jours au même endroit et à la même heure, car cela permet de construire une culture forte. En tant que telles, les ***Réunions Scrum*** sont des rituels qui permettent d'améliorer une certaine forme de socialisation de l'équipe autour du statut d'avancement, des problèmes et des planifications. Le ***Scrum Master*** conduit les réunions et enregistre les tâches de chaque membre de l'équipe dans le ***Backlog*** global du projet. Il enregistre aussi tous les blocages et résout tous les blocages pendant que les développeurs travaillent sur leurs nouvelles tâches.

Les ***Réunions Scrum*** permettent non seulement de planifier les tâches pour les développeurs mais peuvent et devraient également permettre de planifier les activités de toutes les personnes impliquées dans le projet telles que les personnes de l'intégration dédiées à la gestion de configuration, les architectes, les ***Scrum Masters***, les ***Pares-Feu***, les ***Coachs*** et l'équipe d'assurance qualité.

Les ***Réunions Scrum*** permettent aux travailleurs intellectuels d'accomplir des objectifs à moyen terme généralement alloués dans des Sprints ayant une durée d'environ un mois.

**(ce qui change)**
Les ***Réunions Scrums*** peuvent aussi être tenues par des équipes auto-organisées, dans ce cas, une personne est désignée comme scribe, elle enregistre les activités terminées et planifiées du ***Backlog*** et les Blocages existants. Toutes les activités du ***Backlog*** et les blocages à résoudre sont alors répartis au sein des membres de l'équipe.

Le format du ***Backlog*** et des blocages peuvent varier, d'une simple liste d'items sur une feuille de papier, à une représentation informatique sur INTERNET/INTRANET [Schwaber97]. Il est possible d'ajuster le cycle Scrum d'une durée allant généralement de 2 heures à 48 heures.

## Explication

Il est très facile de sous ou de surestimer une estimation, ce qui peut conduire soit à un développeur inactif soit à retarder la réalisation d'une tâche. Il s'avère plus pragmatique de *sonder* fréquemment le statut des petites tâches. Des processus ayant un fort degré d'imprévisibilité ne peuvent pas utiliser **que** des techniques de planification de projet traditionnelles, comme les diagrammes de Gantt ou les diagrammes PERT car le taux de changement de ce qui est analysé, accomplit ou créé est trop élevé. À la place, une re-priorisation constante des tâches offre un mécanisme adaptatif permettant de sonder la connaissance systémique sur des périodes de temps très courtes.

Les réunions SCRUM aident aussi à la création d'une « culture d'anticipation » [Weinberg97] en encourageant des valeurs positives qui :

- augmentent le sentiment général d'urgence,
- promeuvent le partage de connaissance,
- encouragent des communications denses et
- facilitent « l'honnêteté » parmi les développeurs étant donné que tout le monde dit là où il en est quotidiennement.

Ce même mécanisme encourage les membres de l'équipe à socialiser, externaliser, internaliser et combiner la connaissance technique en continu, et permet à l'expertise de devenir la propriété commune d'une communauté de pratique [Nonaka95]. Les ***Réunions Scrum*** sont par conséquent des rituels ayant une transcendance culturelle très profonde. Se réunir au même endroit, à la même heure et avec les mêmes personnes, augmentent le sentiment d'appartenance, et créée ainsi l'habitude de partager la connaissance. Du point de vue de la dynamique des systèmes [Sense94], le développement informatique a un problème de planification, de part la nature plutôt probabiliste des tâches de développement et de part la difficulté des estimations à faire parce que :

1) Des développeurs inexpérimentés, des responsables et des architectes sont impliqués dans les estimations
2) Généralement des dépendances d'architectures s'entremêlent et sont difficiles à gérer
3) Il existe des exigences inconnues ou très peu documentées ou
4) des défis techniques imprévus.

Par conséquent, le développement informatique devient un *jeu de distribution de la bière*[1] chaotique, où il s'avère difficile d'estimer et de contrôler le temps disponible des développeurs, à moins d'accroître le suivi des petites tâches implémentées [Goldratt90], [Senge90]. En ce sens, la ***Réunion Scrum*** devient l'équivalent d'un thermomètre mesurant constamment la température de l'équipe [Schwaber07-2].

Du point de vue de la théorie de la complexité, SCRUM favorise la cohésion du groupe en forçant l'utilisation d'un agent d'interaction plus rapide, accélérant ainsi le processus d'auto-organisation, car il change les ressources de manière opportuniste, grâce aux réunions SCRUM quotidiennes.

C'est compréhensible, parce que le relâchement d'un système multi-agent auto-organisé est proportionnel à la moyenne des échanges entre les agents par unité de temps. Et en fait, le « taux d'interaction » est l'un des leviers que l'on peut utiliser pour contrôler un comportement « émergent » -- c'est comme ajouter un enzyme à une réaction chimique.

Dans SCRUM, cela signifie augmenter la fréquence des réunions SCRUM, et autoriser davantage d'*hyperliens* comme décrit ci-dessous, mais jusqu'à une limite de fréquence maximum optimale de réunions SCRUM (réunions/temps), et ceci jusqu'à la limite maximum optimale des hyperliens ou des membres de l'équipe SCRUM. Sinon l'organisation passe trop de temps à *socialiser la connaissance*, plutôt que de réaliser des tâches.

[1] NdT – Jeu de simulation d'entreprise conçu par le MIT dans les années 1960 - plus d'informations sur https://fr.wikipedia.org/wiki/Beer_Distribution_Game

## Exemples

(Mike Beedle) – Chez Nike Securities à Chicago, nous utilisons les réunions SCRUM depuis février 1997 pour l'ensemble de nos projets y compris pour la réingénierie des processus d'affaires et le développement informatique. Toutes les personnes impliquées dans ces projets reçoivent une semaine de formation dans les techniques SCRUM.

(Yonat Sharon) – Chez Elementrix Technologies, nous avons un projet qui était en dehors des clous après environ 5 mois de développement. Seule une petite partie du périmètre (environ 20%) était réalisée et même cette partie était pleine d'anomalies. Le chef de projet a commencé à tenir une réunion d'avancement deux fois par jour (aucun d'entre nous n'était familier avec le terme de SCRUM alors). Dans le mois qui suivit, l'ensemble du périmètre fut réalisé et le niveau de qualité avait grimpé en flèche. Deux semaines plus tard, une version beta sortait. Les réunions ont été arrêtées plus tard et depuis le projet avance avec beaucoup de mal. Je ne pense pas que cela puisse être attribué uniquement aux réunions Scrum, mais elles ont eu une grande part de responsabilité dans cette réussite.

L'un de mes responsables d'équipes informatiques à RAFAEL, a mis en place une variante des ***Réunions Scrum***. Chaque jour, il va voir chaque développeur et lui pose les 3 questions. Il gère également le backlog. Cette variante n'a pas les effets de construction d'équipe, mais elle offre des éléments d'informations régulièrement.

## Contexte résultant

Une structure telle que le ***DéveloppeurContrôleLeProcessus*** est complètement implémentée grâce à la ***FormeSuitLaFonction***, ou une ***ÉquipeCas*** dans un environnement métier, et est cristallisée dans une structure hautement adaptable et hyperproductive.

À la suite de quoi, l'application de ce modèle conduit aussi à :

- donner une grande visibilité du statut du projet
- donner une grande visibilité de la productivité de chacun
- perdre moins de temps à cause des blocages
- perdre moins de temps à attendre quelqu'un d'autre
- une augmentation de la socialisation de l'équipe

# Sprint

## Contexte

***(DE : DévelopeurContrôleLeProcessus [Coplien95], CompresserLeProcessus [Beedle97], AutoSélectionDeL'équipe [Coplien95])***

Vous êtes un développeur ou un coach s'occupant d'une équipe de développement logiciel dans laquelle il y a un fort degré de découverte, de créativité ou de tests.

Les ***Réunions Scrum*** et les ***Sprints*** peuvent s'appliquer à la réalisation de systèmes informatiques, auxquels à un existant s'ajoute de nouvelles fonctionnalités, permettent de partitionner le travail, avec des interfaces entre les deux, des composants ou des objets, et le tout de manière très propre.

Nous voulons que chaque personne de l'équipe comprenne le problème dans sa globalité et soit au courant de toutes les étapes dans le développement. Cela limite la taille de l'équipe et du système développé. La confiance est la valeur centrale de Scrum, et elle est donc importante tout particulièrement pour le succès des ***Sprints***, donc l'***AutoSélectionDeL'équipe*** est un plus.

Durant un sprint, nous optimisons les communications et maximisons le partage d'information lors des ***Réunions Scrum*** quotidiennes.

Chaque ***Sprint*** prend en charge une quantité pré-allouée de travail à partir du Backlog. L'équipe s'y engage. En principe, rien de l'extérieur n'est ajouté pendant un sprint. Les ajouts venant de l'extérieur sont ajoutés au backlog global. Les problèmes issus du Sprint peuvent également être ajoutés au Backlog. Un Sprint s'achève avec la Démonstration d'une nouvelle fonctionnalité.

## Problème

Nous voulons équilibrer le besoin des développeurs de pouvoir travailler sans être dérangé et le besoin de l'encadrement et des clients de voir de réels progrès.

## Forces

Pour la plupart des gens – chefs de projet, clients, il est difficile d'abandonner le contrôle et d'avoir des preuves d'avancement telles que peut l'offrir la gestion de projets traditionnel. Cela peut paraître risqué de faire de cette manière-là car il n'y a aucune garantie que l'équipe livrera quelque chose.

Souvent, au moment de la livraison des systèmes, ces derniers s'avèrent déjà obsolètes ou ont besoin de faire l'objet de changements majeurs. Le problème est que la plupart des informations sont récoltées au démarrage du projet, alors que l'utilisateur en sait davantage sur ce qu'il veut lorsqu'il utilise le système ou lorsqu'il voit les versions intermédiaires du système.

Certains problèmes sont « retors », c'est-à-dire qu'il s'avère difficile de décrire le problème sans avoir une idée de la solution. Il est faux d'attendre de la part des développeurs qu'ils aient une conception propre et qu'ils soient en mesure de s'engager dès le début sur ce type de problème. De l'expérimentation, des retours d'informations, de la créativité sont nécessaires pour y répondre.

La plupart des processus de développement partent sur un mauvais postulat. Ils présupposent que le processus de développement est une approche comprise de tous, qui peut être planifiée et estimée. Si un projet vient à échouer, cet échec est considéré comme étant la preuve que le processus de

développement nécessite plus de rigueur. Si nous pouvions faire prendre conscience aux développeurs de suivre le processus de développement avec plus de rigueur alors le projet pourrait être réalisé avec succès.

Mais ces approches étape par étape ne fonctionnent pas, parce qu'elles ne permettent pas de gérer les éléments imprévisibles (à la fois humains et techniques) dans le développement de système informatique. Au début du projet, il est impossible d'avoir ni des spécifications complètes ni détaillées ni d'avoir un planification d'aucune sorte en raison de ces incertitudes multiples.

Développer des systèmes est quelque chose d'imprévisible et de chaotique. Le développement est un processus empirique qui demande une réflexion importante durant tout le processus. Une méthode peut uniquement apporter un cadre pour mener à bien les travaux et indiquer là où la créativité est souhaitée. Néanmoins nous traitons les processus de type boîte noire comme étant des processus complètement définis. Des résultats imprévisibles se produisent. Il nous manque les contrôles pour être en capacité de mesurer et de répondre à l'imprévisible.

Pendant que nous construisons le système, des artefacts voient le jour, de nouvelles révélations et des nouvelles perspectives s'offrent à nous. Ces nouveaux artefacts peuvent guider une réflexion future. Une augmentation de la productivité grâce à l'utilisation des bons outils ou à l'arrivée de nouveaux composants peut donner l'opportunité d'ajouter davantage de ***Backlog*** et de fonctionnalités dans notre système ou de livrer le produit plus tôt.

Les clients et les utilisateurs peuvent rarement donner une spécification finale car leurs besoins évoluent rapidement. Le mieux qu'ils puissent faire c'est de faire évoluer un produit au fur et à mesure que leurs besoins évoluent et qu'ils apprennent tout au long du processus. Dans notre processus de développement nous n'utilisons pas ni ce cycle d'apprentissage ni les bénéfices associés.

Les développeurs et les chefs de projets vivent (ou sont forcés à vivre) souvent dans un mensonge. Ils doivent prétendre qu'ils peuvent planifier, prévoir et livrer, et ensuite ils travaillent du mieux qu'ils peuvent pour livrer le système. Ils construisent d'une manière, prétendant construire d'une autre, et résultat ils n'ont aucun contrôle. Une supervision est souvent mis en place pour prouver que le processus est sur les rails et nous suivons des diagrammes de Pert ou autres, croyant qu'un système en résultera. L'automatisation des processus actuels rajoute du travail administratif pour les chefs de projets ainsi que pour les développeurs et aboutit le plus souvent à des processus de développement très peu utilisés et qui finissent par être remiser au fond d'un placard.

## Solution

Donnez de l'espace aux développeurs pour être créatifs, pour apprendre à explorer l'espace de conception, faire leur vrai travail, sans être dérangés par des interruptions extérieures, libres d'adapter leurs manières de travailler en utilisant les opportunités et les révélations qui s'offrent à eux. En même temps faites en sorte que l'encadrement et les parties prenantes gardent  confiance en leur montrant de réels progrès plutôt que des documents et des rapports … produits en guise de preuves. Faites tout ceci en cycles courts, des ***Sprints***, dans lesquels une partie du ***Backlog*** est allouée à une petite équipe. Lors d'un ***Sprint***, pendant une période approximative de 30 jours, une quantité consentie de travail sera réalisée pour créer un livrable. Le ***Backlog*** est assigné aux ***Sprints*** selon les priorités et par approximation de ce qui peut être accompli en un mois. Des morceaux

faiblement cohésifs et ayant un couplage élevés sont sélectionnés. L'accent est mis sur l'habilitation, plutôt que sur le micro-management.

Pendant le ***Sprint*** le chaos extérieur n'est pas autorisé à interférer dans l'incrément. L'équipe, au fur et à mesure de son avancement, peut changer sa course et sa manière de travailler. En la protégeant de l'extérieur, nous lui permettons de se concentrer sur le travail en cours et de livrer du mieux qu'elle peut et de la meilleure manière possible, en utilisant ses compétences, son expériences et sa créativité.

Chaque ***Sprint*** produit un livrable visible et utilisable. Il est démontré dans une ***DémoAprèsSprint***. Un incrément peut être intermédiaire ou livrable, mais il devrait être autonome. L'objectif du Sprint est de réaliser la meilleure qualité logicielle possible et de s'assurer de progrès réels, non pas des jalons de papier en guise d'alibi.

Les ***Sprints*** permettent de mettre en place un environnement sûr et d'avoir des intervalles de temps dans lesquels les développeurs peuvent travailler sans être dérangés par des requêtes extérieures ou inopportunes. Ils permettent aussi d'avoir un morceau de travail pré-alloué que le client, l'encadrement et l'utilisateur peuvent espérer avoir sous la forme d'un livrable utile, comme par exemple un morceau de code opérationnel à la fin du ***Sprint***. L'équipe se focalise sur les bonnes choses à faire, l'encadrement travaille sur l'élimination de ce qui peut empêcher l'équipe de faire encore mieux.

## Explication

Les développeurs ont besoin de temps pour travailler sans être dérangés, ils ont besoin du soutien logistique, du soutien de la part de l'encadrement et les utilisateurs doivent rester convaincus que de réels progrès sont faits.

## Exemples

Chez Argo, dans le département flamand de l'éducation, nous utilisons les ***Sprints*** depuis Janvier 1997 sur un grand nombre de projets pour des utilisateurs finaux ainsi que pour le développement d'un cadre de travail (*framework*) comprenant une base de données, une gestion documentaire et un flux de travail. Le ***Backlog*** est divisé en ***Sprints*** qui durent environ un mois. À la fin de chaque ***Sprint*** une image opérationnelle Smalltalk est livrée comprenant l'intégration de toutes les applications actuelles. L'équipe se rencontre quotidiennement lors des **Réunions Scrum** et le ***Backlog*** est alloué après la ***DémoAprèsSprint*** lors d'une réunion mensuelle dans un comité de pilotage.

## Contexte résultant

Il en résulte un sentiment fort et réel d'adhésion par l'ensemble des participants (y compris les utilisateurs qui restent impliqués durant tout le cycle, avec au minimum la démo et la priorisation du ***Backlog***).

À la fin d'un ***Sprint***, nous avons la meilleure approximation possible de ce qui avait été planifié au démarrage du ***Sprint***. À la fin du ***Sprint***, lors d'une session de revue, les superviseurs ont l'opportunité pour changer le planning futur. Le projet est totalement flexible à ce point. Cible, produit, date de livraison et coût peuvent être affinés.

Avec SCRUM nous avons une grande marge de manœuvre en terme de flexibilité post-planning (à la fois pour le client et le développeur).

Pendant les ***Sprints***, il peut devenir clair, lors des ***Réunions Scrum*** quotidiennes que certaines équipes perdent beaucoup de temps dans des tâches peu ou pas productives. La grande transparence de ***Scrum*** nous permet de gérer cela. Pendant les ***Réunions Scrum*** il peut devenir clair que les personnes ont besoin de plus de temps pour réaliser leurs tâches que le temps alloué à l'origine par l'encadrement. Il peut s'avérer que le personnel n'a pas les compétences ou être moins expérimenté pour la tâche allouée que supposé ou il peut être pris dans des enjeux politiques ou de pouvoir.

Les difficultés de sélection du backlog pour un sprint peut montrer que les priorités ne sont pas claires pour l'encadrement ou le client.

La méthode n'est pas adaptée pour les personnes ayant besoin d'un encadrement fort ; d'autres personnes peuvent considérer qu'il s'agit d'une manière de travailler qui est dangereuse ou folle.

# Backlog

## Contexte
## (DE :)

De par la nature chaotique d'un projet, qu'il soit informatique ou autre, vous ainsi que toute autre personne en faisant partie, avez besoin d'information sur quoi faire ensuite.

## Problème

Quelle est la meilleure manière d'organiser le travail à faire ensuite à n'importe quelle étape du projet ?

## Forces

Les planifications de projets sous la forme de diagramme Pert ou de diagrammes de Gantt essaient souvent de prendre en compte les tâches à faire à priori, mais elles échouent souvent au niveau de l'implémentation parce qu'il leur manque la flexibilité suffisante pour évoluer rapidement. De plus, le temps est pré-alloué aux tâches dans les diagrammes de Gantt ou Pert, mais en contraste, leurs priorités et leur nombre augmentent ou diminuent selon les besoins des *vrais* projets.

N'avoir aucun référentiel de tâches quel que soit sa forme se traduira tout simplement par l'échec du projet. Il doit y avoir d'une manière ou d'une autre une certaine forme de contrôle du projet.

## Solution

Utilisez un ***Backlog*** pour organiser le travail d'une équipe SCRUM.

Le ***Backlog*** est une liste priorisée. La priorité la plus forte du backlog sera traitée en premier, la priorité la plus faible sera traitée en dernier. Aucune fonctionnalité, aucun ajout, aucune amélioration d'un produit ne vaut le coup de se disputer ; il s'agit simplement d'éléments plus ou moins importants quant à la réussite du projet ou à sa pertinence à un moment donné de la vie du projet.

Le ***Backlog*** est le travail à réaliser pour un produit. La réalisation de ce travail transformera le produit de sa forme actuelle à sa vision concrétisée. Mais dans Scrum, le ***Backlog*** évolue de la même manière que le produit et l'environnement dans lequel il sera utilisé évolue. Par conséquent le backlog est dynamique, changé constamment par l'encadrement pour s'assurer que le produit défini en complétant le ***Backlog*** donne le produit le plus approprié, le plus compétitif et le plus utile possible.

Il existe plusieurs sources à un backlog. Le marketing produit ajoute du boulot qui permettra d'accomplir sa vision du produit. Le service commercial ajoute du boulot qui permettra d'accroître les parts de marché ou d'étendre la rentabilité de la part actuelle. Le service technique s'assurera que le produit utilisera la technologie la plus innovante et la plus productive. Le service développement ajoutera du travail pour améliorer les fonctions du produit. Le support client ajoute du boulot pour corriger les anomalies sous-jacentes du produit.

Une seule personne priorise le travail. Cette personne est responsable pour que la vision du produit prenne corps. Son titre est généralement celui de responsable produit, ou de responsable marketing produit. Si quelqu'un d'autre souhaite changer la priorité du travail soit changée, il doit convaincre

cette personne de changer cette priorité. La priorité du backlog la plus haute est celle ayant la définition la plus fournie. Elle est priorisée tout en gardant un œil sur ses dépendances.

Selon la rapidité avec laquelle les produits doivent être mis sur le marché et les budgets de l'organisation, une ou plusieurs équipes Scrum travaillent sur le backlog du produit. Dès qu'une équipe Scrum est disponible (nouvellement formée ou ayant juste fini un ***Sprint***) pour travailler sur le backlog, l'équipe rencontre le responsable produit. Se focalisant sur la priorité la plus forte du backlog, l'équipe sélectionne ce ***Backlog*** que l'équipe pense être en mesure de réaliser dans une itération de ***Sprint*** (30 jours). En faisant cela, l'équipe Scrum peut modifier la priorité du backlog en sélectionnant le backlog qui est interdépendant, c'est-à-dire plus facile à aborder tout de suite plutôt que d'attendre. Il peut s'agir par exemple de plusieurs items ayant besoin qu'un module ou une interface commune soit développée et pour lesquels il fait sens de les inclure dans un seul et même ***Sprint***.

L'équipe sélectionne un groupe cohérent des tops priorités du ***Backlog***, qui – une fois réalisé – permettra d'atteindre un objectif ou un jalon. Il s'exprime sous la forme de l'objectif du Sprint. Pendant le ***Sprint***, l'équipe est libre de ne pas faire un travail donné aussi longtemps que l'objectif du sprint est atteint.

L'équipe décompose ensuite le backlog sélectionné en tâches. Ces tâches sont des éléments distincts de travail que les différents membres de l'équipe se sont assignés à faire. Les tâches sont faites pour terminer le backlog et pour atteindre l'objectif du ***Sprint***.

## Contexte résultant

Toutes les tâches dans le projets sont déterminées dynamiquement et elles sont priorisées selon :

1) les besoins de l'utilisateurs, et
2) ce que l'équipe est en mesure de faire.

## 4. Conclusions

Scrum est un processus créateur de connaissances avec un haut degré de partage d'information durant tout le cycle et l'avancement du travail.

L'élément-clé avec Scrum est de choisir une date à laquelle vous souhaitez mettre en production ou livrer, prioriser la fonctionnalité, identifier les ressources disponibles et prendre les décisions majeures en terme d'architecture. Comparée aux méthodes plus traditionnels la phase de planification est courte étant donné que nous savons que les évènements exigeront des changements par rapport aux plans initiaux. Scrum utilise une approche empirique de développement où l'interaction avec l'environnement n'est non seulement autorisé mais également encouragée, des changements de périmètres, de technologie et de fonctionnalités sont attendus, le partage d'information et de retours d'informations en continu permet de conserver un niveau de performance et de confiance élevé.

Lorsqu'ils sont combinés ensemble, SCRUM et les autres patterns organisationnels [OrgPatt], et notamment ceux de James O. Copien [Coplien95], permettent de former une organisation adaptative de développement logiciel tout en étant structurée.

Leur application permet de générer également une culture forte avec des rôles et des relations clairement définies avec des rituels significatifs et transcendantaux.

## Remerciements

Nous voudrions remercier tous les utilisateurs de SCRUM et les relecteurs qui nous ont fait part de leurs retours d'informations au cours de ces dernières années. Nous tenons à remercier aussi les membres du Chicago Patterns Group qui ont participé à une session de passage en revue du modèle de la ***Réunion Scrum*** (tout particulièrement Brad Appleton, Joe Seda et Bob Haugen). Enfin nous remercions notre berger PLOP98, Linda Rising, pour nous avoir apporté ses observations et ses conseils pour rendre encore meilleur notre document.

(Remerciements de Mike Beedle) Je tiens à remercier Jeff Sutherland et Ken Schwaber pour avoir adapté les techniques SCRUM au développement logiciel au début des années 90, et pour avoir partagé avec moi leurs découvertes. SCRUM a eu un impact significatif au sein des projets dans lesquels j'ai pu l'utiliser.

## Références

[Goldratt90] E. Goldratt, *Theory of Constraints*, North River Press, Great Burlington (MA), 1990.

[Coplien95] J. Coplien and D. Schmidt, *Pattern Languages of Program Design (A Generative Development-Process Pattern Language)*, Addison and Wesley, Reading, 1995.

[Nonaka95] I. Nonaka and H. Takeuchi, *The Knowledge Creating Company*, Oxford University Press, New York, 1995.

[OrgPatt] Org Patterns web site:
http://www.bell-labs.com/cgi-user/OrgPatterns/OrgPatterns?ProjectIndex

[Schwaber97] K. Schwaber's, SCRUM web page: http://www.controlchaos.com

[Schwaber97-2] personal communication. [Senge90] P. Senge, *The Fifth Discipline - The art and Practice of the Learning Organization*, Doubleday/Currency, New York, 1990.

[Sutherland97] J. Sutherland, SCRUM web page: http://www.tiac.net/users/jsuth/scrum/index.html

http://www.jeffsutherland.org/scrum/index.html

[Weinberg97] G. Weinberg, *Quality Software Management – Vol. 4.,* Anticipating Change, Dorset House, New York, 1997.

## Traducteurs

Nicolas MINGO et Nicolas MEREAUX - Les traducteurs agiles