Capsule : Scrum

Ruben Gonzalez-Rubio
Génie électrique et génie informatique
Groupe $\epsilon\chi$it
Université de Sherbrooke

2011 mis à jour en 2014

Introduction

Scrum est un processus de développement agile pour le logiciel. C'est un processus qui est simple à décrire, mais pour réussir un projet, il faut l'appliquer avec beaucoup de rigueur. La première condition de réussite pour un projet Scrum est l'implication de tous les membres de l'équipe à réussir le projet ensemble. Bien entendu, toute l'équipe s'engage à suivre le processus. Ce document est volontairement court, il ne présente pas toutes les raisons justifiant chaque partie du processus. Il présente des concepts et ensuite décrit la séquence des actions qui sont à effectuer. Les termes anglais concernant les références a *Scrum* ont été conservés afin de faciliter la lecture et sont identifiés en italique, comme dans le mot suivant : *Scrum*.

En suivant le processus *Scrum* une équipe de développeurs va produire un logiciel le plus proche possible aux besoins d'un client. L'équipe effectue des *sprints* de travail qui vont influencer les développements (*sprints*) futurs, car le client doit voir à la fin de chaque *release*[1] un logiciel incomplet, mais opérationnel. Le client peut donc réagir et suggérer des changements aux développements.

Scrum n'est pas un processus complet, mais un cadre de travail où certains ajustements sont nécessaires en fonction de l'équipe et du logiciel à développer. Par exemple, la durée

1. une *release* comporte un ou plusieurs *sprints*.

des *sprints* peuvent varier d'une semaine (ou moins) à 4 semaines (ou plus). Beaucoup d'activités sont *time boxed* par exemple, un *sprint*, pour lequel il faut fixer sa durée dès le début du projet et garder celle-ci pour tous les *sprints*.

Le *Scrum team*

Le *Scrum team* est une équipe autogérée où tous les membres vont participer activement sans avoir de catégories tels qu'un chef de projet, des développeurs, des testeurs et autres. Il n'y a pas de module à développer qui est assigné à une personne en particulier, le *Scrum team* est responsable collectivement du succès. Il existe trois rôles : *scrum master*, *product owner* et *developer*. Le *scrum master* doit s'assurer que le *Scrum team* suit le cadre *Scrum* et que tous les obstacles à la progression du travail soient éliminés. Le *product owner* représente le(s) client(s) et il agit comme guide pour arriver au meilleur produit possible. Un *developer* fait les travaux d'analyse, de codage et de test. Il est à noter que le *developer* doit être capable de travailler sur tous les aspects du logiciel ; son bagage technique doit être solide.

Les *artifacts* dans *Scrum*

L'activité principale d'un projet *Scrum* est le *sprint*. Dans un *sprint* le travail d'analyse, de codage et de test est effectué afin d'accomplir les tâches enregistrées dans le *sprint backlog*. En principe, toutes les tâches du *sprint backlog* doivent être complétées dans le *sprint*. Avant un *sprint*, l'équipe effectue un *sprint planning meeting*, qui sert à planifier les tâches à effectuer pendant le sprint, car il s'agit de tâches qui représentent peu de temps de travail, donc quantifiables plus facilement. Un *sprint planning meeting* est composé de deux parties : la mise à jour du *product backlog* et la planification du *sprint* suivant. Dans le *sprint planning meeting* le *Scrum team* va choisir de réaliser des tâches qui vont effectuer les fonctionnalités qui ont la plus grande priorité dans le *product backlog*. Ce dernier est surtout maintenu par le *product owner*.

Afin d'assurer qu'un *sprint* sera complété, il y a le *daily sprint meeting* qui est *time boxed* à 15 minutes. Il faut uniquement savoir ce qui a été fait et ce qui va se faire dans la journée

en plus d'énoncer les obstacles possibles empêchant que le *sprint* puisse être complété. Afin de le garder à 15 min il n'y a pas des discussions, c'est uniquement informatif. Il sert aussi à synchroniser le travail.

À la fin de chaque *sprint*, deux autres activités sont nécessaires. La *sprint review* et la *sprint retrospective*. La première activité sert à démontrer au *product owner* et autres clients les fonctionnalités ajoutées pendant le *sprint*. En fonction de ce qui est démontré, le *product backlog* peut être modifié. La deuxième activité sert à identifier des opportunités d'améliorer le travail de l'équipe pendant le *sprint*.

Une *release* produit un livrable (logiciel en état de fonctionnement utilisable par le(s) client(s)). Avant d'arriver à une *release* il peut y avoir plusieurs *sprints*. Un projet peut comporter aussi plusieurs *releases*.

Il existe deux *artifacts* de plus le *sprint burndown chart* et le *release burndown chart* qui servent à indiquer la quantité de travail qui reste à faire pour un *sprint* ou pour une *release*.

Notons que lorsqu'un *sprint* est planifié, l'équipe doit estimer le temps qui va prendre chaque tâche et que l'addition de ces temps doit être semblable à la durée du *sprint*. Lorsqu'une tâche est effectuée, il faut enregistrer le temps passé pour la réaliser. Ainsi, il est possible de visualiser l'évolution d'un sprint avec *sprint burndown chart*. Ceci donne un aperçu de l'évolution d'un *sprint* et la possibilité de le finir dans le temps alloué.

Le travail dans *Scrum*

En général, le projet démarre par une idée et ensuite une équipe est trouvée pour y travailler jusqu'à la *release* finale ou l'annulation du projet. Beaucoup de variantes sont possibles, mais nous allons décrire un projet normal qui a les ressources nécessaires pour se rendre à un livrable final. Nous pouvons considérer deux phases : l'amorce initiale et la phase de travail, cette dernière est composée de plusieurs *releases* et chacune de plusieurs *sprints*. Il est a signaler que pendant un *sprint* plusieurs réunions techniques peuvent avoir lieu afin de prendre des décisions pour d'effectuer les développements. Ces réunions sont faites par les membres de l'équipe qui ont besoin pour accomplir les tâches. Les réunions

peuvent être de séances en *pair programming* ou concerner plusieurs développeurs. Leur durée n'est pas rigide.

Phase Initiale Normalement le *product owner* qui représente le client va diriger le *Scrum team* afin de produire une première version du *product backlog*. Cette version va évoluer au fur et à mesure que le *product owner* avec le *Scrum team* ajoutent des fonctionnalités, raffinent celles qui se dégagent et coupent celles qui apparaissent comme inutiles. Ensuite, le *Scrum team* convient de la durée de chaque *time box* et travaille pour avoir un *product backlog* contenant les fonctionnalités plus les *releases* et les *sprints*, même s'il s'agit d'une ébauche incomplète. Il est à noter qu'il s'agit d'un travail d'estimation : le *product backlog* va changer selon les développements, même chose pour le contenu et les dates des *releases*. Tout ce travail préparatoire doit prendre un temps qu'il faut convenir dès le départ et celui-ci doit être court. Les fonctionnalités ne doivent pas être des documents de spécifications avec beaucoup de pages- une ligne peut être suffisante. Ceci ne veut pas dire qu'il faut prendre à la légère cette phase, car le temps investi ne peut être récupéré. Tout refaire à chaque fois n'est pas une bonne manière d'avancer.

Phase travail Une fois la phase initiale complétée, l'équipe passe à la phase travail qui est composée de plusieurs *releases*. Chaque *release* comporte plusieurs *sprints*. Au bout de chaque *release* il y a un logiciel fonctionnel, qui est incomplet par rapport à la *release* finale. Chaque *sprint* doit contribuer à la *release* courante sans forcément livrer au client le produit en développement.

Supposons qu'après la phase initiale le *product backlog* comporte une liste de fonctionnalités, éventuellement des *use cases* et des tâches. Un *sprint planning meeting* marque le commencement de la phase et de la *release* courante. Il faut choisir selon les priorités, selon le chemin qui fait diminuer les risques et selon l'objectif de la *release* les fonctionnalités à implémenter. Il faut préciser les tâches qui peuvent par exemple être : développer un *use case* ou coder une première version élémentaire. Tout le *Scrum team* participe à ce *sprint planning meeting*. Il ne s'agit pas de développer les analyses, mais de les planifier. Le résultat est un *sprint backlog* qui doit être suivi rigoureusement pendant le *sprint*. Il faut produire un *sprint burndown chart* où on peut constater les résultats du *sprint*. Il y a

trois possibilités à la fin d'un *sprint* : Il se finit avant la planification ou il finit exactement comme planifié ou il n'est pas fini. S'il est fini exactement selon la planification, tout va bien, les estimations faites ont été justes et si tout continue de cette manière, la *release* se passera bien. Si le *sprint* est fini avant, la planification a été trop pessimiste, l'équipe peut fournir un peu plus de travail pour les *sprints* suivants. La troisième possibilité conduit à une nouvelle planification de tâches pendant le *sprint planning meeting* suivant.

Bien entendu, à la fin de chaque *sprint*, le *product backlog* peut changer, car il y a une adaptation à faire selon les désirs du client et les changements dans les priorités. À chaque *sprint* tout se répète avec les nouvelles tâches.

Phase finale C'est la fin du projet. Une réunion et un rapport post-mortem sont à faire.

Les *time boxes* pour les activités

Nom	Temps suggéré
sprint	une semaine = 40 h = 2400 min
sprint planning meeting	4 h
daily sprint meeting	15 min
sprint review	2 h = 120 min
sprint retrospective	1 h = 60 min

Une *release* = *n sprints*.