HAL
open science

APPRENTISSAGE SITUE DE L'INGENIERIE DES SYSTEMES D'INFORMATION -LA METHODE DU SAUMON

Vincent Ribaud, Philippe Saliou

To cite this version:

Vincent Ribaud, Philippe Saliou. APPRENTISSAGE SITUE DE L'INGENIERIE DES SYSTEMES D'INFORMATION -LA METHODE DU SAUMON. Questions de Pédagogies dans l'Enseignement Supérieur, Jun 2013, Sherbrooke, Canada. pp.279-289. hal-01078439

HAL Id: hal-01078439

https://hal.univ-brest.fr/hal-01078439v1

Submitted on 29 Oct 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

APPRENTISSAGE SITUE DE L'INGENIERIE DES SYSTEMES D'INFORMATION - LA METHODE DU SAUMON

Vincent Ribaud et Philippe Saliou

UEB, Université de Brest, Lab-STICC MOCS, Brest, France

Correspondant: ribaud@univ-brest.fr

Résumé

Cet article présente une méthode d'éducation en ingénierie du logiciel basée sur l'apprentissage par problème et par projet. Les activités ont des visées productives contribuant à la maintenance d'un système d'information en vraie grandeur. La construction des compétences se fait par l'exercice répété d'activités "contre le courant" où les apprenti-es remontent du concret à l'abstrait.

Mots-clés

Apprentissage par problème, didactique professionnelle, induction, ingénierie.

I. INTRODUCTION

Le cadre général de nos recherches est l'analyse de l'activité, dans le courant de l'action et de la cognition située, de l'ergonomie francophone et de la didactique professionnelle. L'activité est certes fonction de la tâche, mais également du sujet [Leplat, 2000]. L'ingénieur du logiciel, comme acteur de son métier, dispose d'un ensemble de ressources qui sont constitutives de ses capacités d'agir. Dans le champ professionnel, explique [Rabardel, 2010], "ce qui est premier, c'est l'action efficace, […] c'est le faire, c'est la réalisation. La connaissance n'est là que comme ressource mobilisable, souvent mobilisée, mais en étant subordonnée à l'agir." Pourtant même si on demande essentiellement à un professionnel une activité productive - qui est une activité de réalisation de tâches, visant à atteindre des buts, l'acteur se construit parallèlement par l'expérience. C'est une dimension ontologique de l'activité, dite activité constructive par [Samurçay et Rabardel, 1995], qui est la constitution, par l'opérateur, des ressources de son action et de son développement. Cet article présente une méthode d'éducation en ingénierie du logiciel basée sur l'apprentissage par problème et par projet, qui articule le faire et l'apprendre comme le décrit la didactique professionnelle. "Le projet de la didactique professionnelle est de rendre compte des processus qui sont en jeu d'une part, dans la formation des compétences lors des situations dans lesquelles intervient une intention de formation et, d'autre part dans le développement de ses compétences au cours de l'activité de travail.

[Rogalsky, 2004]" Toutes les activités effectuées ont des visées productives et elles prennent toutes place dans la maintenance et l'évolution d'un système d'information existant. La croissance constructive se fait par l'exercice répété d'activités "contre le courant" où les apprenti-es remontent du concret à l'abstrait avant de redescendre de l'abstrait au concret. Notre méthode présente de nombreuses analogies avec la manière dont les saumons se développent, d'où son nom: la méthode du saumon. Cet article est organisé comme suit. La section 2 présente les spécificités du développement des systèmes d'information et le positionnement de notre méthode. La section 3 illustre les activités de remontée et de descente du cycle d'abstraction et propose une première analyse de ce qui s'y joue. Nous terminons par une conclusion.

II. DEVELOPPEMENT DES SYSTEMES D'INFORMATION

II.1 Identification de trois cycles

Les auteurs ont été fortement influencés par la méthode Merise. Merise se définit comme une méthode de conception et de développement des systèmes d'information [Tardieu et al., 1983] et distingue trois cycles dans la construction du système: le cycle de vie, le cycle d'abstraction et le cycle de décision.

Les modèles de cycle de vie proposent une définition de haut niveau des différentes phases d'ingénierie d'un système d'information. Le cycle en cascade sera utilisé, qu'on peut résumer en 5 phases principales (si on exclut la maintenance): spécification (ou analyse) des exigences, conception du logiciel, implémentation et intégration, test (ou validation), déploiement (ou installation).

Le cycle d'abstraction est utilisé pour isoler, à un niveau spécifique, les éléments significatifs d'un modèle, tout en ignorant les détails. Merise définit trois niveaux d'abstraction: conceptuel, logique/organisationnel et physique/opérationnel.

Le cycle de décision rend compte de l'ensemble des choix qui doivent être fait pendant le cycle de vie; "le cycle de décision détermine les points de passage où l'organisation décidera des buts et de l'environnement du système d'information et les points de passage où elle décidera de l'adaptation de l'objet artificiel construit par le concepteur à son fonctionnement propre. [Tardieu et al. 1983, p. 39]"

II.2 La métaphore du saumon

Le modèle en cascade, comme beaucoup de modèles et méthodes d'ingénierie des SI, est hérité de l'industrie du bâtiment et des travaux publics. Il considère que les conséquences d'une modification en amont du cycle ont un impact majeur sur les coûts en aval, d'où une importance graduellement décroissante des phases. Comme un cycle de développement est une succession de phases et que les phases amont sont les plus difficiles, l'apprentissage des différentes phases se fait généralement dans l'ordre du cycle de développement. Cette progression va de pair avec une

progression selon le cycle d'abstraction: des modèles les plus abstraits vers les modèles les plus concrets. Si la succession des phases est parfaitement logique et adaptée à la production d'un système, d'un logiciel ou d'un système d'information, elle n'est peut-être pas adaptée pour l'apprentissage, notamment parce qu'elle fait commencer par les phases les plus difficiles et les plus abstraites.

Cet article effectue certaines analogies à partir de la biologie du saumon. Presque tous les saumons sont anadromes, du grec anadromos "qui court en remontant". Le saumon nait en eau douce et se développe d'abord dans les eaux oxygénées des sources; les alevins nagent face au courant pour se nourrir des débris organiques qui y sont charriés. Les jeunes saumons, appelés tacons, vivent de deux à sept ans dans la rivière, jusqu'à leur transformation en smolt. Un smolt a atteint la maturité nécessaire pour quitter sa rivière natale et dévaler jusqu'à l'océan. Le moment venu, il retourne dans la rivière où il est né pour s'y reproduire.

Le cycle de vie du saumon nous inspire des métaphores sans doute inexactes qui nous permettent d'illustrer la problématique d'éducation à l'ingénierie du logiciel:

1. La rivière est le milieu de développement des jeunes saumons, elle est très oxygénée, plus sûre et riche en nourriture en amont et devient progressivement plus hostile en aval; l'estuaire fait la transition avec le milieu maritime.
2. En aquaculture, les saumons juvéniles doivent lutter sans cesse contre un courant artificiel. Ce régime de nage forcée les fait grossir deux fois plus vite.
3. Un saumon traverse plusieurs états: alevin, tacon, smolt. Les smolts subissent de profondes modifications physiologiques pour pouvoir vivre dans l'océan.

Appliqué à l'éducation des futurs ingénieurs en systèmes d'information, notre système d'éducation repose sur quelques principes:

1. Il faut un milieu de développement - un système d'information - où la vie est de plus en plus difficile et où les apprenti-es font leurs expériences.
2. Les apprenti-es se développent en remontant le courant et il est plus facile de commencer par les phases aval, là où le courant est plus calme.
3. L'éducation des apprenti-es se fait à l'aide de techniques d'éducation qui leur permettent de se transformer progressivement mais en profondeur.

II.3 Apprentissage par problème et par projet

[Prince, 2004] définit l'apprentissage par problème (APP) comme "une méthode d'instruction dans laquelle des problèmes significatifs sont introduits au début du cycle d'instruction et sont utilisés pour fournir le contexte et la motivation des apprentissages qui en découlent. […] Typiquement, l'APP implique une quantité importante d'apprentissages autodirigés de la part des étudiant-es." L'essor de la pédagogie par projet est généralement daté dans les années 20, où J. Dewey, C. Freinet, M. Montessori et d'autres ont mis l'élève au centre d'une pédagogie qui le transforme en sujet de sa propre formation. Le projet fait de l'institution éducative un lieu social, [Vassiliev, 1995] parle de socio-pédagogie: "les compétences comportementales s'y acquièrent par le vécu direct d'une mise en situation réelle."

L'approche présentée dans cet article est qualifiée d'apprentissage par problème et par projet, APPP. Notre approche tire du projet sa finalité "l'acte de penser est réglé par sa fin [Dewey, 1935]" où apprendre est la réalisation d'un travail qui mène à un but : de nouveaux composants d'un système d'information. Notre approche tire aussi du cycle de vie d'un projet sa dimension organisationnelle pour l'enseignant-e, "forme de régie d'ensemble de l'activité qu'il[elle] conduit [Morandi, 2002]". Les apprentissages sont situés dans le projet mais la situation n'est pas tout à fait naturelle, elle doit être proposée par l'enseignant-e. D'où la référence première à l'apprentissage par problème - ce qui en fait d'ailleurs la principale complexité : dans notre approche, chaque session met en situation les apprenti-es au carrefour de plusieurs dimensions: 1- comme une activité d'ingénierie et de construction du logiciel donnant du sens à l'action de l'apprenti-e, à la fois un but et les conditions de réalisation; 2 - comme un problème mettant en scène un ensemble de connaissances et de savoir-faire à acquérir, l'apprenti-e se les approprie sur la base de la résolution du problème qu'il-elle conçoit et met en œuvre; 3 - comme une situation qui n'est pas directement pédagogique, car elle doit induire un mode de travail autonome et organiser les ressources mises à disposition de l'apprenti-e. Les situations-problèmes sont tantôt didactiques car elles organisent l'apprentissage et les relations entre l'apprenti-e, l'objet des apprentissages et l'enseignant-e, et tantôt a-didactiques [Brousseau, 1999], c'est-à-dire "dédidactifiée": la situation fait interagir l'élève avec un milieu qui semble non didactique et où "les intentions du professeur sont suffisamment opaques pour qu'il renonce à s'en servir pour guider ses options [Morandi, 2002]." Cela induit une double qualification de l'enseignant-e: la didactique des situations et une certaine expertise du métier d'ingénieur du logiciel.

III. REMONTER PUIS DESCENDRE: UN ENTRAINEMENT AUX ACTIVITES D'INGENIERIE DU LOGICIEL

III.1 Contexte

Depuis septembre 2010, l'université de Brest et 9 universités publiques marocaines (Agadir, Casablanca, Fès, Kenitra, El Jadida, Marrakech, Rabat, Settat, Tanger) ont établi un double diplôme de Master en Offshoring des Technologies de l'Information. Les étudiant-es marocain-es, après une première année de Master dans l'université d'origine, effectuent leur deuxième année de Master à l'université de Brest: un semestre d'études à Brest suivi d'un semestre de stage de fin d'études en France dans la perspective d'une embauche au Maroc.

La grande diversité des Master d'origine ainsi que la nature plutôt théorique des études d'informatique dans l'enseignement supérieur marocain posent un problème d'hétérogénéité et un problème d'adaptation aux exigences du monde du travail. Après deux années d'observation et d'expérimentation de cette formation, nous avons introduit une approche par problème et par projet pour environ un tiers des cours techniques de la deuxième année. La frontière entre les unités d'enseignement

où s'effectue cette approche est supprimée: tous les cours, travaux dirigés, travaux pratiques sont confondus et délivrés sous formes de sessions de travail concernant la maintenance et l'évolution d'un système d'information, initialement développé pour les besoins de notre département d'informatique.

Comme indiqué, l'entrainement aux activités d'ingénierie du logiciel se fait d'abord dans le "sens inverse du courant" dans le cycle de vie comme dans le cycle d'abstraction. Nous présentons dans cette section des exemples d'entrainement. Ce sont des activités de maintenance et d'évolution d'un système d'information complexe et volumineux qui est le milieu de développement des apprenti-es.

III.2 Remonter du modèle physique au modèle logique

Qu'est ce qu'un modèle physique d'un système d'information ? L'ensemble des programmes et des structures de données qui, en s'exécutant, produit un système en fonctionnement. Ce système s'accompagne de documentation-utilisateur: un manuel d'installation, un manuel utilisateur. On peut aussi disposer d'une documentation technique comme le schéma de la base de données, la documentation du code.

III.2.1 Modèles de traitement

Si on considère un programme (dans un langage de programmation) comme un modèle physique de traitement, alors un algorithme est un modèle logique de traitement. L'enseignement de la programmation est par nature déductif, à partir d'algorithmes qu'on traduit en programmes. La plupart des enseignants d'algorithmique et de programmation estiment que l'apprentissage de l'algorithmique est plus difficile que la programmation. Alors pourquoi commencer par là ? Il est difficile de présenter des algorithmes complexes à des apprenti-es alors qu'on peut leur faire manipuler des parties de programmes complexes. Les premières sessions d'APPP ont pour objectif de familiariser les apprenti-es avec des programmes de grande taille, en assemblant, modifiant légèrement, réécrivant, … des composants du système, en travaillant sur des échantillons de taille de plus en plus grande. On pourra souligner que ces activités sont décrites par des verbes d'action associés au niveau "Application" de la taxonomie de Bloom. Lorsque la compréhension d'une partie du système grandit, on peut demander aux apprenti-es de décrire l'organisation du code en paquetages et de faire un résumé des fonctions par paquetage, puis de réorganiser le code; ce qui correspond à des verbes d'action du niveau "Analyse".

Les activités précédentes supposent de disposer du code (ou d'une partie) d'un système d'information, qui est alors vu comme une boîte blanche. Mais, on peut aussi faire manipuler le système comme une boîte noire dont on observe le comportement. L'observation du fonctionnement fournit un modèle abstrait du système, qu'on peut considérer comme un modèle logique de traitement. D'autres sessions d'APPP peuvent ainsi s'appuyer sur l'observation du système (décrire, identifier, lister, reconnaître, … niveau "Connaissance" de Bloom) pour produire une explication, un résumé, une généralisation (niveau supérieur "Application").

III.2.2 Modèles de données

Le propre d'un système d'information est d'être centré sur les données et le modèle physique des données est composée d'un ensemble de constructions qui jouent le même rôle que les programmes (on utilise généralement le langage SQL). A la différence de ce qui se fait généralement en enseignement, le système d'information dans lequel évoluent les apprenti-es a un modèle physique de données complexe: 90 tables, 60 vues, 50 paquetages, 600 triggers, 270 index. Ce qui a été proposé dans la section précédente est appliqué sur les composants de description des données: les premières sessions d'APPP favorisent la compréhension d'un système à grande échelle en réécrivant, restructurant, réorganisant les composants.

De plus, les activités peuvent être assistées par l'emploi d'outils de rétro-ingénierie. La représentation abstraite d'une base de données sous la forme d'un modèle logique (le modèle relationnel) puis l'implémentation de ce modèle logique dans un modèle physique (en SQL) n'a pas fondamentalement changé depuis 30 ans et n'importe quel outil de génie logiciel propose une transformation automatique du niveau logique au niveau physique. Comme cette transformation est parfaitement maîtrisée, de nombreux outils proposent une transformation inverse (appelée rétro-ingénierie ou rétro-conception selon les outils) du niveau physique au niveau logique. Cette capacité de l'outillage fournit de nouvelles activités d'APPP: à partir de la rétro-ingénierie fournie par l'outil, les apprenti-es peuvent produire de nouveau le modèle physique avec le même outil ou avec un outil différent, comparer et analyser les différences, converger vers le même résultat, inférer les règles de transformation qu'utilisent les outils, critiquer les choix de génération, chercher des manières différentes de faire … (niveau "Analyse" - et supérieurs- de Bloom).

III.2.3 Analyse de l'activité

Les sessions décrites précédemment ont pour objectif la maîtrise des activités de construction d'un système d'information: la programmation proprement dite et le passage de la représentation logique aux programmes physiques. A la lecture des activités d'APPP proposées dans les deux sections précédentes, on comprend que l'apprentissage des tâches est basé sur un des piliers de l'analyse du travail: la conceptualisation dans l'action, qui nous vient directement de [Piaget, 1974]. D'après [Pastré, 2011], "la première propriété de la connaissance est d'être opératoire: elle sert à orienter et guider l'action." Dans l'action de programmer, il existe des invariants qu'on retrouve dans toutes les situations. C'est ce que [Vergnaud, 1996] appelle le schème "une organisation invariante de l'activité pour une classe de situations donnée". La transformation d'un modèle logique à un modèle physique est un schème que l'apprentissage doit rendre explicite.

La plupart des activités d'APPP suivent une approche inductive. [Fourez, et al., 1997] la définisse comme "l'induction est une opération mentale au cours de laquelle on passe d'observations multiples à l'énoncé d'une loi (ou d'un modèle) qui en rend compte. Il s'agit d'une sorte de généralisation (qui passe du particulier au général)." La démarche inductive est le moyen d'accéder à la boucle que pratique en permanence un ingénieur du logiciel. Ce qui est remarquable de l'ingénierie des

systèmes d'information est l'existence simultanée de représentations de plus en plus abstraites du système. Un ingénieur expérimenté qui étudie un modèle logique de données "voit" les différents modèles physiques qui en découlent, et inversement dans la pratique d'un modèle physique particulier, l'ingénieur expérimenté "voit" le modèle logique sous-jacent. Il réfléchit et agit sur les deux niveaux simultanément.

III.3 Remonter du modèle logique au modèle conceptuel

Si tout modèle physique résulte en programmes, on est obligé de distinguer au niveau logique les modèles qui conviennent aux données, de ceux qui conviennent aux traitements. Comme dans la section précédente, un modèle logique en opération est guidé par une conceptualisation qu'on appelle un modèle conceptuel.

III.3.1 Modèles de traitements

[Krutchen, 1995] a proposé de modéliser les traitements au travers de 5 perspectives ou vues qui ont fortement inspirées la conception d'UML. Notre expérience industrielle nous a fait comprendre que c'est la vue des composants qui organise tout le développement d'un système. La vue des composants (development view) représente l'organisation du code en termes de modules, est dépendante des concepts des langages et environnements de programmation, et supporte la gestion du code (configuration et version). La notion de CSCI (Computer Software Configuration Item) est primordiale: elle désigne un groupe d'éléments du logiciel, traité comme une entité unique dans la gestion de configuration, mais aussi dans la gestion en général: exigences allouées au CSCI, dépendances entre CSCI, budget, ressources et dates, ... Les CSCI sont décomposables. On utilise le diagramme de paquetage pour la représentation hiérarchique des composants et de leurs interfaces.

Les apprenti-es ont deux activités principales à apprendre et à maîtriser (i) la conception architecturale : transformer les exigences du logiciel en une conception de l'architecture du logiciel qui décrit la structure de haut niveau et identifie les principaux items du logiciel, (ii) la conception détaillée : décomposer la conception architecturale du logiciel en une conception détaillée pour chaque composant du logiciel décrivant toutes les unités du logiciel à produire et à tester.

C'est encore l'approche inductive qui va permettre l'appropriation de ces activités. A partir de modèles logiques de taille significative (décrivant l'organisation des CSCI), les apprenti-es font devoir reconstruire un modèle plus abstrait des fonctions du logiciel. Nous utilisons une méthode systémique dérivée de la méthode Case* proposée par [Barker, 1990]. La hiérarchie de fonctions est le modèle proposé pour analyser les traitements, qui sont hiérarchiquement divisés en un ensemble d'activités appelés fonctions. Le premier niveau de la hiérarchie fait apparaître la division du système d'information en pans fonctionnels (sous-systèmes). Le niveau suivant fait généralement apparaître une certaine unité organisationnelle: la fonction est utilisé par une catégorie d'acteurs, à l'intérieur d'un workflow d'activités, opérant sur un sous-ensemble des données, pour obtenir un but précis. La hiérarchie de fonctions représente le résultat de la conception générale et la conception détaillée.

Un exemple d'activité inductive est de regrouper les différents composants du logiciel en fonctions. Les apprenti-es peuvent effectuer ce regroupement en se basant sur les usages des données: quelles sont les tables utilisés par les modules ? et comment les modules utilisent-ils les tables i.e. est-ce que le module crée, retrouve, met à jour ou détruit (Create, Retrieve, Update, or Delete) des lignes de la table ? On peut regrouper les modules qui ont des usages similaires des données dans la même fonction. Le regroupement peut aussi être opéré sur l'organisation du code, en analysant la hiérarchie des composants, en identifiant quelles sont les dépendances entre les composants, c.à.d. quels sont les composants qui sont utilisés par d'autres (qui en sont donc dépendants). D'autres activités visent à établir une vision de plus haut niveau des services fournis par chaque composant, puis pour chaque ensemble de composants. Une fois une compréhension d'ensemble obtenue pour le système ou un sous-système, les apprenti-es effectuent la réorganisation des composants d'un système ou d'un sous-système, qu'on appelle refactoring. Le refactoring semble opérer seulement au niveau logique et physique mais il met en jeu aussi des activités d'analyse et de synthèse, qui font apparaître un modèle conceptuel des traitements qu'on modifie puis qu'on transforme de nouveau en un modèle logique. Les activités d'APPP permettent aux apprenti-es d'expliciter ces allers-retours entre niveaux.

III.3.2 Modèles de données

La modélisation des données n'a pas beaucoup changé depuis 30 ans. Un modèle conceptuel de données décrit l'organisation des données d'un domaine à l'aide d'entités, d'associations et de propriétés, on l'appelle généralement un modèle entité-association. La transformation d'un modèle conceptuel (entité-association) en un modèle logique (relationnel) repose sur un ensemble de règles, normalement maîtrisées par tout diplômé-e d'une licence d'informatique. Ce qui manque aux apprenti-es, c'est de se confronter avec la complexité et l'hétérogénéité. Avoir appris sur un cas d'école avec quelques entités et associations transformées en quelques tables à l'aide de quelques règles ne prépare pas à la compréhension d'un grand système d'information tel que celui d'une banque ou d'une entreprise de transport. Ces grands systèmes sont composés de sous-systèmes hétérogènes, développés sur plusieurs années de manière différente et avec des technologies différentes par des équipes successives. Le système d'information qui sert de cadre à toutes les activités d'APPP a été développé sur trois années successives en trois sous-systèmes ayant des interactions complexes. Chaque sous-système est structuré en articles de configuration (CSCI); et on dispose, pour chaque CSCI, du modèle conceptuel et du modèle logique correspondant. Qui plus est, pour certains CSCI, nous avons maintenu la synchronisation entre modèles conceptuel et logique. De plus, l'atelier de génie logiciel a une maturité telle que la transformation du niveau conceptuel au niveau logique est complètement paramétrable et parfaitement maîtrisée et que l'outil permet aussi la transformation inverse, une opération appelée rétrofit.

Les activités d'APPP se font toujours selon le schéma inductif puis déductif. Les aprenti-es travaillent sur des CSCI dont ils maîtrisent le niveau logique grâce à des activités d'APPP précédentes. Il-elle-s effectuent le rétrofit des modèles

(logiques) relationnels de ces CSCI pour obtenir un modèle (conceptuel) entité-association. Comme le rétrofit est incomplet et imparfait, les apprenti-es doivent le corriger, l'améliorer et le compléter, ce qui est une activité plus à leur portée qu'une création ex-nihilo. Les différentes options de rétrofit sont expérimentées et l'observation du résultat illustre les règles de transformation utilisées par l'outil dans un sens comme dans l'autre. Les activités d'APPP suivantes visent la modélisation conceptuelle d'un nouveau CSCI - d'une manière classique - mais prenant place dans le système existant. Les activités finales d'APPP visent à transformer le nouveau modèle entité-association en modèle relationnel avec l'aide des services de l'atelier.

III.3.3 Analyse de l'activité

Les sessions décrites ont pour objectif la maîtrise des activités de conception d'un système d'information: la conception architecturale et la conception détaillée et le passage de la représentation conceptuelle aux modèles logiques. Une partie des activités d'APPP appartient au registre de la conceptualisation dans l'action vu précédemment mais une autre partie s'appuie sur une méthodologie qui s'inspire de la psychologie ergonomique: la distinction (et l'écart constaté) entre la tâche prescrite, le faire que l'organisation a défini, et la tâche réelle, ce que l'opérateur fait. Les tâches prescrites pour la transformation d'un modèle conceptuel de données à un modèle logique sont explicites, stables et bien outillées; elles fournissent un excellent point de départ pour arriver à la tâche effective. Pourtant on observe une grande variété de situations dans un système de grande taille: les prescriptions n'ont pas toujours été uniformément appliquées, sciemment ou non; ou bien les prescriptions ont changé car elles étaient trop détaillées ou trop rigides et inadaptées à certaines situations. [Pastré, 2011] explique "qu'on va chercher en didactique professionnelle à identifier, à côté des concepts qui structurent la tâche, des 'jugements pragmatiques', assez souvent implicites, qui permettent de comprendre comment les agents organisent leur activité." Le débutant cherche l'appui d'une représentation "exacte" s'appliquant à toute situation, alors que l'ingénieur expérimenté a une représentation opérative, déformée selon la situation de travail.

IV. CONCLUSION

Les activités d'APPP décrites ici ont été mise en œuvre lors du premier trimestre de la 2ème année du Master qui sert de cadre d'expérimentation. Après une période de flottement de quelques semaines, la plupart des élèves ont compris les apports de l'APPP et se sont engagés assez activement dans ces activités. Il n'est pas sûr que tous les élèves aient adhéré à cette nouvelle forme d'apprentissage, ce qui peut induire comme le souligne [Sweenev, 1999] "inconfort, confusion, antipathie, manque de coopération et méfiance à l'égard de l'APP." L'analyse des notes permettra d'évaluer si l'APPP introduite cette année a réduit l'hétérogénéité des élèves par rapport aux années passées. L'amélioration d'une attitude active sera évaluée à l'issue du stage, en corrélant les évaluations des élèves lors de l'APPP et

l'engagement des stagiaires dans la résolution de problèmes - un des cinq indicateurs qu'on demande aux tuteurs professionnels d'évaluer lors du stage de fin d'études.

REFERENCES

Barker, R. (1990). Case Method: Tasks and Deliverables. Londres : Addison-Wesley Longman (Pearson).

Brousseau, G. (1999). Théorie des situations. Paris : La pensée sauvage.

Dewey, J. (1935-1968). Expérience et éducation. Paris : Armand Colin.

Fourez, G., Engelbert-Lecomte, V. et Mathy, P. (1997). Nos savoirs sur nos savoirs, Un lexique d'épistémologie pour l'enseignement. Bruxelles : De Boeck.

Krutchen, P. (1995). The 4 + 1 View Model of Architecture. IEEE Software, volume 12, n° 6 pp. 42-50.

Leplat, J. (2000). L'analyse psychologique du travail. Toulouse: Octarès.

Morandi, F. (2002). Pratiques et logiques en pédagogie. Paris : Nathan Université.

Pastré. P. (2011). La didactique professionnelle - Un point de vue sur la formation et la professionnalisation. Education Sciences & Society, Competenza e professionalità, volume 2, n°11, pp. 83-95.

Piaget, J. (1974). La prise de conscience. Paris : PUF.

Prince, M. (2004). "Does Active Learning Work? A Review of the Research". Journal of Engineering Education, pp. 223-231.

Rabardel, P. (2010). des motifs et des buts... http://www.cafepedagogique.net/2010 /DDP_Rabardel.aspx (page visitée en Décembre 2012).

Rogalski, J. (2004). La didactique professionnelle : une alternative aux approches de « cognition située » et «cognitiviste » en psychologie des acquisitions. @ctivités, 1 (2), pp. 103-120. http://www.activites.org/v1n2/Rogalski.pdf

Samurçay, R. et, Rabardel, P. (1995). Work competencies: some reflections for a constructivist theoretical framework. In Proceedings Theoretical approaches of competences at work, Courcelle sur Yvette (France).

Sweenev, G. (1999). The challenge for basic science education in problem-based medical curricula. Clinical and Investigative Medicine, volume 22, pp. 15-22.

Tardieu, H., Rochfeld, A. et Coletti, R. (1983). La méthode Merise : Principes et outils, t. 1. Paris : Les éditions d'organisation (1989, 1991, 1994, 2000 et 2003).

Vassilief, J. (1995). Histoire de vie et pédagogie du projet. Lyon : Editions Chroniques sociales.

Vergnaud, G. (1996), "Au fond de l'action, la conceptualisation". In Barbier J.-M. (dir). Savoirs théoriques et savoirs d'action. Paris : PUF.