Développement, Intégration de Moodle : Engagements sur le code

Le développement dans un environnement applicatif complexe et ouvert permet de très nombreuses opportunités d'évolution, amélioration, adaptation. Il permet au client d'obtenir de l'application qu'elle converge petit à petit vers sa propre définition du modèle métier.

Etant donné les coûts significatif de la mobilisation d'expertises diverses (conception, architecture, design, coding et testing, qualité et packaging) et les enjeux parfois importants dans le quotidien de l'exploitation des fonctionnalités, il est nécessaire de projeter ses ambitions dans un modèle crédible de niveau d'achèvement, et d'ajuster ses exigences en relation. Les injonctions de qualité pure ne sont pas de l'ordre de l'absolu. Une péréquation peut et doit être faite entre :

- Les enjeux métiers
- Les budgets mobilisables
- Les garanties demandées (performance, maintenabilité, réversibilité)

Si la Rolls Roice n'est pas toujours la meilleure solution de transport en fonction des circonstances, il est indispensable de pouvoir décider “en toute connaissance et transparence” du degré de finition ou d'exactitude technique demandé, selon des critères normés.

Edunao propose deux modèles de qualification de la qualité techniques des solutions développées ou intégrées. Le premier modèle s'adresse à l'architecture à la conception, le deuxième modèle s'adresse à la réalisation technique elle-même. Ces modèles sont applicables soit comme “niveau d'objectif technique” contractuel dans un contrat de développent de solution technique, soit comme “niveau d'acceptabilité” pour des composants tiers qui sont candidats pour faire partie d'une solution.

Modèle pour l'architecture

L'architecture logicielle ou encore “conception technique” regroupe les décisions d'organisation des données et des traitements qui leur sont appliqués, leur découpage “organique” et la répartition des responsabilités dans les différents “composants” de la solution.

Les objectifs de l'architecture sont :

- Améliorer la maintenabilité à moyen long terme : Un objet technique bien structuré est bien plus facile à revisiter ultérieurement.
- Améliorer la fiabilité : Par la localisation précise d'une responsabilité à un endroit précis, en évitant les écritures multiples des mêmes traitements.
- Améliorer l'intégrabilité : en édifiant des contrats identifiés avec l'extérieur de la solution, isolés dans des endroits qui ne remettent pas en question la totalité du développement à chaque modification.
- Améliorer l'évolutivité : En mettant en place les structures qui “anticipent” les ajouts futurs, et rendent ces ajouts plus simples à faire.
- Augmenter la puissance et la productivité à moyen terme : En encapsulant des fonctions (simples ou complexes) dans un “objet” ou un “composant”, elle crée les briques de base d'un assemblage à une plus grande échelle.

L'architecture a toujours un coût. Et elle est toujours la contrepartie d'un gain ultérieur. Elle constitue

donc un investissement. La structure de coût de l'architecture est divisée en :

- Cout de construction (la mise en place des principes d'architecture)
- Coût d'entretien (le coût de l'effort de changement lorsque des modifications de l'architecture sont décidés).

A trop forte dose, le coûts de l'entretien de l'architecture peut devenir plus grand que le bénéfice réellement réalisé sur le projet. Il convient donc d'être prudent sur le niveau d'architecture exigé.

L'examen à grosse maille des productions logicielles montre une répartition des développements dans 3 grandes catégories :

- **Développements sans architecture :** Le développement vise à la réalisation immédiate de l'effet demandé, sans aucune prise en compte de sa pérennité, sa réutilisablité, son adoptabilité. On appelle souvent aussi ce type de développement du design “jetable”.
- **Architecture opérationnelle :** Les principes d'architecture sont introduits “par nécessité” pratique du développeur, qui y tire un avantage pragmatique et une réduction de l'effort immédiat.
- **Architecture conceptuelle :** Le développement suit des règles strictes d'organisation dès la première ligne de code, en appliquant des règles d'organisation systématiques, structurées et normatives.

Il s'agit bien entendu d'un modèle assez grossier et des variations plus fines existent entre chaque catégorie.

La deuxième approche de l'architecture adresse la répartition des “responsabilités fonctionnelles” inhérentes à tout système de traitement des données. Il s'agit de responsabilités génériques présentes dès qu'un objet technique acquière, stocke, transforme ou restitue des données et des informations.

La réalisation ou la non réalisation de ces responsabilités contribue à livrer un produit plus ou moins complet, plus ou moins capable de s'intégrer en partie ou totalement dans les processus d'usage, mais aucune des fonctionnalités n'a de prédominance sur une autre :

- **Persistance :** Le système ou composant permet-il de sauvegarder son état sous une forme persistance qui peut être récupérée lors de par exemple un redémarrage.
- **Administabilité :** Le système peut il être réglé pour contextualise sa réaction à une situation de gestion particulière (rôle de l'usager, endroit/contexte ou le composant est utilisé) ?
- **Arité, dimensionalité :** Le système est il capable de traiter un problème (arité 1, dimension 0), une collection de problèmes (arité N, dimension 1), une organisation de problèmes (arité N, $1 <$ dimension < 2, ou un ensemble large de problèmes (arité N, dimension M) ?
- **Internationalisation :** L'usage par des communautés culturelles distinctes est-il prévu ?
- **Connectabilité :** Le dispositif dispose-t-il d'une logique de 'connecteurs' qui permettent d'envisager un raccordement facile avec d'autres fonctions du système ?
- **Exportablité :** Le dispositif me permet-il d'extraire (ou respectivement alimenter) sous de format divers en vue de transporter les données vers d'autres outils ?
- **Performance :** Le dispositif est il équipé de stratégies qui permettent de faire croître la volumétrie de données traitées dans une performance compatible avec les autres contraintes de ses opérateurs ?
- **Manoeuvrabilité :** Les opérateurs disposent-t-ils de toutes les commandes qui leur permettent de réaliser toutes les transformation de données dont ils ont besoin ?

Modèle pour l'implémentation (la réalisation technique)

La réalisation technique englobe les phases d'écriture, de test unitaire, de mise au point, de design (interfaces), d'installation de la configurabilité (paramétrisation), de packaging (localisation, installeurs, publication), et enfin de documentation.

L'*écriture* peut faire suite à une action d'architecture (pour les fonctions nouvelles), ou directement engagée (pour des équipes ayant une forte expérience de la culture de développement de Moodle).

Les *tests unitaires* sont réalisés sur une instance de développement, avec des données unitaires de test (tests ALPHA).

La *mise au point* (ou test BETA) suppose le développement globalement achevé dans ses structures principales, et capable d'être opéré dans un environnement de données réelles. La mise au point peut reprendre des éléments de design et modifier l'écriture dans des cycles micro-itératifs de développement.

Le *design* consiste à reprendre, organiser et optimiser les restitutions de données et l'ergonomie de commandes (choix position, organisation et accessibilité des commandes) afin d'obtenir une expérience utilisateur satisfaisante.

La *configurabilité* permet d'obtenir une souplesse d'adaptation de la fonctionnalité à de variation de l'environnement ou du contexte.

Le *packaging* consiste à obtenir un “paquet livrable et installable”, opérationnel pour l'usager final.

La *documentation* consiste à écrire les instructions et communications aidant à l'appropriation de l'usage par les usagers finaux.

Toutes ces opérations peuvent être cadencées selon deux grand modèles :

- Le projet phasé : Chaque opération suit la précédente. Chaque opération traite l'ensemble des points fonctionnels des exigences.
- Les micro-iterations : Une micro-itératio engage tout ou partie (la partie accessible) des opérations de mise en oeuvre dans le but de fournir en sortie un état fonctionnel (mais pas nécessairement complet) du dispositif. C'est le principe privilégié du développement agile.

Revenir à l'index de la TMA/SLA