Un environnement de développement pour l'entreprise

Pour un directeur informatique, le choix d'un atelier de développement est une des décisions les plus difficiles à prendre. Il y a à cela trois raisons : c'est une décision à long terme, elle engage l'entreprise toute entière, et elle doit tenir compte d'un très grand nombre de paramètres. Avant de détailler ces trois points, essayons de définir ce qu'est un atelier (ou environnement) de développement, les types de besoins auxquels il répond, et les contraintes auxquelles il est soumis.

Qu'est-ce qu'un atelier de développement ?

Voici une définition possible d'un atelier, ou environnement, de développement : « un ensemble cohérent d'outils logiciels qui aident une organisation à analyser, concevoir, réaliser, maintenir et documenter ses logiciels, en accord avec ses besoins, sa politique, sa stratégie, son histoire et ses normes de qualité ». Cette définition permet de faire le distinguo entre un outil, qui est un logiciel au service du programmeur, du concepteur, ou d'une petite équipe de développement, et un atelier, qui est véritablement au service de l'entreprise. Lorsque le patron informatique d'une grande banque décide que, dorénavant, tous les projets seront développés avec tel atelier, ce n'est pas pour faire plaisir aux programmeurs, même si, comme lors de la mise en place de tout atelier, les besoins de l'utilisateur final doivent être pris en compte. Le changement d'atelier aura un impact direct sur la date de mise sur le marché des nouveaux produits bancaires, et donc sur la compétitivité de cette banque.

On peut comprendre dès à présent à quel point la décision du choix est difficile : deux banques concurrentes pourront faire le même choix d'environnement de développement avec des conséquences très différentes sur les produits bancaires développés, car l'introduction du même atelier aura un impact différent selon la culture de l'entreprise utilisatrice, ses besoins, son infrastructure.

Les trois facettes

Pour comprendre un outil ou atelier de développement dans sa globalité, il est indispensable de l'examiner sous trois points de vue différents : technologique, méthodologique, et fonctionnel.

Le point de vue technologique est en rapport avec la cible de l'application finale : plate-forme d'exécution de l'application, protocoles de communication entre client et serveur, bases de données supportées, interface utilisateur. C'est, en particulier, le point de vue de l'architecte , mais aussi celui de la maîtrise d'ouvrage et de la production: à quoi ressemblera l'application finale ? quelles seront ses performances ? Quel est le middleware le mieux exploité par l'outil ?

Le point de vue méthodologique concerne le département méthodes et qualité, mais également les chefs de projet et, dans une certaine mesure, l'ensemble des équipes de développement. Quelles sont les méthodes supportées par l'outil ? pourra-t-on continuer à travailler en Merise ? dans quelle mesure devra-t-on modifier nos procédures ? sommes-nous prêts à révolutionner notre manière d'aborder collectivement le développement ?

Le point de vue fonctionnel est celui du développeur en tant qu'individu. L'outil est-il ergonomique ? oblige-t-il le programmeur apprendre un nouveau langage de programmation ? C'est aussi celui de l'administrateur de l'atelier. Celui-ci dispose-t-il d'un référentiel partagé ? avec quels autres outils peut-il s'interfacer ou échanger des données ?

Les contraintes internes de l'outil

Un moyen de comprendre la « philosophie » d'un outil ou d'un atelier consiste à l' examiner, tant que faire se peut, de l'intérieur, c'est à dire du point de vue de son éditeur.

Celui ci doit faire face à une double contrainte.

La première contrainte est d'ordre méthodologique. Elle peut avoir la force tranquille du rouleau compresseur. Les éditeurs d'ateliers de conception et de réalisation sont parfois soumis à une pression énorme. Pour s'en convaincre, il suffit de voir comment UML a forcé les éditeurs des outils à se mettre à cette nouvelle norme. Toutes les méthodes plus ou moins propriétaires ont dû

céder à UML, au moins sur le plan de la notation.
La deuxième contrainte, technologique, demande de la part de ceux qui développent des outils de développement une exceptionnelle flexibilité, des moyens financiers très importants, une capacité presque visionnaire à anticiper les évolutions à venir, ou mieux, la capacité de concevoir des outils selon une architecture « qui défie le temps ». Sur ce dernier point, l'avantage est aux outils dont le noyau dur n'est pas une technologie, mais un langage. Nous détaillerons ce point plus loin.
Ainsi, tout environnement de développement est pris en tenaille entre une évolution méthodologique, lente mais lourde, et une évolution de la technologie, de plus en plus rapide et difficile à maîtriser.
Il est important de remarquer que, pour beaucoup d'outils de développement, l'adaptation aux nouvelles technologies n'est pas qu'une contrainte. C'est aussi souvent leur raison d'être. C'est par ce qu'il est difficile de développer le dialogue client-serveur que les outils de développement client-serveur ont vu le jour. C'est par ce qu'il est délicat de programmer des applications utilisant un moniteur transactionnel que certains outils permettent d'exécuter des verbes transactionnels sans connaissance spécifique de cette technologie de la part du programmeur.

Evolution des outils dans le temps

Les outils évoluent un peu comme les espèces vivantes, par mutations et sélections. Comme dans le monde vivant, plusieurs stratégies d'évolution sont possibles. Certains outils se logent dans une niche écologique et répondent très efficacement à des besoins technologiques ciblés : le client-serveur ou les applications de type « intranet », pour ne citer que les évolutions les plus récentes (ceci ne signifie en aucun cas que tous les outils client-serveur sur le marché doivent être vus comme des outils de niche). D'autres outils partent à la conquête d'un autre milieu, et tentent une percée vers une technologie qu'ils n'adressaient pas jusqu'alors. Par exemple, Delphi et Visual Basic étaient, à l'origine, des outils permettant de construire des applications sur PC. Ils sont venus progressivement au monde du client-serveur.
Examiner un atelier « de l'intérieur » est également le meilleur moyen d'en avoir une vision prospective. Dans un monde en continuelle évolution, rien n'est plus difficile que de savoir ce que vaudra un atelier de développement dans les deux ans à venir. Or, un outil « généraliste » qui a lentement encaissé plusieurs révolutions technologiques (les SGBD relationnels, les environnements multi-fentêtrés, le client-serveur, Internet) aura plus de chance d'encaisser la prochaine révolution que l'outil spécialisé dans une technologie nouvelle, expressément bâti pour répondre aux difficultés particulières à cette technologie.

Le stress du décideur

Il y a donc trois raisons qui rendent le choix d'un environnement de développement difficile.
Tout d'abord, il s'agit, comme nous l'avons vu, d'une décision stratégique, qui engage l'avenir de l'organisation utilisatrice, et non uniquement quelques développeurs du département études.
D'autre part, avec l'acquisition d'un nouvel outil de développement, l'entreprise s'engage pour longtemps. On objectera que, de nos jours, les applications ont une durée de vie de plus en plus courte, et les outils aussi. Certes. Mais lorsqu'une entreprise a acquis un outil pour un développement particulier, il y a des chances pour que cet outil serve à des développements ultérieurs. La formation des développeurs est un investissement lourd. Pourquoi les former tous les cinq ans à un nouvel outil ? D'autre part, conserver la même gamme d'outil favorise la réutilisation des modules déjà développés. Acquérir un outil, c'est donc s'engager pour vingt ans. Affirmation gratuite ? il suffit de remarquer que nombre d'organisations programment encore en COBOL 85 voire en COBOL 74.
Enfin, pour analyser les besoins d'une organisation en outillage de développement il est nécessaire de tenir compte d'un très grand nombre de facteurs, à la fois méthodologiques, technologiques, organisationnels et humains. Les check-lists et grilles de dépouillement permettant de sélectionner un outil comportent plusieurs centaines de lignes. Ceci rend difficile un comparatif point par point. De plus, la très grande variété d'outils de développement sur le marché rend cette comparaison inapplicable. En effet, une fonction qui fait tout l'attrait d'un outil (par exemple, sa possibilité de

générer des programmes transactionnels sans connaissances du moniteur transactionnel sous-jacent) peut être totalement absente d'un autre outil, qui possède, en revanche, une autre caractéristique intéressante (comme de s'adapter à des SGBD non standard). Faire une moyenne pondérée de ces caractéristiques est alors sans grand intérêt.

Les cinq erreurs à éviter

Pour choisir un environnement de développement, il n'y a pas de voie royale. Rappelons simplement une évidence : un outil de développement est un progiciel. La technique de choix d'un atelier est donc, dans ses grandes lignes, analogue à toute méthode classique de sélection de progiciel. Pour faire un choix optimal, il est nécessaire d'en suivre les étapes : étudier l'existant de l'entreprise, examiner l'organisation du développement, définir précisément les besoins, déterminer les axes d'évolution, présélectionner, puis affiner progressivement cette sélection.
Tout au long de ce processus de sélection (et à plus forte raison, est-il besoin de le dire, si la sélection se fait sans démarche rigoureuse) cinq erreurs reviennent fréquemment :

- confondre développement et programmation,
- négliger le langage,
- penser à court terme,
- succomber au charme de la « fée Démo »,
- être seul face au choix.

Ces erreurs perturbent le processus du choix et augmentent inutilement le stress des décideurs, alors qu'elles auraient pu être facilement évitées.
Dans les paragraphes qui vont suivre, nous allons examiner en quoi consistent ces erreurs et comment les éviter.

Erreur 1 : Confondre « développement » et « programmation »

La programmation est, bien entendu, une phase importante du processus de développement, mais ce n'est pas la seule. Développer, c'est aussi définir les besoins, analyser, concevoir, tester, valider, maintenir. Or, il y a une confusion permanente des termes, tant chez les éditeurs que chez leurs clients. L'entreprise a besoin d'un environnement de développement. Les développeurs se focalisent surtout sur un outil de programmation.
Il est donc important de chercher, pour chaque outil, ses potentialités relatives aux différentes phases du cycle de vie : analyse et conception, gestion de configuration, référentiel partagé avec contrôle des accès, etc. Si ces fonctions ne sont pas intégrées à l'outil, elles doivent pouvoir être obtenues par interfaçage avec des outils du marché.

Erreur 2 : Négliger le langage de programmation

Le langage de programmation est très souvent négligé lors du choix d'un outil de développement. Or, il s'agit là du principal outil du développeur. Sur le plan fonctionnel, il devrait être le premier critère de choix. Ceci peut sembler contradictoire avec ce qui a été dit précédemment, mais il n'en est rien. En effet :

- la programmation occupe 15 % de la charge de développement, ce qui n'est tout de même pas négligeable,
- un bon langage réduit l'effort de maintenance à venir,
- s'il est de haut niveau, il sera utilisé, non seulement lors de la phase de programmation, mais également lors des phases de conception en amont,
- le langage, c'est ce qui restera dans cinq ans, lorsque l'outil aura évolué sur le plan méthodologique et technologique.

Il est donc indispensable d'examiner le langage de programmation à fond : syntaxe, caractéristiques particulières, degré de normalisation. Et surtout, n'accorder que peu d'importance aux qualificatifs utilisés par les éditeurs, comme « puissant » ou « de quatrième génération », sans signification tangible.

Erreur 3 : se focaliser sur le court terme

L' acquisition d' un outil engage une équipe de développement pour plusieurs années. Il est donc important de prévoir, non seulement les besoins immédiats, mais aussi les implications à long terme. En tout état de cause, un outil ne sera pleinement rentable que plusieurs mois (parfois un an ou deux) après sa mise en œuvre. Ce temps de latence et d'apprentissage sera d'autant plus long que l'outil est sophistiqué. L'outil miracle, qui s'apprend sur un coin de table et qui multiplie par dix la productivité des équipes n'existe pas, surtout s'il est question d'un outil stratégique au niveau entreprise.

Erreur 4 : Se décider à la vue d'une démonstration

Assister à des démonstrations de produits, sur les stands des salons ou chez les éditeurs, donne un bon aperçu des outils, et apporte des informations intéressantes. Les experts y assistent souvent, car c'est une source d'information pertinente.

Cependant, si cette démarche est utile, elle est loin d'être suffisante. Elle peut même avoir des effets catastrophiques sur le choix de l'outil.

Une démonstration permet de se faire une idée d' un produit et d' avoir une image immédiate de ses caractéristiques les plus saillantes. En aucun cas de se faire une idée précise de ses potentialités.

Un conseil : profitez des démonstrations si vous pensez que cela peut vous être utile, mais, ne vous y fiez pas trop. Et si vous assistez à une démonstration, assistez à plusieurs autres, de manière à avoir un référentiel de comparaison.

Erreur 5 : être seul face au choix

La complexité des outils actuels est telle, qu'elle dépasse largement les compétences d' un seul expert.

Pour pouvoir juger, il est nécessaire de confronter les avis de personnes de métiers et de profils divers. Il est en particulier important de donner la parole à ceux qui vont utiliser l'outil : développeurs et chefs de projet techniques. Mais également à l'architecte, au responsable des méthodes, et, bien entendu, au responsable des études. Lors d'une consultation, il n'est pas rare que les avis d'une douzaine de personnes soient pris en compte. Dans un tel cas de figure, un consultant externe n'est pas uniquement un expert, mais également un animateur et un médiateur.

Le non-choix

Le non-choix est la réponse rampante face à la difficulté de la décision. C'est aussi la meilleure façon de voir proliférer dans l'entreprise une kyrielle d'outils inadaptés au besoin, incompatibles entre eux, et qui finissent par coûter très cher.

Pour l'éviter, trois conseils de bon sens : considérer le choix d'un environnement comme stratégique et en informer toutes les personnes concernées ; se fixer un calendrier à la fois réaliste et précis ; suivre, comme pour tout type de progiciel, une démarche de choix rigoureuse .

En règle générale, une fois prise la décision de changer, trois à quatre mois suffisent pour finaliser le choix. C'est souvent une période difficile. Mais c'est également l'occasion, au travers du processus de choix, de découvrir la maturité et les potentialités de l'entreprise en termes de développement de logiciel, pour en tirer profit dans l'avenir.

Yves CONSTANTINIDIS

Yves Constantinidis

Consultant à BSGL Conseil. Il intervient auprès des grandes entreprises, en France et à l'étranger. Il anime de nombreux séminaires sur le sujet des outils de développement. Il est auteur de l'ouvrage « Outils de construction du logiciel » paru aux éditions Hermès.