Unité Mixte de Recherche 5104 CNRS - INPG - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

Eléments pour le choix de méthodes de développement de systèmes logiciels critiques

Paul CASPI

Rapport de recherche Verimag n^o TR-2005-17

Novembre 2005

Les rapports peuvent être téléchargés à l'adresse suivante
http://www-verimag.imag.fr

Eléments pour le choix de méthodes de développement de systèmes logiciels critiques

Paul CASPI

VERIMAG-CNRS

Novembre 2005

Résumé

Ce rapport a été établi à la demande de la direction générale de la RATP pour la conseiller dans le choix de méthodes de développement de systèmes logiciels critiques tels que mis en œuvre dans des postes de manœuvre informatisés ou des métros sans conducteurs. Il se compose de deux parties : la première partie se consacre à une description des diverses méthodes qui ont été proposées et mises en œuvre pour résoudre le problème des systèmes logiciels critiques. La seconde en déduit quelques critères de choix dans le domaine.

Abstract

This report has been written on request of the Deputy Chief Executive Officer of RATP, the Paris Subway Authory, so as to assist this authority in choosing development methods for software-based safety-critical systems, such as found in computerised signalling systems and driverless subways. It is made of two sections: the first one is devoted to describing the various methods that have been proposed for solving the problems raised by theses critical software systems. The second section extracts from the previous one some criteria of choice in this field.

Mots-clé :

Relecteurs : Nicolas HALBWACHS, Yassine LAKHNECH, Oded MALER, Marie-Laure POTET, Joseph SIFAKIS, Stavros TRIPAKIS

Notes:

Comment citer ce rapport :

@techreport { ,
title = { Eléments pour le choix de méthodes de développement de systèmes logiciels critiques},
authors = { Paul CASPI},
institution = { Rapport de recherche Verimag },
number = {TR-2005-17},
year = { 2005},
note = { }
}

Préambule

Ce rapport a été établi à la demande de la direction générale de la RATP pour la conseiller dans le choix de méthodes de développement de systèmes logiciels critiques tels que mis en œuvre dans des postes de manœuvre informatisés ou des métros sans conducteurs.

Commandé en janvier 2005, il a été fourni en février 2005 puis a fait l'objet de deux présentations devant des publics différents de la RATP. Il faut noter qu'un rapport de même nature a été aussi commandé à Jean-Claude LAPRIE du LAAS-CNRS[1] qui a parcouru les mêmes étapes. Il est resté confidentiel jusqu'en octobre 2005, lorsque nous avons reçu l'autorisation de publier nos rapports. Ce document fait donc suite à cette autorisation. Néanmoins, je me suis autorisé quelques modifications mineures par rapport au rapport initial, modifications issues des remarques que j'ai reçues aux cours des présentations. Autant que faire ce peut, j'ai signalé ces modifications par des notes de bas de page.

Il se compose de deux parties : la première partie se consacre à une description des diverses méthodes qui ont été proposées et mises en œuvre pour résoudre le problème des systèmes logiciels critiques. La seconde en déduit quelques critères de choix dans le domaine. Des questions supplémentaires nous ayant été posées, on trouvera en annexe les réponses correspondantes.

Ce rapport est court et peu technique : cela correspond à la demande qui nous avait été faite, notamment qu'il soit compréhensible par un lecteur scientifique mais non spécialiste. D'autre part, il ne comporte pas de bibliographie. Il y a deux raisons à cela : premièrement, compte-tenu de la variété des sujets abordés, une bibliographie eut été à la fois très volumineuse et forcément incomplète. Deuxièmement, la bibliographie n'est en l'occurrence que la partie émergente de la matière traitée. Beaucoup de ce que je dis ici est du matériel non publié issu d'expériences et de réflexions personnelles :

- Les considérations sur les coûts de développement et de validation sont difficiles à obtenir, car cachés derrière des considérations de secret industriel. Les données que je fournis ici sont issues d'un cours que j'avais monté avec Eric PILAUD qui avait dirigé le projet SPIN de protection intégrée numérique (surveillance et arrêt d'urgence de centrales nucléaires) chez MERLIN GERIN.
- Les considérations sur SACEM, MAGGALY et METEOR viennent de ma participation à la commission SACEM du Ministère des transports et au conseil scientifique du SYTRAL pour l'extension de la ligne D du métro de Lyon.
- Les considérations sur AIRBUS viennent de ma participation au développement de LUSTRE/SCADE, utilisé dans l'avionique ainsi qu'à divers contrats et études que j'ai faites sur le sujet.
- Les considérations sur les normes et la certification viennent, outre les commissions déjà citées, de ma participation comme évaluateur à l'association de certification ferroviaire CERTIFER.
- Enfin, beaucoup de remarques viennent d'avoir pratiqué personnellement un certain nombre d'outils utilisés dans le domaine, outils de preuve, de vérification par modèles, outils de développement et de programmation.

[1] Laboratoire d'analyse et d'architecture des systèmes de Toulouse.

Table des matières

1 Méthodes de développement de systèmes logiciels critiques

1.1 Fiabilité et redondance logicielles

Ces méthodes consistent, en quelque sorte, à tenter de résoudre le problème comme cela a été fait pour le matériel : on mesure la fiabilité d'un système et, si elle est insuffisante, on l'augmente grâce à des techniques de redondance. Cette approche se heurte néanmoins à plusieurs difficultés :

- Contrairement à ce que l'on pourrait croire, il n'y a pas de difficulté majeure pour définir rigoureusement la fiabilité du logiciel, même si les phénomènes impliqués (erreurs de conception) sont différents de ceux impliqués dans le cas du matériel (pannes). En effet, l'événement « une erreur se manifeste dans un système logiciel en service » peut être facilement considéré comme produit par un processus stochastique ponctuel et probabilisé. Mais cette fiabilité est beaucoup plus difficile à mesurer que pour le matériel. En effet, on ne dispose pas de l'effet statistique de masse qu'il y a pour le matériel, surtout s'il s'agit de systèmes de grande qualité qui vont générer très peu d'erreurs, donc des statistiques très peu significatives et des temps d'expérience exorbitants. Songez au temps nécessaire pour garantir, expérimentalement un taux d'erreur inférieur à 10^{-9} par heure.
- Quelles que soient ces difficultés, on aurait pu penser que, comme pour le matériel, des méthodes de redondance permettraient d'augmenter ces fiabilités et compenser ainsi ces mauvaises mesures. Là aussi, il a fallu déchanter : on sait, en effet, que l'indépendance est une condition indispensable pour que les redondances soient opérantes. Or, si on demande à différentes équipes de réaliser le même logiciel, de façon indépendante, avec l'espoir que les erreurs produites seront, elles aussi, indépendantes, il s'avère que cet espoir est, en général, déçu. La raison en est qu'il y a une cause commune aux erreurs, qui est la spécification : les équipes se tromperont aux mêmes endroits, là où la spécification est plus difficile à prendre en compte.
  Ainsi, non seulement on sait mal mesurer la fiabilité de logiciels individuels mais on ne sait pas, non plus, tenir compte, faute d'indépendance, des redondances.
- Enfin, les redondances logicielles sont coûteuses à mettre en œuvre. C'est déjà vrai lorsqu'il s'agit de la version de base mais ce l'est encore plus lorsqu'il s'agit de gérer la maintenance et les évolutions car il faut maintenir et gérer plusieurs équipes dans la durée.

Ces difficultés pourraient paraître rédhibitoires. Cela n'a pas empêché les redondances logicielles d'être utilisées avec succès comme techniques d'appoint, en conjonction, notamment, avec la redondance matérielle. Ainsi, AIRBUS utilise des doubles chaînes de commande, l'une de commande et l'autre de surveillance, avec des programmations légèrement dissymétriques qui contribuent à la sécurité du logiciel. AIRBUS utilise aussi la redondance naturelle apportée par le fait que les gouvernes des avions sont redondantes. Les logiciels qui les contrôlent sont donc, eux aussi, redondants.[2]

[2] On voit ici deux usages différents de la redondance : la redondance commande-surveillance vise à empêcher les erreurs d'agir sur l'extérieur ; c'est une redondance pour la sécurité. La redondance de gouverne vise, elle, à pouvoir continuer de piloter l'avion. C'est une redondance pour la disponibilité.

1.2 Méthodes traditionnelles de développement rigoureux

Mais ces techniques restent des techniques d'appoint qui ne semblent pas suffisantes pour assurer, à elles seules, une sécurité suffisante, au point que certains fiabilistes ont pensé qu'il n'était pas possible de réaliser des fonctions critiques par logiciel et que ce fut une surprise pour eux lorsqu'apparurent des mises en service réussies de systèmes logiciels critiques.

Sur quoi se fondaient ces succès ? Essentiellement sur des méthodes de développement rigoureuses avec une organisation et des étapes bien codifiées.

- Les étapes s'organisent en phase descendante : analyse système[3] spécification fonctionnelle, spécification détaillée, codage, et en phase ascendante : tests unitaires, test d'intégration, tests fonctionnels.
- L'organisation est divisée en équipe de développement et équipe de validation, le plus indépendantes possibles.[4] L'équipe de développement réalise les productions de la phase descendante. L'équipe de validation relit et critique les productions de la phase descendante, élabore les tests et les exécute pendant la phase ascendante.
- Des protocoles organisent et formalisent les relations et la traçabilité entre étapes (documents de traçabilité, réunions et rapports de clôture d'étapes) et entre équipes (fiches d'anomalie, et clôture de fiches d'anomalie).

Si ce processus a connu des succès, il n'en est pas moins long et coûteux. On estime que la phase de validation coûte, en moyenne, au moins deux fois le coût de la phase de développement. D'autre part, c'est un processus itératif qui pourrait très bien ne pas converger. Par exemple, les projets SACEM (RER ligne A) et MAGGALY (ligne D du métro de Lyon) ont pris du retard pour des raisons de validation.

Néanmoins, cette approche a fait ses preuves et a permis à certains industriels expérimentés de produire des systèmes critiques satisfaisants, dans une gamme de complexité donnée.[5] Bien que nous ne disposions pas de statistiques précises sur le sujet, l'honnêteté oblige à concéder que cette approche reste, en dépit des réserves d'universitaires adeptes des méthodes formelles (dont l'auteur de ce rapport fait d'ailleurs partie), la méthode semble-t-il la plus utilisée dans le domaine.

1.3 Construction prouvée de programmes et de systèmes

Comme nous l'avons dit, les universitaires ont des réserves vis-à-vis du processus décrit ci-dessus. Est-ce ainsi que l'on construit un pont ? se demandent-ils, par essais, tests et correction d'erreurs ? Pour eux, l'informatique en est à un stade pré-industriel, comme au Moyen-Age, lorsqu'un pont sur deux s'effondrait en cours de construction.

C'est pourquoi, très tôt, les chercheurs ont essayé de systématiser le processus de construction de systèmes informatiques et de le rendre scientifique. La sémantique axiomatique des langages a été un des beaux succès de cet effort. Elle permet de transformer un programme informatique en un objet de logique formelle sur lequel on peut donc raisonner.

[3] Ajout d'octobre 2005.

[4] Certains ont même préconisé un isolement organisationnel et géographique entre ces équipes.

[5] Qui, d'ailleurs, dépend peut-être de l'industriel considéré.

Cependant la logique est difficile, plus difficile que les calculs de ponts. On dit même qu'elle est (à partir d'un certain niveau) indécidable, c'est-à-dire qu'elle n'est pas complètement accessible à une intelligence mécanique et nécessite impérativement une intervention humaine. Lorsque on veut prouver un gros programme, l'effort est en général trop important et la seule solution est donc d'essayer de découper la preuve en petites étapes. Cette idée a conduit aux méthodes de développement prouvé de programmes, consistant à construire par raffinements successifs, simultanément le programme et sa preuve.

Ces méthodes, préconisées depuis longtemps, ont néanmoins rencontré peu de succès bien qu'un résultat significatif ait été atteint avec la méthode B[6] sur le système METEOR. Les raisons de ces difficultés sont les suivantes :

- Ces méthodes sont encore plus onéreuses que les précédentes. On a déjà souligné combien la logique est difficile. De plus, elle ne se satisfait pas d'informaticiens ordinaires et exige du personnel spécialement formé, ce qui augmente encore le coût, notamment d'entrée dans cette technologie.
- D'autre part, ces méthodes ont, initialement, été mises au point pour des applications purement informatiques. Ce n'est en général pas le cas des applications critiques de type RATP, qui sont des applications de contrôle-commande dans lesquelles des systèmes informatiques interagissent avec des environnements physiques (trains, aiguilles, ...) et humains (conducteurs, aiguilleurs, ...). Dans ces cas, on ne peut prouver le système informatique que vis-à-vis de ces environnements et cela exige que le système logique soit à même de rendre compte, à la fois, des programmes et de leur environnement. C'est donc encore plus difficile. Il faut noter que la méthode B utilisée pour METEOR n'avait pas atteint ce stade de développement, appelé depuis « B Système ». De ce fait, la preuve de METEOR n'était que partielle. Ainsi, le succès de METEOR doit sans doute beaucoup à B, mais aussi au fait que l'aspect système avait bénéficié de l'expérience précédente de MAGGALY, qui, on s'en souvient, avait été développé avec une méthode classique.
- Enfin, au même titre que la méthode traditionnelle, et plus encore, la convergence des raffinements successifs n'est pas garantie. C'est dire combien compte ici l'expérience de l'industriel concernant le système à concevoir.

Il n'en reste pas moins, malgré ces difficultés, que cette approche est susceptible de procurer une plus grande confiance que les méthodes plus empiriques évoquées précédemment (sinon une confiance absolue qui ne saurait exister dans une activité humaine) dans les systèmes ainsi conçus.

1.4 Méthodes formelles partielles

Ces difficultés ont conduit universitaires et industriels à étudier des méthodes formelles partielles, s'adressant à et protégeant contre des types d'erreurs particuliers, susceptibles de nuire à la sécurité des systèmes. Ces méthodes peuvent s'appliquer, en particulier, en complément des méthodes traditionnelles. Les principales sont :

[6] Développée par Jean-Raymond ABRIAL (ajout d'octobre 2005).

1.4.1 Vérification par modèle

L'idée qui a présidé à la naissance de cette technique[7] a été de se restreindre à des fragments de la logique qui soient décidables, c'est-à-dire, dont la résolution pouvait être entièrement mécanisée. Il se trouve que, d'une part, certains systèmes dits d'état fini comme les systèmes composés uniquement de variables binaires en nombre fini, peuvent être entièrement traités de cette façon. D'autre part, même pour les systèmes plus complexes qui ne sont pas de cette nature, certaines propriétés importantes peuvent tout de même être traitées ainsi.

Dans ces cas d'application, les avantages sont importants puisque ici la méthode est automatique et ne nécessite donc pas le lourd investissement humain réclamé par des méthodes à la B. D'autre part, lorsque la preuve que l'on cherche à faire est fausse, cette méthode fournit des contre-exemples, c'est-à-dire des comportements du système qui violent la propriété désirée, que l'on peut donc analyser et qui permettent souvent soit de corriger le système, soit la propriété.[8][9][10] Il faut cependant ne pas croire que la méthode est exempte de difficultés : même d'état fini, les systèmes peuvent comporter un ensemble d'état extrêmement complexe. C'est le phénomène d'*explosion combinatoire* qui limite la méthode. Encore ici, l'expérience du type de système que l'on veut construire et la connaissance de sa complexité est un atout majeur.

1.4.2 Interprétation abstraite

Une autre idée[11] pour contourner les difficultés de la logique a été de chercher à l'approximer de façon à la fois décidable et conservative. Si on y parvient, on obtient une méthode de vérification à la fois sûre (si elle dit que le système est bon, alors il l'est ; si elle dit que le système est mauvais, il est peut-être bon (alarme à tort)) et automatique. Mais les choses sont plus complexes qu'il n'y paraît : l'indécidabilité est une propriété intrinsèque ; tout dépend donc de la qualité de l'approximation utilisée ; si elle est trop grossière, on risque d'être submergé par les alarmes à tort et on ne peut décider. Si elle est trop fine, elle risque de ne pas converger.

Le problème de cette approche est donc de trouver la bonne approximation, ce qui est une tâche aussi difficile que de faire des preuves interactives. Cette méthode a donc trouvé un autre cadre d'application à la vérification de propriétés génériques que l'on va trouver dans beaucoup d'applications très diverses et pour lesquelles on pourra étudier, une fois pour toute, des approximations efficaces. Les plus connues de ces propriétés sont les propriétés d'absence d'erreurs à l'exécution de type débordement de nombre, division par zéro, débordement de mémoire, indice de tableau hors limite, calcul d'adresse erroné, etc., erreurs qui ont été popularisées par le fameux échec du vol inaugural d'ARIANE V. Cet échec a d'ailleurs conduit aux premières mises en œuvre industrielles d'outils d'interprétation abstraite qui connaissent un succès croissant.

[7] Due en particulier à Joseph SIFAKIS de VERIMAG.

[8] Ajout d'octobre 2005.

[9] C'est un avantage par rapport à la preuve interactive : dans celle-ci, lorsque on n'arrive pas à prouver quelque chose, on est beaucoup plus désarmé.

[10] La difficulté de la logique frappe ici aussi : l'expérience montre que lorsqu'une preuve échoue, c'est plus souvent la propriété qui est mal exprimée que le système n'est incorrect.

[11] Due en particulier à Patrick COUSOT de l'Ecole normale supérieure.

Il est important de noter cependant que l'usage d'une méthode telle que B garantit par construction l'absence de ce type d'erreur et rend donc inutile l'emploi de tels outils.

1.4.3 Génération automatique de code

L'idée ici est que, dans des domaines de métiers particuliers comme l'automatique, on peut construire des formalismes aptes à capturer directement les spécifications détaillées des systèmes et susceptibles d'être compilés directement en code sans passer par l'étape de codage manuel. Cette technique, qui évite donc une source d'erreurs importante, a, en outre, l'intérêt de proposer un formalisme métier qui facilite beaucoup la communication au sein des organisations développant et validant les systèmes. C'est dans cette optique que se situe l'atelier SCADE[12] utilisé entre autres chez AIRBUS pour développer les systèmes de commandes de vol.

1.5 Synthèse

1.5.1 Les méthodes ne sont pas exclusives

Si l'on regarde les exemples que nous avons cités, on trouve des situations très diverses, méthode traditionnelle pour MAGGALY, méthode B pour METEOR (mais pas B système et s'appuyant de l'expérience MAGGALY). L'exemple AIRBUS est intéressant car il présente un large spectre de méthodes : la méthode de base est traditionnelle mais elle s'appuie sur SCADE et la génération automatique de code. De plus, SCADE intègre un vérifieur par modèle permettant au concepteur de faire des preuves partielles. Le code produit est ensuite inspecté par interprétation abstraite pour détecter d'éventuelles erreurs à l'exécution. Enfin, on a vu qu'AIRBUS fait appel à de la redondance logicielle.

1.5.2 Elles partagent un fond technologique commun

Si on regarde la structure des logiciels produits par les diverses méthodes, il est frappant de constater qu'elles partagent souvent un fond commun :

- Le programme est cyclique, activé par une horloge temps-réel périodique.
- Le corps du programme est acyclique ou ne comporte que des boucles statiquement bornées, de sorte que son temps d'exécution peut être relativement[13] facilement borné.
- La mémoire nécessaire à l'exécution du corps de programme est statiquement connue (pas d'allocation dynamique).

Ainsi s'est dégagé un certain consensus sur la structure à laquelle doit obéir un logiciel de contrôle-commande critique et ce consensus est largement repris par les diverses normes en vigueur dans les différents domaines d'application. La simplicité de cette structure logicielle, le fait qu'elle produit des logiciels maîtrisables, testables, plus facilement prouvables, n'est pas pour rien dans les succès dont on a fait mention précédemment.

[12] Fondé sur le langage LUSTRE développé à VERIMAG.

[13] L'arrivée sur le marché de processeurs modernes et performants rend cependant cette tâche de plus en plus difficile.

A partir de cette structure de base, des évolutions sont certes possibles (il ne faut pas figer la technologie) mais il faut, chaque fois, montrer que ces évolutions (par exemple le multi-périodique pré-emptif) ne contrarient pas la maîtrise du logiciel obtenue précédemment.

1.5.3 Des trous méthodologiques subsistent

En revanche, il ne semble pas y avoir de consensus concernant les aspects souvent rencontrés d'informatique répartie, lorsque les divers calculateurs intervenant dans le contrôle-commande doivent se coordonner et échanger de l'information. Quels doivent être les systèmes matériels permettant ces échanges, quels protocoles faut-il utiliser, comment les communications s'intègrent-elles dans la structure de chaque programme, comment maîtrise-t-on le comportement d'ensemble, voilà le genre de questions sur lesquelles il y a peu de réponses. Cet état de fait est très grave car il faut bien comprendre qu'un système réparti est considérablement plus complexe qu'un système centralisé. Par exemple, bien des difficultés rencontrées dans MAGGALY étaient dues à cet aspect de répartition.

D'autre part, il faut bien remarquer que les normes sont en général muettes sur ce point, tant il est vrai que les normes ne peuvent appréhender que les phénomènes bien connus et sur lesquels existe un certain consensus. Donc, les normes en général ne procurent pas toujours une protection efficace. De même les agences de certification ont tendance à mettre l'accent sur les aspects qu'elles savent examiner et vérifier, au détriment de ceux qu'elles maîtrisent moins et qui peuvent être tout aussi importants et, parfois, même plus.

2 Eléments pour le choix d'une méthode

De la discussion précédente se dégage l'impression que toutes les méthodes ont des avantages et des inconvénients et donc que le choix est difficile. C'est vrai. D'autre part, les critères de choix peuvent être très divers et englober des aspects très difficiles à maîtriser comme par exemple des éléments de stratégie industrielle, d'hétérogénéité de parc, etc. Nous nous bornons ici à attirer l'attention des décideurs sur des éléments de risque qui nous paraissent essentiels pour le succès d'un projet de système critique.

2.1 La compétence et l'expérience de l'industriel

Quelle que soit la méthode choisie, mener à bien un projet de système critique ne s'improvise pas et exige une organisation et un personnel compétents et expérimentés. C'est vrai évidemment si on se propose d'employer une ou des méthodes formelles mais ce l'est aussi avec des approches traditionnelles. Les métiers de testeur ou de relecteur de code ou d'analyste de sécurité sont de vrais métiers qui sont rares, difficiles et ne s'improvisent pas. Il est important que l'industriel démontre cette compétence et cette expérience.

2.2 La nouveauté et la complexité du problème

Comme nous l'avons vu, quelle que soit la méthode employée, la convergence n'est pas assurée. On peut raisonnablement penser que, si un industriel a su traiter de façon satisfaisante un problème ou une classe de problèmes, il saura traiter de même un problème similaire. En revanche, il est beaucoup plus difficile de s'en convaincre si le problème est entièrement nouveau et si on ne dispose pas d'éléments de comparaison. Dans ces cas, le succès n'est pas assuré. Il faut aussi prendre garde à la complexité. Dans le domaine l'extrapolation est difficile. Si un industriel a su traiter un problème de taille n, il est probable qu'il saura traiter un problème de taille $n + 1$. Mais, que peut-on dire de $2n$? La complexité est souvent exponentielle et, souvent, les différences quantitatives peuvent devenir qualitatives. Ainsi, un problème plus complexe peut apparaître comme un nouveau problème. Parmi les facteurs de complexité, l'aspect de répartition est extrêmement important et il faut y accorder le plus grand soin.

Il est donc très important de s'assurer que l'industriel est à même de maîtriser la complexité du problème, notamment concernant les aspects de répartition.

2.3 La validité de la proposition financière

Développer un système critique coûte cher, même si certaines méthodes sont plus coûteuses que d'autres. On peut estimer qu'il y a un minimum incompressible en deçà du quel on ne peut descendre sauf à dégrader la sécurité du système. En particulier, on estime en général que le coût d'un logiciel critique est au moins trois fois le coût d'un logiciel standard et que, dans la méthode traditionnelle, le coût de la vérification est au moins deux fois celui de la conception (ces deux

estimations sont donc consistantes).[14]

Il est donc très important de vérifier que l'industriel a respecté ces canons en attribuant aux tâches de validation un budget conséquent. Mais, évidemment, ce facteur doit être corrélé avec les aspects de compétence demandés ci-dessus.

2.4 La qualité de la technologie proposée

Nous avons vu aussi qu'il existe un fond technologique commun fondé à la fois sur la raison et l'expérience, dont on ne se départ pas sans risques importants. Il est donc très important d'examiner les déviations proposées par rapport à ce modèle et leurs justifications. Il faut en particulier examiner très soigneusement les aspects essentiels de la technologie de répartition et l'argumentaire qui devrait, à notre avis, obligatoirement l'accompagner tant que les normes ne se sont pas stabilisées sur ce point.

[14] En revanche, il faut remarquer que la technologie logicielle utilisée peut constituer un facteur d'économie important : on considère en effet que la productivité d'un concepteur est constante quelle que soit le niveau de technologie utilisée. Donc, produire une ligne d'un langage de haut niveau ne coûte pas plus cher que de produire une ligne d'un langage de bas niveau. Il y a donc intérêt à utiliser la meilleure technologie possible et c'est de là, entre autre que vient l'intérêt des méthodes par génération automatique de code à partir de modèles de haut niveau.

A Compléments de réponses

1 - Compte tenu de la complexité et de la nécessaire répartition des traitements dans un projet d'automatisation intégrale du mouvement des trains, la difficulté de convergence des tests nécessaires aux "méthodes traditionnelles" permet-elle d'accepter leur utilisation ?

C'est une question difficile. Le seul élément de réponse que je puisse apporter est de se fonder sur l'expérience du soumissionnaire en la matière. A défaut d'expérience comparable, j'aurais personnellement tendance à refuser pour ne pas courir de risque, tout en ayant conscience du caractère quelque peu conservateur d'une telle attitude : si d'autres, précédemment n'avaient pas couru ces risques, peut-être d'ailleurs de façon inconsciente, sans doute le progrès eut été moindre.

2 - Comment appréciez-vous la tentation de faire des impasses (coûts, délais) dans certaines situations de mises au point difficiles et quels sont alors les éléments de traçabilité qui permettent la détection de ces difficultés et la mesure de la sensibilité aux déviations qui pourraient subsister ?

Cette tentation existe certainement. L'expérience montre toutefois qu'elle est limitée. L'honnêteté des acteurs, le fait que leur responsabilité soit engagée, en sont sans doute la cause. Toujours est-il que des retards importants sont souvent constatés dans des projets d'automatisation critique, qui attestent que les acteurs sont plutôt honnêtes et savent résister à la pression des délais.

D'autre part, il existe des moyens pour lutter, si besoin, contre cette tentation. Nous avons souligné, dans notre rapport, l'importance d'avoir un plan qualité soutenu par une organisation qualité qui soit indépendante hiérarchiquement de la direction du projet. D'autre part le maître d'ouvrage peut se doter lui-même (comme l'avait fait le SYTRAL à l'époque de l'extension de la ligne D de Lyon) de sa propre organisation de surveillance, qui vérifie, par exemple par des audits, la bonne exécution du plan qualité.

3 - Comment SECURISER (outils) la première étape de formalisation mathématique des spécifications pour les méthodes de construction prouvées ?

C'est une question importante d'autant plus que les lois de la logique sont implacables à ce sujet : si on part de spécifications inconsistantes, on peut facilement prouver n'importe quoi et, finalement, travailler beaucoup sans parvenir à un résultat valable.

Je n'ai pas de réponses précises à apporter à cette question, qu'il faut plutôt poser à ses promoteurs. Il me semble cependant que cette question a été étudiée, notamment par des gens de l'université de Besançon, qui ont proposé des méthodes d'animation de spécifications logiques, destinées à pallier la difficulté mentionnée ci-dessus.

4 - Avez-vous une appréciation des coûts de développement et de vérification concernant les méthodes formelles partielles par comparaison aux autres méthodes ?

Je n'ai malheureusement pas de réponse à cette question, qui exigerait certainement une étude importante, si tant est même qu'elle soit faisable. Le nombre limité d'expériences, le fait qu'elles s'adressent à des domaines différents font que les comparaisons en la matière risquent d'être peu significatives et qu'il sera difficile d'en tirer des conclusions. Ceci dit, si le projet auquel nous participons concernant la vérification par model-checking d'un poste de signalisation aboutit, vous aurez au moins des éléments de comparaison car vos services connaissent bien le coût de la vérification traditionnelle d'un poste.