Spécifications functionnelles

Christophe Delord ~ http://fun.cdsoft.fr

Dimanche 28 Février 2016

Ce qui ne va pas aujourd'hui

Les principaux problèmes dans la vie de tous les jours d'un ingénieur viennent souvent des spécifications logicielles. Le but de la méthode FUN de CDSoft est assez simple :

Les spécifications doivent être :	mais aussi :
• simples • pratiques • utilisables	• formelles • non ambigues • vérifiables • sûres

Qui suis-je ?

Je suis ingénieur en informatique. Je travaille en informatique et particulièrement dans l'industrie aéronautique depuis 17 ans... à écrire des logiciels mais pas toujours dans de bonnes conditions:

- deadlines incompatibles avec les exigences
- hypothèses insensées pour réduire les coûts
- et une philosophie contre-productive: *ça doit marcher du premier coup, pas de seconde chance pour faire mieux ou tester des solutions alternatives !*

J'ai ainsi accumulé quelques idées pour faire différement.
J'ai aussi une passion pour le logiciel libre et le partage des bonnes idées.
Pour en savoir plus sur moi: `http://cdsoft.fr`

Pourquoi ce livre ?

Ce livre sera une opportunité de:

- trouver du temps pour formaliser mes idées
- s'amuser avec des Arduino ou autres Raspberry Pi, ...
- montrer qu'écrire des logiciels ne nécessite pas toujours des processus, des méthodes et des outils compliqués

Je serais donc flatté si vous décidiez d'aider ce projet et de participer au financement de cette aventure.
J'espère pouvoir travailler à temps partiel pour avoir plus de temps pour ce projet.
En retour vous recevrez une version électronique du livre.

Les réponses de la méhode FUN

Mon point de vue est que la plupart des spécifications, même pour des systèmes critiques, sont écrites :

- dans des langues naturelles ambigues
- dans des formats propriétaires fermés inexploitables
- avec des suites de bureautique propriétaires dangereuses

Voyons ce qui ne va pas et ce qui peut être fait pour améliorer la situation. . .

Problème 1 : Langage naturel ambigu

Un langage naturel est pratique

- pour expliquer ou commenter des idées
- pour communiquer dans la vie de tous les jours.

Mais un logiciel critique (ou non) doit être décrit de manière non ambigue. Les **ambiguités** conduisent à des **interprétations multiples et contradictoires** en pratique.

Problème 1 : Langage naturel ambigu

- Les outils ne comprennent pas facilement le langage naturel.
- Parfois on extrait à la main l'information pour la dupliquer dans d'autres documents (conception, code, ...). Et les humains ne sont pas bons pour le copier/interpréter/coller...
- De tels projets contiennent de nombreux documents différents et contradictoires et l'implémentation peut ne plus suivre la spécification initiale.

Problème 2 : Formats propriétaires fermés

Vous ne pouvez pas maitriser un format propriétaire fermé:

- Le format peut changer à chaque nouvelle version.
- Les anciennes versions et les outils peuvent ne plus être supportés.
- Vous n'avez pas le code source et les anciennes versions seront perdues.

Un format propriétaire est le plus mauvais choix que l'on puisse faire pour écrire un document, en particulier lorsqu'il doit être conservé pendant longtemps (par exemple 80 ans dans l'industrie aéronautique)

Problème 3 : Formats inexploitables

Certains formats, propriétaires ou non, ne sont pas adaptés, en particulier les documents issus de suites bureautiques:

- Très souvent les gens ne comprennent pas la différence entre un éditeur de texte et un traitement de texte.
- Un traitement de texte est très bien pour écrire une lettre à sa grand-mère.
- Mais il enregistre des choses insensées dans le document pour décrire la mise en forme ce qui le rend inexploitable pour de nombreux outils.

Problème 4 : Suites bureautiques dangeureuses

- Vous ne savez pas ce que font les suites de bureautique dans votre dos.
- Confieriez-vous vos documents confidentiels à un pays susceptible d'espionner vos activités ?
- Beaucoup de sociétés donnent leurs documents à des suites bureautiques célèbres et fermées, en utilisant des ordinateurs connectés à internet. Pas de commentaire. . .

Proposition 1 : Langage formel

- Le langage naturel est toujours utilisé pour commenter et expliquer le modèle formel.
- Un langage fonctionnel avec des caractéristiques essentielles:
  - système de typage fort et statique,
  - inférence de type,
  - sémantique mathématique claire et déterminisme.
- Un tel langage est exécutable, déterministe et non ambigu.
- La spécification peut être réutilisée pour les activités ultérieures (conception, code, tests, . . .) grâce à une syntaxe claire et non ambigue.
- Plus d'outils et moins de bugs humains n'est pas une mauvaise chose.

Proposition 2 : formats ouverts

- Formats simples et ouverts :
  - Tous les documents sont écrits dans un format texte (généralement un format UTF-8).
  - Ce format est si simple et basique qu'il sera supporté pendant de nombreuses années.
- Ces formats et les outils associés sont généralement libres et gratuits.

Proposition 3 : formats ouverts (suite)

- Markdown est un format simple que n'importe quel bon éditeur peut lire et modifier.
- Il existe des outils pour convertir ces formats en HTML, PDF ou même dans des formats de suites bureautique mais le document source reste et restera toujours exploitable par des humains et des outils.
- Ces formats séparent clairement le fond et la forme. Il suffit de définir la forme une fois pour toutes et de la réutiliser dans tous les projets. Et de laisser les ingénieurs se concentrer sur le fond.

Proposition 4 : Outils sûrs

Pour écrire du texte, vous avez besoin d'un éditeur de texte :

- Chacun a son éditeur favori.
- La méthode FUN ne force personne à utiliser un éditeur particulier.
- Vous pouvez utiliser celui que vous maitrisez et qui est suffisamment sûr pour votre activité.
- Chacun est ainsi plus productif.

Spécifications exécutables

L'avantage d'écrire des spécifications **fun**ctionnelles est qu'elles sont exécutables. Cela signifie qu'elles peuvent être exécutées pour :

- mettre au point le modèle
- tester et valider le modèle
  - la logique du modèle peut ainsi être vérifiée avant même que la cible matérielle réelle ne soit disponible
- animer un modèle et présenter une maquette à votre client

C'est en quelque sorte la méthode formelle du pauvre...

Plans de tests exécutables

Le concept de specifications exécutables peut être appliqué aux tests !

Parler de plans de test *exécutables* peut sembler étrange puisque la raison d'être d'un test est d'être exécuté pour vérifier une propriété. . .

Il ne devrait pas être nécessaire de dire "plan de test *exécutable*".

Plans de tests exécutables

"Plans de test **exécutables**" n'est pas étrange. Cela devrait être naturel pour tout le monde... Mais les testeurs procèdent la plupart du temps de cette manière:

- écriture d'un plan de test
- implémentation des tests
- exécution des tests
- rédaction d'un rapport de test
- et de nombreuses erreurs de copier/coller à cause d'un process aussi long qu'ennuyeux!

Je vais montrer dans ce livre combien il est facile d'atteindre le même objectif avec des plans de test **fun**ctionnels qui sont auto-exécutables et qui génèrent les rapports de tests eux-mêmes.

Roadmap

La suite présente une roadmap prévisionnelle, un plan du livre.
Le but principal est de montrer comment Haskell (et la programmation **fun**ctionnelle en général) peut servir de langage de formalisation à plusieurs niveaux :

- modélisation / spécification
- simulation
- codage
- tests

Et être appliqué à différents domaines :

- logiciels temps réel et embarqués
- traitement de données (générateurs de code, . . .)

Roadmap / Programmation **fun**ctionnelle

Il sera utile de commencer par une introduction à la programmation fonctionnelle. Cette introduction sera très légère car il existe de nombreux cours et documentations sur le sujet.
Le livre insistera sur les points suivants et leurs intérêts :

- programmation fonctionnelle pure : pourquoi l'absence d'effets de bord est une bonne chose ?
- système de typage statique et fort : comment un système de typage peut être utilisé comme système formel pour décrire un logiciel ?

Roadmap / Quelques applications : formalisation du temps dans les systèmes temps réel (1/2)

Les systèmes temps réels de type réactif sont souvent des automates dont la principale activité est de modifier son état courant en fonction de stimuli externes.
La modélisation de tels systèmes dans un langage fonctionnel pur (sans effets de bord) peut sembler impossible mais nous verrons comment représenter le temps et les changements qu'il provoque sur un système réactif.
Cet exemple montrera les avantages d'un langage fonctionnel pur pour représenter des états, et surtout pour faire des raisonnements sur l'évolution des états dans le temps.

Roadmap / Quelques applications : formalisation du temps dans les systèmes temps réel (2/2)

Applications avec un modèle temps réel :

- générateur de notes de musique à partir de données issues de capteurs de mouvements
- formalisme inspiré de SCADE ou Matlab Simulink et d'applications temps réel embarquées

Roadmap / Quelques applications : Robot Arduino

Une autre modélisation de système réactif donnera lieu à une réalisation concrète : un robot arduino capable de percevoir son environnement et de réagir à quelques stimuli.

- spécification formelle en Haskell : traitement de données issues de capteurs
  - évitement d'obstacles
  - déplacement vers une cible
- simulation du robot dans un environnement (lui aussi simulé)
- implémentation sur un Arduino

Roadmap / Modélisation et Simulation

Modélisation d'un système complet : exemple classique du feux tricolore et de la circulation.

- modélisation d'un feu (machine à état, synchronisation avec l'environnement, ...)
- modélisation d'un usager de la route (comportement, heures de pointes, trajets domicile / travail, MTBF du véhicule, ...)
- modélisation d'un segment de route (vitesse maximale, capacité en nombre de voitures, ...)
- modèle global (réseau routier complet, population d'usagers)
- IHM pour visualiser la simulation, interagir, injecter des pannes, ...

Roadmap / Quelques applications : Une calculatrice bien spécifiée et testée

Ce projet est une réécriture d'une calculatrice écrite précédemment en Lua par CDSoft. L'idée est de réimplémenter cette calculatrice en Haskell proprement :

- spécification formelle en Haskell :
  - typage fort : la vérification de type est une forme de méthode formelle
  - utilisation à l'extrême du compilateur pour détecter les bugs potentiels et le code mort
- tests unitaires complets :
  - la non régression permet d'aborder les évolutions sereinement
  - mesure de la couverture du code par les tests

Roadmap / Quelques applications : Générateur de load ARINC 665

La norme ARINC 665 défini un standard pour le format de données chargeables sur un équipement. Cette norme est très utilisée en aéronautique.
La génération de ces loads est très critique puisqu'ils contiennent des exécutables et des données pour les calculateurs embarqués dans l'avion.
Une spécification **fun**ctionnelle permet de :

- réduire la distance entre les exigences et l'implémentation (traçabilité plus simple)
- tester de manière plus simple et plus sûre

L'idée clé est de décrire une exigence avec ses propriétés pour rendre l'implémentation directement testable à partir du formalisme de l'exigence.

Pour qui ?

À qui s'adresse ce livre et la méthode qu'il décrit ?
Cette méthode n'a ni la puissance de méthodes formelles comme Event-B ni la richesse (encore ?) d'outils comme SCADE ou Matlab Simulink. Ses intérêts sont :

- Apprentissage relativement simple
- Puissance d'un système de typage statique et fort
- Expressivité d'un langage fonctionnel
- Basée sur des outils et des langages libres, donc évolutive simplement et à moindre coût
- Multiplateforme, pas d'IDE imposé, compatible avec tous les bons gestionnaire de version (git, . . .)

Elle s'adresse donc à quiconque cherche un bon compromis entre puissance, efficacité, simplicité et coût.

Et maintenant . . .

Votre aide est la bienvenue. . .

- FUN est encore au stade de projet.
- J'ai besoin de temps pour écrire le livre et quelques exemples.

Je pense que le financement participatif est une solution :

- pour avoir plus de temps
- pour finir plus tôt

Suivez moi

- sur Twitter pour être informé du projet : `https://twitter.com/CDSoftX`
- sur mon site : `http://fun.cdsoft.fr`