Une expérience de notation en masse Revision: 1.10 — soumis à TICE 2002.

Christian Queinnec* & Emmanuel Chailloux**
Université Paris 6 — Pierre et Marie Curie
4, place Jussieu 75252 Paris Cedex 05
*LIP6: `Christian.Queinnec@lip6.fr`
**PPS: `Emmanuel.Chailloux@pps.jussieu.fr`

Résumé

Une expérience d'examen d'informatique mené sur ordinateur et assorti d'une notation automatique (environ 1800 copies corrigées) est décrite dans cet article. L'élaboration des sujets, le processus de notation et l'architecture logicielle de notation ainsi que les techniques d'affichage des notes et de consultation des copies corrigées sont tour à tour exposées et commentées.

Mots clés : Expérimentation et retour d'usage, Evaluation des compétences.

Keywords : Mass-marking, Computerized exams.

La licence d'informatique de l'université Pierre et Marie Curie (UPMC) compte plus de 400 inscrits administratifs. Le contenu scientifique et l'organisation de cette licence ont été rénovés en septembre 2001 et incorpore notamment un *contrôle continu transversal* indépendant des quatre modules que les étudiants ont à suivre par semestre. Ce contrôle continu transversal a pour but de proposer aux étudiants des travaux sur machine pendant tout le semestre afin — de créer une pression de travail légère mais constante et — de leur permettre de maîtriser plus rapidement le savoir-faire de développeur informatique. À la fin du semestre, un projet informatique (dit CFS pour *Contrôle Final Semestriel*) de quatre heures, individuel et sur machine, clôt ce contrôle continu.

Le nombre conséquent de nos étudiants impose un recours important à l'informatique pour pouvoir traiter un tel nombre de copies à corriger. Le but de cet article est de présenter l'expérience acquise lors de cette première épreuve sur machine où tout a été intégralement corrigé automatiquement, de commenter l'architecture informatique de correction, enfin, de décrire le déploiement des résultats de correction.

1 Déroulement

Cette section récapitule chronologiquement les grandes lignes de cette expérience.

Les conditions générales du contrôle continu ont été publiées au début du semestre. Les conditions particulières décrivant le déroulement du CFS ont été publiées dès les premiers jours de janvier 2002, le projet s'est tenu les 30 et 31 janvier 2002 et a concerné au final 253 étudiants (hors dispenses et cas particuliers).

Nous disposons de 105 machines ce qui nous a imposé de faire passer les étudiants en quatre fournées sur deux jours. Nous avons donc été contraints de fabriquer deux sujets et de légères variantes sur le sujet commun aux deux fournées de la journée. Sur la base des notes des partiels, quatre groupes d'étudiants statistiquement comparables (même moyenne et écart-type) ont été déterminés afin de pouvoir comparer l'impact des différents sujets.

Les salles de machine ont été reconfigurées afin de supprimer l'accès à Internet, seuls les serveurs de l'UFR d'informatique restaient consultables. Les outils de communication ont été neutralisés, les partitions d'échange en vrac (*/tmp* en terminologie Unix) démontées, les imprimantes débranchées. L'espace personnel (les *HOME* en terminologie Unix) des étudiants ont été remis en configuration nominale (les contenus précédents ont été restaurés après le CFS). Il n'a fallu qu'une demi-journée pour reconfigurer le centre de calcul et une autre demi-journée pour le restaurer.

Les sujets n'étaient proposés qu'en HTML sur un serveur de l'UFR accompagné du règlement général du projet. Aucun document écrit n'a été autorisé pendant l'épreuve afin de mettre à égalité les étudiants mais aussi d'éviter des fuites entre matin et après-midi. Tous les sites pédagogiques sur les serveurs de l'UFR restaient accessibles ainsi que toute la documentation en ligne des différents outils.

Au cours de ce projet, les étudiants étaient invités à soumettre régulièrement leur production via un système de formulaires (nommé *DeliveryBuilder*[1] créé et maintenu par F. Kordon). Nous avons enregistré près de 1800 soumissions : le sujet comportait cinq questions et il était possible de répondre question après question. Nous avons donc eu de l'ordre de sept soumissions par étudiant.

Les corrigés ont été publiés trois jours après les épreuves. Du fait de notre inexpérience (la notation des 1800 soumissions prenait environ 12 heures), il nous a fallu 15 jours pour achever la correction automatique de ces 1800 soumissions (pour 253 étudiants) et publier les copies notées. Une seule contestation a été enregistrée concernant cette correction.

2 Préparation des sujets

Nos sujets[2] proposaient deux petits exercices indépendants qu'une dernière question (Q5) composait en un jeu solitaire : une variation autour du pendu ou du *master mind*. Les questions demandant des programmes étaient elles-mêmes précédées de questions simples (Q1, Q3) visant à fabriquer des fichiers de données servant de tests à ces programmes. Le premier programme (Q2) correspondait à une double boucle imbriquée, le second (Q4) implantait un dictionnaire de mots, le remplissait et testait l'appartenance à ce dictionnaire.

Nous avons résolu que le langage de programmation ne serait pas imposé. Outre que cela permet à ceux qui n'ont pas suivi le module de programmation (utilisant principalement Ada) d'utiliser un autre langage, cela permet aux étudiants ayant quelques connaissances autres (C, C++, Caml, Java, Pascal, Perl, Prolog, Scheme, sh, etc.) de pouvoir choisir le langage le plus adapté aux questions. Cette décision a de nombreuses conséquences :

– Seul le comportement d'entrées/sorties du programme est testé.

[1] `http://www.infop6.jussieu.fr/licence/2001/delivery_builder/`
[2] `http://www.infop6.jussieu.fr/licence/2001/cct/annales/`

– Le texte du programme n'est pas pris en compte.
– L'étudiant doit fournir le script construisant son programme.

Ces trois conséquences sont assez proches de certaines pratiques industrielles où l'on doit écrire des programmes conformes à leur spécification et où les programmes sont soumis à des batteries de tests automatiques de non-régresssion. Elles apparentent le CFS au concours de programmation[3] annuel que mène ICFP (*International Conference on Functional Programming*). En ce sens, elles sont beaucoup plus exigeantes pour les étudiants car il n'est pas aussi simple d'amadouer un compilateur qu'un correcteur humain. En revanche, les étudiants disposent de toute la puissance informatique pour tester leurs programmes.

Nous avons observé 2 copies en C, 2 en Caml, 1 en Java, toutes les autres étaient en Ada. Cette uniformité nous a étonné d'une part parce que le recrutement de licence est extrêmement hétérogène (45% seulement de l'effectif provient du DEUG MIAS de l'UPMC où ils ont étudié (dans l'ordre) Scheme, Visual Basic (non disponible sur les machines du CFS), l'assembleur Pentium et `bash`). Des langages comme Pascal ou C++ sont largement enseignés dans les filières IUT ou BTS, Caml dans les CPGE. D'autre part, les enquêtes faites en début et fin de semestre indiquent que les étudiants ont le sentiment (56%) de dominer au moins un autre langage de programmation (Pascal 54%, C 43%, bash 39%, Scheme 37%, VisualBasic 33%, Java 27%, Caml 23%). La concentration sur Ada est probablement lié au fait que c'est le dernier langage qu'ils ont manipulé régulièrement.

Puisque seul le comportement du programme importe, il est nécessaire de spécifier très finement le comportement attendu. Nous avons également fourni des programmes solutions pour que, outre les exemples que nous donnions dans l'énoncé, les étudiants puissent les mettre en œuvre dans des contextes différents. Grâce aux droits d'Unix, ces programmes étaient exécutables sans être lisibles ce qui nous a évité de les voir soumis comme solutions originales.

Pour chaque question, nous avons indiqué quels étaient les fichiers à soumettre et quels étaient les critères de notation que nous appliquerions sur ces fichiers. Ce dernier point garantit que les programmes seront testés pour leur comportement nominal (entrées et options sont réputées correctes) et non pour leur robustesse quant aux comportements limites.

Décomposer les questions en préparatifs de tests et programmes a permis de pouvoir expliciter des critères de notation tels que

– nous appliquons *notre* programme sur *votre* test,
– nous appliquons *votre* programme sur *votre* test,
– nous appliquons *votre* programme sur *notre* test.

Pour chaque question, nous nous sommes aussi efforcés de montrer une mise en œuvre interactive et deux mises en œuvre programmatiques permettant d'automatiser les tests.

La plus grande difficulté (déjà mentionnée en [JCL00]) est que, comme les programmes sont jugés sur leurs entrées/sorties et que tous les langages ne les manipulent pas avec la même précision, il faut comparer ces sorties en prenant garde aux blancs superflus ou aux écritures numériques non canoniques.

[3] `http://cristal.inria.fr/ICFP2001/prog-contest/`

3 L'épreuve

L'épreuve en elle-même s'est bien passé. Les salles de machines étaient prêtes, les sujets ont été apportés par un ordinateur portable branché au réseau interne. Les étudiants devaient se placer aux places prévues par la convocation. L'ingénieur système était sur le pied de guerre ainsi qu'un surveillant par salle. Le ramassage des copies en fin d'épreuve est particulièrement rapide car le serveur de réception des soumissions est arrêté à l'heure dite.

C'était le premier CFS et de nombreuses questions ont été posées par des étudiants qui avaient probablement négligé de lire les instructions générales publiées depuis presque un mois. En ce qui concerne les sujets eux-mêmes les questions ont plutôt été d'ordre pratique :

- Où se trouve la documentation en ligne ? Comment chercher dans un document PDF ? Comment demander une page du manuel Unix ?
- Comment créer un script de génération d'un exécutable ?
- Que programmer quand les entrées ne sont pas conformes ?
- Des questions sur Ada (comment s'appelle, s'emploie, l'instruction qui ...) ?

La plupart de ces questions révèle la difficulté de transmission des savoir-faire qui passent naturellement dans une relation maître-apprenti mais qui restent inconnus lors d'un enseignement de masse. De plus ces savoir-faire sont souvent non-scientifiques, contingents et peu communiquables par écrit ou par oral (mais peut-être par vidéo).

Ce genre d'épreuve est conditionné par le bon fonctionnement des machines et du réseau interne. Il ne faut pas procurer plus de possibilités de triche que n'en offre des amphithéâtres bondés. Nous avons donc pris plusieurs mesures (qui figuraient explicitement dans les conditions générales du CFS) :

1. tous les répertoires *HOME* sont capturés à la fin de l'épreuve au cas où un étudiant aurait totalement oublié de soumettre (1 cas !)
2. tous les répertoires *HOME* sont capturés toutes les quelques minutes pendant l'épreuve au cas où une panne (coupure électrique) interromprait l'épreuve (afin de pouvoir noter le dernier état obtenu par les étudiants).
3. tous les caractères frappés et l'identité de la fenêtre auxquelles ils étaient destinés sont archivés.

Ce dernier point permet de pouvoir reconstituer à partir des sauvegardes les dernières modifications apportées par l'étudiant à ses programmes. Il permet aussi de pouvoir enquêter en cas d'accusation de copiage. Bien sûr, les droits des répertoires *HOME* sont surveillés ainsi que les connexions (*login*) afin de limiter les risques de triche.

Enfin et quoiqu'une durée de 4 heures de contrôle soit rare à l'université, nous n'avons eu aucun évanouissement à déplorer. En revanche, quelques étudiants se sont plaints que le temps était limité, qu'il était trop court et que l'épreuve ne correspondait pas à leur façon de travailler (en groupe). Rappelons qu'il s'agit d'un contrôle continu personnel et que les programmes demandés ont été rédigés par quatre enseignants différents en moins d'une heure et demie.

4 Notation des copies

La notation des copies doit être entièrement automatique. Cette section détaille les buts de ce processus de notation.

Des expériences de notation automatique ont eu lieu dans le semestre précédent dont nous avons tenu compte. En particulier, une demande fréquente des étudiants étaient de voir leur « copie » corrigée. Or leur copie n'est techniquement qu'un *.tar.gz* c'est-à-dire un instantané du contenu de leur répertoire de travail. Il nous a donc paru important de verbaliser tous les tests opérés sur la copie ainsi que de commenter toutes les observations menant à des fractions de notes. Une copie corrigée est donc produite qui incorpore les fichiers de l'étudiant et les commentaires associés. Cette copie mène à un document HTML visualisable (cf. figure 1). Un jeu de style permet de montrer en couleur les différents aspects de la copie : les notes, les commandes, les entrées ou sorties (normales ou d'erreur), les annotations, les tests, les questions, etc.

Les copies annotées ont une taille variant de 10 K à 100 K environ. La consultation des copies s'effectue aisément, en parallèle et sur le réseau, ce qui induit un gain de temps en évitant une séance de consultation de copies (avec, approximativement, de 50 à 100 visiteurs) mais permet de ne se pencher que sur les seules contestations. Nous n'en avons enregistré que deux sur les 1800 copies ainsi corrigées. L'une portait sur une copie ayant systématiquement introduit des lignes blanches avant les réponses, ces lignes blanches étaient difficiles à remarquer dans la présentation HTML. L'autre portait sur la compréhension des tests mêmes.

En revanche, nous avons été amené à passer d'un barème classique à un barème beaucoup plus progressif visant à donner des morceaux de point à des faits précis dont l'accumulation totale permettait de retrouver le barème classique (cf. les notes (en blanc sur fond bleu) dans la figure 1 en haut). Nous avons donc décomposé les notes en des myriades de notes parcellaires mais relativement indépendantes, de manière à ne pas présenter d'effet tout ou rien. Tous ces points sont bien sûr verbalisés comme indiqué plus haut.

En particulier, la comparaison des sorties de programmes a également été parcellisée. Plutôt que de comparer brutalement les sorties (même en ignorant les blancs superflus et en tenant compte des écritures variantes des nombres), nous avons (dans le cas des programmes de jeux qui produisaient à chaque tour de jeu 3 lignes nouvelles) comparé les 3 premières lignes, puis les 6 premières, les 9 suivantes, etc. avant de finir par la comparaison totale. Ainsi des programmes qui se fourvoient au bout d'un moment peuvent-ils espérer quelques points pour leur bon début. Nous envisageons d'écrire un nouveau comparateur calculant la distance de Levenshtein (le nombre de lignes qu'il faut supprimer, insérer ou modifier pour passer d'un fichier à l'autre) entre deux fichiers considérés ligne à ligne.

Comme c'était le premier CFS, nous avons aussi toléré des réponses non tout à fait conformes aux spécifications données (comme d'écrire « j'ai gagné ! » ou « Gagné » à la place de « GAGNE » comme spécifié et montré dans au moins 3 exemples). Toutefois ces déviances sont signalées par un avertissement orangé dans la copie annotée. Rappelons qu'il paraît normal que de futurs informaticiens respectent les spécifications des programmes qu'ils ont à construire.

4.1 Questions à problèmes

Deux types de questions nous ont posé des problèmes.

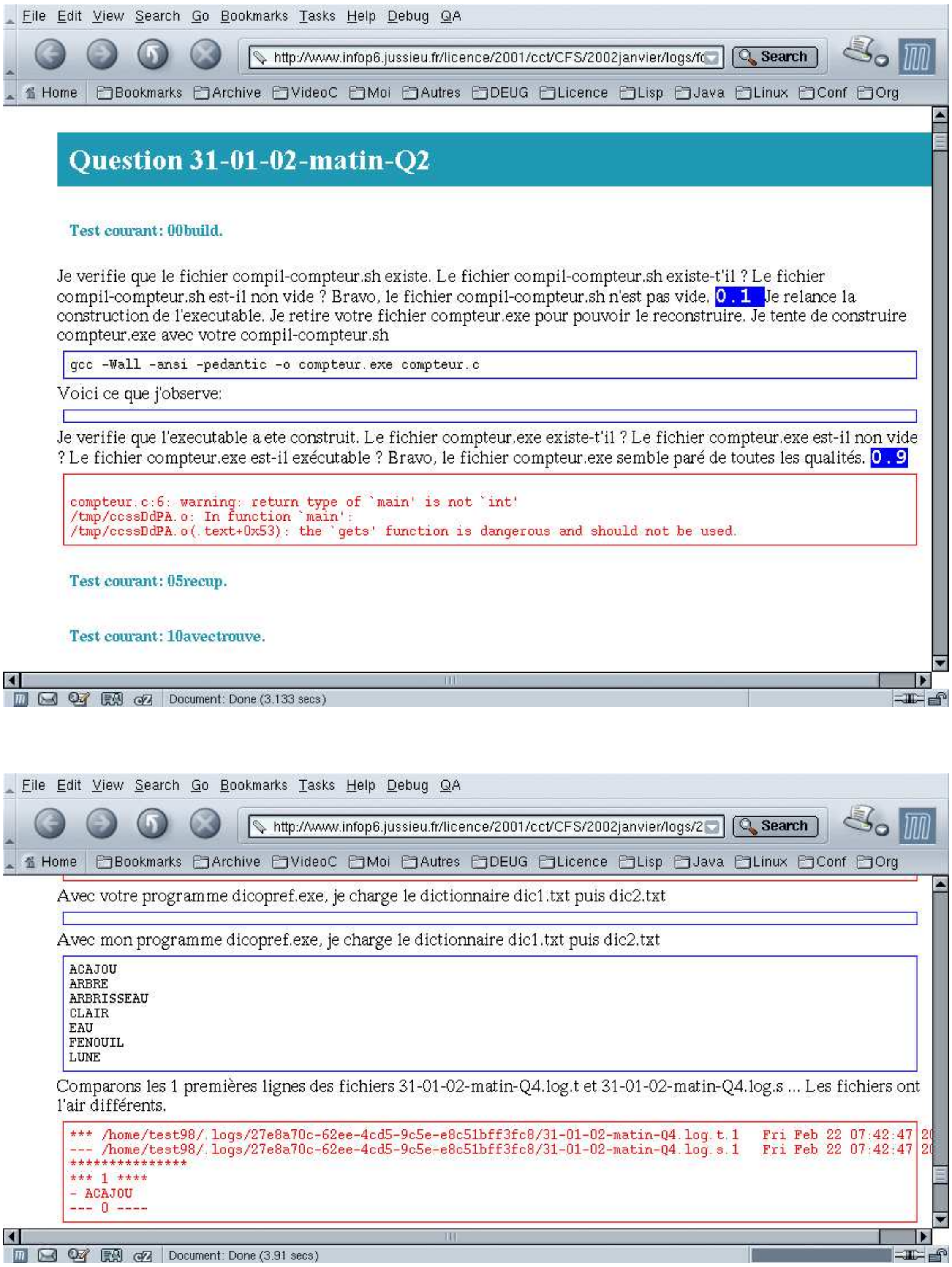


FIG. 1 – Exemples de « copies » annotées.

Pour inciter les meilleurs à rester jusqu'au bout de l'épreuve, nous avons introduit une question assortie d'un bonus. Les programmes figurant dans les 10% les plus rapides gagnaient 1 point. Ce genre de question impose de calculer le temps de réponse de chaque copie. Nous avons eu beau limiter le temps de réponse à 100 secondes (notre meilleure solution n'en nécessitait que 6), cette question a représenté les 9/10 du temps de correction (soit 10 heures environ sur les 12 heures du total). De plus, aucun programme d'étudiant n'a réussi à résoudre ce problème en moins de 100 secondes !

L'introduction de ce type de question doit donc être réfléchi car il impacte fortement la durée de correction. Et il est illusoire de penser que l'on peut ne corriger qu'une seule fois les copies. Nous les avons re-corrigées une quinzaine de fois ! Chaque nouveau cycle de correction conduit à des modifications de notation ou d'annotation. Comme en programmation, le raccourcissement du cycle édition-exécution est primordial et ce type de question va nettement à l'encontre de ce but.

De plus, cette propriété ne peut être calculée que globalement ce qui exclut qu'elle soit engendrée au fil de la copie. D'autres propriétés globales comme la détection de plagiat tombe dans la même catégorie.

Un second type de questions posa problème. Une question demandait à extraire un mot au hasard d'un dictionnaire. Le hasard était quantifié par « au moins trois mots différents sur quatre extractions à partir d'un dictionnaire d'au moins 100 mots, le tout recommencé 4 fois ». Re-corriger les copies ne mène pas forcément aux mêmes notes sur cette question !

4.2 Affichages

Nous avons résolu d'afficher les notes et les copies annotées sur le réseau. Un formulaire présente la note de chaque étudiant (cf. figure 2 en haut) qui peut, à partir de là, obtenir les annotations propres à chacune de ses copies (cf. figure 2 en bas) question par question (cf. figure 1).

Outre cet affichage, ces formulaires nous ont aussi permis, avant affichage pour les étudiants, de vérifier le bon comportement de la procédure de notation automatique en aidant au parcours des copies annotées.

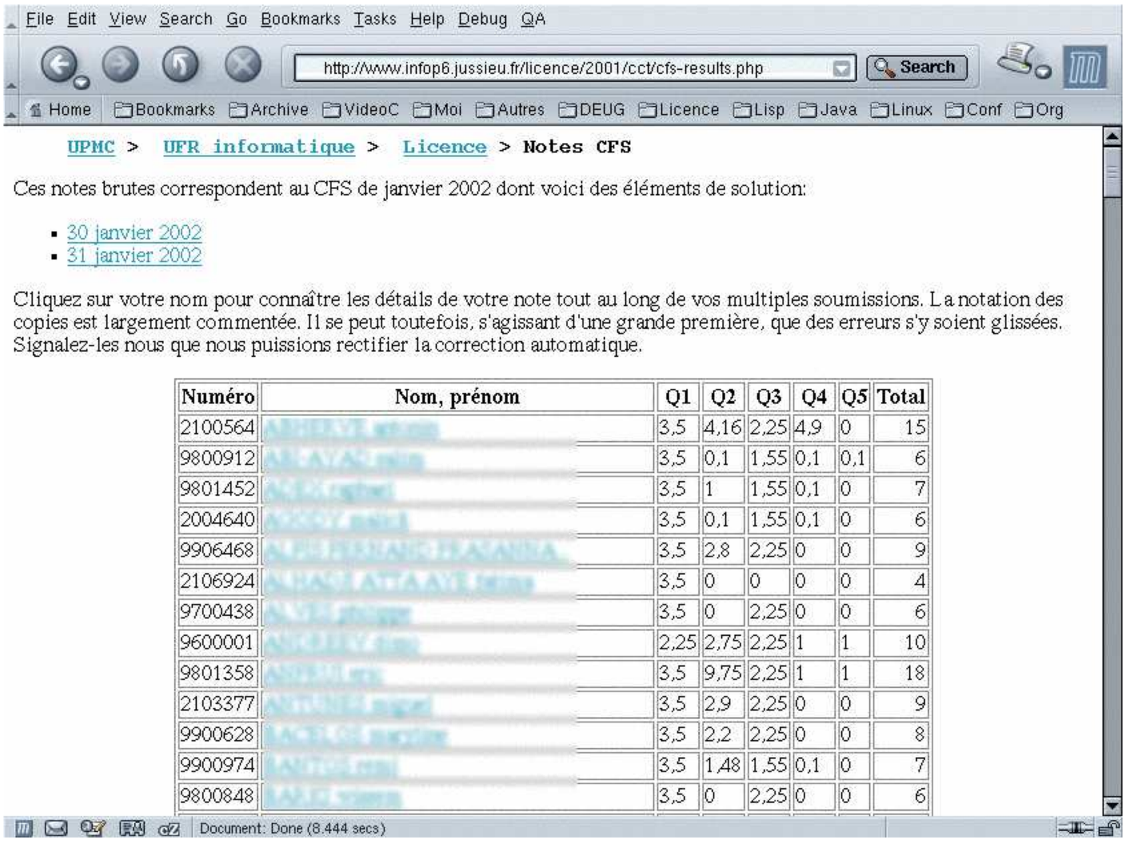
5 Architecture de correction

Dans cette section, nous détaillons l'architecture plus précise de correction des copies.

Tout d'abord, comme il s'agit d'exécuter des programmes inconnus, provenant des étudiants, donc potentiellement dangereux, une machine spéciale (dite de confinement) a été mise en place. Elle comporte les mêmes logiciels que ceux présents sur les machines des étudiants par contre, elle ne comporte aucun serveur, n'a que très peu d'accès sortant et aucun en écriture bref peut être bousillée sans problème sans importuner le reste du réseau.

De plus, chaque programme d'étudiant est limité en temps CPU (50 secondes par défaut), en espace mémoire et disque (10 Mo), etc. afin de ne pas perturber la correction des autres copies. Il est ainsi possible que la correction résiste à un programme qui boucle.

Les soumissions des étudiants sont rendues visibles depuis cette machine. Pour chaque soumission (un fichier *.tar.gz*), on crée un répertoire temporaire où l'on déballe les fichiers soumis. À chaque question est associé un répertoire contenant des scripts (en `bash`) de tests. Afin d'ac-



File Edit View Search Go Bookmarks Tasks Help Debug QA

http://www.infop6.jussieu.fr/licence/2001/cct/cfs-results.php

UPMC > UFR informatique > Licence > Notes CFS

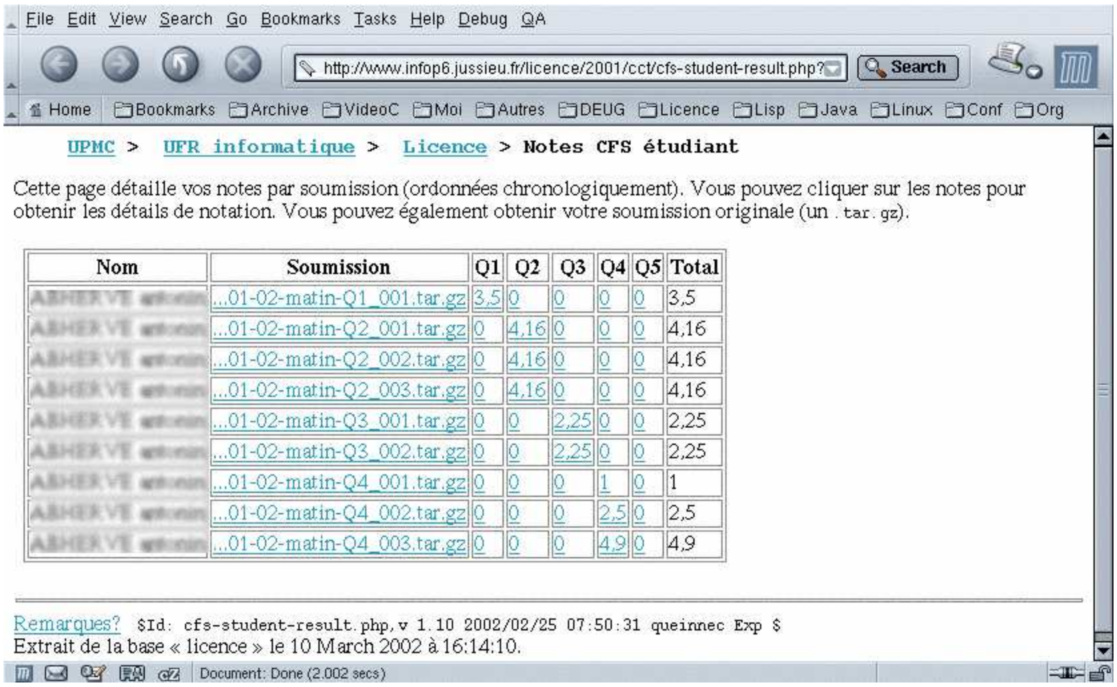
Ces notes brutes correspondent au CFS de janvier 2002 dont voici des éléments de solution:

- 30 janvier 2002
- 31 janvier 2002

Cliquez sur votre nom pour connaître les détails de votre note tout au long de vos multiples soumissions. La notation des copies est largement commentée. Il se peut toutefois, s'agissant d'une grande première, que des erreurs s'y soient glissées. Signalez-les nous que nous puissions rectifier la correction automatique.

Numéro	Nom, prénom	Q1	Q2	Q3	Q4	Q5	Total
2100564	[illegible]	3,5	4,16	2,25	4,9	0	15
9800912	[illegible]	3,5	0,1	1,55	0,1	0,1	6
9801452	[illegible]	3,5	1	1,55	0,1	0	7
2004640	[illegible]	3,5	0,1	1,55	0,1	0	6
9906468	[illegible]	3,5	2,8	2,25	0	0	9
2106924	[illegible]	3,5	0	0	0	0	4
9700438	[illegible]	3,5	0	2,25	0	0	6
9600001	[illegible]	2,25	2,75	2,25	1	1	10
9801358	[illegible]	3,5	9,75	2,25	1	1	18
2103377	[illegible]	3,5	2,9	2,25	0	0	9
9900628	[illegible]	3,5	2,2	2,25	0	0	8
9900974	[illegible]	3,5	1,48	1,55	0,1	0	7
9800848	[illegible]	3,5	0	2,25	0	0	6

Document: Done (8.444 secs)



File Edit View Search Go Bookmarks Tasks Help Debug QA

http://www.infop6.jussieu.fr/licence/2001/cct/cfs-student-result.php?

UPMC > UFR informatique > Licence > Notes CFS étudiant

Cette page détaille vos notes par soumission (ordonnées chronologiquement). Vous pouvez cliquer sur les notes pour obtenir les détails de notation. Vous pouvez également obtenir votre soumission originale (un `.tar.gz`).

Nom	Soumission	Q1	Q2	Q3	Q4	Q5	Total
[illegible]	...01-02-matin-Q1_001.tar.gz	3,5	0	0	0	0	3,5
[illegible]	...01-02-matin-Q2_001.tar.gz	0	4,16	0	0	0	4,16
[illegible]	...01-02-matin-Q2_002.tar.gz	0	4,16	0	0	0	4,16
[illegible]	...01-02-matin-Q2_003.tar.gz	0	4,16	0	0	0	4,16
[illegible]	...01-02-matin-Q3_001.tar.gz	0	0	2,25	0	0	2,25
[illegible]	...01-02-matin-Q3_002.tar.gz	0	0	2,25	0	0	2,25
[illegible]	...01-02-matin-Q4_001.tar.gz	0	0	0	1	0	1
[illegible]	...01-02-matin-Q4_002.tar.gz	0	0	0	2,5	0	2,5
[illegible]	...01-02-matin-Q4_003.tar.gz	0	0	0	4,9	0	4,9

Remarques? `$Id: cfs-student-result.php,v 1.10 2002/02/25 07:50:31 queinnec Exp $`
Extrait de la base « licence » le 10 March 2002 à 16:14:10.

Document: Done (2.002 secs)

FIG. 2 – Affichage des notes et des copies annotées. Les noms ont été volontairement occultés.

croître leur modularité, ces tests sont des petits exécutables indépendants, ne testant qu'une unique caractéristique dont le nom commence par deux chiffres, ce qui permet de les tourner dans l'ordre numérique (un peu comme les fichiers dans */etc/rc.d/rc3.d/*). L'ajout d'un test ne nécessite donc pas de connaître les autres mais seulement le moment où l'on souhaite le tourner.

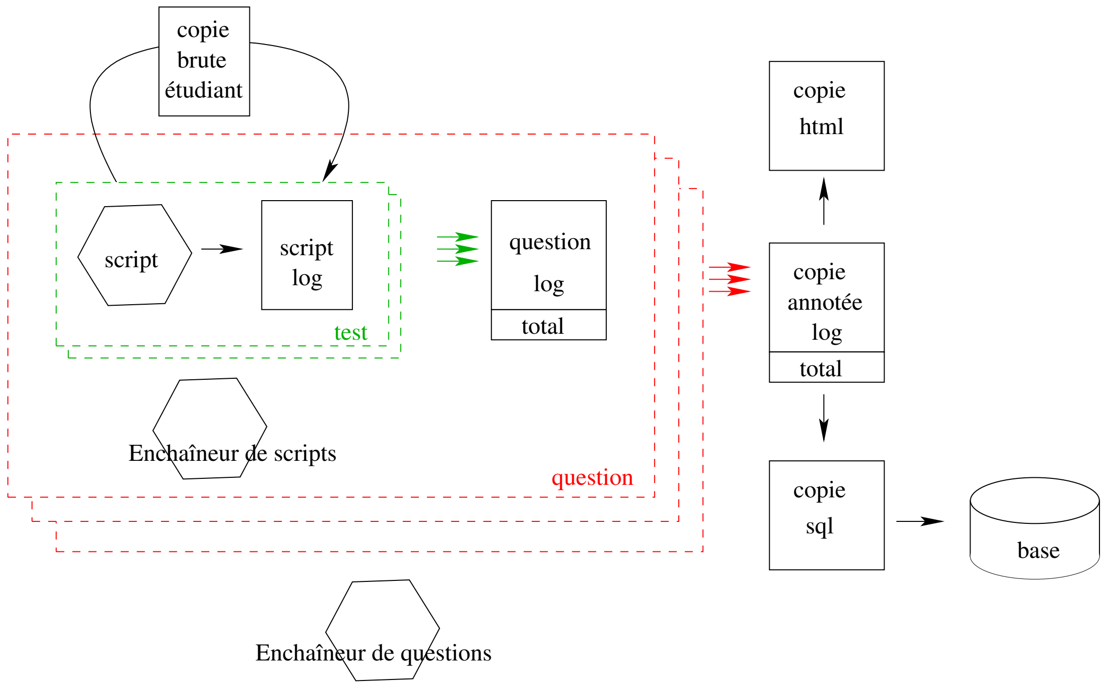


FIG. 3 – Structure des tests.

Chaque test produit un fichier annoté (un *log* sur la figure 3) : un fichier XML pourvu de balises bien parenthésées dans lequel s'accumulent la verbalisation des tests, les commentaires sur les résultats attendus ou obtenus, les notes partielles gagnées (ou perdues) et quelques autres caractéristiques comme les codes de retour des programmes, les temps de calcul, etc. Il est particulièrement pratique de mélanger tout cela dans un unique fichier structuré car ainsi un seul fichier (ou flux) est à manipuler et protéger.

Lorsque tous les tests d'une question sont passés, les fichiers annotés sont concaténés pour former le *log* de la question. Un appendice est engendré qui résume la note obtenue pour la question (en sommant toutes les notes partielles des tests).

Lorsque toutes les questions ont été corrigées, les fichiers annotés sont concaténés pour former le *log* entier de la copie. Un appendice est engendré qui résume la note totale obtenue pour la copie (en sommant toutes les notes des questions). De ce *log* sont obtenus un fichier HTML correspondant à la copie annotée (qui sera visible pour consultation) et un fichier de requêtes SQL qui insèreront les notes (et quelques autres caractéristiques comme les temps d'exécution) de chaque question de la copie dans une base de données.

Cette base de données sera consultable par les étudiants au moyen des formulaires vus en figure 2. La base de données permet d'effectuer divers traitements (ou ajustements) statistiques (moyennes, écart-type) ainsi que de calculer les propriétés globales comme les bonus. Elle permet également le calcul de la note totale de l'étudiant en prenant, par exemple, la somme des maxima obtenus pour chaque question quelque soit la copie la contenant.

Pendant que la machine de confinement note, on peut rapatrier les copies corrigées, les insérer en base et ainsi vérifier le bon comportement de la notation.

En ce qui concerne la génération d'HTML, nous procédons à diverses améliorations surfaciques comme *(i)* la compression des lignes semblables (cas de programmes qui bouclent sur une seule ligne), *(ii)* l'introduction d'ancres permettant de se positionner sur une question particulière dans le document.

5.1 Écriture des tests individuels

Chaque test individuel est décrit par un script en `bash`. Afin de factoriser le code et produire des commentaires uniformes, une bibliothèque de fonctions a été élaborée pour des actions comme la vérification de l'existence d'un fichier, d'un exécutable, sa regénération, la comparaison de deux fichiers par groupes de lignes, etc. Toutes ces fonctions engendrent des fragments d'XML.

Voici un exemple de test individuel correspondant à la figure 3 en haut :

```
#! /bin/sh
# Charger la bibliotheque de fonctions communes:
. ${RT_MARK_DIR}/runtestlib.sh

# Tester si ce fichier existe:
rt_exist "compil-compteur.sh"

# gain d'un dixieme de point alors:
rt_win 0.1

# Tente de reconstruire compteur.exe
rt_rebuild compteur

# A-t'on bien un executable non vide ?
rt_executable_exist compteur.exe

# Si l'on arrive la, c'est que c'est bon!
rt_win 0.9

#fin
```

Les tests doivent être très précautionneux dans leur manipulation des programmes des étudiants. Non seulement ces programmes doivent ainsi être confinés mais leurs résultats doivent être considérés avec circonspection. Nous avons dupliqué certains utilitaires classiques comme `head` (qui renvoie les premières lignes d'un flux ou fichier) or `tee` (qui copie un flux vers un fichier) de manière à limiter la taille des résultats. Par exemple, lire la première ligne produite par un programme qui engendre une unique ligne de plusieurs Moctets doit être contrôlé ! Copier le résultat d'un programme qui boucle mène à des fichiers gigantesques. La règle que nous avons prise est que tout résultat d'un programme provenant d'un étudiant doit être limité par une taille (compatible des résultats attendus). Le confinement général d'Unix permet de limiter le nombre d'octets écrits en fichiers mais ne permet pas de limiter plus précisément le nombre d'octets produits dans un fichier particulier.

Il pourrait être intéressant de faire une analyse statique des tests pour vérifier que toutes les productions estudiantines sont bien limitées (une sorte de *tainted shell* comme il en existe dans certains langages comme Perl ou PHP).

6 Perspectives et conclusions

Dans cet article, nous relatons une expérience d'examen d'informatique (concernant 250 étudiants) sur ordinateur assorti d'une notation entièrement automatique (1800 copies). Le site[4] contient plus de détails et notamment les sujets de CFS. Comme la correction est automatique, les deux CFS actuels sont aussi disponibles pour entrainement sur le réseau : il est donc possible, à cette même adresse[5], de soumettre ses programmes et de recevoir, par courriel, sa correction personnalisée. C'est un excellent moyen d'acclimatation pour le prochain CFS semestriel ou de révision pour le module de programmation de la session de septembre.

A priori, la nature des épreuves (écrire, modifier ou compléter des programmes) est libre.

Parmi les améliorations que nous ont suggérées certains étudiants on trouve la publication des tests exacts (ce qui implique une grande robustesse des programmes de tests (c'est probablement la raison pour laquelle nous n'avons trouvé quasiment aucune référence à qui se comparer sur ce sujet)).

Nous envisageons de corriger, à la volée, pendant l'épreuve même, en indiquant la note minimale que la copie obtiendra. Il faut dans ce dernier cas, être particulièrement sûr de sa procédure de notation mais cela peut rassurer les étudiants.

Pour résumer, les principaux enseignements de cette expérience sont :
- la production d'une copie annotée verbalisant à la fois les tests effectués et les résultats obtenus,
- les précautions à prendre pour l'écriture de tests requérant l'exécution de programmes d'étudiants.

Nous démontrons que cette expérience est viable, elle sera d'ailleurs reproduite en juin prochain. Les outils sont en cours de réutilisation pour d'autres épreuves sur ordinateur. Ils peuvent être demandés aux auteurs.

Références

[JCL00] M.S. Joy, P.-S. Chan, and M. Luck. Networked submission and assessment. In *Proceedings of the 1st Annual Conference of the LTSN Centre for Information and Computer Sciences, LTSN-ICS, 2000*, 2000.

[4] `http://www.infop6.jussieu.fr/licence/cct/annales/`
[5] `http://www.infop6.jussieu.fr/licence/cct/annales/`