Chapitre 1: Présentation du cours

INF3135

Construction et maintenance de logiciels

Alexandre Blondin Massé

Université du Québec à Montréal

v251

Plan

Plan de cours

Informations générales

- **Trimestre**: Hiver 2025
- **Titre du cours**: Construction et maintenance de logiciels
- **Sigle**: INF3135
- **Département**: Informatique
- **Enseignant**: Alexandre Blondin Massé
- **Courriel**: blondin_masse.alexandre@uqam.ca
- **Site personnel**: http://ablondin.uqam.ca
- **Coordonnateur**: Quentin Stiévenart
- **Site du cours**: http://inf3135.uqam.ca
- **Plan de cours**: cliquer ici

Description officielle (site de l'UQAM):

« Notions de base de la programmation procédurale et impérative en langage C sous environnement Unix/Linux (définition et déclaration, portée et durée de vie, fichier d'interface, structures de contrôle, unités de programme et passage des paramètres, macros, compilation conditionnelle). Décomposition en modules et caractéristiques facilitant les modifications (cohésion et couplage, encapsulation et dissimulation de l'information, décomposition fonctionnelle). Style de programmation (conventions, documentation interne, gabarits). Déboggage de programmes (erreurs typiques, traces, outils). Assertions et conception par contrats. Tests (unitaires, intégration, d'acceptation, boîte noire vs. boîte blanche, mesures de couverture, outils d'exécution automatique des tests). Évaluation et amélioration des performances (profils d'exécution, améliorations asymptotiques vs. optimisations, outils). Techniques et outils de base pour la gestion de la configuration. Système de contrôle de version. »

Objectifs du cours (1/2)

- **Développer** et **modifier** des composants logiciels écrits dans un langage **impératif** et **procédural**;
- Bien maîtriser le **langage C** et le **compilateur** du C sous **Unix/Linux**;
- Utiliser les notions de **module**, de **cohésion** et **couplage**, de complexité structurale, de dissimulation de l'information, etc., pour évaluer la **qualité** d'un composant logiciel;
- Expliquer et utiliser les principales techniques de **modularisation**: décomposition fonctionnelle, dissimulation de l'information, filtres et pipelines;
- Utiliser des **assertions** (pré/post-conditions, invariants) pour **documenter** des composants logiciels et assurer leur bon fonctionnement;
- **Déboguer** un programme à l'aide de techniques, stratégies et outils appropriés;

Objectifs du cours (2/2)

- Vérifier le bon **fonctionnement** d'un composant logiciel à l'aide de **tests** fonctionnels et structurels, et d'évaluer à l'aide des notions et outils appropriés la qualité des tests (par ex., complexité cyclomatique, mesures de couvertures des tests);
- Évaluer de façon empirique à l'aide d'outils appropriés (par ex., profils d'exécution) les **performances** d'un composant logiciel de façon à pouvoir, si nécessaire, en améliorer les performances;
- Utiliser divers outils (outil de gestion de configuration, fichiers **Makefile**, **langage de scripts**) pour organiser le développement de programmes comportant plusieurs composants ou modules;
- Expliquer les notions de base de la **maintenance** des logiciels et appliquer certaines techniques de maintenance (tests de **régression** exécutés automatiquement, **remodelage** de programmes).

Modalités d'évaluation

3 travaux pratiques (40%)

- TP1: construction d'un programme (13%)
- TP2: maintenance d'un programme (13%)
- TP3: maintenance d'un programme (14%)

Examens (50%)

- 1 examen intra (25%)
- 1 examen final (25%)

5 quiz (10%)

- En classe, durée de 20 minutes
- 2% chacun

Réussite du cours

- Examen intra + examen final $\geq 50\%$
- Moyenne globale $\geq 50\%$

Contenu détaillé (1/2)

Chapitre 1. Présentation du cours. Plan de cours. Modalités d'évaluation. Systèmes Unix/Linux. Langage C. Construction. Maintenance.

Chapitre 2. Introduction au langage C. Compilation. Types de bases. Variables et constantes. Opérateurs. Mots-clés. Structures de contrôle. Types composés. Pointeurs. Fonctions. Directives.

Chapitre 3. Développement. Environnement de développement. Éditeurs de texte. Gestion de sources. Automatisation de tâches. Tests. Développement dirigé par les tests. Intégration continue. Débogage. Style de programmation.

Chapitre 4. Pointeurs. Tableaux. Chaînes de caractères. Pointeurs de fonction. Fichiers. Entrées et sorties.

Contenu détaillé (2/2)

Chapitre 5. Construction. Documentation. Gestion de la mémoire. Bibliothèques. Gestion des erreurs. Programmation défensive. Programmation par contrats. Intégration et déploiement continus.

Chapitre 6. Structures de données. Tableaux à plusieurs dimensions. Mémoire et allocation dynamique. Tableau dynamique. Ensemble. Tableau associatif. Tableau multidimensionnel. Pile. File. Arbre binaire.

Chapitre 7. Maintenance. Types de maintenance. Réusinage. Retour sur les tests. Modularité. Cohésion et couplage. Encapsulation. Dissimulation de l'information. Décomposition fonctionnelle. Performance.

Références et liens

Contenu du cours

- *The C Programming Language*, de Kernighan et Ritchie
- *Manuel d'utilisation de Git*: disponible en ligne
- *Documentation de Make*: disponible en ligne
- *Documentation de Markdown*: documentation officielle et spécialisation pour GitLab

Politiques de l'UQAM

- Règlement 18 sur la tricherie et l'intégrité académique (plagiat): http://r18.uqam.ca/.
- Politique 16 contre le harcèlement sexuel: document pdf.

Unix/Linux

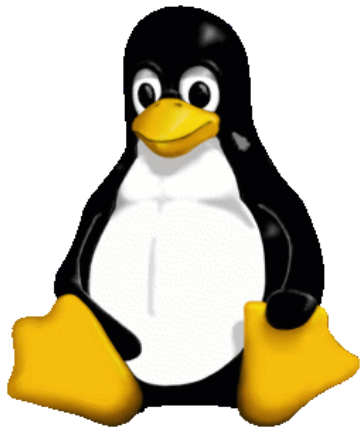
Début d'Unix

- 1969 Ken Thompson et Dennis Ritchie développent un Unix primitif (chez Bell)
- 1972 Ritchie invente le C & Thompson réécrit Unix en C

GNU (GNU is not Unix)

- 1984 Richard Stallman annonce le projet GNU.
  Développement de logiciels Unix libres:
  compilateur C (gcc) et autre outils
- 1985 Stallman crée la *Free Software Foundation* (FSF)
- 1989 Stallman publie la *General Public License* (GPLv1)

Linux



- 1991 Linus Torvalds annonce le développement d'un noyau Unix libre pour PC Intel 80386

Distribution Linux

- Système d'exploitation **complet**
- Ensemble **cohérent** de logiciels
- Basé sur un noyau **linux** et des outils **GNU**
- Organisation et processus de **publication**
- **Outils** d'installation et mise-à-jour
  dont le **gestionnaire de paquets**

Plus de 300 distributions actives existent (selon distrowatch)

- Linux Lite, Zorin OS
- Elementary OS, Trenta OS, Deepin
- Qubes OS, Tail OS

Dans le cours, l'utilisation de Linux est **obligatoire**.

Le langage C

Bref historique

- **Années 70**: Naissance du langage, créé par **Ritchie** et **Kernighan**

→ Origine du langage fortement liée à celle d'Unix

→ 90% du système Unix écrit en C

- **1978**: Publication du livre « The C Programming Language », par Kernighan et Ritchie
- **1983**: ANSI forme un comité pour **normaliser** le langage
- **1989** Apparition de la norme **ANSI-C**
- **1999**: Révision du standard (ISO C99)
- **2011**: Révision du standard (ISO C11)

Caractéristiques du langage

- **Bas niveau**: près du langage machine, contrôle élevé de la mémoire, efficace
- **Déclaratif**: le type des variables doit être déclaré
- **Flexible**: espacement, indentation
- **Structuré**: organisé en blocs (accolades)
- **Modulaire**: division en fichiers, compilation séparée
- **Flexible**: peu de vérification, pointeurs typés mais non contrôlés
- **Spécifique**: pour bien faire une tâche, adapté aux petits programmes et aux bibliothèques
- **Portable**: mais avec parfois un certain effort
- **Simple**: spécification assez courte
- **Verbeux**: il faut écrire beaucoup de code

Exemple

Fichier `maj.c`:

```
#include <stdio.h>
#include <ctype.h>

int main() {
  char c;
  while ((c = getchar()) != EOF) {
    putchar(toupper(c));
  }
  return 0;
}
```

Question

Que fait ce programme?

Logiciel de contrôle de versions

Logiciel de contrôle de versions

- Permet de stocker un ensemble de **fichiers**
- Conserve en mémoire la **chronologie** de toutes les modifications effectuées
- Offre des services de **partage** des fichiers entre plusieurs **personnes**
- Est utilisé pour conserver les différentes **versions** du code source d'un projet
- Permet également de gérer différentes **branches** dont les évolutions sont temporairement indépendantes
- Garantit dans une certaine mesure l'**intégrité** des fichiers, car il est toujours possible de revenir en arrière

Naissance de Git

- **2002**: Linus Torvalds utilises BitKeeper pour conserver l'historique de Linux
- **6 avril 2005**: La version **gratuite** de BitKeeper est supprimée: Torvalds décide de créer son propre logiciel de contrôle de version, Git
- **18 avril 2005**: Git supporte l'opération de fusion de fichiers
- **16 juin 2005**: Git est officiellement utilisé pour conserver l'historique de Linux
- **Fin juillet 2005**: Junio Hamano devient le développeur principal de Git

Commandes les plus courantes

- **Créer** un nouveau projet: `git init`
- **Cloner** un projet existant: `git clone`
- **Sauvegarder** l'état courant du projet: `git commit`
- **Versionner** un nouveau fichier: `git add`
- **Ajouter** un fichier pour le prochain commit: `git add`
- **Consulter** l'historique: `git log`
- **Récupérer** des changements à distance: `git pull`
- **Téléverser** des changements à distance: `git push`

Apprendre les commandes

- Plusieurs commandes vues en **laboratoire**
- Vous devrez aussi en apprendre par **vous-même**
- Beaucoup d'**options** sont disponibles:

```
$ git remote --help
```

Construction et maintenance

Quelques définitions (extraites de Wikipedia)

Construction

« La construction logicielle est le processus de conversion de fichiers de code source en artefacts logiciels autonomes pouvant être exécutés sur un ordinateur, ou le résultat de cette opération. »

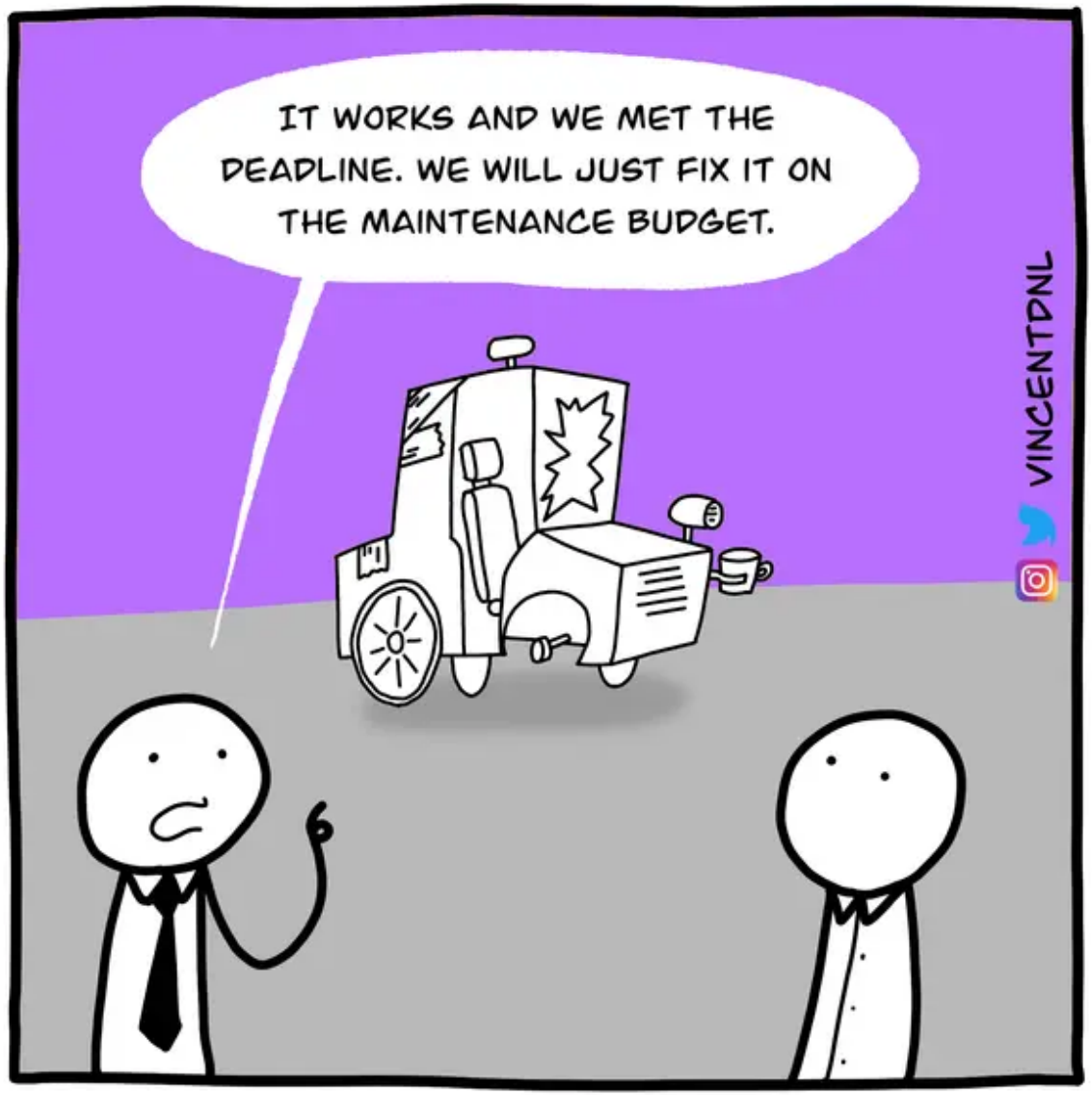
Maintenance

« La maintenance vise à maintenir ou à rétablir un bien dans un état spécifié afin que celui-ci soit en mesure d'assurer un service déterminé. »

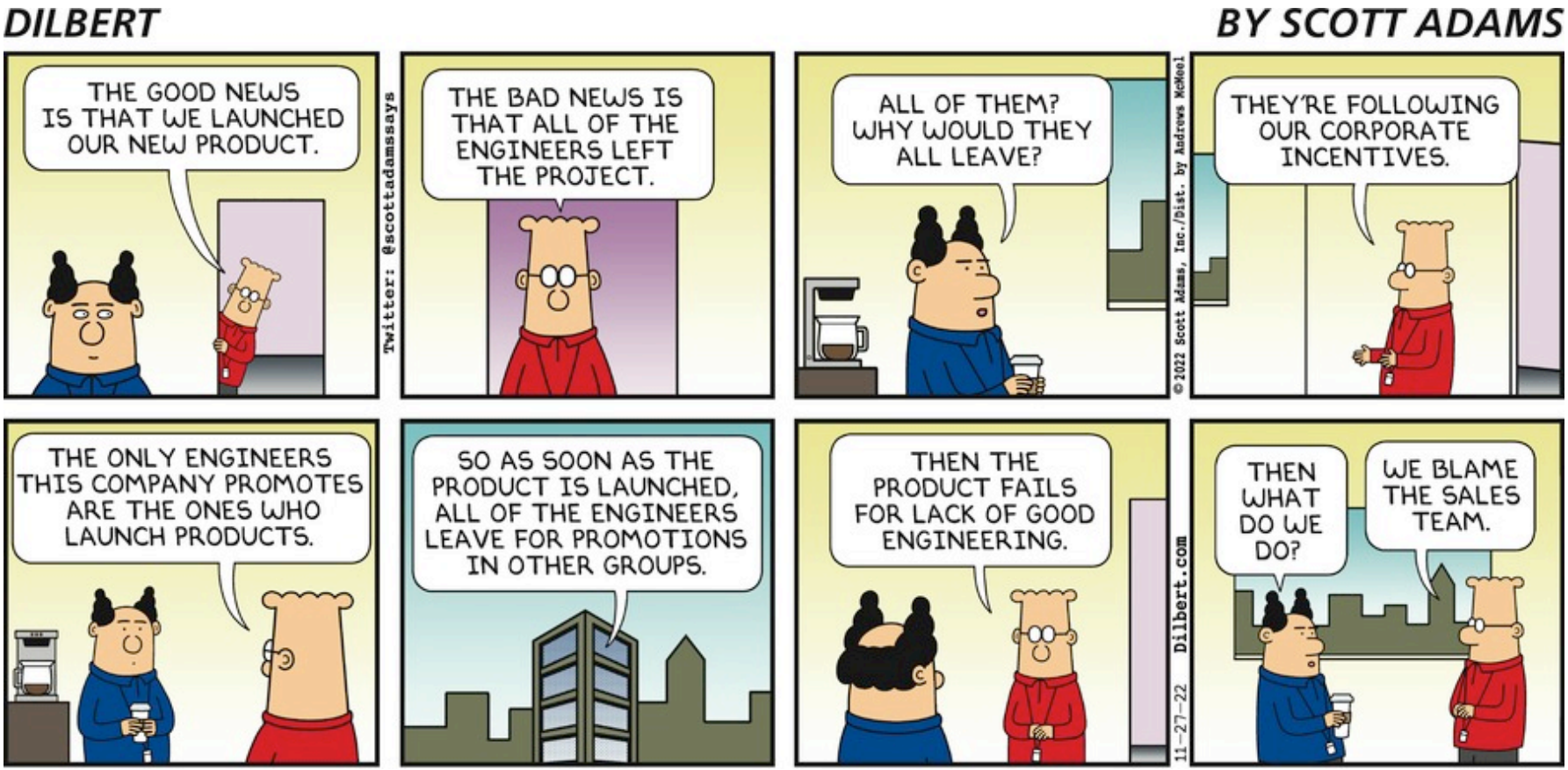
Modularité

« D'une manière générale, la modularité est le degré auquel les composants d'un système peuvent être séparés et recombinés, souvent avec l'avantage de la flexibilité et de la variété d'utilisation. »

Maintenance



Vision à long terme



Pas (seulement) un cours de programmation!

Plusieurs éléments évalués

- Le bon **fonctionnement** des programmes développés
- Le **style** de programmation
- La **modularité** des programmes
- La qualité des **tests** mis en place
- L'utilisation adéquate de **Git**
- La qualité de la **documentation** (notamment les fautes typographiques, d'orthographe, de syntaxe)

Changement de paradigme

- Décomposer le travail en **petites** actions (**itérations**)
- Mettre en place des **tests**
- Effectuer des sauvegardes **fréquentes** (*commit*)
- **Documenter** au fur et à mesure
- **Réusiner** (*refactor*) au fur et à mesure