CMPT 276: Introduction to Software Engineering

A Course Overview

Jeffrey Leung
Simon Fraser University

Spring 2016

Contents

1 Introduction

- *Virtual machine:* Software which emulates bare hardware, on which an operating system can be run
  - Multiple operating systems can be running on the same physical machine ('guests' running on a 'host')
  - Can allow for a standardized development environment
  - Many machine instructions have no performance penalty; calling the operating system is slower

2 Version Control

2.1 Introduction

- *Version control:* Flow of software development in which changes to the code are modular and documented, allowing for clear revision history and simple reversion of changes
- Repositories are hosted on a central website
  - Reasons:
    - Developers can access the code
    - The project can be documented (e.g. issuse, merge/pull requests, features, bugs, code review)
  - E.g. GitHub, Bitbucket, GitLab
- Allows specification of:
  - Which changes constitute a unique version
  - Who has authority to accept changes
- *Workflow:* Convention for how changes move from the developer's repository into the centralized system
- *Development process:* Convention for how software development is handled

2.2 Types of Systems

- *Centralized:* Version control which uses a single central repository to which users can contribute
  - E.g. SVN
- *Distributed:* Version control which allows for multiple repositories, both for the individual programmers and original developers
  - E.g. Git, Mercurial, Bazaar

2.3 Workflow

- Requires Git and a hosting service
- *Issue:* Possible change to a project
  - E.g. New features, bugs, performance enhancements
- Process:
  - An issue is opened
  - Contributors discuss the issue
  - Issue may be:
    - Closed (if it is determined to be impossible, unfeasible, etc.)
    - Altered
    - Divided into multiple, smaller issues
    - Allowed to be implemented through consensus
  - A contributor is assigned the issue
  - The contributor implements the change on a new branch and sends a pull/merge request

  - Issue may be:
    * Altered further and the change revised
    * Accepted and merged into the main branch

2.4 Commits

2.4.1 Overview

- Keep a commit focused on one independent change

2.4.2 Messages

- First word:
  - Capitalized
  - Verb
  - Present tense
- Describes the problem/change/behaviour
- Under 50 characters (commit messages longer than 50 characters are often cut off on the online version control system)
- No end-of-sentence punctuation
- Other sources:
  - *Deliberate Git* by Stephen Ball

3 Unit Testing

- *Unit test:* Method of software programming in which the smallest dividable groups of modules are tested
- Structure:
  - Suites are a collection of fixture(s) and test fixture(s)
  - Fixture classes provide setup and cleanup for testing an operation
  - Test fixtures are a collection of tests, before which a specified fixture is created and after which the fixture is deleted
  - After each test case, the expected results are compared with the actual results
- Types of tests:
  - Correct usages
  - Repeated usages
  - Incorrect usages (e.g. too many/few parameters)
  - Edge cases (e.g. out-of-bounds, 0, invalid characters)
  - Exploits (e.g. database injections)
- Characteristics to test:
  - Input validation
  - Input sanitation
  - States after an operation
  - Usability
  - Performance

4 C++

4.1 Initialization of Variables

- *Initialization:* Setting the value of a variable when it is created
  - *Assignment:* Replacing the current value of a variable
- A default value may or may not be assigned to the variable
- *Universal initialization:*
  - Only in C++11 or later
  - E.g.

```
// Creates an integer variable set to the value 4
int i {4};

// Creates a 4-element vector of integers with the values 3, 1, 5, and 9
std::vector<int> v { 3, 1, 5, 9 };
```

  - Possible syntax mistakes:
    - Creating a 1-element vector of integers with the value 4:

```
std::vector<int> {4};
```

    - Creating a 4-element vector of integers with the value 0 for each integer:

```
std::vector<int> (4);
```

4.2 const Variables/Parameters

- **const** keyword ensures the value of a variable or parameter is not altered
  - Compilation will result in an error
- E.g.

```
const float pi = 3.14;
```

- Order is unimportant
  - E.g.

```
const int i = 4;
int const j = 2;
```

- Overrided by **const_cast<>** or a class member specified as **mutable**

4.3 const Methods

- **const** keyword ensures the members of the associated class are not altered in the method
  - Compilation will result in an error
- E.g.

```
class C
{
  private:
  int num = 0;

  public:
    int return_num() const
    {
      // num (and any other members of this object) cannot be modified in
           this method
      return num;
    }
}
```

- Overrided for a class member specified as **mutable**

4.4 Object Lifetime

- *Scope:* Code in which the given object can be accessed
- *Lifetime:* Time period during which the given object exists and is monitored
- Types of lifetimes:
  - *Static:* Having a lifetime of the entire program
    - Set when a variable is declared without memory specifiers and outside all functions/methods (including **main()**)
    - Static variables are created before and destroyed after the execution of **main()**
    - Order of construction or destruction of static variables cannot be specified
      - Constructor of a static object cannot use another static object
        - E.g.

```
std::string file_name = "file.txt";
File f = file_name;  // Invalid

int main()
{
  // Code here
}
```

  - *Local/stack:* Having a lifetime of a single block or function call
    - Set when a variable is declared without memory specifiers within a block or function call
    - Local variables are destroyed when the block or function call is exited
      - The earlier a local variable is created, the later it is destroyed (stack order; destroyed in reverse order of creation)
    - *Reference/Referring variable:* Object which acts as a direct representation of an assigned object
      - Syntax:

```
Type var;
Type& var_ref = var;
```

- The objects are the same; modifying one modifies the other
- Must be initialized with the referenced object of the same type

– *Dynamic/heap:* Having a lifetime controlled directly by the programmer

* Set when a variable is declared with memory specifiers (i.e. **new** or **new []**)
* Dynamic variables are created and destroyed (with **delete** or **delete []**) by the programmer
* Failure to deallocate dynamic memory can result in memory leaks and data errors
* <**memory**> header includes functions which allow automatic dynamic allocation of variables, as well as automatic de-allocation when all pointers to the object are destroyed
  - **std::unique_ptr<T>** only allows one given pointer to a dynamic variable at any given time, and can be transferred
    - Does not have a copy constructor or assignment function
    - **std::move()** transfers the pointer
    - **uniqueptr.get()** returns a standard pointer to the object; destroying the standard pointer does not destroy the object
  - **std::shared_ptr<T>** allows multiple pointers to a dynamic variable at any given time, and can be duplicated
    - Does not have a copy constructor
    - Assignment function duplicates the pointer
    - **std::move()** transfers the pointer
    - Compared to **std::unique_ptr<T>**:
      – Slower
      – More overhead
      – Reduces multi-threaded performance through locks
  - Due to the ability to use automatic allocation and deallocation, raw pointers should only be used to refer to existing variables rather than to own a variable

– *Thread-local:* Having a lifetime of the given thread

5 C++ REST SDK

5.1 Overview

- C++ REST SDK: Microsoft library which allows for client-server asynchronous communication
- Microsoft Azure Storage Library: Microsoft library which interfaces with Microsoft Azure Storage Tables
  - *Microsoft Azure Storage Tables:* Key-value cloud database
    - *Key-value store:* Database consisting of a collection of arbitrary keys which are mapped to data values
    - Each key in the database consists of a partition name and a row name, and the entity mapped to by a key contains the values for the specific row
      - Values for the specific row are a set of properties, each of which consists of a name and value
- *Uniform Resource Identifier (URI):* Name of a specific resource on a network
  - URL path structure:
    - Consists of: *protocol://address:port/parameter/parameter[/...]*
      - E.g.
        http://localhost:34568/Operation/TableName/PartitionName/RowName
        http://localhost:34568/Operation/TableName/AuthenticationToken
        /PartitionName/RowName
    - **web::http::uri::decode()** converts from HTTP encoding to readable format
    - **web::http::uri::split_path()** converts from readable format to a vector of strings containing the necessary information

5.2 HTTP Requests

- HTTP request is represented by the class **web::http::http_request**
  - Components:
    - HTTP method: **http_request.method()**
    - URI: **http_request.relative_uri().path()**
  - Alteration methods:
    - Decoding URI from HTTP to internal format: **utility::string_t web::http::uri::decode()**
    - Splitting paths at / characters: **utility::string_t web::http::uri::split_path()**
  - Reply method: **http_request.reply(http::status_code)**
    - A JSON body may also be returned:
      http_request.reply(http::status_code, web::json::value)
      - Documentation for creating a JSON body