Project

Overall Project Organization

We use the first 9 weeks of the term for design and construction of projects. Week 10 is reserved for final presentations. A more complete schedule <schedule.php> is available.

Every team will work on two software projects. The first project is intentionally simple with respect to the code that must be developed; the goal is to allow team members to learn to work together, develop an understanding of the software development process, and get feedback on their work products. Feedback will be provided by peer-review as well as by the instructor. Students should complete Project 1 with a good understanding of the issues of collaborative software development and how these issues can be addressed through a disciplined software process.

Project 2 will be a more significant development effort that should produce a usable software application. Students may choose from among projects suggested by the instructor or may submit a proposal for a project of their own.

Project 1: A Simple Address Book

Objectives

For most teams, the hard part of this project will not be the coding, but organizing the team to work effectively together, and being clear on exactly what the team is building, in particular:

1. Requirements: getting clear on precisely what the software should do and communicating this to all team members.
2. Work Assignments: being precise about the tasks assigned to each person; exactly what is to be produced and when it is due.
3. Iteration: building the code up as a sequence of meaningful subsets of the eventual functionality in a way that can easily be extended.
4. Design: communicating the major software design decisions and their rationale.

The software to be designed is a program that can be used to maintain a simple address book. At a minimum, an address book holds a collection of entries, each recording a person's first and last names, address, city, state, zip, phone number and e-mail address.

Initial Requirements

It must be possible to add a new person to an address book, to edit existing information about a person, and to delete a person. It must be possible to sort the entries in the address book alphabetically by last name, or by ZIP code (with ties broken by first name if necessary).

It must be possible to create a new address book, to open an existing address book, to close an address book, and to save an address book to a file. File operations should be performed using standard *New*, *Open*, *Close*, *Save* and *Save As* ... File menu options. The program's File menu will also have a Quit option to allow closing all open address books and terminating the program.

Basic requirements call for the program to work with a single address book. A better implementation should allow for multiple address books to be open, each with its own window that can be closed separately. In this case, New and Open will result in creating a new window, without affecting the current window.

The program should keep track of whether any changes have been made to an address book since it was last saved, and offer the user the opportunity to save changes when an address book is closed either explicitly or as a result of choosing to create/open another, or quitting the program; i.e., it will not lose unsaved data without warning.

A more advanced implementation of the address book should support importing and exporting a set of addresses to other address books (e.g., to share addresses with another user). Import and export will be done using a standard, customer supplied, format (a TAB separated list of entry fields).

It is anticipated that the address book application will be routinely updated to add features, improve capabilities, or improve quality. It must be possible to extend and update the address book design as requirements change.

Quality Requirements:

A useful system must meet at least the following quality requirements:.

- Performance: the software should respond to user requests at a speed equal to or better than competing applications, in any event not to exceed 500ms.
- Reliability: the system should enter and retrieve or exchange properly formatted address book data without loss or error.
- Standards: the address book software should ensure that data entered is consistent with U.S. postal address standards.
- Consistency: it should be possible to edit the name (or other fields) at any time while keeping the associated data.
- Ease of change: it should be possible to make likely changes to the system without extensive re-design. Simple changes should require changes to only a single system component (*module*).

More Advanced Features (Likely Changes)

Once the basic functionality is implemented it is expected that the team will add one or more advanced features. The system should be designed so that such changes are easy to make. An advanced design will implement one or more of these features.

1. As the number of entries gets larger, we will want to be able to search the address book.
2. Ability to keep customized address books with different types of entries in each.
3. Support for user-defined fields.
4. Include links to social media like Facebook

Developmental Objectives

The first objective should be to build something the customer (played by the instructor) actually wants. In addition, speed of development and software quality are critical.

- A bug-free application with reduced capability that is delivered to the market on time is worth more (and will get a better grade) than a buggy or late application with lots of features. All project products should be reviewed against the Project Grading <projectGrading.php> rubric.
- It must be possible to divide the software into a set of distinct components that can be coded and tested concurrently.
- It must be possible to scale the delivered product to meet schedule.
- Likewise, a design that allows for features to be added easily is superior to one that satisfies initial requirements but is difficult to extend.

Suggested Subsets

Teams should employ an incremental development strategy that builds first an application with a small subset of the expected features. Subsequent increments should add additional features. Teams may decide which features to implement first but the following is a reasonable approach.

L0: Minimum Useful Subset

All teams should complete a minimum useful subset as follows:

- Implement basic interaction with graphic user interface (GUI). E.g., send data from the GUI to the back end and store it then recall data from back end and display it.
- Add an addresses to the address book
- Retrieve/view an address

L1: Basic Address Book

- Simple address book version including
  - Store standard postal address and one phone number
  - Sort addresses by name or zip
  - Edit, save, and recall addresses
- GUI for entering accessing address book operations

L2: Advanced Address Book

- Import/export standard-format (customer-supplied) files of addresses
- Support for one or more of the Advanced Features.

Constraints

It is required that at least two team members contribute to each of the significant artifacts. In particular, that two or more must contribute significantly to the code, work on the design, create the documentation, provide reviews, and test results. It should be clear from the schedule and developer log which team members contributed to each artifact.

E-mail: Email instructor
Based on the open source Sinorca design
with content from Prof. Michal Young and Prof. Anthony Hornoff