Software Development Assignments for a Software Architecture Course[*]

David Garlan and Mary Shaw

Department of Computer Science
5000 Forbes Avenue
Carnegie Mellon University
Pittsburgh, PA 15213

Draft of September 1, 1995
Submitted for Publication

Abstract

As software systems grow in size and complexity their design problem extends beyond algorithms and data structures to issues of system design. These issues—the *software architecture* level of software design—are becoming increasingly important to the practicing software engineer. Consequently, it is important to find effective ways to teach this material. To meet this need we developed a course, "Architectures for Software Systems," and have taught it four times. In this paper we describe the principal software development assignments that this course uses to develop skill at applying architectural principles to the design and implementation of software systems.

The major challenges in designing such assignments are (1) making sure that students spend their time on architectural issues rather than coding, and (2) helping students establish and maintain a desired architectural style. We address these issues by providing working examples as starting points. These examples are usable in other courses.

Keywords: *software engineering education, software architecture, programming assignments, software development exercises, software design*

[*]Development of this course and its content was funded in part by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grants MDA972-92-J-1002 and F33615-93-1-1330; by National Science Foundation Grant CCR-9357792; and by a grant from Siemens Corporate Research. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of Wright Laboratory, the US Department of Defense, the United States Government, the National Science Foundation, or Siemens Corporation. The US Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation thereon.

1 Introduction

The software component of the typical undergraduate curriculum emphasizes algorithms and data structures. These curricula include courses on particular system organizations such as those used for compilers, operating systems, or databases are usually offered. However, there is no systematic treatment of the organization of modules into systems or of concepts and techniques at an architectural level of software design. At this level of design the important considerations are the high-level structure of a system, the use of common patterns for system organization, the assignment of responsibility to design elements, and reasoning about the overall conceptual integrity of a system [GS93].

As a result, students now face a large gap between lower-level courses, in which they learn programming techniques, and upper-level project courses, in which they are expected to design more significant systems. Without knowing the alternatives and criteria that distinguish good architectural choices, the already-challenging task of defining an appropriate architecture becomes formidable.

Students with only the conventional background are at a disadvantage when theymust design and develop large, practical software systems. Not only will they lack techniques for making good design decisions, they may even be unaware that critical decisions are being made by default.

2 The Software Architecture Course

We have developed a course that helps to bridge this gap: *Architectures for Software Systems* [GSO$^+$92]. Specifically, the course:

- teaches how to understand and evaluate designs of existing software systems from an architectural perspective,
- provides the intellectual building blocks for designing new systems in principled ways using well-understood architectural paradigms,
- shows how formal notations and models can be used to characterize and reason about a system design, and
- presents concrete examples of actual system architectures that can serve as models for new designs.

This course was developed for senior undergraduates and students in a professional master's program for software engineering. We have now taught it four times.

Our experience has been that certain principles and techniques of good architectural design for software can be taught effectively. However, a critical component of this education is the use of hands-on development projects in which students directly engage architectural design issues.

3 Software Development Assignments

In order to give students practical experience with developing systems from an architectural point of view, ideally we would like to find assignments that

- are concerned with real systems,

- can be completed in a couple of weeks,
- are rich enough to admit variety of solutions, and
- allow students to make meaningful comparisons between different architectural styles.

To meet these goals we designed three two-week assignments. All three assingments are based on the same core design problem. Eachassignment requires the students to explore issue of implementing that problem in a specific style. For the initial course offerings, we assigned an object-oriented, a pipe-and-filter, and an event-driven design.

The major issues we faced in designing these assignments were (1) making sure that students spend their time on architectural issues rather than coding and (2) helping students establish and maintain the desired architectural style. We addressed both concerns by organizing each assignment as an extension of an initial implementation that we provided.

Each task is designed to allow students to become familiar with a particular architectural style. For each task, we supplied an implementation in the required style that used several components from an available collection. The assignment required students to understand the implementatin of a simple version of the problem in that style, then extend the implementation *in the same style* by reconnecting parts, using other components, or minimally changing components.

The choice of this assignment format was driven by two guiding principles: First, the attention of the students should be focused at the architectural level rather than at the algorithms-and-data-structures level. (Students should already know how to do the latter.) Second, it is unreasonable to expect the accurate use of an unfamiliar style without providing illustrative code employing that style. A pleasant side-effect of this choice of format was that problems more closely resembled software maintenance/reuse than building a system from scratch. In addition, our students enter the course with a considerable diversity of programming language background. It's easier for them to work in an unfamiliar language if they have a working starting point.

A major objective of this course is for students to leave with an understanding of how certain properties of a given problem can make a particular architectural choice appropriate or inappropriate. To do this, chose variations of the single core problem that illustrated the relative advantages of the three styles. By assigning the same basic problem for each architectural style, we avoided the risk of students associating problem class X with architectural style Y—instead promoting understanding of the features of each problem that should lead the designer to choose that style. By varying the features related to the architectural choice, we also discouraged students from leaving each solution in the same basic architectural style, adding only the superficial trappings of the second style. For example, by changing the requirements on the system, we ensured that an object-oriented solution would not merely be a pipes-and-filters solution "dressed up" to look like an event-driven system.

To encourage cooperation and to balance unfamiliarity with particular programming languages and systems, students worked in small groups over a period of about two weeks. However, each student was required to independently answer a set of questions on the design issues in the task.

The assignments involved not only the production of a working system but also the analysis of the architectural style in which it was constructed. To help students do their analysis we held design reviews halfway through each assignment. These reviews were presented by the students in the class, with each team making one presentation sometime during the semester. The reviews were not graded; they thereby provided a means for the class to engage in discussions about the architectural style and for the instructors to guide the student solutions (both those being presented and those of the students watching the presentation) by asking pointed questions.

The remainder of this paper gives the three assignments that we used when we taught the course in Spring 1994.

References

[GS93] David Garlan and Mary Shaw. An introduction to software architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, pages 1–39, Singapore, 1993. World Scientific Publishing Company. Also appears as SCS and SEI technical reports: CMU-CS-94-166, CMU/SEI-94-TR-21, ESC-TR-94-021.

[GSO+92] David Garlan, Mary Shaw, Chris Okasaki, Curtis Scott, and Roy Swonger. Experience with a course on architectures for software systems. In *Proceedings of the Sixth SEI Conference on Software Engineering Education*. Springer Verlag, LNCS 376, October 1992. Also available as CMU/SEI technical report, CMU/SEI-92-TR-17, under the title "Experience With a Course on Architectures for Software Systems—Part I: Course Description".

A Assignment 1: KWIC Using an Object-Oriented Architecture

A.1 Description of the problem

This assignment is to implement an interactive version of the KWIC index system (described in Parnas's *On the Criteria To Be Used in Decomposing Systems into Modules*) in the object-oriented paradigm. You will be provided with a partial Ada implementation of the system and asked to identify and make the necessary modifications.

The provided system is simply a line alphabetizer. It interactively inputs a line at a time and upon demand outputs an alphabetized list of the current collection of lines. Here is a transcript of a sample session:

```
Add, Print, Quit: a
> O my son Absalom
Add, Print, Quit: a
> my son my son
Add, Print, Quit: a
> and the king cried
Add, Print, Quit: a
> in a loud voice
Add, Print, Quit: p
        O my son Absalom
        and the king cried
        in a loud voice
        my son my son
```

```
Add, Print, Quit: a
> O Absalom
Add, Print, Quit: p
        O Absalom
        O my son Absalom
        and the king cried
        in a loud voice
        my son my son
Add, Print, Quit: q
```

Your assignment is to modify the existing code to support the following changes:

- Rather than simply outputting an alphabetic list of all the lines, the *Print* command should output an alphabetic list of the circular shifts of all the lines. However, shifts (including the nullary shift) which result in a line beginning with a trivial word—*a*, *an*, *and*, *the*, or the capitalized versions of these words—should be omitted.

  3 different ways of displaying should be supplied. Upon entery of a `p` at the command line, the system should print another "menu line":

  ```
  Add, Print, Quit: p
  Simple, Aligned, Concordance:
  ```

  1. *Simple Display:* Upon entery of a `s` at the command line the system should print the alphabetic list of all line shifts, as described above.
  2. *Aligned Display:* Instead of printing the shifted line, the original sentence is printed, but the word that is at the begining of the shifted sentence is aligned, and capitalized. For example, the line "and the king cried" is printed as follows:

     ```
     and the king CRIED
          and the KING cried
     ```

  3. *Concordance Style Display:* For each shifted variant of the sentence the original sentence is printed, but the word at the beginning of the shifted variant is abbreviated. The line "and the king cried" is printed as follows:

```
        and the king c.
        and the k. cried
```

- three commands, *Original*, *Delete* and *Count* , should be added.

  1. *Original:* Upon entry of an `o` at the command line, the system should output a list of all lines entered by the user (including lines beginning with trivial words, but not including the circular shifts of lines) in their original order.
  2. *Delete:* Upon entery of a `d` at the command line, the system should prompt for a line (much like the Add command). This line should then be deleted from the system.
  3. *Count:* on entery of a `c` at the command line, the system should display the number of original lines currently in the system.

Here is a sample session of the new system:

```
Add, Print, Original, Delete, Count, Quit: a
> and the king cried
Add, Print, Original, Delete, Count, Quit: a
> in a loud voice
Add, Print, Original, Delete, Count, Quit: p
Simple, Aligned, Concordance: s
        cried and the king
        in a loud voice
        king cried and the
        loud voice in a
        voice in a loud
Add, Print, Original, Delete, Count, Quit: c
        2 lines
Add, Print, Original, Delete, Count, Quit: d
> and the king cried
Add, Print, Original, Delete, Count, Quit: c
        1 lines
Add, Print, Original, Delete, Count, Quit: p
Simple, Aligned, Concordance: c
        i. a loud voice
        in a l. voice
        in a loud v.
Add, Print, Original, Delete, Count, Quit: o
        in a loud voice
Add, Print, Original, Delete, Count, Quit:q
```

A.2 The current system

The current system is decomposed into the following modules:

- `words`,
- `lines`,
- `line_collections`, and

- `tree_binary_unbounded_managed`.

In addition there is a top-level module (`session`) which provides the command-line interface.

The source code for the current system will be made available to you by next class period. Watch the class bulletin board for instructions on how and where to obtain the code, along with instructions on accessing an Ada compiler.

A.3 Discussion

On Wednesday, February 2, two or three teams will briefly present their initial designs for class critique and discussion. Volunteers for this presentation will be solicited during the previous class period. Note that each team will be responsible for one such presentation over the course of the three assignments.

This presentation/discussion will not be graded. It is solely for the benefit of you and your classmates.

A.4 Due date and electronic hand-in

The assignment is due by 10:30 am on Wednesday, February 9. You should create a directory called "sa" in one of the team members home directory, and a subdirectory called "hw1". In "sa/hw1" prepare a file called "kwic.doc". This file should contain:

- the names of both team members,
- a list of the modules added/modified and for each such module a list of the resources added/modified.

The directory should also contain the system (source files, especially of all modules modifies/added, and an executable file, named "session").

You should email a pointer to that directory (machine name, user name) to the Teaching Assistant by the due date. After that, none of the files in the directory should be touched. In addition there will be a written commentary (due at the beginning of class on Wednesday, February 9) answering the following questions:

1. Describe the architecture of your system (both the provided part and the parts you added), explaining how it is an example of an object-oriented architecture, and in what ways (if any) it deviates from the basic object-oriented style. For each of the new functionalities required, descibe how your system implements it. Justify your design.
2. For each of the changes you made, explain if the change was of the internals of one of the system components (data structures or algorithms) or of the system architecure.
3. What changes would you have to make to your system to change the representation of line storage? What other components would be affected?
4. What changes would you have to make to your system to add the functionality of only showing lines that start with a particular word?
5. Does the system lend itself to a distributed implementation? If so what changes would have to be made to make it function this way?

The commentary should be your own work: i.e., individuals, not teams for commentary.

A.5 Grading criteria

Your solutions and commentary will be graded by the following criteria:

- Whether or not the resulting system performs as required.
- Use of architectural style in the assignment.
- Your understanding of the kinds of changes easily supported by the architecture.

In particular, the grade will be broken down as follows (100 points maximum):

- the program: 40 points,
- question 1: 20 points,
- questions 2-5: 10 points each.

A.6 Further questions

If you have any further questions, feel free to contact any of us via e-mail or during our office hours. Clarifications (if any) will be posted to the class bulletin board.

B Assignment 2: KWIC Using a Pipe-Filter Architecture

B.1 Description of the problem

This assignment is to implement an interactive version of the KWIC index system (described in Parnas's *On the Criteria To Be Used in Decomposing Systems into Modules*) in the **Pipe-Filter** paradigm. You will be provided with two implementations of the KWIC system - one in Unix shell commands and one in C. You will be asked to extend these implementations with new functionality.

Both versions of the current system accept input at the command line and produce output to the terminal screen. Both versions implement a pipe and filter system that shifts and sorts the input, and then transforms to upper case letters the first word in each line (look at the first example below).

You will be provided with the source code for these systems, as well as source code for two utility programs, *diverge* and *converge* - one to split an input stream and one to join two input streams.

The source code for the current system will be made available to you by next class period. Watch the class bulletin board for instructions on how and where to obtain the code.

Your assignment is to modify the existing code to support the following changes:

1. Extend the shell script version of the system to produce a KWIC index of the login and user names of all users currently logged on a system. Hint: look at *finger* and *cut* and *tail*.

2. Do the same with the C language implementation.

3. Modify the shell script version of the system to produce a KWIC listing that contains no duplicate entries. Hint: look at *uniq*.

4. Modify the C language version of the system to produce a KWIC index in which the login names of users appear as separate entries from the users' real names. Define a set of "trivial" login names to contain "john","smith", "david" and your own login name. In the final output, only nontrivial login names should appear, and only the first and last name of each real user name should appear. (i.e remove middle initials or middle names). Hint: Use the *diverge* and *converge* programs provided. You might find the C function "nxtarg" usefull for some of the C functions you will have to write, or look into diverge.c to see how parsing words is easily done.

```
Here are sample outputs for the solution to each part of the problem:
(The finger part is a real snapshot of some machine, therefore the choice
of names has no deep meaning)

% solution1.csh and solution2
BENNETT jcrb John C R
C R Bennett jcrb John
CERIA santi Santiago
DAFNA Talmor tdafna
DAFNA Talmor tdafna
EHT Eric Thayer
```

```
ERIC Thayer eht
GALMES pepe Jose M
JCRB John C R Bennett
JOHN C R Bennett jcrb
JOSE M Galmes pepe
M Galmes pepe Jose
PEPE Jose M Galmes
R Bennett jcrb John C
SANTI Santiago Ceria
SANTIAGO Ceria santi
TALMOR tdafna Dafna
TALMOR tdafna Dafna
TDAFNA Dafna Talmor
TDAFNA Dafna Talmor
THAYER eht Eric

% solution3.csh
BENNETT jcrb John C R
C R Bennett jcrb John
CERIA santi Santiago
DAFNA Talmor tdafna
EHT Eric Thayer
ERIC Thayer eht
GALMES pepe Jose M
JCRB John C R Bennett
JOHN C R Bennett jcrb
JOSE M Galmes pepe
M Galmes pepe Jose
PEPE Jose M Galmes
R Bennett jcrb John C
SANTI Santiago Ceria
SANTIAGO Ceria santi
TALMOR tdafna Dafna
TDAFNA Dafna Talmor
THAYER eht Eric

Remark: TRIVIAL_NAMES = {'tdafna','john','smith','david'}
% solution4
Bennett John
Ceria Santiago
Dafna Talmor
Dafna Talmor
eht
Eric Thayer
Galmes Jose
jcrb
John Bennett
```

```
Jose Galmes
pepe
santi
Santiago Ceria
Talmor Dafna
Talmor Dafna
Thayer Eric
```

B.2 Discussion

On Wednesday, February 16, one teams will briefly present their initial designs for class critique and discussion. Volunteers for this presentation will be solicited during the previous class period. Volunteers will be drawn from those groups that did not present designs for assignment 1.

This presentation/discussion will not be graded. It is solely for the benefit of you and your classmates.

B.3 Due date and electronic hand-in

The assignment is due by 10:30am on Wednesday, February 23. You should e-mail your solution to the Teaching Assistant by that time. Your solution should consist of

- the names of team members,
- the directory holding the solution.

Your directory should contain 4 text files (besides the c or csh files): "solution1","solution2", "solution3" and "solution4". Each one should a list of the files you use for the solution, with an indication which file is changed or new. All your changes should be well documented within the files.

In addition, there will be a written commentary (due at the beginning of class on Wednesday, February 23) answering the following question:

1. How can the efficiency of the "no duplicates" implementation be changed by using the *sort* and *uniq* filters at different points in the system? (The sorting algorithm has O(n log n) complexity).

The commentary should be your own work: i.e., individuals, not teams for commentary.

B.4 Grading criteria

Your solutions and commentary will be graded by the following criteria:

- Whether or not the resulting system performs as required.
- Use of architectural style in the assignment.
- Your understanding of the implications of changes made to the system architecture.

In particular, the grade will be broken down as follows (100 points maximum):

- the program: 80 points, and
- question 1: 20 points.

B.5 Further questions

If you have any further questions, feel free to contact any of us via e-mail or during our office hours. Clarifications (if any) will be posted to the class bulletin board.

C Assignment 3: KWIC Using an Implicit Invocation Architecture

C.1 Description of the Problem

This assignment, once again, is to implement an interactive version of the KWIC index system (described in Parnas's *On the Criteria To Be Used in Decomposing Systems into Modules*) in the implicit invocation paradigm. You will be provided with a partial Ada implementation of the system and asked to identify and make the necessary modifications.

The provided system, as in Assignment 1, is simply a line alphabetizer. It interactively inputs a line at a time and upon demand outputs an alphabetized list of the current collection of lines. Unlike the first assignment this version also allows a delete command. Here is a transcript of a sample session:

```
Add, Delete, Print, Quit:
a
> Star Wars
Add, Delete, Print, Quit:
a
> The Empire Strikes Back
Add, Delete, Print, Quit:
a
> Return of the Jedi
Add, Delete, Print, Quit:
p
Return of the Jedi
Star Wars
The Empire Strikes Back
Add, Delete, Print, Quit:
d
> Star Wars
Add, Delete, Print, Quit:
p
Return of the Jedi
The Empire Strikes Back
```

Your assignment is to modify the existing code to support the following changes:

1. Rather than simply outputting an alphabetic list of all the lines, the *Print* command should output an alphabetic list of the circular shifts of all the lines. However, shifts (including the nullary shift) which result in a line beginning with a trivial word—*a, an, and, the* and the capitalized versions of these words—should be omitted.

2. On a *Print* command the system should also print a counter of the number of original lines added by the system.

Here is a sample session of the new system:

```
Add, Delete, Print, Quit:
a
> Star Wars
Add, Delete, Print, Quit:
a
> The Empire Strikes Back
Add, Delete, Print, Quit:
a
> Return Of The Jedi
Add, Delete, Print, Quit:
p
-- Number of Original Lines:   3--
Back The Empire Strikes
Empire Strikes Back The
Jedi Return Of The
Of The Jedi Return
Return Of The Jedi
Star Wars
Strikes Back The Empire
Wars Star
Add, Delete, Print, Quit:
d
> Star Wars
Add, Delete, Print, Quit:
p
-- Number of Original Lines:   2--
Back The Empire Strikes
Empire Strikes Back The
Jedi Return Of The
Of The Jedi Return
Return Of The Jedi
Strikes Back The Empire
```

C.2 The Current System

The current system is decomposed into the following modules:

- `Words`
- `Lines`
- `Line_Collections`
- `Alphabetized_List`
- `KWIC_Session`

In addition, the following additional modules will be used in the final system (they have already been written for you):

- `Shifter_1`
- `Shifter_2`
- `Trivial_Eater`

There is also a file, called `event_bindings.ada` which contains the bindings from events to methods. **To complete your solution, you should modify this file only, and add one new module.**

You will also need to generate the event manager itself. This is automatically generated from the event description language embedded in the Ada code and in `event_bindings.ada`. To generate the event manager, type:

```
make_events *.ada
```

This will create two files: `event_manager.ada` and `event_manager.adb`. They should be compiled into your system as well.

The format of the event description language is as follows:

- All lines in the event description language are preceeded by the `--!` symbol. This symbol indicates to Ada that these lines are to be ignored, and to the event description language processor (which is made primarily of `awk` scripts) that these lines are to be processed. Note that `event_bindings.ada` contains nothing other than lines in the event description language.
- Each section of the event description language is bracketed by two lines that indicate what package the enclosed declarations are associated with. These lines are:

  ```
  --! for <package_name>
  --!  ...
  --! end for <package_name>
  ```

  where `<package_name>` represents the Ada package name of the associated package. All other declarations go between these two statements (where the ellipsis is).
- To create a new event in the system, include a `declare` statement of the form:

  ```
  --!   declare <event_name> <args>
  ```

  where

  - `<event_name>` represents the name of the event, and

  - `<args>` represents the data associated with that event (if any). Each argument is of the form:

    ```
    <identifier> : <type>;
    ```

    where

    * `<identifier>` is an Ada identifier for the datum, and
    * `<type>` is the Ada type name of the type of the datum.

  All of the event declarations required for this system are included in the specifications of the various packages provided. You should not have to add any on your own.

- Bindings from an event to a method associated with that event can be found in `event_bindings.ada`. For each binding, the following format is used:

  ```
  --!  when <event_name> => <method_name> <argnames>
  ```

  where

  - `<event_name>` represents the name of the event upon whose announcement the method should be called.
  - `<method_name>` represents the name of the procedure (within the package specified by the `for` statement) which should be called when the event is announced.
  - `<argnames>` is a list of the identifiers of data associated with the event in a `declare` statement which are to be passed to the procedures. You do not have to pass every argument, nor do you need to pass them in the same order they are defined. However, every name which appears in `<argnames>` must have been part of the event declaration.

When a component wishes to announce an event, it calls `Announce_Event`, signaling the name of the event and any parameters. (All calls to `Announce_Event` have already been provided in the code. It will help you in your solution to know that this particular implicit invocation system guarantees that whatever activity was caused by the event announcement is complete when the `Announce_Event` procedure returns, so that there are no pending events in the system once the call returns.

C.3 Discussion

On Monday March 7, a team will briefly present their initial designs for class critique and discussion. Volunteers for this presentation will be solicited during the previous class period. Note that each team will be responsible for one such presentation over the course of the three assignments.

This presentation/discussion will not be graded. It is solely for the benefit of you and your classmates.

C.4 Due Date and Electronic Hand-In

The assigment is due by 10:30am on Monday March 14. You should e-mail your solution to the Teaching Assistant. Your solution should include:

- the names of your team members,
- a pointer to a directory containing a modified source of `event_bindings.ada`, the added module, and a running system.

In addition, there will be a written commentary (due at the beginning of class on March 14) answering the following questions:

1. Are implicit systems easier or harder to modify than object-oriented architectures? Why? Describe specific modifications (other than the one which you performed) which would be easier in an implicit invocation system, and other modifications which would be harder.
2. Could the system specified be implemented using a dataflow architecture? If so, how? If not, why not?
3. Explain how your implementation differs from the one proposed in the paper by Garlan, Kaiser, and Notkin for handling trivial line removal. Would that have been a better approach? If so, why? If not, why not?
4. The implicit invocation system provided by `make_events` assures that all events which are caused by a single `Announce_Event`, whether directly or indirectly, are all complete and all methods called before the `Announce_Event` call returns. Identify any differences in your solution which would have been caused if the system delivered the events in arbitrary order, and did not guarantee their delivery prior to returning from an announcement.
5. Does your system handle line deletions properly? Defend.

The commentary should be your own work; i.e., individuals, not teams for commentary.

C.5 Grading Criteria

Your solutions and commentary will be graded by the following criteria:

- Whether or not the resulting system performs as required.
- Use of architectural style in the assignment.
- Your understanding of the kinds of changes easily supported by the architecture.

In particular, the grade will be broken down as follows (100 points maximum):

- the program: 40 points,
- questions 1-5: 12 points each.

C.6 Further Questions

If you have any further questions, feel free to contact any of us via e-mail or during our office hours. Clarifications (if any) will be posted to the class mailing list.