University of Waterloo
Department of Electrical and Computer Engineering

Final Examination E&CE 355
Software Engineering Fall 2003

2:00 Sat. Dec. 13, 2003 **180 minutes**
Instructors: N. Young, M. Hembruch
NO ADDITIONAL MATERIAL ALLOWED
NO CALCULATORS ALLOWED

NOTES:

Answer all questions.

Guesses on two-valued questions have an expected value of zero.

Question weights are indicated in brackets [...].

Proctors and TA's are NOT allowed to answer questions. Instructors will only correct obvious typos or other problems with the exam. They will not answer questions.

If information appears to be missing from a question, make a reasonable assumption, state your assumption, and proceed. Do not simplify the question.

Attempt to answer questions in the space provided. If necessary, you may use the back of another page. If you do this, please indicate it clearly.

If you separate the pages, make sure your initials are on the top of every page.

The last page includes copies of figures for questions 2, 4, 8 and 9. Tear the last page off, for quick reference. Do not write answers on this last reference page.

Hand in all pages except possibly the last page.

NAME _____

Signature: _____

SN: _____

1. Software lifecycles		15
2. SDL and MSC		25
3. Real time		15
4. Design patterns		20
5. Brook's Complexity		15
6. Build tools		10
7. V&V		15
8. Cyclomatic complexity		20
9. Coverage		30
10. Integration testing		15
Total		**180**

1. Software lifecycles [15]

We discussed two software development lifecycles in class, *waterfall* and *incremental*.

Our guest speaker, Mauricio De Simone of Nitido, made the following statements:

> Build the bones, then put on the meat.
> Have a small team that puts together the bones.
> Bring development muscle once the bones are in place.

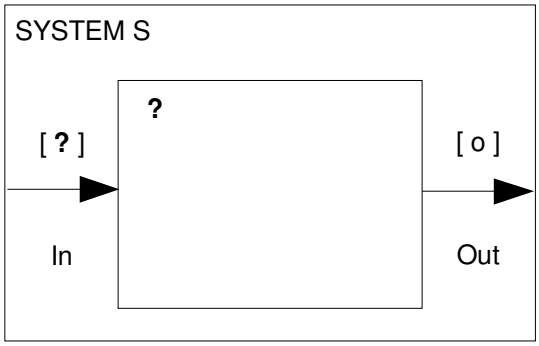
a) [5] Circle "waterfall" or "incremental" indicating whether Mauricio is describing the waterfall or incremental lifecycles. In two or three sentences, briefly explain your answer.

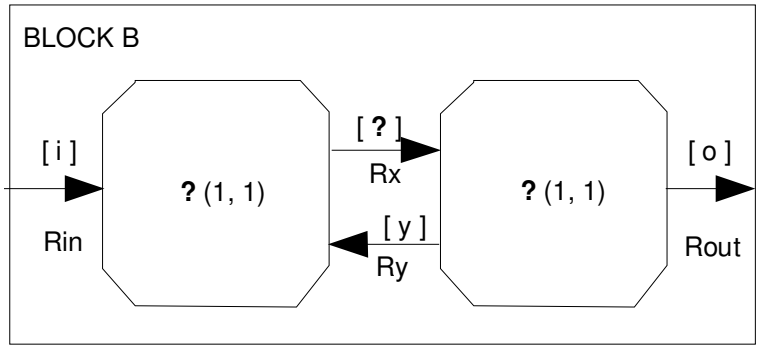
Waterfall Incremental

b) [10] Sketch the software development lifecycle that you circled in part (a) Label all blocks and arcs.

2. SDL and MSC [25]

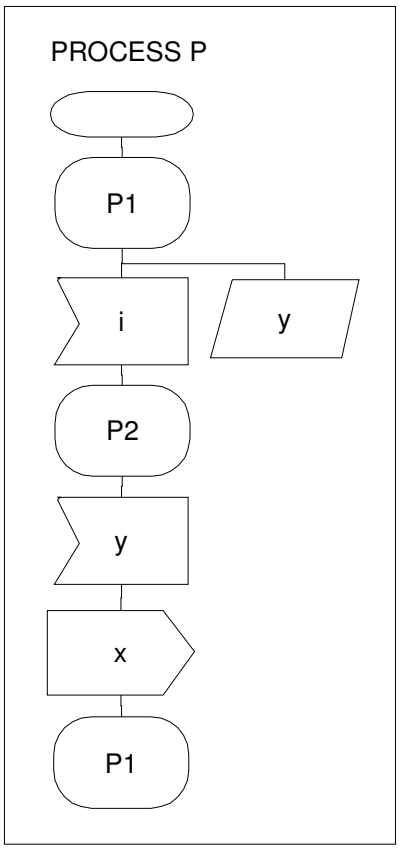
a) [5] The Specification and Description (SDL) figures shown below describe a small system, S. The five question marks (?) show where labels are missing. Complete the figures by replacing the question marks with the correct labels.

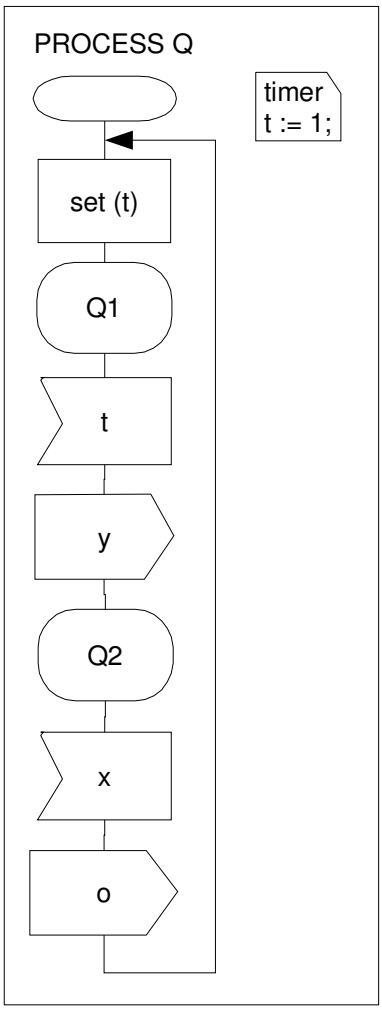




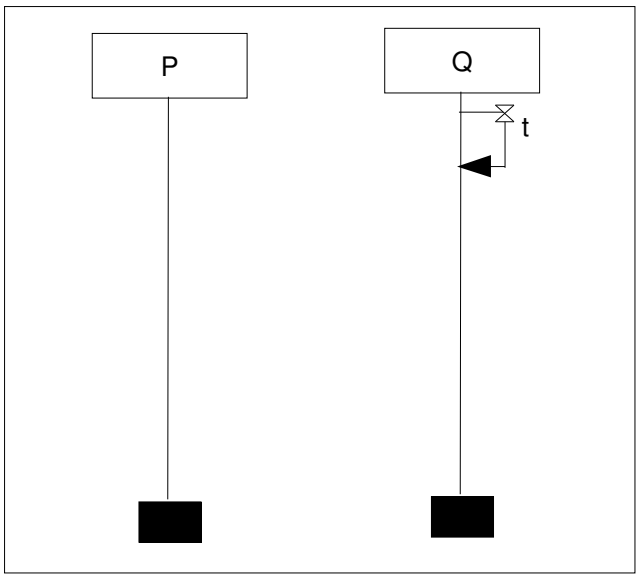
b) [5] Complete the following table.

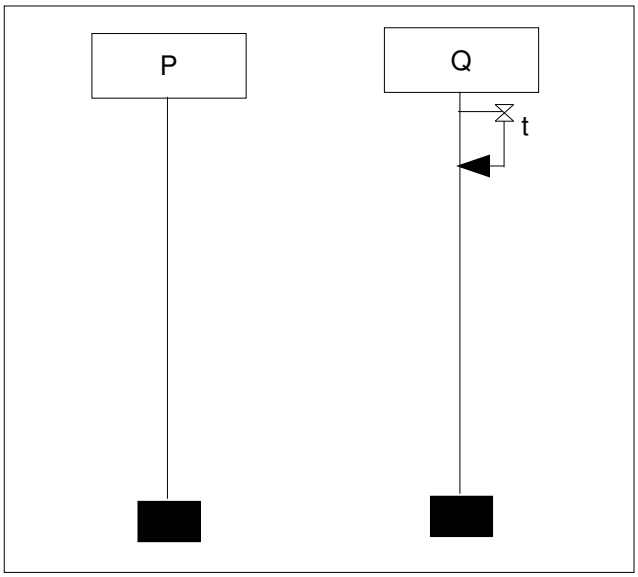
Number of blocks	
Number of signals	
Number of states in process P	
Number of inputs in process P	
Number of outputs in process Q	





c) There are at least two message sequences for the system S that produce exactly one output message ("o") to the environment. Complete the two Message Sequence Charts (MSC's) shown below. showing different message sequences that produce exactly one output of the signal "o".





3. Real time [15]

a) [3] In one or two sentences, define the term real-time requirement as we discussed it in class.

_____

_____

b) [12] The following table lists four software applications. Classify each application by circling "Not", "Soft" or "Hard", to indicate whether the application includes real-time requirements. Explain each answer with a sentence or two.

Application / Real time	***Explanation***
1. Stop light, i.e., 3-coloured traffic light (red, yellow, green) Not Soft Hard	
2. Word processor Not Soft Hard	

Application / Real time	*Explanation*
3. Missile flight surface control Not Soft Hard	
4. Phantom Dialer PBX program Not Soft Hard	

4. Design patterns [20]

The following C++ code fragments show an example of a design pattern discussed in lecture.

```
class ioPattern
{
   public:
      virtual file *newfile(String filename);
      virtual shmem *newShmem(int key);
      virtual msgq *newMsgq(int key);
}

class unixIoPattern: public
{
   public:
      unixIoPattern() {
      file *newFile(String filename)
         {
            if (fileType == 1)
               return new unixFile(filename);
            else
               return new unixPipe(filename);
         }
      shmem *newShmem(int key)
         { return new unixShmem(key); }
      ...
}
```

```
...
main()
{
 ioPattern *iof;
 #ifdef UNIX
    iof = new unixIoPattern();
 #ifdef WINDOWS
    iof = new winIoPattern();
 #elseif QNX
    iof = new qnxIoPattern();
 #endif
 file xyz = iof->newFile("abc");
}
```

a) [3] Name the design pattern that this example illustrates.

b) [3] How many instances of the design pattern are created during the execution of `main()`?

c) [5] Files (`file`), shared memory (`shmem`) and message queues (`msgq`) are examples of resources. List the names of the classes to change and/or add to extend this example for a socket resource?

Classes changed _____ Classes added _____

d) [6] List the names of the classes that you would need to change and/or add to extend the original example (i.e., without the socket resource) for the Apple Macintosh ("mac") operating system?

Classes changed _____ Classes added _____

e) [3] The use of the `#ifdef` directive shown in this example illustrates a common technique for implementing a product family, i.e., for using the same source code to generate a similar but distinct products for multiple different operating systems. We also discussed this technique in the lectures on configuration management. What name did we call the different members of a product family?

_____

5. Brooks' complexity [15]

Our guest speaker, Mauricio De Simone of Nitido, referred to Brooks' two categories of complexity. Mauricio defined the categories as follows:

Category A: Complexity arising from our choice of tools to use in solving a problem

Category B: Complexity inherent in the problem itself

Mattias Hembruch also used the same two catgories to explain the purpose of the Requisite Skills Package (RSP) portion of the project, where you implemented the Phantom Dialer (PhD).

a) [3] Brooks, De Simone and Hembruch used specific names for the categories. Name the categories.

Category A _____ Category B _____

b) [6] On the back of the facing page, in one or two short paragraphs, explain the purpose for the Requisite Skills Package (RSP) using Brooks' two categories. (If you don't know the specific terms, just use "Category A" and "Category B".)

Write answer on back of the facing page.

c) [4] Circle "agree" or "disagree", depending on whether you agree that the RSP fulfilled its purpose for your project team. In about two or three sentences, explain why you agree or disagree. Refer to your project team's actual experience.

Agree Disagree

6. Build tools [10]

Assume the file, `t.c`, shown at right. Also, assume the function `test()` in file `test.c` as shown on the last page (and as mentioned again in the cyclomatic complexity question).

```
/* t.c */

#include <stdlib.h>
extern void test(int a, int b);
int main(int argc, char *argv[])
{
   if( argc == 3 )
      test( atoi(argv[1]), atoi(argv[2]));
}
```

a) [6] Complete the Makefile shown at right such that it will compile the program "`t`" from `t.c` and `test.c`. Use the command `gcc` to compile and/or link the files. Do not assume the use of built-in makefile rules.

```
### Makefile to compile t

all: t

t: t.o test.o
```

b) [4] Assume `t.c` and `test.c` have been checked into a version control system which creates a repository file in the current directory with a "`.vc`" extension (that is, `test.c.vc` and `t.c.vc`). Assume the command "`cmd update <filename>`" will retrieve the latest version of the file. Write the additional Makefile rules to automatically check that the working copies of `t.c` and `test.c` are the latest version.

```
### Makefile to retrieve latest versions
```

7. V&V [15]

a) [3] In one or two sentences, define verification.

b) [3] In one or two sentences, define validation.

The Cleanroom software development method emphasizes fault prevention over failure detection. All failures of the product are traced to faults in the development process: the failed product is discarded; the process fault is isolated and corrected. The Cleanroom method was created by Harlan Mills at IBM Federal Systems and has been used at organizations such as NASA, Ericsson and Texas Instruments.

c) [9] The following table outlines three important aspects of the Cleanroom method. For each aspect, provide two pieces of information. First, circle "static" or "dynamic" indicating whether the aspect applies static or dynamic verification or validation. Second, write about two or three sentences explaining the benefit that the aspect brings to the development process and/or product quality.

Cleanroom	***Benefits explanation***
1. Individual developers are not allowed to compile or run their own modules. Each developer is responsible for their own modules' entire syntactic and semantic correctness, which the developer achieves purely by inspection. Static Dynamic	
2. Modules are specified as the functional composition of other modules, through the elementary constructs of sequence, alternation and iteration. Each module's correctness is verified by mathematically proving that the module implements its specified function. Static Dynamic	
3. Completed systems are validated through tests that are selected according to statistical models of how the system will be used. Features that are used more often are tested more than features that are used less often. Static Dynamic	

8. Cyclomatic complexity [20]

a) [12] In the space below, draw the control flow graph for the C function, `test()`. Label the edges to show which condition each edge represents. (Assume short-circuit evaluation, i.e., the same as in class.)

```
    /* test() function */
    void test(int a, int b)
    {
       int i, x, y;

01     x=a*b+2;
02     y=(b+a)*5;

03     if( (a<0) && (b<0) ) return;

04     for(i=0;i<4;i++)
       {
05        if( (a>b) && (x>a) )
06           printf("Case 1!\n");
07        else if( (b>a) && (y>b) )
08           printf("Case 2!\n");
09        else printf("Case 3!\n");
10       y++;
11       x--;
       }
    }
```

b) [4] The cyclomatic complexity, V(G), can be calculated by counting the regions of the graph. Label the regions of the graph, R1, R2, etc., and state the cyclomatic complexity.

V(G) =

c) [4] In addition to counting the regions of the graph, we discussed two other methods of calculating the cyclomatic complexity. Choose either method and calculate the cyclomatic complexity again. Clearly show all steps in detail, including the original equation and variable substitutions.

V(G) =

Draw your control flow graph here, or on the back of the facing page.

9. Coverage [30]

Consider the C function, `test()`, in the previous question, and the three test cases listed in the table.

a) [20] Complete the table by listing the statements and edges covered by each of the test cases. The first test case is done for you, as an example.

b) [5] Circle "yes" or "no", indicating whether the set of three cases give statement coverage of the program. If you circle "no", write at least one statement not covered.

c) [5] Circle "yes" or "no", indicating whether the set of three cases give edge coverage of the program. If you circle "no", write at least one edge not covered.

Case	***Statements***	***Edges***
(a = 0, b = 0)	01, 02, 03, 04, 05, 07, 09, 10, 11	a >= 0, a <= b, b <= a, i < 4, i >= 4
(a = 2, b = 2)		
(a = 9, b = 1)		
Coverage	Yes No	Yes No
Not covered		

10. Integration testing [15]

Our guest speaker, Mauricio De Simone of Nitido, made the following statements.

> Integration is the hardest part.
> Independently test layers. Divide and conquer the layers of your system.

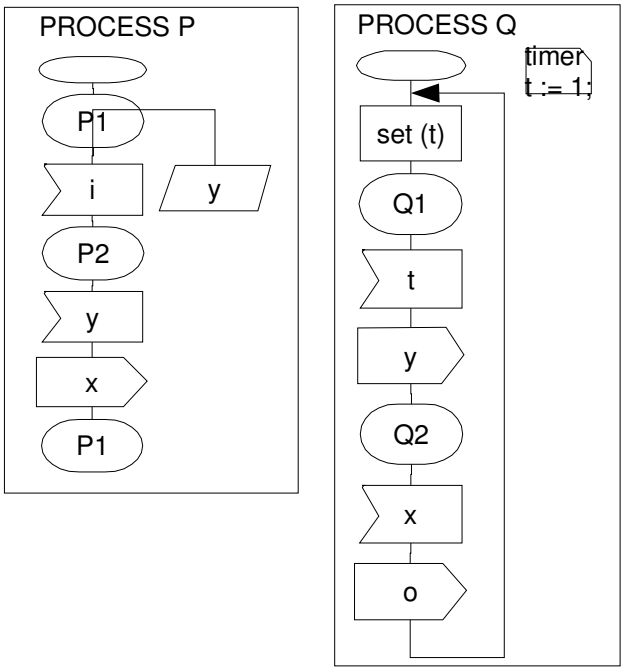
a) [2] Define the term “integration”.

b) [6] In a short paragraph, explain why Mauricio would say “Integration is the hardest part.” Give at least two reasons. Use the the concepts and terms we discussed in class.

c) [2] Circle “yes” or “no” indicating whether Mauricio recommends performing integration testing separately from system testing. In one or two sentences, briefly explain your answer.

Yes No

2. SDL and MSC



8. Cyclomatic Complexity & 9. Coverage

```
    /* test() function */
    void test(int a, int b)
    {
       int i, x, y;

01     x=a*b+2;
02     y=(b+a)*5;

03     if( (a<0) && (b<0) ) return;

04     for(i=0;i<4;i++)
       {
05        if( (a>b) && (x>a) )
06           printf("Case 1!\n");
07        else if( (b>a) && (y>b) )
08           printf("Case 2!\n");
09        else printf("Case 3!\n");
10       y++;
11       x--;
       }
    }
```

4. Design patterns

```
class ioPattern
{
   public:
      virtual file *newfile(String filename);
      virtual shmem *newShmem(int key);
      virtual msgq *newMsgq(int key);
}

class unixIoPattern: public
{
   public:
      unixIoPattern() {
      file *newFile(String filename)
         {
            if (fileType == 1)
               return new unixFile(filename);
            else
               return new unixPipe(filename);
         }
      shmem *newShmem(int key)
         { return new unixShmem(key); }
      ...
}
```

```
...
main()
{
 ioPattern *iof;
 #ifdef UNIX
    iof = new unixIoPattern();
 #ifdef WINDOWS
    iof = new winIoPattern();
 #elseif QNX
    iof = new qnxIoPattern();
 #endif
 file xyz = iof->newFile("abc");
}
```