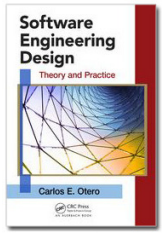
CHAPTER 8: PRINCIPLES OF CONSTRUCTION DESIGN

SESSION II: PROGRAMMING DESIGN LANGUAGE, STYLES, AND QUALITY EVALUATION IN CONSTRUCTION DESIGN

Software Engineering Design: Theory and Practice
by Carlos E. Otero



SESSION'S AGENDA

- Programming Design Language
- Construction (Programming) Styles
  - ✓ Formatting
  - ✓ Naming
  - ✓ Documentation
- Quality Evaluation of Construction
  - ✓ Completeness
  - ✓ Correctness
  - ✓ Testability
  - ✓ Maintainability
- McCabe's Complexity Theory
  - ✓ Used to evaluate testability and maintainability
- What's next…

PROGRAMMING DESIGN LANGUAGE

- Programming Design Language (PDL) is a form of pseudo-code used widely for designing internal function behavior.
  - ✓ PDL's popularity stems from the use of natural languages—as opposed to computer languages or graphical techniques—to define the required behavior of functions.
  - ✓ This result in detailed function designs that are easier to create, easier to review, and easier to translate to code.
- The usage of a natural language in PDL provides a "comments first approach" to constructing code for any programming language.
  - ✓ Therefore, PDL should be written without concern of the target programming language and detailed enough that code can be generated with minimal effort.
- PDL can be used as both design technique and effective documentation approach for functions and code within functions.

PROGRAMMING DESIGN LANGUAGE

- Example of PDL:

```
This function writes events occurring in the system to the event console.
Events are displayed together with their classification and description,
which are both provided by the client's calling function. In addition, the
number of events received for each type of event is computed and displayed
in the event console. Events are required to have id and description. The
function returns a value indicating success/failure of the operation.

Set the value of the function status to "failure"
Determine if the event id is valid

If the event id is valid
  Use the event id to determine the type of event

  Increment by one the counter that keeps track of the number of events
  logged of this type to reflect this newly received event.

  Write the event id, description, and event counter to the event console
  and set the return value to "success"

If the event id is not valid
  Set the event id to "unknown"
  Set the event description to "invalid event ..."
  Write the event id and description to the event console

Return the value of the function status
```

SOFTWARE CONSTRUCTION USING STYLES

- Construction styles are not about programming, but about defining the rules or guidelines for everyone in the project when writing the source code for a computer program.

- As a general principle, construction styles must provide rules that ensure consistency, simplicity, and clarity of code.

- Typical styles used during construction design include:
  - ✓ Formatting Conventions
  - ✓ Naming Conventions
  - ✓ Documentation Conventions

Software Construction using Styles
- Formatting Conventions -

Styles for Whitespaces To Separate Keywords, Parenthesis, and Curly Braces.

```
// Style #1: White space after keyword.
if (x) {

}

// Style #2: White space after parenthesis and parameter.
if( x ) {

}

// Style #3: White space after keyword, parenthesis and parameter.
if ( x ) {

}
```

Styles for Whitespaces in Binary Operators, after Commas, and Semicolons

```
// White before and after binary operators.
int x = y + z;

// Using Style #2 of white spaces.
// White space after comma.
int x, y, z;

// White space after semicolon.
for( int i = 0; i < maxSize; i++ ) {

}
```

SOFTWARE CONSTRUCTION USING STYLES
- FORMATTING CONVENTIONS -

Styles for Whitespaces in Nested Statements.

```
// Indentation of nested statements.
if( x == y ) {

··// Code here is appropriately indented with 2 white spaces.
··if( z == r ) {

····// Code here is appropriately indented with 4 white spaces.
····if( c == a ) {

//////// Code here is appropriately indented with 6 white spaces.
····}
··}
}
```

Matching Closing and Opening Brackets in Nested Statements.

```
void computeValue() {

  while( /*some condition*/ ) {

    for( /*some condition, iterate*/ ) {

      if( /*some condition*/ ) {

        switch( /*some condition*/ ) {
          // ...

        } // end switch
      } // end if
    } // end for
  } // end while
}
```

SOFTWARE CONSTRUCTION USING STYLES
- FORMATTING CONVENTIONS -

Inline Bracket Placement Style.

```
// Inline brace placement style in C++ class definition.
class List {
  // ...
};

// Inline brace placement style in function definition.
void append() {
  // ...
}

// Inline brace placement style in conditional statements.
if( condition == true ) {
  // ...
}
else {
  // ...
}

// Inline brace placement style in loops.
while( condition == true ) {
  // ...
}
```

New-line Bracket Placement Style.

```
// Newline brace placement style in C++ class definition.
class List
{
  // ...
};

// Newline brace placement style in function definition.
void append()
{
  // ...
}

// Newline brace placement style in conditional statements.
if( condition == true )
{
  // ...
}

// Newline brace placement style in loops.
while( condition == true )
{
  // ...
}
```

SOFTWARE CONSTRUCTION USING STYLES
- NAMING CONVENTIONS -

- Naming conventions can help programmers develop models for differentiating different aspects of software programs and maintain consistency through software items.
  - ✓ This can in turn result in code that is self-documented by the use of conventions applied consistently in the code.

- Naming conventions can be used to differentiate all elements that compose a software program; therefore, consistent use of naming style can help software teams to better understand the work done by each other.

- When applied consistently, styles can easily help software developers, testers, and maintainers understand the code and make assumptions about it, based on the programming style.

SOFTWARE CONSTRUCTION USING STYLES
- NAMING CONVENTIONS -

```
// Example 1:
// Incomplete variable names.
Radio rdo1;
Radio rdo2;

// Complete variable name.
Radio activeRadio;
Radio backupRadio;

// Example 2:
class Radio {
  public:
    // Incomplete function name.
    void xmit(Message* message);
};

class Radio {
  public:
    // Complete function name.
    void transmit(Message* message);
};
```

```
// Example 3:
// Incomplete and contextually inappropriate variable names.
if( s > MaxAmount ) {
  // s is ambiguous!
  // MaxAmount of what?
  // MaxAmount does not reflect the appropriate context!
}

// Complete and contextually appropriate variable names.
if( salary > MaxSalaryAmount ) {
  // ...
}

// Example 4:
// Contextually inappropriate names.
DirectoryManager reaper;
reaper.destroy();

// Contextually appropriate names.
DirectoryManager directoryManager;
directoryManager.deleteFiles();
```

SOFTWARE CONSTRUCTION USING STYLES - DOCUMENTATION CONVENTIONS -

- Similar to formatting and naming conventions, documentation conventions can also be (almost universally) applied to projects in different domains and with different programming languages.

- Documentation conventions deal with styles and specifications for what to document and how to document during construction.
  - ✓ Generally, if the naming conventions are followed, comments should provide information that describes why an operation is written as opposed to what the operation is doing.

- In many cases, the actions performed by operations (or blocks of codes) can be inferred from the naming conventions or programming syntax; however, the reasons behind the choice of code cannot be inferred as easily.
  - ✓ Therefore, comments should provide the reasons why code was written, and when necessary, what the code is doing.

Software Construction using Styles
- Documentation Conventions -

```
//****************************************************************************
// FILE:           MyFile.h
//
// DESCRIPTION:    This file contains the definition of class x. Class x
//                 is used in the system for ...  Clients of this class
//                 are required to ...
//
// REVISION:       Revision 1.0
//
// CLASSIFICATION: Unclassified
//
// RESTRICTIONS:   None
//
// AUTHOR:         Joe Developer
//
// HISTORY:
//    PROBLEM #  INITIALS  DATE      DESCRIPTION
// ----------------------------------------------------------------------
//    N/A        JD        1/1/2011  Initial Design and Code.
//    10         TAE       5/1/2011  Removed dead code.
//    ...        ...       ...       ...
// ----------------------------------------------------------------------
//
//****************************************************************************
```

Classified / Unclassified / Company sensitive / etc.

Who, why and when the file was changed?

Documenting files!

Software Construction using Styles
- Documentation Conventions -

Straight from the PDL!

```
//*************************************************************************
// METHOD:          EventLogger::log(EventId id, string description)
//
// DESCRIPTION:     This function writes events occurring in the system to
//                  the event console. Events are displayed together with
//                  their classification and description, which are both
//                  provided by the client's calling function. In addition,
//                  the number of events received for each type of event is
//                  computed and displayed in the event console. Events
//                  are required to have id and description.
//
// RETURNS:         The function returns a boolean value indicating
//                  success/failure of the operation.
//
// PRE-CONDITIONS:  ...
// POST-CONDITIONS: ...
//
//*************************************************************************
```

Documenting functions!

QUALITY EVALUATION OF CONSTRUCTION DESIGN

- The construction design activity is the last major design step performed before construction. Therefore, it provides the last opportunity to evaluate the quality of the system to be built.

- There are numerous project-specific quality characteristic (e.g., security, etc.) that can be identified and evaluated for construction designs.
  - ✓ However, at a minimum, the design's completeness, correctness, testability, and maintainability should be evaluated, since these generally apply to all software projects.

- Completeness and correctness deal with the degree to which construction designs correctly meet the allocated requirements.
  - ✓ Construction designs that are correct, but incomplete, complete but incorrect, or incomplete and incorrect—those that do not meet all requirements, are incorrect, or both—need to be addressed and resolved to maintain the envisioned software quality of the product.

QUALITY EVALUATION OF CONSTRUCTION DESIGN - CORRECTNESS-

- Completeness and correctness can both be evaluated through:
  - ✓ Peer reviews
  - ✓ Unit testing
  - ✓ Audits

- Peer reviews are tasks that concentrate on verifying and validating designs and code (i.e., design reviews and code reviews, respectively).
  - ✓ Peer reviews must be planned, organized, and conducted in such way that a collective approval between all members of the project (with different disciplines) is reached.

- Ultimately, the quality of construction designs in terms of completeness and correctness are evaluated through unit testing. Therefore, as construction designs are created, so are unit tests. One or more unit test cases are essential for verifying and validating construction designs.
  - ✓ Unit test cases can be created straight from Use Cases!

Quality Evaluation of Construction Design
- Unit Tests -

UC-00-Search Product Main Scenario			
Description: This scenario describes the main flow of operations for requesting a product-search with the server system.			
Actors: Operator, Server System			
Pre-Conditions: The client and server system have been initialized.			
Requirements: MCR-001, MCR-002			
Alternate Scenario: UC-10-Invalid Search, UC-11-Connection Failure, UC-12-Response Timeout			
Revision History			
Date	Version	Description	Revised By
9/17/2010	1.0	Initial scenario creation.	John Doe
Description			
Step	Operator Action	System Action	
1	Operator enters valid product ID and clicks on the search button.	*Validates the data*. Retrieves server's communication information from config file.	
2		Establishes a connection with the server system and sends product request data to the server.	
3		Waits a maximum of 3 seconds for a server response.	
4		. .	
5		Response received and product information is displayed.	
6		*Save response data* in file system and ask user to search for another product.	
7	Operator clicks the cancel button to finish searching for products.		
Notes			
For details of data validation and saving response data to file system see use cases UC-05 and UC-06 respectively.			
Approval Signatures			
Software Engineer:			
Stakeholder(s):			
Quality Auditor:			

Sample Use Case Scenario from Chapter 3

Unit Test Case				
Unit Test Name:				
Description:				
Requirements:				
Pre-Conditions:				
S	**Operator Action**	**System Action**	**P/F**	**N**
1	Operator enters invalid product data and clicks on the send button.	Detects invalid data and displays error message to the operator.		
2	Operator enters valid product data and clicks on the send button.	Validates the data. Retrieves server's IP address and port number.		
3		Opens a socket connection and send product request data to the server.		
4		Waits a maximum of 3 seconds for a server response.		
5		. .		
6		Response received and product information is displayed.		
7		Save response data in file system and ask user to search for another product.		
8	Operator clicks the cancel button to finish searching for products.			
Test Result Notes				
Approval Signatures				
Software Engineer:				
Test Engineer:				
Quality Auditor:				

Sample Test Case from Scenario

QUALITY EVALUATION OF CONSTRUCTION DESIGN - TESTABILITY AND MAINTAINABILITY -

- Testability quality (in construction design) deals with the amount of effort required to test artifacts that are the result of construction design.

- On the other hand, a design's maintainability quality deals with the amount of effort required to maintain a tested artifact that is the result of construction design.

- Both testability and maintainability are goals that can be achieved in many ways, as determined by non-functional requirements of the project.
  - ✓ A common approach for evaluating the testability and maintainability of construction designs include the measurement of the design's cyclomatic complexity [1].

- Testability and maintainability goals can be transformed into requirements that are based on the cyclomatic complexity.
  - ✓ Maintainability quality can also be evaluated by the compliance of the resulting implementation of construction design to the programming style defined for the project.

QUALITY EVALUATION OF CONSTRUCTION DESIGN - CYCLOMATIC COMPLEXITY -

- Cyclomatic complexity is a technique developed by Thomas J. McCabe that can be used for evaluating the quality of flow-based designs.

- It is a mathematical technique based on graph theory that provides a quantitative justification for making design decisions that lead to higher quality in terms of a design's maintainability and testability.

- The cyclomatic complexity computation allows designers to measure the complexity of flow-based operational designs by determining the complexity of the decision structure of operations instead of lines of code.

- The cyclomatic complexity technique works by computing the cyclomatic number v*(G)* of a graph *G* with *n* vertices, *e* edges, and *p* connected components, as seen in $v(G) = e - n + 2p$

QUALITY EVALUATION OF CONSTRUCTION DESIGN - CYCLOMATIC COMPLEXITY -

- Computing the cyclomatic complexity for large operations can be tedious, therefore two simplifications methods are available for easily computing the cyclomatic complexity of single-component graphs (i.e., $p=1$).

- The first method allows for the computation of complexity in terms of a program's decision constructs, such as *if* statements, *while* loop, *for* loop, and *case* statements.

- Harlan Mills proved that the cyclomatic complexity (C) of as structured program meeting the control graphs requirements mentioned above is equal to the number of conditions in the code (π) plus 1 [2], as seen in
  - ✓ Where the number of conditions (π) is measured as follows: $C = \pi + 1$
    - If, while, and for loops all count as one unit of complexity
    - Compound conditional statements (e.g., if x and y) count as two units of complexity.
    - Case statements containing n branches count as *n-1* units of complexity.

QUALITY EVALUATION OF CONSTRUCTION DESIGN - CYCLOMATIC COMPLEXITY -

- The second simplification approach allows for the visual determination of complexity via the program control graph.

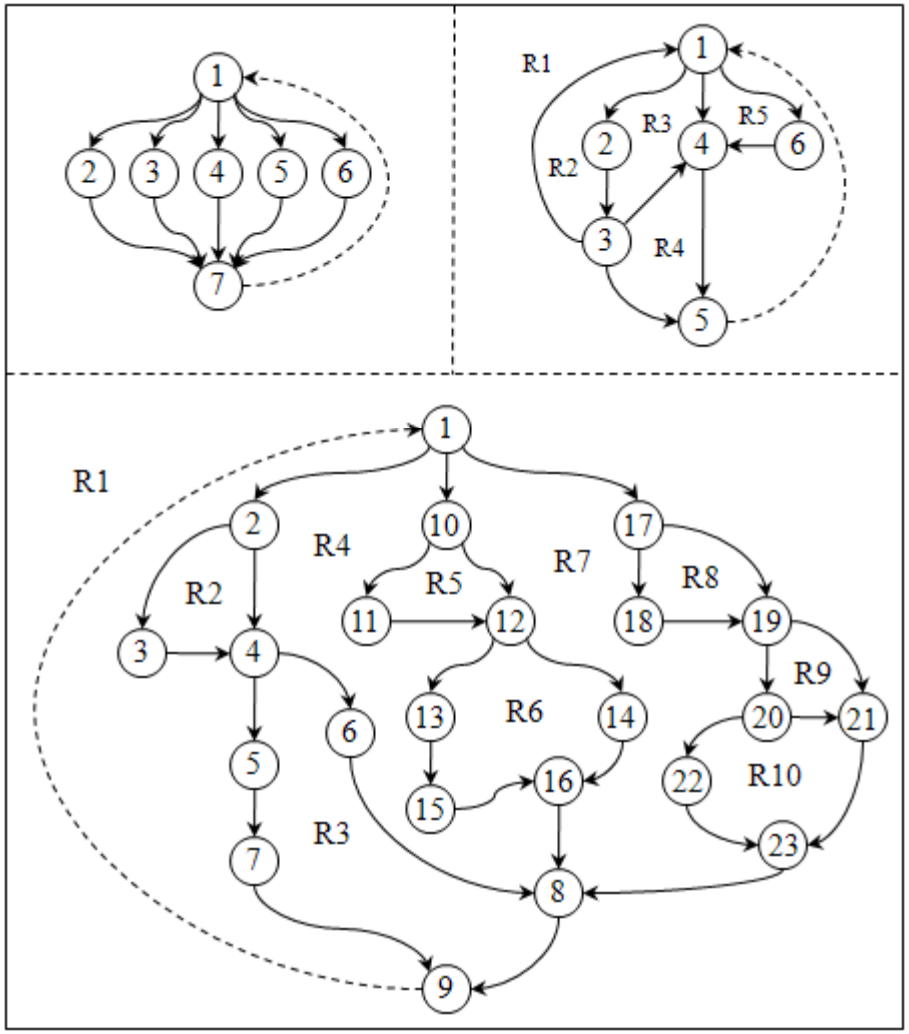
- This approach is based on the work of mathematician Leonhard Euler, who proved that for connected planar graphs—those without intersecting edges—the regions (r) of a graphs can be computed using $2 = n - e + r$

- A regions is an area enclosed by arcs; therefore, given the characteristics of the program control chart, the *number of regions enclosed by arcs*, *plus one that resides outside the graph*, is equal to the cyclomatic complexity of the graph.

- Typically, a cyclomatic complexity value of 10 is acceptable, based on McCabe's work.

QUALITY EVALUATION OF CONSTRUCTION DESIGN
- CYCLOMATIC COMPLEXITY -

- First Program Flow:
  - ✓ e = 10, n = 7, p = 1
  - ✓ V(G) = e – n + 2p
  - ✓ V(G) = 5 **or**
  - ✓ C = π + 1
  - ✓ For switch statement, π = n-1, π = 4, therefore,
  - ✓ C = 5
- Second Program Flow:
  - ✓ r = 5
  - ✓ Complexity = r + 1
  - ✓ Complexity = 6
- Third Program Flow:
  - ✓ C = 11



QUALITY EVALUATION OF CONSTRUCTION DESIGN - MAINTAINABILITY -

- As stated before, maintainability quality can also be evaluated by the compliance of the resulting implementation of construction design to the programming style defined for the project. In many practical situations, where requirements have been established to meet a specific programming style, it can be time-consuming to read code line-by-line to validate that the code meets the style's requirements.
  - ✓ In these cases, the use of automated style checkers can provide significant benefits.

- Automated style checkers (e.g., CheckStyle) are tools that can be configured to enforce a specific style of programming. Some of the capabilities included by these tools include:
  - ✓ Naming conventions of attributes and methods
  - ✓ Formatting conventions
  - ✓ Limit of the number of function parameters, line lengths
  - ✓ Comments (Javadoc) for classes, attributes and methods
  - ✓ Presence of mandatory headers
  - ✓ Good practices for class design
  - ✓ Checks for duplicated code sections
  - ✓ Checks cyclomatic complexity against a specified threshold
  - ✓ Other multiple complexity measurements

WHAT'S NEXT…

- In this session, we continued the discussion on construction design, including:
  - ✓ Programming Design Language
  - ✓ Construction (Programming) Styles
    - Formatting
    - Naming
    - Documentation
  - ✓ Quality Evaluation of Construction
    - Completeness
    - Correctness
    - Testability
    - Maintainability
  - ✓ McCabe's Complexity Theory
    - Used to evaluate testability and maintainability

- This concludes the presentation of construction design. In the next module, we will present an introduction to another very important form of design, human-computer interface design.

REFERENCES

- [1] McCabe, Thomas J. “A Complexity Measure.” IEEE Transactions on Software Engineering SE-2, no. 4 (1976):308-320.
- [2] Mills, Harlan D. Mathematical Foundations for Structured Programming. Gaithersburg, MD: IBM Federal Systems Division, IBM Corporation, 1972.