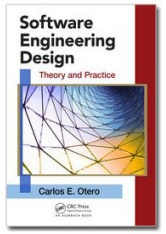
CHAPTER 5: PRINCIPLES OF DETAILED DESIGN

CIS 6930 – SOFTWARE ARCHITECTURE AND DESIGN

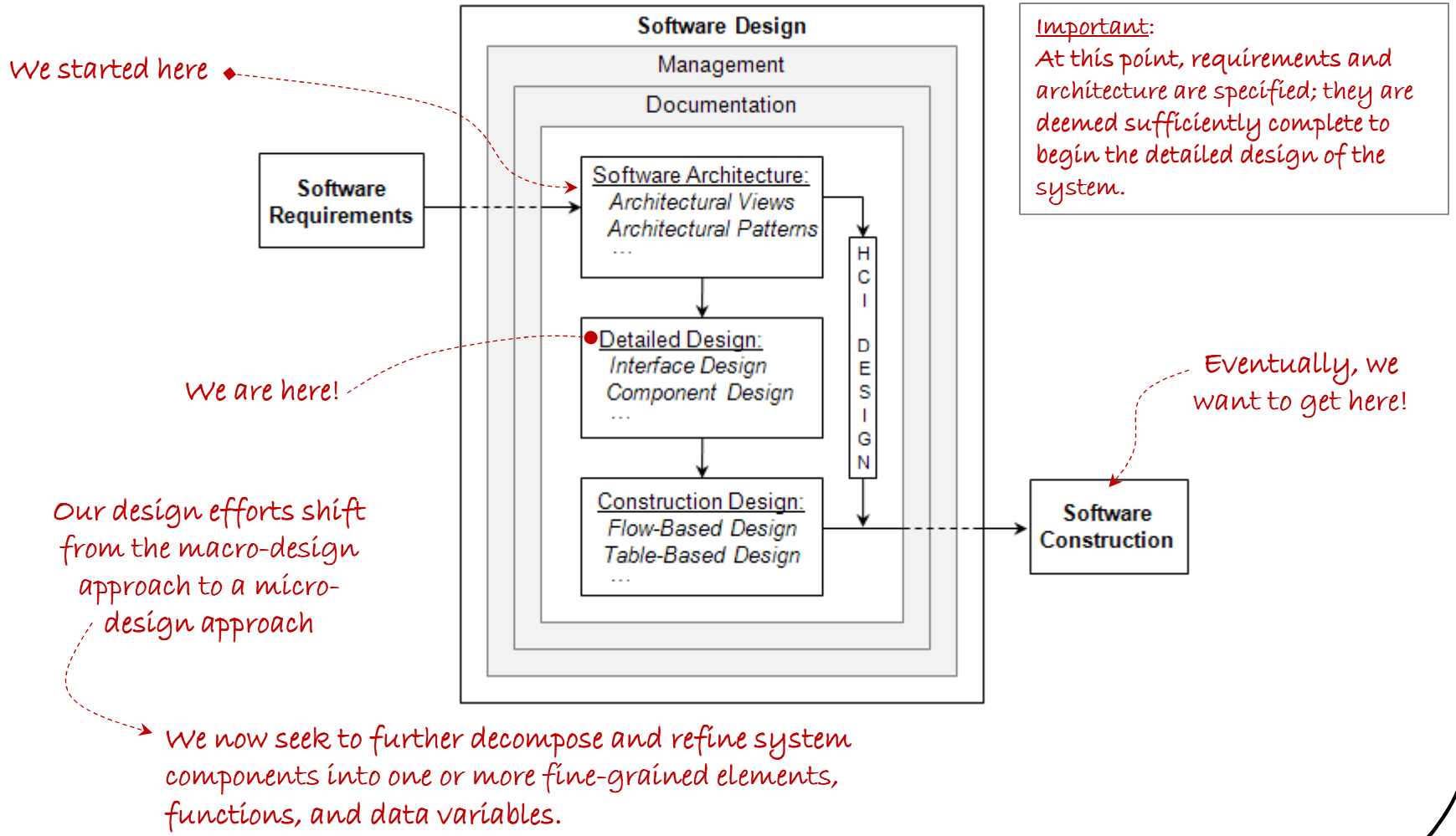
Software Engineering Design: Theory and Practice
by Carlos E. Otero



SESSION'S AGENDA

- Overview of Detailed Design
  - ✓ What is detailed design?
  - ✓ Where does it fit?
- Key Tasks in Detailed Design
  - ✓ Understanding architecture and requirements
  - ✓ Creating detailed designs
  - ✓ Evaluating detailed designs
  - ✓ Documenting detailed designs
  - ✓ Monitoring and controlling implementation
- What's next…

First, Let's Think About Where We Are…



WHAT IS DETAILED DESIGN?

- According to the IEEE [1],
  1. The process of refining and expanding the preliminary design phase of a system or component to the extent that the design is sufficiently complete to be implemented .
  2. The result of the process in 1.

- To keep terminology consistent, we'll use the following definition:
  1. The process of refining and expanding the ~~preliminary design phase~~ *software architecture* of a system or component to the extent that the design is sufficiently complete to be implemented .
  2. The result of the process in 1.

- During *Detailed Design* designers go deep into each component to define its internal structure and behavioral capabilities, and the resulting design leads to natural and efficient construction of software.

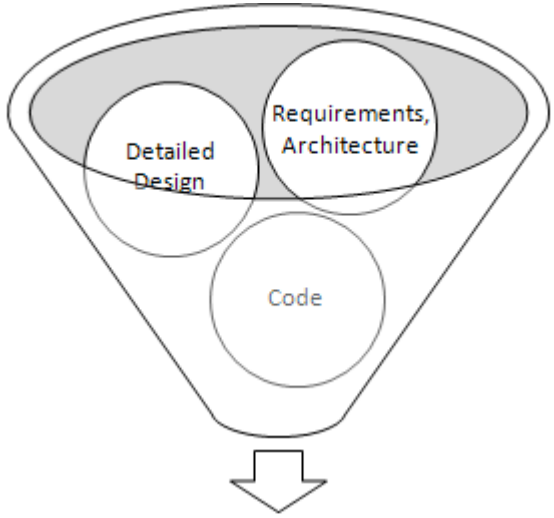
WHAT IS DETAILED DESIGN?

- Clements et al. [2] differentiate between architectural and detailed design as follows:
  - ✓ "*Architecture is design, but not all design is architecture. That is, many design decisions are left unbound by the architecture and are happily left to the discretion and good judgment of downstream designers and implementers. The architecture establishes constraints on downstream activities, and those activities must produce artifacts—finer-grained design and code—that are compliant with the architecture, but architecture does not define an implementation.*"

- Detailed design is closely related to architecture and construction; therefore successful designers (during detailed design) are required to have or acquire full understanding of the system's requirements and architecture.
  - ✓ They must also be proficient in particular design strategies (e.g., object-oriented), programming languages, and methods and processes for software quality control.
  - ✓ Just as architecture provides the bridge between requirements and design, detailed design provides the bridge between design and code.

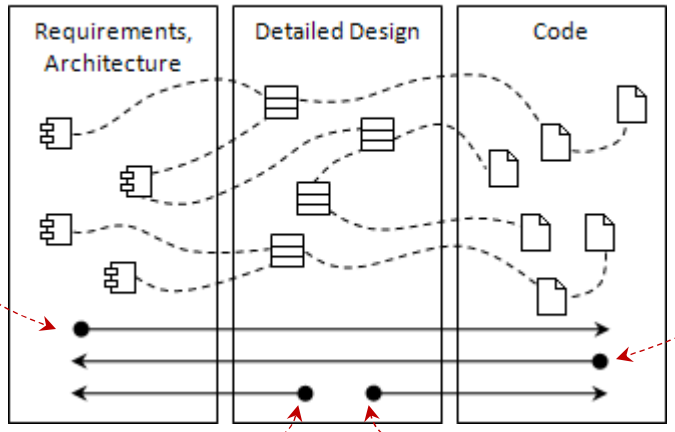
What is Detailed Design?

Designer's Mental Model During Detailed Design!



Important:
During detailed design, the use of industry-grade development tools are essential for modeling, code generation, compiling generated code, reverse engineering, software configuration management, etc.

If given requirements and architecture, detailed designers must move the project forward all the way to code



If given code, detailed designers must be able to reverse engineer the code to produce detailed and architectural designs.

When starting at detailed design, designers must be able to produce both code and architectural designs

KEY TASKS IN DETAILED DESIGN

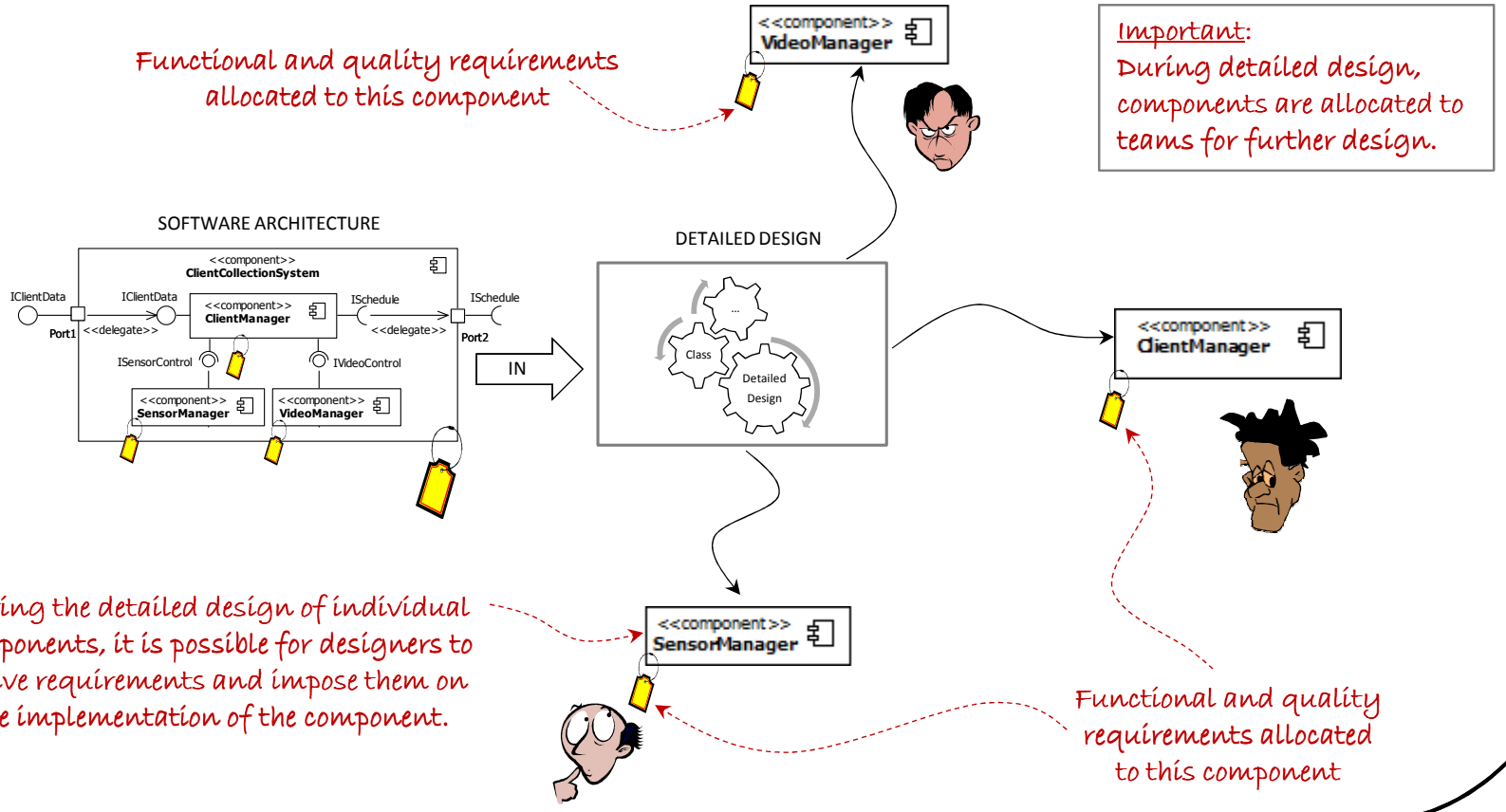
- In practice, it can be argued that the detailed design phase is where most of the problem-solving activities occur. Consider the case in which a formal process is followed, so that the requirements is followed by architecture and detailed design.
  - ✓ In many practical applications, the architectural design activity defers complex problem solving to detailed design, mainly through abstraction.
  - ✓ In some cases, even specifying requirements is deferred to detailed design!

- For these reasons, detailed design serves as the gatekeeper for ensuring that the system's specification and design are sufficiently complete before construction begins.
  - ✓ This can be especially tough for large-scale systems built from scratch without experience with the development of similar systems.

- The major tasks identified for carrying out the detailed design activity include:
  - ✓ Understanding the architecture and requirements
  - ✓ Creating detailed designs
  - ✓ Evaluating detailed designs
  - ✓ Documenting software design
  - ✓ Monitoring and controlling implementation

UNDERSTANDING THE ARCHITECTURE AND REQUIREMENTS

- Unlike the software architecture, were the complete set of requirements are evaluated and well understood, designers during detailed design activity focus on requirements allocated to their specific components.



CREATING DETAILED DESIGNS

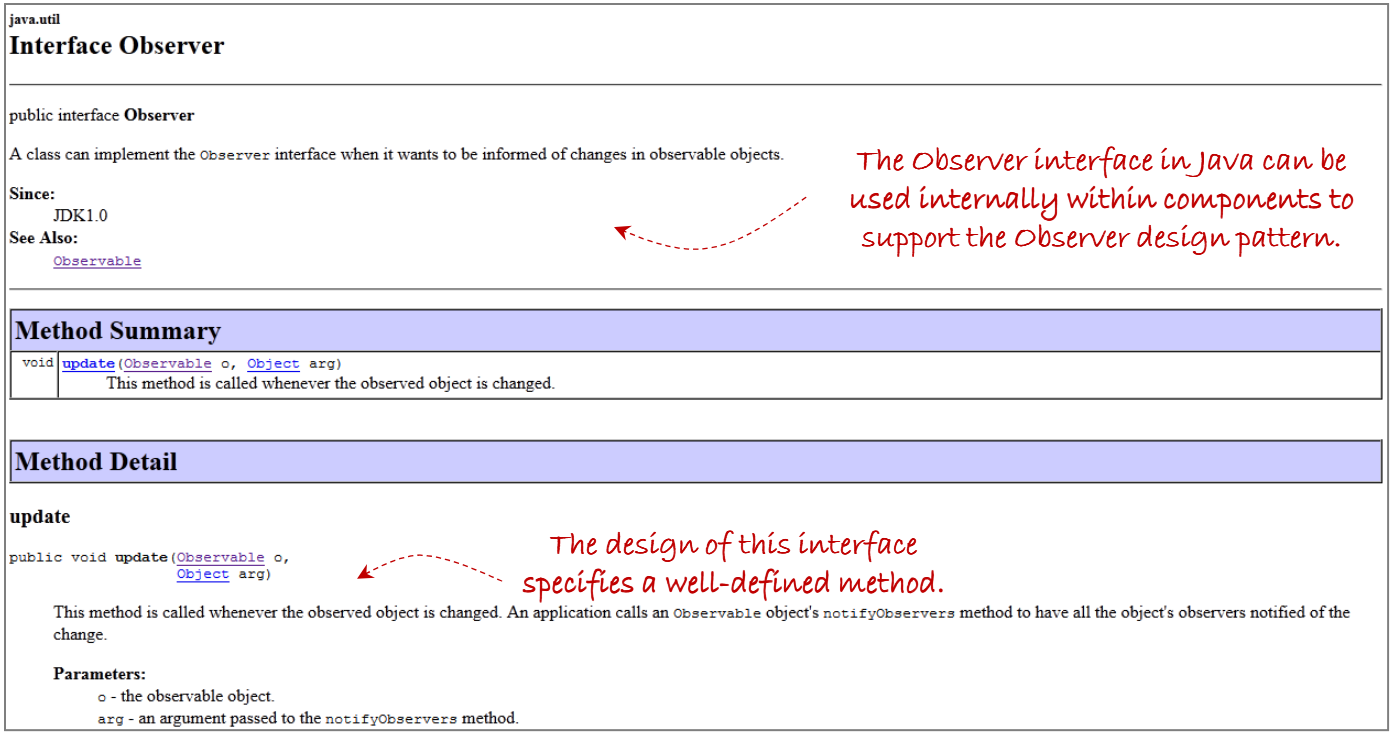
- After the architecture and requirements for assigned components are well understood, the detailed design of software components can begin.
  - ✓ Detailed design consist of both structural and behavioral designs.

- When creating detailed designs, focus is placed on the following:
  - ✓ Interface Design
    - Internal interface design
    - External interface design
  - ✓ Graphical User Interface Design
    - This may be a continuation of designs originated during architecture.
  - ✓ Internal Component Design
    - Structural
    - Behavioral
  - ✓ Data Design

CREATING DETAILED DESIGNS

- Interface Design
  - ✓ Refers to the design task that deals with specification of interfaces between components in the design [3]. It can be focused on:
    - Interfaces internally within components
    - Interfaces used externally across components
- An example of an internal interface design can be seen below:

Note:
We can model this interface easily with UML.



java.util

Interface Observer

public interface **Observer**

A class can implement the `Observer` interface when it wants to be informed of changes in observable objects.

Since:
JDK1.0

See Also:
`Observable`

The Observer interface in Java can be used internally within components to support the Observer design pattern.

Method Summary

void	**update**(`Observable` o, `Object` arg) This method is called whenever the observed object is changed.

Method Detail

update

```
public void update(Observable o,
                   Object arg)
```

The design of this interface specifies a well-defined method.

This method is called whenever the observed object is changed. An application calls an `Observable` object's `notifyObservers` method to have all the object's observers notified of the change.

Parameters:
`o` - the observable object.
`arg` - an argument passed to the `notifyObservers` method.

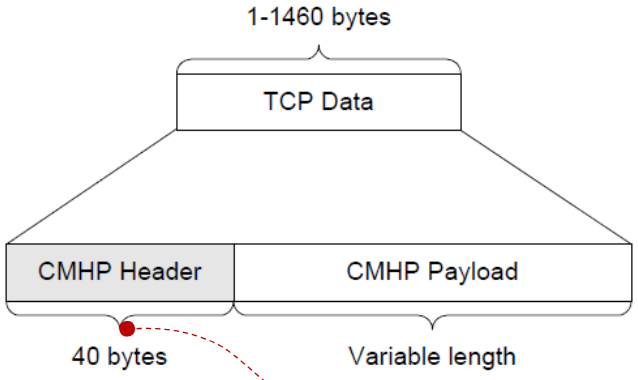
CREATING DETAILED DESIGNS

- Example of external interface design (from Wikipedia)

802.3 Ethernet frame structure

Preamble	Start of frame delimiter	MAC destination	MAC source	802.1Q tag (optional)	Ethertype (Ethernet II) or length (IEEE 802.3)	Payload	Frame check sequence (32-bit CRC)	Interframe gap
7 octets	1 octet	6 octets	6 octets	(4 octets)	2 octets	42[note 2]–1500 octets	4 octets	12 octets
		64–1522 octets						
72–1530 octets								
84–1542 octets								

This is already specified by the 802.3 standard, but, you may design your own application-specific messaging specification at the application level. When you do so, you end up with an Interface Design Document containing all the information about the messaging format. For example, see below



The CMHP Header is designed as seen in the table to the right.

Table 10-1 CMHP Header V1.1

Name	Length (Bytes)	Format	Description
Message Length	4	Numeric	Length of the message including the header – this means that a system can always find a message regardless of the header size
Message Type	2	Numeric	Defines the type of message – WMO, Keep Alive, Registration Request etc (See Section 10.3.1 and the Application-specific data message section)
Major Version	1	Numeric	Set to 0x01
Minor Version	1	Numeric	Set to 0x01
M(s)	1	Numeric	Message Sent Count
M(r)	1	Numeric	Message Receive Count
Flags	1	Bit	Bit 0 – Poll Flag; Bit 1 – Final Flag; 6 flags unused
Spare	1	Numeric	Set to 0x0
Status	2	Numeric	This field will provide supporting information based upon the Message Type – Default value 0x0000
Timestamp - Minutes	2	Numeric	Minute of the Day message sent (0 – ((24*60)-1)
Timestamp - Seconds	4	Numeric	Microsecond of the Minute (0 – (60*100000)-1)
Source Location ID	8	ASCII	Identifier used to depict the sender of the message. Pad with zeros
Spare	8	ASCII	To be used for future options – Set to 0x0.
Checksum	4	Numeric	32-byte CRC

Example extracted from : https://faaco.faa.gov/attachments/5B9EDCCD-D566-F405-67840C21B590D68C.pdf

CREATING DETAILED DESIGNS

- Another example of external interface design in XML

```
<xs:schema targetName_pace="http://learnxmlws.com/Weather"
elementFormDefault="qualified"
  xmlns="http://learnxmlws.com/Weather"
  xmlns:mstns="http://learnxmlws.com/Weather"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="WeatherRequest" type="xs:string"/>
    <xs:element name="CurrentWeather">
        <xs:complexType>
            <xs:sequence>
               <xs:element name="Conditions"
                           type="xs:string" />
               <xs:element name="IconUrl" type="xs:string" />
               <xs:element name="Humidity"
                           type="xs:float" />
               <xs:element name="Barometer"
                           type="xs:float" />
               <xs:element name="FahrenheitTemperature"
                           type="xs:float" />
                <xs:element name="CelsiusTemperature"
                           type="xs:float" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

```
<!-- this is the request message with the zip code in it-->
<WeatherRequest
    xmlns="http://learnxmlws.com/Weather">20171</WeatherRequest>

<!-- this is the response message with weather information-->
<CurrentWeather
    xmlns="http://learnxmlws.com/Weather">
    <Conditions>Sunny</Conditions>
    <IconUrl>http://www.LearnXmlws.com/images/sunny.gif</IconUrl>
    <Humidity>0.41</Humidity>
    <Barometer>30.18</Barometer>
    <FahrenheitTemperature>75</FahrenheitTemperature>
    <CelsiusTemperature>23.89</CelsiusTemperature>
</CurrentWeather>
```

Example extracted from link below. For more details of this example, please navigate to the link below

http://msdn.microsoft.com/en-us/magazine/cc188900.aspx#S2

EVALUATING DETAILED DESIGNS

- Logical designs are verified using static techniques; that is, through non-execution of the software application.
  - ✓ This makes sense since at this point, the software has not been constructed!

- The most popular technique for evaluating detailed designs involves *Technical Reviews*. When conducting technical reviews, keep in mind the following:
  - ✓ Send a review notice with enough time for others to have appropriate time to thoroughly review the design.
  - ✓ Include a technical expert in the review team, as well as stakeholders of your design.
  - ✓ Include a member of the software quality assurance or testing team in the review.
  - ✓ During the review, focus on the important aspects of your designs; those that show how your design helps meet functional and non-functional requirements.
  - ✓ Document the review process.
    - Make sure that any action items generated during the review are captured and assigned for processing.

DOCUMENTING DETAILED DESIGNS

- Documentation of a project's software design is mostly captured in the software design document (SDD), also known as software design description. The SDD is used widely throughout the development of the software.
  - ✓ Used by programmers, testers, maintainers, systems integrators, etc.

- Other forms of documentation include:
  - ✓ Interface Control Document
    - Serves as written contract between components of the system software as to how they communicate.
  - ✓ Version Control Document
    - Contains information about what is included in a software release, including different files, scripts and executable. Different versions of the design depend on specific software release.

DOCUMENTING DETAILED DESIGNS

- The sections of the SDD and sample table of contents:

Section	Description
Date of issue and status	Date of issue is the day on which the SDD has been formally released. Every time the SDD is updated and formally released, there should be a new date of issue.
Scope	Scope provides a high level overview of the intended purpose of the software. It sets a limit as to what the SDD will describe and defines the objectives of the software.
Issuing organization	Issuing organization is the company which produced the SDD.
Authorship	Authorship pertains to who wrote the SDD and certain copyright information.
References	References provide a list of all applicable documents that are referred to within the SDD. If there is a certain technology that is used within the design, it is important to refer to the corresponding documentation on that technology, so it may be referenced. When reading the referenced documents, stakeholders may uncover inconsistencies in how the technology should be used and how it is used in the software design.
Context	Description of the context of the SDD.
Body	Body is the main section of the SDD where the design is documented. This is where stakeholders look to understand the software and how it is to be constructed.
Summary	
Glossary	A glossary provides definitions for all software related terms and acronyms used in the SDD.
Change history	Change history is a brief description of the items added to, deleted from, or changed within the SDD.

MANAGING IMPLEMENTATION

- Detailed design synchronicity is concerned with the degree of how well detailed designs adhere to the software architecture and how well software code adheres to the detailed design.
  - ✓ Low degree of synchronicity points to a flaw in the process and can lead to software project failure.
- Particular attention needs to be paid when projects enter the maintenance phase or when new engineers are brought into the project.
- Processes must be in place to ensure that overall synchronicity is high

WHAT'S NEXT…

- In this session, we presented fundamentals concepts of the detailed design activity, including:
  - ✓ What is detailed design?
  - ✓ Key tasks in detailed design

- In the next session, we will present a more in depth coverage of designing the internal structure of components. This will be necessary to become effective designers, especially when working with design patterns.

REFERENCES

- [1] IEEE. "IEEE Standard Glossary of Software Engineering Terminology." IEEE, 1990, p.34.
- [2] Clements, Paul, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford. *Documenting Software Architectures. Boston*, MA: Addison Wesley, 2001.
- [3] Sommerville, Ian. Software Engineering, 9th ed. *Boston*, MA: Addison Wesley, 2010.