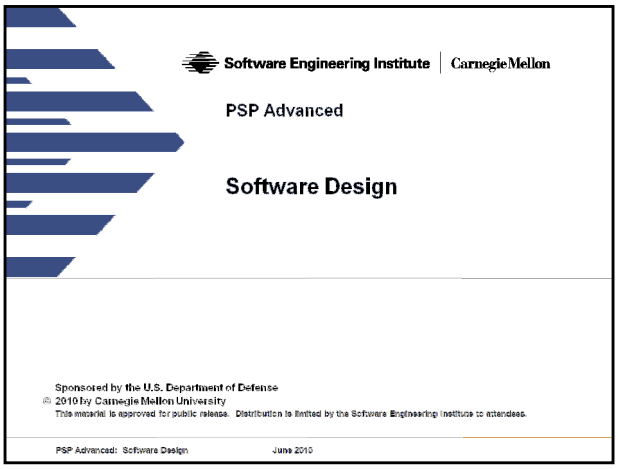
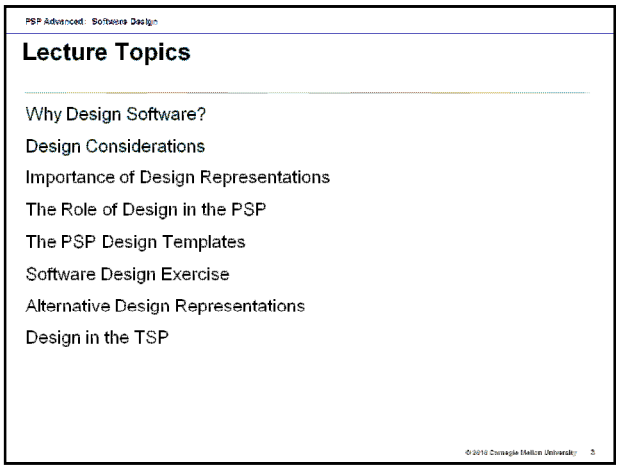

Software Engineering Institute | Carnegie Mellon
PSP Advanced
Software Design
Sponsored by the U.S. Department of Defense
© 2010 by Carnegie Mellon University
This material is approved for public release. Distribution is limited by the Software Engineering Institute to attendees.
PSP Advanced: Software Design
June 2010

Lecture Objectives

After this lecture, you will

- understand why design is important,
- appreciate how DLD fits in a personal process,
- know the principles and ideas that apply to design,
- recognize what a complete design is,
- be able to represent your designs in the PSP templates.


PSP Advanced: Software Design
Lecture Topics
Why Design Software?
Design Considerations
Importance of Design Representations
The Role of Design in the PSP
The PSP Design Templates
Software Design Exercise
Alternative Design Representations
Design in the TSP
© 2010 Carnegie Mellon University 3

See next slide for discussion points.

Why Design Software?

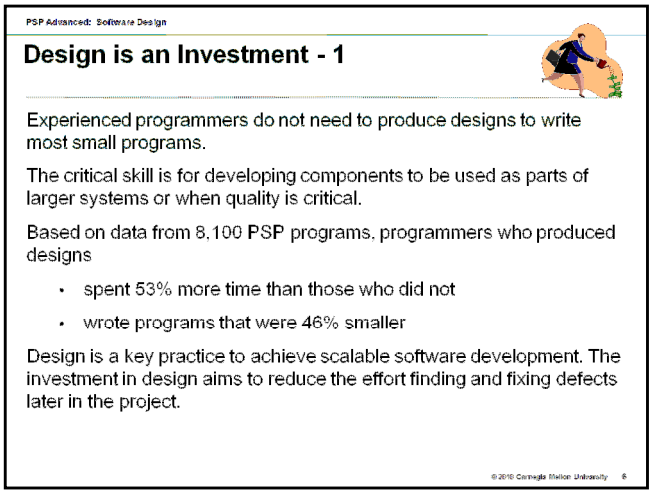
Software design starts with a problem (expressed as requirements) and generates a “blueprint” for a solution to be implemented in software.

The design process should generate an overall view of the solution unobscured by low-level implementation details.

The design process does not construct the solution, but explores the potential solution space and makes decisions about the structure and behaviour of the intended software product.

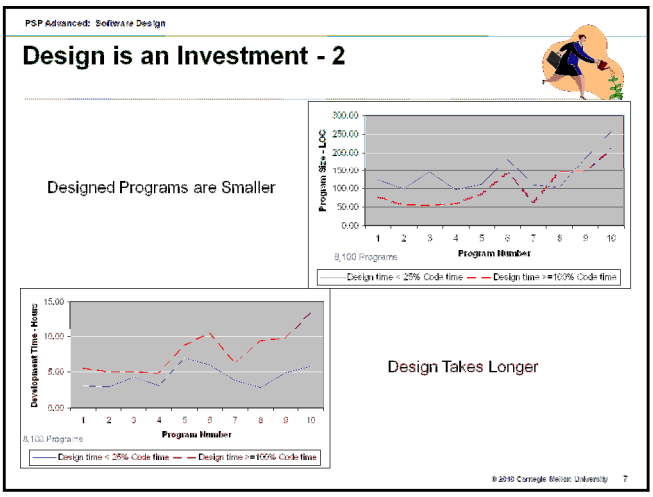
The resulting design will guide the subsequent construction process.

Effective design will minimize defects that arise from inconsistent or incomplete understanding of the software to be built.



Note: if doing good design leads to smaller sized programs, this will typically reduce productivity as measured in LOC/hour, although the total development time may be unaffected. This does identify one of the weaknesses of relying excessively on LOC/hr as a productivity measure and highlights the difference between individual productivity and organizational productivity (functionality delivered per developer per month).

Note: if people do good design, Program 7 will be straight forward, if they don't "oh oh" (and instructor can follow up on day 4 with the data to reinforce the point).



From the overall database of 8,100 programs written by 810 engineers in PSP training courses, the developers for 80 of the programs spent between 100 and 150% of coding time in design time. Those who spent more time were omitted because their excessive design time was often due to poor design practices or requirements misunderstandings. Of this same group, 151 had design times of less than 25% of coding time.

151 code intensive programs

80 design intensive programs

Average development time increase for design-intensive programs is 53%.

Design is Creative

Software design is a creative process.

The design process cannot be

- reduced to a routine procedure
- automated
- precisely controlled or predicted

The design process can be structured to

- separate the routine from the creative activities
- ensure that the design work is performed properly
- identify effective design tools and methods

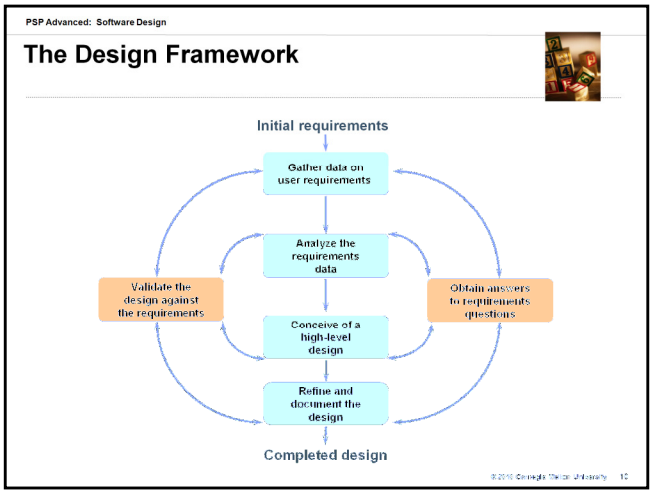
Design is a Learning Process

Design involves discovery, invention, and intuitive leaps from one abstraction level to another.

While the design must reflect the requirements, requirements usually are not stable until the product has been used, if then.

Design work is iterative, and it must be driven by feedback from all involved parties.

The critical problem is knowing when to freeze the design to produce the next iteration.



Structuring the Design Process

Good software designers follow a dynamic process. They

- jump from concept to detail
- simultaneously consider issues at several design levels
- explore multiple alternatives

A structured design process can help you to manage the dynamics of design.

- capture what has been learned
- record and manage issues
- track design status

A properly-implemented design process will reduce rework, manage routine tasks, and give the designer the freedom to be creative.

Design Considerations

During design, there are many issues to consider

- requirements uncertainty
- unfamiliar technologies
- scalability and quality concerns
- prototyping

Requirements Uncertainty

For most new software systems, requirements will not be completely known until after the users have used the finished product.

This creates a challenge for designers.

Analysing the impact of every change to requirements is important.

Unfamiliar Technologies

The computing industry is continually developing new and updating existing technologies for software.

Examples

- operating systems
- programming languages
- middleware
- databases

A design needs to

- satisfy the users' requirements, AND
- be efficiently implementable in the implementation technology

Scalability and Quality Concerns

Many desirable or essential properties of a software product require a global perspective.

Examples

- security
- safety
- scalability of memory usage, execution time, bandwidth requirements

Considering these issues during design is better than trying to retrofit them to a finished but unsatisfactory software product.

Prototyping

Developing a prototype can help resolve many of these issues

- requirements uncertainty
- unfamiliar technologies
- scalability concerns

Before developing a prototype, specify its purpose and define the questions it is to answer.

Commonly, prototypes are discarded once they have served their purpose.

If part of a prototype is to be included in the product, additional effort is required to ensure that it meets normal development standards.

Design Quality

Design is a defect prevention activity.

Poor quality designs are a major source of rework, maintenance, and user dissatisfaction.

A quality design

- is complete and precise
- is documented sufficiently so it can be reviewed
- meets the users' needs
- precisely guides implementation

Users of Design Information

The principal users of the design are

- implementers
- design reviewers and verifiers
- testers and test developers
- documenters, maintainers, and enhancers

These users potentially need a large amount of material.

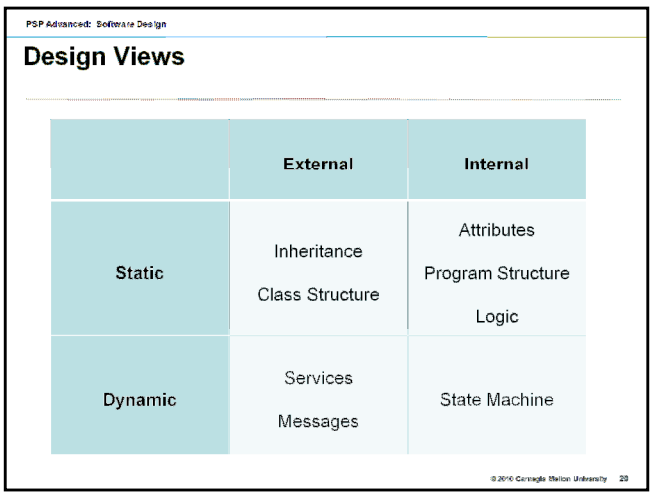
- Not all information is needed immediately.
- Some information can be obtained from other sources.
- It is wise to limit the design workload as much as possible.

Essential Design Information

The information that designers should provide includes

- where the program fits into the system
- a description of how the program will be used
- a specification for all classes and parts
- a structural view of the product
- a specification of all external calls and references
- a list of all external variables, parameters, and constants
- a clear statement of the program's logic

The essential design information can be categorized into static or dynamic views, and internal or external views.



1. External-static: This category defines the static relationships of this part to other parts or system elements. Examples are the call-return behavior and the inheritance hierarchy.
2. External-dynamic: This category defines the interactions of this part with other parts or system elements. Examples would be used scenarios, system or regression-testing scripts, or real-time interrupt-response behavior
3. Internal-static: This category contains a static description of a module or part, such as its detailed logical structure.
4. Internal-dynamic: This category defines the part's dynamic characteristics. Examples of internal-dynamic behavior are state machines, response-time specifications, and interrupt-handling performance.

Importance of Design Representations

It is important to separate two issues.

- how to do design
- how to represent the design when it is completed

Since the PSP can be used with any design method, it does not specify a specific design approach.

However, the PSP does address design representation.

Poor Design Notations Cause Defects

Design visibility

- Complex designs are difficult to visualize.
- A poor representation compounds visualization problems.
- A well-represented design captures all design decisions unambiguously.

Design redundancy

- A redundant design is often inconsistent.
- Inconsistency breeds errors and causes defects.
- A quality design has minimum duplication.
- Where possible, use design tools to ensure consistency.

Requirements for Design Representations

The design notation must

- precisely define all significant design aspects
- be commonly understood
- communicate the designers' intent
- help identify design problems and omissions
- be suitable for representing a broad range of designs

The design should also

- be concise and easy to use
- provide a complete and accessible reference
- have minimum redundancy

Formal notations meet these criteria.

Design Defects

Defects introduced during design are often difficult to detect.

Such defects have the potential to cause significant rework if they are not detected early.

Design reviews offer the first and best opportunity to identify and remove design defects.

Later phases tend to assume the design is correct and are focusing on other types of defects.

For a design review to be effective, the design must be precise and comprehensible so omissions, inconsistencies and ambiguities are readily apparent.

The Role of Design in the PSP

The PSP is specifically designed for individual developers constructing small programs and components for larger systems.

Consequently, design in the PSP focuses on detailed-level design.

However, the principles of design scale to systems of any size and the importance of good design increases with the size of the overall system.

The PSP Design Process

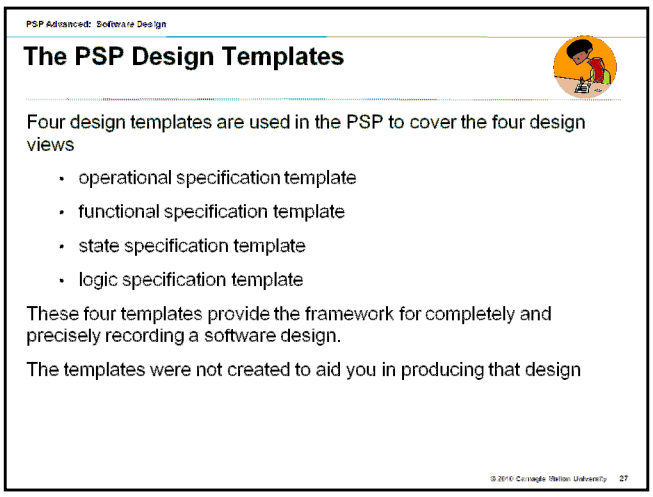
Since there is no single best design method, the PSP supports multiple methods.

The PSP focuses on what a complete design should contain.

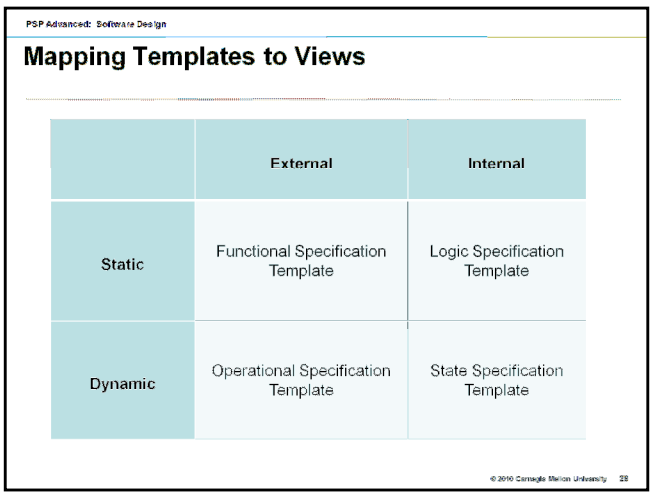
The goal is to eliminate requirements and design defects.

The PSP uses design templates to provide

- criteria for design completeness
- reviewable designs



Go through this slide quickly. Detailed examples will follow.



Note, this differs from the chart in the book which shows the functional specification template in the ***external dynamic*** block as well as the external static block. Discussion with Watts revealed he could not recall why it was there so I removed it (db 04/09).

Operational Specification Template

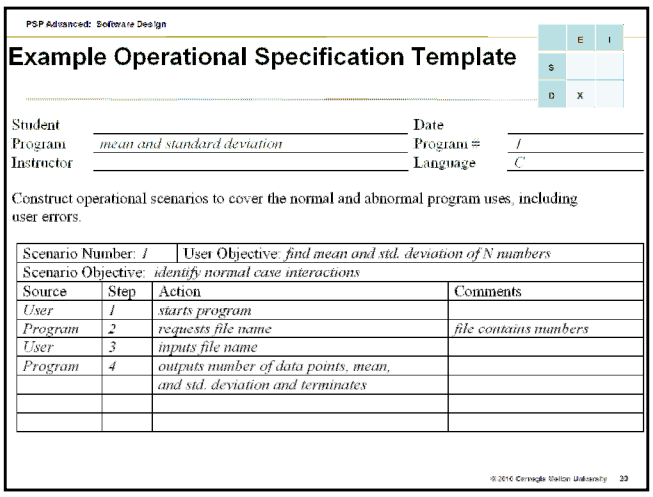
The operational specification template (OST) describes the users' normal and abnormal interactions with the system.

It contains the

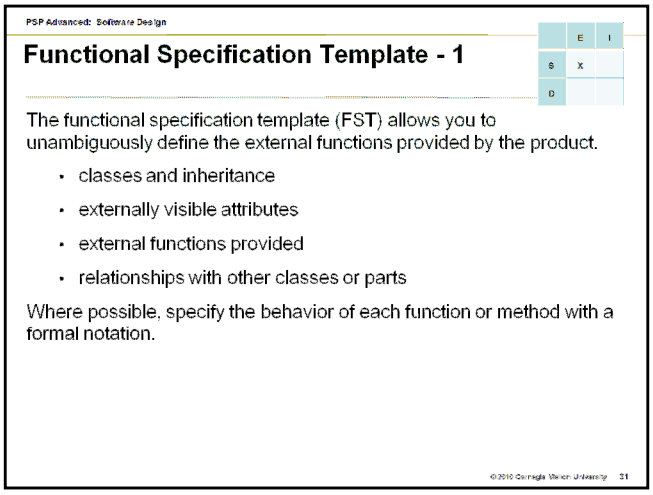
- principal user actions and system responses
- anticipated error and recovery conditions

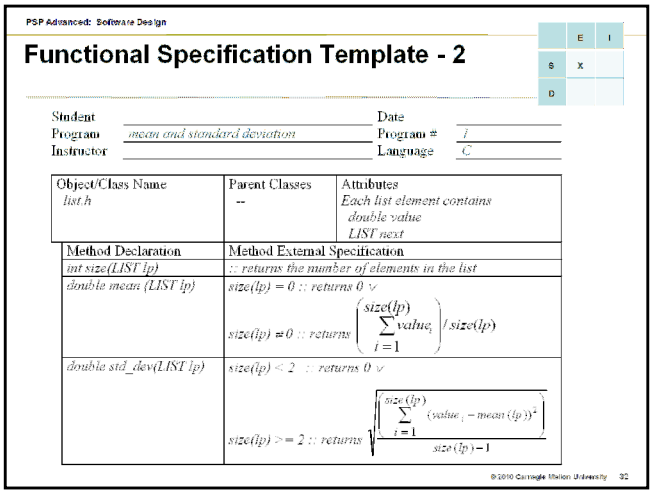
The operational specification template can be used to

- define test scenarios and test cases
- resolve development questions about operational issues
- resolve requirements discussions with users



Just an example of what one looks like. No need to walk through Template step by step. The student will get practice during the design exercise, as well as with each programming assignment.


PSP Advanced: Software Design
Functional Specification Template - 1
E
I
S
X
D
The functional specification template (FST) allows you to unambiguously define the external functions provided by the product.
• classes and inheritance
• externally visible attributes
• external functions provided
• relationships with other classes or parts
Where possible, specify the behavior of each function or method with a formal notation.
© 2010 Carnegie Mellon University 31



PSP Advanced: Software Design

Functional Specification Template - 2

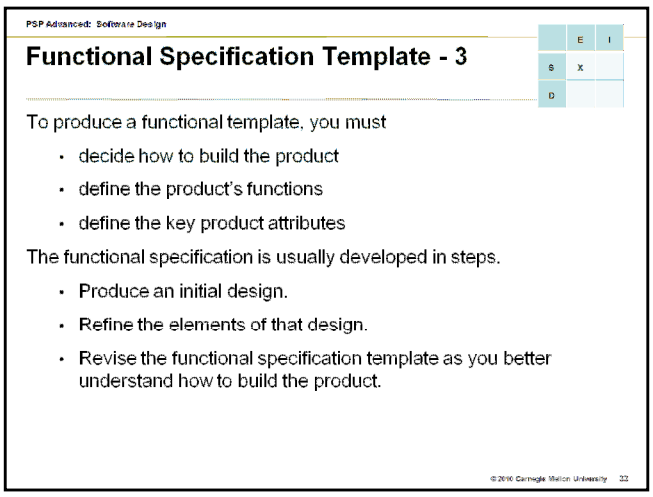
E I S X D

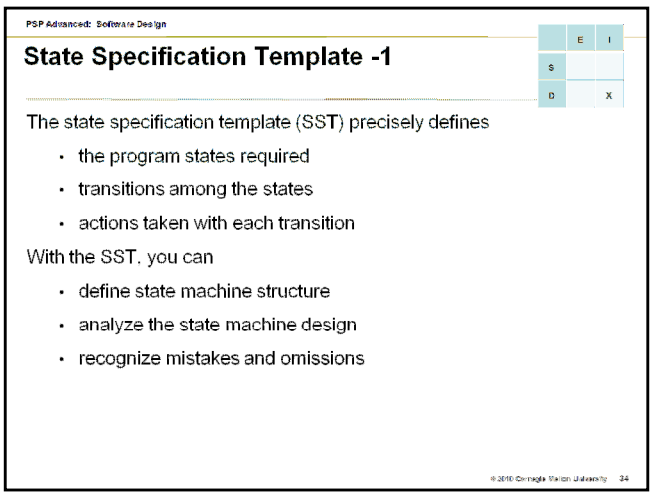
Student _____ Date _____
Program *mean and standard deviation* Program # *1*
Instructor _____ Language *C*

Object/Class Name	Parent Classes	Attributes
list.h	--	*Each list element contains* *double value* *LIST next*

Method Declaration	Method External Specification
int size(LIST lp)	*:: returns the number of elements in the list*
double mean (LIST lp)	*size(lp) = 0 :: returns 0 ∨* *size(lp) ≠ 0 :: returns* $\left(\sum_{i=1}^{size(lp)} value_i\right) / size(lp)$
double std_dev(LIST lp)	*size(lp) < 2 :: returns 0 ∨* *size(lp) >= 2 :: returns* $\sqrt{\frac{\left(\sum_{i=1}^{size(lp)} (value_i - mean(lp))^2\right)}{size(lp)-1}}$

Just an example of what one looks like. No need to walk through Template step by step. The student will get practice during the design exercise, as well as with each programming assignment.


PSP Advanced: Software Design
Functional Specification Template - 3
E
I
S
X
D
To produce a functional template, you must
• decide how to build the product
• define the product's functions
• define the key product attributes
The functional specification is usually developed in steps.
• Produce an initial design.
• Refine the elements of that design.
• Revise the functional specification template as you better understand how to build the product.
© 2010 Carnegie Mellon University 33


PSP Advanced: Software Design
State Specification Template -1
E
I
S
D
X
The state specification template (SST) precisely defines
• the program states required
• transitions among the states
• actions taken with each transition
With the SST, you can
• define state machine structure
• analyze the state machine design
• recognize mistakes and omissions
© 2010 Carnegie Mellon University 34

State Specification Template -2

The SST specifies

- the name of every state
- a brief description of each state
- the name and description of any functions or parameters used in the SST
- the conditions that cause transitions from the state to itself or to any other state
- the conditions that cause transitions from any other state to this state
- the actions taken during each transition

Example State Machine

A simple stopwatch has three buttons (start/stop, reset and hold) and a single display. Initially, the stopwatch displays zero.

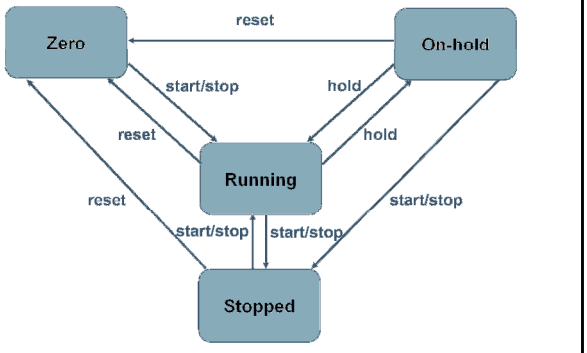
Pressing the start/stop button causes the timer to start incrementing and the display shows the timer values.

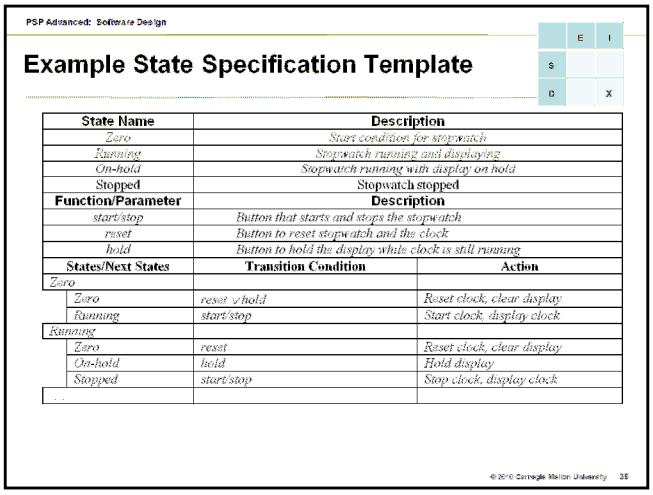
Pressing the hold button while the stopwatch is running causes the display to hold its current value while the timer continues incrementing. If the hold button is pressed again, the display again shows the timer values.

Pressing the start/stop button when the stopwatch is running or on-hold causes the timer to stop incrementing.

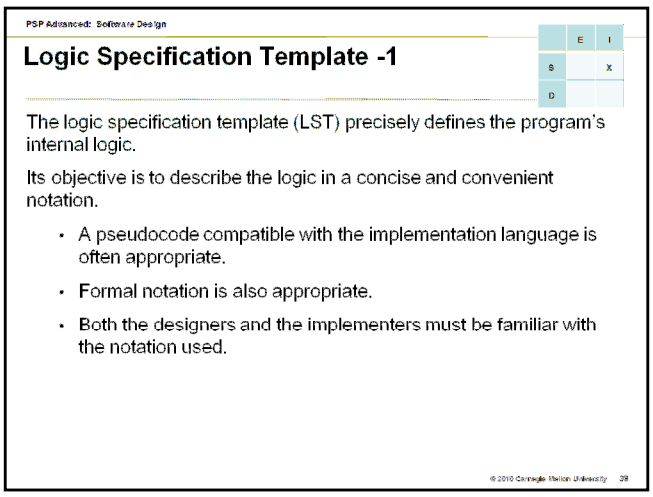
Pressing the reset button at any time causes the stopwatch to return to its initial state.

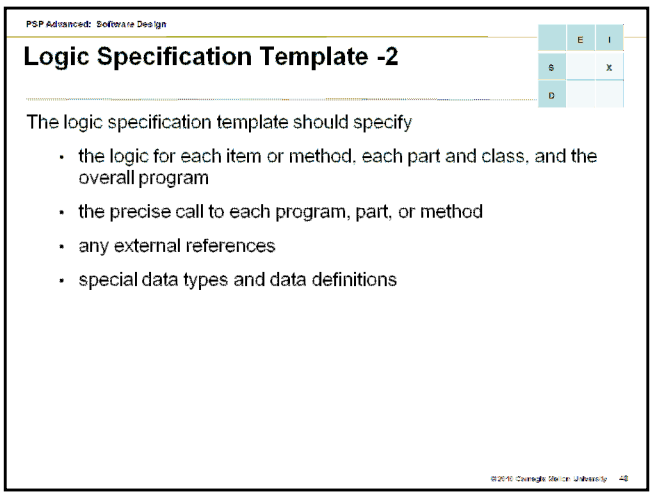
Example State Diagram

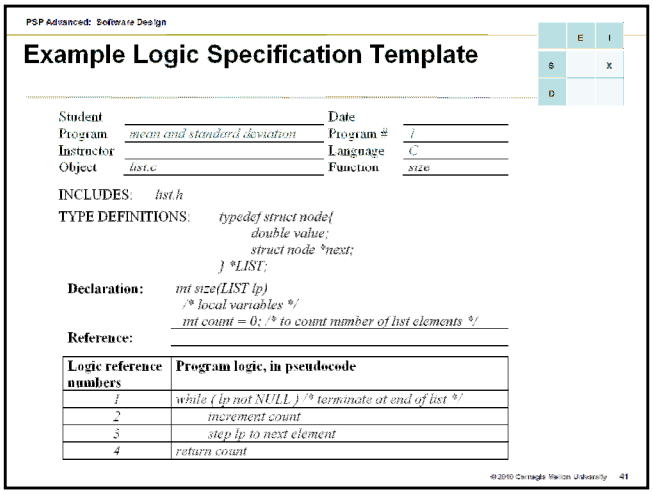




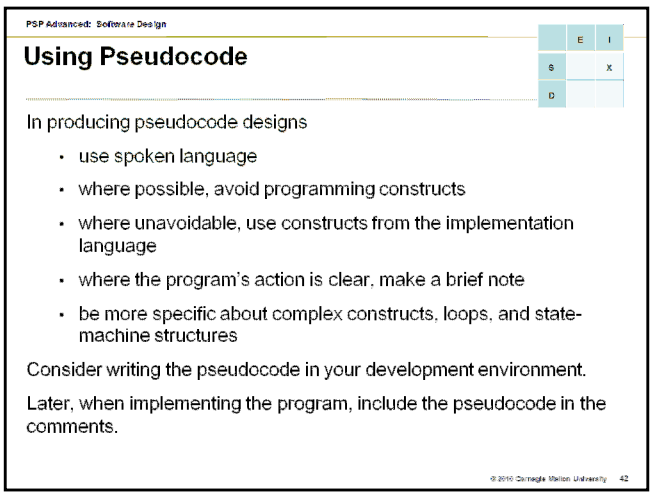
Just an example of what one looks like. No need to walk through Template step by step. The student will get practice during the design exercise, as well as with each programming assignment.


PSP Advanced: Software Design
Logic Specification Template -1
E
I
S
X
D
The logic specification template (LST) precisely defines the program's internal logic.
Its objective is to describe the logic in a concise and convenient notation.
• A pseudocode compatible with the implementation language is often appropriate.
• Formal notation is also appropriate.
• Both the designers and the implementers must be familiar with the notation used.
© 2010 Carnegie Mellon University 39


PSP Advanced: Software Design
Logic Specification Template -2
E
I
S
X
D
The logic specification template should specify
the logic for each item or method, each part and class, and the overall program
the precise call to each program, part, or method
any external references
special data types and data definitions
©2010 Carnegie Mellon University 48



Just an example of what one looks like. No need to walk through Template step by step. The student will get practice during the design exercise, as well as with each programming assignment.


PSP Advanced: Software Design
Using Pseudocode
E
I
S
X
D
In producing pseudocode designs
use spoken language
where possible, avoid programming constructs
where unavoidable, use constructs from the implementation language
where the program's action is clear, make a brief note
be more specific about complex constructs, loops, and state-machine structures
Consider writing the pseudocode in your development environment.
Later, when implementing the program, include the pseudocode in the comments.
© 2010 Carnegie Mellon University 42

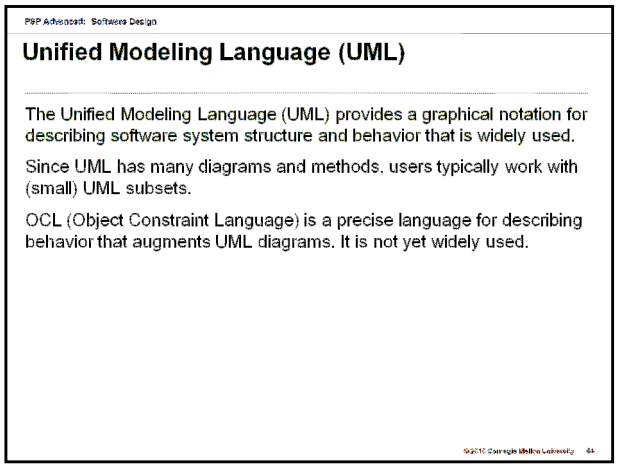
Alternative Design Representations

The PSP design templates have been successfully used by thousands of PSP-trained engineers.

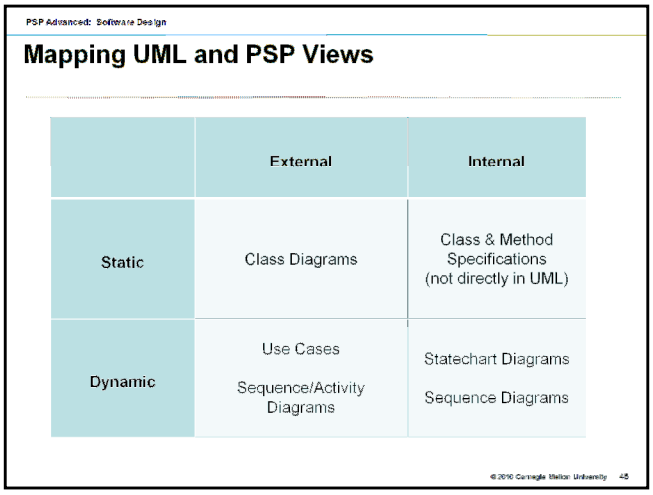
Using these templates in this course is required so you learn and understand their benefits.

Before using an alternative design representation, you should be convinced that it provides at least equivalent capability for

- precision
- completeness
- design review effectiveness



UML is based on notations developed by Booch Rumbaugh and Jacobson.



UML and the PSP templates are complementary:

•UML covers the logical and physical construction of a software system

•The PSP templates focus on precision descriptions of interfaces and system and component behavior

UML and the PSP Templates:

•The UML use case and sequence diagrams provide the same information as the PSP operational specification template

•UML class and sequence diagrams provide relationship and interaction information that is not captured in the PSP templates

•UML class diagrams record the method signature, but the behavioral specification should be documented with the PSP functional specification

•The PSP logic specification template has no UML equivalent, so it should be used to record program logic

•The statechart and state diagram are equivalent

•UML does not have an equivalent to the state specification template

•In designing state machines with UML, record the finished design on the state specification template.

The following material should really only discussed with a class full of UML users:

Use Cases

•Use-case diagrams link actors (external agents) with use cases.

•Each use-case describes a unit of functionality and is documented in text (UML does not define a format).

•A use-case describes sequences of normal and abnormal interactions among actors and the system

•A use-case description is an external perspective, with the system viewed as a “black box”.

•UML activity diagrams can also describe usage details.

Sequence Diagrams

•Sequence diagrams map the use actions to the sequence of messages between objects and actors

•Sequence diagrams also specify the dynamic interactions among objects within a system

•Sequence diagrams describe the time ordering of the interactions.

•UML also provides collaboration diagrams which describe the structural interconnections among objects.

Class Diagrams

•UML class diagrams describe the static relationships between a system and its classes including associations and inheritance

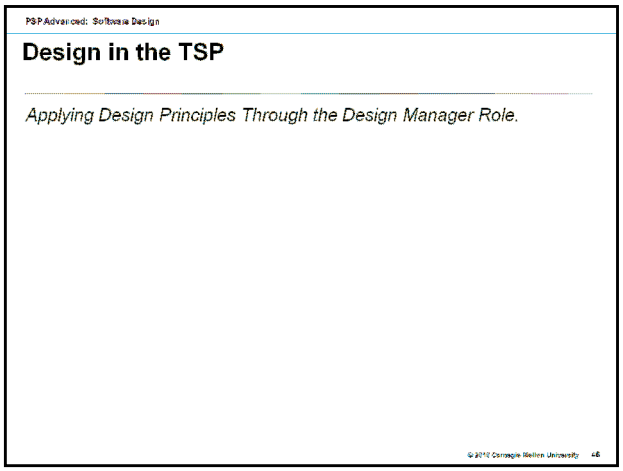
•UML class diagrams also specify the methods and attributes of the class and the class external interfaces.

Statechart Diagrams

•The UML statechart diagram describes all states that an object can have and the events that cause state transitions.

•A statechart diagram identifies

- •The states associated with an object
- •Its transitions (how an object changes state)
- •Its activities (what an object does in a state)



Self-Directed Teams

Manage their own work

Share responsibilities for managing the project through eight team roles:

Planning Manager	Design Manager
Quality Manager	Test Manager
Process Manager	Customer Interface Manager
Support Manager	Implementation Manager

Teams may customize or create special roles depending on the need

When all team members execute their role responsibilities, follow the defined processes, and meet their goals and commitments, the team will perform efficiently and effectively.

Design Manager Team Role

- Maintain a focus on design issues throughout the project
- Identify and resolve all design issues
- Document and confirm design issue resolution
- Provide the team focus for anticipating and addressing product performance and size issues
- Lead the team in producing, refining, and verifying the product design
- Ensure that all the design issues and assumptions are identified documented and resolved
- Manage the design change process
- Establish the standards and procedures the team will use to produce the design materials
- Reports weekly to the team on the status of design standards and product design work

Questions to ask as Design Manager - 1

Are the team's design methods and notations capable of producing a quality design?

Do all team members understand how to use these design methods?

If some team members are not fluent with the design methods, what remedial action do you recommend?

Is the team's design work of high quality?

Has a sound system architecture been produced and documented?

Is the architecture properly controlled and maintained?

Does the architecture consider future product evolution?

Questions to ask as Design Manager - 2

Does the design conform to the architecture?

Is the design properly documented and maintained?

Are the interfaces and other design dependencies with other related teams properly identified and managed?

Are there any other design issues that the team should be aware of?

Messages to Remember

While design is a creative process, its routine aspects can be defined.

A goal of the PSP is to capture design information in a reviewable representation so defects can be identified and eliminated early.

A good design notation will reduce design defects.

A precise design notation helps you to produce clear and correct designs.

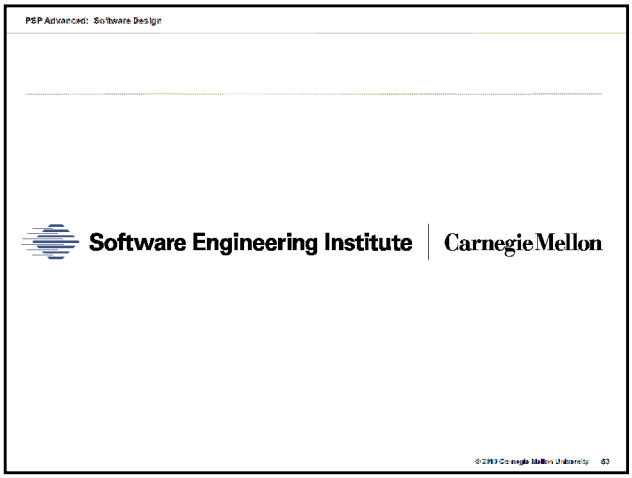
Software design is a skill that is developed through practice, review and reflection on performance.

Use the PSP design templates in the course exercises, and whenever you can do so in your work.

Software Design Exercise

The exercise aims to give you practice with the PSP design templates on a familiar problem (Program 3 from the PSP Fundamentals course).

Using a familiar problem lets you focus on understanding the information to be recorded on the templates, rather than understanding the problem requirements and creating a design to satisfy those requirements.


PSP Advanced: Software Design
Software Engineering Institute
Carnegie Mellon