Software Design

Outline
- Review of PSP Levels
- Overview
- The Design Process
- Design Quality
- Structuring the Design Process
- Design Notation
- Templates for use in Design
- Design Guidelines

Review of PSP Levels
(Humphrey, 1995, p. 11)

PSP0
- Current process
- Time recording
- Defect recording
- Defect type standard

PSP1
- Size estimating
- Test report

PSP2
- Code reviews
- Design reviews

PSP2.1
- Design templates

PSP3
- Cyclic development

Overview
- Good SW design transforms (ill-defined) requirements into an implementable product design specification.
  - Ill-defined requirements?
  - Requirements are generally less-than-perfectly defined. Thus we say they are ill-defined.
  - Ideally we would have well-defined requirements.
- Two aspects of design quality:
  - Content
  - Representation
- Even a good design will probably be poorly implemented if its representation is bad
- The PSP addresses design from a defects-prevention perspective
- Design defects are more difficult to reduce than are coding defects

The Design Process
- Design is creative and cannot be reduced to a routine.
- However, it need not be totally unstructured.
- Design involves many parallel, cooperating activities in which discovery, invention, and intuition are frequently required.

The Design Framework
(cf. Humphrey, 1995, p. 211)
The (Simplified) Systems Development Framework

Design is a Learning Process

Conceptual Design

SASY Iterative Incremental Process

Design Quality

Design Decisions are Based on Design Users’ Needs
**Products Controlled by Design**

**Product Owners**

(cf. Humphrey, 1995, p. 315-316)

- System / Product Mgt
  - Issues log
  - Program’s intended function & how it should be used
  - System-level use scenarios
  - System constraints

- System Engineers
  - File descriptions
  - System messages
  - Reasons why system design decisions were made
  - Special error check / conditions

- Software Designers
  - List of related objects
  - External variables, calls, references
  - Statement of program’s logic
  - Diagram of where the program fits into the system

**Change Control**

(cf. Humphrey, 1995, p. 316)

- Because of the large size of the design of any reasonably large system, the number of changes will be large / frequent and change control is absolutely necessary.
- Make sure that you only specify the absolute minimum of information, and
- Document each piece of information in just one place (so that multiple occurrences do not become inconsistent).
- The PSP deals with design standards for individual developers.

**Design Levels**

(cf. Humphrey, 1995, p. 317)

- Design proceeds at multiple levels of abstraction. (cf. Fig 10.3 Design Pyramid)
- Decisions should be documented at each level where they are made.
- If not, they will have to be reconstructed at each successively higher level.
- This reconstruction is an error-prone process.
- Attempting to work at multiple levels at one time causes difficulty and facilitates errors.

**Structuring the Design Process**

(cf. Humphrey, 1995, p. 318-320)

Design is a dynamic, iterative-incremental, and creative process, yet it is best performed within a structured process framework:

- Requirements
- System specification
- System high-level design
- Product specification
- Product high-level design
- Component 1-1 specification
- Component 1-1 detailed design
- Module 1nk specification
- Module 1nk detailed design

**Requirements Definition**

(cf. Humphrey, 1995, p. 318-319)

- A requirements definition statement describes the problem and/or need in user terms. It does not propose a solution.
- It is rare that you can get a complete and accurate req’s statement before you begin work because:
  - Few people have the specialized skills needed for req’s specification
  - Req’s change: over time and as you ask questions the users will think more deeply about their needs.
  - New solutions will cause needs, and thus req’s, to change.
  - This is a feedback loop...
- Thus, your focus is to work with users to help them generate as clear, precise, and specific a req’s statement as they can at a given point in time.

**Design Specification**

(cf. Humphrey, 1995, p. 319-322)

- The goal of software design is “to produce concise and precise statements of exactly what the program is to do and how to do it”.
- A design specification describes solutions to the problem in both user and technical terms. One or more potential solutions are proposed.
- Designs are specified at multiple levels:
  - High-Level
  - Detailed
  - Implementation
Multiple Design Levels
(cf. Humphrey, 1995, p. 319-322)

- **High-Level**
  - Conceptual/overall design.
  - Critical trade-off decisions are made here.
  - Balances development economics, application needs, and technology: what is feasible, desirable, and affordable (And, we should add what is politically/organizationally acceptable…)
  - Thus to make proper high-level designs you must have accurate development estimates. This will allow you to present in economic terms the costs of each request the user has for system features.

- **Detailed**
  - Reduces high-level design to implementable form: functions, objects, states, …

- **Implementation**
  - While implementation is not design, it implements detailed design, provides feedback (testing) on the quality of the design, and may in fact motivate changes in the design.

Design Notation
(cf. Humphrey, 1995, p. 322-324)

- English (and any other natural language) is too redundant and imprecise to use as a design notation.
- The PSP provides a set of design templates & logic notation to facilitate documenting the various aspects of design.
- Design notation criteria:
  - Can precisely and completely represent the design.
  - Is understandable and usable by the people who must use the design.
  - Helps in efficiently producing a design.
- Design notation used for high-level design work should be implementation dependent, but as lower and lower-level design is performed the notation should become more and more implementation dependent, even to the point of using constructs from the implementation language.

Learning Design Notations
(cf. Humphrey, 1995, p. 323-324)

- It takes time to learn design notations.
- Thus, at first your design work will be harder and will take longer.
- So, give yourself time to first learn a variety of notations.
- Then analyze the effectiveness of various techniques in contrast to not using these techniques.
- Keep techniques that help you address problem areas, and discard techniques that are not helpful.
- Summary: learn, experiment/measure, analyze, select.
- If the data you collect does not indicate that a technique is useful, find something that does!

The PSP’s Design Notation
(cf. Appendix B)

- cf. Appendix B
- cf. Tables 10.1/2, p. 325, 326

- Do Appendix B examples in-class.

Design Templates
(cf. Humphrey, 1995, p. 324-327)

- The PSP focuses on OO design; however, non-OO designs can use the very same techniques:
  - Define ADT’s, organize your designs around “logical” classes, the functions that implement them, state diagrams for these logical “objects”, etc.
- The PSP provides templates that help lead to complete and precise designs, and minimize duplication of information.
  - Information is stored in one place and is then simply referenced other places.

Template Dimensions
(cf. Humphrey, 1995, p. 325-327)

- The elements of a complete design can be organized as follows:
  - Internal Static
    - logical design
    - attributes, semantics
  - Internal Dynamic
    - dynamic behavior
  - External Static
    - relationships to other objects
    - structural hierarchy
    - logical behavior

### Take this slide out and don’t even talk about this model? It doesn’t quite seem to map directly to the four templates as Humphrey suggests.

- The functional specification describes several aspects of a system, including:
  - Class / object names & attributes
  - Inheritance hierarchy (parent classes)
  - Method names (declarations)
  - Method preconditions and actions
- These aspects describe each class conceptually (inheritance, pre-conditions & actions), and specify how the class will be used (method names and calling format).
- Thus we see that this template describes both internal requirements and external uses of each class / method, as well as both static and dynamic aspects.
- cf. Appendix B1-5 on design notation


- The state specification describes the internal dynamic behavior of an object. This includes:
  - The object's states
  - All allowed transitions between these states
  - All conditions that cause transitions.
- What we desire is a "proper" state machine. Proper state machines have the following properties:
  - States are complete & orthogonal.
  - State transitions are complete & orthogonal.
  - Can reach an end state from every other state.
- cf. Example template and notation on p. 331-335. (State machine can be shown both graphically and functionally.)
- cf. Appendix B6 on "proper state machines"


- The logic specification describes the internal processing logic of each method. It provides:
  - Pseudocode describing the method's internal processing logic
  - The object's language-specific internal attributes and actual definition and calling / return protocol
  - #defines, #includes, ...
- cf. CRC cards are conceptually a better way to do this. They can be used to combine the functional and logic templates all together.


- Operational scenarios are descriptions of how a user might expect to interact with the system. They describe things users will want to be able to do. They can also describe incorrect ways the system might be used.
- cf. Ivar Jacobson's "Use Cases"

Using Templates in Design (cf. Humphrey, 1995, p. 343-347)

- At each level you specify external behavior, with functions and operational spec's. Internal behavior is specified with state and logic spec's.
- The design and implementation hierarchies parallel each other, with implementation following naturally on the heels of design.
- Design: At each level you specify external behavior, with functions and operational spec's. Internal behavior is specified with state and logic spec's. Implementation: Design is specified with function and operational spec's. Specification of internal behavior is handled with state and logic spec's.


- Design Levels
  - Work up and down the design hierarchy, however:
    - When possible complete higher-level designs first.
    - Do not consider a higher-level design complete until all abstractions it uses are fully specified.
    - Do not consider program element designs complete until all the elements that call them are complete.
    - Document assumptions as you go.
  - Do not consider lower-level design decisions if they do not affect other parts of the system.
- Prototyping
  - Prototyping can help you resolve difficult issues so you can specify designs about which uncertainty remains until actual implementation is performed.
- Redesign
  - Use the design templates when you have to reverse engineer or redesign an already-existing product.