Lecture 1:**SOFTWARE LIFE CYCLE MODELS** Some popular software life cycle models: 1. Waterfall 2. Build-and-Fix 3. Rapid Prototype 4. Spiral 5. Incremental 6. V-Model 7. Synchronize-and-Stabilize 8. Evolutionary/Technology-Based 9. Evolutionary 10. Rapid Application Development (RAD) 11. Concurrent Development 12. Formal Methods 13. 4th Generation 14. Object-Oriented Models Note: a model is a representation of a process or system **Waterfall** (or Linear Sequential or Classic) Model Phases follow one another in linear order, with iteration. **Build and Fix** build first version, keep modifying until client is satisfied **V Model** SQA team performs verification & validation (V&V) after each phase of life cycle, culminating in final overall V&V **SOFTWARE LIFE CYCLE PHASES** Explicit Software Life Cycle Phases(shown in life cycle models)1. Conception 2. Requirements Analysis 3. Design 4. Implementation (Programming) 5. Testing/Verification/ Validation 6. Maintenance 7. Retirement Some Implicit Software Life Cycle Phases (not shown in life cycle models) 1. Management 2. Documentation 3. SQA **Some Major Reports (Documentation) for Life Cycle Phases:** 1. Software Requirements Specifications (SRS) 2. Project Management Plan (PMP) 3. Software Quality Assurance Plan (SQAP) 4. Design Document (DD) 5. Test Plan (TP) 6. Final Report (all of the above) Some Diagrams Used for Documenting the Phases: 1. UML Diagrams (Use Case, Sequence, Statechart, . . . ) 2. Entity-Relationship (ER) Diagrams 3. Gantt Chart 4. and so on , . .

**EXERCISE 1a**: Consider the list of implicit life cycle phases already mentioned. Note that implicit phases do not show up in most life cycle models. Keeping this in mind, give one or two additional implicit life cycle phases: **ANSWER** Project Management Plan (PMP), Software Verification and Validation Plan (SVVP), Risk Management Monitoring & Mitigation Plan (RMMM), Software Quality Assurance Plan (SQAP), **EXERCISE 1b True:** Most implicit life cycle phases tend to go on for the entire life cycle.

THE CONCEPTION PHASE 1. Conception Input: <various> Action: concept of a product Output Documentation: vague statement of product

THE REQUIREMENTS PHASE 2. Requirements Analysis Input: vague statement of product Action:

System analyst (or software engineer) interacts with customer to understand the needs (requirements) of customer Output: software requirements specification (SRS), precise, detailed, and unambiguous statement of what will be delivered to customer (what software does)

NOTE: A distinction is sometimes made between requirements and specification Requirements = what customer wants Specifications = what is delivered to customer ideally, requirements should equal specifications

THE DESIGN PHASE 3. Software Design Input: software requirements specification (SRS) Action:

Software designer develops architectural and detailed description of software Output: design document (DD), technical description of software, from overview to specifics (how software works)

IMPLEMENTATION PHASE 4. Implementation (previously programming) Input: design document (DD) Action: programmer generates code using tools and/or programming languages Output: test plan (TP), program modules (code)

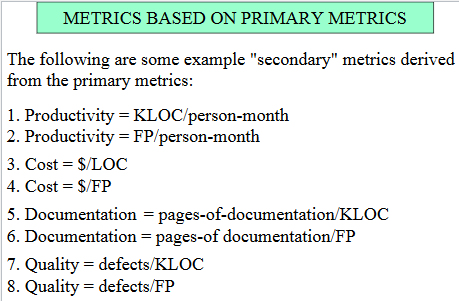
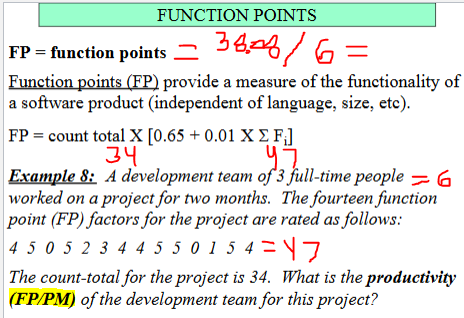
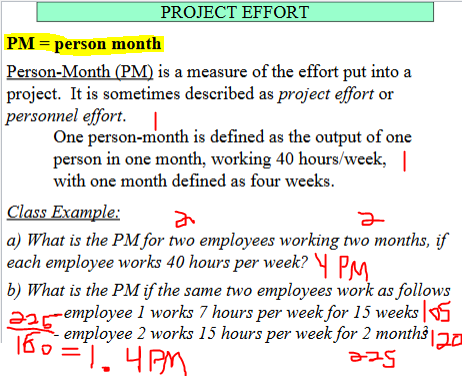
TESTING PHASE 5. Testing (or SQA/Test/V&V) Input: test plan (TP) and program modules Action:

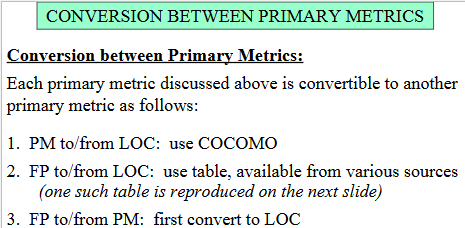
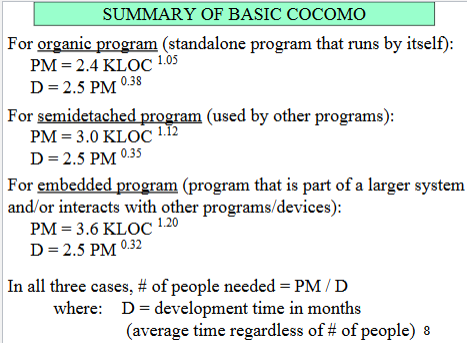
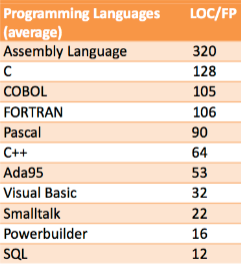
Program modules are integrated and tested Output: working product and maintenance/operation documentation

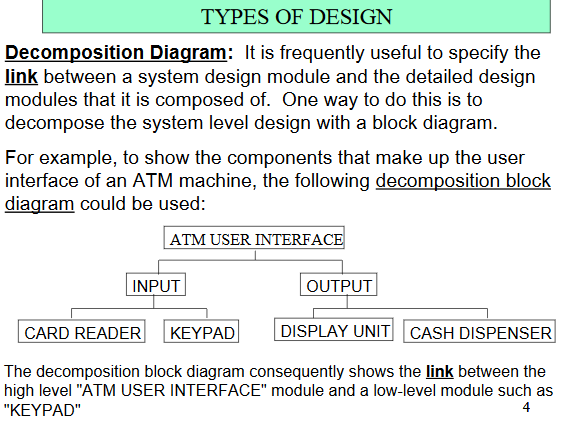
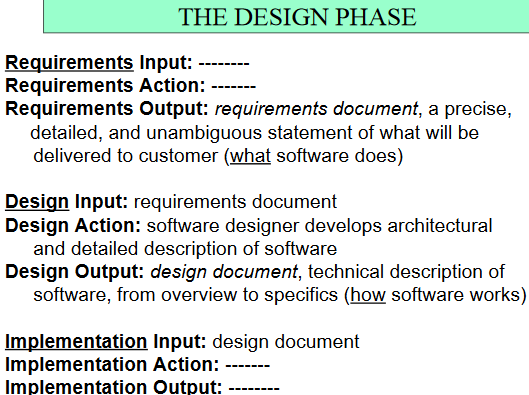
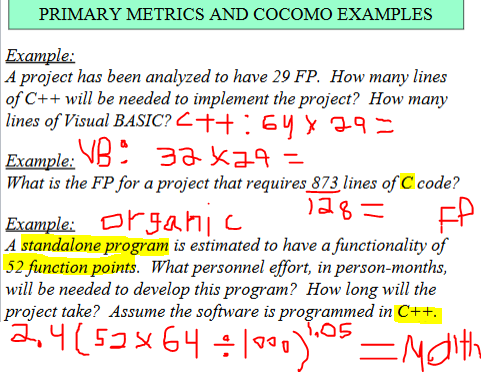
OPERATION/MAINTENANCE PHASE 6. Operation and Maintenance Input: working product and maintenance/operation documentation Action: product is used, defects are fixed, and changes/upgrades are made Output: evolving product and maintenance record

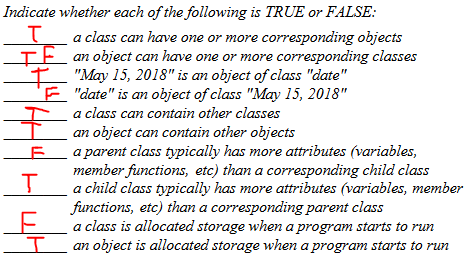
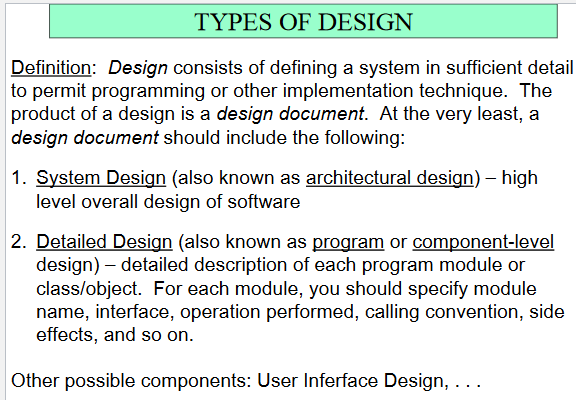
RETIREMENT PHASE 7. Retirement Input: evolving product Action: product is discarded and/or replaced (cycle begins again) Output:

**Lecture 2:** A metric is a numeric measurement. A software metric is a numeric software measurement. Three popular software metrics, previously discussed, are: PM = Person-Month (previously called Man-Month) LOC = Lines of Code (1000 LOC = 1 KLOC) FP = Function Points These three metrics are sometimes referred to as the primary metric variables. A fundamental primary metric is time, but will not be our focus since it is not a basic software metric, and it is included in Person-Month. Several other metrics, called secondary or computed metrics, are based on the primary metrics. Note: Hewlett-Packard defines and uses six primary metrics.. LOC = LINES OF CODE: Measured by 1.Each Statement, Comments, statement delimiters









**Types of Design**: The design process is strongly influenced by the designer "focus" (sometimes referred to as "viewpoint") and

this consequently results in different design types. Some common ones are: 1. Program-Flow Oriented Design === also called process-oriented design|| focus is on the flow of program execution|| suggested notations: flowcharts, Nassi-Schneidermann, pseudocode, structure charts, UML sequence diagrams 2. Data-Flow Oriented Design === Focus is on flow of data and transformations on data || suggested notation: DFDs 3. Data-Oriented Design === approach is to identify major data structures and then design around the structures ||| suggested notation: block diagrams 4. Function-Oriented Design === system is broken down into modules called objects (more correctly, classes)- objects communicate among themselves and with the outside world using messages- subject, a grouping of related objects (helps a reader in understanding a large complex system) - other considerations: inheritance, polymorphism, virtual functions 5. Object-Oriented Design (OOD) === focus is on the functions performed by the various components/modules of the design - suggested notation: block diagrams, DFDs, UML use case diagrams 6. Decomposition/Composition === also called top-down design or stepwise decomposition - start from overview, proceed to details (or, general to specifics)- suggested notation: hierarchy charts opposite of decomposition (from details, to overview) - suggested notation: hierarchy charts 7. Database-Centric === design is developed around a database - frequently utilizes a commercial database management system (DBMS) such as mySQL, SQLite, Oracle - suggested notation: database schema 8. Real-Time Design 9. Hybrid Design 10. Combinations of above

DESIGN EXAMPLE: Give an architectural (high level) software design for an ATM

(automatic teller machine) to perform deposits and withdrawals on savings and checking accounts. Give the design using each of the following approaches: (a) Decomposition (b) Program-Flow (c) Data Structure (or Database) Centric (d) Data-Flow

From Object-Oriented Analysis (OOA) to Object-Oriented Design(OOD): Unlike traditional software design, the distinction between the

Object-Oriented (OOA) phase and the Object-Oriented Design (OOD) phase tends to be blurry. This forces software engineers to begin making object-oriented design decisions during the analysis of the project.

Consider the object-oriented design of an ATM that provides the

following functions for saving and checking accounts: 1. Withdrawal 2. Deposit 3. Balance Enquiry 4. Close Account Give the names of five

objects that could be part of the design: Give the names of two subjects that could be part of the design: Five objects: Account Type, Customer name, Acct num, Acct balance, Cust address, cust SSN. Subject – Account, ATM, Customer.

Three major types of software interfaces are: 1. Interface between internal software components. 2. Interface between software and external non-human objects. 3. Interface between software and human user. The focus in the present discussion will be on the third type of interface, usually referred to as the user interface **user interface** – Graphical User Interface(GUI), Command Line Interface (CLI)||| User Familiarity Use concepts that are familiar to user, not necessarily to developer 2. User Aesthetics Pleasant to current user application 3. Consistency Comparable operations should be activated in similar manner 4. Minimal User Memory Interface should minimize the need for user to memory anything 5. Minimal Surprise User should never be surprised by the behavior of the user interface 6. Recoverability Interface should include mechanisms that allow user to recover from an error 7. User Guidance Interface should provide context-sensitive user help facilities 8. User Feedback Interface should provide meaningful feedback when errors occure 9. Confirmation of Destructive Request Interface should re-prompt user before performing a destructive action||| Design = Technical, Requirements = How||||