International Association Of Certified Practicing Engineers	IACPE INTERNATIONAL ASSOCIATION OF CERTIFIED PRACTICING ENGINEERS www.iacpe.com Knowledge, Certification, Networking	Page : 1 of 95 Rev: 01 Rev. 01- Feb 2016
IACPE No 19, Jalan Bilal Mahmood 80100 Johor Bahru Malaysia	Introduction to Software Engineering CPE LEVEL II TRAINING MODULE	

TABLE OF CONTENT

LIST OF TABLE

LIST OF FIGURE

INTRODUCTION

Scope

National infrastructures and utilities are controlled by computer-based systems and most electrical products include a computer and controlling software. Industrial manufacturing and distribution is completely computerized, as is the financial system. Entertainment, including the music industry, computer games, and film and television, is software intensive. Therefore, software engineering is essential for the functioning of national and international societies.

There are many different types of software systems, from simple embedded systems to complex, worldwide information systems. It is pointless to look for universal notations, methods, or techniques for software engineering because different types of software require different approaches. Developing an organizational information system is completely different from developing a controller for a scientific instrument. Neither of these systems has much in common with a graphics-intensive computer game. All of these applications need software engineering; they do not all need the same software engineering techniques.

Software engineers can be rightly proud of their achievements. Of course we still have problems developing complex software but, without software engineering, we would not have explored space, would not have the Internet or modern telecommunications. All forms of travel would be more dangerous and expensive. Software engineering has contributed a great deal.

Software has become critical to advancement in almost all areas of human endeavor. The art of programming only is no longer sufficient to construct large programs. There are serious problems in the cost, timeliness, maintenance and quality of many software products. Software engineering has the objective of solving these problems by producing good quality, maintainable software, on time, within budget. To achieve this objective, we have to focus in a disciplined manner on both the quality of the product and on the process used to develop the product.

This module is made to provide fundamental knowledge about basic object-oriented concepts, basic concepts in complexity analysis, design and implement simple GUIs. In addition, this module describes about techniques for testing and debugging. It assists engineers to understand design and be able to translate the design into a working program and code written by others.

This module also covers program constructs such as assignment statements, if statements, while and for loops. In addition, engineers can understand about functional style programming in Python including higher-order functions, list comprehension and generators.

General Considerations

I. Software Engineering

At the first conference on software engineering in 1968, Fritz Bauer defined software engineering as “The establishment and use of sound engineering principles in order to obtain economically developed software that is reliable and works efficiently on real machines”. Stephen Schach defined the same as “A discipline whose aim is the production of quality software, software that is delivered on time, within budget, and that satisfies its requirements”. [1]

Both the definitions are popular and acceptable to the majority. However, due to increase in cost of maintaining software, objective is now shifting to produce quality software that is maintainable, delivered on time, within budget, and also satisfies its requirements.

Software engineers adopt a systematic and organized approach to their work, as this is often the most effective way to produce high-quality software. However, engineering is all about selecting the most appropriate method for a set of circumstances so a more creative, less formal approach to development may be effective in some circumstances. Less formal development is particularly appropriate for the development of web-based systems, which requires a blend of software and graphical design skills.

Essential attributes of good Software can be described in the following table:[3]

Table 1 Essential attributes of good software

Product characteristics	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security, and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilization, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable, and compatible with other systems that they use.

II. Process, Methods, and Tools

Software engineering is a layered technology. Referring to Figure1 , any engineering approach (including software engineering) must rest on an organizational commitment to quality. Total quality management and similar philosophies foster a continuous process improvement culture, and this culture ultimately leads to the development of increasingly more mature approaches to software engineering. The bedrock that supports software engineering is a quality focus.

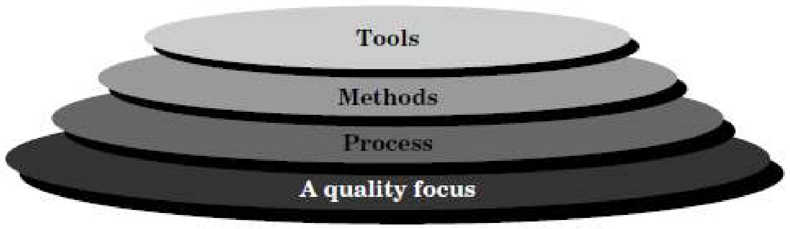

Figure 1 Software engineering layers

The foundation for software engineering is the process layer. Software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software. Process defines a framework for a set of key process areas (KPAs) that must be established for effective delivery of software engineering technology. The key process areas form the basis for management control of software projects and establish the context in which technical methods are applied, work products (models, documents, data, reports, forms, etc.) are produced, milestones are established, quality is ensured, and change is properly managed.

Software engineering methods provide the technical how-to for building software. Methods encompass a broad array of tasks that include requirements analysis, design, program construction, testing, and support. Software engineering methods rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

Software engineering tools provide automated or semi-automated support for the process and the methods. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering, is established. CASE combines software, hardware, and a software engineering database (a repository containing important information about analysis, design, program construction, and testing) to create a software engineering environment analogous to CAD/CAE (computer-aided design/engineering) for hardware.[2]

III. Software Process Models

To solve actual problems in an industry setting, a software engineer or a team of engineers must incorporate a development strategy that encompasses the process, methods, and tools layers. This strategy is often referred to as a process model or a software engineering paradigm. A process model for software engineering is chosen based on the nature of the project and application, the methods and tools to be used, and the controls and deliverables that are required. In an intriguing paper on the nature of the software process, L. B. S. Raccoon uses fractals as the basis for a discussion of the true nature of the software process.

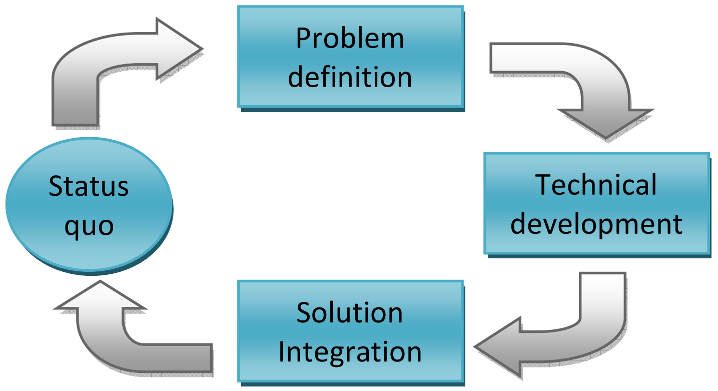

Figure 2 The phases of a problem solving loop

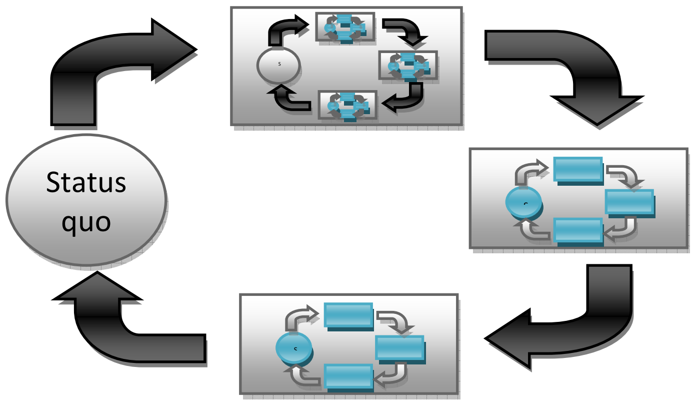

Figure 3 The phases within phases of the problem solving loop

All software development can be characterized as a problem solving loop (Figure 2) in which four distinct stages are encountered: status quo, problem definition, technical development, and solution integration. Status quo “represents the current state of affairs”; problem definition identifies the specific problem to be solved; technical development solves the problem through the application of some technology, and solution integration delivers the results (e.g., documents, programs, data, new business function, new product) to those who requested the solution in the first place.

This problem solving loop applies to software engineering work at many different levels of resolution. It can be used at the macro level when the entire application is considered, at a mid-level when program components are being engineered, and even at the line of code level. Therefore, a fractal representation can be used to provide an idealized view of process. In Figure 3, each stage in the problem solving loop contains an identical problem solving loop, which contains still another problem solving loop (this continues to some rational boundary; for software, a line of code).

In the sections that follow, a variety of different process models for software engineering are discussed. Each represents an attempt to bring order to an inherently chaotic activity. It is important to remember that each of the models has been characterized in a way that (ideally) assists in the control and coordination of a real software project.

i. The Linear Sequential Model

Sometimes called the classic life cycle or the waterfall model, the linear sequential model suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing, and support. **Figure 4**. illustrates the linear sequential model for software engineering

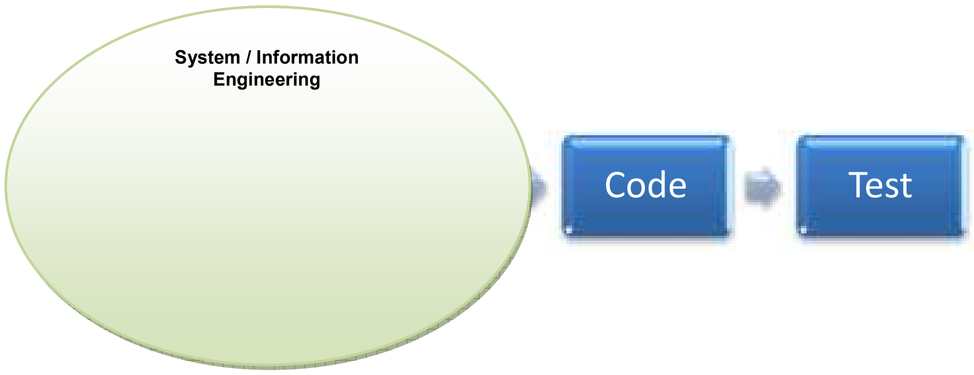

Figure 4 The linear Sequential Model

ii. The Prototyping Model

Often, a customer defines a set of general objectives for software but does not identify detailed input, processing, or output requirements. In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human/machine interaction should take. In these, and many other situations, a prototyping paradigm may offer the best approach.

The prototyping paradigm

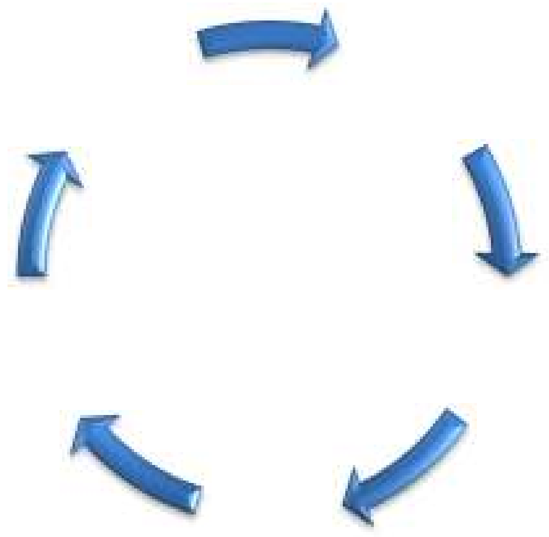

Figure 5

begins with requirements gathering. Developer and customer meet and define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory. A "quick design" then occurs. The quick design focuses on a representation of those aspects of the software that will be visible to the customer/user (e.g. Input approaches and output formats). The quick design leads to the construction of a prototype. The prototype is evaluated by the customer/user and used to refine requirements for the software to be developed. Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done.

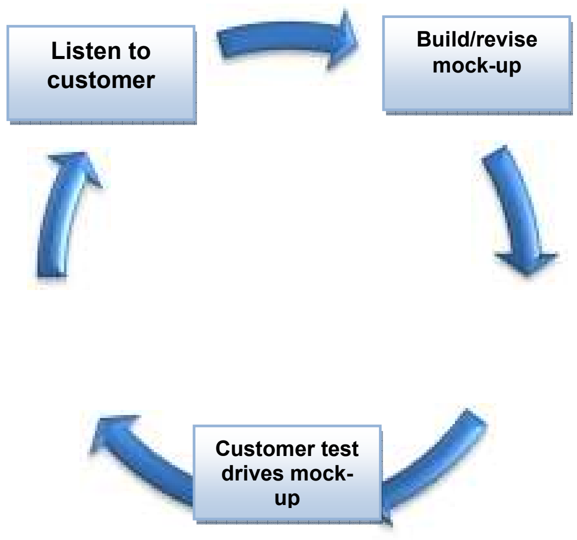

Figure 5 The prototyping paradigm

iii. The RAD Model

Rapid application development (RAD) is an incremental software development process model that emphasizes an extremely short development cycle. The RAD model is a “high-speed” adaptation of the linear sequential model in which rapid development is achieved by using component-based construction. If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a “fully functional system” within very short time periods (e.g., 60 to 90 days). Used primarily for information systems applications, the RAD approach encompasses the phases; business modeling, data modeling, process modeling, application generation, testing and turnover.

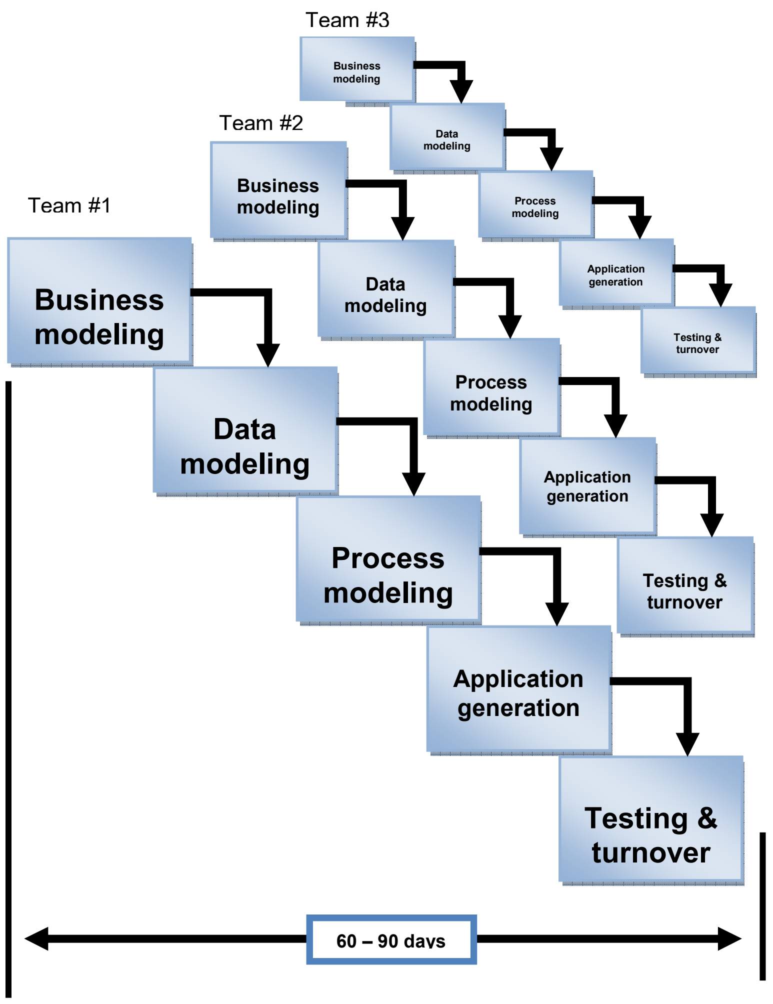

Figure 6 The RAD Model

iv. Evolutionary Software Process Models

There is growing recognition that software, like all complex systems, evolves over a period of time. Business and product requirements often change as development proceeds, making a straight path to an end product unrealistic; tight market deadlines make completion of a comprehensive software product impossible, but a limited version must be introduced to meet competitive or business pressure; a set of core product or system requirements is well understood, but the details of product or system extensions have yet to be defined. In these and similar situations, software engineers need a process model that has been explicitly designed to accommodate a product that evolves over time.

The linear sequential model (Figure 4) is designed for straight-line development. In essence, this waterfall approach assumes that a complete system will be delivered after the linear sequence is completed. The prototyping model is designed to assist the customer (or developer) in understanding requirements. In general, it is not designed to deliver a production system. The evolutionary nature of software is not considered in either of these classic software engineering paradigms.

Evolutionary models are iterative. They are characterized in a manner that enables software engineers to develop increasingly more complete versions of the software.

- The Incremental Model
- The Spiral Model
- The WINWIN Spiral Model
- The Concurrent Development Model

v. Component – Based Development

Object-oriented technologies provide the technical framework for a component-based process model for software engineering. The object-oriented paradigm emphasizes the creation of classes that encapsulate both data and the algorithms used to manipulate the data. If properly designed and implemented, object-oriented classes are reusable across different applications and computer-based system architectures.

The component-based development (CBD) model (Figure 7) incorporates many of the characteristics of the spiral model. It is evolutionary in nature, demanding an iterative approach to the creation of software. However, the component-based development model composes applications from pre-packaged software components (called classes).

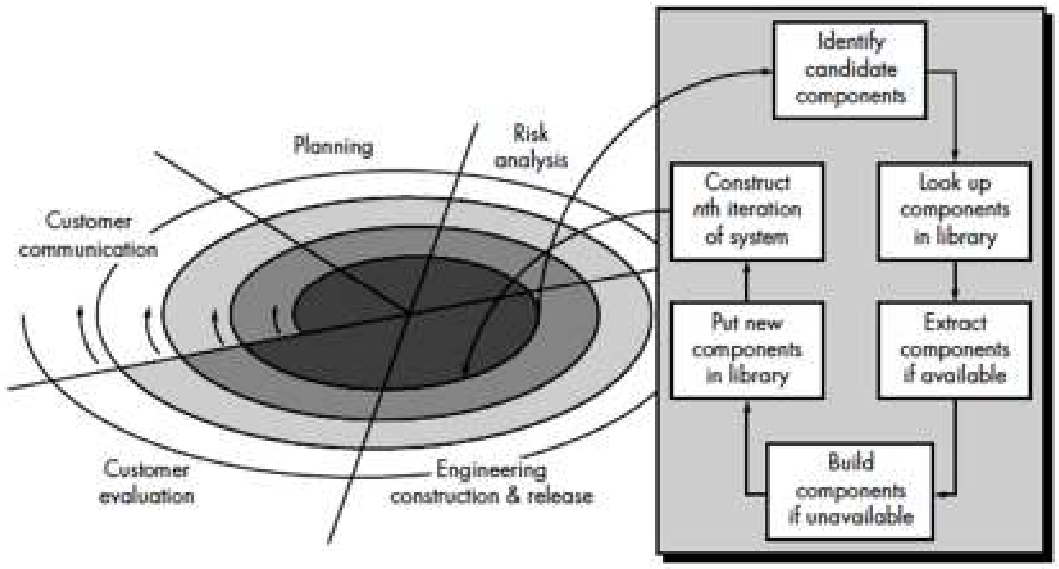
Planning
Risk analysis
Customer communication
Customer evaluation
Engineering construction & release
Identify candidate components
Construct nth iteration of system
Look up components in library
Put new components in library
Extract components if available
Build components if unavailable

Figure 7 Component - based development

The component-based development model leads to software reuse, and reusability provides software engineers with a number of measurable benefits. Based on studies of reusability, industry studies, report component assembly leads to a 70 percent reduction in development cycle time; an 84 percent reduction in project cost, and a productivity index of 26.2, compared to an industry norm of 16.9. Although these results are a function of the robustness of the component library, there is little question that the component-based development model provides significant advantages for software engineers.

vi. The Formal Methods Model

The formal methods model encompasses a set of activities that leads to formal mathematical specification of computer software. Formal methods enable a software engineer to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation. A variation on this approach, called cleanroom software engineering, is currently applied by some software development organizations.

DEFINITIONS

Software Engineering - A discipline whose aim is the production of quality software, software that is delivered on time, within budget, and that satisfies its requirements.

Algorithm- An algorithm (pronounced AL-go-rith-um) is a procedure or formula for solving a problem

Business model- A business model is the conceptual structure supporting the viability of a business, including its purpose, its goals and its ongoing plans for achieving them.

Computer-aided design/engineering - CAD (computer-aided design) software is used by architects, engineers, drafters, artists, and others to create precision drawings or technical illustrations. CAD software can be used to create two-dimensional (2-D) drawings or three-dimensional (3-D) models.

Data modelling - Data modeling is a process used to define and analyze data requirements needed to support the business processes within the scope of corresponding information systems in organizations.

Hardware - In information technology, hardware is the physical aspect of computers, telecommunications, and other devices.

Information systems - is the collection of technical and human resources that provide the storage, computing, distribution, and communication for the information required by all or some part of an enterprise.

Operating system -An operating system (OS) is system software that manages computer hardware and software resources and provides common services for

computer programs - The operating system is a component of the system software in a computer system.

Product requirements - defines how the product will do for the interface then allows designers and engineers to use their expertise to provide optimal solutions to requirements

Requirements specification -is a comprehensive description of the intended purpose and environment for software under development

Software design - Software design is actually a multistep process that focuses on four distinct attributes of a program: data structure, software architecture, interface representations, and procedural (algorithmic) detail. The design process translates requirements into a representation of the software that can be assessed for quality before coding begins.

Software requirements analysis - The requirements gathering process is intensified and focused specifically on software. To understand the nature of the program(s) to be built, the software engineer ("analyst") must understand the information domain for the software, as well as required function, behavior, performance, and interface.

Testing - Conducting tests to uncover errors and ensure that defined input will produce actual results that agree with required results.

UML - Unified Modeling language (UML) is a standardized modeling language enabling developers to specify, visualize, construct and document artifacts of a software system.

GUI - A graphical user interface (GUI) is an interface through which a user interacts with electronic devices such as computers, hand-held devices and other appliances.

Object-oriented programming (OOP) - A schematic paradigm for computer programming in which the linear concepts of procedures and tasks are replaced by the concepts of objects and messages.

Syntax - the grammatical rules and structural patterns governing the ordered use of appropriate words and symbols for issuing commands, writing code, etc., in a particular software application or programming language.

Class - A class is used in object-oriented programming to describe one or more objects. It serves as a template for creating, or instantiating, specific objects within a program.

Attribute - in computing, an attribute is a specification that defines a property of an object, element, or file. It may also refer to or set the specific value for a given instance of such.

Tuple - A tuple in Python is much like a list except that it is immutable changeable) once created.