Advanced Software Engineering
(CS6401)

Autumn Semester (2022-2023)

Dr. Judhistir Mahapatro
Department of Computer Science and Engineering
National Institute of Technology Rourkela

Model View Controller Pattern

```
class Student
{
        private String name;

        public String getName()
        {
                return name;
        }

        public void setName(String name)
        {
                this.name = name;
        }
}
```

```
class StudentView
{
        public void printStudentDetails(String studentName)
        {
                System.out.println("Student: ");
                System.out.println("Name: " + studentName);
        }
}
```

```
class StudentController
{
        private Student model;
        private StudentView view;

    public StudentController(Student model, StudentView view)
        {
            this.model = model;
            this.view = view;
        }

public String getStudentName() {     return model.getName();         }

public void setStudentName(String name) {   model.setName(name);   }

public void updateView()
        {
            view.printStudentDetails( model.getName() );
        }
}
```

```
class MVCPattern
{
   public static void main(String[] args)
   {
            Student student = new Student();
            student.setName(“Ram”);


        StudentView view = new StudentView();

         StudentController controller = new StudentController(model, view);

            controller.updateView();

            controller.setStudentName(“Laxman");

            controller.updateView();
            }

}
```

Software Project Management

Software project Management

The main goal of this is to enable a group a developers to work effectively towards the successful completion of a project.

Project Planning

- Project planning is undertaken and completed before any development activity starts.
- It required utmost care and attention since commitment to unrealistic time and resource estimates result in schedule slippage and project failure.
- For effective project planning, in addition to a thorough knowledge of the various estimation techniques, past experience is crucial.

Project Planning

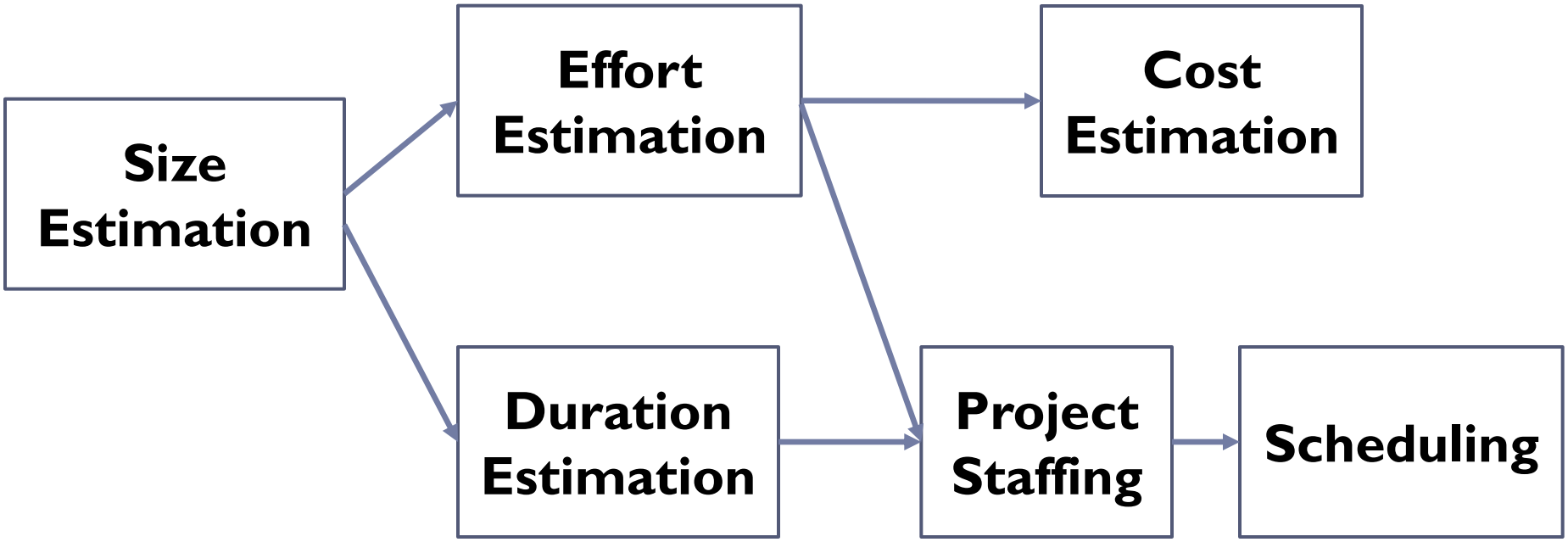
- Following activities are performed by a project manager
  - **Estimation:**
    - The following project attributes are estimated
      - **Cost:** How much is it going to cost to develop the software product?
      - **Duration:** How long is it going to take to develop?
      - **Effort:** How much effort would be necessary to develop the product?
    - Scheduling and staffing are dependent on the accuracy with which these three estimates have been made.

Project Planning (Cont..)

- Following activities are performed by a project manager
  - **Scheduling:** after project parameters have been estimated, schedules for manpower and other resources are developed.
  - **Staffing:** staff organization and staffing plans are made.
  - **Risk management:** other several plans such as quality assurance plan and configuration management plan etc.

Project Planning

- Size is the most fundamental parameter based on which all other estimations and project plans are made.



Organization of the software project management plan (SPMP) document

1. Introduction
    (a) objectives
    (b) major functions
    (c) performance issues
    (d) management and technical constraints
2. Project estimates
    (a) Historical data used
    (b) estimation techniques used
    (c) effort, resource, cost and project duration estimates

Organization of the software project management plan (SPMP) document

3. Schedule
   (a) work breakdown structure
   (b) task network representation
   (c) Gantt chart representation
   (d) PERT CHART representation
4. Project resources
   (a) people
   (b) Hardware and software
   (c) special resources

Organization of the software project management plan (SPMP) document

5. staff organization
   (a) team structure
   (b) management reporting
6. Risk management plan
   (a) risk analysis
   (b) risk identification
   (c) risk estimation
   (d) risk abatement procedures

Metrics for Project Size Estimation

- Accurate project size estimation is central to satisfactory estimation of all other project parameters such as effort, completion time and total project cost.
- Project Size is not the size of the executable code or number of bytes source code occupies.
- It is a measure of the problem complexity in terms of effort and time required to develop the product.
- Two popular metrics – Lines of Code and Function Point.

Lines of Code

- Simplest among all metrics, and extremely popular
- It measures the size of the project by counting the number of source instructions in the developed program.
- Comment lines and header lines are ignored.
- Accurate estimation of LOC count at the beginning of the project is a very difficult task
  - but easy at the end of the project.

Lines of Code

- One can estimate the count at the beginning of the project by a systematic guess
  - The problem is divided into modules and modules further divides into sub-modules and so on, until the LOC of leaf-level modules are small enough to be predicted.
- Past experience in developing similar modules is very useful.
- Total size estimation is carried out by adding estimates of all leaf-level modules.

Shortcomings

- LOC is a measure of coding activity alone.
  - It focuses on the coding activity alone.
    - Coding is only a small part of the overall software development effort.
    - The implicit assumption made by the LOC metric is flawed.
    - Design and testing efforts can be grossly disproportional to the coding effort.
    - Code size is obviously an improper indicator of the problem size.

Shortcomings

- LOC count depends on the choice of specific instructions
  - It depends on the Coding Style
    - Choice of code layout
    - Choice of instructions in writing the program
    - Specific algorithm used
  - Different programmers may lay out their code in very different way.

    Example: one programmer might write several source instructions on a single line whereas another might split a single instruction across several lines

Shortcomings

- LOC count depends on the choice of specific instructions (Cont..)
  - Different size measures for essentially identical programs
  - Problem can be overcome to a large extent by counting language tokens in a program rather than the lines of code
  - A more intricate problem arises due to specific choices of instructions made in writing the program.

Example: One might use Switch statement whereas other may use if …then …else statement

Shortcomings

- LOC count depends on the choice of specific instructions (Cont..)
  - **Note:** For same programming problem, different programmers might come up with programs having different LOC counts.

    This situation does not improve, even if language tokens are counted instead of lines of code.

Shortcomings

- LOC measure correlates poorly with the quality and efficiency of the code
  - Some programmers produce lengthy and complicated code as they do not make effective use of available instruction set or use improper algorithms
  - Poor and sloppily written piece of code have larger number of source of instructions than thoughtfully written code
  - Calculating productivity as LOC generated per man-month may encourage programmers to write lots of poor quality code

Shortcomings

- LOC metric penalizes use of higher-level programming languages and code reuse
  - LOC count will be lower when a programmer consciously use of several library routines.
  - If managers use LOC count to measure effort put in by the programmers then it would be discouraging code reuse

Shortcomings

- LOC metric measures the lexical complexity of a program and does not address the more important issues of logical and structural complexities

  - Two programs with equal LOC counts but a program incorporating complex logic would require more effort to develop than a program with very simple logic

    Example: a program having multiple nested loops and decision constructs whereas other having only sequential control flow.

Shortcomings

- It is very difficult to accurately estimate LOC of the final program from problem specification
  - LOC count is very difficult to estimate during project planning stage, and can only be accurately computed after the software development is complete.

Function Point Metric

- One of the important advantage of FP metric over LOC is that it can easily be computed from the problem specification itself.

- More the number of data items that a function reads from the user and outputs and the more number of files accessed, the higher is the complexity of the function.

Function Point Metric Computation

- The size of a software product (in units of function points) is computed using different characteristics of the product identified in its requirement specification.
  - **Step1:** Compute the unadjusted function point (UFP) using a heuristic expression.
  - **Step2:** Refine UFP to reflect the actual complexities of the different parameters used in UFP computation.
  - **Step3:** Compute FP by further refining UFP to account for the specific characteristics of the project that can influence the entire development effort.

Function Point Metric Computation

Step1: UFP Computation

- UFP is the weighted sum of five characteristics of a product.
- The weights associated with the five characteristics were determined empirically by Albrecht through data gathered from many projects.

Function Point Metric Computation

- **UFP Computation**

  UFP = (#Inputs) * 4 + (#Outputs) *5 +
  (#Inquiries) *4 + (#Files) * 10 +
  (#Interfaces)*10

- **Number of inputs:**
  - Each data item input by the user is counted.
  - Related inputs are grouped together and counted as a single input.
  - Employee age, name, sex, address, phone number etc. since they describe a single employee.

Function Point Metric Computation

- **Number of outputs:**
  - Includes reports printed, screen outputs, and error messages produced etc.
- **Number of inquiries:**
  - An inquiry is a user command (without any input data) and only requires some actions to be performed.
- **Number of files:**
  - Logical files represents a group of logically related data.
  - It includes data structures as well as physical files.

Function Point Metric Computation

- **Number of interfaces:**
  - Different mechanisms that are used to exchange information with other systems.
  - Example: data files on tapes, disks, communication links with other systems etc.

Function Point Metric Computation

Step 2: Refine parameters

- UFP computed in step1 is a gross indicator of the problem size.
- UFP needs to be refined.
- Each parameter is assumed to be of average complexity. This is rarely true.

Function Point Metric Computation

Refinement of Function Point Entities

Type	Simple	Average	Complex
Input(I)	3	4	6
Output (O)	4	5	7
Inquiry (E)	3	4	6
Number of files (F)	7	10	15
Number of Interfaces	5	7	10

Function Point Metric Computation

- **Step 3: Refine UFP based on complexity of the overall project**
  - Albrecht identified 14 parameters that can influence the development effort.
    - Requirement for reliable backup and recovery
    - Requirement for data communication
    - Extent of distributed processing
    - Performance requirements
    - Expected operational environment
    - Extent of online data entries
    - Extent of multi-screen

Function Point Metric Computation

- **Step 3: Refine UFP based on complexity of the overall project (cont..)**
  - Extent of online updating of master files
  - Extent of complex input, outputs, online queries and files
  - Extent of complex data processing
  - Extent of currently developed code can be designed for reuse
  - Extent of conversion and installation included in the design
  - Extent of multiple installations in an organization and variety of customer organizations
  - Extent of change and focus on ease of use

Function Point Metric Computation

Step 3: Refine UFP based on complexity of the overall project (cont..)

- Each of 14 parameters is assigned a value from 0 (no influence) to 6 (strong influence).
  - Resulting numbers are summed yielding a total degree of influence (DI).
- A technical complexity factor (TCF) for the project is computed.
  - TCF = (0.65+0.01*DI), where DI vary from 0 to 84.
  - TCF vary from 0.65 to 1.49.
- Finally, FP = UFP * TCF.

Shortcomings of Function Point Metric

- It does not take into account the algorithmic complexity of a function.
- It assumes same efforts required to design and develop any two different functionalities of the system is same
- To overcome this above issue, feature point metric has been developed.
  - Feature point metric incorporates algorithmic complexity as an extra parameter.

Shortcomings of Function Point Metric

- Function point and Feature point are language-independent and can be easily computed from SRS during project planning itself

- Subjective nature of Function point

E**xample:** functionality requires the employee name and employee address to be input. One considers both items as a single data unit whereas other take it as two different data units. Ambiguity leave scope to project managers to come with different function point measures essentially for the same problem.