Surprise Test 2

Student Name: Sahil Kaundal UID: 21BCS8197

Branch: CSE (Lateral Entry) Section/Group:20BCS-807\_B Semester: 4th Date of Performance: 04/04/2022

Subject Name: SE Subject Code: 20CSP-254

Q1.

Highlights the various types of risks associated during software development.

Answer:

The various type of risks are:

Schedule Risk:

Schedule related risks refers to time related risks or project delivery related planning risks. The wrong schedule affects the project development and delivery.

Time is not estimated perfectly  
Improper resource allocation  
Tracking of resources like system, skill, staff etc  
Frequent project scope expansion  
Failure in function identification and its’ completion

Budget Risk:

Budget related risks refers to the monetary risks mainly it occurs due to budget overruns.

Wrong/Improper budget estimation  
Unexpected Project Scope expansion  
Mismanagement in budget handling  
Cost overruns  
Improper tracking of Budget

Operational Risks:

Operational risk refers to the procedural risks means these are the risks which happen in day-to-day operational activities during project development due to improper process implementation or some external operational risks.

Technical Risks:

Technical risks refers to the functional risk or performance risk which means this technical risk mainly associated with functionality of product or performance part of the software product.

Programmatic Risks:

Programmatic risks refers to the external risk or other unavoidable risks. These are the external risks which are unavoidable in nature. These risks come from outside and it is out of control of programs.

Q2.

Explain statistical methods for software metric estimation and evaluation.

Answer:

Just as we typically need to determine the weight, volume, and dynamic flight characteristics of a developmental aircraft as part of the planning process, you need to determine how much software to build. One of the main reasons software programs fail is our inability to accurately estimate software size. Because we almost always estimate size too low, we do not adequately fund or allow enough time for development. Poor size estimates are usually at the heart of cost and schedule overruns.

The key to credible software sizing is to use a variety of software sizing techniques, and not to rely on a single source or method for the estimate. Reliance on a single source or technique represents a major contribution to program cost and schedule risk, especially if it is based exclusively on a contractor’s bid. There are two common types of size inaccuracies for which you can compensate to some degree.

1. Normal statistical inaccuracies can be dealt with by using multiple data sources and estimating methodologies, or by using multiple organizations to do the estimating and check and analyze results.

2. The earlier the estimate is made — the less is known about the software to be developed — and the greater the estimating errors.

Basing your estimates on more than one source is sound advice for both types of discrepancies. Because accuracy can be improved if estimates are performed with smaller product elements, base your estimates on the smallest possible unit of each component. Then compile these calculations into composite figures.

Given our shortcomings in size estimation, it is absolutely critical that you measure, track, and control software size throughout development. You need to track the actual software size against original estimates (and revisions) both incrementally and for the total build. Analysis is necessary to determine trends in software size and functionality progress. Data requirements for these measures are stated in Contract Data Requirements List (CDRL) items and should include:

* The number of distinct functional requirements in the Software Requirement Specification (SRS) and Interface Requirement Specification (IRS),
* The number of software units contained in the Software Development Plan (SDP) or Software Design Description (SDD), and Source lines-of-code (SLOC) or function point estimates for each computer software configuration item (CSCI) and build compared to the actual source code listing for each software unit.

Software size has a direct effect on overall development cost and schedule. Early significant deviations in software size data indicate problems such as:

* Problems in the model(s), logic, and rationale used to develop the estimates,
* Problems in requirements stability, design, coding, and process,
* Unrealistic interpretation of original requirements and resource estimates to develop the system, and
* Faulty software productivity rate estimates.

Significant departures from code development estimates should trigger a risk assessment of the present and overall effort. Size-based models should be revisited to compare your development program with those of similar domain, scope, size, and complexity, if possible.

Q3.

There exist different types of COCOMO Models discuss in detail and give phase-wise distribution of effort. How is function point metric advantageous over LOC metric? Explain.

Answer:

COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. Any of the three forms can be adopted according to our requirements. These are types of COCOMO model:

1. Basic COCOMO Model
2. Intermediate COCOMO Model
3. Detailed COCOMO Model

Basic COCOMO can be used for quick and slightly rough calculations of Software Costs. Its accuracy is somewhat restricted due to the absence of sufficient factor considerations.

Intermediate COCOMO takes these Cost Drivers into account and Detailed COCOMO additionally accounts for the influence of individual project phases, i.e in case of Detailed it accounts for both these cost drivers and also calculations are performed phase-wise henceforth producing a more accurate result.

Design  
Effort : 16% to 18%  
Time : 19% to 38%

Plan / Requirements  
EFFORT : 6% to 8%  
DEVELOPMENT TIME : 10% to 40%

Programming  
Effort : 48% to 68%  
Time : 24% to 64%

Integration & Test  
Effort : 16% to 34%  
Time : 18% to 34%

*% depend on mode & size*

In general, people prefer the functional size of software indicated as Function Point for one very important reason, i.e, the size expressed using the Function point metric stays constant in any case and whichever language or languages are used.