V516 Homework_2
Sai Su

Part_1
The “Inventory System” is set up to assure that the supplies and raw materials the business depends on are always on hand (available). To accomplish this, there is an inventory-monitor that continuously monitors the current level of all supplies and raw materials (stock items). The Inventory-monitor provides the data about stock-on-hand to the Inventory System. The Inventory system receives invoices from Suppliers when supplies and raw materials (stock items) are delivered and added to the inventory. The Inventory System is capable of determining when to re-order supplies or raw-materials from a supplier. To accomplish the re-ordering task, the Inventory system has threshold values for each stock item; when the inventory level of a given stock item reaches it’s threshold, the Inventory System generates an order to a Supplier to replenish the inventory level for that stock item(s).

For the following Context and Level_0 data flow diagram, redraw the Level_0 diagram using appropriate diagramming software, labeling each data flow. Use snipping tool or other recording method to capture your completed diagram. Add additional sub-processes (5.0,6.0…) if you feel they are necessary. (5 points)
Part 2: (10 points)
Create a Level_0 DFD for this version of the ParkIT_DC Context Diagram:

**What is Park It DC?**
Park It DC is an application that allows you to check a specific area in the district for parking information. Why not check parking information before you leave and see what streets you can park on. Check what meters cost money and what ones are free. Park It DC will even help you not get a ticket for parking in an RPP zone!

**How does it Work?**
In Short, we store all data in a database. Every 10 mins. we make a call to the crime and service request feed and either update the already existing record or insert a new one. Then we use Google Maps API calls to read XML files that we generate from the database and plot the data on the map.

**RPP = Residential Parking Permit**
**Data Flow Diagram (context level) for Park It information system:**
Part 3:
Write a memo - 500 words or less- to your future public_sector boss, advising them why your agency (organization) should adopt and standardize on an Open Source platform like LAMP. (5 points)

- What is open source platform, like LAMP
  Open Source Software (OSS) was developed as a method of software distribution and testing and has become a method of software production that is accompanied by flexible licensing and a potential means of invention and social aggregation that is characterized by growth and open diffusion. LAMP is an evolving integrated architecture for writing and deploying applications assembled from open source products. It has been viewed as a low-end development platform, users can mix and match components to suit their needs.

- Why public sector should adopt OSS (LAMP)?
Public sectors should adopt free and open source software such as LAMP because it can protect and extend transparency and accountability in e-governments and offer scope for technology to be socially shaped by citizens and associations as well as by administrators and private interests. The detailed reasons for adoption are listed below:

1) Access and participation. Document formats must allow citizens to read and submit documents without purchasing expensive tools, so that citizens and outside developers can comment and contribute.

2) Public ownership. Whatever tools are developed or purchased by the government or other public agencies should belong to the public, as long as no security issues are involved.

3) Easier archiving. A vendor can stop supporting a format at any time or make changes that leave documents incompatible with earlier ones, then the agencies’ preservation of documents, especially big data will be difficult and costly without open source software.

4) Transparency or security. Unlike any closed software, LAMP allows the public to trust that the tools are accurate and have no security flaws, or users can even find and fix security flaws by accessing the source code without waiting for the vendor.

5) Low-cost. In the long run, agencies can save a lot of money by investing in programming or system administration skills, or hiring a firm to maintain the free software.

- Why should standardize OSS?

It is necessary to standardize OSS because there are potential risks to run away with the standardization mantle. Good and impartial third-party governance helps avoid the creation of overlapping, nondiverse, fragmental and confusing projects, and creates connectivity for a project. Additionally, standardization establishes an effective development process—not only for new contributions, but also for maintenance, updates, and releases to prevent security flaws. Proper governance also provides essential business, legal, management, and strategic processes that ensure a proper ownership and licensing of contributions, release management, and open community involvement.

Conclusively, the fundamental reason for the standardization of open source is to promote communications and to establish governance over the use of a global commercial and information infrastructure.

- Tips

An organizational change in the form of policy, governance, staff skills/training has to happen within the public organization in order to successfully implement OSS. Even though there are some barriers in the adoption and standardization of the open source platform, the benefit it brings would outweigh the cost as long as public leaders can have a thorough understanding of OSS (LAMP) and make good use of it.

**Part 4:**

There have been many failures in the development of information systems using traditional SDLC. Discuss some characteristics of the methodology that could contribute to the high failure rate. Provide a description/overview of a failed system and suggest “what went wrong”. (5 points)

The systems development life cycle (SDLC) is a conceptual model composed of a number of clearly defined and distinct work phases which are used by systems engineers and systems developers to start from an initial feasibility study through maintenance of the completed application. The general phases in SDLC are: Requirements, Design, Construction, Testing,
Delivery and Maintenance, across different LC models. There are some characteristics of the methodology that could contribute to the high failure rate.

1) Before the project really starts, the IT team need to understand what the client wants, the domain intricacies, decide the best suited technology and accordingly calculate the cost and make time schedule. But sometimes the team does not contain business analyst, or has a poor communication with the clients, so that cannot construct the project professionally according to the specific requirements. A failed system often reveals that the team provides less time for requirement phase, or the team’s technology is not robust enough, or testing team does not involve during the requirement phase to do error checking.

2) A failed system usually has weak design since the team spends least time on designing and pushed to coding, or rarely update their design documents. So the design does not withstand user load and data load, not flexible to change and the interface is incompatible, and the environments the application should work in is not properly addressed.

3) The goals of construction phase are to implement the design and to create maintainable source code. But the system may fail in this step because of the problems on source code. For example, a failed system contains source code that is not compliant to coding standards, or with junk code accumulation, and sometimes has bad tinkering of generated source code. Besides, the problems may also result from construction on non-baselined requirements, memory leaks, security threats and other performance issues rampant.

4) A system usually fails to validate whether the applications address all user requirements, validate technical performance and concern all possible conditions. This failure is due to many reasons during testing phase, for example: the late entering of tester in the project cycle; test environment does not fully simulate end user environment; test cases are not comprehensive. Problems also possibly lie in the unidentified root causes of defects, which actually could be bad requirements gathering or inconsistent design.

5) Problems can also happen during the delivery of system, causing several defect and various issues during acceptance testing. This kind of problems may be resulted from manpower ramp up while delivery date nears. And if there is a flare-up in the delivery, project would go in full throttle, revenue leakages would happen in various channel, and there would be no proper accounting for many of the above referred revenue leakages.

6) During the system’s maintenance period, the system also may not avoid failure when regression testing is missed out; or existing source is not understandable, leading to huge re-work; or any deficient design is carried forward for want of huge extra effort to do permanent fix.