**Chapter 1: Foundation of system development**

**Information Systems Analysis and Design**

-Defined as the complex, challenging, and simulating organizational process that a team of business and systems professionals uses to develop and maintain information systems

**Application Software**

Software designed to support organizational function or process

**Systems Analyst**

Organizational role most responsible for analysis and design of information systems

**Define information systems analysis and design**

1950s

- Goal was efficiency of processing

- Emphasis was on automating existing processes

- All applications developed in machine or assembly language

1960s

- Advent of procedural (third-generation) languages

- Enabled development of smaller, faster, less expensive computers

1970s

- System development came to be more disciplined

- Became more like engineering as focus shifted from process first to data first

1980s

- Marked by major breakthroughs in organizations as microcomputers became key organizational tools

- Software industry expanded writing off-the-shelf software

- 4G L development led to instructing computers what to do instead of how to do it

1990s

- Focused on system integration

- Developers used visual programming environments (Visual Basic)

- Relational and object-oriented databases developed

- Enterprise-wide systems developed

- Web and Internet applications begun and expanded

Present day

- Continued focus on developing systems for the Internet and for firm’s intranets and extranets

- Implementation involving three-tier design

* + - Database on one server
    - Application on second server
    - Client logic located on user machines

- Move to wireless system components (access from anywhere)

- Continuing trend toward assembling systems from programs and components purchased off the shelf

* Get the system user involved
* Use a problem-solving approach
* Establish phases and activities
* Document through development
* Establish standards
* Manage process and the projects
* Justify systems as capital investments
* Don’t be afraid to cancel or revise scope
* Divide and conquer

**Describe the information systems development life cycle (SDLC)**

**Systems development methodology**

- A standard process followed in an organization to conduct all the steps necessary to analyze, design, implement, and maintain information systems

**The systems development life cycle (S D L C)**

- The traditional methodology used to develop, maintain, and replace information systems

* + - Features several phases that mark the progress of the systems analysis and design efforts

**Evolutionary Model**

- A spiral process in which one is constantly cycling through phases at different levels

**Phases of the SDLC**

**Planning**

- Need for a new or enhanced system is identified

- Needs are identified, analyzed, prioritized, and arranged

- Determine the scope of the proposed system

-Baseline project plan is developed

**Analysis**

- System requirements are studied from user input and structured

- Requires careful study of current systems, manual and computerized, that might be replaced or be enhanced

- Output is description of the alternate solution recommend by the analysis team

**Design**

- Analyst converts the alternate solution into logical and physical specifications

**- Logical Design**

* + - The design process part that is independent of any specific hardware or software platform

**- Physical Design**

* + - The logical specifications of the system from logical design are transformed into technology-specific details from which all programing/system construction can be accomplished

- Choices of language, database, and platform are many times already decided by the organization or client

**Implementation**

- Occurs when the information system is coded, tested, installed, and supported in the organization

- New systems become part of the daily activities of the organization

**Maintenance**

- The phase in which an information system is systematically repaired and improved

- Organization’s needs may change over time requiring changes to the system based on user’s needs

|  |  |
| --- | --- |
| **Phase** | **Products, Outputs, or Deliverables** |
| Planning | * Priorities for system and projects; an architecture for data, networks, and selection hardware, and information systems management are the result of associated systems * Detailed steps, or work plan, for project * Specification of system scope and planning and high-level system requirements or features * Assignment of team members and other resources * System justification or business case |
| Analysis | * Description of current system and where problems or opportunities exist, with a general recommendation on how to fix, enhance, or replace current system * Explanation of alternative systems and justification for chosen alternative |
| Design | * Functional, detailed specifications of system elements (data, processes, inputs, and outputs) * Technical, detailed specifications of all system elements (programs, files, network, system software, etc.) * Acquisition plan for new technology |
| Implementation | * Code, documentation, training procedures, and support capabilities |
| Maintenance | * New versions or releases of software with associated updates to documentation, training, and support |

- The Analysis-Design-Code-Test Loop is an example of traditional practice

**Heart of Systems Development**

- Current practice combines analysis, design, and implementation into a single process

**The SDLC Traditional Waterfall Problems**

- Once one phase ends another begins, going downhill until complete

- Makes it difficult to go back

- Results in great expense to make changes

- Role of system users or customers narrowly defined

- Focused on deadlines

**Agile methodologies share three key principles:**

* 1. A focus on adaptive rather than predictive methodologies
  2. A focus on people rather than roles
  3. A focus on self-adaptive processes

- The agile methodologies group argues that software development methodologies adapted from engineering generally do not fit with real world software development

**The Manifesto for Agile Software Development** (Table 1-2)

- Seventeen anarchists agree

- We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

* + - Individuals and interactions over processes and tools
    - Working software over comprehensive documentation
    - Customer collaboration over contract negotiation
    - Responding to change over following a plan

- That is, while we value the items on the right, we value the items on the left more. We follow the following principles:

* + - The highest priority is to satisfy the customer through early and continuous delivery of valuable software.
    - Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.
    - Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
    - Businesspeople and developers work together daily throughout the project.
    - Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.
    - The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
    - Working software is the primary measure of progress.
    - Continuous attention to technical excellence and good design enhances agility.
    - Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
    - Simplicity—the art of maximizing the amount of work not done—is essential.
    - The best architectures, requirements, and designs emerge from self-organizing teams.
    - At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.
    - Agile methodologies are not for everyone
    - Fowler recommends an agile process if your project involves

- unpredictable or dynamic requirements

- responsible and motivated developers

|  |  |  |
| --- | --- | --- |
| **Factor** | **Agile Methods** | **Traditional Methods** |
| Size | Well matched to small products and teams Reliance on tacit knowledge limits scalability | Methods evolved to handle large products and teams Hard to tailor down to small products |
| Criticality | Untested on safety-critical products  Potential difficulties with simple design and lack of documentation | Methods evolved to handle highly critical products Hard to tailor down to products that are not critical. |
| Dynamism | Simple design and continuous refactoring are excellent for highly dynamic environments but a source of potentially expensive rework for highly stable environments | Detailed plans and Big Design Up Front, excellent for highly stable environment but a source of expensive rework for highly dynamic environments |
| Personnel | Requires continuous presence of a critical mass of scarce experts  Risky to use non-agile people | Needs a critical mass of scarce experts during project definition but can work with fewer later in the project, unless the environment is highly dynamic |
| Culture | Thrives in a culture where people feel comfortable and empowered by having many degrees of freedom (thriving on chaos) | Thrives in a culture where people feel comfortable and empowered by having their roles defined by clear practices and procedures (thriving on order) |

- customers who understand the process and will get involved

**eXtreme Programming**

* Short, incremental development cycles
* Focus on automated tests written by programmers
* Emphasis on two-person programming teams
* Customers to monitor the development process
* Relevant parts of eXtreme Programming that relate to design specifications are
  1. How planning, analysis, design, and construction are all fused into a single phase of activity
  2. Its unique way of capturing and presenting system requirement and design specifications
* Coding and testing are related parts of the same process
* Advantages include
  1. Increased communications among developers
  2. Higher levels of productivity
  3. Higher quality code
  4. Reinforcement of other practices in eXtreme Programming
     + Include code-and-test discipline

**Scrum**

* Originated in 1995 by Sutherland and Schwaber
* Most popular methodology for agile (58%)
  + Scrum framework includes
  + Scrum teams with associated roles, events, artifacts, and rules
  + Each team consists of three roles
    - Product owner
    - Development team
    - Scrum master
* Scrum designed for speed and multiple functional product releases
* Primary unit is the Sprint (runs two weeks to a month)
  + Starts with an eight-hour planning meeting
    - What needs to be delivered by the end of the sprint
    - How will the team accomplish that work
  + Daily Standup: A 15-minute meeting held to evaluate progress made within the past 24 hours and what needs to be done
  + At the end of the sprint, two additional meetings
    - The Sprint Review: (4 hours) focusing on the product, what has been accomplished, and what needs to be done
    - The Sprint Retrospective: (3 hours) focusing on team performance and how it can improve
  + Three primary artifacts in the Scrum process
    - Product Backlog: Listing of potential requirements
    - Sprint Backlog: Listing of only items to be addressed in a particular sprint
    - Increment: Represents the sum of all the Product Backlog items completed during a sprint.

**Agile in Practice**

* Three primary factors critical for success
  + Delivery strategy: Continuous delivery of working software in short time scales
  + Following agile software engineering practices
  + Team capability: Agile principle of building projects around motivated individuals
* Agile development offers managers and programmers more choice in their efforts to produce good systems that come in on time and under budget

**Explain object-oriented analysis and design and the Rational Unified Process (RUP)**

* Based on objects rather than data or processes
* Combines data and processes (called **methods**) into single entities call objects
* **Object**: A structure that encapsulates attributes and methods that operate on those attributes
* **Inheritance**: Hierarchical arrangement of classes enabling subclasses to inherit properties of superclasses
* **Object Class**: Logical grouping of objects that have the same attributes and behaviors
* **Rational Unified Process (R U P)** is an object-oriented systems development methodology
* Based on an iterative, incremental approach to systems development
* R U Ps four phases (each can be further divided)

- Inception

- Elaboration

- Construction

- Transition

* Criticisms of the SDLC include

- Forced timed phases on intangible and dynamic processes were doomed to fail

- Life-cycle reliance has resulted in massive amounts of process and documentation

- Cycles are not necessarily waterfalls

**Chapter 2: Software development life cycle**

**Describe the RUP methodologies**

A software engineering process based on best practices in modern software development

- A disciplined approach to assigning and managing tasks and responsibilities in a development organization

- Focus on high quality software that meets the needs of its end users within a predictable schedule and budget

A process framework that can be tailored to specific organization or project needs

RUP is a methodology for delivering projects in a maximum performance manner

Risk-driven process

* + Risk management integrated into the development process
  + Iterations are planned based on high priority risks

Use-Case driven development

* + Use cases express requirements on the system’s functionality and model the business as context for the system
  + Use cases are defined for the intended system and are used as the basis of the entire development process

Architecture-centric design

* + Architecture is the primary artefact to conceptualize, construct, manage, and evolve the system
  + Consists of multiple, coordinated views (or models) of the architecture
* In RUP, the process is described at two levels: the discipline level and the workflow detail level. A **Workflow** is a grouping of activities that are often performed "together" to produce a specific result. In particular, workflow details describe groups of activities performed together in a discipline.
* The workflows for the RUP disciplines and workflow details are described using Unified Modeling Language (UML) activity diagrams. Discipline diagrams contain the workflow details of the discipline.

**Inception Phase**

*Inception is the first of four RUP phase its all about getting familiar with Project goal and Scope .this phase help you determine the project feasibility , what customer want and how will you get into more resource consumable phase.*

Objective:

* Understand what to build.
  + A vision document:
  + Optional business model
  + An initial project glossary
* Identify key system functionality.
  + A initial use-case model (10% -20%) complete.
* Determine at least one possible solution.
  + One or several prototypes.
* Understand the costs, schedule, and risks associated with the project.
  + An initial risk assessment.
  + Business case
* Decide what process to follow and what tools to use.
  + A project plan

**Elaboration phase**

*Elaboration is the second of the four phases in the RUP approach. The goal of the Elaboration phase is to define and baseline the architecture of the system in order to provide a stable basis for the bulk of the design and implementation effort in the Construction phase. The architecture evolves out of a consideration of the most significant requirements (those that have a great impact on the architecture of the system) and an assessment of risks.*

* Deeper Requirement understanding
  + - At least 80% complete use-case model
    - Supplementary requirements capturing
      * non functional requirements
      * None Use case requirement
* Architect consideration.
  + - A Software Architecture Description.
    - An executable architectural prototype.
      * Risk mitigation and Accurate Cost/Scapulae
    - A revised risk list and a revised business case.
      * Development Case refinement
    - A development plan for the overall project
      * coarse-grained project plan
      * showing iterations
      * evaluation criteria for each iteration.

**Construction Phase**

Construction is really about cost-efficient development of a complete product—an operational version of your system—that can be deployed in the user community

**What is Agile**

Agile is a time boxed, iterative approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver it all at once near the end

* Minimize development costs and achieve some degree of parallelism
* Iteratively develop a complete product that is ready to transition to its user community
* The software product integrated on the adequate platforms.
* The user manuals.
* A description of the current release.

**Transition Phase**

The purpose of the transition phase is to transition the software product to the user community. Once the product has

been given to the end user, issues usually arise that require you to develop new releases, correct some problems, or

finish the features that were postponed.

* “beta testing” to validate the new system against user expectations
* parallel operation with a legacy system that it is replacing
* conversion of operational databases
* training of users and maintainers
* roll-out the product to the marketing, distribution, and sales teams
* Improve future project performance through lessons learned
* In RUP, the process is described at two levels: the discipline level and the workflow detail level. A **Workflow** is a grouping of activities that are often performed "together" to produce a specific result. In particular, workflow details describe groups of activities performed together in a discipline.
* The workflows for the RUP disciplines and workflow details are described using Unified Modeling Language (UML) activity diagrams. Discipline diagrams contain the workflow details of the discipline.

**Describe the Agile methodologies**

Agile is a time boxed, iterative approach to software delivery that builds software incrementally from the start of the project, instead of trying to deliver it all at once near the end

It works by breaking projects down into little bits of user functionality called **user stories**, prioritizing them, and then continuously delivering them in short two week cycles called **sprints**

|  |  |
| --- | --- |
| **RUP**: | **Agile:** |
| **Heavyweight**, **Is a Framework** | **Lightweight**, **Like a Philosophy** |
| * Use-case driven from inception to deployment, small team size * Architecture-centric, model visually, well-documented, lots of artifact * Iterative and incremental, where large projects are divided into smaller projects * Long-term detailed planning * It mostly relies on the ability of experts and professionals to assign the activities to individuals * It is a rather complex method which makes its implementation challenging, particularly for smaller businesses, teams or projects * A long time is spent doing requirements or design work before programming starts * Too many roles, artifacts required * High cost in managing, following RUP project * Tool-set are useful but expensive | * Small cross-functional teams are used * Daily status meetings are held, less artifact can be better * Short timeframe increments for each  change, short-term planning * A working project is completed at  end of each iteration and  demonstrated to stakeholders * It can be difficult to keep the interest of customers / users who are involved in the process. * Team members may be unsuited to the intense involvement that characterizes agile methods. * Prioritizing changes can be difficult where there are multiple stakeholders. * Maintaining simplicity requires extra work. * Contracts may be a problem as with other approaches to iterative development. * Because of their focus on small, tightly-integrated teams, there are problems in scaling agile methods to large systems. * Less emphasis on documentation - harder to maintain when you get a new team for maintenance |

**Chapter 3: Determining software requirements**

* **Characteristics of a good systems analyst:**
  + Impertinence – question everything
  + Impartiality – consider all issues to find the best solution
  + Relax constraints – assume anything is possible and eliminate the infeasible
  + Attention to detail – every fact must fit with every other fact
  + Reframing – challenge yourself to look at the organization in new ways
* **Systems analysts need to understand:**
  + Business objectives that drive what and how work is done
  + Information people need to do their jobs
  + The data (definition, volume, size) handled in support of jobs
  + Data transformation and storage (when, how, by whom)
  + Data handling dependencies and sequences
  + Data handling and processing rules
  + Policies and guidelines that describe the nature of the business and market and the environment it operates in
  + Key events that affect data values and when they occur

|  |
| --- |
| **Deliverables for Requirements Determination** |
| 1. Information collected from conversations with or observations of users: interview transcripts, notes from observation, meeting minutes |
| 1. Existing written information business mission and strategy statements, sample business forms and reports and computer displays, procedure manuals, job descriptions, training manuals, flowcharts and documentation of existing systems, consultant reports |
| 1. Computer-based information: results from J A D sessions, reports of existing systems, and displays and reports from system prototypes |

|  |
| --- |
| **Traditional Methods of Collecting System Requirements** |
| * Individually interview people informed about the operation and issues of the current system and future systems needs |
| * Interview groups of people with diverse needs to find synergies and contrasts among system requirements |
| * Observe workers at selected times to see how data are handled and what information people need to do their jobs |
| * Study business documents to discover reported issues, policies, rules, and directions as well as concrete examples of the use of data and information in the organization |

* **Open-ended questions –** questions in interviews that have no prespecified answers
* **Closed-ended questions –** questions in interviews that ask those responding to choose from among a set of specified responses

**Interviewing Guidelines**

* Don’t phrase a question in a way that implies a right or wrong answer
* Listen carefully to what is being said
* Record notes within 48 hours after an interview
* Don’t set expectations about the new system unless you know these will be deliverables
* Seek a variety of perspectives from the interviews

**Interviewing Groups**

* Drawbacks to interviewing individuals:
  + Reconciling contradictions in information collected
  + New interviews may require new questions
  + Not an efficient process
* Group interview advantages:
  + More effective use of time
  + Allows synergy when groups can hear each other
* Primary disadvantage is difficulty in scheduling with multiple people involved
* **Nominal group technique (NGT)** – facilitated process that supports idea generation by groups. At the beginning of the process, group members work alone to generate ideas. The ideas are then pooled under the guidance of a trained facilitator.
* End result is a listing of either problems or features generated and prioritized by the group
* Can be used as part of a JAD effort

**Directly Observing Users**

* Direct observation of workers:
  + Watching users work at their jobs
  + Observe actual measure of how employees interact with information systems and how they do their jobs
  + More accurate than interview
  + People can change their normal behavior when they know they are being observed
  + Observation cannot be continuous, thus you are getting only a snapshot of how they work

**Analyzing Procedures and Other Documents**

* An analysis of existing documents can give you a wealth of information:
  + Problems with existing systems
  + Opportunities to meet new needs with critical information
  + Identify key people of current system
  + Values of organization who help determine priorities desired by different users
  + Special information processing circumstances that might not otherwise be identified
  + Identify left out features of current software that may lead to needed features in future systems
  + Identify processing rules that must be enforced
* A written work procedure describes how a job or task is performed
* **Formal system** – official way a system works as described in organizational documentation.
* **Informal system** – way a system actually works
* Four major documents analyzed when creating a new system:
  1. Written work procedure
  2. A form such as the invoice form on the previous slide

Gives crucial information about the nature of the organization

* 1. A report such as the one on the next slide

Can be used to analyze to determine which data to capture

* 1. Documents used to describe the system and how it is used

Examples include flowcharts, data dictionaries, user manuals

**Comparison of Observation and Document Analysis**

|  |  |  |
| --- | --- | --- |
| **Characteristic** | **Observation** | **Document Analysis** |
| Information Richness | High (many channels) | Low (passive) and old |
| Time Required | Can be extensive | Low to moderate |
| Expense | Can be high | Low to moderate |
| Chance for Follow-Up and Probing | Good: probing and clarification questions can be asked during or after observation | Limited: probing possible only if original author is available |
| Confidentiality | Observee is known to interviewer; observee may change behavior when observed | Depends on nature of document; does not change simply by being read |
| Involvement of Subject | Interviewees may or may not be involved and committed depending on whether they know if they are being observed | None, no clear commitment |
| Potential Audience | Limited numbers and limited time (snapshot) of each | Potentially biased by which documents were kept or because document was not created for this purpose |

|  |
| --- |
| **Contemporary Methods for Collecting System Requirements** |
| Bringing session users, sponsors, analysts, and others together in a J A D session to discuss and review system requirements |
| Iteratively developing system prototypes that refine the understanding of system requirements in concrete terms by showing working versions of system features |

* **Joint Application Design (JAD)** – structured process in which users, managers, and analysts work together for several days in a series of intensive meetings to specify or review system requirements
  + Started by IBM in the late 1970s
  + Primary purpose is to collect system requirements simultaneously from key people involved with the system
  + Enables conflict resolution
* Typical JAD participants include:
  + **JAD session leader** – organizes and runs session
  + **Users** – key users of the system
  + **Managers** – managers of the work groups
  + **Sponsor** – high level company executive
  + **Systems analysts** – member of the systems analysis team
  + **Scribe**– records notes from session
  + **IS Staff** – I S staff composed of programmers, database analysts, I S planners, and data center personnel
* **JAD session leader** – trained individual who plans and leads Joint Application Design sessions
* **Scribe** – person who makes detailed notes of the happenings at a Joint Application Design session
* End results of a JAD:
  + Documentation detailing existing system
  + Features of proposed system

**Using Prototyping During Requirement Determination**

* **Prototyping** – iterative process of systems development in which requirements are converted to a working system that is continually revised through close collaboration between an analyst and users
  + Quickly converts basic requirements into working, limited version of final information system
  + Viewed and tested by the user
  + Prompts user for modifications for final system
* Evolutionary Prototyping
  + Begin by modeling part of the target system
    - If successful, evolve rest of the system from those parts
  + Prototype becomes the actual production system
* Throwaway Prototyping
  + Prototype is not preserved once system is built
  + Quickly developed as a mockup
* Prototyping is most useful when:
  + User requirements are not clear
  + Few users are involved in the system
  + Designs are complex and require concrete form to evaluate
  + All want specific system requirements as communication problems have existed in the past
  + Tools and data are readily available to rapidly build a prototype
* Drawbacks of prototyping as a tool include:
  + A tendency to avoid creating formal documentation
  + Difficult to adapt to other potential users
  + Built as standalones makes it difficult to adapt to other users
  + SDLC checks are often bypassed
* **Business process reengineering (BPR)** – search for, and implementation of, radical change in business processes to achieve breakthrough improvements in products and services
  + Reorganize data flow to eliminate unnecessary steps
  + Achieve synergy between previously separate steps
  + Become more responsive to future changes
  + Can be achieved through creative application of information technologies

**Identifying Processes to Reengineer**

* **Key business processes –** structured, measured set of activities designed to produce a specific output for a particular customer or market
  + Focused on organizational outcome (e.g., products)
  + Same techniques used as requirements determination
* Which activities need radical change?
  + Importance of activity to delivering an outcome
  + Feasibility of changing the activity
  + Level of dysfunction of current activity

**Disruptive Technologies**

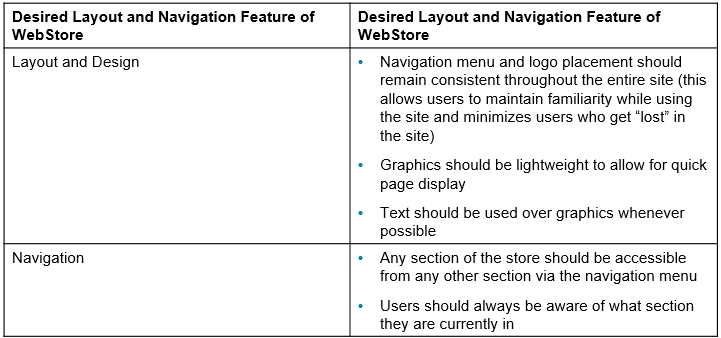
* Information technologies must be applied to radically improve business processes
* **Induction** – reasoning from the specific to the general
  + Managers learn power of new technologies and ways to innovate and alter how work is done
* **Disruptive technologies** – technologies that enable breaking long-held business rules that inhibit organizations from making radical business changes

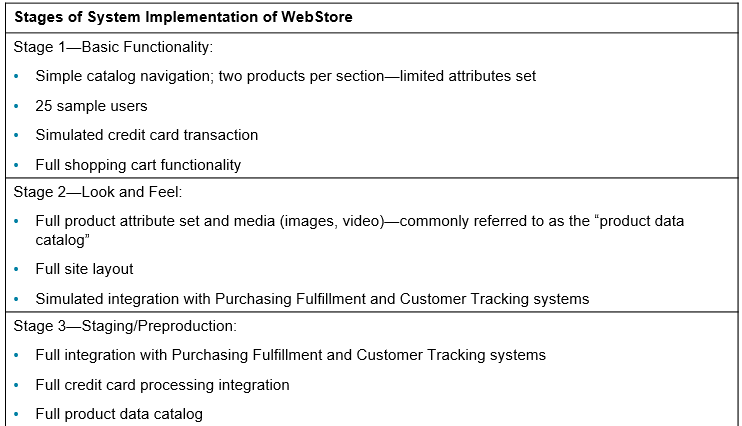
**Long-Held Organizational Rules That Are Being Eliminated through Disruptive Technologies**

|  |  |
| --- | --- |
| **Rule** | **Disruptive Technology** |
| Information can appear in only one place at one time. | Distributed databases allow the sharing of information. |
| Businesses must choose between centralization and decentralization. | Advanced telecommunications networks can support dynamic organizational structure. |
| Managers must make all decisions. | Decision-support tools can aid nonmanagers. |
| Field personnel need offices where they can receive, store, retrieve, and transmit information. | Wireless data communication and portable computers provide a “virtual” office for workers. |
| The best contact with a potential buyer is personal contact. | Interactive communication technologies allow complex messaging capabilities. |
| You have to find out where things are. | Automatic identification and tracking technology knows were things are. |
| Plans get revised periodically. | High-performing computing can provide real-time updating. |

**Steps in the Agile Usage-Centered Design Method for Requirements Determination**

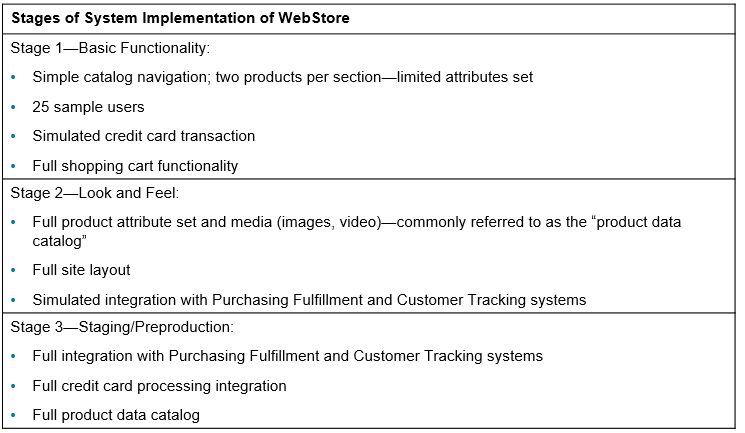
|  |
| --- |
| **Steps in the Agile Usage-Centered Design Method for Requirements Determination** |
| 1. Gather a group of people, including analysts, users, programmers, and testing staff, and sequester them in a room to collaborate on this design. Include a facilitator who knows this process. |
| 1. Give everyone a chance to vent about the current system and to talk about the features everyone wants in the new system. Record all of the complaints and suggestions for change on whiteboards or flip charts for everyone to see. |
| 1. Determine what the most important user roles would be. Determine who will be using the system and what their goals are for using the system. Write the roles on 3 × 5 cards. Sort the cards so that similar roles are close to each other. Patton (2002) calls this a **role model.** |
| 1. Determine what tasks user roles will have to complete in order to achieve their goals. Write these down on 3 × 5 cards. Order tasks by importance and then by frequency. Place the cards together based on how similar the tasks are to each other. Patton calls this a **task model.** |
| 1. Task cards will be grouped together on the table based on their similarity. Grab a stack of cards. This is called an **interaction context.** |
| 1. For each task card in the interaction context, write a description of the task directly on the task card. List the steps that are necessary to complete the task. Keep the descriptions conversational to make them easy to read. Simplify. |
| 1. Treat each stack as a tentative set of tasks to be supported by a single aspect of the user interface, such as a screen, page, or dialogue, and create a paper-and-pencil prototype for that part of the interface. Show the basic size and placement of the screen components. |
| 1. Take on a user role and step through each task in the interaction context as modeled in the paper-and-pencil prototype. Make sure the user role can achieve its goals by using the prototype. Refine the prototype accordingly. |

**Desired Layout and Navigation Feature of WebStore**

**Stages of System Implementation of WebStore** ****

**Electronic Commerce Applications: Determining System Requirements**

* Determining system requirements for Pine Valley Furniture’s WebStore
  + System layout and navigation characteristics
  + WebStore and site management system capabilities
  + Customer and inventory information
  + System prototype evolution

**Stages of System Implementation of WebStore**

**Chapter 4: Designing architecture**

**Architectural Abstraction**

* **Architecture in the small** is concerned with the architecture of individual programs.
  + At this level, we are concerned with the way that an individual program is decomposed into components.
* **Architecture in the large** is concerned with the architecture of complex enterprise systems that include other systems, programs, and program components.
  + These enterprise systems are distributed over different computers, which may be owned and managed by different companies

**Why is architecture important?**

* The architecture of a system is an abstract description of its structure and behavior
* The level of abstraction is chosen such that all critical design decisions are apparent and meaningful analysis is feasible

**Architectural Representations**

* Simple, informal block diagrams showing entities and relationships are the most frequently used method for documenting software architectures.
* But these have been criticized because they lack semantics, do not show the types of relationships between entities nor the visible properties of entities in the architecture.
* Depends on the use of architectural models. The requirements for model semantics depends on how the models are used.

**Architecture Reuse**

* Systems in the same domain often have similar architectures that reflect domain concepts
* Application product lines are built around a core architecture with variants that satisfy particular customer requirements
* The architecture of a system may be designed around one of more architectural patterns or ‘styles’
* *A major goal of the software architecture discipline is to develop an environment that promotes the reuse of components across separate software projects (Ryan O'Farrell, J. A. Hamilton)*

**Layered Architecture Pattern**

* Usage:
  + Applications that are needed to be built quickly.
  + Appropriate for teams with inexperienced developers and limited knowledge of architecture patterns.
  + Require strict standards of maintainability and testability

**Event-driven Architecture Pattern**

* Usage:
  + For applications where individual data blocks interact with only a few modules.
  + Helps with user interfaces

**Microservices Architecture Pattern**

* Usage:
  + Businesses and web applications that require rapid development.
  + Websites with small components, data centers with well-defined boundaries, and remote teams globally

**Client-server Architecture Pattern**

* Usage:
  + Applications that focus on real-time services
  + Applications that require controlled access and offer multiple services for a large number of distributed clients
  + An application with centralized resources and services that has to be distributed over multiple servers

**Designing Systems for a Client/Server Architecture**

* **Database engine** – (back-end) portion of the client/server database system running on the server that provides database processing and shared access functions
* **Client** – (front-end) portion of the client/server database system that provides the user interface and data manipulation functions
* **Application program interface (API)** – the software building blocks that are used to ensure that common system capabilities, such as user interfaces and printing, as well as modules are standardized to facilitate data exchange between clients and servers

**Advanced Forms of Client/Server Architectures**

* **Application server** – computing server where data analysis functions primarily reside
* **Virtual machine** – software emulation of a physical computer system, both hardware and operating system, that allows more efficient sharing of physical hardware resources
* **Virtualization** – act of creating virtual (rather than physical) versions of a variety of computing capabilities including hardware platforms, operating systems, storage devices, and networks
* **Three-tiered client/server architecture** – advanced client/server architectures in which there are three logical and distinct applications—data management, presentation, and analysis—that are combined to create a single information system
* **Middleware** – combination of hardware, software, and communication technologies that brings data management, presentation, and analysis together into a three-tiered (or n-tiered) client/server environment

**What is Cloud Computing?**

* **Cloud computing** – the provision of applications over the Internet where customers do not have to invest in the hardware and software resources needed to run and maintain the applications, but are charged on a per-use basis
* **Utility computing** – form of on-demand computing where resources in terms of processing, data storage, or networking are rented on an as-needed basis. The organization only pays for the services used.

**Service Models**

* **Infrastructure as a service (IaaS)** – cloud computing model in which only the basic capabilities of processing, storage, and networking are provided
* **Platform as a service (PaaS)** – cloud computing model in which the customer can run his or her own applications that are typically designed using tools provided by the service provider; the customer has limited or no control over the underlying infrastructure
* **Software as a service (SaaS)** – cloud computing model in which a service provider offers applications via a cloud infrastructure

**Cloud Characteristics**

* Cloud computing has the following characteristics that distinguish it form in-house infrastructure:
  + On-demand self-service
  + Rapid elasticity
  + Broad network access
  + Resource pooling
  + Measured service

**Service-Oriented Architecture**

* **Service-oriented architecture (S O A)** – software architecture in which business processes are broken down into individual components (or services) that are designed to achieve the desired results for the service consumer (which can be either an application, another service, or a person)
* Three main principles of S O A include:
  + **Reusability** – service should be usable in different applications
  + **Interoperability** – service should work with other service
  + **Componentization** – service should be simple and mobile

**Web Services**

* **Web service** – method of communication between two electronic devices over a network
* **eXtensible Markup Language (X M L)** – Internet authoring language that allows designers to create customized tags, enabling the definition, transmission, validation, and interpretation of data between applications
* **JavaScript Object Notation (J S O N)** – lightweight data interchange approach that is relatively easy for humans to understand and for computers to generate or interpret
* **Simple Object Access Protocol (S O A P)** – protocol for communicating X M L data between Web service applications and the operating system
* **Representational State Transfer (R E S T)** – relatively simple and fast protocol for communicating J S O N data between Web service applications and the operating system

**Designing Internet Systems**

* Most new system development focuses on Internet-based applications (for internal processing, business-to-business, and business-to-consumer)
* Internet design is simpler than client/server due to proliferation of standards

**Standards Drive the Internet**

* Types of standards include:
  + **Domain naming system (BIND)** – method for translating Internet domain names into Internet Protocol (IP) addresses. B I N D stands for Berkeley Internet Name Domain.
  + **Hypertext Transfer Protocol (HTTP)** – communication protocol or exchanging information on the Internet
  + **Hypertext Markup Language (HTML)** –standard language for representing content on the Web through the use of hundreds of command tags

**Site Consistency**

* Professionalism requires a consistent look-and-feel across all pages of a website
  + Makes it easier for users to navigate
* **Cascading Style Sheets (CSSs)** –set of style rules that tells a Web browser how to present a document
* **eXtensible Stylesheet Language (XSL)** –specification for separating style from content when generating X M L pages

**Design Issues Related to Site Management**

* Customer loyalty and trustworthiness results in customers feeling like the site, and their data, are secure. Ways to convey this:
  + **Design quality** (professional appearance)
  + **Up-front disclosure**(open and honest relationship)
  + **Comprehensive, correct, and current content**(provides users with current information)
  + **Connected to the rest of the Web**(credibility)
* **Personalization** – providing Internet content to a user based upon knowledge of that customer
* **Customization** – results in Internet sites that allow users to customize the content and look of the site based on their personal preferences

**Chapter 5: Designing Databases**

**Database Design**

* File and database design occurs in two steps:
  1. Develop a logical database model, which describes data using notation that corresponds to a data organization used by a database management system
     + Relational database model
  2. Prescribe the technical specifications for computer files and databases in which to store the data
     + Physical database design provides specifications
     + Logical and physical database design in parallel with other system design steps

**Process of Database Design**

* Four key steps in logical database modeling and design:
  1. Develop a logical data model for each known user interface for the application using normalization principles
  2. Combine normalized data requirements from all user interfaces into one consolidated logical database model (view integration)
  3. Translate the conceptual E-R data model for the application into normalized data requirements
  4. Compare the consolidated logical database design with the translated E-R model and produce one final logical database model for the application

**Physical Database Design**

* Key physical database design decisions:
  + Choosing a storage format (data type) for each attribute from the logical database model
  + Grouping attributes from the logical database model into physical records
  + Arranging related records in secondary memory (hard disks and magnetic tapes) so that records can be stored, retrieved and updated rapidly
  + Selecting media and structures for storing data to make access more efficient

**Deliverables and Outcomes**

* During logical database design you must account for every data element on a system input or output
  + Normalized relations are the primary deliverable
* **Primary key** – attribute (or combination of attributes) whose value is unique across all occurrences of a relation
* Physical database design converts relations into database tables
  + Programmers and database analysts code the definitions of the database using Structured Query Language (S Q L)

**The Relational Database Model**

* **Relational database model** – data represented as a set of related tables or relations
* **Relation** – named, two-dimensional table of data. Each relation consists of a set of named columns and an arbitrary number of unnamed rows.
* Relations have several properties that distinguish them from nonrelational tables:
  + Entries in cells are simple
  + Entries in columns are from the same set of values(domain)
  + Each row is unique
  + The sequence of columns can be interchanged without changing the meaning or use of the relation
  + The rows may be interchanged or stored in any sequence

**Well-Structured Relations**

* **Well-structured relation** – relation that contains a minimum amount of redundancy and that allows users to insert, modify, and delete the rows without error or inconsistencies; also known as a table
* 3 Anomalies:
* ->3 loại bất thường (insert, update, delete) -> làm cho ko thể lưu , xóa dât như mong muốn
* -> **Well-structured relation** ko đc tòn tại những cái đó

**Normalization**

* **Normalization** – process of converting complex data structures into simple, stable data structures
* The result of normalization is that every nonprimary key attribute depends upon the whole primary key and nothing but the primary key
* 3 normal form
* fully functional dependency
* partial dependency(2/ a part of primary key)
* transitive dependency
* **First Normal Form (1NF):**
  + Has no multivalued attributes, unique rows, and all relations are in 1N F
* **Second Normal Form (2NF):**
  + Each nonprimary key attribute is identified by the whole key (referred to as a full functional dependency)
* **Third Normal Form (3NF):**
  + Nonprimary key attributes do not depend on each other (referred to as a transitive dependency)

**Functional Dependency**

* **Functional dependency** – constraint between two attributes in which the value of one attribute is determined by the value of another attribute
* Example of attribute B being functionality dependent on attribute A
  + Dependency is represented by an arrow ( → )
  + Attribute B is functionally dependent on attribute A if, for every valid value of A, that value of A uniquely determines the value of B
  + Represented as: A → B
  + For each value of A you will find one and only one value of B mean B dependent on A
  + Ví dụ mỗi id chỉ tìm đc 1 tên
* Functional dependency is not a mathematical dependency
* Instances (or sample data) in a relation do not prove the existence of a functional dependency
* Knowledge of problem domain is most reliable method for identifying functional dependency

**Second Normal Form (2NF)**

* **Second normal form (2N F)** – relation is in second normal form if every nonprimary key attribute is functionally dependent on the whole primary key
* To convert a relation into 2N F , decompose the relation into new relations using the attributes, called **determinants**, that determine other attributes
* The determinants are the primary keys of the new relations

**Third Normal Form (3NF)**

* **Third normal form (3N F)** – relation is in second normal form and has no functional (transitive) dependencies between two (or more) nonprimary key attributes
* **Foreign key** – attribute that appears as a nonprimary key attribute in one relation and as a primary key attribute (or part of a primary key) in another relation
* **Referential integrity** – rule that states that either each foreign key value must match a primary key value in another relation or the foreign key value must be null (i.e., have no value)

**Transforming E-R Diagrams into Relations**

* Transforming an E-R diagram into normalized relations and merging all of them into one consolidated set of relations takes four steps:
  1. Represent entities (each becomes a relation)
  2. Represent relationships (each must be represented in the relational database design)
  3. Normalize the relations (make them well structured)
  4. Merge the relations (renormalize if necessary to remove redundancy)

**Representing Entities**

* Each regular entity is transformed into a relation
* The identifier of the entity type becomes the primary key of the corresponding relation
* The primary key must satisfy the following two conditions
  + The value of the key must uniquely identify every row in the relation
  + The key should be nonredundant
* The entity type label is translated into a relation name

**Representing Relationships**

* The procedure for representing relationships depends on:
  + The degree of the relationship (unary, binary, ternary)
  + The cardinality of the relationship
* **Binary 1:*N* Relationship** – represented by adding the primary key attribute (or attributes) of the entity on the one side of the relationship as a foreign key in the relation that is on the many side of the relationship
* **Binary or Unary 1:1 relationship** is represented by any of the following:
  + Add the primary key of A as a foreign key of B
  + Add the primary key of B as a foreign key of A
  + Both of the above
* **Binary and Higher-Degree *M*:*N* Relationships –** represented by creating another relations, include the primary keys of all relations into the new one as a primary key
  + Becomes a composite key
  + Any non-key attributes associated with the *M*:*N* relationship are included in the new relation
* **Unary 1:*N* Relationship** (also called recursive relationships):
  + Is modeled as a relation
  + Primary key of that relation is the same as for the entity type
  + Foreign key is added to the relation that references the primary key values
* **Recursive foreign key** – foreign key in a relation that references the primary key values of that same relation
* **Unary *M*:*N* Relationship** is modeled as one relation, then:
  + Create a separate relation to represent the *M*:*N* relationship
  + The primary key of the new relation is a composite key of two attributes that both take their values from the same primary key
  + Any attribute associated with the relationship is included as a nonkey attribute in this new relation

**E-R Diagrams to Relational Transformation**

|  |  |
| --- | --- |
| **E-R Structure** | **Relational Representation** |
| Regular entity | Create a relation with primary key and nonkey attributes. |
| Weak entity | Create a relation with a composite primary key (which includes the primary key of the entity on which this weak entity depends) and nonkey attributes. |
| Binary or unary 1:1 relationship | Place the primary key of either entity in the relation for the other entity or do this for both entities. |
| Binary 1:*N* relationship | Place the primary key of the entity on the one side of the relationship as a foreign key in the relation for the entity on the many side. |
| Binary or unary *M:N*  relationship or  associative entity | Create a relation with a composite primary key using the primary keys of the related entities, plus any nonkey attributes associative entity of the relationship or associative entity. |
| Binary or unary *M:N*  relationship or associative entity with additional key(s) | Create a relation with a composite primary key using the primary keys of the related entities and additional primary key attributes associated with the relationship or associative entity, plus any nonkey attributes of the relationship or associative entity. |
| Binary or unary *M:N*  relationship or associative entity with its own key | Create a relation with the primary key associated with the relationship or associative entity, plus any nonkey attributes of the relationship or associative entity and the primary keys of the related entities (as foreign key attributes). |
| Supertype/subtype | Create a relation for the superclass, which contains the primary relationship key and all nonkey attributes in common with all subclasses, plus create a separate relation for each subclass with the same primary key (with the same or local name) but with only the nonkey attributes related to that subclass. |

**Merging Relations**

* Merging relations
  + Is the last step in the logical database design
  + Purpose is to remove redundant relations
  + Example when given two relations:
    - EMPLOYEE1(Emp\_ID,Name,Address,Phone)
    - EMPLOYEE2(Emp\_ID,Name,Address,Jobcode,Number\_of\_Years)
  + They can be merged together:
    - EMPLOYEE(Emp\_ID,Name,Address,Phone,Jobcode,Number\_of\_Years)

**View Integration Problems**

* Must understand the meaning of the data and be prepared to resolve any problems that arise in the process
* **Synonym** – two different names used for the same attribute
  + When merging, get agreement from users on a single, standard name
  + Example of two relations with synonym primary keys (representing S S N numbers) of different names:
    - STUDENT1(Student\_ID,Name)
    - STUDENT2(Matriculation\_Number,Name,Address)
* **Homonym –** single attribute name that is used for two or more different attributes
  + Resolved by creating a new descriptive name
  + Example: home address versus local address?
    - STUDENT1(Student\_ID,Name,Address)
    - STUDENT2(Student\_ID,Name,Phone\_Number,Address)
* **Dependencies between nonkeys** occurs when two 3N F relations are merged to form a single relation such as:
  + STUDENT1(Student\_ID,Major)
  + STUDENT2(Student\_ID,Adviser)
  + Since both have the same primary key they can be merged as follows:
    - STUDENT(Student\_ID,Major,Adviser)
* If a transitive dependency exists such as Major → Advisor
  + You need to **normalize** to remove the transitive dependency
    - STUDENT(Student\_ID,Major)
    - MAJOR ADVISER(Major,Adviser)
* **Class/Subclass** relationships may be hidden in user views or relations. Consider the following:
  + PATIENT1(Patient\_ID,Name,Address,Date\_Treated)
  + PATIENT2(Patient\_ID,Room\_Number)
* The answer? Convert it to a Supertype/Subtype!
  + PATIENT(Patient\_ID,Name,Address)
  + INPATIENT(Patient\_ID,Room\_Number)
  + OUTPATIENT(Patient\_ID,Date\_Treated)

**Physical File and Database Design**

* Designing physical files/databases requires the following information:
  + Normalized relations, including volume estimates
  + Definitions of each attribute
  + Descriptions of where and when data are used: entered, retrieved, deleted, and updated (including frequencies)
  + Expectations or requirements for response time and data integrity
  + Descriptions of the technologies used for implementing the files and database so that the range of required decisions and choices for each is known

**Designing Fields**

* **Field** – smallest unit of named application data recognized by system software
  + An attribute from a relation is now recognized as a field in a database
* **Data type** – coding scheme recognized by system software for representing organizational data

**Choosing Data Types**

* Selecting a data type requires balancing four objectives:
  + Minimize storage space
  + Represent all possible values of the field
  + Improve data integrity of the field
  + Support all data manipulations desired on the field

**Calculated Fields**

* **Calculated field** – field that can be derived from other database fields. Also known as a computed field or a derived field.
* It is common for an attribute to be mathematically related to other data
* The database will either stored or compute the calculated field when requested

**Controlling Data Integrity**

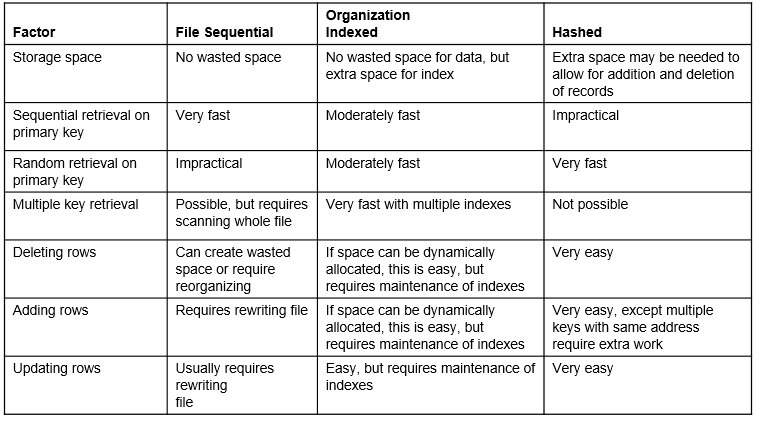
* **Default value** – value a field will assume unless an explicit value is entered for that field
* **Range control –** limits values (numeric or alpha-numeric data) that can be entered into a field
* **Referential integrity** – constraint specifying that the value (or existence) of an attribute in one relation depends on the value (or existence) of the same attribute in another relation
* **Null value** – special field value, distinct from zero, blank, or any other value, that indicates that the value for the field is missing or otherwise unknown

**Designing Physical Tables**

* A relational table is a set of related tables related by foreign keys referencing primary keys
* **Physical table** – named set of rows and columns that specifies the fields in each row of the table
* **Denormalization** – process of splitting or combining normalized relations into physical tables based on affinity of use of rows and fields
* Partitioning is the capability to split a table into separate sections. Partitioning types include:
  + **Range partitioning:** partitions are defined by nonoverlapping ranges of values for a specified attribute
  + **Hash partitioning:** a table row is assigned to a partition by an algorithm and then maps the specified attribute value to a partition
  + **Composite partitioning:** combines range and hash partitioning by first segregating data by ranges on the designated attribute, and then within each of these partitions
* Partitioning helps speed up system performance
* Denormalization can increase change of errors
* Three common situations where denormalization is used are:
  + Two entities with a one-to-one relationship
  + A many-to-many relationship (associative entity) with nonkey attributes
  + Reference data

**Arranging Table Rows**

* The result of denormalization is the definition of one or more physical files
* **Physical file** – named set of table rows stored in a contiguous section of secondary memory
* **File organization** – technique for physically arranging the records of a file
* Objectives for choosing file organization include:
  + Fast data retrieval
  + High throughput for processing transactions
  + Efficient use of storage space
  + Protection from failures or data loss
  + Minimizing need for reorganization
  + Accommodating growth
  + Security from unauthorized use
* **Pointer** – field of data that can be used to locate a related field or row of data
* Three basic families of file organization:
  + **Sequential file organization** – file organization in which rows in a file are stored in sequence according to a primary key value
  + **Indexed file organization** – file organization in which rows are stored either sequentially or nonsequentially, and an index is created that allows software to locate individual rows
  + **Hashed file organization** – file organization in which the address of each row is determined using an algorithm
* **Index** – table used to determine the location of rows in a file that satisfy some condition
* **Secondary key –** represents one or a combination of fields for which more than one row may have the same combination of values
  + Allows an index to point to more than one record
* Indexed file organization
  + Advantage is allowing for both random and sequential processing
  + Disadvantages include:
    - Extra space required to store indexes
    - Extra time necessary to access and maintain indexes
  + Guidelines for choosing indexes include:
    - Specify a unique index for the primary key of each table
    - Specify an index for foreign keys
    - Specify an index for nonkey fields that are referenced in qualification, sorting and grouping commands for the purpose of retrieving data

**Comparative Features of Sequential, Indexed, and Hashed File Organizations** ****

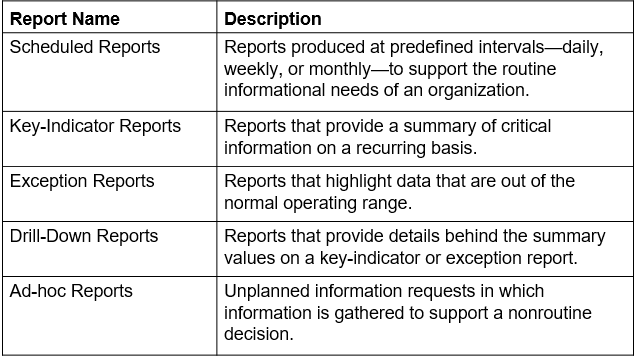
**Designing Controls for Files**

* Two goals of physical table design:
  1. Protection from failure or data loss
  2. Security from unauthorized use
* These goals are achieved primarily by implementing controls on each file
* Two additional types of controls address
  1. Backup
  2. Security
* Techniques for file restoration:
  1. Periodically making a backup copy of a file
  2. Storing a copy of each change to a file in a transaction log or audit trail
  3. Storing a copy of each row before or after it is changed
* Means of building data security into a file:
  1. Coding, or encrypting, the data in the file
  2. Requiring data file users to identify themselves by entering user names and passwords
  3. Prohibiting users from directly manipulating any data in the file by forcing users to work with a copy (real or virtual)

**Chapter 6: Designing Interfaces**

**Designing Interfaces: Forms and Reports**

* **Form** – business document that contains some predefined data and may include some areas where additional data are to be filled in. An instance of a form is typically based on one database record.
  + Usually in a stylized format and not a simple row and column look
* **Report** – business document that contains only predefined data; it is a passive document used solely for reading or viewing. A report typically contains data from many unrelated records or transactions
  + Reports typically contain data from many unrelated records or transactions

**Common Types of Business Reports** ****

**The Process of Designing Interfaces**

* A user-centered activity that follows a **prototyping** approach:
  + Understanding the target audience and their needs
  + Collect initial requirements
  + Structure and refine this information into an initial prototype
  + Users then evaluate the prototype
  + Make any needed refinements until users are satisfied and accept the prototype
* **Paper prototype** – series of mock screens that can be used to test content, look, and feel, as well as the task flow and other usability factors
  + Focus is on the design (content, layout, flow)
* **Wireframe** – simple design to show the placement of information elements on a screen and the space needed for each element
  + Allows users to get a sense and feel of a design
* A coding sheet is an “old” tool for designing forms and reports in text-based format (see figure 10-2)
* A wireframe is an input screen roughed out on a sheet of paper (see figure 10-3A)
* A data input screen designed in Visual Basic allows a variety of font sizes, colors, and highlighting (see figure 10-3B)

**Deliverables and Outcomes**

* Design specifications are the major deliverables and contain three sections:
  1. Narrative overview
     + Characterizes, users, tasks, system, and environmental factors
  2. Sample design
     + Image of the form is more thoroughly tested and assessed
  3. Testing and usability assessment
     + Provides all testing and assessment information
     + Determine usability

**General Guidelines for the Design of Interfaces**

* **Meaningful Titles:**
  + Clear and specific titles describing content and use of form or report
  + Revision date or code to distinguish a form or report from prior versions
  + Current date, which identifies when the form or report was generated
  + Valid date, which identifies on what date (or time) the data in the form or report were accurate
* **Meaningful Information:**
  + Only needed information should be displayed
  + Information should be provided in a manner that is usable without modification
* **Balance the Layout:**
  + Information should be balanced on the screen or page
  + Adequate spacing and margins should be used
  + All data and entry fields should be clearly labeled
* **Design an Easy Navigation System:**
  + Clearly show how to move forward and backward
  + Clearly show where you are (e.g., page 1 of 3)
  + Notify user when on the last page of a multipaged sequence

**Methods of Highlighting**

* Blinking and audible tones
* Color differences
* Intensity differences
* Size differences
* Font differences
* Reverse video
* Boxing
* Underlining
* All capital letters
* Offsetting the position of nonstandard information

**Highlighting Information**

* Several situations when highlighting can be a valuable technique for conveying special information include:
  + Notifying users of errors in data entry or processing
  + Providing warnings to users regarding possible problems such as unusual data values or an unavailable device
  + Drawing attention to keywords, commands, high-priority messages, and unusual data values

**Color Use Guidelines**

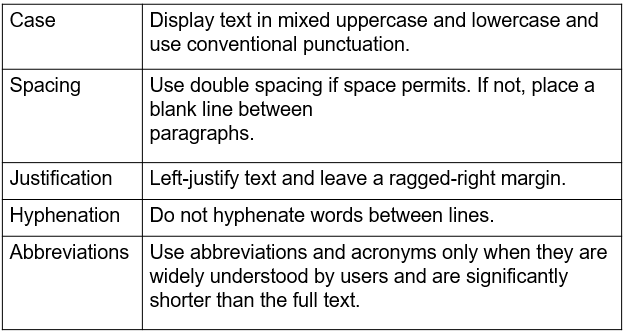
* Don’t use too many colors
* Use color coding to support use tasks
* Allow users to control color coding
* Design for monochrome then add color
* Use color coding consistently
* Avoid color pairings which clash
* Use color change to show status change
* Be aware that color displays are usually lower resolution

**Benefits and Problems from Using Color**

* **Benefits from Using Color:**
  + Soothes or strikes the eyes
  + Accents an uninteresting display
  + Facilitates subtle discriminations in complex displays
  + Emphasizes the logical organization of information
  + Draws attention to warnings
  + Evokes more emotional reactions
* **Problems with Using Color:**
  + Color pairings may wash out or cause problems for some users (e.g., color blindness)
  + Resolution may degrade with different displays
  + Printing or conversion to other media may not easily translate

**Rule 60-30-10**

* Is a simple theory for creating color palettes that are well-balanced and visually interesting:
* Generally something fairly neutral (either literally or psychologically) makes up 60% of the palette
* Another complementary color makes up 30% of the palette
* And then a third color is used as an accent for the remaining 10% of the design

**Guidelines for Displaying Text** ****

**General Guidelines for Displaying Tables and Lists**

* **Use meaningful labels:**
  + All columns and rows should have meaningful labels
  + Labels should be separated from other information by using highlighting
  + Redisplay labels when the data extend beyond a single screen or page
* **Formatting columns, rows, text:**
  + Sort in a meaningful order (e.g., ascending, descending, or alphabetic)
  + Place a blank line between every five rows in long columns
  + Similar information displayed in multiple columns should be sorted vertically (i.e., read from top to bottom, not left to right)
  + Columns should have at least two spaces between them
  + Allow white space on printed reports for user to write notes
* **Formatting columns, rows, text: (cont.)**
  + Use a single typeface, except for emphasis
  + Use same family of typefaces within and across displays and reports
  + Avoid overly fancy fonts
* **Formatting numeric, textual, and alphanumeric date:**
  + **Right-justify numeric data** and align columns by decimal points or other delimiter
  + **Left-justify textual data**. Use short line length, usually 30–40 characters per line (this is what newspapers use, and it is easier to speed-read).
  + Break long sequences of **alphanumeric****data**into small groups of three to four characters each

**Guidelines for Selecting Tables Versus Graphs**

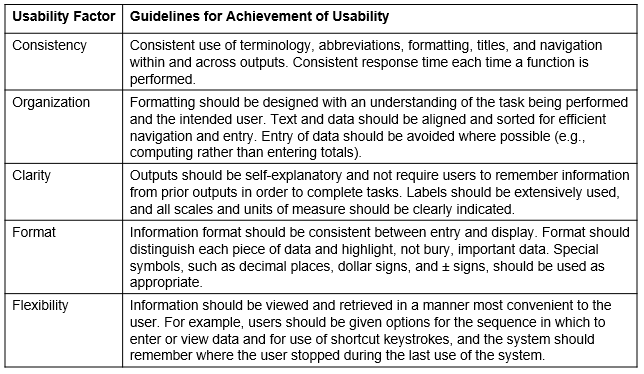
* **Tables**
  + Reading individual data values
* **Graphs**
  + Providing a quick summary of data
  + Detecting trends over time
  + Comparing points and patterns of different variables
  + Forecasting activities
  + Reporting vast amounts of information when relatively simple impressions are to be drawn

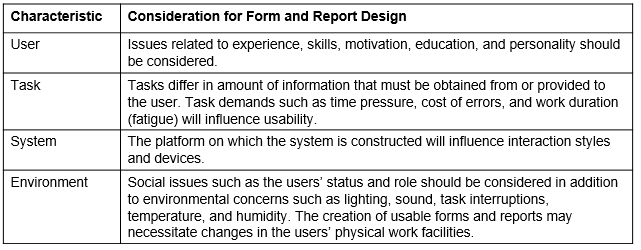
**Paper versus Electronic Reports**

* Laser and ink-jet printers can print reports that look like display screen
  + Same guidelines apply
* High-speed impact printers are faster, but less user-friendly in appearance
  + Good for large batch reports
  + The coding sheet from figure 10-2 may be useful for these types of reports

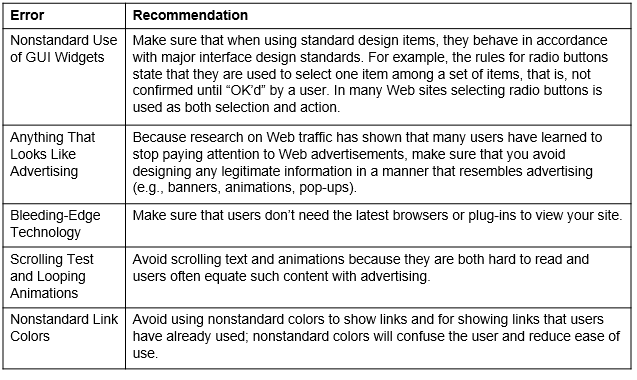
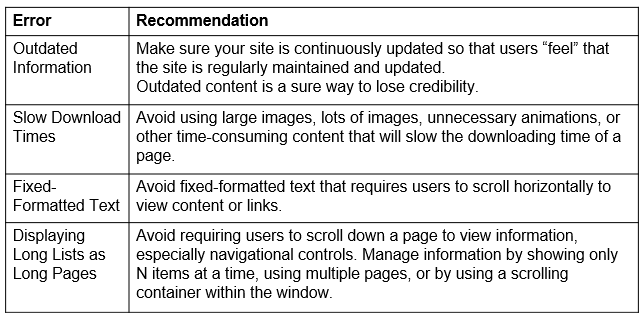
**Assessing Usability**

* **Usability** – overall evaluation of how a system performs in supporting a particular user for a particular task
* Three characteristics of usability:
  1. Speed – Can you complete a task efficiently?
  2. Accuracy – Does the system provide what you expect?
  3. Satisfaction – Do you like using the system?

**General Design Guidelines for Usability of Forms and Reports** ****

**Characteristics for Consideration When Designing Forms and Reports** ****

**Measures of Usability**

* Methods to assess usability:
  + **Learnability** – usability dimension concerned with how difficult it is for the user to perform a task for the first time
  + **Efficiency** – usability dimension concerned with how quickly users can perform tasks once they know how to perform them
  + **Error rate** – usability dimension concerned with how many errors a user might encounter and how easy it is to recover from those errors
  + **Memorability**—How easy is it to remember how to accomplish a task when revisiting the system after some period of time?
  + **Satisfaction and aesthetics**—How enjoyable is the system’s visual appeal and how enjoyable is the system to use?
* **Common Errors When Designing the Layout of Web Pages** **** 

**Designing Forms and Reports at PVF**

* Guidelines established at Pine Valley Furniture (P V F) include:
  + Use lightweight graphics
  + Establish forms and data integrity rules
  + Use stylesheet-based H T M L

**Lightweight Graphics**

* **Lightweight graphics** – small, simple images that allow a Web page to be displayed more quickly
* Forms and data integrity rules:
  + All forms that request information should be clearly labeled and provide adequate room for input
  + Specific fields requiring specific information must provide a clear example
  + Forms must designate which fields are optional, required, and which have a range of values

**Stylesheet-Based H T M L** – Web design approach that separates content from the way in which it is formatted and presented, making ongoing maintenance easier and site-wide consistency much higher

* + Allows content of a Web page to remain separate from the way it is formatted
  + Facilitates site-wide consistency