# Chapter 7: Development Strategies

**Chapter 7 – Development Strategies**: Chapter 7 considers various development strategies for the new system, and plans for the transition to the systems design phase.

### Questions

1. Explain the difference between horizontal and vertical application software. Suggest two examples of each type.

Ans:

* Difference between Horizontal and Vertical Application Software

**Horizontal application software** is designed to serve a wide range of industries and organizations by addressing general business needs. These solutions are not tailored to any specific sector but instead provide functionalities that are widely applicable. Examples include:

* **Microsoft Office**: Used for word processing, spreadsheets, and presentations across all types of organizations.
* **Customer Relationship Management (CRM) systems like Salesforce**: Used for managing customer data and interactions, adaptable to various industries.

**Vertical application software**, on the other hand, is developed to meet the unique requirements of a specific industry or sector. These solutions are specialized and incorporate features, workflows, and compliance needs relevant to that field. Examples include:

* **Medical practice management software**: Designed specifically for healthcare providers, handling patient records, appointments, and billing.
* **Banking software**: Tailored for financial institutions, often including modules for loan processing, compliance, and transaction management.

1. What are three typical reasons why companies develop their own information systems?

Ans:

Three Typical Reasons Why Companies Develop Their Own Information Systems

* **To satisfy unique business requirements**: Companies may have specific needs that cannot be met by commercial, off-the-shelf solutions. Custom systems can be designed to align with unique workflows, policies, and business processes.
* **To meet constraints of existing systems**: Sometimes, new software must integrate with legacy systems or existing infrastructure. Developing in-house ensures compatibility and smooth operation within the current environment.
* **To meet constraints of existing technology**: Organizations may have investments in certain hardware or platforms. Custom development allows them to leverage these assets, avoiding the need for costly replacements or upgrades.

1. What are user applications? Suggest three examples that could boost user productivity.

Ans:

**User applications** are software tools that utilize standard business software to perform specific tasks, often providing user-friendly interfaces for non-technical staff. These applications help users accomplish their work more efficiently.

Three examples that can boost productivity:

* **Screen generators**: Allow users to create custom data entry forms tailored to their workflow, reducing errors and speeding up data collection.
* **Report generators**: Enable users to build and customize reports without programming skills, providing quick insights and supporting better decision-making.
* **Form and Report Wizards**: Tools (such as those in Microsoft Access) that guide users through the process of creating forms or reports, automating much of the setup and reducing the learning curve.

1. What are main steps in the software acquisition process?

Ans:

Main Steps in the Software Acquisition Process

1. **Evaluate Information System Requirements**: Define the key features, performance needs, and constraints. Consider future growth and network or web-related issues.
2. **Identify Potential Vendors or Outsourcing Options**: Research and shortlist vendors or service providers who can deliver the required solution.
3. **Evaluate the Alternatives**: Gather feedback from current users, conduct application testing, and use benchmarking to compare options.
4. **Perform Cost-Benefit Analysis**: Calculate the total cost of ownership, including licensing, maintenance, and support. Weigh these against the expected benefits.
5. **Prepare a Recommendation**: Compare the alternatives, summarize findings, and formally recommend the best solution.
6. **Implement the Solution**: Install, configure, and test the software. Train users and convert existing data to the new system as needed.
7. What is an RFP, and how does it differ from an RFQ?

Ans:

RFP, and How Does It Differ from an RFQ?

* **Request for Proposal (RFP)**: A document that describes a company’s needs and invites vendors to propose solutions. It is used when the company is open to different approaches and wants vendors to suggest how they would address the requirements.
* **Request for Quotation (RFQ)**: A more specific document used when the company knows exactly what it needs and is seeking the best price for a defined product or service. Vendors are asked to provide pricing and delivery terms for the specified items.

**Key difference**: An RFP seeks creative solutions to a problem, while an RFQ seeks the best price for a clearly defined need.

1. What is the purpose of a benchmark test? Suggest at least two examples of benchmarks.

Ans:

A **benchmark test** is used to measure and compare the performance of software solutions under specific conditions. It helps organizations objectively assess how well a system handles expected workloads.

Examples of benchmarks:

* **Transaction Processing Speed**: Measures how many transactions (such as orders or payments) the system can process per minute.
* **Database Query Response Time**: Assesses how quickly the system retrieves information from a database, especially under heavy user load.

1. What is an evaluation model? How would you create a weighted evaluation model?

Ans:

An **evaluation model** is a structured method for comparing software options based on multiple criteria, ensuring an objective and comprehensive assessment.

To create a **weighted evaluation model**:

* Identify all important evaluation criteria (e.g., cost, performance, scalability, vendor support).
* Assign a weight to each criterion based on its importance to the organization.
* Rate each software option on each criterion, typically using a scale (such as 1 to 5).
* Multiply each rating by the criterion’s weight.
* Sum the weighted scores for each option.
* The option with the highest total score is considered the best fit.

This approach ensures that the most critical factors have the greatest influence on the final decision.

1. What decisions might management reach at the end of the systems analysis phase, and what would be the next step in each case?

Ans:

Management Decisions at the End of the Systems Analysis Phase and Next Steps

At the end of systems analysis, management may:

* **Implement an outsourcing alternative**: Proceed to contract negotiations and transition planning with the selected vendor.
* **Develop an in-house system**: Move forward to the systems design phase, allocating resources to internal development teams.
* **Purchase or customize a software package**: Begin procurement and customization, followed by implementation.
* **Perform additional systems analysis**: Return to the analysis phase to gather more information or explore other options.
* **Stop all further work**: Terminate the project if it is deemed unfeasible or unnecessary.

After a decision is made, the project typically transitions to the systems design phase, unless more analysis or a project halt is required.

1. Explain the relationship between logical and physical design.

Ans:

Relationship Between Logical and Physical Design

* **Logical design** outlines what the system must do, focusing on business requirements and system functionality without specifying how it will be implemented.
* **Physical design** details how the system will be built, including technologies, hardware, databases, and network infrastructure.

The two are closely related: logical design sets the requirements, and physical design provides the technical blueprint to fulfill those requirements. Proper alignment ensures the system meets business goals effectively.

### Projects

1. Investigate the ROI of cloud-based software development environments.

**Investigation of ROI for Cloud-Based Software Development Environments**

**1. Key Factors Influencing Cloud ROI**

Cloud-based development environments transform how organizations build, deploy, and manage software. Calculating ROI involves analyzing both quantitative and qualitative benefits against costs. Below are the critical factors:

**1.1 Upfront Costs vs. Subscription Models**

* **On-Premises Costs**: Traditional setups require significant capital expenditure (CapEx) for servers, data centers, and licensing. For example, a company might spend **$2M+** on hardware and **$500k/year** on maintenance.
* **Cloud Costs**: Cloud services like AWS, Azure, or Gitpod use operational expenditure (OpEx) models with pay-as-you-go pricing. For instance, a team of 100 developers might pay **$30–$50/user/month** for cloud IDEs, avoiding upfront infrastructure investments.

**1.2 Scalability and Elasticity**

* **Dynamic Resource Allocation**: Cloud environments auto-scale compute resources during peak loads (e.g., CI/CD pipelines), eliminating over-provisioning. A fintech startup could reduce server costs by **40%** by scaling down non-production environments overnight.
* **Cost per Workload**: Tools like AWS Lambda charge **$0.20 per million requests**, making sporadic workloads (e.g., testing) far cheaper than maintaining idle servers.

**1.3 Reduced Maintenance Overhead**

* **Managed Services**: Cloud providers handle security patches, backups, and compliance (e.g., HIPAA, GDPR). A healthcare SaaS company might save **$200k/year** by outsourcing infrastructure management.
* **IT Staff Efficiency**: By eliminating server maintenance, developers reclaim **5–10 hours/week** (valued at **$75–$150/hour** for senior engineers).

**1.4 Accelerated Development Cycles**

* **Preconfigured Environments**: Tools like GitHub Codespaces or Gitpod provide instant, standardized dev environments. A survey showed developers save **5 hours/week** on setup, reducing time-to-market by **20%**.
* **CI/CD Integration**: Cloud-native pipelines (e.g., AWS CodePipeline) automate testing/deployment, cutting release cycles from weeks to days.

**1.5 Collaboration and Remote Work**

* **Real-Time Collaboration**: Cloud IDEs like CodeSandbox enable simultaneous editing, reducing miscommunication delays. Distributed teams report **15–30% faster** project completion.
* **Centralized Tooling**: Unified platforms (e.g., GitLab) consolidate version control, issue tracking, and monitoring, slashing context-switching by **25%**.

**2. Financial Metrics for ROI Calculation**

**2.1 Payback Period**

* **Formula**:

Payback Period (months)=Initial Migration Costs/Monthly Savings

**Example**: Migrating to AWS costs **$150k** upfront but saves **$50k/month** in hardware and labor. Payback occurs in **3 months**.

**2.2 ROI Formula**

* **Formula**:

ROI=(Net Benefits−Cloud Costs) ×100/Cloud Costs

**Example**: A company saves **$1.2M/year** in TCO with cloud adoption, spending **$300k/year** on subscriptions.

ROI=(1,200,000−300,000)/300,000×100=300%

**2.3 Net Present Value (NPV)**

* **Formula**: Discount future cash flows to present value. For a 3-year cloud migration saving **$1.5M/year** at a 10% discount rate:

NPV=∑(Annual Savings/(1+Discount Rate) Year)

NPV=$3.7*M*

**3. Case Studies and Real-World Data**

**3.1 GitHub Codespaces**

* **Productivity Gains**: A telecom company reduced developer onboarding from **10 days to 2 days** using prebuilt environments, saving **$3.2M/year** in training costs.
* **Cost Reduction**: By retiring legacy tools, the company saved **$7.1M** over three years.

**3.2 Gitpod ROI Calculator**

* **Inputs**: 100 engineers, **5 hours/week** saved on environment setup, **€85k average salary**.
* **Outputs**:
  + **24,000 hours/year** reclaimed (19% capacity boost).
  + **ROI: 913%** (€1.97M annual savings vs. €215k cloud costs).

**3.3 Automotive Manufacturing**

* A carmaker using cloud-based DevOps tools reduced software defects by **30%**, saving **$1.8M/year** in post-release fixes.

**4. Risks and Mitigation Strategies**

**4.1 Hidden Costs**

* **Data Transfer Fees**: AWS charges **$0.09/GB** for cross-region transfers. Mitigate by using CDNs or edge caching.
* **Vendor Lock-In**: Multi-cloud strategies (e.g., Kubernetes) prevent dependency.

**4.2 Security and Compliance**

* **Shared Responsibility Model**: Cloud providers secure infrastructure; users manage access controls.
* **Encryption**: Use AWS KMS or Azure Key Vault for data-at-rest encryption.

**4.3 Overprovisioning**

* **Auto-Scaling Policies**: Set thresholds to terminate unused instances.
* **Spot Instances**: Use AWS Spot Instances for non-critical workloads at **70–90%** discount.

**5. Best Practices for Maximizing ROI**

1. **Adopt FinOps**: Cross-functional teams track cloud spend using tools like CloudHealth or Kubecost.
2. **Leverage Reserved Instances**: Commit to 1–3 year terms for **40–60%** discounts.
3. **Monitor Usage**: Tools like AWS Cost Explorer identify underutilized resources.
4. **Optimize Architecture**: Serverless (e.g., AWS Lambda) and microservices reduce idle resource costs.

**6. Conclusion**

Cloud-based development environments offer **300–900% ROI** through faster deployment, collaboration, and scalability. While risks like hidden costs exist, strategic planning-using FinOps, auto-scaling, and multi-cloud-ensures sustainable returns. Organizations adopting cloud tools report **20–40% productivity gains**, making the shift critical for staying competitive in agile