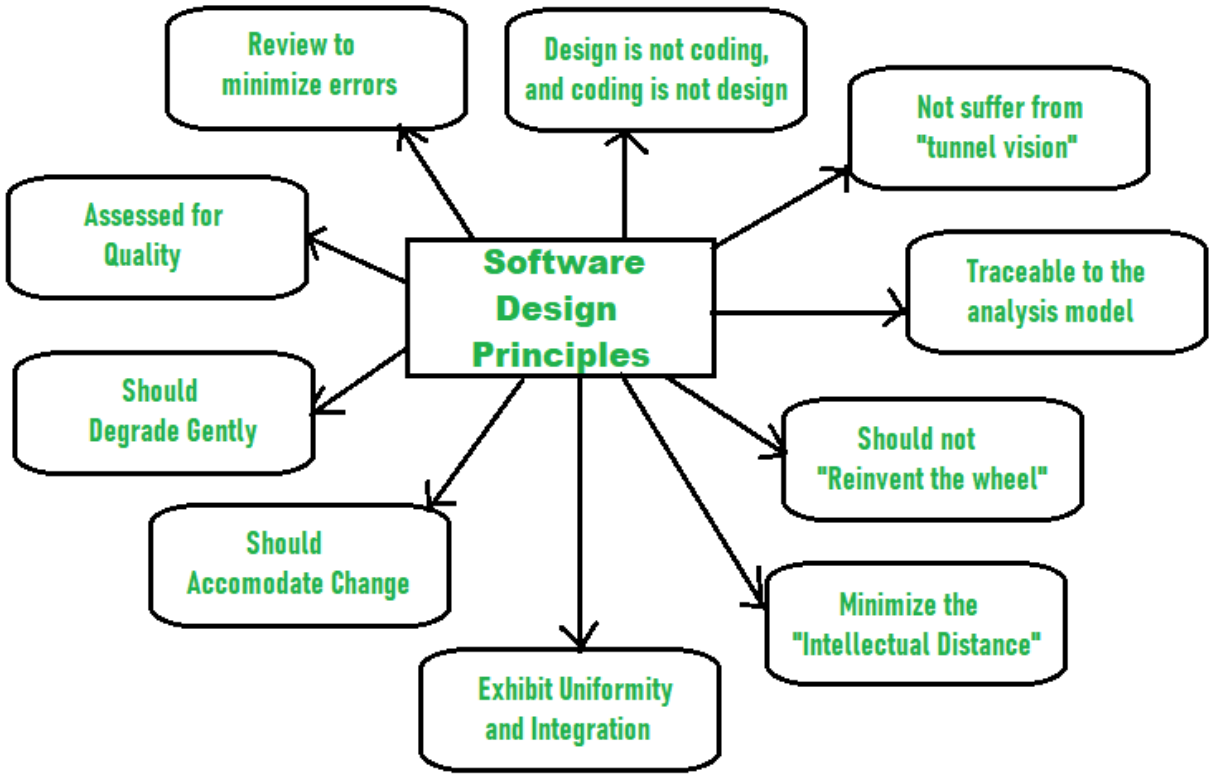
BCA3 (NEW) UNIT -2
SOFTWARE DESIGN PRINCIPLES

Principles of Software Design

- डिज़ाइन का अर्थ है किसी चीज़ का रूप, कार्य और कार्यप्रणाली दर्शाने के लिए उसका चित्र बनाना या योजना बनाना। **सॉफ़्टवेयर डिज़ाइन** भी सॉफ़्टवेयर आवश्यकताओं की योजना बनाने या उन्हें एक ऐसे चरण में बदलने की प्रक्रिया है जो किसी सॉफ़्टवेयर सिस्टम को विकसित करने के लिए आवश्यक हैं। सॉफ़्टवेयर डिज़ाइन के संरचनात्मक घटकों को व्यवस्थित और व्यवस्थित करने के लिए कई सिद्धांतों का उपयोग किया जाता है। जिन सॉफ़्टवेयर डिज़ाइनों में इन सिद्धांतों को लागू किया जाता है, वे शुरुआत से ही सॉफ़्टवेयर की सामग्री और कार्य प्रक्रिया को प्रभावित करते हैं। ये सिद्धांत नीचे दिए गए हैं:



सॉफ़्टवेयर डिज़ाइन के सिद्धांत:

1. **"सुरंग दृष्टि" से ग्रस्त नहीं होना चाहिए** - प्रक्रिया को डिजाइन करते समय, इसे "सुरंग दृष्टि" से ग्रस्त नहीं होना चाहिए, जिसका अर्थ है कि केवल उद्देश्य को पूरा करने या प्राप्त करने पर ही ध्यान केंद्रित नहीं करना चाहिए, बल्कि अन्य प्रभावों पर भी ध्यान केंद्रित करना चाहिए।

2. **विश्लेषण मॉडल के लिए अनुरेखणीय** - डिजाइन प्रक्रिया विश्लेषण मॉडल के लिए अनुरेखणीय होनी चाहिए, जिसका अर्थ है कि इसे उन सभी आवश्यकताओं को पूरा करना चाहिए जो एक उच्च गुणवत्ता वाले उत्पाद को विकसित करने के लिए सॉफ्टवेयर की आवश्यकता होती है।

3. **पहिया का पुनःआविष्कार " नहीं करना चाहिए** - डिज़ाइन प्रक्रिया को पहिया का पुनःआविष्कार नहीं करना चाहिए, अर्थात पहले से मौजूद चीज़ों को बनाने में समय या प्रयास बर्बाद नहीं करना चाहिए। इससे समग्र विकास में वृद्धि होगी।

4. **बौद्धिक दूरी को न्यूनतम करें** - डिजाइन प्रक्रिया को वास्तविक दुनिया की समस्याओं और उस समस्या के लिए सॉफ्टवेयर समाधानों के बीच के अंतर को कम करना चाहिए, जिसका अर्थ है कि इसे बौद्धिक दूरी को न्यूनतम करना चाहिए।

5. **एकरूपता और एकीकरण प्रदर्शित करें** - डिज़ाइन में एकरूपता प्रदर्शित होनी चाहिए, अर्थात यह पूरी प्रक्रिया में बिना किसी परिवर्तन के एक समान होनी चाहिए। एकीकरण का अर्थ है कि यह सॉफ़्टवेयर के सभी भागों, यानी सबसिस्टम, को एक सिस्टम में मिला दे या संयोजित कर दे।

6. **परिवर्तन को समायोजित करना** - सॉफ्टवेयर को इस तरह से डिजाइन किया जाना चाहिए कि वह परिवर्तन को समायोजित कर सके, अर्थात सॉफ्टवेयर को उपयोगकर्ता की आवश्यकता के अनुसार परिवर्तन को समायोजित करना चाहिए।

7. **धीरे से गिरावट** - सॉफ्टवेयर को इस तरह से डिजाइन किया जाना चाहिए कि यह सुचारू रूप से गिरावट हो, जिसका अर्थ है कि निष्पादन के दौरान कोई त्रुटि होने पर भी यह ठीक से काम करना चाहिए।

8. **गुणवत्ता का मूल्यांकन** - डिजाइन का गुणवत्ता के लिए मूल्यांकन किया जाना चाहिए, अर्थात मूल्यांकन के दौरान डिजाइन की गुणवत्ता की जांच की जानी चाहिए और उस पर ध्यान केंद्रित किया जाना चाहिए।

9. **त्रुटियों का पता लगाने के लिए समीक्षा करें** - डिजाइन की समीक्षा की जानी चाहिए, जिसका अर्थ है कि समग्र मूल्यांकन किया जाना चाहिए ताकि यह पता लगाया जा सके कि क्या कोई त्रुटि मौजूद है या उसे कम किया जा सकता है।

10. **डिज़ाइन कोडिंग नहीं है और कोडिंग डिज़ाइन नहीं है** - डिज़ाइन का अर्थ है किसी समस्या को हल करने के लिए प्रोग्राम के तर्क का वर्णन करना और कोडिंग एक प्रकार की भाषा है जिसका उपयोग किसी डिज़ाइन के कार्यान्वयन के लिए किया जाता है।

सिस्टम मॉडलिंग सॉफ्टवेयर इंजीनियरिंग में किसी सिस्टम की संरचना, व्यवहार और अंतःक्रियाओं को समझने के लिए उसके अमूर्त निरूपण तैयार करना, संचार में सहायता करना और डिज़ाइन व कार्यान्वयन को सुगम बनाना शामिल है। ये मॉडल, जो अक्सर ग्राफ़िकल होते हैं, UML जैसे संकेतन का उपयोग करते

हैं। (यूनिफाइड मॉडलिंग लैंग्वेज) का उपयोग सिस्टम के विभिन्न दृष्टिकोणों को दर्शाने के लिए किया जाता है।

सिस्टम मॉडलिंग के मुख्य पहलू:

- **उद्देश्य:**

  सिस्टम मॉडलिंग किसी सिस्टम की कार्यक्षमता को समझने, हितधारकों (ग्राहकों सहित) के साथ संवाद करने और गुणवत्ता का आकलन करने में मदद करती है।

- **मॉडल के प्रकार:**

  विभिन्न मॉडल मौजूद हैं, जिनमें शामिल हैं:

- **संरचनात्मक मॉडल :** वर्ग आरेख की तरह, प्रणाली की स्थिर संरचना का प्रतिनिधित्व करते हैं।
- **व्यवहार मॉडल :** अनुक्रम आरेख और स्थिति आरेख की तरह दर्शाते हैं कि सिस्टम समय के साथ कैसे व्यवहार करता है।
- **संदर्भ मॉडल :** सिस्टम की सीमाओं और उसके पर्यावरण के साथ उसकी अंतःक्रियाओं को दर्शाते हैं।
- **डेटा मॉडल :** सिस्टम के भीतर डेटा संरचनाओं और संबंधों का प्रतिनिधित्व करते हैं।

यूएमएल :

एकीकृत मॉडलिंग भाषा प्रणाली मॉडलिंग के लिए व्यापक रूप से प्रयुक्त मानक है, जो किसी प्रणाली के विभिन्न पहलुओं को दर्शाने के लिए विभिन्न प्रकार के आरेख प्रस्तुत करता है।

मॉडल-संचालित इंजीनियरिंग :

यह दृष्टिकोण मॉडल को प्राथमिक आर्टिफैक्ट के रूप में उपयोग करता है, जिससे कोड या अन्य सिस्टम घटक स्वचालित रूप से उत्पन्न किए जा सकते हैं।

सॉफ्टवेयर विकास जीवन चक्र (एसडीएलसी) मॉडल:

विकास प्रक्रिया को निर्देशित करने के लिए सिस्टम मॉडल का उपयोग विभिन्न SDLC मॉडल, जैसे वाटरफॉल मॉडल , V-मॉडल , पुनरावृत्त मॉडल और सर्पिल मॉडल के साथ किया जाता है ।

फ़ायदे:

सिस्टम मॉडलिंग से संचार में सुधार होता है, त्रुटियां कम होती हैं, बेहतर डिजाइन की सुविधा मिलती है, तथा संभावित समस्याओं की शीघ्र पहचान संभव होती है।

उदाहरण:

उपयोग केस आरेख, वर्ग आरेख, अनुक्रम आरेख, गतिविधि आरेख और स्थिति आरेख सिस्टम मॉडलिंग में प्रयुक्त UML आरेख के सामान्य उदाहरण हैं।

A data flow model, डेटा प्रवाह मॉडल का उपयोग करके डेटा प्रवाह को रेखांकन द्वारा दर्शाता है , जिसमें इनपुट से फ़ाइल संग्रहण और रिपोर्ट निर्माण तक डेटा ले जाने की प्रक्रियाओं की व्याख्या की जाती है।

डेटा प्रवाह आरेख, सिस्टम के भीतर डेटा प्रवाह का एक **दृश्य निरूपण है। यह** पूरे सिस्टम में इनपुट से आउटपुट तक डेटा प्रवाह को समझने और इस प्रक्रिया में इसके रूपांतरण को समझने में मदद करता है । ये मॉडल **सॉफ़्टवेयर इंजीनियरों** , ग्राहकों और उपयोगकर्ताओं को आवश्यकताओं के विश्लेषण और **विनिर्देशन के दौरान प्रभावी ढंग से एक साथ काम करने में सक्षम बनाते हैं** ।

डेटा फ्लो डायग्राम (DFD) क्या है?

डेटा फ्लो डायग्राम (DFD) किसी भी सिस्टम में डेटा प्रवाह का एक **ग्राफिकल निरूपण है। यह आने वाले डेटा प्रवाह, बाहर जाने वाले डेटा प्रवाह और संग्रहीत डेटा को दर्शाने में सक्षम है। DFD आने वाले** और **बाहर जाने वाले दोनों डेटा प्रवाहों को दर्शाता है** और सिस्टम की कार्यक्षमता का एक उच्च-स्तरीय अवलोकन प्रदान करता है। यह सीखने और उपयोग करने के लिए अपेक्षाकृत सरल तकनीक है, जिससे यह तकनीकी और गैर-तकनीकी दोनों हितधारकों के लिए सुलभ है।

डेटा फ़्लो डायग्राम को कई तरीकों से दर्शाया जा सकता है। डेटा फ़्लो डायग्राम (DFD) संरचित-विश्लेषण 1 टूल से संबंधित है। डेटा फ़्लो डायग्राम बहुत लोकप्रिय हैं क्योंकि ये हमें **सॉफ़्टवेयर-सिस्टम प्रक्रियाओं में शामिल प्रमुख चरणों और डेटा को देखने में मदद करते हैं** ।

डीएफडी की मूल संरचना

डेटा फ्लो डायग्राम (DFD) की विशेषताएँ

डेटा फ्लो डायग्राम (DFD) की कुछ विशेषताएं नीचे दी गई हैं:

1. ग्राफ़िकल निरूपण : डेटा प्रवाह आरेख (DFD) सिस्टम के भीतर डेटा प्रवाह को दर्शाने के लिए विभिन्न प्रतीकों और संकेतन का उपयोग करते हैं। ये जटिल सिस्टम को समझने योग्य दृश्य तत्वों में सरल बना देते हैं । इससे **तकनीकी** और **गैर-तकनीकी** दोनों हितधारकों के लिए उनकी व्याख्या करना आसान हो जाता है ।

2. समस्या विश्लेषण: डेटा फ्लो डायग्राम (DFD) किसी सिस्टम को समझने में बहुत उपयोगी होते हैं और विश्लेषण के दौरान इनका प्रभावी ढंग से उपयोग किया जा सकता है। डेटा फ्लो डायग्राम (DFD) काफी सामान्य होते हैं और सॉफ़्टवेयर आवश्यकता विनिर्देशन के लिए समस्या विश्लेषण तक सीमित नहीं होते हैं।

3. अमूर्तन : डीएफडी कार्यान्वयन विवरणों को अमूर्त करते हैं और सिस्टम के भीतर **डेटा प्रवाह और प्रक्रियाओं पर ध्यान केंद्रित करते हैं। वे एक उच्च-स्तरीय अवलोकन प्रदान करते हैं और अनावश्यक तकनीकी जानकारी को छोड़ देते हैं।**

4. पदानुक्रम : डेटा प्रवाह आरेख (DFD) किसी प्रणाली का पदानुक्रम प्रदान करता है। उच्च-स्तरीय आरेख, अर्थात् 0-स्तरीय आरेख, संपूर्ण प्रणाली का अवलोकन प्रदान करता है, जबकि निम्न-स्तरीय आरेख, जैसे 1-स्तरीय DFD और उससे आगे, प्रत्येक प्रक्रिया का विस्तृत डेटा प्रवाह प्रदान करते हैं।

डेटा प्रवाह आरेख (DFD) में स्तर

डीएफडी को विभिन्न स्तरों में वर्गीकृत किया गया है, और प्रत्येक स्तर अलग-अलग स्तर का विवरण प्रदान करता है। स्तरों को **0 से** आगे क्रमांकित किया गया है। स्तर जितना ऊँचा होगा, आरेख उतना ही विस्तृत होगा। डीएफडी के चार स्तर निम्नलिखित हैं:

0-स्तरीय डेटा प्रवाह आरेख (DFD)

लेवल 0 DFD उच्चतम-स्तरीय आरेख है , जो सिस्टम को एक **एकल प्रक्रिया के रूप में दर्शाता है** और बाहरी संस्थाओं के साथ उसकी अंतःक्रियाओं को दर्शाता है। यह सिस्टम में प्रमुख प्रक्रियाओं, डेटा प्रवाह और डेटा भंडार को दर्शाता है, लेकिन इन प्रक्रियाओं की आंतरिक कार्यप्रणाली के बारे में कोई विवरण नहीं देता। इसे **संदर्भ आरेख भी कहा जाता है** , जो सिस्टम के संचालन को सारगर्भित करता है और दिखाता है कि डेटा सिस्टम में कैसे प्रवेश करता है और कैसे बाहर निकलता है।

1-स्तरीय डेटा प्रवाह आरेख (DFD)

लेवल 1 DFD, लेवल 0 डेटा फ़्लो डायग्राम (DFD) में पहचानी गई प्रमुख प्रक्रियाओं को उप-प्रक्रियाओं में विभाजित करके सिस्टम का अधिक विस्तृत दृश्य प्रदान करता है। लेवल 1 डेटा फ़्लो डायग्राम (DFD) पर प्रत्येक उप-प्रक्रिया को एक अलग प्रक्रिया के रूप में दर्शाया गया है। प्रत्येक उप-प्रक्रिया से जुड़े डेटा प्रवाह और डेटा भंडार भी दिखाए गए हैं।

स्तर 1 DFD प्रणाली का अधिक **विस्तृत दृश्य प्रदान करता है, जो प्रमुख कार्यात्मक पहलुओं पर केंद्रित है। स्तर 0** से **संदर्भ आरेख को कई बबल्स/प्रक्रियाओं** में विस्तारित किया गया है ।

2-स्तरीय डेटा प्रवाह आरेख (DFD)

स्तर 2 DFD, स्तर 1 DFD की उप-प्रक्रियाओं को **अतिरिक्त उप-प्रक्रियाओं** में विभाजित करता है , जिससे और भी **विस्तृत दृश्य प्राप्त होता है** । यह स्तर **विशिष्ट आवश्यकताओं** या सिस्टम के उन हिस्सों से निपटने में उपयोगी होता है जिनकी प्रक्रियाओं और अंतःक्रियाओं की गहन जाँच की आवश्यकता होती है।

3-स्तरीय डेटा प्रवाह आरेख (DFD)

3-स्तर डेटा प्रवाह आरेख (DFDs) का **सबसे विस्तृत स्तर है** , जो सिस्टम में प्रक्रियाओं, डेटा प्रवाहों और डेटा भंडारों का विस्तृत दृश्य प्रदान करता है। यह स्तर आमतौर पर जटिल प्रणालियों के लिए उपयोग किया जाता है, जहाँ सिस्टम को समझने के लिए उच्च स्तर के विवरण की आवश्यकता होती है। इसमें प्रत्येक

प्रक्रिया, डेटा प्रवाह और डेटा भंडार का **विस्तृत विवरण शामिल होता है** , और आमतौर पर इसका उपयोग तब किया जाता है जब सिस्टम की व्यापक समझ की आवश्यकता होती है।

डेटा फ्लो डायग्राम (DFD) के प्रकार

डीएफडी को दो मुख्य प्रकारों में वर्गीकृत किया जा सकता है, जिनमें से प्रत्येक सिस्टम डिजाइन के एक अलग परिप्रेक्ष्य पर ध्यान केंद्रित करता है:

डेटा फ्लो डायग्राम (DFD) के प्रकार

1. तार्किक डेटा प्रवाह आरेख (DFD)

लॉजिकल **डेटा फ्लो डायग्राम मुख्य रूप से सिस्टम प्रक्रिया** पर केंद्रित होता है । यह दर्शाता है कि सिस्टम में डेटा कैसे प्रवाहित होता है। लॉजिकल डेटा फ्लो डायग्राम (DFD) मुख्य रूप से तकनीकी कार्यान्वयन विवरणों में गहराई से जाए बिना उच्च-स्तरीय प्रक्रियाओं और डेटा प्रवाह पर केंद्रित होता है।

तार्किक DFD का उपयोग विभिन्न संगठनों में सिस्टम के सुचारू संचालन के लिए किया जाता है। बैंकिंग सॉफ्टवेयर सिस्टम की तरह, इसका उपयोग यह बताने के लिए किया जाता है कि डेटा एक इकाई से दूसरी इकाई में कैसे स्थानांतरित होता है।

ऑनलाइन किराना स्टोर का तार्किक डेटा प्रवाह आरेख 2. भौतिक डेटा प्रवाह आरेख

भौतिक डेटा प्रवाह आरेख दर्शाता है कि सिस्टम में **डेटा प्रवाह वास्तव में कैसे कार्यान्वित होता है । भौतिक डेटा प्रवाह आरेख (DFD) में, हम डेटा संग्रहण, डेटा संचरण, और विशिष्ट तकनीक या सिस्टम घटकों जैसे अतिरिक्त विवरण शामिल करते हैं। भौतिक DFD अधिक विस्तृत होते हैं और सिस्टम के वास्तविक कार्यान्वयन** , जिसमें हार्डवेयर, सॉफ़्टवेयर और डेटा प्रोसेसिंग के भौतिक पहलू शामिल हैं, पर एक नज़दीकी नज़र प्रदान करते हैं।

ऑनलाइन किराना स्टोर का भौतिक डेटा प्रवाह आरेख

डेटा फ्लो डायग्राम (DFD) के घटक

डीएफडी **में चार मुख्य घटक** होते हैं जो सिस्टम के भीतर डेटा के प्रवाह का प्रतिनिधित्व करने के लिए एक साथ काम करते हैं:

डेटा प्रवाह आरेख विधियाँ और प्रतीक

1. प्रक्रिया

किसी सिस्टम में इनपुट से आउटपुट में परिवर्तन प्रक्रिया फलन के कारण होता है। प्रक्रिया के प्रतीक गोल कोनों वाला आयताकार, अंडाकार, आयत या वृत्त होते हैं। प्रक्रिया का सार व्यक्त करने के लिए उसे एक छोटे वाक्य, एक शब्द या वाक्यांश में नाम दिया जाता है।

2. डेटा प्रवाह

डेटा प्रवाह, सिस्टम के विभिन्न भागों के बीच सूचना के स्थानांतरण को दर्शाता है। तीर का चिह्न डेटा प्रवाह का प्रतीक है। स्थानांतरित की जा रही सूचना को निर्धारित करने के लिए प्रवाह को एक संबंधित नाम दिया जाना चाहिए।

डेटा प्रवाह, स्थानांतरित की जा रही सूचना के साथ-साथ सामग्री का भी प्रतिनिधित्व करता है। सामग्री परिवर्तन उन प्रणालियों में मॉडल किए जाते हैं जो केवल सूचनात्मक नहीं होतीं। किसी दिए गए प्रवाह को केवल एक ही प्रकार की सूचना स्थानांतरित करनी चाहिए। प्रवाह की दिशा तीर द्वारा दर्शाई जाती है, जो द्वि-दिशात्मक भी हो सकती है।

3. वेयरहाउस (डेटा स्टोर)

डेटा को बाद में इस्तेमाल के लिए वेयरहाउस में संग्रहित किया जाता है। दो क्षैतिज रेखाएँ स्टोर के प्रतीक का प्रतिनिधित्व करती हैं। वेयरहाउस केवल एक डेटा फ़ाइल तक ही सीमित नहीं है, बल्कि यह कुछ भी हो सकता है, जैसे दस्तावेज़ों वाला एक फ़ोल्डर, एक ऑप्टिकल डिस्क, एक फ़ाइलिंग कैबिनेट।

डेटा वेयरहाउस को उसके कार्यान्वयन से स्वतंत्र रूप से देखा जा सकता है। जब डेटा वेयरहाउस से प्रवाहित होता है, तो उसे डेटा रीडिंग माना जाता है और जब डेटा वेयरहाउस में प्रवाहित होता है, तो उसे डेटा एंट्री या डेटा अपडेटिंग कहा जाता है।

4. टर्मिनेटर (बाह्य इकाई)

टर्मिनेटर एक बाहरी इकाई है जो सिस्टम के बाहर स्थित होती है और सिस्टम के साथ संचार करती है। उदाहरण के लिए, यह बैंक जैसे संगठन, ग्राहकों जैसे लोगों के समूह या एक ही संगठन के विभिन्न विभाग हो सकते हैं, जो मॉडल सिस्टम का हिस्सा नहीं होते और एक बाहरी इकाई होते हैं। मॉडल किए गए सिस्टम भी टर्मिनेटर के साथ संचार करते हैं।

डेटा प्रवाह आरेख (DFD) के नियम

डीएफडी के नियम निम्नलिखित हैं:

1. डेटा प्रवाहित हो सकता है

- टर्मिनेटर या बाहरी इकाई → प्रक्रिया
- प्रक्रिया → टर्मिनेटर या बाहरी इकाई
- प्रक्रिया → डेटा संग्रह
- डेटा संग्रह → प्रक्रिया
- प्रक्रिया → प्रक्रिया

2. डेटा प्रवाहित नहीं हो सकता

- टर्मिनेटर या बाह्य इकाई → टर्मिनेटर या बाह्य इकाई
- टर्मिनेटर या बाहरी इकाई → डेटा स्टोर
- डेटा स्टोर → टर्मिनेटर या बाहरी इकाई
- डेटा स्टोर → डेटा स्टोर

डेटा प्रवाह आरेख के स्तर (DFD) का उदाहरण

डेटा प्रवाह आरेख (DFD) पारदर्शिता बनाए रखने के लिए पदानुक्रम का उपयोग करता है, जिससे बहुस्तरीय डेटा प्रवाह आरेख (DFD) बनाए जा सकते हैं। **डेटा प्रवाह आरेख (DFD) के स्तर** निम्नानुसार हैं :

0-स्तरीय DFD

इसे संदर्भ आरेख भी कहा जाता है। इसे एक अमूर्त दृश्य के रूप में डिज़ाइन किया गया है, जो सिस्टम को एक एकल प्रक्रिया के रूप में दिखाता है और बाहरी

संस्थाओं के साथ उसके संबंध को दर्शाता है। यह पूरे सिस्टम को एक एकल बुलबुले के रूप में दर्शाता है जिसमें इनपुट और आउटपुट डेटा को आने/जाने वाले तीरों द्वारा दर्शाया जाता है।

रेलवे आरक्षण प्रणाली का स्तर 0 1-स्तर DFD

यह स्तर, स्तर 0 DFD में पहचानी गई प्रमुख प्रक्रियाओं को उप-प्रक्रियाओं में विभाजित करके, सिस्टम का अधिक विस्तृत दृश्य प्रदान करता है। प्रत्येक उप-प्रक्रिया को स्तर 1 DFD पर एक अलग प्रक्रिया के रूप में दर्शाया गया है। प्रत्येक उप-प्रक्रिया से जुड़े डेटा प्रवाह और डेटा भंडार भी दिखाए गए हैं।
1-स्तरीय DFD में, संदर्भ आरेख को कई बुलबुलों/प्रक्रियाओं में विभाजित किया जाता है। इस स्तर पर, हम सिस्टम के मुख्य कार्यों पर प्रकाश डालते हैं और 0-स्तरीय DFD की उच्च-स्तरीय प्रक्रिया को उप-प्रक्रियाओं में विभाजित करते हैं ।

रेलवे आरक्षण प्रणाली का स्तर 1 डीएफडी 2-स्तरीय डीएफडी

यह स्तर, स्तर 1 DFD में पहचानी गई उप-प्रक्रियाओं को और अधिक उप-प्रक्रियाओं में विभाजित करके, सिस्टम का और भी विस्तृत दृश्य प्रदान करता है। प्रत्येक उप-प्रक्रिया को स्तर 2 DFD पर एक अलग प्रक्रिया के रूप में दर्शाया गया है। प्रत्येक उप-प्रक्रिया से जुड़े डेटा प्रवाह और डेटा भंडार भी दिखाए गए हैं।

डेटा फ्लो डायग्राम (DFD) के लाभ

- **सिस्टम को समझना** : डीएफडी यह समझने में मदद करते हैं कि सिस्टम के माध्यम से सूचना कैसे प्रवाहित होती है, तथा महत्वपूर्ण कार्यात्मक घटकों का खुलासा करते हैं।
- **ग्राफिकल प्रस्तुतिकरण** : डीएफडी एक **सरल, दृश्य प्रस्तुतिकरण प्रदान करते हैं** जिसे समझना आसान है, जिससे वे तकनीकी और गैर-तकनीकी दोनों हितधारकों के लिए उपयोगी बन जाते हैं।
- **विस्तृत प्रणाली विखंडन : डीएफडी सिस्टम को अलग-अलग प्रक्रियाओं** में विभाजित कर सकता है , जिससे स्पष्ट दस्तावेजीकरण और कार्यप्रवाह की बेहतर समझ प्राप्त होती है।

- **सिस्टम दस्तावेज़ीकरण** : डीएफडी सिस्टम के **दस्तावेज़ीकरण में उपयोगी होते हैं** , यह सुनिश्चित करते हुए कि प्रक्रियाएं वर्तमान और भविष्य की विकास आवश्यकताओं दोनों के लिए अच्छी तरह से परिभाषित हैं।
-

डेटा फ्लो डायग्राम (DFD) के नुकसान

- **समय लेने वाला** : DFDs बनाना, विशेष रूप से जटिल प्रणालियों के लिए, **समय लेने वाला हो सकता है** और इसके लिए व्यापक प्रयास की आवश्यकता हो सकती है।
- **सीमित दायरा** : डीएफडी केवल **डेटा प्रवाह पर ध्यान केंद्रित करते हैं और सिस्टम सुरक्षा** , **प्रदर्शन** या **उपयोगकर्ता इंटरफेस** जैसे अन्य पहलुओं को शामिल नहीं कर सकते हैं ।
- **अद्यतन करने की चुनौतियाँ** : यदि सिस्टम में बार-बार बदलाव किए जाते हैं, तो DFD पुराने हो सकते हैं। उन्हें अद्यतन रखने के लिए काफ़ी रखरखाव की आवश्यकता हो सकती है।
- **विशेषज्ञता की आवश्यकता** : यद्यपि समझना सरल है, लेकिन सटीक DFD बनाने के लिए सिस्टम का विश्लेषण करने और डेटा प्रवाह को परिभाषित करने में **तकनीकी विशेषज्ञता की आवश्यकता होती है।**

एक सिमेंटिक डेटा मॉडल सॉफ्टवेयर इंजीनियरिंग में, यह डेटा का एक वैचारिक निरूपण है जो केवल उसके भौतिक भंडारण या कार्यान्वयन विवरणों पर ध्यान केंद्रित करने के बजाय, उसके अर्थ और एक विशिष्ट डोमेन के भीतर संबंधों पर ज़ोर देता है। इसका उद्देश्य डेटा के वास्तविक-विश्व अर्थविज्ञान को समझना है, जिससे डेवलपर्स और व्यावसायिक उपयोगकर्ताओं, दोनों को एक स्पष्ट और सहज समझ प्राप्त हो सके।

सिमेंटिक डेटा मॉडल के प्रमुख पहलुओं में शामिल हैं:

- **सार्थक प्रतिनिधित्व:**

यह डेटा को वास्तविक दुनिया की संस्थाओं, विशेषताओं और संबंधों के संदर्भ में परिभाषित करता है, तथा यह दर्शाता है कि अनुप्रयोग के डोमेन में जानकारी को कैसे समझा और उपयोग किया जाता है।

- **कार्यान्वयन से अमूर्तन:**

  पारंपरिक डेटा मॉडल के विपरीत, जो अक्सर विशिष्ट डेटाबेस प्रौद्योगिकियों (जैसे, संबंधपरक, पदानुक्रमित) से बंधे होते हैं, एक सिमेंटिक डेटा मॉडल एक उच्च-स्तरीय, वैचारिक दृश्य प्रदान करता है जो अंतर्निहित डेटाबेस प्रबंधन प्रणाली से स्वतंत्र होता है।

- **उन्नत समझ:**

  डेटा के अर्थ और संदर्भ को स्पष्ट रूप से परिभाषित करके, यह हितधारकों के बीच संचार और सहयोग में सुधार करता है, तथा यह सुनिश्चित करता है कि सभी लोग डेटा की एक समान समझ साझा करें।

- **डेटा एकीकरण और अंतरसंचालनीयता को सुगम बनाता है:**

  सिमेंटिक मॉडल, जो प्रायः ऑन्टोलॉजी और औपचारिक ज्ञान प्रतिनिधित्व तकनीकों का उपयोग करके बनाए जाते हैं, विविध स्रोतों से डेटा के आसान एकीकरण को सक्षम करते हैं और विभिन्न प्रणालियों के बीच अंतर-संचालन को समर्थन प्रदान करते हैं।

- **उन्नत विश्लेषण और AI के लिए समर्थन:**
  इन मॉडलों में संग्रहित समृद्ध अर्थगत जानकारी उन्नत विश्लेषण, तर्क और कृत्रिम बुद्धिमत्ता अनुप्रयोगों के लिए आधार प्रदान करती है, जिससे अधिक गहन डेटा विश्लेषण और निर्णय लेने में सहायता मिलती है।

संक्षेप में, एक सिमेंटिक डेटा मॉडल डेटा भंडारण के जटिल तकनीकी विवरण और सूचना की सहज, व्यवसाय-उन्मुख समझ के बीच एक सेतु के रूप में कार्य करता है, जिससे डेटा विभिन्न सॉफ्टवेयर इंजीनियरिंग गतिविधियों के लिए अधिक सुलभ और मूल्यवान बन जाता है।

सॉफ्टवेयर इंजीनियरिंग में ऑब्जेक्ट मॉडल

ऑब्जेक्ट **मॉडल एक मूलभूत अवधारणा है जो ऑब्जेक्ट्स** का उपयोग करके किसी सिस्टम की संरचना और व्यवहार का प्रतिनिधित्व करती है । इसका उपयोग मुख्य रूप से **ऑब्जेक्ट-ओरिएंटेड प्रोग्रामिंग (OOP)** और **ऑब्जेक्ट-ओरिएंटेड डिज़ाइन (OOD) में किया जाता है** ।

परिभाषा

ऑब्जेक्ट **मॉडल** सिद्धांतों और अवधारणाओं का एक संग्रह है जो किसी सिस्टम में ऑब्जेक्ट्स की संरचना और उनके परस्पर क्रिया के तरीके का वर्णन करता है। इसमें शामिल हैं:

- **ऑब्जेक्ट्स** : क्लासों के उदाहरण जो डेटा और व्यवहार को समाहित करते हैं।
- **कक्षाएं** : उन वस्तुओं के लिए ब्लूप्रिंट जो विशेषताओं और विधियों को परिभाषित करते हैं।
- **वंशानुक्रम** : मौजूदा वर्गों से नए वर्ग बनाने की प्रणाली।
- **एनकैप्सुलेशन** : आंतरिक स्थिति को छिपाना तथा सभी अंतःक्रियाओं को ऑब्जेक्ट की विधियों के माध्यम से निष्पादित करना।
- **बहुरूपता (Polymorphism)**: एकाधिक रूपों (जैसे, विधि ओवरलोडिंग और ओवरराइडिंग) के लिए साझा इंटरफ़ेस का उपयोग करने की क्षमता।

ऑब्जेक्ट मॉडल की मुख्य अवधारणाएँ

1. **वस्तुओं**
    - वास्तविक दुनिया की संस्थाओं का प्रतिनिधित्व करें.
    - इसमें **स्थिति (विशेषताएँ)** और **व्यवहार (विधियाँ) शामिल हैं** ।

- **संदेश पासिंग** (विधि कॉल) के माध्यम से संवाद करें ।

2. **कक्षाओं**
   - ऑब्जेक्ट्स के गुण और व्यवहार को परिभाषित करें।
   - ऑब्जेक्ट बनाने के लिए टेम्पलेट के रूप में कार्य करें।
3. **कैप्सूलीकरण**
   - डेटा और विधियों को एक इकाई (वर्ग) के भीतर डेटा पर संचालित करता है।
   - एक्सेस विनिर्देशकों (सार्वजनिक, निजी, संरक्षित) के माध्यम से **एक्सेस नियंत्रण** प्रदान करता है ।
4. **विरासत**
   - कक्षाओं को अन्य कक्षाओं से सुविधाएँ प्राप्त करने की अनुमति देता है।
   - **कोड पुनः उपयोग** और **पदानुक्रमित वर्गीकरण को** बढ़ावा देता है ।
5. **बहुरूपता**
   - एक इंटरफ़ेस को सामान्य वर्ग की क्रियाओं के लिए उपयोग करने की अनुमति देता है।
   - **संकलन-समय बहुरूपता** (विधि ओवरलोडिंग) और **रन-समय बहुरूपता** (विधि ओवरराइडिंग) शामिल है ।
6. **मतिहीनता**
   - जटिल कार्यान्वयन विवरण छुपाता है और केवल आवश्यक विशेषताएं दिखाता है।

ऑब्जेक्ट मॉडल बनाम अन्य मॉडल

पहलू	ऑब्जेक्ट मॉडल	प्रक्रियात्मक मॉडल
संरचना	ऑब्जेक्ट और क्लासेस	प्रक्रियाएँ और कार्य
केंद्र	वास्तविक दुनिया की संस्थाएँ चरण और तर्क	

पहलू	ऑब्जेक्ट मॉडल	प्रक्रियात्मक मॉडल
डेटा + व्यवहार	वस्तुओं में संयुक्त	अलग
पुनर्प्रयोग	उच्च (विरासत के कारण)	मध्यम

सॉफ्टवेयर विकास में उपयोग

- **यूएमएल (यूनिफाइड मॉडलिंग लैंग्वेज)** सिस्टम डिजाइन करने के लिए ऑब्जेक्ट मॉडल का उपयोग करता है।
- इसमें मदद करता है:
  - **सिस्टम विश्लेषण और डिजाइन**
  - **कोड जनरेशन**
  - **हितधारकों के बीच संचार**

UML में उदाहरण (ऑब्जेक्ट आरेख)

वर्ग: कार

+ बनाना: स्ट्रिंग
+ मॉडल: स्ट्रिंग
+ गति: int

+ त्वरित करें( ): शून्य
+ ब्रेक( ): शून्य

- विशेषताओं और विधियों (व्यवहार) के साथ एक वर्ग कार का प्रतिनिधित्व करता है।
- ऑब्जेक्ट्स (जैसे myCar) इस वर्ग के उदाहरण हैं।

ऑब्जेक्ट मॉडल के लाभ

- **मॉड्यूलरिटी** और **रखरखाव को** बढ़ावा देता है
- कोड की **पुन: प्रयोज्यता** सक्षम करता है
- **मापनीयता** और **लचीलेपन को** बढ़ाता है
- सॉफ्टवेयर डिज़ाइन को **वास्तविक दुनिया की सोच के साथ संरेखित करता है**

सॉफ्टवेयर इंजीनियरिंग में वंशानुक्रम मॉडल

इनहेरिटेंस **मॉडल, ऑब्जेक्ट-ओरिएंटेड पैराडाइम** का एक प्रमुख हिस्सा है । यह एक **नए वर्ग** (जिसे **उपवर्ग** या **व्युत्पन्न वर्ग कहा जाता है**) को **मौजूदा वर्ग** (जिसे **सुपरक्लास** या **बेस क्लास कहा जाता है**) के **गुणों और व्यवहारों** (विशेषताओं और विधियों) को प्राप्त करने की अनुमति देता है।

विरासत की परिभाषा

वंशानुक्रम एक ऐसी प्रणाली है जिसके द्वारा **एक वर्ग दूसरे वर्ग की विशेषताओं को वंशानुक्रम में प्राप्त कर सकता है** , जिससे **कोड पुनःप्रयोग** , **विस्तारशीलता** और वस्तुओं के **श्रेणीबद्ध वर्गीकरण को बढ़ावा मिलता है।**

उत्तराधिकार का उद्देश्य

- **कोड पुनः उपयोग** : मौजूदा कोड का पुनः उपयोग करके दोहराव से बचा जाता है।
- **विस्तारशीलता** : व्यवहार में आसानी से संशोधन या विस्तार संभव बनाता है।

- **पदानुक्रम मॉडलिंग** : वास्तविक दुनिया के "है-एक" संबंधों का प्रतिनिधित्व करता है (उदाहरण के लिए, एक कुत्ता *एक* जानवर *है*) /

विरासत के प्रकार

प्रकार	विवरण	उदाहरण
एकल उत्तराधिकार	एक उपवर्ग एक सुपरक्लास से विरासत में प्राप्त होता है ।	कार, वाहन से विरासत में मिली है
बहुस्तरीय विरासत	एक वर्ग एक व्युत्पन्न वर्ग (बहु-स्तरीय श्रृंखला) से विरासत में प्राप्त होता है।	इलेक्ट्रिककार → कार → वाहन
पदानुक्रमित विरासत	एकाधिक वर्ग एक ही आधार वर्ग से विरासत में प्राप्त होते हैं।	कार , बाइक वाहन से विरासत में मिली
बहुविध वंशानुक्रम	एक वर्ग एक से अधिक आधार वर्गों से विरासत में प्राप्त होता है।	फ्लाइंगकार को कार और एयरक्राफ्ट से विरासत में प्राप्त किया गया है (C++ में समर्थित, जावा में नहीं)
संकर वंशानुक्रम	दो या अधिक प्रकार की विरासत का संयोजन।	बहुस्तरीय और बहुस्तरीय का मिश्रण

वाक्यविन्यास उदाहरण (जावा-जैसे स्यूडोकोड में)

```
वर्ग वाहन {
    शून्य प्रारंभ() {
    System.out.println ( "वाहन शुरू हुआ");
}
}

क्लास कार वाहन का विस्तार करता है {
    शून्य ड्राइव() {
```

```
    System.out.println ( "कार चल रही है");
}
}
```

- कार को वाहन से start( ) विधि विरासत में मिलती है ।

यूएमएल प्रतिनिधित्व

यूएमएल आरेखों में, वंशानुक्रम को एक ठोस रेखा और **आधार वर्ग की ओर इशारा करते हुए एक खोखले त्रिभुज के साथ दिखाया जाता है** ।

वाहन
▲
|
कार

उत्तराधिकार मॉडल के लाभ

- **कोड पुन: उपयोग को बढ़ावा देता है**
- **रखरखाव में सुधार**
- **बहुरूपता का समर्थन करता है**
- **वास्तविक दुनिया के रिश्तों को दर्शाता है**

विरासत की कमियाँ

- कक्षाओं के बीच **तंग युग्मन** हो सकता है ।
- अत्यधिक उपयोग से प्रणालियाँ **जटिल और कठोर हो सकती हैं** ।
- **बहुविध वंशानुक्रम** अस्पष्टता पैदा कर सकता है (**डायमंड समस्या की तरह**)।

सर्वोत्तम प्रथाएं

- जब स्पष्ट "है-एक" संबंध हो तो वंशानुक्रम का उपयोग करें।
- जब उचित हो तो **उत्तराधिकार की अपेक्षा संरचना को** प्राथमिकता दें ।
- गहरी विरासत श्रृंखलाओं (2-3 स्तरों से अधिक) से बचें।

वास्तविक दुनिया का उदाहरण

सुपरक्लास : कर्मचारी
- नाम
- वेतन

उपवर्ग: प्रबंधक (कर्मचारी को विरासत में मिलता है)
- विभाग
- बोनस

यहां, प्रबंधक *एक कर्मचारी है* , इसलिए प्रबंधक को कर्मचारी से बुनियादी गुण और व्यवहार विरासत में मिलते हैं।

सॉफ्टवेयर इंजीनियरिंग में ऑब्जेक्ट एग्रीगेशन

ऑब्जेक्ट एग्रीगेशन, ऑब्जेक्ट-ओरिएंटेड डिज़ाइन में एक अवधारणा है जो ऑब्जेक्ट्स के बीच एक **"हैस-ए" संबंध को दर्शाती है। यह एक प्रकार का जुड़ाव है** जो एक **संपूर्ण-भाग** संबंध का मॉडल बनाता है जहाँ भाग संपूर्ण से **स्वतंत्र रूप से मौजूद रह सकता है ।**

परिभाषा

एकत्रीकरण एक विशेष प्रकार का संघ है जहां एक वर्ग (**संपूर्ण**) **में दूसरे वर्ग** (**भाग**) का संदर्भ होता है , लेकिन दोनों **स्वतंत्र रूप से मौजूद हो सकते हैं** ।

इसे ऐसे समझें:

"एक वस्तु की दूसरी वस्तु होती है।"

उदाहरण

- एक **विभाग** *के पास* कई **कर्मचारी हैं** .
- एक **कर्मचारी** किसी विभाग से **स्वतंत्र रूप से** कार्य कर सकता है ।

```
वर्ग कर्मचारी {
स्ट्रिंग नाम;
}

कक्षा विभाग {
सूची<कर्मचारी> कर्मचारी ; / / एकत्रीकरण
}
```

इस मामले में:

- विभाग कर्मचारी को एकत्रित करता है ।
- यदि कोई विभाग हटा दिया जाता है, तो कर्मचारी ऑब्जेक्ट अभी भी मौजूद रह सकते हैं।

यूएमएल प्रतिनिधित्व

यूएमएल (यूनिफाइड मॉडलिंग लैंग्वेज) में, **एकत्रीकरण को** संबंध के पूरे (कंटेनर) छोर पर एक **खोखले हीरे के साथ दिखाया जाता है।**

विभाग ◇ —— कर्मचारी

एकत्रीकरण बनाम संयोजन बनाम संघटन

अवधारणा	UML में प्रतीक	जीवनकाल निर्भरता	उदाहरण

अवधारणा	UML में प्रतीक	जीवनकाल निर्भरता	उदाहरण
संगठन	सरल रेखा	कोई स्वामित्व निहित नहीं	शिक्षक विद्यार्थी
एकत्रीकरण	खोखला हीरा	भाग स्वतंत्र रूप से रह सकता है	विभाग ◇ —— कर्मचारी
संघटन	ठोस हीरा	भाग का जीवन पूरे पर निर्भर करता है	घर ◆ —— कमरा

एकत्रीकरण का उपयोग कब करें

एकत्रीकरण का उपयोग तब करें जब:

- एक **has -a** संबंध का मॉडल बनाना चाहते हैं ।
- अंश सम्पूर्ण से अधिक **समय तक जीवित रह सकता है ।**
- **साझा संबंध** दिखाना चाहते हैं (उदाहरण के लिए, एक ही कर्मचारी को साझा करने वाले कई विभाग)।

वास्तविक दुनिया के उदाहरण

- एक **पुस्तकालय पुस्तकों को** एकत्रित करता है ।
- एक **कंपनी कर्मचारियों को** एकत्रित करती है .
- एक **टीम खिलाड़ियों को** एकत्रित करती है .

एकत्रीकरण के लाभ

- **मॉड्यूलरिटी** और **कोड पुनः उपयोग को** बढ़ावा देता है ।
- वर्गों के बीच **ढीले युग्मन का** समर्थन करता है ।
- **वास्तविक दुनिया के रिश्तों को** अधिक स्वाभाविक रूप से प्रतिबिंबित करता है ।

सॉफ्टवेयर इंजीनियरिंग में सेवा उपयोग मॉडल

सेवा **उपयोग मॉडल सेवा-उन्मुख वास्तुकला (एसओए)** और **घटक-आधारित सॉफ्टवेयर इंजीनियरिंग** में एक अवधारणा है , जो इस बात पर ध्यान केंद्रित करती है कि **ग्राहक (उपयोगकर्ता या सिस्टम)** सॉफ्टवेयर सिस्टम द्वारा प्रदान की गई **सेवाओं** के साथ कैसे बातचीत करते हैं ।

यह उन **तरीकों , संदर्भों** और **अनुक्रमों को परिभाषित करता है** जिनमें **सेवाओं की खोज की जाती है, उन्हें बाध्य किया जाता है, आह्वान किया जाता है और उनका उपयोग किया जाता है** ।

परिभाषा

सेवा **उपयोग मॉडल** वर्णन करता है:

- उपयोगकर्ताओं या अन्य प्रणालियों द्वारा सेवाओं का उपभोग **किस प्रकार किया जाता है।**
- सेवाएँ **कब** और **किस क्रम में लागू की जाती हैं।**
- **निर्भरताएँ** .
- **इंटरफेस और अनुबंध** जो उपयोग को नियंत्रित करते हैं।

यह किसी प्रणाली में सेवा अंतःक्रियाओं को डिजाइन करने, दस्तावेजीकरण करने और कार्यान्वित करने के लिए एक **ब्लूप्रिंट के रूप में कार्य करता है।**

सेवा उपयोग मॉडल के प्रमुख तत्व

तत्व	विवरण
सेवा उपभोक्ता	वह सिस्टम या उपयोगकर्ता जो सेवा का उपयोग करता है.

तत्व	विवरण
सेवा प्रदाता	सेवा प्रदान करने वाली प्रणाली या घटक।
सेवा इंटरफ़ेस	यह परिभाषित करता है कि उपभोक्ता सेवा के साथ किस प्रकार इंटरैक्ट करता है (इनपुट, आउटपुट, प्रोटोकॉल)।
सेवा अनुबंध	सेवा अपेक्षाओं को परिभाषित करने वाला औपचारिक समझौता (पूर्व शर्तें, पश्चात शर्तें , डेटा प्रारूप)।
सेवा बंधन	रनटाइम या डिज़ाइन-टाइम पर सेवा से कनेक्ट करने की प्रक्रिया।
सेवा आह्वान	सेवा कार्यक्षमता को निष्पादित करने के लिए वास्तविक कॉल.
सेवा जीवनचक्र	सेवा उपयोग के चरणों का वर्णन करता है: खोज → बाइंडिंग → निष्पादन → समाप्ति।

सेवा उपयोग के चरण

1. **सेवा खोज**
   - सेवा का पता लगाएं (मैन्युअल रूप से या UDDI जैसी रजिस्ट्री के माध्यम से)।
2. **सेवा बंधन**
   - gRPC) का उपयोग करके सेवा से कनेक्ट करें ।
3. **सेवा आह्वान**
   - सेवा पर परिचालन निष्पादित करें (उदाहरण के लिए, HTTP कॉल या विधि आह्वान के माध्यम से).
4. **प्रतिक्रिया प्रबंधन**
   - सेवा द्वारा लौटाए गए परिणाम या आउटपुट को संसाधित करें।
5. **सेवा रिलीज़**

- वैकल्पिक रूप से उपयोग के बाद सेवा को रिलीज़ या अपंजीकृत करें (मुख्यतः प्रबंधित वातावरण में)।

UML अनुक्रम आरेख (सरलीकृत)

ग्राहक सेवा रजिस्ट्री सेवा
--- findService --->	
<-- सेवासूचना ----	
--- बाइंडसर्विस ---------- >
--- इनवोकऑपरेशन ---------- >
<-- returnResult ----------

उदाहरण (वेब सेवा उपयोग)

एक **ई-कॉमर्स ऐप की कल्पना करें** जो **भुगतान सेवा का उपयोग करता है** ।

क्लाइंट ऐप → भुगतान सेवा (REST API)
- पोस्ट/भुगतान
- पेलोड: कार्ड विवरण, राशि
- प्रतिक्रिया: लेनदेन आईडी

यहां, सेवा उपयोग मॉडल दस्तावेज करेगा:

- क्लाइंट API से कैसे जुड़ता है.
- यह कौन सा डेटा भेजता और प्राप्त करता है।
- परिचालनों का अपेक्षित अनुक्रम (जैसे, प्रमाणित करें → भुगतान करें → पुष्टि करें)।

सेवा उपयोग मॉडलिंग के लाभ

- **सेवा निर्भरता** और जिम्मेदारियों को स्पष्ट करता है।
- कई ग्राहकों के लिए सेवाओं के **पुनः उपयोग की सुविधा प्रदान करता है।**

- वितरित प्रणालियों में **अंतरसंचालनीयता में सुधार करता है।**
- **ढीले युग्मन** और **स्केलेबिलिटी का समर्थन करता है** ।
- **स्वचालित परीक्षण** , दस्तावेज़ीकरण और ऑर्केस्ट्रेशन सक्षम करता है।

उपयोग के मामले

- **माइक्रोसर्विसेज आर्किटेक्चर**
- **क्लाउड-नेटिव अनुप्रयोग**
- **व्यवसाय प्रक्रिया मॉडलिंग**
- **API डिज़ाइन और दस्तावेज़ीकरण**

उपकरण और मानक

- **यूएमएल (उपयोग केस और अनुक्रम आरेख)**
- **WSDL (वेब सेवा विवरण भाषा)**
- **OpenAPI / स्वैगर (REST API के लिए)**
- **बीपीएमएन (बिजनेस प्रोसेस मॉडल और नोटेशन)**

सॉफ्टवेयर इंजीनियरिंग में डेटा शब्दकोश

डेटा **डिक्शनरी** एक **केंद्रीकृत संग्रह है** जिसमें सॉफ़्टवेयर सिस्टम में उपयोग किए जाने वाले **डेटा तत्वों की परिभाषाएँ और विवरण होते हैं। यह टीमों को डेटा को लगातार समझने, व्यवस्थित करने और प्रबंधित करने में मदद करके सिस्टम विश्लेषण और डिज़ाइन में महत्वपूर्ण भूमिका निभाता है** ।

परिभाषा

डेटा **शब्दकोश** एक **सूची या तालिका है** जिसमें निम्नलिखित शामिल हैं:

- सभी डेटा तत्वों के **नाम** ,
- उनका **वर्णन** ,
- **डेटा के प्रकार** ,
- **रिश्ते** ,
- **मान्य मान (डोमेन)** ,
- **उपयोग नियम** .

यह सुनिश्चित करता है कि परियोजना पर काम करने वाले प्रत्येक व्यक्ति को **डेटा की साझा समझ हो** ।

डेटा शब्दकोश का उद्देश्य

उद्देश्य	विवरण
मानकीकरण	पूरे सिस्टम में नामकरण और स्वरूपण में एकरूपता सुनिश्चित करता है।
स्पष्टता	डेवलपर्स, विश्लेषकों और हितधारकों के लिए स्पष्ट परिभाषाएँ प्रदान करता है।
दस्तावेज़ीकरण	सिस्टम डेटा के औपचारिक दस्तावेज़ीकरण के रूप में कार्य करता है।
विकास सहायता	डेटाबेस डिजाइन, कोडिंग और परीक्षण में सहायता करता है।
सत्यापन	डेटा की शुद्धता और अखंडता की जांच करने में मदद करता है।

डेटा डिक्शनरी के घटक

मैदान	विवरण	उदाहरण

मैदान	विवरण	उदाहरण
डेटा तत्व का नाम	डेटा आइटम का नाम	ग्राहक आईडी
विवरण	डेटा तत्व क्या दर्शाता है	ग्राहक के लिए विशिष्ट पहचानकर्ता
डेटा प्रकार	डेटा का प्रकार	पूर्णांक, स्ट्रिंग, दिनांक
लंबाई	अधिकतम आकार	10 (अक्षर)
प्रारूप	इनपुट/आउटपुट प्रारूप	वर्ष-माह-दिन
डिफ़ॉल्ट मान	यदि प्रदान नहीं किया गया है तो पूर्वनिर्धारित मान	व्यर्थ
शून्य की अनुमति दें?	क्या फ़ील्ड खाली रह सकता है	हां नहीं
मान्य मान (डोमेन)	अनुमत मान	"एम", "एफ", "अन्य"
स्रोत	डेटा कहाँ से आता है?	उपयोगकर्ता इनपुट / बाहरी API
प्रतिबंध	कोई नियम	अद्वितीय होना चाहिए

डेटा डिक्शनरी प्रविष्टि का उदाहरण

गुण	कीमत
नाम	आर्डर की तारीख
विवरण	आदेश दिए जाने की तिथि
प्रकार	तारीख

गुण	कीमत
प्रारूप	वर्ष-माह-दिन
आवश्यक	हाँ
डिफ़ॉल्ट मान	वर्तमान सिस्टम तिथि
स्रोत	उपयोगकर्ता इनपुट
प्रतिबंध	भविष्य में नहीं हो सकता

डेटा शब्दकोश में डेटा के प्रकार

1. **आधार डेटा** - एप्लिकेशन में उपयोग किया गया वास्तविक डेटा (उदाहरण के लिए, customer_name , order_id)।
2. **व्युत्पन्न डेटा** - आधार डेटा से गणना किया गया डेटा (उदाहरण के लिए, total_price)।
3. **मेटाडेटा** - अन्य डेटा के बारे में जानकारी (जैसे, प्रकार, लंबाई, स्रोत)।

इसका उपयोग कहाँ किया जाता है

- **डेटाबेस डिज़ाइन** (ERDs, स्कीमा)
- **सॉफ़्टवेयर आवश्यकता दस्तावेज़ (एसआरएस)**
- **सिस्टम विश्लेषण और मॉडलिंग**
- **एपीआई डिज़ाइन**
- **डेटा वेयरहाउसिंग और BI सिस्टम**

लाभ

डेटा की स्थिरता को बढ़ावा देता है
 टीमों के बीच
संचार में सुधार करता है अतिरेक और **त्रुटियों को कम करता है**
परिवर्तनों के दौरान
प्रभाव विश्लेषण में सहायता करता है **डेटा शासन** और **अनुपालन का समर्थन करता है**

डेटा शब्दकोश बनाने के लिए उपकरण

- **स्प्रेडशीट (एक्सेल, गूगल शीट्स)**
- **ईआर मॉडलिंग उपकरण** (उदाहरण के लिए, ईआर/स्टूडियो, ल्यूसिडचार्ट)
- **डेटाबेस सिस्टम** (SQL सर्वर, Oracle)
- **डेटा कैटलॉग उपकरण** (अपाचे एटलस, कोलिब्रा , एलेशन)
- **दस्तावेज़ीकरण प्लेटफ़ॉर्म** (कॉन्फ्लुएंस, नोशन)

सॉफ्टवेयर इंजीनियरिंग में डिजाइन प्रक्रिया

डिजाइन **प्रक्रिया सॉफ्टवेयर आवश्यकताओं को** एक विस्तृत **ब्लूप्रिंट में बदलने** की एक संरचित विधि है जो सॉफ्टवेयर सिस्टम के **विकास का मार्गदर्शन करती है।**

यह **सॉफ्टवेयर विकास जीवन चक्र (एसडीएलसी) का एक महत्वपूर्ण चरण है** जो इस बात पर केंद्रित है कि सिस्टम कैसे बनाया जाएगा - जिसमें **आर्किटेक्चर, घटक, इंटरफेस, डेटा और एल्गोरिदम शामिल हैं** ।

परिभाषा

डिजाइन **प्रक्रिया** वह **नियोजन चरण है** जहां डेवलपर्स सिस्टम की तकनीकी संरचना बनाते हैं जो विश्लेषण चरण के दौरान परिभाषित कार्यात्मक और गैर-कार्यात्मक आवश्यकताओं को पूरा करती है।

डिजाइन प्रक्रिया के उद्देश्य

- **समग्र सिस्टम आर्किटेक्चर को** परिभाषित करें
- सिस्टम को **प्रबंधनीय मॉड्यूल में विभाजित करें**
- **मॉड्यूलरिटी** , **पुन: प्रयोज्यता** और **रखरखाव** सुनिश्चित करें
- **कुशल कार्यान्वयन** के लिए तैयारी करें

डिज़ाइन प्रक्रिया के चरण

1. आवश्यकता विश्लेषण (इनपुट)

- विश्लेषण चरण से आउटपुट (एसआरएस - सॉफ्टवेयर आवश्यकता विनिर्देश)

2. सिस्टम डिज़ाइन (उच्च-स्तरीय डिज़ाइन / वास्तुशिल्प डिज़ाइन)

- **सिस्टम आर्किटेक्चर** को परिभाषित करता है : प्रमुख मॉड्यूल, डेटा प्रवाह और संचार।
- उत्तर **"कौन से घटकों की आवश्यकता है?"**
- कलाकृतियाँ:
    - ब्लॉक आरेख
    - आर्किटेक्चर पैटर्न (जैसे, स्तरित, क्लाइंट-सर्वर, माइक्रोसर्विसेस)
    - प्रौद्योगिकी विकल्प

3. विस्तृत डिज़ाइन (निम्न-स्तरीय डिज़ाइन)

- **प्रत्येक मॉड्यूल** या घटक पर ध्यान केंद्रित करता है ।
- **आंतरिक तर्क** , **इंटरफेस** , **डेटा संरचनाएं** और **एल्गोरिदम** को परिभाषित करता है ।
- उत्तर **"प्रत्येक घटक कैसे काम करेगा?"**
- कलाकृतियाँ:
  - वर्ग आरेख
  - अनुक्रम आरेख
  - छद्म कोड
  - डेटा स्कीमा

4. इंटरफ़ेस डिज़ाइन

- निर्दिष्ट करता है कि मॉड्यूल/घटक किस प्रकार परस्पर क्रिया करेंगे।
- इसमें UI डिज़ाइन और API शामिल हैं।

5. डेटा डिज़ाइन

- यह निर्दिष्ट करता है कि डेटा को कैसे संग्रहीत, एक्सेस और प्रबंधित किया जाएगा।
- डेटाबेस, फ़ाइल प्रारूप, डेटा मॉडल (ईआरडी, स्कीमा) का डिज़ाइन।

डिज़ाइन सिद्धांत

गुणवत्ता सुनिश्चित करने के लिए निम्नलिखित सिद्धांत लागू किए जाते हैं:

सिद्धांत	विवरण
प्रतिरूपकता	सिस्टम को अलग-अलग मॉड्यूल में विभाजित करें

सिद्धांत	विवरण
मतिहीनता	आंतरिक विवरण छिपाएँ; कार्यक्षमता पर ध्यान केंद्रित करें
कैप्सूलीकरण	डेटा और उस पर काम करने वाली विधियों को बाँधें
चिंताओ का विभाजन	प्रत्येक मॉड्यूल एक चिंता का विषय संभालता है
पुनर्प्रयोग	अन्य भागों/परियोजनाओं में पुन: उपयोग के लिए डिज़ाइन
सामंजस्य और युग्मन	उच्च संसक्ति (एक मॉड्यूल के भीतर) और निम्न युग्मन (मॉड्यूल के बीच)

डिज़ाइन उपकरण और तकनीकें

- **यूएमएल आरेख** : वर्ग, अनुक्रम, उपयोग मामला, गतिविधि
- **डेटा प्रवाह आरेख (DFD)**
- **इकाई-संबंध आरेख (ERD)**
- **फ़्लोचार्ट**
- **छद्म कोड**
- **वायरफ्रेम / UI मॉकअप**

उच्च-स्तरीय डिज़ाइन का उदाहरण

ऑनलाइन लाइब्रेरी प्रणाली:

- घटक: उपयोगकर्ता इंटरफ़ेस, पुस्तक सूचीपत्र, उधार सेवा, डेटाबेस
- आर्किटेक्चर: क्लाइंट-सर्वर मॉडल
- डेटा प्रवाह: UI → सेवा परत → डेटाबेस

डिज़ाइन प्रक्रिया के परिणाम

- **सॉफ्टवेयर डिज़ाइन दस्तावेज़ (SDD)**
- वास्तुकला आरेख
- डेटा स्कीमा
- इंटरफ़ेस विनिर्देश
- वर्ग और वस्तु परिभाषाएँ

एक अच्छी तरह से परिभाषित डिज़ाइन प्रक्रिया के लाभ

विकास समय और लागत कम करता है
 सिस्टम **स्केलेबिलिटी** और **रखरखाव में सुधार करता है**
 परीक्षण और डिबगिंग को आसान बनाता है
 बेहतर **टीम सहयोग का समर्थन करता है**
 SDLC में समस्याओं का शीघ्र पता लगाने में मदद करता है

सामान्य डिज़ाइन मॉडल

- **वाटरफॉल मॉडल** : डिजाइन विश्लेषण के बाद किया जाता है और कोडिंग से पहले तय किया जाता है।
- **चंचल डिजाइन** : डिजाइन पुनरावृत्तीय विकास के साथ विकसित होता है।
- **मॉडल-संचालित डिज़ाइन (एमडीडी)** : कोडिंग से पहले मॉडल बनाने पर ध्यान केंद्रित करें।

सॉफ्टवेयर इंजीनियरिंग में डिजाइन विधियाँ

डिज़ाइन विधियाँ उन संरचित दृष्टिकोणों और तकनीकों को संदर्भित करती हैं जिनका उपयोग सॉफ़्टवेयर **आवश्यकताओं को** एक **सुसंरचित सिस्टम आर्किटेक्चर**

और विस्तृत **घटक डिज़ाइन** में बदलने के लिए किया जाता है । ये विधियाँ यह सुनिश्चित करने में मदद करती हैं कि सॉफ़्टवेयर **कुशल** , **मॉड्यूलर** , **रखरखाव योग्य हो** और **उपयोगकर्ता की आवश्यकताओं को पूरा करे** ।

डिज़ाइन विधियाँ क्यों महत्वपूर्ण हैं

- डिज़ाइन के लिए एक **व्यवस्थित दृष्टिकोण प्रदान करें**
- **पुन: प्रयोज्यता** , **मापनीयता** और **स्पष्टता** को बढ़ावा देना
- **जटिलता को** प्रबंधित करने में सहायता करें
- बेहतर **सहयोग** और **दस्तावेज़ीकरण की अनुमति दें**

सामान्य सॉफ्टवेयर डिज़ाइन विधियाँ

डिजाइन विधि विवरण

1. संरचित डिज़ाइन (एसडी)

- **शीर्ष-से-नीचे** दृष्टिकोण और **कार्यात्मक अपघटन** पर आधारित
- **स्पष्ट रूप से परिभाषित इनपुट/आउटपुट** के साथ सिस्टम को **मॉड्यूल में तोड़ने पर ध्यान केंद्रित करता है**
- अक्सर उपयोग:
  - **डेटा प्रवाह आरेख (DFDs)**
  - **संरचना चार्ट**

इसके लिए सर्वश्रेष्ठ: प्रक्रियात्मक या फ़ंक्शन-आधारित सिस्टम
 जटिल, ऑब्जेक्ट-ओरिएंटेड सिस्टम के लिए कम उपयुक्त

---

2. ऑब्जेक्ट-ओरिएंटेड डिज़ाइन (OOD)

- **परस्पर क्रिया करने वाली वस्तुओं** के संग्रह के रूप में मॉडलिंग करने पर आधारित
- इस तरह की अवधारणाओं का उपयोग करता है:
  - **कक्षाएं** , **ऑब्जेक्ट्स**
  - **वंशानुक्रम** , **बहुरूपता**
  - **एनकैप्सुलेशन** , **अमूर्तन**
- **यूएमएल आरेखों** का उपयोग करता है : वर्ग, अनुक्रम, उपयोग-मामला, आदि।

इसके लिए सर्वश्रेष्ठ: बड़े, जटिल, वास्तविक दुनिया के सिस्टम
 पुन: उपयोग और रखरखाव को बढ़ावा देता है अधिक अग्रिम योजना की आवश्यकता होती है

---

3. टॉप-डाउन डिज़ाइन

- **समग्र प्रणाली** से शुरू करें और इसे **छोटे उप-प्रणालियों/मॉड्यूलों** में विभाजित करें
- अमूर्त से ठोस तक **प्रगतिशील परिशोधन** पर ध्यान केंद्रित करता है

अमूर्तता और सरलीकरण में
मदद करता है शुरुआत में निम्न-स्तरीय अनुकूलन को अनदेखा कर सकता है

---

4. नीचे से ऊपर की ओर डिज़ाइन

- **अलग-अलग मॉड्यूल/घटकों** से शुरू करें और पूर्ण सिस्टम तक आगे बढ़ें
- अक्सर **लाइब्रेरी विकास में** या जब पुन: प्रयोज्य घटक पहले से मौजूद हों, तब इसका उपयोग किया जाता है

पूर्व-निर्मित मॉड्यूल को एकीकृत करने के लिए उपयोगी
 शुरुआत में बड़े-चित्र प्रणाली प्रवाह को याद किया जा सकता है

---

5. मॉड्यूलर डिज़ाइन

- सिस्टम को **स्वतंत्र, विनिमेय मॉड्यूल में विभाजित करने पर जोर देता है**
- प्रत्येक मॉड्यूल **एक ही कार्य करता है** और इसे अलग से विकसित/परीक्षण किया जा सकता है

रखरखाव और पठनीयता बढ़ाता है
 समानांतर विकास का समर्थन करता है अच्छी तरह से परिभाषित इंटरफेस की आवश्यकता होती है

---

6. डेटा-केंद्रित डिज़ाइन

- कार्यों के बजाय **डेटा** पर केंद्रित करें
- पर ध्यान देता है:
  - डेटा संरचनाएं
  - डेटाबेस
  - डेटा एक्सेस परतें
- अक्सर **इकाई-संबंध आरेख (ERD) के साथ जोड़ा जाता है**

डेटा-गहन प्रणालियों के लिए बढ़िया
 नियंत्रण प्रवाह और व्यवहार पर कम ध्यान

---

7. पहलू-उन्मुख डिज़ाइन (एओडी)

- **क्रॉस-कटिंग चिंताओं को** कोर लॉजिक से अलग करता है

- **पहलू-उन्मुख प्रोग्रामिंग (AOP)** के साथ उपयोग किया जाता है

स्वच्छ कोड को बढ़ावा देता है
 विशेष उपकरण/फ्रेमवर्क की आवश्यकता होती है (उदाहरण के लिए, AspectJ)

---

8. घटक-आधारित डिज़ाइन

- **स्वतंत्र घटकों** से निर्मित है जो सुपरिभाषित इंटरफेस प्रदान करते हैं
- **पुन: उपयोग** , **प्लग-एंड-प्ले** आर्किटेक्चर को बढ़ावा देता है
- **माइक्रोसर्विसेज** और **वितरित प्रणालियों** में उपयोग किया जाता है

मापनीयता और परिनियोजन लचीलेपन का समर्थन करता है
 इंटरफेस के सावधानीपूर्वक प्रबंधन की आवश्यकता होती है

सही डिज़ाइन विधि का चयन

विधि का चुनाव इस पर निर्भर करता है:

- परियोजना का **आकार और जटिलता**
- **विकास दृष्टिकोण** (एजाइल, वाटरफॉल, आदि)
- **टीम विशेषज्ञता**
- प्रणाली की प्रकृति (वास्तविक समय, डेटा-संचालित, उद्यम-स्तर)

डिज़ाइन विधियों में प्रयुक्त उपकरण

- **यूएमएल उपकरण** : ल्यूसिडचार्ट , ड्रा.आईओ, स्टारयूएमएल
- **ईआर मॉडलिंग उपकरण** : MySQL वर्कबेंच, ईआर/स्टूडियो
- **IDE समर्थन** : विजुअल स्टूडियो, इंटेलीज , एक्लिप्स

- **आर्किटेक्चर टूल्स** : आर्किमेट , एंटरप्राइज़ आर्किटेक्ट

सारांश तालिका

तरीका	केंद्र	के लिए उपयुक्त
संरचित डिज़ाइन	फ़ंक्शन/डेटा	प्रक्रियात्मक प्रणालियाँ
ओओडी	वस्तुओं	वास्तविक दुनिया, जटिल प्रणालियाँ
उपर से नीचे	सिस्टम-प्रथम	बड़ी नई परियोजनाएँ
नीचे से ऊपर	मॉड्यूल-प्रथम	घटक एकीकरण
मॉड्यूलर	स्वतंत्रता	बड़े और रखरखाव योग्य ऐप्स
डेटा-केंद्रित	डेटा संरचनाएं	डेटा-संचालित अनुप्रयोग
पहलू-ओरिएंटेड	पार काटने	लॉगिंग, सुरक्षा, आदि.
घटक-आधारित	पुनर्प्रयोग	वितरित प्रणालियाँ, माइक्रोसर्विसेज

सॉफ्टवेयर इंजीनियरिंग में डिजाइन विवरण

डिज़ाइन **विवरण** एक विस्तृत दस्तावेज़ या प्रस्तुतिकरण होता है जो बताता है कि **निर्दिष्ट आवश्यकताओं को पूरा करने के लिए एक सॉफ़्टवेयर सिस्टम कैसे** बनाया जाएगा । यह **आवश्यकता विनिर्देश** और **कार्यान्वयन (कोडिंग)** चरण के बीच की खाई को पाटता है ।

परिभाषा

डिज़ाइन **विवरण** सॉफ्टवेयर सिस्टम का एक व्यापक प्रतिनिधित्व है **आर्किटेक्चर** , **घटक** , **डेटा संरचनाएं** , **इंटरफेस** और **तर्क** , सॉफ्टवेयर डेवलपमेंट लाइफ साइकिल (एसडीएलसी) के **डिजाइन चरण** के दौरान बनाए गए ।

डिज़ाइन विवरण का उद्देश्य

- आवश्यकताओं को तकनीकी योजना में परिवर्तित करें
- कार्यान्वयन में डेवलपर्स का मार्गदर्शन करें
- परीक्षण और सत्यापन का समर्थन करें
- भविष्य के रखरखाव और उन्नयन में सहायता
- टीमों में एकरूपता और समझ सुनिश्चित करें

डिज़ाइन विवरण के घटक

सॉफ्टवेयर डिज़ाइन विवरण (SDD) में निम्नलिखित शामिल होता है:

---

1 परिचय

- प्रणाली का अवलोकन
- डिजाइन का उद्देश्य
- दायरा
- संदर्भ (एसआरएस, मानक, आदि)

---

2. वास्तुशिल्प डिजाइन

- उच्च-स्तरीय प्रणाली संरचना
- प्रमुख मॉड्यूल और उनके संबंध
- उपयोग किए गए डिज़ाइन पैटर्न या वास्तुशिल्प शैलियाँ (जैसे, MVC, क्लाइंट-सर्वर, माइक्रोसर्विसेस)
- **आरेख** : वास्तुकला आरेख, घटक आरेख

---

3. विस्तृत डिज़ाइन

- प्रत्येक मॉड्यूल/घटक का विवरण
- आंतरिक तर्क और एल्गोरिदम
- घटक के भीतर डेटा प्रवाह
- नियंत्रण प्रवाह और निर्णय संरचनाएं
- **स्यूडोकोड या फ़्लोचार्ट**

---

4. इंटरफ़ेस डिज़ाइन

- आंतरिक इंटरफेस (मॉड्यूल के बीच)
- बाहरी इंटरफेस (UI, API, तृतीय-पक्ष सेवाएँ)
- डेटा प्रारूप, प्रोटोकॉल और विधियाँ
- UI/UX डिज़ाइन (वायरफ्रेम, स्क्रीन विवरण)

---

5. डेटा डिज़ाइन

- डेटाबेस स्कीमा
- डेटा संरचनाएं और प्रकार
- ईआर आरेख या वर्ग आरेख

- फ़ाइल प्रारूप, मेटाडेटा, सत्यापन नियम

---

6. क्लास और ऑब्जेक्ट डिज़ाइन (यदि OOP का उपयोग कर रहे हैं)

- कक्षा की ज़िम्मेदारियाँ और रिश्ते
- वंशानुक्रम, बहुरूपता
- UML वर्ग आरेख

---

7. अनुक्रम और गतिविधि आरेख

- संचालन का चरण-दर-चरण प्रवाह
- समय के साथ घटकों की परस्पर क्रिया

---

8. डिज़ाइन बाधाएँ

- प्रदर्शन, मापनीयता और मेमोरी सीमाएँ
- प्लेटफ़ॉर्म, प्रोग्रामिंग भाषा, या टूल प्रतिबंध

---

9. मान्यताएँ और निर्भरताएँ

- बाहरी पुस्तकालय या प्रणालियाँ
- तृतीय-पक्ष API या उपकरण

---

10. शब्दावली और संदर्भ

- शब्दों की परिभाषाएँ
- स्रोत दस्तावेज़ों के लिंक

उदाहरण: डिज़ाइन विवरण (स्निपेट)

मॉड्यूल: उपयोगकर्ता प्रमाणीकरण

- **उद्देश्य** : लॉगिन के दौरान उपयोगकर्ताओं को सत्यापित करना
- **इनपुट** : उपयोगकर्ता नाम , पासवर्ड
- **आउटपुट** : प्राधिकरण टोकन, त्रुटि संदेश
- **डेटा संरचनाएं** :
  - उपयोगकर्ता : { आईडी, ईमेल, हैश_पासवर्ड }
- **तर्क** :
  - जांचें कि उपयोगकर्ता नाम मौजूद है या नहीं
  - हैश इनपुट पासवर्ड और संग्रहीत हैश के साथ तुलना करें
- **इंटरफ़ेस** :
  - API समापन बिंदु: POST /login
  - प्रतिक्रिया: {टोकन: <JWT>}

डिज़ाइन विवरण के लिए उपयोग किए जाने वाले उपकरण

- **यूएमएल उपकरण** : ल्यूसिडचार्ट , स्टारयूएमएल , ड्रा.आईओ
- **दस्तावेज़ीकरण उपकरण** : माइक्रोसॉफ्ट वर्ड, गूगल डॉक्स, मार्कडाउन
- **आर्किटेक्चर टूल्स** : आर्किमेट , एंटरप्राइज़ आर्किटेक्ट
- **डेटाबेस डिज़ाइन** : MySQL वर्कबेंच, DBDesigner

एक अच्छे डिज़ाइन विवरण के लाभ

- गलत संचार को कम करता है
- कार्यान्वयन को आसान और अधिक सटीक बनाता है

- परीक्षण और डिबगिंग को सरल बनाता है
- मापनीयता और रखरखाव का समर्थन करता है

सॉफ्टवेयर इंजीनियरिंग में डिजाइन की गुणवत्ता

डिजाइन गुणवत्ता से तात्पर्य है कि एक सॉफ्टवेयर डिजाइन वांछित **कार्यात्मक** और **गैर-कार्यात्मक** आवश्यकताओं को कितनी अच्छी तरह पूरा करता है, और यह सिस्टम के विकास, रखरखाव और भविष्य के विकास को कितनी प्रभावी ढंग से समर्थन देता है।

डिज़ाइन की गुणवत्ता क्यों मायने रखती है

- उच्च **गुणवत्ता वाला डिज़ाइन** ऐसे सॉफ्टवेयर की ओर ले जाता है जो:
  - **भरोसेमंद**
  - **अनुरक्षणीय**
  - **कुशल**
  - **स्केलेबल**
  - **समझने में आसान**
- खराब डिज़ाइन के कारण हो सकते हैं:
  - **विकास समय में** वृद्धि
  - उच्च **रखरखाव लागत**
  - **परीक्षण** और **उन्नयन** में कठिनाइयाँ
  - सिस्टम **विफलताएँ**

अच्छे सॉफ्टवेयर डिज़ाइन की प्रमुख विशेषताएँ

गुण	विवरण

गुण	विवरण
यथार्थता	डिज़ाइन सभी आवश्यकताओं को सटीक रूप से कार्यान्वित करता है।
क्षमता	यह डिज़ाइन प्रदर्शन, गति और संसाधन उपयोग को अनुकूलित करता है।
रख-रखाव	डिज़ाइन को संशोधित करना, ठीक करना या बढ़ाना आसान है।
विश्वसनीयता	डिजाइन यह सुनिश्चित करता है कि सिस्टम अपेक्षित परिस्थितियों में लगातार कार्य करता रहे।
प्रयोज्य	डिज़ाइन उपयोग में आसानी (विशेषकर UI/UX पहलुओं) का समर्थन करता है।
पोर्टेबिलिटी	यह डिज़ाइन विभिन्न प्लेटफार्मों पर तैनाती का समर्थन करता है।
पुनर्प्रयोग	घटकों का अन्य प्रणालियों या मॉड्यूल में पुनः उपयोग किया जा सकता है।
FLEXIBILITY	यह डिज़ाइन न्यूनतम प्रयास से बदलती आवश्यकताओं के अनुरूप ढल सकता है।
प्रतिरूपकता	यह प्रणाली स्पष्ट जिम्मेदारियों के साथ अलग-अलग घटकों में विभाजित है।
परीक्षण योग्यता	यह डिज़ाइन घटकों के प्रभावी परीक्षण को सुगम बनाता है।

डिज़ाइन की गुणवत्ता सुनिश्चित करने के सिद्धांत

सिद्धांत	स्पष्टीकरण
चिंताओ का विभाजन	प्रत्येक मॉड्यूल एक विशिष्ट चिंता या कार्यक्षमता को संबोधित करता है।
कम युग्मन	मॉड्यूल/घटकों की एक दूसरे पर न्यूनतम निर्भरता होती है।
उच्च सामंजस्य	एक मॉड्यूल के भीतर तत्व निकट रूप से संबंधित और केंद्रित होते हैं।

सिद्धांत	स्पष्टीकरण
जानकारी छिपाना	आंतरिक मॉड्यूल विवरण अन्य मॉड्यूल से छिपाए गए हैं।
एकल जिम्मेदारी	प्रत्येक वर्ग/मॉड्यूल में परिवर्तन का एक कारण होता है।
खुला-बंद सिद्धांत	मॉड्यूल विस्तार के लिए खुले हैं लेकिन संशोधन के लिए बंद हैं।
अपने आप को दोहराएँ नहीं (DRY)	कोड या तर्क की नकल करने से बचें।

डिज़ाइन की गुणवत्ता मापना

- **कोड मेट्रिक्स** (बाद में विकास में, लेकिन डिज़ाइन इन पर प्रभाव डालता है)
  - चक्रीय जटिलता
  - प्रति मॉड्यूल कोड की पंक्तियाँ
  - इंटरफेस की संख्या
- **समीक्षाएं और निरीक्षण**
  - सहकर्मी डिज़ाइन समीक्षाएँ
  - डिज़ाइन वॉकथ्रू
- **प्रोटोटाइपिंग और परीक्षण**
  - डिज़ाइन निर्णयों का शीघ्र सत्यापन

डिज़ाइन गुणवत्ता समस्याओं के उदाहरण

- **स्पेगेटी कोड:** खराब मॉड्यूलरिटी और उलझी हुई निर्भरताएं।
- **गॉड ऑब्जेक्ट:** एक मॉड्यूल/क्लास जो बहुत अधिक कार्य करता है।
- **कठोरता:** व्यापक प्रभाव के बिना डिजाइन को बदलना कठिन है।
- **नाज़ुकता:** परिवर्तन अप्रत्याशित दुष्प्रभाव पैदा करते हैं।
- **गतिहीनता:** घटकों का पुनः उपयोग या स्थानांतरण नहीं किया जा सकता।

उच्च-गुणवत्ता वाले डिज़ाइन के लाभ

- तेज़ विकास चक्र
- आसान डिबगिंग और परीक्षण
- कम रखरखाव लागत
- बेहतर मापनीयता और प्रदर्शन
- टीम के सदस्यों के बीच बेहतर सहयोग

वास्तुकला डिजाइन - सॉफ्टवेयर इंजीनियरिंग

- सॉफ़्टवेयर के डिज़ाइन को दर्शाने के लिए सॉफ़्टवेयर को एक आर्किटेक्चरल डिज़ाइन की आवश्यकता होती है। IEEE आर्किटेक्चरल डिज़ाइन को "हार्डवेयर और सॉफ़्टवेयर घटकों और उनके इंटरफेस के एक संग्रह को परिभाषित करने की प्रक्रिया" के रूप में परिभाषित करता है ताकि कंप्यूटर सिस्टम के विकास के लिए रूपरेखा तैयार की जा सके। कंप्यूटर-आधारित सिस्टम के लिए बनाया गया सॉफ़्टवेयर इन कई आर्किटेक्चरल शैलियों में से एक को प्रदर्शित कर सकता है।

सिस्टम श्रेणी में शामिल हैं

- घटकों का एक समूह (जैसे : एक डेटाबेस, कम्प्यूटेशनल मॉड्यूल) जो सिस्टम द्वारा आवश्यक कार्य निष्पादित करेगा।
- कनेक्टर्स का यह सेट घटकों के बीच समन्वय, संचार और सहयोग में मदद करेगा।
- वे शर्तें जो परिभाषित करती हैं कि सिस्टम बनाने के लिए घटकों को कैसे एकीकृत किया जा सकता है।
- सिमेंटिक मॉडल जो डिजाइनर को सिस्टम के समग्र गुणों को समझने में मदद करते हैं।

वास्तुकला शैलियों का उपयोग प्रणाली के सभी घटकों के लिए एक संरचना स्थापित करने के लिए किया जाता है।

स्थापत्य शैलियों का वर्गीकरण

1] डेटा केंद्रित आर्किटेक्चर:

- डेटा स्टोर इस आर्किटेक्चर के केंद्र में स्थित होगा और अन्य घटकों द्वारा बार-बार इसका उपयोग किया जाएगा, जो स्टोर के भीतर मौजूद डेटा को अद्यतन, जोड़ते, हटाते या संशोधित करते हैं।
- यह चित्र एक विशिष्ट डेटा-केंद्रित शैली को दर्शाता है। क्लाइंट सॉफ़्टवेयर एक केंद्रीय रिपॉज़िटरी तक पहुँच प्राप्त करता है। इस दृष्टिकोण के विभिन्न रूपों का उपयोग रिपॉज़िटरी को एक ब्लैकबोर्ड में बदलने के लिए किया जाता है, जब क्लाइंट से संबंधित डेटा या क्लाइंट के लिए रुचिकर डेटा क्लाइंट सॉफ़्टवेयर को सूचनाएँ बदलता है।
- यह डेटा-केंद्रित आर्किटेक्चर एकीकरण को बढ़ावा देगा । इसका मतलब है कि मौजूदा घटकों को बदला जा सकता है और अन्य क्लाइंट की अनुमति या चिंता के बिना आर्किटेक्चर में नए क्लाइंट घटक जोड़े जा सकते हैं।
- ब्लैकबोर्ड तंत्र का उपयोग करके ग्राहकों के बीच डेटा पारित किया जा सकता है।

डेटा केंद्रित वास्तुकला के लाभ:

- डेटा का भंडार ग्राहकों से स्वतंत्र है
- ग्राहक एक दूसरे से स्वतंत्र होकर काम करते हैं
- अतिरिक्त ग्राहकों को जोड़ना सरल हो सकता है।
- संशोधन बहुत आसान हो सकता है

डेटा केंद्रित वास्तुकला

2] डेटा प्रवाह आर्किटेक्चर:

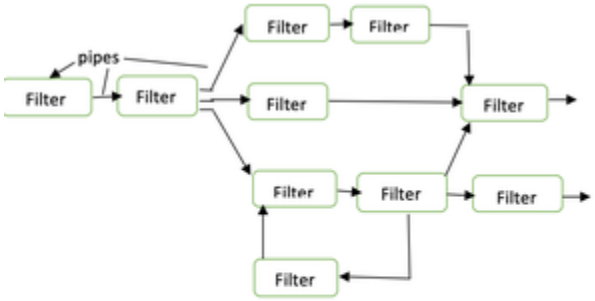
- इस प्रकार की वास्तुकला का उपयोग तब किया जाता है जब इनपुट डेटा को कम्प्यूटेशनल मैनिपुलेटिव घटकों की एक श्रृंखला के माध्यम से आउटपुट डेटा में परिवर्तित किया जाता है।
- यह चित्र पाइप-एवं-फिल्टर वास्तुकला को दर्शाता है, क्योंकि इसमें पाइप और फिल्टर दोनों का उपयोग किया जाता है तथा इसमें फिल्टर नामक घटकों का एक समूह होता है, जो रेखाओं द्वारा जुड़ा होता है।
- पाइप्स का उपयोग एक घटक से दूसरे घटक तक डेटा संचारित करने के लिए किया जाता है।
- प्रत्येक फ़िल्टर स्वतंत्र रूप से कार्य करेगा और एक निश्चित रूप का डेटा इनपुट लेने और निर्दिष्ट रूप के अगले फ़िल्टर को डेटा आउटपुट देने के लिए डिज़ाइन किया गया है। इन फ़िल्टरों को पड़ोसी फ़िल्टरों की कार्यप्रणाली के बारे में किसी ज्ञान की आवश्यकता नहीं होती है।
- यदि डेटा प्रवाह रूपांतरणों की एक ही पंक्ति में विघटित हो जाता है, तो इसे बैच अनुक्रमिक कहा जाता है। यह संरचना डेटा के बैच को स्वीकार करती है और फिर उसे रूपांतरित करने के लिए अनुक्रमिक घटकों की एक श्रृंखला लागू करती है।

डेटा फ्लो आर्किटेक्चर के लाभ:

- यह रखरखाव, पुनःप्रयोजन और संशोधन को प्रोत्साहित करता है।
- इस डिज़ाइन के साथ, समवर्ती निष्पादन समर्थित है।

डेटा फ्लो आर्किटेक्चर का नुकसान:

- यह अक्सर बैच अनुक्रमिक प्रणाली में बदल जाता है
- डेटा प्रवाह आर्किटेक्चर ऐसे अनुप्रयोगों की अनुमति नहीं देता है जिनमें अधिक उपयोगकर्ता सहभागिता की आवश्यकता होती है।
- दो अलग-अलग लेकिन संबंधित धाराओं में समन्वय स्थापित करना आसान नहीं है

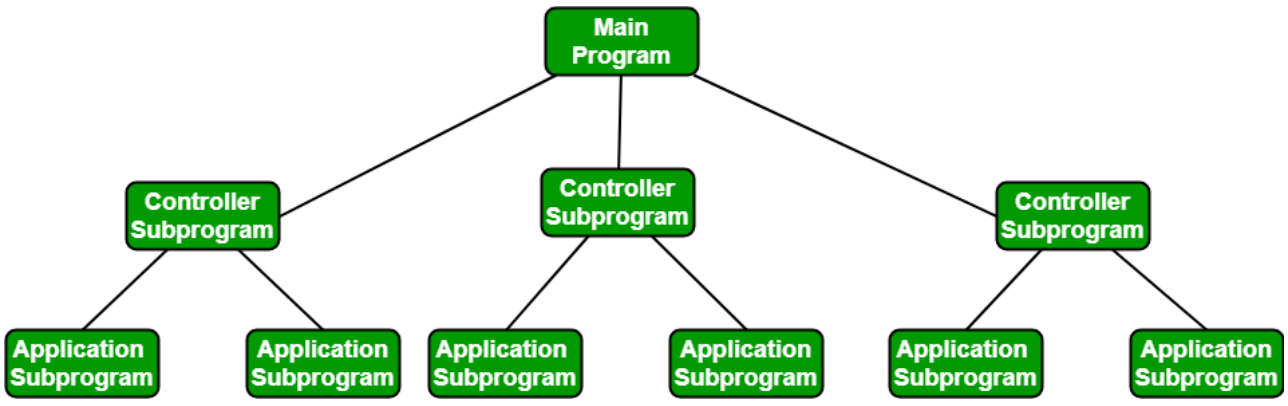


डेटा प्रवाह वास्तुकला

3] कॉल और रिटर्न आर्किटेक्चर

इसका उपयोग ऐसे प्रोग्राम बनाने के लिए किया जाता है जिसे आसानी से स्केल और संशोधित किया जा सके। इस श्रेणी में कई उप-शैलियाँ मौजूद हैं। उनमें से दो का विवरण नीचे दिया गया है।

- **रिमोट प्रोसीजर कॉल आर्किटेक्चर:** इस घटक का उपयोग नेटवर्क पर एकाधिक कंप्यूटरों के बीच वितरित मुख्य प्रोग्राम या उप-प्रोग्राम आर्किटेक्चर में प्रस्तुत करने के लिए किया जाता है।
- **मुख्य प्रोग्राम या उप-प्रोग्राम आर्किटेक्चर:** मुख्य प्रोग्राम संरचना कई उप-प्रोग्रामों या कार्यों में विभाजित होकर एक नियंत्रण पदानुक्रम बनाती है। मुख्य प्रोग्राम में कई उप-प्रोग्राम होते हैं जो अन्य घटकों को क्रियान्वित कर सकते हैं।



4] ऑब्जेक्ट ओरिएंटेड आर्किटेक्चर

किसी सिस्टम के घटक डेटा और उन क्रियाओं को समाहित करते हैं जिन्हें डेटा में हेरफेर करने के लिए लागू किया जाना चाहिए। घटकों के बीच समन्वय और संचार संदेश प्रेषण के माध्यम से स्थापित होता है।

ऑब्जेक्ट ओरिएंटेड आर्किटेक्चर की विशेषताएँ :

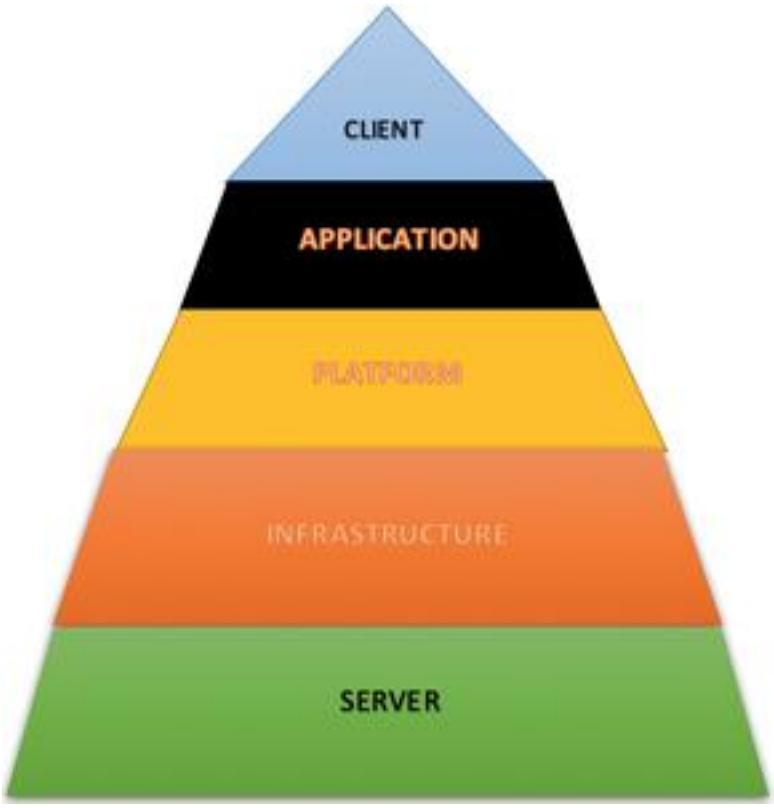
- ऑब्जेक्ट सिस्टम की अखंडता की रक्षा करता है।
- एक वस्तु अन्य वस्तुओं के चित्रण से अनभिज्ञ होती है।

ऑब्जेक्ट ओरिएंटेड आर्किटेक्चर के लाभ:

- यह डिजाइनर को एक चुनौती को स्वायत्त वस्तुओं के संग्रह में अलग करने में सक्षम बनाता है।
- अन्य ऑब्जेक्ट्स को ऑब्जेक्ट के कार्यान्वयन विवरण की जानकारी होती है, जिससे अन्य ऑब्जेक्ट्स पर प्रभाव डाले बिना परिवर्तन किया जा सकता है।

5] स्तरित वास्तुकला

- कई अलग-अलग परतें परिभाषित की जाती हैं, जिनमें से प्रत्येक परत एक सुपरिभाषित संचालन सेट निष्पादित करती है। प्रत्येक परत कुछ ऐसे संचालन करेगी जो क्रमशः मशीन निर्देश सेट के करीब पहुँचते जाते हैं ।
- बाहरी परत पर, घटक उपयोगकर्ता इंटरफ़ेस संचालन प्राप्त करेंगे और आंतरिक परतों पर, घटक ऑपरेटिंग सिस्टम इंटरफेसिंग (ओएस के साथ संचार और समन्वय) करेंगे।
- उपयोगिता सेवाओं और अनुप्रयोग सॉफ्टवेयर कार्यों के लिए मध्यवर्ती परतें।
- इस स्थापत्य शैली का एक सामान्य उदाहरण ओएसआई-आईएसओ (ओपन सिस्टम इंटरकनेक्शन-अंतर्राष्ट्रीय मानकीकरण संगठन) संचार प्रणाली है ।



स्तरित वास्तुकला

सॉफ्टवेयर इंजीनियरिंग और संगठनात्मक सिद्धांत में एक संरचना चार्ट (SC) **एक चार्ट होता है जो किसी सिस्टम के सबसे छोटे हिस्से को उसके प्रबंधनीय न्यूनतम स्तर तक दर्शाता है** । इनका उपयोग संरचित प्रोग्रामिंग में प्रोग्राम मॉड्यूल को एक वृक्ष के रूप में व्यवस्थित करने के लिए किया जाता है। प्रत्येक मॉड्यूल को एक बॉक्स द्वारा दर्शाया जाता है, जिसमें मॉड्यूल का नाम होता है।

रिपॉजिटरी डिज़ाइन पैटर्न

- रिपॉजिटरी डिज़ाइन पैटर्न एक संरचनात्मक पैटर्न है जो डेटा एक्सेस को अमूर्त बनाता है और डेटा संचालनों के प्रबंधन का एक केंद्रीकृत तरीका प्रदान करता है। डेटा लेयर को व्यावसायिक तर्क से अलग करके, यह कोड की रखरखाव क्षमता, परीक्षण क्षमता और लचीलेपन को बढ़ाता है, जिससे किसी एप्लिकेशन में विभिन्न डेटा स्रोतों के साथ काम करना आसान हो जाता है।

रिपॉजिटरी डिज़ाइन पैटर्न: डेटा एक्सेस को सरल बनाना

रिपॉजिटरी डिज़ाइन पैटर्न एक सॉफ्टवेयर डिज़ाइन पैटर्न है जो किसी एप्लिकेशन के व्यावसायिक तर्क और डेटा भंडारण के बीच एक मध्यस्थ परत के रूप में कार्य करता है।

- इसका प्राथमिक उद्देश्य डेटा भंडारण प्रौद्योगिकियों के अंतर्निहित विवरणों को सारगर्भित करते हुए डेटा तक पहुंचने, प्रबंधित करने और हेरफेर करने के लिए एक संरचित और मानकीकृत तरीका प्रदान करना है।
- यह पैटर्न चिंताओं के स्पष्ट पृथक्करण को बढ़ावा देता है, जिससे सॉफ्टवेयर अधिक रखरखाव योग्य, परीक्षण योग्य और डेटा स्रोतों में परिवर्तन के लिए अनुकूलनीय बन जाता है।

रिपोजिटरी मॉडल **सॉफ़्टवेयर इंजीनियरिंग** में इस्तेमाल किया जाने वाला एक प्रकार का **आर्किटेक्चरल मॉडल,** किसी सिस्टम के विभिन्न घटकों के बीच डेटा एक्सचेंज को प्रबंधित और समन्वित करने के लिए इस्तेमाल किया जाता है। यह उन सिस्टम में विशेष रूप से उपयोगी है जहाँ **कई घटकों को डेटा के एक सामान्य सेट को साझा और प्रबंधित करने की आवश्यकता होती है** ।

रिपोजिटरी मॉडल क्या है?

रिपॉजिटरी मॉडल में , **सभी डेटा एक केंद्रीय रिपॉजिटरी में प्रबंधित होता है** , जो सभी सिस्टम घटकों द्वारा सुलभ होता है। ये घटक डेटा को पुनः प्राप्त करने, अपडेट करने या संग्रहीत करने के लिए रिपॉजिटरी के साथ इंटरैक्ट करते हैं।

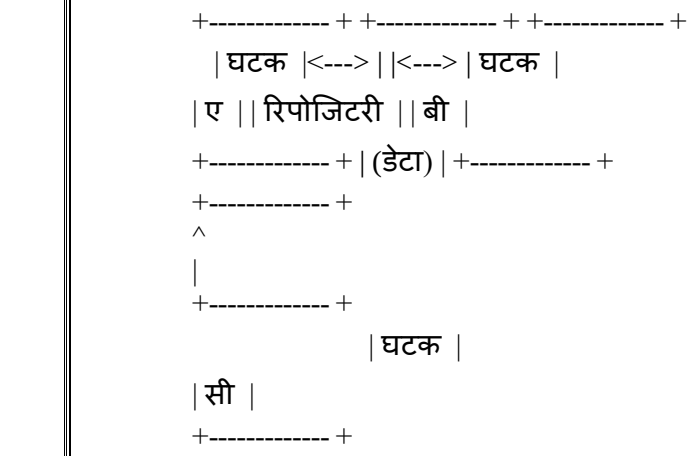
रिपॉजिटरी मॉडल के प्रमुख घटक:

1. **रिपॉजिटरी (केंद्रीय डेटा स्टोर)**

- एक साझा डेटाबेस या डेटा संरचना जहाँ सभी जानकारी संग्रहीत होती है।

2. **सॉफ़्टवेयर घटक (क्लाइंट/टूल)**
    - ये घटक (जैसे संपादक, विश्लेषक, प्रोसेसर) डेटा प्राप्त करने या अद्यतन करने के लिए रिपोजिटरी का उपयोग करते हैं।
3. **रिपॉजिटरी इंटरफ़ेस**
    - परिभाषित करता है कि घटक रिपॉजिटरी में डेटा तक कैसे पहुंचते हैं (उदाहरण के लिए, API या क्वेरी भाषाएं)।

आरेख (पाठ प्रतिनिधित्व)



रिपॉजिटरी मॉडल के लाभ:

- **केंद्रीकृत डेटा प्रबंधन** - डेटा प्रबंधन को अधिक सुसंगत और सुरक्षित बनाता है।
- **आसान रखरखाव** - डेटा संरचना का अद्यतन एक ही स्थान पर होता है।
- **पुन: प्रयोज्यता को बढ़ावा देता है** - घटक डेटा और उपकरणों का पुन: उपयोग कर सकते हैं।

नुकसान:

- **रिपॉजिटरी अड़चन** - विफलता या प्रदर्शन संबंधी समस्याओं का एकल बिंदु।
- **मापनीयता सीमाएँ** - बहुत बड़ी या वितरित प्रणालियों के साथ अच्छी तरह से मापनीय नहीं हो सकती हैं।
- **तंग युग्मन** - घटक भंडार संरचना पर बहुत अधिक निर्भर हो सकते हैं।

रिपॉजिटरी मॉडल का उपयोग कब करें:

- **संकलक प्रणालियों** में (उदाहरण के लिए, लेक्सिकल विश्लेषक, पार्सर, सिमेंटिक विश्लेषक के बीच साझा प्रतीक तालिका)।
- **CASE टूल्स** (कम्प्यूटर-एडेड सॉफ्टवेयर इंजीनियरिंग) में , जहां डिज़ाइन टूल्स एक सामान्य डेटा मॉडल साझा करते हैं।
- **डेटाबेस-केंद्रित अनुप्रयोगों** में .

संबंधित वास्तुशिल्प मॉडल:

नमूना	मुख्य विचार	उदाहरण उपयोग मामला
कोष	घटकों द्वारा साझा किया गया केंद्रीकृत डेटा	IDEs, CASE उपकरण
ग्राहक सर्वर	क्लाइंट सर्वर से सेवाओं का अनुरोध करते हैं	वेब अनुप्रयोग
पाइप और फ़िल्टर	चरणों में संसाधित डेटा (फ़िल्टर)	कंपाइलर, डेटा प्रोसेसिंग
बहुस्तरीय	परतों में व्यवस्थित कार्यक्षमता	ओएस, नेटवर्क प्रोटोकॉल स्टैक

क्लाइंट-सर्वर मॉडल

•

क्लाइंट-सर्वर मॉडल एक वितरित आर्किटेक्चर है जहाँ क्लाइंट सेवाओं का अनुरोध करते हैं और सर्वर उन्हें प्रदान करते हैं। क्लाइंट सर्वर को अनुरोध भेजते हैं, जो उन्हें संसाधित करके परिणाम लौटाते हैं। क्लाइंट आपस में संसाधन साझा नहीं करते, बल्कि सर्वर पर निर्भर रहते हैं। इसके सामान्य उदाहरणों में ईमेल सिस्टम और वर्ल्ड वाइड वेब शामिल हैं, जहाँ क्लाइंट (जैसे ब्राउज़र या ईमेल ऐप) सामग्री तक पहुँचने या डेटा भेजने के लिए सर्वर से इंटरैक्ट करते हैं।

क्लाइंट-सर्वर मॉडल कैसे काम करता है?

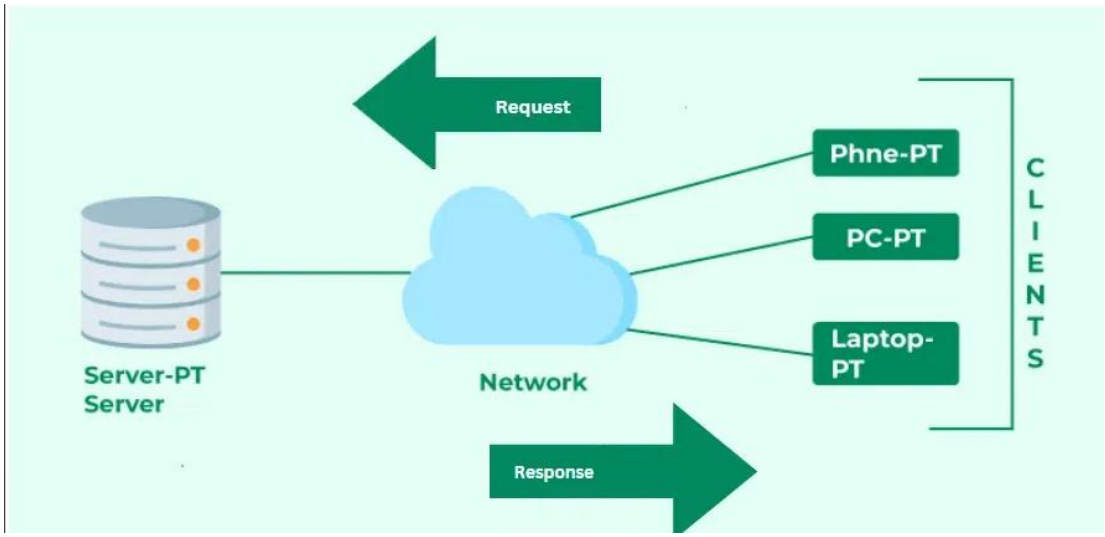
ग्राहक

जब हम "क्लाइंट" की बात करते हैं, तो इसका मतलब एक ऐसा उपकरण होता है जो सर्वर से सेवाओं का अनुरोध करता है और उन्हें प्राप्त करता है। क्लाइंट वह इकाई है जो संचार शुरू करती है, सर्वर से डेटा या संसाधन मांगती है। उदाहरण के लिए, गूगल क्रोम, मोज़िला फ़ायरफ़ॉक्स या सफारी जैसे वेब ब्राउज़र सामान्य क्लाइंट एप्लिकेशन हैं जो वेब पेज रेंडर करने के लिए सर्वर से डेटा का अनुरोध करते हैं।

सर्वर

दूसरी ओर, सर्वर एक दूरस्थ कंप्यूटर या सिस्टम होता है जो क्लाइंट को डेटा, संसाधन या सेवाएँ प्रदान करता है। यह आने वाले क्लाइंट अनुरोधों को सुनता है, उन्हें संसाधित करता है और आवश्यक जानकारी वापस भेजता है। एक सर्वर एक साथ कई क्लाइंट अनुरोधों को संभाल सकता है।

उदाहरण: *वेब सर्वर वेबसाइटों को होस्ट करते हैं, जबकि डेटाबेस सर्वर डेटा संग्रहीत करते हैं और उस तक पहुँच प्रदान करते हैं। क्लाइंट अनुरोध भेजते हैं और सर्वर तब तक प्रतिक्रिया देते हैं जब तक अनुरोधित डेटा या सेवा उपलब्ध हो।*



क्लाइंट सर्वर मॉडल

ब्राउज़र सर्वर के साथ कैसे इंटरैक्ट करता है?

ब्राउज़र के ज़रिए सर्वर से इंटरैक्ट करने की प्रक्रिया में कई चरण शामिल होते हैं। ब्राउज़र में URL दर्ज करने और वेबसाइट डेटा प्राप्त करने के दौरान उठाए जाने वाले कदमों का विवरण यहां दिया गया है:

1. उपयोगकर्ता URL (यूनिफ़ॉर्म रिसोर्स लोकेटर) दर्ज करता है: उपयोगकर्ता ब्राउज़र के एड्रेस बार में एक वेबसाइट का पता (जैसे, www.example.com) टाइप करता है।

2. DNS (डोमेन नाम सिस्टम) लुकअप: ब्राउज़र मानव-पठनीय URL को IP पते में बदलने के लिए DNS सर्वर को अनुरोध भेजता है (चूंकि कंप्यूटर एक दूसरे को पहचानने और कनेक्ट करने के लिए IP पते का उपयोग करते हैं)।

3. DNS सर्वर पता हल करता है: DNS सर्वर डोमेन नाम को खोजता है और अनुरोधित वेबसाइट को होस्ट करने वाले वेब सर्वर का IP पता लौटाता है।

4. ब्राउज़र HTTP/HTTPS अनुरोध भेजता है: ब्राउज़र वेबसाइट का डेटा प्राप्त करने के लिए वेब सर्वर के IP पते पर एक HTTP/HTTPS अनुरोध भेजता है। HTTP (हाइपरटेक्स्ट ट्रांसफर प्रोटोकॉल) या HTTPS (सुरक्षित संस्करण) वह प्रोटोकॉल है जिसका उपयोग ब्राउज़र (क्लाइंट) और वेब सर्वर (सर्वर) के बीच संचार के लिए किया जाता है।

5. सर्वर वेबसाइट फ़ाइलें भेजता है: सर्वर अनुरोध को संसाधित करता है और आवश्यक वेबसाइट फ़ाइलें (HTML, CSS, JavaScript, चित्र, आदि) ब्राउज़र को वापस भेजता है।

6. वेबसाइट रेंडर करना: ब्राउज़र फ़ाइलों को रेंडर करता है और उपयोगकर्ता को वेबसाइट दिखाता है। इस रेंडरिंग प्रक्रिया में कई घटक एक साथ काम करते हैं:

- **DOM (डॉक्यूमेंट ऑब्जेक्ट मॉडल) इंटरप्रेटर:** HTML संरचना को संसाधित करता है और पृष्ठ का वृक्ष जैसा प्रतिनिधित्व बनाता है।
- **सीएसएस इंटरप्रेटर:** HTML तत्वों पर शैलियाँ लागू करता है।
- **JS इंजन:** अन्तरक्रियाशीलता के लिए जावास्क्रिप्ट कोड निष्पादित करता है। अधिकांश आधुनिक जावास्क्रिप्ट इंजन प्रदर्शन को अनुकूलित करने के लिए जस्ट-इन-टाइम (JIT) संकलन का उपयोग करते हैं।

सॉफ्टवेयर इंजीनियरिंग में, एक अमूर्त मशीन मॉडल, कंप्यूटर सिस्टम का एक सैद्धांतिक मॉडल होता है जो विश्लेषण और डिज़ाइन के आवश्यक पहलुओं पर ध्यान केंद्रित करने के लिए अनावश्यक विवरणों को अमूर्त कर देता है। यह एक "मशीन" है क्योंकि यह प्रोग्रामों के चरण-दर-चरण निष्पादन की अनुमति देता है, लेकिन "अमूर्त" इसलिए है क्योंकि यह वास्तविक हार्डवेयर के कई पहलुओं की उपेक्षा करता है। इन मॉडलों का उपयोग सिस्टम के व्यवहार को समझने, सॉफ्टवेयर डिज़ाइन करने और यहाँ तक कि यह पता लगाने के लिए किया जाता है कि विभिन्न हार्डवेयर घटक प्रदर्शन को कैसे प्रभावित कर सकते हैं।

यहां अधिक विस्तृत विवरण दिया गया है:

मुख्य विशेषताएं:

- **अमूर्तन:**

  किसी सिस्टम के आवश्यक पहलुओं पर ध्यान केंद्रित करता है, विशिष्ट हार्डवेयर या प्रोग्रामिंग भाषाओं जैसे कार्यान्वयन विवरणों को अनदेखा करता है।

- **औपचारिकीकरण:**

मॉडल को परिभाषित करने के लिए सेट, तर्क और फ़ंक्शन जैसी गणितीय संरचनाओं का उपयोग करता है।

- **चरण-दर-चरण निष्पादन:**

  यह मशीन किस प्रकार इनपुट को संसाधित करती है और आउटपुट उत्पन्न करती है, इसका वर्णन करके प्रोग्राम व्यवहार के अनुकरण या विश्लेषण की अनुमति देता है।

- **सैद्धांतिक आधार:**
  यह गणना, जटिलता और जटिल प्रणालियों के व्यवहार को समझने के लिए आधार प्रदान करता है।

- ## **Abstract machine model** का उपयोग क्यों करें?

- **सिस्टम व्यवहार को समझना:**

  यह किसी प्रणाली के कार्य करने के तरीके का विश्लेषण करने का एक स्पष्ट तरीका प्रदान करता है, चाहे वह उसके ठोस कार्यान्वयन से स्वतंत्र हो।

- **डिजाइन और विकास:**

  सिस्टम को कैसे संचालित किया जाना चाहिए, इसका खाका प्रदान करके सॉफ्टवेयर डिजाइन करने में सहायता करता है।

- **अदाकारी का समीक्षण:**

  प्रॉक्सी आर्किटेक्चर का उपयोग करके यह पता लगाने की अनुमति देता है कि विभिन्न हार्डवेयर कॉन्फ़िगरेशन सॉफ्टवेयर प्रदर्शन को कैसे प्रभावित कर सकते हैं , आईईईईई एक्सप्लोर के अनुसार .

- **औपचारिक सत्यापन:**

  सॉफ्टवेयर और प्रणालियों की शुद्धता साबित करने के लिए एक ढांचा प्रदान करता है।

- **भाषा कार्यान्वयन:**
  प्रोग्रामिंग भाषाओं के निष्पादन के लिए एक मॉडल प्रदान करके उनके डिजाइन और कार्यान्वयन में सहायता करता है।

अमूर्त मशीनों के उदाहरण:

- **ट्यूरिंग मशीन :**

  गणना का एक सैद्धांतिक मॉडल जो गणना योग्यता सिद्धांत का आधार बनता है।

- **परिमित राज्य मशीन :**

सीमित संख्या में अवस्थाओं और उनके बीच संक्रमण वाली प्रणालियों का वर्णन करने के लिए एक मॉडल।

- **Abstract machine model:-**
  सॉफ्टवेयर और हार्डवेयर सहित जटिल प्रणालियों के बारे में निर्दिष्टीकरण और तर्क के लिए एक औपचारिक मॉडल।

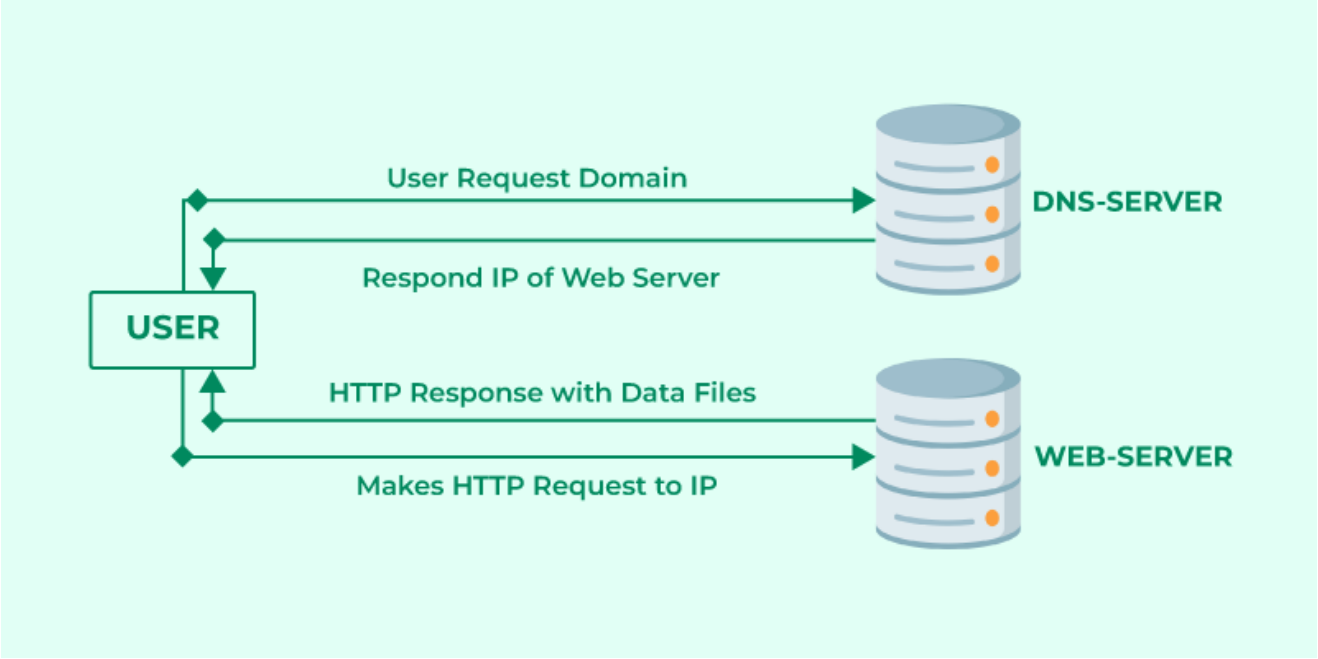
- **<u>जावा वर्चुअल मशीन (JVM)</u> :**
  एक अमूर्त मशीन का ठोस उदाहरण जो जावा बाइटकोड के लिए रनटाइम वातावरण प्रदान करता है ।

सॉफ्टवेयर इंजीनियरिंग के संदर्भ में:

अमूर्त मशीन मॉडल का उपयोग सॉफ्टवेयर विकास के विभिन्न चरणों में किया जाता है:

- **आवश्यकता विनिर्देश:** ASM का उपयोग उच्च स्तर के अमूर्तन पर किसी प्रणाली के वांछित व्यवहार को मॉडल करने के लिए किया जा सकता है।
- **सॉफ्टवेयर डिजाइन:** इनका उपयोग किसी सिस्टम की वास्तुकला और घटकों को डिजाइन करने के लिए किया जा सकता है।
- **कार्यान्वयन:** वे अमूर्त मॉडल को ठोस कोड में अनुवाद करने के लिए आधार प्रदान करते हैं।
- **परीक्षण और सत्यापन:** इनका उपयोग सिस्टम की शुद्धता और प्रदर्शन को सत्यापित करने के लिए किया जा सकता है।

संक्षेप में, अमूर्त मशीन मॉडल अनावश्यक विवरणों को हटाकर तथा मूल कार्यक्षमता और व्यवहार पर ध्यान केंद्रित करके सॉफ्टवेयर प्रणालियों को समझने, डिजाइन करने और तर्क करने के लिए एक शक्तिशाली उपकरण प्रदान करते हैं।



क्लाइंट सर्वर अनुरोध और प्रतिक्रिया

क्लाइंट-सर्वर मॉडल के वास्तविक उदाहरण

1. ईमेल सिस्टम

- **क्लाइंट:** ईमेल ऐप्स (जैसे, आउटलुक, जीमेल)
- **सर्वर:** ईमेल प्रदाता (जैसे, जीमेल सर्वर, याहू मेल सर्वर)
- **यह कैसे काम करता है:** क्लाइंट सर्वर के माध्यम से ईमेल प्राप्त करते हैं और भेजते हैं।

2. वर्ल्ड वाइड वेब

- **क्लाइंट:** वेब ब्राउज़र (जैसे, क्रोम, फ़ायरफ़ॉक्स)
- **सर्वर:** वेब सर्वर (जैसे, Apache, Nginx)
- **यह कैसे काम करता है:** ब्राउज़र वेब पेजों का अनुरोध करते हैं; सर्वर सामग्री के साथ प्रतिक्रिया देते हैं।

3. क्लाउड स्टोरेज सेवाएँ

- **क्लाइंट:** उपयोगकर्ता डिवाइस (जैसे, पीसी, स्मार्टफोन)
- **सर्वर:** क्लाउड प्लेटफ़ॉर्म (जैसे, गूगल ड्राइव, ड्रॉपबॉक्स)
- **यह कैसे काम करता है:** क्लाइंट फ़ाइलें अपलोड/डाउनलोड करते हैं; सर्वर उन्हें दूर से संग्रहीत और प्रस्तुत करते हैं।

क्लाइंट-सर्वर मॉडल के लाभ

क्लाइंट-सर्वर मॉडल कई लाभ प्रदान करता है जो इसे नेटवर्क और वितरित प्रणालियों में लोकप्रिय बनाते हैं:

- **केंद्रीकृत डेटा प्रबंधन** : सभी डेटा एक केंद्रीकृत सर्वर में संग्रहीत किया जाता है, जिससे इसे प्रबंधित करना, अद्यतन करना और बैकअप करना आसान हो जाता है।
- **लागत दक्षता** : चूंकि सर्वर अधिकांश प्रसंस्करण का काम संभालता है, इसलिए क्लाइंट को कम संसाधनों की आवश्यकता होती है और वे सरल डिवाइस हो सकते हैं, जिससे लागत कम हो जाती है।
- **स्केलेबिलिटी** : क्लाइंट और सर्वर, दोनों को अलग-अलग स्केल किया जा सकता है। सर्वर को ज़्यादा क्लाइंट्स को संभालने के लिए अपग्रेड किया जा सकता है और सर्वर इंफ्रास्ट्रक्चर में बिना कोई बड़ा बदलाव किए नए क्लाइंट्स जोड़े जा सकते हैं।
- **डेटा रिकवरी** : सर्वर पर केंद्रीकृत डेटा भंडारण बेहतर डेटा रिकवरी और आसान बैकअप रणनीतियों की अनुमति देता है।
- **सुरक्षा** : फायरवॉल, एन्क्रिप्शन और प्रमाणीकरण जैसे **सुरक्षा उपायों को** सर्वर पर केंद्रीकृत किया जा सकता है, जिससे यह सुनिश्चित होता है कि संवेदनशील डेटा सुरक्षित है।

क्लाइंट-सर्वर मॉडल के नुकसान

- **संक्रमित डाउनलोड के प्रति क्लाइंट की भेद्यता:** यदि सर्वर दुर्भावनापूर्ण फ़ाइलों (जैसे, वायरस या ट्रोजन) को होस्ट करता है, तो क्लाइंट अनजाने में उन्हें डाउनलोड और निष्पादित कर सकते हैं, जिससे उनके सिस्टम को खतरा हो सकता है।
- **सर्वर लक्ष्य होते हैं** : सर्वर सेवा निषेध (DOS) हमलों के प्रति संवेदनशील होते हैं, जहां सर्वर पर ट्रैफिक का दबाव बढ़ जाता है और वह वैध ग्राहकों के लिए अनुपलब्ध हो जाता है।
- **डेटा स्पूफिंग और संशोधन** : यदि उचित सुरक्षा उपाय (जैसे, एन्क्रिप्शन) लागू नहीं किए जाते हैं, तो डेटा पैकेट को ट्रांसमिशन के दौरान स्पूफ या संशोधित किया जा सकता है।
- **मैन-इन-द-मिडिल (MITM) हमले** : हमलावर, विशेष रूप से असुरक्षित नेटवर्क पर, लॉगिन क्रेडेंशियल या संवेदनशील डेटा चुराने के लिए क्लाइंट और सर्वर के बीच संचार को बाधित कर सकते हैं।

सॉफ़्टवेयर इंजीनियरिंग में, एक नियंत्रण मॉडल यह परिभाषित करता है कि सॉफ़्टवेयर सिस्टम के विभिन्न भाग कैसे परस्पर क्रिया करते हैं और अपने निष्पादन प्रवाह का प्रबंधन कैसे करते हैं। यह अनिवार्य रूप से उस क्रम को निर्धारित करता है जिसमें घटक निष्पादित होते हैं और वे घटनाओं या डेटा पर कैसे प्रतिक्रिया देते हैं। नियंत्रण मॉडल मज़बूत, रखरखाव योग्य और मापनीय सॉफ़्टवेयर बनाने के लिए महत्वपूर्ण हैं।

नियंत्रण मॉडल के प्रमुख पहलुओं का विवरण इस प्रकार है:

1. केंद्रीकृत बनाम विकेन्द्रीकृत नियंत्रण:

- **केंद्रीकृत नियंत्रण :**

  एक घटक "नियंत्रक" के रूप में कार्य करता है, जो अन्य घटकों के निष्पादन को निर्देशित करता है। यह एक टॉप-डाउन कॉल-रिटर्न मॉडल (सबरूटीन की तरह) या समवर्ती प्रणालियों के लिए एक प्रबंधक मॉडल हो सकता है।

- **विकेन्द्रीकृत नियंत्रण :**
  घटक एक दूसरे के साथ सीधे तौर पर, अक्सर घटनाओं या संदेशों के माध्यम से, बिना किसी केंद्रीय प्राधिकरण के, अंतःक्रिया करते हैं।

2. नियंत्रण प्रवाह आरेख:

- **नियंत्रण प्रवाह आरेख** : किसी सिस्टम के भीतर क्रियाओं और निर्णयों के अनुक्रम को दृश्य रूप से दर्शाते हैं। ये आरेख दर्शाते हैं कि सॉफ़्टवेयर के विभिन्न भागों के बीच नियंत्रण कैसे चलता है, जिसमें स्थितियों के आधार पर शाखाएँ बनाना भी शामिल है।
- ये आरेख प्रणाली के तर्क को समझने और नियंत्रण प्रवाह में संभावित समस्याओं की पहचान करने में मदद करते हैं।

3. मॉडल-व्यू-कंट्रोलर (एमवीसी) :

- एमवीसी एक लोकप्रिय आर्किटेक्चरल पैटर्न है जो किसी एप्लिकेशन को तीन परस्पर जुड़े भागों में विभाजित करता है: मॉडल, व्यू और कंट्रोलर।
- **मॉडल:** डेटा और व्यावसायिक तर्क का प्रबंधन करता है।
- **दृश्य :** उपयोगकर्ता को डेटा प्रदर्शित करता है और उपयोगकर्ता इनपुट को संभालता है।

- **नियंत्रक :** मध्यस्थ के रूप में कार्य करता है, उपयोगकर्ता इनपुट को संभालता है, मॉडल को अद्यतन करता है, और दृश्य का प्रबंधन करता है।
- एमवीसी घटकों के बीच ढीले युग्मन को बढ़ावा देता है, जिससे अनुप्रयोग अधिक मॉड्यूलर हो जाता है और रखरखाव आसान हो जाता है।

4. नियंत्रण विनिर्देश:

- **नियंत्रण विनिर्देशन** : सिस्टम को कैसे नियंत्रित किया जाता है इसके लिए नियमों और प्रक्रियाओं को परिभाषित करता है।
- **स्वचालित प्रणालियाँ :**

  साइंसडायरेक्ट.कॉम के अनुसार , स्वचालित प्रणालियों में नियंत्रण मॉडल आवश्यक हैं, जैसे कि विनिर्माण या रोबोटिक्स में उपयोग किए जाने वाले, कार्यों के निष्पादन का प्रबंधन करने और वास्तविक समय की घटनाओं पर प्रतिक्रिया देने के लिए।

- **वेब अनुप्रयोग :**
  वेब विकास में MVC का व्यापक रूप से उपयोग उपयोगकर्ता इंटरफेस (दृश्य), डेटा (मॉडल) और उपयोगकर्ता क्रियाओं (नियंत्रकों) के बीच अंतःक्रिया को प्रबंधित करने के लिए किया जाता है।

संक्षेप में, नियंत्रण मॉडल सॉफ्टवेयर प्रणालियों के डिजाइन और कार्यान्वयन के लिए मौलिक हैं, जो यह सुनिश्चित करते हैं कि वे सही ढंग से, कुशलतापूर्वक कार्य करें, तथा बदलती आवश्यकताओं के अनुकूल हों।

मॉड्यूल अपघटन | सिस्टम डिज़ाइन

- मॉड्यूल विघटन, सिस्टम डिज़ाइन का एक महत्वपूर्ण पहलू है, जो जटिल प्रणालियों को छोटे, अधिक प्रबंधनीय मॉड्यूल या घटकों में विभाजित करने में सक्षम बनाता है। ये मॉड्यूल विशिष्ट कार्यात्मकताओं को समाहित करते हैं, जिससे सिस्टम को समझना, रखरखाव करना और मापना आसान हो जाता है। इस लेख में, हम सिस्टम डिज़ाइन में मॉड्यूल विघटन की अवधारणा, इसके लाभों, सामान्य तकनीकों और सर्वोत्तम प्रथाओं का पता लगाएंगे।

मॉड्यूल decomposition क्या है?

मॉड्यूल विघटन, सिस्टम डिज़ाइन और आर्किटेक्चर में एक प्रक्रिया है जहाँ एक जटिल सिस्टम को छोटे, अधिक प्रबंधनीय मॉड्यूल या घटकों में विभाजित किया जाता है। प्रत्येक मॉड्यूल सिस्टम की एक विशिष्ट कार्यक्षमता, डेटा या व्यवहार को समाहित करता है, और मॉड्यूल को सुपरिभाषित इंटरफेस के माध्यम से एक-दूसरे के साथ अंतःक्रिया करने के लिए डिज़ाइन किया जाता है।

सिस्टम डिज़ाइन में मॉड्यूल अपघटन का महत्व

- **समझ** : प्रणाली को अधिक प्रबंधनीय, छोटे घटकों में विभाजित करके इसकी समझ को बढ़ाता है।
- **रखरखाव** : विशेष मॉड्यूल में संशोधनों को अलग करने से रखरखाव और उन्नयन आसान हो जाता है।
- **प्रयोज्यता** : एक ही परियोजना के भीतर या अलग-अलग परियोजनाओं के बीच मॉड्यूल के पुन: उपयोग को प्रोत्साहित करता है।
- **मापनीयता** : बड़ी, अधिक जटिल प्रणालियों को छोटे, प्रबंधन में आसान टुकड़ों में विभाजित करता है, जिससे अधिक कुशल मापनीयता संभव होती है।
- **परीक्षण** : परीक्षण और डिबगिंग की प्रक्रिया को आसान बनाने के लिए मॉड्यूल को अलग करता है।

मॉड्यूल अपघटन के सिद्धांत और उद्देश्य

- **युग्मन** : परिवर्तनों के प्रभाव को कम करने के लिए, मॉड्यूल में कम युग्मन होना चाहिए, जिसका अर्थ है कि वे एक दूसरे से काफी हद तक स्वतंत्र होने चाहिए या शिथिल रूप से जुड़े होने चाहिए।
- **मापनीयता** : मॉड्यूल को भविष्य के विस्तार और आवश्यकता संशोधनों को ध्यान में रखकर बनाया जाना चाहिए।
- **जानकारी छिपाना** : केवल अन्य मॉड्यूलों के साथ संचार करने के लिए आवश्यक इंटरफेस को ही मॉड्यूलों द्वारा उजागर किया जाना चाहिए, जिससे उनकी आंतरिक कार्यप्रणाली समाहित हो सके।

- **संसक्ति** : यदि किसी मॉड्यूल के घटक एक दूसरे से घनिष्ठ रूप से जुड़े हुए हैं तथा किसी विशेष कर्तव्य या कार्य पर केन्द्रित हैं तो उसमें उच्च स्तर की संसक्ति होनी चाहिए।
- **प्रयोज्यता** : दोहराव को कम करने और उत्पादकता बढ़ाने के लिए, मॉड्यूल को इस प्रकार बनाया जाना चाहिए कि उनका विभिन्न स्थितियों में पुनः उपयोग किया जा सके।

सिस्टम आर्किटेक्चर में मॉड्यूल अपघटन की भूमिका

मॉड्यूल विघटन, जो किसी सिस्टम के घटकों और उनकी अंतःक्रियाओं को व्यवस्थित करता है, सिस्टम की वास्तुकला को डिज़ाइन करने के लिए आवश्यक है। यह सिस्टम के संरचनात्मक संगठन में प्रभावी विकास, रखरखाव और विकास को बढ़ावा देने में सहायता करता है।

- **मॉड्यूलरिटी** :
    - मॉड्यूल विघटन एक जटिल प्रणाली को छोटे, अधिक प्रबंधनीय मॉड्यूल में तोड़ने में मदद करता है।
    - प्रत्येक मॉड्यूल को स्वतंत्र रूप से विकसित, परीक्षण और रखरखाव किया जा सकता है, जिससे समग्र प्रणाली को समझना और संशोधित करना आसान हो जाता है।
- **एनकैप्सुलेशन** :
    - मॉड्यूल संबंधित कार्यक्षमता, डेटा और व्यवहार को समाहित करते हैं, तथा आंतरिक विवरण को अन्य मॉड्यूल से छिपाते हैं।
    - इससे निर्भरता कम हो जाती है और यह सुनिश्चित करने में मदद मिलती है कि एक मॉड्यूल में परिवर्तन से अन्य मॉड्यूल प्रभावित न हों।
- **अमूर्तन** :
    - मॉड्यूल अमूर्तता का एक स्तर प्रदान करते हैं, जिससे डेवलपर्स को प्रत्येक मॉड्यूल के आंतरिक कार्यान्वयन को समझने की आवश्यकता के बिना उसके इंटरफेस और कार्यक्षमता पर ध्यान केंद्रित करने की अनुमति मिलती है।
    - इससे डिजाइन सरल हो जाता है और रखरखाव में सुधार होता है।
- **प्रयोज्यता** :

- मॉड्यूलर डिजाइन पुन: प्रयोज्यता को बढ़ावा देता है, क्योंकि मॉड्यूल को सिस्टम के विभिन्न भागों या अन्य प्रणालियों में आसानी से पुन: उपयोग किया जा सकता है।
- इससे विकास का समय और प्रयास कम हो जाता है, क्योंकि डेवलपर्स पहिये को फिर से आविष्कार करने के बजाय मौजूदा मॉड्यूल का लाभ उठा सकते हैं

कुल मिलाकर, मॉड्यूल विघटन, सिस्टम आर्किटेक्चर में ऐसे सिस्टम बनाने के लिए ज़रूरी है जो लचीले, रखरखाव योग्य और स्केलेबल हों। यह जटिल सिस्टम डिज़ाइन करने के लिए एक संरचित दृष्टिकोण प्रदान करता है, जिससे उन्हें समय के साथ विकसित करना, समझना और रखरखाव करना आसान हो जाता है।

मॉड्यूल अपघटन की तकनीकें

नीचे मॉड्यूल अपघटन की विभिन्न तकनीकें दी गई हैं:

1. ऊपर से नीचे अपघटन:

पूरे सिस्टम से शुरुआत करें और धीरे-धीरे इसे ज़्यादा प्रबंधनीय सबसिस्टम या मॉड्यूल में विभाजित करें। वेब एप्लिकेशन डिज़ाइन करते समय, आप समग्र आर्किटेक्चर (जैसे, फ्रंट-एंड, बैक-एंड, डेटाबेस) को परिभाषित करके शुरुआत कर सकते हैं, और फिर प्रत्येक भाग को घटकों और मॉड्यूल में विभाजित कर सकते हैं।

2. नीचे से ऊपर तक अपघटन:

अलग-अलग भागों या मॉड्यूल से शुरू करके और उन्हें क्रमिक रूप से संयोजित करके बड़े सबसिस्टम या संपूर्ण सिस्टम बनाएँ। सॉफ़्टवेयर लाइब्रेरी विकसित करते समय, आप छोटे, पुन: प्रयोज्य घटकों को लागू करके शुरुआत कर सकते हैं, और फिर उन्हें संयोजित करके बड़ी, अधिक जटिल कार्यात्मकताएँ बना सकते हैं।

3. कार्यात्मक अपघटन:

सिस्टम को उसके कार्यात्मक गुणों या आवश्यकताओं के अनुसार विघटित करें। इसे कार्यात्मक विघटन कहते हैं। बैंकिंग सिस्टम डिज़ाइन करते समय, आप

सिस्टम को खाता प्रबंधन, लेनदेन प्रसंस्करण और रिपोर्टिंग जैसे मॉड्यूल में विघटित कर सकते हैं।

4. वस्तु-उन्मुख अपघटन:

सिस्टम को तोड़ने के लिए बहुरूपता, वंशानुक्रम और एनकैप्सुलेशन जैसी अवधारणाओं का उपयोग करें। गेम डिज़ाइन करते समय, आप सिस्टम को खिलाड़ियों, दुश्मनों, हथियारों और वातावरण जैसी वस्तुओं में विभाजित कर सकते हैं, जिनमें से प्रत्येक का अपना व्यवहार और गुण होते हैं।

ये विघटन तकनीकें परस्पर अनन्य नहीं हैं और जटिल प्रणालियों को प्रभावी ढंग से डिज़ाइन करने के लिए इनका संयोजन में उपयोग किया जा सकता है। प्रत्येक तकनीक परियोजना की विशिष्ट आवश्यकताओं और बाधाओं के आधार पर, सिस्टम घटकों को विभाजित और व्यवस्थित करने के तरीके पर एक अलग दृष्टिकोण प्रदान करती है।

प्रभावी मॉड्यूल अपघटन के लिए मानदंड

- **उच्च सामंजस्य** :
    - मॉड्यूल में उच्च सामंजस्य होना चाहिए, जिसका अर्थ है कि मॉड्यूल के भीतर के तत्व निकट रूप से संबंधित होने चाहिए और एक एकल, सुपरिभाषित उद्देश्य में योगदान करने चाहिए।
    - इससे मॉड्यूल को केंद्रित रखने में मदद मिलती है और जटिलता कम हो जाती है।
- **कम युग्मन** :
    - मॉड्यूलों में कम युग्मन होना चाहिए, अर्थात वे एक दूसरे से अपेक्षाकृत स्वतंत्र होने चाहिए।
    - इससे एक मॉड्यूल में परिवर्तन का अन्य मॉड्यूल पर प्रभाव कम हो जाता है, जिससे प्रणाली अधिक लचीली हो जाती है तथा उसका रखरखाव आसान हो जाता है।
- **स्पष्ट इंटरफ़ेस** :

    - मॉड्यूल में स्पष्ट और सुपरिभाषित इंटरफेस होना चाहिए, जो यह निर्दिष्ट करे कि वे अन्य मॉड्यूल के साथ किस प्रकार इंटरैक्ट करते हैं।
    - इससे निर्भरताओं को प्रबंधित करने में मदद मिलती है और सिस्टम के बाकी हिस्सों को प्रभावित किए बिना मॉड्यूल को बदलना या संशोधित करना आसान हो जाता है।
- **एकल उत्तरदायित्व सिद्धांत (एसआरपी)** :
    - प्रत्येक मॉड्यूल में परिवर्तन के लिए एक ही जिम्मेदारी या कारण होना चाहिए।
    - इससे मॉड्यूल को केंद्रित रखने में मदद मिलती है और उन्हें समझना और संशोधित करना आसान हो जाता है।
- **पुन: उपयोग की संभावना** :
    - मॉड्यूल को पुनः उपयोग को ध्यान में रखकर डिजाइन किया जाना चाहिए, ताकि उन्हें सिस्टम के अन्य भागों या अन्य प्रणालियों में आसानी से पुनः उपयोग किया जा सके।
    - इससे विकास के समय और प्रयास को कम करने में मदद मिलती है।
- **मापनीयता** :
    - मॉड्यूल को पैमाने के अनुसार डिजाइन किया जाना चाहिए, जिसका अर्थ है कि वे सिस्टम आर्किटेक्चर में बड़े संशोधन की आवश्यकता के बिना आकार या जटिलता में परिवर्तन को समायोजित करने में सक्षम होने चाहिए।
- **परीक्षण योग्यता** :
    - मॉड्यूलों को इस तरह से डिजाइन किया जाना चाहिए कि उन्हें अलग-अलग परीक्षण करना आसान हो।
    - इससे प्रणाली की शुद्धता और विश्वसनीयता सुनिश्चित करने में मदद मिलती है।

मॉड्यूल अपघटन की प्रक्रिया

मॉड्यूल विघटन की प्रक्रिया में एक जटिल प्रणाली को छोटे, अधिक प्रबंधनीय मॉड्यूल या घटकों में तोड़ना शामिल है। मॉड्यूल विघटन की सामान्य प्रक्रिया नीचे दी गई है:

चरण 1: सिस्टम को समझें

सिस्टम की आवश्यकताओं, कार्यक्षमता और संरचना को अच्छी तरह समझने से शुरुआत करें। मुख्य घटकों की पहचान करें और जानें कि वे एक-दूसरे के साथ कैसे परस्पर क्रिया करते हैं।

चरण 2: कार्यात्मक इकाइयों की पहचान करें

सिस्टम की विभिन्न कार्यात्मक इकाइयों या विशेषताओं की पहचान करें। ये उच्च-स्तरीय कार्य हो सकते हैं जो सिस्टम करता है, जैसे उपयोगकर्ता प्रमाणीकरण, डेटा प्रोसेसिंग, या रिपोर्ट तैयार करना।

चरण 3: मॉड्यूल सीमाएँ परिभाषित करें

पहचानी गई कार्यात्मक इकाइयों के आधार पर, प्रत्येक मॉड्यूल की सीमाएँ निर्धारित करें। एक मॉड्यूल में सिस्टम की एक एकल, सुपरिभाषित कार्यक्षमता या विशेषता समाहित होनी चाहिए।

चरण 4: मॉड्यूल इंटरफेस निर्धारित करें

प्रत्येक मॉड्यूल के इंटरफ़ेस को परिभाषित करें, यह निर्दिष्ट करते हुए कि यह अन्य मॉड्यूल के साथ कैसे इंटरैक्ट करता है। इसमें इनपुट पैरामीटर, आउटपुट डेटा और अन्य मॉड्यूल पर निर्भरताएँ शामिल हैं।

चरण 5: उच्च सामंजस्य सुनिश्चित करें

सुनिश्चित करें कि प्रत्येक मॉड्यूल के तत्व आपस में घनिष्ठ रूप से जुड़े हों और एक ही, सुपरिभाषित उद्देश्य में योगदान दें। मॉड्यूल को केंद्रित और रखरखाव योग्य बनाए रखने के लिए उनमें उच्च सामंजस्य का लक्ष्य रखें।

चरण 6: युग्मन को न्यूनतम करें

मॉड्यूल के बीच युग्मन को न्यूनतम रखने का लक्ष्य रखें, यह सुनिश्चित करते हुए कि वे एक-दूसरे से अपेक्षाकृत स्वतंत्र हों। इससे एक मॉड्यूल में होने वाले परिवर्तनों का अन्य मॉड्यूल पर प्रभाव कम हो जाता है, जिससे सिस्टम अधिक लचीला और रखरखाव में आसान हो जाता है।

चरण 7: परिष्कृत करें और पुनरावृत्त करें

फीडबैक और परीक्षण के आधार पर मॉड्यूल विघटन को परिष्कृत करें। सामंजस्य में सुधार, युग्मन को कम करने और यह सुनिश्चित करने के लिए डिज़ाइन में पुनरावृत्ति करें कि मॉड्यूल सिस्टम आवश्यकताओं को पूरा करते हैं।

चरण 8: अपघटन का दस्तावेजीकरण करें

प्रत्येक मॉड्यूल के उद्देश्य, उसके इंटरफ़ेस और अन्य मॉड्यूल पर उसकी निर्भरता सहित मॉड्यूल विघटन का दस्तावेज़ीकरण करें। यह दस्तावेज़ीकरण भविष्य में सिस्टम को समझने और बनाए रखने में मदद करता है।

इस प्रक्रिया का पालन करके, डेवलपर्स एक जटिल प्रणाली को प्रभावी ढंग से प्रबंधनीय मॉड्यूल में विघटित कर सकते हैं, जिसके परिणामस्वरूप एक मॉड्यूलर, लचीला और रखरखाव योग्य सिस्टम डिज़ाइन प्राप्त होता है।

मॉड्यूल विघटन के लिए उपकरण और पद्धतियाँ

- **यूएमएल (एकीकृत मॉडलिंग भाषा)** :
    - यूएमएल, मॉड्यूल विघटन सहित सॉफ्टवेयर प्रणाली की कलाकृतियों को दृश्यमान करने, निर्दिष्ट करने, निर्माण करने और दस्तावेजीकरण करने के लिए एक मानक संकेतन प्रदान करता है।
    - यूएमएल में केस आरेख, वर्ग आरेख और पैकेज आरेख का उपयोग मॉड्यूल और उनके संबंधों को दर्शाने के लिए किया जा सकता है।
- **आईडीई (एकीकृत विकास वातावरण)** :
    - इंटेलीज आईडिया और एक्लिप्स जैसे आईडीई कोड को मॉड्यूल में विज़ुअलाइज़ और व्यवस्थित करने के लिए उपकरण प्रदान करते हैं।
    - क्लास डायग्राम, प्रोजेक्ट संरचना दृश्य और कोड रिफैक्टरिंग टूल जैसी सुविधाओं का उपयोग सिस्टम को मॉड्यूल में विघटित करने के लिए किया जा सकता है।
- **डिज़ाइन पैटर्न** :
    - डिज़ाइन पैटर्न सामान्य डिज़ाइन समस्याओं के लिए पुन: प्रयोज्य समाधान प्रदान करते हैं और सिस्टम में मॉड्यूल की संरचना में मदद कर सकते हैं।

- फैक्टरी विधि, रणनीति और ऑब्जर्वर जैसे पैटर्न का उपयोग मॉड्यूल इंटरफेस और इंटरैक्शन को परिभाषित करने के लिए किया जा सकता है।

- **चुस्त कार्यप्रणाली** :
  - स्क्रम और कानबन जैसी एजाइल पद्धतियां पुनरावृत्तीय विकास और निरंतर सुधार पर जोर देती हैं, जो समय के साथ मॉड्यूल विघटन को परिष्कृत करने में मदद कर सकती हैं।
  - कहानियां, स्प्रिंट योजना और पूर्वव्यापी जैसी चुस्त प्रथाओं का उपयोग मॉड्यूल विघटन को प्राथमिकता देने और परिष्कृत करने के लिए किया जा सकता है।
- **सॉफ्टवेयर आर्किटेक्चर उपकरण:**
  - मॉड्यूल विघटन को एंटरप्राइज़ आर्किटेक्ट, विज़ुअल पैराडाइम या ल्यूसिड चार्ट जैसे उपकरणों के उपयोग से दृश्य रूप से दर्शाया और प्रलेखित किया जा सकता है।

मॉड्यूल अपघटन के लाभ

- **समझ** : प्रणाली को छोटे, अधिक प्रबंधनीय घटकों में विभाजित करके, प्रणाली की समझ में सुधार किया जाता है।
- **रखरखाव** : विशेष मॉड्यूल में परिवर्तन को सीमित करके, संशोधनों को बनाए रखना और अद्यतन करना आसान होता है।
- **मापनीयता (Scalability)** : किसी प्रणाली को छोटे, प्रबंधन में आसान घटकों में विभाजित करने की क्षमता, जिससे अधिक प्रभावी मापनीयता संभव हो सके।
- **मॉड्यूलरिटी** : मॉड्यूलर डिजाइन को प्रोत्साहित करता है, जो सिस्टम स्केलेबिलिटी, रखरखाव और समझ को सुविधाजनक बनाता है।
- **प्रयोज्यता** : एक ही परियोजना के भीतर या परियोजनाओं के बीच मॉड्यूल की पुन: प्रयोज्यता को सुविधाजनक बनाता है।

मॉड्यूल अपघटन की चुनौतियाँ

- **रखरखाव** : यह सुनिश्चित करना कि एक मॉड्यूल में संशोधन से अन्य मॉड्यूल पर अप्रत्याशित प्रभाव न पड़े।

- **मापनीयता** : यह सुनिश्चित करना कि भविष्य में विस्तार और आवश्यक परिवर्तनों को मॉड्यूल विघटन द्वारा समायोजित किया जा सके।
- **जटिलता** : बड़ी, जटिल प्रणालियों को छोटे, अधिक प्रबंधनीय घटकों में तोड़ना कठिन हो सकता है और इसके लिए सावधानीपूर्वक विचार करने की आवश्यकता होती है।
- **दस्तावेज़ीकरण** : भविष्य के रखरखाव और संदर्भ के लिए, मॉड्यूल अपघटन प्रक्रिया और उसके तर्क का एक लिखित रिकॉर्ड प्रदान किया जाता है।
- **युग्मन और संसक्ति** : यह सुनिश्चित करने के लिए कि मॉड्यूल दृढ़तापूर्वक संसक्तिशील हों, लेकिन शिथिल रूप से जुड़े हों, युग्मन और संसक्ति को संतुलित किया जाना चाहिए।

सॉफ़्टवेयर इंजीनियरिंग में डोमेन-विशिष्ट आर्किटेक्चर (DSA) एक सॉफ़्टवेयर आर्किटेक्चर को संदर्भित करता है जिसे किसी विशिष्ट अनुप्रयोग डोमेन के लिए विशेष रूप से डिज़ाइन और अनुकूलित किया जाता है। सामान्य-उद्देश्य आर्किटेक्चर के विपरीत, जिनका उद्देश्य व्यापक प्रयोज्यता होता है, DSA को किसी विशिष्ट समस्या क्षेत्र की विशिष्ट विशेषताओं, आवश्यकताओं और बाधाओं के अनुरूप बनाया जाता है। यह विशेषज्ञता उस डोमेन के भीतर बेहतर दक्षता, प्रदर्शन और रखरखाव की अनुमति देती है।

डोमेन-विशिष्ट आर्किटेक्चर के प्रमुख पहलुओं में शामिल हैं:

- **डोमेन फोकस:**

  यह आर्किटेक्चर एक विशिष्ट डोमेन (जैसे, स्वास्थ्य सेवा, वित्तीय व्यापार, ई-कॉमर्स, एम्बेडेड सिस्टम) की मुख्य अवधारणाओं, संस्थाओं और प्रक्रियाओं के आसपास बनाया गया है।

- **विशेष घटक:**

  इसमें प्रायः विशेष सॉफ्टवेयर घटक, मॉड्यूल या सेवाएं शामिल होती हैं, जो डोमेन के विशिष्ट परिचालनों और डेटा संरचनाओं के लिए अनुकूलित होती हैं।

- **संदर्भ वास्तुकला:**

डीएसए में आम तौर पर एक संदर्भ आर्किटेक्चर शामिल होता है जो डोमेन के भीतर कई संबंधित प्रणालियों में लागू सामान्य डिजाइन निर्णयों, पैटर्न और सिद्धांतों को परिभाषित करता है, साथ ही अनुकूलन के लिए भिन्नता के बिंदुओं की पहचान भी करता है।

- **पुन: उपयोग और दक्षता:**

  डोमेन-विशिष्ट ज्ञान और सर्वोत्तम प्रथाओं को प्राप्त करके, डीएसए सॉफ्टवेयर घटकों और वास्तुशिल्प पैटर्न के पुनः उपयोग को बढ़ावा देते हैं, जिससे उस डोमेन के भीतर अनुप्रयोगों के लिए तेजी से विकास, कम लागत और बेहतर गुणवत्ता प्राप्त होती है।

- **डोमेन मॉडलिंग:**

  डीएसए का विकास प्रायः समस्या क्षेत्र को व्यापक रूप से समझने और वास्तुशिल्प डिजाइन को सूचित करने के लिए गहन डोमेन विश्लेषण और मॉडलिंग से पहले किया जाता है।

  संक्षेप में, डोमेन-विशिष्ट आर्किटेक्चर एक परिभाषित डोमेन के भीतर सॉफ्टवेयर विकास के लिए एक संरचित और अनुरूप दृष्टिकोण प्रदान करते हैं, जो कुशल, प्रभावी और रखरखाव योग्य प्रणालियों के निर्माण को सुविधाजनक बनाता है।