Software Engineering का परिचय

सीबीएसई पाठ्यक्रम पर आधारित

इन्फार्मेटिक्स प्रैक्टिसेज कक्षा -12

अध्याय -5

द्वारा:

संजीव भदौरिया

स्नातकोत्तर शिक्षक (संगणक विज्ञान)

के० वि० बाराबंकी (लखनऊ संभाग)

Software Engineering क्या है?

Software Engineering किसी भी सॉफ्टवेयर सिस्टम के design, development और maintenance के लिए एक सुगठित और व्यवस्थित कार्य का तरीका है |

Software Engineering की आवश्यकता –

इनदिनों सारी application software पर निर्भर हैं और आजकल के software बहुत बड़े बड़े और जटिल होते हैं | software को निम्न रूप से विकसित किया जाना अत्यंत आवश्यक होता है -

a. निर्देशों के अनुरूप हो
b. समय पर प्राप्त हो.
c. इरादे के अनुसार काम करता हो और उस समस्या को हल करता हो जिसके लिए इसे विकसित किया गया है।
d. बजट के अनुरूप हो अर्थात बजट से अधिक मंहगा न हो |

Software को विकसित करने के लिए software engineering बहुत ज़रूरी होता है इसके निम्न कारण हैं -

i. सही निर्देश
ii. अनुमापकता (Scalability) का स्कोप
iii. लागत नियंत्रण
iv. गुणवत्ता

सॉफ्टवेर प्रक्रिया की गतिविधियाँ (Software Process Activities)-

सॉफ्टवेर प्रक्रिया शब्द एक कई गतिविधियों का एक समूह है जो आपस में जुडी हैं और जो एक क्रम में सम्पन्न होती हैं और एक software का निर्माण करती हैं |

कुछ आधारभूत गतिविधियाँ हैं जो सभी software प्रोसेस में सामान रहती हैं | इनमे से कुछ -

1. Software विनिर्देश (specification)- यह गतिविधि निम्न के लिए उत्तरदायी है -
    a. Software के मुख्या कार्य के लिए .
    b. Operation के ऊपर नियंत्रण (Constraints).
2. Software Design और Implementation- यह गतिविधि निम्न के लिए उत्तरदायी है -
    a. निर्देशानुसार software की design बनाना
    b. Design के अनुसार प्रोग्राम लिखना
3. **Software वेरिफिकेशन और मान्यकरण (validation)-** यह क्रिया निम्न के लिए उत्तरदायी होती है -
    a. Software सभी दिए गए निर्देशों के अनुरूप है या नहीं |
    b. Software मानित design के अनुरूप कार्य कर रहा है या नहीं |
4. **Software Evolution(software maintenance)-** यह क्रिया निम्न दो बातों का ध्यान रखती है -
    a. बनाया गया software, customer के समस्त बिन्दुओं को पूरा कर रहा है या नहीं |
    b. बनाया गया software design में customer के अनुसार सुधार करने और रख-रखाव की गुंजाईश है अथवा नहीं |

Software Process Model-

यह software process का एक साधारण प्रतिनिधित्व है | हम तीन सबसे ज्यादा प्रयुक्त होने वाले मॉडल के बारे में बात करेंगे -

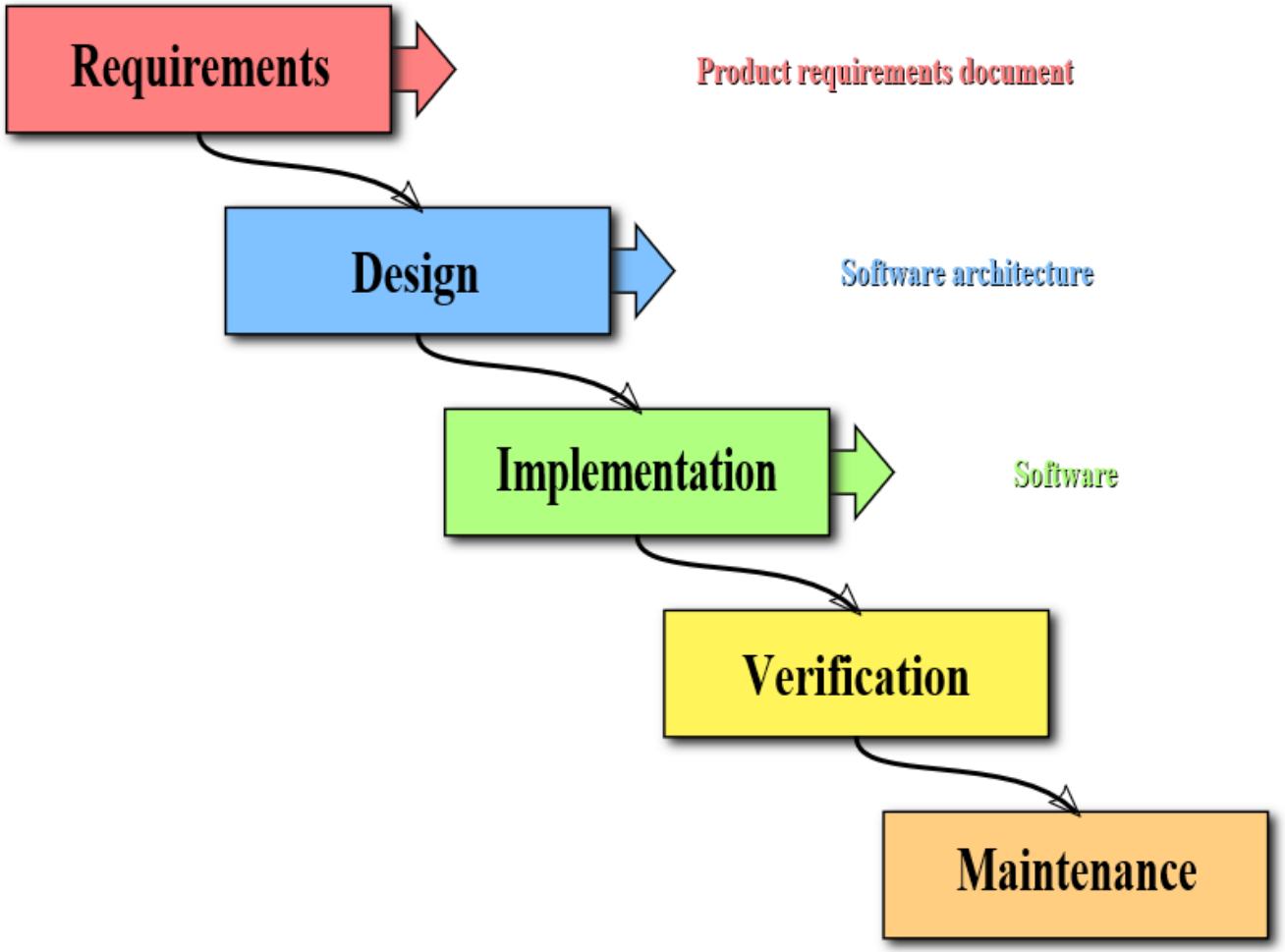
1. वाटर-फाल मॉडल (The Waterfall Model)
2. विकासवादी मॉडल (The Evolutionary Model)
3. घटक आधारित मॉडल (The Component-based Model)

The Waterfall Model- यह मॉडल software development का एक क्रमिक और रैखिक(linear) दृष्टिकोण है जहां सॉफ्टवेयर एक चरण से दूसरे चरण में व्यवस्थित रूप से नीचे की ओर विकसित होता है यह मॉडल २ विभिन्न चरणों में विभाजित है और एक चरण का आउटपुट दूसरे चरण के लिए इनपुट का काम करता है |

इस मॉडल की मूलभूत विकास गतिविधियाँ नीचे सूचीबद्ध हैं।

चरणों में कोई overlapping नहीं है |

1. आवश्यकता विनिर्देश |
2. सिस्टम डिजाईन और विश्लेषण |
3. कार्यान्वयन (implementation) और इकाई परीक्षण |
4. एकीकरण और सिस्टम परीक्षण |
5. सञ्चालन और अनुरक्षण |



The Waterfall Model

यह मॉडल उन प्रोजेक्ट्स के लिए पूर्णतया उपयुक्त है जिनमे -

- Requirements स्पष्ट रूप से निर्धारित हों.
- प्रत्येक चरण को स्पष्ट रूप से रखा गया है और क्रमिक रूप से अगले चरण में बढ़ रहा है।
- प्रत्येक चरण पूरी तरह से पूरा हो जाता है और अगले चरण के इनपुट के रूप में अपना आउटपुट उपलब्ध कराता है।

Waterfall Model के लाभ -

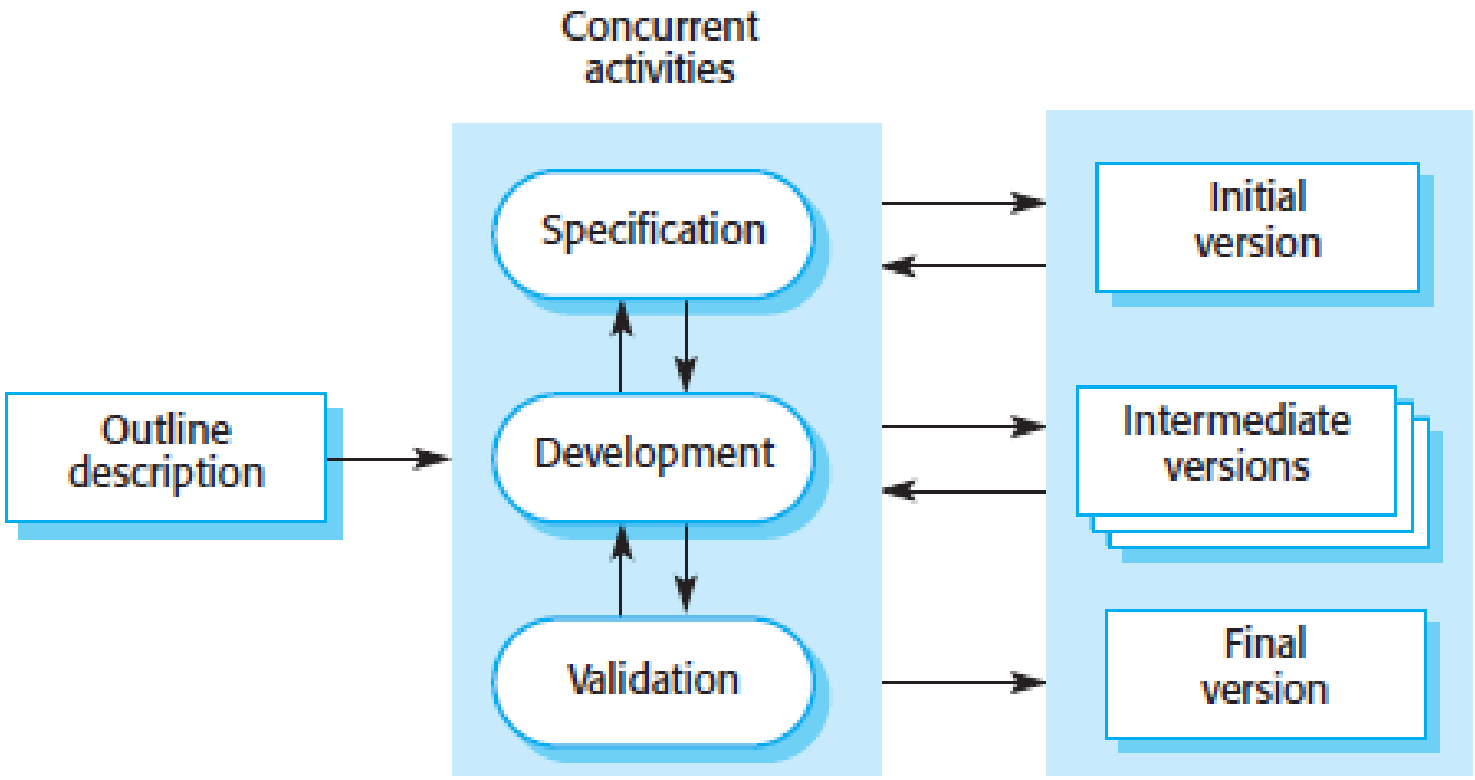
- **विभागीयकरण-** यह पूरी प्रक्रिया को विभागों में विभाजित करता है। यह प्रत्येक विभाग के लिए अलग-अलग कार्यक्रम और समय-सीमा तय करता है।
- मॉडल को समझना आसान है।
- मॉडल का प्रबंधन करना आसान है।
- जटिल मॉडल नहीं।

Waterfall Model की कमियां -

- समय और लागत का कोई अनुमान नहीं।
- परिवर्तनों को शामिल करना मुश्किल है।
- जटिल प्रणालियों के लिए नहीं।

The Evolutionary Model-

यह एक तेज़ रफ़्तार का सॉफ्टवेयर डेवलपमेंट मॉडल है जहां एक प्रारंभिक सॉफ्टवेयर इम्प्लीमेंटेशन बहुत संक्षिप्त विनिर्देशों से तेजी से विकसित होता है, जो तब सॉफ्टवेयर के उपयोगकर्ताओं के मूल्यांकन के अनुसार पुनरावृत्त रूप से संशोधित होता है। इस सॉफ़्टवेयर प्रक्रिया में, एक प्रारंभिक सॉफ़्टवेयर इम्प्लीमेंटेशन विकसित और उपयोगकर्ताओं को दिया या दिखाया जाता है। उपयोगकर्ता इस सॉफ्टवेयर इम्प्लीमेंटेशन के साथ काम करता है और उस पर टिप्पणी और प्रतिक्रिया देता है जिसके आधार पर प्रणाली को परिष्कृत किया जाता है और इसमें परिवर्तन शामिल किए जाते हैं। यह प्रक्रिया तब तक दोहराई जाती है जब तक कि एक पूर्ण-विकसित प्रणाली विकसित नहीं हो जाती। सॉफ्टवेयर विनिर्देश, विकास और सत्यापन और परीक्षण गतिविधियाँ गतिविधियों में व्यापक रूप से सक्रिय प्रतिक्रिया हैं।



The Evolutionary Model

Evolutionary Model के लाभ -

- वास्तव में पूर्ण प्रणाली विकसित करने से पहले संभावित निवेशकों के लिए अवधारणा को प्रदर्शित करना उपयोगी है।
- यह विफलता के जोखिम को कम करता है, क्योंकि संभावित जोखिमों को सॉफ्टवेयर इम्प्लीमेंटेशन के निरंतर मूल्यांकन द्वारा जल्दी पहचाना जा सकता है और सुधारात्मक कदम उठाए जा सकते हैं।
- विकास, विकास टीम और ग्राहक के संयुक्त योगदान से होता है।
- उपयोगकर्ता को अंतिम उत्पाद के कार्य का उचित विचार मिलता है क्योंकि सिस्टम का एक कार्यशील मॉडल प्रदान किया जाता है।
- उपयोगकर्ता प्रतिक्रिया बेहतर समाधान के लिए अग्रणी प्रारंभिक चरण में उपलब्ध है।

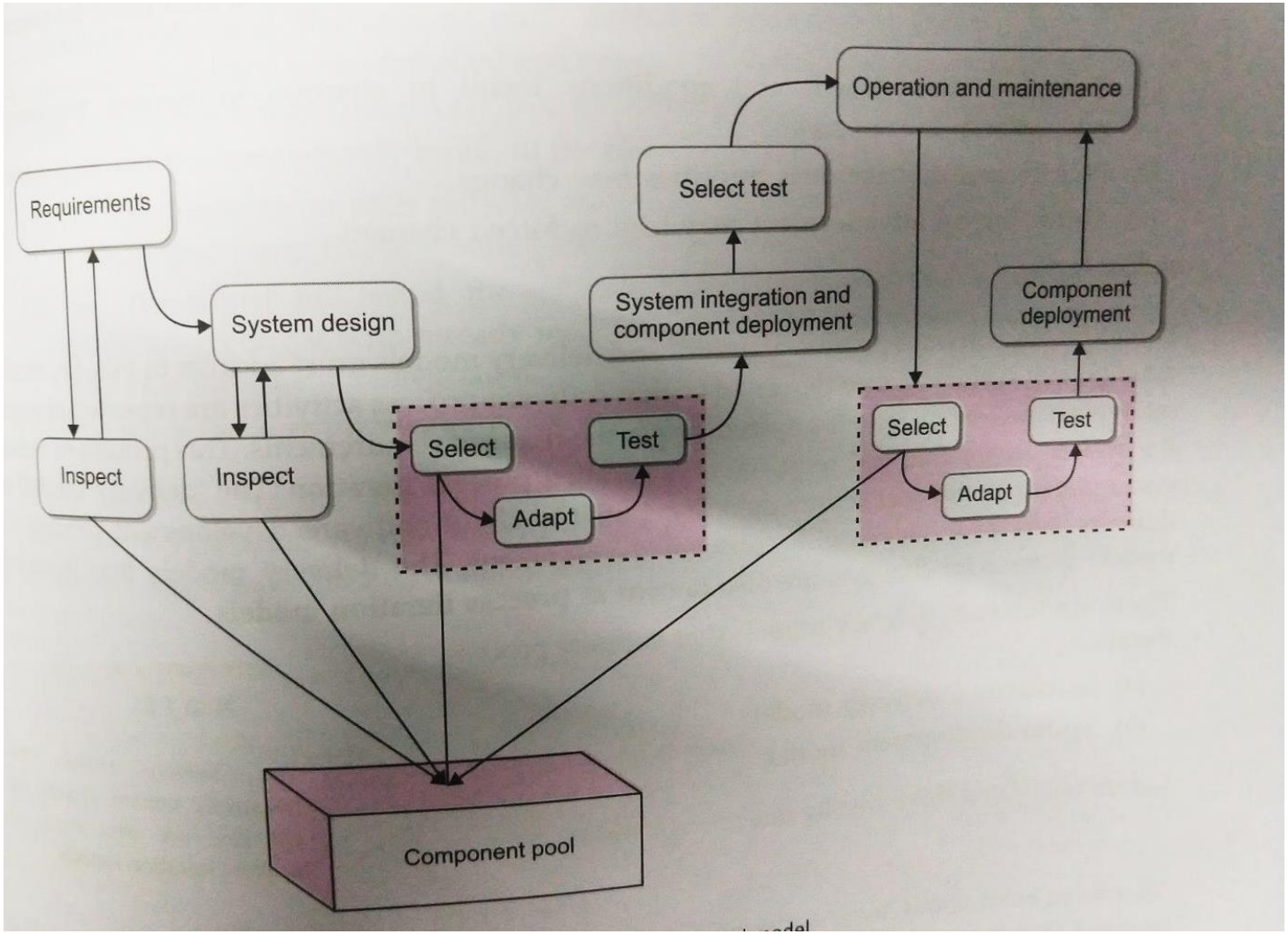
Evolutionary Model की कमियां -

- यदि सुझाए गए परिवर्तन भी हो सकते हैं, तो यह विकास टीम की लय को बिगाड़ सकता है।
- यदि उपयोगकर्ता मांग करता है तो इसमें बदलाव हो सकता है, यह सिस्टम की जटिलता को बढ़ा सकता है।
- बार-बार किए गए बदलावों से विकास की लागत बढ़ सकती है और यह सॉफ्टवेयर सिस्टम के बजट को पटरी से उतार सकता है।

Component-Based Model-

यह एक सॉफ्टवेयर एलिमेंट है जो एक सॉफ्टवेयर मॉडल की पुष्टि करता है और एक रचना मानक के अनुसार संशोधन के बिना, स्वतंत्र रूप से तैनात और तैयार किया जा सकता है। सॉफ़्टवेयर विकसित करते समय, आप महसूस कर सकते हैं कि एक निश्चित कार्यक्षमता आपके द्वारा पहले विकसित किए गए सॉफ़्टवेयर घटक के समान है। यह मॉडल इस बहुत ही विचार पर आधारित है, यानी संभव और संभव होने पर वर्तमान सॉफ़्टवेयर विकास में मौजूदा सॉफ़्टवेयर घटकों को शामिल करने और पुन: उपयोग करने के लिए। इस मॉडल के निम्नलिखित चरण हैं-

- घटक विश्लेषण।
- आवश्यकताएँ संशोधन।
- पुन: उपयोग के साथ सिस्टम डिजाइन।
- विकास और एकीकरण।



The Component-Based Model

Component-based model के लाभ –

- यह विकसित होने वाले सॉफ्टवेयर की मात्रा को कम करता है।
- यह कम लागत और जोखिम के परिणामस्वरूप होता है, यदि पुन: प्रयोज्य घटक उपलब्ध हैं।
- यह सॉफ्टवेयर के तेजी से वितरण की ओर भी ले जाता है।

Component-based model की हानियाँ -

- कभी-कभी आवश्यकताओं से समझौता किया जाता है।
- यह एक ऐसी प्रणाली को जन्म दे सकता है जो उपयोगकर्ताओं की वास्तविक जरूरतों को पूरा नहीं करती है।

Delivery Models-

आधुनिक युग के सॉफ्टवेयर सिस्टम की डिलीवरी एक बार और सभी के लिए अंतिम डिलीवरी हो सकती है। परिवर्तन होते हैं और अपरिहार्य होते हैं। इस प्रकार, सॉफ्टवेयर सिस्टम की डिलीवरी में बदलावों को शामिल करना चाहिए। इसलिए, आधुनिक डेवलेपमेंट सिस्टम को इतना डिज़ाइन किया गया है कि सॉफ़्टवेयर प्रक्रिया गतिविधियाँ नियमित अंतराल पर दोहराई जाती हैं या सिस्टम को परिवर्तित आवश्यकताओं के अनुसार फिर से तैयार करती हैं। परिवर्तित आवश्यकताओं के अनुसार प्रणाली को फिर से तैयार करने की इस प्रक्रिया को प्रक्रिया पुनरावृत्ति कहा जाता है। वितरण मॉडल जो बदलावों के लिए पुनरावृति प्रणाली को अद्यतन करते हैं, उन्हें प्रक्रिया पुनरावृत्ति मॉडल भी कहा जाता है। दो ऐसे वितरण मॉडल हैं जो प्रक्रिया पुनरावृत्ति का समर्थन करते हैं-

- Incremental delivery model.
- Spiral development model.

Incremental delivery model-

यह मॉडल एक विकास और वितरण मॉडल है जो दो सॉफ्टवेयर प्रोसेस मॉडल- वाटरफॉल मॉडल और इवोल्यूशनरी मॉडल की ताकत को जोड़ती है। इस मॉडल में मुख्य रूप से ये कार्य-चरण हैं-

1. वितरित की जाने वाली समग्र सेवाएं निर्धारित करें।
2. सॉफ्टवेयर वृद्धिशील बनाता है यह निर्धारित करें- सॉफ़्टवेयर वृद्धि या सॉफ़्टवेयर बिल्ड का निर्धारण करने के लिए निम्न तंत्र किया जाता है, जिसका अर्थ है कि एक वितरण-चरण में वितरित की जाने वाली सेवाओं का सेट।

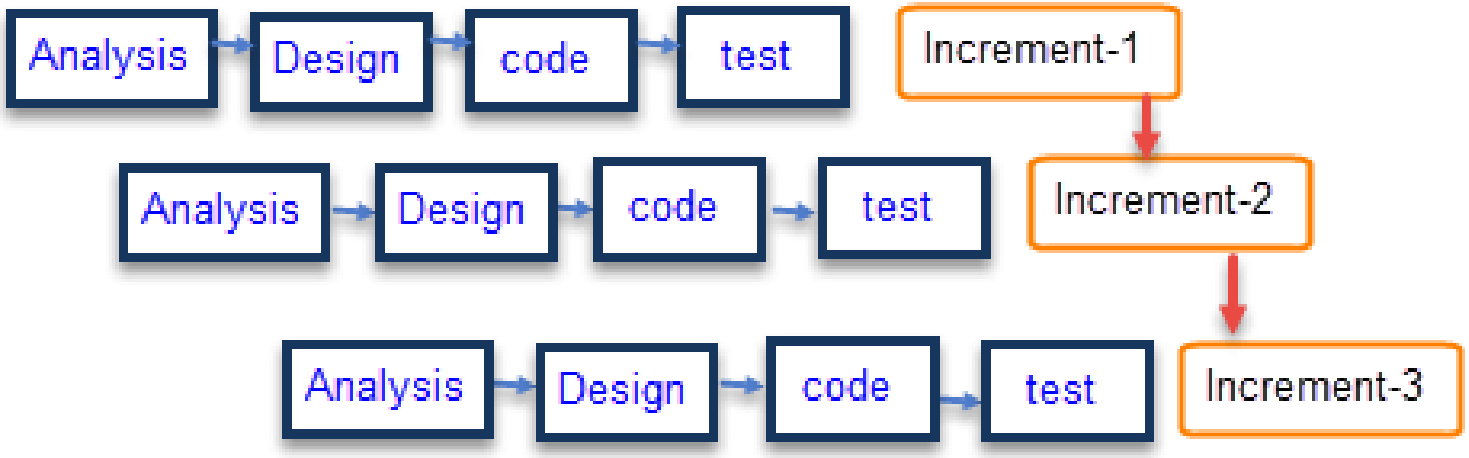
I. ग्राहक को इन सेवाओं को प्राथमिकताओं के लिए सबसे महत्वपूर्ण सेवाओं के रूप में कम से कम महत्वपूर्ण सेवाओं के लिए कहा जाता है।.
II. प्राथमिकता स्तरों के अनुसार, सॉफ़्टवेयर बिल्ड की संख्या को परिभाषित किया जाता है, जहां प्रत्येक सॉफ़्टवेयर इंक्रीमेंट प्रत्येक प्राथमिकता स्तर सेवा प्रदान करता है। वेतन वृद्धि के लिए सेवाओं का आवंटन सेवा की प्राथमिकता पर निर्भर करता है।

3. **प्रत्येक software incremental build का विकास और वितरण** - एक बार जब सिस्टम **incremental build** की पहचान की गई और प्राथमिकता दी गई, तो प्रत्येक सॉफ्टवेयर वेतन वृद्धि के लिए निम्नलिखित गतिविधियों को उनकी प्राथमिकता के क्रम में किया जाता है।

i. प्रत्येक सॉफ़्टवेयर development की आवश्यकताओं को विवरणों में परिभाषित किया गया है, और उस development को सर्वोत्तम अनुकूल सॉफ़्टवेयर प्रोसेस का उपयोग करके विकसित किया गया है।
ii. एक बार विकसित और परीक्षण करने के बाद, सॉफ़्टवेयर बिल्ड क्लाइंट के लिए वितरित और execute किया जाता है। नई development के एकीकरण के बाद काम कर रहा सिस्टम पुनर्प्राप्त है।
iii. अगली प्राथमिकता सॉफ्टवेयर बिल्ड के लिए काम शुरू होता है और पूरी प्रक्रिया को दोहराया जाता है।

Advantages:-

i. यह काम करने वाला सॉफ्टवेयर जल्दी तैयार करता है।
ii. यह अधिक लचीला है, इसमें गुंजाइश और आवश्यकताओं को बदलने के लिए कम खर्च होता है।
iii. छोटे पुनरावृत्ति के दौरान परीक्षण और डिबग करना आसान है।
iv. कुल मिलाकर परियोजना की विफलता का कम जोखिम है | क्योंकि परीक्षण इन्क्रेमेंट्स को जोड़ा जाता है।
v. जोखिम का प्रबंधन करना आसान है क्योंकि जोखिम भरे टुकड़ों की पहचान की जाती है और उन्हें सबसे पहले नियंत्रित किया जाता है



The Incremental delivery Model

कमियां:

1. सिस्टम के साथ प्रत्येक सॉफ्टवेयर इंक्रीमेंट को जोड़ने के बाद, सिस्टम को पूरी तरह से काम करने के लिए एकीकरण परीक्षण किया जाता है, जिससे परीक्षण भार बढ़ता है।
2. हर सॉफ्टवेयर इंक्रीमेंट को विकसित किया जाता है। ऐसा प्रत्येक चरण कठोर है और एक दूसरे को overlap नहीं करते हैं।
3. यदि सभी आवश्यकताओं को स्पष्ट रूप से पहचाना नहीं गया है और उपयुक्त प्राथमिकता नहीं दी गई है, तो इससे समग्र system architecture में बड़ी समस्याएं हो सकती हैं।

The Spiral Model-

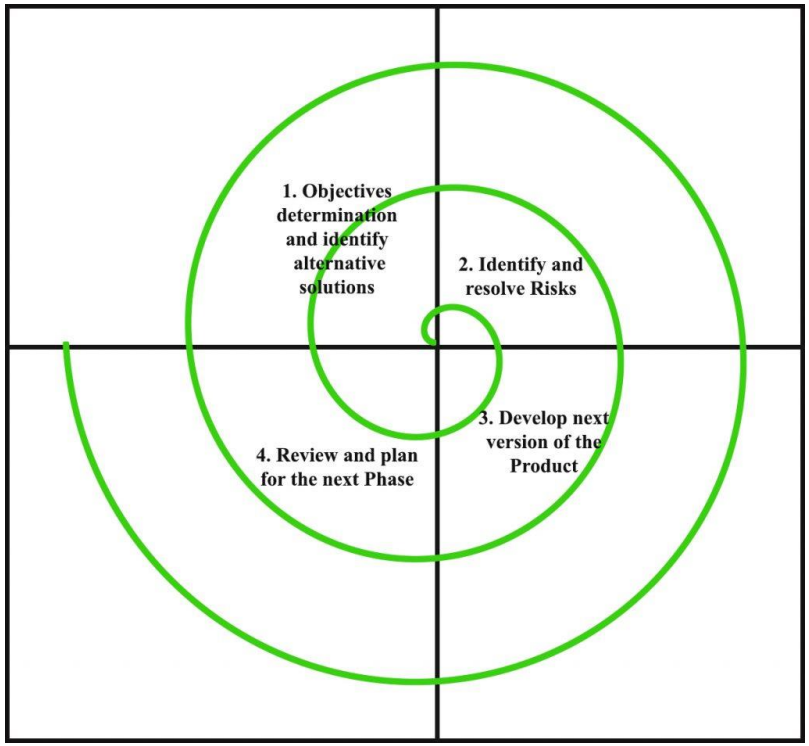
इस model मे, software को इसके प्रोटोटाइप से लेकर इसके incremental releases के एक क्रम में विकसित किया जाता है | जैसे की इस model का नाम है spiral - इस मॉडल में होने वाली गतिविधियों का क्रम एक गतिविधि से दूसरी गतिविधि के साथ होता है | इस मॉडल की गतिविधियों के अनुक्रम को एक spiral के माध्यम से दर्शाया जाता है, जहां प्रत्येक लूप सॉफ्टवेयर प्रक्रिया के एक चरण से मेल खाती है। इस प्रकार, अंतरतम लूप (innermost loop) का संबंध सिस्टम की व्यावहारिकता, आवश्यकताओं की परिभाषा के साथ अगला लूप, सिस्टम के डिजाइन के साथ अगला लूप आदि से हो सकता है।

Spiral मॉडल के प्रत्येक चरण के दौरान निम्नलिखित गतिविधियां की जाती हैं-

1. **पहला चतुर्थांश (उद्देश्य सेटिंग)** - इस चरण में, चरणों के उद्देश्यों को निर्धारित किया जाता है और संबंधित जोखिमों की जांच की जाती है.
2. **दूसरा चतुर्थांश(जोखिम मूल्याङ्कन और कमी)** - इस चरण में, प्रत्येक पहचाने गए जोखिम के लिए एक विस्तृत विश्लेषण किया जाता है। यह चरण जोखिम में कमी के लिए भी जिम्मेदार है, इसलिए जहां भी संभव हो जोखिम में कमी के उपाय किए जाते हैं।

3. **तृतीय चतुर्थांश (विकास और मान्यता) -** यह चरण पहचान किए गए जोखिमों को हल करने के बाद उत्पाद के अगले स्तर के थिरता और सत्यापन के लिए है।

4. **चौथा चतुर्थांश (समीक्षा और योजना) -** इस चरण के दौरान, अब तक प्राप्त परिणाम ग्राहक के साथ समीक्षा हैं। और spiral के चारों ओर अगले पुनरावृत्ति की योजना भी बनती है।



The Spiral Model

spiral मॉडल बड़े, जटिल और महंगे सॉफ्टवेयर सिस्टम के विकास के लिए फायदेमंद है। यह उन प्रणालियों के लिए भी उपयुक्त है जहां प्रत्येक विकासवादी स्तर पर जोखिमों की प्रतिक्रिया की आवश्यकता होती है।

- कृपया हमारे ब्लॉग को फॉलो करिए और youtube channel को subscribe करिए | ताकि आपको और सारे chapters मिल सकें |

www.pythontrends.wordpress.com