Fundação CECIERJ - Vice-Presidência de Educação Superior a Distância

Curso de Tecnologia em Sistemas de Computação
Disciplina: Engenharia de Software
AD2 2° Semestre de 2017

1) O conhecimento aplicado no gerenciamento de projetos pode ser dividido em várias disciplinas. As disciplinas centrais" (gerenciamento do escopo, tempo e custo) dominam a etapa de planejamento no gerenciamento de projetos. Qual é o papel de cada uma delas? Como a dimensão da qualidade pode afetar seu equilíbrio? (valor 2,0 pontos).

A disciplina de gerenciamento de escopo tem como objetivo delimitar o trabalho que deve ser feito no projeto. Ela é fundamental para determinar as atividades que deverão ser realizadas, que por sua vez comporão o principal elemento de interesse na disciplina de gerenciamento de tempo. Estas atividades também formarão a base para a identificação dos recursos humanos e materiais que são necessários para a condução do projeto. Estes recursos, e seu uso ao longo do tempo, formarão a base para o gerenciamento dos custos de execução do projeto. A disciplina de qualidade pode exigir que novas atividades sejam incluídas no cronograma e no plano de custos de modo a garantir que os critérios de qualidade desejados sejam atingidos.

2) No material de aula falamos sobre diagramas de estados. O que este diagrama consegue representar? De que forma ele pode ser usado no processo de desenvolvimento? (valor: 2,0 pontos)

Um diagrama de estados representa a dinâmica de execução de cada classe do projeto de software. Ele representa os estados que a classe terá ao longo do seu ciclo de vida e as transições que são permitidas entre estes estados. Os estados são representados por retângulos de bordas arredondadas, enquanto as transições são representadas por setas que conectam estes estados. Os estados da classe, assim como as suas transições, devem estar representados no diagrama de casos de uso e nas descrições textuais dos casos de uso, de modo que os requisitos do sistema e seu projeto estejam compatíveis.

3) Atividade de pesquisa: O que são métodos ágeis? Que características estes métodos apresentam que os diferenciam de modelos de desenvolvimento convencionais? Dê um exemplo de um método ágil e aponte que características este método apresenta. (valor: 2,0 pontos)

Métodos ágeis, Desenvolvimento ágil ou simplesmente *Agile*, engloba um conjunto de metodologias utilizadas no desenvolvimento de software que propõem, essencialmente, menor foco no processo e maior foco em pessoas, interações entre pessoas, produto, colaboração com cliente e respostas a mudanças.

Agile engloba diversas metodologias (também chamadas de práticas) como, por exemplo, *Adaptive Software Development (ASD) [3], Agile Unified Process (AUP) [4], Crystal Methods [5], Dynamic Systems Development Methodology (DSDM) [6], eXtreme Programming (XP) [7], Feature Driven Development (FDD) [8], Kanban [9], Lean Software Development [10], Scrum [11], Scrumban [Ladas 2009 and several variant methods of agile]*.

Como características de XP podem ser citadas grande foco no código fonte, intensidade nos testes e simplicidade nas soluções.

Referências

[1] Al-Zewairi1, M., Biltawi, M., Etaiwi, W. and Shaout, A. (2017) Agile Software Development Methodologies: Survey of Surveys. Journal of Computer and Communications, 5, 74-97. https://doi.org/10.4236/jcc.2017.55007

[2] Williams, L. and Tomayko, J. (9 sep 2002). Agile Software Development. Computer Science Education, v. 12, n. 3, p. 167–168.

[3] Highsmith, J. (1997) Messy, Exciting, and Anxiety-Ridden: Adaptive Software Development. American Programmer, 10, 23-29.

[4] Ambler, S. (2006) The Agile Unified Process (AUP) http://www.ambysoft.com/unifiedprocess/agileUP.html

[5] Cockburn, A. (2004) Crystal Clear a Human-Powered Methodology for Small Teams. Addison-Wesley, Reading.

[6] Tuffs, D., Stapleton, J., West, D. and Eason, Z. (1999) Inter-Operability of DSDM with the Rational Unified Process. DS DM Consortium, 1, 1-29.

[7] Anderson, A., Beattie, R. and Beck, K. (1998) Chrysler Goes to Extremes. Mp Disruted Couters, 24-28.

[8] Coad, P., de Luca, J. and Lefebvre, E. (1999) Java Modeling Color with Uml: Enter- prise Components and Process with Cdrom. Prentice Hall, Upper Saddle River.

[9] Ladas, C. (2009) Scrumban-Essays on Kanban Systems for Lean Software Develop- ment. Modus Cooperandi Press, Seattle.

[10] Poppendieck, M. and Poppendieck, T. (2003) Lean Software Development: An Agile Toolkit. Addison-Wesley, Boston.

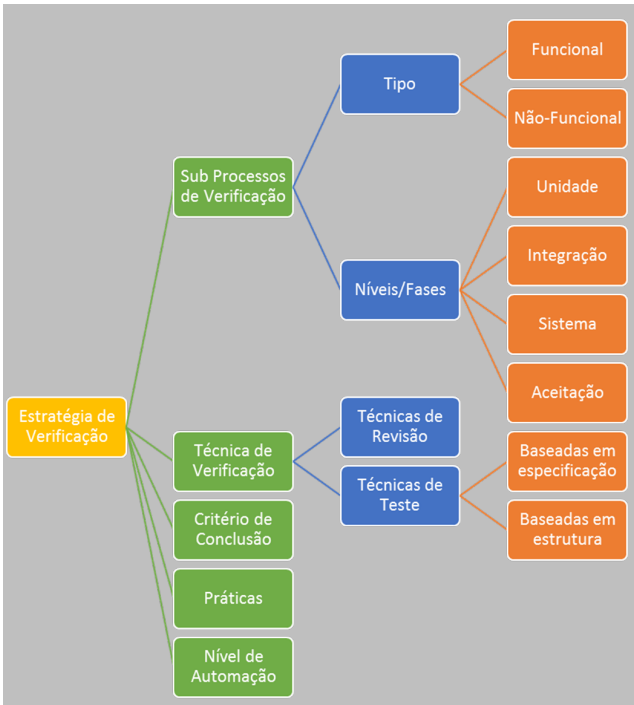
[11] Schwaber, K. (1996) Controlled Chaos: Living on the Edge.

4) Explique o que é teste funcional. Apresente pelo menos dois exemplos de critérios de teste que podem ser utilizados para projetar casos de teste para este tipo de teste. Utilize os critérios exemplificados para projetar os casos de teste para o seguinte requisito: (valor: 2,0 pontos)

Em uma indústria química o controle da temperatura e pressão de um determinado processo depende de dois fatores: a temperatura de ebulição do produto sendo processado e de um coeficiente Z. As entradas do programa de controle são a temperatura de ebulição (te) do produto em graus Celsius limitada ao intervalo [90,200] e o coeficiente Z limitado ao intervalo [0.2, 2.4]. O sistema de controle deve suspender o processo se a temperatura ou a pressão atingirem valores críticos.

O cálculo do valor crítico da temperatura (tk) é obtido pela fórmula: tk = te*3.
O valor crítico da pressão (pk) pela fórmula:
pk = tk * Z, quando te for menor que 110;

Como pode ser visto na figura a seguir, Teste Funcional é um Tipo de teste que tem a finalidade de avaliar a *corretude* do software em relação a seus requisitos funcionais, suas funcionalidades. Os critérios de teste mais conhecidos para testes funcionais são Particionamento em Classes de equivalência e Análise de Valor Limite. Normalmente, esses critérios são utilizados em conjunto.



Exemplo

Primeiro, determinar os valores de entrada a serem utilizados através da análise de valor limite a particionamento em classes de equivalência:

TE[inválidos] = 89, 201 | TE[válidos] = 90, 91, 199, 200

Z[inválidos] = 0.2, 2.5 | Z[válidos] = 0.2, 0.3, 2.3, 2.4

Depois, gerar casos de testes com valores inválidos

CT[TE=89; Z=0.2] = Inválido

CT[TE=201; Z=0.3] = Inválido

CT[TE=90; Z=0.1] = Inválido

CT[TE=91; Z=2.5] = Inválido

Gerar casos de testes válidos

CT[TE=90; Z=0.2] = Válido [TK=270; PK=54]

CT[TE=91; Z=0.3] = Válido [TK=273; PK=81.9]

CT[TE=199; Z=2.3] = Válido [TK=597; PK=457,7]

CT[TE=200; Z=2.4] = Válido [TK=600; PK=480]

Gerar casos de testes para terceiro e quarto parâmetro quando: CT[TE=90; Z=0.2] = Válido [TK=270; PK=54]

CT[TK=269;PK=53] = Não interrompe

CT[TK=270;PK=53] = Interrompe pela temperatura

CT[TK=271;PK=53] = Interrompe pela temperatura

CT[TK=269;PK=54] = Interrompe pela pressão

CT[TK=269;PK=55] = Não interrompe pela pressão

Seguir mesmo raciocínio para gerar os casos de testes para as outras possibilidades de TE e Z

5) Explique o que você entende pelos termos defeito, falta, erro e falha. Explique a diferença entre eles, e cite pelo menos uma taxonomia de defeitos pode ser usada para classificar defeitos. Dê um exemplo de como defeitos podem ser identificados. (valor 2,0 pontos)

Defeito (defect): o termo defeito pode ser utilizado de maneira genérica para designar faltas ou falhas (LYU, 1996)

Falta (fault): um passo, processo ou definição de dados incorretos em um programa de computador (IEEE 610, 1990).

Erro (error): Uma ação humana que produz um resultado incorreto.

Falha (failure): é a inabilidade do sistema ou componente de executar suas funções da maneira como foi especificado. Ou seja, é o comportamento operacional do software diferente do esperado pelo usuário. Pode se dizer também que é a manifestação de uma falta (fault) (IEEE 610, 1990). Uma falta pode gerar diversas falhas e diversas faltas podem gerar uma falha (HUANG e LIN, 2010).

Defeitos podem ser identificados através de atividades de revisões, como as inspeções.

Referências

LYU, M. R., (1996). “Handbook of Software Reliability Engineering”, McGraw-Hill.

IEEE Standard 610-1990: IEEE Standard Glossary of Software Engineering Terminology, IEEE Press

HUANG, C. Y., LIN, C.-T. (2010). “Analysis of software reliability modeling considering testing compression factor and failure-to-fault relationship”, IEEE Transactions on Computers, 59, 283 - 288.