Na aula de 1/8/2012, nós realizamos uma introdução à disciplina, comentando sobre a programação e a bibliografia.

Recomendamos a obra dos três amigos (Ivar Jacobson, Grady Booch, James E. Rumbaugh: The unified software development process - the complete guide to the unified process from the original designers. Addison-Wesley, 1999) e outras duas obras como fontes primárias de estudos (Philippe Kruchten: The Rational Unified Process-An Introduction. Addison-Wesley, 3rd edition, 2003 e Martin Fowler: UML Distilled: A Brief Guide to the Standard Object Modeling Language, Addison-Wesley, 3rd edition, 2004). Estes livros fazem parte do acervo da biblioteca da Unifor e os dois últimos possuem versões em português. Outros papers, artigos, tutoriais, slides e fontes serão usados como complementação do estudo. Para estudr mais sobre requisitos e o modelo de caso de uso recomendamos COCKBURN, Alistair. Escrevendo Casos de Uso Eficazes. Bookman, 2004.

Foi realizada uma discussão sobre o conceito de Processo de Software e Modelo de Processo de Software. Também discutimos o que diferencia as atividades de Requisitos, Análise e Projeto, concentrando-se em comparar os objetivos, artefatos e papeis envolvidos com essas disciplinas.

Em resumo, a discussão em grupo concordou que a disciplina de requisitos preocupa-se em levantar as funcionalidades do software, definindo e priorizando o que será construído, mostrando que restrições se aplicam ao software e o que está fora de escopo, firmando um acordo formal (baseline) ou tácito com o cliente / usuário.

Já a disciplina de análise ainda preocupa-se com o que será construído, mas com duas diferenças fundamentais: (1) a especificação é construída na linguagem dos desenvolvedores (diagramas e modelos UML – por exemplo) e (2) ocorrem refinamentos, eliminando-se ambiguidades, redundâncias, inconsistências, etc.

Durante o projeto, é definido como o sistema será implementado (atividade seguinte), construindo-se a arquitetura da aplicação e o design como um todo, de acordo com as restrições não funcionais (por exemplo, confiabilidade, manutenibilidade, tolerância a falhas, disponibilidade, recuperabilidade, etc) e com o ambiente físico de execução (plataforma, ambientes, protocolos, linguagens, frameworks, legados, etc).

Por fim, deixamos claro nesta aula que o curso terá abordagens prática e teórica. Ambas farão parte da pontuação das notas de cada aluno, trabalhando de forma individual e em equipe.

Para a próxima aula, discutiremos mais sobre Processo, Modelos de Processo e detalharemos as disciplinas, atividades, papéis e artefatos que importam para o nosso escopo de estudo. Vamos entender quais as diferenças entre as disciplinas de Requisitos, Análise e Projeto. Par a melhor acompanhamento da aula, sugiro a leitura prévia do capítulo de processo da obra de Ian Sommerville, Software Engineering, 8 edition (capítulo 4 – em edições diferentes o capítulo muda, ver índice).

Na aula de 3/8/2012, nós apresentamos alguns conceitos gerais aplicáveis a todos os Modelos de Processo de Software.

Começamos definindo Software Produto e mostrando que o mesmo é composto pelos artefatos gerados em todas as atividades de um Processo. Depois apresentamos as razões para se efetuar manutenção em software, destacando exemplos de erro, evolução, prevenção e demandas advindas do ambiente.

A seguir mostramos a curvas de erros do software, destacando que, estatisticamente, todos os softwares possuem erros e que cada manutenção acrescenta novos erros ao software. Ao longo de todo o ciclo de vida, o software tende a chegar a um estágio de difícil manutenção, quando podem entrar em cena necessárias ações de Reengenharia ou simplesmente se descomissionar (aposentar) o software.

Enfocamos também um enfoque sobre custo e tempo gasto em cada atividade do Processo e percebemos que a manutenção (quando um software entra em operação) tem a maior fatia do bolo. Daí as técnicas principais de A&P (nosso foco na disciplina) darem muita ênfase à tornar o software o mais extensível possível.

Por fim, fornecemos uma visão geral sobre os modelos de processo clássicos. Frisamos que, na prática, nenhum modelo genérico de processo é usado de forma pura. Em geral, combinam-se características selecionadas de vários modelos na montagem de um processo em uma organização que desenvolve software. Para efeito de estudo, porém, analisamos os modelos em separado.

Assim, o modelo em Cascata tem como características a separação entre fases (atividades) e a progressão linear. Da mesma forma como ocorre em uma cascata de água real, a água desce uma elevação, um degrau (fase) por vez, e não retorna mais ao topo.

O modelo Evolucionário (também chamado por outros autores como Iterativo – não confundir com interativo), em vez de quebrar um projeto em atividades (como o Cascata), quebra-o por funcionalidades. Além disso, em geral há uma atividade de estudo ou prototipação (não confundir com protótipo de interface), antes que as iterações do modelo evolucionário realmente iniciem.

Já o modelo baseado em componentes usa ativos (componentes próprios) ou componentes e frameworks de terceiros para a montagem de aplicações, gerando esforço de integração entre esses componentes e, por vezes, renegociando requisitos. Em contrapartida, oferece ganhos em custo, prazo e qualidade.

Para estudar os conteúdos dessas aulas de processo, reforçamos a leitura (1) do capítulo 2 de Processo da obra de Ian Sommerville, Software Engineering, 9th edition e (2) do capítulo 2 da obra de Martin Fowler, UML Distilled - UML essencial é o título em português.

Na aula de 6/8/2012, iniciamos o estudo dos Modelos Clássicos Processo de software revisando os conceitos relacionados a Processo, suas as quatro macros atividades principais (especificação, desenvolvimento, validação e evolução) e revisitando a conceituação de verificação e validação discutido na aula anterior.

Depois iniciamos o estudo do Modelo Cascata (Waterfall). Este modelo caracteriza-se pela progressão linear das suas atividades, de tal forma que uma atividade só inicia após a anterior haver sido completada. Tampouco volta-se a uma atividade anterior quando a mesmo tiver sido completada.

E isso pode ser um problema, pois demonstramos que os requisitos de sistemas de negócios (comércio, finanças, bancos, serviços, etc) são muito voláteis e mudam constantemente. Logo, pode haver problemas se esse modelo for aplicado, em um ciclo longo, a um sistema cujos requisitos podem mudar ao longo do tempo. Há ainda problemas inerentes á própria natureza da especificação de requisitos, como o fato da mesma poder, com certa frequência, ser ambígua, redundante, inconsistente e incompleta.

Outra característica do Modelo Cascata é que os testes de integração e de sistema são realizados mais para o final do ciclo, potencialmente aumentando a sua complexidade e os custos dessa atividade.

Depois iniciamos o estudo das características do modelo de processo Evolucionário, fazendo uma comparação com o Modelo em Cascata, principalmente no tocante a como os dois modelos tratam a complexidade do processo de desenvolvimento de software (Cascata divide o processo em atividades e o Evolucionário em subconjuntos de requisitos).

Ficou claro que, seja qual for a estratégia adotada, no modelo evolucionário há uma ação inicial de exploração de requisitos para fechar escopo, restrições e elementos relacionados ao desenvolvimento em geral. A partir daí, o desenvolvimento evolui com liberação de versões intermediárias do sistema (de acordo com os requisitos priorizados), até que se obtenha a versão final.

Feedback (com testes realizados a partir do início e por todo o ciclo do desenvolvimento), melhores oportunidades para provisões de tempo e custo, bem como melhor motivação das equipes envolvidas (desenvolvimento e usuários) foram pontos fortes destacados do Modelo Evolucionário.

Como desvantagens (pontos de preocupação) do Modelo Evolutivo, destacamos a possibilidade de podermos ter uma solução menos estruturada (em relação ao Modelo Cascata), pois não se possui todos os requisitos em mãos na hora de definir a arquitetura do software, bem como a ausência de marcos claros, também em comparação com o Modelo Cascata.

O Modelo Evolucionário prevê duas formas de iniciar o processo de desenvolvimento: através de throw away prototypes (protótipos descartáveis para entendimento dos requisitos) ou exploratory development (construção de um protótipo inicial com requisitos em alto nível, evoluindo com detalhamentos e incorporação de novas features até a versão final do sistema).

Para estudar os conteúdos dessas aulas de processo, reforçamos a leitura (1) do capítulo 2 de Processo da obra de Ian Sommerville, Software Engineering, 9th edition, publicada no Unifor On Line e (2) do capítulo 2 da obra de Martin Fowler, UML Distilled - UML essencial é o título em português.

A aula de 8/8/2012 focou no modelo de processo baseado em componentes e no desenvolvimento incremental.

Iniciamos fazendo um resumo dos Modelos já estudados, Cascata e Evolucionário, mostrando aplicações e desvantagens de cada um. Depois mostramos uma situação real no qual os objetivos institucionais e do projeto estavam sendo distorcidos, em uma construção de artefatos que poderiam, com vantagens, ser adquiridos. Dessa forma, através desse exemplo, introduzimos o conceito de componente (diferente de framework) e da Engenharia de Software baseada em componentes, como uma forma de se ganhar tempo, qualidade, e custo no processo de software.

Em seguida, revisitamos e aprofundamos os principais pontos do Modelo baseado em componentes, especialmente o conceito e tipos de componentes COTS (de pequenos itens a subsistemas completos). A análise “Make/Buy/Reuse”, a adequação de requisitos e a preocupação em integrar componentes que muitas vezes não foram projetados para trabalhar juntos também foram discutidas.

Neste ponto fizemos um exercício, baseado em uma questão de uma avaliação de semestre anterior e discutimos a resposta.

Terminamos a aula abordamos sobre o conceito e as vantagens da abordagem iterativa (que pode ser aplicada a qualquer modelo de processo genérico), especificamente do desenvolvimento incremental: feedback, melhor provisionamento, teste das principais funcionalidades primeiro e com mais frequência, melhor motivação, etc . Demonstramos (matematicamente) que a complexidade dos sistemas pode ser melhor gerenciada através dessa abordagem, bem como através da componentização.

Para estudar os conteúdos dessas aulas de processo, reforçamos a leitura (1) do capítulo 2 de Processo da obra de Ian Sommerville, Software Engineering, 9th edition, publicada no Unifor On Line e (2) do capítulo 2 da obra de Martin Fowler, UML Distilled - UML essencial é o título em português.

A aula de 10/8/2012 iniciou rediscutindo a questão de dominar a complexidade dos sistemas, usando a abordagem iterativa incremental, quando da construção de softwares.

Ainda sobre o tema complexidade, mostramos como a teoria geral dos sistemas (aplicada tanto a sistemas mecânicos, administrativos, biológicos, etc) ajuda explicar como se pode medir e gerenciar essa complexidade, o que pode ser muito útil na A&P do software produto. Dentre outras características, mostramos que todo sistema: (1) possui um escopo definido (tudo o que faz parte do sistema); (2) é um todo coeso, porém dividido em partes que se relacionam; (3) cada parte ou subsistema, em si, também é um sistema; (4) são os relacionamentos que definem como o sistema se comporta e qual a sua complexidade; (5) todo sistema está imerso em um ambiente, influenciando e sendo influenciado por ele.

A seguir mostramos a abordagem de Iteração em Espiral. Vimos que, a rigor, é um preciosismo diferenciar o ciclo incremental do desenvolvimento espiral, mas destacamos duas facetas deste conceito criado por Barry Boehm, em 1988. O primeiro é a preocupação com riscos e suas características, como probabilidade, severidade (impacto), ações de mitigação (para diminuir a probabilidade de um risco se efetivar) e planos de contingência (para diminuir o impacto do risco, uma vez que ele venha a se efetivar). O segundo é que cada ciclo do espiral pode representar qualquer fase, atividade ou tarefa necessária do projeto.

Mostramos ainda que o ciclo espiral está dividido em quatro setores, cada um com papeis distintos (definição de objetivos; avaliação e controle de riscos; desenvolvimento e validação; planejamento).

Por fim, mostramos a separação de conceitos envolvendo (1) técnicas de modelagem / especificação de sistemas e (2) Metodologias / Modelo de Processo de desenvolvimento. Ao primeiro grupo, pertence a UML e, ao segundo, pertence o RUP e as técnicas ágeis.

Para estudar os conteúdos dessas aulas de processo, reforçamos a leitura (1) do capítulo 2 de Processo da obra de Ian Sommerville, Software Engineering, 9th edition, publicada no Unifor On Line e (2) do capítulo 2 da obra de Martin Fowler, UML Distilled - UML essencial é o título em português.

A aula de 13/8/2012 iniciou o estudo do RUP.

Começamos apresentando as três perspectivas desse modelo de processo: a Dinâmica, com a divisão do desenvolvimento em fases (Iniciação, Elaboração, Construção e Transição) e suas iterações, cada qual com objetivos e características específicos; a Estática, que mostra as atividades (disciplinas) desenvolvidas durante o processo (Modelagem de Negócio, Requisitos, A&P, Implementação, Teste, Gestão de Configuração e Mudança, Gestão de Projeto, Ambiente), mostrando a execução das mesmas través do gráfico das ”baleias”; a Prática, que sugere boas práticas a serem aplicadas durante um projeto e que analisaremos mais detalhadamente ao final do estudo atual.

Destacamos ainda os pilares básicos do RUP: Iterativo e incremental (aspectos já amplamente estudados no nosso curso); Centrado em uma solução de arquitetura (quando se define a estrutura e organização do software, bem como se fornece soluções técnicas para todos os aspectos do projeto); Dirigido por casos de uso, que possuem papel importante em todas as fases, conforme foi citado em aula.

Analisamos em detalhe os marcos da Iniciação e da Construção, bem como o conceito de Arquitetura de Software, Requisitos Funcionais e Não Funcionais (Confiabilidade, Disponibilidade, recuperabilidade, usabilidade, Escalabilidade, Tolerância a Falhas, Segurança, Tempo, Protocolos, dentre outros).

Sobre os objetivos da iniciação, destacando o business case (análise custo / benefício), o levantamento do escopo e detalhamento inicial dos requisitos críticos (média de 5% do esforço de refinamento total), análise dos riscos, sugestão em alto nível de uma arquitetura, planejamento do projeto, ajustes no ambiente.

Em relação à Elaboração, firmamos que os objetivos principais são o refinamento dos requisitos em geral (Casos de Uso, Visão, Glossário, especificação suplementar, etc) e a definição da arquitetura, em conjunto com outras atividades complementares, tais como, o acompanhamento dos riscos, levantamento dos ativos (análise make/buy/reuse), refinamento do planejamento e do ambiente, etc. Apesar do alvo ser 100% de requisitos detalhados, ao final da elaboração, a estatística diz que, em média, 85% dessa meta é atingida, ficando o restante para a fase de Construção

Para as aulas de RUP, sugerimos a leitura dos capítulos 1, 3, 4 e 5 da obra dos três amigos (JACOBSON, Ivar; BOOCH, Grady; RUMBAUGH, James; Unified Software Development Process. Addison Wesley, 1999). Alternativamente, estude os capítulos 1 a 6 da obra de Kruchten (KRUCHTEN, Philippe. The Rational Unified Process: An Introduction, Third Edition. Addison Wesley, 2003). Uma terceira alternativa é um resumo no capítulo 2 do livro (ARLOW, Jim; NEUSTADT, Ila; UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design, second edition . Addison Wesley, 2005).