MC437—Projeto de Sistemas de Informação

Plano de Aulas

1° Semestre 2017

Prof. Adín Ramírez Rivera
adin@ic.unicamp.br

1 Conteúdos prévios

Espera-se que o aluno tenha os seguintes conhecimentos prévios mínimos para o desenvolvimento na disciplina.

1.1 Pré-requisitos

- Programação
  - Ler e escrever código.
  - Realizar o design de algoritmos para resolver problemas.
- Engenheira de software
  - Descrever as etapas principais de ciclo de vida do software
  - Descrever diferenças principais entre metodologias de software.
  - Descrever um fluxo de versionamento.
- Bancos de dados
  - Criar modelos relacionais que refletem um problema.
  - Gerenciar uma base de dados.
- Interface homem-computador (IHC)
  - Descrever métodos de design e avaliação de interfaces.
  - Descrever diferenças principais entre paradigmas em IHC.

1.2 Autoavaliação inicial

Recomenda-se que o aluno faça uma autoavaliação inicial para garantir os conhecimentos prévios, e estudar os que tem deficiência.

2 Programa

A disciplina está dividida em dois projetos: um introdutório (warm-up) e um projeto de maior complexidade. A convenção a seguir na definição dos projetos é: os conteúdos (teóricos ou práticos) da disciplina denotam-se com "C"; os objetivos de aprendizado de cada unidade de conteúdo com "O"; e as atividades a desenvolver para alcançar os objetivos de cada unidade de conteúdo denotam-se com "A"[1] (as atividades avaliativas estão marcadas com "*").

[1] As atividades seguem o formato P-A-P. O **P**rofessor inicia a atividade com instruções ou dando um contexto, os **A**lunos fazem atividades, e depois o **P**rofessor fecha a atividade.

2.1 Warm-up

O warm-up é um exemplo do desenvolvimento de software esperado pelos alunos. Durante cinco (5) aulas se explicará os conteúdos teóricos e práticos para o desenvolvimento do projeto. As avaliações em cada aula são qualitativas e não incidem na nota final.

C1. Introdução. (1 aula: 03/03) Atividades:

A1.1. Apresentação da disciplina: conteúdo, bibliografia, avaliação, atividades práticas, páginas na internet, tecnologias, etc.

A1.2. Apresentação do ambiente de trabalho: gitlab, login, etc.

Sugestões: Familiarizar-se com as ferramentas: git, bash, ssh, screen, postgresql, OpenCV.

C2. Administração da configuração. (1 aula: 08/03)

O2.1. Compreender o papel da administração da configuração.

O2.2. Utilizar várias ferramentas comumente adotadas na implementação de processos ágeis de software: controle de versão (git), integração contínua (jenkins), etc.

Atividades:

A2.1. Descrever os conceitos e atividades relacionadas com a Administração da Configuração (AC).

P. Pede aos alunos escrever o que se lembram é a AC e as atividades dela.

A. Cada aluno escreve os itens que lembra.

P. Recolhe os itens dos alunos.

A2.2. Discussão entre a turma sobre o que os alunos lembram.

P. Dirige a discussão entre a turma.

A. Discutem os itens mencionados.

P. Discute temas chave que não são mencionados por os alunos.

A2.3. (*) Fazer exemplos de configuração de ferramentas de processos ágeis de software.

P. Dá instruções para realizar uma prática de configuração de um repositório e fazer mudanças para mostrar o uso de controle de versões.

A. Fazem a atividade prática.

P. Retroalimentação dos problemas que os alunos encontraram.

Avaliação. O aluno configura o repositório corretamente.

A2.4. Explicar porque os problemas aconteceram na prática anterior, e como AC ajuda a resolvê-los.

P. Inicia a discussão dos problemas que os alunos encontraram na atividade anterior, e dá as perguntas para dirigir a discussão.

A. Respondem e dão exemplos de sua experiencia na atividade anterior e relacionam os conceitos iniciais.

P. Fecha a discussão com os pontos chave dela.

Avaliação. O aluno identifica os problemas comuns da ausência de controle de versões e identifica exemplos de seu uso.

A2.5. (*) Explicar o papel da administração da configuração no desenvolvimento anterior.

P. Inicia a discussão ligando os problemas com a falta de processo.

A. Discutem.

P. Fecha a discussão com os pontos chave e faz uma relação da prática com a teoria.

Avaliação. O aluno identifica os problemas comuns da ausência de controle de versões e identifica exemplos de uso correto da AC.

C3. Técnicas de trabalho em equipe. (0.5 aula: 10/03)

O3.1. Reconhecer a importância de reuniões (*sprint meetings*, *daily standup*, *sprint review meeting*, entre outras) durante cada iteração do processo de software (*sprint*).

O3.2. Compreender os papeis e a distribuição de trabalho em uma equipe de desenvolvimento de software.

Atividades:

A3.1. Explicar o processo de trabalho em equipe em uma metodologia ágil.

P. Pergunta sobre os tipos distintos de reuniões em um processo ágil e os atores principais nelas.

A. Respondem as perguntas.

P. Fecha a discussão com as reuniões importantes e os atores que participam nelas.

A3.2. (*) Entender a diferença entre as reuniões e pontos de sincronização da equipe no processo de desenvolvimento de software.

P. Apresenta perguntas relacionadas com as reuniões e pede aos alunos anotar as respostas que aparecem na apresentação. Apresenta o ciclo de interação e as reuniões envolvidas nele.

A. Tomam notas e tentam encontrar as respostas. Discutem suas respostas.

P. Retroalimentação das respostas dadas, e fecha com os pontos principais.

Avaliação. As respostas dos alunos são consistentes com as definições da metodologia ágil.

A3.3. (*) Explicar os papeis e as responsabilidades dos participantes em um processo ágil.

P. Apresenta uma situação com um grupo de pessoas e um problema a resolver. Dá instruções para os alunos atribuir responsabilidades e tarefas a os participantes.

A. Atribuem responsabilidades e tarefas de acordo à situação.

P. Guia uma discussão entre as propostas dos alunos. Fecha com os pontos principais e as recomendações da metodologia ágil.

Avaliação. A atribuição de papeis e tarefas é consistente com o problema e a metodologia de software.

C4. Definição de requisitos. (0.5 aula: 10/03)

O4.1. Analisar um problema para extrair seus requisitos de acordo com uma metodologia ágil.

Atividades:

A4.1. Dividir um problema em subproblemas.

P. Entrega o enunciado de um problema, e dá as instruções para os alunos para encontrar os subproblemas para resolver o problema maior.

A. Dividem o problema.

P. Dirige a discussão para encontrar os subproblemas distintos e encontrar uma solução.

A4.2. Converter problemas em requisitos de software.

P. Dá instruções para converter um dos problemas encontrados em requisitos formais de software.

A. Realizar o design de um requisito.

P. Discute os resultados encontrados, os problemas e soluções para os requisitos projetados.

Avaliação. A divisão dos problemas é consistente com a definição do requisito de software.

A4.3. (*) Integrar os requisitos com uma metodologia de software ágil.

P. Dá instruções para agregar um requisito a uma ferramenta de controle de versões.

A. Adiciona o requisito de acordo com a metodologia de software ágil.

P. Retroalimentação do trabalho. Pode demostrar uma forma de fazer o trabalho.

Avaliação. Aplica a metodologia corretamente.

C5. Design de software. (1 aula: 15/03)

O5.1. Descrever formalmente os componentes de um requisito funcional de software.

Atividades:

A5.1. (*) Definir os componentes de software a partir de um requisito dado.

P. Dá um requisito definido na atividade anterior, e pede para os alunos definir os componentes de software que resolvem o problema.

A. Realizam o design de um conjunto de componentes para um problema.

P. Discute as soluções propostas e as contrasta entre elas.

Avaliação. O design do componente é consistente e auto-contido, e o aluno pode justificá-lo.

C6. Implementação de código. (1 aula: 17/03)

O6.1. Programar e modificar um componente de software a partir de um design dado utilizando as ferramentas de desenvolvimento definidas.

Atividades:

A6.1. (*) Programar usando as ferramentas definidas.

P. Dá um componente e funcionalidade definida anteriormente para programar usando uma ferramenta de controle de versões.

A. Programam.

P. Retroalimentação do trabalho feito.

Avaliação. Uso correto das boas práticas de programação.

A6.2. (*) Modificar um programa feito e administrar as mudanças.

P. Dá instruções para modificar o componente de software.

A. Programam.

P. Retroalimentação do trabalho feito. Discute boas práticas e erros comuns no trabalho.

Avaliação. Uso correto das boas práticas e ferramentas de controle de versões.

C7. Documentação. (1 aula: 22/03)

O7.1. Comunicar oralmente e por escrito a descrição de métodos (procedimentos), resultados e conclusões.

Atividades:

A7.1. Escrever um registro das atividades de uma reunião.

P. Dá instruções para os alunos fazerem uma reunião breve, e depois escreverem o que foi feito.

A. Fazem a reunião, escrevem um registro das atividades.

P. Revisa os registros e componentes deles.

A7.2. Reconhecer partes relevantes a documentar no desenvolvimento de software.

P. Dá instruções para uma discussão sobre as partes relevantes dos registros elaborados antes.

A. Discutem.

P. Fecha a discussão com as partes principais a considerar. Dá o padrão a utilizar para a disciplina.

Avaliação. O registro é consistente com a metodologia usada e cobre os pontos chave.

A7.3. (*) Documentar código.

P. Mostra exemplos de documentação de código feita pelos alunos durante a aula anterior.

A. Discutem as caraterísticas dos cometários e sua funcionalidade.

P. Fecha a discussão com as boas práticas para documentar software. Dá o padrão a utilizar na disciplina.

Avaliação. O código é compressível por outra pessoa que não é o programador.

A7.4. (*) Documentar eventos e avanços no controlador de versões.

P. Mostra exemplos de documentação de repositórios de versões dos alunos durante a aula anterior.

A. Discutem as caraterísticas dos cometários e sua funcionalidade.

P. Fecha a discussão com as boas práticas para documentar atividades em controle de versões. Dá o padrão a utilizar na disciplina.

Avaliação. As descrições na ferramenta de controle de versões são compressíveis por outras pessoas.

2.2 Projeto de desenvolvimento de software

O projeto de desenvolvimento de software compreende três iterações (*sprints*) de ciclo de software. Cada iteração será desenvolvida en aproximadamente oito (8) aulas. Cada *sprint* tem uma demostração e apresentação ao final dela, mas espera-se que os alunos continuem trabalhando durante as apresentações. Portanto, considera-se fora do *sprint* as demonstrações e espera-se que os alunos tenham um fluxo continuo de trabalho.

Espera-se que os resultados de aprendizado (conteúdos) e as atividades melhorem com cada iteração. O enunciado do problema a desenvolver será entregado com antecedência a os alunos. Os equipes de trabalho serão feitos pelos instrutores.

As avaliações intermédias no *sprint* são qualitativas e não incidem na nota final. A retroalimentação das avaliações debe ser usada para melhorar o processo de desenvolvimento de software e o planejamento feito pelo equipe.

A avaliação final (apresentação) de cada iteração sera de qualificação de acordo com a definição do "Plano de Desenvolvimento". Os equipes deveram criar um *release tag* dentro do *sprint* para a avaliação final. Se não existe um *release tag* válido (dentro do prazo do *sprint*), o último *commit* do *sprint* será utilizado.

Cada *sprint* está composto de seis aulas (mas as aulas de apresentação podem ser usadas para trabalhar no seguinte *sprint*). O calendário dos *sprints* é

S_1: 24/03, 29/03, 31/03, 05/04, 07/04, e **12/04**;

S_2: 28/04, 03/05, 05/05, 10/05, 12/05, e **17/05**;

S_3: 26/05, 31/05, 02/06, 07/06, 09/06, e **14/06**;

onde as datas em negrita denotam o final de cada *sprint* (só *commits* anteriores o desse dia serão usados para avaliação final do *sprint*), e as datas das demonstrações são:

D_1: 19/04 e 26/04;

D_2: 19/05 e 24/05;

D_3: 21/06 e 23/06.

Em seguida, as atividades dos *sprints* são definidas de acordo ao numero de aulas.

C8. Definição de requisitos. (1 aula)

O8.1. Analisar um problema para extrair seus requisitos de acordo com a metodologia ágil e com o tempo da iteração.

Atividades:

A8.1. Analisar e dividir um problema em subproblemas.

P. Dá as instruções para os alunos para encontrar os subproblemas a resolver de acordo com o enunciado do problema.

A. Dividem o problema.

P. Chuva de ideias dos subproblemas encontrados.

A8.2. Converter problemas em requisitos de software.

P. Dá instruções para converter um dos problemas encontrados a requisitos formais de software.

A. Realizam design dos requisitos.

P. Discute os resultados encontrados, os problemas e soluções para os requisitos projetados.

A8.3. (*) Escrever os requisitos formais em uma ferramenta de administração de desenvolvimento de software.

P. Dá instruções para agregar os requisitos a uma ferramenta de controle de versões.

A. Adicionam os requisitos de acordo com a metodologia de software ágil.

P. Retroalimentação do trabalho.

Avaliação. Os requisitos definidos para a iteração são consistentes com o tempo e com as prioridades definidas no problema e iterações anteriores.

C9. Design de software. (5 aulas concorrentes com C10 e C11)

O9.1. Descrever formalmente os componentes dos requisitos funcionais de software definidos anteriormente.

Atividades:

A9.1. (*) Definir os componentes de software a partir de um requisito dado.

P. Dá instruções para definir os componentes de software dos requisitos anteriores.

A. Realizam o design de um conjunto de componentes para um problema.

P. Discute as soluções propostas e as contrasta entre elas.

Avaliação. Os requisitos são modelados correta e consistentemente.

C10. Implementação de código. (5 aulas concorrentes com C9 e C11)

O10.1. Programar ou modificar os componentes de software a partir de um design dado utilizando as ferramentas de desenvolvimento definidas.

Atividades:

A10.1. (*) Programar ou modificar um componente feito e administrar as mudanças.

P. Dá instruções para programar ou modificar o componente de software.

A. Programam.

P. Retroalimentação do trabalho feito. Discute boas práticas e erros comuns no trabalho.

Avaliação. O código compila e ajusta ao design.

C11. Documentação e comunicação. (5 aulas concorrentes com C9 e C10)

O11.1. Comunicar oralmente e por escrito a descrição de métodos (procedimentos), resultados e conclusões.

Atividades:

A11.1. Escrever um registro das atividades das reuniões.

P. Dá instruções para os alunos fazerem uma reunião de organização, pelo menos uma vez cada duas aulas.

A. Fazem a reunião, escrevem um registro das atividades.

P. Revisa os registros e componentes deles.

Avaliação. O planejamento permite alcançar os objetivos do projeto.

A11.2. (*) Documentar eventos e avanços no controlador de versões.

P. Dá instruções para os alunos fazerem melhoras à documentação.

A. Discutem e escrevem.

P. Fecha a discussão com dicas sobre a documentação existente.

Avaliação. A documentação é compressível por outra pessoa que não é o programador.

C12. Comunicação e apresentação. (2 aulas)

Esta unidade de conteúdo inclui as apresentações e demonstrações dos grupos. Portanto, compartilha o objetivo O11.1.

Metade dos grupos (selecionados aleatoriamente) apresentarão em cada aula. Todos os alunos participarão nesta atividade (como público ou apresentadores). Cada grupo apresentará o *release tag* (ou último *commit*) do *sprint* (o protótipo do software deve ser funcional).

Atividades:

A12.1. (*) Apresentar os resultados do trabalho realizado na iteração.

P. Escolhe os responsáveis pela presentação.

A. Apresentam os resultados e o trabalho feito pela equipe, e faz uma demostração do protótipo.

P. Retroalimentação do trabalho feito. Discute possíveis erros e destaca boas práticas feitas.

Avaliação. Os requisitos foram implementados correta e consistentemente. O planejamento permitirá terminar o projeto nas iterações planejadas. Os problemas encontrados foram discutidos e resolvidos.