① Processos de desenvolvimento de software ágeis são aqueles que se baseiam nos princípios do manifesto ágil, como ~~pessoas~~ valorizar mais pessoas no lugar de processo, entrega de software funcional ao lugar de documentos e integração com o cliente. Exemplos ~~diretos~~ de processos ágeis são o Extreme Programing (XP) e crystal. Vale destacar que o Scrum é um processo de gestão de projetos ágil, mas que muitas de suas práticas são adotadas em processos de desenvolvimento de software ágil, como no XP. Vale destacar também que a adoção de processos de desenvolvimento de software ágil em sua totalidade é rara, mas que suas práticas são adotadas com frequência no mercado. Como o XP foi o processo de desenvolvimento ~~com~~ que ganhou maior destaque quando surgiram os métodos ágeis utilizarei ele como exemplo, explicando suas práticas e como diferem de processos prescritivos tradicionais.

No XP a gestão do projeto é baseada no Scrum. Como etapa de requisitos inicial é construído um Backlog de produto onde as histórias de usuário são registradas. Este backlog é priorizado segundo o valor para o negócio ~~definido~~ pelo ~~a~~ product owner, o representante do cliente/negócio. No XP, o product owner (PO) deve fazer parte da equipe ou ter contato constante com ela idealmente trabalhando no mesmo espaço físico dela. Isto é fundamental no XP para que dúvidas ~~de~~ sobre os requisitos possam ser retiradas diretamente com o PO, visto que não há documentação extensiva de requisitos como nos processos prescritivos.

Após a formação de um product backlog inicial, uma parte dele é selecionado para ser desenvolvido em um período fixo de 2 a 4 semanas chamado Sprint. Esta seleção é feita em uma reunião própria chamada de planejamento

do sprint. Esta reunião envolve também o detalhamento e estimativa de esforço para as histórias de usuário. A estimativa faz uso de técnica ~~de~~ chamada planig poker, onde cada membro da equipe faz uso de cartas para estimar o tamanho, complexidade ou esforço da ~~tarefa~~ história. Caso não haja consenso, o participante que colocou a maior carta e o que colocou a menor apresentam seus argumentos e uma nova rodada é executada. ~~Como~~ Este processo é repetido até que se chegue a um consenso. Caso não ocorra o Scrum master, especialista na metodologia de gestão, ~~pode~~ que faz parte do time, pode intervir para facilitar o consenso.

Como parte do planejamento, após as estimativas serem realizadas, histórias de usuário podem ser removidas ou incluídas no ~~Sprint~~ Sprint backlog, desde que ~~caibam~~ caibam no tempo dele. Após o planejamento, é iniciado o sprint, onde o time deve selecionar as histórias do sprint backlog seguindo o maior valor para o negócio estabelecido pelo PO para a execução, uma a uma até que todo o trabalho do sprint seja concluído ou que seu tempo termine.

O desenvolvimento de software no XP segue uma série de práticas ágeis. Primeiro a composição do time deve conter todas as competências necessárias para executar o projeto. ~~O~~ O time também deve ficar alocado todo no mesmo espaço físico, facilitando a comunicação. Não deve haver barreira ~~&~~ físicas entre os membros do time, também para facilitar a comunicação. A programação é feita em pares, o que ~~deve~~ visa facilitar a troca de conhecimento entre os integrantes do time e também melhorar a qualidade do código. Os testes são automatizados e desenvolvidos antes do código-fonte, seguindo o Test-driven development (TDD).

Sempre que uma história é concluída ela deve ser integrada ao repositório de código, realizando integração contínua. Os testes devem passar para o código ser entregue e evitar código quebrado no repositório.

Para controle e coordenação das atividades da equipe é utilizado o quadro kanban, que mostra visualmente quem está realizando qual atividade e qual é o status de cada uma, que pode ser a fazer, fazendo, feito ou outras situações especiais como pausado, com impedimento, etc.

Quando um Sprint é concluído, ocorrem duas reuniões, uma para entrega do produto para o PO e outra para melhoria de processo. Na entrega do sprint (review), as histórias de usuário completas durante o sprint são apresentadas e os testes de aceitação são executados para que o PO aceite ou não as histórias como entregues. Também são justificados os motivos de histórias não terem sido entregues, caso necessário. Na retrospectiva do Sprint, PO não participa, e a equipe reflete como foi o andamento do sprint, o que deu certo, o que não deu, e o que pode ser melhorado nos próximos sprints. Isto serve para melhoria do processo de desenvolvimento de software da própria equipe.

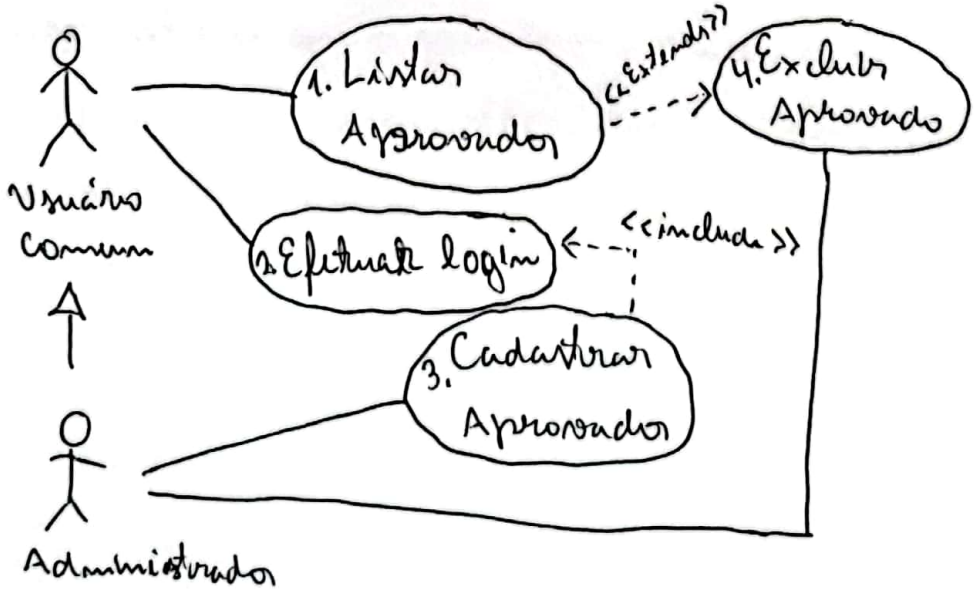
Um ponto que havia me esquecido de mencionar, é que durante o desenvolvimento do software, na execução do sprint, são realizadas reuniões diárias de equipe chamadas de daily meeting. Estas reuniões devem ser realizadas de pé (para ser rápido) e são respondidas três perguntas por cada membro da equipe: o que foi feito desde a última reunião? o que será feito até a próxima? e se há algum impedimento (problema)? Não devem ser resolvidos problemas durante a reunião, mas sim após ela. É recomendado que a reunião ocorra sempre no mesmo horário e que dure no máximo 15 minutos.

Um outro mecanismo de controle e representação visual que é utilizado durante os sprints é o burndown chart. Este é um gráfico que representa no eixo X a quantidade de dias ou tempo do Sprint e no eixo Y a quantidade de pontos ou esforço do Sprint. É traçada uma linha reta do topo do eixo Y até o final do eixo x para indicar o ritmo linear de desenvolvimento do escopo no Sprint. A cada dia é marcado no burndown chart o quanto de esforço ainda falta ser feito, ~~para~~ fazendo também a linha entre os pontos marcados. Caso a linha esteja acima da reta ideal, o ~~projeto~~ Sprint está em atraso, caso esteja abaixo está adiantado.

Por fim, o processo se repete por sprints, ou seja, após a conclusão de um sprint é iniciado um próximo, repetindo até que se conclua o backlog do produto.

(2) Casos de uso são representações de como um software pode ser utilizado. Os casos de uso podem ser representados em forma de diagrama, pelo diagrama de casos de uso da UML, e também possuem descrição em formato de texto estruturado. No contexto da engenharia de requisitos, a modelagem e descrição dos casos de uso são fundamentais para processos prescritivos de software, pois eles descrevem em detalhes quais serão as funcionalidades do sistema, quais usuários terão acesso a quais funcionalidades e as sequências de interações possíveis com seus resultados esperados. O diagrama de casos de uso e sua descrição fazem parte do documento de especificação de requisitos em sua perspectiva estrutural. Os casos de uso ~~são~~ devem ser validados e aprovados pelo cliente antes do desenvolvimento do software em processos tradicionais.

Para exemplificar considere o seguinte diagrama de casos de uso UML de um sistema para divulgação da lista de aprovados em um determinado concurso público.



O diagrama representa os usuários do sistema (atores), no caso, usuário comum e administrador, os casos de uso, 1. Listar aprovadores, 2. Efetuar login, 3. cadastrar aprovador e 4. Excluir aprovado. O diagrama também representa as relações entre atores, entre atores e casos de uso e entre casos de uso. A relação entre os atores é uma herança e indica que ~~todos~~ todos os casos de uso do ator apontado pela seta podem ser executados pelo ator que aponta. A relação entre ator e caso de uso indica que um ator pode executar o caso de uso. Já a relação entre casos de uso são de dois tipos, include, que indica que o caso de uso apontado será executado dentro do caso de uso que o aponta obrigatoriamente, já o extends representa que o caso de uso apontado pode ser executado opcionalmente a partir de um ponto de extensão (para) do caso de uso que o aponta.

Cada ~~a~~ elipse (caso de uso) possui uma descrição textual de forma estruturada contendo os seguintes campos: nome do caso de uso, atores que o executam, pré-condição, pós-condição, ~~fl~~ fluxo principal, fluxos alternativos e fluxos de exceção. Como exemplo irei descrever o caso de uso de listar aprovador.

Caso de uso: 1. Listar Aprovadores
Atores: Usuário Comum e Administrador
Pré-condição: não há.
Pós-condição: Usuários aprovadores exibidos ~~na~~ na tela (listagem)
Fluxo Principal:

1. O caso de uso se inicia quando o usuário clica no botão exibir aprovadores no menu principal do sistema
2. O sistema exibe uma listagem com os participantes

do concurso que foram aprovados mais [RN01] contendo os seguintes campos: nome, número de inscrição e nota.

Fluxos Alternativos

A.1 - no passo 1 do fluxo principal o usuário logado é administrador do sistema

A.2 - O sistema exibe além dos dados contidos no fluxo principal também um botão de excluir candidato para cada linha da listagem.

Fluxos de exceção:

não há

Regras de negócio (poderia estar em outra parte do doc. de requisitos)

[RN01] - Para ser aprovado o candidato deve tirar nota maior que 7,0 na prova escrita.

O fluxo principal do sistema indica o uso típico do caso de uso mostrando sempre a ação do usuário e a reação do sistema. O fluxo alternativo representa alternativas de uso do sistema, no caso mostramos o comportamento do sistema com outro perfil de acesso. Os fluxos excepcionais mostram o comportamento do sistema em caso de erro de uso ou problemas. Já as regras de negócio, que podem estar em outra parte do documento de requisitos, mostram regras de externas que influenciam o comportamento do sistema.

(3) Contagem de pontos de função é uma técnica de estimativa de tamanho (dimensão e volume também são termos utilizados) de software. Seu uso se dá pela contagem de características funcionais do software a partir de artefatos de requisitos e análise. A contagem de pontos de função tem como objetivo muitas vezes estimar o valor base para contratos de software de escopo e preço fixo, como ~~muitas vezes~~ o por exemplo em licitações públicas no Brasil. Isto é feito multiplicando a quantidade de pontos de função por um valor por ponto de função, por exemplo, mil Reais por ponto de função.

A contagem inclui as seguintes características: entradas, saídas, consultas, arquivos e interfaces externas. Além disso, a contagem é ajustada pela complexidade dos ~~itens~~ elementos envolvidos na funcionalidade, considerando subcaracterísticas e limites da quantidade delas para definir se a complexidade será baixa, média ou alta. A complexidade define pesos para ~~a contagem~~ que são multiplicados pela contagem dos itens para chegar a quantidade de pontos de função ajustados de uma funcionalidade.

Uma entrada representa uma entrada de dado, um campo de dados em um formulário que é submetido ao sistema seria uma entrada.

Uma saída, da mesma forma, indica um campo de dado que é exibido pelo sistema em uma funcionalidade.

Uma consulta representa a ida do sistema em um arquivo lógico buscar ou gravar um conjunto de dados.

Um arquivo representa um agrupamento ~~lógico~~ lógico de dados que fica armazenado no sistema.

Por fim, uma interface externa representa um ponto de acesso a outro sistema que é utilizado para acessar ou enviar dados para outro sistema.

Como exemplo de contagem utilizarei o caso de uso 1. Listar aprovados do sistema descrito na questão 2. Considere que a lista de aprovados é armazenada em um arquivo lógico interno contendo 3 campos de dados, matrícula, CPF e nota do candidato. Considere também que o nome do candidato é recuperado a partir de uma API externa a partir do seu CPF. Considere também que na nossa contagem de subcaracterísticas consideraremos duas formas de visualização de ~~uma~~ saída como complexidade média (admin e usuário comum). Segue a contagem na tabela abaixo.

Entradas	Simples	média	complexa
Saídas		3	
consultas	2		
Arquivos	1		
Interface Externa	1		

Os pesos e a fórmula para calcular o total de pontos de função ajustados a partir da contagem da tabela acima não me recordo, mas estabelecer pesos para cada linha x coluna da tabela acima.