Problems and Programmers

Eduardo Figueiredo

http://www.dcc.ufmg.br/~figueiredo

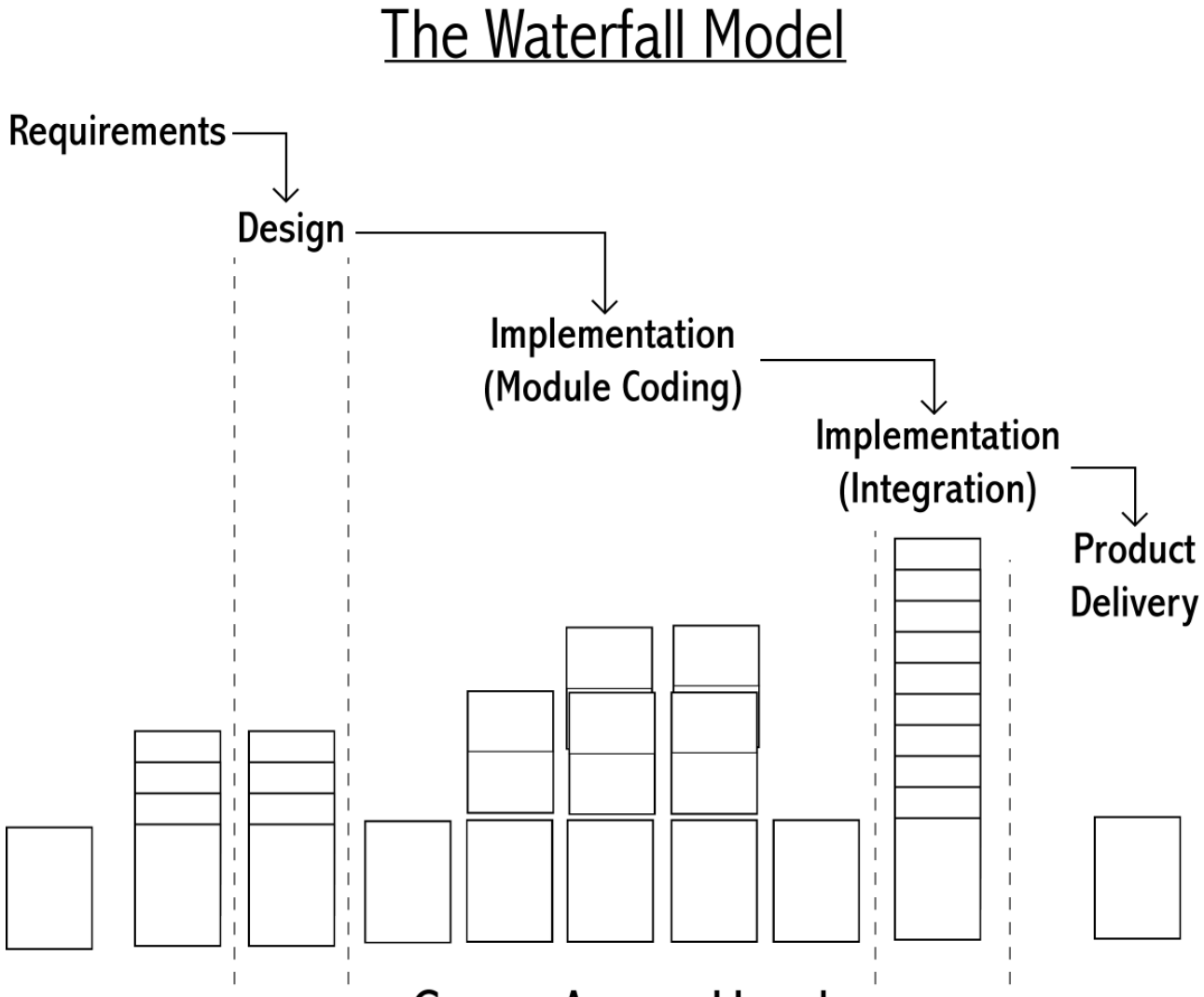
Visão Geral do PnP

- O jogo Problems and Programmers (PnP) simula um processo de software
  - Fase de requisitos até a entrega
  - O jogo tira o foco do produto que é geralmente explorado em TPs
  - O jogo prioriza conceitos gerenciais
- Eventos que ocorrem no jogo tem correspondência no mundo real

Regras Elementares

- Jogadores assumem o papel de gerentes de projetos
- Todos têm que implementar o mesmo projeto no menor tempo possível
  - Quem completar o projeto primeiro ganha o jogo
- Jogadores devem seguir o Modelo Cascata

Disposição das Cartas



Fases do jogo PnP

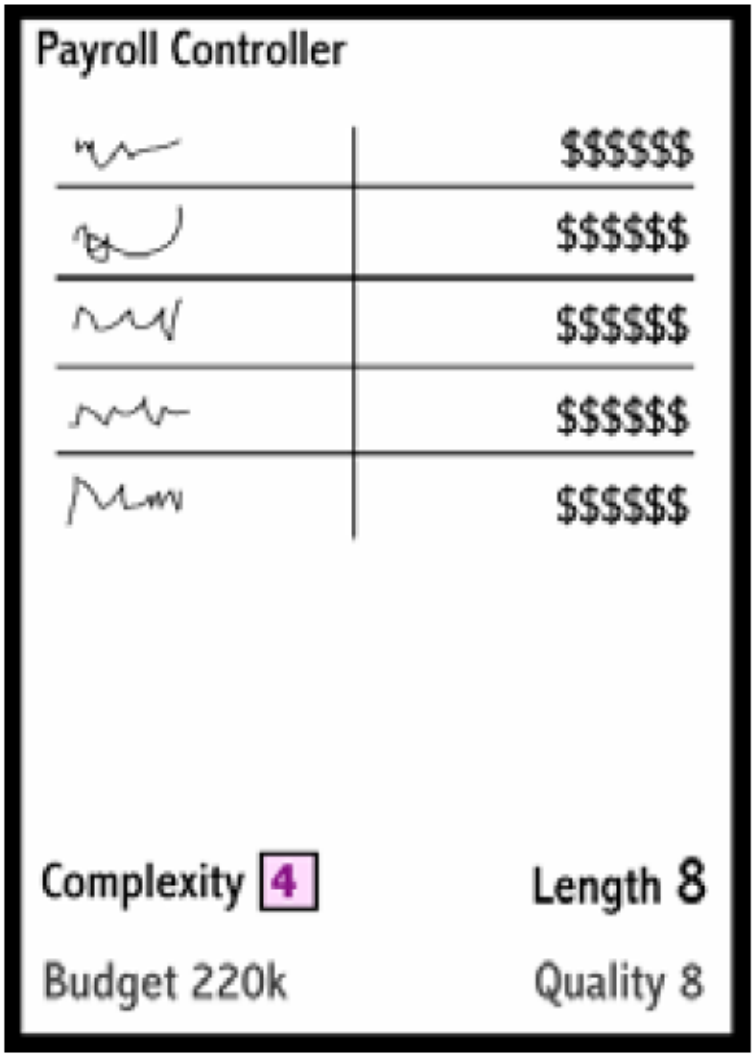
- O jogo é dividido em seis fases
  1. Definir configuração inicial
  2. Fase de Requisitos
  3. Fase de Projeto
  4. Fase de Implementação
  5. Fase de Integração
  6. Entrega do Produto

Configuração Inicial

Fase 1: Configuração Inicial

- Uma carta de projeto é sorteada
  - Existem vários projetos com atributos diferentes
- Um projeto define quatro atributos
  - Tamanho (*length*)
  - Complexidade (*complexity*)
  - Qualidade (*quality*)
  - Orçamento (*budget*)

Exemplo de Carta de Projeto



Atributos de um projeto

- **Tamanho** define quanto de código é necessário para completar o projeto
- **Complexidade** determina quanto de habilidade os programadores precisam para gerar um unidade de código
- **Qualidade** diz quanto do código precisa ser verificado ao final do projeto
- **Orçamento** limita a quantidade de programadores e cartas de conceitos

Início do Jogo

- Após o projeto ser sorteado, cada jogador pega 5 cartas no monte principal
- Existem três montes de cartas
  - **Monte principal** possui cartas de problemas, conceitos e programadores
  - **Monte de documentação** possui cartas de requisitos e de projeto
  - **Monte de implementação** possui cartas de código

Tipos de Cartas

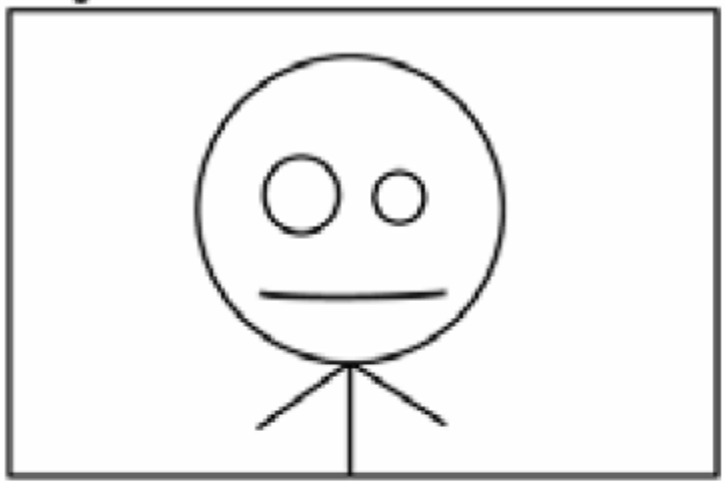
- **Cartas de Conceito** representam boas decisões de desenvolvimento
- **Cartas de Programadores** representam os membros da equipe
- **Cartas de Problemas** definem uma penalidade quando boas práticas não são adotadas de jogo

Exemplos de Cartas

Concept - Walkthrough

You may discard this card to replace an unclear requirements and/or an unclear design with new cards from the documentation deck.

Programmer - Arnold



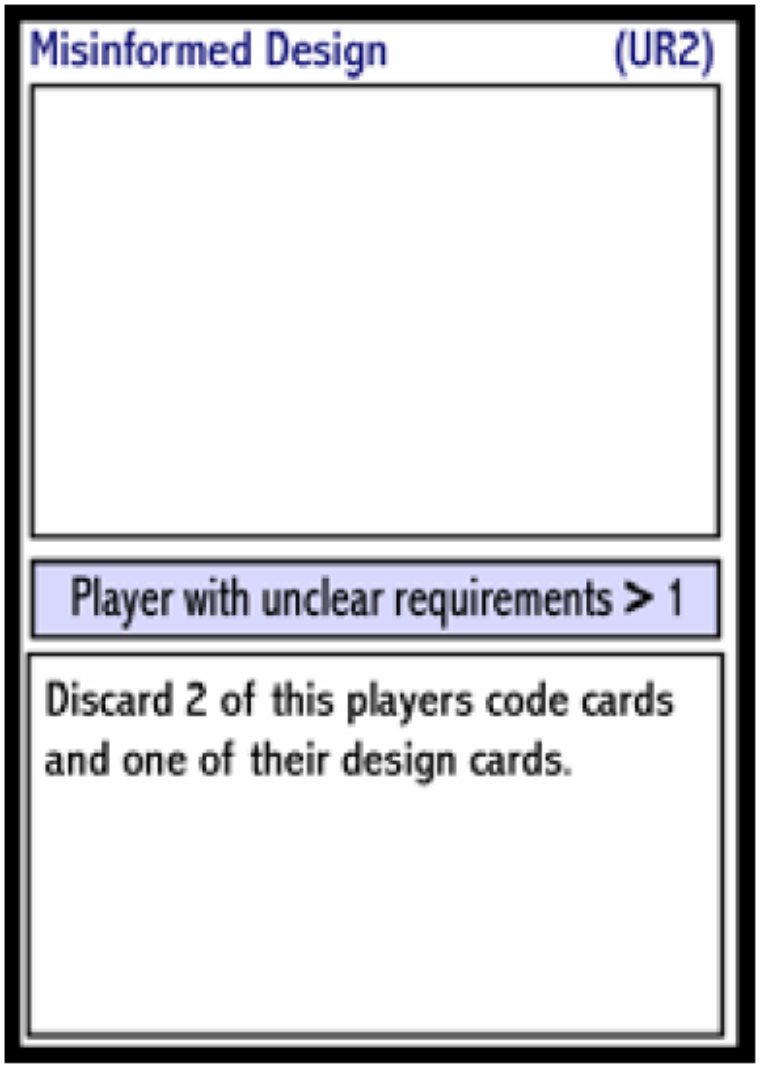
Good experience, but is aloof and could be difficult to work with.

Salary: 90k

Skill 5 Personality 2

Cartas de Problemas

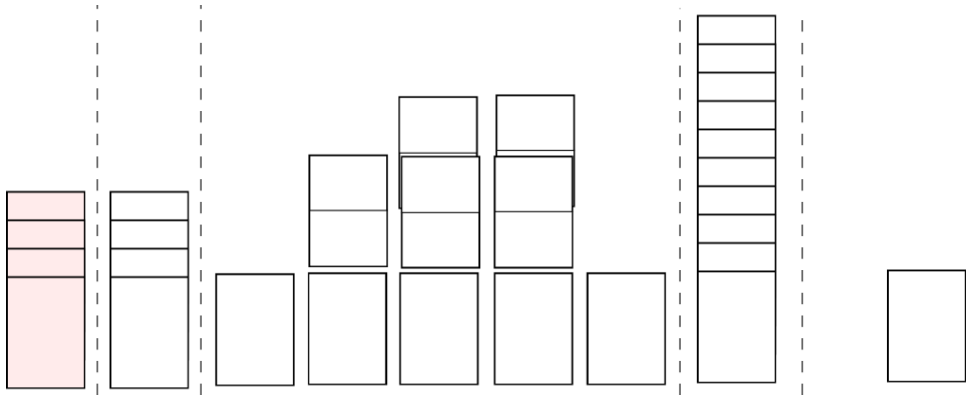
- Cartas de problemas são jogadas para os adversários
- Se o jogador receber uma carta e satisfizer sua condição, ele será penalizado conforme a consequência



Estrutura das Rodadas

- Na sua vez em cada rodada, o jogador deve seguir os seguintes passos
  1. Decidir se vai passar para a próxima fase do ciclo de desenvolvimento
  2. Comprar cartas
  3. **Fazer ações que são permitidas para a fase que ele se encontra**
  4. Empregar conceitos e/ou programadores
  5. Descartar cartas desnecessárias

Fase de Requisitos



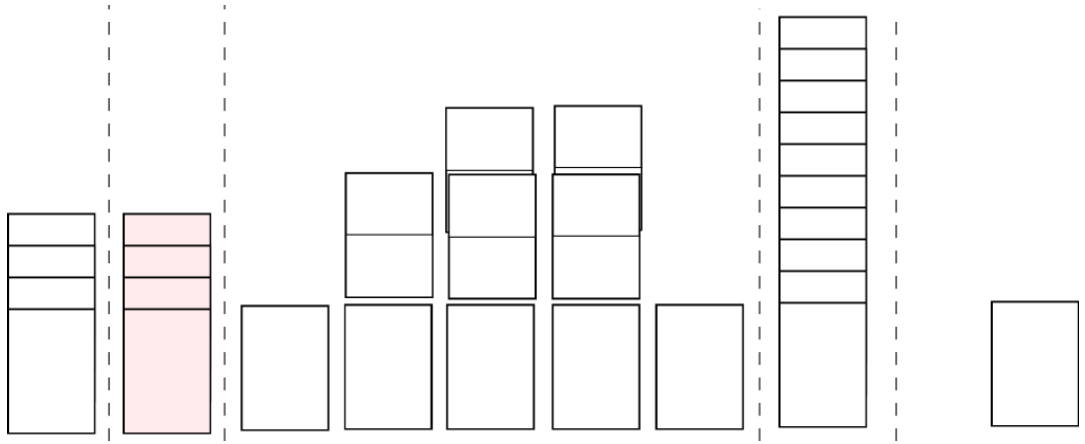
Fase de Requisitos

- Em sua vez, o jogador na fase de requisitos pode comprar duas cartas do monte de documentação
- Jogadores são encorajados a especificar os requisitos antes de moverem para as fases seguintes
  - Nenhum jogador é obrigado a trabalhar os requisitos

Requisitos Claros e Não Claros

- A maioria das cartas de documentação são brancas
  - Representam requisitos claros
- Podem haver cartas de documentação "*Unclear*"
  - Representam requisitos não claros
- Para transformar um requisito não claro em requisito claro, é necessário gastar uma das duas opções de compra

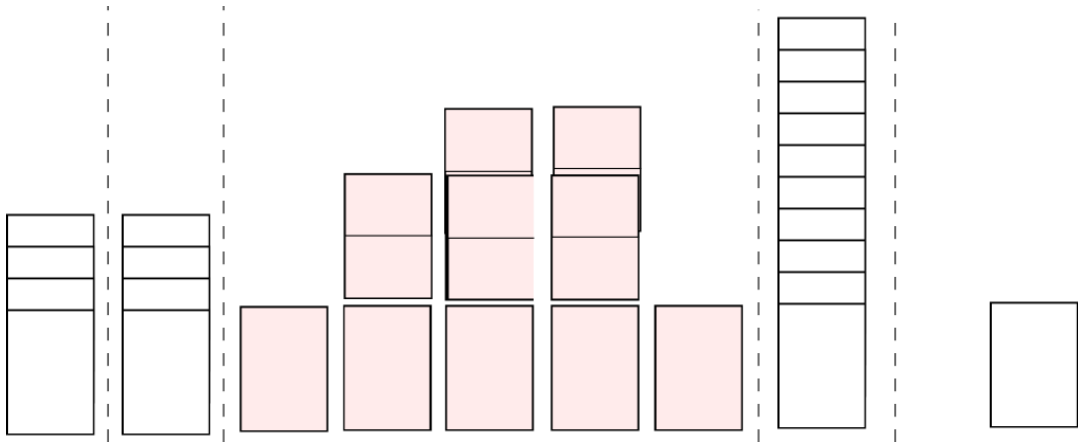
Fase de Projeto



Fase de Projeto

- É muito semelhante a fase de requisitos
  - Compra-se duas cartas no mesmo monte de documentação
  - Pode haver documentação de projeto não claro (os mesmo procedimentos são adotados)
- As cartas são colocadas na coluna de projeto

Fase de Implementação



Fase de Implementação

- Nesta fase, os programadores fazem ações referentes ao seus respectivos valores de habilidades
- As ações possíveis são
  1. Produzir código bem estruturado
  2. Produzir código “espaguete”
  3. Inspecionar código
  4. Resolver *bugs*

Ação de Codificar

- Para produzir uma carta de **código estruturado**, o programador gasta tempo equivalente a complexidade do projeto

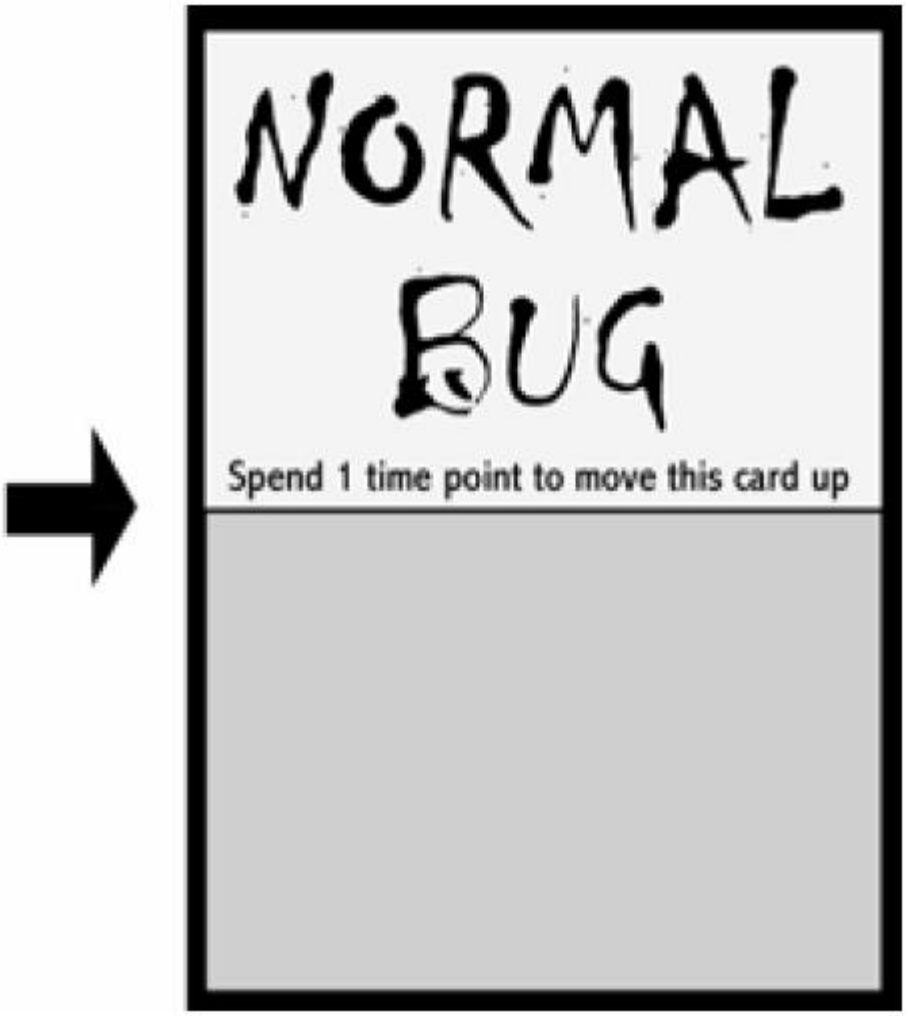
- Para produzir uma carta de **código espaguete**, o programador gasta metade da complexidade do projeto

Código por Programador

- Cada carta de código produzido é colocada em uma coluna acima do programador que a produziu
  - Inicialmente, a carta é colocada com a face para baixo
- A face da carta para baixo significa que o código não foi inspecionado
  - Possíveis *bugs* no código estão ocultos

Código Não Inspecionado

Código Inspecionado



Código Estruturado x Espaguete

- A mesma carta é usada para código estruturado ou espaguete
- A carta possuir duas metades
  - Metade escura (azul) para cima representa código estruturado
  - Metade clara (vermelho) para cima representa código espaguete
- Código espaguete revela *bugs* mais frequentemente que código estruturado

Ação de Inspecionar

- Inspecionar uma carta de código requer uma unidade de tempo
  - Unidade de tempo equivale a uma unidade de habilidade do programador
- Após inspecionar uma carta de código, o programador pode virar a face da carta para cima
  - Revela a existência ou não de *bug*

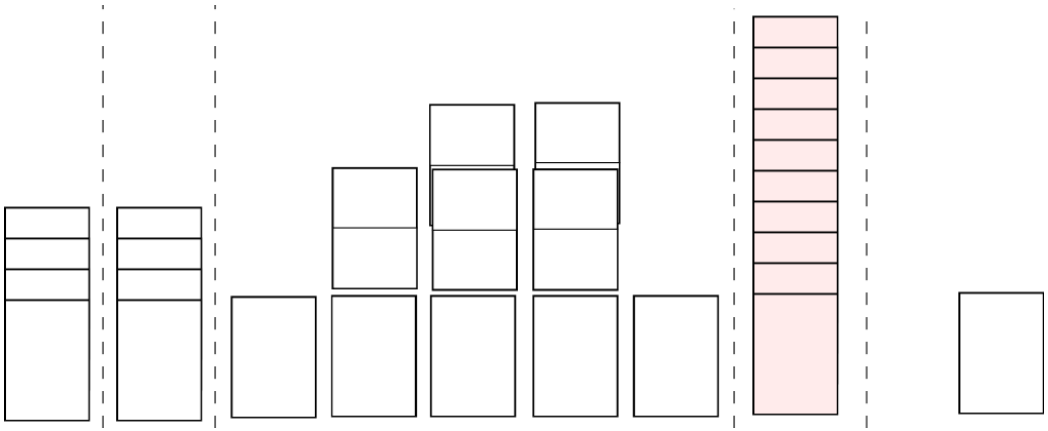
Ação de Resolver *Bug*

- Os programadores podem usar um ponto de tempo tentar remover um *bug*
  - Entretanto, o *bug* pode não ser removido completamente da primeira vez
- Para remover um *bug* pode ser necessário várias unidades de tempo
  - Apenas *bugs* simples (*Simple*) podem ser trocados diretamente por uma nova carta de código usando um ponto de tempo

Normal e *Nasty Bug*

- Cada unidade de tempo gasto pelo programador move o *bug* uma posição para cima na coluna
  - Troca a carta com *bug* pela carta imediatamente acima na coluna do programador
- Quando a carta com *bug* atingir o topo
  - Com uma unidade de tempo, a carta com *bug* pode ser trocada por uma nova carta

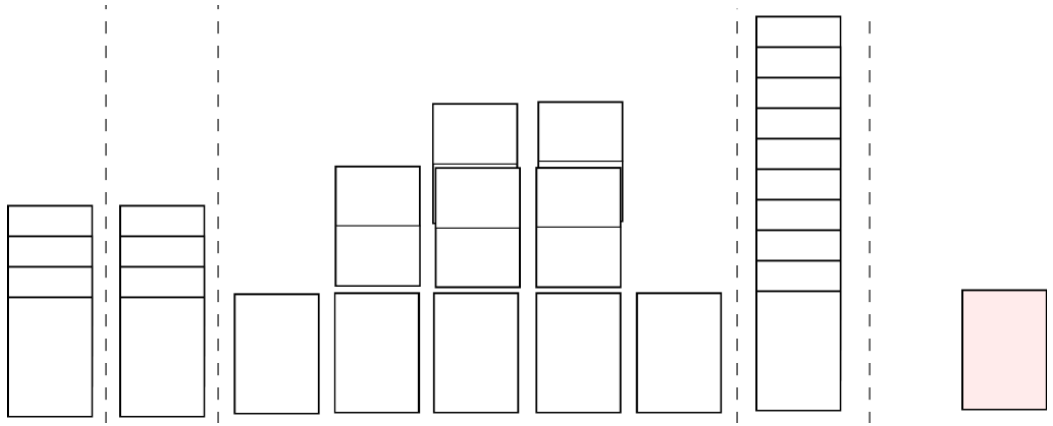
Fase de Integração



Fase de Integração

- Em cada rodada de integração, o código de um programador pode ser movido para a parte de código integrado
- O projeto termina quando for integrada quantidade de código equivalente ao tamanho do projeto
  - O jogador pode integrar código que não foi inspecionado (ou mesmo com *bug*)

Entrega do Produto



Fase de Entrega do Produto

- Durante a entrega do produto, é feita a verificação e validação com o cliente
- O cliente verifica quantidade de cartas de código equivalente a qualidade do projeto
- Se o cliente descobrir algum *bug* nessa fase, o jogador será penalizado
  - Podendo até mesmo ser desclassificado

Penalidades por Bug na Entrega

- Se o cliente encontrar apenas *bugs* simples (*Simple*) ou normais (*Normal*)
  - O jogador terá que criar e integrar novamente quantidade equivalente de cartas para substituir o código com *bug*
- Se o cliente encontrar algum *bug* grave (*Nasty*)
  - O jogador será desqualificado e o jogo continua com os demais jogadores

Término do Jogo

- Se nenhuma das cartas reveladas pelo cliente possuírem *bugs*, o jogador ganha!

Bibliografia da Aula

- A. Baker, E. Navarro, and A. van der Hoek. **An Experimental Card Game for Teaching Software Engineering Processes**. Journal of Systems of Software (JSS), 2004.

- A. Baker, E. Navarro, A. van der Hoek. **Problems and Programmers: an Educational Software Engineering Card Game**. International Conference on Software Engineering (ICSE), 614-619, 2003.