

Metodologia Simplified

António Rocha - 2003

Metodologias

- As empresas precisam de uma metodologia simples e eficaz para realizarem o seu primeiro projecto OO
- “Uma metodologia tem mais probabilidades de ser usada quando é simples, pequena e eficaz”
 - A.R. Cockburn na Object Magazine
- A maior parte das metodologias de desenvolvimento de software são muito grandes e complexas para serem utilizadas em projectos de software reais.

Metodologias

- Em vez de notações de modelação complexas e detalhadas são precisos métodos práticos com notações claras e um processo simples para descrições.
- O problema é que tais métodos têm muitas fraquezas, facilmente detectadas pelos “gurus”. Mas será que isso interessa ? Será que uma boa metodologia tem de cobrir à partida todos os detalhes ?
- Por exemplo: a introdução da metodologia OMT (Rumbaugh, Object Oriented Modeling and Design), e a OOSE (Jacobson, Object-Oriented software engineering) estão compiladas em cerca de 500 páginas.
- É necessário uma metodologia simples para principiantes e para equipas que trabalhem em pequenos projectos.

Requisitos para uma metodologia

- Uma metodologia, mesmo uma simples, tem de ter os seguintes requisitos:
 - Servir de guia em todo processo de desenvolvimento do sistema, desde os requisitos até aos testes.
 - Ter notações e descrição dos processos
 - Especificar as fases do produto, como documentos e figuras
 - Permitir extensões
 - Ser fácil de aprender e usar

Requisitos para uma metodologia

- **Tem de permitir modelar:**
 - **A funcionalidade do sistema, isto é, o que o sistema dará ao utilizador final**
 - **Quais são os objectos do sistema e como se relacionam entre si. O método tem de ajudar a descobrir os objectos baseando-se na análise do domínio do problema. O método tem também de permitir refinar e transformar o domínio objectos numa forma que possa ser implementada através de uma linguagem de programação.**
 - **Como os objectos colaboram para fornecer a funcionalidade desejada.**

A metodologia Simplified

● **Notação:**

■ **Linguagem natural**

- **É a principal ferramenta para capturar os requisitos, é tipicamente utilizada sempre que existe a necessidade de comunicar com os utilizadores finais. Esta linguagem é, também, utilizada para dar ênfase a algo relacionado com os diagramas ou classes.**

■ **Diagramas de classes**

- **Fornecem uma vista estática dos objectos do sistema nas várias fases de desenvolvimento**

■ **Diagramas de sequência**

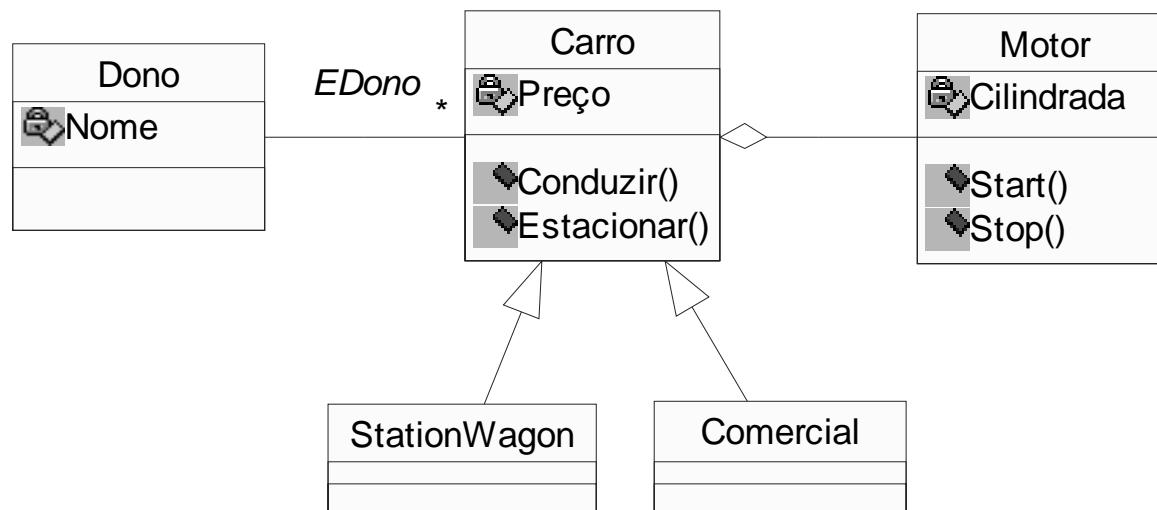
- **Fornecem uma vista funcional dos objectos através da ilustração da cooperação entre eles.**

Notação

- Todas as figuras devem ser simples e legíveis. Os diagramas de classes e de sequência devem apenas ilustrar o que é necessário. Se for necessário escolher entre uma modelação profunda ou menos profunda, normalmente é sempre melhor escolher a menos profunda e adicionar mais comentários em texto.

Notação

- O método Simplified utiliza a notação dos diagramas de classes do UML, embora não sejam necessários todos os detalhes da notação. Tipicamente, são suficientes :
 - As associações 1:1, 1:N, N:N
 - Agregações
 - Herança

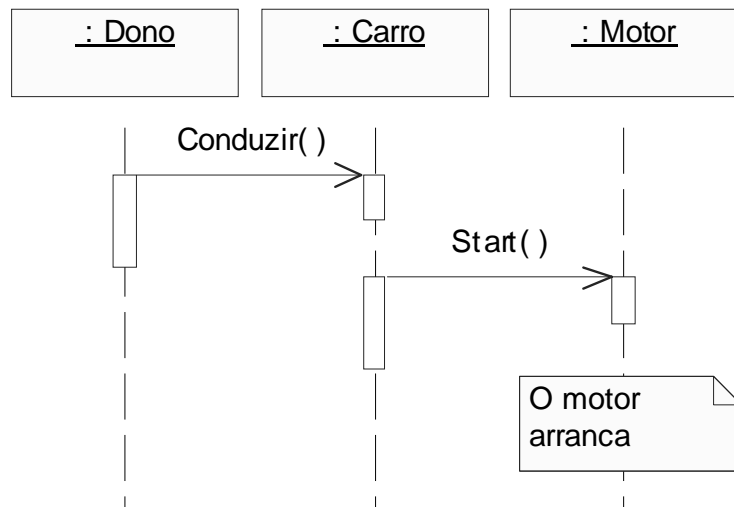


Notação

- Os diagramas de sequência ilustram como é que instancias de várias classes comunicam entre si. Cada diagrama de sequência mostram um fluxo sequencial de eventos. O fluxo pode ser posto em movimento pelas acções do utilizador final, como por exemplo carregar num botão, ou por acções internas, como por exemplo um *timer*.
- Os diagramas de sequência mostram como um conjunto de objectos comunicam entre si para fornecer uma determinada funcionalidade.

Notação

- **Descrição:** O dono começa a conduzir o carro
- **PreConditions** : O carro está estacionado
- **PostConditions:** O dono está a guiar o carro
- **Exceptions** : Não consegue colocar o motor a trabalhar



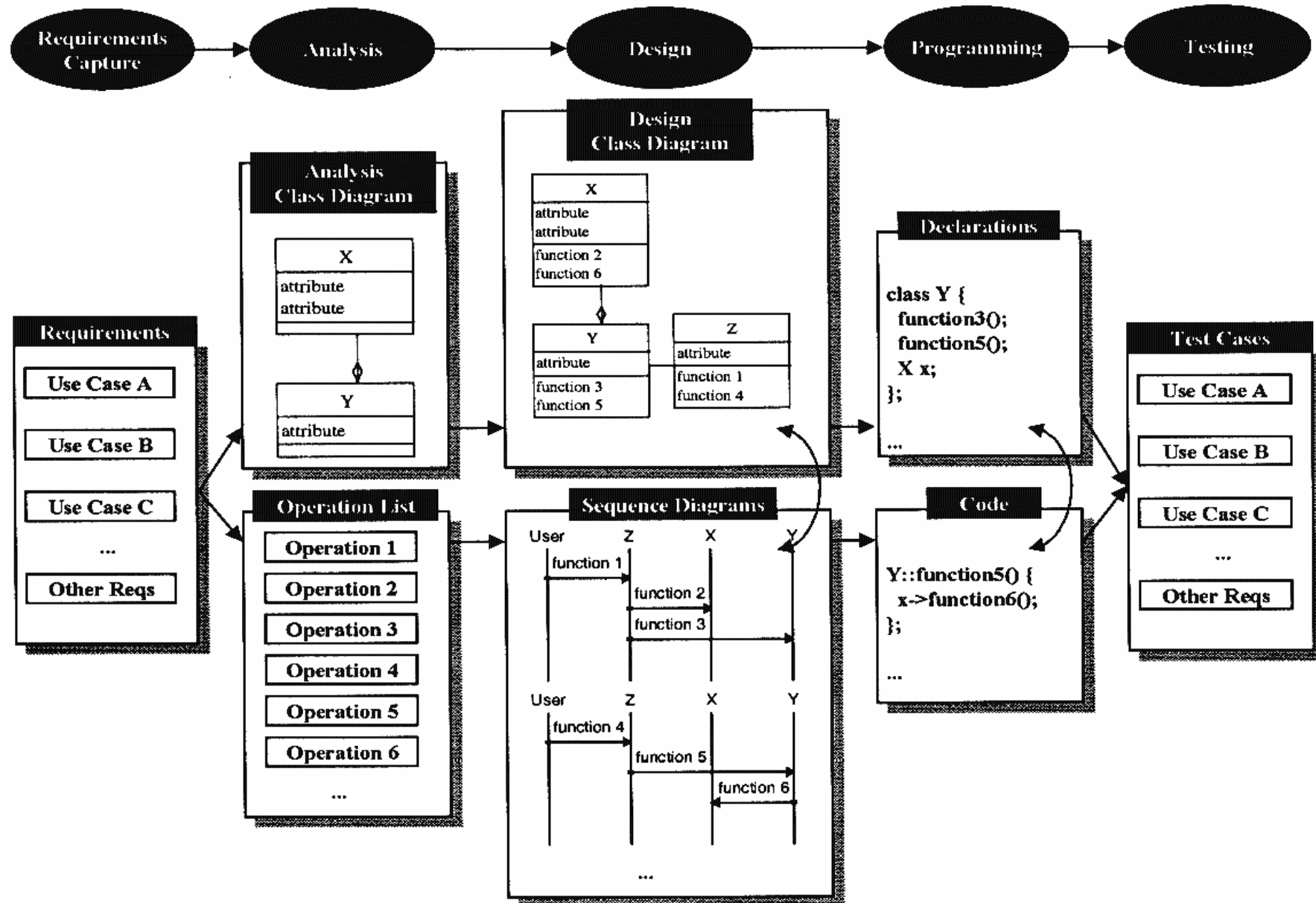
Fases e paralelismo

- Captura de requisitos
 - Recolhe todos os requisitos que existem para o sistema poder ser desenvolvido
- Análise
 - Modela os conceitos, isto é, os objectos do domínio do problema, e analisa também as operações do sistema.
- Desenho
 - Nesta fase, os produtos da análise são transformados numa forma que possa ser programada. O desenho mostra as estruturas dos objectos, as suas interfaces e como colaboram
- Programação
 - Produz o código e concentra-se tipicamente numa classe de cada vez.
- Testes
 - Testa o sistema tendo em conta os requisitos.

Fases e paralelismo

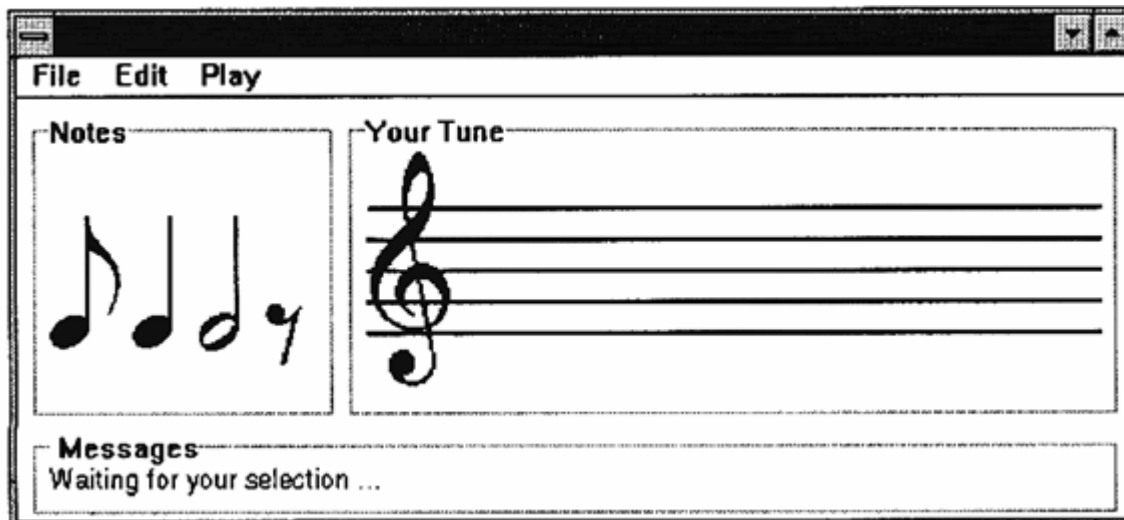
- Existem dois caminhos paralelos no processo de desenvolvimento do sistema :
 - O estático, que utiliza os diagramas de classes para mostrar as propriedades estáticas do sistema.
 - O funcional, que utiliza a descrição das operações e diagramas de sequência para mostrar as propriedades funcionais do sistema.
- Os dois estão relacionados

Fases e paralelismo



Exemplo

- Exemplo: desenvolvimento de uma aplicação que permite a estudantes de música (iniciais) compor músicas simples. A aplicação chama-se 'Compositor Elementar'.



Requisitos funcionais

- Use Case 1 : Compor uma música
 - O estudante inicia a aplicação e esta mostra uma pauta vazia e um quadro com tipos de notas possíveis para selecção. O estudante selecciona um tipo de nota com o rato(ex. quarter, half, quarter rest, etc), depois arrasta-a para o lugar da pauta onde a pretende colocar. Seleccionando as várias notas e colocando-as na pauta, o estudante vai compondo uma música. O estudante põe a música a tocar e grava-a no disco. Finalmente ele fecha a aplicação.
- Use Case 2 : Ouvir uma música composta previamente
 - O estudante inicia a aplicação e carrega a música que quer ouvir do disco. Todas as notas da aplicação aparecem na pauta. Depois disso o estudante põe a música a tocar.

Requisitos não funcionais

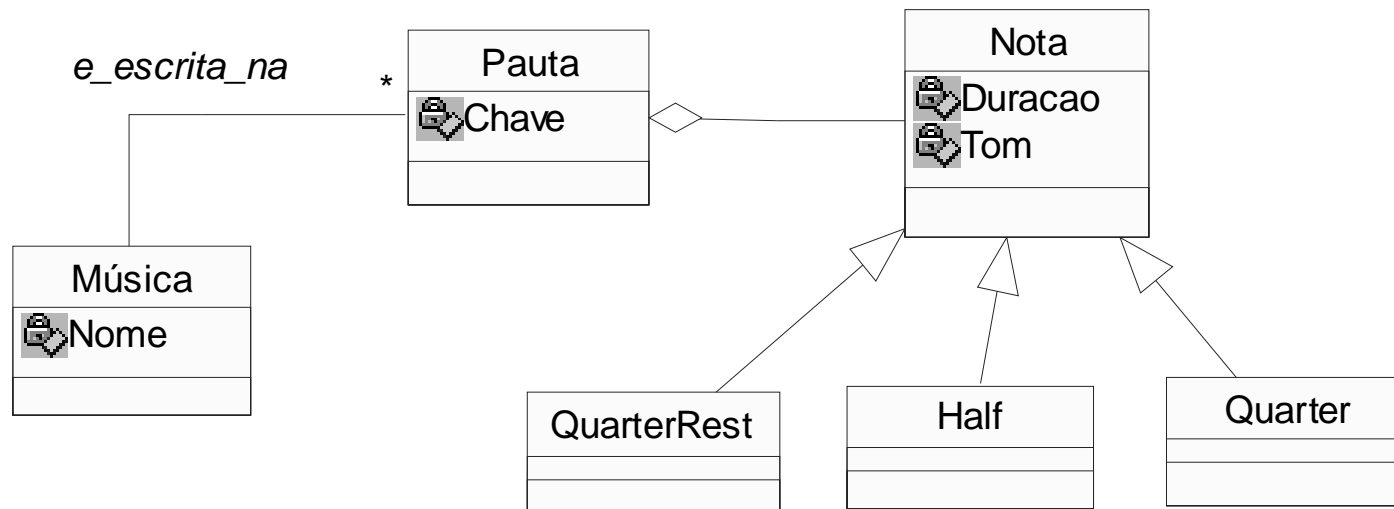
- **Requisito não funcional 1 :**
 - **A aplicação suporta as escalas x, y e z...**
- **Requisito não funcional 2 :**
 - **O número máximo de notas por composição é 20.**
- **Requisito não funcional 3 :**
 - **As composições são guardadas em ASCII**

Análise

- **O objectivo da análise é entender o domínio do problema e o sistema a ser desenvolvido. A fase de análise é baseada nos requisitos especificados na fase anterior e tem dois processos :**
 - **Análise de objectos**
 - **Tem como objectivo especificar todos os conceitos chave, relacionados com o sistema a desenvolver. Produz diagramas de análise de classes que documentam os conceitos do problema**
 - **Análise de comportamento**
 - **Modela o sistema como uma caixa preta, modelando apenas as funcionalidades externas do sistema e produz uma lista de operações. O sistema final tem de suportar a funcionalidade de todas estas operações.**

Análise

- Análise de classes



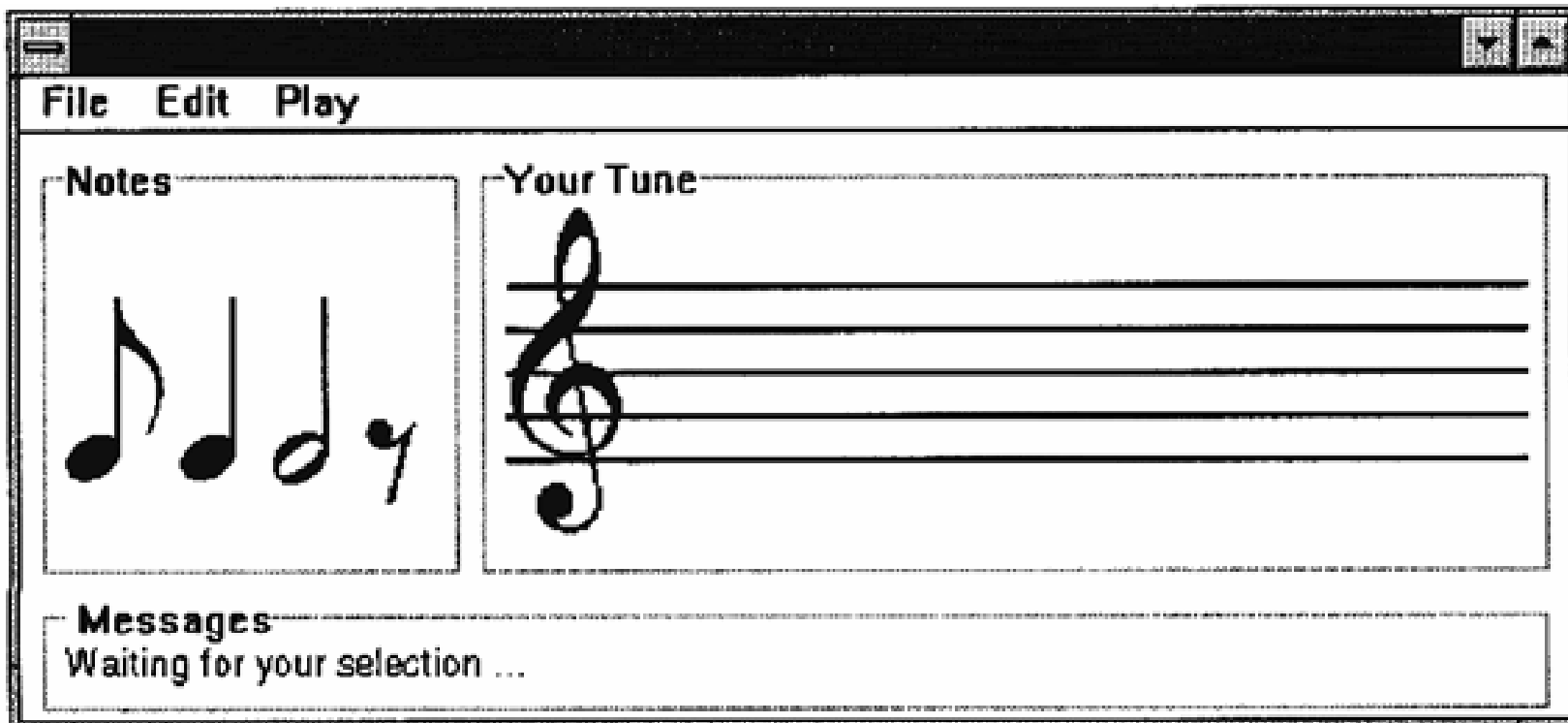
Análise

- A análise de comportamento produz a lista de operações, que é construída com base nos *use cases*. A fig. 7 mostra todas as operações da aplicação, as operações 1 a 5, e 7 são encontradas no primeiro *use case* e as operações 1, 4, 6 e 7 são encontradas no segundo *use case*.
 1. Arrancar com a aplicação
 2. Seleccionar um tipo de nota
 3. Colocar uma nota na pauta
 4. Tocar uma música
 5. Gravar uma música
 6. Carregar uma música
 7. Fechar a aplicação

Interfaces utilizadores

- O método Simplified não inclui uma fase separada para a especificação dos interfaces dos utilizadores. Normalmente, interfaces simples, como a da fig.8, pode ser desenhada em qualquer editor gráfico. Os protótipos das interfaces dos utilizadores podem também ser construídos com a ajuda dos utilizadores finais. No entanto, interfaces complicados podem requerer métodos de análise mais completos (ex.OMT++).

Interfaces utilizadores



Desenho

- O objectivo da fase de desenho é transformar os produtos da análise numa forma que possa ser implementada numa linguagem de programação. Enquanto que a análise concentra-se nos objectos e nas funcionalidades que são relevantes para o utilizador final, a fase de desenho lida com os objectos e funções que têm de ser programadas.
- Esta fase inclui dois caminhos, como mostra a fig.3, os diagramas de análise de classes são transformados em diagramas de desenho de classes, e as operações são modeladas como diagramas de sequência.
- Os objectos do domínio descobertos na análise são modificados para que possam ser implementados. Estas modificações podem ser :
 - Especificações de implementação
 - Modificação da estrutura das classes
 - Identificação de atributos e operações

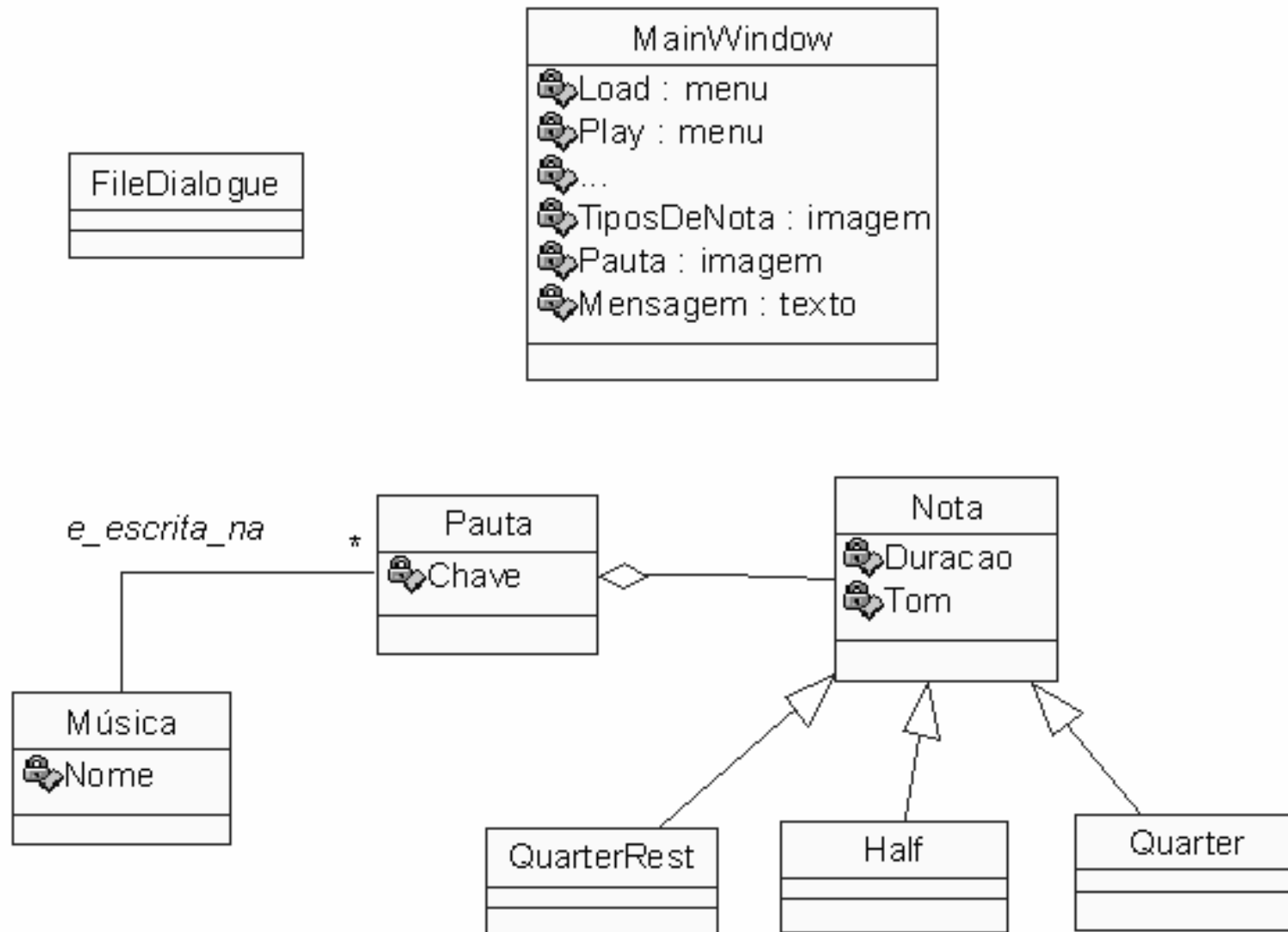
Desenho

- O objectivo da fase de desenho é transformar os produtos da análise numa forma que possa ser implementada numa linguagem de programação. Enquanto que a análise concentra-se nos objectos e nas funcionalidades que são relevantes para o utilizador final, a fase de desenho lida com os objectos e funções que têm de ser programadas.
- Esta fase inclui dois caminhos, como mostra a fig.3, os diagramas de análise de classes são transformados em diagramas de desenho de classes, e as operações são modeladas como diagramas de sequência.
- Os objectos do domínio descobertos na análise são modificados para que possam ser implementados. Estas modificações podem ser :
 - Especificações de implementação
 - Modificação da estrutura das classes
 - Identificação de atributos e operações

Desenho

- Para iniciar a fase de desenho poderá ser construído uma primeira versão do diagrama de classes que está na fig.9.
- Nesta primeira versão o diagrama da análise de classes da fase anterior é a base do diagrama de desenho de classes, e as classes que representam as interfaces do utilizador são adicionadas no topo do modelo

Desenho



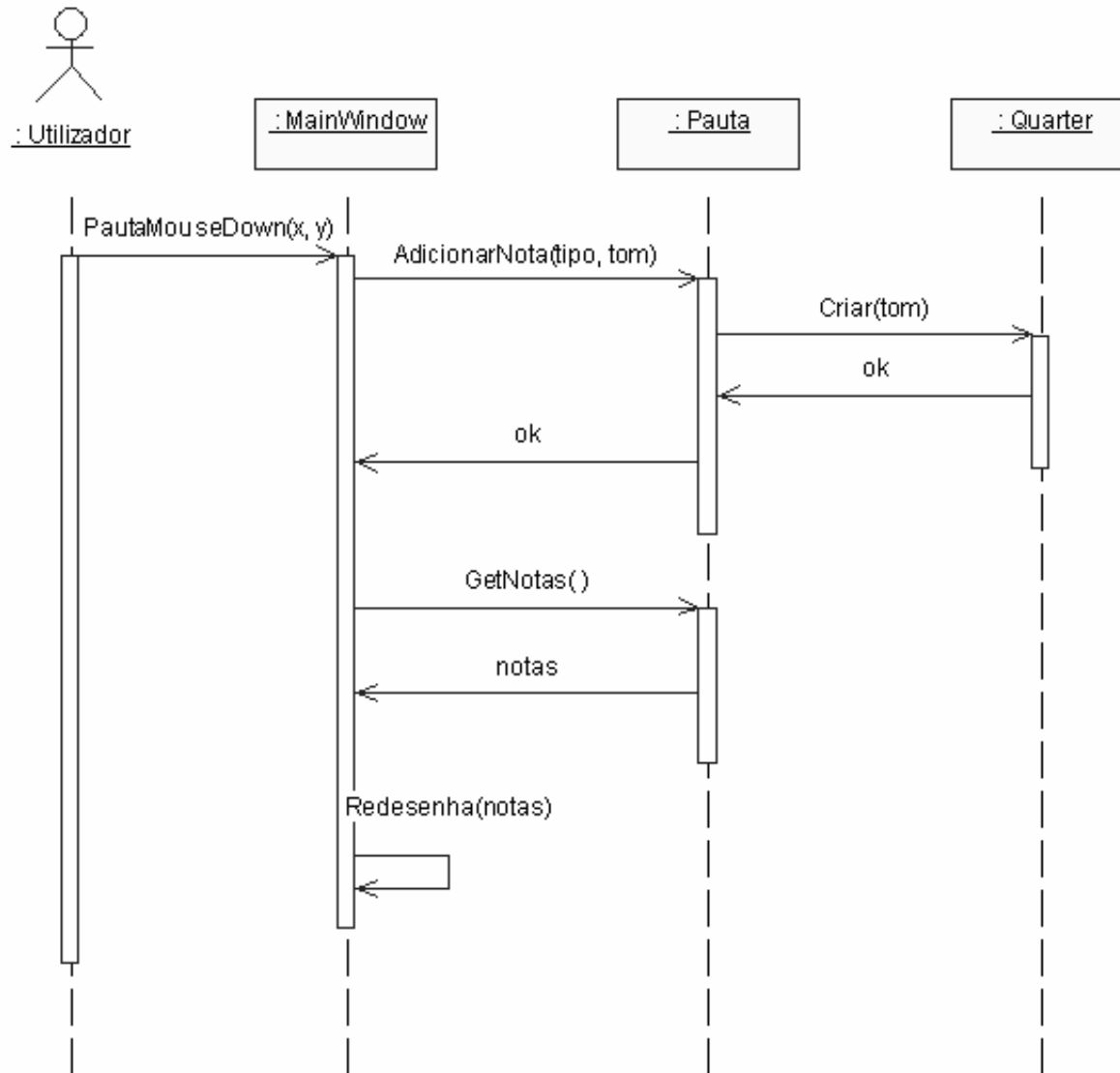
Desenho

- A primeira versão do diagrama de desenho de classes é imatura, e necessita de muitos refinamentos e afinações.
- O refinamento é feito mediante a utilização sistemática de diagramas de sequência. São analisadas todas as operações na lista de operações e são desenhados diagramas de sequência para cada operação. Ao fazê-lo vão ser refinadas as ligações entre as classes e vão ser adicionadas novas operações, atributos e classes.

Desenho

- Diagrama de sequência para a operação “Colocar uma nota na pauta”
 - Descrição: Colocar uma nota na pauta (o objecto nota pode ser de qualquer tipo, e o diagrama de sequência ilustra o caso da nota quarter).
 - Preconditions : O tipo da nota está seleccionado
 - PostConditions: Uma nova nota é adicionada a pauta
 - Exceptions: O número máximo de notas é excedido, não é possível adicionar mais: é mostrada uma mensagem de erro.

Desenho



Desenho

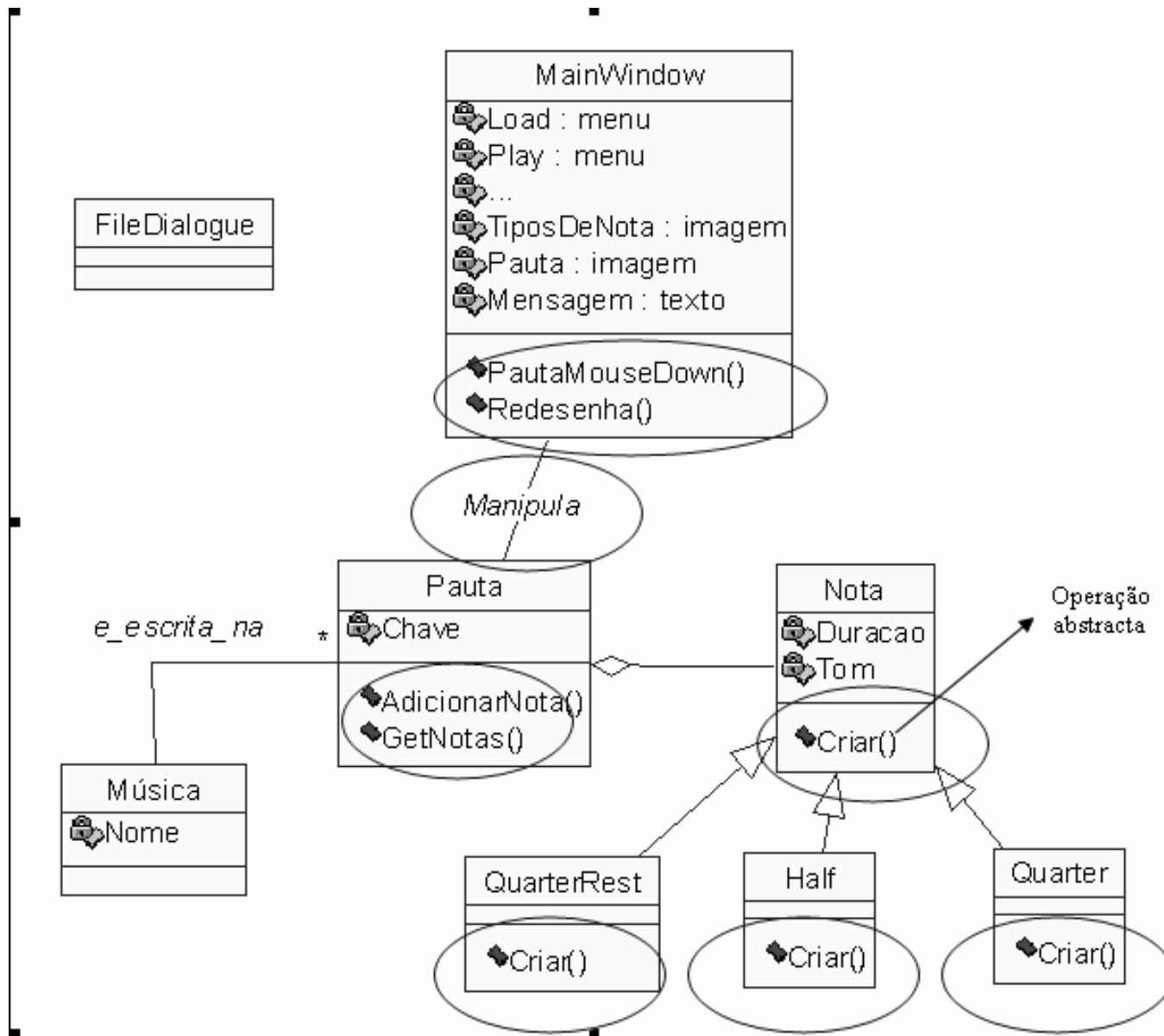


Fig.11: Segunda versão do diagrama de desenho de classes

Desenho

- Diagrama de sequência para a operação “Tocar uma música”
 - Descrição: Tocar uma música
 - Preconditions: Existe uma música na pauta
 - PostConditions: A música é tocada
 - Exceptions: Problemas com os *drivers*, não é possível tocar a música: é mostrada uma mensagem de erro.

Desenho

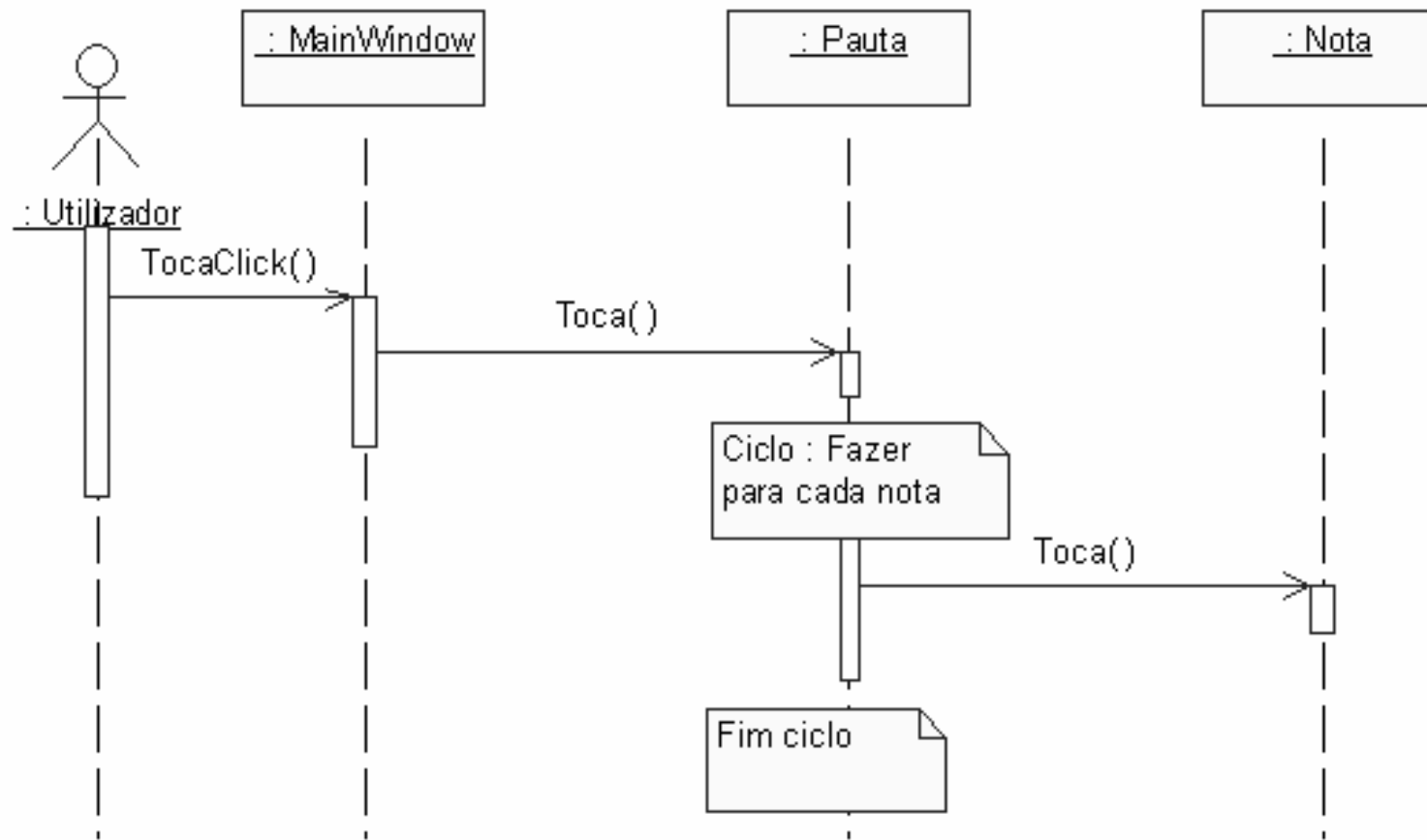


Fig.12 : cooperação entre os objectos quando o utilizador toca uma música

Desenho

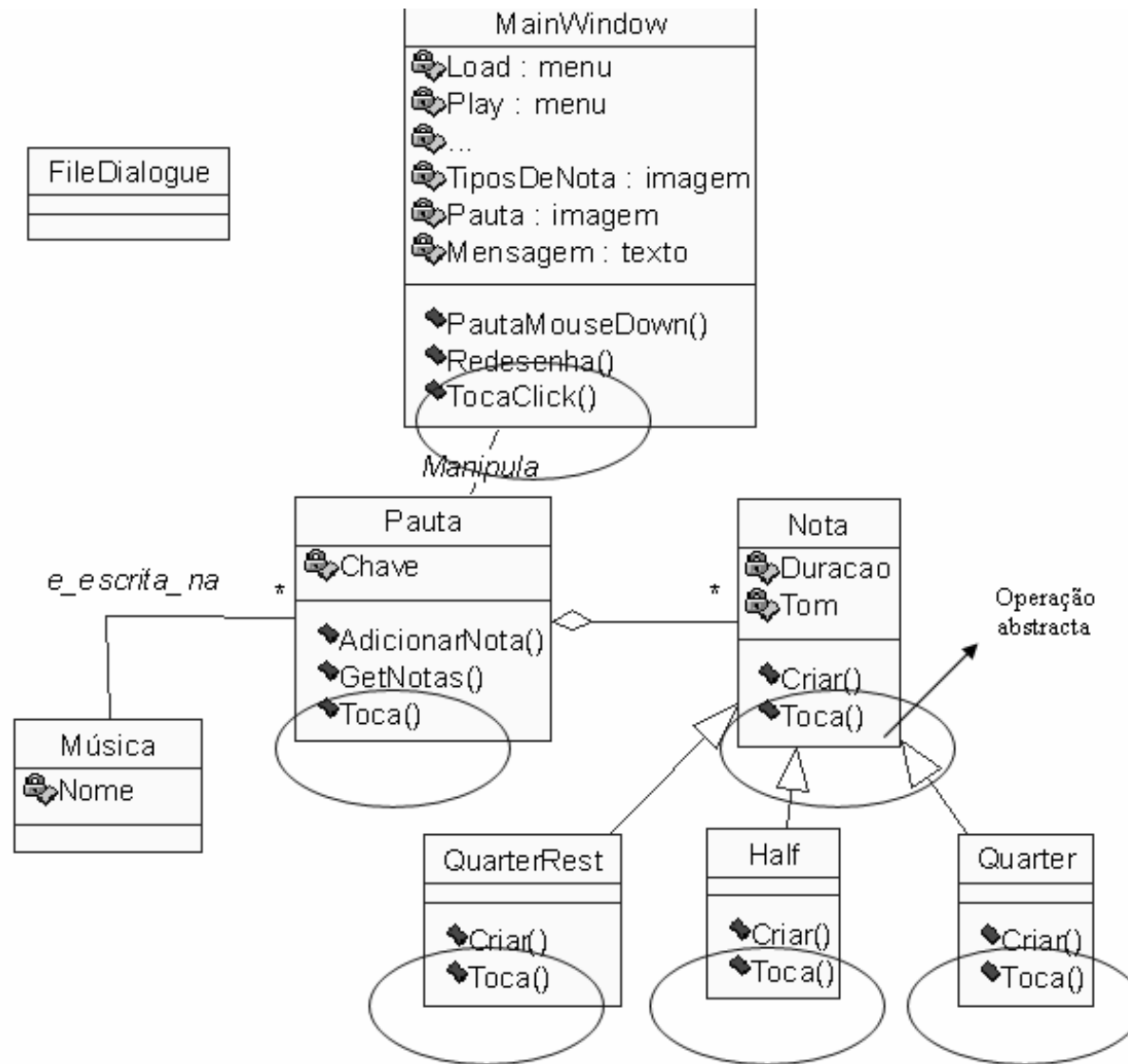


Fig.13.: Terceira versão do diagrama de desenho de classes

Desenho

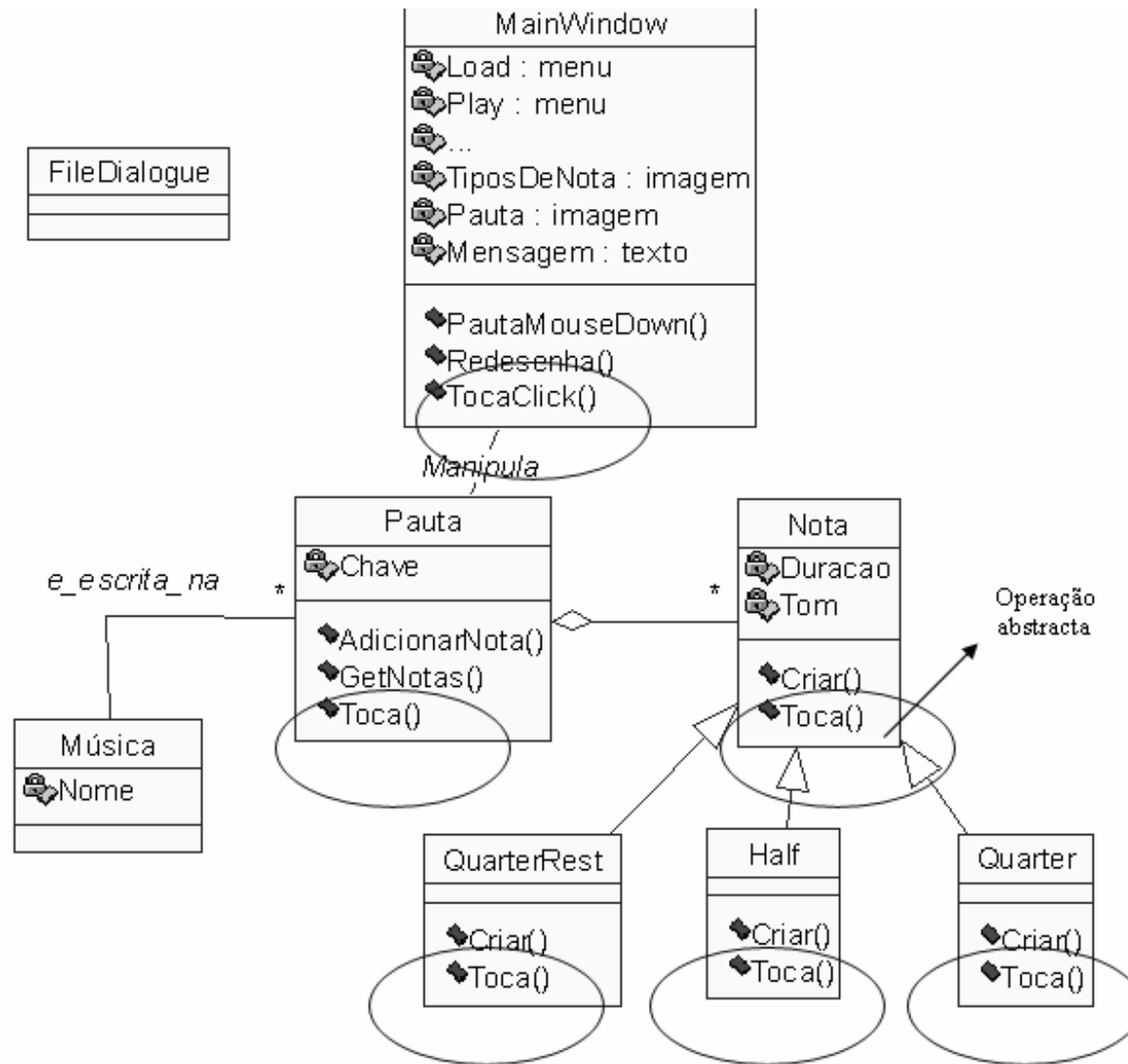


Fig.13.: Terceira versão do diagrama de desenho de classes

Programação

- O objectivo da programação é transformar os diagramas de classes e os diagramas de sequência em código, através de uma linguagem de programação.
- O desenho já modelou todas as classes do sistema e ligações entre essas classes. Segundo o método Simplified, no desenho não é modelada toda a funcionalidade interior dos objectos individualmente, em vez disso, a programação das classes é baseada nos diagramas de classes e diagramas de sequência

Programação

```
EstruturaDeNotas = array[0..20] of ^Nota;
```

```
Pauta = classe
```

```
private
```

```
    Key : integer;
```

```
    Notas: EstruturaDeNotas;
```

```
public
```

```
    function AdicionarNota(tipo : integer, tom : integer) : Boolean;
```

```
    function GetNotas: PChar;
```

```
    procedure Toca;
```

```
end;
```

Fig.14 : Declaração da classe Pauta

```
procedure TmainWindow PautaMouseDown(x, y : integer);
```

```
var    tom : integer;
```

```
    ok : Boolean;
```

```
    notas : Pchar;
```

```
begin
```

```
    {Calcular o tom baseado na variável y}
```

```
    ...
```

```
    ok := MyPauta.AdicionarNota(tipo, tom);
```

```
    if (ok = false) then
```

```
        MessageText.Caption := 'Não é possível adicionar mais notas';
```

```
    Notas := MyPauta.GetNotas;
```

```
    Redesenhar(notas);
```

```
end;
```

Fig.15 : Declaração do procedimento PautaMouseDown da classe MainWindow

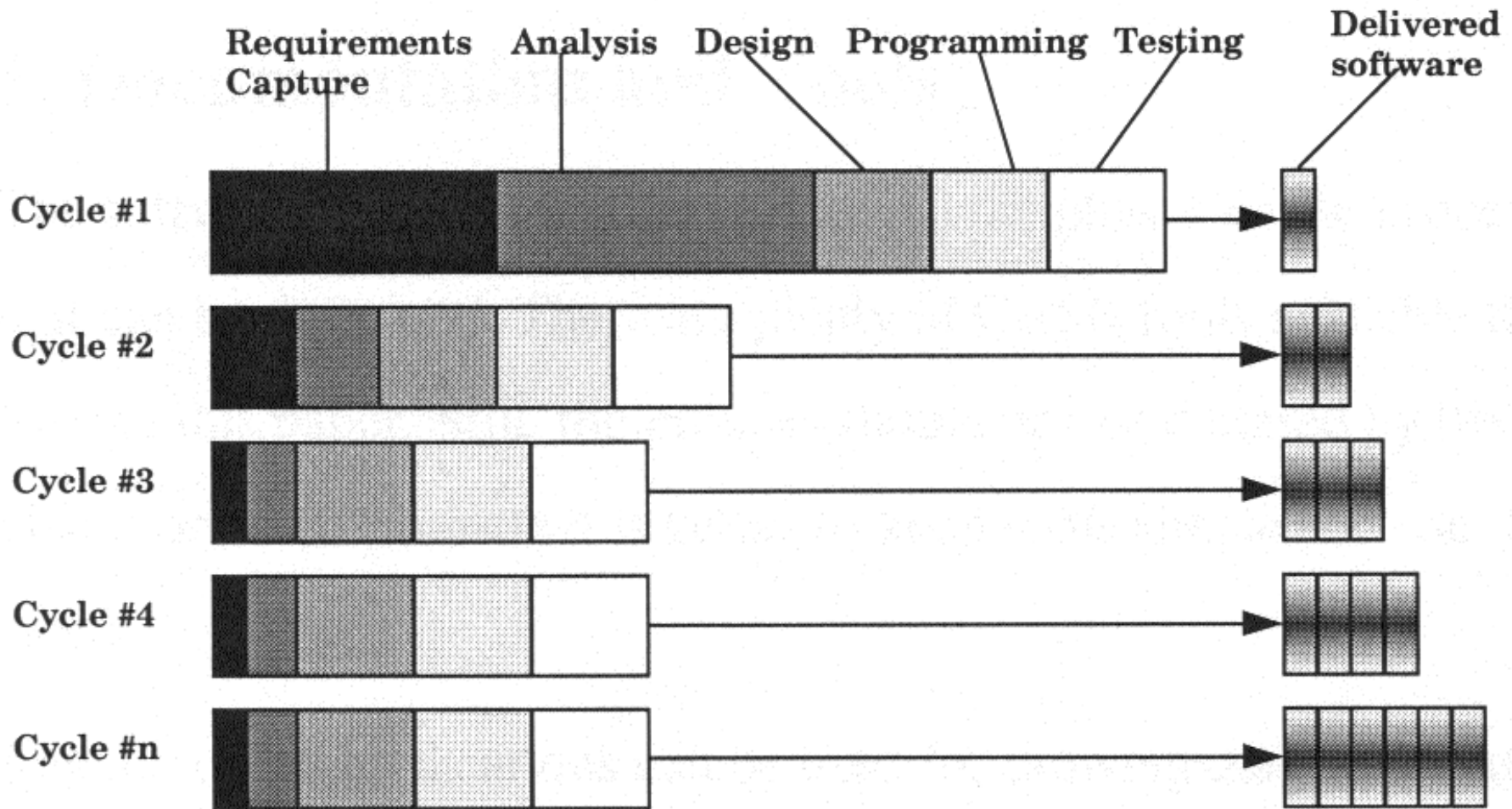
Testes

- O objectivo dos testes é descobrir erros e assegurar que o software funciona conforme o planeado. Portanto, os testes são executados conforme os requisitos. De acordo com o método Simplified, todos os *use case* do sistema e todos os requisitos não funcionais são verificados.

Desenvolvimento iterativo de software

- O desenvolvimento de software por iterações constrói o software peça por peça. Os sistemas são implementados em ciclos sequenciais e cada ciclo implementa apenas uma parte da funcionalidade requerida. Cada nova fase é construída em cima da fase anterior, e cada nova fase pode beneficiar da experiência obtida nos ciclos anteriores. Após cada ciclo é também possível validar os requisitos testando a parte implementada do sistema e até entregando versões aos clientes.

Desenvolvimento iterativo de software



Fim

- Perguntas?