A metodologia Simplified

António Rocha
ARocha@dei.isep.ipp.pt

1999/2000

1. Introdução 3
2. Requisitos para uma metodologia 5
3. A metodologia Simplified 6
  3.1 Notação 6
  3.2 Fases e paralelismo 9
4. Fases do método 11
  4.1 Captura de requisitos 11
  4.2 Análise 12
  4.3 Desenho 15
  4.3 Desenho 15
  4.4 Programação 22
  4.5 Testes 23
5. Desenvolvimento iterativo de software 24
6. Documentação e ferramentas 26
7. Conclusões 27

O metodologia OO Simplified

Uma metodologia para o primeiro projecto OO

1. Introdução

Existem vários livros de OO com diferentes metodologias e notações, que explicam como desenvolver software OO de "modo correcto", existem empresas a fazer consultoria e cursos sobre OO.

Informação sobre como desenvolver um software de um modo OO está disponível para quem tem tempo e disposição para estudar.

Na prática, as empresas/pessoas não conseguem dispender esforços para estudar, tirar cursos ou ter consultoria por um longo período de tempo, especialmente as pequenas e médias empresas que necessitam de um começo fácil, não podem contratar consultores especialistas em OO.

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.

Em vez de notações de modelação complexas e detalhadas são precisos métodos práticos com notações claras e um processo simples para descrições.

Estes métodos deverão ser suficientemente simples para a maioria das empresas o utilizarem, e devem ser fáceis de aprender.

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 ?

Não e não, a metodologia deverá primeiro preocupar-se em cobrir os aspectos mais importantes do desenvolvimento do software, só depois é que os detalhes deverão ser tratados.

As metodologias OO correntes não podem ser classificadas como simples e fáceis de entender porque tentam cobrir todos os pequenos aspectos do desenvolvimento, pelo que se tornam complexas.

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.

Portanto é necessário uma metodologia simples para principiantes e para equipas que trabalhem em pequenos projectos.

2. 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

O método 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.

3. A metodologia Simplified

3.1 Notação

A notação deste metodologia inclui três elementos principais :

- 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.

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.

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

È também uma boa prática dar nomes as associações que não sejam herança nem agregação. A fig.1 mostra os principais elementos da notação para um diagrama de classes.

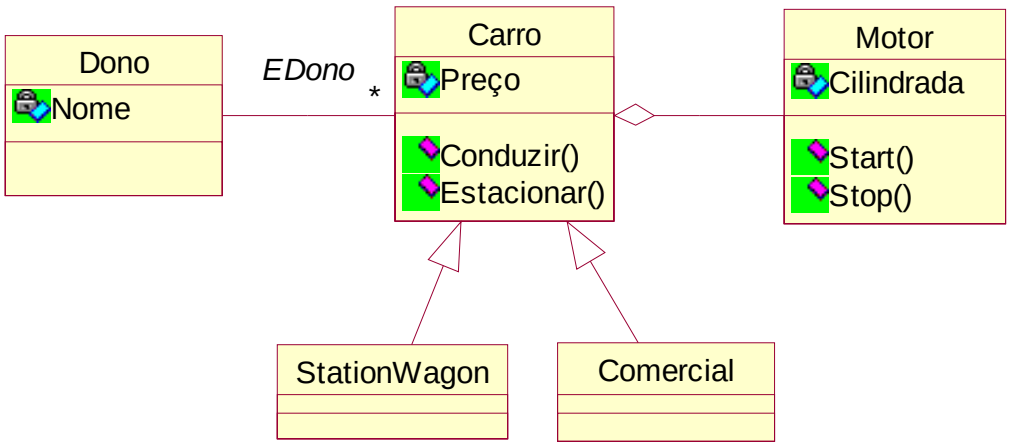

Fig.1 : Notação básica do UML para um diagrama de classes

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.
A fig.2 mostra os principais elementos da notação para um diagrama de sequência.

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

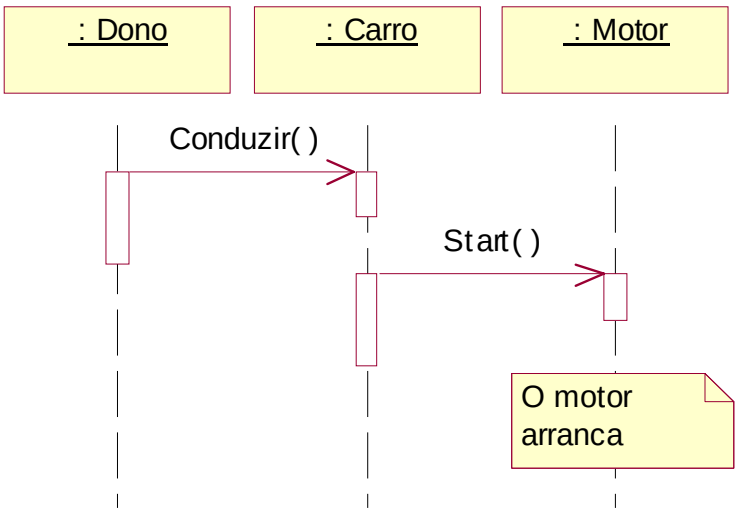

Fig.2 : Notação básica do UML para um diagrama de sequência

O nome dos objectos está no topo do diagrama, as setas são mensagens de um objecto para outro e começam num objecto *caller* e apontam para o objecto *called*. O nome de cada mensagem é escrita em cima de cada seta, e cada mensagem pode ter parâmetros. Podem também ser incluídos comentários.

3.2 Fases e paralelismo

A fig.3 mostra as cinco fases do método Simplified:

- 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 classes de cada vez.

- Testes
  Testa o sistema tendo em conta os requisitos.

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.

Estes dois caminhos estão relacionados entre si como mostra a fig.3.

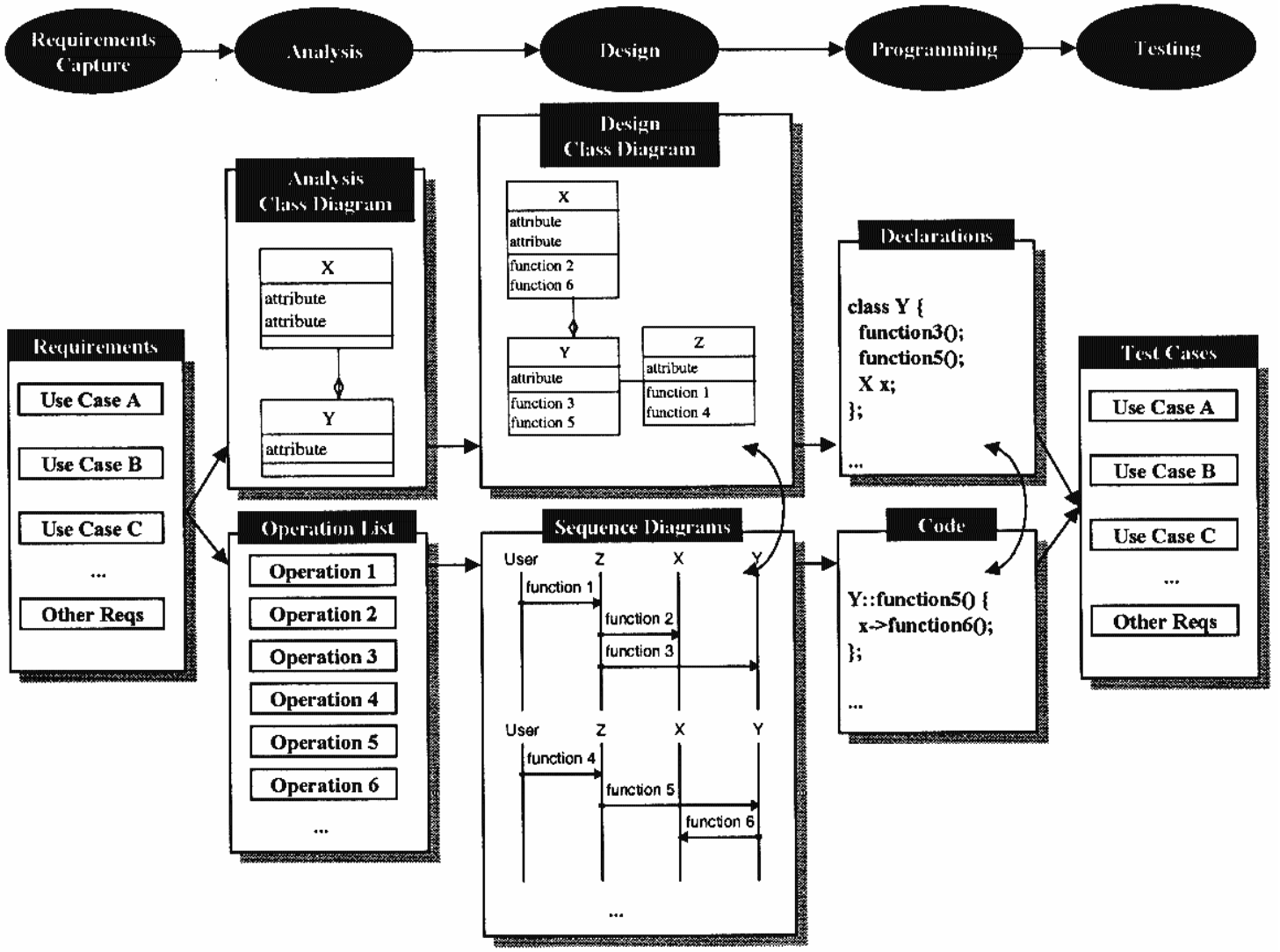

Fig.3 : O caminho estático e o funcional

4. Fases do método

4.1 Captura de requisitos

O processo de desenvolvimento de um sistema começa pela captura de requisitos. O objectivo desta fase é comunicar com os utilizadores finais e documentar os requisitos que têm de ser exactos.
Os requisitos podem ser divididos em funcionais e não funcionais. Os funcionais podem ser modelados através de *use cases*, que são textos que descrevem a utilização do futuro sistema. Para simplificação, o método Simplified não utiliza nenhuma da notação para *use cases*.

Para ilustrar as várias fases do método Simplified, vai ser utilizado como exemplo o desenvolvimento de uma aplicação que permite a estudantes de música (iniciais) compor músicas simples. A aplicação chama-se 'Compositor Elementar'.

A fig.4 mostra os requisitos funcionais do sistema, e a fig.5 mostra os requisitos não 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.

Fig.4 : Requisitos funcionais para o 'Compositor Elementar'

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

Fig.5 : Requisitos não funcionais para o 'Compositor Elementar'

O requisitos são discutidos com o cliente, se possível o cliente deverá participar na escrita dos *use cases*. Todos os *use cases* são escritos de um modo que o cliente possa perceber e possa fazer comentários. Esboços dos interfaces do utilizador podem por os *use cases* ainda mais esclarecedores.

Após os *use cases* e todos os outros requisitos estarem documentados e acordados com o cliente, formarão a base das fases seguintes do processo de desenvolvimento. Em cada passo, as fases do produto devem ser verificadas com os use cases e os requisitos não funcionais.

Os *use cases* formam também o conjunto de testes básicos para o teste do sistema.

4.2 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

- Analise 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.

A lista de operações e os diagramas de análise de classes são modelos separados, as operações incluem e usam os conceitos definidos nos diagramas de classes. Nesta fase não se deve tentar arrastar as operações para o diagrama de análise de classes, tentando adivinhar para onde. O diagrama de análise de classes inclui poucas operações, onde normalmente são estão presentes os atributos e as classes.

A fig.6 mostra o diagrama de análise de classes da aplicação 'Compositor Elementar'. A música é escrita em pautas, cada pauta consiste em notas, sendo também especificado o duração e tom de cada nota.
As notas podem ser de vários tipos, tais como quarter, half e quartes rests.

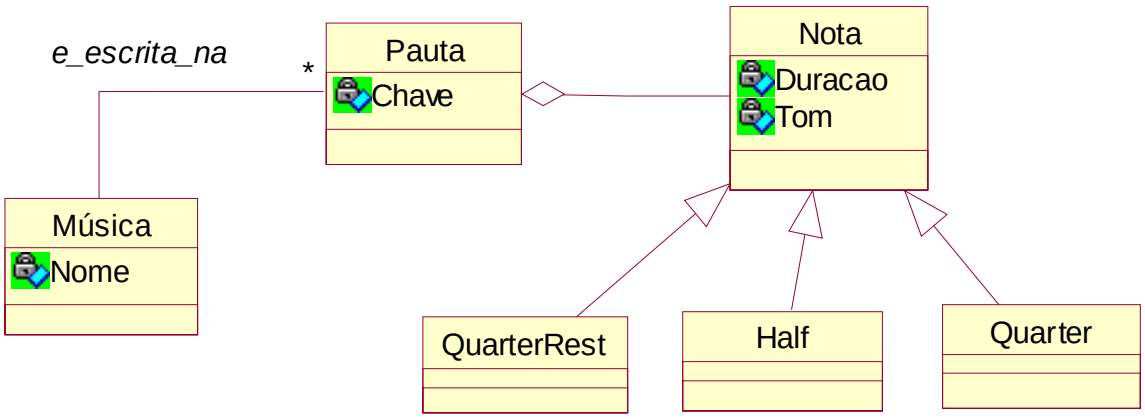

Fig.6 : Diagrama de análise de classes da aplicação 'Compositor Elementar'

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

Fig.7 : Lista de operações

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++).

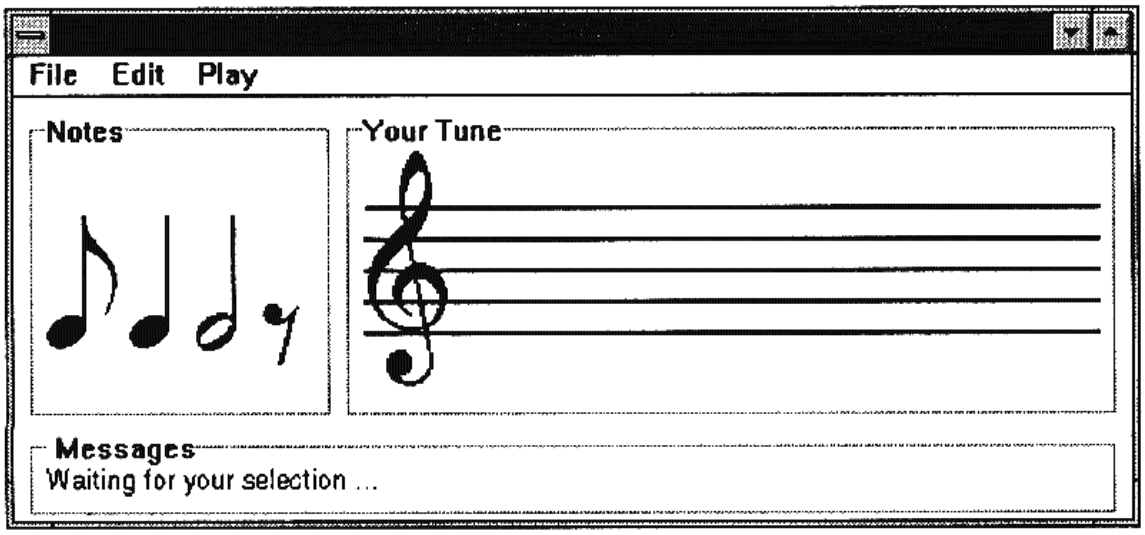

Fig.8 : A janela principal da aplicação

4.3 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

Na fase de análise os caminhos estáticos e funcionais estavam separados, mas na fase de desenho estes encontram-se. Os diagramas de sequência modificam os diagramas de classes, e os diagramas de classes fornecem objectos e atributos para os diagramas de sequência.

Um desenho com sucesso requer um bom conhecimento da linguagem de programação, sistema operativo, hardware e outros que afectem o código final.

Regras de desenho básicas, tais como manipular interfaces do utilizador e manipular bases de dados são, normalmente, ditadas pelas ferramentas escolhidas.

Para este exemplo vamos assumir a implementação numa linguagem Visual, que tenha classes para tratamento de interfaces. Normalmente, cada *window* ou *dialogue* é um objecto. Botões, menus e outros controles são também objectos.

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

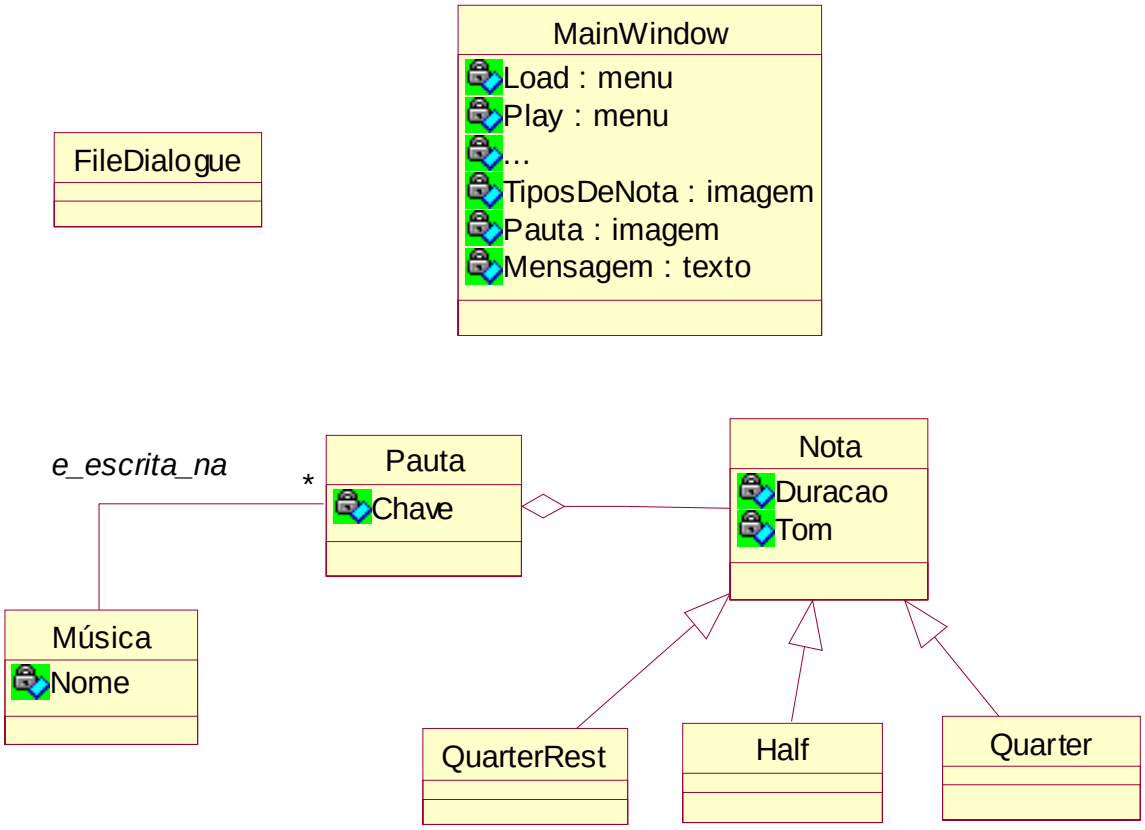

Fig.9 : Primeira versão do diagrama de desenho de classes

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 faze-lo vão ser refinadas as ligações entre as classes e vão ser adicionadas novas operações, atributos e classes.

Ao desenhar diagramas de sequência são especificadas as responsabilidades que cada objecto tem e como funcionam na prática. Uma seta começa no objecto que chama a funcionalidade de outro objecto e aponta para esse outro objecto. O nome da função é escrita em cima da seta. As funções podem ter parâmetros que ficam entre parêntesis, mas os valores de retorno não tem parêntesis.

Para o primeiro passo de refinamento do diagrama de classes, é desenhado um diagrama de sequência para ilustrar como é que os objectos comunicam entre si, a forma como o utilizador final coloca uma nota na pauta. A fig. 10 mostra como a operação ‘Colocar uma nota na pauta’ vai ser programada. Antes da operação começar, o utilizador tem de escolher um tipo de nota, este tipo é guardado no objecto MainWindow, esta acção não vai estar ilustrada.
Como primeira acção o utilizador efectua um click na pauta no ecrã. O click arranca com a operação da MainWindow ‘PautaMouseDown(x,y)’. Primeiro a função determina o tom da nota, mediante o cálculo da posição relativa do rato em relação à pauta. Depois o objecto MainWindow chama a operação ‘AdicionarNota(tipo, tom)’ do objecto Pauta, que por sua vez cria o objecto Nota, neste caso QuarterNote. A operação ‘AdicionarNota(tipo, tom)’ retorna a mensagem ‘ok’ se a operação tiver corrido bem. Finalmente a MainWindow pede o conjunto de todas as notas para redesenhar a pauta. Para isso a MainWindow chama o seu método ‘Redesenhar(notas)’.

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.

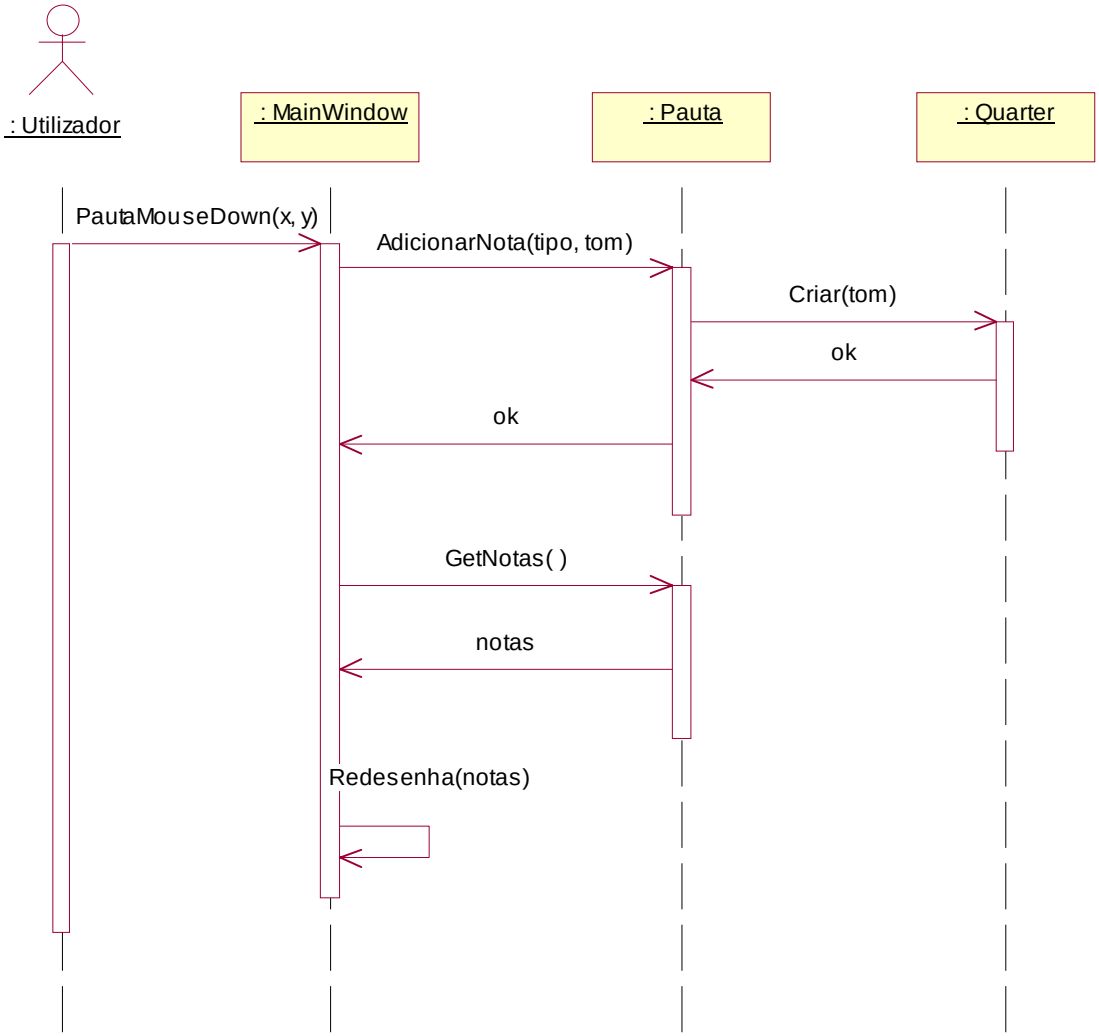

Fig.10 : Diagrama de sequência ilustrando a cooperação entre os objectos quando o utilizador coloca uma nota na pauta

Cada diagrama de sequência especifica um conjunto de métodos, atributos e associações e adiciona-os ao desenho do diagrama de classes. No entanto, os diagramas de sequência não mostram a comunicação dos objectos em todos os detalhes, em vez disso, devem ser simples e legíveis, devendo apenas referir-se aos modos de execução quando decorrem normalmente, assim, não é dada grande relevância as condições de excepção.

A fig. 11 ilustra a segunda versão do desenho do digrama de classes. As operações e ligações sugeridas pelo diagrama de sequência da fig.10 são adicionadas ao modelo.

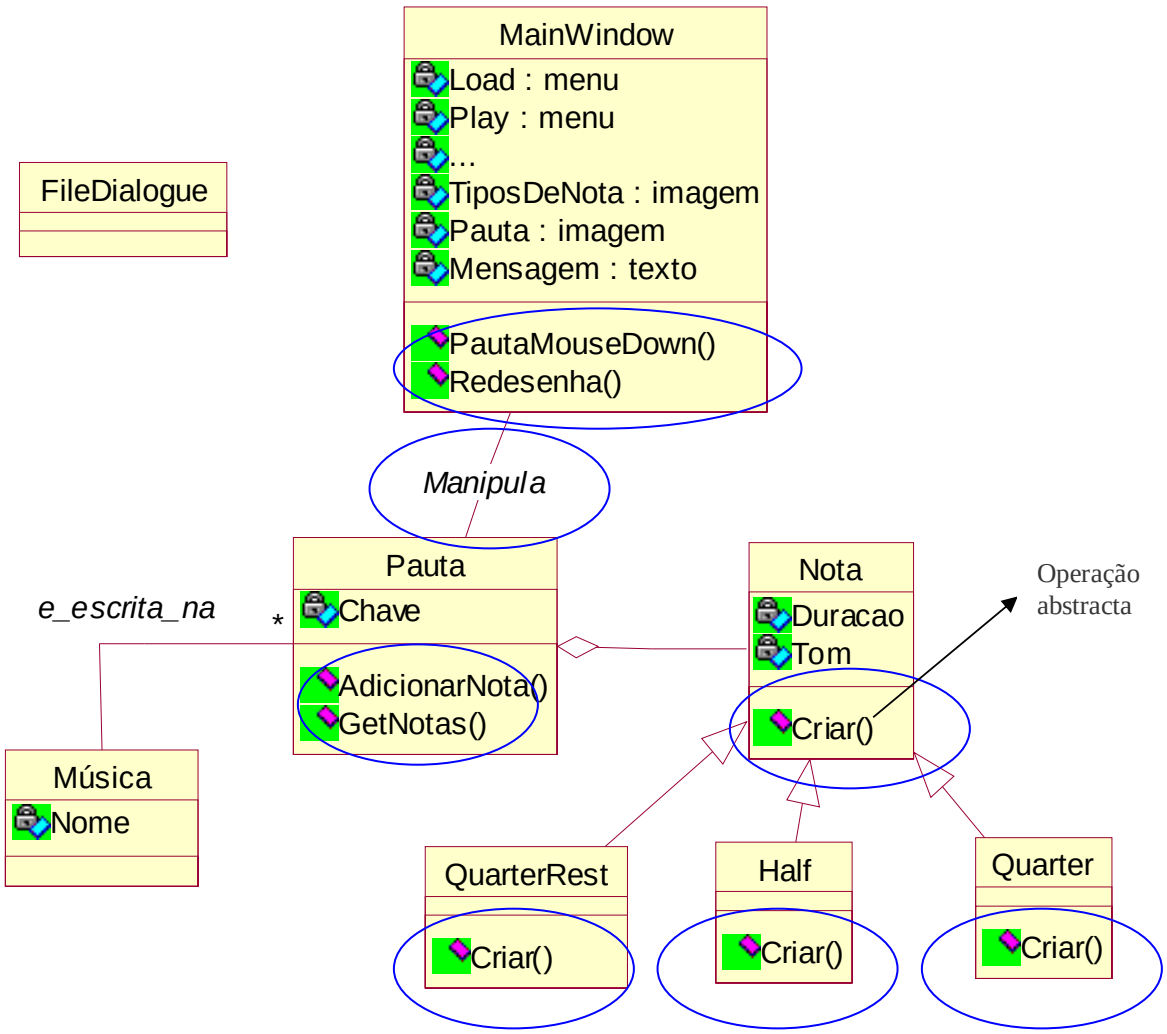

Fig.11 : Segunda versão do diagrama de desenho de classes

A figura 12 mostra como os objectos comunicam entre si quando o utilizador quer ouvir a sua música e, em seguida a fig. 13 apresenta a terceira versão do diagrama de classes, com as modificações sugeridas pelo segundo diagrama de sequência.

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.

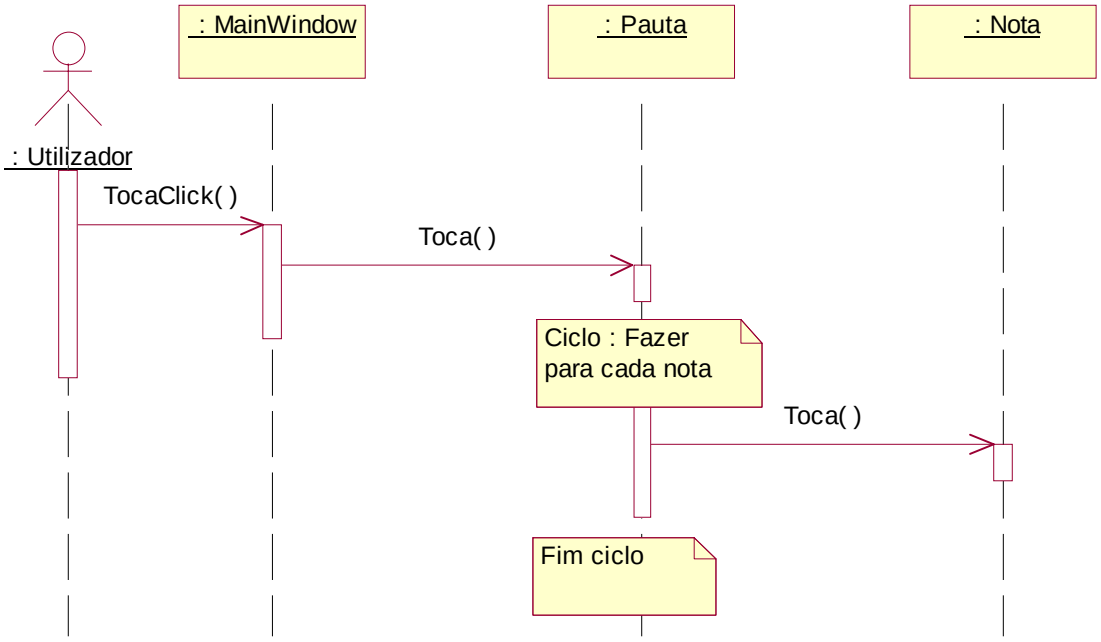

Fig.12 : cooperação entre os objectos quando o utilizador toca uma música

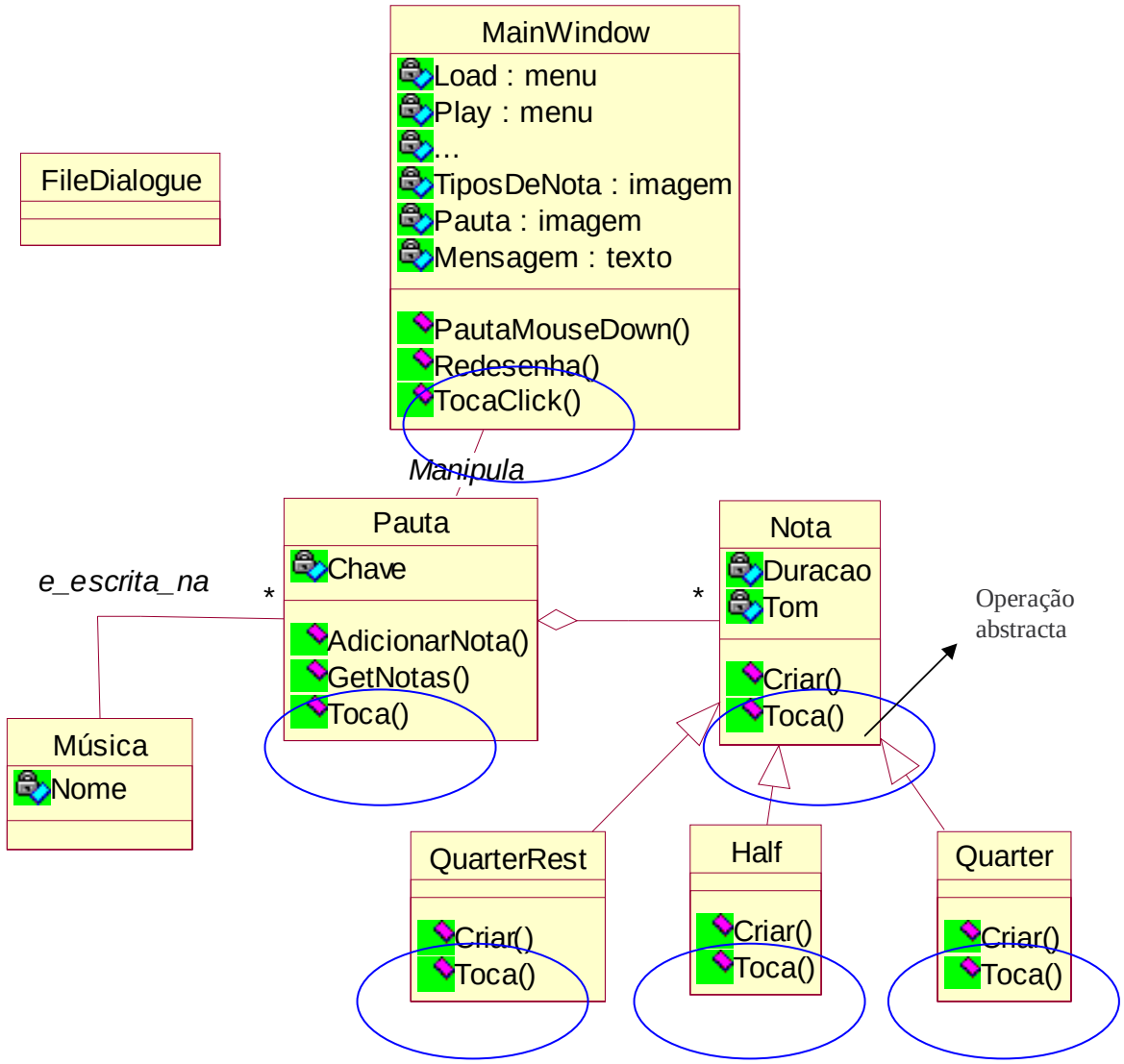

Fig.13 : Terceira versão do diagrama de desenho de classes

É necessário especificar todas as operações da lista de operações como diagramas de sequência e refinar o diagrama de classe de acordo com estas. Quando é desenhado o diagrama de sequência já está a ser pensada a implementação final, isto é, como é que o software vai funcionar na prática. Ao desenhar diagramas de sequência, é muitas vezes necessário adicionar especificidades ao diagrama de classes. A estrutura dos objectos especificada na fase de análise pode ter de ser alterada e podem também ser removidas as classes que não estão envolvidas em nenhum diagrama de sequência. O objectivo é desenhar classes que simulam objectos do real-world e que são, ao mesmo tempo, implementáveis.

4.4 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. Então, enquanto que o desenho se concentra na estrutura dos objectos, e cooperação entre eles, a programação lida com a funcionalidade interna dos objectos individualmente.
A fig. 14 mostra a declaração da classe Pauta, que é baseada no diagrama de classes da fig.13.
A fig. 15 mostra o código do método PautaMouseDown da classe MainWindows, está escrito em Object Pascal e é baseado no diagrama de sequência da fig. 10.

```
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 possivel adicionar mais notas';
        Notas :=MyPauta.GetNotas;
        Redesenhar(notas);
end;
```

Fig.15 : Declaração do procedimento PautaMouseDown da classe MainWindow

É importante que todas as classes, as suas interfaces e a cooperação entre objectos tenham sido modelados durante a fase de desenho. Com base nesse desenho o programador pode concentrar-se numa classe e numa operação de cada vez. Esta é a razão pela qual o método Simplified não utiliza nenhuma notação para a funcionalidade interna do objecto, na maioria dos casos não é necessário.

4.5 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. Existem várias ferramentas de teste que podem aumentar a qualidade dos testes. No entanto a tarefa mais importante nos testes é correr cada use case e comparar os resultados com o especificado nos use case iniciais, é aqui que podem ser descobertos os erros mais graves.

5. 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.

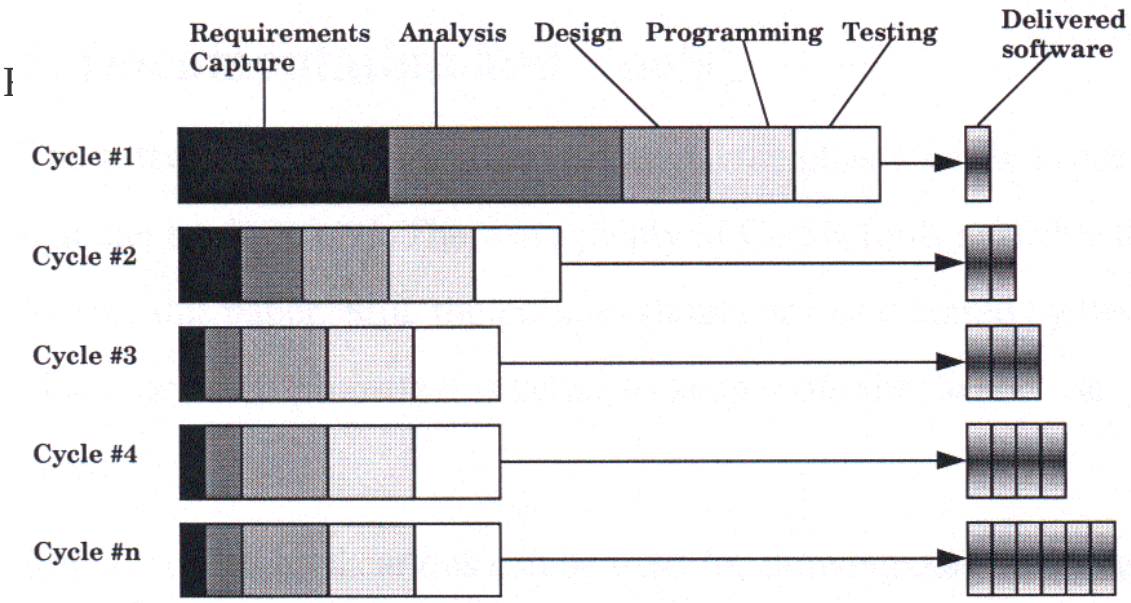

Fig.16 : Fases do desenvolvimento iterativo do software

A fig. 16 mostra as fases de desenvolvimento iterativo do software. É necessário capturar os requisitos e efectuar a análise cuidadosamente logo no primeiro ciclo por forma a assegurar que temos os objectivos correctos e bem definidos para os ciclos seguintes. Deverão ser escritos todos os *use cases* que são possíveis de especificação.

A análise tem de estar o mais completa possível durante o primeiro ciclo, caso contrário pode acontecer, facilmente, que as soluções do desenho não possam ser expandidas após o primeiro ciclo.
Também deve ser identificado o conjunto de operações que deverão ser implementadas durante o primeiro ciclo. A implementação destas

operações constitui a primeira versão do sistema. Esta versão, capaz apenas de executar um número limitado de operações, deverá ser dada ao utilizador final para comentários.

Tipicamente as operações de um único use case são um bom ponto de partida. O use case seleccionado para a primeira iteração deverá ser o mais importante, segundo o ponto de vista do utilizador final.

Apenas as operações seleccionados são transformadas na forma de diagramas de sequência durante o desenho, e apenas estas são programadas. Alguns objectos e operações que possam ser descobertos na fase de análise não são implementados. O próximo ciclo encarregarsse-á do resto das operações.

É essencial que o desenho seja baseado numa arquitectura sólida para que possam ser adicionadas novas classes e implementados novos métodos sem haver necessidade de voltar a implementar o que já foi feito nos ciclos anteriores.

6. Documentação e ferramentas

O processo de desenvolvimento de software tem de ser legível. Todas as fases do processo têm de produzir algo visual que possa ser discutido. Existem muitas ferramentas CASE disponíveis que suportam as notações usadas neste método, no entanto, o processo não deverá ser ditado pela ferramenta.

Todo o processo deverá estar bem documentado :

- Documento de requisitos – Tem de incluir todos os use cases e todos os requisitos não funcionais

- Documento da análise – Tem de conter o diagrama de analise de classes e a lista de operações, podendo também conter *windows* e *dialogues* do interface do utilizador e, possivelmente um protótipo.

- Documento de desenho – Tem de ter o diagrama de desenho de classes e todos os diagramas de sequência e, também todas as outras decisões de desenho, como por exemplo as soluções para a BD.

7. Conclusões

Quando uma empresa começa a desenvolver software numa perspectiva OO é necessário um projecto piloto. Durante esse projecto piloto a empresa pode aprender a nova tecnologia e adoptar novos métodos de trabalho.
De modo a tirar o maior proveito possível do projecto piloto, a empresa precisa de um método simples e poderoso para desenvolver o software, é o caso do método Simplified.
O método que consiste em cinco passos claros e uma notação simples cobre todas as fases desde a captura de requisitos até aos testes e inclui notações e descrições de processos. É uma metodologia completa que poderá ser expandida e modificada

O método Simplified é um *subset* do método OMT++ que é utilizado na Nokia Telecommunications e em outras empresas europeias. A metodologia OMT++ inclui outros mecanismos para desenvolvimento de sistemas maiores e mais complexos como, por exemplo, especificação de interfaces do utilizador.
O método Simplified poderá ser utilizado apenas em projectos pequenos/médios e é mais vocacionado para o primeiro projecto OO.

Existem riscos na adopção de um método simples para o desenvolvimento do software, pois a sua simplicidade pode falhar nas especificações chave do sistema. No entanto, o verdadeiro problema das metodologias existentes são a sua complexidade, um método tem de ser simples e intuitivo, e em vez de cobrir todos os detalhes do desenvolvimento do software, o método deverá, primeiro, apenas suportar as fases mais importantes. Só depois delas serem aplicadas na prática é que o método poderá ser expandido.