Infográfico

Uma linha de produção de *software* pode ser determinada pela especialização associada às funcionalidades do sistema. Essas funcionalidades podem ser modificadas de acordo com as regras de negócios de cada empresa. Para isso, existem notações que são usadas para criar modelos de funcionalidades (*features*, em inglês). Esses modelos têm representações gráficas que auxiliam na interpretação tanto sintática quanto semântica dos módulos, melhorando a capacidade de fazer alterações de acordo com os novos requisitos em uma linha de produção de *software*.

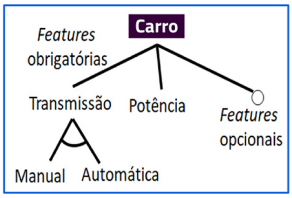
Veja, no Infográfico a seguir, as principais notações utilizadas para criar modelos de funcionalidades em uma linha de produção de *software*.

Em uma linha de produto de *software*, as características de um produto pertencentes a uma mesma família podem ser realizadas com uma **modelagem de *features*.** Podem-se modelar tanto as características comuns do sistema base quanto as consideradas específicas de cada produto em particular.

A seguir, você vai entender a evolução do sistema de uma forma visual.

Feature oriented domain analysis (Foda)

Definida na década de 1990, foi a primeira notação proposta para modelagem gráfica de funcionalidades, tornando-se base para as demais notações referenciadas por estrutura de árvore.

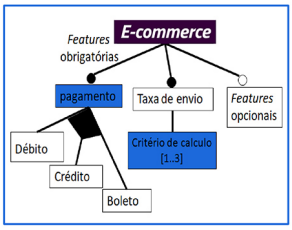


Esse tipo de notação faz a representação das funcionalidades via texto. As variabilidades das funcionalidades são interligadas por arestas. Outras associações, como agregação e composição, são representadas por quadro simples.

As *features* obrigatórias são as funcionalidades comuns e as opcionais, as específicas.

Czarnecki Eisenecker extend

Nome oriundo dos cientistas que propuseram a notação: Krzysztof Czarnecki e Ulrich Eisenecker. Essa notação teve origem nos anos 2000. Foi a primeira derivação da notação Foda, sendo talvez a mais utilizada.

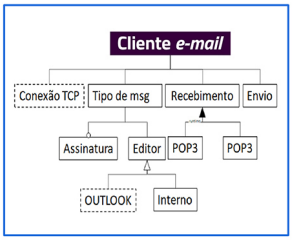


As funcionalidades são interligadas por arestas, mas essa notação difere da Foda, pois as funcionalidades são representadas por quadros. Podem-se representar *subfeatures* alternativas e opcionais. Além disso, faz uso da representação de cardinalidade em seus nodos.

Com o uso da cardinalidade, é possível representar *features* obrigatórias e opcionais, tanto na forma [1..1] e [0..1], respectivamente, como [0..2][6..6], indicando que essa *feature* pode estar 0, 1, 2 ou 6 vezes no produto.

Feature reuse-driven software engineering business (FeatuResb)

Essa notação é baseada na linguagem de modelagem unificada (UML – *unified modeling language*) e foi proposta por Bosh *et al.*, em 2000.

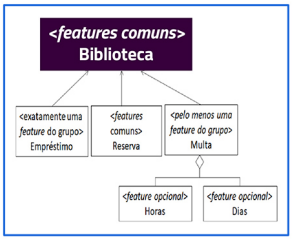


Um diferencial dessa notação é utilizar as arestas praticamente na horizontal, deixando os níveis da árvore de *features* bem distintos. Outra diferença é a representação de feature externa, indicada por um retângulo pontilhado.

Essas funcionalidades não fazem parte diretamente do sistema, mas são importantes porque ele as usa e depende delas para, por exemplo, representar sistemas operacionais que definam a forma de compilação do produto.

Gomaa

Essa é outra notação baseada na *unified modeling language* (UML), mas proposta no método Plus *(product lines UML-based software engineering)*. O nome foi também retirado do autor da notação, Hassan Gomaa.



Essa notação faz uso da representação UML, como as técnicas e componentes existentes nos diagramas UML na modelagem das *features*. As características e os grupos de características são representados como classes sem atributos e métodos, especificadas por estereótipos.

A organização do modelo é feita em formato de árvore, e os relacionamentos são expressos por setas, acompanhadas de um dos estereótipos *requires* ou *mutually includes*.

Logo, a modelagem de *features* está sendo cada vez mais utilizada para modelagem gráfica das funcionalidades dos componentes no desenvolvimento de aplicações em larga escala. Nesse cenário, torna-se uma ferramenta fundamental em uma linha de produção de *software*.

É uma linguagem de modelagem de propósito geral. O principal objetivo da UML é definir uma maneira-padrão de visualizar como um sistema foi projetado. É muito semelhante aos projetos usados em outras áreas da engenharia. A UML não é uma linguagem de programação; é, antes, uma linguagem visual. Usam-se diagramas UML para retratar o comportamento e a estrutura de um sistema. A UML ajuda engenheiros de *software*, empresários e arquitetos de sistemas com modelagem, projeto e análise.	O termo cardinalidade se refere ao número de membros cardinais (básicos) em um conjunto. A cardinalidade pode ser finita (um número inteiro não negativo) ou infinita. O termo é usado em outros modelos, como o de entidade de relacionamento para modelagem de banco de dados. Uma cardinalidade é simbolizada por números entre colchetes, como [0..3], que representa 0, 1, 2 e 3 opções na modelagem de funcionalidades (*features*).	Um estereótipo é representado pelos símbolos "<<" e ">>", sendo usado em representações, como UML, para classificar um componente. Ao modelar funcionalidades, por exemplo, usando notação Gomaa, podem-se ter estereótipos predefinidos, como <<Entity>> e <<Document>>, mas também desenvolvidos pela equipe de desenvolvimento, como <<textopersonalizado>>. Além disso, é possível utilizar estereótipos para demonstrar comportamentos de métodos como <<construtor>> ou <<getter>>.