# GitBob

## Estratégia usada na implementação

Decidimos para o desenvolvimento deste projeto dividir a implementação por partes, sendo que cada uma dessas partes corresponde a uma máquina que é um refinamento da máquina criada antes dela.   
Criámos 2 contextos diferentes, um para os Users, contendo o set USERS (com todos os users existentes), UTYPES (com todos os tipos existentes, no nosso caso BASIC e PREMIUM que são constantes do contexto) e UEMAILS (com todos os e-mails existentes). O outro contexto foi para os Files sendo que este contem os sets FILES (com todos os ficheiros existentes que podem ser adicionados ao GitBob) e MODES (com todos os modos possíveis de partilha de um ficheiro, no nosso caso REGULAR, SECURE, READONLY).

A divisão que fizemos foi a seguinte e pela seguinte ordem de refinamentos:

* GitBob, onde foi introduzido o conjunto de users registados, separados em dois conjuntos diferentes, um em que o user está associado ao seu tipo (registeredUserEMails) e outro em que user está associado ao seu e-mail(registeredUserUTypes). Nesta máquina é possível criar e remover um utilizador, bem como fazer upgrade/downgrade do seu tipo.
* GitBobWFiles, refinamento do GitBob, onde foi introduzido os ficheiros, criando por isso as seguintes relações: fileSize (FILES ⇸ ℕ) que associa um ficheiro activo ao seu tamanho, fileOwner (FILES ⇸ USERS) que associa um ficheiro activo ao seu criador, fileBMode (FILES ⇸ BOOL), que associa um ficheiro activo o seu estado de backup (utilizámos um booleano para representar o estado, visto que este apenas consiste em on e of), fileVersion (FILES ⇸ ℕ1), fileBackups (FILES ↔ ℕ1) que regista todas as versões de um ficheiro, desde que o seu modo de backup foi activado. Com a introdução dos ficheiros, foram implementados os seguintes eventos: addFile, removeFile, uploadFile, downloadFile, downloadFromArchive, turnOnBU, TurnOffBU.A razão que nos levou a separar os ficheiros dos utilizadores, foi a falta de dependência entre ambos.
* GitBoWFilesSharing, refinamento do GitBobWFiles, onde agora os ficheiros podem ser partilhados entre os vários users registados, assim como alterar o tipo de partilha dos ficheiros. Além disso, existe um arquivo, que contêm os ficheiros a pagados (o arquivo apenas foi introduzido nesta máquina, devido à dificuldade acrescida de manter os acessos que existiam no momento da remoção de um ficheiro do GitBob). Como tal, acrescentou-se as seguintes relações: fileModes (FILES ⇸ MODES), que associa um ficheiro activo a um tipo de partilha, fileSharing  (FILES ↔ USERS), que associa um ficheiro activo aos vários utilizadores, archFSize (FILES ⇸ ℕ), que regista no arquivo o tamanho do ficheiro na altura da sua remoção, archFAccess (FILES ↔ USERS), que regista no arquivo os users com acesso ao ficheiro na altura da sua remoção, archFVersion (FILES ⇸ ℕ1), que regista no arquivo a versão do ficheiro na altura da sua remoção. Como isto, foram implementados novos eventos como shareFile, removeShare, changeSharingMode e downloadFromArchive.
* GitBoWFilesSharingVersions, refinamente do GitBoWFilesSharing, onde foram introduzidos os repositórios locais. Apenas introduzimos agora os repositórios locais e o controlo de versões entre os repositórios e o GitBob, devido à complexidade de implementação do controlo de versões. Para resolver esta complexidade, introduziu-se duas novas relações, uma para guardar os ficheiros que o user tem no seu repositório, domainLF (FILES ↔ USERS), e outra para guardar a versão de cada ficheiro que o user tem, localFiles (domainLF ⇸ ℕ1). A função localFiles tem como domínio um par file user, pois cada um destes pares é único.