Plano de Desenvolvimento de Software

Baseado em [Tsiu&Karan2014] e [DI-IPSC81427A-00]

Versão 3.0 | 2016.01.09
RESOURCE TEAM

RESTRIÇÕES DE ACESSO

Documento público.

Contactos dos Autores

Nome do Autor	Email do Autor
Mª Mafalda Fernando	uc2013136659@student.uc.pt
Rui Pedro Ruivo	uc2012158444@student.uc.pt
Adriana Barbosa	uc2012138601@student.uc.pt
Jorge Luís Costa	uc2012149260@student.uc.pt
Ana Margarida Pereira	uc2012138634@student.uc.pt
João Guilherme Craveiro	uc2013136429@student.uc.pt

TABELA DE REVISÕES

Versão	Data	Autor	Descrição
3.0	2016.01.09	Resource Team	Correções em todo o documento de acordo com o *feedback* fornecido pelo professor, para entrega final
2.1	2015.11.29	Resource Team	Correções de forma/gramática
2.0	2015.11.27	Resource Team	Correção/atualização de secções já preenchidas; preenchimento da secção 4
1.2	2015.11.14	Resource Team	Arquitetura & Design
1.1	2015.10.31	Resource Team	Especificação de Requisitos
1.0	2015.10.18	Resource Team	Preenchimento inicial das secções, exceto a secção 4
0.2	2015.10.14	MZR	Additional information taken from reference [DI-IPSC81427A-00]
0.1	2015.10.07	MZR	Initial document, with the doc structure and a few guideline contents

Índice

Glossário

GSIIC	Gestão de Sistemas e Infraestruturas de Informação e Comunicação
IDE	*Integrated Development Environment* (Ambiente de Desenvolvimento Integrado)
PL	Prático-Laboratorial
RT	Request Tracker
SDP	*Software Development Plan* (Plano de Desenvolvimento de Software)
SO	Sistema Operativo
UC	Universidade de Coimbra

Lista de Figuras

Referências Externas

[DI-IPSC81427A-00] Software development Plan (SDP)

[DropBox] https://www.dropbox.com/

[Facebook] https://www.facebook.com/

[Linux Mint] http://www.linuxmint.com/

[Linux Ubuntu] http://www.ubuntu.com/

[Microsoft Office] http://www.products.office.com/

[PyCharm] https://www.jetbrains.com/pycharm/

[Python] https://www.python.org/

[RT] https://www.bestpractical.com/rt/

[SCRUM] https://www.scrum.org/

[SO09] What is a good software development plan?

[Sublime Text] http://www.sublimetext.com/

[Trello] https://www.trello.com/

[UC] http://www.uc.pt/

[VirtualBox] https://www.virtualbox.org/

1.Scope

.1 Introdução

Este documento é o manual de procedimento da equipa de Engenharia de Software "Resource Team", constituída por Mª Mafalda Sousa do Carmo Fernando, Rui Pedro Dias Ruivo, Adriana Pereira Barbosa, Jorge Luís Neto Costa, Ana Margarida de Matos Pereira e João Guilherme Assafrão Craveiro, estudantes da Licenciatura em Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Aqui descreveremos os nossos métodos de trabalho, a estrutura dos nossos processos e as ferramentas que utilizaremos. Referenciaremos artefactos 'vivos' com links externos, não os colocando diretamente no documento.

.2 Overview do Sistema

O software em desenvolvimento será aplicado no trabalho do *helpdesk* da Universidade de Coimbra (UC). Em particular, no contexto de gestão de tickets de serviço: o *helpdesk* recebe tickets de estudantes e funcionários da UC por linhas de suporte, faz a sua triagem de acordo com as suas características e tenta dar-lhes resposta da forma mais eficaz e eficiente possível. Para tal, utilizam o software *Request Tracker* (RT), da *Best Practical*, mantendo registo do estado dos tickets e aplicando uma variação do sistema de gestão de processos *Kanban*: uma abordagem sistemática ao processamento evolutivo e incremental.

O nosso cliente é a unidade de infraestrutura da GSIIC (Gestão de Sistemas e Infraestruturas de Informação e Comunicação) da UC, representada pelo Engenheiro Pedro Vale Pinheiro. Ele implementou um *dashboard* simples e básico para servir de *frontend* alternativo à interface do RT, dada a sua deficiente usabilidade, e abordou-nos procurando torná-lo adequado ao uso por parte da sua equipa.

O cliente tem grande experiência no campo da tecnologia, tendo feito uma descrição clara dos problemas com que se depara primeiramente, os quais motivaram o desenvolvimento deste *software*: o *dashboard* criado pelo Eng. Vale Pinheiro requer um sistema web que permita a sua utilização como plataforma de gestão de processos, através da atualização e visualização de dados presentes na base de dados do RT; necessitam também de acesso em tempo-real à atividade de processamento dos *tickets,* e da geração automática de relatórios de atividade, com estatísticas de trabalho e funcionamento. O cliente dá especial importância à qualidade da interação, procurando evitar o acesso direto à pobre interface do RT.

Uma vez desenvolvido na sua totalidade, este *software* deverá resolver os problemas mencionados acima, possivelmente integrando aspetos extra que o cliente poderá ou não requisitar mais adiante.

.3 Overview do Documento

Este documento é o Plano de Desenvolvimento de *Software* (SDP) que se centra nos requisitos e condicionantes do software, no seu modelo de processo, planeamento e calendarização, recursos e ferramentas. Uma vez que serão utilizados métodos ágeis de desenvolvimento, irá ser submetido a diversas alterações com o avançar do projeto.

O SDP segue a seguinte estrutura:

1. ***Scope*:** Descrição do documento, do que este inclui ou não, identificação do contexto de desenvolvimento do *software* e identificação dos elementos intervenientes.

2. ***Overview* de Requisitos e Restrições:** *Overview* dos requisitos e restrições que afetam o projeto, subdivididos em secções de acordo com o trabalho a realizar.

3. **Descrição do Modelo de Processo de *Software*:** Descrição do modelo de processo de *software* adotado pela equipa e seu funcionamento.

4. **Planeamento do Projeto:** Descrição dos vários aspetos do planeamento das atividades de desenvolvimento do *software* e identificação de riscos/incertezas associadas, assim como as técnicas previstas para a sua mitigação.

5. **Calendarização do Projeto:** Calendarização do projeto segundo as metas estabelecidas, sujeita a alteração conforme a atualização destas.

6. **Organizações e Recursos do Projeto:** Estrutura interna do projeto, organizações envolvidas e relações entre elas, recursos (humanos e materiais) e suas responsabilidades.

7. **Ferramentas:** Lista das ferramentas adotadas para o desenvolvimento do *software* e gestão da equipa

8. **Anexos:** Descrição sucinta dos perfis dos membros da equipa, SCRUM *Charts*, diagramas (de classes e de fluxo) e *Mockups*

2. *Overview* de Requisitos e Restrições

.1 Requisitos e Restrições no Sistema e no *Software* a Ser Desenvolvido

O *software* a ser desenvolvido deverá implementar uma *interface* que permita a utilização do RT sem acesso direto à *interface* do mesmo, com acrescento de uma componente de registo estatístico.

.1 Requisitos Técnicos

Partindo do código funcional fornecido pelo cliente, foram identificados os seguintes requisitos funcionais:

1. Remover os comandos *back/forward* implementados nos URLs, para que os tickets não sejam movidos com cada carregamento da página
2. Remover o link http://suporte.uc.t implementado no URL para que carregar num *ticket* redirecione o utilizador para o RT atual
3. Correr os *scripts* Python de atualização sem necessidade de recorrer à linha de comandos

4. Substituir as contas de utilizador já existentes por contas de membros da equipa
5. Visualizar as estatísticas corretamente
6. Aceder à descrição de um *ticket* diretamente dentro do *dashboard* RT (sem recorrer à plataforma RT)

.1 Requisitos Não Funcionais

Foram identificados os seguintes requisitos não funcionais:

1. *Interface* organizado de acordo com o contexto em que o *software* será utilizado (ie, tendo em conta que será utilizado em ambiente de trabalho, tendo de haver um equilíbrio entre apelo visual e motivação)
2. Minimização da curva de aprendizagem do utilizador (*software* simples de usar)
3. *Performance*

.2 Requisitos e Restrições na Calendarização e Recursos do Projeto

A meta fixa do projeto é o final do semestre, Dezembro de 2015, pelo que o *software* deverá estar completamente implementado e cumprir todos os requisitos (já definidos e a definir) até essa data.

O desenvolvimento rege-se por metas móveis, em cada uma das quais deverá ser entregue uma nova parte do que está a ser desenvolvido. Terá de haver uma (difícil mas) eficiente gestão do tempo, dado que não só dispomos de pouco, mas também temos de o distribuir não unicamente pela cadeira de Engenharia de *Software*. O tempo é, portanto, uma forte restrição.

Em termos de recursos, o projeto requer, para além dos temporais, recursos físicos (máquinas com sistema operativo UNIX ou imagem deste, espaços para reuniões e trabalho em grupo) e intelectuais (capacidade de programação *frontend* e *backend,* agilidade na manipulação, compreensão e construção de código em Python, compreensão dos requisitos do cliente e do que este já implementou, entre outros).

.3 Outros Requisitos (Segurança, *Performance,* Usabilidade…)

O *software* deverá ser de fácil utilização, simples, direto, acessível e prático, minimizando o quanto possível a curva de aprendizagem do utilizador (usabilidade).

Deverá ainda apresentar uma boa *performance*, ou seja, não só cumprir os requisitos acima enumerados, mas fazê-lo de eficiente (i.e., mantendo uma boa relação entre os recursos que consome e os resultados que apresenta).

3.Descrição do Modelo de Processo de *Software*

Um método ágil é uma estrutura conceptual para gerir projetos de Engenharia de *Software.* Estes métodos enfatizam a comunicação em tempo real, preferencialmente cara-a-cara. Valorizam a satisfação do consumidor, entregando rápida e continuamente *software* funcional, em comparação com os métodos tradicionais, nos quais a entrega de *software* é feita em meses ao invés de semanas (como ocorre nos ágeis). Estes métodos devem ser simples, adaptar-se rapidamente à mudança, privilegiando um maior conteúdo de *software* funcional em contrapartida com a documentação extensa. Em suma, estes métodos devem responder a mudanças mais do que seguir um plano

O modelo escolhido pelo grupo foi o SCRUM. Em gestão de processos ágeis, este modelo tem por base um planeamento não linear, uma elaboração mais exaustiva e focada na agregação do valor para o cliente, gerindo os riscos e fornecendo um ambiente seguro.

No SCRUM existem três papéis principais: *Product Owner, Scrum Master* e *Scrum Team.*

No nosso projeto, o *Product Owner* é o professor da cadeira: ele fala e decide, com o cliente, o que deve ser mudado a cada meta. É também ele que tem acesso direto aos conteúdos e resultados produzidos.

O Sc*rum Master* é a Mafalda, responsável por:

- Ajudar a equipa a chegar a um consenso acerca do que será alcançado em dado período de tempo
- Ajudar a equipa a chegar a um consenso durante as reuniões presenciais ou em questões debatidas diariamente via *Facebook*
- Ajudar a equipa a manter o foco e a seguir as regras que regem os processos de trabalho
- Analisar e procurar corrigir problemas que impeçam o progresso da equipa

Por último, temos a *Scrum Team:* os membros da equipa, cujo objetivo é o desenvolvimento de cada meta.

Funcionamento do Modelo SCRUM

O *Product Owner* recebe da parte do cliente a próxima meta a cumprir. Este comunica-a à equipa e avalia o progresso do grupo. A equipa decide quais as tarefas a cumprir no imediato – dada a duração curta das metas, cada uma é vista como um *sprint*.

Com o decorrer da meta e definidas as tarefas distribuídas pelo grupo, o *Scrum Master* comunica diariamente com a equipa para verificar e controlar os avanços conseguidos. São realizados *charts* para uma melhor visualização do trabalho realizado.

Por fim, na entrega da meta a equipa faz uma revisão do trabalho desenvolvido e a análise do que tenha corrido melhor ou pior para na próxima meta melhorar.

4.Planeamento do Projeto

.1 Planeamento e Monitorização do Projeto

A equipa de trabalho organiza-se maioritariamente via *Facebook,* tendo um grupo privado específico para o projeto. Por essa via é feita não só partilha de informação, como recolha de métricas (nomeadamente o número de minutos de trabalho semanal), marcação de reuniões ou eventos de *team building*, votações caso haja uma decisão a ser tomada e partilha de alguns ficheiros.

O principal meio de partilha de ficheiros e gestão de versões é a plataforma *Dropbox.*

O principal meio de registo de planeamento das atividades é a plataforma *Trello*, onde o grupo tem implementado um esquema de processo tipo *Kanban* para organização das tarefas definidas segundo o modelo SCRUM.

Devido à natureza do projeto (métodos ágeis; metas semanais), as atividades são planeadas semanalmente. Todas as semanas são revistas metas técnicas na aula PL, sendo aí feita uma primeira distribuição de tarefas a realizar para a semana seguinte. Para além disso, é ainda marcada, quando possível, uma reunião presencial para a semana em questão.

As reuniões semanais são marcadas pela Mafalda Fernando, e nelas se reveem as tarefas em progresso/concluídas da semana e da semana anterior. O plano das reuniões é elaborado previamente pelo Rui Ruivo, estando a Margarida Pereira encarregue de registar a ata (guardada na *Dropbox*, para possíveis alterações por parte de outros membros e gestão de versões) e o Jorge Costa de reservar uma sala no Departamento de Engenharia Informática, onde possam ter lugar.

De acordo com as metas técnicas e de desenvolvimento definidas, cada elemento do grupo fica responsável por dada tarefa, sendo que tem normalmente o prazo de uma semana para a cumprir. Uma vez que as reuniões se dão a meio da semana, são revistas as tarefas concluídas da segunda metade da semana anterior e primeira metade da atual, a partir da apresentação na aula PL.

Uma tarefa é dada como concluída quando é acordado por todos os membros do grupo que cumpre os requisitos para ela estabelecidos. Uma tarefa de cariz técnico (por exemplo, uma funcionalidade) tem de ser executada sem erros, passar em testes definidos pela equipa e pelo professor com base nos possíveis contextos de execução, e o mais rápida e eficientemente possível (sendo isto assegurado pela revisão do código por parte dos elementos do grupo). Uma tarefa de cariz organizacional (por exemplo, o preenchimento de uma secção do SDP) considera-se concluída após revisão e aprovação por parte de todos os membros do grupo (e, eventualmente, por parte do professor).

As métricas recolhidas são o número de tarefas concluídas por semana e o número de minutos gastos por cada elemento do grupo no desenvolvimento do software por semana (submetido no formulário fornecido pelo professor pela Mafalda, após recolha dos valores por inquérito no grupo do *Facebook*). No caso da recolha do número de minutos gastos por cada elemento por semana, caso haja algum dado em falta é completado com a média de entre os fornecidos.

.2 Ambiente de Desenvolvimento de *Software*

O *software* está a ser desenvolvido no sistema operativo *Linux Mint,* em imagem virtual na plataforma *VirtualBox* da *Oracle*, com o IDE *Pycharm* em conjunto com o software de edição de código *Sublime Text*, na linguagem de programação Python. O desenvolvimento é centralizado, sendo que o sistema em que o *software* se insere é um sistema distribuído.

Os artefactos intermédios são testados em duas máquinas dos elementos do grupo (na de desenvolvimento, com SO *Linux Mint*, e numa com SO *Linux Ubuntu*), através da simulação de diversas situações de aplicação real, tais como a criação, movimento e tratamento de *tickets*, a criação e autenticação de utilizadores, entre outras. É feita a apresentação das funcionalidades técnicas nas aulas PL.

Não haverá *software* aparte da entrega final.

.3 Gestão de Requisitos de *Software*

Os requisitos do *software* são recolhidos a partir da descrição do problema por parte do cliente, de indicações dadas pelo professor e de perguntas que são enviadas aos gestores de cliente para que possam obter esclarecimentos do mesmo (partilha de informação no fórum de discussão da disciplina Engenharia de *Software*, na plataforma *Inforestudante*).

Após a recolha da informação, o grupo em conjunto analisa e define os requisitos, podendo estes sofrer alterações com maior ou menor frequência. Os requisitos são registados e guardados num documento partilhado na *Dropbox*, subdividindo-se em funcionais e não funcionais, e sendo organizados do geral para o particular.

Os requisitos mais específicos são tratados primeiro, segundo uma abordagem *bottom-up*, sendo definidas tarefas com base neles. Quando esse conjunto de tarefas é dado como concluído (segundo os critérios de conclusão de tarefas enunciados anteriormente), considera-se que o *software* cumpre o requisito (ou sub-requisito), possivelmente após testes adicionais.

.4 Design de *Software*

Do ponto de vista arquitetural, o *software* integra-se num sistema do tipo cliente-servidor, sendo o navegador *web* o cliente e tendo como servidor final o RT. O *dashboard* implementado surge como servidor intermédio, encarregado de redirecionar páginas *web*.

O *dashboard* tem as seguintes classes principais:

- ***UserAuth* –** classe responsável pela autenticação dos utilizadores, verificando dados como *username*, *email* e *id*
- ***DITICConfig* –** classe que guarda dados dos utilizadores, *tickets* e configurações do sistema
- ***RTApi* –** *interface* com o servidor do RT
- ***ProjectObject* –** classe que representa um objeto na árvore do projeto, e seus atributos
- ***ManageProjectElements* –** classe que permite a manipulação da árvore de elementos do projeto
- ***ProjectElementFastSearch* –** classe que pesquisa um elemento na árvore do projeto

O diagrama de classes e relações entre elas encontra-se em anexo no documento.

.5 Design de Interação

Do ponto de vista dos processos, há dois fluxos definidos: a *pipeline* de um *ticket* e o *workflow* do utilizador (ambos em anexo).

Aquando da criação de um novo *ticket*, caso tenha sido colocado "INBOX" no Custom Field "IS – Informatica e Sistemas" este deverá aparecer no *dashboard* na lista de *tickets* da tabela INBOX.

Nessa tabela, um utilizador pode alterar a prioridade de um *ticket* ou marcá-lo como urgente. Caso um utilizador decida tratar esse *ticket*, deverá premir a opção *Take*, o que o promoverá a *owner* do *ticket* em questão.

O *ticket* ao qual foi feito *Take* é colocado na coluna IN da tabela *Kanban*, podendo ser alterada a sua prioridade, podendo avançar para ACTIVE (operação *forward*) ou ser recolocado na tabela INBOX (operação *back*).

Em ACTIVE encontram-se os *tickets* em resolução. Caso haja a necessidade de aguardar por algum recurso/evento, um *ticket* nesta tabela pode "posto em espera", sendo o seu processamento pausado e passando este para a coluna STALLED. Aí, o *ticket* pode ser recolocado na pipeline principal (*back*).

Novamente em ACTIVE, o *ticket* pode ser marcado como resolvido e colocado na coluna RESOLVED (*forward*). Uma vez aí, é assumido que esse *ticket* foi terminado e pode ser eliminado posteriormente, embora possa ser reaberto para revisão ou para novo processamento (*back*).

Do ponto de vista do utilizador, este é direcionado para a página *Home*, de onde, caso tenha já fornecido as suas credenciais para autenticação, segue para a página INBOX, o seu ambiente de trabalho. Caso contrário, é-lhe pedida a autenticação e só depois prossegue.

Qualquer opção de configuração direciona o utilizador para o RT. Caso este pretenda criar um *ticket* (*Create Ticket*) ou selecione um *ticket* para consultar detalhes, é para as páginas específicas para os efeitos que é direcionado.

.6 Implementação de *Software* e Teste de Unidade

O código está organizado em seis classes: *UserAuth, DITICConfig, RTApi, ProjectObject, ManageProjectElements* e *ProjectElements*. Como os próprios nomes indicam, a primeira trata da autenticação do utilizador do *dashboard*, a segunda das configurações do sistema, a terceira da comunicação com o RT e as últimas três dos elementos do projeto, organizando-os em forma de lista. As relações entre as classes encontram-se nos diagramas de classes, em anexo.

Cada unidade alterada/acrescentada será sujeita a testes, dependendo estes da sua natureza e estabilidade, em conjunto com as restantes. Gradualmente, os testes de unidade e as revisões efetuadas serão mais exigentes e abrangentes, culminando com a conclusão de que a unidade em questão passa e, para além disso, o requisito a ela associado é dado como satisfeito/a tarefa concluída.

Defeitos encontrados são procurados no código, identificados, analisados, discutidos e corrigidos individualmente. Não são utilizadas ferramentas adicionais de *debugging*.

.7 Integração e Teste de Unidade

Como referido acima, cada unidade é sujeita a testes específicos, pensados em conjunto pela equipa ou definidos pelo professor, para a sua funcionalidade, uma vez alterada/desenvolvida. Estes testes envolvem todo o sistema, embora não todas as suas funcionalidades.

A integração de todas as unidades não envolve nenhum processo específico, uma vez que os próprios testes envolvem todo o sistema e o conjunto dos seus componentes.

.8 Integração e Teste de Alto Nível

O sistema é iterativamente desenvolvido e cada alteração é testada e melhorada até cumprir o requisito ou sub-requisito a que corresponde. A abordagem *bottom-up* e subdivisão dos requisitos permite que os requisitos mais gerais e abrangentes vão sendo construídos com o cumprimento dos mais pequenos, pelo que a sua validação estará implícita no software final.

Os testes ao produto final abrangerão todas as funcionalidades do sistema, nomeadamente:

- Criação/eliminação de *tickets*
- Criação de novos utilizadores
- Atualização e apresentação das estatísticas de trabalho individuais
- Atualização e apresentação das estatísticas de trabalho coletivas (para apresentação no ecrã *master*)
- *Forward* e *Back* de tickets em todos os estados nos quais tal é permitido
- *Stall* e *Interrupt* de *tickets*, no estado ACTIVE
- Consulta de detalhes de *tickets* em formato *pop-up*
- Fecho de *tickets,* com *pop-up* para justificar a ação

Serão realizados testes em diferentes máquinas com SO *Linux,* de forma a garantir a independência dos resultados e portabilidade do código.

Nesta fase, defeitos menores serão corrigidos, quando possível, e defeitos de maior importância serão minimizados, se possível. Caso sejam identificados defeitos não tratáveis no imediato, serão registados e dados a conhecer na entrega do produto final.

.9 Avaliação do Produto

A avaliação final do produto dependerá não só do seu desempenho (se passa ou não em todos os testes efetuados, se é rápido na execução das tarefas, se não consome demasiados recursos, se tem bom desempenho em máquinas com SOs diferentes), como também do seu *interface* e usabilidade. Conhecendo o ambiente e contexto em que o *software* será utilizado, e com base nas indicações do próprio cliente, será avaliado o resultado final na sua totalidade por parte da equipa de trabalho e, ultimamente, pelo professor da cadeira Engenharia de *Software.*

.10 Entrega

O produto final será entregue ao cliente (por meio do professor) sob a forma de código-fonte, a ser executado diretamente.

Juntamente com o código, será entregue ao professor este artefacto, o SDP, na sua versão final mais completa, em formato PDF.

.11 Gestão de Configuração de *Software*

As alterações efetuadas por cada elemento da equipa ao longo do desenvolvimento do *software* são coordenadas e sincronizadas nas reuniões realizadas/via *Facebook*. A comunicação eficaz entre os elementos do grupo impede que haja conflitos de concorrência, e, dada a própria heterogeneidade da equipa, geralmente não há mais do que um elemento a tratar de dada unidade de código.

O código completo é guardado na *Dropbox*, que permite a gestão das suas versões e recuperação de versões anteriores, sendo registadas também as alterações efetuadas em relação à versão anterior. Não há atualização dos ficheiros sem que todos os elementos do grupo tenham conhecimento das alterações que se pretendem efetuar e as aprovem.

Dada a dimensão do projeto, não há a necessidade de muitos artefactos de desenvolvimento, pelo que a sua gestão e identificação é relativamente simples.

A confiança entre os membros da equipa é o principal meio de assegurar que não há quebra de regras ou violação dos processos definidos.

.12 Planos de Garantia de Qualidade

Há duas entidades que contribuem para a garantia de qualidade dos artefactos intermédios e finais: o professor e a equipa.

A equipa averigua e melhora a qualidade do *software* através dos testes que efetua e a qualidade dos restantes artefactos (SDP, registo de métricas, plataforma *Trello* e *charts* de calendarização) com base no consenso entre os seus elementos.

Nas aulas PL semanais, o professor confirma a qualidade do *software* após cada alteração realizada (pelo menos, a nível de interface e funcionamento), e revê também artefactos como o SDP, fornecendo *feedback* que é então analisado pela equipa e adaptado no melhoramento da qualidade final.

.13 Planos de Ação Corretivos

Caso surja a necessidade de realizar alterações no *software* já desenvolvido (por alteração de requisitos do cliente, por exemplo), é criado um *backup* da versão atual e funcional, e a partir desta são definidas as alterações específicas a realizar e as tarefas a atribuir a cada elemento da equipa.

No caso de atraso em dada tarefa/requisito, caso seja possível, mobilizam-se membros da equipa que de momento não tenham tarefas a tratar ou tenham tarefas de menor importância, de forma a dar assistência a quem está responsável pela parte atrasada. Caso seja, ainda assim, impossível compensar o atraso até à meta em questão, procura-se compensar com esforço extra/nova organização de tarefas para a meta seguinte.

Embora possam surgir exceções, são adotadas as mesmas estratégias de adaptação caso surjam problemas relacionados com pessoas, com processos ou com tecnologias; sendo que, no caso dos processos, os problemas podem envolver a adoção de novos processos de trabalho ou mesmo a reorganização da equipa.

.14 Gestão de Risco

A gestão de risco é distribuída por toda a equipa, sendo que depende das tarefas atribuídas iterativamente pelos membros. Cada membro está encarregue de gerir os riscos relacionados com a tarefa pela qual está encarregue, e apenas em casos excecionais se recorre a toda a equipa para tratar de certo fator de risco.

5.Calendarização do Projeto

Como o projeto segue uma metodologia ágil, existem várias metas ao longo do desenvolvimento do projeto, cada uma durando entre uma e duas semanas. Uma vez que o modelo de processo de *software* adotado se baseia no SCRUM, há uma calendarização iterativa para cada *sprint* (meta), sofrendo esta alterações progressivas.

São atualizados dois *charts: velocity chart* e *burndown chart* (ver anexos), que dão à equipa uma noção geral da sua progressão e capacidade de trabalho, essenciais para o planeamento das atividades para cada *sprint.*

Meta 1	Meta 2	Meta 3
Especificação do *scope* do *software*	Especificação de Requisitos do Software	Especificação da arquitetura e *design* do software
Overview de requisitos e restrições	Correção dos *mockups* iniciais	Implementação de um *pop-up* para visualização da descrição de um *ticket*
Descrição do modelo de processo de *software*	Início da adaptação da interface aos *mockups*	Continução da adaptação da interface aos *mockups*
Calendarização do projeto	Implementação da criação de *tickets* a partir do *dashboard*	
Especificação das organizações e recursos do projeto		
Anexos do SDP		
Remoção de *back* e *forward*		
Remoção de 'http://suporte.uc.pt'		
Automatização dos *scripts* de atualização		
Substituição das contas de utilizador existentes pelas de membros da equipa		
Visualização correta de estatísticas		
Mockups iniciais		

Meta 4	Meta 5	Meta 6
Especificação do planeamento e monitorização do projeto	Testes finais	Análise do *feedback* fornecido pelo professor
Especificação do ambiente de desenvolvimento do *software*	Demonstração ao vivo perante o professor	Correção do *Software Development Plan*
Descrição dos processos de gestão de requisitos do *software*	Organização do código para a entrega final	Entrega final do SDP
Descrição do *design* do *software*	Entrega final do código-fonte	
Descrição da implementação do *software* e testesde unidade		
Descrição da integração e testes de alto nível		
Especificação do meio de entrega do produto final		
Descrição dos processos de gestão de configuração do *software*		
Descrição dos meios de avaliação do produto		
Especificação dos planos de garantia de qualidade		
Especificação dos planos de ação corretiva		
Descrição dos processos de gestão de risco		
Continuação da adaptação da interface aos *mockups*		

Figura 1 - Calendarização por sprints

6.Organizações e Recursos do Projeto

6.1. Organizações

Há duas organizações principais envolvidas neste projeto, ambas fazendo parte da UC: o cliente – a unidade de infraestrutura da GSIIC –, e as diferentes equipas de estudantes da cadeira de Engenharia de *Software.* Há um contacto regular entre as diferentes equipas e o cliente através de uma reunião com 3 representantes dos estudantes, os gestores de cliente, utilizada para tirar dúvidas com o cliente e esclarecer requisitos.

A nossa equipa tem uma estrutura interna simples, tendo sido atribuídos os seguintes cargos: a Mafalda Fernando é a porta-voz, o Rui Ruivo o planeador de reuniões e a Margarida Pereira responsável pelas atas das mesmas.

6.2.Recursos do Projeto

A equipa para este projeto é constituída por 6 elementos: 5 estudantes de LEI e uma estudante de LDM. Como tal, existe uma divisão inicial de tarefas entre as que são relacionadas com a parte de design do projeto e as mais ligadas à área da informática. Entre todos os membros da equipa, reúnem-se diferentes competências necessárias para completar o projeto, tais como a eficiência e a capacidade de trabalhar com Python, a capacidade para estruturar, planear e testar código, a criatividade e *resourcefulness* necessárias para superar desafios inesperados, e também a capacidade para o planeamento de atividades.

Também temos como recursos necessários ao projeto os computadores onde vamos desenvolver o trabalho e software como o RT, o *Pycharm*, a *VirtualBox* e o código dado pelo cliente. Por fim, a informação que é dada pelo cliente nas reuniões com os seus gestores.

7.Ferramentas

Para o desenvolvimento do *software* e artefactos relacionados, são utilizadas as ferramentas abaixo resumidas, já mencionadas em secções anteriores:

Software (ver Referências Externas)

- SOs *Linux Mint* e *Linux Ubuntu*
- *Oracle VirtualBox*
- IDE *Pycharm*
- Editor de texto *Sublime Text*
- *Microsoft Office* 2013 (*Word* e *Excel*)
- Plataformas *web* (*Trello, Dropbox, Facebook)*

Hardware

- Computadores pessoais com o software acima descrito instalado

Gestão da Equipa

- Comunicação e organização presenciais (reuniões semanais e aulas PL)
- Comunicação e organização via remota (plataformas *web* já referidas, outros meios)
- Modelo de processo de *software* SCRUM

As decisões tomadas a nível de *software* e *hardware* foram motivadas tanto pelas condições iniciais do problema proposto, como pelas preferências consensuais dos membros da equipa, segundo a sua experiência de trabalho de desenvolvimento.

As decisões tomadas a nível da gestão da equipa surgiram também do consenso de todos os membros, com base no fundamento teórico estudado nas aulas de Engenharia de *Software.*

Anexos

A.Perfis da Equipa

Adriana

Adriana Pereira Barbosa, estudante de Design e Multimédia.

Pontos Fortes	Pontos Fracos	Nº de Estudante
• Criatividade • Html + CSS • Sentido de humor	• Programação • Gestão de recursos • Foco único	2012138601

João

João Guilherme Assafrão Craveiro, estudante de Engenharia Informática.

Pontos Fortes	Pontos Fracos	Nº de Estudante
• Bom a trabalhar sob pressão • Programação com Python	• Design • Concentração • Gestão de recursos	2013136429

Jorge

Jorge Luís Neto Costa, estudante de Engenharia Informática.

Pontos Fortes	Pontos Fracos	Nº de Estudante
• Criatividade • Bom a trabalhar sob pressão • Sentido de humor	• Concentração • Simplificação • Fraca memória	2012149260

Mafalda

Maria Mafalda Fernando, estudante de Engenharia Informática.

Pontos Fortes	Pontos Fracos	Nº de Estudante
• Criatividade • Boa a trabalhar sob pressão • *Resourcefulness*	• Calma • Teimosia • Perfecionismo	2013136659

Margarida

Ana Margarida de Matos Pereira, estudante de Engenharia Informática.

Pontos Fortes	Pontos Fracos	Nº de Estudante
• Criatividade • Boa a trabalhar sob pressão • *Resourcefulness*	• Fraca memória • Difícil compreensão • Difícil interpretação	2012138634

Rui

Rui Pedro Dias Ruivo, estudante de Engenharia Informática.

Pontos Fortes	Pontos Fracos	Nº de Estudante
• Criatividade • Bom a trabalhar sob pressão • *Resourcefulness* • Sentido de humor • Programação com Python • Eficiência	• Gestão de recursos • *Design* • Fraca memória	2012158444

B. SCRUM *Charts*

Velocity Chart

Representa, em minutos, o tempo médio despendido pela equipa em cada meta (*sprint* do modelo SCRUM). Este *chart* permite à equipa ter uma noção geral da sua velocidade, ou seja, do que consegue executar em determinado período de tempo.

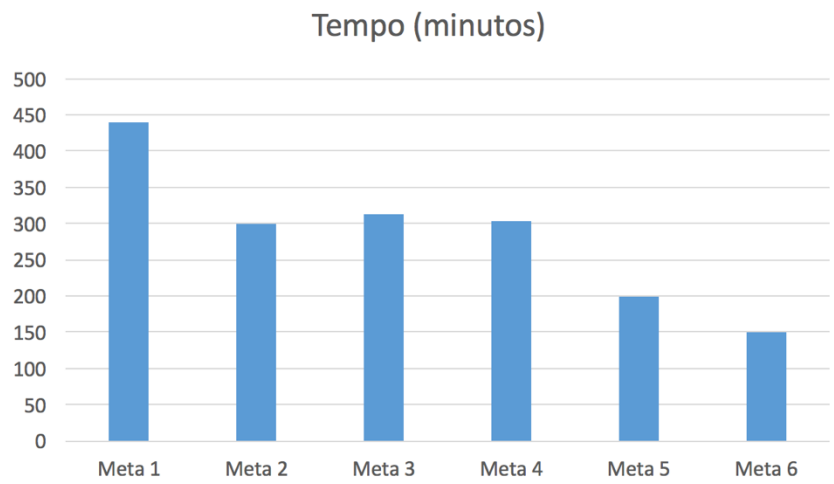

Figura 2- Gráfico do tempo médio dispendido por meta (em minutos)

O tempo médio despendido pelos elementos da equipa para cada meta repartiu-se tanto pelas aulas teóricas e práticas de Engenharia de Software como pelo trabalho realizado fora destas, tanto em grupo como individualmente.

Verifica-se que:

- A primeira meta foi aquela em que o tempo médio investido pelos elementos da equipa foi maior, dado que envolveu a maior parte das atividades de organização e planeamento iniciais.
- As metas intermédias levaram aproximadamente o mesmo tempo, dado que nenhuma representou um esforço adicional ou menor (tratando-se essencialmente de implementações e documentação)
- As duas últimas metas foram aquelas que requereram menos esforço, uma vez que se incluem já na entrega final.

Embora estes dados não reflitam especificamente o esforço empregue por cada elemento do grupo, contemplando o grupo como um todo, é possível concluir que o trabalho foi realizado de forma equilibrada, não havendo grandes variações no tempo despendido por elemento por meta. Tendo em conta o resultado final, pode também concluir-se que este tempo foi adequado às tarefas em execução, dado que a sua maioria foi concluída e os requisitos cumpridos, segundo os critérios estabelecidos.

Burn-down charts

Relação entre a estimativa da progressão ideal do cumprimento das tarefas definidas e a progressão atual, para cada meta (*sprint*) cumprida.

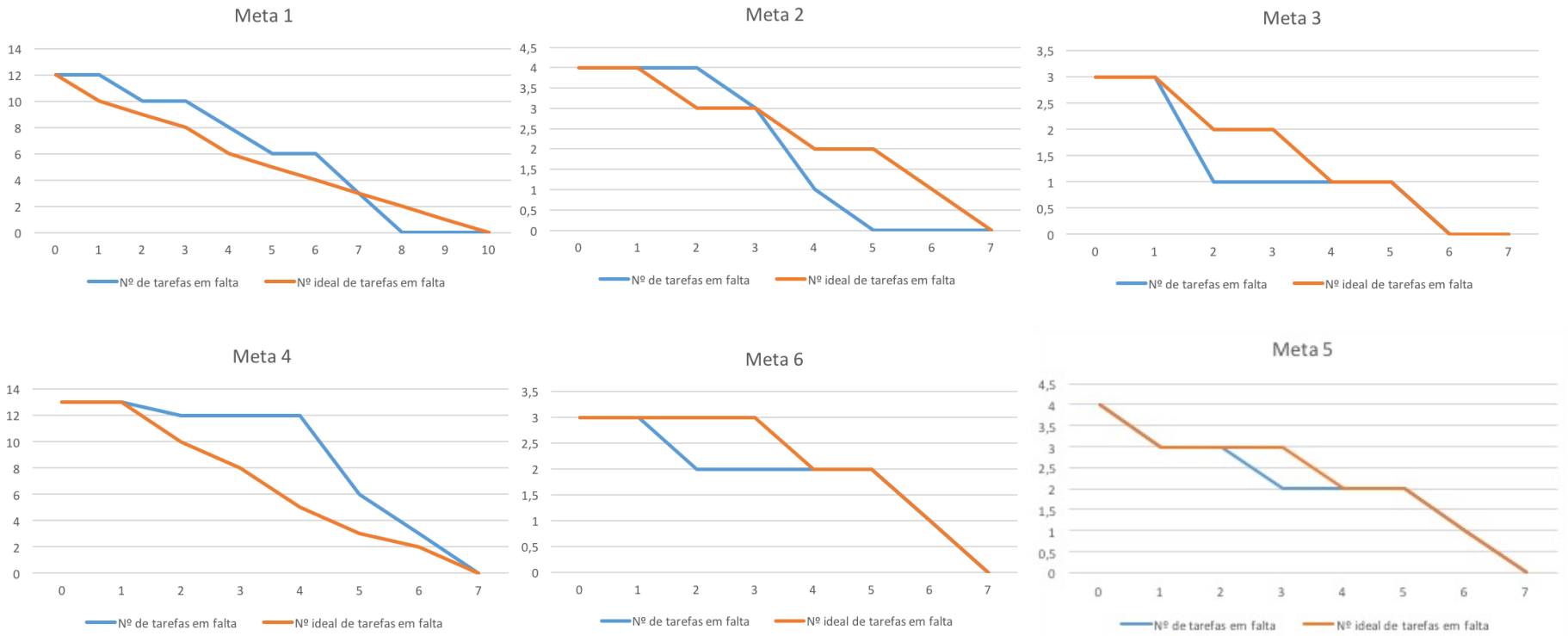

Figura 3 - Burn-down charts das metas 1 a 6

Os gráficos apresentados baseiam-se numa estimativa semi-linear da progressão do cumprimento das tarefas definidas para cada meta (ou seja, assentam na ideia da distribuição uniforme do esforço). Como se pode verificar – e como seria de esperar – , a progressão real não segue o modelo "ideal", dada a natureza diferenciada das tarefas e o facto do esforço da equipa não ser uniforme ao longo das semanas. Para tal contribuiu, entre outros fatores, o facto de Engenharia de *Software* não ser a única cadeira em que os elementos do grupo tiveram de se concentrar no decorrer do semestre.

C. Diagrama de Classes

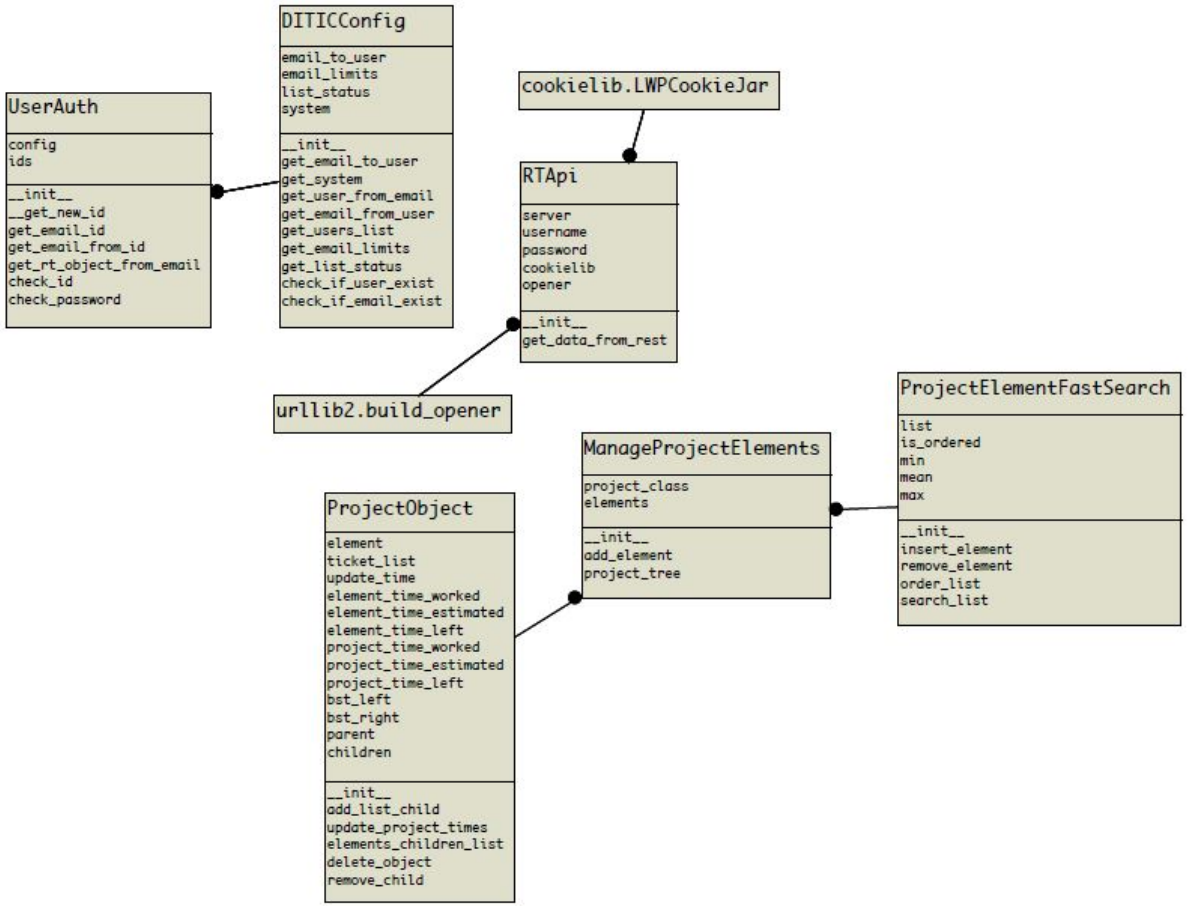

Figura 4 - Diagrama de relação entre classes

D. Diagramas de Fluxo

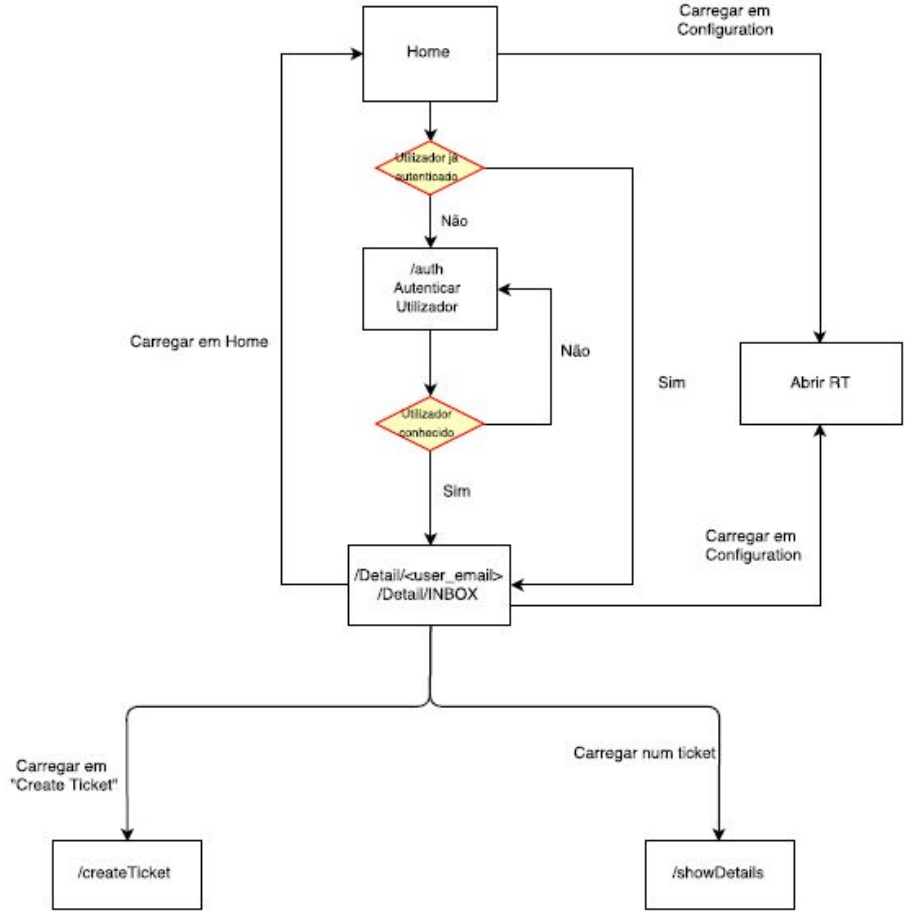

Figura 5 - Workflow do utilizador

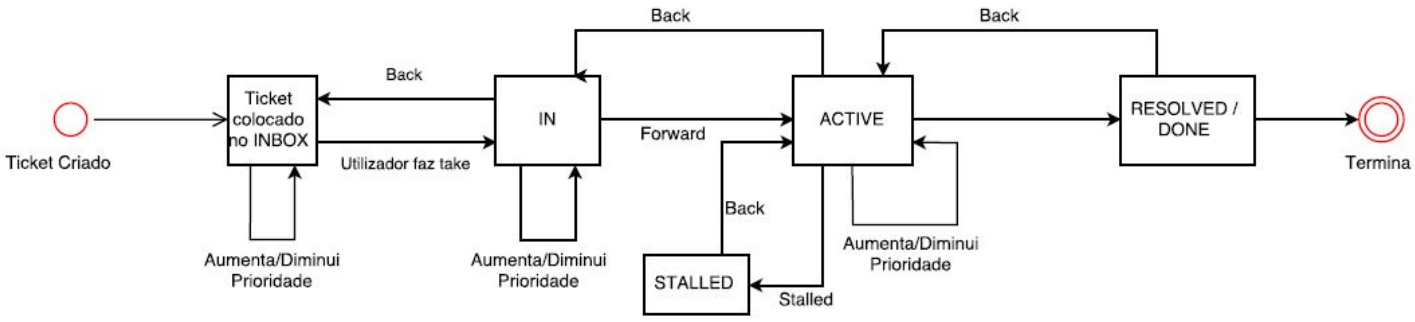

Figura 6 - Fluxo de um ticket

E. *Mockups*

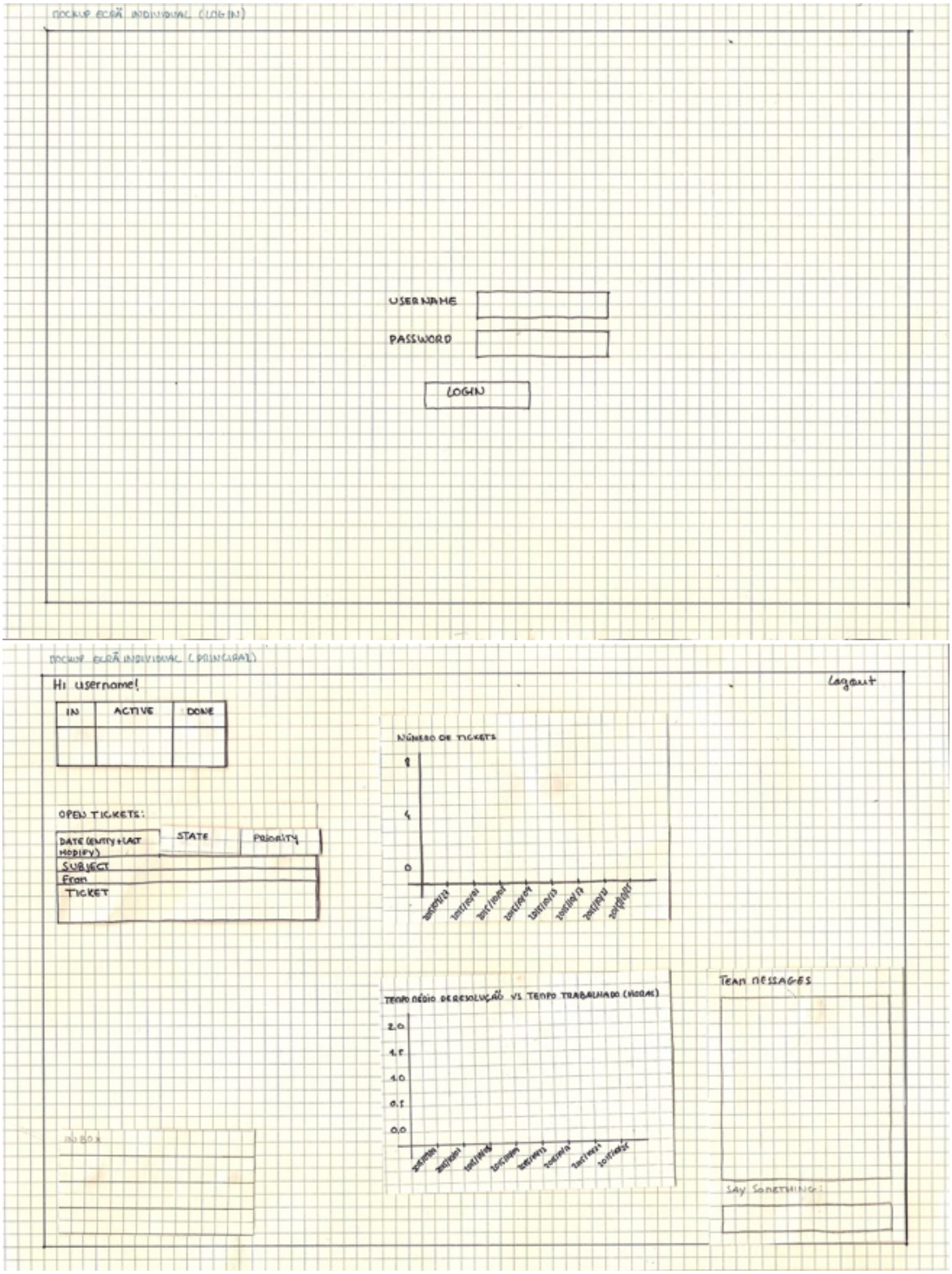

Figura 7- Esboço de Mockups

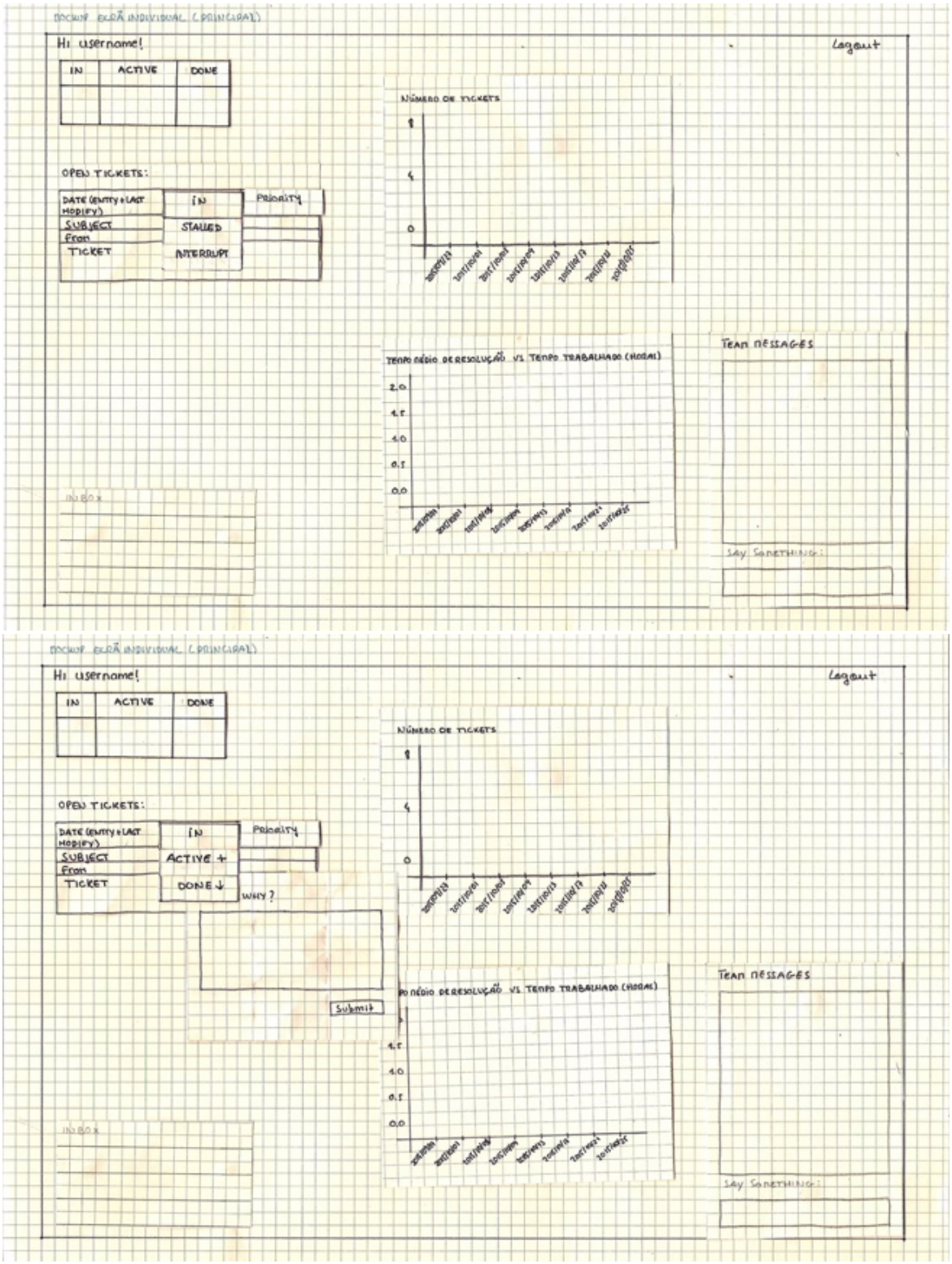

Figura 8- Esboço de Mockups

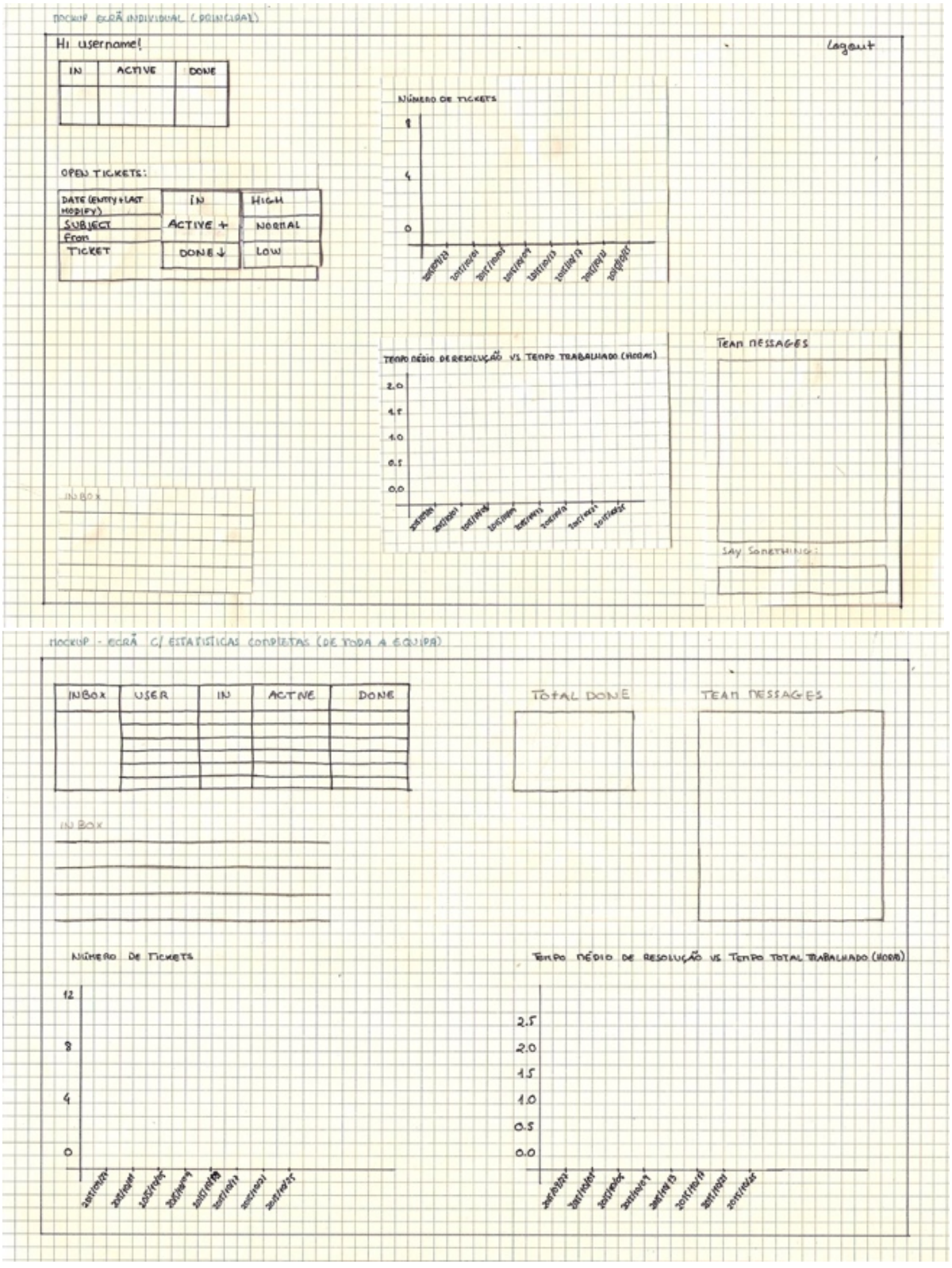

Figura 9 - Esboço de Mockups