

**Faculdade de Engenharia da Universidade do Porto**



**ESOF 2016/2017 - Mestrado Integrado em Engenharia Informática e Computação**

**Autores:**

Andreia Rodrigues - [up201404691@fe.up.pt](mailto:up201404691@fe.up.pt)

Eduardo Leite - [gei12068@fe.up.pt](mailto:gei12068@fe.up.pt)

Gonçalo Leão - [up201406036@fe.up.pt](mailto:up201406036@fe.up.pt)

Francisco Queirós - [up201404326@fe.up.pt](mailto:up201404326@fe.up.pt)

ESOF - Relatório 2

WorldEdit

Processo de engenharia dos requisitos

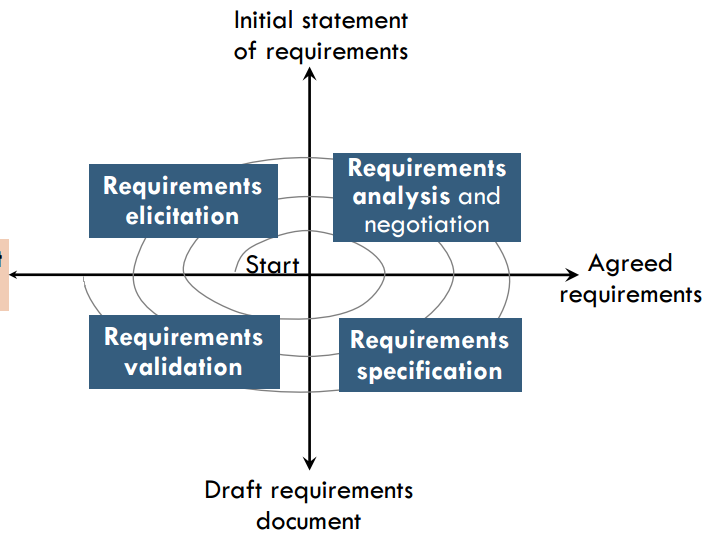
O processo de engenharia de requisitos consiste no estudo das necessidades do cliente e dos utilizadores da aplicação para chegar à definição dos requisitos do sistema, do *hardware* e do *software*.

No âmbito deste relatório, vamos-nos focar nos requisitos de *software*. Um requisito de *software* consiste numa propriedade que o *software* deve incorporar para resolver um problema específico.

Assim, a motivação dos processos de engenharia de requisitos é de converter problemas ou necessidades vagamente definidas em propriedades bem definidas do sistema.

O ciclo do processo de engenharia de requisitos é a primeira fase do processo de engenharia de software. Trata-se de um passo crítico para a qualidade do produto final pois muitos dos defeitos da aplicação têm como origem erros deste ciclo, e o custo de correção de erros cresce exponencialmente ao longo das fases do processo de software. Uma parte significativa destes erros consiste em falhas de comunicação entre os *stakeholders* e na má compreensão dos requisitos a implementar.

O ciclo pode ser dividido em quatro fases.



*Figura 1: Representação visual do ciclo do processo de engenharia de requisitos*

O levantamento de requisitos consiste na interação entre a equipa de desenvolvimento e os *stakeholders* para identificar os problemas que a aplicação deve resolver e as necessidades que deve satisfazer.

A análise e negociação dos requisitos envolve detetar e resolver conflitos entre os requisitos e resolver outros problemas que lhes possam estar associados, tal como a sua ambiguidade. Trata-se de uma fase de refinamento dos requisitos levantados na primeira fase. Em particular, esta fase permite elaborar os requisitos do sistema que serão usados para determinar os requisitos de *software*.

A especificação dos requisitos passa pela elaboração de um documento que especifica os requisitos do sistema de uma forma mais formal. Normalmente, passa pela elaboração de um *software requirements specification* (SRS), uma declaração oficial do que os desenvolvedores da aplicação devem implementar. Nesta fase, podem também ser produzidos outros artefactos (por exemplo, glossários ou certos modelos como modelos de caso de uso ou de domínio) para ajudar a formalizar os requisitos.

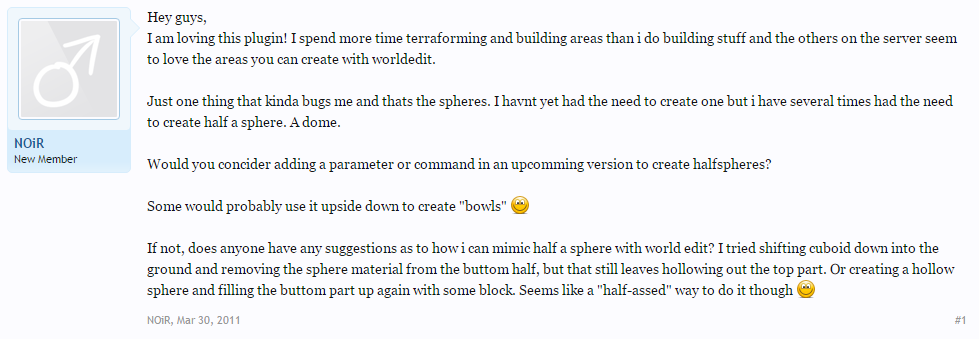
A validação dos requisitos envolve averiguar se os requisitos definem realmente o sistema concebido pelo cliente. Exemplos de técnicas usadas incluem a revisão e inspeção dos requisitos e a geração de testes de aceitação.

Vamos agora ver como é que as quatro fases do ciclo do processo de engenharia de requisitos estão a ser implementadas no caso do nosso projeto de análise, WorldEdit. Convém ter em mente que o esquema apresentado trata-se apenas de um entre vários modelos válidos e viáveis para a engenharia de requisitos. Além disso, num contexto prático, tal como se viu no primeiro relatório para o processo de *software*, o processo empregue pode ter muitas semelhanças aos modelos teóricos, mas a correspondência entre os dois nunca é total.

Levantamento

No caso do WorldEdit, o levantamento de requisitos tem lugar no *issue tracker* do WorldEdit, na maior parte dos casos. Pode também ser usado o fórum do WorldEdit e o canal IRC do desenvolvedor mais ativo do projeto, sk89q, apesar destes meios de comunicação servirem sobretudo para identificar *bugs* e fazer perguntas sobre o *plugin*. Nestes meios, novas ideias de *features* são apresentadas tanto por utilizadores do *plugin*, como pelos seus desenvolvedores, que constituem os s*takeholders* deste projeto (não existem “clientes” pois trata-se de um projeto open-source).

Dado que este projeto tem um processo de software próximo do modelo incremental de desenvolvimento (como se viu no primeiro relatório), os requisitos não são todos definidos de uma vez. Com efeito, os requisitos vão surgindo como fruto das vontades e necessidades dos utilizadores.

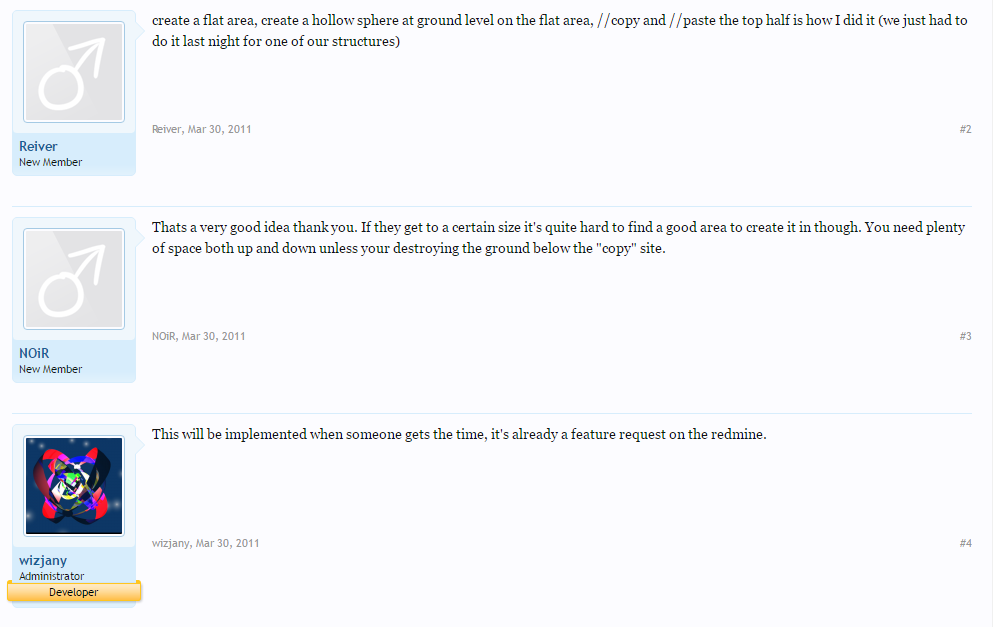
Na figura abaixo, pode-se ver um exemplo da proposta de uma nova *feature* (requisito funcional) através do fórum do WorldEdit. A nova *feature* consiste em criar semi-esferas de certos tipos de blocos, tendo em conta que já existe uma funcionalidade semelhante para desenhar esferas. 

*Figura 2: Exemplo da proposta de uma “feature” através do fórum do WorldEdit  
(Fonte:* <http://forum.enginehub.org/threads/half-a-sphere-a-dome.168/#post-653>*)*

Análise e negociação

A análise e negociação de requisitos ocorre nos mesmos canais de comunicação que o levantamento de requisitos. Esta fase passa pela discussão entre quem propôs a *feature* e tanto os desenvolvedores como outros utilizadores do plugin.

A título de exemplo, vejamos o início da discussão no fórum do WorldEdit para a proposta da *feature* apresentada anteriormente, para desenhar semi-esferas.



*Figura 3: Análise e negociação da “feature”proposta pelo fórum do WorldEdit*

No exemplo acima, o utilizador “Reiver” propõe uma solução alternativa para o problema apresentado inicialmente. Em seguida, o utilizador “NOiR” expressa a sua opinião, dizendo que apesar de ser uma solução alternativa viável em alguns casos, noutros não o será (quando não houver espaço suficiente para desenhar uma esfera). Por fim, o utilizador “wizjany”, um dos principais developers do projeto, informa que já existe uma proposta igual no redmine, ilustrando a importância que esta fase tem para eliminar colisões entre propostas.

Especificação

No caso do WorldEdit, não existe a fase de especificação de requisitos. Tratando-se de um projeto *open-source* de pequena/média dimensão em que não há uma organização muito rígida, não é elaborado um documento que formalize os novos requisitos do *software*.

O que existe de mais próximo a este gênero de documentos é a wiki do WorldEdit, onde são apresentadas as *features* deste *plugin*. Assim, a wiki funciona como um manual de utilização do *plugin*. Contudo, a wiki só é atualizada depois das *features* serem implementadas, o que é uma diferença fundamental com os modelos estudados de engenharia de *software* e de requisitos, onde os documentos a formalizar os requisitos são elaborados antes da sua implementação.

Validação

A fase de validação também não é facilmente identificável no caso do WorldEdit, visto que não existe nenhum método para formalizar os requisitos.

Podemos apenas dizer que, no contexto do WorldEdit, a fase de validação de requisitos ocorre de forma natural e informal ao longo das outras fases do processo de engenharia de software.

Identificação dos requisitos

Os requisitos de *software* podem ser separados em duas categorias: requisitos funcionais e não-funcionais.

Vamos agora ver como se podem definir estes dois tipos de requisitos e ver alguns exemplos para o caso do projeto WorldEdit.

Requisitos funcionais

Um requisito funcional é uma declaração de serviços que a aplicação deve fornecer e de como deve reagir face a um dado *input* ou situação.

No caso do WorldEdit, os *inputs* dados pelo utilizador do plugin são feitos através de comandos, com ou sem argumentos dependendo na utilização destes. Estes comandos são introduzidos na janela de chat do jogo Minecraft e permitem alterar o mundo 3D a partir da manipulação dos blocos já existentes ou da geração de novos blocos.

Alguns dos comandos existentes e respectiva resposta do WorldEdit após serem inseridos na consola são:

* //limit - o plugin deve limitar o número máximo de blocos que o utilizador seleciona ao mesmo tempo

- Comandos para manipulação do histórico de ações (todas as operações são guardadas num histórico de ações, que guarda as 15 operações mais requisitos. Este histórico muda sempre que é alterado um bloco através do WorldEdit):

* //undo - programa deve desfazer a última ação feita
* //redo - programa deve refazer a última ação desfeita
* /clearhistory - programa deve apagar o histórico de ações

- Comandos para a seleção de regiões (o WorldEdit é baseado na edição de regiões de blocos que são previamente selecionadas utilizando estes comandos):

* //wand - plugin dá a ferramenta de seleção de regiões ao utilizador para que os comandos relativos a essa operação possam ser utilizados. Para o plugin cancelar essa ação, deverá ser inserido o comando /toggleeditwand pelo utilizador.
* //sel - programa deve escolher a forma da região a editar
* //desel - deve cancelar seleção efetuada
* //chunk - programa deve selecionar “chunk” (grupo de blocos com dimensões predefinidas pelo Minecraft) no local onde o jogador se encontra
* //expand e //contract - programa deve expandir ou contrair a região selecionada conforme os argumentos que lhe sejam passados no comando
* //shift - plugin deve mover a região selecionada para outro local escolhido pelo utilizador

- Comandos para manipulação de regiões (o WorldEdit tem comandos simples e intuitivos para os utilizadores que ajudam na manipulação de regiões de blocos previamente selecionadas por comandos deste mesmo plugin):

* //set - WorldEdit deve transformar todos os blocos da região selecionada no tipo especificado pelo comando
* //replace - plugin deve substituir todos os blocos da região de um determinado tipo por blocos de outro tipo (ambos especificados pelo comando)
* //walls - plugin deve construir paredes à volta da região selecionada
* //outline - plugin deve construir paredes, teto e chão à volta da região selecionada
* //smooth - plugin deve nivelar a superfície da região selecionada
* //deform - plugin deforma a superfície da região selecionada segundo uma função especificada na chamada do comando
* //forest - plugin deve preencher a região selecionada com uma floresta

- Comandos para clipboard (funcionalidade implementada no plugin que permite copiar/colar uma região de blocos, guardá-la ou até carregá-la a partir de um ficheiro):

* //copy - plugin deve copiar a região selecionada
* //cut - plugin deve cortar a região selecionada
* //paste - plugin deve colar a região selecionada
* //rotate e //flip - plugin deve rodar/virar a região em clipboard
* //schem - plugin deve fazer a operação especificada no comando em relação ao clipboard existente no momento (guardar em ficheiro, fazer load de um clipboard guardado, etc)

- Comandos para geração de regiões (o WorldEdit permite gerar regiões com formas mais complexas que os quadrados tradicionais do Minecraft a partir de comandos simples, usando como bloco original o bloco abaixo da posição do jogador):

* //generate - WorldEdit deve gerar uma região de acordo com a fórmula indicada pelo utilizador na chamada do comando
* //cyl - plugin deve gerar um cilindro
* //sphere - plugin deve gerar uma esfera
* //pyramid - plugin deve gerar uma pirâmide
* /forestgen - plugin deve gerar uma floresta

- Comandos utilitários (o plugin disponibiliza comandos que facilitam a manipulação do mundo 3D):

* //fill - WorldEdit deve colocar blocos na zona vazia dentro da região selecionada
* //drain - plugin deve retirar líquidos (água ou lava) de dentro da região selecionada
* /fixwater - plugin deve nivelar zonas com água dentro da região selecionada
* /fixlava - plugin deve nivelar zonas com lava dentro da região selecionada
* /removeabove - plugin deve retirar blocos que estejam acima da posição do jogador
* /removebellow - plugin deve retirar blocos que estejam abaixo da posição do jogador
* /butcher - WorldEdit deve eliminar todos os inimigos (mobs) dentro da zona selecionada
* /now - WorldEdit deve simular queda de neve dentro da zona selecionada
* /ex - WorldEdit deve eliminar fogos dentro da zona selecionada

- Comandos para manipulação de “chunks” (“chunks” são segmentos 16x16x256 do mundo de Minecraft. Estes comandos ajudam à sua listagem e edição):

* /chunkinfo - WorldEdit deve devolver o nome do ficheiro onde está o chunk onde o jogador está
* /listchunks - plugin deve listar todos os “chunks” a serem usados
* /delchunks - plugin deve apagar os “chunks” dentro da região selecionada

- Comandos para seleção de ferramentas (o WorldEdit fornece ferramentas que podem ser associadas a um item para poderem ser utilizadas e que ajudam na edição da região selecionada pelo utilizador):

* // - plugin deve mudar a arma do jogador para “super pick axe” que permite minar e destruir blocos mais facilmente
* /sp single - plugin deve mudar a “super pick axe” para atuar em apenas um bloco de cada vez
* /sp area - plugin deve mudar a “super pick axe” para atuar numa área de blocos
* /tool - WorldEdit deve selecionar a ferramenta descrita no comando introduzido pelo utilizador
* /none - plugin deve retirar a ferramenta a ser utilizada pelo utilizador

- Comandos para seleção de pincéis(o plugin permite construir e editar regiões de blocos a grandes distâncias a partir da utilização de pincéis que podem ser ligados ao item a ser utilizado de momento):

* /brush sphere - plugin deve selecionar o pincel de gerar esferas
* /brush cylinder - plugin deve selecionar o pincel de gerar cilindros
* /brush clipboard - plugin deve mudar a ferramenta atual para a ferramenta de clipboard
* /brush smooth - plugin deve selecionar o pincel que nivela regiões de blocos

- Comandos de *Getting Around* (o plugin fornece comandos que ajudam o utilizador a mexer a sua personagem de forma a conseguir editar as regiões de blocos de forma mais eficiente):

* /unstuck - plugin deve teleportar jogador para o primeiro espaço livre perto da posição onde chama o comando
* /ascend - plugin deve teleportar jogador para um nível de blocos acima
* /descend - plugin deve teleportar jogador para um nível de blocos abaixo
* /thru - plugin deve teleportar jogador para o outro lado da parede para onde está a olhar

- Comandos para snapshots (funcionalidade que permite carregar uma secção do mundo definida por uma selecção de uma região e fazer restore para um backup feito anteriormente sem ter de desligar o servidor):

* //restore - plugin deve restaurar um snapshot especificado pelo utilizador ao chamar o comando
* //snapshot use - plugin deve usar um snapshot especificado pelo utilizador ao chamar o comando
* //snapshot sel - plugin deve selecionar um snapshot com um id especificado pelo utilizador ao chamar o comando

- Comandos para scripting (o WorldEdit permite integrar scripts para permitir fazer pequenas tarefas de forma rápida e sem complicações):

* /cs - WorldEdit deve executar o script especificado ao chamar o comando
* /.s - WorldEdit deve re-executar o último script executado mas com novos argumentos

- Comandos Gerais:

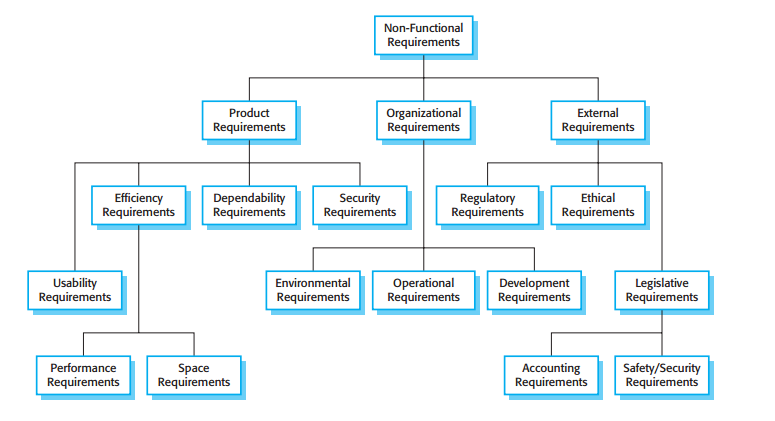
* /searchitem - WorldEdit deve procurar por um item com o nome especificado na chamada do comando
* /worldedit - plugin deve listar todos os comandos disponibilizados pelo plugin
* /worldedit help - plugin deve mostrar documentação sobre o comando cujo nome é especificado na chamada deste comando ou lista documentação sobre todos os comandos caso não sejam declarados argumentos

- Comandos para biomas (um bioma é um região do mundo do Minecraft com características geográficas específicas (clima, humidade, altitude…)):

* /biomelist - plugin deve listar todos os biomas disponibilizados pelo plugin
* /biomeinfo - plugin deve mostrar qual o bioma no bloco selecionado
* //setbiome - plugin deve colocar bioma especificado nos blocos da região selecionada

Requisitos não-funcionais

Um requisito não-funcional é uma restrição nos serviços e funcionalidades da aplicação. Tratam-se de requisitos que tendem a abranger todo o sistema e não apenas alguns dos seus componentes específicos. As restrições podem estar associadas a questões de eficiência, segurança, fiabilidade, portabilidade ou facilidade de uso do sistema, entre outras.



*Figura 2: Classificação dos requisitos não-funcionais  
(Fonte: Software Engineering, Ian Sommerville, 9th Edition)*

Desenvolvimento

No WorldEdit, as funcionalidades a adicionar no projeto têm de satisfazer alguns requisitos gerais. Não devem interferir com a API, nem devem alterar a forma como os utilizadores usam o plugin, a menos que haja uma boa razão para isso. Devem evitar implementar funcionalidades que vão ser difíceis de manter mais tarde. Também devem procurar ser independentes das outras funcionalidades implementadas.

Segurança

Quanto aos requisitos de segurança, estes têm de ser cumpridos de forma imperativa, pois algumas funções do software poderiam sofrer abuso por parte de alguns utilizadores se não lhes fossem impostas restrições por parte dos desenvolvedores do software. Isto deve-se pelo facto de certas funções, sem restrições de utilização, deterem bastante poder em relação a todo o software. A título de exemplo, um utilizador pode fazer paste de uma área muito grande, o que pode conduzir a um crash do server. É por isso importante restringir o acesso de certas ferramentas apenas a pessoas que o devem ter, por exemplo, os desenvolvedores do projeto.

Portabilidade

O código deste projeto é escrito maioritariamente em Java e evita o uso de código específico de certas plataformas sempre que possível, usando ainda mecanismos de recurso caso algo não funcione. Desta forma, o plugin funciona em todas as grandes plataformas.

Compatibilidade

Em termos de compatibilidade, o WorldEdit é um dos poucos plugins na comunidade de Minecraft que funciona em várias plataformas de plugins de forma consistente. Alguns exemplos são hMod, SCP Bukkit, Spigot, Forge, Canary, Spout, entre outros.

Dependências

O WorldEdit depende de algumas bibliotecas de Java como o Google Guava (que contém, em particular, novas estruturas de dados como multimap e multiset), entre outras. No entanto, não tem quaisquer dependências com outros plugins ou mods de Minecraft.

Usabilidade

Não são necessárias habilidades para trabalhar com este plugin. Apenas é necessário ler a documentação.

Espaço

Não existem quaisquer requisitos de espaço.

Casos de uso

Um diagrama de casos de uso tem como objetivo esquematizar, de forma simples e clara, os diferentes cenários de utilização da aplicação por parte dos utilizadores finais. O diagrama identifica os atores (utilizadores) e as suas interações com o sistema.

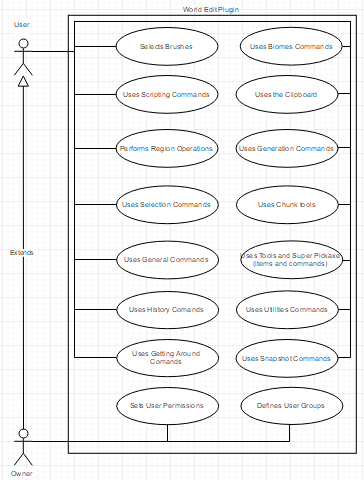


Figura - *Use Case Diagram* do *World Edit*

Após uma análise do contexto do World Edit, chegamos à conclusão de que podemos distinguir 2 actores para o nosso diagrama:

*User* - O utilizador, que através da linha de comandos fornecida pelo Minecraft, pode executar comandos para o plugin. Sendo assim, ligamos este actor a use cases que representam grupos de comandos. Não descemos mais um nível até aos comandos em si, para não complicar demasiado o gráfico, pois tal não seria interessante no contexto deste relatório.

*Owner* - O dono do servidor, que é um tipo específico de utilizador, consegue fazer tudo o que o utilizador pode fazer e pode também gerir as permissões e grupos a que os jogadores pertencem, apesar de estas ações funcionarem em conjunto com plugins externos que auxiliam à gestão de permissões, as permissões pertencem e correspondem aos comandos do World Edit e portanto achamos pertinente incluir estas acções como casos de uso.

Vamos agora definir algumas templates do diagrama de casos de uso, para os casos mais importantes:

1. Nome - “Uses Selection Commands”

Actors - User

Goal description - utilizador usa um comando que seleciona uma região de blocos, ou modifica uma seleção pré-existente

Referência a requisitos - comandos para a seleção de regiões

Pré-condições - utilizador deve ter feito login no seu servidor de Minecraft e deve ter a permissão para este comando

Descrição:

1. Utilizador seleciona o chat de Minecraft clicando na tecla “T”

2. Utilizador escreve o comando desejado, com os argumentos necessários no chat e clica na tecla “Enter”

3. Utilizador seleciona a região do mundo que quer editar e procede à edição desse conjunto de blocos

Pós-condições - A selecção que o utilizador tinha feito, foi alterada de acordo com o comando selecionado dentro do grupo de comandos que este use case representa

Variações:

1. Se utilizador cancelar a escrita do comando

* este não vai ser registado pelo chat e plugin não vai executar nada até que um comando seja recebido com sucesso

Excepções:

1. Se utilizador escrever mal o comando

* chat vai mostrar uma mensagem de erro, que diz ao utilizador que esse comando não existe

2. Se utilizador colocar argumentos inválidos, ou nenhum argumento

* chat vai mostrar uma mensagem de erro que informa o utilizador da utilização correta desse comando ou remete-o para o comando “help”

3. Se utilizador colocar tudo corretamente mas não tiver permissão para executar o comando escolhido nas condições atuais

* chat vai mostrar uma mensagem de erro que informa o utilizador que não tem permissão para executar o comando escolhido

1. Nome - “Performs Region Operations”

Actors - User

Goal description - utilizador usa um comando que altera uma região de blocos previamente selecionada

Referência a requisitos - comandos para a manipulação de regiões

Pré-condições - utilizador deve ter feito login no seu servidor de Minecraft e deve ter a permissão para este comando. Para além disto, o utilizador deve ainda ter feito os passos necessários à seleção de uma região de blocos de forma bem sucedida.

Descrição:

1. Utilizador seleciona o chat de Minecraft clicando na tecla “T”

2. Utilizador escreve o comando desejado, com os argumentos necessários no chat e clica na tecla “Enter”

3. Utilizador manipula a região do mundo que selecionou previamente, utilizando os comandos de seleção de regiões

Pós-condições - A região editada pelo utilizador apresenta agora as características que este definiu através do comando selecionado dentro do grupo de comandos que este use case representa.

Variações:

1. Se utilizador cancelar a escrita do comando

* este não vai ser registado pelo chat e plugin não vai executar nada até que um comando seja recebido com sucesso

Excepções:

1. Se utilizador escrever mal o comando

* chat vai mostrar uma mensagem de erro, que diz ao utilizador que esse comando não existe

2. Se utilizador colocar argumentos inválidos, ou nenhum argumento

* chat vai mostrar uma mensagem de erro que informa o utilizador da utilização correta desse comando ou remete-o para o comando “help”

3. Se utilizador colocar tudo corretamente mas não tiver permissão para executar o comando escolhido nas condições atuais

* chat vai mostrar uma mensagem de erro que informa o utilizador que não tem permissão para executar o comando escolhido

1. Nome - “Uses Generation Commands”

Actors - User

Goal description - utilizador usa um comando que gera automaticamente uma região de blocos de acordo com o comando selecionado e os argumentos inseridos

Referência a requisitos - comandos para a geração de regiões

Pré-condições - utilizador deve ter feito login no seu servidor de Minecraft e deve ter a permissão para este comando.

Descrição:

1. Utilizador seleciona o chat de Minecraft clicando na tecla “T”

2. Utilizador escreve o comando desejado, com os argumentos necessários no chat e clica na tecla “Enter”

Pós-condições - O comando gerou uma região de blocos com as dimensões passadas nos argumentos e com a forma respetiva descrita pelo comando introduzido. Esta região de blocos vai ser gerada no bloco abaixo do jogador.

Variações:

1. Se utilizador cancelar a escrita do comando

* este não vai ser registado pelo chat e plugin não vai executar nada até que um comando seja recebido com sucesso

Excepções:

1. Se utilizador escrever mal o comando

* chat vai mostrar uma mensagem de erro, que diz ao utilizador que esse comando não existe

2. Se utilizador colocar argumentos inválidos, ou nenhum argumento

* chat vai mostrar uma mensagem de erro que informa o utilizador da utilização correta desse comando ou remete-o para o comando “help”

3. Se utilizador colocar tudo corretamente mas não tiver permissão para executar o comando escolhido nas condições atuais

* chat vai mostrar uma mensagem de erro que informa o utilizador que não tem permissão para executar o comando escolhido

1. Nome - “Uses History Commands”

Actors - User

Goal description - utilizador usa um comando que manipula o histórico de operações feitas previamente. Estas operações são originadas por outros comandos inseridos pelo utilizador que manipulam regiões de blocos

Referência a requisitos - comandos para manipulação do histórico de ações

Pré-condições - utilizador deve ter feito login no seu servidor de Minecraft e deve ter a permissão para este comando

Descrição:

1. Utilizador seleciona o chat de Minecraft clicando na tecla “T”

2. Utilizador escreve o comando desejado e clica na tecla “Enter”

Pós-condições - O comando refez a operação apagada anteriormente ou apagou a última operação feita, conforme o comando inserido e de acordo com o estado do mundo quando o comando é inserido

Variações:

1. Se utilizador cancelar a escrita do comando

* este não vai ser registado pelo chat e plugin não vai executar nada até que um comando seja recebido com sucesso

Excepções:

1. Se utilizador escrever mal o comando

* chat vai mostrar uma mensagem de erro, que diz ao utilizador que esse comando não existe

2. Se utilizador colocar tudo corretamente mas não tiver permissão para executar o comando escolhido nas condições atuais

* chat vai mostrar uma mensagem de erro que informa o utilizador que não tem permissão para executar o comando escolhido

Modelo do domínio

Um modelo do domínio tem como objetivo representar classes conceituais da aplicação num contexto mais real, de alto nível. Este modelo não pretende especificar as classes envolvidas no *design* do *software* mas sim os procedimentos, eventos, objetos e relações no domínio dos problemas que a aplicação pretende resolver.

**Conclusões**

É do entender dos autores deste relatório que a fase de engenharia de requisitos tem uma importância fulcral no desenvolvimento de *software.* Com efeito, seguir um bom processo de engenharia de requisitos permite detetar vários erros que, se não fossem tratados numa fase mais inicial, seriam muito mais custosos (nomeadamente em termos de esforço e tempo, neste caso) de resolver mais à frente. Estes erros consistem sobretudo em falhas de comunicação entre os *stakeholders* e numa má compreensão dos requisitos a incorporar no sistema. Assim, o processo de engenharia de requisitos mostra bem a importância do diálogo entre as diferentes partes envolvidas num projeto de engenharia de software.

Apesar do processo de engenharia de requisitos do WorldEdit não seguir um modelo próximo do que é apresentado na figura 1, foi adotada uma boa metodologia para a gestão dos requisitos, visto que, em particular, lida bem com a evolução contínua dos requisitos do sistema.

Bibliografia

Software Engineering, Ian Sommerville, 9th Edition, capítulos 4, 5 e glossário.

Glossário do IEEE para a terminologia de engenharia de software, disponível em: <http://www.mit.jyu.fi/ope/kurssit/TIES462/Materiaalit/IEEE_SoftwareEngGlossary.pdf>