**RELAZIONE TERZO SPRINT**

**SPRINT REVIEW TERZA SETTIMANA**

All’inizio del terzo sprint era stata stimata una velocità di **58 sp** . Nella riunione tenuta all’inizio della settimana di lavoro, erano state definite **6 user stories**, da completare entro il 12/12/21. Tutti i task sono stati completati in modo corretto, e in più sono stati aggiunti due ulteriori task per migliorare l’interfaccia grafica. Attualmente la velocità dal team è di **64 sp,** in quanto col miglioramento della grafica dell’interfaccia sono stati aggiunti altri 6 sp. Inoltre è stato riportato un technical debt per un test non completato sulla classe Parser.

Nella riunione tenuta all’inizio del terzo sprint, il team ha deciso di adottare nuove scelte progettuali, ricorrendo all’utilizzo di un Command Pattern, un pattern utilizzato quando si ha la necessità di disaccoppiare l’invocazione di un comando dai suoi dettagli implementativi, separando colui che invoca il comando da chi esegue l’operazione. Sono quindi state ridefinite delle classi per ogni opportuno Command, un classe Invoker e per svolgere l’user story riguardante le user operation è stata definita una classe **UserDefinedCommand**.

Nella classe **CommandInvoker** sono stati definiti dei metodi che consentono all’utente di aggiungere un nuovo comando, eseguirlo, modificarlo e cancellarlo.

Inoltre è possibile tramite l’interfaccia, salvare le nuove operazioni definite dall’utente, ricaricarle attraverso un *JFileChooser* che dà la possibilità di accedere direttamente al file System dell’utilizzatore, dal quale l’utente può scegliere il file di estensione adatto alla visualizzazione in tabella delle user defined operations. Il salvataggio e il reload dei comandi tramite file è stato sempre gestito mediante due classi specifiche: **SaverFile** e **ImportFile.**

* **SaverFile:** impone il vincolo all’utente dipoter salvare i file solo utilizzando l’estensione *(\*.txt)* .
* **ImportFile:** rende visibile all’utente solo i file (\*.txt).

E’ stato necessario ridefinire una nuova classe **Parser** per i comandi inseriti dall’utente, per l’implementazione della user story riguardante le user defined operations. Nuove eccezioni sono state definite per gestire tutti i possibili casi d’inserimento da parte del cliente, sia nel caso di operazione andata a buon fine che non riuscita per il non riconoscimento di un comando (compreso l’inserimento di numeri).

Tutti i nuovi comandi inseriti dall’utente sono visibili direttamente dalla **GUI** attraverso una JTable; è possibile cancellare o modificare i singoli comandi con una selezione della riga, tramite con un pulsante. L’operazione di modifica di una singola user defined operation, si abilita nel momento il cui l’utente seleziona dalla JTable la relativa riga.

E’ stata inoltre migliorata la grafica dell’interfaccia per la presentazione finale.

**Terzo Sprint – Retrospective**

* **Stop (cose da smettere di fare):**

Non gestire tutti i casi d’eccezione durante la progettazione ed implementazione di un task.

* **Less of (cose da fare di meno):**

Definire le unità di testing in maniera poca dettagliata.

* **Keep doing (cose da continuare a fare):**

Continuare a fare commit di codice testato, e accorparlo al ramo principale.

Continuare a comunicare con il team.

* **More of (cose da fare di più):**

Riuscire a dedicare una piccola parte del tempo di lavoro per supportare gli altri, in caso di necessità.

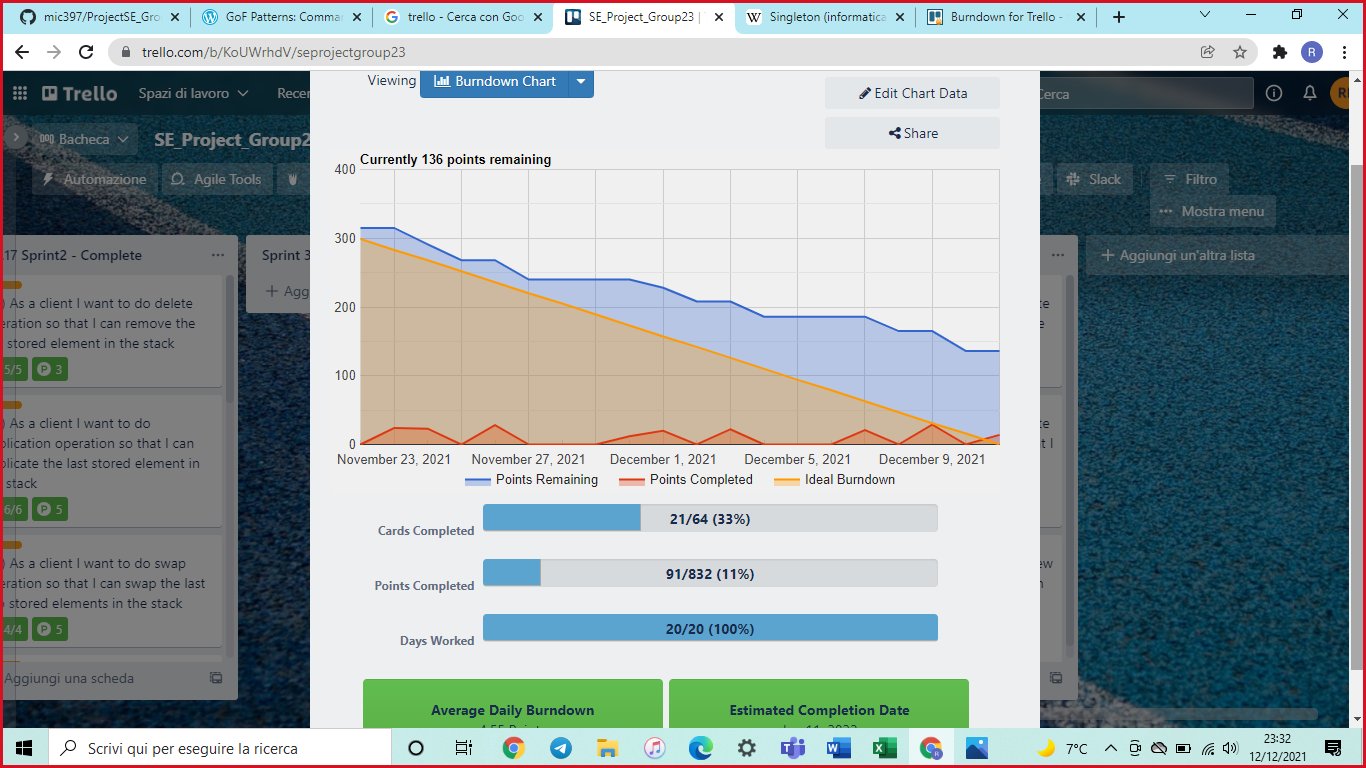
* **Start (cosa da iniziare a fare):**

**BURNDOWN CHART**

Il burndown chart complessivo è disponibile su Trello alla seguente sezione.

Immagine che contiene testo, screenshot, interni

Descrizione generata automaticamente[**https://trello.com/b/KoUWrhdV/seprojectgroup23**](https://trello.com/b/KoUWrhdV/seprojectgroup23)



**SCELTA ARCHITETTURALE**

Come già stabilito durante la fase di pre-gaming, il team ha scelto di basare l’intero progetto su un’architettura **MVC**, per l’implementazione della calcolatrice scientifica programmabile. Questa architettura è stata scelta in quanto l’applicativo mantiene una vista principale che fornisce i principali “comandi” concessi all’utente. È basata su eventi generati dall’utente, tramite il (Viewer), gestiti in back-end dal Controller.

* **Model:** definisce i metodi per accedere e modificare i dati; rappresenta lo stato dell'applicazione. In tal caso il ruolo è ricoperto principalmente dalla classe Model
* **Viewer:** è l’interfaccia GUI, tramite la quale l’utente può eseguire i vari comandi. Non contiene logica, semplicemente sono stati inseriti in essa tutti gli elementi grafici.
* **Controller:** è la classe Calculator, che contiene l’intera logica della calcolatrice, ovvero tutti I metodi necessari per eseguire le varie operazioni.

