Norme di Progetto

Gruppo VarTmp7 - Progetto Stalker vartmp7@gmail.com

Informazioni sul documento

Versione	4.0.0-R1
Approvatore	Ricardo Tassetto
Redattori	Giuseppe Zatta Riccardo Tassetto Xiaowei Wen
Verificatori	Marco Ferrati Lorenzo Taschin
Uso	Interno
Distribuzione	Prof. Tullio Vardanega Prof. Riccardo Cardin VarTmp7

Descrizione
In questo documento sono descritte le regole, le convenzioni e gli strumenti adottati durante lo sviluppo del progetto Stalker dal gruppo VarTmp7.

Registro delle modifiche

Versione	Data	Descrizione	Nominativo	Ruolo
4.0.0-R1	2020-05-14	Versione 4.0.0-R1 approvata	Riccardo Tassetto	*Responsabile*
4.0.0-INC-14	2020-05-06	Verificata nuova struttura del documento. Documento pronto per l'approvazione.	Lorenzo Taschin	*Verificatore*
3.1.0-INC-14	2020-05-06	Modificata struttura documento	Giuseppe Zatta	*Redattore*
3.1.0-INC-11	2020-04-15	Verificata norma sul versionamento per il rilascio	Lorenzo Taschin	*Verificatore*
3.0.0-INC-11	2020-04-15	Aggiunta norma sul versionamento per il rilascio	Riccardo Tassetto	*Redattore*
3.0.0-INC-10	2020-04-12	Versione 3.0.0-INC-10 approvata	Lorenzo Taschin	*Responsabile*
3.0.0-INC-10	2020-04-08	Verifica correzioni e aggiunte in §3.1. Documento pronto per l'approvazione	Marco Ferrati	*Verificatore*
2.2.0-INC-10	2020-04-06	Correzioni e aggiunte in §3.1	Riccardo Tassetto	*Redattore*
2.2.0-INC-10	2020-04-06	Verifica correzioni e aggiunte in §3.4	Marco Ferrati	*Verificatore*
2.1.1-INC-9	2020-04-05	Correzioni e aggiunte in §3.4	Riccardo Tassetto	*Redattore*
2.1.1-INC-8	2020-03-24	Verifica correzioni di §3.1.1, §3.1.2 e §3.1.3	Lorenzo Taschin	*Verificatore*
2.1.0-INC-8	2020-03-24	Correzione di §3.1.1, §3.1.2 e §3.1.3	Riccardo Tassetto	*Redattore*
2.1.0-INC-7	2020-03-21	Verifica §3.3.2.1	Lorenzo Taschin	*Verificatore*
2.0.0-INC-7	2020-03-20	Stesura §3.3.2.1	Xiaowei Wen	*Redattore*
2.0.0-PoC_v_2	2020-03-09	Versione 2.0.0-PoC_v_2 approvata	Marco Ferrati	*Responsabile*

1.4.0-PoC_v_1	2020-03-08	Verifica generale del documento per la consegna con esito positivo	Lorenzo Taschin	*Verificatore*
1.3.0-PoC_v_1	2020-03-03	Verifica di §2.2.1.2.7 e §2.2.1.8 con esito positivo	Lorenzo Taschin	*Verificatore*
1.3.0-PoC_v_1	2020-03-03	Stesura §2.2.1.2.7 e §2.2.1.2.8	Marco Ferrati	*Redattore*
1.2.0-PoC_v_1	2020-03-01	Verifica di §2.2.1.2.3, §2.2.1.2.4, §2.2.1.2.5 e §2.2.1.2.6 con esito positivo	Lorenzo Taschin	*Verificatore*
1.2.0-0	2020-03-01	Correzione §2.2.1.2.3, §2.2.1.2.4, §2.2.1.2.5 e §2.2.1.2.6	Xiaowei Wen	*Redattore*
1.2.0-0	2020-03-01	Verifica di §2.2.1.2.3, §2.2.1.2.4, §2.2.1.2.5 e §2.2.1.2.6 con esito negativo	Lorenzo Taschin	*Verificatore*
1.2.0-0	2020-02-29	Stesura §2.2.1.2.3, §2.2.1.2.4, §2.2.1.2.5 e §2.2.1.2.6	Xiaowei Wen	*Redattore*
1.2.0-0	2020-02-28	Verifica di §2.2.1.3.2 con esito positivo	Lorenzo Taschin	*Verificatore*
1.1.0-0	2020-02-26	Aggiunte delle specifiche dei file Java di Android in §2.2.1.3.2	Xiaowei Wen	*Redattore*
1.1.0-0	2020-02-25	Verifica di §2.1.1.1.2 con esito positivo	Lorenzo Taschin	*Verificatore*
1.0.1-0	2020-02-24	Aggiunto come pianificare gli incrementi in §2.1.1.1.2	Xiaowei Wen	*Redattore*
1.0.1-0	2020-02-23	Verifica dei paragrafi corretti superata	Stefano Cavaliere	*Verificatore*
1.0.0-0	2020-02-22	Correzione dei paragrafi secondo la precedente verifica	Giuseppe Zatta	*Redattore*

1.0.0-0	2020-02-21	Verifica dei paragrafi 2.1.1.4 e 2.2.1.2 non superata	Stefano Cavaliere	*Verificatore*
1.0.0-0	2019-02-18	Correzione post RR	Giuseppe Zatta	*Redattore*
1.0.0-0	2019-12-30	Documento approvato e pronto per la consegna	Xiaowei Wen	*Responsabile*
0.1.0-0	2019-12-29	Terza revisione generale con esito positivo	Stefano Cavaliere	*Verificatore*
0.0.0-0	2019-12-28	Correzioni varie sezione 4	Giuseppe Zatta	*Redattore*
0.0.0-0	2019-12-28	Correzioni varie sezione 3	Riccardo Tassetto	*Redattore*
0.0.0-0	2019-12-27	Seconda revisione generale	Lorenzo Taschin	*Verificatore*
0.0.0-0	2019-12-26	Termine stesura sezione 3 e 4	Riccardo Tassetto	*Redattore*
0.0.0-0	2019-12-25	Continuazione stesura Processi organizzativi	Riccardo Tassetto	*Redattore*
0.0.0-0	2019-12-22	Correzione sezione 3 e 4 dopo verifica	Riccardo Tassetto	*Redattore*
0.0.0-0	2019-12-22	Correzione sezione 1 e 2 dopo verifica	Giuseppe Zatta	*Redattore*
0.0.0-0	2019-12-22	Prima revisione generale	Stefano Cavaliere, Lorenzo Taschin	*Verificatore*
0.0.0-0	2019-12-19	Terminata stesura sezione 4 e modifica Processi di supporto	Riccardo Tassetto	*Redattore*
0.0.0-0	2019-12-16	Terminata prima stesura sezione 3	Riccardo Tassetto	*Redattore*
0.0.0-0	2019-12-13	Inizio stesura sezioni 3 e 4	Riccardo Tassetto	*Redattore*
0.0.0-0	2019-12-12	Scrittura Sviluppo in Processi Primari	Giuseppe Zatta	*Redattore*
0.0.0-0	2019-12-02	Scrittura Introduzione e Processi Primari	Giuseppe Zatta	*Redattore*
0.0.0-0	2019-11-24	Creazione scheletro del documento	Claudia Zattara	*Redattore*

Tabella 1: Registro delle Modifiche.

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Lo scopo di questo documento è di definire le best practice e il way of working che ogni membro del gruppo *VarTmp7* è tenuto ad adottare durante tutto il ciclo di vita del progetto *Stalker* al fine di garantire quanto più possibile l'uniformità del materiale prodotto.

1.2 Glossario

Al fine di evitare ogni ambiguità relativa al linguaggio impiegato nei documenti viene fornito il *Glossario_ v.4.0.0-R1*, contenente la definizione dei termini in corsivo marcati con una $_{G}$ a pedice.

1.3 Maturità del documento

Questo documento sarà soggetto a futuri incrementi per cui non si può definire completo alla attuale versione del documento. Questa decisione è presa al fine di trattare preventivamente e con maggior cura i problemi maggiormente impellenti e ricorrenti. È possibile trovare la pianificazione degli incrementi nel capitolo 4 del *Piano di progetto_ v.3.0.0-INC-11*.

1.4 Riferimenti

1.4.1 Normativi

- capitolato d'appalto C5:
  https://www.math.unipd.it/~tullio/IS-1/2019/Progetto/C5.pdf;

1.4.2 Informativi

- ISO-IEC 12207:1997:
  https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf;
- ISO 31-0:
  https://bsol.bsigroup.com/Bibliographic/BibliographicInfoData/000000000030150465.

2 Processi primari

2.1 Fornitura

Il processo di fornitura è un processo essenziale nello sviluppo software, ed infatti è uno dei processi primari. Il suo principale scopo è di fornire al cliente il prodotto da lui richiesto, ovvero che soddisfa le sue richieste (requisiti). Per fare ciò è necessario determinare le *procedure*$_{\text{G}}$ e le risorse necessarie al corretto svolgimento del progetto. A tale fine è necessario:

- comprendere le richieste del proponente e da ciò determinarne la fattibilità;
- redigere un *Piano di progetto* che ci aiuti a portare a termine tutte le richieste;
- mantenere una discussione proficua con il proponente in merito ai seguenti temi di discussione:
  - determinare gli aspetti prioritari;
  - determinare vincoli su requisiti e processi;
  - stimare i costi (temporali) del lavoro;
  - accordarsi sulla qualifica del prodotto.

2.1.1 Attività

2.1.1.1 Studio di fattibilità

L'obiettivo di questa attività è analizzare ogni *capitolato*$_{\text{G}}$ proposto al fine di discutere le varie possibilità che poi sfoceranno nello *Studio di fattibilità* e di conseguenza nella scelta del capitolato stesso. In tal documento, per ogni capitolato proposto verranno indicati:

- informazioni sul capitolato;
- descrizione ed obiettivo finale;
- dominio tecnologico;
- considerazioni del gruppo.

Queste ultime, a loro volta, saranno suddivise in:

- aspetti positivi;
- criticità e fattori di rischio;
- conclusioni.

2.1.1.2 Piano di progetto

Il *Piano di progetto*, redatto dal *Responsabile* affiancato dagli *Amministratori* ha come obiettivo la pianificazione delle *attività*$_{\text{G}}$ che guideranno il processo durante le tutte le *fasi*$_{\text{G}}$ del progetto. A questo fine vengono dunque organizzate le risorse disponibili, che verranno quindi distribuite lungo i vari periodi di tempo previsti dal *Piano di progetto*. In particolare questo documento dovrà affrontare le seguenti tematiche:

- **Analisi dei Rischi**: si analizzano i rischi nei quali si potrebbe incorrere durante lo sviluppo del progetto. Questi rischi vengono poi classificati per gravità e per la probabilità di incorrere in essi;

- **Modello di ciclo di vita**: si definisce il *modello di ciclo di vita*$_G$ scelto per questo progetto, ivi comprese le motivazioni che hanno portato alla scelta;
- **Pianificazione**: si rende necessario pianificare con attenzione le attività da eseguire nei vari incrementi del progetto e le loro scadenze temporali. Considerato il modello di ciclo di vita incrementale è necessario normare anche gli incrementi stessi, che verranno poi definiti nel *Piano di Progetto*. Essi dovranno essere strutturati nel seguente modo:
  - **Codice Identificativo Univoco**;
  - **Pre-condizione**: si tratta di attività o altri incrementi che risultano essere bloccanti per l'inizio di questo incremento;
  - **Post-condizione**: condizioni attraverso le quali si può stabilire che l'incremento è terminato;
  - **Data inizio**: data di inizio prevista;
  - **Data fine**: data prevista per il completamento dell'incremento;
  - **Ruoli attivi**: ruoli lavorativi previsti per l'incremento;
  - **Attività previste**;
  - **Requisiti da soddisfare**: elenco dei requisiti, del documento *Analisi dei Requisiti*, che l'incremento deve soddisfare;
  - **Note**.
- **Preventivi e Consuntivo**: al fine di calcolare un preventivo vengono stimati i costi del lavoro di ogni attività. In seguito verranno anche forniti i preventivi a finire ed i consuntivi al fine di calcolare le differenze tra il preventivo iniziale, il preventivo a finire ed il consuntivo per poter ricalcolare i successivi preventivi con una precisione maggiore.

2.1.1.3 Piano di qualifica

Il *Piano di qualifica*, redatto dai *Verificatori* ha come obiettivo la scelta delle strategie per la *qualifica*$_G$ delle attività al fine di garantire la *qualità*$_G$ del materiale prodotto dal gruppo. I contenuti principali del *Piano di qualifica* saranno:

- **Qualità di processo**: si identificano le metriche per il controllo della qualità del processo;
- **Qualità di prodotto**: si identificano le metriche per il controllo della qualità del prodotto;
- **Specifica dei Test**: qui vengono specificati dei test necessari a garantire che il prodotto soddisfi tutti i requisiti. Vengono raggruppati in tabelle, una per ogni categoria di test:
  - Test di sistema: identificati da TCS-X con X un numero incrementale.
  - Test di integrazione: identificati da TI-X con X un numero incrementale
  - Test di unità: identificati da TU-X con X un numero incrementale

  Ogni test ha uno stato; tale stato può assumere valore **NI** che sta per non implementato oppure **I** che sta per implementato.
  Ogni test ha una descrizione che deve specificare come deve avvenire il test.
  La specifica dei test segue il modello a V.
- **Standard di qualità**: si delineano gli *standard di qualità*$_G$ scelti;

- **Resoconto delle attività di verifica**: qui vengono riportati i risultati delle attività di *verifica*$_{\text{G}}$ e di *validazione*$_{\text{G}}$;
- **Valutazioni per il Miglioramento**: al fine di un miglioramento continuativo nel tempo qui vengono riportate le problematiche relative al ricoprire un determinato ruolo o all'utilizzo di un determinato strumento, insieme con le relative soluzioni proposte.

2.1.1.4 Gestione contrattuale

Questa attività consiste nella accettazione del contratto e nella negoziazione con il proponente riguardo al contenuto dello stesso o a richieste di modifica dei requisiti funzionali del prodotto software. Essa prevede anche la chiusura dello stesso al termine dell'attività "Consegna del prodotto".
Per favorire un dialogo continuativo con il proponente, rappresentato da Zanetti Davide e Cardona Tommaso, il gruppo VarTmp7 si impegna a contattarlo tramite canali di messaggistica quali Slack, email e videochiamate in particolare per:

- chiarimenti sui requisiti richiesti;
- consigli di natura tecnica;
- qualità richiesta nella realizzazione del prodotto.

2.1.1.5 Consegna del prodotto

Questa attività prevede la consegna del prodotto software realizzato, insieme alla documentazione relativa, al proponente al termine del lavoro.
Per adempiere a ciò il gruppo VarTmp7 renderà open-source il codice sorgente realizzato tramite la piattoforma GitHub. Oltre al codice sorgente verrà reso disponibile anche la documentazione prodotta e il report richiesto dal proponente.
Tutto questo sarà disponibile all'indirizzo `https://github.com/varTmp7/Stalker` organizzato in quattro cartelle (RR, RP, RQ, RA). Il prodotto finito si troverà all'interno della cartella *RA*, disponibile da una settimana prima della Revisione di Accettazione.
A seguito della consegna, non seguirà l'attività di manutenzione del prodotto.

2.1.1.5.1 Contenuto delle comunicazioni

Nell'ottica di mantenere con il proponente un buon rapporto, si ritiene necessario normare le comunicazioni con esso. In particolar modo questi rapporti devono tenere conto di:

- **Interesse da entrambe le parti**: per rendere il proponente partecipe al lavoro del gruppo, quest'ultimo si impegna a tenere contatti costanti nel tempo; inoltre queste comunicazioni devono incentrarsi su argomenti che possano risultare interessanti anche per esso, e quindi non su temi banali;
- **Preferire un approccio concreto alle sole parole**: un testo per quanto esplicativo possa essere non è paragonabile ad un prototipo funzionante, ad uno schema o a un disegno del sistema (o di un suo sottosistema). Inoltre questi ultimi, vista la loro natura specifica e non ambigua, sono molto meno fraintendibili delle parole.

2.1.2 Strumenti

Gli strumenti relativi alle attività del processo di fornitura sono descritti in seguito.

2.1.2.1 Microsoft Project

2.1.2.1.1 Descrizione generale
Microsoft Project è un software di pianificazione sviluppato e venduto da Microsoft. è uno strumento per assistere i project manager nella pianificazione, nell'assegnazione delle risorse, nella verifica del rispetto dei tempi, nella gestione dei budget e nell'analisi dei carichi di lavoro.

2.1.2.1.2 Applicazione al progetto
Ai fini del nostro progetto viene usato in particolar modo per la produzione di *Diagrammi di Gantt*$_{\mathrm{G}}$.

2.1.2.2 Microsoft Excel

2.1.2.2.1 Descrizione generale
Microsoft Excel è un programma prodotto da Microsoft, dedicato alla produzione ed alla gestione di fogli elettronici. è parte della suite di software di produttività personale Microsoft Office, ed è disponibile per i sistemi operativi Windows e Macintosh.

2.1.2.2.2 Applicazione al progetto
Il gruppo ha scelto di utilizzare Microsoft Excel per la produzione delle tabelle presenti nel *Piano di progetto*, dei grafici a torta e a barre.

2.2 Sviluppo

Questo processo contiene le attività ed i compiti necessari alla realizzazione del prodotto finale, in particolar modo esso deve tenere conto di:

- fissare gli obiettivi di sviluppo;
- fissare i vincoli tecnologici e di design;
- realizzare un prodotto finale che supera i test;
- realizzare un prodotto finale che soddisfa i requisiti e le richieste del proponente.

Seguendo le linee guida *ISO*$_{\mathrm{G}}$ *IEC*$_{\mathrm{G}}$ 12207:1995 abbiamo suddiviso il processo nelle seguenti attività:

- Analisi dei Requisiti;
- Progettazione;
- Codifica.

2.2.1 Attività

2.2.1.1 Analisi dei Requisiti

Questo documento, redatto dagli analisti, ha lo scopo di riportare i requisiti del progetto in maniera formale.
Questi ultimi vengono estrapolati da diverse fonti:

- documenti di specifica del capitolato;
- fonti normative;
- casi d'uso;
- incontri con il proponente.

2.2.1.1.1 Casi d'uso

I casi d'uso rappresentano dei comportamenti offerti o desiderati espressi sotto forma di *diagrammi UML*$_{\mathrm{G}}$. La loro struttura è composta da:

- codice identificativo univoco, nella forma **UC-x.y** , dove le componenti significano:
  - UC specifica che si tratta di un caso d'uso;
  - x è un numero progressivo che identifica univocamente il caso d'uso;
  - y è un numero progressivo che individua un sottocaso d'uso.
- titolo;
- attori;
- scopo;
- descrizione;
- precondizioni;
- postcondizioni;

- flusso degli eventi principali;
- scenario alternativo (se presente);
- estensioni (se presenti).

2.2.1.1.2 Requisiti

I requisiti rappresentano le specifiche che il sistema deve adempiere. Questi vengono estrapolati da diverse fonti e sono composti da:

- codice identificativo univoco nella forma R-X-T-I, dove:
  - **R** sta per requisito;
  - **X** è un carattere numerico ordinato crescente che identifica il requisito univocamente all'interno della sua tipologia di requisiti;
  - **T** è un carattere alfabetico che distingue la tipologia di requisito ovvero **F**unzionale, **Q**ualitativo, **D**ichiarativo;
  - **I** è un carattere alfabetico che segnala l'importanza di un requisito che può essere **O**bbligatorio, **D**esiderabile, **F**acoltativo.
- descrizione esaustiva;
- fonte dalla quale viene estratto.

2.2.1.2 Progettazione

Questo processo viene inizializzato dopo la prima stesura del documento *Analisi dei requisiti*. A questo punto bisogna delineare le caratteristiche del prodotto e degli incrementi che esso subirà in base al modello di ciclo di vita incrementale; tali caratteristiche devono venire approvate da tutti gli *stakeholder*$_{\text{G}}$. Questo processo verrà ulteriormente suddiviso nei due suoi sottoprocessi principali:

- **Progettazione Architetturale**: qui si decide il design architetturale del prodotto software. Esso comprende lo sviluppo e la predisposizione delle baseline relative al prodotto software come anche la descrizione delle interfacce interne ed esterne dei software items.
- **Progettazione di Dettaglio**: una volta svolta la progettazione architetturale, quindi di alto livello, bisogna delinare come l'architettura software andrà realizzata e quindi per i sotfware items andranno codificati e combinati per formare il prodotto software.

2.2.1.2.1 Technology baseline

Si tratta della base tecnologica (linguaggi di programmazione, librerie esterne, framework) con la quale viene svolto il progetto; Essa comprende un PoC_{G} nel quale si mostra che le tecnologie scelte sono state comprese e implementare

2.2.1.2.2 Product baseline

La Product Baseline è il prodotto della Technology Baseline in forma incrementale applicata prototipo. Con l'aggiunta di incrementi essa diverrà il prodotto finale. Al fine di seguire e verificare lo sviluppo del prototipo nella Product Baseline ci saranno i diagrammi UML che specificano le modalità di costruzione del prototipo.

2.2.1.2.3 Design patterns

I design patterns che possono essere adottati nella progettazione del sistema Stalker sono quelli presentati a lezione e suddivisi in quattro categorie:

- **architetturale**:
  - Model-View-Controller (**MVC**);
  - Model-View (**MV**);
  - Dependency Injection;
  - Model-View-Presenter (**MVP**).
- **creazionale**:
  - Singleton;
  - Builder;
  - Abstract Factory.
- **strutturale**:
  - Adapter;
  - Decorator;
  - Façade;
  - Proxy.
- **comportamentale**:
  - Command;
  - Iterator;
  - Observer;
  - Strategy;
  - Template method.

2.2.1.2.4 Diagramma delle classi

È stato adottato lo standard **OMG UML 2.5** per rappresentare i diagrammi delle classi in modo chiaro e seguendo la semantica **UML Semantics**.
I diagrammi delle classi devono essere composti da tre parti:

1. **nome della classe**: deve essere significativo e la presenza di questa parte è obbligatoria. Deve essere scritto nel formato **CamelCase**. Per esempio: NomeClasse.
   Nel caso in cui la classe che si sta modellando sia un'interfaccia, allora il nome deve essere scritto in questo modo: **«NomeInterfaccia»**. Nel caso in cui la classe che si sta modellando sia una classe astratta, allora il nome deve essere scritto in ***italico***.
2. **elenco degli attributi**: nel caso in cui non ci siano attributi nella classe che si sta modellando bisogna lasciare uno spazio vuoto.
   Ogni attributo è composto da:
   - l'indicatore di visibilità, che può essere:

- +: pubblico, un attributo con questo indicatore di accessibilità è accessibile da qualunque classe;
- ~: package, un attributo con questo indicatore di accessibilità è accessibile solo alle classi che appartengono allo stesso package;
- #: protetto, un attributo con questo indicatore di accessibilità è accessibile solo alle proprie sotto classi;
- **-**: privato, un attributo con questo indicatore di accessibilità è accessibile solo all'interno della propria classe.

Di default, tutti gli attributi devono essere privati anche se possono esistere delle eccezioni in situazioni particolari. Per esempio i valori costanti e statici possono essere pubblici se il progettista ritiene che quel valore possa essere utile anche all'esterno della classe che lo contiene.

- **il nome dell'attributo**: deve essere significativo e deve essere scritto nel formato **CamelCase** con la prima lettera in minuscolo. Per esempio: nomeAttributo. Se l'attributo dovesse essere costante allora il nome deve essere scritto tutto in maiuscolo e utilizzare _(carattere underscore) come spazio. Per esempio: NOME_COSTANTE.
- **il tipo dell'attributo**: può essere sia primitivo che una classe definita dal progettista. Nel secondo caso bisogna indicare il nome della classe come tipo;
- **la molteplicità**: di default è **0..1** in questo caso può essere omessa, altrimenti può assumere seguenti valori:
  - **0..***: da zero a N;
  - ***: N;
  - **...**: indefinita;

3. **elenco delle operazioni**: se non ci sono operazioni nella classe che si sta modellando bisogna lasciare uno spazio vuoto.
   Ogni operazione è composta da:
   - **livello di accessibilità**: definito nello stesso modo degli attributi;
   - **nome**: definito nello stesso modo degli attributi;
   - **lista dei parametri**: indica l'elenco dei parametri che l'operazione richiede;
   - **tipo ritornato**: può essere un tipo semplice o un tipo definito.

Nel caso in cui l'operazione venisse implementata nuovamente in una sottoclasse, essa dev'essere riportata nel diagramma della sottoclasse, mentre in caso contrario ciò non è necessario. Per quanto riguardano i commenti e le note: bisogna valutare caso per caso la necessità di introdurli in quanto in caso se ne usino troppi lo svantaggio della loro manutenibilità supererebbe il vantaggio della loro utilità. Per indicare un commento si usa un rettangolo, con un angolo piegato, collegato al diagramma della classe per mezzo di una linea tratteggiata.

2.2.1.2.5 Relazione di dipendenza

Si ha l'accoppiamento tra due elementi di un diagramma se una modifica alla definizione del primo può rendere necessaria una modifica della definizione del secondo, causando in questo modo difficoltà di manutenzione del codice sorgente e del diagramma stesso. Sarà quindi opportuno

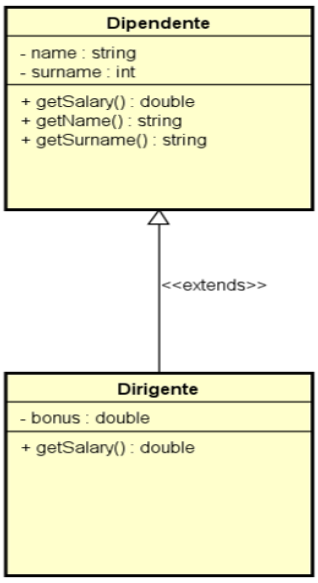

Figura 1: Esempio di diagramma corretto.

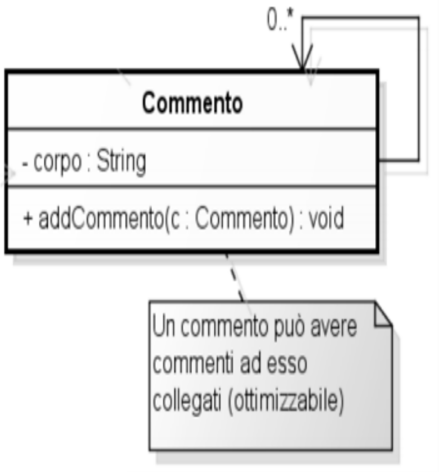

Figura 2: Esempio di diagramma di classe con commento corretto.

minimizzare le dipendenze per aumentare la manutenibilità del software. Come illustrato nella figura 3, ci sono cinque tipi di accoppiamento:

- **Dipendenze**: quando un'istanza di una classe collabora brevemente con l'istanza di un'altra classe. Viene indicato con una linea tratteggiata terminante con una freccia orientata. È il tipo di accoppiamento che si preferisce avere;
- **Associazione**: quando le istanze di una classe collaborano con le istanze di un'altra classe

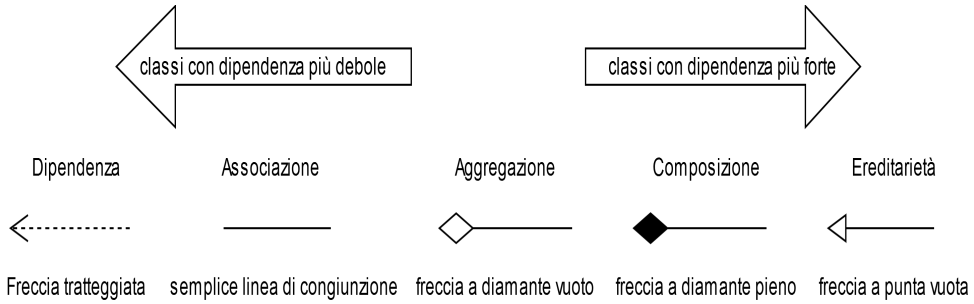

Figura 3: Diversi tipi di dipendenze.

per qualche unità di tempo prolungato. Viene indicato con una linea continua;

- **Aggregazione**: quando una classe possiede ma condivide reference a oggetti di un'altra classe. Viene indicato con una freccia con la punta a diamante vuota;
- **Composizione**: quando una classe contiene istanze di un'altra classe. Viene indicato con una freccia con la punta a diamante piena;
- **Ereditarietà**: quando una classe è un sottotipo di un'altra. Viene indicato con una freccia a punta triangolare vuota.

Per specificare meglio le dipendenze, si possono usare le seguenti direttive:

- **«call»**: la sorgente invoca un'operazione della classe destinazione;
- **«create»**: la sorgente crea istanze della classe destinazione;
- **«derive»**: la sorgente è derivata dalla classe destinazione;
- **«instantiate»**: la sorgente è una istanza della classe destinazione;
- **«permit»**: la classe destinazione permette alla sorgente di accedere ai suoi campi privati;
- **«realize»**: la sorgente è un'implementazione di una specifica o di un'interfaccia definita dalla sorgente;
- **«refine»**: raffinamento tra differenti livelli semantici;
- **«substitute»**: la sorgente è sostituibile alla destinazione;
- **«trace»**: tiene traccia dei requisiti o di come i cambiamenti di una parte del modello si colleghino ad altre;
- **«use»**: la sorgente richiede la destinazione per la sua implementazione.

2.2.1.2.6 Diagramma dei package

Un package in UML individua uno spazio dei nomi, in cui non si possono avere due entità con lo stesso nome e serve per raggruppare un numero arbitrario di elementi UML in un'unità di livello più alto. Gli elementi sono individuati da un nome univoco nel package e preceduto da un indicatore di visibilità:

- **+**: pubblico, l'elemento è visibile alle classi di un altro package;
- **-**: privato, l'elemento non è visibile alle classi di un altro package.

Il principio di progettazione adottato è il **Common Reuse Principle** ovvero le classi dello stesso package dovrebbero essere sempre riusate insieme per garantire la facilità di versionamento.
Un package si dice che dipende da un altro quando esiste una classe appartenente al primo che dipende da una classe appartenente al secondo. Bisogna quindi evitare le dipendenze circolari in questo modo:

1. individuare le classi che causano la dipendenza circolare dei due package;
2. creare un terzo package e spostare le classi del punto uno nel package appena creato.

Oppure si possono unire le classi dei due package all'interno di uno stesso package, il progettista deve valutare qual è la soluzione migliore.

2.2.1.2.7 Diagrammi di sequenza

I diagrammi di sequenza modellano il modo in cui i diversi oggetti collaborano tra di loro per implementare un comportamento. Per la loro realizzazione il gruppo VarTmp7 si appoggia allo standard *OMG UML 2.5*.
Ogni diagramma è identificato da un nome nella forma **sd Nome dell'interazione**.
Gli oggetti coinvolti (detti partecipanti) sono identificati da un rettangolo con all'interno un'etichetta nella forma **istanza: NomeClasse**. Sotto ad ogni istanza vi è una linea tratteggiata raffigurante la vita dell'istanza sovrastante.

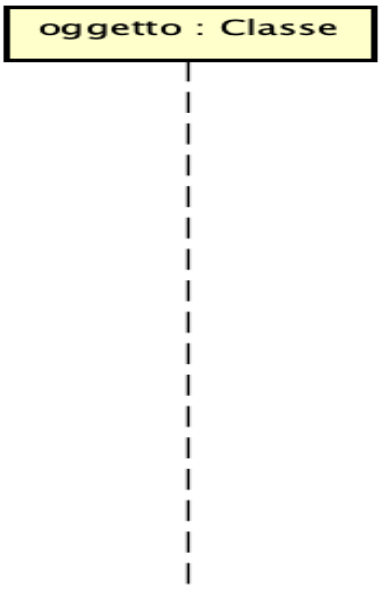

Figura 4: Esempio di oggetto o partecipante e della sua vita in un diagramma di sequenza.

Un intervallo di tempo nel quale un partecipante è attivo viene indicato con un rettangolo bianco, detto *barra di attivazione*, sovrastante la linea tratteggiata.

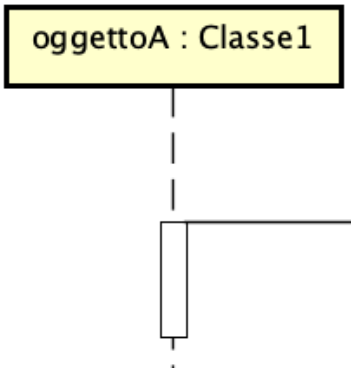

Figura 5: Esempio della barra di attivazione di un oggetto in un diagramma di sequenza.

Da una barra di attivazione partiranno dei *messaggi* rappresentanti le chiamate ai metodi degli oggetti coinvolti. Il primo messaggio, che scaturisce dall'esterno, viene chiamato *messaggio trovato* e viene rappresentato con una pallina nera al suo capo sinistro. Le diverse tipologie di messaggi si rappresentano mediante delle frecce che vanno da una barra di attivazione ad un'altra.

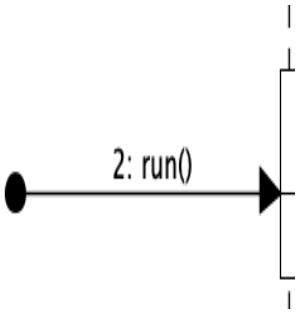

Figura 6: Esempio del messaggio trovato in un diagramma di sequenza.

Le diverse tipologie di messaggi possono essere:

- **messaggio sincrono**: freccia continua con la punta chiusa e piena con etichetta il nome del metodo invocato nel formato **nomeMetodo(lista parametri formali)**;
- **messaggio asincrono**: freccia continua con la punta aperta con etichetta il nome del metodo invocato nel formato **nomeMetodo(lista parametri formali)**;
- **messaggio di ritorno**: freccia tratteggiata con la punta aperta;
- **messaggio per la creazione di un nuovo partecipante**: freccia tratteggiata con la punta aperta direzionata verso il nuovo oggetto e con l'etichetta **«create»**;
- **messaggio per la distruzione di un partecipante**: freccia continua con la punta piena terminante con una **X** sulla vita dell'istanza che si sta distruggendo e con l'etichetta **«destroy»**.

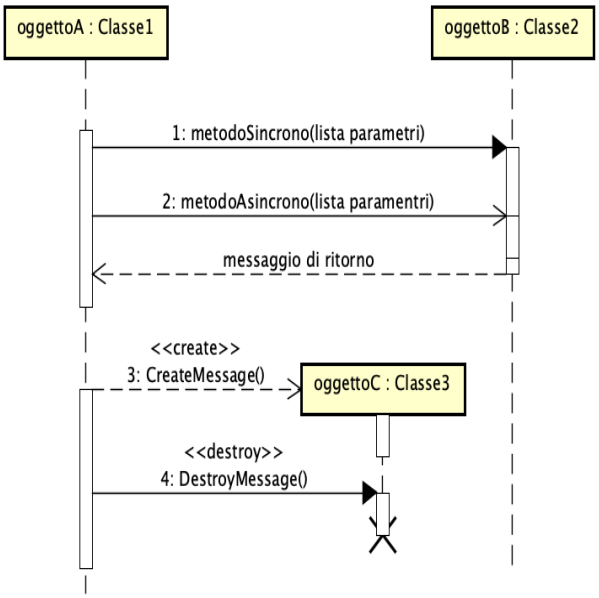

Figura 7: Messaggi che due oggetti si possono scambiare in un diagramma di sequenza.

È inoltre possibile definire dei *frame di interazione* utili a modellare cicli e condizioni. Essi devono avere definito l'operatore nell'etichetta in alto a sinistra del rettangolo in cui sono contenuti, e la guardia quando necessaria.
I diversi tipi di operatore sono:

- **alt**: frammenti multipli in alternativa; verrà eseguito solo quello per cui è soddisfatta la guardia;
- **opt**: il frammento viene eseguito solo se la guardia viene superata;
- **par**: ogni frammento viene eseguito in parallelo;
- **loop**: il frammento viene eseguito più volte, la base dell'iterazione è indicata dalla guardia;
- **region**: il frammento può essere eseguito da un solo thread alla volta;
- **neg**: il frammento mostra un'interazione non valida;
- **ref**: si fa riferimento ad un'interazione definita in un altro diagramma;
- **sd**: racchiude l'intero diagramma di sequenza.

2.2.1.2.8 Diagrammi di attività

I diagrammi di attività descrivono la logica procedurale, i processi di business e gli aspetti dinamici dei casi d'uso. Per la loro realizzazione il gruppo VarTmp7 si appoggia allo standard *OMG UML 2.5*.
Ogni diagramma è identificato da un nome nel formato indicante l'attività che quel diagramma rappresenta.
Un'attività è un insieme di più azioni (*nodes*) collegate tra loro mediante frecce continue con la punta aperta. Le possibili azioni rappresentabili sono:

- **nodo iniziale**: pallino pieno che indica da dove inizia l'esecuzione del processo, genera un *token*;
- **fork**: linea spessa orizzontale o verticale, da quel punto in poi l'attività si parallelizza, ha una freccia in ingresso e n in uscita; vengono generati n token;
- **join**: linea spessa orizzontale o verticale, in quel punto avviene la sincronizzazione di processi generati da una fork, ha n frecce in ingresso e una in uscita; vengono consumati n token e generato uno;
- **activity**: rettangolo con una descrizione molto breve; consuma e produce un token;
- **sotto-attività**: rettangolo non direttamente collegato all'attività che si sta descrivendo che può però essere richiamato mediante un'activity decorata con un simbolo raffigurante un tridente;
- **branch**: rombo con una freccia in ingresso e n in uscita, si può prendere solo uno degli n rami; in prossimità di ogni ramo dev'esserci scritta la guardia nel formato **[guardia]** indicante secondo quali condizioni si può percorrere quel percorso;
- **merge**: rombo con n frecce in ingresso e una in uscita, è il punto in cui gli n rami generati da un branch si uniscono;
- **pin**: piccolo quadratino dove entrano/escono frecce delle activity, indicano il passaggio di parametri tra un'activity e un'altra. Per ognuno di essi va indicato il **TipoParametro**;
- **nodo di fine flusso**: cerchio vuoto con una X al centro, rappresenta il caso in cui uno dei rami di esecuzione termina (l'attività può proseguire sui rami paralleli ancora attivi); conusma un token;
- **nodo finale**: due cerchi conentrici di cui il più interno è nero, rappresenta il punto in cui termina l'esecuzione; consuma un token.

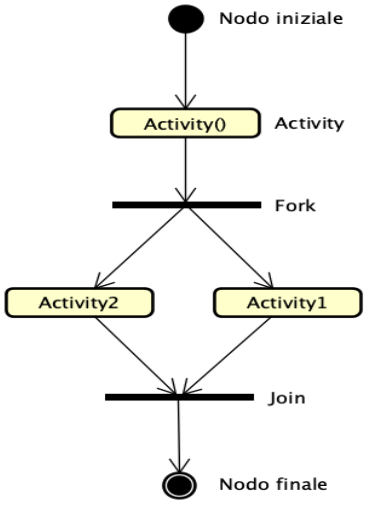

Figura 8: Nodo iniziale, activity, fork, join e nodo finale in un diagramma di attività.

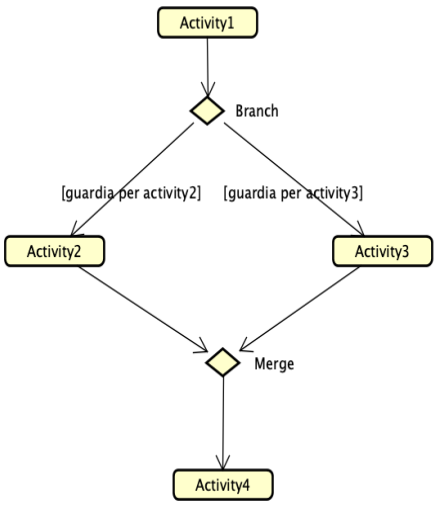

Figura 9: Branch e merge in un diagramma di attività.

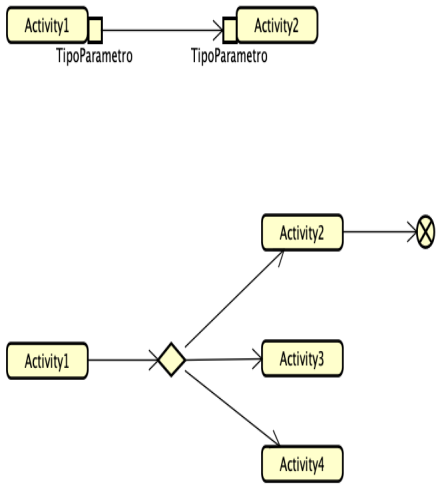

Figura 10: Esempio di pin e nodo di fine flusso in un diagramma di attività.

Per modellare un evento si fa uso dei segnali. Ne esistono di diverso tipo:

- **eventi provenienti da un processo esterno**: si usano i segnali "a incastro", due figure complementari. La prima rappresenta un evento non bloccante sul quale non si ha controllo sull'esecuzione, essa deve avere un'etichetta nel formato **«signal sending» nome del segnale**. La seconda rappresenta un evento bloccante rappresentante la ricezione del messaggio di risposta dalla prima figura, essa deve avere un'etichetta nel formato **«signal recepit» nome del segnale**;
- **eventi temporali**: rappresentati da una clessidra si differenziano in due categorie:

- **timeout**: hanno una freccia entrante ed una uscente, devono avere un'etichetta nel formato **wait *n* min** con n il numero di minuti da attendere prima di proseguire con l'esecuzione;
- **eventi ripetuti**: hanno solo una freccia uscente, devono avere un'etichetta nel formato **every *n* min** con n il numero di minuti ogni quanto viene inviato il segnale.

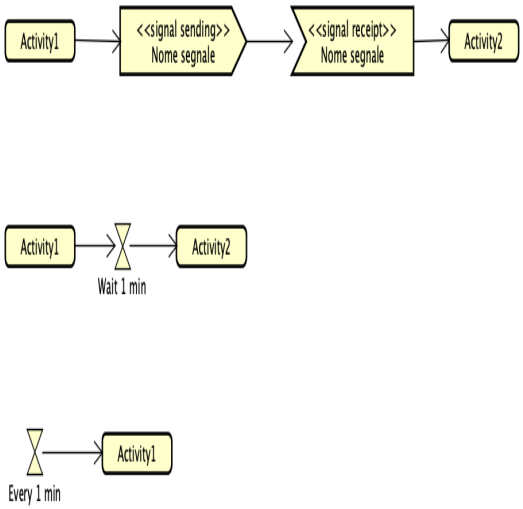

Figura 11: Esmepio di segnali in un diagramma di attività.

2.2.1.3 Codifica

Normare la codifica è necessario per:

- una corretta gestione della qualità;
- permettere la generazione di codice leggibile ed uniforme;
- migliorare la qualità del prodotto;
- agevolare le fasi di verifica, validazione e manutenzione del prodotto.

2.2.1.3.1 Regole generali

Le seguenti norme dovranno essere applicate a tutti i file, a prescindere dal linguaggio di programmazione scelto:

- **indentazione**: i blocchi di testo devono venire correttamente indentati, con una indentazione di 4 caratteri spazio;
- **parentesizzazione**: le parentesi andranno messe "in linea" e non a capo;
- **nomi**: i nomi dovranno essere semplici ed esplicativi ed a seconda di ciò a cui si riferiscono si può avere un ulteriore definizione di queste stesse norme:
  - per i nomi delle **classi** verrà seguita la notazione *CamelCase*$_{\text{G}}$ (esempio: NomeClasse);
  - per i nomi delle **funzioni** verrà seguita la notazione CamelCase, con l'eccezione della prima lettera che andrà minuscola (esempio: nomeFunzione);

- i nomi delle **costanti** verrà seguita la notazione *SnakeCase*$_G$ e con tutte le lettere maiuscole (esempio: NOME_COSTANTE).

2.2.1.3.2 File Java

Per quanto riguarda i file scritti in questo linguaggio di programmazione, si dovranno seguire anche le seguenti norme:

- **attribuzione classi**: ogni classe Java creata dovrà comparire un campo TAG statico, costante, privato che dovrà contenere come valore la stringa "com.vartmp7.Stalker.[percorso package].nome classe" dove

  $$percorsopackage$$

  indica la gerarchia di *package*$_G$ della classe Java in oggetto;
- **metodi Log**: per debugging nell'ambiente di sviluppo Android Studio, al posto di usare funzioni generali di stampa (System.out.println()) dovranno essere usati i seguenti metodi statici. Questi avranno come parametri il campo TAG(vedi punto precedente) come primo parametro e il messaggio da stampare. I metodi usabili saranno:
  - Log.v,*verbose*;
  - Log.d, *debug*;
  - Log.i, *info*;
  - Log.w, *warning*;
  - Log.e, *error*;
  - Log.wtf, *What the failure.*

  Questi metodi statici stampano dei messaggi di output che possono essere filtrati sia in base al metodo utilizzato:

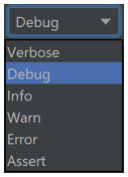

Figura 12: Filtri di default messi a disposizione da Android Studio

  Inoltre è possibile creare dei filtri attraverso la funzione **Edit filter configuration** con la quale è possibile filtrare attraverso dei parametri il log:
  - il parametro TAG;
  - il messaggio di Log;
  - il nome dei Package;
  - il *PID*$_G$

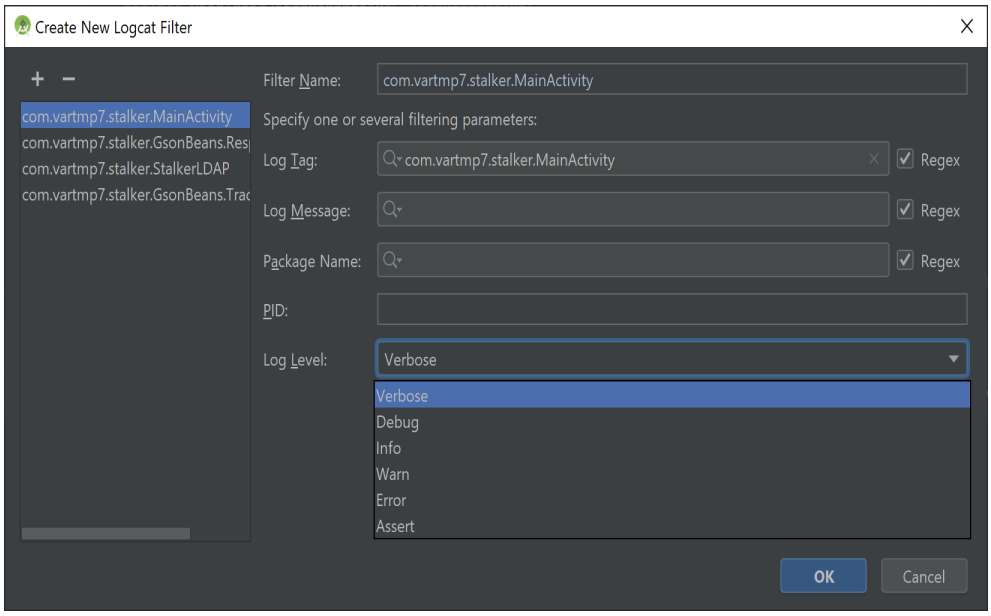

Figura 13: Creazione dei filtri personalizzati

2.2.1.3.3 Test applicazione Android

Per specificare un nuovo file di test questo deve essere nel formato *Test.java che deve essere salvato nella cartella test/.

2.2.1.3.4 Test backend

Per specificare un nuovo file di test questo deve essere nel formato test_*.py che deve essere salvato nella cartella tests/.
I metodi di test devono essere parlanti e nel formato `test_*()`.

2.2.1.3.5 Test applicazione web

Per specificare un nuovo file di test questo deve essere nel formato *.spec.ts. Usando la Angular CLI quando viene generato un componente o un servizio vengono generati anche i file di test.

2.2.2 Strumenti

2.2.2.1 Astah UML

2.2.2.1.1 Descrizione generale

Astah UML è un editor di modellazione UML.

2.2.2.1.2 Applicazione al progetto

Abbiamo scelto di utilizzare questo editor veloce ed intuitivo per la produzione dei diagrammi dei casi d'uso, di attività, di classi e di sequenza.

2.2.2.2 Android Studio

2.2.2.2.1 Descrizione generale
Android Studio è un ambiente di sviluppo integrato (IDE) per lo sviluppo per la piattaforma Android. È disponibile gratuitamente sotto licenza Apache 2.0.

2.2.2.2.2 Applicazione al progetto
Abbiamo usato questo ambiente di sviluppo per la codifica e la verifica della applicazione Android. del sistema Stalker.

2.2.2.3 PyCharm

2.2.2.3.1 Descrizione generale
PyCharm è un ambiente di sviluppo integrato utilizzato nella programmazione informatica, in particolare per il linguaggio Python. È sviluppato dalla società ceca JetBrains.

2.2.2.3.2 Applicazione al progetto
Abbiamo usato questo ambiente di sviluppo per la codifica e la verifica della servizio di backend.

2.2.2.4 WebStorm

2.2.2.4.1 Descrizione generale
WebStorm è un IDE utile per lo sviluppo software attraverso il linguaggio Javascript. I principali vantaggi di questo IDE sono:

- numerose funzioni per l'auto completamento del codice;
- supporto per TypeScript e NodeJS;
- supporto per i sistemi di versionamento.

2.2.2.4.2 Applicazione al progetto
Abbiamo usato questo ambiente di sviluppo per la codifica e la verifica della Web Application.

3 Processi di supporto

3.1 Documentazione

Lo scopo di questa sezione è definire gli standard riguardanti i documenti relativi ad ogni processo e attività significativi, volti allo sviluppo del progetto. I documenti sono consultabili nelle apposite sezioni del *repository*$_G$: *https://github.com/varTmp7/Stalker-Documents*. Ogni documento segue le fasi del seguente ciclo di vita:

- **creazione del documento**: viene creato il documento in conformità alle norme adeguandolo a documenti precedenti dello stesso tipo, utilizzando il template reperibile nel repository;
- **realizzazione**: il documento viene scritto progressivamente, utilizzando un approccio incrementale;
- **in revisione**: il documento è prodotto interamente e rimane in attesa di verifica. Ogni documento viene verificato da almeno una persona, diversa dal redattore dello stesso.
- **approvato**: il responsabile di progetto dichiara che il documento è completo in ogni sua parte e pronto per essere rilasciato.

3.1.1 Attività

3.1.1.1 Template

Per facilitare la stesura dei documenti, è stato creato un template *LaTeX* che contiene la struttura grafica e lo stile di formattazione adottati. In questo modo, oltre a velocizzare la stesura permettendo ai redattori di concentrarsi solo sulla scrittura dei contenuti, il template permette di adottare automaticamente le conformità previste dalle *Norme di progetto*.

3.1.1.2 Struttura dei documenti

Ogni documento, ad eccezione della lettera di presentazione, deve essere realizzato secondo la seguente struttura.

3.1.1.2.1 Prima pagina

Il frontespizio è la prima pagina del documento ed è cosí formato:

- logo del gruppo: logo di *VarTmp7* posizionato in alto a centro pagina;
- gruppo e progetto: nome del gruppo e del progetto *Stalker*, visibili centralmente subito sotto il logo;
- titolo: nome del documento, posizionato centralmente in grassetto;
- tabella: presente sotto il titolo del documento, centrale e contenente le seguenti informazioni:
  - versione: versione del documento;
  - approvatori: nome e cognome dei membri del gruppo incaricati dell'approvazione del documento;
  - redattori: nome e cognome dei membri del gruppo incaricati della redazione del documento;

- verificatori: nome e cognome dei membri del gruppo incaricati della verifica del documento;
- uso: tipo d'uso che puo essere interno o esterno;
- distribuzione: destinatari del documento.

- descrizione: descrizione sintetica relativa al documento, centrale, posta a fondo pagina.

3.1.1.2.2 Registro delle modifiche
Ogni documento dispone di un *Registro delle Modifiche*, ovvero un *changelog*$_{\text{G}}$ in formato tabellare che riporta le seguenti affermazioni:

- versione del documento dopo la modifica;
- data della modifica;
- descrizione sintetica della modifica apportata;
- nominativo di chi ha apportato la modifica;
- ruolo di chi ha apportato la modifica.

3.1.1.2.3 Indice
Lo scopo dell'indice è dare una visione d'insieme della struttura del documento, mostrando le parti gerarchiche che lo compongono. Ogni documento ha al suo interno l'indice dei contenuti, posizionato dopo il *Registro delle Modifiche*.

3.1.1.2.4 Elenco immagini
I documenti muniti di figure saranno provvisti di un indice delle figure contenente, per ciascuna, una breve descrizione e il numero della pagina in cui è collocata.

3.1.1.2.5 Elenco tabelle
I documenti muniti di tabelle saranno provvisti di un indice delle tabelle contenente, per ciascuna, una breve descrizione e il numero della pagina in cui è collocata.

3.1.1.2.6 Contenuto principale
La struttura delle pagine di contenuto è la seguente:

- il logo del gruppo è posizionato in alto a sinistra;
- il numero e nome della sezione sono posizionati in alto a destra;
- una riga divide l'intestazione dal contenuto;
- tra l'intestazione e il piè di pagina troviamo il contenuto;
- una riga divide il contenuto dal piè di pagina;
- in basso a destra è presente il numero della pagina corrente e il numero complessivo di pagine del documento.

3.1.1.3 Classificazione dei documenti
Tutti i documenti devono ricevere l'approvazione del *Responsabile* prima della loro divulgazione.

3.1.1.3.1 Documenti ad uso interno

Sono da considerarsi documenti ad uso interno al gruppo:

- tutti i *Verbali interni*;
- lo *Studio di fattibilità*;
- le *Norme di progetto*.

3.1.1.3.2 Documenti ad uso esterno

I documenti destinati ad essere divulgati all'esterno del gruppo sono:

- l'*Analisi dei requisiti*;
- il *Piano di qualifica*;
- il *Piano di progetto*;
- la *Lettera di presentazione*;
- il *Glossario*.

3.1.1.3.3 Verbali

I verbali vengono prodotti da un incaricato che avrà il ruolo di *redattore*, ad ogni incontro interno al gruppo o con i proponenti del progetto. Per i verbali è prevista un'unica stesura, in quanto una modifica successiva all'incontro implicherebbe un cambio nelle decisioni prese in modo retroattivo. Oltre alla struttura standard di tutti gli altri documenti descritta in precedenza, il verbale è costituito nelle pagine di contenuto da una sezione di *Informazioni generali*, che contengono nell'ordine:

- luogo di svolgimento della riunione;
- data nella quale si è svolta;
- ora di inizio;
- ora di fine;
- membri del gruppo/esterni presenti.

Segue l'*Ordine del giorno*, una breve descrizione degli argomenti da discutere nella riunione. Successivamente è presente il *Verbale della riunione*, una descrizione più dettagliata, di tutti gli argomenti discussi. In seguito troviamo una tabella che raccoglie il tracciamento delle decisioni prese durante l'incontro. Ogni riga è costituita da:

- un codice che identifica la scelta fatta nel formato *YYYY-MM-DD/N*, dove al posto di *N* sarà presente un numero incrementale che identifica la decisione nella giornata, preceduto dalla data dell'incontro;
- una descrizione della scelta fatta.

Ogni verbale dovrà essere denominato secondo il seguente formato

Verbale_X_YYYY-MM-DD

dove per *X* si intende il tipo di verbale, che sarà:

- **I**: se l'incontro è stato interno al gruppo;

- **E**: se l'incontro si è svolto con la partecipazione dei proponenti.

In conformità allo standard ISO 8601, le date devono essere scritte secondo il formato: *YYYY-MM-DD*, dove:

- **YYYY**: rappresenta l'anno con 4 cifre;
- **MM**: rappresenta il mese con 2 cifre;
- **DD**: rappresenta il giorno con 2 cifre.

Questo formato verrà utilizzato in tutti i documenti, ad eccezione dei diagrammi creati con *MS Project*, le cui date saranno scritte nel modo seguente: *YY-MM-DD*. Gli orari invece, sempre secondo lo stesso standard, devono essere scritti nel formato *HH:MM*, dove:

- **HH**: rappresenta l'ora con 2 cifre in formato 24 ore;
- **MM**: rappresenta i minuti con 2 cifre.

3.1.1.4 Norme tipografiche

3.1.1.4.1 Nomi dei file

I nomi dei file (estensione esclusa) e delle cartelle utilizzano la convenzione *CamelCase* e alcune regole aggiuntive elencate di seguito:

- i nomi dei file e delle cartelle avranno sempre come lettera iniziale una lettera maiuscola;
- i nomi composti verranno scritti unendo le parole tra loro e lasciando le loro iniziali maiuscole;
- le preposizioni non si omettono;
- i nomi dei documenti includeranno la versione e seguiranno il formato: *NomeDelFile_ v.X.Y.Z-INC-WW*, dove:
  - X.Y.Z indicano la versione attuale del documento;
  - INC-WW indicano l'incremento numero WW raggiunto.

Un esempio corretto è: *NormeDiProgetto_ v.1.0.0-INC-05*. Farà eccezzione la nomenclatura dei documenti al momento della consegna per la Revisione di Accettazione, che avrà il seguente formato: *NomeDelFile_ v.X.Y.Z-R*, dove R stà per "rilascio".

3.1.1.4.2 Separatore migliaia

Per facilitare la lettura dei numeri con piú di tre cifre intere, si è deciso di fare riferimento allo standard internazionale ISO 31-0, che raccomanda l'uso dello spazio per raggruppare le cifre. Un esempio corretto è: *12 320*, che indica il numero *dodicimilatrecentoventi*.

3.1.1.4.3 Elenchi puntati

Gli elenchi puntati presenti nei documenti rispetteranno il seguente formato:

- la frase che precede il primo elemento dell'elenco deve terminare con il carattere ":";
- se un elemento dell'elenco a sua volta precede un sottoelenco, dovrà terminare con il carattere ":";

- ogni elemento dell'elenco deve cominciare con la lettera minuscola, tranne quando è un nome proprio o una parola da glossario;
- ogni elemento dell'elenco, se è costituito da un termine o da un solo periodo, deve terminare con il carattere ";" altrimenti se è costituito da più periodi, divisi tra loro dal carattere ".", l'elemento terminerà con il carattere ".";
- l'ultimo elemento dell'elenco terminerà in ogni caso con il carattere ".".

3.1.1.4.4 Glossario

I termini del *Glossario* presenti nel testo dei documenti sono marcati con una G maiuscola a pedice alla loro prima occorrenza e scritti in corsivo. Se nel *Glossario* un termine presenta una descrizione che utilizza termini da glossario, è necessario utilizzare il comando LaTeX appositamente creato che formatta la parola in corsivo e con una G maiuscola a pedice. Le parole da glossario presenti nei titoli o nelle didascalie di immagini e tabelle non vengono segnalate. Solo la prima occorrenza della parola da glossario presente nel corpo del documento viene segnalata. Se il termine da inserire nel *Glossario* è composto da più parole, tutte le parole comprese le preposizioni saranno scritte in corsivo. Un esempio corretto è: *ciclo di vita*$_{\mathrm{G}}$.

3.1.1.4.5 Sigle

In tutti i documenti verranno utilizzate le seguenti sigle per abbreviare i nomi di documenti da citare nel testo, con l'esclusione dei titoli di sezione o paragrafo:

- **AdR**: indica l'*Analisi dei requisiti*;
- **NdP**: indica le *Norme di progetto*;
- **PdP**: indica il *Piano di progetto*;
- **PdQ**: indica il *Piano di qualifica*;
- **SdF**: indica lo *Studio di fattibilità*.

Le seguenti sigle invece sono utilizzate per abbreviare i nomi delle varie revisioni, verranno utilizzate nei nomi delle cartelle e nei documenti:

- **RR**: indica la *Revisione dei Requisiti*;
- **RP**: indica la *Revisione della Progettazione*;
- **RQ**: indica la *Revisione di Qualifica*;
- **RA**: indica la *Revisione di Accettazione*.

3.1.1.4.6 Iniziali maiuscole

In tutti i documenti saranno utilizzate le iniziali maiuscole solo nei seguenti casi:

- l'iniziale della prima parola del titolo;
- l'iniziale della prima parola del corpo del paragrafo;
- l'iniziale dei nomi propri.

3.1.1.5 Componenti grafiche

3.1.1.5.1 Tabelle

Di seguito sono riportate le norme da utilizzare per lo stile delle tabelle che compaiono nei diversi documenti.

- l'*header*$_G$ della tabella ha lo sfondo blu e il colore del testo bianco;
- la prima parola di ogni cella ha l'iniziale maiuscola;
- lo sfondo delle righe di contenuto della tabella è di colore bianco alternato a grigio;
- la tabella è centrata al documento.

Di seguito troviamo un esempio corretto.

Parametro 1	Parametro 2
Contenuto 1	Questa è la prima riga di contenuto
Contenuto 2	Questa è la seconda riga di contenuto

Tabella 2: Esempio corretto di tabella.

3.1.1.5.2 Immagini

Le immagini devono essere messe in una cartella, dentro alla cartella del documento a cui appartengono, denominata *img*. Il formato usato per nominare i file immagine è il seguente: *S_ss_#_nome.estensione*, dove:

- **S** è il numero di sezione in cui verrà inserita l'immagine;
- **ss** è il numero di sottosezione in cui verrà inserita l'immagine;
- # è il numero progressivo dell'immagine all'interno della sottosezione;
- **nome** è un nome significativo per l'immagine;
- **estensione** è il formato dell'immagine.

Tutte le immagini saranno centrate e seguite da una breve descrizione.

3.1.1.5.3 Diagrammi UML

I diagrammi UML presenti nei vari documenti andranno inseriti in formato immagine *png*.

3.1.2 Strumenti

3.1.2.1 LaTeX

3.1.2.1.1 Descrizione generale

LaTeX è un linguaggio di markup per la preparazione di testi, basato sul programma di composizione tipografica TeX.

3.1.2.1.2 Applicazione al progetto

Abbiamo utilizzato questo strumento per realizzare tutta la documentazione relativa al progetto.

3.2 Gestione della configurazione

La gestione della configurazione è un processo dello sviluppo software. Esso prevede il controllo e la gestione formale di oggetti documentali e non. Tale gestione viene basata sui database dove vengono controllati questi oggetti, comunemente denominati *configuration item*$_{\text{G}}$.

3.2.1 Attivitá

3.2.1.1 Versionamento

3.2.1.1.1 Codifica della versione del documento

Ogni documento può essere ricostruito attraverso le sue versioni, di cui si tiene traccia nel registro delle modifiche, dove ogni riga corrisponde ad una versione. Ogni versione del documento viene codificata con il formato *X.Y.Z-W*, dove:

- **X** rappresenta una versione completa del documento, verificata e rilasciata dal responsabile:
  - parte da 0;
  - viene incrementata dal verificatore al termine di una verifica generale positiva. Tali verifiche vengono eseguite prima di un rilascio.
- **Y** rappresenta una versione la cui stesura è stata completata e verificata:
  - parte da 0;
  - viene incrementato dal verificatore al termine di una verifica positiva, successivamente ad una modifica o ad una aggiunta al documento;
  - all'incremento di X riparte da 0.
- **Z** rappresenta una modifica per correzione ad una versione Z precedente:
  - parte da 0;
  - viene incrementato dal verificatore dopo una verifica positiva, successivamente ad una correzione dovuta ad una verifica negativa;
  - all'incremento di X o Y riparte da 0.
- **W** rappresenta un riferimento alla versione del software.

3.2.1.1.2 Codifica della versione del PoC

Anche il Proof Of Concept può essere ricostruito attraverso le sue versioni, di cui si tiene traccia attraverso i diff di Github. Ogni versione del documento viene codificata con il formato *PoC_v_X*, dove X sta per la versione del PoC, verificata ed approvata dal responsabile. Ogni aumento di versione coincide con il cambio di incremento del modello di sviluppo, sempre a patto che tale incremento preveda una modifica al PoC.

3.2.1.2 Gestione del repository

3.2.1.2.1 Strutturazione

Ogni membro del gruppo ha in locale una copia del repository, clonata dal principale ospitato in remoto. L'organizzazione delle cartelle è la seguente:

- **RR**: cartella che contiene i documenti richiesti alla revisione dei requisiti, strutturata in questo modo:
  - **Documenti interni**: cartella che contiene i *Verbali interni*, le *NdP* e lo *SdF*;
  - **Documenti esterni**: cartella che contiene i *Verbali esterni*, l'*AdR*, il *PdP*, il *PdQ* e il *Glossario*;
  - **Lettera di presentazione**.
- **TemplateDocumento**: cartella che contiene i file necessari a creare un nuovo documento e un file di comandi personalizzati per facilitarne la stesura.

3.2.1.2.2 Utilizzo

Ogni membro del gruppo seguirà questa procedura:

- scelto il branch in cui lavorare sulla base del documento (o parte software) a lui assegnato, eseguire un $pull_{\text{G}}$, in modo da aggiornare la sua copia locale con le ultime modifiche;
- svolto il lavoro assegnato, confermare le modifiche apportate nell'area di $staging_{\text{G}}$;
- eseguire il $commit_{\text{G}}$, lasciando un breve commento che ne descriva il contenuto;
- effettuare il $push_{\text{G}}$ delle modifiche, in modo da aggiornare il repository in remoto.

3.2.1.2.3 Gestione dei cambiamenti

Per problematiche di maggiore entità come cambio strumenti di sviluppo o norme si seguirá tale prassi:

- verranno contattati tutti i membri del gruppo tramite il relativo canale *Slack*;
- si aprirà un'issue su *Glo Boards* con una descrizione il più possibile precisa;
- il gruppo discuterà la problematica, valuterà eventuali soluzioni e una volta scelta la soluzione più appropriata, essa verrà attuata apportando le modifiche necessarie;
- verrà creato un verbale interno in modo da tenere traccia dei cambiamenti;
- ogni *redattore* provvederà ad aggiornare i documenti modificando le parti interessate;
- a questo punto sarà possibile chiudere l'issue.

Se invece la problematica è relativa a requisiti, funzionalità, progettazione architetturale o tecnologie utilizzate invece:

- verranno contattati tutti i membri del gruppo tramite il relativo canale *Slack*;
- si aprirà un'issue su *Glo Boards* con una descrizione il più possibile precisa;
- il gruppo discuterà la problematica, valuterà eventuali soluzioni e una volta scelta la soluzione più appropriata, essa sarà sottoposta al proponente;
- se il proponente dovesse accettare la soluzione proposta, questa verrà attuata apportando le modifiche necessarie altrimenti si troverà una soluzione che possa soddisfare il proponente e allo stesso tempo venire incontro ai problemi del gruppo;

- verrà creato un verbale esterno in modo da tenere traccia dei cambiamenti;
- ogni *redattore* provvederà ad aggiornare i documenti modificando le parti interessate;
- a questo punto sarà possibile chiudere l'issue.

3.2.2 Strumenti

3.2.2.1 Git

3.2.2.1.1 Descrizione generale
Git è un software di controllo versione distribuito utilizzabile da interfaccia a riga di comando, creato da Linus Torvalds nel 2005. Git nacque per essere un semplice strumento per facilitare lo sviluppo del kernel Linux ed è diventato uno degli strumenti di controllo versione più diffusi.

3.2.2.1.2 Applicazione al progetto
È utilizzato dal nostro gruppo per gestire e mantenere aggiornato il repository locale, e anche per connettersi al repository remoto su Github

3.2.2.2 Github

3.2.2.2.1 Descrizione generale
GitHub è un servizio di hosting per progetti software. Il nome "GitHub" deriva dal fatto che GitHub è una implementazione dello strumento di controllo versione distribuito Git.

3.2.2.2.2 Applicazione al progetto
Noi lo usiamo come locazione per il repository remoto condiviso dal gruppo e per la condivisione del prodotto e dei relativi documenti con il fornitore

3.2.2.3 GitKraken

3.2.2.3.1 Descrizione generale
GitKraken é una GUI per Git, che mette a disposizione anche altri servizi per il lavoro collaborativo come le Glo Board.

3.2.2.3.2 Applicazione al progetto
Viene usato dal gruppo per una più semplice gestione del repository Git.

3.2.2.4 Google Drive

3.2.2.4.1 Descrizione generale
Google Drive è un servizio web, in ambiente cloud computing, di memorizzazione e sincronizzazione online introdotto da Google il 24 aprile 2012. Basato su software open source[1], comprende il file hosting, il file sharing e la modifica collaborativa di documenti.

3.2.2.4.2 Applicazione al progetto

Il gruppo lo usa per la condivisione di materiale non soggetto a versionamento, come per esempio ricerche e documenti relativi allo studio personale ma anche per la gestione degli ordini del giorno degli incontri.

3.2.2.5 Microsoft OneDrive

3.2.2.5.1 Descrizione generale

Microsoft OneDrive è un servizio di cloud storage e backup offerto da Microsoft, accessibile tramite browser e app desktop e smartphone.

3.2.2.5.2 Applicazione al progetto

Il gruppo lo usa per la gestione delle slide, in quanto questo strumento permette l'utilizzo collaborativo delle applicazioni del pacchetto Office attraverso l'applicativo web Office 365.

3.3 Gestione della qualità

Lo scopo è fornire un'adeguata garanzia che i prodotti e i processi software rispettino gli obiettivi di qualità, aderiscano ai piani stabiliti nel ciclo di vita del software e che i bisogni del proponente siano soddisfatti. Per la trattazione approfondita della gestione della qualità si fa riferimento al *PdQ*, nel quale vengono descritte le modalità di perseguimento della qualità. Nello specifico troviamo:

- gli standard utilizzati;
- i processi di interesse negli standard;
- gli attributi del software più significativi per il progetto.

Per ogni processo vengono descritti:

- gli obiettivi;
- la strategia;
- le metriche.

Le caratteristiche che rendono un prodotto di qualità sono descritte attraverso:

- gli obiettivi;
- le metriche.

3.3.1 Attività

3.3.1.1 Pianificazione

Questa attività prevede di porre degli obiettivi di qualità e disporre di conseguenza le persone e le risorse nel migliore dei modi.

3.3.1.2 Valutazione

Dopo la pianificazione, devono venire confrontati gli obiettivi previsti con i risultati ottenuti.

3.3.1.3 Reazione

Dopo la valutazione bisogna adattare le future strategie, adottando o meno correzioni in base ai risultati ottenuti.

3.3.2 Strumenti

Gli strumenti predefiniti per la qualità sono:

- gli standard (*ISO/IEC/IEEE-12207:1995* per i processi e *ISO/IEC 9126* per i prodotti);
- le metriche di seguito descritte.

3.3.2.1 Metriche

Di seguito sono descritte le metriche utilizzate per ogni processo. Le soglie e i range di valori accettabili vengono descritti in dettaglio nel *Piano di qualifica*.

3.3.2.1.1 Analisi dei requisiti

Per verificare la percentuale di requisiti obbligatori soddisfatti si usa l'indice *PROS* che viene calcolato con questa formula: $PROS = \frac{requisitiobbligatorisoddisfatti}{requisitiobbligatoritotali}$.

3.3.2.1.2 Progettazione

Per quanto riguarda la progettazione architetturale e di dettaglio, gli indici utilizzati sono:

- **Structural Fan-in** (SFIN): indica quante componenti utilizzano un dato modulo. Un valore alto indica un alto riuso della componente. Si misura attraverso il conteggio delle componenti;
- **Structural Fan-out** (SFOUT): indica quante componenti vengono utilizzate dalla componente in esame. Un alto valore indica un alto accoppiamento della componente. Si misura attraverso il conteggio delle componenti;
- **CBO**: accoppiamento tra le classi di oggetti. Una classe è accoppiata ad una seconda se usa metodi o variabili definiti nella seconda. Si misura contando il numero di metodi o di variabili definiti in una classe e utilizzati in un'altra.

3.3.2.1.3 Pianificazione

Per la misura della qualità nell'attività di pianificazione faremo affidamento alle seguenti metriche:

- **Budget at Completion** (BAC): budget totale allocato per il progetto;
- **Actual Cost** (AC): il costo sostenuto fino al momento del calcolo;
- **Earned Value** (EV): metrica di utilità per il calcolo di SV e CV (spiegate successivamente). Si tratta del valore delle attività realizzate fino al momento del calcolo;
- **Planned Value**(PV): metrica di utilità per il calcolo di SV e CV (spiegate successivamente). Indica il costo pianificato per la realizzazione delle attività di progetto fino al momento del calcolo. Si misura con BAC_{G} * % di lavoro pianificato;
- **Schedule Variance** (SV): esprime lo stato di anticipo o ritardo nello svolgimento rispetto alla pianificazione delle attività di progetto. Si calcola come $SV = EV - PV$;
- **Cost Variance**(CV): differenza tra il costo del lavoro effettivamente completato e quello pianificato. Si calcola come $CV = EV - AC$.

3.3.2.1.4 Verifica codice

Le metriche utilizzate per la verifica del codice sono:

- **Statement Coverage** (SC): indica il numero di comandi percorsi cumulativamente dai test durante la loro esecuzione, ciascuno con esito corretto. Per comandi totali si intende tutti quelli appartenenti all'unità in fase di test. Si misura con la formula: $SC = \frac{numerodicomandipercorsi}{numerodicomandipercorsi}$;
- **Branch Coverage** (BC): indica il numero di rami di flusso di controllo percorsi complessivamente dai test durante la loro esecuzione, ciascuno con esito corretto. Per rami totali si intendono tutti quelli appartenenti all'unità in fase di test. Si misura con la formula: $SC = \frac{ramipercorsi}{ramitotali}$;

- **Modified Condition/Decision Coverage** (MCDC): indica il numero di decisioni e condizioni testate complessivamente dai test durante la loro esecuzione, ciascuno con esito corretto. Una condizione è un'espressione booleana semplice, non contenente operatori booleani. Una decisione è un'espressione composta, contenente condizioni combinate da operatori booleani. Si misura con la formula: $SC = \frac{decisionipercorse}{decisionitotali}$.

Utilizzeremo ,inoltre, le metriche fornite dallo strumento Sonarqube che sono:

- il numero di bug: è un errore nella logica del codice;
- il numero di vulnerabilità: sono delle violazioni alle regole di sicurezza imposte da Sonarqube che devono essere rispettate, con la presenza di una vulnerabilità potrebbe venir violata la sicurezza di tutta l'applicazione;
- il numero di Security Hotspot: segnala che delle righe di codice potrebbero produrre una falla nella sicurezza del codice, ma la sicurezza di tutta l'applicazione potrebbe non venir violata. È a discrezione dello sviluppatore correggere tale errore o meno;
- Code Coverage che permette di individuare la percentuale di righe di codice eseguite rispetto al totale, durante la loro esecuzione. Un valore alto in code coverage garantisce che ci sia una bassa possibilità di errore.
- la percentuale di codice ripetuto: indica quante volte un'insieme di righe di codice è stato ripetuto, un valore alto potrebbe portare a diminuire la manutenibilità del codice sorgente.

3.3.2.1.5 Documentazione

Le metriche utilizzate per la misura della qualità nella documentazione sono:

- **indice di Gulpease**, indice che determina la leggibilità di un testo. Viene calcolato attraverso la seguente formula: $I_G = 89 + \frac{(300 * numerodifrasi - 10 * numerodilettere)}{numerodiparole}$;
- numero di errori grammaticali ed ortografici.

3.3.2.1.6 Gestione della qualità

Per misurare la gestione della qualità verrà usato l'indice di PMS, che indica la percentuale di metriche soddisfatte e valuta quante metriche raggiungono soglie accettabili sul numero totale delle metriche calcolate. Una bassa percentuale di soddisfazione può indicare poca qualità, metriche inadeguate o mancata correttezza nel calcolo. Si misura con la formula: $PMS = \frac{numerodimetrichesoddisfatte}{numerodimetrichetotali}$.

3.4 Verifica

Il processo di verifica ha lo scopo di individuare gli errori introdotti nel prodotto durante la fase di redazione o sviluppo in modo da permettere la realizzazione di prodotti corretti, coesi e completi. Sono soggetti a verifica sia il software che i documenti. Per ottenere questo risultato, il processo si affida alle attività di analisi e di test.

3.4.1 Attività

3.4.1.1 Analisi

Il processo di analisi si divide in due metodi: analisi statica e analisi dinamica.

3.4.1.1.1 Analisi statica

L'analisi statica studia la documentazione e il codice, accertandone la conformità alle regole, l'assenza di difetti e la presenza delle proprietà desiderate. Non richiede l'esecuzione del codice, quindi è attuabile già nelle prime fasi. Esistono due tecniche per effettuare analisi statica, che sono:

- **Walkthrough**: tecnica che consiste in una lettura a largo spettro in cerca di anomalie. Verrà utilizzata nella fase iniziale del progetto non avendo ancora una profonda conoscenza delle NdP e del PdQ;
- **Inspection**: tecnica che consiste in una lettura mirata e strutturata, utilizzando le *liste di controllo*$_{\text{G}}$ e le misurazioni effettuate in precedenza, in modo da rendere sempre più efficace la tecnica con l'esperienza.

Segue una prima lista di controllo per la verifica dei documenti. Se ne prevede l'ampliamento.

Oggetto	Descrizione
Formato data	La data deve rispettare il formato *YYYY-MM-DD*
Sintassi	Frase semplificabile o errore nella sua struttura
Elenchi puntati	Uso della punteggiatura non conforme alle norme
Glossario	Termine già presente segnalato da inserire nel Glossario
Indice	Deve essere presente e corretto l'indice del documento
Numero pagina	Deve essere presente e corretto il numero totale di pagine
Elenco tabelle/immagini	Devono essere corretti l'elenco delle immagini e delle tabelle del documento
Maiuscola	Uso dell'iniziale maiuscola non conforme alle norme

Tabella 3: Lista di controllo per la verifica

3.4.1.1.2 Analisi dinamica

L'analisi dinamica viene effettuata tramite dei test che verificano il corretto funzionamento del

prodotto software e rilevano eventuali errori. Richiede l'esecuzione del prodotto software, quindi è attuabile solamente dopo l'attività di codifica.

3.4.1.2 Test

3.4.1.2.1 Test di unità
I test di unità si eseguono su singole unità di software e permettono di verificarne la correttezza. Le singole unità possono essere testate con l'ausilio di *driver*$_{\text{G}}$ e *stub*$_{\text{G}}$ che rispettivamente simulano un'unità chiamante e un'unità chiamata.

3.4.1.2.2 Test di integrazione
I test di integrazione sono l'estensione dei test di unità. Si concentrano sulle interfacce tra i componenti. Al superamento dei test si formano agglomerati sempre più grandi di unità, fino ad arrivare alla dimensione totale del sistema.

3.4.1.2.3 Test di sistema
Il test di sistema viene fatto alla validazione del prodotto finale, una volta integrati tutti i suoi componenti. Ne verifica il comportamento dinamico rispetto ai requisiti software.

3.4.1.2.4 Test di regressione
Il test di regressione viene effettuato a seguito di una modifica per correzione o estensione di parti, in modo da accertare che non comportino errori nel sistema.

3.4.1.2.5 Test di accettazione
Il test di accettazione si effettua con la presenza del committente e serve ad accertare il soddisfacimento dei requisiti utente. Il suo superamento garantisce che il software è pronto per essere rilasciato.

3.4.2 Strumenti

3.4.2.1 Compilazione automatica

3.4.2.1.1 Descrizione generale
Per evitare l' occorrenza di documenti incompleti (ad esempio mancanza dell'indice o del numero di pagina) causato da una compilazione solo parziale di alcuni documenti, abbiamo deciso di ricorrere alla compilazione automatica.

3.4.2.1.2 Applicazione al progetto
Abbiamo automatizzato il processo di compilazione sfruttando una *Github Action*$_{\text{G}}$, che ad ogni push ricompila i documenti modificati.

3.4.2.2 Verifica ortografica

3.4.2.2.1 Descrizione generale
Per migliorare i documenti e velocizzare il lavoro dei verificatori abbiamo ritenuto necessario introdurre dei sistemi di verifica ortografica.

3.4.2.2.2 Applicazione al progetto
La verifica ortografica viene effettuata nei plugin offerti dagli editor di studio utilizzati dal gruppo: *TexStudio*, *JEdit* e *Overleaf*.

3.4.2.3 Verifica codice
Si fa affidamento allo strumento offerto da W3C per la validazione del codice di markup *HTML5* e *CSS3*, disponibile al link `https://validator.w3.org/`. Verranno adottati altri strumenti successivamente.

3.4.2.4 Sonarqube

3.4.2.4.1 Descrizione generale
È uno strumento per monitoraggio dei risultati dell'analisi statica del codice, in cui non mantiene solo il valore attuale, ma l'andamento dei valori utilizzando dei grafici.

3.4.2.4.2 Applicazione al progetto
Abbiamo utilizzato lo strumento per la verifica del codice attraverso la misura dei seguenti valori:

- il numero di bug;
- il numero di vulnerabilità;
- il numero di Security Hotspot;
- la percentuale di codice coperto dai test;
- la percentuale di codice ripetuto.

3.4.2.5 Indice di Gulpease

3.4.2.5.1 Descrizione generale
L'Indice di Gulpease è un indice di leggibilità di un testo tarato sulla lingua italiana. Rispetto ad altri ha il vantaggio di utilizzare la lunghezza delle parole in lettere anziché in sillabe, semplificandone il calcolo automatico.

3.4.2.5.2 Applicazione al progetto
È stato creato uno script in *Python* che, dato in input un file in formato *pdf*, restituisce l'*indice di Gulpease*$_{\mathrm{G}}$.

3.5 Validazione

Lo scopo della validazione è accertare l'efficacia del prodotto finale. L'esito positivo del processo garantisce che il software rispetti tutti i requisiti e soddisfi i bisogni del committente.

3.5.1 Attività

3.5.1.1 Identificazione
Identificare gli oggetti da validare.

3.5.1.2 Scelta della strategia
Orientandosi verso un riutilizzo delle procedure di verifica.

3.5.1.3 Applicazione
Tenendo traccia dei risultati ottenuti, bisogna attuare la strategia scelta.

3.5.1.4 Valutazione
Bisogna valutare i risultati ottenuti in confronto con le aspettative iniziali.

4 Processi organizzativi

4.1 Gestione organizzativa

La gestione organizzativa è un processo tramite il quale si stabiliscono le norme relative all'organizzazione del gruppo di lavoro, dalla suddivisione delle risorse umane in ruoli, alle norme sugli incontri fisici e/o telematici

4.1.1 Attività

4.1.1.1 Ruoli di progetto

Ogni membro del gruppo ricoprirà, a rotazione, i ruoli di progetto di seguito descritti. Nel *PdP* verranno assegnate e organizzate le attività relative ai vari ruoli.

4.1.1.1.1 Responsabile di progetto

Il responsabile di progetto ha il compito di gestire le risorse umane, i rischi, la pianificazione, il controllo, il coordinamento e le relazioni esterne. Inoltre, ha il compito di fare da intermediario tra il gruppo e le persone esterne; competono a lui quindi le comunicazioni con committente e proponente.

4.1.1.1.2 Amministratore di progetto

L'amministratore ha il compito di supportare e controllare l'ambiente di lavoro. Si occupa quindi di gestire il controllo della configurazione del prodotto, del versionamento e della documentazione.

4.1.1.1.3 Analista

L'analista si occupa dell'attività di analisi del problema, definendone i requisiti espliciti, impliciti e del dominio applicativo. Ha il compito di redigere lo *SdF* e l'*AdR*. Questa figura non sarà sempre presente durante il progetto.

4.1.1.1.4 Progettista

Il progettista si occupa degli aspetti tecnologici e tecnici del progetto. Ha il compito di effettuare lo SdF del prodotto, sviluppare un'architettura efficiente ed efficace sulla base del lavoro dell'analista e di redigere la specifica tecnica del progetto.

4.1.1.1.5 Verificatore

Il verificatore ha il compito di gestire le attività di verifica e validazione, durante l'intero ciclo di vita del progetto. Si occupa di verificare la correttezza della documentazione e del codice affidandosi alle norme definite nelle *NdP*, segnalando eventuali errori alle persone che hanno in carico l'oggetto in discussione.

4.1.1.1.6 Programmatore

Il programmatore ha il compito di produrre il codice necessario allo sviluppo del prodotto e alle componenti di supporto utili alla verifica e validazione. Deve attenersi alle specifiche fornite dal responsabile e ha il compito di produrre codice che sia facilmente mantenibile.

4.1.1.2 Gestione degli incontri

4.1.1.2.1 Incontri interni
Gli incontri interni al gruppo sono organizzati dal responsabile, che si occupa di scegliere in accordo con gli altri membri, data e orario dell'incontro sulla base della disponibilità di quest'ultimi, resa pubblica in un documento condiviso in *Google Drive*, aggiornato settimanalmente.

4.1.1.2.2 Incontri esterni
Gli incontri esterni sono, allo stesso modo, organizzati dal responsabile. Quando si presenta la necessità di un incontro di questo tipo, sia per volontà di uno o piú membri del gruppo, del proponente o del committente, il responsabile si occupa di comunicare una data e un orario deciso in accordo con le parti, comunicandolo attraverso gli strumenti di comunicazione.

4.1.1.3 Distanziamento sociale
A causa del distanziamento sociale obbligatorio attualmente in vigore gli incontri, sia interni al gruppo che esterni con i committenti, andranno svolti necessariamente per via telematica, tramite Hangouts.

4.1.1.4 Formazione

4.1.1.4.1 Conoscenze pregresse
Di seguito sono elencate le varie conoscenze, affini al progetto, dichiarate dai membri del gruppo:

- **Stefano Cavaliere**: C++, Java, HTML, CSS, Bootstrap, SQL;
- **Marco Ferrati**: C++, Swift, iOS, Python, Java, Git, SQL;
- **Lorenzo Taschin**: C++, Java, Python, Javascript, HTML, SQL;
- **Riccardo Tassetto**: C++, C#, Java, CSS, HTML, PHP, Bootstrap, SQL, Git;
- **Marcel Junior Wandji**: C++, CSS, Git, HTML, Java, Javascript, SQL, PHP;
- **Xiaowei Wen**: Android (in Java), Java, HTML, Python, C++, Git, MS Excel, SQL, MS Project;
- **Giuseppe Zatta**: C++, Java, HTML, CSS, PHP, MS Office, Git, SQL;
- **Claudia Zattara**: C++, Java, HTML, CSS, Javascript, PHP, SQL, Git, MS Office.

È prevista una formazione autonoma da parte di tutti i membri del gruppo, in modo da sfruttare al meglio le tecnologie che verranno utilizzate durante il progetto. Per facilitare la formazione i membri del gruppo sono tenuti a condividere, oltre alle loro conoscenze, eventuali materiali utili caricandoli nell'apposita cartella condivisa di Google Drive.

4.1.1.4.2 Processo di formazione del personale
Essendo il gruppo VarTmp7 composto da membri con conoscenze pregresse disomogenee, per aumentare l'efficienza e l'efficacia della formazione è necessario seguire la seguente procedura: quando il gruppo incontra una tecnologia sconosciuta, due membri vengono incaricati dello studio e approfondimento di tale tecnologia. Successivamente devono esporre quanto acquisito dal loro studio agli altri membri del gruppo nella maniera più adattata (spiegazione verbale, realizzazione di un video esemplificativo, condivisione di materiali informativi).

4.1.2 Strumenti

4.1.2.1 Telegram

4.1.2.1.1 Descrizione generale
Telegram è un servizio di messaggistica istantanea e broadcasting basato su cloud ed erogato senza fini di lucro dalla società Telegram LLC.

4.1.2.1.2 Applicazione al progetto
Questo strumento è utilizzato dal gruppo per le comunicazioni organizzative interne al gruppo.

4.1.2.2 Slack

4.1.2.2.1 Descrizione generale
Slack è un software che rientra nella categoria degli strumenti di collaborazione aziendale utilizzato per inviare messaggi in modo istantaneo ai membri del team.

4.1.2.2.2 Applicazione al progetto
Il gruppo utilizza un *workspace*$_{\text{G}}$ di Slack per comunicare internamente. Sono presenti diversi canali, uno per ogni documento, in aggiunta di un canale generale per informazioni utili a tutto il gruppo e ad un canale riservato alle comunicazioni con un referente dei proponenti, *Imola Informatica*

4.1.2.3 Hangouts

4.1.2.3.1 Descrizione generale
Hangouts è un software di messaggistica istantanea e di VoIP sviluppato da Google. è disponibile per le piattaforme mobili Android e iOS e come estensione per il browser web Google Chrome.

4.1.2.3.2 Applicazione al progetto
Il gruppo fa uso di questo strumento per le riunioni virtuali in canali vocali.

4.1.2.4 Glo Boards

4.1.2.4.1 Descrizione generale
Come accennato in precedenza, *Glo Boards* è uno strumento fornito da *GitKraken* per la gestione delle issue. Permette a tutti i membri del gruppo di creare nuove issue, assegnandole agli interessati e notificando le relative scadenze. Lo strumento consiste in una lavagna virtuale dove sono visibili i vari task creati, associati ad una data di scadenza e assegnati ad uno o più membri. Anche gli stadi in cui transitano le issue sono configurabili.

4.1.2.4.2 Applicazione al progetto

La scelta è ricaduta su questo strumento per l'intuitività dell'interfaccia e la facilità di utilizzo. Gli stati in cui si possono trovare i task, nel nostro progetto sono:

- **To do**: il lavoro deve ancora essere preso in carico;
- **In progress**: i membri assegnati stanno lavorando a questo compito;
- **Verify**: il lavoro svolto è terminato ed è sotto verifica;
- **Done**: il task è stato completato superando la verifica.

A ISO/IEC 9126

ISO/IEC 9126$_{G}$ è uno standard internazionale per valutare la qualità del software.
Questo standard è diviso in quattro parti che vengono riportate di seguito.

A.1 Metriche per la qualità interna

Metriche che si applicano al software non eseguibile durante le fasi di progettazione e codifica. Permettono di individuare eventuali problemi che potrebbero influire sulla qualità finale del prodotto prima che venga realizzato un eseguibile. Grazie alle misure effettuate tramite le metriche interne è possibile prevedere il livello di qualità esterna e di qualità in uso del prodotto finale, poichè entrambe vengono influenzate dalla qualità interna.
Viene rilevata tramite analisi statica. Idealmente la qualità interna determina la qualità esterna.

A.2 Metriche per la qualità esterna

Metriche applicabili al software in esecuzione che ne misurano il comportamento attraverso dei test, in funzione degli obiettivi stabiliti.
Viene rilevata tramite analisi dinamica. Idealmente la qualità esterna determina la qualità in uso.

A.3 Metriche per la qualità in uso

Metriche applicabili solo al prodotto finito ed in uso in condizioni reali.
La qualità in uso viene raggiunta solo se è stato raggiunto il livello di qualità interna e di qualità esterna.

A.4 Modello della qualità del software

Il modello di qualità del software suddivide la qualità in sei caratteristiche generali e varie sotto caratteristiche, misurabili attraverso delle metriche, utilizzate per fornire una scala ed un metodo per la misurazione. Di seguito sono riportate queste caratteristiche.

A.4.1 Funzionalità

Capacità del software di soddisfare i requisiti, descritti nell'*Analisi dei requisiti*, in un determinato contesto.
Nello specifico il software deve soddisfare le seguenti caratteristiche:

- **appropriatezza**: capacità di fornire funzioni appropriate per attività specifiche, che permettano di raggiungere gli obiettivi prefissati;
- **accuratezza**: capacità di fornire i risultati concordati o la precisione richiesta;
- **interoperabilità**: capacità di interagire ed operare con uno o più sistemi specificati;
- **conformità**: capacità di aderire a standard;
- **sicurezza**: capacità di proteggere informazioni e dati.

A.4.2 Affidabilità

Capacità del software di mantenere uno specifico livello di prestazioni quando usato in condizioni specificate.
Nello specifico il software deve soddisfare le seguenti caratteristiche:

- **maturità**: capacità di evitare il verificarsi di errori, malfunzionamenti o risultati non corretti;
- **tolleranza agli errori**: capacità di mantenere livelli prefissati di prestazioni anche in presenza di malfunzionamenti o usi scorretti del prodotto finale;
- **recuperabilità**: capacità di ripristinare un livello appropriato di prestazioni o di recupero di informazioni rilevanti a seguito di un malfunzionamento;
- **aderenza**: capacità di aderire a standard, regole e convenzioni che riguardano l'affidabilità.

A.4.3 Efficienza

Capacità del prodotto software di eseguire le proprie funzioni minimizzando il tempo necessario e sfruttando al meglio le risorse che necessita.
Nello specifico il software deve soddisfare le seguenti caratteristiche:

- **nel tempo**: capacità di fornire adeguati tempi di risposta, elaborazione e velocità di attraversamento in determinate condizioni;
- **nello spazio**: capacità di utilizzo di quantità e tipo di risorse in maniera adeguata;

A.4.4 Usabilità

Capacità del prodotto software di essere compreso, appreso, usato e accettato dall'utente, quando usato sotto determinate condizioni.
Nello specifico il software deve soddisfare le seguenti caratteristiche:

- **comprensibilità**: capacità di essere chiaro riguardo le proprie funzionalità ed il proprio utilizzo;
- **apprendibilità**: capacità di essere facilmente apprendibile dagli utenti;
- **operabilità**: capacità di permettere all'utente di eseguire i suoi scopi e controllarne l'uso;
- **attrattività**: capacità di essere piacevole all'utente che l'utilizza.

A.4.5 Manutenibilità

Capacità del software di essere modificato al fine di aggiungere correzioni, miglioramenti o adattamenti.
Nello specifico il software deve soddisfare le seguenti caratteristiche:

- **analizzabilità**: capacità di essere facilmente analizzato al fine di localizzare un errore;
- **modificabilità**: capacità di poter essere agevolmente modificato nel codice, nella progettazione o nella documentazione;
- **stabilità**: capacità di evitare effetti indesiderati a seguito di una modifica;
- **testabilità**: capacità di essere facilmente testato per validare le modifiche apportate.

A.4.6 Portabilità

Capacità del software di essere trasportato da un ambiente di lavoro ad un altro, sia esso hardware che software.
Nello specifico il software deve soddisfare le seguenti caratteristiche:

- **adattabilità**: capacità di essere facilmente adattato a differenti ambienti operativi, senza applicare modifiche;
- **installabilità**: capacità di poter essere installato in un determinato ambiente;
- **conformità**: capacità di coesistere con altre applicazioni e di condividere risorse;
- **sostituibilità**: capacità di essere utilizzato al posto di un altro software per svolgere gli stessi compiti, nello stesso ambiente.