



DeltaX

Progetto *Produlytics*

Norme di Progetto

Informazioni

Versione	2.0
Data Approvazione	2022-05-29
Responsabile	Giacomo Stevanato
Redattori	Leila Dardouri Alberto Lazari Christian Micheletti Riccardo Pavan Diego Stafa Giacomo Stevanato Daniele Trentin
Verificatori	Leila Dardouri Alberto Lazari Christian Micheletti Riccardo Pavan Diego Stafa Giacomo Stevanato Daniele Trentin
Stato	Approvato
Destinatari	DeltaX Prof. Riccardo Cardin Prof. Tullio Vardanega
Uso	Interno

Sommario

Raccolta delle norme che andranno a regolare lo sviluppo del progetto.

Ver.	Data	Nominativo	Ruolo	Descrizione
2.0	2022-05-29	Giacomo Stevanato	Responsabile	Approvazione del documento
1.14	2022-04-16	Giacomo Stevanato, Leila Dardouri	Amministratore, Verificatrice	Modifica norme struttura repository codifica
1.13	2022-04-11	Leila Dardouri, Alberto Lazari	Amministratrice, Verificatore	Modifica norme codifica
1.12	2022-03-28	Daniele Trentin, Diego Stafa	Amministratore, Verificatore	Regolamentazione redazione <i>Manuale Utente</i>
1.11	2022-03-26	Giacomo Stevanato, Leila Dardouri	Amministratore, Verificatrice	Aggiunte norme struttura repository codifica
1.10	2022-03-16	Christian Micheletti, Riccardo Pavan	Amministratore, Verificatore	Aggiunte norme struttura codifica
1.9	2022-03-15	Diego Stafa, Riccardo Pavan	Amministratore, Verificatore	Aggiunte metriche
1.8	2022-03-14	Diego Stafa, Daniele Trentin	Amministratore, Verificatore	Aggiunte norme codifica SQL, HTML e CSS
1.7	2022-03-12	Riccardo Pavan, Christian Micheletti	Amministratore, Verificatore	Aggiunte norme codifica TypeScript
1.6	2022-03-09	Leila Dardouri, Giacomo Stevanato	Amministratrice, Verificatore	Aggiunte norme codifica Java
1.5	2022-03-02	Christian Micheletti, Riccardo Pavan	Amministratore, Verificatore	Aggiunte norme codifica TypeScript
1.4	2022-02-28	Daniele Trentin, Leila Dardouri	Amministratore, Verificatrice	Aggiornamento regolamentazione sui processi
1.3	2022-02-25	Daniele Trentin, Christian Micheletti	Amministratore, Verificatore	Aggiornamento regolamentazione sui processi
1.2	2022-02-20	Alberto Lazari, Riccardo Pavan	Amministratore, Verificatore	Aggiornamento metriche
1.1	2022-02-19	Alberto Lazari, Christian Micheletti	Amministratore, Verificatore	Nuovo formato changelog
1.0.0	2022-02-04	Diego Stafa	Responsabile	Approvazione
0.4.0	2022-01-09	Leila Dardouri	Verificatrice	Verifica
0.3.1	2022-01-04	Giacomo Stevanato	Amministratore	Modifica metriche
0.3.0	2021-12-12	Riccardo Pavan	Verificatore	Verifica
0.2.2	2021-12-10	Leila Dardouri	Amministratrice	Stesura appendici A e B

Ver.	Data	Nominativo	Ruolo	Descrizione
0.2.1	2021-12-07	Daniele Trentin	Amministratore	Conclusione §4
0.2.0	2021-12-02	Diego Stafa	Verificatore	Verifica
0.1.2	2021-12-01	Daniele Trentin	Amministratore	Inizio stesura §4
0.1.1	2021-12-01	Daniele Trentin	Amministratore	Conclusione §2
0.1.0	2021-11-30	Leila Dardouri	Verificatrice	Verifica
0.0.4	2021-11-30	Giacomo Stevanato	Amministratore	Fine stesura §3
0.0.3	2021-11-24	Daniele Trentin	Amministratore	Inizio stesura §2
0.0.2	2021-11-23	Giacomo Stevanato	Amministratore	Inizio stesura §3
0.0.1	2021-11-23	Christian Micheletti	Responsabile	Stesura §1
0.0.0	2021-11-23	Christian Micheletti	Responsabile	Creazione del documento

Indice

1	Introduzione	10
1.1	Scopo del documento	10
1.2	Scopo del prodotto	10
1.3	Glossario	10
1.4	Riferimenti	10
1.4.1	Normativi	10
1.4.2	Informativi	11
2	Processi primari	13
2.1	Fornitura	13
2.1.1	Aspettative	13
2.1.2	Analisi dei rischi	13
2.1.3	Pianificazione	14
2.1.3.1	Metriche	14
2.1.4	Preventivazione e consuntivazione	14
2.1.5	Controllo della qualità	14
2.1.6	Coordinazione verifica e validazione	15
2.1.7	Definizione dei termini	15
2.1.8	Stesura Manuale Utente	15
2.1.9	Documenti	15
2.1.9.1	Piano di Progetto	15
2.1.9.2	Piano di Qualifica	16
2.1.9.3	Glossario	16
2.1.9.4	Manuale Utente	16
2.1.10	Strumenti	16
2.2	Sviluppo	16
2.2.1	Scopo	16
2.2.2	Aspettative	17
2.2.3	Descrizione	17
2.2.4	Analisi dei requisiti	17
2.2.4.1	Scopo	17
2.2.4.2	Descrizione	17
2.2.4.3	Analisi iniziale	17
2.2.4.4	Comunicazione con il proponente	18
2.2.4.5	Rielaborazione	18
2.2.4.6	Documento	18
2.2.4.7	Casi d'uso	18
2.2.4.8	Requisiti	19
2.2.4.9	UML	20
2.2.5	Progettazione	21
2.2.5.1	Scopo	21
2.2.5.2	Aspettative	21
2.2.5.3	Descrizione	21
2.2.5.4	Definizione dei test di accettazione e di sistema	21
2.2.5.5	Scelta delle tecnologie e <i>Proof of Concept</i>	21
2.2.5.6	Progettazione architetturale di sistema	22
2.2.5.7	Progettazione architetturale delle componenti	22
2.2.5.8	Progettazione di dettaglio	22
2.2.5.9	Documento	23

2.2.5.9.1	Interfaccia del front-end	23
2.2.5.9.2	Interfaccia del back-end e API rilevazioni	24
2.2.5.9.3	Requisiti	24
2.2.5.9.4	Definizione delle classi	24
2.2.5.9.5	Diagrammi delle classi	25
2.2.5.10	Metriche	25
2.2.6	Codifica	26
2.2.6.1	Scopo	26
2.2.6.2	Aspettative	26
2.2.6.3	Implementazione delle unità	26
2.2.6.4	Implementazione dei moduli	26
2.2.6.5	Stile di codifica - <i>Java_G</i>	26
2.2.6.5.1	Struttura dei file	26
2.2.6.5.2	Struttura delle classi	26
2.2.6.5.3	Pratiche di programmazione	27
2.2.6.5.4	Formattazione del codice	27
2.2.6.5.5	Convenzioni sui nomi	28
2.2.6.6	Stile di codifica - <i>TypeScript_G</i>	28
2.2.6.6.1	Struttura dei file	28
2.2.6.6.2	Struttura delle classi	28
2.2.6.6.3	Pratiche di programmazione	29
2.2.6.6.4	Formattazione del codice	29
2.2.6.6.5	Convenzioni sui nomi	29
2.2.6.7	Stile di codifica - <i>SQL_G</i>	30
2.2.6.7.1	Struttura dei file	30
2.2.6.7.2	Convenzioni sui nomi	30
2.2.6.8	Stile di codifica - <i>HTML_G</i> e <i>CSS_G</i>	30
2.2.6.9	Metriche	30
3	Processi di supporto	31
3.1	Documentazione	31
3.1.1	Descrizione	31
3.1.2	Scopo	31
3.1.3	Lista documenti	31
3.1.4	Ciclo di vita	31
3.1.5	Template	31
3.1.6	Struttura	32
3.1.6.1	Indentazione	32
3.1.6.2	Annotazioni	32
3.1.6.3	Prima pagina	32
3.1.6.4	Registro delle modifiche	33
3.1.6.5	Indice	33
3.1.6.6	Contenuto	34
3.1.6.7	Verbali	34
3.1.7	Convenzioni stilistiche	35
3.1.7.1	Nomi dei file	35
3.1.7.2	Stile del testo	35
3.1.7.3	Elenchi puntati	36
3.1.7.4	Tabelle	36
3.1.7.5	Formati delle date	36
3.1.8	Strumenti	36
3.1.9	Metriche	36

3.2	Gestione della configurazione	37
3.2.1	Descrizione	37
3.2.2	Scopo	37
3.2.3	Codice di versione	37
3.2.4	Tecnologie	37
3.2.4.1	Git	37
3.2.4.2	GitHub	37
3.2.5	Repository	37
3.2.5.1	Lista repository	37
3.2.5.2	Gerarchia dei file	38
3.2.6	Sincronizzazione	39
3.2.6.1	Branch	39
3.2.6.2	Rebase	39
3.2.6.3	Pull Request	39
3.3	Gestione della qualità	39
3.3.1	Descrizione	39
3.3.2	Scopo	39
3.3.3	PDCA	40
3.3.3.1	Plan	40
3.3.3.2	Do	40
3.3.3.3	Check	40
3.3.3.4	Act	40
3.3.4	Strumenti	41
3.3.5	Struttura delle metriche	41
3.3.6	Struttura degli obiettivi	41
3.3.7	Metriche	41
3.4	Verifica	42
3.4.1	Scopo	42
3.4.2	Aspettative	42
3.4.3	Descrizione	42
3.4.4	Analisi statica	42
3.4.4.1	Walkthrough	42
3.4.4.2	Ispezione	42
3.4.5	Analisi dinamica	43
3.4.5.1	Test di unità	43
3.4.5.2	Test di integrazione	43
3.4.5.3	Test di sistema	44
3.4.5.4	Test di regressione	44
3.4.5.5	Codici identificativi dei test	44
3.4.5.6	Stato dei test	45
3.5	Validazione	45
3.5.1	Scopo	45
3.5.2	Aspettative	45
3.5.3	Descrizione	45
3.5.4	Test di accettazione	45
3.5.5	Metriche	45

4	Processi organizzativi	46
4.1	Gestione organizzativa	46
4.1.1	Scopo	46
4.1.2	Aspettative	46
4.1.3	Descrizione	46
4.1.4	Ruoli	46
4.1.4.1	Responsabile	47
4.1.4.2	Amministratore	47
4.1.4.3	Analista	47
4.1.4.4	Progettista	47
4.1.4.5	Programmatore	48
4.1.4.6	Verificatore	48
4.1.5	Gestione oraria	48
4.1.6	Gestione delle comunicazioni	48
4.1.6.1	Comunicazioni interne	48
4.1.6.2	Comunicazioni esterne	49
4.1.7	Gestione degli incontri	49
4.1.7.1	Incontri interni	49
4.1.7.2	Verbali di riunioni interne	49
4.1.7.3	Compiti del Responsabile	49
4.1.7.4	Doveri dei partecipanti	49
4.1.7.5	Approvazione delle decisioni	49
4.1.7.6	Tracciamento delle decisioni	49
4.1.7.7	Incontri esterni del gruppo	50
4.1.7.8	Verbali di riunioni esterne	50
4.1.8	Gestione degli strumenti di coordinamento	50
4.1.8.1	Ticketing	50
4.1.9	Gestione dei rischi	50
4.1.9.1	Struttura dei rischi	51
4.1.10	Strumenti	51
4.2	Formazione	51
4.2.1	Scopo	51
4.2.2	Aspettative	52
4.2.3	Formazione dei membri del gruppo	52
4.2.4	Guide e documentazione	52
A	Standard per la qualità	53
A.1	Funzionalità	53
A.1.1	Adeguatezza	53
A.1.2	Accuratezza	53
A.1.3	Interoperabilità	53
A.1.4	Sicurezza	53
A.1.5	Aderenza alle funzionalità	53
A.2	Affidabilità	53
A.2.1	Maturità	53
A.2.2	Tolleranza ai guasti	53
A.2.3	Recuperabilità	54
A.2.4	Aderenza all'affidabilità	54
A.3	Usabilità	54
A.3.1	Comprensibilità	54
A.3.2	Apprendibilità	54
A.3.3	Operabilità	54

A.3.4	Attrattività	54
A.3.5	Aderenza all'usabilità	54
A.4	Efficienza	54
A.4.1	Comportamento rispetto al tempo	54
A.4.2	Utilizzo delle risorse	55
A.4.3	Aderenza all'efficienza	55
A.5	Manutenibilità	55
A.5.1	Analizzabilità	55
A.5.2	Modificabilità	55
A.5.3	Stabilità	55
A.5.4	Provabilità	55
A.5.5	Aderenza alla manutenibilità	55
A.6	Portabilità	55
A.6.1	Adattabilità	55
A.6.2	Installabilità	55
A.6.3	Coesistenza	56
A.6.4	Sostituibilità	56
A.6.5	Aderenza alla portabilità	56
B	Metriche per la qualità	57
B.1	Metriche per la qualità di processo	57
B.2	Metriche per la qualità di prodotto	59

Elenco delle tabelle

Tabella 2	Metriche per la pianificazione.	14
Tabella 3	Metriche per la progettazione.	25
Tabella 4	Metriche per la codifica.	30
Tabella 5	Metriche per la documentazione.	36
Tabella 6	Metriche per il miglioramento.	41
Tabella 7	Tipi di test di unità.	43
Tabella 8	Tipi di test di integrazione.	44
Tabella 9	Metriche per il testing.	45

Elenco delle figure

Figura 1	Ciclo di Deming.	40
----------	--------------------------	----

1 Introduzione

1.1 Scopo del documento

Lo scopo del documento è descrivere il *way of working_G* che il gruppo adotta per lo svolgimento del *progetto_G*, dunque i processi scelti, la loro istanziiazione e le modalità di miglioramento.

I processi sono divisi per le categorie individuate dallo standard *ISO/IEC 12207:1995_G*.

Il documento deve essere fruibile e le norme al suo interno devono essere rispettate da ogni membro del gruppo. Queste permettono di avere coerenza e consistenza nei prodotti, nei processi e nella documentazione, oltre che a lavorare in maniera efficace.

1.2 Scopo del prodotto

Lo scopo è realizzare una *web application_G* per supervisionare l'andamento della produzione di alcune macchine, tramite *carte di controllo_G*. Le *macchine_G* sono configurate da un amministratore, mentre l'applicazione delle carte di controllo è visualizzabile da tutti gli utenti su una o più *caratteristiche_G* scelte.

1.3 Glossario

Onde evitare ambiguità nei termini utilizzati nei documenti redatti, viene fornito in allegato anche il *Glossario v2.0*, dove vengono definiti tutti i termini con un significato particolare.

Un termine presente nel *Glossario* viene contrassegnato dal corsivo e da una 'G' aggiunta a pedice.

1.4 Riferimenti

1.4.1 Normativi

- Standard ISO/IEC 12207:1995:
https://www.math.unipd.it/~tullio/IS-1/2009/Approfondimenti/ISO_12207-1995.pdf;
- Standard ISO/IEC 9126:
 - https://it.wikipedia.org/wiki/ISO/IEC_9126;
 - La qualità del software secondo il modello ISO/IEC 9126 di Ercole F. Colonese:
http://www.colonese.it/00-Manuali_Pubblicatii/07-ISO-IEC9126_v2.pdf.
- Standard ISO 8601:
https://it.wikipedia.org/wiki/ISO_8601;
- Norme per la formattazione di Java:
<https://google.github.io/styleguide/javaguide.html#s4-formatting>;
- Norme per la formattazione di TypeScript:
<https://google.github.io/styleguide/tsguide.html#comments-documentation>;
- Norme per la formattazione di HTML e CSS:
<https://google.github.io/styleguide/htmlcssguide.html>;
- Capitolato d'appalto C3 - CC4D:
 - <https://www.math.unipd.it/~tullio/IS-1/2021/Progetto/C3p.pdf>;
 - <https://www.math.unipd.it/~tullio/IS-1/2021/Progetto/C3.pdf>.
- Verbale esterno: *VE_2021-11-05*;

- Verbale esterno: *VE_2021-12-23*;
- Verbale esterno: *VE_2022-01-11*;
- Verbale esterno: *VE_2022-02-21*;
- Verbale esterno: *VE_2022-03-15*;
- Verbale esterno: *VE_2022-04-26*.

1.4.2 Informativi

- *Piano di Progetto v2.0*;
- *Piano di Qualifica v2.0*;
- Slide dell'insegnamento di Ingegneria del Software, in particolare:
 - Processi di ciclo di vita del software:
<https://www.math.unipd.it/~tullio/IS-1/2021/Dispense/T03.pdf>;
 - Gestione di progetto:
<https://www.math.unipd.it/~tullio/IS-1/2021/Dispense/T06.pdf>;
 - Verifica e validazione - analisi statica:
<https://www.math.unipd.it/~tullio/IS-1/2021/Dispense/T15.pdf>;
 - Verifica e validazione - analisi dinamica:
<https://www.math.unipd.it/~tullio/IS-1/2021/Dispense/T16.pdf>.
- Software Engineering - Ian Sommerville (10th Edition), in particolare:
 - Chapter 25: configuration management.
- GitHub Documentation, in particolare:
 - about Git:
<https://docs.github.com/en/get-started/using-git/about-git>;
 - pushing commits to a remote repository:
<https://docs.github.com/en/get-started/using-git/pushing-commits-to-a-remote-repository>;
 - getting changes from a remote repository:
<https://docs.github.com/en/get-started/using-git/getting-changes-from-a-remote-repository>;
 - dealing with non-fast-forward errors:
<https://docs.github.com/en/get-started/using-git/dealing-with-non-fast-forward-errors>.
- Git Glossary:
<https://git-scm.com/docs/gitglossary>;
- L'arte di scrivere con L^AT_EX- Lorenzo Pantieri, Tommaso Gordini:
http://www.lorenzopantieri.net/LaTeX_files/ArteLaTeX.pdf, in particolare:
 - Capitolo 3: basi;
 - Capitolo 4: testo;
 - Capitolo 6: tabelle e figure;

– Capitolo 9: personalizzazioni.

- Ciclo di Deming:
https://it.wikipedia.org/wiki/Ciclo_di_Deming
- Metriche di progetto:
https://it.wikipedia.org/wiki/Metriche_di_progetto
- Metriche per *testing_G* di qualità:
<https://www.tricentis.com/blog/64-essential-testing-metrics-for-measuring-quality-assurance-success/>.

2 Processi primari

2.1 Fornitura

L'obiettivo del processo di fornitura è descrivere ogni compito e attività svolta dal fornitore con lo scopo di comprendere e soddisfare adeguatamente le richieste del proponente. Il fornitore, dopo aver compreso in maniera completa la domanda del proponente, stabilisce successivamente con il proponente tramite un contratto la data di consegna del prodotto e la gestione della manutenzione dello stesso. Viene scritto poi il *Piano di Progetto* che scaglionerà le varie attività da svolgere.

Il processo di fornitura è formato dalle seguenti fasi:

- avvio;
- approntamento di risposte alle richieste;
- contrattazione;
- pianificazione;
- esecuzione e controllo;
- revisione e valutazione;
- consegna e completamento.

2.1.1 Aspettative

Nell'intero svolgimento del progetto, il gruppo *DeltaX* ha intenzione di mantenere un contatto costante con l'azienda *Sanmarco Informatica S.p.A.* mediante un dialogo continuo.

Grazie a questa vicinanza, è possibile:

- stabilire i bisogni che il prodotto deve soddisfare;
- stilare requisiti e vincoli sui processi;
- stimare i costi;
- effettuare una verifica continua;
- chiarire eventuali dubbi emersi durante il progetto.

2.1.2 Analisi dei rischi

L'attività è svolta dal Responsabile insieme agli Amministratori, che:

- a inizio progetto analizzano i potenziali rischi che possono verificarsi durante lo svolgimento ostacolando, come illustrato in §4.1.9;
- se durante il progetto si verifica un rischio non preventivato, si riuniscono per:
 - attribuirgli una probabilità e una pericolosità;
 - valutare se il piano di contingenza attuato sia stato efficace, modificandolo altrimenti;
 - aggiungere il rischio alla lista di quelli possibili.
- al termine di ogni incremento si riuniscono per discutere di:
 - rischi preventivati che non si sono verificati, per decidere se:
 - * modificarne probabilità e pericolosità;

- * rimuoverli totalmente.
- rischi preventivati che si sono verificati, per decidere se:
 - * modificarne probabilità e pericolosità;
 - * il piano di contingenza attuato sia stato efficace, modificandolo altrimenti.

2.1.3 Pianificazione

L'attività è svolta dal Responsabile con l'aiuto degli Amministratori, che, sulla base della data di consegna stabilita dal Responsabile e del *modello di sviluppo_G* scelto:

- redigono un piano temporale per la realizzazione del progetto, in particolare:
 1. dividono il tempo disponibile tra le tre revisioni;
 2. dividono il tempo per ogni revisione in vari incrementi;
 3. organizzano il tempo di ogni incremento per svolgere le attività;
 4. producono dei *diagrammi di Gantt_G* che riassumono le scelte prese.
- alla fine di ogni incremento aggiornano la pianificazione in risposta a eventuali anticipi e ritardi del periodo appena concluso.

2.1.3.1 Metriche

Metrica	Nome	Riferimento
M2VP	Variazione di Piano	§B.1 M2VP
M3VC	Variazione di Costo	§B.1 M3VC
M4VR	Variazione Numero Requisiti	§B.1 M4VR

Tabella 2: Metriche per la pianificazione.

2.1.4 Preventivazione e consuntivazione

L'attività viene svolta dal Responsabile con l'aiuto degli Amministratori, che:

- a inizio progetto, sulla base della pianificazione redatta e delle risorse a disposizione, preventivano il costo di ogni incremento;
- al termine di ogni incremento, sulla base dell'andamento effettivo delle attività:
 - consuntivano il costo reale apportato dall'incremento;
 - aggiornano il preventivo a finire.

2.1.5 Controllo della qualità

L'attività viene svolta dagli Amministratori, conseguentemente alla definizione delle metriche nell'appendice B. Per ogni metrica, stabiliscono:

- un valore accettabile;
- un valore preferibile.

In qualsiasi momento del progetto possono aggiungere all'appendice nuove metriche, qualora ce ne fosse la necessità.

Successivamente, alla fine di ogni incremento:

- raccolgono le misurazioni delle metriche;
- aggiornano il cruscotto con i nuovi valori;
- discutono i risultati al primo incontro utile, per valutare eventuali azioni migliorative.

2.1.6 Coordinazione verifica e validazione

L'attività viene svolta dagli Amministratori, che, conseguentemente alla definizione dei requisiti di progetto:

- raccolgono i test di test di accettazione, definiti durante l'attività descritta in §2.2.5.4;
- raccolgono i test di test di sistema, definiti durante l'attività descritta in §2.2.5.4 ;
- raccolgono i test di integrazione, definiti durante l'attività descritta in §2.2.5.7;
- raccolgono i test di unità, definiti durante l'attività descritta in §2.2.5.8.

Inoltre, i Verificatori si occupano di raccogliere in delle liste gli errori riscontrati più frequentemente, come descritto in §3.4.4.2.

2.1.7 Definizione dei termini

L'attività viene svolta dagli Amministratori, che, dopo aver aggiunto un termine in un documento che potrebbe non essere noto ai destinatari o dipendere fortemente dal contesto del progetto:

- lo aggiungono alla lista nel foglio di calcolo condiviso apposito:
 - se il documento è il *Manuale Utente*, il foglio di calcolo è “Termini Glossario MU”;
 - se il documento è la *Specifica Architettuale*, il foglio di calcolo è “Termini Glossario SA”;
 - altrimenti, il foglio di calcolo è “Termini Glossario”.
- premono “Ordina foglio dalla A alla Z”, così da notare subito un eventuale termine doppiato, quindi eliminarlo e non continuare con la procedura;
- associano al termine una spiegazione concisa;
- controllano tutti gli altri documenti, cosicché, qualora il termine fosse presente, possano segnalarne la presenza nel *Glossario* tramite il comando `\gloss{termine}`.

2.1.8 Stesura Manuale Utente

2.1.9 Documenti

2.1.9.1 Piano di Progetto

Comprende:

- **analisi dei rischi:** riporta il risultato dell'attività descritta in §2.1.2;
- **modello di sviluppo:** illustra il modello di sviluppo scelto durante l'attività di pianificazione, descritta in §2.1.3;
- **pianificazione:** riporta il risultato dell'attività descritta in §2.1.3;
- **preventivo e consuntivo:** riporta il risultato dell'attività descritta in §2.1.4;
- **la composizione del gruppo.**

2.1.9.2 Piano di Qualifica

Comprende:

- **obiettivi di qualità:** contenente i valori che devono assumere le metriche per garantire la qualità del progetto, stabiliti durante l'attività descritta in §2.1.5. Si divide poi in:
 - **qualità di processo:** contenente i valori che le metriche dell'appendice B.1 devono assumere per garantire la qualità di processo;
 - **qualità di prodotto:** contenente i valori che le metriche dell'appendice B.2 devono assumere per garantire la qualità di prodotto;
 - **qualità per obiettivo:** contenente i valori che le metriche devono assumere per garantire la qualità, raggruppate in base ai relativi processi.
- **strategie di verifica:** contenente il risultato dell'attività descritta in §2.1.6;
- **cruscotto delle metriche:** contenente in forma schematica le misurazioni delle metriche per tracciare l'andamento della qualità del progetto, come definito dall'attività descritta in §2.1.5.

2.1.9.3 Glossario

Contiene il risultato dell'attività descritta in §2.1.7, organizzando in sezioni alfabeticamente ordinate tutti i termini elencati nel foglio di calcolo "Termini Glossario".

2.1.9.4 Manuale Utente

Contiene in appendice il risultato dell'attività descritta in §2.1.7, organizzando in sezioni alfabeticamente ordinate tutti i termini elencati nel foglio di calcolo "Termini Glossario MU".

2.1.10 Strumenti

Il gruppo nel corso del processo di fornitura utilizza:

- **Google Calendar:** sistema di calendari, usato per segnare le riunioni, scadenze e in generale gli eventi significativi che riguardano il gruppo;
- **Google Sheet_G:** parte del Google Workspace, suite di software e strumenti di produttività in cloud collaborativi, sono dei fogli di calcolo, usati come spiegato in §2.1.9.3;
- **Microsoft Excel:** parte del pacchetto suite Microsoft Office, è un software che permette la scrittura di fogli di calcolo tramite tabelle. Il gruppo li utilizza per calcolare i preventivi e la suddivisione delle ore per il *Piano di Progetto*;
- **Microsoft PowerPoint:** parte del pacchetto suite Microsoft Office, è un software che il gruppo usa per realizzare ed esporre le presentazioni;
- **ProjectLibre:** software di gestione progettuale utilizzato per la creazione dei diagrammi di Gantt.

2.2 Sviluppo

2.2.1 Scopo

Lo scopo del processo di sviluppo, come descritto dallo standard ISO/IEC 12207:1995, è descrivere i task e le attività di analisi, progettazione, codifica, integrazione, test, installazione e accettazione, riguardanti il prodotto software da sviluppare.

2.2.2 Aspettative

Includono:

- determinare vincoli tecnologici;
- determinare gli obiettivi di sviluppo;
- determinare vincoli di design;
- realizzare un prodotto finale che superi i test e soddisfi i requisiti e le aspettative del proponente.

2.2.3 Descrizione

Di seguito sono elencate e successivamente approfondite le attività caratterizzanti di questo processo:

- analisi dei requisiti;
- progettazione architettuale;
- codifica del software.

2.2.4 Analisi dei requisiti

2.2.4.1 Scopo

Lo scopo dell'attività di analisi dei requisiti è:

- assistere l'attività di progettazione, fornendo requisiti semplici e precisi;
- concordare con il proponente le aspettative sul prodotto, descrivendole con un linguaggio chiaro e semplice;
- assistere l'attività di pianificazione, fornendo informazioni utili al calcolo della mole di lavoro;
- assistere l'attività di verifica, facilitando il tracciamento dei requisiti.

2.2.4.2 Descrizione

L'analisi dei requisiti viene effettuata dagli Analisti, che si occupano di produrre il documento *Analisi dei Requisiti*, il quale deve contenere:

- gli attori, ovvero coloro che interagiscono con il prodotto;
- i casi d'uso, ovvero le possibili interazioni con il prodotto;
- i requisiti, ovvero le caratteristiche che il prodotto deve soddisfare;
- un tracciamento dai requisiti alle fonti e viceversa.

Il documento deve essere modificato finché non ottiene l'approvazione del proponente.

2.2.4.3 Analisi iniziale

L'analisi dei requisiti parte dallo studio del capitolato d'appalto per individuare i principali attori, casi d'uso (come illustrato in §2.2.4.7) e requisiti del prodotto (come illustrato in §2.2.4.8). In questa fase viene analizzato anche il regolamento di progetto per individuare eventuali requisiti interni.

2.2.4.4 Comunicazione con il proponente

Una volta prodotta una prima bozza, inizia una comunicazione continua con il proponente, atta a:

- risolvere dubbi;
- raffinare i casi d'uso e i requisiti già individuati;
- individuarne di nuovi.

Da ogni incontro emergono delle richieste del proponente che devono essere fissate nel relativo verbale e poi rielaborate per aggiornare l'analisi dei requisiti. In seguito alla rielaborazione potrebbero sorgere altri dubbi o alcuni casi d'uso potrebbero essere considerati non chiari, in tal caso si procede con lo svolgimento di un ulteriore incontro con il proponente.

2.2.4.5 Rielaborazione

Le richieste del proponente vanno rielaborate, in particolare:

- vanno prodotti casi d'uso per ogni macro-richiesta;
- ogni caso d'uso va scomposto in sottocasi d'uso più semplici;
- casi e sottocasi d'uso vanno associati a dei requisiti, e questa operazione va tracciata con una relativa tabella.

Alcune richieste del proponente potrebbero non essere azioni che un attore deve poter effettuare: in tal caso dovranno produrre direttamente dei requisiti, che dovranno avere come fonte il verbale ove è riportata tale richiesta.

2.2.4.6 Documento

L'analisi dei requisiti produce il documento *Analisi dei Requisiti*, contenente:

- una descrizione generale del prodotto;
- una lista contenente gli attori individuati e un diagramma UML che li riassume;
- una lista contenente i casi e sottocasi d'uso individuati, come descritto in §2.2.4.7;
- delle liste contenenti i requisiti individuati, divisi per tipologia e come descritti in §2.2.4.8;
- il tracciamento tra i requisiti e le loro fonti.

2.2.4.7 Casi d'uso

Un caso d'uso è un insieme di scenari con lo stesso obiettivo per un utente. Vuole descrivere l'insieme di funzionalità del sistema dal punto di vista degli utenti, quindi senza nessun dettaglio implementativo. È formato da:

- **diagramma UML_G** (opzionale): per rappresentare le relazioni con altri casi d'uso;
- **intestazione**, nel formato:

UC [codice] - [titolo]

dove:

- UC sta per “*Use Case*”, cioè “caso d'uso”;

- [codice] è il codice identificativo del caso d’uso, descritto successivamente;
- [titolo] è il titolo del caso d’uso.

Tale intestazione può essere generata automaticamente con il comando `\uc{[titolo_uc]}`. Per gerarchie con più di un livello si possono usare `\subuc`, `\subsubuc` e `\subsubsubuc` al posto di `\uc`;

- **attore_G/i primario/i**: entità esterna al sistema e che vi interagisce;
- **attore/i secondario/i** (opzionale): entità esterna al sistema di supporto per il raggiungimento dell’obiettivo dell’attore primario;
- **descrizione**: breve descrizione del caso d’uso;
- **scenario principale**: elenco puntato che scandisce il flusso degli eventi;
- **estensioni** (opzionali): gli scenari alternativi, che iniziano la loro esecuzione interrompendo il caso d’uso che ne ha verificato le precondizioni;
- **precondizioni**: le condizioni del sistema prima del verificarsi del caso d’uso;
- **postcondizioni**: le condizioni del sistema dopo che si è verificato il caso d’uso.

Il codice di un caso d’uso senza padre è un numero progressivo univoco tra i casi d’uso di questo tipo. Il codice di un sottocaso d’uso è nel formato:

[codice_padre].[numero_figlio]

dove:

- **codice_padre** è il codice che identifica in maniera univoca il padre;
- **numero_figlio** è un numero progressivo univoco tra i figli dello stesso padre.

2.2.4.8 Requisiti

I requisiti dell’*Analisi dei Requisiti* devono essere divisi in quattro tipologie:

- **funzionali**: descrivono le funzionalità e il comportamento che il prodotto deve avere;
- **di qualità**: descrivono vincoli sulla qualità del prodotto, per esempio componenti di cui deve essere composto il prodotto, come codice sorgente e schemi delle basi di dati;
- **di vincolo**: descrivono vincoli sull’implementazione del prodotto, per esempio sulle tecnologie utilizzabili;
- **prestazionali**: descrivono vincoli di tempo e spazio che il prodotto deve soddisfare durante la sua esecuzione.

Ogni requisito ha una certa importanza:

- **obbligatorio**: il requisito deve essere soddisfatto nel prodotto finale;
- **desiderabile**: il requisito non è indispensabile, ma è molto gradita la sua soddisfazione;
- **opzionale**: il requisito è completamente facoltativo.

Ogni requisito ha un codice univoco nel formato:

R[importanza][tipologia][numero]

dove:

- [importanza] è:
 - **O** se il requisito è obbligatorio;
 - **D** se il requisito è desiderabile;
 - **P** se il requisito è opzionale.
- [tipologia] è:
 - **F** se il requisito è funzionale;
 - **Q** se il requisito è di qualità;
 - **V** se il requisito è di vincolo;
 - **P** se il requisito è prestazionale.
- [numero] è un numero progressivo univoco se il requisito non ha padre, altrimenti è nel formato:

[numero_padre].[numero_figlio]

dove:

- [numero_padre] è il [numero] del requisito padre;
- [numero_figlio] è un numero progressivo univoco tra i figli dello stesso padre.

Ogni requisito deve essere creato come risposta a un bisogno espresso in una o più fonti. Le fonti possono essere:

- il capitolato;
- i verbali degli incontri con il proponente;
- interne, se derivano dal fornitore;
- i casi d'uso.

I requisiti vengono elencati in quattro tabelle, una per tipologia, e ciascuna deve contenere le seguenti colonne:

- **codice**: il codice univoco del requisito;
- **importanza**: l'importanza del requisito;
- **descrizione**: una breve descrizione del requisito;
- **fonti**: la o le fonti che hanno portato alla creazione del requisito.

2.2.4.9 UML

Tutti i diagrammi UML devono essere realizzati nella versione 2.0.

2.2.5 Progettazione

2.2.5.1 Scopo

L'obiettivo dell'attività di progettazione è individuare in base ai requisiti specificati nel documento *Analisi dei Requisiti v2.0* tutte le caratteristiche che dovranno essere presenti nel prodotto finale. Questo per produrre la miglior soluzione che soddisfi a pieno gli *stakeholder_G*. Il processo di progettazione è inverso rispetto all'analisi dei requisiti, infatti:

- **analisi dei requisiti:** divisione di un problema in parti per capirne il dominio applicativo;
- **progettazione:** ricostruzione di un problema specificando la funzionalità di ogni parte.

La progettazione permette di scomporre l'implementazione in componenti così da poter suddividere il lavoro tra i Programmatori, ottimizzando tempi e *risorse_G*.

2.2.5.2 Aspettative

Terminata questa attività, si deve:

- avere chiara l'architettura del sistema;
- aver svolto uno studio approfondito sulle possibili tecnologie applicabili, con relativi pro e contro di ognuna, dal quale poi deve derivare la scelta effettiva sulle tecnologie.

2.2.5.3 Descrizione

La progettazione è formata da due parti:

- ***Technology Baseline_G***, che comprende a sua volta:
 - le definizioni dei test di accettazione e di sistema;
 - le scelte tecnologiche;
 - il *Proof of Concept_G* che ne dimostra l'utilizzo.
- ***Product Baseline_G***, che completa l'attività di progettazione, ed è diviso a sua volta in:
 - progettazione architeturale di sistema;
 - progettazione architeturale delle componenti;
 - progettazione di dettaglio.

2.2.5.4 Definizione dei test di accettazione e di sistema

In seguito al completamento dei requisiti nell'*Analisi dei Requisiti* gli Amministratori devono redigere le specifiche dei test di accettazione (§3.5.4) e di sistema (§3.4.5.3).

2.2.5.5 Scelta delle tecnologie e *Proof of Concept*

Viene effettuata dal Responsabile con l'aiuto degli Amministratori, in seguito al termine dell'analisi dei requisiti. La scelta va effettuata analizzando vantaggi e svantaggi di ogni tecnologia alla luce dei requisiti del progetto. Tale scelta viene validata da Progettisti e Programmatori attraverso la realizzazione di un *Proof of Concept* che dimostri l'attuabilità del progetto.

2.2.5.6 Progettazione architettuale di sistema

Viene effettuata dai Progettisti in seguito alla revisione RTB. Essi si devono occupare di dare una definizione ad alto livello del sistema software da produrre. Include:

- la descrizione del modello architettuale scelto;
- la definizione delle componenti software del sistema;
- la definizione dell'interfaccia di ogni componente individuata, suddivise per componenti del sistema:
 - le prime interfacce sono individuate sulla base dei requisiti che sono stati individuati nell'*Analisi dei Requisiti* (§2.2.4), accorpendo nella stessa interfaccia requisiti di funzionalità simili;
 - per ogni interfaccia si individuano i requisiti necessari per permettere la loro implementazione;
 - l'operazione si ripete partendo dai nuovi requisiti.
- il comportamento in esecuzione delle componenti del sistema, attraverso diagrammi di sequenza;
- qualora i comportamenti siano particolarmente articolati, la descrizione è corredata di un diagramma di attività;
- la specifica dei dettagli tecnici della configurazione di sistema, ovvero quali strumenti usare, quali porte di comunicazione e quali indirizzi;
- come le componenti vengono assemblate.

2.2.5.7 Progettazione architettuale delle componenti

Viene effettuata dai Progettisti in seguito alla definizione di ogni interfaccia delle componenti durante la progettazione architettuale di sistema. Il loro compito è descrivere per ogni componente la sua architettura interna e i design pattern che usa. Include:

- la definizione dei suoi moduli e unità attraverso i diagrammi delle classi;
- la specifica dei test di integrazione (§3.4.5.2) per testare tali moduli.

2.2.5.8 Progettazione di dettaglio

Viene effettuata dai Progettisti in seguito alla specifica statica della componente. Essi si occupano di definire il comportamento in esecuzione delle unità software. Include:

- la definizione dell'interazione degli oggetti concreti della componente attraverso i diagrammi di sequenza;
- qualora i comportamenti degli oggetti siano particolarmente articolati, la descrizione è corredata di un diagramma di attività;
- la specifica dei test di unità (§3.4.5.1).

2.2.5.9 Documento

La progettazione produce il documento *Specifica Architetturale*, contenente:

- una descrizione generale delle tecnologie utilizzate, scelte durante l'attività descritta in §2.2.5.5;
- l'architettura di sistema, definita durante l'attività descritta in §2.2.5.6, la quale contiene:
 - la descrizione dell'architettura complessiva del prodotto;
 - la divisione in componenti e le loro connessioni;
 - la configurazione di sistema, la quale definisce per ogni componente:
 - * l'interfaccia;
 - * eventuali requisiti per i livelli inferiori.
- l'architettura delle componenti, come descritto in §2.2.5.7, la quale contiene, per ogni componente:
 - una descrizione generale della sua architettura;
 - i design pattern principali usati;
 - la struttura dei package;
 - la lista di moduli, ognuno con:
 - * una descrizione generale;
 - * un diagramma delle classi;
 - * una lista di definizioni di classi, interfacce e altri componenti del modulo.
- la progettazione di dettaglio, ovvero la specifica di ogni unità, come descritto in §2.2.5.8, la quale contiene:
 - per ogni metodo e costruttore, raggruppati per classe:
 - * una descrizione di come debbano essere implementati, ovvero delle operazioni che devono effettuare;
 - * i metodi privati che utilizzano;
 - * se sono privati, allora anche la loro firma come quelle riportate nell'architettura delle componenti.
 - per ogni metodo di java con annotazione `@Query`, raggruppati per interfaccia:
 - * una descrizione della query da inserire nell'annotazione `@Query`.
 - i diagrammi di sequenza, ognuno con una descrizione contenente:
 - * le principali interazioni tra le classi;
 - * le scelte che hanno portato a quell'esecuzione piuttosto che a un'altra (ad esempio eventuali controlli ed errori);
 - * il significato delle classi esterne non presenti nell'architettura delle componenti.
- in appendice il risultato dell'attività descritta in §2.1.7, organizzando in sezioni alfabeticamente ordinate tutti i termini elencati nel foglio di calcolo "Termini Glossario SA".

2.2.5.9.1 Interfaccia del front-end

Per ogni interfaccia del front-end è specificato lo scopo e sono elencati:

- **percorso**: il percorso alla schermata;
- **elementi**: gli elementi che la compongono, tracciati con i requisiti dell'*Analisi dei Requisiti* che soddisfano.

2.2.5.9.2 Interfaccia del back-end e API rilevazioni

Per ogni interfaccia del back-end e dell'API rilevazioni è specificato lo scopo e sono elencati:

- **endpoint**: il percorso dell'URL;
- **metodo HTTP**: il metodo HTTP usato;
- **body**: il contenuto del corpo della richiesta, con il formato usato e i campi che contiene;
- la risposta, eventualmente divisa in esito positivo ed esito negativo, specificando:
 - **codice di stato HTTP**: lo stato HTTP ritornato;
 - **body**: il contenuto del body.

2.2.5.9.3 Requisiti

Ogni requisito per i livelli inferiori deve avere un codice univoco nel formato:

RI [C] [numero]

dove:

- [C] è:
 - F se il requisito è del front-end;
 - B se il requisito è del back-end;
 - A se il requisito è dell'API rilevazioni.
- [numero] è un numero progressivo univoco.

Ogni requisito deve essere creato come risposta a un bisogno espresso in una o più fonti, dove le fonti sono le interfacce individuate in §2.2.5.6.

2.2.5.9.4 Definizione delle classi

La definizione di ogni classe, interfaccia o componente di un modulo deve contenere:

- una descrizione generale;
- il tipo di elemento (classe, interfaccia, record, enumerazione);
- (solo per i component di Angular) la composizione della *View_G*, che la descrive mediante i tag HTML5 da usare;
- la lista delle annotazioni necessarie;
- la lista dei campi privati, ognuno con il suo tipo, nome e descrizione;
- la lista dei campi pubblici, ognuno con il suo tipo, nome e descrizione;
- la lista dei costruttori;
- la lista dei metodi, ognuno con:
 - le sue annotazioni, eccetto `@Override`, ognuna su una nuova riga;
 - il suo tipo di ritorno;
 - una lista dei parametri, ognuno su una nuova riga, con il suo tipo, nome e descrizione;

- le eventuali eccezioni lanciate;
- una descrizione generale, su una nuova riga.

Per le componenti scritte nel linguaggio Java sono implicite:

- l'esistenza di un costruttore che accetta un parametro per campo e lo assegna al relativo campo, a meno che non siano definiti altri costruttori;
- l'esistenza di un getter per ogni campo di un record, con lo stesso nome del relativo campo.

2.2.5.9.5 Diagrammi delle classi

I diagrammi delle classi devono rappresentare in modo schematico le classi e interfacce delle componenti, mostrando le relazioni di dipendenza tra di esse. Durante la loro scrittura:

- va seguito lo standard UML 2.0;
- vanno segnati tutti i campi, o direttamente nelle classi o sulle frecce di dipendenza;
- vanno segnati i metodi più importanti;
- è possibile e preferibile evitare di segnare i metodi privati e ereditati;
- è possibile segnare la presenza di design pattern mediante l'uso di note;
- vanno segnate le eccezioni con *«throws»*;
- nei diagrammi delle classi in linguaggio Java vanno colorate:
 - in verde le classi che dipendono dal framework Spring;
 - in rosso tutte le altre.

Diagrammi di sequenza

I diagrammi delle classi devono rappresentare in modo schematico le interazioni a runtime tra istanze delle varie classi. Durante la loro scrittura:

- va seguito lo standard UML 2.0;
- per le istanze di classi esterne con nomi non noti si possono utilizzare nomi descrittivi composti da più parole, avendo cura di definirne il significato nella descrizione del diagramma di sequenza;
- nei diagrammi di sequenza in linguaggio Java vanno colorate:
 - in verde le classi che dipendono dal framework Spring;
 - in rosso tutte le altre.

2.2.5.10 Metriche

Metrica		Nome	Riferimento
M11PRCI	Percentuale di	<i>Requisiti di Configurazione_G</i> Implementati	§B.2 M11PRCI
M12PMG		Profondità Media delle Gerarchie	§B.2 M12PMG

Tabella 3: Metriche per la progettazione.

2.2.6 Codifica

2.2.6.1 Scopo

La codifica, svolta dai Programmatori, ha come scopo l'effettiva realizzazione del prodotto software richiesto. Permette la trasformazione in codice dell'intera architettura pensata dai Progettisti. Il codice potrà poi essere eseguito da un elaboratore. È importante che i Programmatori seguano queste norme per creare del codice conforme.

2.2.6.2 Aspettative

La codifica deve avere come risultato un prodotto software conforme alle caratteristiche e ai requisiti concordati con il proponente. L'uso delle norme in questa fase è fondamentale per:

- creare codice leggibile, uniforme, robusto ed efficiente;
- migliorare la qualità del prodotto;
- facilitare la verifica;
- facilitare le procedure di manutenzione ed estensione.

2.2.6.3 Implementazione delle unità

Viene svolta dai Programmatori a seguito della progettazione di dettaglio di ogni unità. Essi si devono occupare di implementare le unità e i relativi test di unità seguendo la specifica precedentemente prodotta. L'esito positivo dell'esecuzione dei test di unità è requisito obbligatorio per procedere alla verifica dell'unità.

2.2.6.4 Implementazione dei moduli

Viene svolta dai Programmatori a seguito dell'implementazione di tutte le unità che compongono un modulo. Essi si devono occupare di implementare i moduli, unendo le unità che ne fanno parte, e i relativi test di integrazione. L'esito positivo dell'esecuzione dei test di integrazione è requisito obbligatorio per procedere alla verifica del modulo.

2.2.6.5 Stile di codifica - *Java_G*

2.2.6.5.1 Struttura dei file

La struttura di un sorgente Java è definita dalle seguenti parti:

1. **package:** dichiarazione del package di appartenenza;
2. **import:** istruzioni di import, nell'ordine: librerie di progetto e librerie standard Java;
3. **classe:** esattamente una classe, interfaccia o enumerazione di primo livello. Questa regola non viene applicata in fase di *testing_G*.

2.2.6.5.2 Struttura delle classi

La struttura di una classe Java è definita dalle seguenti parti, ove ritenute necessarie:

1. le classi estese e poi le interfacce implementate;
2. i membri statici, prima i campi e poi i metodi;
3. i campi, nell'ordine: costanti e variabili;

4. i costruttori;
5. i getter e setter;
6. i metodi.

Ogni classe deve avere sopra di essa un blocco di commenti in formato *Javadoc_G* che la descriva brevemente.

Ogni metodo di una classe deve avere sopra di esso un blocco di commenti in formato *Javadoc_G*, che ne descriva:

1. lo scopo;
2. i parametri;
3. il valore di ritorno.

2.2.6.5.3 Pratiche di programmazione

- **metodi**: optare per metodi brevi, se un metodo è troppo lungo, scorporarlo in più metodi;
- **getters e setters**: preferire *getters* e *setters* a campi pubblici o protetti;
- **ricorsione**: la ricorsione va evitata quanto più possibile poiché potrebbe indurre a una maggiore occupazione di memoria rispetto a soluzioni iterative e rende il codice più difficile da verificare;
- **this**: all'interno di una classe si deve usare il parametro con **this** per riferirsi a campi e metodi;
- **visibilità**: per ogni dichiarazione bisogna usare sempre la minor visibilità concessa;
- **tipi primitivi**: preferire i tipi primitivi ai corrispettivi oggetti del linguaggio;
- **membri statici**: i membri statici devono essere immutabili e acceduti dalla classe e non dalle istanze;
- **eccezioni**: la classe `RuntimeException` deve essere estesa solo per errori implementativi;
- **commenti**: i commenti che svolgono una delle seguenti funzioni devono prefiggere le seguenti abbreviazioni:
 - **TODO**: per definire una sezione di codice non ancora sviluppata;
 - **HACK**: per definire una sezione di codice sviluppata, ma migliorabile;
 - **FIXME**: per definire una sezione di codice non funzionante.
- **final**: dove possibile, marcare con **final**: classi, metodi, campi, parametri e variabili;
- **optional**: l'uso di `Optional` deve essere preferito ai valori `null`.

2.2.6.5.4 Formattazione del codice

Per la formattazione del codice Java è stato adottato il modello indicato al seguente indirizzo: <https://google.github.io/styleguide/javaguide.html#s4-formatting>, affiancato allo strumento di formattazione: <https://github.com/google/google-java-format>.

2.2.6.5.5 Convenzioni sui nomi

- **leggibilità:** i nomi assegnati devono racchiudere il significato di ciò a cui si riferiscono, evitando di essere troppo concisi e mal interpretabili;
- **package:** i nomi dei package devono usare solo lettere minuscole;
- **classi:** i nomi delle classi seguono lo stile *PascalCase_G*;
- **costruttori:** i nomi dei parametri dei costruttori devono corrispondere al nome del campo a cui si riferiscono, all'interno di un costruttore i campi devono essere acceduti tramite la parola chiave `this`;
- **metodi:** i nomi dei metodi seguono lo stile *camelCase_G*;
- **getter e setter:** i nomi dei getter e dei setter seguono lo stile *camelCase_G*, ma con “get” o “set” come prefisso;
- **costanti:** le costanti seguono lo stile *UPPER_SNAKE_CASE_G*;
- **variabili:** i nomi delle variabili seguono lo stile *camelCase*;
- **parametri:** i nomi dei parametri seguono lo stile *camelCase*;
- **tipi generici:** i nomi dei tipi generici vanno indicati con una lettera maiuscola.

2.2.6.6 Stile di codifica - *TypeScript_G*

2.2.6.6.1 Struttura dei file

La struttura di un sorgente TypeScript è definita dalle seguenti parti:

1. **import:** istruzioni di import;
2. **classe:** esattamente una classe, interfaccia o enumerazione di primo livello. Questa regola non viene applicata in fase di testing.

2.2.6.6.2 Struttura delle classi

La struttura di una classe TypeScript è definita dalle seguenti parti, ove ritenute necessarie:

1. le classi estese e poi le interfacce implementate;
2. i membri statici, prima i campi e poi i metodi;
3. i campi, nell'ordine: costanti e variabili;
4. i costruttori;
5. i *value accessor*;
6. i metodi.

Ogni metodo di una classe deve avere sopra di esso un blocco di commenti *JSDoc_G* che ne descrive:

1. lo scopo;
2. i parametri;
3. il valore di ritorno.

2.2.6.6.3 Pratiche di programmazione

- **import:** i percorsi usati nelle istruzioni di `import` devono essere relativi;
- **moduli:** i moduli devono essere preferiti ai *namespace*;
- **metodi:** optare per metodi brevi, se un metodo è troppo lungo, scorporarlo in più metodi;
- **ricorsione:** la ricorsione va evitata quanto più possibile poiché potrebbe indurre a una maggiore occupazione di memoria rispetto a soluzioni iterative e rende il codice più difficile da verificare;
- **visibilità:** per ogni dichiarazione bisogna usare sempre la minor visibilità concessa;
- **modificatore public:** evitare di scrivere esplicitamente il modificatore `public`;
- **identificatori privati:** preferire il modificatore `private` agli identificatori privati;
- **costruttori:** i costruttori devono sempre essere richiamati con le parentesi;
- **value accessor:** i value accessor seguono queste regole:
 - il “get” deve essere una funzione pura;
 - se c’è solo un “get” ed è senza operazioni, deve essere sostituito da un campo pubblico `readonly`;
 - se sia il “get” che il “set” sono senza operazioni, deve essere sostituito da un campo pubblico (da notare che questo significa, il più delle volte, un errore di progettazione).
- **valori di ritorno:** esplicitare sempre il tipo di ritorno delle funzioni;
- **tipi primitivi:** preferire i tipi primitivi ai corrispettivi oggetti del linguaggio;
- **array:** gli array devono essere costruiti con la notazione a parentesi quadre;
- **commenti:** i commenti che svolgono una delle seguenti funzioni devono prefiggere le seguenti abbreviazioni:
 - **TODO:** per definire una sezione di codice non ancora sviluppata;
 - **HACK:** per definire una sezione di codice sviluppata, ma migliorabile;
 - **FIXME:** per definire una sezione di codice non funzionante.
- **optional:** il costrutto `optional` deve essere preferito al valore `undefined`.

2.2.6.6.4 Formattazione del codice

Per la formattazione del codice TypeScript è stato adottato il modello indicato al seguente indirizzo: <https://google.github.io/styleguide/tsguide.html#comments-documentation>.

2.2.6.6.5 Convenzioni sui nomi

- **leggibilità:** i nomi assegnati devono racchiudere il significato di ciò a cui si riferiscono, evitando di essere troppo concisi e mal interpretabili;
- **univocità:** tutti i nomi assegnati in una stessa classe devono essere univoci per evitare ambiguità;
- **file sorgenti:** i file sorgenti devono essere chiamati con lo stesso nome della classe che contengono;
- **interfacce:** i loro nomi seguono lo stile PascalCase;

- **classi:** i nomi delle classi seguono lo stile PascalCase e, se implementano una e una sola interfaccia e sono l'unica classe a farlo, aggiungono il suffisso `Impl`;
- **metodi:** i nomi dei metodi seguono lo stile camelCase;
- **value accessor:** i nomi dei value accessor seguono le stesse regole dei nomi degli attributi di classe e sono preceduti da una delle due keyword “get” o “set”;
- **costanti:** i nomi delle costanti seguono lo stile UPPER_SNAKE_CASE;
- **variabili:** i nomi delle variabili seguono lo stile camelCase;
- **parametri:** i nomi dei parametri seguono lo stile camelCase;
- **tipi generici:** i nomi dei tipi generici vanno indicati con una lettera maiuscola.

2.2.6.7 Stile di codifica - SQL_G

2.2.6.7.1 Struttura dei file

La struttura di una file SQL è definita da costrutti SQL separati da una linea vuota, eccezione fatta per le istruzioni in singola riga.

2.2.6.7.2 Convenzioni sui nomi

- **leggibilità:** i nomi assegnati devono racchiudere il significato di ciò a cui si riferiscono, evitando di essere troppo concisi e mal interpretabili;
- **univocità:** tutti i nomi delle relazioni devono essere univoci per evitare ambiguità;
- **parole chiave:** le parole chiave del linguaggio devono essere scritte in maiuscolo;
- **relazioni:** i nomi delle relazioni seguono lo stile *snake_case_G*;
- **attributi:** gli attributi seguono lo stile *snake_case*.

2.2.6.8 Stile di codifica - $HTML_G$ e CSS_G

Per lo stile di codifica dei linguaggi HTML e CSS è stato adottato il modello indicato al seguente indirizzo: <https://google.github.io/styleguide/htmlcssguide.html>.

2.2.6.9 Metriche

Metrica	Nome	Riferimento
M7CCM	<i>Complessità Ciclomantica_G</i> Media	§B.1 M7CCM
M8SC	Statement Coverage	§B.2 M8SC
M9BC	Branch Coverage	§B.2 M9BC
M13LMCM	Linee Medie di Codice per Metodo	§B.2 M13LMCM
M14LCC	Linee di Commenti per Codice	§B.2 M14LCC

Tabella 4: Metriche per la codifica.

3 Processi di supporto

3.1 Documentazione

3.1.1 Descrizione

Questa sezione contiene le norme relative alla documentazione di progetto, cioè come essa vada creata, strutturata e aggiornata. I sorgenti dei documenti sono tracciati nel *repository*_G <https://github.com/DeltaXswe/documentazione-interna>.

3.1.2 Scopo

Lo scopo di questo processo è documentare e tenere traccia di tutti i processi e scelte avvenute durante lo sviluppo di questo progetto.

3.1.3 Lista documenti

I documenti che devono essere prodotti sono:

- *Norme di Progetto*;
- *Piano di Progetto*;
- *Piano di Qualifica*;
- *Analisi dei Requisiti*;
- *Specifica Architettuale*;
- *Manuale Utente*;
- *Verbali interni ed esterni*;
- *Glossario*.

3.1.4 Ciclo di vita

Ogni documento ha un ciclo di vita composto dalle seguenti fasi:

- **creazione**: il documento viene creato seguendo il template contenuto nella cartella template del repository, il quale deve seguire la struttura e le convenzioni successivamente riportate;
- **realizzazione**: il documento viene progressivamente aggiornato man mano che i contenuti vengono prodotti e aggiornati;
- **revisione**: ogni modifica o gruppo di modifiche al documento viene controllata da almeno un membro del gruppo, diverso da coloro che hanno apportato tali modifiche;
- **approvazione**: il documento viene dichiarato completato dal Responsabile e può essere quindi rilasciato.

3.1.5 Template

Per i documenti viene utilizzato un template in formato $LaTeX_G$. Per poter essere utilizzato bisogna inserire nella prima riga del documento il comando `\documentclass{posizione/del/file/template}`. Il template ha lo scopo di uniformare dal punto di vista grafico i documenti e di facilitarne la produzione grazie ai comandi che mette a disposizione. Nelle seguenti sezioni vengono ampiamente descritte le sue funzionalità e le convenzioni adottate per i documenti.

3.1.6 Struttura

Per agevolare la scrittura parallela di parti diverse del documento, ogni documento è diviso in più file `.tex`, uno per sezione. I file vengono poi uniti in un file utilizzando il comando `\input{file.tex}`, ottenendo così dalla compilazione un unico file `.pdf`.

3.1.6.1 Indentazione

L'indentazione del contenuto dei file `.tex` e i ritorni a capo all'interno delle frasi di contenuto devono essere evitati il più possibile in quanto complicano la lettura del codice sorgente durante la revisione delle *pull request*_G su *GitHub*_G. È concesso l'uso del tabulatore per far rientrare gli `\item` degli elenchi puntati.

3.1.6.2 Annotazioni

Eventuali annotazioni lasciate durante la redazione del documento per segnalare parti incomplete o spunti con cui ampliare le sezioni devono seguire il formato:

`%TODO commento`

in quanto il `TODO` viene scritto in rosso su Texmaker e diventa quindi di facile identificazione, riducendo così la probabilità di incorrere in refusi.

3.1.6.3 Prima pagina

La prima pagina deve contenere i seguenti elementi, in ordine:

- **logo del gruppo:** reperibile nella cartella template del repository in formato `.svg`;
- **nome del gruppo:** *DeltaX*;
- **nome del progetto:** Progetto *Produlytics*;
- **indirizzo email del gruppo:** `deltax.swe@gmail.com`;
- **nome del documento**;
- una tabella con le seguenti informazioni:
 - **versione del documento:** il numero di versione del documento, in formato `X.Y`;
 - **data approvazione:** la data in cui il Responsabile ha approvato il documento;
 - **responsabile dell'approvazione:** il nominativo del Responsabile che ha approvato il documento;
 - **redattori:** i nominativi di chi ha contribuito alla stesura del documento;
 - **verificatori:** i nominativi di chi ha contribuito alla verifica del documento;
 - **stato del documento:** può essere “approvato” o “non approvato” e indica se il Responsabile ha approvato il documento o meno;
 - **destinatari:** a chi è rivolto il documento;
 - **uso:** può essere “esterno” o “interno”, a seconda se ha come destinatario anche il proponente o meno.
- un breve sommario.

Per la creazione della pagina sono stati creati i seguenti comandi, che sono utilizzabili prima del `\begin{document}`:

- `\titolo{[titolo]}` per impostare `[titolo]` come titolo del documento;
- `\versione{[versione]}` per impostare `[versione]` come la versione del documento;
- `\data{[data]}` per impostare `[data]` come la data di approvazione del documento;
- `\responsabile{[responsabile]}` per impostare `[responsabile]` come il Responsabile del documento;
- `\redattori{[redattori]}` per impostare `[redattori]` come i redattori del documento, separando ciascuno con `\\`;
- `\verificatori{[verificatori]}` per impostare `[verificatori]` come i Verificatori del documento, separando ciascuno con `\\`;
- `\stato{[stato]}` per impostare `[stato]` come lo stato del documento (ad esempio “Approvato”);
- `\destinatari{[destinatari]}` per impostare `[destinatari]` come i destinatari del documento, separando ciascuno con `\\`;
- `\uso{[uso]}` per impostare `[uso]` come il tipo di uso del documento (ad esempio “Interno” o “Esterno”);
- `\descrizione{[descrizione]}` per impostare il sommario del documento.

Una volta usati i precedenti comandi, il comando `\firstpage`, da usare dopo il comando `\begin{document}`, genera automaticamente una prima pagina conforme.

3.1.6.4 Registro delle modifiche

La seconda pagina deve essere il registro delle modifiche, ovvero una tabella riportante ogni modifica apportata al documento:

- **ver.:** la versione del documento dopo la modifica;
- **data:** la data della verifica;
- **nominativi:** i membri del gruppo che hanno contribuito alla modifica e alla sua verifica oppure il membro del gruppo che ha approvato il documento;
- **ruoli:** i ruoli che ricoprono i membri, è sempre presente un Verificatore, a meno che non si tratti di un’approvazione del Responsabile;
- **descrizione:** una descrizione sintetica della modifica.

È possibile inserire tali informazioni in un file `.csv` i cui nomi delle colonne corrispondono a quelli appena descritti. Una volta fatto ciò il comando `\changelog{[file.csv]}` genera automaticamente il registro delle modifiche.

3.1.6.5 Indice

Successivamente al registro delle modifiche, in una nuova pagina, deve essere presente l’indice, ovvero un elenco di tutte le parti che compongono il documento. Se presenti nel documento, le due pagine successive sono dedicate rispettivamente all’elenco delle tabelle e all’elenco delle figure.

3.1.6.6 Contenuto

Le pagine di contenuto, che comprendono anche le pagine con il registro delle modifiche e gli indici, sono strutturate in:

- un'intestazione con:
 - il **logo** e il **nome del gruppo** a sinistra;
 - la **sezione corrente** a destra.
- una sezione centrale, con il **contenuto vero e proprio**;
- un piè di pagina con:
 - il **titolo del documento**;
 - il **numero della pagina attuale** con accanto quello delle **pagine totali** nel formato “Pagina x di y”.

Se il template è stato incluso, le pagine sono automaticamente strutturate in questo modo. L'uso del comando `\titolo{[titolo]}` è necessario per l'inserimento del titolo a piè di pagina.

Il contenuto dei documenti deve essere organizzato in sezioni (`\section{[titolo]}`), sottosezioni (`\subsection{[titolo]}`), sottosottosezioni (`\subsubsection{[titolo]}`) e paragrafi (`\paragraph{[titolo]}`) utilizzando i relativi comandi \LaTeX . L'importanza di ogni livello deve essere coerente all'interno dell'intero documento. Gli ultimi livelli ovviamente possono essere omessi se il contenuto non ha ragione di essere ulteriormente diviso.

3.1.6.7 Verbalì

I *Verbalì*, per loro natura e a differenza degli altri documenti, hanno un'unica stesura, in quanto non sono evolvibili nel tempo, ma sono comunque soggetti a verifica e approvazione. Il loro contenuto comprende, in ordine:

- un'intestazione con:
 - **luogo**, che può essere:
 - * la sede fisica in cui si è tenuto l'incontro;
 - * “online” se si è svolto online.
 - **data**;
 - **orario d'inizio**;
 - **orario di fine**;
 - **partecipanti**, che possono essere:
 - * la totalità o una parte dei membri del gruppo *DeltaX*;
 - * persone esterne al gruppo, come i rappresentanti del proponente.
 - **Segretario**: il nominativo di chi ricopre il ruolo di Segretario durante l'incontro;
 - **ordine del giorno**: un riassunto di ciò che sarà discusso durante l'incontro.
- **considerazioni** (opzionali);
- **decisioni prese**, ognuna composta da:
 - **codice**: un codice identificativo nel formato:

`V[I/E]_[YYYY-MM-DD] . [X]`

dove:

- * [I/E] è:
 - I se il *Verbale* è interno;
 - E se il *Verbale* è esterno.
 - * [YYYY-MM-DD] è la data del *Verbale*;
 - * [X] è il numero della decisione.
 - **decisione**: una breve descrizione della decisione presa;
 - **incaricati** (facoltativo): i nominativi di chi prenderà in carico la decisione, se è una decisione che non necessita di incaricati contiene “-”;
 - **id issue** (facoltativo): l’id della issue associata, vedi §4.1.8.1, se è una decisione che non necessita di una issue contiene “-”.
- gli **argomenti eventualmente** lasciati in sospeso, che verranno recuperati all’incontro successivo.

3.1.7 Convenzioni stilistiche

3.1.7.1 Nomi dei file

I nomi di file e cartelle utilizzano lo stile “snake case”, ovvero:

- i caratteri sono tutti in minuscolo;
- i nomi composti da più parole vengono separati da un underscore “_” invece che da uno spazio.

In particolare, i nomi dei file principali dei singoli documenti sono:

- **Norme di Progetto**: norme_di_progetto_vX.Y;
- **Piano di Progetto**: piano_di_progetto_vX.Y;
- **Piano di Qualifica**: piano_di_qualifica_vX.Y;
- **Analisi dei Requisiti**: analisi_dei_requisiti_vX.Y;
- **Specifica Architetture**: specifica_architetturale_vX.Y;
- **Glossario**: glossario_vX.Y;
- **Manuale Utente**: manuale_utente_vX.Y;
- **Verbali Interni**: VI_YYYY-MM-DD;
- **Verbali Esterni**: VE_YYYY-MM-DD.

Dove X.Y è la versione del documento e YYYY-MM-DD è la data del *Verbale*.

3.1.7.2 Stile del testo

- **grassetto**: viene utilizzato per i titoli, i termini delle voci degli elenchi e in generale per le parole da enfatizzare fortemente;
- **corsivo**: viene utilizzato per il nome del gruppo *DeltaX*, il nome del progetto *Produlytics*, il nome dell’azienda proponente *Sanmarco Informatica S.p.A.*, i nomi dei documenti e più in generale altri nomi di prodotti;
- **maiuscolo**: viene utilizzato per gli acronimi, che vengono scritti con sole lettere maiuscole, le iniziali dei nomi dei documenti, i nomi dei ruoli dei membri del gruppo ed eventualmente per il nome di ciò a cui ci si riferirà successivamente con una sigla, per enfatizzare con quali lettere è stata composta la sigla.

3.1.7.3 Elenchi puntati

Le voci di ogni elenco iniziano per lettera minuscola e terminano con un punto e virgola “;” eccetto per l’ultima voce di un elenco, che termina con un punto “.”. Nel caso di un elenco di definizioni o nella forma “termine: descrizione” allora il termine, e non i “:” che lo seguono, va posto in grassetto, come già specificato nella sezione sullo stile del testo.

3.1.7.4 Tabelle

Nelle tabelle le righe sono colorate in modo alternato con il colore bianco e grigio chiaro per facilitarne la lettura. Il colore dell’intestazione della tabella è un grigio più scuro per risaltare rispetto alle righe del contenuto. I grigi devono avere abbastanza contrasto con il colore nero del testo e tra di essi. Il template fornisce un ambiente `tabella` che applica automaticamente questo stile alle tabelle create con esso.

3.1.7.5 Formati delle date

Il formato per le date è conforme allo standard ISO 8601 ed è:

YYYY-MM-DD

dove:

- YYYY è il numero dell’anno con 4 cifre;
- MM è il numero del mese con 2 cifre;
- DD è il numero del giorno con 2 cifre.

3.1.8 Strumenti

Il gruppo come strumenti di supporto utilizza:

- **L^AT_EX**: linguaggio per la stesura dei documenti, che vengono compilati usando il software TeXLive;
- **Visual Studio Code**: editor di codice sorgente sviluppato da Microsoft. Grazie all’estensione LaTeX Workshop diventa un editor di testo per L^AT_EX;
- **Texmaker**: utilizzabile in alternativa a Visual Studio Code, è un editor L^AT_EX *open source* multiplatforma con un visualizzatore PDF integrato;
- **StarUML_G**: strumento UML di MKLab, è utilizzato per realizzare i diagrammi UML;
- **GitHub**: servizio di hosting per progetti software usato come spiegato in §3.2.4.2.

3.1.9 Metriche

Metrica	Nome	Riferimento
M10IG	Indice di Gulpease	§B.2 M10IG

Tabella 5: Metriche per la documentazione.

3.2 Gestione della configurazione

3.2.1 Descrizione

Questa sezione contiene le norme relative all'organizzazione e tracciabilità della documentazione e del codice prodotto.

3.2.2 Scopo

Lo scopo di questo processo è organizzare, coordinare e rendere tracciabili le modifiche effettuate alla documentazione e codice prodotto, così da facilitare le altre attività del progetto.

3.2.3 Codice di versione

Quando una verifica conferma una modifica si incrementa la versione del documento. Ogni versione di un documento è identificata univocamente da un codice di versione nel formato:

X.Y

dove:

- X: rappresenta la versione approvata dal Responsabile, che è anche colui che può autorizzare un suo incremento;
- Y: rappresenta la versione approvata dal Verificatore a seguito di modifiche del redattore.

Tutte le componenti di un codice di versione iniziano a 0 e ci ritornano ogni qualvolta una componente alla loro sinistra viene incrementata.

3.2.4 Tecnologie

3.2.4.1 Git

Per il versionamento del codice sorgente, sia del software che della documentazione, si usa il software di versionamento distribuito Git.

3.2.4.2 GitHub

Il coordinamento per il versionamento con il software Git viene effettuato sulla piattaforma GitHub.

3.2.5 Repository

3.2.5.1 Lista repository

Si utilizzano due repository:

- <https://github.com/DeltaXswe/documentazione-interna>, il repository privato per il codice sorgente della documentazione;
- <https://github.com/DeltaXswe/cc4d>, il repository pubblico per il codice sorgente del software e le versioni stabili della documentazione.

3.2.5.2 Gerarchia dei file

Il repository **documentazione-interna** è composto da una cartella per documento all'interno della quale è presente il codice sorgente per la compilazione di tale documento. È inoltre presente una cartella **template** contenente il codice sorgente per il template \LaTeX utilizzato e condiviso tra i vari documenti e una cartella **risorse** per eventuali risorse esterne utili alla stesura dei documenti.

Il repository **cc4d** è composto dalle seguenti cartelle e file:

- **Candidatura:** contenente il documento di candidatura del gruppo al progetto didattico;
- **RTB:** contenente i documenti da consegnare alla revisione RTB, quando saranno disponibili, ed è a sua volta diviso in:
 - **Documenti esterni:** contenente i documenti a uso esterno, ovvero:
 - * *Piano di Progetto v1.0.0*;
 - * *Piano di Qualifica v1.0.0*;
 - * *Analisi dei Requisiti v1.0.0*;
 - * *Glossario v1.0.0*;
 - * una cartella **Verbal**i contenente i *Verbal*i Esterni.
 - **Documenti interni:** contenente i documenti a uso interno, ovvero:
 - * *Norme di Progetto v1.0.0*;
 - * una cartella **Verbal**i contenente i *Verbal*i Interni.
- **PB:** contenente i documenti da consegnare alla revisione PB, quando saranno disponibili, ed è a sua volta diviso in:
 - **Documenti esterni:** contenente i documenti a uso esterno, ovvero:
 - * *Piano di Progetto v2.0*;
 - * *Piano di Qualifica v2.0*;
 - * *Analisi dei Requisiti v2.0*;
 - * *Specifica Architetture v1.0*;
 - * *Manuale Utente v1.0*;
 - * *Glossario v2.0*;
 - * una cartella **Verbal**i contenente i *Verbal*i Esterni.
 - **Documenti interni:** contenente i documenti a uso interno, ovvero:
 - * *Norme di Progetto v2.0*;
 - * una cartella **Verbal**i contenente i *Verbal*i Interni.
- **CA:** contenente i documenti da consegnare alla revisione CA, quando saranno disponibili, ed è a sua volta diviso in:
 - **Documenti esterni:** contenente i documenti a uso esterno, ovvero:
 - * *Piano di Progetto v3.0*;
 - * *Piano di Qualifica v3.0*;
 - * *Analisi dei Requisiti v3.0*;
 - * *Specifica Architetture v2.0*;
 - * *Manuale Utente v2.0*;
 - * *Glossario v3.0*;
 - * una cartella **Verbal**i contenente i *Verbal*i Esterni.

- **Documenti interni:** contenente i documenti a uso interno, ovvero:
 - * *Norme di Progetto v3.0*;
 - * una cartella **Verbal**i contenente i *Verbal*i Interni.
- **Proof of Concept:** contenente l'implementazione del *Proof of Concept*, quando sarà disponibile, per la revisione RTB;
- **Produlytics:** contenente il codice sorgente del prodotto, quando sarà disponibile.

3.2.6 Sincronizzazione

3.2.6.1 Branch

Le repository sono divise nei seguenti *branch_G*:

- **main:** contiene il codice e i documenti approvati dal Responsabile;
- **developing:** contiene il codice e i documenti approvati dai Verificatori;
- un branch per ogni *issue_G* da cui verranno aperte le pull request. Questi branch possono essere modificati da una sola persona per evitare potenziali conflitti.

3.2.6.2 Rebase

Per sincronizzare modifiche effettuate al branch main con i branch secondari si deve effettuare un rebase. Questo permette di mantenere intatta la cronologia del branch main ed evita commit di merge.

3.2.6.3 Pull Request

Quando un branch è pronto per la verifica, il suo creatore deve aprire una pull request, chiedendo quindi la sua integrazione nel branch main, che potrà avvenire solo dopo un esito positivo da parte dei test automatici e dal processo di verifica.

3.3 Gestione della qualità

3.3.1 Descrizione

Questa sezione contiene le norme relative alla gestione della qualità, cioè il processo atto a garantire la qualità di processi e prodotti, e atto alla soddisfazione delle aspettative del cliente e proponente. Il gruppo ambisce a ottenere un miglioramento continuo dei processi, pertanto adotta la metodologia PDCA, descritta in seguito.

3.3.2 Scopo

Lo scopo di questo processo è garantire che i processi e il prodotto rispettino degli obiettivi di qualità prefissati.

3.3.3 PDCA



Figura 1: Ciclo di Deming.

Il PDCA, anche detto *ciclo di Deming*, è un metodo iterativo per il controllo e il miglioramento continuo dei processi e dei prodotti. Per migliorare la qualità è necessario svolgere tutte e quattro le fasi che compongono l'acronimo costantemente. Le fasi sono:

3.3.3.1 Plan

La fase di pianificazione consiste nello stabilire gli obiettivi e i processi necessari per ottenere dei risultati coerenti con quanto atteso.

3.3.3.2 Do

L'attività di esecuzione consiste nello svolgere ciò che è stato deciso, nel modo deciso, nella fase precedente. Vengono inoltre misurati i risultati attraverso la raccolta di dati.

3.3.3.3 Check

L'attività di test e controllo consiste nello studio di ciò che emerso durante l'esecuzione. Ai dati misurati viene attribuito un significato grazie alle metriche e i risultati ottenuti vengono confrontati con le attese stabilite nella fase di "Plan".

3.3.3.4 Act

L'attività di azione consiste nel consolidare quanto di buono è emerso da ciò che è stato svolto e nell'attuare delle strategie correttive per migliorare laddove ci sono state discrepanze tra i risultati aspettati e quelli ottenuti, analizzando le cause. In questo modo il ciclo PDCA verrà raffinato e si otterranno risultati migliori all'iterazione successiva.

3.3.4 Strumenti

Per la gestione della qualità vengono usate le metriche.

3.3.5 Struttura delle metriche

Le metriche sono definite nell'appendice A, in degli elenchi con i seguenti punti:

- **codice:** il codice della metrica nel formato:

$M[\text{numero}] [\text{sigla}]$

dove:

- M sta per “*metrica_G*”;
 - [numero] è un numero progressivo univoco per ogni metrica;
 - [sigla] è una sigla composta dalle iniziali del nome della metrica.
- **nome:** il nome della metrica;
 - **descrizione:** cosa rappresenta;
 - **scopo:** il motivo per cui è importante misurarla;
 - ed eventualmente anche:
 - **formula:** come viene calcolata;
 - **strumento:** lo strumento che viene utilizzato per calcolarla.

Vengono inoltre riferite in ogni sezione del processo che vanno a misurare, in sottosezione “Metriche”.

3.3.6 Struttura degli obiettivi

Gli obiettivi per le metriche sono definiti nel *Piano di Qualifica v2.0* in tabelle con le seguenti colonne:

- **metrica:** il codice della metrica;
- **nome:** il nome della metrica;
- **valore accettabile:** il valore che deve assumere la metrica per poterla considerare soddisfatta;
- **valore preferibile:** il valore ideale che dovrebbe assumere la metrica.

3.3.7 Metriche

Metrica	Nome	Riferimento
M1PMS	Percentuale di Metriche Soddisfatte	§B.1 M1PMS
M5NRNP	Numero di Rischi Non Previsti	§B.1 M5NRNP
M6PRGM	Percentuale di Rischi Gestiti Male	§B.1 M6PRGM

Tabella 6: Metriche per il miglioramento.

3.4 Verifica

3.4.1 Scopo

Lo scopo del processo è determinare se i prodotti di una attività siano conformi con i vincoli e requisiti imposti nelle attività precedenti.

3.4.2 Aspettative

Un processo di Verifica ben istanziato garantisce correttezza ed efficienza delle attività.

3.4.3 Descrizione

Questo processo viene svolto dai Verificatori, che analizzano i prodotti di una attività per valutarne la conformità con i vincoli qualitativi specificati nel *Piano di Qualifica v2.0*. Le verifiche vengono fatte seguendo il *modello a V_G*, come illustrato nel *Piano di Qualifica v2.0*.

L'attività di Verifica deve essere documentata nel *Piano di Qualifica v2.0*, descrivendo la procedura adottata, il risultato sperato e il risultato ottenuto.

Le possibili attività di Verifica sono descritte di seguito:

3.4.4 Analisi statica

L'analisi statica è un tipo di analisi che non richiede alcun tipo di esecuzione. Questo tipo di analisi può anche essere applicato ai documenti. Due tecniche fondamentali di analisi sono *walkthrough_G* e *ispezione_G*.

3.4.4.1 Walkthrough

Il walkthrough è una tecnica che coinvolge un Verificatore e l'autore del prodotto, che può essere un qualsiasi altro ruolo. Questa tecnica si articola in:

1. **pianificazione**, alla quale partecipano sia il Verificatore che l'autore;
2. **lettura**: il Verificatore legge il prodotto (codice o documento) cercando errori, non conformità con le *Norme di Progetto* e sezioni migliorabili;
3. **discussione**, tra il Verificatore e l'autore, in cui il Verificatore riferisce tutto ciò che ha trovato e ritiene migliorabile all'autore;
4. **correzione**: se l'autore acconsente alle proposte, svolge le correzioni.

Ognuna di queste fasi viene documentata. La debolezza di questa tecnica è il non essere automatizzabile, inoltre la discussione non è formale e i criteri stessi di conformità possono essere fraintesi dalle due parti. Tuttavia, all'inizio del progetto non è possibile agire diversamente; pertanto, i membri del gruppo utilizzeranno questa pratica estensivamente nelle fasi iniziali del processo.

I membri del gruppo sono tenuti a documentare l'esecuzione del walkthrough con dei commenti nelle pull request associate su GitHub.

3.4.4.2 Ispezione

L'ispezione è una tecnica simile al walkthrough, che però adopera delle liste di controllo per effettuare controlli mirati agli errori più frequenti, invece di effettuare letture complete. Appena saranno disponibili liste di controllo sufficientemente grandi per assicurare una verifica completa, questa tecnica dovrà essere preferita al walkthrough in quanto più veloce. Per essere efficace è fondamentale che le liste di controllo siano basate su presupposti ben fondati.

I Verificatori sono tenuti a effettuare il walkthrough a campione e a valutare se gli errori trovati siano sufficientemente ricorrenti da dover essere inseriti nelle liste di controllo. Le liste di controllo sono riportate nel *Piano di Qualifica v2.0*.

3.4.5 Analisi dinamica

L'analisi dinamica è una categoria di tecniche di analisi che richiedono che il prodotto sia in esecuzione, pertanto non sono applicabili ai documenti. L'analisi dinamica deve poter essere automatizzata e ripetibile. La tecnica principale di analisi dinamica è il test. Esistono diversi tipi di test; ogni tipo di test può rispondere a delle esigenze maturate in una specifica fase dello sviluppo.

3.4.5.1 Test di unità

I test di unità sono definiti dai Verificatori durante la progettazione di dettaglio. Sono definiti in base alle specifiche di una *unità software_G*. Più test potranno essere associati a una singola unità, in tal caso formeranno una *test suite_G* per quella unità. I test di unità possono richiedere l'utilizzo di *stub_G* o *driver_G*, i quali permettono di testare le singole unità in ambienti simulati, il che è necessario nelle prime fasi dello sviluppo, quando l'intero ambiente non è ancora disponibile. I test di unità possono essere di due tipi, funzionali o strutturali, i Verificatori li adoperano entrambi, con strumenti automatici messi a disposizione dalla tecnologia usata.

Tipo	Descrizione	Vantaggi	Svantaggi
Funzionali (<i>black box_G</i>)	I test si basano sul comparare, dato un input, l'output effettivo con quello atteso.	Sono veloci da realizzare.	Non assicurano la copertura completa del flusso di esecuzione.
Strutturali (<i>white box_G</i>)	I test sono costruiti ciascuno per eseguire un flusso diverso. Una batteria di test deve assicurare copertura completa del codice in esame.	Assicurano che ogni flusso venga eseguito e valutato.	Se il codice ha una complessità ciclomantica elevata, l'implementazione diventa lunga e prona a errori umani.

Tabella 7: Tipi di test di unità.

3.4.5.2 Test di integrazione

I test di integrazione vengono definiti dai Verificatori durante la fase di progettazione architetturale e sono specifici per identificare errori nella specifica e nei test di unità¹. Essi si applicano alle singole componenti dell'architettura, verificando che le loro interfacce siano coerenti con la specifica, valutando ogni possibile flusso di dati tra le parti. Ogni passo di integrazione è incrementale e reversibile, ovvero è *baseline_G* per i passi successivi. L'integrazione può avvenire con approccio *bottom-up_G* o *top-down_G*.

¹Se i test di unità passano ma i test di integrazione no, e le interfacce sono coerenti con la specifica, allora significa che i test di unità non coprono tutti i casi possibili.

Tipo	Descrizione	Vantaggi	Svantaggi
bottom-up	L'integrazione parte dalle componenti del sistema che hanno più dipendenze entranti.	Richiedono pochi stub.	Ritarda l'implementazione delle funzionalità utente.
top-down	L'integrazione parte dalle componenti del sistema che hanno meno dipendenze entranti.	Le funzionalità con maggiore valore esterno vengono sviluppate prima.	Richiedono molti stub, i quali non incrementano le funzionalità.

Tabella 8: Tipi di test di integrazione.

3.4.5.3 Test di sistema

Questo tipo di test è effettuato alla fine dei test di integrazione, quando tutte le componenti sono state integrate nel sistema. I test di sistema sono preludio del collaudo, si concentrano sulla misura della copertura delle funzionalità, vengono infatti definiti dai Verificatori durante l'attività di analisi dei requisiti. Questo tipo di test viene svolto anche durante la validazione.

3.4.5.4 Test di regressione

I *test di regressione_G* si effettuano dopo la modifica di un componente del sistema e consistono nel ripetere quei test di unità, di integrazione e di sistema necessari per essere sicuri che dei cambiamenti a una componente non pregiudichino funzionalità già verificate, causando una *regressione_G*.

3.4.5.5 Codici identificativi dei test

Ogni test è associato a un codice identificativo nel formato **T[tipo][codice]**, dove:

- **[tipo]** è il tipo di test:
 - U per i test di unità;
 - I per i test d'integrazione;
 - S per i test di sistema;
 - A per i test di accettazione.
- **[codice]** è il codice identificativo del test all'interno del suo tipo:
 - se il test non ha padre, è un numero progressivo univoco tra quelli del suo tipo;
 - se il test ha un padre allora è nel formato:

[codice_padre].[numero_figlio]

dove:

- * **codice_padre** è il codice che identifica in maniera univoca il padre all'interno del suo tipo;
- * **numero_figlio** è un numero progressivo univoco tra i figli dello stesso padre.

3.4.5.6 Stato dei test

Ogni test è associato a uno stato che ne descrive il risultato. Esso può essere:

- **N/I**: non implementato;
- **S**: superato;
- **N**: non superato.

3.5 Validazione

3.5.1 Scopo

Lo scopo del processo è determinare se il prodotto finale rispetti i requisiti e i vincoli da contratto.

3.5.2 Aspettative

Se il prodotto viene approvato dal committente, il gruppo provvede alla consegna e alla chiusura del progetto.

3.5.3 Descrizione

Il processo di validazione si appoggia ai test di sistema, normati nella sezione del processo di verifica. A questo tipo di test se ne aggiunge uno ulteriore, ovvero il test di accettazione.

3.5.4 Test di accettazione

I test di accettazione vengono effettuati durante il collaudo, dimostrando in presenza del committente che il prodotto soddisfi i requisiti individuati. Un test di accettazione è composto da una sequenza di azioni che un attore deve poter soddisfare.

Prima di richiedere un collaudo, i Verificatori si preoccupano di eseguire la test suite di sistema in un ambiente identico a quello di installazione: è preconditione necessaria del collaudo che i test di sistema diano esito positivo.

3.5.5 Metriche

Metrica	Nome	Riferimento
M15PTUS	Percentuale di Test di Unità Superati	§B.2 M15PTUS
M16PTIS	Percentuale di Test di Integrazione Superati	§B.2 M16PTIS

Tabella 9: Metriche per il testing.

4 Processi organizzativi

4.1 Gestione organizzativa

4.1.1 Scopo

In questa sezione vengono espone le modalità e gli strumenti di coordinamento adottati dal gruppo per quanto riguarda la comunicazione interna ed esterna e l'assegnazione di ruoli e compiti ai membri del gruppo. Il processo di gestione organizzativa è strutturato come segue:

- comunicazione:
 - comunicazione interna;
 - comunicazione esterna.
- riunioni:
 - riunioni interne;
 - riunioni esterne.

4.1.2 Aspettative

Ciò che ci si aspetta da questo processo è:

- ricavare una pianificazione sulle attività da seguire;
- monitorare il gruppo, i processi e i prodotti;
- gestire il gruppo assegnando ruoli e compiti;
- facilitare la comunicazione fra i membri del gruppo;
- facilitare la comunicazione con gli esterni.

4.1.3 Descrizione

Le attività di gestione sono:

- assegnazione di ruoli e compiti ai componenti del gruppo;
- istanziazione dei processi;
- stima dei tempi, risorse e costi;
- esecuzione dei ruoli, compresa l'attività di controllo;
- monitoraggio, controllo e valutazione periodica delle attività.

4.1.4 Ruoli

Al fine di gestire al meglio i diversi compiti e attività da svolgere, il progetto stabilisce sei diversi ruoli con specifiche mansioni e responsabilità. Per l'intera durata del progetto, a ogni membro del gruppo viene attribuito un ruolo, con lo scopo di accumulare un significativo numero di ore per ciascuna figura aziendale, stabilito a priori nel *Piano di Progetto v2.0*. I ruoli vengono descritti di seguito:

4.1.4.1 Responsabile

Il Responsabile è il punto di riferimento per le comunicazioni con il committente e ha responsabilità decisionali di scelta e approvazione, costituisce il “centro di coordinamento” per l'intera durata del progetto. Rappresenta il gruppo di lavoro nei confronti del committente e del proponente. Deve avere competenze tecniche per valutare rischi e scelte alternative. Ha responsabilità su:

- la pianificazione;
- la gestione delle risorse umane;
- il coordinamento e le relazioni esterne;
- il controllo dei progressi del progetto;
- la cura delle relazioni esterne;
- l'approvazione della documentazione.

4.1.4.2 Amministratore

L'Amministratore si occupa dell'efficienza e dell'operatività dell'ambiente di sviluppo. Ha la gestione della configurazione del prodotto, del versionamento e della documentazione. Redige le *Norme di Progetto*. Inoltre, amministra le infrastrutture di supporto e risolve problemi legati alla gestione dei processi. Si occupa di:

- amministrare le risorse come: infrastrutture, strumenti e documentazione;
- risolvere problemi legati alla gestione dei processi;
- scrivere e aggiornare le regole e le procedure di lavoro;
- controllare il versionamento e la configurazione dei prodotti;
- assicurare la correttezza della documentazione.

4.1.4.3 Analista

L'Analista dovrebbe avere una grande conoscenza del campo ed è colui che cerca di capire il problema. Redige l'*Analisi dei Requisiti* e studia appieno il dominio del problema per comprenderlo al meglio. Non segue il progetto fino alla consegna.

4.1.4.4 Progettista

Si occupa di trovare una soluzione al problema con vincoli accettabili, effettua scelte tecniche e tecnologiche e segue lo sviluppo, non la manutenzione.

Si occupa di:

- sviluppare un'architettura seguendo un insieme di *best practices_G* per mantenere coerenza e consistenza;
- effettuare scelte per l'ottenimento di soluzioni affidabili, efficienti, sostenibili e che rispettino i requisiti;
- definire un'architettura logica facile da mantenere;
- decomporre il sistema in componenti e organizzarne le interazioni, i ruoli e le responsabilità per favorire la modularizzazione e il riutilizzo;

- sviluppare un'architettura robusta e sicura ai malfunzionamenti;
- definire una struttura che abbia un basso grado di accoppiamento;
- utilizzare soluzioni ottimizzate.

4.1.4.5 Programmatore

Ha competenze tecniche specifiche e si occupa di implementare la soluzione tramite l'attività di codifica. Concretizza la soluzione dei Progettisti e non è compito suo inventare e aggiungere nuovi elementi. Partecipa alla fase di realizzazione e manutenzione del prodotto. Si occupa di:

- scrivere codice versionato e manutenibile seguendo le norme fissate;
- creare le componenti di supporto per verifica e validazione del codice, implementando test ad hoc;
- redarre il *Manuale Utente*.

4.1.4.6 Verificatore

Verifica il lavoro svolto dagli altri membri. È presente per l'intera durata del progetto, ha capacità di giudizio e di relazione. Si assicura che le *Norme di Progetto* siano rispettate.

4.1.5 Gestione oraria

Ogni membro del gruppo è tenuto a segnare sul Google Sheet condiviso ogni giorno che svolge del lavoro, nella cella appartenente alla riga corrispondente al proprio nome e alla colonna corrispondente al ruolo ricoperto, il numero di ore effettive svolte. Questo tracciamento è essenziale per il calcolo del consuntivo di ogni fase del Piano di Progetto.

4.1.6 Gestione delle comunicazioni

4.1.6.1 Comunicazioni interne

Le comunicazioni interne avvengono tramite due principali strumenti: *Telegram_G* e *Discord_G*.

Telegram è un servizio di messaggistica istantanea che viene utilizzato per le comunicazioni veloci e più informali tra i membri. Discord è un'applicazione nella quale è possibile sia messaggiare che videochiamare, infatti viene utilizzata per le riunioni del gruppo in remoto. Consente inoltre di creare più canali di comunicazione. Per organizzare la discussione sono stati creati i canali:

- **capitolati:** per raccogliere quanto emerso durante gli incontri iniziali con le aziende, svolti per scegliere il capitolato su cui lavorare;
- **appunti:** dove appuntare subito quanto emerso durante una riunione interna. È la base per la stesura dei verbali;
- **github:** per le decisioni prese sull'organizzazione del repository su GitHub;
- **way-of-working:** per raccogliere proposte sugli strumenti da utilizzare per migliorare il way of working del gruppo.

È inoltre previsto che se il gruppo si dividerà in sottogruppi per svolgere dei lavori venga creato un canale specifico per ciascuno di essi, dove potersi coordinare autonomamente.

Nel caso in cui Telegram non fosse disponibile per malfunzionamenti, il gruppo si sposterà temporaneamente su Discord, anche per le comunicazioni più informali.

4.1.6.2 Comunicazioni esterne

La principale via di comunicazione con il proponente è l'applicazione *Google Meet_G*, con la quale vengono svolte le videochiamate. Le email vengono usate per fissare le riunioni e per porre domande brevi. L'indirizzo mail utilizzato sarà sempre e solo quello ufficiale del gruppo. Il gruppo sfrutta al meglio le riunioni scrivendo un elenco delle domande da porre al proponente, così da poter risolvere i dubbi sorti fino a quel momento in un'unica sede. Tutti i membri si impegnano a prendere parte a ogni riunione esterna.

4.1.7 Gestione degli incontri

4.1.7.1 Incontri interni

Le riunioni del gruppo sono organizzate dal Responsabile, in accordo con tutti gli altri membri del gruppo. La data e l'ora vengono segnate su Google Calendar, nello specifico nel calendario dell'account del gruppo, a cui tutti i membri hanno accesso.

4.1.7.2 Verbali di riunioni interne

Durante ogni riunione interna, il Responsabile nomina un Segretario che si occupa di trascrivere in sintesi il contenuto dell'incontro. Al termine dell'incontro il Responsabile nomina un Redattore e un Verificatore, il primo formalizza la bozza del Segretario, il secondo ne verifica successivamente il contenuto. La struttura del *Verbale* viene ampiamente descritta nella sezione dei processi di supporto.

4.1.7.3 Compiti del Responsabile

Il Responsabile deve occuparsi di:

- fissare la data delle riunioni interne;
- stabilire l'ordine del giorno;
- ascoltare e valutare tutte le richieste di ogni membro del gruppo, approvandole o rifiutandole;
- verificare e approvare il *Verbale* redatto dal Redattore.

4.1.7.4 Doveri dei partecipanti

I partecipanti hanno il dovere di:

- arrivare puntuali alle riunioni;
- comunicare al Responsabile, il prima possibile, eventuali ritardi o assenze;
- partecipare attivamente agli argomenti dell'ordine del giorno;
- mantenere un comportamento corretto durante tutta la riunione.

4.1.7.5 Approvazione delle decisioni

Una decisione viene considerata presa se approvata dalla maggioranza del gruppo e dal Responsabile. Una riunione viene considerata valida e ufficiale se sono presenti almeno cinque sui sette componenti del gruppo.

4.1.7.6 Tracciamento delle decisioni

Ogni decisione approvata viene tracciata all'interno del verbale come descritto in §3.1.6.7.

4.1.7.7 Incontri esterni del gruppo

È compito del Responsabile organizzare gli incontri con il proponente e il committente. Deve decidere una data in accordo tra le due parti, che poi comunicherà a tutto il gruppo.

4.1.7.8 Verbali di riunioni esterne

Anche in questo caso, come per le riunioni interne, viene redatto un *Verbale* con le stesse modalità.

4.1.8 Gestione degli strumenti di coordinamento

4.1.8.1 Ticketing

Il ticketing è un ottimo strumento che permette ai membri del gruppo di sapere in ogni momento quali sono le attività in corso. Il Responsabile può assegnare i compiti ai vari membri e monitorarne l'andamento. Lo strumento di ticketing scelto è il servizio di gestione delle issue integrato in GitHub. Una issue viene creata con lo scopo di tracciare idee, miglioramenti, compiti o problemi da risolvere. In generale, quando viene individuata un'attività da svolgere:

1. viene creata una issue associata;
 - se la issue è particolarmente grande viene scomposta in più issue.
2. una issue:
 - può essere risolta tramite un branch apposito;
 - può essere svolta insieme a più issue in uno stesso branch.
3. i Verificatori verificano quanto è stato svolto;
 - se la verifica ha esito positivo:
 - (a) viene fatto il merge del branch nell'upstream;
 - (b) viene cancellato il branch;
 - (c) l'issue viene chiusa;
 - (d) il requisito è considerato soddisfatto.
 - se la verifica ha esito negativo:
 - (a) vengono corretti gli errori segnalati dai Verificatori e si torna al punto (3).

Le issue sono create dal Responsabile oppure dai Verificatori e hanno:

- **title:** un titolo, che deve essere conciso e significativo;
- **description:** una descrizione, che deve essere il più breve possibile, chiara e specificare i motivi dell'apertura della issue;
- **assignee** (opzionale): il membro o i membri a cui è stata assegnata la issue. Potrebbe variare durante il progetto;
- **milestone_G:** la milestone associata;
- **labels:** le etichette, che specificano lo stato della issue (dopo la creazione è "open").

4.1.9 Gestione dei rischi

È compito del Responsabile, insieme agli Amministratori, individuare eventuali rischi e documentarli nel *Piano di Progetto*. Dopo la loro individuazione, è necessario definire una o più strategie per la gestione dei rischi.

4.1.9.1 Struttura dei rischi

I rischi sono divisi in tre tipologie:

- rischi tecnologici;
- rischi organizzativi;
- rischi comunicativi.

Ogni rischio è formato da:

- intestazione, nel formato:

`R[tipo][numero] - [nome]`

dove:

- `[tipo]` è la tipologia del rischio, cioè:
 - * `T` se il rischio è tecnologico;
 - * `O` se il rischio è organizzativo;
 - * `C` se il rischio è comunicativo.
 - `[numero]` è un numero progressivo univoco per ogni rischio della sua tipologia;
 - `[nome]` è il nome del rischio.
- descrizione;
 - probabilità di manifestazione;
 - pericolosità;
 - metodi di rilevamento;
 - piano di contingenza.

4.1.10 Strumenti

Il gruppo nel corso dello sviluppo del progetto utilizza:

- **Telegram**, come spiegato in §4.1.6.1;
- **Discord**, come spiegato in §4.1.6.1;
- **Google Gmail**, per la mail del gruppo come spiegato in §4.1.6.2;
- **Google Sheet**, come spiegato in §4.1.5;
- **Google Meet**, come spiegato in §4.1.6.2;
- **Google Calendar**, come spiegato in §4.1.7.1;
- **GitHub**, come spiegato in §4.1.8.1.

4.2 Formazione

4.2.1 Scopo

Lo scopo di questo processo è definire le norme riguardanti la formazione dei membri del gruppo *DeltaX* sulle tecnologie richieste per la produzione dei documenti e la costruzione del prodotto richiesto.

4.2.2 Aspettative

Il processo deve garantire buone conoscenze su tutte le tecnologie necessarie alla produzione dei documenti e al completamento del prodotto.

4.2.3 Formazione dei membri del gruppo

I membri del gruppo *DeltaX* provvedono in modo autonomo allo studio delle varie tecnologie scelte per la costruzione del prodotto. Per facilitare la formazione, i membri del gruppo sono tenuti a condividere eventuali conoscenze già possedute.

4.2.4 Guide e documentazione

Di seguito una lista delle guide e della documentazione da prendere come riferimento per la formazione di ogni membro del gruppo *DeltaX*:

- per \LaTeX : <https://www.latex-project.org/>;
- per *TypeScript*_G: <https://www.typescriptlang.org/docs/>;
- per *Angular*_G: <https://angular.io/docs>;
- per *Java*_G: <https://docs.oracle.com/en/java/>;
- per *Spring*_G: <https://docs.spring.io/spring-framework/docs/current/reference/html/>;
- per *PostgreSQL*_G: <https://www.postgresql.org/docs/current/>;
- per *TimeScale*_G: <https://docs.timescale.com/>;
- per GitHub: <https://docs.github.com/en>.

Ogni membro del gruppo è tuttavia libero, qualora lo ritenesse necessario, di utilizzare altro materiale diverso da quello sopra menzionato ai fini della propria formazione e, eventualmente, di condividerlo con gli altri membri.

A Standard per la qualità

Il concetto di qualità del software si è evoluto nel tempo, includendo esigenze diverse ai fini del corretto funzionamento del prodotto e del suo utilizzo da parte degli utenti; pertanto è importante valutare oggettivamente e in modo preciso il livello di qualità del prodotto software, attraverso la definizione di caratteristiche e metriche. Ogni caratteristica rilevante del prodotto è quindi misurata applicando metriche adeguate e valutata in base a intervalli di accettabilità e valori soglia stabiliti.

Le norme ISO/IEC 9126 descrivono un modello di qualità del software, le cui caratteristiche e metriche vengono prese come riferimento dal gruppo *DeltaX* e vengono riportate di seguito.

A.1 Funzionalità

È la caratteristica che rappresenta la capacità del software di fornire le funzioni, esplicite e non, necessarie per operare in un certo contesto.

I suoi attributi sono:

A.1.1 Adeguatezza

È la capacità del software di fornire un appropriato insieme di funzioni che permettano agli utenti di svolgere determinati task e il raggiungimento di obiettivi prefissati.

A.1.2 Accuratezza

È la capacità del software di fornire i risultati attesi con la precisione richiesta.

A.1.3 Interoperabilità

È la capacità del software di fornire i risultati attesi con la precisione richiesta.

A.1.4 Sicurezza

È la capacità del software di proteggere le informazioni e i dati in modo che entità non autorizzate non possano accedervi.

A.1.5 Aderenza alle funzionalità

È la capacità del software di aderire a standard, convenzioni e regolamenti di carattere legale relative alle funzionalità.

A.2 Affidabilità

È la caratteristica che rappresenta la capacità di un prodotto software di mantenere il livello di prestazione quando usato in condizioni specificate.

I suoi attributi sono:

A.2.1 Maturità

È la capacità del software di evitare che si verifichino errori o che vengano prodotti risultati sbagliati in esecuzione.

A.2.2 Tolleranza ai guasti

È la capacità del software di mantenere un buon livello di funzionalità anche in caso di errori.

A.2.3 Recuperabilità

È la capacità del software di ripristinare il livello di prestazioni e di recuperare i dati in caso di errori o malfunzionamenti. Valuta anche il periodo di tempo per cui il software è inaccessibile a seguito di un guasto.

A.2.4 Aderenza all'affidabilità

È la capacità del software di aderire a standard, convenzioni e regole relative all'affidabilità.

A.3 Usabilità

È la caratteristica che rappresenta la capacità di un prodotto software di essere comprensibile. Chiarisce tutti gli ambienti e scenari di utilizzi del prodotto, inclusa la preparazione all'utilizzo del software e la valutazione dei risultati.

I suoi attributi sono:

A.3.1 Comprensibilità

È la capacità del software di consentire all'utente di capire le funzionalità di cui dispone e come poterle usare per raggiungere con successo gli obiettivi di un certo task, in specifiche condizioni di utilizzo. Dipende dalla documentazione disponibile e dalla prima impressione data dalla ricezione del prodotto.

A.3.2 Apprendibilità

È la capacità del software di permettere all'utente di imparare l'applicazione.

A.3.3 Operabilità

È la capacità del software di permettere all'utente di usarlo e controllarlo. Influiscono tutti gli attributi della funzionalità, la modificabilità, adattabilità e installabilità.

A.3.4 Attrattività

È la capacità del software di risultare attraente all'utente. Influiscono l'aspetto grafico delle interfacce: i colori, le immagini, la disposizione, ecc.

A.3.5 Aderenza all'usabilità

È la capacità del software di aderire a standard, convenzioni e regole relative all'usabilità.

A.4 Efficienza

È la caratteristica che rappresenta la capacità di un prodotto software di realizzare le funzioni richieste nel minor tempo possibile e utilizzando nel miglior modo le risorse necessario, quando opera in determinate condizioni.

I suoi attributi sono:

A.4.1 Comportamento rispetto al tempo

È la capacità del software di possedere tempi di risposta, di elaborazione e quantità di lavoro appropriati eseguendo le funzionalità previste sotto determinate condizioni di utilizzo.

A.4.2 Utilizzo delle risorse

È la capacità del software di utilizzare un numero e tipo appropriato di risorse quando svolge le funzionalità previste, in specifiche condizioni di utilizzo.

A.4.3 Aderenza all'efficienza

È la capacità del software di aderire a standard e convenzioni relative all'efficienza.

A.5 Manutenibilità

È la caratteristica che rappresenta la capacità di un prodotto essere modificabile, sia per correzioni che per adattare il software a modifiche all'ambiente o ai requisiti.

I suoi attributi sono:

A.5.1 Analizzabilità

È la capacità del software di rendere individuabili le cause degli errori attraverso l'analisi.

A.5.2 Modificabilità

È la capacità del software di essere modificabile, in termini di modifiche al codice, alla progettazione o alla documentazione.

A.5.3 Stabilità

È la capacità del software di non generare effetti indesiderati a seguito di modifiche.

A.5.4 Provabilità

È la capacità del software di consentire verifica e validazione, attraverso i test, del software modificato.

A.5.5 Aderenza alla manutenibilità

È la capacità del software di aderire a standard e convenzioni relative alla manutenibilità.

A.6 Portabilità

È la caratteristica che rappresenta la capacità di un prodotto essere trasportato a ambienti diversi, quindi con organizzazione e tecnologie diverse.

I suoi attributi sono:

A.6.1 Adattabilità

È la capacità del software di essere adattabile a ambienti diversi senza azioni specifiche. Include la scalabilità.

A.6.2 Installabilità

È la capacità del software di essere installato in un determinato ambiente.

A.6.3 Coesistenza

È la capacità del software di coesistere con altre applicazioni indipendenti in ambienti comuni e di condividersi risorse.

A.6.4 Sostituibilità

È la capacità del software di sostituire un altro software specifico e indipendente, per lo stesso scopo e nello stesso ambiente.

A.6.5 Aderenza alla portabilità

È la capacità del software di aderire a standard e convenzioni relative alla portabilità.

B Metriche per la qualità

B.1 Metriche per la qualità di processo

I valori delle metriche vengono resettati a ogni incremento del progetto, per garantire una visione sull'andamento priva di condizionamenti da quelli precedenti.

- **M1PMS:**
 - **nome:** Percentuale di Metriche Soddisfatte;
 - **descrizione:** percentuale che rappresenta le metriche che rientrano nei range di accettabilità e quindi anche di preferibilità;
 - **scopo:** massimizzarla per perseguire processi di qualità;
 - **formula:** $\frac{\text{metriche soddisfatte}}{\text{metriche totali}} \cdot 100$.
- **M2VP:**
 - **nome:** Variazione di Piano;
 - **descrizione:** numero rappresentante il numero di giorni di differenza rispetto alla pianificazione. Se:
 - * è > 0 si è in anticipo;
 - * è 0 si è in linea;
 - * è < 0 si è in ritardo.
 - **scopo:** controllare come si procede rispetto alla pianificazione ed evitare ritardi;
 - **formula:** $(FP - IP) - (FC - IC)$, dove:
 - * FP (Fine Pianificata) è il giorno pianificato di fine attività;
 - * IP (Inizio Pianificato) è il giorno pianificato di inizio attività;
 - * FC (Fine Consuntivata) è il giorno consuntivato di fine attività;
 - * IC (Inizio Consuntivato) è il giorno consuntivato di inizio attività.
- **M3VC:**
 - **nome:** Variazione di Costo;
 - **descrizione:** numero rappresentante lo stato dei costi rispetto al preventivo. Se:
 - * è > 0 si è speso di più;
 - * è 0 si è in linea;
 - * è < 0 si è speso di meno.
 - **scopo:** controllare come si procede rispetto al preventivo ed evitare rincari;
 - **formula:** $CAS - CAP$, dove:
 - * CAS (Costo Attività Svolte) sono i costi delle attività svolte finora;
 - * CAP (Costo Attività Pianificate) sono i costi delle attività che dovrebbero essere state svolte finora.
- **M4VR:**
 - **nome:** Variazione dei Requisiti;
 - **descrizione:** numero rappresentante la variazione dei requisiti durante il progetto;
 - **scopo:** misurare eventuali discostamenti dall'analisi dei requisiti, in modo da adattare la pianificazione di conseguenza;

- **formula:** $NRA + NRR + NRM$, dove:
 - * NRA (Numero Requisiti Aggiunti) è la quantità di requisiti aggiunti dall'ultimo incremento;
 - * NRR (Numero Requisiti Rimossi) è la quantità di requisiti rimossi dall'ultimo incremento;
 - * NRM (Numero Requisiti Modificati) è la quantità di requisiti modificati dall'ultimo incremento.
- **M5NRNP:**
 - **nome:** Numero di Rischi Non Previsti;
 - **descrizione:** numero che rappresenta la quantità di rischi verificati e non previsti dall'analisi dei rischi;
 - **scopo:** minimizzarlo per perseguire un'attività di analisi dei rischi efficace.
- **M6PRGM:**
 - **nome:** Percentuale di Rischi Gestiti Male;
 - **descrizione:** percentuale che rappresenta la quantità di rischi verificati che sono stati gestiti male;
 - **scopo:** minimizzarla per perseguire un buon piano di contingenza per i rischi.
- **M7CCM:**
 - **nome:** Complessità Ciclomatica Media;
 - **descrizione:** numero razionale positivo rappresentante, la media della complessità ciclomatica di ogni metodo;
 - **scopo:** ridurla limita i test case, semplificando il testing;
 - **formula:** $\sum_{metodo} \frac{e - n + 2}{numero\ metodi}$, dove:
 - * e è il numero di archi del grafo del flusso di esecuzione del metodo;
 - * n è il numero di vertici del grafo del flusso di esecuzione del metodo;
 - * $numero\ metodi$ è il numero totale di metodi.
- **M8SC:**
 - **nome:** Statement Coverage;
 - **descrizione:** percentuale che rappresenta gli statement eseguiti durante i test;
 - **scopo:** massimizzarla per ottenere un processo di testing che apporti valore aggiunto;
 - **formula:** $\frac{statement\ eseguiti}{statement\ totali} \cdot 100$.
- **M9BC:**
 - **nome:** Branch Coverage;
 - **descrizione:** percentuale che rappresenta i branch eseguiti durante i test;
 - **scopo:** massimizzarla per ottenere un processo di testing che apporti valore aggiunto;
 - **formula:** $\frac{branch\ eseguiti}{branch\ totali} \cdot 100$.

B.2 Metriche per la qualità di prodotto

- **M10IG:**
 - **nome:** Indice di Gulpease;
 - **descrizione:** numero intero positivo rappresentante la leggibilità di un testo italiano. Se è:
 - * < 80 il testo è difficile da leggere per chi ha la licenza elementare;
 - * < 60 il testo è difficile da leggere per chi ha la licenza media;
 - * < 40 il testo è difficile da leggere per chi ha la licenza superiore.
 - **scopo:** massimizzarlo per rendere i documenti prodotti di facile fruizione;
 - **formula:** $89 + \frac{300 \cdot (N_f - 10 \cdot N_l)}{N_p}$, dove:
 - * N_f è il numero delle frasi contenute nel documento;
 - * N_l è il numero delle lettere;
 - * N_p è il numero delle parole.
 - **strumento:** si utilizza uno strumento di calcolo online:
https://farfalla-project.org/readability_static/.
- **M11PRCI:**
 - **nome:** Percentuale di Requisiti di Configurazione Implementati;
 - **descrizione:** percentuale rappresentante la quantità dei requisiti di configurazione implementati;
 - **scopo:** misurare la progressione della codifica prima dell'integrazione;
 - **formula:** $\frac{\text{requisiti di configurazione implementati}}{\text{requisiti di configurazione totali}} \cdot 100$.
- **M12PMG:**
 - **nome:** Profondità Media delle Gerarchie;
 - **descrizione:** numero razionale positivo rappresentante, per ogni gerarchia di classi, il massimo numero di classi parenti. Una gerarchia composta da una sola classe ha profondità 0;
 - **scopo:** limitarla, in quanto una gerarchia troppo profonda complica testing e manutenzione, soprattutto se si modifica una classe alla base.
- **M13LMCM:**
 - **nome:** Linee Medie di Codice per Metodo;
 - **descrizione:** numero razionale positivo rappresentante, in media, il numero di linee di codice che ogni metodo ha, escludendo le linee vuote o composte solo da parentesi chiuse;
 - **scopo:** limitarlo, in quanto metodi troppi lunghi potrebbero star violando il single responsibility principle, oltre che essere meno comprensibili.
- **M14LCC:**
 - **nome:** Linee di Commenti per Codice;
 - **descrizione:** numero razionale positivo rappresentante il numero di linee di codice che ci sono rapportato al numero di linee di commenti, escludendo le linee vuote;
 - **scopo:** assicurare che ci siano abbastanza commenti a supporto della lettura del codice, particolarmente utili per perseguirne la manutenibilità;
 - **formula:** $\frac{\text{linee di codice}}{\text{linee di commento}}$.

- **M15PTUS:**
 - **nome:** Percentuale di Test di Unità Superati;
 - **descrizione:** percentuale rappresentante la quantità di test di unità implementati che restituiscono esito positivo;
 - **scopo:** ottenere un prodotto comprovatamente robusto e maturo;
 - **formula:** $\frac{\text{numero di test di unità implementati superati}}{\text{numero di test di unità implementati}} \cdot 100$.
- **M16PTIS:**
 - **nome:** Percentuale di Test di Integrazione Superati;
 - **descrizione:** percentuale rappresentante la quantità di test di integrazione implementati che restituiscono esito positivo;
 - **scopo:** ottenere un prodotto comprovatamente robusto e maturo;
 - **formula:** $\frac{\text{numero di test di integrazione implementati superati}}{\text{numero di test di integrazione implementati}} \cdot 100$.