Università degli Studi di Padova

# KaleidosCode

# SWEDesigner

## Software per diagrammi UML

## Piano di qualifica v1.0.0


**Informazioni sul documento**

| | |
|---|---|
| **Versione** | 1.0.0 |
| **Data Redazione** | 09/03/2017 |
| **Redazione** | Bonato Enrico |
| | Bonolo Marco |
| | Pace Giulio |
| | Sovilla Matteo |
| **Verifica** | Pezzuto Francesco |
| **Approvazione** | Sanna Giovanni |
| **Uso** | Esterno |
| **Distribuzione** | *Prof. Vardanega Tullio* |
| | *Prof. Cardin Riccardo* |
| | *Zucchetti s.p.a.* |

kaleidos.codec6@gmail.com

# Diario delle Modifiche

| Versione | Data | Autore | Descrizione |
|---|---|---|---|
| 1.0.0 | 31/03/2017 | Sanna Giovanni | Approvazione Documento |
| 0.2.0 | 29/03/2017 | Pezzuto Francesco | Verifica Documento |
| 0.1.1 | 26/03/2017 | Bonolo Marco | Incremento documento |
| 0.1.0 | 15/03/2017 | Pezzuto Francesco | Verifica Documento |
| 0.0.5 | 10/03/2017 | Bonolo Marco | Stesura parte capitolo 4 |
| 0.0.4 | 10/03/2017 | Sovilla Matteo | Stesura parte capitolo 3 e parte capitolo 4 |
| 0.0.3 | 09/03/2017 | Bonato Enrico | Stesura parte capitolo 3 |
| 0.0.2 | 09/03/2017 | Pace Giulio | Stesura capitolo 2 |
| 0.0.1 | 08/03/2017 | Bonolo Marco | Creazione scheletro del documento e stesura della sezione Introduzione |

# Indice

# 1 Introduzione

## 1.1 Scopo del documento

Questo documento definisce gli obbiettivi e le metodologie che ogni membro del gruppo *KaleidosCode* adotterà per garantire un determinato livello di qualità del prodotto. A tal proposito ogni membro del gruppo è tenuto a leggere, perseguire e raggiungere gli obbiettivi definiti in esso.

## 1.2 Scopo del prodotto

Lo scopo del progetto è la realizzazione di un software di costruzione di diagrammi $\text{UML}_{\text{G}}$ con la relativa generazione di codice $\text{Java}_{\text{G}}$ e $\text{Javascript}_{\text{G}}$ utilizzando tecnologie web. Il prodotto deve essere conforme ai vincoli qualitativi richiesti dal committente.

## 1.3 Glossario

Al fine di evitare ogni ambiguità di linguaggio e massimizzare la comprensione dei documenti, i termini tecnici, di dominio, gli acronimi e le parole che necessitano di essere chiarite, sono riportate nel documento *Glossario v1.0.0*.
Ogni occorrenza di vocaboli presenti nel *Glossario* è marcata da una "G" maiuscola in pedice.

## 1.4 Riferimenti utili

### 1.4.1 Riferimenti normativi

- **Capitolato d'appalto**:
  http://www.math.unipd.it/~tullio/IS-1/2016/Progetto/C6.pdf (09/03/2017).

### 1.4.2 Riferimenti informativi

- **Qualità del software (Slide del Corso di Ingegneria del Software)**:
  http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/L10.pdf (09/03/2017);
- **Qualità di Processo (Slide del Corso di Ingegneria del Software)**:
  http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/L11.pdf (09/03/2017);
- **Glossario**: *Glossario v1.0.0*.

# 2 Definizione obiettivi di qualità

Prendendo come riferimento lo standard [ISO/IEC 9126] e lo standard [ISO/IEC 12207] il team si impegna a garantire che SWEDesigner abbia le seguenti qualità:

## 2.1 Funzionalità

Si garantisce che il sistema prodotto abbia tutte le funzionalità che il documento *Analisi dei requisiti v1.0.0* indica. L'implementazione di ogni requisito deve essere quanto più completa ed economica.

- **Misura**: l'unità di misura utilizzata sarà la quantità di requisiti mappati in componenti del sistema create e funzionanti;
- **Metrica**: la sufficienza è raggiunta quando vengono soddisfatti tutti i requisiti obbligatori;
- **Strumenti**: il sistema deve superare tutti i test previsti dai dalla documentazione prodotta e consegnata in sede di Revisione dei Requisiti.

## 2.2 Affidabilità

Il sistema deve essere quanto più possibile robusto. Nel caso di eventuali errori deve essere di facile ripristino.

- **Misura**: l'unità di misura utilizzata sarà la quantità di esecuzioni che vanno a buon fine;
- **Metrica**: visto che non è possibile valutare a monte tutte le possibili casistiche di utilizzo le esecuzioni dovranno il più possibile coprire la possibile gamma di possibilità. Per questo motivo è impossibile stabilire oggettivamente una esatta soglia che corrisponda alla sufficienza;
- **Strumenti**: ancora da definire.

## 2.3 Usabilità

Il sistema deve risultare per quanto possibile intuitivo e di facile utilizzo. Deve coniugare una facilità di apprendimento e utilizzo con il soddisfacimento di tutte le necessità dell'utente.

- **Misura**: poichè non esiste una metrica oggettiva che riguarda questo ambito l'unità di misura utilizzata sarà una valutazione soggettiva dell'usabilità;
- **Metrica**: non esistendo una metrica oggettiva è impossibile determinare con certezza quale sia la sufficienza. In ogni caso i membri del gruppo si impegneranno a garantire un'usabilità più alta possibile;
- **Strumenti**: si vedano le *Norme di progetto v1.0.0* .

## 2.4 Efficienza

Il sistema deve ridurre al minimo l'utilizzo delle risorse impiegate e deve fornire le funzionalità richieste nel minor tempo possibile.

- **Misura**: il tempo di latenza dell'editor in seguito a un comando;
- **Metrica**: la sufficienza viene definita come un tempo di latenza inferiore ai 2 secondi;
- **Strumenti**: si vedano le *Norme di progetto v1.0.0* .

## 2.5 Manutenibilità

Il sistema deve essere più possibile estensibile e comprensibile.

- **Misura**:l'unità di misura utilizzata sarà quella descritta nella sezione "Metriche per il codice"
- **Metrica**: il prodotto deve avere la sufficienza in tutte le metriche descritte nella sezione "Metriche per il codice".
- **Strumenti**: si vedano le *Norme di progetto v1.0.0* .

## 2.6 Portabilità

Il sistema deve essere più portabile possibile. Il front end$_G$ dovrà funzionare correttamente su più browser possibile. Inoltre dovrà essere supportato da più sistemi operativi possibili.

- **Misura**: il front end deve rispettare gli standard W3C$_G$;
- **Metrica**: Il software dovrà avere le caratteristiche di portabilità descritte. Per questo motivo sarà necessario raggiungere la sufficienza in tutte le metriche descritte nella sezione "Metriche per il codice";
- **Strumenti**: si vedano le *Norme di progetto v1.0.0* .

## 2.7 Altre qualità

Saranno importanti per la qualità del progetto anche i seguenti aspetti:

- **incapsulamento**: un buon livello di incapsulamento è preferibile in quanto aumenta la riusabilità e la manutenibilità del codice. A questo scopo saranno quindi utilizzate interfacce dove possibile
- **coesione**: le funzionalità che concorrono a uno stesso obiettivo devono risiedere nello stesso componente in modo da favorire semplicità e manutenibilità. In questo modo viene inoltre ridotto l'indice di dipendenza.

# 3 Visione generale della strategia

Per garantire la qualità dei prodotti realizzati durante lo sviluppo del progetto, è indispensabile definire e perseguire strategie che assicurino la qualità dei processi adottati, nonchè il loro continuo miglioramento; inoltre, è necessario definire metriche e pianificare attività che valutino in modo preciso la qualità dei prodotti ottenuti e dei processi adottati. A tal scopo, verranno adottate le seguenti strategie:

- Definizione accurata di norme che regolamentano e standardizzano i processi coinvolti nel progetto, in termini di:
  - Processi di fornitura;
  - Processi di sviluppo;
  - Processi di supporto;
  - Processi organizzativi;
- Descrizione dettagliata delle strategie di pianificazione adottate per sviluppo del progetto, in termini di:
  - Modello di sviluppo adottato;
  - Analisi dei rischi che si possono incontrare;
  - Pianificazione delle attività e dei tempi;
  - Stima preventiva delle risorse che saranno impiegate;
  - Assegnazione delle risorse, al fine di portare a termine le attività pianificate nei tempi previsti;
  - Consultivo, durante lo sviluppo del progetto, delle risorse impiegate;
- Ad ogni processo coinvolto nello sviluppo del progetto verrà applicato il principio $\text{PDCA}_{\text{G}}$, affiancato dal modello $\text{CMM}_{\text{G}}$. Essi permettono il controllo, la valutazione e il miglioramento continuo dei processi, nonchè la determinazione del livello di maturità dell'organizzazione nel gestire tali processi.

## 3.1 Organizzazione

La gestione della strategia di verifica si basa sull'attuazione delle relative attività descritte nelle *Norme di progetto* . Tali attività vengono eseguite per ogni processo attuato, allo scopo di verifica della qualità del processo stesso e dell'eventuale prodotto ottenuto, facendo riferimento anche alle metriche definite nel presente documento [..]. Ogni documento prevede un diario delle modifiche che permette di concentrare l'attività di verifica solo nelle parti modificate dopo l'ultima verifica eseguita. Data la diversa natura dei prodotti ottenuti dalle fasi del progetto, si necessita, per ognuna di esse, una diversa procedura di verifica:

- **Analisi**: I metodi di verifica utilizzati per questa fase sono descritti nelle *Norme di progetto* ;

- **Analisi di dettaglio**: in tale fase verranno verificati i processi che porteranno all'incremento dei prodotti realizzati nella fase precedente; verrà inoltre garantito che tutti i requisiti possano essere rintracciabili. I metodi utilizzati per la verifica di questa fase saranno descritti nelle *Norme di progetto* e incrementati nelle fasi successive.
- **Progettazione e codifica**: in tale fase verranno verificati i processi che porteranno all'incremento dei prodotti realizzati nella fase precedente. L' attività di verifica per questa fase prevede l'esecuzione di test pianificati, come in [???]. I metodi utilizzati per la verifica di questa fase saranno descritti nelle *Norme di progetto* e incrementati nelle fasi successive.

## 3.2 Scadenze temporali

Dato l'obiettivo di rispettare le scadenze fissate nel *Piano di progetto* è indispensabile pianificare l'attività di verifica della documentazione e del codice prodotto, in modo che risulti sistematica e organizzata; grazie all'applicazione di tale strategia, l'individuazione e la correzione degli errori averrà il prima possibile, impedendo la loro rapida diffusione e mitigando la possibilità che gli stessi si ripresentino in futuro; diminuendo così il rischio di ritardi. Tale pianificazione è documentata nel *Piano di progetto* il quale contiene, nella sottosezione 1.4, anche le scadenze temporali che il gruppo *KaleidosCode* si impegna a rispettare.

## 3.3 Responsabilità

I ruoli responsabili delle attività di verifica sono il *Responsabile di progetto* e il *Verificatore*. I loro compiti e responsabilità, descritti nelle *Norme di progetto*, permettono alle attività di verifica di essere efficienti e sistematiche.

# 4 La strategia di gestione della qualità nel dettaglio

## 4.1 Risorse

### 4.1.1 Necessarie

Per la realizzazione del prodotto sono necessarie le risorse umane e tecnologiche citate di seguito.

- **Risorse umane**: sono descritte dettagliatamente nel *Piano di progetto*.
  - *Responsabile di progetto*;
  - *Amministratore*;
  - *Analista*;
  - *Progettista*;
  - *Programmatore*;
  - *Verificatore*.
- **Risorse software**: sono descritte dettagliatamente nelle *Norme di progetto*. Si tratta di software che permettano:
  - la comunicazione e la condivisione del lavoro tra gli elementi del team;
  - la stesura della documentazione in formato LaTeX;
  - la creazione di diagrammi UML;
  - la codifica nei linguaggi di programmazione scelti;
  - la semplificazione delle attività di verifica;
  - la gestione dei test sul codice.
- **Risorse hardware**: ciascun componente del gruppo ha bisogno di un computer con tutti i software necessari. È necessario avere a disposizione almeno un luogo dove poter effettuare le riunioni del team.

### 4.1.2 Disponibili

Ogni membro del team ha a disposizione uno o più computer personali dotati degli strumenti necessari.
Le riunioni interne si svolgono presso le aule del dipartimento di Matematica dell'Università degli Studi di Padova.

## 4.2 Misure e metriche

Il processo di verifica deve essere quantificabile per fornire informazioni utili, bisogna quindi stabilire le metriche da adottare per le misurazioni. Si definiranno due intervalli di misure:

- **Range di accettazione**: intervallo di valori vincolante per l'accettazione del prodotto;

- **Range ottimale**: intervallo di valori entro cui è consigliabile rientri la misurazione. Il mancato rispetto di questa condizione non pregiudica l'accettazione del prodotto, ma richiede verifiche più approfondite in merito.

### 4.2.1 Metriche per i processi

**Schedule Variance**

È una metrica di progetto standard, indica se si è in linea, in anticipo o in ritardo rispetto alla schedulazione pianificata delle attività di progetto. È pari alla differenza tra il valore delle attività pianificate e il valore delle attività svolte alla data corrente.

**Parametri utilizzati**

- **Range di accettazione**: $\geq -(preventivo * 5\%)$;
- **Range ottimale**: $\geq 0$.

**Budget Variance**

È una metrica di progetto standard, indica se si spende di più o di meno rispetto a quanto preventivato alla data corrente. È pari alla differenza tra costo pianificato e costo effettivamente sostenuto alla data corrente.

**Parametri utilizzati**

- **Range di accettazione**: $\geq -(preventivo * 10\%)$;
- **Range ottimale**: $\geq 0$.

### 4.2.2 Metriche per i documenti

**Indice Gulpease**

Definito nel 1988 all'Università degli Studi di Roma "La Sapienza" per valutare la leggibilità di un documento redatto in lingua italiana, l'indice $\text{Gulpease}_{\text{G}}$ si basa sul calcolo del numero di caratteri contenuto in una parola rapportato con altri fattori quali il numero di parole e di frasi. La formula per il calcolo dell'indice Gulpease è la seguente:

$$89 + \frac{300\,(numero\ di\ frasi) - 10\,(numero\ di\ lettere)}{numero\ di\ parole}$$

Il risultato indica quindi la complessità del documento con un valore compreso tra 0 e 100, dove 100 indica la più alta leggibilità. Attraverso gli studi condotti, risulta che testi con un indice:

- **inferiore a 80** sono difficili da leggere per chi ha la licenza elementare;
- **inferiore a 60** sono difficili da leggere per chi ha licenza media;
- **inferiore a 40** sono difficili da leggere per chi ha un diploma superiore.

Tale indice, però, non indica la comprensibilità del testo. Il documento potrebbe contenere frasi incomprensibili ed avere comunque un alto indice Gulpease. Per la tipologia dei documenti redatti, la formalità nella scrittura e gli argomenti trattati risulta difficile adeguare la stesura del testo ad un indice Gulpease ottimale. Per questo motivo, ogni documento sarà valutato anche da un individuo che avrà il compito di valutare se parti di testo dovranno essere semplificate o meno. Inoltre, i limiti imposti da tale indice saranno sufficientemente rilassati per accettare anche frasi poco più complesse.

**Parametri utilizzati**

- **Range di accettazione**: 40 - 100;
- **Range ottimale**: 50 - 100.

### 4.2.3 Metriche per il codice

**Rapporto linee di commento su linee di codice**
Indica il rapporto tra linee di commento e linee di codice in un file (linee vuote escluse). Ritenendo importante la rapidità di comprensione del codice, questa metrica è utile per stimare la manutenibilità.

**Parametri utilizzati**

- **Range di accettazione**: $\geq 0.25$;
- **Range ottimale**: $\geq 0.30$.

**Numero di parametri**
Indica il numero di parametri formali di un metodo. Più alto è il numero dei parametri formali, più aumenta la quantità di memoria occupata nella pila dei processi.

**Parametri utilizzati**

- **Range di accettazione**: 0 - 8;
- **Range ottimale**: 0 - 5.

**Numero di campi dati**
Indica il numero di campi dati interni ad una classe. Un numero elevato può rendere difficile la manutenibilità del codice della classe, oltre ad essere indice di cattiva programmazione.
È possibile ridurre il numero di campi dati attraverso l'incapsulamento di ulteriori classi.

**Parametri utilizzati**

- **Range di accettazione**: 0 - 16;
- **Range ottimale**: 0 - 10.

**Complessità ciclomatica**
Indica il numero di cammini linearmente indipendenti attraverso il grafo di controllo di flusso del metodo/funzione: i nodi del grafo corrispondono a gruppi indivisibili di istruzioni, mentre gli archi connettono due nodi se il secondo gruppo può essere eseguito immediatamente dopo il primo.
È possibile ridurre l'indice di complessità attraverso la suddivisione del metodo/funzione in più parti.

**Parametri utilizzati**

- **Range di accettazione**: 0 - 10;
- **Range ottimale**: 0 - 6.

È accettato anche un valore più elevato, qualora dovesse influire positivamente sulla velocità di esecuzione.

**Livello di annidamento** Indica quante volte le strutture di controllo sono inserite l'una all'interno dell'altra. Un alto grado di annidamento può portare a una complessità maggiore del codice che porta a una difficoltà nella verifica, nella comprensione e nella modifica.

**Parametri utilizzati**

- **Range di accettazione**: 0 - 6;
- **Range ottimale**: 0 - 4.

**Grado di accoppiamento** Viene calcolato in base a due indici:

- **Accoppiamento afferente**: Numero di classi esterne al package che dipendono da classi del package. Se il numero è troppo alto troppe classi dipendono da tale package quindi modifiche una classe del package ha effetti che si ripercuotono su più classi. Se il numero è troppo basso il package risulta poco utile;
- **Accoppiamento efferente**: numero di classi interne a un package dipendenti da classi esterne al package. Un numero alto può essere sintomo di una scarsa progettazione.

**Grado di instabilità** Tale metrica viene utilizzata per misurare l'instabilità delle componenti di un sistema basandosi sul grado di accoppiamento delle classi. Un valore alto indica alta instabilità e quindi una bassa libertà di modifica del codice, in quanto ogni modifica ha effetti su più classi. Tale indice viene calcolato così:

$$\frac{C_\mathrm{e}}{C_\mathrm{e} + C_\mathrm{e}}$$

dove:

- $\mathbf{C_a}$ : rappresenta l'accoppiamento afferente;
- $\mathbf{C_e}$ : rappresenta l'accoppiamento efferente;

**Parametri utilizzati**

- **Range di accettazione**: 0 - 0.8;
- **Range ottimale**: 0.3 - 0.7.

**Chiamate innestate di metodi** Indica quante volte le strutture di controllo sono inserite l'una all'interno dell'altra. Un alto grado di annidamento può portare a una complessità maggiore del codice che porta a una difficoltà nella verifica, nella comprensione e nella modifica.

**Parametri utilizzati**

- **Range di accettazione**: 0 - 6;
- **Range ottimale**: 0 - 4.

**Copertura del codice** Rappresenta la percentuale di codice eseguita durante i test. Maggiore è questo valore, più esaurienti saranno i test e maggiori sono le probabilità di scoprire eventuali errori.

**Parametri utilizzati**

- **Range di accettazione**: 80 % -100 % ;
- **Range ottimale**: 90 % -100 % ;

**Numero di linee per metodo** Indica il numero di statement che compongono un metodo. Se un metodo risulta troppo lungo il suo funzionamento risulterà più complicato da comprendere, quindi può essere opportuno dividerlo in più sotto-funzioni.

**Parametri utilizzati**

- **Range di accettazione**: meno di 60;
- **Range ottimale**: meno di 40.

**Validazione W3C** L'applicativo web deve superare il test di validazione offerto da W3C con 0 errori gravi. Sono accettati gli avvisi e le incertezze che non compromettono la funzionalità del sito fino a un massimo di 10 per pagina.

**Parametri utilizzati**

- **Range di accettazione**: 0 - 10 (per pagina);
- **Range ottimale**: 0 - 0 (per pagina).