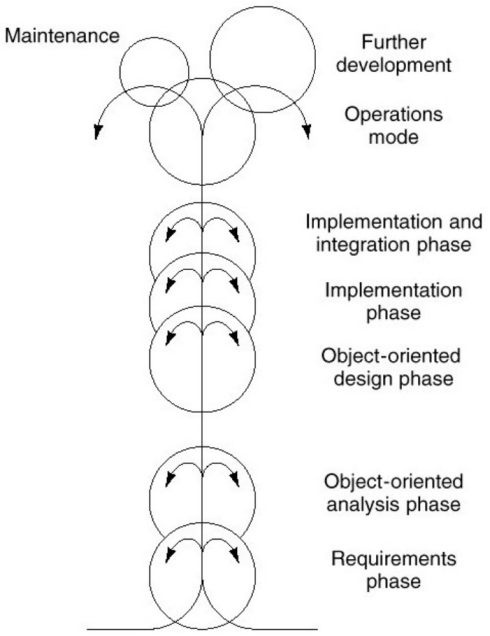
- ***<u>Lezione 6 – (28/10/2024)</u>***

Altri modelli che vengono usati sono:

- *<u>Modello object – oriented (o “Modello a Fontana”)</u>*

Questo modello è noto anche come "modello a fontana" a causa delle frecce che ricordano zampilli d'acqua. Si basa sull'idea di applicare i principi della programmazione orientata agli oggetti anche nelle fasi di analisi dei requisiti e progettazione del software. Si tratta di un modello altamente iterativo, in cui ogni cerchio rappresenta una fase specifica dello sviluppo. In particolare, il modello evidenzia che le fasi di specifica e progettazione devono essere affrontate con un approccio orientato agli oggetti. Tuttavia, una volta completata la fase di progettazione, non è strettamente necessario adottare una codifica orientata agli oggetti. Il modello prevede inoltre un'interazione sia all'interno di ciascuna fase (intrafase) che tra le fasi (interfase), con alcune forme di sovrapposizione tra le attività (cerchi sovrapposti nella figura).

Grazie a questo approccio, l'attività di manutenzione risulta ridotta rispetto al modello a cascata (Waterfall).



- *<u>Modello di ingegneria simultanea (o concorrente)</u>*

Ha come obiettivo la riduzione di tempi e costi di sviluppo, mediante un approccio sistematico al progetto integrato e concorrente di un prodotto software e del processo ad esso associato. Le fasi di sviluppo coesistono invece di essere eseguite in sequenza. Questo approccio è stato usato di più dalle case automobilistiche.

[Ha accennato ricordi del suo lavoro e ha introdotto un approccio chiamato M.o.n.è. (non so come è scritto)]

- *Modello basato su metodi formali*

Comprende una serie di attività che conducono alla specifica formale matematica del software, al fine di eliminare ambiguità, incompletezze ed inconsistenze e facilitare la verifica dei programmi mediante l'applicazione di tecniche matematiche.
La Cleanroom Software Engineering (1987) ne rappresenta un esempio di realizzazione, in cui viene enfatizzata la possibilità di rilevare i difetti del software in modo più tempestivo rispetto ai modelli tradizionali.

- *Il Modello Microsoft*

La Microsoft è un'azienda che sin dai tempi degli anni '80 ha affrontato, durante lo sviluppo di software commerciali, problemi di incremento della qualità dei prodotti software e di riduzione di tempi e costi.

Per risolvere ciò si è adottato un processo che è sia iterativo, sia incrementale sia concorrente, permettendo di esaltare doti di creatività delle persone che si occupano di sviluppare il Prodotto Software.

Per quanto riguarda le informazioni sull'approccio adottato da Microsoft per lo sviluppo dei propri software, non disponiamo di molte conoscenze, poiché l'azienda non ne ha mai parlato apertamente. Le uniche fonti che abbiamo a disposizione provengono da *M. A. Cusumano e R. W. Selby*, i quali hanno pubblicato un articolo intitolato *"How Microsoft Builds Software"* sulla rivista *"Communications of the ACM"*, vol. 40, n. 6, nel giugno del 1997.

Attualmente la Microsoft usa l'approccio denominato *"Synchronize-and-Stabilize".* Questo approccio è basato su:

- sincronizzazione → si sincronizzano quotidianamente le attività svolte da persone che lavorano sia individualmente che all'interno di piccoli team (da 3 a 8 persone). I componenti software, anche se solo parzialmente sviluppati, vengono assemblati in un prodotto (daily build), che viene poi testato e corretto.
- stabilizzazione → si stabilisce periodicamente il prodotto in incrementi (milestone) successivi durante l'avanzamento del progetto, piuttosto che un'unica volta alla fine

Questo approccio ha un ciclo di sviluppo a 3 fasi:

- 1° FASE → *Fase progettuale/di pianificazione:* definisce la visione, la descrizione dettagliata e la tabella di marcia del prodotto;
- 2° FASE → *Fase di sviluppo:* presenta lo sviluppo in 3/4 sottoprogetti sequenziali ognuno derivante da una pubblicazione importante (milestone release);
- 3° FASE → *Fase di stabilizzazione:* test onnicomprensivi interni ed esterni, prodotto finale, consolidamento e invio;

Planning Phase Define product vision, specification, and schedule

- **Vision Statement** Product and program management use extensive customer input to identify and priority-order product features.
- **Specification Document** Based on vision statement, program management and development group define feature functionality, architectural issues, and component interdependencies.
- **Schedule and Feature Team Formation** Based on specification document, program management coordinates schedule and arranges feature teams that each contain approximately 1 program manager, 3–8 developers, and 3–8 testers (who work in parallel 1:1 with developers).

Fase di Pianificazione
Definisci la visione del prodotto, le specifiche e il programma

- **Dichiarazione di Visione**: Il management del prodotto e del programma utilizza un ampio input dei clienti per identificare e ordinare in base alla priorità le caratteristiche del prodotto.
- **Documento delle Specifiche**: Sulla base della dichiarazione di visione, il management del programma e il gruppo di sviluppo definiscono la funzionalità delle caratteristiche, le problematiche architetturali e le interdipendenze dei componenti.
- **Pianificazione e Formazione dei Team di Funzionalità**: Basandosi sul documento delle specifiche, il management del programma coordina il programma e organizza i team di funzionalità, ognuno dei quali include circa 1 responsabile di programma, 3-8 sviluppatori e 3-8 tester (che lavorano in parallelo 1:1 con gli sviluppatori).

Development Phase Feature development in 3 or 4 sequential subprojects that each results in a milestone release

Program managers coordinate evolution of specification. Developers design, code, and debug. Testers pair with developers for continuous testing.

- **Subproject I** First 1/3 of features (Most critical features and shared components)
- **Subproject II** Second 1/3 of features
- **Subproject III** Final 1/3 of features (Least critical features)

Fase di Sviluppo
Sviluppo delle funzionalità in 3 o 4 sotto-progetti sequenziali, ognuno dei quali si conclude con una release milestone

I responsabili di programma coordinano l'evoluzione delle specifiche. Gli sviluppatori progettano, codificano e risolvono i bug. I tester collaborano con gli sviluppatori per eseguire test continui.

- **Sotto-progetto I**: Primo terzo delle funzionalità (le più critiche e i componenti condivisi)
- **Sotto-progetto II**: Secondo terzo delle funzionalità
- **Sotto-progetto III**: Ultimo terzo delle funzionalità (le meno critiche)

Stabilization Phase Comprehensive internal and external testing, final product stabilization, and ship

Program managers coordinate OEMs and ISVs and monitor customer feedback. Developers perform final debugging and code stabilization. Testers recreate and isolate errors.

- **Internal Testing** Thorough testing of complete product within the company
- **External Testing** Thotough testing of complete product outside the company by "beta" sites, such as OEMs, ISVs, and end users
- **Release preparation** Prepare final release of "golden master" disks and documentation for manufacturing

Fase di Stabilizzazione
Test approfonditi interni ed esterni, stabilizzazione finale del prodotto e rilascio

I responsabili di programma coordinano gli OEM e gli ISV e monitorano il feedback dei clienti. Gli sviluppatori effettuano il debug finale e la stabilizzazione del codice. I tester ricreano e isolano gli errori.

- **Test Interni**: Test approfonditi del prodotto completo all'interno dell'azienda
- **Test Esterni**: Test approfonditi del prodotto completo al di fuori dell'azienda da parte di siti "beta", come OEM, ISV e utenti finali
- **Preparazione alla Release**: Preparazione per il rilascio finale dei dischi "golden master" e della documentazione per la produzione

M. A. Cusumano e R. W. Selby, nel loro articolo definiscono pure le strategie che stanno alla base dello sviluppo software in Microsoft. Queste strategie presentate nell'articolo sono due:

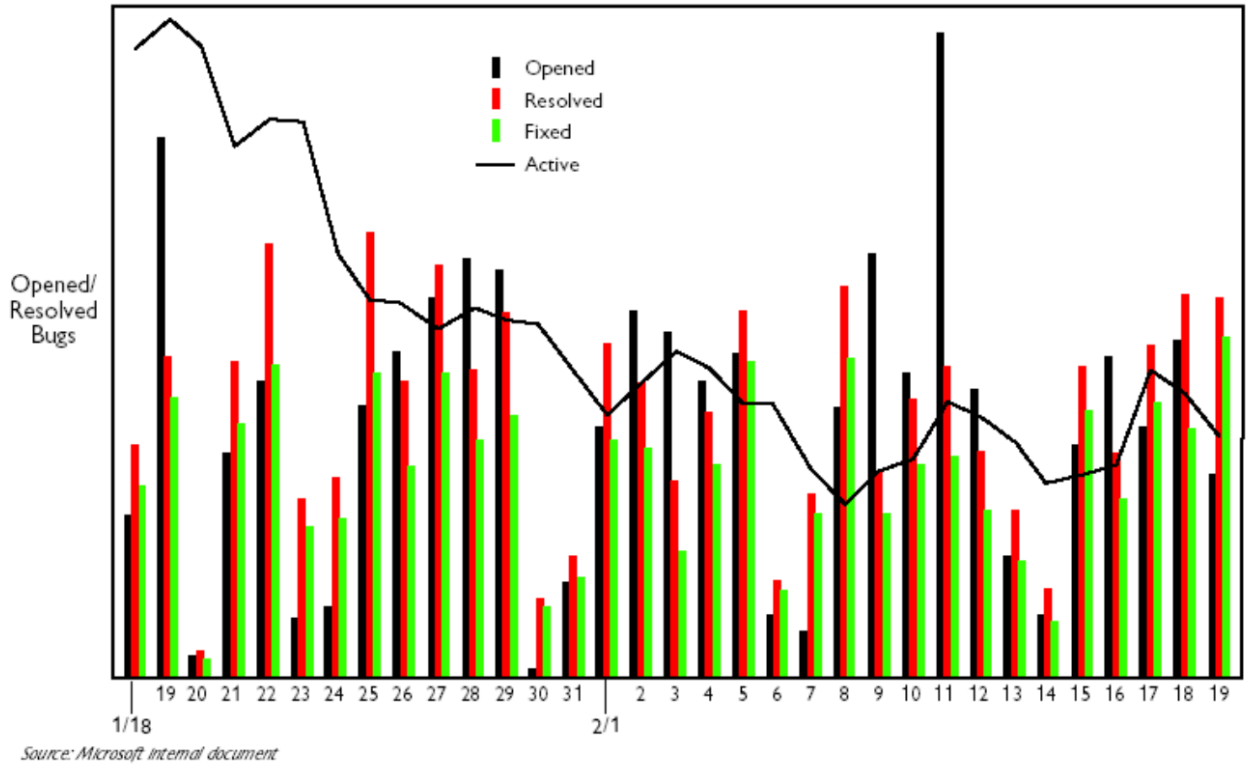
1. Strategia per definire prodotto e processo → "considerare la creatività come elemento essenziale". Principi di realizzazione:

   a. Dividere il progetto in *milestone* (da 3 a 4)
   b. Definire una "product vision" e produrre una specifica funzionale che evolverà durante il progetto
   c. Selezionare le funzionalità e le relative priorità in base alle necessità utente
   d. Definire un'architettura modulare per replicare nel progetto la struttura del prodotto
   e. Assegnare task elementari e limitare le risorse

2. Strategia per lo sviluppo e la consegna dei prodotti → "lavorare in parallelo con frequenti sincronizzazioni". Principi di realizzazione:

   a. Definire team paralleli ed utilizzare daily build perla sincronizzazione
   b. Avere sempre un prodotto da consegnare, con versioni per ogni piattaforma e mercato
   c. Usare lo stesso linguaggio di programmazione all'interno dello stesso sito di sviluppo
   d. Testare continuamente il prodotto durante il suo sviluppo
   e. Usare metriche per il supporto alle decisioni

L’unica metrica che viene collezionata è quella riportata in foto



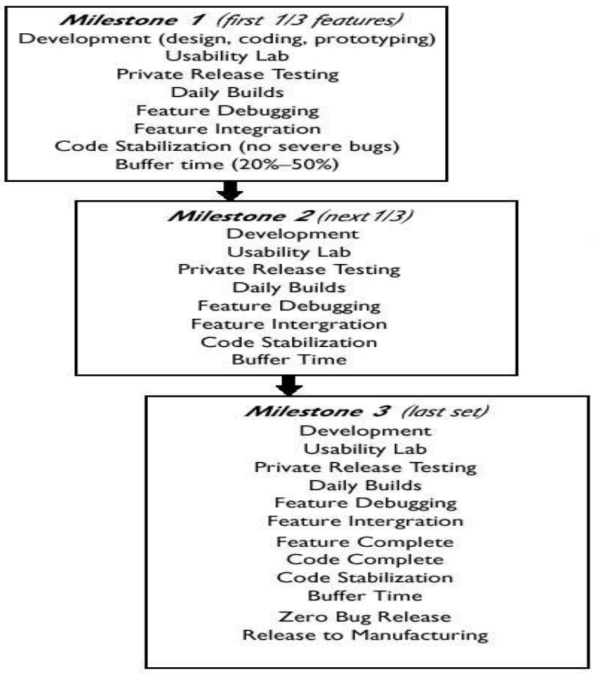
Ad ogni giorno sono associate delle metriche che definiscono i difetti. Nel grafico ogni giorno vedremo 3 colori:

- La barra nera indica i “difetti aperti”, ovvero quando si riscontra un malfunzionamento bisogna capire perché il software non funziona correttamente
- La barra rossa indica i difetti già localizzati, ovvero quando io ho capito dove si trova già il difetto
- La barra verde indica i difetti localizzati e che ho rimosso

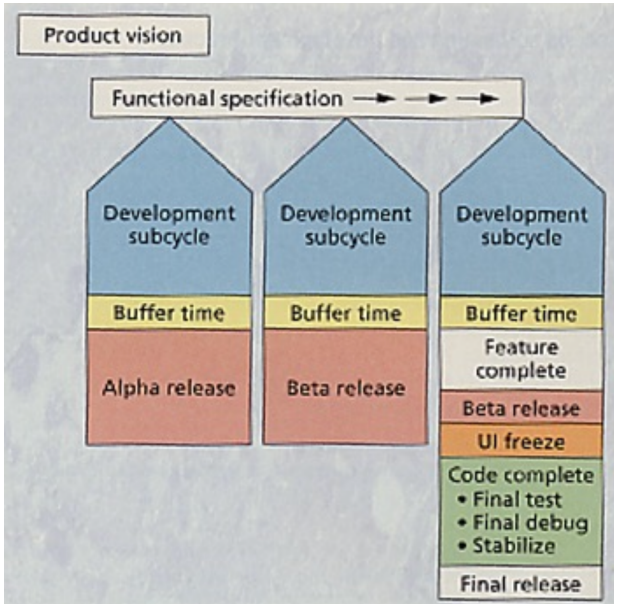
La linea tratteggiata rappresenta il numero di difetti attivi, ovvero quelli che sono stati identificati, ma non ancora risolti.

La qualità del prodotto è determinata dalle aspettative del cliente; in questo caso, Microsoft funge sia da cliente che da produttore, stabilendo i propri standard di qualità. L'affidabilità del prodotto è legata all'andamento di questa linea tratteggiata: se rimane al di sotto di una soglia prestabilita, possiamo considerare il prodotto pronto per essere immesso sul mercato, come promesso.

Un esempio di sviluppi di *Milestones*



Modello del ciclo di sviluppo *synch-and-stabilize*



Possiamo vedere che partiamo da una *Product Vision*, che è quella che viene messa sul mercato, poi ci sta una *Specifica Funzionale*, che è seguita da freccette per specificare che va avanti fino alla fine. Poi ci stanno le varie fasi di *milestones.*

La figura mostra il ciclo di sviluppo del prodotto, che parte dalla *Product Vision*, cioè la visione complessiva del prodotto che si vuole immettere sul mercato.

Successivamente si passa alla *Specificazione Funzionale*, che continua lungo tutto il ciclo di sviluppo (indicata dalle frecce).

Il processo è suddiviso in tre *sottocicli di sviluppo* (*Development Subcycle*), ognuno dei quali include un *Buffer Time*, cioè un tempo aggiuntivo rispetto a quello stimato per gestire eventuali imprevisti e mantenere il progetto nei tempi previsti.

Ogni sottociclo porta a diverse fasi di rilascio:

- Il primo sottociclo culmina nella *Alpha Release*, una versione iniziale del software.

- Il secondo sottociclo rilascia la *Beta Release*, una versione più avanzata ma ancora soggetta a test.

- Il terzo sottociclo prevede la fase di *UI Freeze*, in cui l'interfaccia utente viene congelata, seguita dal completamento del codice e delle funzionalità principali, per poi procedere con i test finali e le ultime correzioni.

Una volta superate tutte queste fasi, si arriva alla *Final Release*, la versione stabile e pronta per il mercato.

Confronto tra modelli *synch - and - stabilize* e *waterfall*

Synch-and-Stablize	Sequential Development
Product development and testing done in parallel	Separate phases done in sequence
Vision statement and evolving specification	Complete "frozen" specification and detailed design before building the product
Features prioritized and built in 3 or 4 milestone subprojects	Trying to build all pieces of a product simultaneously
Frequent synchronizations (daily builds) and intermediate stabilizations (milestones)	One late and large integration and system test phase at the project's end
"Fixed" release and ship dates and multiple release cycles	Aiming for feature and product "perfection" in each project cycle
Customer feedback continuous in the development process	Feedback primarily after development as inputs for future projects
Product and process design so large teams work like small teams	Working primarily as a large group of individuals in a separate functional department