Esempi di APM

Metodologie e Approcci di Agile Project Management

Kanban

Kanban: origine

kanban (看板), termine giapponese che letteralmente significa «**insegna**»
(**kan** = 看= ***visuale***, **ban** = 板 = ***segnale***)

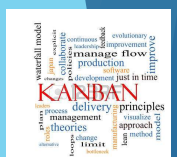
La metodologia Kanban nasce all'interno del sistema **JIT** (Just-in-time)
di reintegrazione delle scorte mano a mano che vengono consumate

La produzione JIT è stata un'idea di Kiichirō Toyoda, fondatore della Toyota.

«Il just in time (spesso abbreviato in JIT), espressione inglese che significa "appena in tempo", è una filosofia industriale che ha invertito il "vecchio metodo" di produrre prodotti finiti per il magazzino in attesa di essere venduti (detto **logica push**) passando alla **logica pull** secondo cui occorre produrre solo ciò che è stato già venduto o che si prevede di vendere in tempi brevi.»

Fonte: https://it.wikipedia.org/wiki/Just_in_time_(produzione)

Kanban: cosa è un kanban?



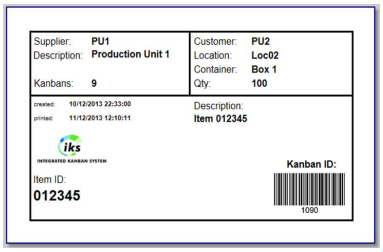
- Il kanban si configura come un **cartellino quadrato** che contiene le informazioni necessarie per produrre, acquistare o movimentare componenti e materiali nel sistema produttivo

- I kanban nel loro originale significato si possono distinguere in due grandi tipologie:
  A. **di movimentazione** (o di trasporto)
     - servono per spostare componenti e materiali verso un processo produttivo
  B. **di produzione**
     - rappresentano veri e propri ordini di produzione mediante i quali si autorizza il processo a monte a produrre un certo componente per un processo a valle
- Nel Project Management il kanban rappresenta un'**attività da svolgere**

Kanban: informazioni sul kanban

Le informazioni che generalmente si possono trovare su un cartellino kanban sono:

1. Il codice del componente interessato
2. Il fornitore di quel componente
3. Il cliente che lo richiede
4. Il tempo a disposizione per il ripristino
5. La quantità da ripristinare
6. Il contenitore da utilizzare
7. Altre informazioni personalizzate



- Nel Project Mgnt solitamente contengono una **frase rappresentativa dell'attività da svolgere**

Kanban nel Project Mgnt

- Metodologia di gestione LEAN ispirata:
  - ai principi dello sviluppo software agile
  - alla teoria dei vincoli
  - ai principi del TPM (Toyota Production System)
- Inizialmente proposta da *David J. Anderson* nell'agosto 2007 …
- … successivamente ufficializzata nel libro "Kanban" scritto dallo stesso *Anderson* e pubblicato nel 2010
- OBIETTIVO:

Evitare la **sovrapproduzione**

che è lo spreco più impattante

sulle performance di un sistema produttivo

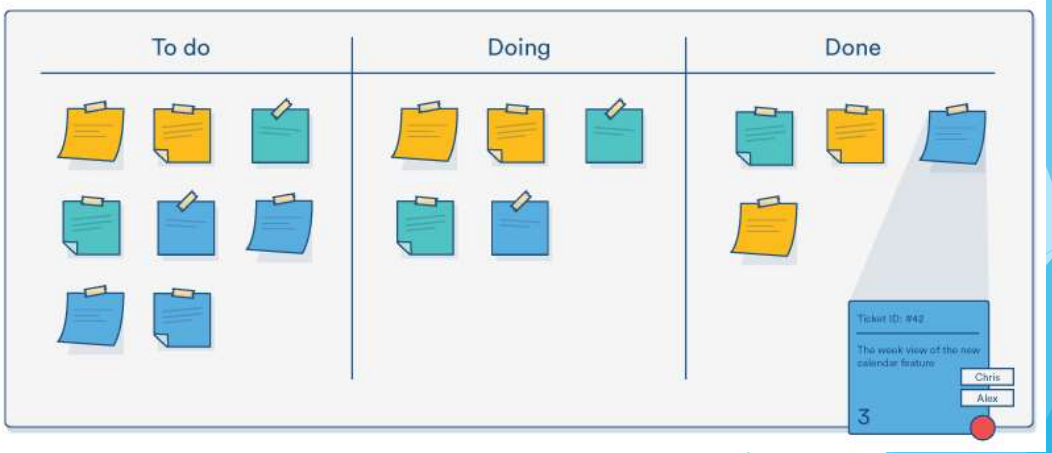
Kanban: "Kanban" vs "kanban"

Nella comunità dello sviluppo software si fa riferimento a:

"**Kanban**" (con la 'K' maiuscola) per indicare il **processo di sviluppo software**

"**kanban**" (con la 'k' minuscola) per indicare **l'uso di cartellini** come segnalatori visuali, per la realizzazione di sistemi pull così come descritto nel Toyota Production System.

Kanban: how to



Kanban: le 5 proprietà

Le cinque proprietà Kanban di *David J. Anderson*:

1. Visualizzare il flusso di lavoro
2. Limitare il Work-in-Progress
3. Misurare e gestire il flusso
4. Rendere le politiche di processo esplicite
5. Utilizzare i modelli per riconoscere le opportunità di miglioramento

XP (Extreme Programming)

XP (Extreme Programming)

- Metodologia agile di «programmazione estrema»
- Enfatizza:
  - scrittura di codice di qualità
  - rapidità di risposta ai cambiamenti di requisiti
- Fa uso di tecniche di controllo e ottimizzazione del codice:
  - **Pair programming** (vedi dopo)
  - **Unit test** che rappresentano i requisiti
    (= test automatizzati che devono essere superati ad ogni integrazione del codice software)
  - **Refactoring** (vedi dopo)
- Vieta agli sviluppatori di scrivere codice non strettamente necessario
- Richiede comunicazione diretta e frequente del tipo:
  - sviluppatore-cliente
  - sviluppatore-sviluppatore

XP: basi

Si basa su 12 regole (o pratiche) raggruppate in 4 aree:

1. Feedback a scala fine
2. Processo continuo
3. Comprensione condivisa
4. Benessere dei programmatori

XP: basi

Si basa su 12 regole (o pratiche) raggruppate in 4 aree:

1. **Feedback a scala fine**
    1. Pair programming
    2. Planning Game
    3. TDD (Test Driven Development)
    4. Whole Team
2. Processo continuo
3. Comprensione condivisa
4. Benessere dei programmatori

XP: basi

Si basa su 12 regole (o pratiche) raggruppate in 4 aree:

1. **Feedback a scala fine**
    1. **Pair programming**
        - 2 programmatori lavorano insieme su una sola workstation, uno sta alla tastiera e l'altro ragiona sull'approccio e pensa se può funzionare.
        - Questo rende il codice prodotto di migliore qualità.
        - I due programmatori devono avere la stessa esperienza.
    2. Planning Game
    3. TDD (Test Driven Development)
    4. Whole Team
2. Processo continuo
3. Comprensione condivisa
4. Benessere dei programmatori

XP: basi

Si basa su 12 regole (o pratiche) raggruppate in 4 aree:

1. **Feedback a scala fine**
   1. Pair programming
   2. **Planning Game**
      - è una riunione di pianificazione che avviene una volta per ciascuna iterazione (tipicamente una volta a settimana).
   3. TDD (Test Driven Development)
   4. Whole Team
2. Processo continuo
3. Comprensione condivisa
4. Benessere dei programmatori

XP: basi

Si basa su 12 regole (o pratiche) raggruppate in 4 aree:

1. **Feedback a scala fine**
   1. Pair programming
   2. Planning Game
   3. **TDD (Test Driven Development)**
      - i test automatici (sia unit test che di accettazione) vengono scritti prima di scrivere il codice
   4. Whole Team
2. Processo continuo
3. Comprensione condivisa
4. Benessere dei programmatori

XP: basi

Si basa su 12 regole (o pratiche) raggruppate in 4 aree:

1. **Feedback a scala fine**
    1. Pair programming
    2. Planning Game
    3. TDD (Test Driven Development)
    4. **Whole Team** (Team «totale»)
        1. In XP, il "cliente" non è colui che paga il conto, ma la persona che realmente utilizza il sistema.
        2. Il cliente deve essere presente e disponibile a verificare (sono consigliate riunioni settimanali o **Jour fixe**).
2. Processo continuo
3. Comprensione condivisa
4. Benessere dei programmatori

XP: basi

Si basa su 12 regole (o pratiche) raggruppate in 4 aree:

1. Feedback a scala fine
2. **Processo continuo**
    5. Continuous integration
    6. Refactoring o Design Improvement
    7. Small Releases
3. Comprensione condivisa
4. Benessere dei programmatori

XP: basi

Si basa su 12 regole (o pratiche) raggruppate in 4 aree:

1. Feedback a scala fine
2. **Processo continuo**
   5. **Continuous integration**
      - Integrare continuamente i cambiamenti al codice eviterà ritardi più avanti nel ciclo del progetto, causati da problemi d'integrazione.
   6. Refactoring o Design Improvement
   7. Small Releases
3. Comprensione condivisa
4. Benessere dei programmatori

XP: basi

Si basa su 12 regole (o pratiche) raggruppate in 4 aree:

1. Feedback a scala fine
2. **Processo continuo**
   5. Continuous integration
   6. **Refactoring o Design Improvement**
      - riscrivere il codice senza alterarne le funzionalità esterne, cambiando l'architettura, in modo da renderlo più semplice e generico.
   7. Small Releases
3. Comprensione condivisa
4. Benessere dei programmatori

XP: basi

Si basa su 12 regole (o pratiche) raggruppate in 4 aree:

1. Feedback a scala fine
2. **Processo continuo**
   5. Continuous integration
   6. Refactoring o Design Improvement
   7. **Small Releases**
      - consegna del software avviene tramite **frequenti rilasci** di funzionalità che creano del valore concreto.
3. Comprensione condivisa
4. Benessere dei programmatori

XP: basi

Si basa su 12 regole (o pratiche) raggruppate in 4 aree:

1. Feedback a scala fine
2. Processo continuo
3. **Comprensione condivisa**
   8. Coding standards
   9. Collective code ownership
   10. Simple design
   11. System metaphor
4. Benessere dei programmatori

XP: basi

Si basa su 12 regole (o pratiche) raggruppate in 4 aree:

1. Feedback a scala fine
2. Processo continuo
3. **Comprensione condivisa**
   8. **Coding standards**
      - Scegliere ed utilizzare un preciso **standard di scrittura del codice**.
      - Rappresenta un insieme di **regole concordate**, che l'intero team di sviluppo accetta di rispettare nel corso del progetto.
   9. Collective code ownership
   10. Simple design
   11. System metaphor
4. Benessere dei programmatori

XP: basi

Si basa su 12 regole (o pratiche) raggruppate in 4 aree:

1. Feedback a scala fine
2. Processo continuo
3. **Comprensione condivisa**
   8. Coding standards
   9. **Collective code ownership**
      - significa che ognuno è responsabile di tutto il codice
        → contribuisce alla stesura chiunque sia coinvolto nel progetto.
   10. Simple design
   11. System metaphor
4. Benessere dei programmatori

XP: basi

Si basa su 12 regole (o pratiche) raggruppate in 4 aree:

1. Feedback a scala fine
2. Processo continuo
3. **Comprensione condivisa**
   8. Coding standards
   9. Collective code ownership
   10. **Simple design**
       - i programmatori dovrebbero utilizzare un approccio del tipo «Simple is better" alla progettazione software.
       - Progettare al minimo e con il cliente.
   11. System metaphor
4. Benessere dei programmatori

XP: basi

Si basa su 12 regole (o pratiche) raggruppate in 4 aree:

1. Feedback a scala fine
2. Processo continuo
3. **Comprensione condivisa**
   8. Coding standards
   9. Collective code ownership
   10. Simple design
   11. **System metaphor**
       - descrivere il sistema con una metafora, anche per la descrizione formale.
       - Questa può essere considerata come una storia che ognuno - clienti, programmatori, e manager - può raccontare circa il funzionamento del sistema.
4. Benessere dei programmatori

XP: basi

Si basa su 12 regole (o pratiche) raggruppate in 4 aree:

1. Feedback a scala fine
2. Processo continuo
3. Comprensione condivisa
4. **Benessere dei programmatori**
   12. **Sustainable pace**
       - il concetto è che i programmatori o gli sviluppatori software non dovrebbero lavorare più di 40 ore a settimana.

XP: linee guida

4 linee guida:

1. **Comunicazione**
   tutti possono parlare con tutti, persino l'ultimo dei programmatori con il cliente
2. **Semplicità**
   gli analisti mantengano la descrizione formale il più semplice e chiara possibile
3. **Verifica**
   sin dal primo giorno si prova il codice
4. **Coraggio**
   si dà in uso il sistema il prima possibile e
   si implementano i cambiamenti richiesti man mano

XP: fasi

4 fasi di progetto, ognuna delle quali con le sue regole interne:

1. **Pianificazione**
   user stories, release planning, small releases, project velocity, load factor, iterative development, iteration planning, move people around, daily stand up meeting, fix XP
2. **Progettazione**
   simplicity, system metaphor, CRC cards, spike solution, never add early, refactoring
3. **Sviluppo**
   Customer Always Available, Standards, Unit Test First, Pair Programming, Sequential Integration, Integrate Often, Collective Code Ownership, Optimize Last, No Overtime
4. **Collaudo**
   unit test framework, bug's found, functional test o acceptance tests